AAF-GAN: APLICAÇÃO DE FUNÇÕES DE ATIVAÇÃO HIPERBÓLICAS ADAPTATIVAS EM REDES NEURAIS ADVERSÁRIAS GENERATIVAS

Christian da Silva Cabral Cardozo

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Geraldo Zimbrão da Silva

Rio de Janeiro
Maio de 2021

AAF-GAN: APLICAÇÃO DE FUNÇÕES DE ATIVAÇÃO HIPERBÓLICAS ADAPTATIVAS EM REDES NEURAIS ADVERSÁRIAS GENERATIVAS

Christian da Silva Cabral Cardozo

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Geraldo Zimbrão da Silva

Aprovada por: Prof. Geraldo Zimbrão da Silva
Prof. Alexandre de Assis Bento Lima
Prof. Leandro Guimarães Marques Alvim

RIO DE JANEIRO, RJ – BRASIL
MAIO DE 2021

Cardozo, Christian da Silva Cabral

AAF-GAN: Aplicação de funções de ativação hiperbólicas adaptativas em redes neurais adversárias generativas/Christian da Silva Cabral Cardozo. – Rio de Janeiro: UFRJ/COPPE, 2021.

XIII, 99 p.: il.; 29, 7cm.

Orientador: Geraldo Zimbrão da Silva

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2021.

Referências Bibliográficas: p. 73 – 81.

1. Redes Neurais Adversárias Generativas. 2. Penalização Hiperbólica. 3. Função de Ativação. I. Silva, Geraldo Zimbrão da. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

À minha avó Cremilda
(In memoriam)

Agradecimentos

Gostaria de primeiramente agradecer aos meus pais. Mônica e José, vocês foram a base de tudo que eu pude conquistar. Graças à vocês minha realidade mudou, e espero que eu possa retribuir cada gota de suor que vocês dedicaram à mim para mudar a de vocês também.

Agradeço à minha esposa, Lívia, pelos longos anos de amizade, companheirismo e amor que temos juntos. Você tem sido minha força diária, do meu lado sempre que eu preciso de um sorriso ou de alguém para chorar junto. Me sinto a pessoa mais feliz do mundo por viver esse amor com você.

Agradeço carinhosamente à toda a minha família que sempre acreditou em mim e me apoiou, especialmente meu irmão caçula Caio e minha tia madrinha Cristina. É ótimo ter uma família que vibra com você a cada etapa concluída, te dando forças para continuar no caminho tão sonhado.

Aos meus amigos, minha eterna gratidão. Vocês são parte da minha essência e eu não reconheceria a mim mesmo se não fosse por vocês. Mesmo agora de muito longe, vocês continuam sendo irmãos para mim. Obrigado aos meus amigos de Realengo e de infância, aos amigos que conheci durante o ensino médio e faculdade, e aos que recentemente conheci na Irlanda. Agradeço em especial aos meus amigos Luiz Alberto, Francisco, Filipe, Wenderson, Wesley, Nicholas, Igor, Aluizio, Felipe Milepe, Beatriz, Camilla e Thamires.

Agradeço aos professores que fizeram parte da minha formação acadêmica e profissional, transferindo seus conhecimentos e experiências para que eu me tornasse um profissional na área que eu sempre me identifiquei. Agradeço especialmente ao meu orientador Geraldo Zimbrão, que me orientou em diversas iniciativas ao longo da minha trajetória desde o meu segundo período na graduação.

"Portanto, não se preocupem com o amanhã, pois o amanhã trará as suas próprias preocupações. Basta a cada dia o seu próprio mal". Obrigado Deus, por mais uma vitória.

Ego vici mundum.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

AAF-GAN: APLICAÇÃO DE FUNÇÕES DE ATIVAÇÃO HIPERBÓLICAS ADAPTATIVAS EM REDES NEURAIS ADVERSÁRIAS GENERATIVAS

Christian da Silva Cabral Cardozo

Maio/2021

Orientador: Geraldo Zimbrão da Silva

Programa: Engenharia de Sistemas e Computação

As redes neurais adversárias generativas (GANs) são uma arquitetura recente de modelo generativo em que duas redes neurais são treinadas competindo entre si, onde o gerador busca produzir amostras semelhantes aos dados reais sem acesso direto à base de dados e o discriminador busca maximizar seus acertos em classificar quais exemplos são falsos e quais são reais. Umas das alternativas que busca melhorar o desempenho de redes neurais até então não explorada em redes GANs são as funções de ativação adaptativas. Estudos recentes indicam que funções adaptativas baseadas na técnica de penalização hiperbólica são capazes de acelerar a convergência durante o treinamento de redes neurais multicamadas.

Neste trabalho, apresentamos uma nova abordagem para redes GANs através da aplicação de funções de ativação adaptativas baseadas em penalização hiperbólica com o objetivo de analisar seu impacto na qualidade das amostras geradas na tarefa de geração de imagens. Propomos também uma nova versão para a função Mish com parâmetros treináveis utilizando uma função bi-hiperbólica em sua fórmula. Por fim, exploramos uma limitação teórica da função bi-hiperbólica assimétrica adaptativa (BHAA) no que diz respeito ao seu intervalo de atuação, propondo uma versão normalizada. Resultados mostraram que certas estratégias podem melhorar a convergência da rede GAN utilizando as funções estudadas. Além disso, validamos que a função MiDA é capaz de melhorar, ou no mínimo, manter a performance da rede GAN quando comparada com uso da função Mish. As descobertas formam diretrizes de aplicações na arquitetura de redes adversárias que podem ser base para mais estudos na área.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

AAF-GAN: HYPERBOLIC ADAPTIVE ACTIVATION FUNCTIONS APPLIED TO GENERATIVE ADVERSARIAL NETWORKS

Christian da Silva Cabral Cardozo

May/2021

Advisor: Geraldo Zimbrão da Silva

Department: Systems Engineering and Computer Science

Generative adversarial networks (GANs) are a generative model architecture proposed a few years ago where two neural networks train against each other. The generator network tries to produce samples similar to real data with no direct contact with them, and the discriminator network attempts to maximize the accuracy of predicting which data is fake generated and legit from the dataset. Adaptive activation functions are an approach yet not explored in GANs that could improve their performance. Specifically, functions based on the hyperbolic penalization technique indicate that convergence can be improved in the multi-layer network's training's early epochs.

In this work, we propose a new approach to GANs networks by applying adaptive activation functions based on hyperbolic penalization to analyze the impact on the generated samples quality, considering the image generation task. We also propose a new version using adaptive parameters for the Mish function by changing its formula to use a hyperbolic function to self-gating. Finally, we explore the theoretical limitation of the Adaptive Asymmetric Bi-hyperbolic function due to non-limited boundaries by proposing a new normalized formulation. We found strategies resulting in better performance for early convergence for GANs. Also, we validate that the MiDA function can, at minimum, improve the convergence compared to Mish performance. The discoveries were compiled into guidelines that can be used as the base for applying adaptive functions to GANs.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

As inteligências artificiais e os sistemas inteligentes estão cada vez mais presentes no dia a dia da sociedade humana. Os avanços de *hardware* dos dispositivos e dos algoritmos de aprendizado de máquina possibilitaram que diversas ferramentas e novas técnicas fossem desenvolvidas. Carros autônomos, análise para diagnóstico de doenças, sistemas de recomendação, IA para jogos e transcrição automática de fala são alguns exemplos onde a sociedade aplica as descobertas de aprendizado de máquina para melhorar a sua qualidade de vida, saúde e entretenimento.

As redes neurais são um dos modelos de aprendizado de máquina que tornam possíveis os exemplos citados acima. Inúmeras aplicações fazem uso de diferentes modelos e arquiteturas de redes neurais para a resolução de tarefas como reconhecimento facial e de expressões faciais (JIANG e LEARNED-MILLER, 2017, LAWRENCE *et al.*, 1997, MENG JOO ER *et al.*, 2002), transcrição e reconhecimento de fala (ABDEL-HAMID *et al.*, 2014, GRAVES e JAITLY, 2014, MIRSAMADI *et al.*, 2017), processamento de linguagem natural (DOS SANTOS e GATTI, 2014, SOCHER e MANNING, 2013), detecção de objetos em imagens (ERHAN *et al.*, 2014, SZEGEDY *et al.*, 2013) e imagens sinteticamente geradas (GREGOR *et al.*, 2015, KARRAS *et al.*, 2019).

Uma das terminologias para classificar modelos de aprendizado de máquina leva em consideração qual o objetivo da saída do modelo. Desta forma temos como classificação modelos discriminativos e generativos. Os modelos discriminativos têm por objetivo classificar uma amostra dada como entrada. Já os modelos generativos possuem o objetivo de gerar amostras baseadas nas estatísticas probabilísticas aprendidas dado uma entrada. Desta forma, modelos generativos são utilizados para sintetizar dados artificialmente. Algumas das aplicações para modelos generativos listadas por THEIS *et al.* (2016) são: remoção de ruídos em imagens/vídeos, *data*

augmentation, síntese de texturas, entre outras.

Dentro do conjunto de tarefas que compreendem a síntese de dados artificialmente gerados, um dos modelos de grande destaque e relativamente recente, mas amplamente estudado, é conhecido como *generative adversarial networks* (GAN) (GOODFELLOW *et al.*, 2014). Desde sua formulação, o interesse em estudos relacionados à redes GAN vem crescendo como podemos notar no gráfico da figura 1.1 com dados coletados do repositório *dblp*[1].

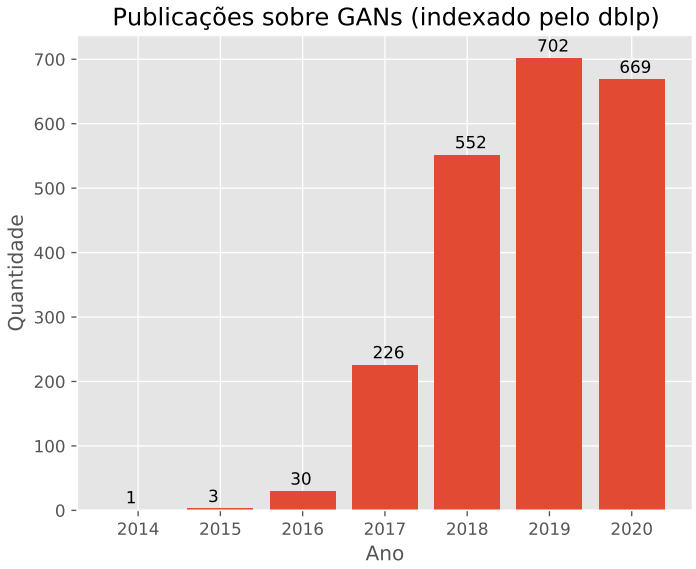


Figura 1.1: Gráfico de publicações sobre redes GANs (contendo o termo '*generative adversarial network*') por ano indexados pelo *dblp*.

O grande interesse em modelos GAN pode ser atribuído ao seu potencial em aprender distribuições complexas de dados independentemente do domínio da base. HONG *et al.* compila as diversas variantes de GAN e lista onde seu poder de aprendizado é aplicado com sucesso. Tarefas de aprendizado de máquina como tradução imagem-para-imagem, predição de *frames* em vídeos, *data augmentation*, e tratamento de imagens são exemplos onde modelos GANs obtém grande performance na qualidade dos resultados.

Além dos estudos das arquiteturas de redes neurais, que deram origem por exemplo às redes GANs, outro aspecto amplamente estudado em redes neurais se diz respeito às funções de ativação. As funções de ativação são um dos componentes essenciais em redes neurais e sua correta aplicação pode influenciar consideravelmente os resultados durante o processo de aprendizagem (MAAS *et al.*, 2013). Funções de ativação tradicionais possuem formato fixo, o que teoricamente pode limitar a flexibilidade de aprendizado durante o treinamento (AGOSTINELLI *et al.*, 2015), além

[1]dblp - Computer Science Bibliography: `https://dblp.org/`

de serem na maioria das vezes mais suscetíveis ao problema de gradiente minguante (BENGIO *et al.*, 1994).

Neste contexto, o estudo de funções de ativação é uma área ativa de pesquisas e as funções com parâmetros adaptativos surgem como uma opção que busca facilitar a escolha de uma função com o formato apropriado para o conjunto de dados aplicado (AGOSTINELLI *et al.*, 2015, HOU *et al.*, 2017, JAGTAP *et al.*, 2020, QIAN *et al.*, 2018). Dentre as opções de funções adaptativas, as funções baseadas em penalização hiperbólica (XAVIER, 1982) se apresentam como candidatas capazes de acentuar a convergência de resultados durante o treinamento de redes neurais multi-camadas (CANALLI, 2017).

Por fim, a ausência de estudos de funções de ativação adaptativas aplicadas nos modelos de rede GAN se apresenta como uma possibilidade ainda não explorada pela literatura até o momento. Modelos de redes neurais profundas, como o GAN, se caracterizam pela profundidade de camadas da sua arquitetura, o que exige mais recursos computacionais de memória e processamento durante o processo de aprendizado. A aplicação de funções adaptativas que melhoram a convergência do treinamento em épocas iniciais pode significar um ganho computacional para o modelo quando executado em ambientes com recursos de tempo, processamento e/ou memória limitados.

1.2 Objetivos

O presente trabalho tem por objetivo estudar o impacto que funções de ativação adaptativas baseadas em penalização hiperbólica podem causar para o treinamento de redes GANs na tarefa de geração de imagens. Espera-se que as imagens geradas pelo modelo GAN alcancem uma determinada qualidade baseada na métrica de avaliação mais rapidamente quando comparado ao modelo tradicional.

Além da aplicação das funções hiperbólicas tradicionais, elaboramos uma nova função de ativação candidata a substituir a função ReLU em camadas ocultas de redes neurais: a função MiDA (*Mish Dual Adaptive*). Esta função, baseada em uma recente proposta de função de ativação com formato fixo (Mish), possui dois parâmetros ajustáveis advindos do uso de uma função hiperbólica em sua fórmula substituindo a função tangente hiperbólica presente na função original. Assim como as outras funções hiperbólicas, aplicamos a função MiDA com o objetivo de melhorar a convergência da qualidade das imagens geradas pelo modelo.

Por fim, realizamos uma análise teórica de uma limitação da função bi-hiperbólica assimétrica adaptativa (BHAA) no que diz respeito ao seu intervalo de atuação. Propomos uma versão normalizada da função utilizando sua derivada como uma possibilidade de solucionar, em teoria, o problema de extrapolação dos limites de

atuação da função.

1.3 Contribuições

Este trabalho explora uma nova abordagem utilizando funções adaptativas em redes neurais adversárias. Para tal abordagem tanto funções baseadas em técnica de penalização hiperbólica já definidas na literatura quanto novas funções propostas neste trabalho foram utilizadas. No contexto de utilização de funções hiperbólicas adaptativas e redes neurais adversárias, este trabalho trás como contribuições: (i) análise do impacto do uso de funções hiperbólicas adaptativas em diferentes configurações de redes neurais; (ii) quais as melhores estratégias de aplicação de funções adaptativas hiperbólicas de acordo com os resultados experimentais; (iii) uma nova formulação para a função Mish, substituindo a aplicação de uma *tanh* por uma função BHSA, criando uma nova função adaptativa que denominamos MiDA; (iv) proposta de normalização matemática da função BHAA para atacar o problema de extrapolação dos limites de atuação da função.

1.4 Organização

A presente dissertação está organizada da seguinte forma:

1. no capítulo 2 tratamos da fundamentação teórica necessária para o desenvolvimento da proposta e dos experimentos. Os conceitos de redes neurais, *generative adversarial networks* e funções de ativação são abordados durante este capítulo.

2. no capítulo 3 descrevemos as propostas deste trabalho, apresentando duas novas funções de ativação adaptativas baseadas em penalização hiperbólica (MiDA e BHANA) além de definir uma nova família de modelos de redes GANs para quando funções de ativação adaptivas são utilizadas: *adaptive activation functions generative adversarial networks* (AAF-GAN).

3. no capítulo 4 apresentamos a metodologia utilizada e os resultados das avaliações experimentais do modelo proposto, comparando seu desempenho com arquiteturas *baseline*.

4. por fim, no capítulo 5 discutimos as considerações finais pontuando as vantagens e desvantagens do modelo proposto baseando-se nos resultados obtidos, além de abordar as contribuições científicas do presente estudo e possíveis trabalhos futuros.

Capítulo 2

Revisão Bibliográfica

Neste capítulo abordamos a fundamentação teórica base deste trabalho. Iniciamos a discussão com o assunto de redes neurais artificiais, introduzindo suas origens, evolução e seus componentes fundamentais. Logo após, introduzimos o modelo de redes neurais adversárias, elucidando seu funcionamento e citando algumas das arquiteturas já elaboradas. Por fim, discutimos sobre funções de ativação, como funcionam as funções de ativação com parâmetros adaptativos e introduzimos as funções baseadas na técnica de penalização hiperbólica.

2.1 Redes Neurais Artificiais

2.1.1 Perceptron

A busca pela simulação do comportamento de neurônios presentes no cérebro humano data dos anos 40. MCCULLOCH e PITTS (1943) introduziram o primeiro estudo de um modelo computacional com tal objetivo. Posteriormente, ROSENBLATT (1958) propôs um modelo inovador denomidado *perceptron*, que pode ser definido como um modelo de classificação linear capaz de lidar com duas classes (BISHOP, 2006). O *perceptron* recebe um vetor numérico x como entrada e computa o resultado de saída aplicando a seguinte transformação matemática:

$$y(x) = f(x \cdot w + b) \tag{2.1}$$

Onde w é denominado *matriz de pesos* (ou *weights*), b é denominado *viés* (ou *bias*) e a função $f(\cdot)$ é denominada *função de ativação*. No caso base do *perceptron*, a função de ativação é definida pela função degrau:

$$f(x) = \begin{cases} 1 & \text{se } x > 0, \\ 0 & \text{caso contrário} \end{cases} \tag{2.2}$$

Onde 0 e 1 representam as classes alvo.

A figura 2.1 é um diagrama que representa a equação 2.1 do *perceptron*. Neste diagrama, transformamos o parâmetro *bias* para que ele componha parte da entrada. Adicionamos então um elemento a mais na entrada, igual a 1, que será aplicada à multiplicação pelo peso w_0, que na verdade é o *bias*.

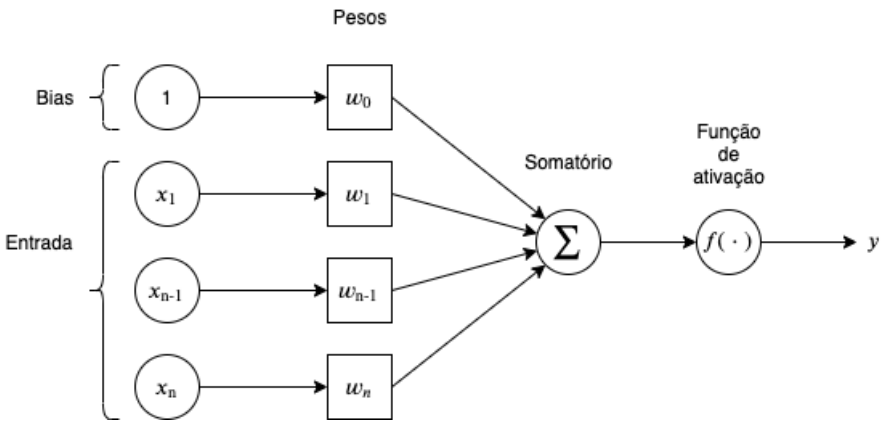


Figura 2.1: Representação gráfica de um *perceptron*

Através da transformação matemática da equação 2.1 o *perceptron* é capaz de realizar associações que mapeiam as entradas às classes alvo de saída de acordo com o ajuste dos parâmetros *weights* e *bias*.

Apesar de inovador para a época em que surgiu, o *perceptron* possui limitações que impedem o modelo de efetuar tarefas não-triviais (MINSKY e PAPERT, 1969). Caso o conjunto de dados envolvido no problema não seja linearmente separável, por exemplo, o *perceptron* não será capaz de oferecer uma solução. Para problemas com um conjunto de dados não-trivial precisamos fazer uso de uma arquitetura mais robusta.

2.1.2 Feedforward Neural Networks

Feedforward neural networks são redes neurais artificiais com multicamadas de neurônios capazes de aproximar funções não-lineares de um espaço dimensional para outro desde que seja fornecido a quantidade necessária de neurônios e camadas ocultas para tal tarefa (GOODFELLOW *et al.*, 2016, HORNIK *et al.*, 1989). As *feedforward neural networks* também são conhecidas como *multilayer perceptrons* (MLP). Porém, BISHOP (2006) destaca que o termo é impróprio pois um modelo com múltiplos perceptrons caracterizaria um modelo não-linear e não-contínuo, quando na verdade as *feedforward neural networks* representam um modelo não-linear porém contínuo. Afim de simplificar o texto, deste ponto em diante mencionaremos as *feedforward neural networks* como rede neurais multicamadas ou simplesmente redes neurais.

As redes neurais multicamadas são formadas por uma camada de entrada, uma camada de saída e pelo menos uma camada oculta, onde cada camada possui sua própria quantidade de neurônios. O modelo mais simples de uma rede neural funciona da seguinte forma:

1. a rede neural recebe através da camada de entrada um vetor x (chamado também de *features*). As unidades da camada de entrada representam então os valores do vetor de *features* x.

2. é aplicado uma sequência de transformações matemáticas através de suas camadas ocultas (como apresentado na equação 2.1).

3. a camada de saída retorna um valor y como resultado.

Cada camada de uma rede neural produzirá uma saída que servirá de entrada para a camada seguinte, até que o final da rede neural seja alcançado. A figura 2.2 representa o modelo básico de uma rede neural com múltiplas camadas.

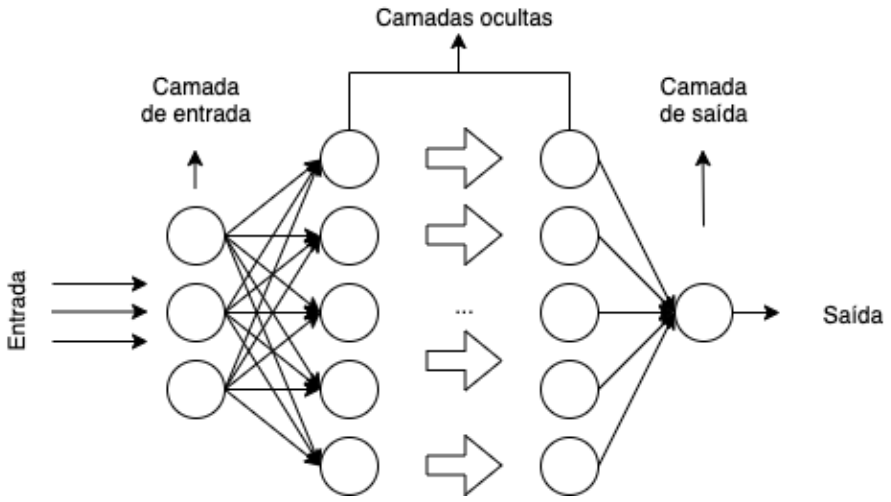


Figura 2.2: Representação gráfica de uma rede neural multi-camadas

A quantidade de camadas de uma rede neural pode ser denominada como profundidade (*depth*), terminologia que deu origem ao nome da área de estudos conhecida como *deep learning* (GOODFELLOW *et al.*, 2016).

Devido à dimensionalidade dos parâmetros a serem ajustados, é inviável a atualização manual dos pesos w em uma rede neural em busca da melhor solução possível. Para isso, precisamos utilizar algoritmos de otimização que buscam minimizar o erro da rede neural a cada passo do treinamento. Precisamos então definir uma função de custo (*loss function*) para usar no treinamento. A função de custo calcula a representação do erro entre o valor predito na saída da rede neural e o valor esperado e, logo, temos como objetivo minimizar o erro calculado pela função. A escolha da

função de custo dependerá do tipo de tarefa que a rede neural se propõe a aprender. As funções *mean squared error* (MSE) e *root mean squared error* (RMSE) são utilizadas geralmente em modelos de regressão. Essa família de funções porém geram resultados insatisfatórios pois sofrem com a saturação do gradiente e aprendizado lento (GOODFELLOW *et al.*, 2016). A família de funções *cross-entropy* é utilizada como uma alternativa mais robusta e eficiente, e a escolha da função dependerá de qual tipo de representacão temos na saída da rede neural (GOODFELLOW *et al.*, 2016, SIMARD *et al.*, 2003). Por exemplo, em tarefas de classificação binária podemos utilizar a função *binary cross-entropy* (BCE) e em tarefas de classificação multi-classes utilizamos a *categorical cross-entropy* (CCE). Por fim, com a função de custo definida, utilizamos os algoritmos de *backpropagation* e gradiente descendente no processo de otimização.

O algoritmo de *backpropagation* que redes neurais utilizam nos dias atuais foi primeiramente publicado na década de 70 (LINNAINMAA, 1970, 1976), apesar do estudo original não focar seu uso em redes neurais. Estudos posteriores exploraram o uso do *backpropagation* em redes neurais e ajudaram a consolidar o seu uso para a otimização dos pesos (LECUN, 1985, NIELSEN, 1989, RUMELHART *et al.*, 1986, WERBOS, 1974, 1981). O *backpropagation* é encarregado de calcular o gradiente da função de custo, partindo da última camada da rede neural em direção à primeira. O gradiente calculado é utilizado no algoritmo de otimização, o gradiente descendente.

O gradiente descendente é um algoritmo de otimização executado de forma iterativa com o objetivo de minimizar uma função alvo, de forma que a cada iteração o algoritmo mova-se na direção oposta do gradiente da função a ser minimizada (RUDER, 2016). Em redes neurais, a função alvo da minimização é a função de custo. Uma das características do algoritmo de gradiente descendente é que a solução ótima não é garantida, ou seja, o algoritmo se encarrega de fornecer um mínimo local (RUDER, 2016). A taxa de aprendizado (ou *learning rate*) é um hiperparâmetro numérico utilizado para controlar a taxa que o valor do gradiente influenciará no ajuste dos parâmetros. Em outras palavras, a taxa de aprendizado influencia no tamanho do passo a ser dado no ajuste dos parâmetros (RUDER, 2016). Definir uma taxa de aprendizado adequada é importante não só para definir a convergência do modelo, mas também para que o aprendizado não fique necessariamente estagnado em uma solução mínima local.

Uma das variantes do algoritmo de gradiente descendente é o *stochastic gradient descent*, onde o gradiente calculado sofre uma atualização de valor para cada amostra de treinamento no conjunto de dados. A estratégia de utilizar todos os exemplos do conjunto de treinamento para atualização do gradiente é uma abordagem ineficaz e redundante (BISHOP, 2006, RUDER, 2016). Já a versão do algoritmo chamada de *mini-batch gradient descent* computa o gradiente atualizando seu valor para todo

mini-batch de exemplos de treinamento. RUDER (2016) lista as vantagens do *mini-batch gradient descent*, dentre elas ter uma convergência mais estável e ser adaptável para utilizar diferentes otimizações dos *frameworks* de *deep learning*.

O cálculo do gradiente requer que as operações matemáticas envolvidas no processo de aprendizado da rede neural sejam diferenciáveis. Vimos que o *perceptron* utiliza a função degrau como função de ativação. A função degrau não é diferenciável em todos os pontos exatamente porque possui uma descontinuidade quando x se aproxima de zero. Além disso, qualquer valor de derivada para os outros pontos é exatamente zero. Portanto, é inviável utilizá-la em uma rede neural com otimização baseada em gradiente descendente. Há diversas alternativas de funções de ativação que podem ser utilizadas para substituir a função degrau. A seção 2.3 aborda com mais detalhes as funções de ativação em redes neurais.

Além dos conceitos abordados até o presente momento, uma questão importante dentre do escopo de redes neurais é a escolha da arquitetura. Diversos modelos se destacaram pela performance em determinadas tarefas. Esse é o caso por exemplo das redes neurais recorrentes em tarefas que envolvem processaremento de dados sequenciais, e das redes neurais convolucionais, abordadas na sub-seção 2.1.3, em tarefas de visão computacional.

2.1.3 Redes Neurais Convolucionais

As redes neurais convolucionais, em inglês *convolucional neural networks* (CNN), são um tipo de arquitetura de rede neural. As CNNs são construídas a partir dos mesmo conceitos fundamentais de aprendizado de uma rede neural, porém as estratégias arquiteturais de uma CNN são diferentes das redes neurais tradicionais, como abordaremos nesta seção. GOODFELLOW *et al.* (2016) define esse tipo de rede neural como uma especialização capaz de lidar com dados topologicamente representados em forma de grade, como por exemplo imagens (grade em duas dimensões) e séries temporais (grade em uma dimensão).

A primeira abordagem mais próxima de uma rede neural convolucional é conhecida como *Neocognitron* (FUKUSHIMA, 1980). Baseada em uma área presente no córtex visual de gatos, a rede *Neocognitron* possui diversos conceitos arquiteturais que são implantadas em redes neurais convolucionais modernas até o presente momento (SCHMIDHUBER, 2014). Mas, apesar da sua semelhança com as CNNs atuais, a *Neocognitron* não utilizava o algoritmo de *backpropagation* no seu processo de aprendizado. Um estudo posterior consolidou o uso do *backpropagation* e introduziu formalmente o conceito das redes neurais convolucionais (LECUN *et al.*, 1989, LECUN *et al.*, 1998). O emprego de técnicas como *backpropagation* em redes neurais convolucionais ajudaram a consolidar bons resultados em conjunto de dados de

imagens com especializações posteriores na arquitetura (SCHMIDHUBER, 2014).

As CNNs foram desenvolvidas primeiramente com o intuito de classificar imagens e muitos aspectos do seu funcionamento podem ser explicados ao levarmos em consideração como a rede processa uma imagem como entrada. Três características da sua arquitetura são fundamentais para entendermos como as redes convolucionais funcionam: campos receptivos locais, pesos compartilhados e *pooling* (BISHOP, 2006).

O diagrama da figura 2.2 exibe como a camada de entrada está interligada à todas as unidades da primeira camada oculta da rede convolucional. Semanticamente, isto significa que cada *feature* de entrada será processada pela camada oculta sem levar em consideração características como a proximidade de elementos em sub-regiões na entrada. Por exemplo, caso o dado de entrada seja uma imagem, os *pixels* mais próximos entre si possuem um significado semântico que podem descrever alguma característica presente na imagem. BISHOP (2006) lembra de que diversos algoritmos na área de visão computacional fazem uso do conceito de extração de características locais em sub-regiões da imagem, e o mesmo conceito é adotado nas camadas convolucionais de uma CNN. Este é o primeiro conceito fundamental de uma rede convolucional a descrevermos nesta seção: campos receptivos locais. A figura 2.3 mostra como alguns neurônios da camada de convolução estão conectados aos dados de entrada de forma local. Nesta figura temos uma imagem de entrada 4x4 *pixels*, onde cada quadrado representa um pixel, e campos receptivos definidos pelo tamanho de 2x2 *pixels*. Para cada sub-região formada por esses campos, os *pixels* estarão conectados à somente um neurônio na camada de convolução. Com isso, neste exemplo temos uma camada de convolução de 3x3 neurônios, onde cada neurônio possui quatro conexões de entrada de *pixels* da imagem. Desse modo, neurônios podem aprender se uma determinada característica está presente naquela sub-região conectada.

O conjunto de 3x3 neurônios na figura 2.3 é chamado de mapa de características (ou *feature map*). Os *feature maps* são uma das dimensões de uma camada convolucional. Cada *feature map* de uma rede convolucional é treinada para aprender alguma característica do dado de entrada, especializando assim seus neurônios a identificarem características locais na entrada. Podemos adicionar quantos *features maps* quisermos na camada de convolução, onde cada *feature map* será responsável por tentar aprender algum tipo de padrão de entrada. A figura 2.4 mostra as dimensões de uma camada de convolução. A largura e a altura representam a quantidade de neurônios presentes por dimensão, e a profundidade a quantidade de *feature maps*.

As imagens podem ser definidas por canais de cores como RGB ou CMYK. Caso o dado de entrada possua mais de um canal, o conceito de campo receptivo

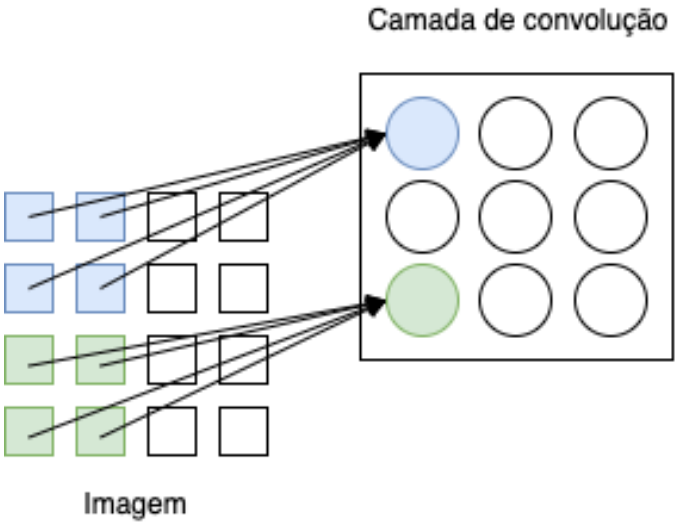


Figura 2.3: Conexões locais em redes convolucionais

local funciona da mesma forma: cada neurônio é conectado à uma sub-região da imagem e consequentemente seus canais de profundidade. Os *feature maps* são chamados de canais de saída (*output channels*) em algumas *frameworks* de *deep learning*, como uma alusão ao dado de entrada. Com essa analogia podemos entender que cada camada de convolução subsequente da rede tratará semanticamente a saída da camada anterior da mesma forma: um dado organizado em formato de grade com n canais.

O segundo conceito fundamental de uma rede convolucional é o de pesos compartilhados. Descrevemos nos parágrafos anteriores como neurônios estão conectados localmente à uma região da imagem. Os pesos de cada conexão são compartilhados entre os neurônios de um mesmo *feature map*. Isso significa que na figura 2.3 os pesos utilizados na operação de convolução para cada neurônio serão os mesmos. Conhecido como *kernel size*, as dimensões do campo local dos neurônios definirá a quantidade de pesos que a rede deve operar na camada de convolução. Caso tenhamos um *kernel size* de dimensões 2x2, significa que teremos 4 pesos mais 1 parâmetro de viés, totalizando 5 parâmetros por *feature map*. As duas principais vantagens de se fazer uso do conceito de pesos compatilhados são (BISHOP, 2006):

1. redução do numero total de parâmetros que a rede neural necessita ajustar durante o treinamento;

2. invariância no valor de saída da rede em caso de translações ou distorções na imagem de entrada.

Além de definirmos o *kernel size*, outro hiperparâmetro da rede convolucional que define o formato de saída de uma camada é o modo que o *kernel* desliza pelo dado de entrada. Há duas características que definem esse comportamento: *padding*

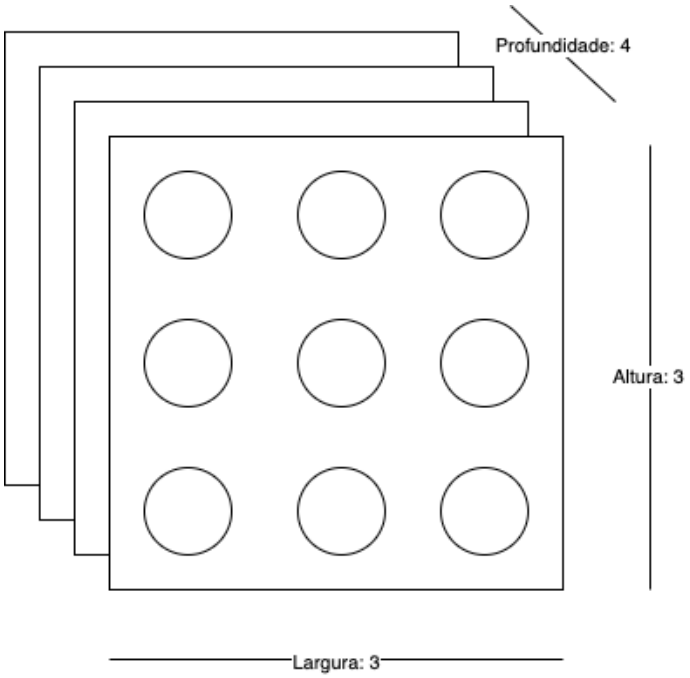


Figura 2.4: Dimensões em uma camada de convolução

e *stride*, ambos numéricos. O *padding* define se o dado entrada terá bordas e qual será o tamanho dessas bordas. Caso uma borda seja adicionada, é como se estivéssemos aumentando as dimensões largura e altura da imagem de entrada. O *stride* define de que modo o *kernel* se moverá pela imagem definindo os campos locais. Um *stride* com valor 1 significa que o *kernel* deslizará a cada um pixel formando os campos locais. Caso o valor seja igual à 2, o *kernel* pulará dois *pixels* sempre que for deslizando pela imagem ao formar os campos locais.

As camadas de convolução recebem esse nome devido ao tipo de operação matemática que os neurônios executam para calcular sua saída. A equação 2.3 descreve a operação matematicamente conhecida como convolução:

$$conv(i,j) = b + \sum_{c=0}^{K} \sum_{d=0}^{K} w_{c,d} * x_{i+c,j+d} \tag{2.3}$$

Onde i e j representam respectivamente coluna e linha do pixel na imagem de entrada, b o viés, K o valor do *kernel size*, w os pesos compartilhados no *kernel*, e x o valor dos *pixels* nas posições $i+c, j+d$. A equação 2.4 representa o valor de saída do neurônio após aplicada uma função de ativação ao resultado da convolução:

$$y(i,j) = f(conv(i,j)) \tag{2.4}$$

Onde *f(·)* é a função de ativação aplicada.

A terceira característica fundamental de uma rede convolucional é a camada de *pooling*. O papel dessa camada é realizar uma sub-amostragem do resultado da

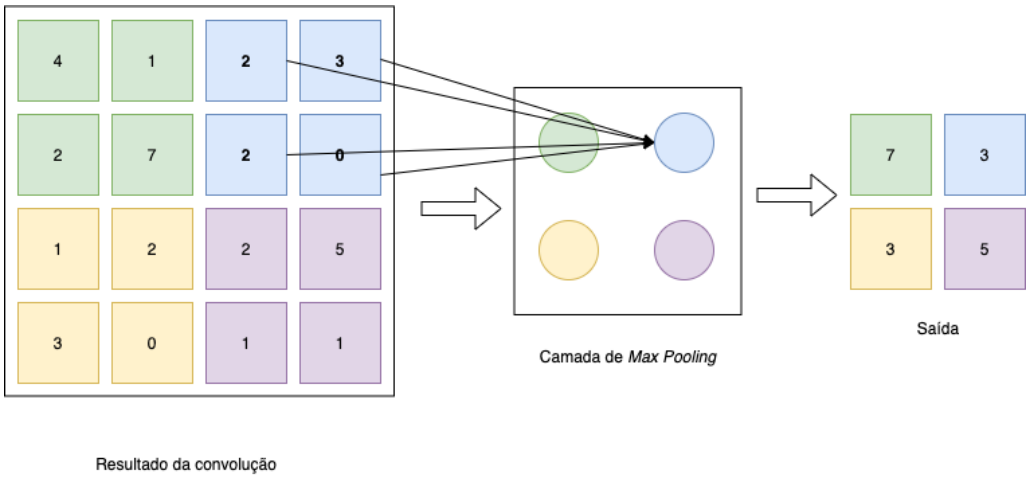


Figura 2.5: Cada unidade da camada de *max-pooling* é responsável por computar o valor máximo de uma região do resultado da camada convolucional.

camada de convolução. Cada *feature map* da camada de convolução passará pelo processo de sub-amostragem de pequenos campos receptivos. O resultado de saída das unidades da camada de *polling* será definido pelo tamanho do campo receptivo, a forma que o campo desliza pelo *output* de cada *feature map* e pelo tipo de operação de amostragem. O tamanho do campo e o modo em que ele desliza pelo plano de saída dos *feature maps* são definidos pelos hiper-parâmetros *kernel size*, *padding* e *stride*, e funcionam da mesma forma como definimos anteriormente para o modo que a camada de convolução processa a entrada. Diversos tipos de operações de sub-amostragem são validamente utilizados nesta camada. As camadas de sub-amostragem do tipo *max pooling* por exemplo geram o valor máximo do campo local como resultado da unidade de *pooling*. Já as camadas do tipo *average pooling* geram a média aritmética como saída. A figura 2.5 exemplifica como uma camada do tipo *max pooling* funciona.

As redes convolucionais desenvolveram papel de destaque na área de *deep learning*. Além de aplicações comerciais, diversos modelos de redes convolucionais conquistaram prêmios em diversas competições de variados temas, como reconhecimento de padrões, detecção de objetos e segmentação de imagens (GOODFELLOW *et al.*, 2016, SCHMIDHUBER, 2014). Alguns destes modelos de sucesso foram: LeNet (LECUN *et al.*, 1998), AlexNet (KRIZHEVSKY *et al.*, 2012), ZF Net (ZEILER e FERGUS, 2013), VGG Net (SIMONYAN e ZISSERMAN, 2014), GoogLeNet (SZEGEDY *et al.*, 2015) e ResNet (HE *et al.*, 2015).

2.2 Generative Adversarial Networks

2.2.1 Definição do modelo GAN

As *Generative Adversarial Networks* (GAN) (GOODFELLOW *et al.*, 2014) são um modelo generativo de aprendizado de máquina. Essa arquitetura propõe que duas redes neurais, denominadas discriminador e gerador, sejam treinadas de forma adversária (GOODFELLOW *et al.*, 2014, 2016). Enquanto a rede geradora deve tentar produzir amostras de itens que sejam similares às amostras da base de dados, a rede discriminadora tenta distinguir quais amostras foram geradas e quais amostras são exemplos reais da base de dados. Além disso, a rede geradora à princípio não possui visibilidade direta sobre os dados reais e recebe como entrada ruído gerado aleatoriamente. Desta forma, ambas as redes competem entre si durante o processo de aprendizado buscando aprimorar seus resultados, caracterizando o treinamento adversário. Uma possível analogia que podemos fazer para entender melhor este processo é imaginar que a rede geradora é um falsificador de itens, tentando gerar exemplos falsos, porém similares aos itens reais, e a rede discriminadora é um avaliador que busca identificar e diferenciar os itens reais dos artificialmente gerados. A figura 2.6 representa o modelo básico da arquitetura e funcionamento de uma GAN.

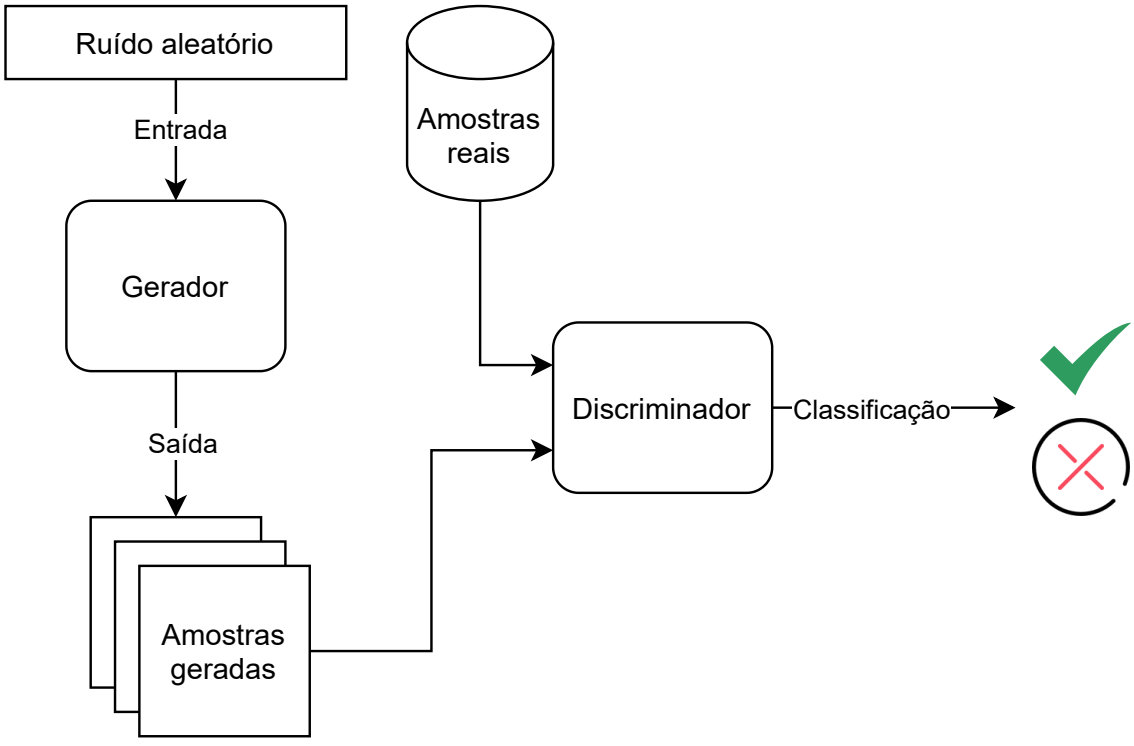


Figura 2.6: Esquema geral básico da arquitetura de redes adversárias (GAN)

A arquitetura GAN é inspirada no algoritmo *minimax* da teoria dos jogos (GOODFELLOW *et al.*, 2014), e seu treinamento consiste em encontrar, teoricamente, o equilíbrio de Nash entre as redes. No entanto, a busca pelo equilíbrio como forma

de treinamento das redes não é viável, fazendo com que o modelo não obtenha a convergência necessária à princípio (GOODFELLOW, 2015, SALIMANS *et al.*, 2016). Alternativamente, os algoritmos de gradiente descendente são utilizados como uma opção viável para o aprendizado das redes.

Para o treinamento de redes GAN, definimos p_g como sendo a distribuição das amostras geradas por $G(z;\theta_g)$, onde G é uma função diferenciável representada por uma rede neural (gerador) com parâmetros θ_g, e z é denomidado ruído de entrada. Definimos também $D(x;\theta_d)$ como sendo o resultado probabilístico de uma amostra x ter sido retirada do conjunto de dados real, ou seja, não sintetizada pela rede geradora. D é uma função diferenciável representada por uma rede neural (discriminador) com parâmetros θ_d, e x a amostra a ser avaliada. A rede discriminadora é então treinada para maximizar a probabilidade de acertar se a amostra é real ou falsa, enquanto a rede geradora é treinada para minimizar os acertos do discriminador, ou seja, produzir amostras cada vez mais fiéis aos da base de dados original. A função *minimax* $V(D,G)$ é então definida pela equação 2.5:

$$min_g max_d V(D,G) = \mathbb{E}_{x\sim p_{dados}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (2.5)$$

Dado o tempo e recursos necessários, o modelo de redes neurais adversárias pode em teoria atingir a solução ótima, obtendo assim um estimador para p_{dados}. Nesse cenário, o discriminador gera em média como resultado de saída $1/2$ para qualquer amostra, seja ela gerada ou real (GOODFELLOW *et al.*, 2014).

Os conceitos definidos até o momento caracterizam o modelo básico de redes GAN. No entanto, diversas arquiteturas foram posteriormente propostas utilizando diferentes estratégias com o intuito de alavancar o desempenho e a usabilidade das redes GAN em diferentes tipos de tarefas. Dois dos modelos precursores que impactaram os componentes de treinamento e qualidade dos itens gerados são os modelos CondGAN e DCGAN, discutidos respectivamente nas sub-seções 2.2.2 e 2.2.3.

2.2.2 Conditional Generative Adversarial Networks (CondGAN)

As *Conditional Generative Adersarial Networks* (CondGAN) (MIRZA e OSINDERO, 2014) são uma das arquiteturas de redes neurais adversárias. O modelo propõe a incorporação das classes do conjunto de dados ao treinamento, concatenando o dado da classe ao ruído fornecido como entrada para a rede geradora e ao exemplo a ser examinado pela rede discriminadora. Tal incorporação pode ser feita através da conversão da classe em um *one-hot encoding vector*, por exemplo.

A aplicação da estratégia condicional possibilita o controle de qual classe a rede geradora deverá gerar a amostra. Apesar de o objetivo da estratégia ser similar ao modelo InfoGAN (CHEN *et al.*, 2016), ambos os modelos se diferem pois o CondGAN adiciona uma nova dimensionalidade à entrada da rede geradora, enquanto o InfoGAN busca controlar a classe de saída da rede geradora através das variáveis do espaço latente do ruído. Devido à essa diferença essencial entre os dois modelos, podemos considerar que o modelo CondGAN representa uma solução mais direta à geração de amostras condicionadas, assumindo que as classes do conjunto de dados são uma informação conhecida na construção do problema.

O treinamento de uma GAN condicional é similar ao de uma não-condicional, com apenas uma pequena mudança na distribuição de probabilidade das redes geradora e discriminadora. Definimos então a função objetivo de uma CondGAN baseada em um jogo *minimax* como sendo:

$$min_g max_d V(D,G) = \mathbb{E}_{x \sim p_{dados}(x)}[\log D(x|y)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z|y)))] \quad (2.6)$$

Note que diferentemente da equação 2.5, na equação 2.6 as probabilidades D e G são condicionais, representando a nova dimensionalidade de entrada nas redes introduzida pelo modelo. As figuras 2.7 e 2.8 representam o diagrama do modelo condicional para as redes geradora e discriminadora, respectivamente.

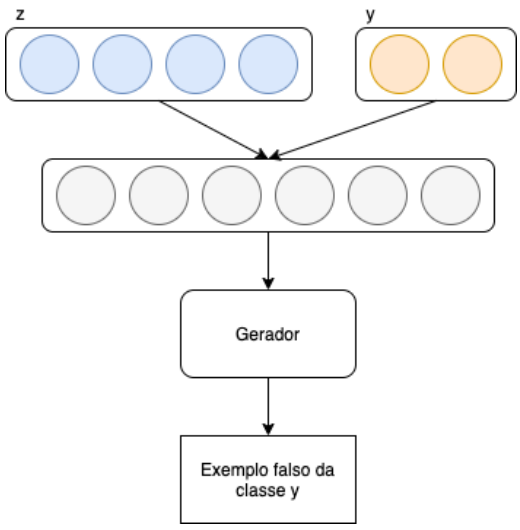


Figura 2.7: Representação do fluxo na rede geradora no modelo CondGAN, onde z é o ruído de entrada e y é a classe representada por um *one-hot vector*.

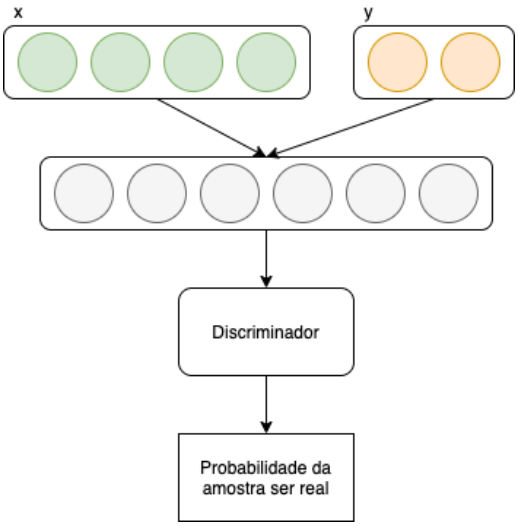


Figura 2.8: Representação do fluxo na rede discriminadora no modelo CondGAN, onde x é o vetor de *features* de entrada (por exemplo a intensidade dos pixels em uma imagem preto e branco) e y é a classe dessa amostra representada por um *one-hot vector*.

2.2.3 DeepConv Generative Adversarial Networks (DCGAN)

Redes convolucionais começaram a ser utilizadas em GANs a partir da proposta do modelo DCGAN (RADFORD *et al.*, 2015). Até então, as redes geradora e discriminadora eram compostas por *feedforward neural nets* multi-camadas. A combinação dos modelos trouxe uma nova abordagem ao campo de aprendizado não-supervisionado em tarefas de visão computacional, como destaca o autor do modelo.

Como visto na sub-seção 2.1.3, as CNN possuem um componente chamado *filtro*, que serve como uma componente de aprendizado de uma característica da imagem de entrada. RADFORD *et al.* destaca que os filtros das redes convolucionais de uma DCGAN de fato são capazes de identificar características intrínsecas ao dado de entrada, culminando no aperfeiçoamento da amostra gerada.

As camadas da rede discriminadora são camadas convolucionais, discutidas na sub-seção 2.1.3. A novidade está na rede geradora, que precisa atuar com um tipo de camada que realize o caminho inverso da decomposição da imagem (como é o caso das camadas convolucionais). Neste espaço, as camadas da rede geradora são camadas do tipo de-convolucional, também referenciadas como convolução transposta, responsáveis por realizar a aritmética oposta à das camadas convolucionais (ZEILER *et al.*, 2010). Estas camadas transformam o *feature map* para uma amostragem maior, ou seja, alteram as dimensões de entrada aumentando a amostragem nas dimensões de saída. A figura 2.9 representa um exemplo de operação de convolução transposta.

Além da arquitetura básica DCGAN, a proposta inicial recomenda certas abordagens para o desenvolvimento e treinamento de redes GANs convolucionais (RAD-

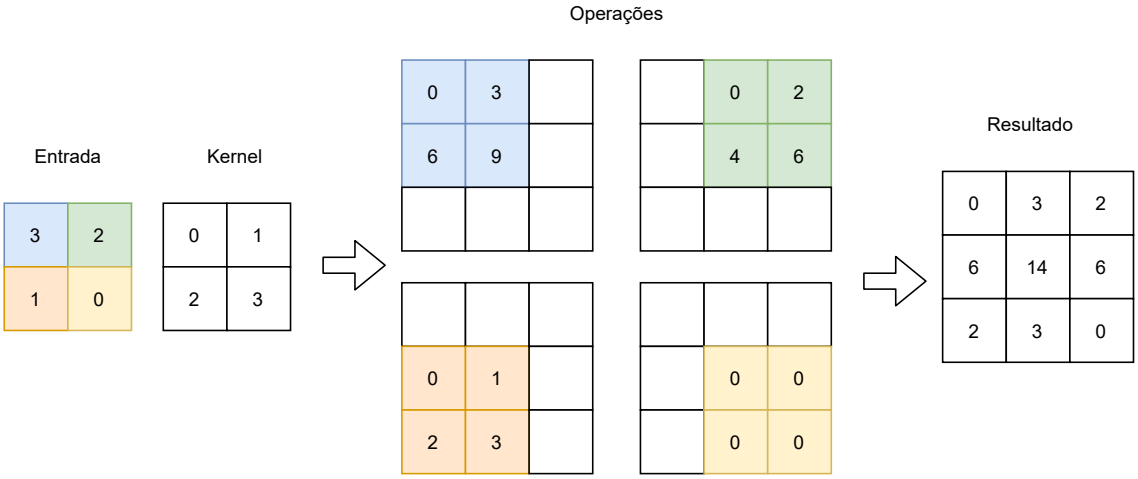


Figura 2.9: Exemplo de uma operação de de-convolução, com tamanho do *kernel* 2x2, *stride* 1 e *padding* 0.

FORD *et al.*, 2015), como:

1. utilizar *batch-normalization* durante o treinamento;
2. não utilizar camadas *fully-connected*;
3. utilizar a função ReLU como ativação nas camadas ocultas do gerador;
4. utilizar a função LeakyReLU como ativação nas camadas ocultas do discriminador;
5. utilizar convoluções com *stride* ao invés de camadas de *pooling*.

2.3 Funções de Ativação

2.3.1 Funções de ativação tradicionais

Em redes neurais, as funções de ativação são funções não-lineares diferenciáveis, tipicamente aplicadas em cada neurônio da rede, promovendo uma transformação numérica sobre a saída de cada unidade (BISHOP, 2006, GOODFELLOW *et al.*, 2016). É importante citar a importância do fato da função ser não-linear, o que dá ao modelo de redes neurais a capacidade de generalização necessária para encontrar soluções de problemas não linearmente separáveis. Além disso, idealmente a função deve ser diferenciável para que possamos utilizar os algoritmos de otimização baseados em gradiente visto na seção 2.1.

Função degrau

A função degrau aplicada no *perceptron* não é uma função de ativação utilizada em redes neurais na prática. Esta função além de possuir uma descontinuidade na origem, possui derivada igual à zero em todos os outros pontos. A figura 2.10 representa o gráfico da função degrau.

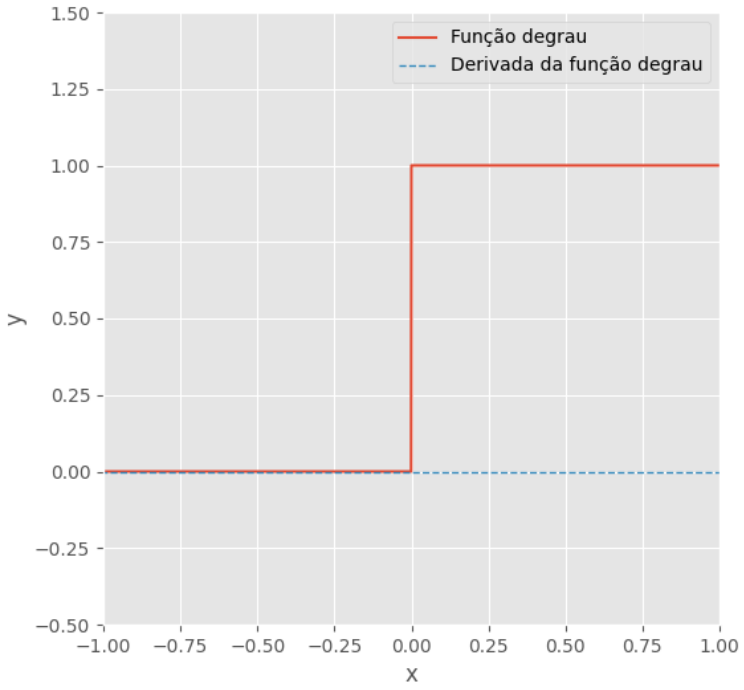


Figura 2.10: Gráfico da função degrau e sua derivada. Note que na origem $x = 0$ há uma descontinuidade. Além disso, o valor da derivada da função degrau em qualquer outro ponto diferente de zero, onde é indefinida, é igual à zero, tornando-se inviável o seu uso para a otimização usando gradiente.

Segundo BISHOP (2006), algumas das funções mais populares geralmente escolhidas para desempenhar o papel de ativação em redes neurais são as funções: logística, *softmax*, tangente hiperbólica e ReLU (*rectified linear unit*) (MAAS *et al.*, 2013).

Função logística

A função logística é uma função sigmoidal comumente utilizada na camada de saída de redes neurais aplicadas à problemas de classificação binária, onde o valor de saída representa a probabilidade $p(c = 1|x)$, sendo x o vetor de entrada da rede e c a classe (0 ou 1). Na figura 2.11 temos o gráfico da função logística e sua derivada, onde a função logística é definida pela seguinte equação:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

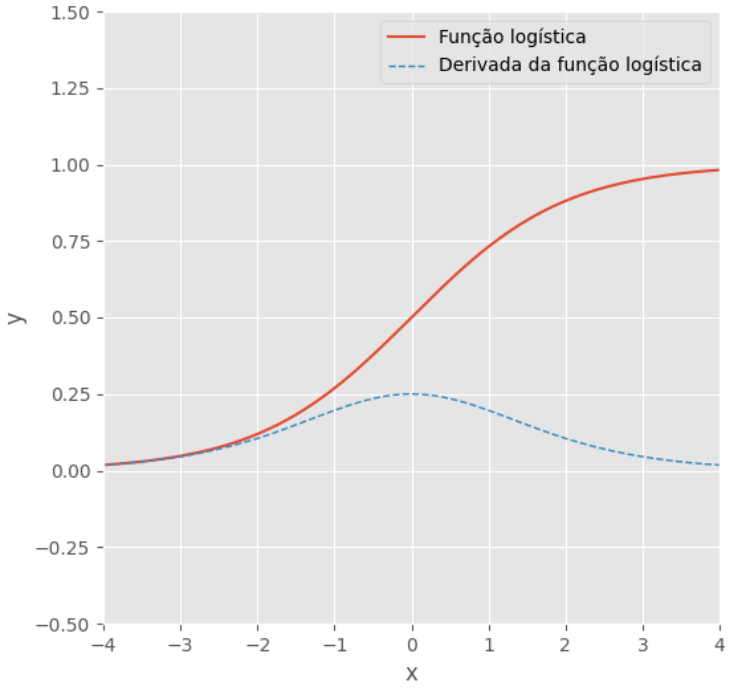


Figura 2.11: Função logística e sua derivada

A generalização da função logística para problemas de classificação com n classes mutualmente exclusivas é chamada de função *softmax*. A função *softmax* é normalmente utilizada na unidade de saída da rede neural de forma que a soma das probabilidades de cada classe baseado na entrada seja igual à 1, ou seja, criando uma distribuição de probabilidade onde somos capazes de escolher a classe com maior probabilidade e assim selecionar a classificação para cada amostra dentre n classes.

Função tangente hiperbólica

A função tangente hiperbólica ($tanh$) tem uma relação próxima à função logística, pois:

$$tanh(x) = 2g(2x) - 1 \tag{2.8}$$

Onde g é a função logística definida na equação 2.7.

A *tanh*, assim como a logística, também possui uma forma sigmoidal, porém seu intervalo de atuação no eixo vertical é assintoticamente limitado em -1 e 1. O uso

de funções sigmoidais em camadas ocultas da rede não é recomendado, mas quando necessário aconselha-se utilizar a função tangente hiperbólica ao invés da função logística (GOODFELLOW *et al.*, 2016). Substituindo o termo g na equação 2.8, obtemos a definição da função tangente hiperbólica:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.9}$$

Na figura 2.12 temos o gráfico da função tangente hiperbólica e sua derivada.

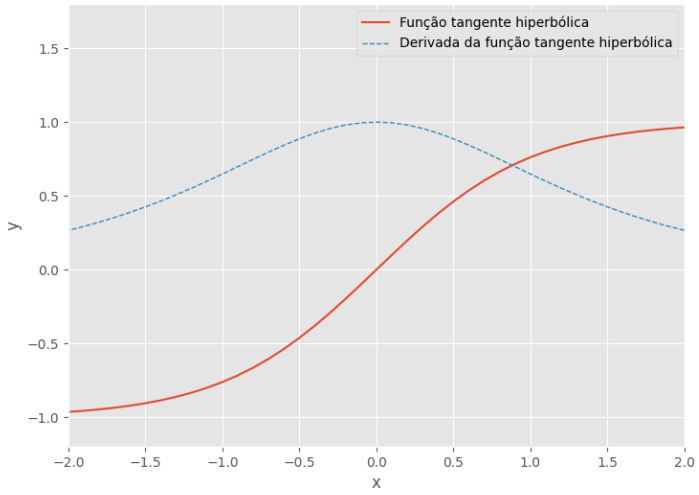


Figura 2.12: Função tangente hiperbólica e sua derivada

Função ReLU

A função ReLU é considerada a escolha inicial padrão, principalmente em *deep learning*, para neurônios das camadas ocultas (GOODFELLOW *et al.*, 2016). Apesar desta função não ser derivável quando $x = 0$, GOODFELLOW *et al.* (2016) explica que é uma situação delicadamente aceitável já que não se espera que o algoritmo de otimização alcance o ponto em que a função é exatamente igual à zero. A figura 2.13 representa o gráfico da função ReLU e sua derivada, onde a função ReLU é definida pela equação 2.10:

$$ReLU(x) = max(0, x) \tag{2.10}$$

Como podemos observar na figura 2.13, a derivada da função ReLU quando $x < 0$ é igual à zero. Esse fenômeno é conhecido na literatura como um tipo de gradiente minguante (*vanish gradient*) denominado *dying ReLU* (LU *et al.*, 2020, MISRA, 2020, TROTTIER *et al.*, 2016). Como discutimos anteriormente, esse cenário não é útil para o cálculo do gradiente, e com isso algumas outras funções de ativação

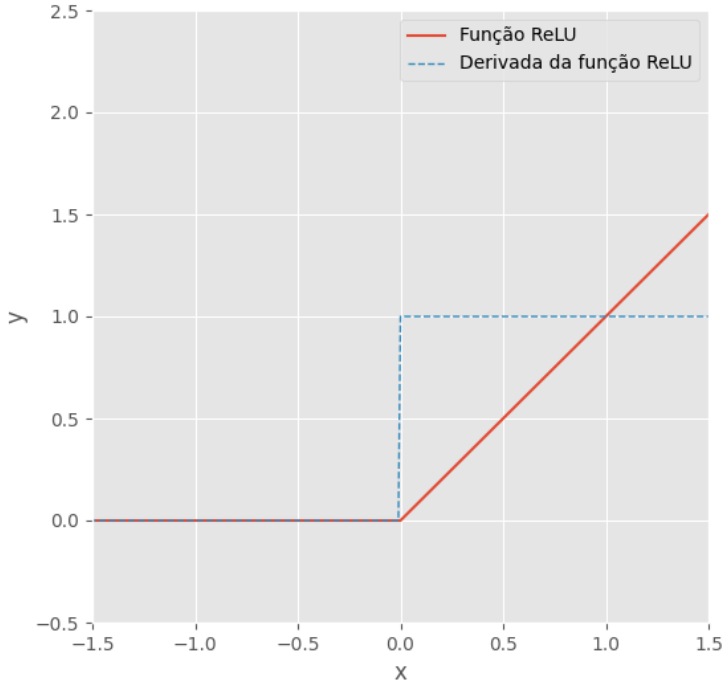


Figura 2.13: Gráfico da função ReLU e sua derivada

foram elaboradas para minimizar esse problema. As funções LeakyReLU (MAAS *et al.*, 2013) e Mish (MISRA, 2020), com formato similar à ReLU, são exemplos de funções que atacam o problema do gradiente igual à zero quando x é negativo.

Função LeakyReLU

A função LeakyReLU define que quando $x < 0$ seu valor é igual a $\alpha * x$, onde α é uma constante definida como hiperparâmetro do modelo, e quando $x \geq 0$ tem comportamente similar à ReLU. Na figura 2.14 temos o gráfico da função LeakyReLU e sua derivada, onde a função é definida pela equação:

$$LeakyReLU(\alpha, x) = \begin{cases} x & \text{se } x \geq 0, \\ \alpha * x & \text{caso contrário} \end{cases} \tag{2.11}$$

Função Mish

A função Mish é definida como uma função auto-regularizada e não-monotônica (MISRA, 2020). Experimentos em MISRA (2020) demostram que a função Mish tende a se igualar ou melhorar o desempenho do modelo comparado aos resultados do mesmo modelo utilizando ReLU. Inspirada na função Swish (RAMACHANDRAN

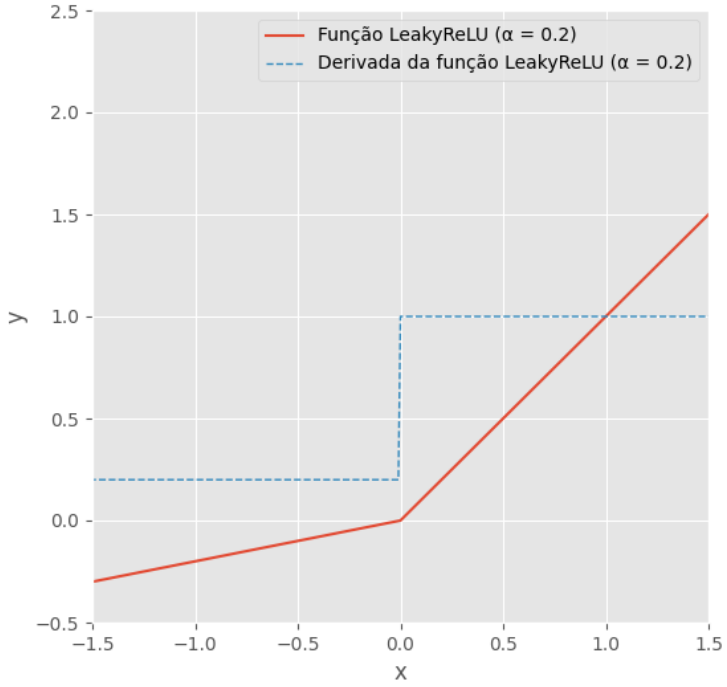


Figura 2.14: Gráfico da função LeakyReLU e sua derivada

et al., 2017), a função Mish também possui algumas propriedades que podem, teoricamente segundo MISRA (2020), explicar sua performance melhorada:

1. Continuamente diferenciável: a função Mish é diferenciável em todos os pontos.

2. *Self-gating*: assim como na função Swish, a função Mish possui a propriedade *self-gating*, onde a entrada é multiplicada pelo resultado de uma transformação não-linear aplicada à própria entrada.

3. *Self-regularizing*: ao observarmos a primeira derivada da função Mish na equação 2.12, podemos identificar que o termo $\Delta(x)$ atua, especulativamente, como um termo pré-condicionador do gradiente melhorando a convergência do modelo.

$$\begin{aligned} \text{Mish}'(x) &= \text{sech}(\text{softplus}(x))^2 * x * \text{sigmoid}(x) + \frac{\text{Mish}(x)}{x} \\ &= \Delta(x) * \text{swish}(x) + \frac{\text{Mish}(x)}{x} \end{aligned} \tag{2.12}$$

Onde:

$$\begin{aligned} \text{softplus}(x) &= \ln(1 + e^x) \\ \text{swish}(x) &= x * \text{sigmoid}(x) \\ \Delta(x) &= \text{sech}(\text{softplus}(x))^2 \end{aligned} \tag{2.13}$$

Na figura 2.15 temos o gráfico da função Mish e sua derivada, onde a função é definida pela equação 2.14:

$$\begin{aligned} Mish(x) &= x * \tanh(\text{softplus}(x)) \\ &= x * \tanh(\ln(1 + e^x)) \end{aligned} \tag{2.14}$$

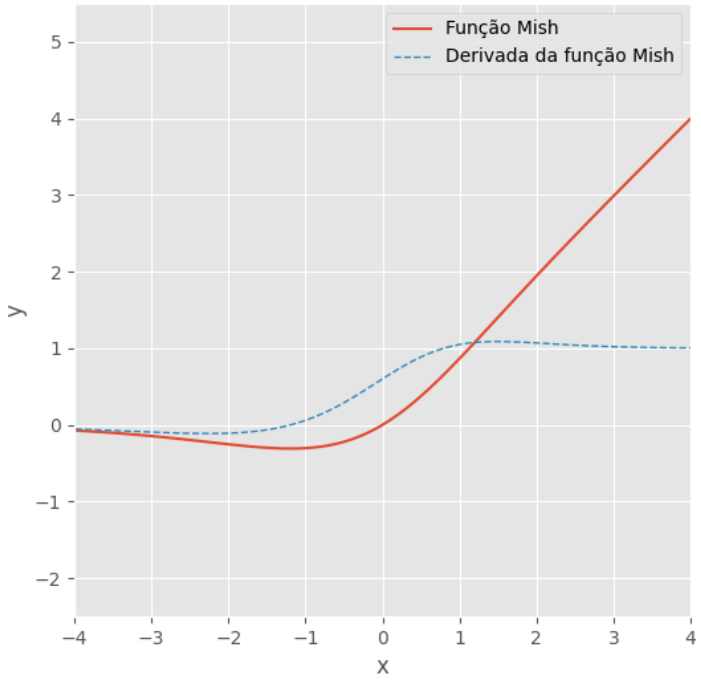


Figura 2.15: Gráfico da função Mish e sua derivada

2.3.2 Funções de ativação adaptativas

As alternativas listadas até o momento são funções de ativação pré-definidas, ou seja, elas não possuem parâmetros a serem otimizados pela rede e permanecem com sua definição estática durante todo o treinamento. Apesar de simples e computacionalmente leves, a abordagem de utilizar ativações fixas e pré-definidas pode significar uma limitação no desempenho do modelo (AGOSTINELLI *et al.*, 2015). Uma classe de funções conhecida como funções de ativação adaptativas propõe exatamente uma definição dinâmica através de parâmetros a serem otimizados pelo próprio processo de treinamento da rede neural.

Função APL

Os primeiros estudos que propuseram que as unidades de ativação deveriam ser aprendidas pelo modelo de aprendizado se baseavam em algoritmos evolutivos (YAO,

1999) e no uso de um único parâmetro compartilhado pelas unidades de ativação (TURNER e MILLER, 2014). O trabalho de AGOSTINELLI *et al.* (2015) introduziu a ideia de utilizar parâmetros adaptativos dedicados para cada uma das unidades de ativação adaptativas. Desta forma, cada função de ativação adaptativa utilizada no modelo tem seus próprios parâmetros a serem ajustados. Esta estratégia foi demonstrada através da função proposta denominada *Adaptive Piecewise Linear Units* (APL), onde seu uso foi capaz de melhorar o desempenho de redes neurais em diversos conjuntos de dados (AGOSTINELLI *et al.*, 2015). A função APL é definida pela seguinte equação 2.15:

$$f_i(x) = \max(0, x) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s) \tag{2.15}$$

Onde f_i é a função de ativação no *i-ésimo* neurônio, S é um hiperparâmetro que define o número de articulações, a e b parâmetros a serem ajustados pelo processo de aprendizado da rede. As variáveis a e b controlam, respectivamente, a inclinação da reta e o ponto da articulação. A figura 2.16 representa como a função APL seria se os parâmetros fossem $a = 0.2$ e $b = 0$.

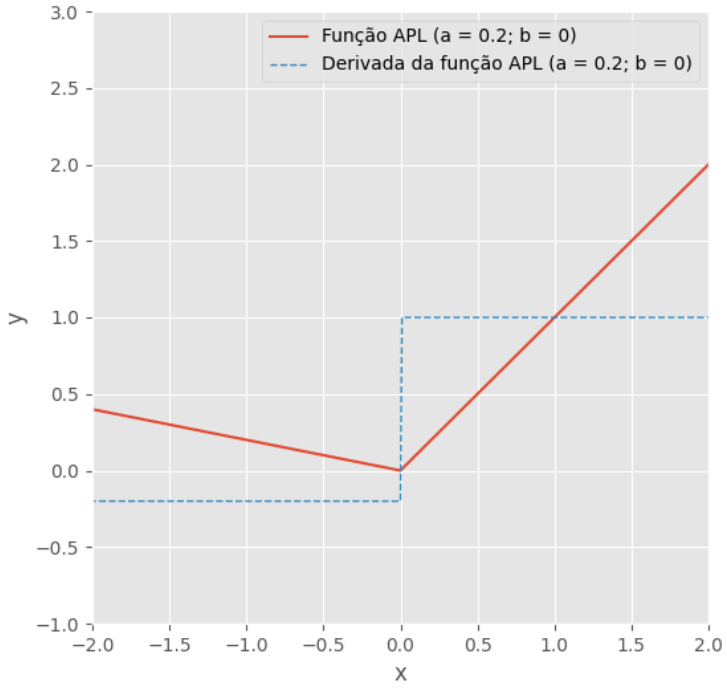


Figura 2.16: Gráfico da função APL e sua derivada com parâmetros $a = 0.2$ e $b = 0$

Funções com penalização hiperbólica

Além da função APL, outras funções adaptativas vêm sendo propostas para uso em redes neurais. A família de funções baseadas na técnica de penalização hiperbólica (XAVIER, 1982) foi primeiramente empregada em redes neurais sem a atualização automática dos pesos pelo algoritmo de gradiente descendente (MIGUEZ, 2012, XAVIER e XAVIER, 2016). O uso do algoritmo de aprendizado para atualização dos parâmetros das funções de ativação baseadas em penalização hiperbólica permitiu identificar quais obtiveram melhor desempenho em uma rede multi-camadas (CANALLI, 2017):

1. Função bi-hiperbólica simétrica adaptativa (BHSA)
2. Função bi-hiperbólica assimétrica adaptativa (BHAA)
3. Função suavização hiperbólica da ReLU (SH-ReLU)

SH-ReLU

A função SH-ReLU (CANALLI, 2017) é uma versão da ReLU suavizada utilizando a técnica de penalização hiperbólica. A suavização proposta ataca, assim como as funções Mish e LeakyReLU citadas anteriormente, o problema de *Dying ReLU*, buscando minimizar o impacto que o problema do gradiente minguante pode causar durante o treinamento da rede. A SH-ReLU possui um parâmetro adaptativo, o τ, que é otimizado durante o treinamento buscando melhorar o desempenho do modelo através da adaptação da própria função de ativação. Na figura 2.17 temos o gráfico da função SH-ReLU com diferentes valores para τ e suas respectivas derivadas, onde a função SH-ReLU é definida pela equação:

$$\text{SH-ReLU}(x, \tau) = \frac{x + \sqrt{x^2 + \tau^2}}{2} \tag{2.16}$$

BHSA

A função bi-hiperbólica simétrica adaptativa (BHSA), inicialmente formulada em XAVIER (2005), vem sendo desde então estudada por seus impactos durante o treinamento em redes neurais (CANALLI, 2017, MIGUEZ, 2012, XAVIER, 2005). Trata-se de uma função de formato sigmoidal parametrizada através de λ e τ baseada no método de penalização hiperbólica XAVIER (1982). Além da versão tradicional, a função BHSA possui ainda uma versão escalada que atua no intervalo $[-1, 1]$ (CANALLI, 2017).

Os dois parâmetros treináveis desta função controlam duas características distintas do formato da curva de ativação. O parâmetro λ controla a inclinação da

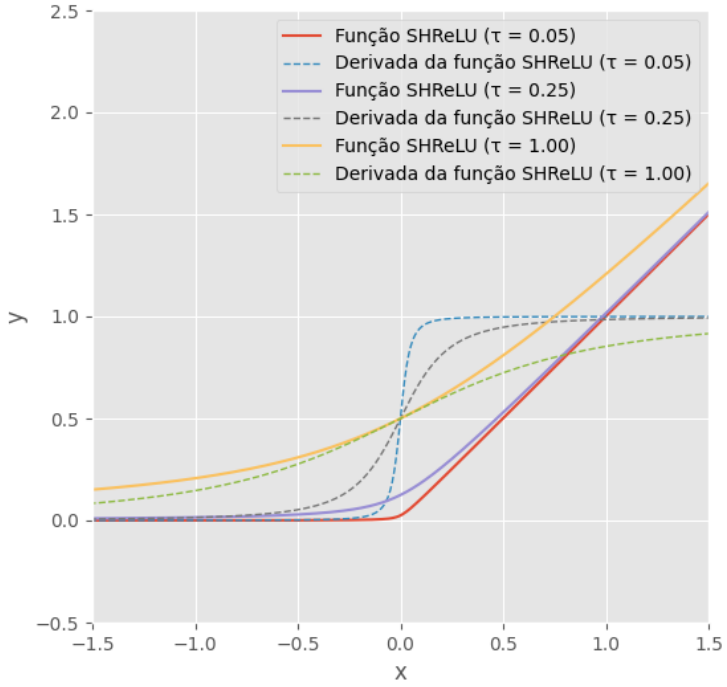


Figura 2.17: Gráfico da função *SH-ReLU* e suas derivadas com parâmetros $\tau = 0.05, 0.25, 1$

reta de transição que ascende da parte inferior da sigmoide para a parte superior. O parâmetro τ controla a intensidade dos cantos curvados da sigmoide (superior e inferior, de forma simétrica). Na figura 2.18 podemos observar algumas possíveis curvas da BHSA quando variamos seus parâmetros treináveis.

A função BHSA, atuante no intervalo $[-1, 1]$, é definida pela equação 2.17. Para obter a versão tradicional atuante no intervalo $[0, 1]$ basta substituirmos os termos $1/2\lambda$ por $1/4\lambda$. Podemos simplificar o entendimento da equação 2.17 pela simples subtração de duas hipérboles h_1 e h_2, representadas respectivamente pelas equações 2.18 e 2.19.

$$\text{BHSA}(x, \lambda, \tau) = \sqrt{\lambda^2 \left(x + \frac{1}{2\lambda}\right)^2 + \tau^2} - \sqrt{\lambda^2 \left(x - \frac{1}{2\lambda}\right)^2 + \tau^2} \tag{2.17}$$

$$h_1(x, \lambda, \tau) = \sqrt{\lambda^2 \left(x + \frac{1}{2\lambda}\right)^2 + \tau^2} \tag{2.18}$$

$$h_2(x, \lambda, \tau) = \sqrt{\lambda^2 \left(x - \frac{1}{2\lambda}\right)^2 + \tau^2} \tag{2.19}$$

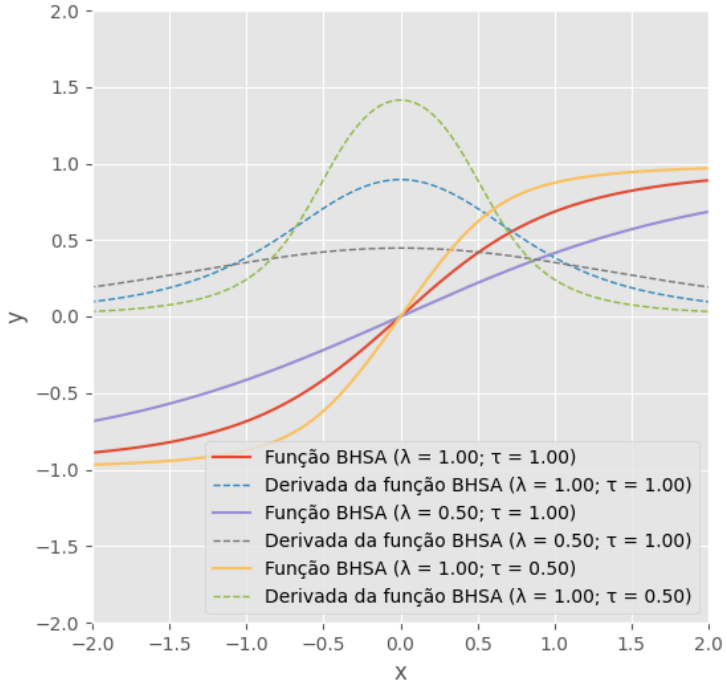


Figura 2.18: Gráfico da função BHSA e suas derivadas com parâmetros: $(\lambda = 1; \tau = 1)$, $(\lambda = 0.5; \tau = 1)$ e $(\lambda = 1; \tau = 0.5)$.

BHAA

A função BHSA pode ser entendida como uma versão especializada da função Bi-hiperbólica assimétrica adaptativa (BHAA) (XAVIER, 2005), uma versão mais generalizada. Na função BHAA o parâmetro único τ dá lugar à dois novos parâmetros: τ_1 e τ_2. Com essa substituição, o formato sigmoidal passa a não ser mais simétrico dependendo da relação entre τ_1 e τ_2. Quando $\tau_1 > \tau_2$, a curva superior da sigmóide recebe uma distorção em formato de cume, enquanto quando $\tau_1 < \tau_2$ a curva inferior da sigmóide recebe uma distorção em formato de vale. Quando $\tau_1 = \tau_2$, temos exatamente o mesmo caso da versão simétrica. O parâmetro λ permanece com a mesma característica descrita na função BHSA. A figura 2.19 representa como a variação destes parâmetros afeta a curva da função e suas derivadas.

Assim como a BHSA, a função BHAA possui uma versão escalada atuante no intervalo $[-1, 1]$ formulada em CANALLI (2017). A equação 2.20 define a função BHAA atuante no intervalo $[-1, 1]$, enquanto que para obter a versão atuante em $[0, 1]$ basta substituírmos os termos $1/2\lambda$ por $1/4\lambda$.

$$\text{BHAA}(x, \lambda, \tau_1, \tau_2) = \sqrt{\lambda^2 \left(x + \frac{1}{2\lambda}\right)^2 + \tau_1^2} - \sqrt{\lambda^2 \left(x - \frac{1}{2\lambda}\right)^2 + \tau_2^2} \tag{2.20}$$

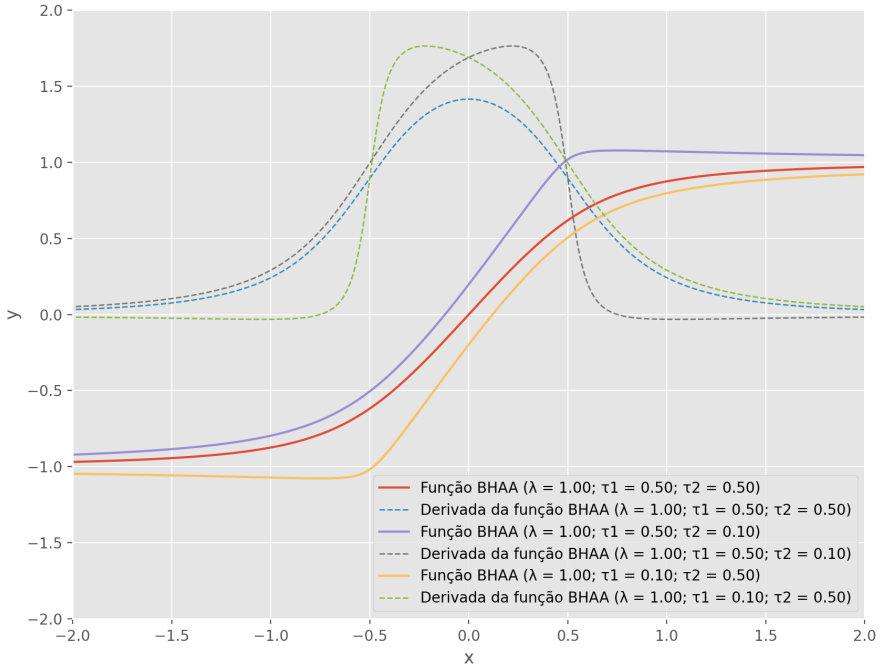


Figura 2.19: Gráfico da função BHAA e suas derivadas com parâmetros: $(\lambda = 1; \tau_1 = 0.5; \tau_2 = 0.5)$, $(\lambda = 0.5; \tau_1 = 0.5; \tau_2 = 0.5)$, $(\lambda = 1; \tau_1 = 0.5; \tau_2 = 0.1)$ e $(\lambda = 1; \tau_1 = 0.1; \tau_2 = 0.5)$

2.3.3 Gradiente minguante

A questão do gradiente minguante (*vanish gradient*) foi citado anteriormente como um complicador para o treinamento de redes neurais. Nesta sub-seção discutiremos um pouco sobre esse problema, como ele surge e como podemos, se não evitar, pelo menos reduzir seu impacto no processo de aprendizado da rede.

Na literatura disponível, diversos textos destacam o impacto que o gradiente minguante pode causar durante o treinamento de uma rede neural. Essencialmente, o problema de gradiente minguante ocorre quando o valor do gradiente calculado é consideravelmente pequeno fazendo que a atualização dos pesos na rede não impacte o resultado atual que a rede está produzindo, ou seja, causando estagnação no processo de aprendizado. (BENGIO *et al.*, 1994, BISHOP, 2006, GOODFELLOW *et al.*, 2016)

A análise das funções de ativação que utilizamos em redes neurais nos ajuda a identificar em que cenário alcançamos o estado de gradiente minguante e como podemos reduzir seu impacto. Ao analisarmos a função logística, citada na sub-seção 2.3.1, vemos que sua derivada decresce conforme nos afastamos da origem $x = 0$. De fato, note que a derivada da função 2.7 quando $x \to \pm\infty$ é igual a zero:

$$\lim_{x \to \pm\infty} \left(\frac{d}{dx} \left(\frac{1}{1 + e^{-x}} \right) \right) = \lim_{x \to \pm\infty} \frac{e^x}{(e^x + 1)^2} = 0 \tag{2.21}$$

Funções com fronteiras superior e inferior, ou que são assintoticamente limitadas, sofrem do mesmo problema que a função logística. Uma das estratégias para mitigar o impacto da saturação da rede é a utilização de funções não limitadas quando $x \to \pm\infty$. A função ReLU, por exemplo, cresce indefinidamente quando $x \to +\infty$, portanto, é uma boa candidata à reduzir o impacto causado pelo gradiente minguante, já que:

$$\lim_{x \to +\infty} \left(\frac{d}{dx} ReLU(x) \right) = 1 \tag{2.22}$$

Contudo, para valores negativos a função ReLU possui derivada igual à zero, gerando o problema de gradiente minguante conhecido como *Dying ReLU* (LU *et al.*, 2020, MISRA, 2020, TROTTIER *et al.*, 2016). Neste caso, há outras alternativas que alteram o comportamento da função quando $x < 0$, como é o caso das funções LeakyReLU, Mish e SH-ReLU, mencionadas anteriormente nas sub-seções 2.3.1 e 2.3.2. Tais funções reduzem o impacto que o gradiente minguante pode eventualmente causar durante o treinamento pois eliminam o caso de *Dying ReLU*.

Os casos em que não é possível utilizar a função ReLU ou suas variantes são geralmente nas camadas de saída da rede. O uso de uma função com formato sigmoidal, como a função logística, traz naturalmente um resultado compatível com a expectativa de obter um número que representa uma probabilidade. E, para reduzir os impactos do gradiente minguante, uma das abordagens empregadas é o uso de funções de ativação adaptativas. CANALLI (2017) demonstra como as funções que utilizam penalização hiperbólica (BHSA, BHAA, SH-ReLU) podem, teoricamente, retardar o processo de gradiente minguante.

Capítulo 3

Método Proposto

Neste capítulo descrevemos a principal proposta para o presente trabalho: a aplicação de funções de ativação adaptativas em redes neurais adversárias generativas. Descrevemos também a proposta de duas novas funções de ativação adaptativas baseadas em funções pré-existentes com o objetivo de melhorar seu desempenho aplicando-as em redes GANs.

3.1 BHANA: Bi-hiperbólica assimétrica normalizada adaptativa

Durante a seção 2.3, apresentamos a função BHAA, uma função de ativação com parâmetros treináveis de formato sigmoidal que utiliza a técnica de penalização hiperbólica. Conforme sua definição, quando os parâmetros τ_1 e τ_2 são diferentes, uma distorção é criada no formato da função de forma que um pico (quando $\tau_1 > \tau_2$) ou um vale (quando $\tau_1 < \tau_2$) seja formado. Esse comportamento fica claro ao obervarmos os gráficos da função quando variamos tais parâmetros na figura 2.19.

A deformação da função BHAA acaba ocasionando um problema de extrapolação dos limites de atuação de acordo com a relação entre seus parâmetros. MIGUEZ (2012) clarifica que nas funções bi-hiperbólicas assimétricas o valor mínimo da função pode ser menor do que 0 ou maior 1 (ou, no caso da versão escalada (CANALLI, 2017), menor que -1 ou maior do que 1). Com isso, obtemos um erro de cálculo quando avaliamos a função de custo da rede neural, pois diversos dos valores de saída calculados pela rede podem estar fora do intervalo de atuação da função de custo (como é o caso da função de custo BCE (*binary cross-entropy*)). Note que quando $\tau_1 = \tau_2$ obtemos exatamente uma função BHSA, que não sofre do problema de extrapolação dos intervalos de atuação.

Para mitigar o problema de extrapolação dos limites de atuação, a implementação usual coloca em prática o truncamento destes valores, como mostrado na figura 3.1.

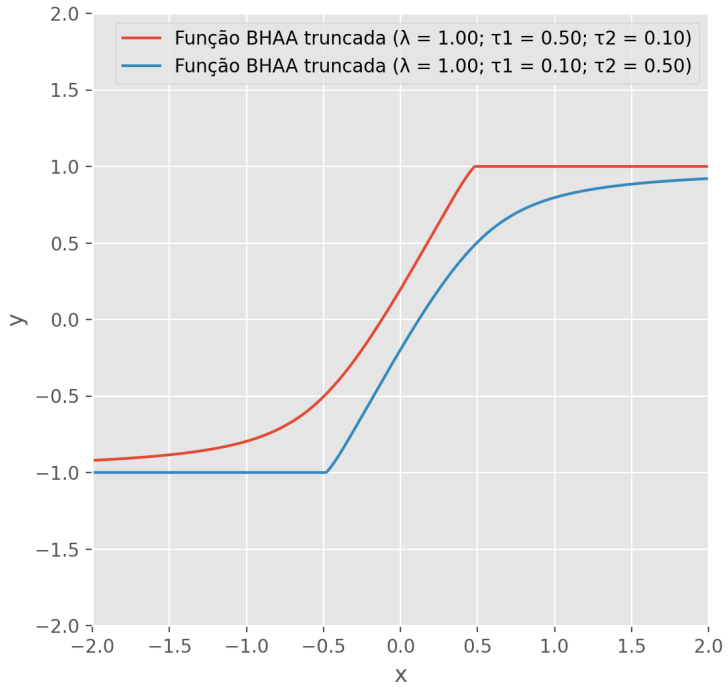


Figura 3.1: Gráfico da função BHAA com valores truncados para limitar o resultado de sua aplicação dentro dos limites de atuação apropriados para os modelos de aprendizado.

Note que, a partir do ponto que os valores começam a serem truncados, a derivada em tais pontos se torna zero, causando uma desvantagem teórica entre a própria BHAA e sua versão simétrica BHSA.

Tendo em consideração o problema definido das funções BHAA que extrapolam o intervalo de atuação, propomos uma versão normalizada da função onde temos como objetivos:

- Criar uma versão matematicamente normalizada, onde os limites de atuação da função não sejam extrapolados independentemente dos valores dos parâmetros τ_1 e τ_2.
- Manter a característica da função original onde melhoramos a convergência do modelo em épocas iniciais do treinamento, como observado em CANALLI (2017).

Neste ponto, propomos utilizar a própria derivada da função BHAA para normalizar seu resultado dentro do intervalo de atuação esperado. Sabemos que, por definição, o valor da icógnita x da derivada de uma função quando igualada à zero é um ponto extremo da função. A derivada da função BHAA (2.20) é definida a seguir como:

$$\frac{d}{dx}\text{BHAA}(x,\lambda,\tau_1,\tau_2) = \frac{\lambda(1-2\lambda x)}{\sqrt{(1-2\lambda x)^2+4\tau_2^2}} + \frac{\lambda(1+2\lambda x)}{\sqrt{(1+2\lambda x)^2+4\tau_1^2}} \tag{3.1}$$

Igualamos a equação 3.1 à zero e resolvemos em relação à x:

$$\frac{\lambda(1-2\lambda x)}{\sqrt{(1-2\lambda x)^2+4\tau_2^2}} + \frac{\lambda(1+2\lambda x)}{\sqrt{(1+2\lambda x)^2+4\tau_1^2}} = 0 \tag{3.2a}$$

$$\frac{\lambda(1-2\lambda x)}{\sqrt{(1-2\lambda x)^2+4\tau_2^2}} = -\frac{\lambda(1+2\lambda x)}{\sqrt{(1+2\lambda x)^2+4\tau_1^2}} \tag{3.2b}$$

$$\lambda(1-2\lambda x)\sqrt{(1+2\lambda x)^2+4\tau_1^2} = -\lambda(1+2\lambda x)\sqrt{(1-2\lambda x)^2+4\tau_2^2} \tag{3.2c}$$

$$(1-2\lambda x)\sqrt{(1+2\lambda x)^2+4\tau_1^2} = -(1+2\lambda x)\sqrt{(1-2\lambda x)^2+4\tau_2^2} \tag{3.2d}$$

$$(1-2\lambda x)^2((1+2\lambda x)^2+4\tau_1^2) = (1+2\lambda x)^2((1-2\lambda x)^2+4\tau_2^2) \tag{3.2e}$$

$$4\tau_1^2 - 4\tau_2^2 - 16\tau_1^2\lambda x - 16\tau_2^2\lambda x + 16\tau_1^2\lambda^2x^2 - 16\tau_2^2\lambda^2x^2 = 0 \tag{3.2f}$$

$$4(-\tau_1-\tau_2+2\tau_1\lambda x-2\tau_2\lambda x)(-\tau_1+\tau_2+2\tau_1\lambda x+2\tau_2\lambda x) = 0 \tag{3.2g}$$

Podemos notar que para que a equação 3.2g seja igual à zero, ou o segundo termo ou o terceiro termo do produto precisam ser iguais a zero. Com isso, podemos resolver os dois termos separadamente igualando eles à zero para encontrarmos os possíveis valores de x:

$$-\tau_1-\tau_2+2\tau_1\lambda x-2\tau_2\lambda x = 0 \tag{3.3a}$$

$$2\lambda x(\tau_1-\tau_2) = \tau_1+\tau_2 \tag{3.3b}$$

$$x_1 = \frac{\tau_1+\tau_2}{2\lambda(\tau_1-\tau_2)} \tag{3.3c}$$

$$-\tau_1+\tau_2+2\tau_1\lambda x+2\tau_2\lambda x = 0 \tag{3.4a}$$

$$2\lambda x(\tau_1+\tau_2) = \tau_1-\tau_2 \tag{3.4b}$$

$$x_2 = \frac{\tau_1-\tau_2}{2\lambda(\tau_1+\tau_2)} \tag{3.4c}$$

Desta forma, obtemos em 3.3c e 3.4c as raízes da equação 3.1. Tais valores representam pontos extremo da função BHAA. Avaliamos tais valores para x na equação 2.20 e:

- (Caso 1) se $\tau_1 > \tau_2$: o resultado de maior grandeza numérica será o ponto máximo da função (p_{max}) e a função nunca terá valores menores do que zero

(ou -1 no caso escalado).

- (Caso 2) se $\tau_1 < \tau_2$: o resultado de menor grandeza numérica será o ponto mínimo (p_{min}) da função e a função nunca terá valores maiores do que 1.

Dependendo da relação entre os parâmetros τ_1 e τ_2, usamos os valores encontrados de mínimo e máximo da função para normalizá-la no intervalo $[0, 1]$. Deste modo obtemos uma nova versão para a função BHAA, que será denominada como BHANA (bi-hiperbólica assimétrica normalizada adaptativa):

$$\text{BHANA}(x, \lambda, \tau_1, \tau_2) = \frac{\text{BHAA}(x, \lambda, \tau_1, \tau_2) - p_{min}}{p_{max} - p_{min}} \tag{3.5}$$

Onde:

$$p_{min} = (-1)^k (v_{min})^{1-k} \tag{3.6a}$$

$$p_{max} = (v_{max})^k (1)^{1-k} \tag{3.6b}$$

$$k = \frac{|\tau_1 - \tau_2|}{2(\tau_1 - \tau_2)} + 0.5 \tag{3.6c}$$

$$v_{max} = max(\text{BHAA}(x_1, \lambda, \tau_1, \tau_2), \text{BHAA}(x_2, \lambda, \tau_1, \tau_2)) \tag{3.6d}$$

$$v_{min} = min(\text{BHAA}(x_1, \lambda, \tau_1, \tau_2), \text{BHAA}(x_2, \lambda, \tau_1, \tau_2)) \tag{3.6e}$$

$$\tag{3.6f}$$

Os termos v_{max} e v_{min} representam, respectivamente, os resultados máximos e mínimos quando avaliamos a função BHAA utilizando x_1 e x_2. O termo k representa qual o caso de relação entre τ_1 e τ_2 temos (0 se $\tau_1 < \tau_2$, e 1 se $\tau_1 > \tau_2$), e auxilia na seleção dos pontos mínimos e máximos da função, representados respectivamente por p_{min} e p_{max}. Podemos visualizar na figura 3.2 qual o formato que a função BHANA assume como resultado da normalização dos valores de saída.

3.2 MiDA: Mish Dual Adaptive

Outra função de ativação abordada durante a sub-seção 2.3.1, é a função Mish. Nesta seção propomos uma modificação nesta função de forma que ela passe a utilizar a bi-hiperbólica BHSA no lugar da função tangente hiperbólica, que fornece a propriedade de *self-gating* da função original.

Ao utilizarmos a função BHSA como substituta da tangente hiperbólica na Mish, esperamos manter todas as três propriedades que caracterizam as vantagens de usar tal função (descritos na sub-seção 2.3.1): *self-gating*, derivável em todos os pontos e *self-regularizing*. Além disso, como a BHSA é uma função de ativação adaptativa,

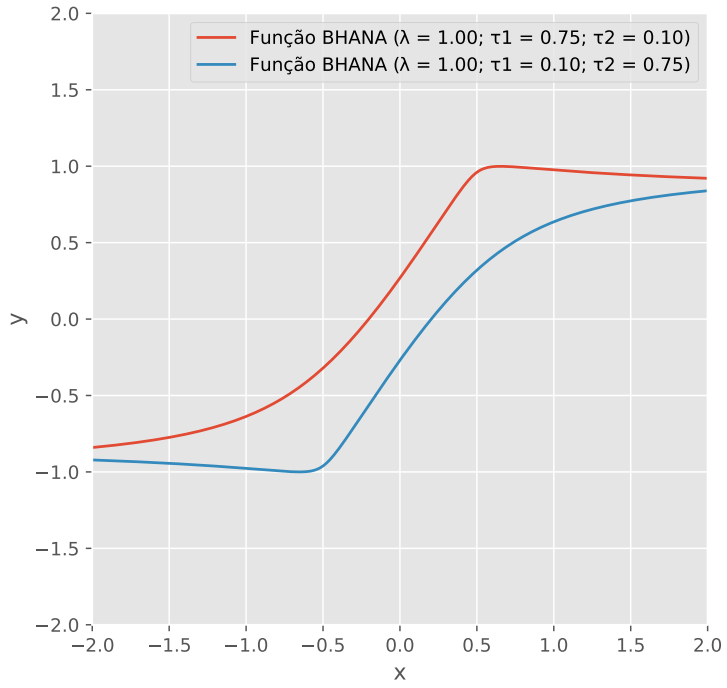


Figura 3.2: Gráfico da função BHANA com diferentes parâmetros τ_1 e τ_2.

esperamos também que as características de melhor convergência em épocas iniciais de treinamento sejam preservados (como observado em CANALLI (2017)). Não consideramos nenhuma versão da função bi-hiperbólica assimétrica pois esta ainda possui certas inconsistências sobre o limite de atuação apresentados na seção 3.1.

Denominamos a função proposta como MiDA (*Mish Dual Adaptive*), a qual possui dois parâmetros treináveis (herdados do uso da função BHSA: τ e λ) e é definida matematicamente a seguir pela equação 3.7 como:

$$\text{MiDA}(x, \lambda, \tau) = x * \text{BHSA}(\text{softmax}(x), \lambda, \tau) \tag{3.7}$$

Onde a função BHSA é definida pela equação 2.17 e $\text{softmax(x)} = \ln(1 + e^x)$. A derivada da função MiDA é defindа a seguir pela equação 3.8:

$$\begin{aligned}\frac{d}{dx}\text{MiDA}(x, \lambda, \tau) = {} & 0.5\sqrt{(1 + 2\lambda \ln(1 + e^x))^2 + 4\tau^2} \\ & -0.5\sqrt{(1 - 2\lambda \ln(1 + e^x))^2 + 4\tau^2} \\ & + \frac{\lambda e^x x(1 - 2\lambda \ln(1 + e^x))}{(1 + e^x)\sqrt{(1 - 2\lambda \ln(1 + e^x))^2 + 4\tau^2}} \\ & + \frac{\lambda e^x x(1 + 2\lambda \ln(1 + e^x))}{(1 + e^x)\sqrt{(1 + 2\lambda \ln(1 + e^x))^2 + 4\tau^2}}\end{aligned} \tag{3.8}$$

Podemos reescrever a equação 3.8 para uma fórmula mais simples. Note que o

primeiro e segundo termos da soma são exatamente BHSA(softmax$(x), \lambda, \tau)$). Além disso, as raízes quadradas presentes nos denominadores dos terceiro e quarto termos são, respectivamente, $h_1 * 2$ e $h_2 * 2$ da função BHSA (onde h_1 é definido em 2.18 e h_2 em 2.19). Extraímos ainda os valores em comum do terceiro e quarto termos, obtendo a versão simplificada da derivada a seguir:

$$\frac{d}{dx}\text{MiDA}(x, \lambda, \tau) = \Delta(x) + \frac{\text{MiDA}(x, \lambda, \tau)}{x} \tag{3.9}$$

Onde:

$$\Delta(x) = \frac{\lambda e^x x}{2(1+e^x)} \left(\frac{1 + 2\lambda \text{softmax}(x)}{h_1(\text{softmax}(x))} + \frac{1 - 2\lambda \text{softmax}(x)}{h_2(\text{softmax}(x))} \right) \tag{3.10}$$

Com essa análise da derivada da função MiDA, podemos fazer um paralelo com a derivada da função Mish na equação 2.12. Podemos notar que ambas possuem um termo da soma que fornecem o indicativo de um regularizador para a função: $\Delta(x)$.

A propriedade de *self-gating* continua válida no caso da função MiDA. A função BHSA, com seu formato sigmoidal, continua exercendo o papel de *gating* que a função tangente hiperbólica exerce na Mish.

A propriedade de diferenciabilidade em todos os pontos na função MiDA pode ser facilmente observada quando analisamos o gráfico da função na figura 3.3.

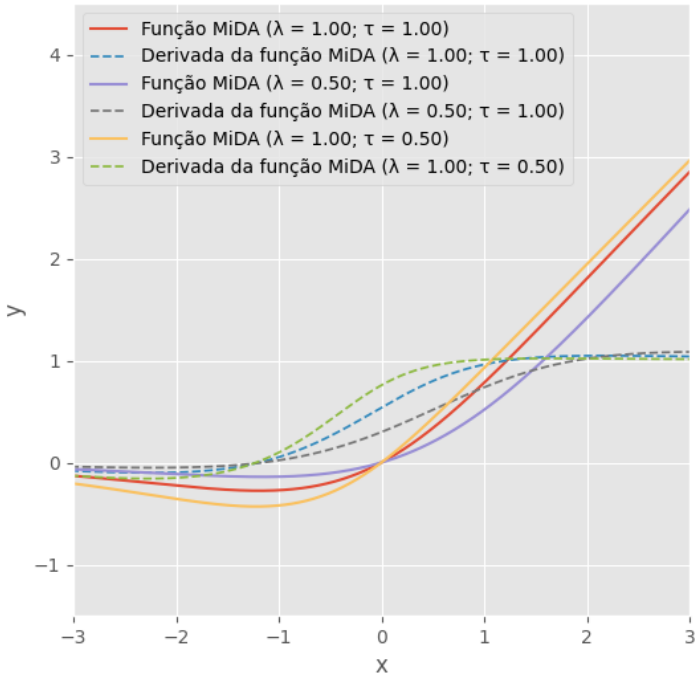


Figura 3.3: Gráfico da função MiDA e sua derivada com diversos parâmetros

3.3 Funções de ativação adaptativas em Generative Adersarial Nets

Os conceitos das redes generativas adversárias, assim como seu crescente uso, foram discutidos no presente trabalho durante a seção 2.2. Um dos pontos destacados durante a discussão teórica é o fato desta arquitetura ser um modelo de *deep learning*, ou seja, redes neurais profundas multicamadas são geralmente utilizadas para compor as redes geradora e discriminadora. Com o avanço das pesquisas nesta área, modelos como em KARRAS *et al.* (2019) passam a ficar cada vez mais complexos e demandam mais recursos de computação.

Já as funções de ativação adaptativas podem ser aplicadas em redes neurais GANs para melhorar seu resultado. Na seção 2.3.2 vimos que funções de ativação adaptativas baseadas na técnica de penalização hiperbólica podem melhorar o desempenho da rede em épocas iniciais durante o treinamento de redes neurais, como mostram experimentos disponíveis na literatura (CANALLI, 2017).

Neste contexto, propomos a utilização das funções de ativação baseadas em técnicas de penalização hiperbólica em redes neurais adversárias. Como as redes GANs demandam mais recursos computacionais, consequentemente aumentando o tempo de treinamento em ambientes com disponibilidade de processamento e memória restritos, a utilização de funções adaptativas hiperbólicas para atingir um *threshold* de qualidade de resultado em épocas mais iniciais do treinamento pode significar um menor tempo necessário de treinamento para alcançar o limite desejado.

Separamos então as funções adaptativas em duas categorias: funções com formato base sigmoidal e funções de formato base retificadora. As funções com formato base sigmoidal deverão ser utilizadas como substitutas para funções sigmoidais estáticas. Funções sigmoidais são geralmente utilizadas em camadas de saída das redes geradora e discriminadora, como por exemplo as funções tangente hiperbólica e logística respectivamente.

Já as funções com formato base retificadora deverão ser utilizadas como substitutas para funções retificadoras estáticas, como as funções ReLU e LeakyReLU. Estas funções são geralmente utilizadas em camadas internas (ou camadas ocultas) das redes neurais. A tabela 3.1 lista as funções adaptativas que propomos o uso em redes GAN, assim como sua categoria e seus parâmetros adaptativos.

Denominamos esta arquitetura de rede GAN utilizando funções adaptativas como AAF-GAN (*Adaptive activation functions GAN*). Na figura 3.4 esquematizamos onde cada função estática será substituída nas redes adversárias baseado na categoria das funções da tabela 3.1.

Tabela 3.1: Funções de ativação adaptativas com utilização de técnica de penalização hiperbólica a serem utilizadas em redes GANs

Função	**Categoria**	**Nº de parâmetros adaptativos**	**Parâmetros adaptativos**
BHSA	Formato sigmoidal	2	λ e τ
BHANA	Formato sigmoidal	3	λ, τ_1 e τ_2
SHReLU	Formato retificadora	1	λ
MiDA	Formato retificadora	2	λ e τ

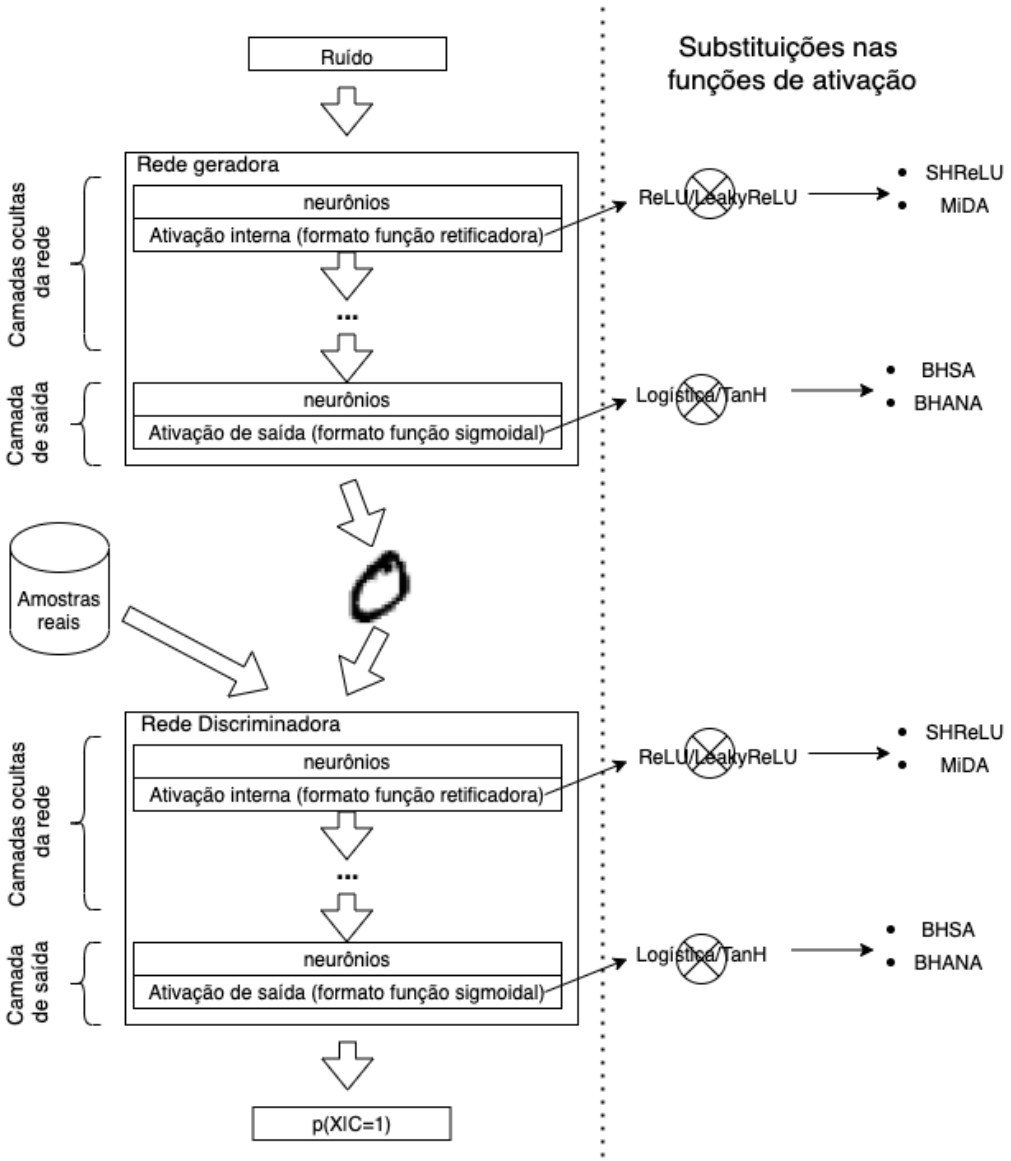


Figura 3.4: Arquitetura AAF-GAN: substituição de ativações estáticas por funções adaptativas que utilizam a técnica de penalização hiperbólica.

Capítulo 4

Experimentos e Resultados

Neste capítulo abordamos e discutimos os experimentos realizados afim de avaliar as propostas do presente trabalho. Primeiro, descrevemos a metodologia empregada assim como os conjuntos de dados utilizados em cada experimento. Depois, descrevemos os experimentos e analisamos seus resultados de acordo com sua métrica de avaliação. Os experimentos foram divididos em duas partes com o intuito de facilitar a compreensão dos modelos e resultados. Na parte 1, os experimentos foram realizados utilizando a arquitetura CondGAN com o *dataset* MNIST. Já na parte 2 utilizamos a arquitetura DCGAN com o *dataset* CelebA.

4.1 Metodologia

A metodologia empregada para avaliar o método proposto consiste em validar seu desempenho em duas diferentes arquiteturas de redes GAN. Nesta seção descrevemos os detalhes da estratégia empregada além de descrever os conjuntos de dados utilizados, as arquiteturas das redes e as métricas de avaliação para cada experimento. Demais configurações de redes, inicializações e arquiteturas possíveis não foram levadas em consideração na variação metodológica com o intuito de definir o escopo do espaço explorado e consequentemente não causar desnecessariamente o aumento exagerado da complexidade de combinações de possíveis experimentos.

4.1.1 Conjuntos de dados

MNIST

O *dataset* MNIST (LECUN e CORTES, 2010) é um conjunto de dados formado por imagens de dígitos de 0 à 9 escritos à mão livre. Ao todo, são 60 mil amostras consideradas como amostras de treinamento e mais 10 mil amostras para teste/validação, totalizando 63 MB de dados. As amostras representam imagens 28x28 *pixels* em escala cinza. O MNIST é um conjunto de dados bem consolidado e amplamente

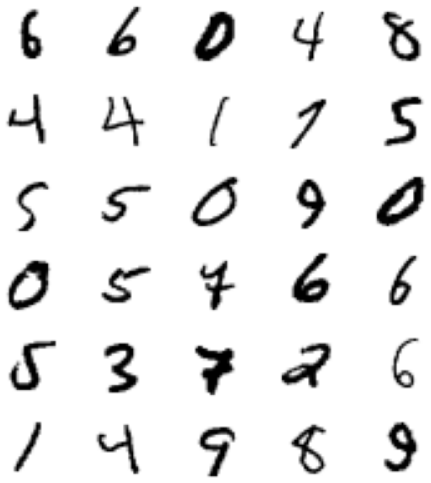


Figura 4.1: Amostra de imagens do *dataset MNIST*.

utilizado em diversos trabalhos e pesquisas para diferentes tarefas de aprendizado de máquina, como por exemplo classificação e geração de imagens. Um ranqueamento de modelos classificadores pode ser encontrado no website oficial[1], enquanto que podemos citar dentre os diversos modelos generativos que exploraram este *dataset* os trabalhos de: GOODFELLOW *et al.* (2014), MIRZA e OSINDERO (2014), RADFORD *et al.* (2015), KINGMA e WELLING (2014) e SONG *et al.* (2019). Algumas das amostras deste dataset podem ser encontradas na figura 4.1.

CelebA

Outro *dataset* que utilizamos neste trabalho é o CelebA (LIU *et al.*, 2015). O CelebA é um conjunto de dados de imagens em larga escala de rostos de celebridades, contendo 202,599 amostras de 10,177 indivíduos, totalizando 1,62 GB de dados. As amostras são imagens em RGB com dimensões 218x178 *pixels*. Algumas das amostras deste dataset podem ser encontradas na figura 4.2.

O *CelebA* é amplamente utilizado em tarefas de aprendizado de máquina para geração de imagens (KARRAS *et al.*, 2019, LIN *et al.*, 2019, MENICK e KALCHBRENNER, 2018), reconhecimento de atributos faciais (CRESWELL *et al.*, 2017, GANJU *et al.*, 2018) e detecção de faces (KIM *et al.*, 2019).

4.1.2 Arquitetura das redes GAN

Vanilla CondGAN

A primeira arquitetura que utilizamos na avaliação experimental a ser descrita é a *Conditional GAN* (CondGAN). Como visto anteriormente durante a revisão bibli-

[1]MNIST dataset website: `http://yann.lecun.com/exdb/mnist/`

Figura 4.2: Amostra de imagens do *dataset CelebA*.

ográfica na sub-seção 2.2.2, a rede CondGAN é um modelo de GAN onde concatenamos às entradas das redes geradora e discriminadora as respectivas os respectivos valores das *labels* de cada amostra. Nossa rede segue os mesmos princípios arquiteturais propostos pela publicação original (MIRZA e OSINDERO, 2014), o qual é baseado na primeira arqitetura GAN (GOODFELLOW *et al.*, 2014) - popularmente conhecido também como *Vanilla GAN*.

A rede geradora possui 4 camadas *fully-connected*, onde a primeira camada é composta por 256 neurônios, a segunda por 512, a terceira por 1024 e a última por 786 neurônios. Em cada camada é aplicada uma função de ativação, onde, na arquitetura original é escolhida a função LeakyReLU com parâmetro $\alpha = 0.2$, com exceção da última camada onde é aplicada a função tangente hiperbólica. A figura 4.3 representa o diagrama da arquitetura da rede geradora.

A rede discriminadora também possui 4 camadas *fully-connected*. A primeira camada possui 1024 neurônios, a seguda possui 512, a terceira possui 256 e a última camada possui 1 neurônio. Da primeira à terceira camada são aplicadas à saída dos neurônios uma função de ativação, onde no trabalho original utiliza-se a LeakyReLU com $\alpha = 0.2$, além de uma camada de *Dropout* com $p = 0.3$. Na última camada apenas a função de ativação logística é aplicada. A figura 4.4 representa o diagrama da rede discriminadora.

DCGAN

Outra arquitetura de rede GAN que utilizamos para validar nossa proposta é conhecida como DCGAN. Neste modelo, as redes geradora e discriminadora são constituídas de redes convolucionais, ao invés de simplesmente *fully-connected networks*. O mesmo modelo de arquitetura proposto pelo trabalho original (RADFORD *et al.*,

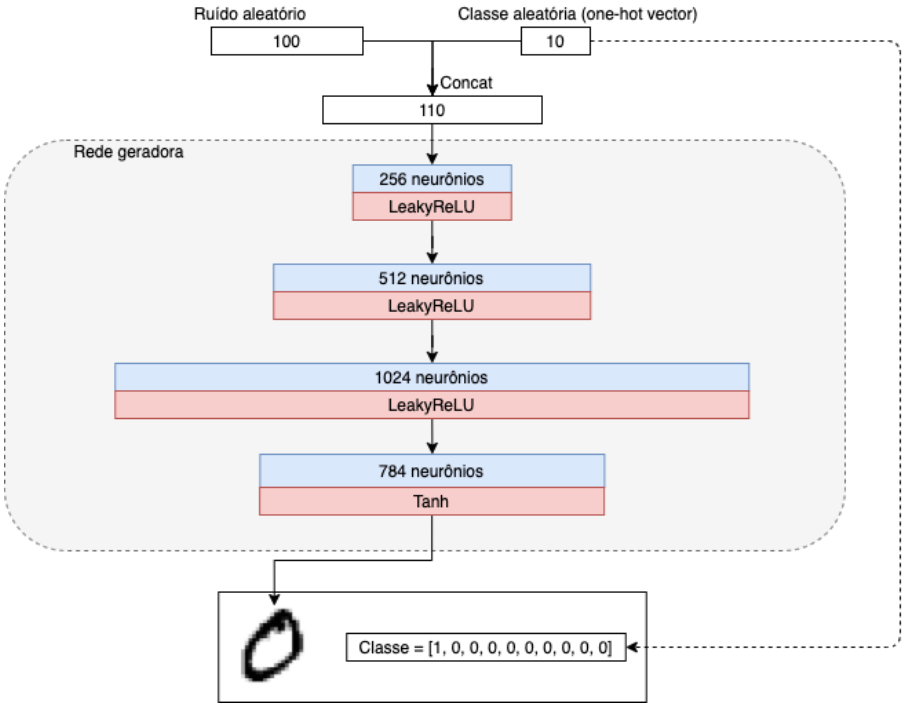


Figura 4.3: Diagrama da rede geradora da arquitetura CondGAN utilizada durante experimentos.

2015) foi seguido durante nossa avaliação experimental.

A rede geradora é uma rede convolucional com 5 camadas, onde as quatro primeiras camadas são compostas por uma de-convolução, seguido de um *BatchNorm* e uma função de ativação, no caso do modelo padrão uma ReLU. A última camada é formada por uma de-convolução seguida da função de ativação tangente hiperbólica. Os detalhes de cada camada da rede geradora podem ser encontrados na tabela 4.1 e seu diagrama na figura 4.5.

A rede discriminadora também possui 5 camadas, onde da segunda à quarta camada são formadas por uma convolução, seguida de um *BatchNorm* e uma função de ativação LeakyReLU (com parâmetro $\alpha = 0.2$). A primeira camada é composta por uma convolução seguida da ativação LeakyReLU ($\alpha = 0.2$). A quinta e última camada é composta por uma convolução seguida da ativação logística. Os detalhes de cada camada da rede discriminadora podem ser encontrados na tabela 4.2 e seu diagrama na figura 4.6.

4.1.3 Configuração geral das redes neurais

No quesito configuração da rede e seus componentes, levamos em consideração algumas das melhorias que redes GAN podem tirar proveito. Baseado na literatura disponível (GOODFELLOW *et al.*, 2014, GOODFELLOW, 2017, RADFORD *et al.*, 2015, SALIMANS *et al.*, 2016), empregamos as seguintes configurações, hiperparâ-

Tabela 4.1: Arquitetura da rede geradora do modelo DCGAN utilizado nos experimentos.

Camada	**Deconv.**	**BatchNorm**	**Função de ativação**
Camada 1	$canais_{in} = 100$ $canais_{out} = 1024$ $kernel = 4$ $stride = 1$ $padding = 0$	Sim	ReLU
Camada 2	$canais_{in} = 1024$ $canais_{out} = 512$ $kernel = 4$ $stride = 2$ $padding = 1$	Sim	ReLU
Camada 3	$canais_{in} = 512$ $canais_{out} = 256$ $kernel = 4$ $stride = 2$ $padding = 1$	Sim	ReLU
Camada 4	$canais_{in} = 256$ $canais_{out} = 128$ $kernel = 4$ $stride = 2$ $padding = 1$	Sim	ReLU
Camada 5	$canais_{in} = 128$ $canais_{out} = 3$ $kernel = 4$ $stride = 2$ $padding = 1$	Não	Tanh

Tabela 4.2: Arquitetura da rede discriminadora do modelo DCGAN utilizado nos experimentos.

Camada	**Conv.**	**BatchNorm**	**Função de ativação**
Camada 1	$canais_{in} = 3$ $canais_{out} = 128$ $kernel = 4$ $stride = 2$ $padding = 1$	Não	LeakyReLU
Camada 2	$canais_{in} = 128$ $canais_{out} = 256$ $kernel = 4$ $stride = 2$ $padding = 1$	Sim	LeakyReLU
Camada 3	$canais_{in} = 256$ $canais_{out} = 512$ $kernel = 4$ $stride = 2$ $padding = 1$	Sim	LeakyReLU
Camada 4	$canais_{in} = 512$ $canais_{out} = 1024$ $kernel = 4$ $stride = 2$ $padding = 1$	Sim	LeakyReLU
Camada 5	$canais_{in} = 1024$ $canais_{out} = 1$ $kernel = 4$ $stride = 1$ $padding = 0$	Não	Logística

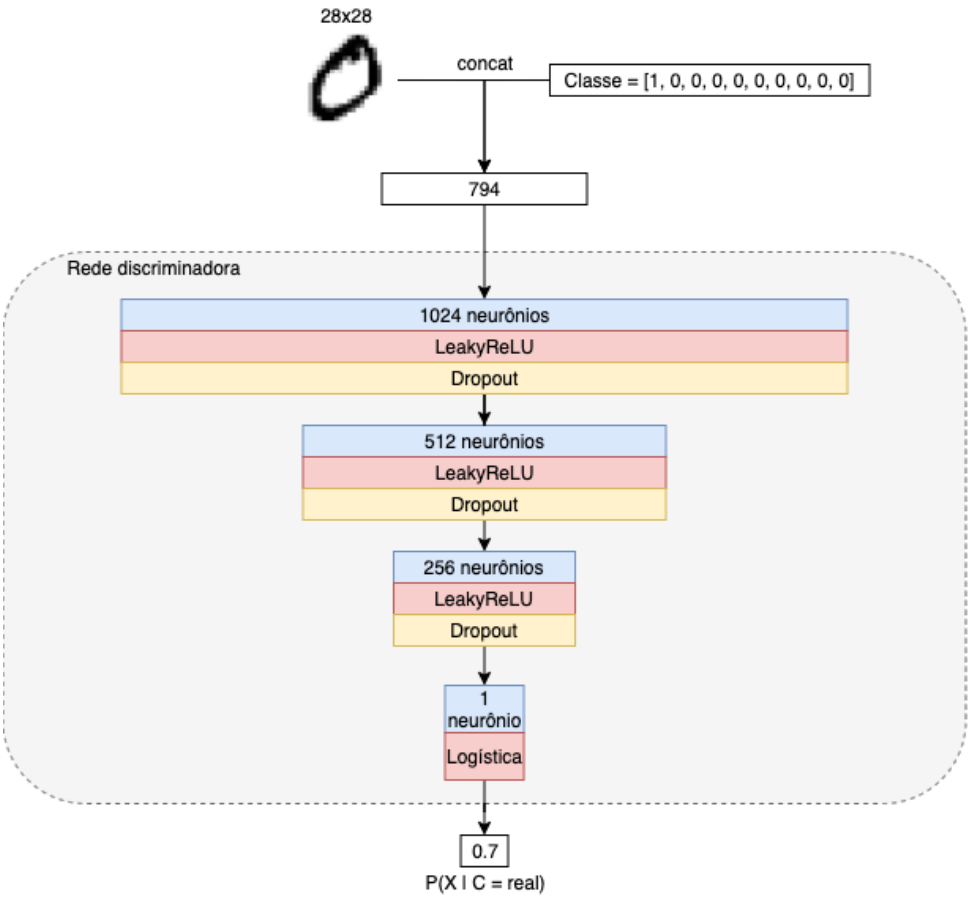


Figura 4.4: Diagrama da rede discriminadora da arquitetura CondGAN utilizada durante experimentos.

metros e técnicas:

- Método de otimização: utilizamos *Adam* (KINGMA e BA, 2015) como otimizador em ambas as redes para ambas arquiteturas, com taxa de aprendizado igual à 0.0002 e $\beta_1 = 0.5$;
- *One-sided label smoothing*: substituímos a *label* que representa exemplos reais no discriminador por 0.9, ao invés de usar o valor 1;
- Inversão de valores das *labels*: para treinar o gerador, invertemos o valor das classes alvo. As amostras reais são tratados como da classe falsa, enquanto as amostras gerados são tratadas como da classe positiva. Este mecanismo é mais eficiente na prática do que treinar o gerador para minimizar $1 - loss_D$, onde $loss_D$ é o *loss* da rede discriminadora.
- Função de custo: *Binary cross-entropry* foi utilizada para calcular o *loss* durante o treinamento.
- Inicialização dos pesos dos neurônios: para componentes convolucionais, inicializamos os pesos dos neurônios seguindo uma distribuição normal com média

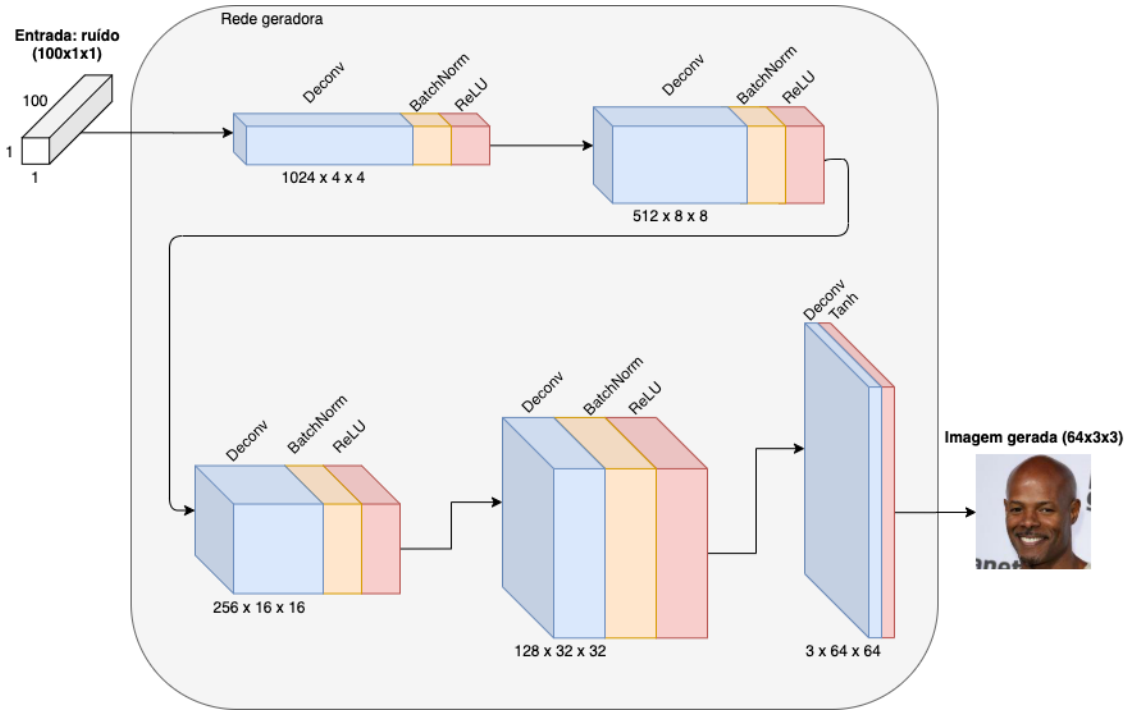


Figura 4.5: Diagrama da rede geradora da arquitetura DCGAN utilizada durante experimentos.

0.0 e variância 0.02. Para componetes *BatchNorm*, inicializamos usando também uma normal mas com média 1 e variância 0.02

4.1.4 Inicialização dos parâmetros em funções de ativação adaptativas

Para a inicialização dos parâmetros das funções de ativação adaptativas utilizadas, adotamos a estratégia de inicializar tais valores de modo que o formato da função ficasse, em média, próximo ao formato da função não-adaptativa que esta estava substituindo no modelo. A inicialização utilizando a distribuição normal foi utilizada para garantir que houvesse uma variação mínima nos parâmetros de forma que pudesse guiar o treinamento das redes à uma melhor convergência. Listamos na tabela 4.3 o valor de inicialização de cada parâmetro para cada função de ativação adaptativa em cada rede.

4.1.5 Métricas de avaliação

Em redes neurais generativas adversárias, devido à dinâmica de treinamento, os valores das funções de custo das redes não são métricas válidas para mensurar a qualidade das amostras geradas. Com o sucesso das redes GAN diversas métricas já foram propostas e estudadas como forma de avaliar do desempenho do modelo (BORJI, 2018, SHMELKOV *et al.*, 2018).

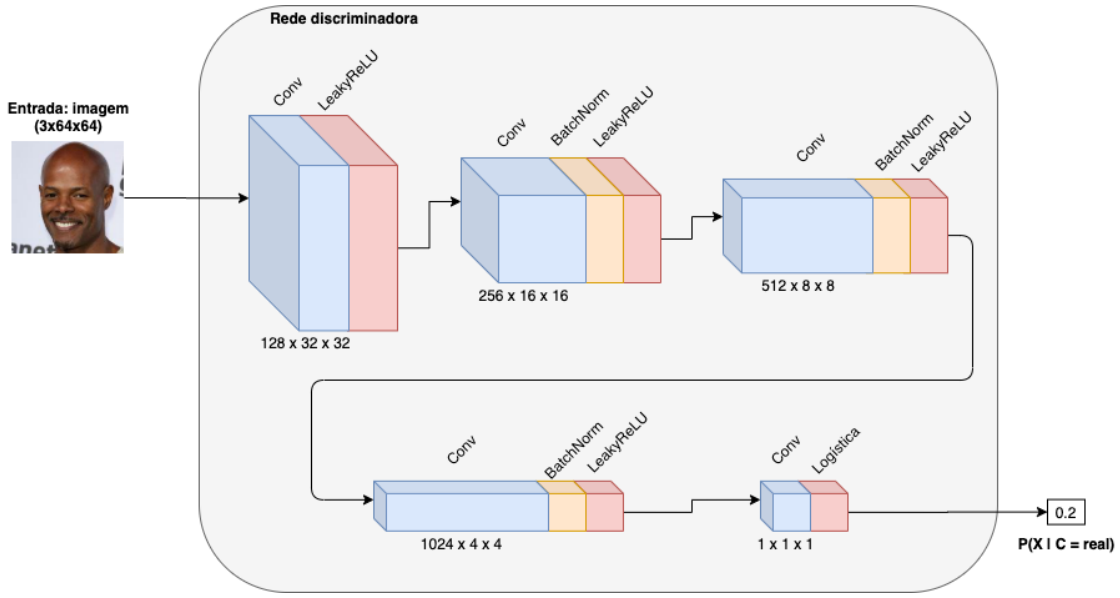


Figura 4.6: Diagrama da rede discriminadora da arquitetura DCGAN utilizada durante experimentos.

Tabela 4.3: Inicialização dos parâmetros nas funções de ativação adaptativas.

Função	**Rede**	**Função que substitui**	**Média**	**Variância**
BHSA/BHANA	Gerador	Tangente hiperbólica	$\tau_{media} = 0.25$ $\lambda_{media} = 0.5$	$\tau_{var} = 0.05$ $\lambda_{var} = 0.05$
BHSA/BHANA	Discriminador	Logística	$\tau_{media} = 0.7$ $\lambda_{media} = 0.5$	$\tau_{var} = 0.05$ $\lambda_{var} = 0.05$
SHReLU	Gerador/ Discriminador	ReLU/ LeakyReLU	$\tau_{media} = 0$	$\tau_{var} = 0.2$
MiDA	Gerador/ Discriminador	ReLU/ LeakyReLU	$\alpha_{media} = 1$ $\beta_{media} = 0$	$\alpha_{var} = 0.05$ $\beta_{var} = 0.001$

Apesar de ter sido a primeira métrica utilizada para avaliar a qualidade do gerador (GOODFELLOW *et al.*, 2014) (devido ao fato de ser geralmente utilizada em modelos generativos) e amplamente utilizada durante o surgimento da arquitetura GAN, a métrica de *log-likelihood* médio utilizando *Kernel Density Estimation* (KDE) vem sendo desde então questionada em relação à sua utilidade para avaliar a qualidade das amostras geradas (BORJI, 2018). Portanto, duas diferentes métricas, uma para cada modelo, foram selecionadas para avaliar a qualidade das amostras geradas.

Desempenho de classificação

A métrica de desempenho de classificação consiste em avaliar a acurácia das amostras condicionalmente geradas em um modelo classificador pré-treinado para o mesmo *dataset* (BORJI, 2018). A hipótese proposta por essa métrica é bastante intuitiva: caso o classificador resulte bons valores para métrica de acurácia ao tentar prever a classe das amostras geradas, então significa que as amostras geradas são qualitativamente informativas o suficiente para distinguir a classe do objeto.

Em nossos experimentos, utilizamos esta métrica para avaliar o desempenho das amostras geradas no modelo *Vanilla CondGAN*, apresentado na sub-seção 4.1.2, junto ao *dataset* do MNIST. O classificador escolhido foi o modelo implementado na documentação oficial da biblioteca *PyTorch* [2], o qual obtém um ótimo resultado de 99.25% de acurácia no conjunto de teste do MNIST.

FID score

A métrica *Fréchet Inception Distance* (*FID score*) (HEUSEL *et al.*, 2017) é uma métrica de avaliação para modelos generativos. O cálculo da métrica é realizado levando em consideração as médias e covariâncias dos dados reais e gerados quando aplicados à uma camada específica de uma *Inception Network* (SZEGEDY *et al.*, 2014).

O *FID score* assume que os valores das *features* resultantes da camada interna da *Inception Net* seguem uma distribuição normal multivariada. A distância entre as duas distribuições é então calculada usando *Fréchet Distance* entre duas normais como definido pela equação 4.1 a seguir:

$$FID(r, g) = ||\mu_r - \mu_g||^2 + Tr\left(\Sigma_r + \Sigma_g - 2\sqrt{\Sigma_r \Sigma_g}\right) \tag{4.1}$$

Na equação 4.1, *Tr* é a operação traço da matriz (soma dos itens na diagonal principal) resultante do cálculo interno entre parêntesis. Os parâmetros r e g são, respectivamente, as amostras dos dados reais e gerados. Em seu resultado, quanto menor o valor do *FID score* mais similares são as distribuições, ou seja, melhor a qualidade das amostras geradas pelo modelo.

4.1.6 Implementação e implantação

O código desenvolvido para os experimentos do presente trabalho está disponível no GitHub [3]. Utilizamos a biblioteca *PyTorch* (PASZKE *et al.*, 2017) para construir o modelo de redes neurais GAN e treinar com o auxílio de GPUs. Outras bibliotecas

[2] Classificador MNIST em *PyTorch*: `https://github.com/pytorch/examples/tree/master/mnist`

[3] Link para download do código: `https://github.com/chriiscardozo/msc_aafgan`

populares em Python também foram utilizadas neste trabalho como: *numpy* para operações aritiméticas, *matplotlib* para criar imagem dos gráficos, *multiprocessing* para paralelização de cálculos em CPU, *scikit-learn* e *scipy* para cálculos e fórmulas científicas. Uma implementação já pronta do *FID score* (SEITZER, 2020) foi utilizada para calcular a métrica durante os experimentos.

A parte 1 dos experimentos foi executada utilizando uma máquina com uma placa de vídeo NVIDIA GTX 1080ti, processador Intel i7-7740X e 32 GB de memória RAM. A parte 2 dos experimentos foi executada com o auxílio da computação em núvem da AWS (*Amazon Web Services*), utilizando instâncias EC2 do tipo p3.2xlarge com 8 núcleos de processamento virtuais (vCPU), 61 GB de memória RAM e uma placa de vídeo Tesla V100. Para ambos os ambientes, disponibilizamos no apêndice C o tempo médio gasto por época para cada modelo empregado nos experimentos.

4.2 Parte 1: *MNIST-CondGAN*

A primeira parte executada dos experimentos tem por objetivo observar o comportamento das funções adaptativas por meio da validação de performance das amostras geradas quando substituem as funções padrões no modelo *Vanilla CondGAN* (referenciado como *baseline* durante essa fase de experimentação). Para este experimento, utilizamos o *dataset* MNIST e, como neste caso estamos lidando com um modelo condicional, a métrica de avaliação selecionada será a acurácia em um classificador pré-treinado.

O pré-processamento realizado nas amostras reais do *dataset* consiste apenas no passo de normalizar o valor de cada *pixel* das amostras dentro do intervalo $[-1, 1]$, levando em consideração toda a base de dados para a normalização. Este valor representa a tonalidade do *pixel* em escala cinza dentro do intervalo citado.

Para o cálculo da métrica de avaliação, o mesmo modelo foi executado 10 vezes com inicializações de *seeds* diferentes para cada função adaptativa que usamos como substituta do modelo original.

O treinamento foi executado com um total de 100 épocas, utilizando *mini-batches* de tamanho 100. As dimensões da imagem de saída que o gerador produz é o mesmo encontrado na base de dados: 28x28 *pixels*. O ruído de entrada do gerador tem dimensão igual à 100 e é gerado a partir de uma distribuição normal com média 0 e variância 1.

As imagens geradas pelos modelos que obtiveram convergência durantes estes experimentos podem ser encontrados no apêndice A.

4.2.1 Experimento 1: BHSA vs *baseline*

O primeiro experimento da Parte 1 que iremos abordar é a utilização da função BHSA. Neste experimento, verificamos o desempenho de 3 diferentes modelos de AAF-GAN que utilizando a BHSA nas redes adversárias:

- Gen_BHSA: BHSA na camada de saída da rede geradora
- Dis_BHSA: BHSA na camada de saída da rede discriminadora
- Dis_BHSA_Gen_BHSA: BHSA na camada de saída de ambas as redes adversárias

Nas figuras 4.7, 4.8 e 4.9 temos, respectivamente, o desempenho dos modelos Gen_BHSA, Dis_BHSA e Dis_BHSA_Gen_BHSA com relação à métrica de acurácia do classificador pré-treinado. Podemos notar que durante as épocas iniciais o AAF-GAN com BHSA consegue superar o modelo *baseline*.

Destacamos que o modelo Dis_BHSA_Gen_BHSA se mantém melhor que o modelo padrão consistentemente até a época 45. Já os outros dois modelos que usam a função BHSA estritamente no gerador ou no discriminador conseguem ser superior ao modelo base em média até a época 35. Após a referida quantidade de épocas, o modelo base se iguala em performance de métrica com o AAF-GAN. Na figura 4.10 podemos comparar o delta da acurácia de cada modelo AAF-GAN usando BHSA, subtraindo o valor de acurácia do modelo base. O modelo AAF-GAN que utiliza a função BHSA em ambas as redes adversárias obtém resultados melhores em épocas iniciais comparado aos outros modelos.

4.2.2 Experimento 2: BHANA vs *baseline*

Os três modelos AAF-GAN formulados para o experimento usando a função BHANA são definidos como:

- Gen_BHANA: BHANA na camada de saída da rede geradora
- Dis_BHANA: BHANA na camada de saída da rede discriminadora
- Dis_BHANA_Gen_BHANA: BHANA na camada de saída de ambas as redes adversárias

Durante os experimentos utilizando a função BHANA identificamos dois problemas: falha na convergência do modelo e falhas computacionais nos cálculos do algoritmo de *backpropagation*.

A função BHANA, apesar de matematicamente coerente, se mostrou computacionalmente inviável. Observamos que o modelo de treinamento falhou em diversas

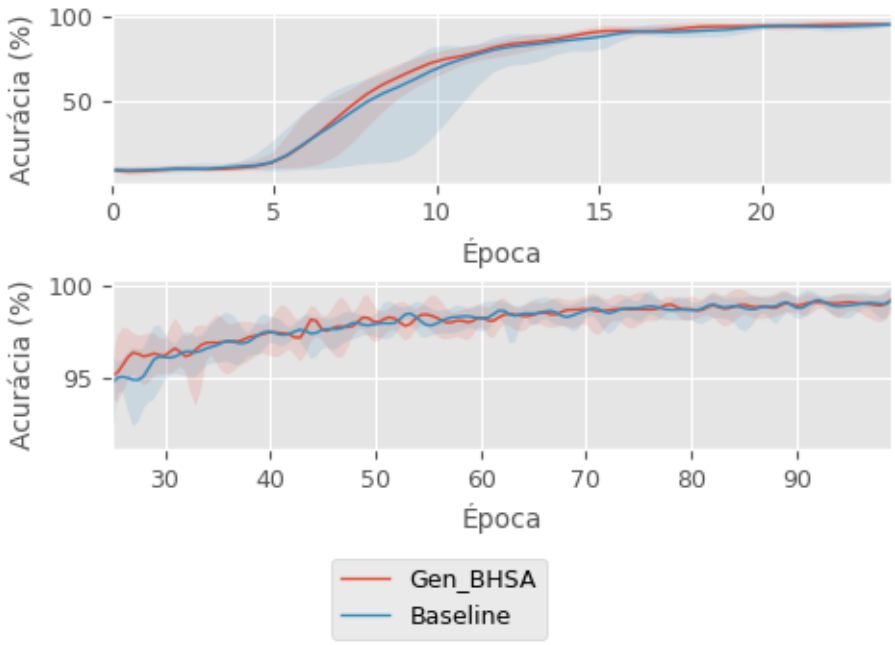


Figura 4.7: Acurácia média (com limites máximo e mínimo observado nos experimentos) do modelo *baseline* e AAF-GAN usando BHSA na rede geradora.

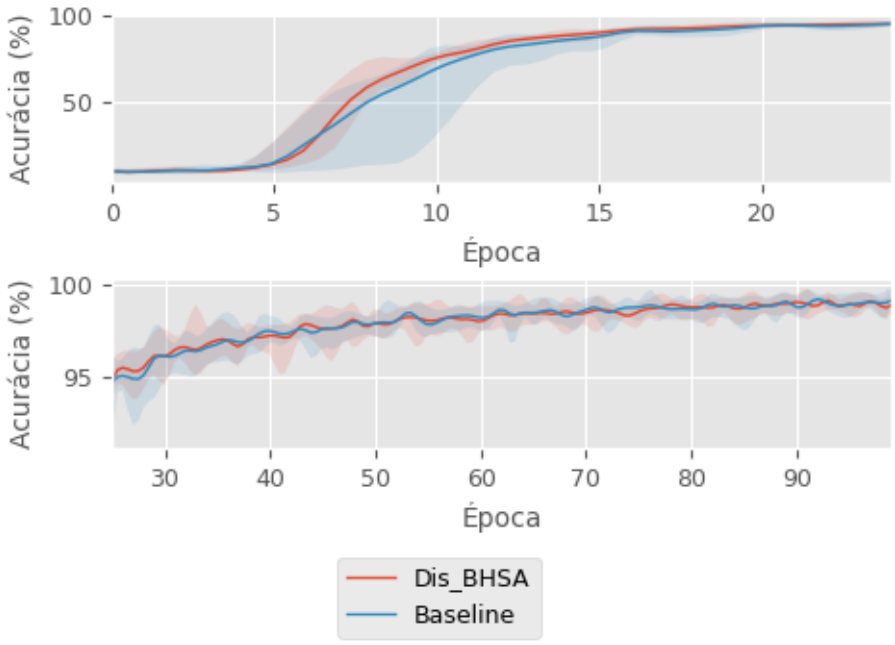


Figura 4.8: Acurácia média (com limites máximo e mínimo observado nos experimentos) do modelo *baseline* e AAF-GAN usando BHSA na rede discriminadora.

rodadas (com diferentes *seeds*) antes de terminar as épocas configuradas com erros de *underflow*. Sua formulação onde precisamos utilizar o cálculo da derivada condicionada à relação de τ_1 e τ_2 (equação 3.5) torna não-trivial a sua implementação e é a principal hipótese das falhas computacionais.

No entanto, para os experimentos que concluíram o processo de treinamento,

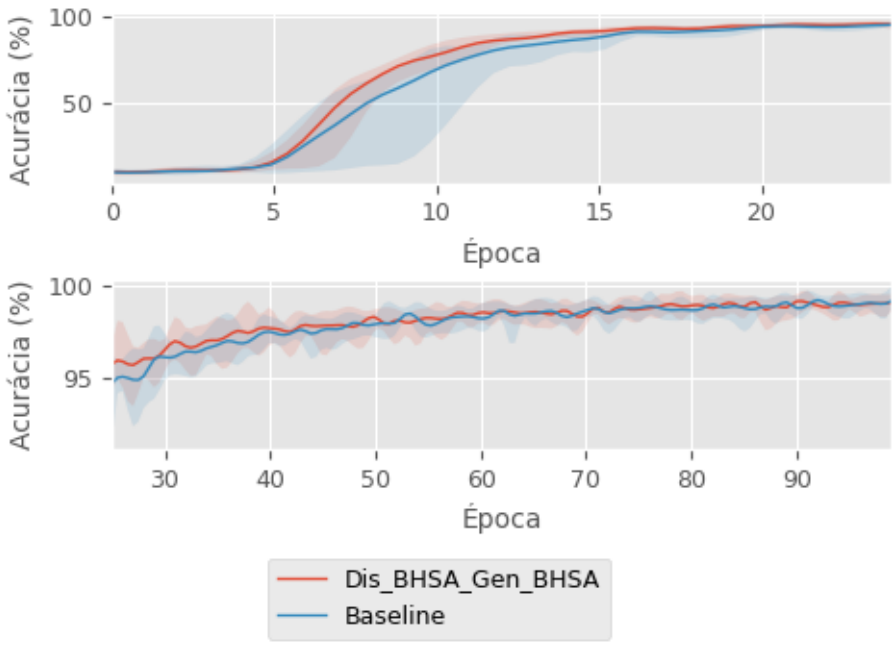


Figura 4.9: Acurácia média (com limites máximo e mínimo observado nos experimentos) do modelo *baseline* e AAF-GAN usando BHSA em ambas as redes adversárias.

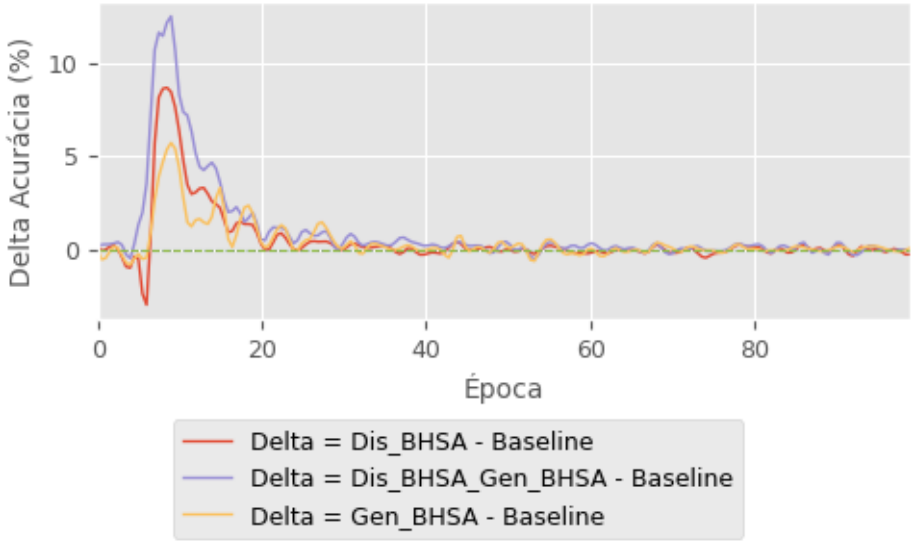


Figura 4.10: Gráfico do delta da acurácia dos modelos AAF-GAN usando BHSA e *baseline*: quanto mais positivo for o valor de delta, melhor foi o desempenho do AAF-GAN.

alguns ainda assim não convergiram, ou seja, a rede geradora não produziu resultados coerentes. Consequentemente os valores de acurácia média sofreram pelo fato de diversas rodadas não convergirem.

Afim de comparar se o mesmo tipo de falha de convergência ocorre na função BHAA original, realizamos experimentos com a função na sua versão primária com a estratégia de limitação do valor gerado. Definimos essa implementação específica

Tabela 4.4: Porcentagem de falhas computacionais e falhas de convergência nos modelos utilizando as funções BHANA e BHAA truncada (BHATA) durante a execução dos experimentos.

Função	**Nome do modelo**	**Experimentos com falha computacional (%)**	**Experimentos com falha de convergência (%)**
BHANA	Gen_BHANA	**0%**	**30%**
	Dis_BHANA	30%	85%
	Dis_BHATA_Gen_BHATA	50%	80%
BHATA	Gen_BHATA	**0%**	**0%**
	Dis_BHATA	**0%**	100%
	Dis_BHATA_Gen_BHATA	**0%**	100%

nos experimentos como BHATA (bi-hiperbólica assimétrica truncada adaptativa), onde os três modelos GAN são:

- Gen_BHATA: BHATA na camada de saída da rede geradora
- Dis_BHATA: BHATA na camada de saída da rede discriminadora
- Dis_BHATA_Gen_BHATA: BHATA na camada de saída de ambas as redes adversárias

Observamos que a função BHAA truncada (BHATA) não sofre do problema de *underflow*, já que sua implementação é exatamente a função BHAA original com valores truncados em pontos fora do intervalo de saída esperado. Porém, observamos que a versão truncada também não obtém bons resultados de convergência. Isso nos leva a acreditar que o problema de convergência da função BHANA é inerente à natureza matemática da função BHAA em si. Ou seja, encontrar uma metodologia com inicialização de pesos, arquitetura da rede e definição de hiperparâmetros para utilizar funções bi-hiperbólicas assimétricas é mais complicado quando comparado à utilização das versões de funções bi-hiperbólicas simétricas. Na tabela 4.4 listamos a porcentagem de experimentos que falharam computacionalmente (*underflow*) e que não convergiram para ambas as funções.

4.2.3 Experimento 3: SH-ReLU vs *baseline*

Neste terceiro experimento da parte 1, utilizamos a função SH-ReLU como substituta das funções retificadoras das camadas ocultas das redes. Os modelos AAF-GAN que utilizam SH-ReLU serão identificados como:

- GSHReLU: SH-ReLU nas camadas ocultas da rede geradora
- DSHReLU: SH-ReLU nas camadas ocultas da rede discriminadora
- DSHReLU_GSHReLU: SH-ReLU nas camadas ocultas de ambas as redes adversárias

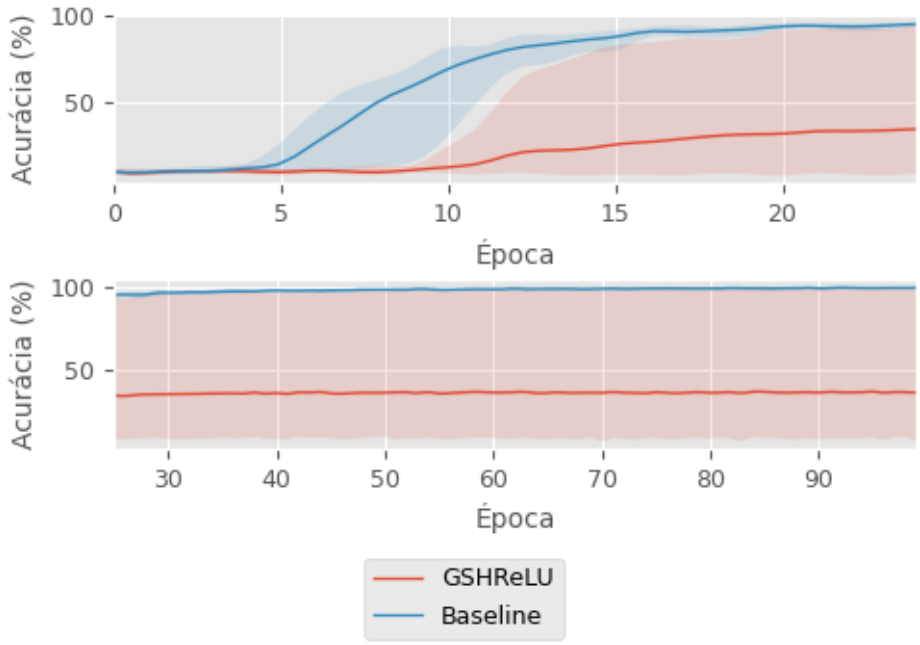


Figura 4.11: Acurácia média (com limites máximo e mínimo observado nos experimentos) do modelo *baseline* e AAF-GAN usando SH-ReLU na rede geradora.

As figuras 4.11, 4.12 e 4.13 possuem os gráficos de acurácia, respectivamente, dos modelos GSHReLU, DSHReLU e DSHReLU_GSHReLU. Notamos que o desempenho do AAF-GAN neste caso deixa muito a desejar comparado ao *baseline*. Nenhuma *seed* do modelo AAF-GAN com DSHReLU convergiu, enquanto que apenas um experimento do modelo DSHReLU_GSHReLU convergiu, e três do modelo GSHReLU.

4.2.4 Experimento 4: MiDA vs *baseline*

Nesta sub-seção avaliamos o desempenho da função MiDA. Ao substituirmos as funções estáticas das camadas ocultas obtemos três modelos de AAF-GAN usando a função MiDA:

- GMiDA: MiDA nas camadas ocultas da rede geradora
- DMiDA: MiDA nas camadas ocultas da rede discriminadora
- DMiDA_GMiDA: MiDA nas camadas ocultas de ambas as redes adversárias

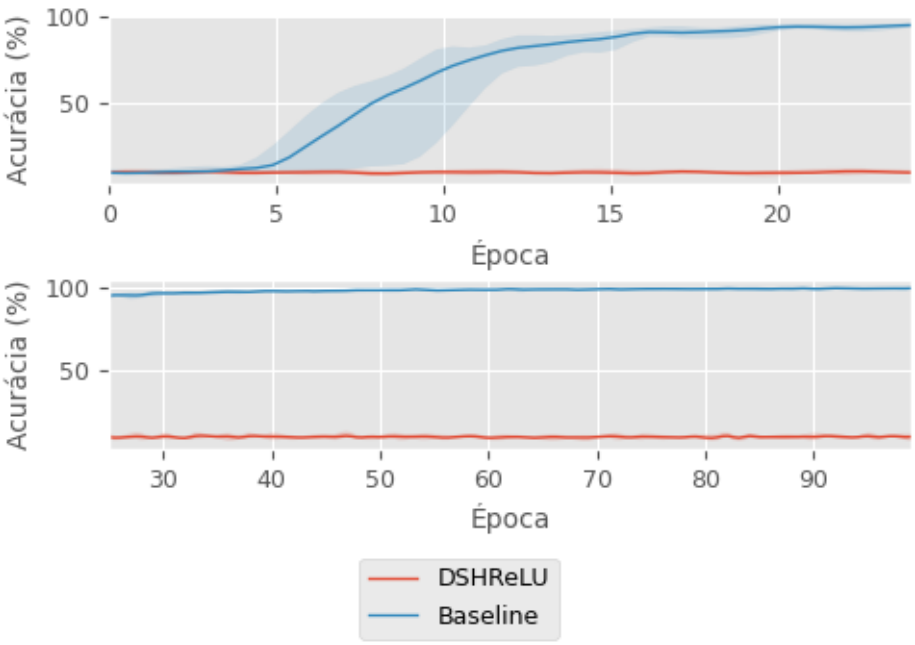


Figura 4.12: Acurácia média (com limites máximo e mínimo observado nos experimentos) do modelo *baseline* e AAF-GAN usando SH-ReLU na rede discriminadora.

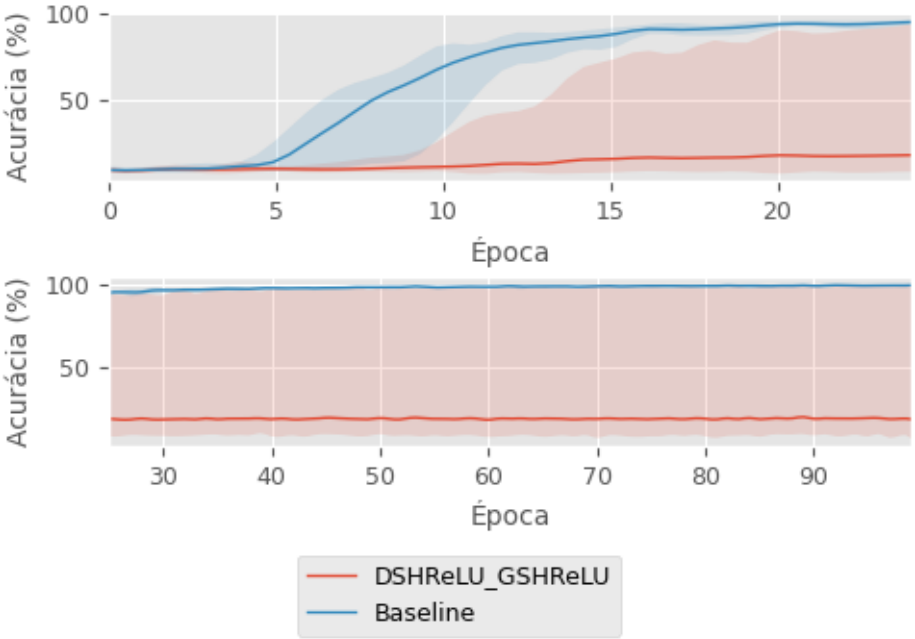


Figura 4.13: Acurácia média (com limites máximo e mínimo observado nos experimentos) do modelo *baseline* e AAF-GAN usando SH-ReLU em ambas as redes adversárias.

Os resultados mostram que o modelo AAF-GAN GMiDA consegue um desempenho notavelmente melhor que o modelo *baseline*, e se mantém em média melhor até a época 30. Diferentemente do modelo AAF-GAN usando BHSA (sub-seção 4.2.1), após as épocas iniciais, o GMiDA não mantém seu desempenho e passa a ser

mais fraco que o modelo base. Os modelos DMiDA e DMiDA_GMiDA não tiveram resultados satisfatórios comparados ao modelo padrão. Um experimento do modelo DMiDA inclusive não convergiu durante o treinamento. Nas figuras 4.14, 4.15 e 4.16 temos, respectivamente, os gráficos de acurácia dos modelos GMiDA, DMiDA e DMiDA_GMiDA.

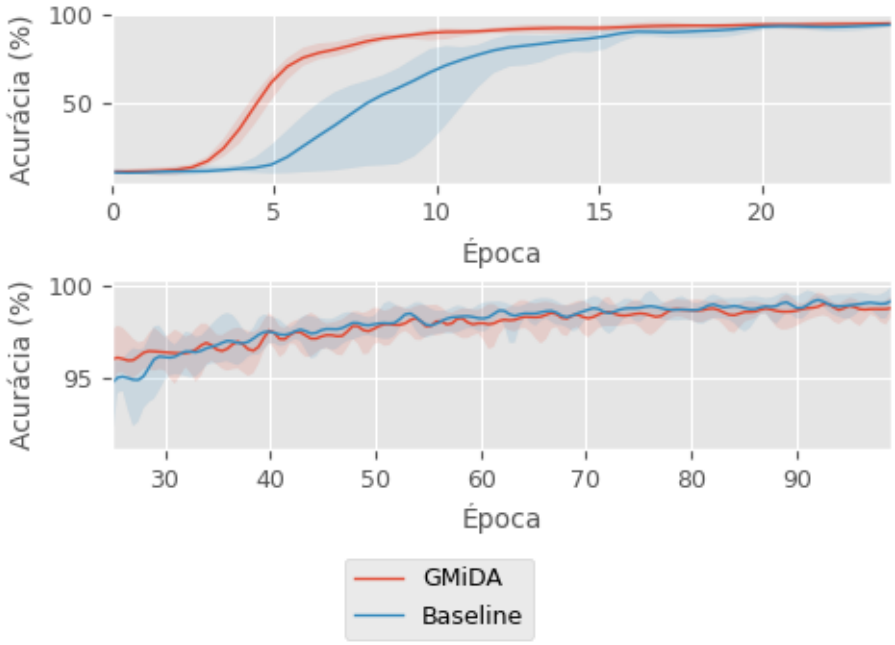


Figura 4.14: Acurácia média (com limites máximo e mínimo observado nos experimentos) do modelo *baseline* e AAF-GAN usando MiDA na rede geradora.

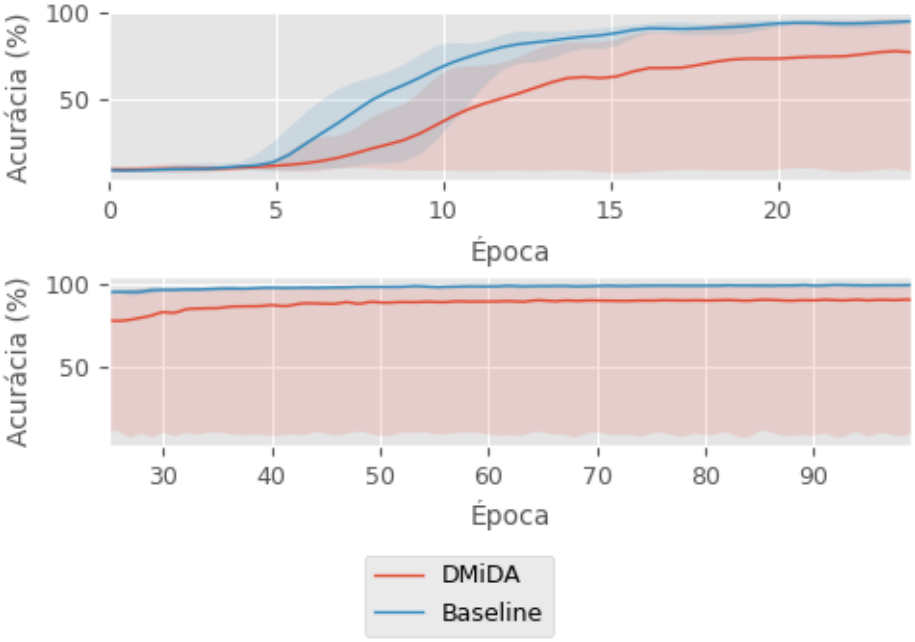


Figura 4.15: Acurácia média (com limites máximo e mínimo observado nos experimentos) do modelo *baseline* e AAF-GAN usando MiDA na rede discriminadora.

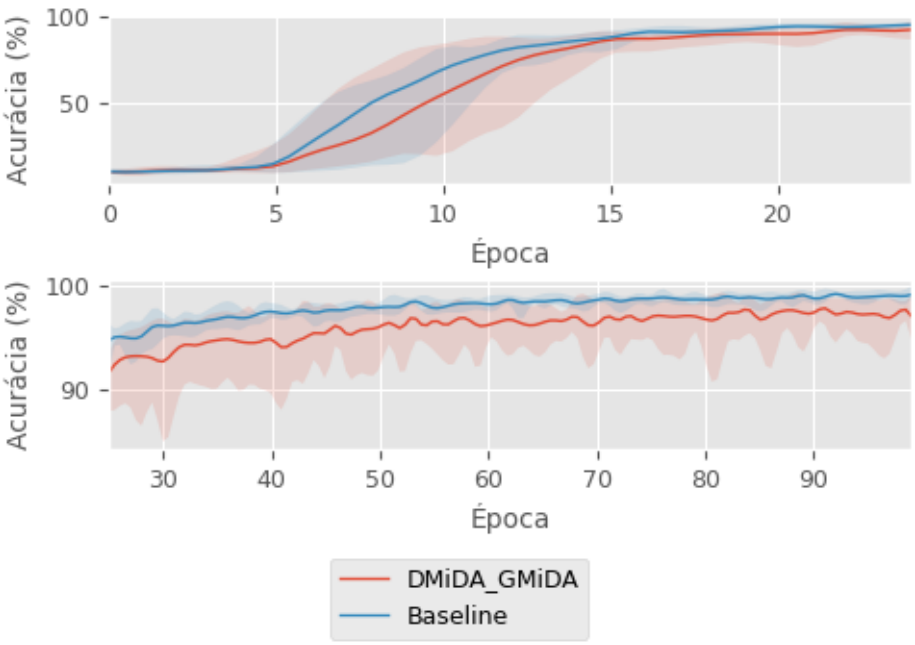


Figura 4.16: Acurácia média (com limites máximo e mínimo observado nos experimentos) do modelo *baseline* e AAF-GAN usando MiDA em ambas as redes adversárias.

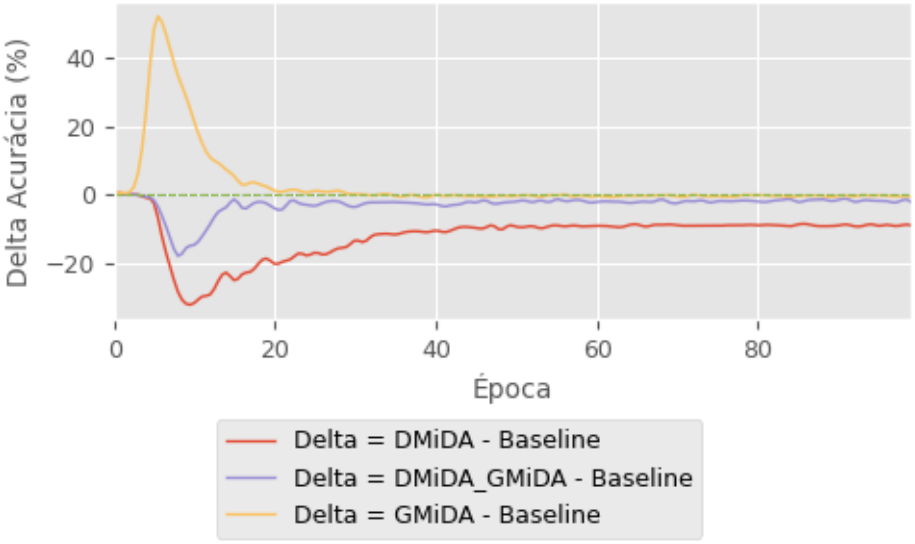


Figura 4.17: Gráfico do delta da acurácia dos modelos AAF-GAN usando MiDA e *baseline*: quanto mais positivo for o valor de delta, melhor foi o desempenho do AAF-GAN.

Neste ponto identificamos que há uma certa dificuldade inerente ao modelo de redes adversárias onde as redes discriminadoras são negativamente sensíveis ao uso de funções que não são as do modelo padrão em camadas ocultas. Assim como durante os experimentos da função SH-ReLU (sub-seção 4.2.3), os modelos AAF-GAN para funções de camadas ocultas que modificaram apenas a rede geradora obtiveram melhor desempenho comparado às outras duas substituições possíveis.

Podemos notar este comportamento para a função MiDA ao observarmos o gráfico na figura 4.17.

4.2.5 Experimento 5: MiDA vs Mish

Nesta sub-seção comparamos o desempenho da função MiDA com o desempenho da função Mish. Realizamos este experimento pois a função MiDA é baseada na função Mish, logo é de interesse identificar se a modificação da função adaptativa melhora, piora ou se iguala ao desempenho da função estática. Definimos mais três modelos de rede GAN utilizando a função de ativação Mish nas camadas ocultas da rede:

- GMish: Mish nas camadas ocultas da rede geradora
- DMish: Mish nas camadas ocultas da rede discriminadora
- DMish_GMish: Mish nas camadas ocultas de ambas as redes adversárias

Ao substituirmos as funções na rede geradora apenas, podemos notar que a performance do AAF-GAN com a função MiDA é consistentemente melhor do que o GAN usando Mish durante todo o treinamento. Já quando substituímos as funções na rede discriminadora, podemos notar que o a função MiDA começa pior do que a função Mish, e após a época 30 a função MiDA passa a obter um resultado melhor. Quando substituímos as funções em ambas as redes - modelos DMish_GMish e DMiDA_GMiDA -, observamos que os modelos possuem desempenho similar e que da época 30 em diante o modelo com funções Mish é em média melhor.

Já havíamos citado durante o experimento da sub-seção 4.2.4 que, segundo observado nos experimentos, modificar as funções retificadoras das camadas ocultas no discriminador tem um impacto negativo no resultado do treinamento. Isso é corroborado quando observamos que a função Mish aplicada ao descriminador tem desempenho melhor quando comparada com os mesmos modelos da MiDA. Porém, observamos a exceção ao final do treinamento do modelo DMiDA, onde o modelo AAF-GAN se sai melhor do que utilizar a função Mish.

Em relação à rede geradora, onde obtivemos melhores resultados em modelos que modificam as ativações internas da rede, a função MiDA representa um avanço à Mish levando em consideração que a primeira se sai consistentemente melhor do que a segunda.

4.2.6 Estatísticas gerais e ranking

Nesta sub-seção analisamos algumas estatísticas sobre a execução do modelo para consolidar as suposições feitas durante a análise de cada experimento desta Parte

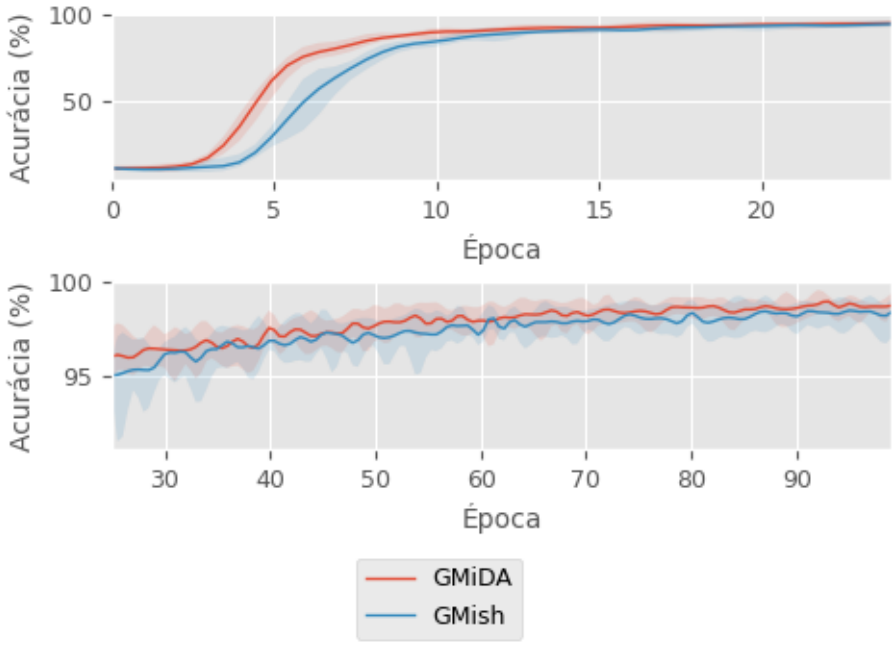


Figura 4.18: Acurácia média (com limites máximo e mínimo observado nos experimentos) dos modelos GAN usando Mish e AAF-GAN usando MiDA nas redes geradoras.

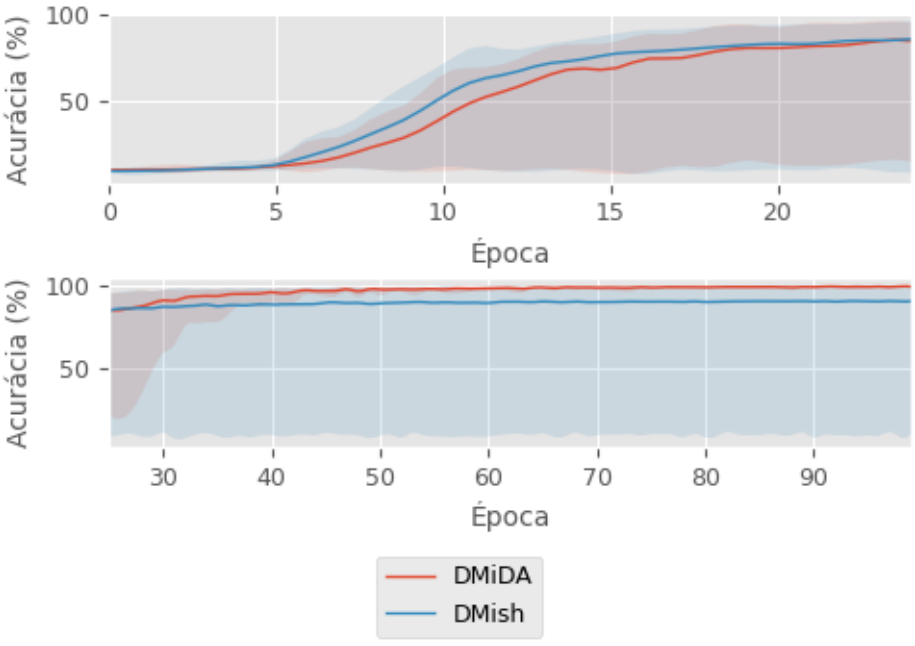


Figura 4.19: Acurácia média (com limites máximo e mínimo observado nos experimentos) dos modelos GAN usando Mish e AAF-GAN usando MiDA nas redes discriminadoras.

1. As estatísticas foram computadas utilizando a média de cada respectiva época. Além disso, foram excluídos do cálculo da média rodadas em que o modelo não obteve convergência em nenhum momento do treinamento (pelo menos 90% de acurácia). As demais estatísticas utilizadas são descritas a seguir:

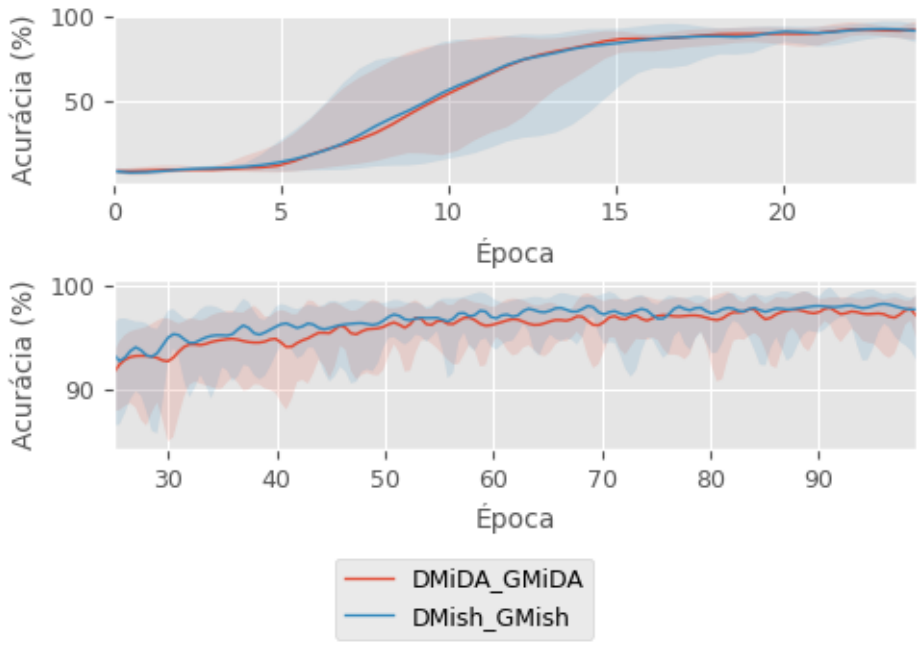


Figura 4.20: Acurácia média (com limites máximo e mínimo observado nos experimentos) dos modelos GAN usando Mish e AAF-GAN usando MiDA em ambas as redes adversárias.

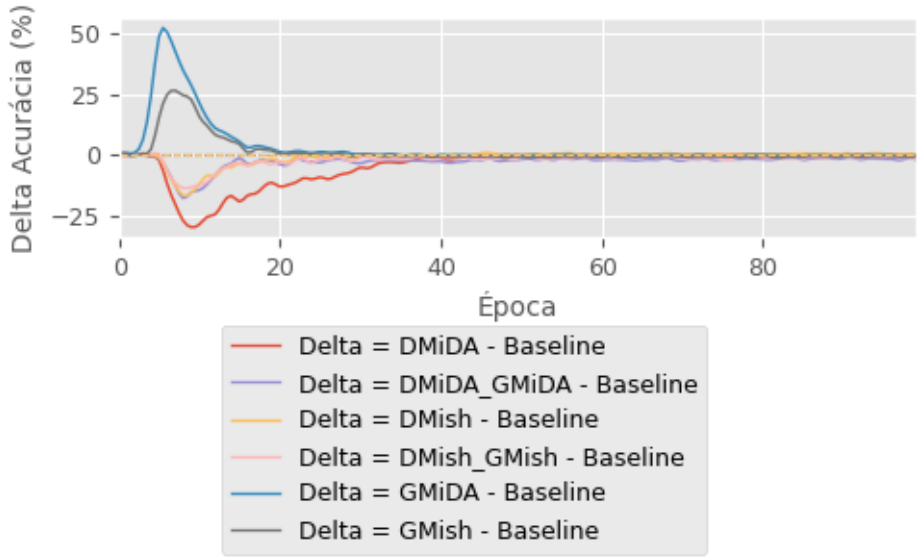


Figura 4.21: Gráfico do delta da acurácia (com o modelo *baseline*) dos modelos AAF-GAN usando MiDA e GAN usando Mish: quanto mais positivo for o valor de delta, melhor foi o desempenho do modelo.

- **Top acurácia**: melhor acurácia alcançada durante o treinamento no classificador pré-treinado. (tabela 4.6)
- **Top-5 acurácia**: média das cinco melhores acurácias alcançadas durante o treinamento no classificador pré-treinado. (tabela 4.6)
- **Épocas para atingir determinada acurácia**: quantidade de épocas para

atingir um certo valor de acurácia. Utilizamos 50%, 75% e 90% como valores de *thresholds*. (tabela 4.6)

- **Porcentagem de deltas positivos**: porcentagem de deltas positivos durante um determinado intervalo de épocas, onde delta é a subtração da acurácia do modelo corrente com o modelo *baseline*. Os intervalos de épocas utilizados foram [1, 50], [51, 100] e [1,100]. Porcentagens maiores que 50% significam que o modelo se mantém consistentemente melhor em média do que o *baseline* durante o intervalo analisado. (tabela 4.7)

- **Porcentagem de falhas computacionais e de convergência**: falhas computacionais representam a porcentagem de rodadas que o modelo resultou em erro de computação (*underflow*), enquanto que falha de convergência representa a porcentagem que o modelo não convergiu em nenhum momento do treinamento para uma acurácia aceitável (90% de acurácia foi o limite mínimo exigido para calcular esta estatística). (tabela 4.5)

Como listado na tabela 4.5, os modelos utilizando a função BHANA, SHReLU e BHATA não obtiveram quantidade suficiente de amostras de experimentos convergindo para que fossem levados em consideração na computação das estatísticas e, por isso, são omitidos das tabelas 4.6 e 4.7.

As estatísticas das tabelas 4.6 e 4.7 deixa explicito que modelos MiDA e Mish quando aplicados em ambas as redes adversárias não conseguem um desempenho melhor do que o *baseline*. Porém, quando aplicadas em apenas uma das redes, GMish e GMiDA obtém uma curva de convergência melhor no início do treinamento, enquanto que DMish e DMiDA conseguem superar o *baseline* no final do treinamento obtendo melhores valores de *top* acurácia.

Em relação às funções aplicadas na camada de saída, podemos notar que utilizar a função BHSA em ambas as redes adversárias torna o modelo consistentemente superior ao *baseline* se levarmos em consideração o delta das acurácias.

4.3 Parte 2: *CelebA-DCGAN*

Nesta segunda parte de experimentos, observamos a qualidade das imagens geradas das funções adaptativas em uma arquitetura DCGAN utilizandoo *dataset CelebA* (*CelebA-DCGAN*). Este modelo sem nenhuma modificação em relação à funções de ativação será referenciado como *baseline* durante os experimentos da Parte 2.

As imagens do conjunto de dados foram pré-processadas realizando um *center crop* de 178 *pixels* e logo em seguida redimensionadas para 64x64 *pixels*. Este passo é necessário pois a imagem no *dataset* original possui dimensões 218x178 e a arquitetura DCGAN empregada nos experimentos trabalha com imagens de dimensões

Tabela 4.5: Tabela de estatísticas sobre falhas computacionais (*underflow*) e de convergência dos experimentos realizados na Parte 1 (*MNIST-CondGAN*).

Modelo	**Falhas computacionais**	**Falhas de convergência**
baseline	0 de 10	10 de 10
Dis_BHSA	0 de 10	10 de 10
Dis_BHSA_Gen_BHSA	0 de 10	10 de 10
Gen_BHSA	0 de 10	10 de 10
Dis_BHANA	3 de 10	1 de 7
Dis_BHANA_Gen_BHANA	5 de 10	1 de 5
Gen_BHANA	0 de 10	7 de 10
Dis_BHATA	0 de 10	0 de 10
Dis_BHATA_Gen_BHATA	0 de 10	0 de 10
Gen_BHATA	0 de 10	10 de 10
DMiDA	0 de 10	9 de 10
DMiDA_GMiDA	0 de 10	10 de 10
GMiDA	0 de 10	10 de 10
DMish	0 de 10	9 de 10
DMish_GMish	0 de 10	10 de 10
GMish	0 de 10	10 de 10
DSHReLU	0 de 10	0 de 10
DSHReLU_GSHReLU	0 de 10	1 de 10
GSHReLU	0 de 10	3 de 10

Tabela 4.6: Tabela de estatísticas sobre a métrica de acurácia dos experimentos realizados na Parte 1 (*MNIST-CondGAN*).

Modelo	**Top Acurácia (%)**	**Top-5 Acurácia (%)**	**Épocas p/ acurácia 50%**	**Épocas p/ acurácia 75%**	**Épocas p/ acurácia 90%**
baseline	99.2	99.09	8	11	16
Dis_BHSA	99.13	99.03	8	10	16
Dis_BHSA_Gen_BHSA	99.12	99.08	8	10	14
Gen_BHSA	99.22	99.09	8	11	15
DMiDA	99.29	99.2	11	18	30
DMiDA_GMiDA	97.83	97.65	10	13	22
GMiDA	98.99	98.84	**5**	**6**	**10**
DMish	**99.56**	**99.41**	10	13	19
DMish_GMish	98.2	98.05	10	13	20
GMish	98.49	98.45	6	8	13

Tabela 4.7: Tabela de estatísticas sobre os deltas da métrica de acurácia entre modelos AAF-GAN e *baseline* durante os experimentos da Parte 1 (*MNIST-CondGAN*)

Modelo	**(%) Deltas positivos [1,50]**	**(%) Deltas positivos [51,100]**	**(%) Deltas positivos [1,100]**
baseline	0	0	0
Dis_BHSA	72	38	55
Dis_BHSA_Gen_BHSA	**96**	58	**77**
Gen_BHSA	72	50	61
DMiDA	8	46	27
DMiDA_GMiDA	6	0	3
GMiDA	72	8	40
DMish	18	**98**	58
DMish_GMish	8	0	4
GMish	58	0	29

64x64. Além disso, os canais RGB foram normalizados resultando em valores dentro do intervalo [-1, 1].

Durante os experimentos, analisaremos a métrica *FID score* dos exemplos gerados com o objetivo de determinar sua qualidade. Executamos cada modelo 5 vezes com inicializações (*seeds*) diferentes, 25 épocas e *mini-batches* de tamanho 128. Por fim, mantivemos as mesmas configurações de ruído de entrada empregados durante a Parte 1 dos experimentos.

Como o modelo convolucional DCGAN é mais robusto e seu treinamento é mais demorado por conta da dimensionalidade das redes, selecionamos os modelos com melhores desempenhos da Parte 1 de acordo com as métricas analisadas. Com isso, além do modelo *baseline*, executamos a parte 2 dos experimentos para os seguintes modelos:

- Dis_BHSA_Gen_BHSA: pois se mostrou, se não consistentemente melhor, no mínimo de desempenho similar comparado ao *baseline* durante todo o treinamento.
- GMiDA: pois obteve grande desempenho em épocas iniciais comparado ao *baseline*.
- GMish: para fins de comparação com o modelo GMiDA.

As imagens geradas pelos modelos desta Parte 2 dos experimentos para o conjunto de dados CelebA podem ser encontradas no apêndice B.

4.3.1 Experimento 6: BHSA vs *baseline*

No experimento 6 avaliamos o modelo AAF-GAN com BHSA aplicada em ambas as redes adversárias (Dis_BHSA_Gen_BHSA) no modelo *CelebA-DCGAN*. Nas figuras 4.22 e 4.23 podemos visualizar o desempenho deste modelo comparado ao *baseline* para a métrica *FID score*.

Podemos notar que neste experimento o desempenho do modelo AAF-GAN Dis_BHSA_Gen_BHSA se manteve consistentemente melhor do que o *baseline* em média até a época 10. Após este ponto, os modelos possuem desempenhos similares na métrica de avaliação.

4.3.2 Experimento 7: MiDA vs *baseline*

Neste experimento, analisamos os resultados da aplicação da função MiDA na rede geradora individualmente. Na figura 4.24 temos o resultado do *FID score* para o modelo AAF-GAN GMiDA. Podemos observar neste gráfico o mesmo comportamento identificado na sub-seção 4.2.4: o modelo proposto tem convergência melhor no início do treinamento e, a partir de determinada época (época 7, neste caso) o modelo *baseline* passa a se sair melhor na geração das imagens. Na figura 4.25 podemos observar o delta do *FID score* entre os modelos AAF-GAN GMiDA e o *baseline*.

4.3.3 Experimento 8: MiDA vs Mish

Assim como realizado na Parte 1 dos experimentos, executamos os modelos GAN usando Mish para compararmos seu desempenho com a função MiDA.

Podemos notar pela figura 4.26 que a aplicação da função MiDA não obteve grandes vantagens no modelo convolucional GAN comparada à função Mish. A aplicação da função MiDA nas camadas ocultas da rede geradora não gera instabilidade no aprendizado e, como visto na sub-seção 4.3.2, obtém melhor convergência comparado ao *baseline*. Porém, seu desempenho comparado ao uso da função Mish no gerador são similares.

4.3.4 Estatísticas gerais e ranking

Analisamos nesta sub-seção estatísticas de desempenho dos modelos durante a Parte 2 dos experimentos. Assim como realizado para calcular o *FID score* médio, as estatísticas a seguir foram calculadas utilizando o valor médio observado para cada experimento. Todas as diferentes inicializações randômicas convergiram nesta Parte 2 do experimento, e por isso omitiremos a tabela de convergência como apresentada na sub-seção 4.2.6.

Tabela 4.8: Tabela de estatísticas sobre a métrica *FID score* dos experimentos realizados na Parte 2 (*CelebA-DCGAN*).

Modelo	**Top FID**	**Top-5 FID**	**Épocas para FID = 50**	**Épocas para FID = 30**	**Épocas para FID = 15**
baseline	**13.33**	**13.49**	4	6	**12**
Dis_BHSA_Gen_BHSA	**13.2**	**13.67**	**2**	**4**	**12**
GMiDA	14.62	15.09	**2**	**4**	24
GMish	15.16	15.36	**2**	5	Nunca

Tabela 4.9: Tabela de estatísticas sobre os deltas da métrica de *FID score* entre modelos AAF-GAN e *baseline* durante os experimentos da Parte 2 (*CelebA-DCGAN*).

Modelo	**(%) Deltas positivos [1,10]**	**(%) Deltas positivos [11,25]**	**(%) Deltas positivos [1,25]**
baseline	0	0	0
Dis_BHSA_Gen_BHSA	**100**	**53**	**72**
GMiDA	**80**	0	32
GMish	**100**	0	40

As estatísticas computadas para esta seção estão descritas a seguir:

- **Top *FID score***: melhor *FID score* alcançado em média durante o treinamento dos modelos. (tabela 4.8)
- **Top-5 *FID score***: média dos cinco melhores valores de *FID score*. (tabela 4.8)
- **Épocas para atingir determinado *FID score***: número de épocas que o modelo leva em média para alcançar um certo valor de *FID score*. Utilizamos os valores 50, 30 e 15 para tal estatística. (tabela 4.8)
- **Porcentagem de deltas positivos**: porcentagem de deltas entre os modelos propostos e o *baseline* positivos dentro de um intervalo de épocas. Os intervalos utilizados foram [1,10], [10,25] e [1,25]. (tabela 4.9)

Podemos visualizar que pelos resultados das tabelas 4.8 e 4.9 que o modelo AAF-GAN Dis_BHSA_Gen_BHSA obteve os melhores resultados. Seu desempenho no início do treinamento é tão bom quanto aos modelo utilizando funções MiDA e Mish, porém se mantém competitivo com o modelo *baseline* também em épocas mais tardias do treinamento. Já os modelos utilizando MiDA e Mish mantiveram seu comportamento onde iniciam melhores comparados ao *baseline* mas não conseguem manter seu desempenho superior após as épocas iniciais.

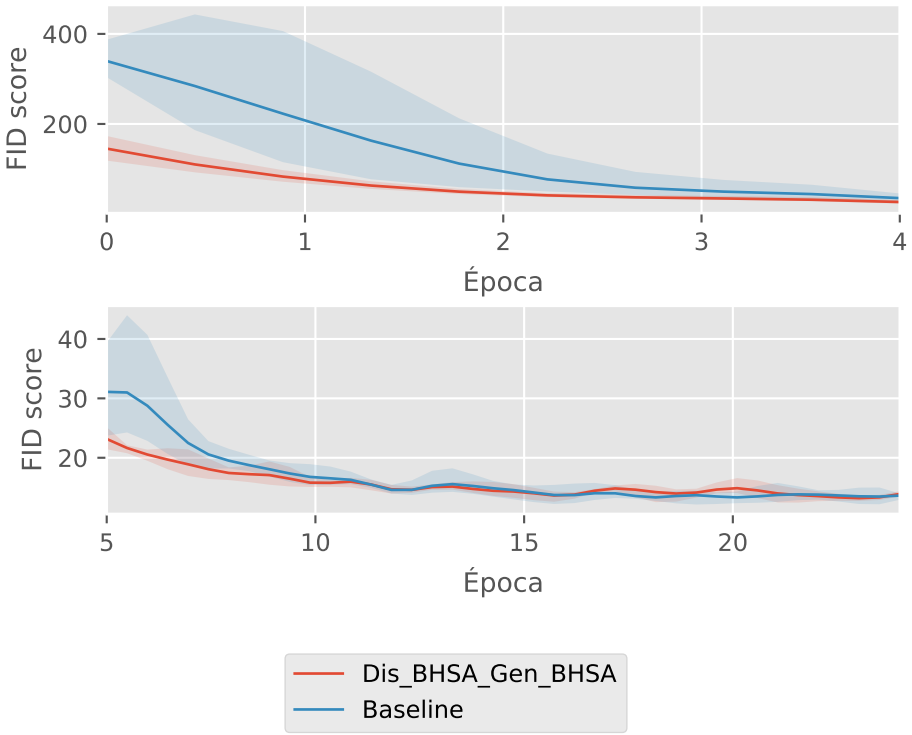


Figura 4.22: *FID score* médio (com limites máximo e mínimo observado nos experimentos) do modelo *baseline* e AAF-GAN usando BHSA em ambas as redes adversárias.

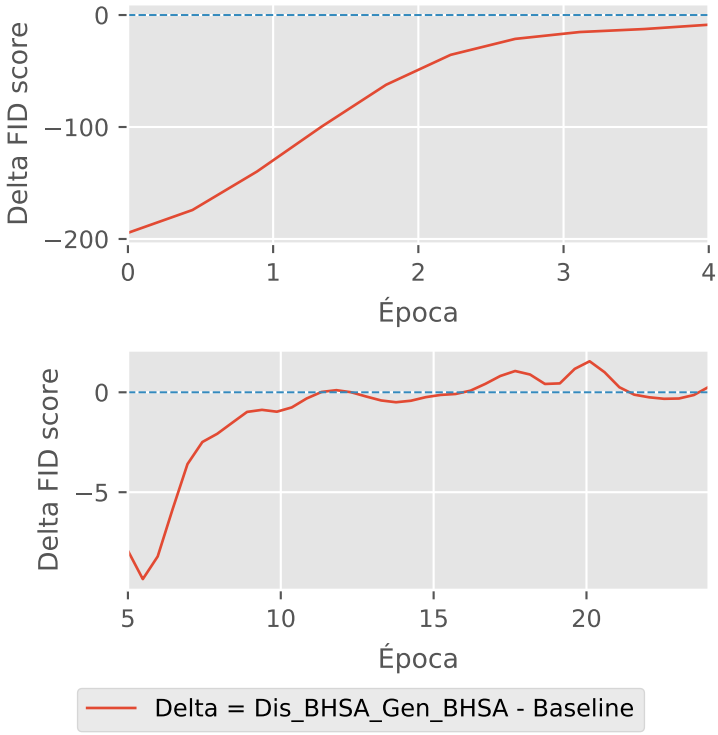


Figura 4.23: Gráfico do delta do *FID score* dos modelos AAF-GAN Dis_BHSA_Gen_BHSA e *baseline*: quanto mais negativo for o valor de delta, melhor foi o desempenho do AAF-GAN.

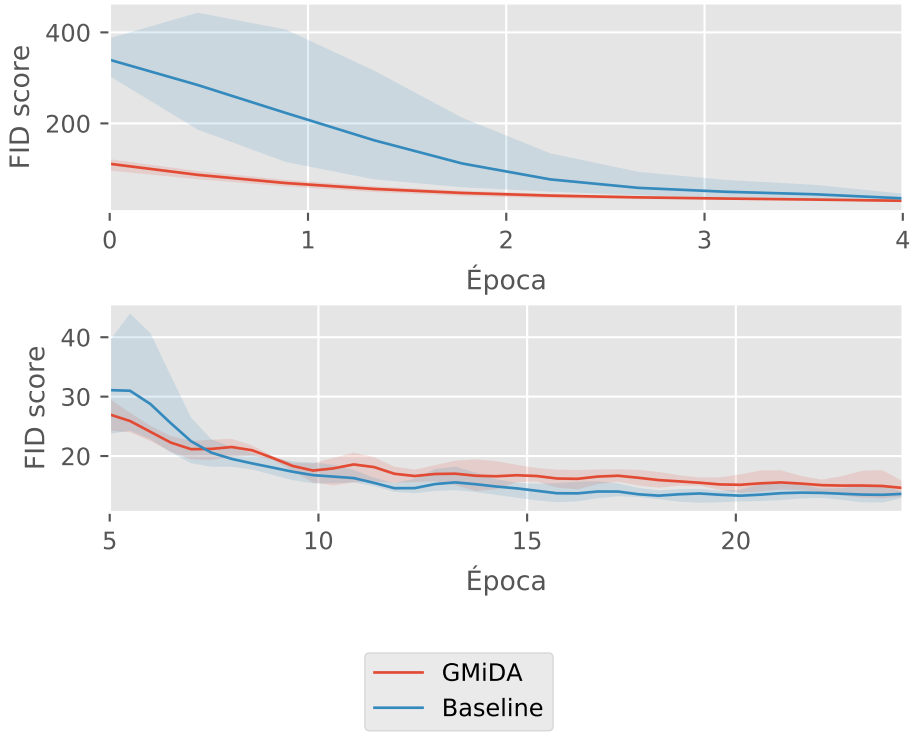


Figura 4.24: *FID score* médio (com limites máximo e mínimo observados nos experimentos) do modelo *baseline* e AAF-GAN usando MiDA na rede geradora.

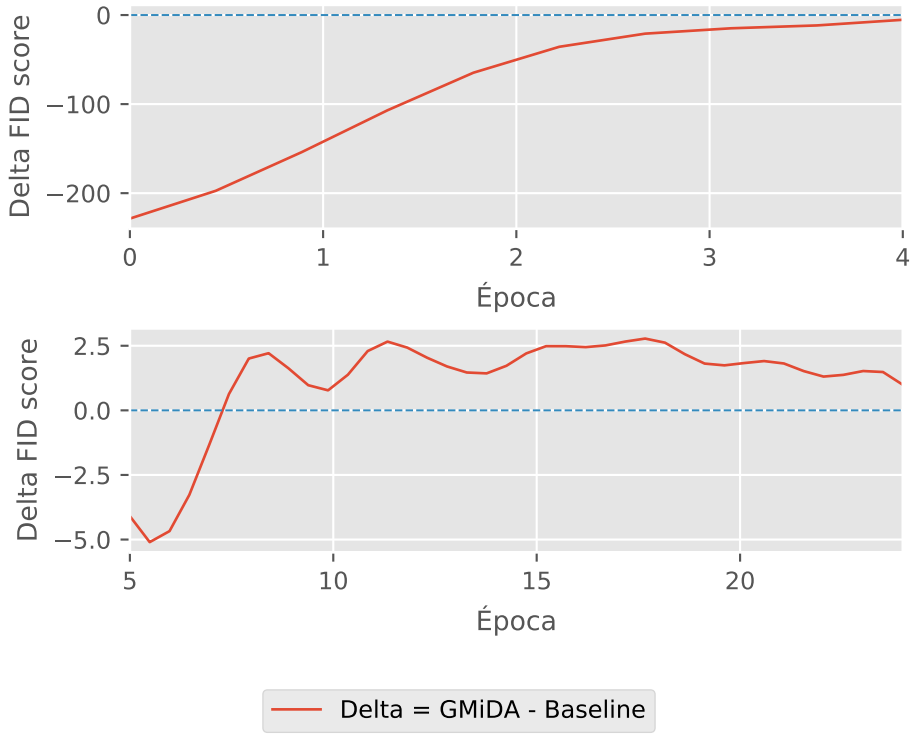


Figura 4.25: Gráfico delta do *FID score* dos modelos AAF-GAN usando MiDA no gerador e *baseline*: quanto mais negativo for o valor de delta, melhor foi o desempenho do AAF-GAN.

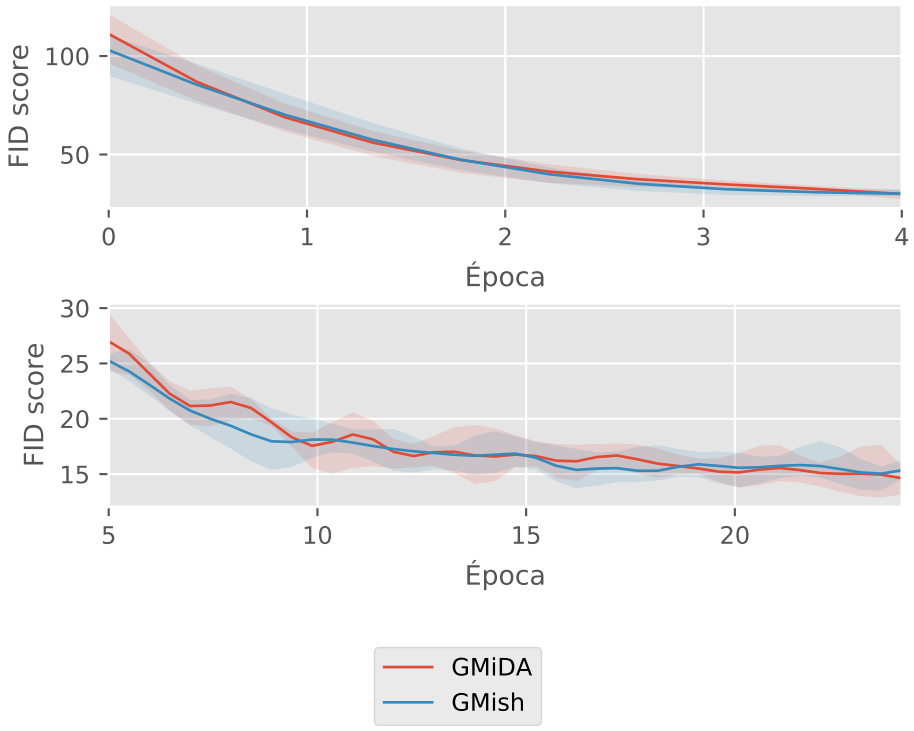


Figura 4.26: *FID score* médio (com limites máximo e mínimo observados nos experimentos) dos modelos GMish e GMiDA.

Capítulo 5

Conclusões

O capítulo 5 é dedicado à discussão sobre as considerações finais e dificuldades encontradas durante o desenvolvimento do presente trabalho. Por fim, encerramos o texto pontuando sobre os possíveis trabalhos futuros que podem direcionar investigações adicionais de pesquisa na área.

5.1 Considerações finais

Durante o desenvolvimento do presente trabalho abordamos uma área de pesquisa sobre funções de ativação adaptativas até então não explorada em redes neurais GAN. Iniciamos então seu estudo com a proposta de aplicação de funções adaptativas nas camadas ocultas ou de saída das redes utilizando funções baseadas em penalização hiperbólica como ativação dos neurônios. A hipótese de que tais funções poderiam modificar o desempenho da rede GAN em épocas iniciais foi levada como a premissa de que poderíamos melhorar a qualidade das amostras geradas pela rede geradora. Exploramos então a aplicação de diversas funções adaptativas hiperbólicas e como tais funções influenciaram na qualidade das amostras dependendo de qual rede era modificada.

Além disso, destacamos o problema do intervalo de atuação da função BHAA, onde a extrapolação dos valores leva à inconsistência durante o processo de aprendizado da rede ocasionado por presunções características das funções de custo utilizadas.

Por fim, observamos que na literatura uma recente proposta de função de ativação estática era capaz de obter melhores resultados, ou no mínimo competitivos, quando comparada à ativações retificadoras tradicionais. Com isso, utilizamos a função Mish como base substituindo em sua fórmula o componente responsável pela propriedade de *self-gating* com o objetivo de analisar se era possível alavancar seu desempenho através de uma função hiperbólica adaptativa.

5.2 Contribuições

A principal contribuição deste trabalho é a validação da aplicação de funções hiperbólicas adaptativas em redes GAN. Durante os experimentos, pudemos identificar as funções que obtiveram um desempenho de convergência mais acelerada em relação ao modelo *baseline*, além de qual estratégia de aplicação seria a mais adequada. As descobertas formam uma diretriz inicial para a aplicação direta das funções abordadas em redes GAN, listadas a seguir:

- A função BHSA pode ser aplicada como substituta na camada de saída de ambas as redes adversárias (individualmente ou simultaneamente) para causar um impacto positivo de convergência inicial do modelo e mantendo um bom desempenho comparado ao *baseline* em épocas mais adiante.
- Utilizar a função BHSA em ambas as redes adversárias simultaneamente se mostrou ser a melhor combinação sendo consistentemente melhor, ou no mínimo competitiva, que o modelo *baseline*.
- Utilizar a função MiDA nas camadas ocultas da rede geradora impulsiona a convergência inicial do modelo não-convolucional, superando inclusive a melhora que a função Mish proporciona.
- No modelo convolucional, a função MiDA tem desempenho similar à Mish e não caracterizou nenhuma melhora ou piora nos resultados quando comparadas.
- O uso das funções MiDA e Mish na rede geradora pode melhorar a convergência inicial, porém ao custo de piorar o resultado em épocas mais avançadas do treinamento.
- No geral, a rede discriminadora se mostrou mais sensível à substituição por funções adaptativas em camadas ocultas, ocasionando em um impacto negativo para o treinamento.

Além das diretrizes, outras duas contribuições no presente trabalho são da formulação de duas novas funções de ativação adaptativas. A função BHANA, que apesar de computacionalmente limitada, é formulada como uma alternativa matemática para a versão original solucionando o problema de extrapolação dos limites de atuação. Já a função MiDA é proposta como uma alternativa com parâmetros adaptativos à função Mish com o potencial de melhorar a convergência de modelos não-convolucionais especialmente em épocas iniciais.

5.3 Desafios encontrados

Durante o trabalho, pudemos identificar desafios e dificuldades que de certa forma limitaram o escopo do trabalho realizado. Propusemos uma metodologia onde não modificaríamos a arquitetura e nem as características das redes adversárias com o intuito de identificar individualmente o impacto das funções adaptativas aplicadas. A quantidade de combinações de experimentos a serem realizados caso decidíssemos variar tais características do modelo culminaria em uma dimensionalidade de experimentos não adequadas para um único trabalho.

Como consequência, não abordamos diferentes inicializações de pesos das redes, diferentes profundidades de camadas e inicializações de hiperparâmetros. Além disso, restringimos o escopo de inicialização dos pesos das funções adaptativas para que seu formato fosse o mais similar inicialmente com a função tradicional a ser substituída. O presente trabalho limitou seu escopo às inicializações advindas da metodologia do formato da função, não explorando o impacto de diferentes estratégias de inicialização dos pesos ou de estratégias de substituição.

Por fim, destacamos um dos principais desafios que se diz respeito à implementação da função BHANA. Apesar de matematicamente correta, a versão normalizada da função BHAA se mostrou computacionalmente limitada em relação aos recursos de precisão de cálculo necessário.

5.4 Trabalhos futuros

A partir dos resultados encontrados e dos desafios do estudo realizado, identificamos algumas das sugestões de trabalhos futuros que podem dar continuidade para as pesquisas nesta área:

- Realizar um estudo com o escopo limitado à uma função adaptativa com o objetivo de detalhar as melhores inicializações, arquiteturas e hiperparâmetros a serem utilizados. Deixamos aqui a sugestão das funções BHSA e MiDA como candidatas devido ao seu desempenho observado.

- Investigar como estabilizar o treinamento ao aplicar funções adaptativas hiperbólicas em camadas internas da rede discriminadora.

- Averiguar o desempenho de tais funções em outras arquiteturas de redes GAN. Como visto no capítulo de introdução, há diversos estudos e modelos diferentes de redes GAN que poderiam se beneficiar dos mesmos resultados positivos obtidos durante os experimentos deste trabalho.

- Aplicar outros tipos de funções adaptativas em redes neurais adversárias com o objetivo de melhorar seu desempenho, já que o presente trabalho limitou o escopo à versões de funções adaptativas hiperbólicas.

- Estudar uma nova versão da função Mish, mantendo as mudanças da função MiDA e substituindo também a função $softmax$ pela função SH-ReLU.

- Estudar o impacto e o desempenho da função MiDA em modelos que não sejam GAN. A função Mish foi originalmente proposta e aplicada em modelos de redes neurais para resolver tarefas de classificação. Logo, espera-se que a função MiDA possa ocasionar em uma melhora em tais classificadores assim como foi possível observar nas arquiteturas estudas nesta dissertação.

- Desenvolver uma técnica de implementação para a função BHANA que seja computacionalmente viável, ou uma nova versão normalizada que seja matematicamente coerente e computacionalmente possível de implementação com os recursos de tecnologia atuais.

Referências Bibliográficas

ABDEL-HAMID, O., MOHAMED, A., JIANG, H., et al., 2014, "Convolutional Neural Networks for Speech Recognition", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, v. 22, n. 10, pp. 1533–1545. doi: 10.1109/TASLP.2014.2339736.

AGOSTINELLI, F., HOFFMAN, M. D., SADOWSKI, P. J., et al., 2015, "Learning Activation Functions to Improve Deep Neural Networks". In: Bengio, Y., LeCun, Y. (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*. Disponível em: <http://arxiv.org/abs/1412.6830>.

BENGIO, Y., SIMARD, P., FRASCONI, P., 1994, "Learning long-term dependencies with gradient descent is difficult", *IEEE Transactions on Neural Networks*, v. 5, n. 2, pp. 157–166. doi: 10.1109/72.279181.

BISHOP, C. M., 2006, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 1 ed. New York, NY, Springer. ISBN: 0387310738.

BORJI, A., 2018, "Pros and Cons of GAN Evaluation Measures", *CoRR*, v. abs/1802.03446. Disponível em: <http://arxiv.org/abs/1802.03446>.

CANALLI, Y. D. M., 2017, *Funções de ativação hiperbólicas em redes neurais*. Tese de Doutorado, Universidade Federal do Rio de Janeiro.

CHEN, X., DUAN, Y., HOUTHOOFT, R., et al., 2016. "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets". .

CRESWELL, A., BHARATH, A. A., SENGUPTA, B., 2017, "Conditional Autoencoders with Adversarial Information Factorization", *CoRR*, v. abs/1711.05175. Disponível em: <http://arxiv.org/abs/1711.05175>.

DOS SANTOS, C., GATTI, M., 2014, "Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts". In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pp. 69–78, Dublin, Ireland, ago. Dublin City University and Association for Computational Linguistics. Disponível em: <https://www.aclweb.org/anthology/C14-1008>.

ERHAN, D., SZEGEDY, C., TOSHEV, A., et al., 2014, "Scalable Object Detection using Deep Neural Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

FUKUSHIMA, K., 1980, "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position", *Biological Cybernetics*, v. 36, pp. 193–202.

GANJU, K., WANG, Q., YANG, W., et al., 2018, "Property Inference Attacks on Fully Connected Neural Networks Using Permutation Invariant Representations". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, p. 619–633, New York, NY, USA. Association for Computing Machinery. ISBN: 9781450356930. doi: 10.1145/3243734.3243834. Disponível em: <https://doi.org/10.1145/3243734.3243834>.

GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., et al., 2014, "Generative Adversarial Nets". In: Ghahramani, Z., Welling, M., Cortes, C., et al. (Eds.), *Advances in Neural Information Processing Systems 27*, Curran Associates, Inc., pp. 2672–2680. Disponível em: <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>.

GOODFELLOW, I., BENGIO, Y., COURVILLE, A., 2016, *Deep Learning*. Cambridge, MA, USA, MIT Press. http://www.deeplearningbook.org.

GOODFELLOW, I. J., 2015. "On distinguishability criteria for estimating generative models". .

GOODFELLOW, I. J., 2017, "NIPS 2016 Tutorial: Generative Adversarial Networks", *CoRR*, v. abs/1701.00160. Disponível em: <http://arxiv.org/abs/1701.00160>.

GRAVES, A., JAITLY, N., 2014, "Towards End-To-End Speech Recognition with Recurrent Neural Networks". In: Xing, E. P., Jebara, T. (Eds.), *Proceedings of the 31st International Conference on Machine Learning*, v. 32, *Proceedings of Machine Learning Research*, pp. 1764–1772, Bejing, China,

22–24 Jun. PMLR. Disponível em: <http://proceedings.mlr.press/v32/graves14.html>.

GREGOR, K., DANIHELKA, I., GRAVES, A., et al., 2015, "DRAW: A Recurrent Neural Network For Image Generation". In: Bach, F., Blei, D. (Eds.), *Proceedings of the 32nd International Conference on Machine Learning*, v. 37, *Proceedings of Machine Learning Research*, pp. 1462–1471, Lille, France, 07–09 Jul. PMLR. Disponível em: <http://proceedings.mlr.press/v37/gregor15.html>.

HE, K., ZHANG, X., REN, S., et al., 2015, "Deep Residual Learning for Image Recognition", *CoRR*, v. abs/1512.03385. Disponível em: <http://arxiv.org/abs/1512.03385>.

HEUSEL, M., RAMSAUER, H., UNTERTHINER, T., et al., 2017, "GANs Trained by a Two Time-Scale Update Rule Converge to a Nash Equilibrium", *CoRR*, v. abs/1706.08500. Disponível em: <http://arxiv.org/abs/1706.08500>.

HONG, Y., HWANG, U., YOO, J., et al., 2019, "How Generative Adversarial Networks and Their Variants Work: An Overview", *ACM Comput. Surv.*, v. 52, n. 1 (fev.). ISSN: 0360-0300. doi: 10.1145/3301282. Disponível em: <https://doi.org/10.1145/3301282>.

HORNIK, K., STINCHCOMBE, M., WHITE, H., 1989, "Multilayer feedforward networks are universal approximators", *Neural Networks*, v. 2, n. 5, pp. 359–366.

HOU, L., SAMARAS, D., KURC, T., et al., 2017, "ConvNets with Smooth Adaptive Activation Functions for Regression". In: Singh, A., Zhu, J. (Eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, v. 54, *Proceedings of Machine Learning Research*, pp. 430–439, Fort Lauderdale, FL, USA, 20–22 Apr. PMLR. Disponível em: <http://proceedings.mlr.press/v54/hou17a.html>.

JAGTAP, A. D., KAWAGUCHI, K., KARNIADAKIS, G. E., 2020, "Adaptive activation functions accelerate convergence in deep and physics-informed neural networks", *Journal of Computational Physics*, v. 404 (Mar), pp. 109136. ISSN: 0021-9991. doi: 10.1016/j.jcp.2019.109136. Disponível em: <http://dx.doi.org/10.1016/j.jcp.2019.109136>.

JIANG, H., LEARNED-MILLER, E., 2017, "Face Detection with the Faster R-CNN". In: *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*, pp. 650–657. doi: 10.1109/FG.2017.82.

KARRAS, T., LAINE, S., AILA, T., 2019, "A style-based generator architecture for generative adversarial networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410.

KIM, D., KIM, M., KWON, G., et al., 2019, "Progressive Face Super-Resolution via Attention to Facial Landmark", *CoRR*, v. abs/1908.08239. Disponível em: <http://arxiv.org/abs/1908.08239>.

KINGMA, D. P., BA, J., 2015, "Adam: A Method for Stochastic Optimization". In: Bengio, Y., LeCun, Y. (Eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Disponível em: <http://arxiv.org/abs/1412.6980>.

KINGMA, D. P., WELLING, M., 2014, "Auto-Encoding Variational Bayes". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E., 2012, "ImageNet Classification with Deep Convolutional Neural Networks". In: Pereira, F., Burges, C. J. C., Bottou, L., et al. (Eds.), *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., pp. 1097–1105. Disponível em: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.

LAWRENCE, S., GILES, C. L., AH CHUNG TSOI, et al., 1997, "Face recognition: a convolutional neural-network approach", *IEEE Transactions on Neural Networks*, v. 8, n. 1, pp. 98–113. doi: 10.1109/72.554195.

LECUN, Y., 1985, "Une procédure d'apprentissage pour réseau à seuil asymétrique", *Proceedings of Cognitiva 85, Paris*, pp. 599–604.

LECUN, Y., BOSER, B., DENKER, J. S., et al., 1989, "Backpropagation Applied to Handwritten Zip Code Recognition", *Neural Computation*, v. 1, pp. 541–551.

LECUN, Y., CORTES, C., 2010, "MNIST handwritten digit database", Disponível em: <http://yann.lecun.com/exdb/mnist/>.

LECUN, Y., BOTTOU, L., BENGIO, Y., et al., 1998, "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE*, pp. 2278–2324.

LIN, C. H., CHANG, C., CHEN, Y., et al., 2019, "COCO-GAN: Generation by Parts via Conditional Coordinating", *CoRR*, v. abs/1904.00284. Disponível em: <http://arxiv.org/abs/1904.00284>.

LINNAINMAA, S., 1970, *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. Tese de Mestrado, Univ. Helsinki.

LINNAINMAA, S., 1976, "Taylor expansion of the accumulated rounding error", *BIT Numerical Mathematics*, v. 16, n. 2, pp. 146–160.

LIU, Z., LUO, P., WANG, X., et al., 2015, "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*, December.

LU, L., SHIN, Y., SU, Y., et al., 2020. "Dying ReLU and Initialization: Theory and Numerical Examples". .

MAAS, A. L., HANNUN, A. Y., NG, A. Y., 2013, "Rectifier nonlinearities improve neural network acoustic models". In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

MCCULLOCH, W., PITTS, W., 1943, "A Logical Calculus of Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, v. 5, pp. 127–147.

MENG JOO ER, SHIQIAN WU, JUWEI LU, et al., 2002, "Face recognition with radial basis function (RBF) neural networks", *IEEE Transactions on Neural Networks*, v. 13, n. 3, pp. 697–710. doi: 10.1109/TNN.2002.1000134.

MENICK, J., KALCHBRENNER, N., 2018, "Generating High Fidelity Images with Subscale Pixel Networks and Multidimensional Upscaling", *CoRR*, v. abs/1812.01608. Disponível em: <http://arxiv.org/abs/1812.01608>.

MIGUEZ, G. A., 2012, *Otimização do Algoritmo de Backpropagation Pelo Uso da Função de Ativação Bi-Hiperbólica*. Tese de Doutorado, Universidade Federal do Rio de Janeiro.

MINSKY, M., PAPERT, S., 1969, *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA, MIT Press.

MIRSAMADI, S., BARSOUM, E., ZHANG, C., 2017, "Automatic speech emotion recognition using recurrent neural networks with local attention". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2227–2231. doi: 10.1109/ICASSP.2017.7952552.

MIRZA, M., OSINDERO, S., 2014. "Conditional Generative Adversarial Nets". .

MISRA, D., 2020. "Mish: A Self Regularized Non-Monotonic Activation Function". .

NIELSEN, R. H., 1989, "Theory of the Backpropagation Neural Network". In: *Proceedings of the International Joint Conference on Neural Networks* (Washington, DC), v. I, pp. 593–605. Piscataway, NJ: IEEE.

PASZKE, A., GROSS, S., CHINTALA, S., et al., 2017, "Automatic differentiation in PyTorch". In: *NIPS-W*.

QIAN, S., LIU, H., LIU, C., et al., 2018, "Adaptive Activation Functions in Convolutional Neural Networks", v. 272, n. C (jan.), pp. 204–212. ISSN: 0925-2312. doi: 10.1016/j.neucom.2017.06.070. Disponível em: <https://doi.org/10.1016/j.neucom.2017.06.070>.

RADFORD, A., METZ, L., CHINTALA, S., 2015. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". Disponível em: <http://arxiv.org/abs/1511.06434>. cite arxiv:1511.06434Comment: Under review as a conference paper at ICLR 2016.

RAMACHANDRAN, P., ZOPH, B., LE, Q. V., 2017, "Searching for Activation Functions", *CoRR*, v. abs/1710.05941. Disponível em: <http://arxiv.org/abs/1710.05941>.

ROSENBLATT, F., 1958, "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review*, v. 65, n. 6, pp. 386–408. ISSN: 0033-295X. doi: 10.1037/h0042519. Disponível em: <http://dx.doi.org/10.1037/h0042519>.

RUDER, S., 2016. "An overview of gradient descent optimization algorithms". Disponível em: <http://arxiv.org/abs/1609.04747>.

RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J., 1986, "Learning Representations by Back-propagating Errors", *Nature*, v. 323, n. 6088, pp. 533–536. doi: 10.1038/323533a0. Disponível em: <http://www.nature.com/articles/323533a0>.

SALIMANS, T., GOODFELLOW, I., ZAREMBA, W., et al., 2016. "Improved Techniques for Training GANs". .

SCHMIDHUBER, J., 2014. "Deep Learning in Neural Networks: An Overview". Disponível em: <http://arxiv.org/abs/1404.7828>. cite arxiv:1404.7828Comment: 88 pages, 888 references.

SEITZER, M., 2020. "pytorch-fid: FID Score for PyTorch". https://github.com/mseitzer/pytorch-fid, August. Version 0.1.1.

SHMELKOV, K., SCHMID, C., ALAHARI, K., 2018, "How good is my GAN?" *CoRR*, v. abs/1807.09499. Disponível em: <http://arxiv.org/abs/1807.09499>.

SIMARD, P. Y., STEINKRAUS, D., PLATT, J. C., 2003, "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis." In: *ICDAR*, pp. 958–962. IEEE Computer Society. ISBN: 0-7695-1960-1. Disponível em: <http://dblp.uni-trier.de/db/conf/icdar/icdar2003.html#SimardSP03>.

SIMONYAN, K., ZISSERMAN, A., 2014, "Very Deep Convolutional Networks for Large-Scale Image Recognition", *CoRR*, v. abs/1409.1556. Disponível em: <http://arxiv.org/abs/1409.1556>.

SOCHER, R., MANNING, C. D., 2013, "Deep Learning for NLP (without Magic)." In: Vanderwende, L., III, H. D., Kirchhoff, K. (Eds.), *HLT-NAACL*, pp. 1–3. The Association for Computational Linguistics. Disponível em: <http://dblp.uni-trier.de/db/conf/naacl/naacl2013.html#SocherM13>.

SONG, Y., MENG, C., ERMON, S., 2019, "MintNet: Building Invertible Neural Networks with Masked Convolutions". In: Wallach, H., Larochelle, H., Beygelzimer, A., et al. (Eds.), *Advances in Neural Information Processing Systems*, v. 32, pp. 11004–11014. Curran Associates, Inc. Disponível em: <https://proceedings.neurips.cc/paper/2019/file/70a32110fff0f26d301e58ebbca9cb9f-Paper.pdf>.

SZEGEDY, C., TOSHEV, A., ERHAN, D., 2013, "Deep Neural Networks for Object Detection". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, p. 2553–2561, Red Hook, NY, USA. Curran Associates Inc.

SZEGEDY, C., LIU, W., JIA, Y., et al., 2014, "Going Deeper with Convolutions", *CoRR*, v. abs/1409.4842. Disponível em: <http://arxiv.org/abs/1409.4842>.

SZEGEDY, C., LIU, W., JIA, Y., et al., 2015, "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9.

THEIS, L., VAN DEN OORD, A., BETHGE, M., 2016, "A note on the evaluation of generative models". In: *International Conference on Learning Representations*, Apr. Disponível em: <http://arxiv.org/abs/1511.01844>.

TROTTIER, L., GIGUÈRE, P., CHAIB-DRAA, B., 2016, "Parametric Exponential Linear Unit for Deep Convolutional Neural Networks", *CoRR*, v. abs/1605.09332. Disponível em: <http://arxiv.org/abs/1605.09332>.

TURNER, A. J., MILLER, J. F., 2014, "NeuroEvolution: Evolving Heterogeneous Artificial Neural Networks", *Evolutionary Intelligence*, v. 7, n. 3 (nov.), pp. 135–154. ISSN: 1864-5909. doi: doi:10.1007/s12065-014-0115-5. Disponível em: <http://dx.doi.org/10.1007/s12065-014-0115-5>. Special Issue: Evolution in UK 20.

WERBOS, P. J., 1974, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Tese de Doutorado, Harvard University.

WERBOS, P. J., 1981, "Applications of Advances in Nonlinear Sensitivity Analysis". In: *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, pp. 762–770.

XAVIER, A. E., 1982, *Penalização Hiperbólica: Um Novo Método para Resolução de Problemas de Otimização*. Tese de Mestrado, Universidade Federal do Rio de Janeiro.

XAVIER, A. E., 2005, "Uma função ativação para Redes Neurais artificiais mais flexível e poderosa e mais rápida", *Learning & Nonlinear Models*, v. 3, n. 1, pp. 1–7.

XAVIER, V. L., XAVIER, A. E., 2016, "Função Ativação Hiperbólica para Redes Neurais Artificiais", pp. 1–14.

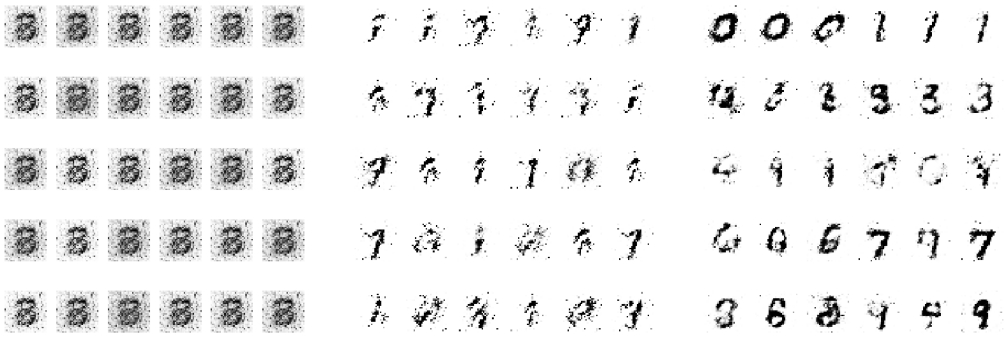
YAO, X., 1999, "Evolving artificial neural networks", *Proceedings of the IEEE*, v. 87, n. 9, pp. 1423–1447.

ZEILER, M. D., FERGUS, R., 2013, "Visualizing and Understanding Convolutional Networks." *CoRR*, v. abs/1311.2901. Disponível em: <http://dblp.uni-trier.de/db/journals/corr/corr1311.html#ZeilerF13>.

ZEILER, M., KRISHNAN, D., TAYLOR, G., et al., 2010, "Deconvolutional networks". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2010*, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 2528–2535. ISBN: 9781424469840. doi: 10.1109/CVPR.2010.5539957. 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2010 ; Conference date: 13-06-2010 Through 18-06-2010.

Apêndice A

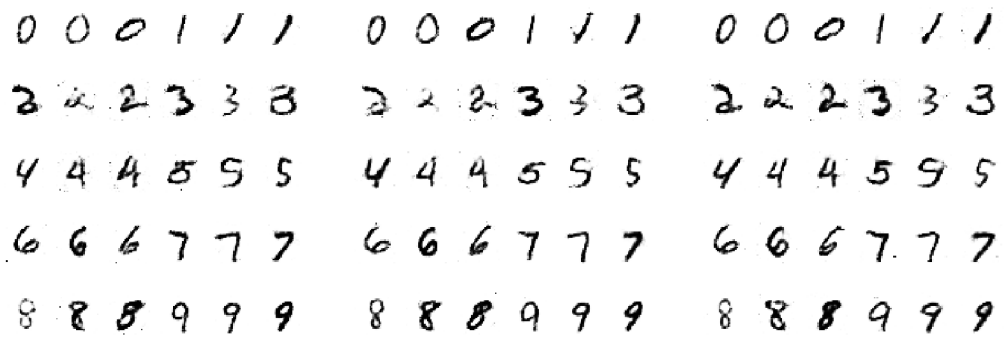
Amostras geradas - Parte 1 (MNIST-CondGAN)



(a) Época 1 (b) Época 5 (c) Época 10

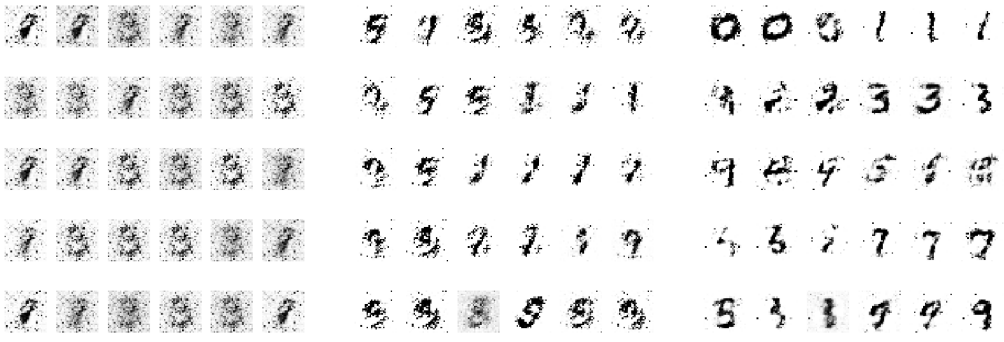
(d) Época 15 (e) Época 20 (f) Época 25

(g) Época 30 (h) Época 40 (i) Época 50

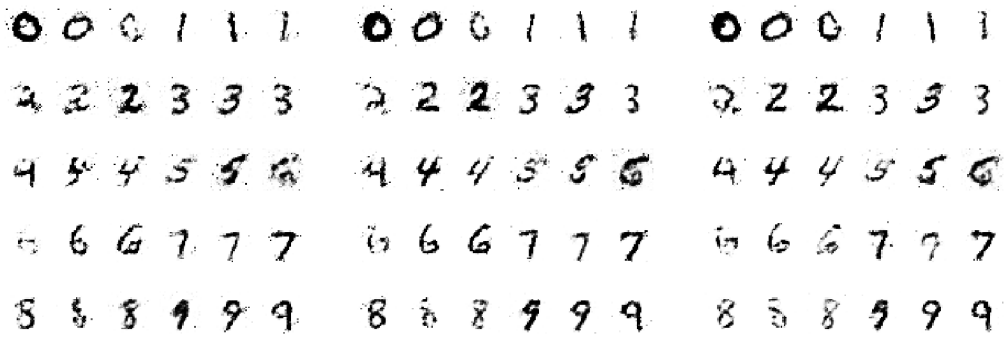


(j) Época 65 (k) Época 80 (l) Época 100

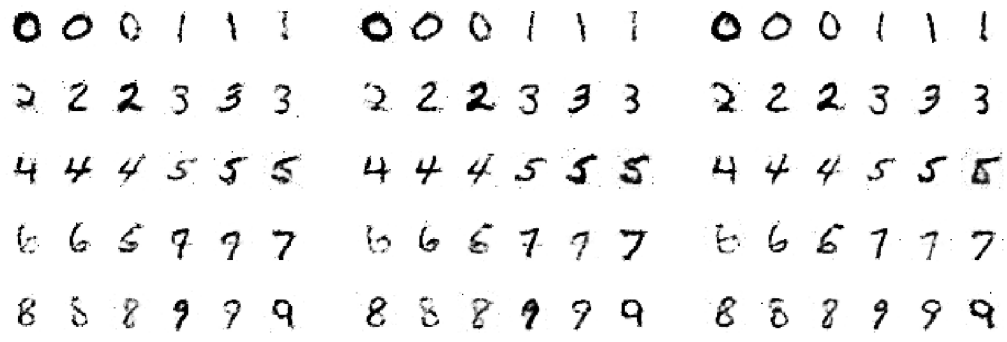
Figura A.1: Amostras geradas pelo modelo *baseline* no experimento MNIST-CondGAN (Seção 4.2).



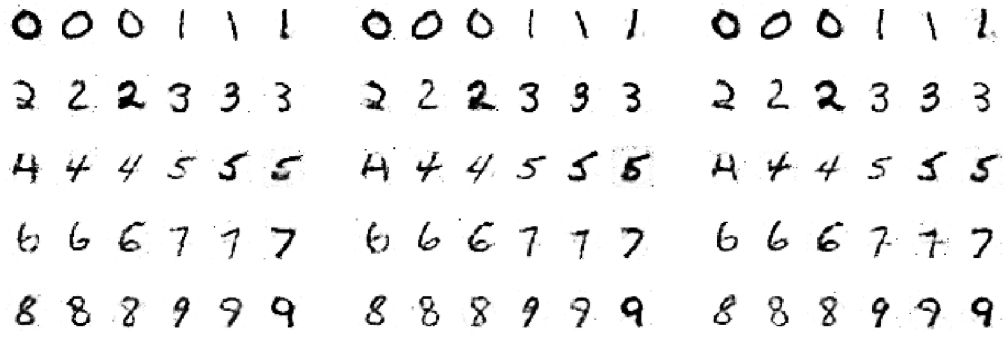
(a) Época 1 (b) Época 5 (c) Época 10



(d) Época 15 (e) Época 20 (f) Época 25

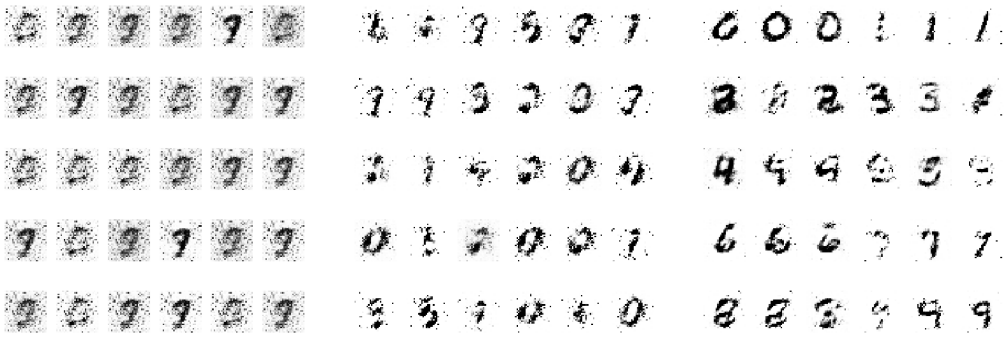


(g) Época 30 (h) Época 40 (i) Época 50

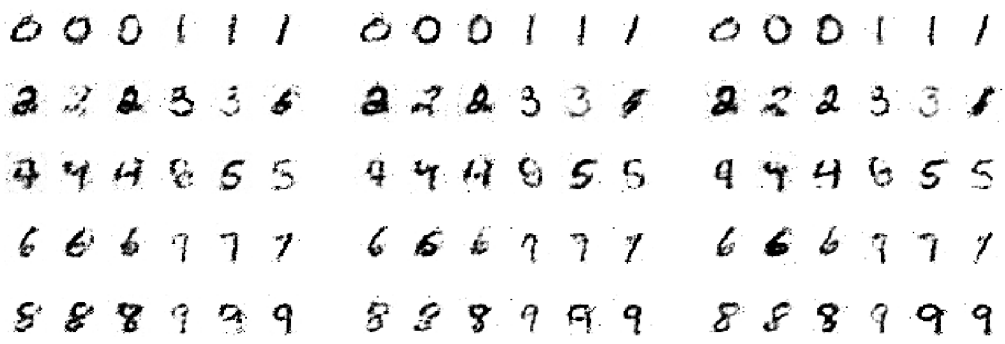


(j) Época 65 (k) Época 80 (l) Época 100

Figura A.2: Amostras geradas pelo modelo AAF-GAN Dis_BHSA no experimento MNIST-CondGAN (Seção 4.2).



(a) Época 1 (b) Época 5 (c) Época 10



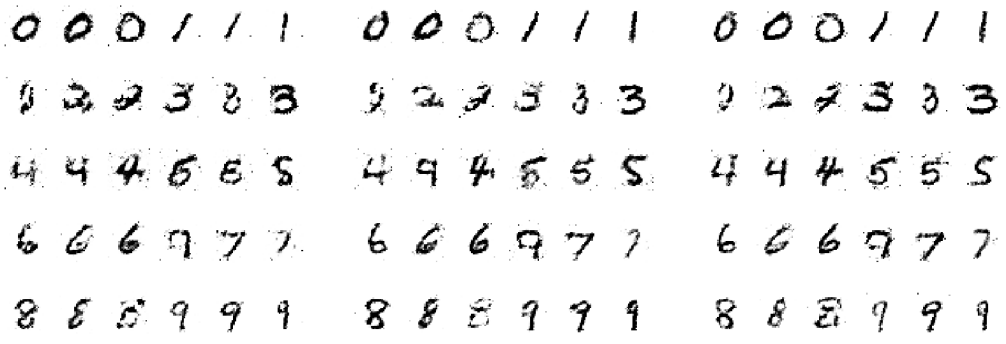
(d) Época 15 (e) Época 20 (f) Época 25

(g) Época 30 (h) Época 40 (i) Época 50

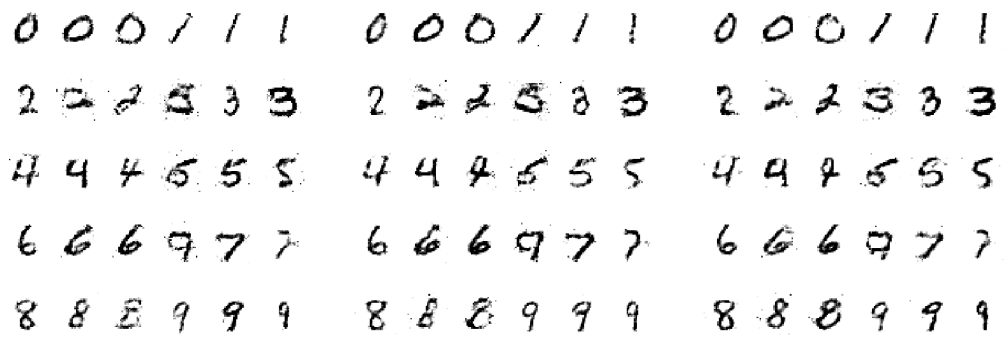
(j) Época 65 (k) Época 80 (l) Época 100

Figura A.3: Amostras geradas pelo modelo AAF-GAN Gen_BHSA no experimento MNIST-CondGAN (Seção 4.2).

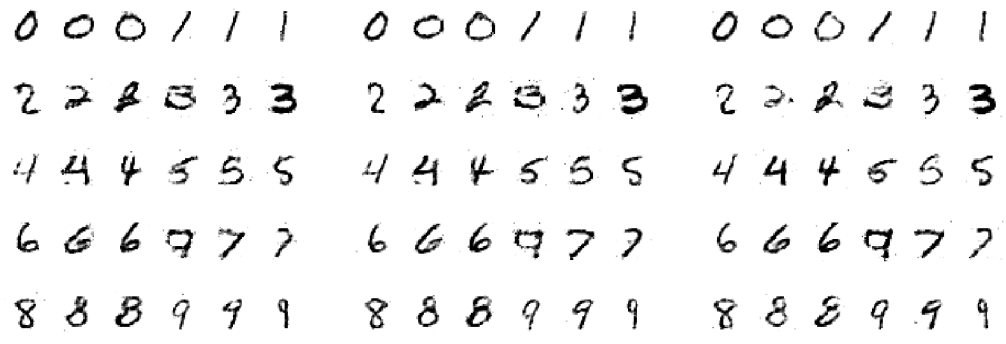
(a) Época 1 (b) Época 5 (c) Época 10



(d) Época 15 (e) Época 20 (f) Época 25

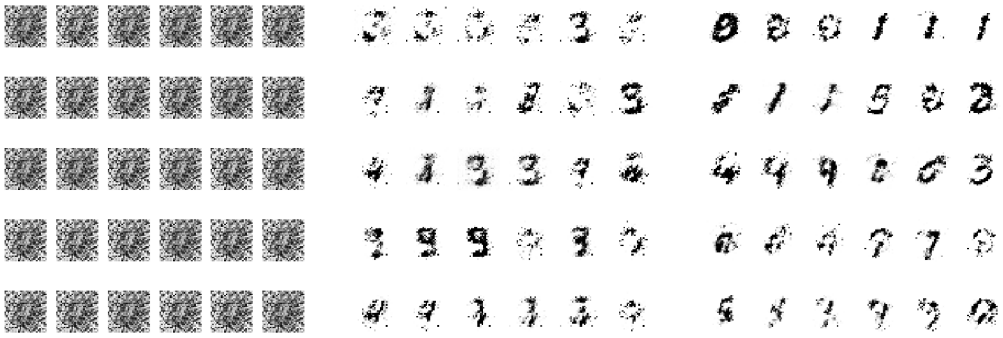


(g) Época 30 (h) Época 40 (i) Época 50

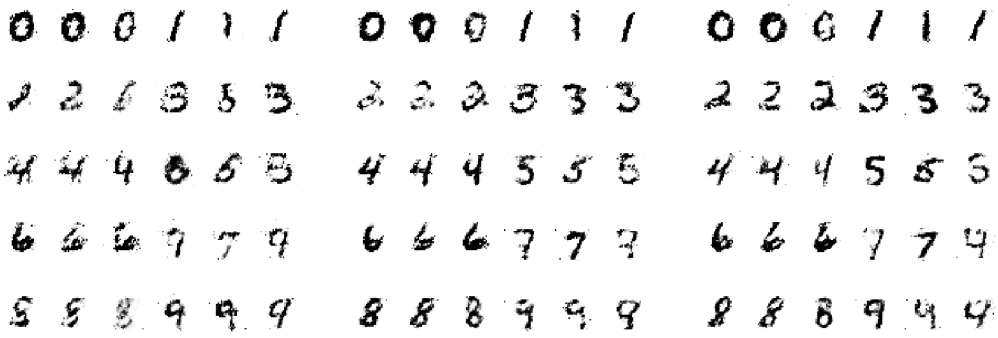


(j) Época 65 (k) Época 80 (l) Época 100

Figura A.4: Amostras geradas pelo modelo AAF-GAN Dis_BHSA_Gen_BHSA no experimento MNIST-CondGAN (Seção 4.2).

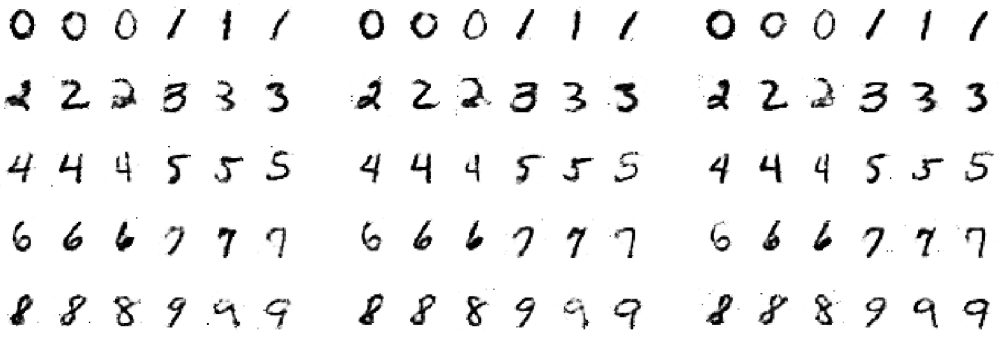


(a) Época 1 (b) Época 5 (c) Época 10



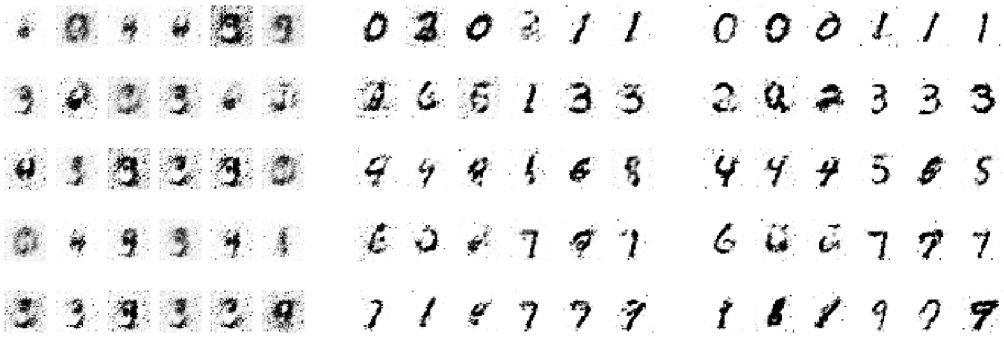
(d) Época 15 (e) Época 20 (f) Época 25

(g) Época 30 (h) Época 40 (i) Época 50



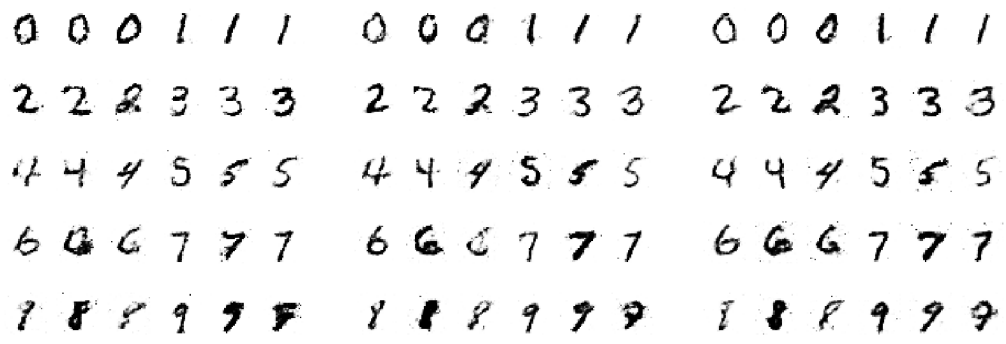
(j) Época 65 (k) Época 80 (l) Época 100

Figura A.5: Amostras geradas pelo modelo AAF-GAN DMiDA no experimento MNIST-CondGAN (Seção 4.2).



(a) Época 1 (b) Época 5 (c) Época 10

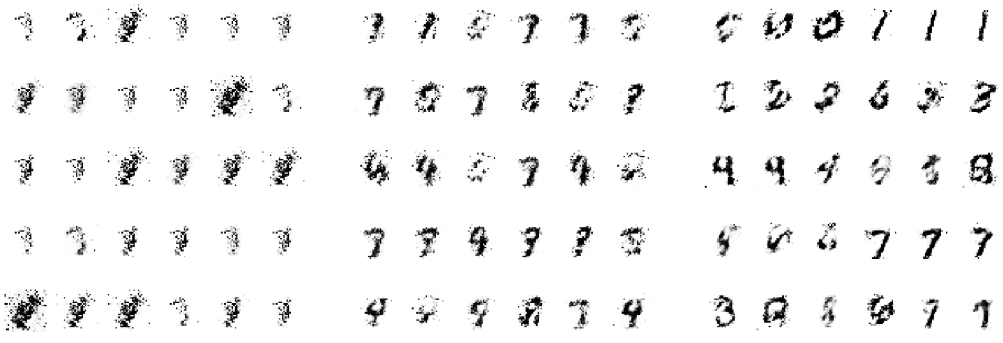
(d) Época 15 (e) Época 20 (f) Época 25



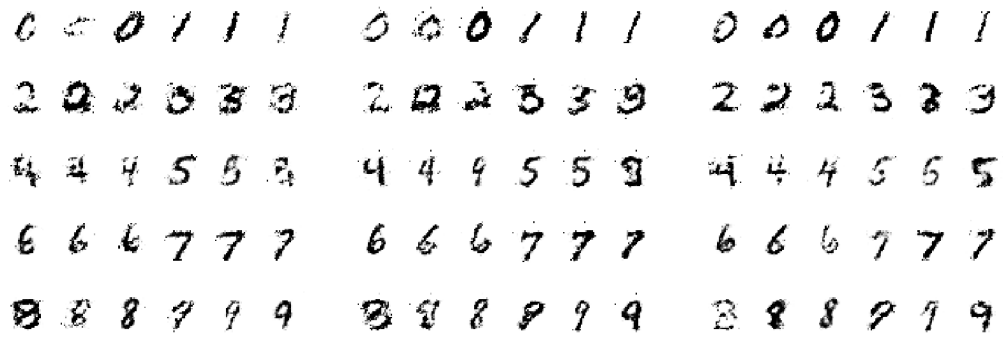
(g) Época 30 (h) Época 40 (i) Época 50

(j) Época 65 (k) Época 80 (l) Época 100

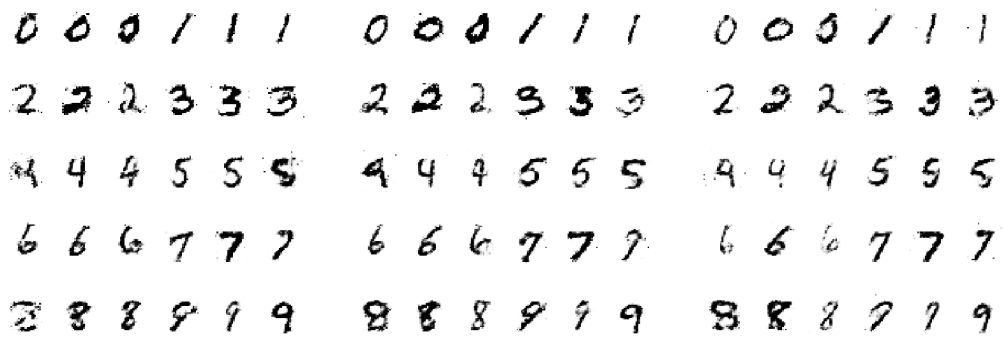
Figura A.6: Amostras geradas pelo modelo AAF-GAN GMiDA no experimento MNIST-CondGAN (Seção 4.2).



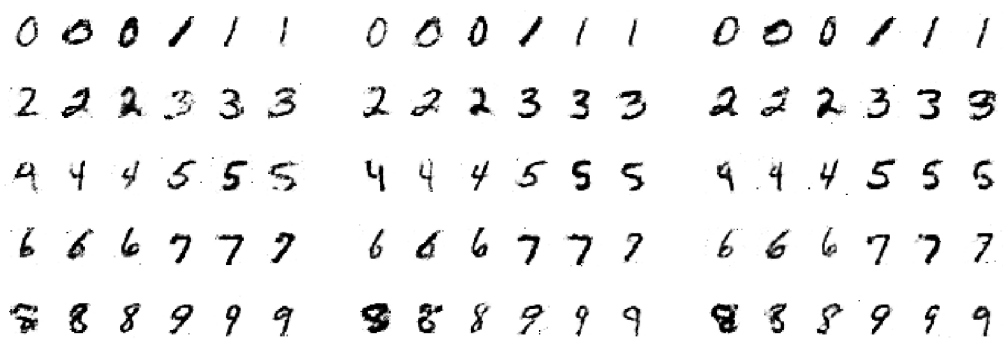
(a) Época 1 (b) Época 5 (c) Época 10



(d) Época 15 (e) Época 20 (f) Época 25

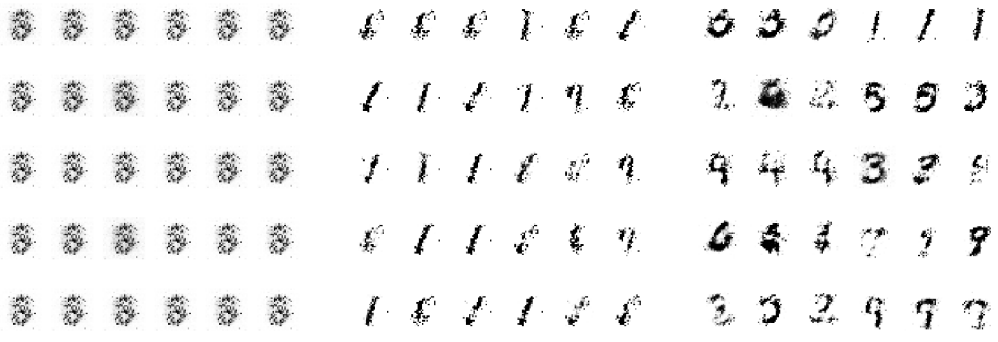


(g) Época 30 (h) Época 40 (i) Época 50

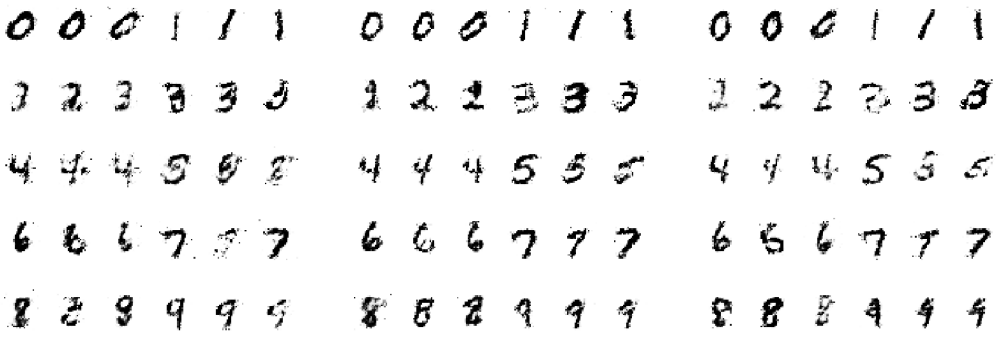


(j) Época 65 (k) Época 80 (l) Época 100

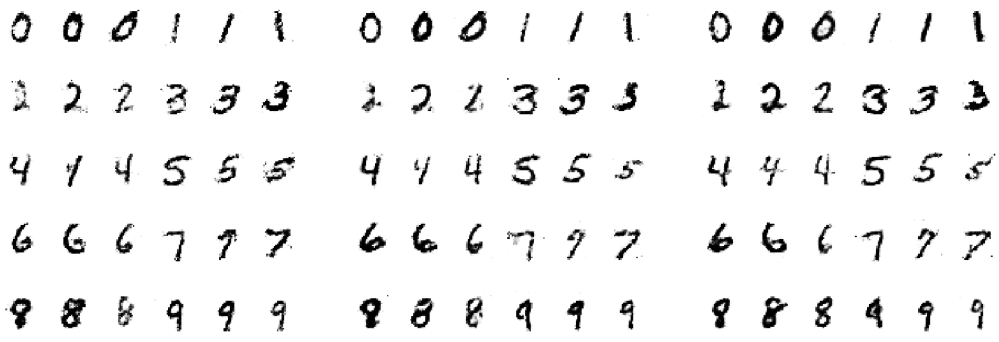
Figura A.7: Amostras geradas pelo modelo AAF-GAN DMiDA_GMiDA no experimento MNIST-CondGAN (Seção 4.2).



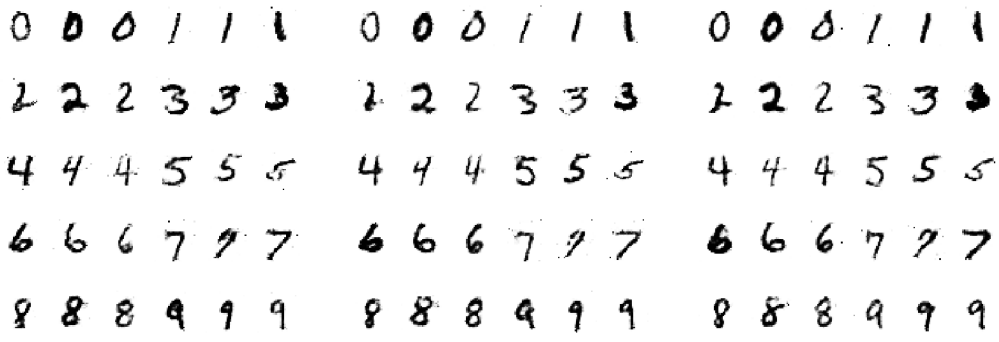
(a) Época 1 (b) Época 5 (c) Época 10



(d) Época 15 (e) Época 20 (f) Época 25

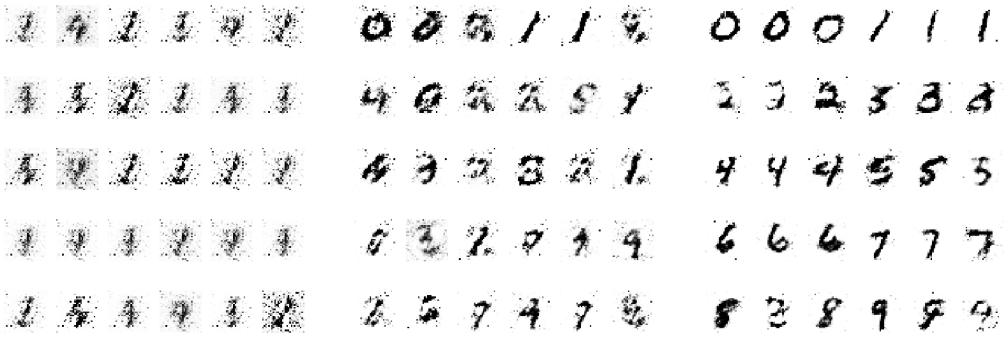


(g) Época 30 (h) Época 40 (i) Época 50

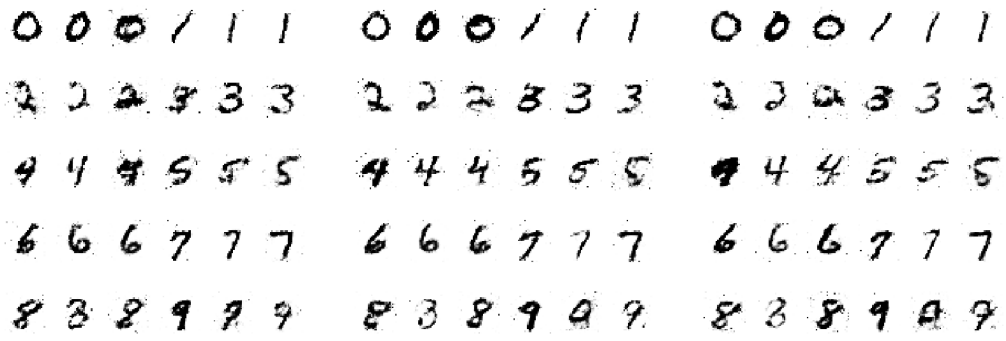


(j) Época 65 (k) Época 80 (l) Época 100

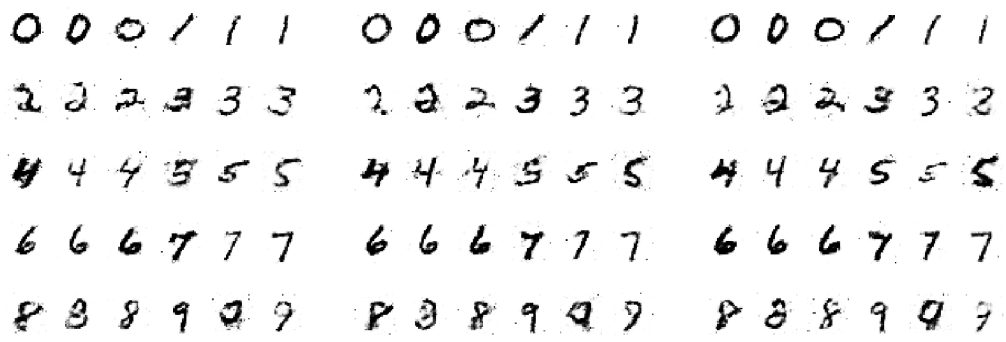
Figura A.8: Amostras geradas pelo modelo AAF-GAN DMish no experimento MNIST-CondGAN (Seção 4.2).



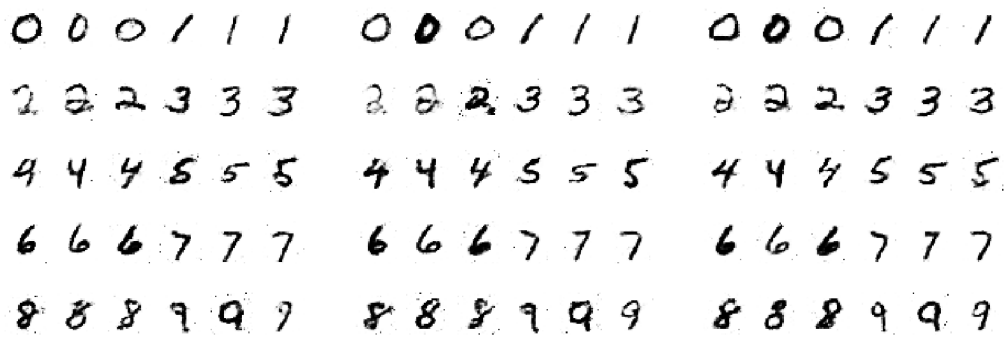
(a) Época 1 (b) Época 5 (c) Época 10



(d) Época 15 (e) Época 20 (f) Época 25

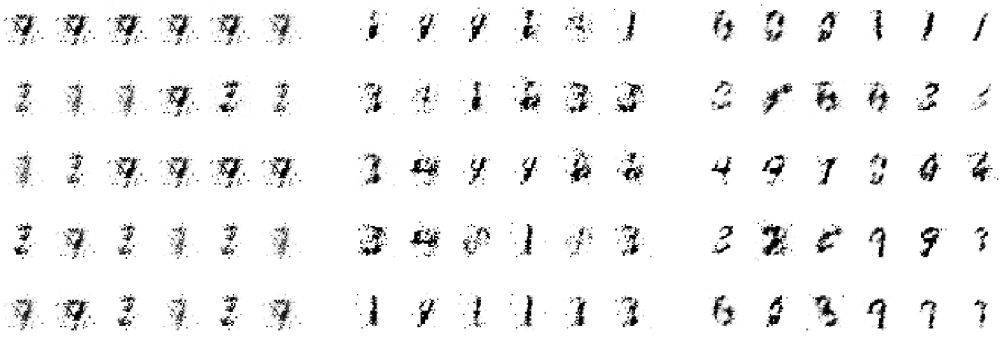


(g) Época 30 (h) Época 40 (i) Época 50

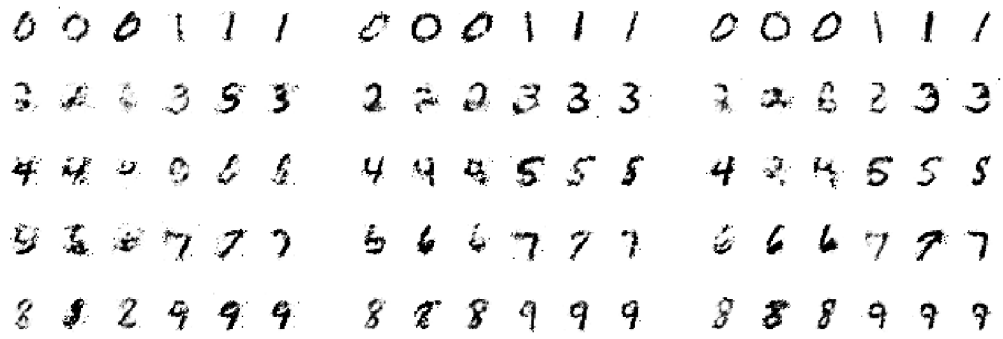


(j) Época 65 (k) Época 80 (l) Época 100

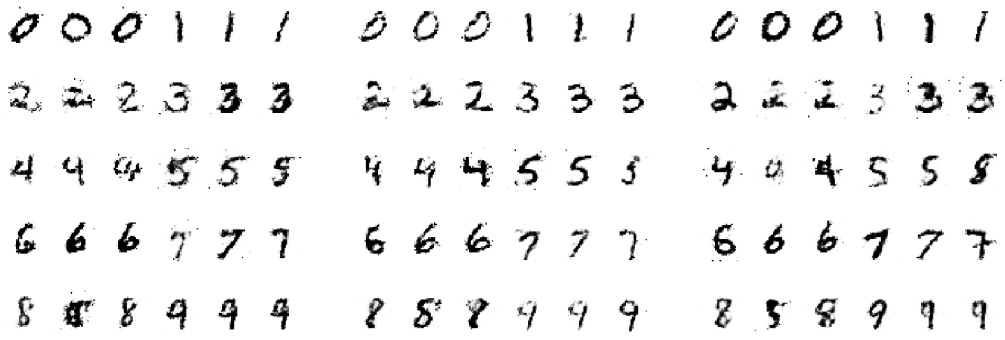
Figura A.9: Amostras geradas pelo modelo AAF-GAN GMish no experimento MNIST-CondGAN (Seção 4.2).



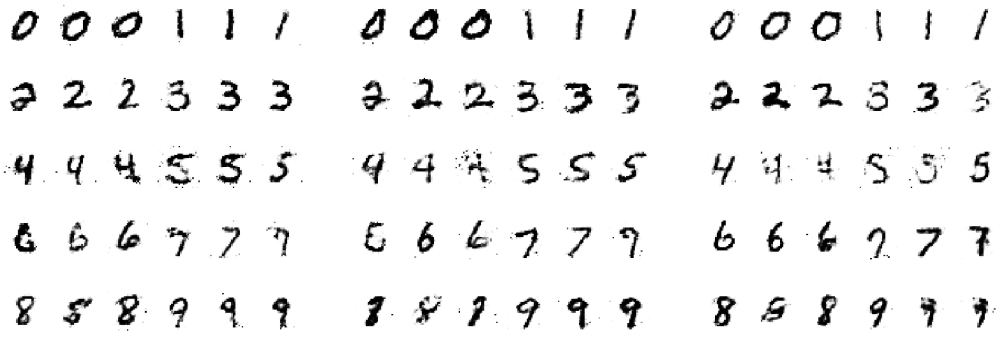
(a) Época 1 (b) Época 5 (c) Época 10



(d) Época 15 (e) Época 20 (f) Época 25



(g) Época 30 (h) Época 40 (i) Época 50

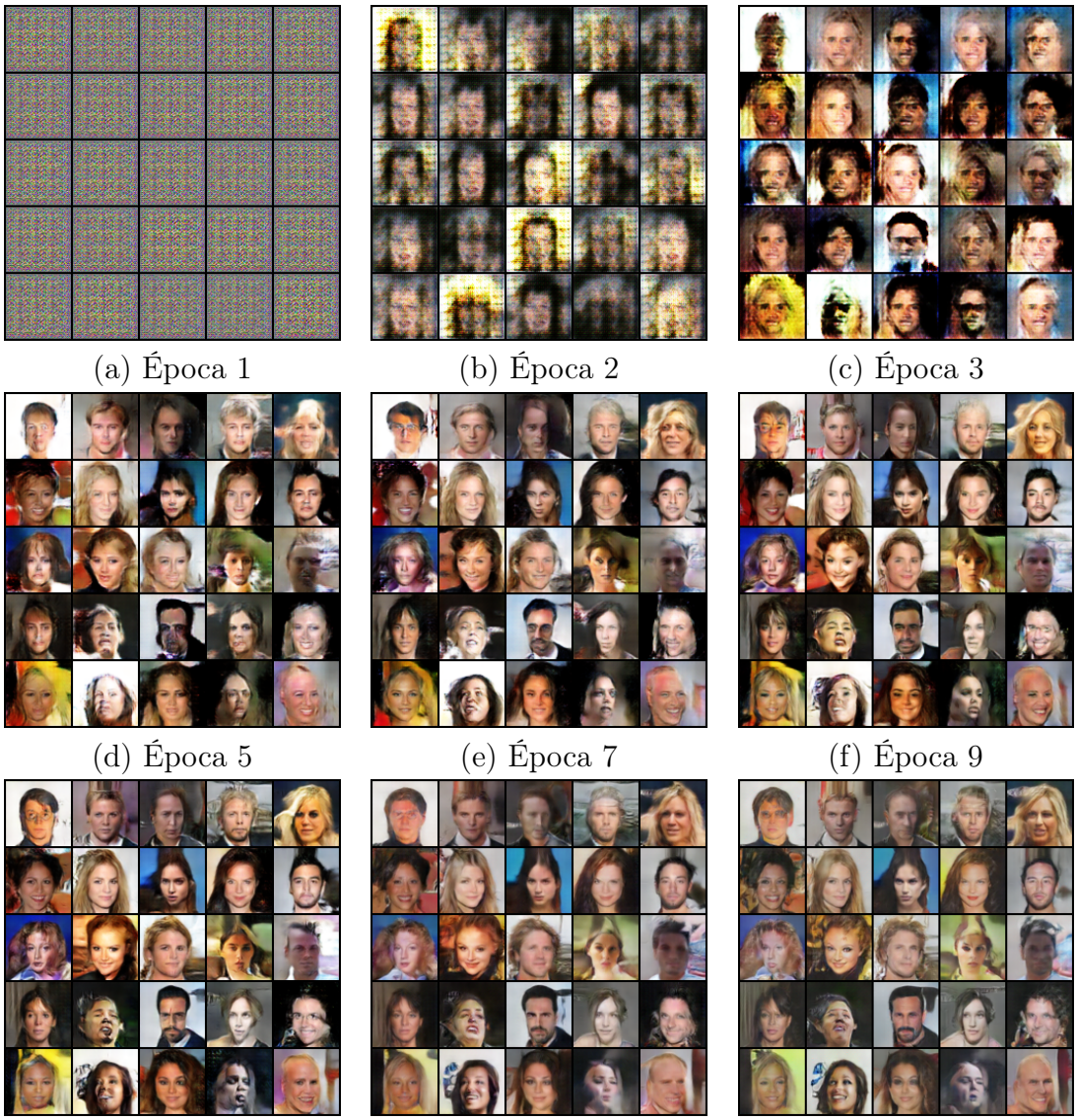


(j) Época 65 (k) Época 80 (l) Época 100

Figura A.10: Amostras geradas pelo modelo AAF-GAN DMish_GMish no experimento MNIST-CondGAN (Seção 4.2).

Apêndice B

Amostras geradas - Parte 2 (CelebA-DCGAN)



(a) Época 1 (b) Época 2 (c) Época 3

(d) Época 5 (e) Época 7 (f) Época 9

(g) Época 12 (h) Época 15 (i) Época 25

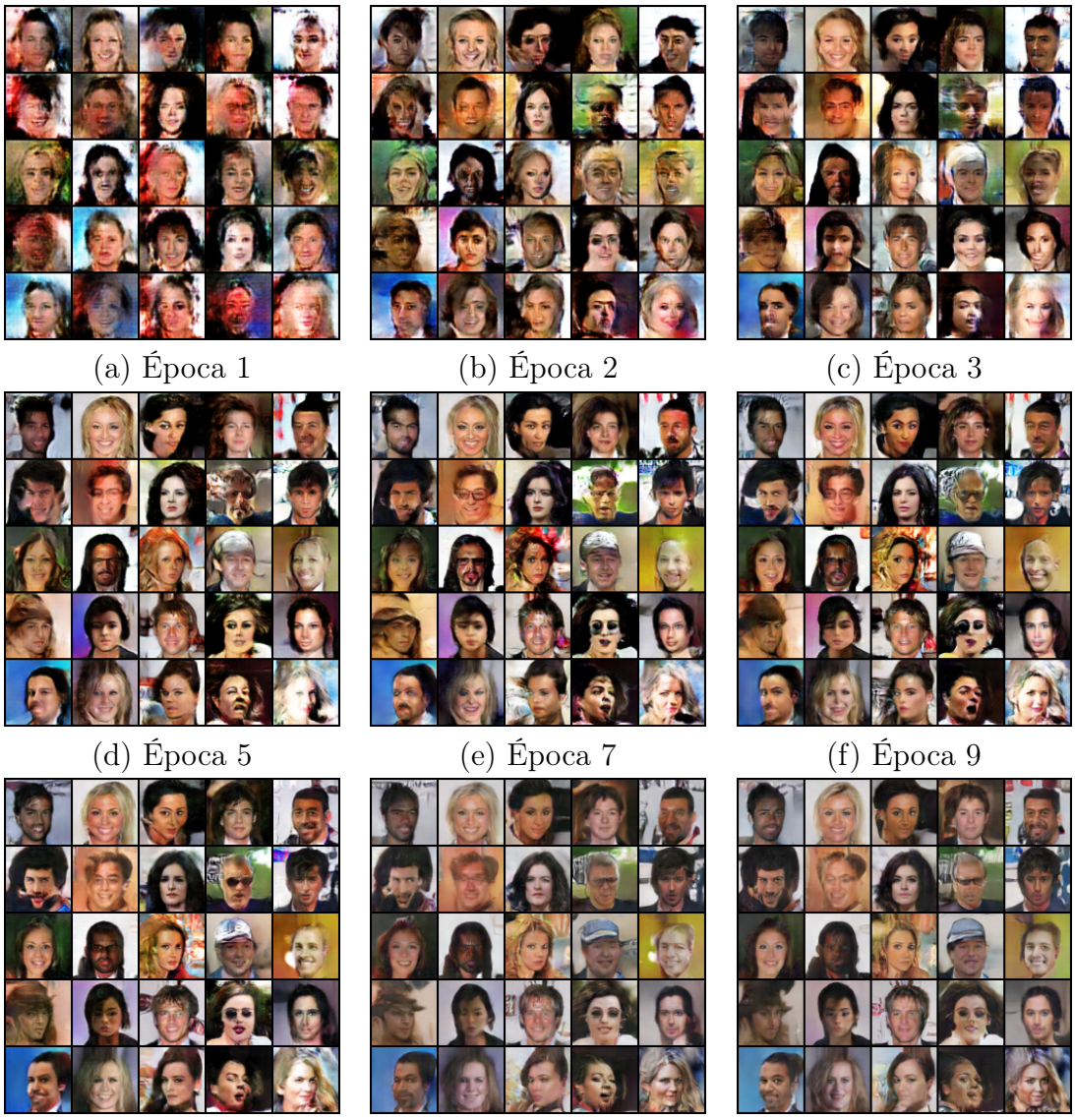
Figura B.1: Amostras geradas pelo modelo *baseline* no experimento CelebA-DCGAN (Seção 4.3).

(a) Época 1 (b) Época 2 (c) Época 3

(d) Época 5 (e) Época 7 (f) Época 9

(g) Época 12 (h) Época 15 (i) Época 25

Figura B.2: Amostras geradas pelo modelo AAF-GAN Dis_BHSA_Gen_BHSA no experimento CelebA-DCGAN (Seção 4.3).



(a) Época 1 (b) Época 2 (c) Época 3

(d) Época 5 (e) Época 7 (f) Época 9

(g) Época 12 (h) Época 15 (i) Época 25

Figura B.3: Amostras geradas pelo modelo AAF-GAN GMiDA no experimento CelebA-DCGAN (Seção 4.3).

(a) Época 1 (b) Época 2 (c) Época 3

(d) Época 5 (e) Época 7 (f) Época 9

(g) Época 12 (h) Época 15 (i) Época 25

Figura B.4: Amostras geradas pelo modelo AAF-GAN GMish no experimento CelebA-DCGAN (Seção 4.3).

Apêndice C

Tempo de execução médio por época dos experimentos

Tabela C.1: Tempo médio gasto (em segundos) para a execução de uma época para cada modelo na máquina com GPU NVIDIA GTX 1080ti

Modelo	**MNIST-CondGAN**	**CelebA-DCGAN**
baseline	12 s	471 s
Dis_BHANA	17 s	Não aplicado
DMish_GMish	13 s	Não aplicado
Dis_BHSA	13 s	Não aplicado
DSHReLU_GSHReLU	13 s	Não aplicado
Dis_BHANA_Gen_BHANA	18 s	Não aplicado
DSHReLU	13 s	Não aplicado
GMish	12 s	486 s
GSHReLU	12 s	Não aplicado
Dis_BHATA_Gen_BHATA	15 s	Não aplicado
Dis_BHSA_Gen_BHSA	13 s	477 s
DMish	13 s	Não aplicado
Gen_BHANA	14 s	Não aplicado
DMiDA_GMiDA	16 s	Não aplicado
Gen_BHATA	13 s	Não aplicado
DMiDA	14 s	Não aplicado
GMiDA	13 s	531 s
Dis_BHATA	14 s	Não aplicado
Gen_BHSA	12 s	Não aplicado

Tabela C.2: Tempo médio gasto (em segundos) para a execução de uma época para cada modelo em uma instância p3.2xlarge na AWS

Modelo	**MNIST-CondGAN**	**CelebA-DCGAN**
baseline	12 s	300 s
Dis_BHANA	20 s	Não aplicado
DMish_GMish	12 s	Não aplicado
Dis_BHSA	13 s	Não aplicado
DSHReLU_GSHReLU	14 s	Não aplicado
Dis_BHANA_Gen_BHANA	26 s	Não aplicado
DSHReLU	13 s	Não aplicado
GMish	12 s	308 s
GSHReLU	12 s	Não aplicado
Dis_BHATA_Gen_BHATA	16 s	Não aplicado
Dis_BHSA_Gen_BHSA	14 s	303 s
DMish	12 s	Não aplicado
Gen_BHANA	16 s	Não aplicado
DMiDA_GMiDA	22 s	Não aplicado
Gen_BHATA	13 s	Não aplicado
DMiDA	18 s	Não aplicado
GMiDA	14 s	328 s
Dis_BHATA	14 s	Não aplicado
Gen_BHSA	12 s	Não aplicado