Perceptron Multicamada

1st Philippe José Alves
Departamento de Engenharia Elétrica
Universidade de Brasília
Brasília, Brasil
philippejosealves@gmail.com

2nd Marcelo Guedes França
Departamento de Engenharia Elétrica
Universidade de Brasília
Brasília, Brasil
marcelo_guedes53@hotmail.com

***Resumo*—Neste relatório, avaliou-se a performance do perceptron multi-camadas para 4 arquiteturas distintas na identificação dos dez dígitos manuscritos que compõem a base decimal(0 a 9). Para treinar, validar e testar cada uma das redes foi utilizada uma base de dados do MNIST que contém 70 mil imagens. Em média, a acurácia nessas 4 arquitetura ficaram em torno de 97% mas pequenas diferenças foram observadas de arquitetura para arquitetura.**

I. Introdução

A. Apresentação do problema

No geral, os dígitos manuscritos são facilmente reconhecidos pelos seres humanos devido ao alto grau de performance do cérebro humano no reconhecimento de padrões presentes em imagens. Contudo, essa tarefa não tem uma solução simples e direta quando uma máquina é responsável por fazê-la. Há inúmeras aplicações úteis quando a tarefa de reconhecimento de dígitos ou, de forma geral, de caracteres manuscritos é feita de forma automática sem a necessidade do ser humano. Tarefas de tradução como de libras para alguma linguagem textual, reconhecimento de textos a partir das suas imagens manuscritas se mostram importantes para a nossa sociedade.

O projeto tem como objetivo propor uma solução para o problema de reconhecimento de dígitos manuscritos a partir de uma rede neural artificial organizada em camadas denominada de perceptron multicamada. Trata-se portanto de um algoritmo de aprendizagem de máquinas para um resolver um problema de classificação em multi-classes.

B. Base de dados

Para treinar a rede neural utilizou-se a base de dados de imagens manuscritas de dígitos de 0 a 9 pertencentes ao ao *Modified National Institute of Standards and Technology*(MNIST) database. Nessa base dados as imagens estão em níveis de cinza, sendo que cada imagem é um conjunto de 28x28 pixels de 0 (branco) a 255 (preto). A base de dados está disponível para download em [2]. Essa base de dados é composta por duas partições. A primeira parte possui 60000 imagens que estão disponíveis para treinamento. O tamanho das imagens é de 28x28 e todas estão em escala de cinza, de 0 a 255 (branco a preto). Semelhantemente, o conjunto de 10000 imagens para teste, que é a segunda parte, está em escala de cinza. Para a nossa aplicação dividimos o conjunto de teste em dois subconjuntos: o de validação e o novo conjunto teste cada uma com 5000 imagens.

A função de ativação é aquela que orienta-se na saída no neurônio depois que ele faz o produto escalar entre os pesos.

C. Fundamentação teórica

O perceptron multicamada é uma rede neural artificial em que cada camada se conecta sempre à próxima camada (*feed-forward*). O MLP (Multi Layer Perceptron) apresenta uma ou mais camadas ocultas em sua arquitetura. Essa camadas são as intermediárias, isto é, entre as camadas de entrada e de saída. A figura em 1 ilustra uma rede MLP.

O MLP pode ser considerada com uma generalização do percepetorn simples para múltiplas camadas. O perceptron simples pode ser resumido do ponto de vista matemático pela equação em 1.

$$y = \varphi(\sum_{i=1}^{n} w_i x_i + b) = \varphi(\underline{w}^T x + b) \tag{1}$$

Nessa expressão w é a matriz de pesos, x as entradas, b o *bias* φ é a função de ativação e por fim y é a saída do neurônio. No MLP a saída do perceptron simples é conectado a neurônios de outras camadas e assim sucessivamente até a camada de saída.

Para que o MLP superasse a performance do perceptron simples foi essencial a estrutura em múltiplas camadas, o uso de funções de ativação não lineares e diferenciáveis e adequação do algoritmo de backpropagation para atualizar os pesos de todas as camadas da rede. Dessa forma, o MLP conseguiu solucionar problemas cujos dados eram não-linearmente separáveis.

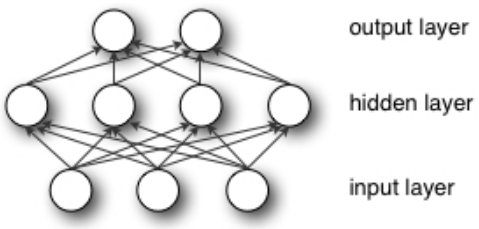

Figura 1. Perceptron Multicamada [1]

No treinamento em multicamadas do MLP, o valor do vetor pesos será recalculado a partir da última camada até a camada

de entrada da rede. Com o erro da camada de saída propaga-se ele para as demais camadas usando a regra da cadeia. Esse método de atualizar os pesos é chamado de *backpropagation*. Nota-se que esse algoritmo só é possível em treinamentos supervisionados porque de outro modo não é possível calcular o erro na saída sem conhecermos o valor verdadeiro da saída.

Com isso, o MLP segue uma sequência de processos: Inicialização dos pesos da rede com pequenos valores aleatórios; Cálculo da saída da rede; cálculo da função de erro diferenciável a partir dos dados de entrada; cálculo dos gradientes de cada peso da rede (a fim de que o sentido contrário do gradiente forneça um caminho para a propagação dos pesos); e aplicação da iteração para atualizar o peso de cada camada (calculando repetidamente o gradiente em cada iteração).

Dessa forma, a atualização de pesos segue a seguinte relação:

$$w \leftarrow w - \eta \frac{\partial E}{\partial w} y \tag{2}$$

Nessa equação η é um hiperparâmetro da rede a ter seu valor escolhido. Ele é chamado de taxa de aprendizado e indica o tamanho do passo da atualização do pesos no *backpropagation*. W é vetor de pesos de uma das camadas, $\frac{\partial E}{\partial w}$ é erro da saída de uma determinada camada devido aos pesos dessa camada e y representa o vetor de saída dos neurônios dessa camada.

As funções de ativação usadas nesse projeto basicamente decidem se a informação que o neurônio recebe é essencial para a informação fornecida ou deve ser desconsiderada. Usamos as funções de ativação logística (ou sigmóide), ReLU e Softmax. Cada uma dessas funções de ativação foram usadas em situações específicas. Para as camadas intermediárias foi usada a função ReLU, e para a camada de saída a função logística ou a softmax, já que é mais propício para classificação multiclasse.

A função logística (ou sigmóide) é não-linear. Quanto a isso sabe-se que quando há vários neurônios com função sigmóide como função de ativação a saída será igualmente não-linear. A função varia de 0 a 1 tendo um formato de S, e descrita da seguinte forma:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3}$$

A função ReLU também é não-linear, o que quer dizer que existe a possibilidade de facilmente copiar os erros para trás e obter diversas camadas de neurônios ativados por esta função. Entretanto, os gradientes que se deslocam em direção a zero podem ser empecilho para a ReLU. ReLU é definida por:

$$f(x) = max(0, x) \tag{4}$$

A Softmax é um tipo de sigmóide usada para tratar problemas de classificação. A função softmax transforma as saídas para cada classe para valores entre 0 e 1 e também divide pela soma das saídas. Isso essencialmente dá a probabilidade de a entrada estar em uma determinada classe. A função Softmax pode ser descrita da seguinte forma:

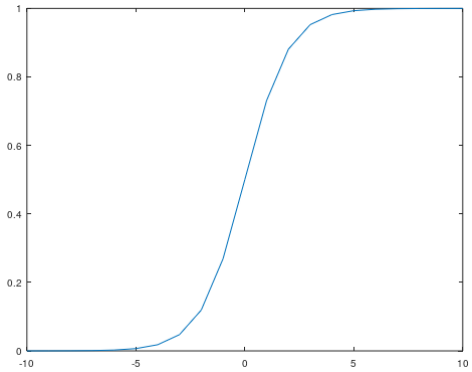

Figura 2. Função de Ativação Logística

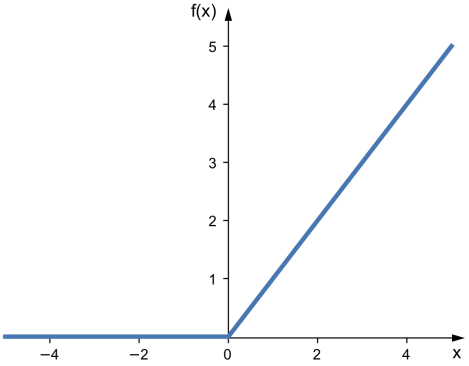

Figura 3. Função de Ativação ReLU

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, j = 1...K \tag{5}$$

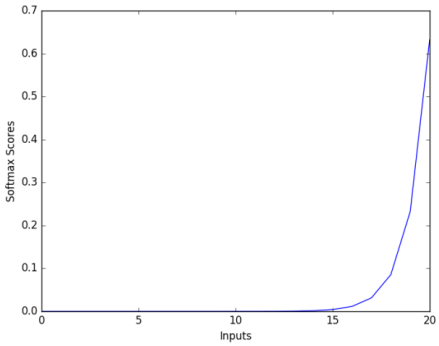

Figura 4. Função de Ativação Softmax

Foi usado a entropia cruzada em alternativa e o erro quadrático como funções de custo. Na entropia cruzada as saídas da rede são tratadas como hipótese independente, e as ativações como probabilidade da hipótese ser verdadeira. A entropia cruzada H dado em 6 quantifica a diferença entre duas distribuições de probabilidade.

$$H(p,q) = -\sum_{x} p(x) \log q(x) \tag{6}$$

O erro qudrático 7 é dado simplesmente pela diferença entre a saída predita pelo vetor y e a saída deseja, vetor d, elevado ao quadrado e somado para cada par (y_i,d_i) obtido.

$$E(d,y) = \sum_{i} (y_i - d_i)^2 \tag{7}$$

Como utilizaremos uma quantidade de camadas consideráveis (acima de 4) caracterizando uma rede profunda, é necessário evitar que haja overfitting, pois do contrário a rede irá se especializar nos dados de treinamento e não conseguirá uma boa generalização. Neste projeto foram usadas as regularizações dropout, L1 e L2 que serão usadas para não ocorrer overfitting, assim a rede será mais robusta.

As regularizações L1 e L2 penalizam os pesos que estão muito alto. Isso se torna necessário porque se o peso estiver muito grande, a função de custo terá um valor maior, e se a função de custo tem um valor maior será subtraído dos pesos no momento que for feito o backpropagation.

Pela abordagem de regularização L1 a função de custo é modificada, de modo que é adicionado dos valores absolutos dos pesos da seguinte forma:

$$C = C_0 + \frac{\lambda}{n} \sum_{w} |w| \tag{8}$$

A expressão que traz a resultante para a atualização de uma rede regularizada L1 será:

$$w \to w' = w - \frac{\eta\lambda}{n} sgn(w) - \eta \frac{0}{\partial w} \tag{9}$$

Os pesos reduzem em quantidade constante para zero quando a regularização é L1. Os pesos diminuem proporcionalmente a w quando a regularização é do tipo L2. Quando o peso tem uma grande magnitude, a Regularização L1 reduz o peso muito menos do que a Regularização L2.

Já o dropout atua desativando neurônios das camadas. Nesta regularização não há modificação da função de custo, no caso do dropout a própria rede é modificada. Nessa regularização, elimina-se casualmente neurônios ocultos presentes na rede, sem mexer na saída. É encaminhada para frente a entrada através da rede modificada, e depois é retropropagado o resultado. Esse processo é repetido primeiro restaurando os neurônios removidos, depois escolhendo um novo subconjunto aleatório de neurônios ocultos para excluir, assim a rede aprenderá um conjunto de pesos e viés.

II. Materiais e Métodos

A linguagem para resolver o problema de reconhecimento de dígitos e implementar o MLP foi o TensorFlow 1.13.1. O TensorFlow é uma biblioteca de software de código aberto lançada pela Google para computação numérica usando grafos computacionais. O aplicativo Jupyter Notebook foi utilizado como plataforma para criar e compartilhar o código em TensorFLow. Usamos o Keras para descarregar o dataset do MNIST. O Keras é uma biblioteca de rede neural open-source em Python que é executada sob o TensorFlow.

A. Pré-processamento

O pré processamento dos dados consistiu em normalizar os pixels por 255. Dessa forma, seus valores estão entre 0 a 1. Os dados foram separados em 3 categorias: treino, validação e teste. Os dados de treino é formado por 60 mil amostras ao passo que a validação e o teste possuem 5 mil imagens cada.

Para usar o MLP para o caso multi-classe codificou-se a saída para a codificação one-hot. Como há 10 classes, então cada classe será representada por um vetor de tamanho 10 valendo zero em 9 desses valores e 1 no índice que representa a classe ao qual pertence. Arbitrou-se o índice 0 como o representante do dígito 0 (classe 0), índice 1 representa o dígito 1 (classe 1) e assim por diante.

B. Implementação

A programação para implementar o Percepetrom Multi-Camadas dar liberdade para definir o número de neurônios por camada, a quantidade de camadas escondidas. É possível definir qual regularização usar dentre L1, L2 e dropout ou se abster de usar essa técnica. A regularização é indicada para redes profundas e visa ampliar a capacidade de generalização da MLP. Foram implementadas duas funções de custo: a entropia cruzada e o erro quadrático. A base para elaborar o código foi retirada de [4]. Adaptações e implementações de novas funções foram feitas no código original.

As funções de ativações são disponibilizadas pelo TensorFlow. Nas arquiteturas de redes MLP propostas foi usado a função de ativação ReLU até a penúltima camada. A última camada foi utilizada a softmax ou a logística

A atualização dos pesos é feita pelo método de descida do gradiente em que tentar-se minimizar o erro da função de custo em função dos pesos e bias da rede. A camada de saída sempre tem 10 neurônios que representam as 10 classes.

Dado esse conjunto de decisões a serem tomadas foram escolhidas algumas arquiteturas de redes neurais para resolver o problema e comparar o performance de cada uma. Essas arquiteturas compreendem 4 cenários ou abordagens. Em todos os cenários o modelo foi treinados utilizando mini-conjuntos de imagens (*batches*) de tamanho 32. No treinamento os parâmetros da rede neural são recalculados após cada batch passar pela rede e ter o seu erro calculado.

III. Resultados e Discussão

A. Primeiro cenário

A arquitetura em camadas para esse caso consistiu em:

1) Camada de entrada
    a) 512 neurônios
    b) Ativação Relu
2) Primeira camada escondida
    a) 256 neurônios
    b) Ativação Relu
    c) Regularização L2

3) Segunda camada escondida
   a) 128 neurônios
   b) Ativação Relu
4) Terceira camada escondida
   a) 64 neurônios
   b) Ativação Relu
5) Camada de saída
   a) 10 neurônios
   b) Ativação Softmax

Esse modelo foi treinado por 200 épocas. A atualização dos pesos é feita conforme o método da descida do gradiente. Foi fixado 0.01 como taxa de regularização. A função de custo utilizada foi a entropia cruzada cuja fórmula foi dada em 6. A taxa de aprendizagem foi fixada em 0.005. Como podemos notar essa arquitetura temos 5 camadas e então cabe aplicar uma forma de regularização. A regularização escolhida foi a L2 presente na primeira camada escondida.

Para treinar esse modelo fizemos a validação para verificar o número de épocas em que atingimos a melhor performance e não incorremos em sobre-ajuste do modelo. A figura em 5 mostra o gráfico da validação. Notamos que a partir da época de número 150 a taxa de acurácia permanece praticamente constante. appendixA acuŕacia geral (envolvendo a classificação das 10 classes) usando as amostras de teste foi de 97.4%. É interessante notar que a curva de acurácia da validação acompanha a curva de acurácia treinamento sem discrepâncias consideráveis. Isso indica que o nosso modelo foi treinado de forma robusta sem que houvesse ajuste excessivo (*overfiting*).

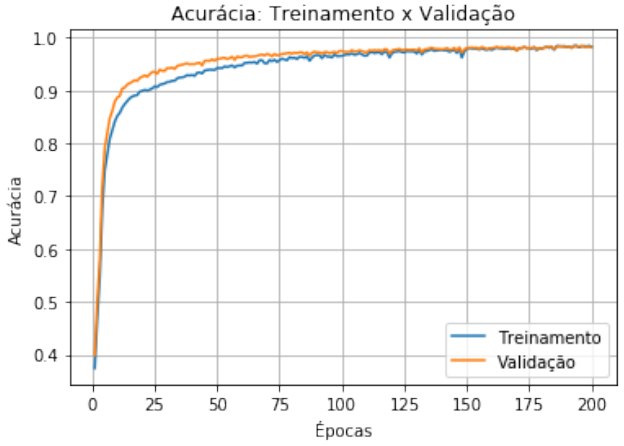

Figura 5. Cenário 01. Validação do treinamento

A matriz de confusão, para essa proposta de MLP, está disponível na figura 6. Os seus resultados foram normalizada pelo número de exemplos de cada classe. Esse tipo de normalização é interessante para identificar quais classes o nosso classificador tem dificuldade para acertar. Nessa matriz, nota-se que a classificação do dígito 9 foi a que teve maior taxa de erro, com 94% de acurácia. Vemos ainda que dos 6% de erro para essa classe 2% ocorrem quando o classificar prediz que o dígito é 4 enquanto a classe verdadeira é o 9. Esse erro era esperado dado que os dígitos 4 e 9 são semelhantes do ponto de vista da escrita. Além disso, em alguns casos esse dígitos podem ser classificados erroneamente até pelos seres humanos quando escritos com uma ”caligrafia ruim”.

Outro erro relevante do classificador foi entre as classes ou dígitos 7 e 1. De fato, 2% das classificações em que se esperava a classe 7 foi predito pelo classificador a classe 1. Mais uma vez, é um tipo de erro aceitável dados a semelhança de escrita entre os números 7 e 1 que ocorrem em algumas das imagens do conjunto de teste.

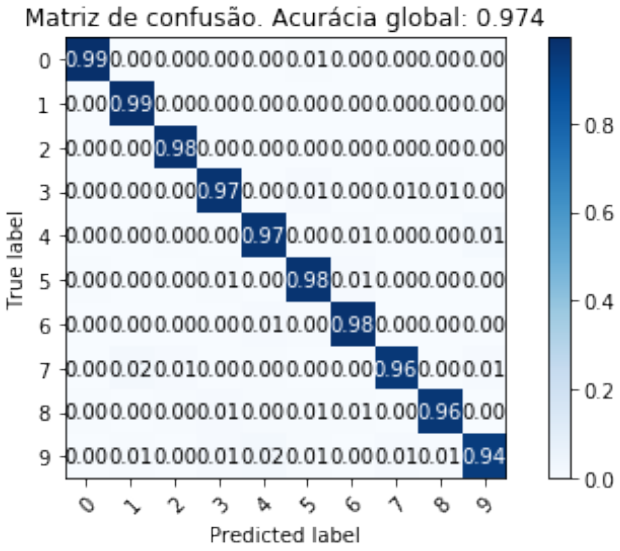

Figura 6. Cenário 01. Matriz de Confusão

B. *Segundo cenário*

Nesse segundo cenário aumentou-se a profundidade da rede e também o número de regularizações. A ideia é verificar a resposta da nova rede e se a profundidade é parâmetro relevante para a melhora nos resultados. A arquitetura, nesse cenário, consiste em:

1) Camada de entrada
   a) 512 neurônios
   b) Ativação Relu
2) Primeira camada escondida
   a) 256 neurônios
   b) Ativação Relu
   c) Regularização L2
3) Segunda camada escondida
   a) 128 neurônios
   b) Ativação Relu
   c) Dropout 30%
4) Terceira camada escondida
   a) 64 neurônios
   b) Ativação Relu
   c) Dropout 30%

5) Quarta camada escondida
   a) 32 neurônios
   b) Ativação Relu
6) Quinta camada escondida
   a) 16 neurônios
   b) Ativação Relu
7) Camada de saída
   a) 10 neurônios
   b) Ativação: Função logística

Esse modelo foi treinado por 200 épocas. A taxa de aprendizagem foi de 0.005 e a taxa de regularização 0.01. A função de custo foi, novamente, a entropia cruzada. Como o número de camadas aumentou em comparação com o primeiro cenário então as penalizações da rede através das regularizações se fazem mais presente a fim de proporcionar robustez à rede.

O gráfico mostrado em 7 indica as acurácias de treinamento e da validação ao passar das épocas. Observa-se que a velocidade na qual a rede converge é mais lenta que o primeiro cenário. Em parte, isso deve-se ao grande número de camadas e dos pesos na qual a rede deve ajustar para proporcionar uma crescente taxa de acurácia. As regularizações também influenciam na velocidade de convergência da rede.

A acurácia geral no teste foi de 97.08%. Dessa forma, nesse conjunto de dados e nessa arquitetura proposta o aumento do número de camadas escondidas em comparação ao primeiro cenário não tem uma correlações com a performance da MLP em termos de acurácia no conjunto de teste.

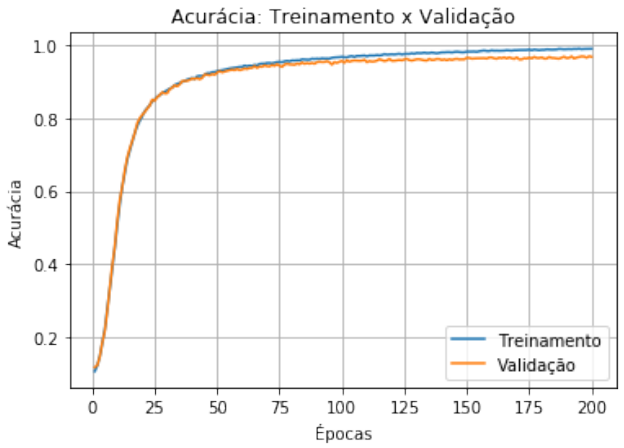

Figura 7. Cenário 02. Validação do treinamento

C. Terceiro cenário

A arquitetura, nesse cenário apresenta o mesmo número de camadas que o primeira proposta apresenta. As mudanças estão na aplicação de 3 categorias de regularização: L1,L2 e dropout e o uso do ADAM como função de otimização. A arquitetura, em camadas, está disponível abaixo.

1) Camada de entrada
   a) 512 neurônios
   b) Ativação Relu
2) Primeira camada escondida
   a) 256 neurônios
   b) Ativação Relu
   c) Regularização L2
3) Segunda camada escondida
   a) 128 neurônios
   b) Ativação Relu
   c) Dropout 40%
4) Terceira camada escondida
   a) 64 neurônios
   b) Ativação Relu
   c) Regularização L1
5) Camada de saída
   a) 10 neurônios
   b) Ativação Softmax

Esse modelo foi treinado por 200 épocas. Foi fixado 0.01 como taxa de regularização e 0.005 como taxa de aprendizagem.

A atualização dos pesos foi feita conforme o método da descida estocástica do gradiente adaptativo. Esse método de otimização é chamado de Adam e ele difere do método tradicional de descida do gradiente por apresentar uma taxa de aprendizagem adaptativa que leva em consideração as estimativas do primeiro e segundo momentos dos gradientes [3].

O gráfico acurácia x época nessa abordagem está ilustrado na figura 8. Desse gráfico observa-se que com menos épocas atingimos taxas de acurácia comparáveis aos dois cenários apresentados anteriormente. Esse comportamento é muitas vezes desejado para uma rede neural porque, no geral, consome menos tempo para atingir resultados satisfatórios.

Porém, em virtude do ADAM o treinamento apresenta certa 'instabilidade' nos valores de acurácia computados, motivo pelo qual o gráfico apresenta uma taxa de ruído considerável. A acurácia geral obtida de 97.04% e, portanto, um pouco abaixo dos casos anteriores.

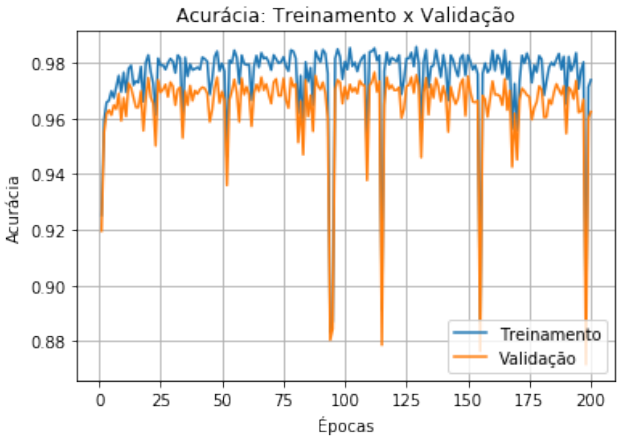

Figura 8. Cenário 03. Validação do treinamento

D. Quarto cenário

A quarta proposta consiste na arquitetura apresentada no terceiro cenário com a mudança na função de custo. Aqui utilizamos a função de erro quadrático cuja equação está mostrada em 7 como função de custo. O método de otimização será, novamente, o Adam. No treinamento ilustrado em 9 notamos que os ruidos das curvas persistem porém de forma mais 'branda' que o terceiro cenário. A partir da época de número 50 a acurácia da validação parece ter atingido um valor médio em torno de 98% enquanto que o treinamento cresce continuamente em direção a acurácia de 100%. Esse comportamento pode indicar que o modelo esteja se especializando excessivamente nos dados de treinamento. Todavia, como a validação não tem uma tendência de decrecimento ao longo das épocas isso provavelmente não ocorreu.

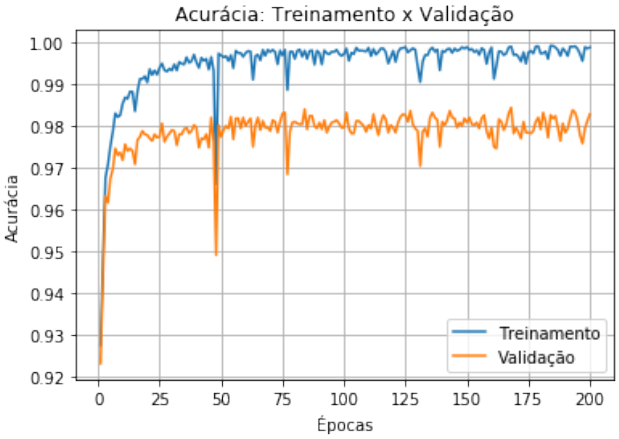

Figura 9. Cenário 04. Validação do treinamento

A matriz de confusão ilustrada em 10 mostra que a acurácia geral foi de 98.3%. Esse foi o melhor resultado dentre os casos apresentados neste trabalho. Em comparação ao primeiro cenário observa-se, por exemplo, menos erro na classificação dos dígitos 9 e 4 e entre 7 e 1. Os dígitos 9 e 3 foram os que tiveram a menor taxa de acurácia dentre as 10 classes com um valor de 97%.

A utilização do erro qudrático no nosso exemplo representou um ganho em termos de acucrácia.

IV. Conclusões

Do exposto, é possível concluir que o MLP é uma rede neural capaz de classificar dígitos manuscritos de 0 a 9 com acurácia acima de 97% através de uma das nossas arquiteturas propostas e treinando a rede com o dataset do MNIST. A partir dos resultados obtidos, percebeu-se que a utilização de cada um dos propostas de rede possui prós e contras. A utilização de um número de camadas acima de 5 não representou melhora na generalização do problema. A utilização do ADAM como técnica para variar a taxa de aprendizagem melhorou o desempenho em termos de velocidade de convergência mas não implicou, necessariamente, em melhora nas taxas de acurácia na classificação dos dígitos (como visto na terceira proposta).

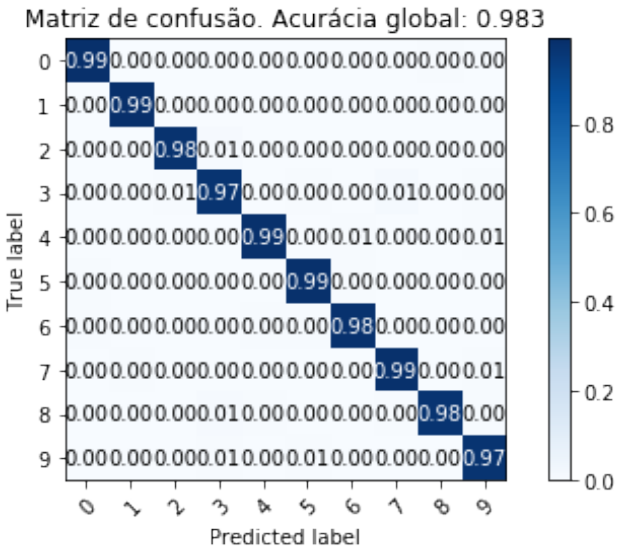

Figura 10. Cenário 04. Matriz de Confusão

A estrutura em 5 camadas e usando o erro quadrático com função de custo apresentou o melhor resultado na predição das classes. Seu aspecto ruidoso, visto durante o treinamento, pode não ser desejável para as aplicações práticas.

Referências

[1] Multilayer Perceptron, DeepLearning 0.1 documentation. Disponível em: http://deeplearning.net/tutorial/mlp.html. Acessado em: 14 de maio de 2019.

[2] The MNIST Database of handwritten digits. Disponível em: http://yann.lecun.com/exdb/mnist/. Acessado em: 17 maio de 2019.

[3] Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, Jason Brownlee. Disponível em: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. Acessado em: 14 de maio de 2019.

[4] Multi-Layer perceptron using Tensorflow, Aayush Agrawal. Disponível em: https://towardsdatascience.com/multi-layer-perceptron-using-tensorflow-9f3e218a4809. Acessado em: 08 de maio de 2019.

[5] Redes Neurais, Perceptron Multicamadas e o Algoritmo Backpropagation https://medium.com/ensina-ai/redes-neurais-perceptron-multicamadas-e-o-algoritmo-backpropagation-eaf89778f5b8. Acessado em: 13 maio 2019.

[6] Deep Learning Book - Capítulo 8, Função de Ativação http://deeplearningbook.com.br/funcao-de-ativacao/. Acessado em: 13 maio 2019.

[7] Deep Learning Book - Capítulo 23, Como funciona o Dropout? http://deeplearningbook.com.br/capitulo-23-como-funciona-o-dropout/. Acessado em: 17 maio 2019.

[8] Deep Learning Book - Capítulo 22, Regularização L1 http://deeplearningbook.com.br/capitulo-22-regularizacao-l1/. Acessado em: 17 maio 2019.