Análise do desempenho de diferentes configurações de redes neurais MLP no problema de classificação de imagens de dígitos MNIST

André Luiz Moreira Dutra - 2019006345
Aprendizado de Máquina

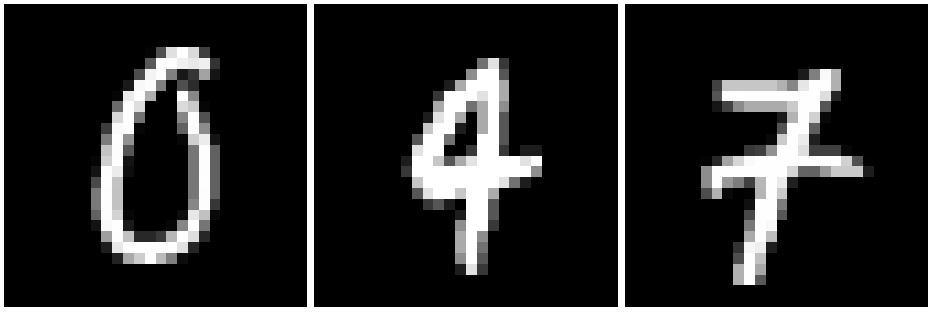
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

cienciandre@ufmg.br

1. Introdução

Este trabalho se propõe a resolver o problema de classificação de imagens do conjunto de dados MNIST utilizando como modelo redes neurais do tipo MLP, bem como apresentar uma análise comparativa dos desempenhos do modelo para diferentes escolhas de parâmetros.

A base de dados MNIST constitui um conjunto de imagens em preto e branco de tamanho 28x28 de dígitos escritos à mão, associados a seus respectivos rótulos representando o dígito escrito. Observe alguns exemplos:



Rótulo: 0 Rótulo: 4 Rótulo: 7

O objetivo do classificador é, ao receber uma imagem arbitrária de um dígito escrito à mão, na forma de um vetor 28x28 com as cores da imagem, definir corretamente o rótulo associado à imagem. Ou seja, definir qual dígito foi desenhado.

Para isso, foi usada uma rede MLP (multi-layer perceptron) com três camadas: uma camada de entrada, uma camada oculta e uma camada de saída. Variações de diversos parâmetros do modelo foram avaliadas, e a calibragem dos mesmos foi feita utilizando validação cruzada. Desse modo, neste trabalho serão apresentadas todas as escolhas feitas

com relação à montagem do modelo, bem como uma análise comparativa dos resultados do modelo para os dados de treino, validação e teste.

2. Estrutura e Organização da Rede

Conforme pré-definido, a rede foi estruturada em três camadas: uma camada de entrada com 28*28=784 nós, que recebe a imagem em formato achatado em uma dimensão (flattened), uma camada oculta de tamanho variável e sujeito a calibragem na validação, variando entre 25, 50 e 100 nós, e uma camada de saída, com 10 nós. Cada nó (com exceção dos nós de entrada, que apenas fornecem os dados iniciais) utiliza a função sigmoide como função de ativação.

O treinamento da rede foi feito utilizando o algoritmo backpropagation, utilizando uma base total de 5000 exemplos. A função de perda utilizada foi a entropia cruzada binária (binary cross-entropy), já implementada no framework utilizado, representada pela seguinte expressão:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right]$$

Além do tamanho da camada oculta, os demais parâmetros variados na validação tinham relação com o treinamento do modelo: a taxa de aprendizado e o algoritmo de cálculo do gradiente. A taxa de aprendizado utilizada foi testada para os valores 0.5, 1 e 10. Já para o algoritmo de cálculo do gradiente, foram usadas quatro variações:

- Gradient Descent
- Mini-batch com batches de tamanho 50
- Mini-batch com batches de tamanho 10
- Stochastic Gradient Descent

Na prática, os quatro algoritmos são quase idênticos, variando apenas no tamanho do batch (subconjunto dos dados) utilizado. Para o GD, todos os exemplos do treino são passados de uma vez a cada passo, enquanto para o mini-batch o número de exemplos passados é definido pelo tamanho do batch. Por fim, o SGD passa apenas um exemplo a cada passo.

3. Escolhas de Implementação

Nesta seção serão detalhadas as escolhas feitas sobre a estruturação e implementação do modelo que não foram calibradas pela validação cruzada. São elas:

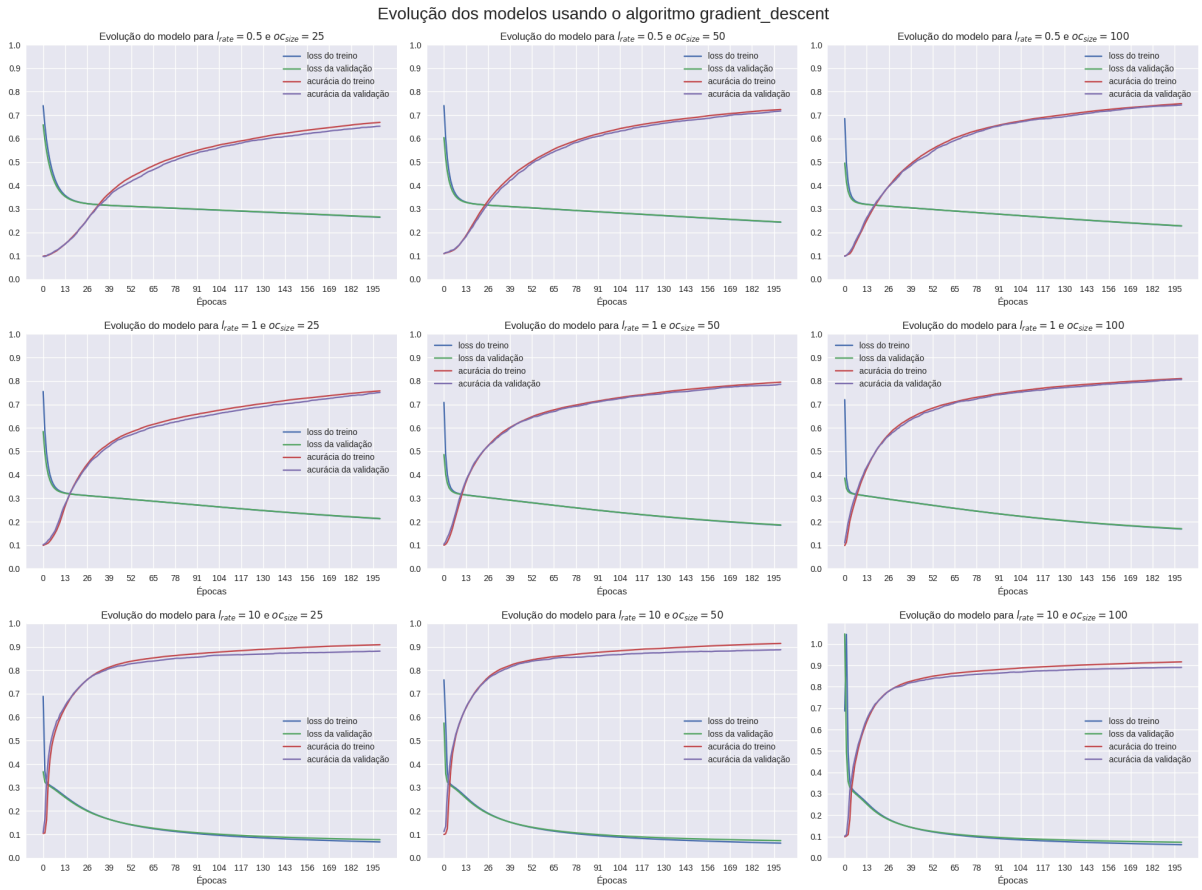
- **Framework de implementação do modelo:** O modelo foi implementado utilizando como framework principal o Tensorflow. Outras bibliotecas foram utilizadas, no entanto, como auxiliares, como o pandas para tratamento dos dados, numpy para organização dos vetores, sklearn apenas para a separação dos cortes da validação cruzada e pyplot para o plot dos resultados.
- **Tamanho dos conjuntos de treino, teste e validação:** A escolha da divisão dos dados em treino e teste foi feita utilizando, dos 5000 exemplos totais, 90% para treino e 10% para teste. Foi definido também que, do total dos dados de treino, 10% seria utilizado para a validação, correspondendo a 9% do total. Como foi feita a validação cruzada, os dados de treino foram divididos em 10 partes contendo 9% dos dados cada e o mesmo modelo foi avaliado 10 vezes, utilizando cada parte como validação do treino das demais.
- **Número de épocas**: Como o Tensorflow foi utilizado com a GPU, o tempo de cálculo das métricas de cada batch foi independente do tamanho do batch. Assim, o tempo de execução de uma época do treino do modelo dependia somente do número de batches rodado na época pelo algoritmo de cálculo do gradiente utilizado, já que o treino com as batches é sequencial e não pode ser paralelizado. Desse modo, para atingir as restrições de tempo de entrega deste trabalho, algoritmos com mais batches por época foram executados durante menos épocas, já que a cada época executavam mais batches, que correspondiam a passos. Com isso, o GD, com 1 batch por época, foi executado por 200 épocas, o mini_batch de tamanho 50 foi executado durante 50 épocas, o mini_batch de tamanho 10 durante 10 épocas e o SGD durante 4 épocas, totalizando cerca de uma hora de treino.

4. Análise Comparativa

Dada a estrutura da rede, devemos agora calibrar os parâmetros do modelo. Conforme mencionado, os parâmetros variáveis, sujeitos a validação, e suas respectivas variações são os seguintes:

- **taxa de aprendizado (l_{rate}):** 0.5, 1 ou 10
- **tamanho da camada oculta (oc_{size}):** 25, 50 ou 100
- **algoritmo de cálculo do gradiente:** gradient_descent, mini_batch_50, mini_batch_10 ou stochastic_gradient_descent

Como os resultados do modelo utilizando diferentes algoritmos foram os mais discrepantes, os resultados foram agrupados por algoritmo. Observe os resultados das variações que utilizaram o Gradient Descent:



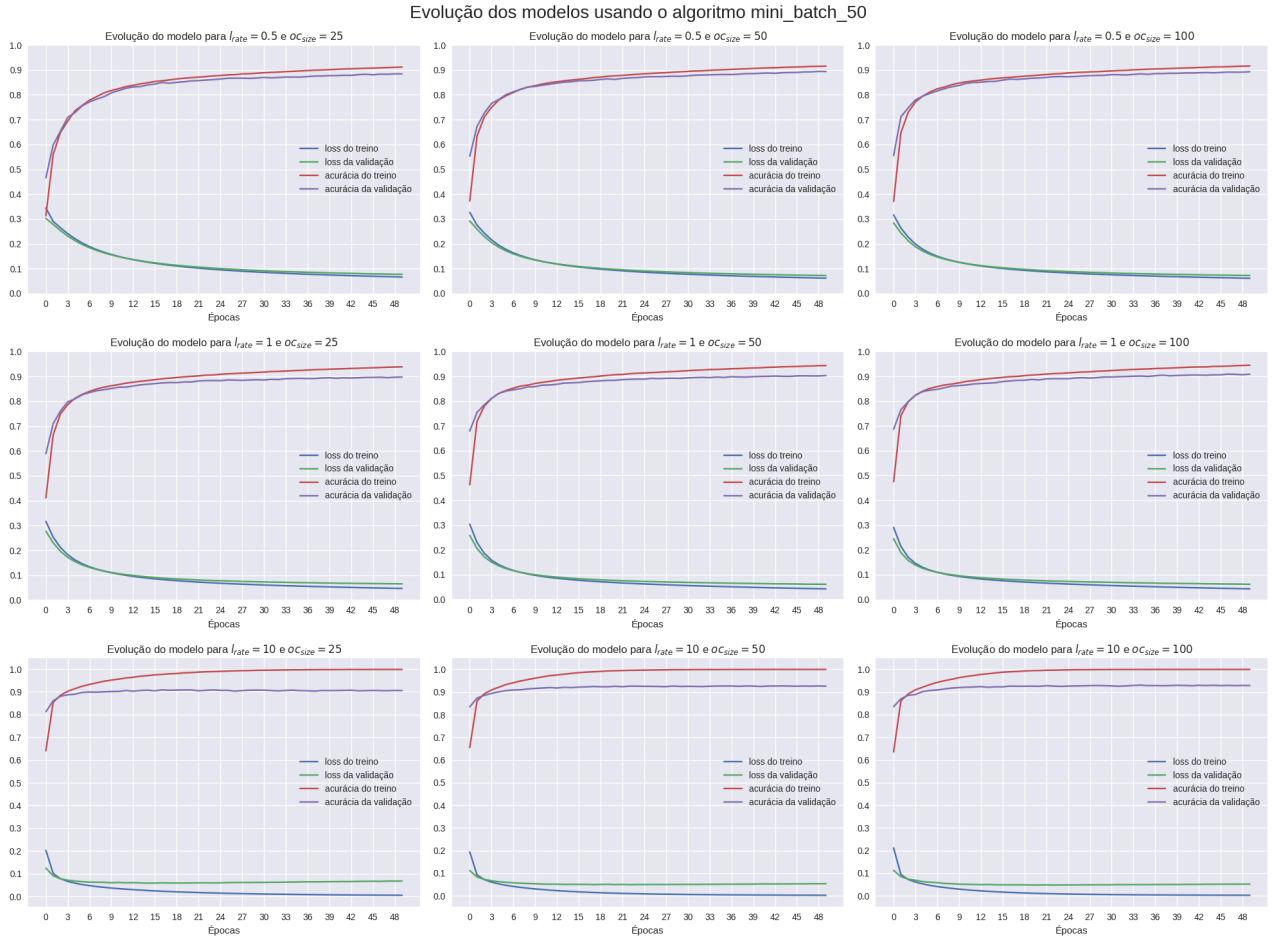
É notável que, dentre as demais, as variações que usaram o GD foram as que mais demoraram para que a loss convergisse a algum valor. Após aproximadamente 10 épocas, a loss começou a cair quase que linearmente, sendo a com menor taxa de queda entre as losses de todos os algoritmos. No entanto, o GD foi também o algoritmo com menor oscilação dentro das métricas de treino, o que é esperado, já que o gradiente do GD aponta com maior precisão para o mínimo local buscado dentre todos os algoritmos testados.

Com relação às diferenças entre as outras variáveis, elas foram menos perceptíveis. Quase não se vê diferença entre modelos que variam apenas no número de nós na camada oculta, exceto por um leve, quase imperceptível aumento na acurácia para modelos com camada oculta maior, principalmente para menores taxas de aprendizado. No entanto, esta diferença foi tão irrisória que não nos permite tirar nenhuma conclusão. A taxa de aprendizado, no entanto, apresentou diferenças mais evidentes: para este algoritmo, maiores taxas de aprendizado ocasionaram em acurácias maiores ao fim do treino. Isso muito

provavelmente ocorre pelo fato de que o GD converge muito lentamente, fazendo com que uma taxa de aprendizado maior acelere mais a convergência do algoritmo sem estourar pra fora do mínimo local.

Este também foi o algoritmo onde o erro empírico (loss do treino) menos descolou da previsão do erro esperado (loss da validação). Mas isso muito provavelmente ocorreu porque o algoritmo nas 200 épocas sequer teve tempo de convergir o suficiente paraque ocorresse overfitting e esse descolamento fosse visível. O fato de a maior taxa de aprendizado ser a única a apresentar este descolamento em certo grau corrobora esta hipótese.

Observe agora os resultados das variações que utilizaram o mini_batch_50:

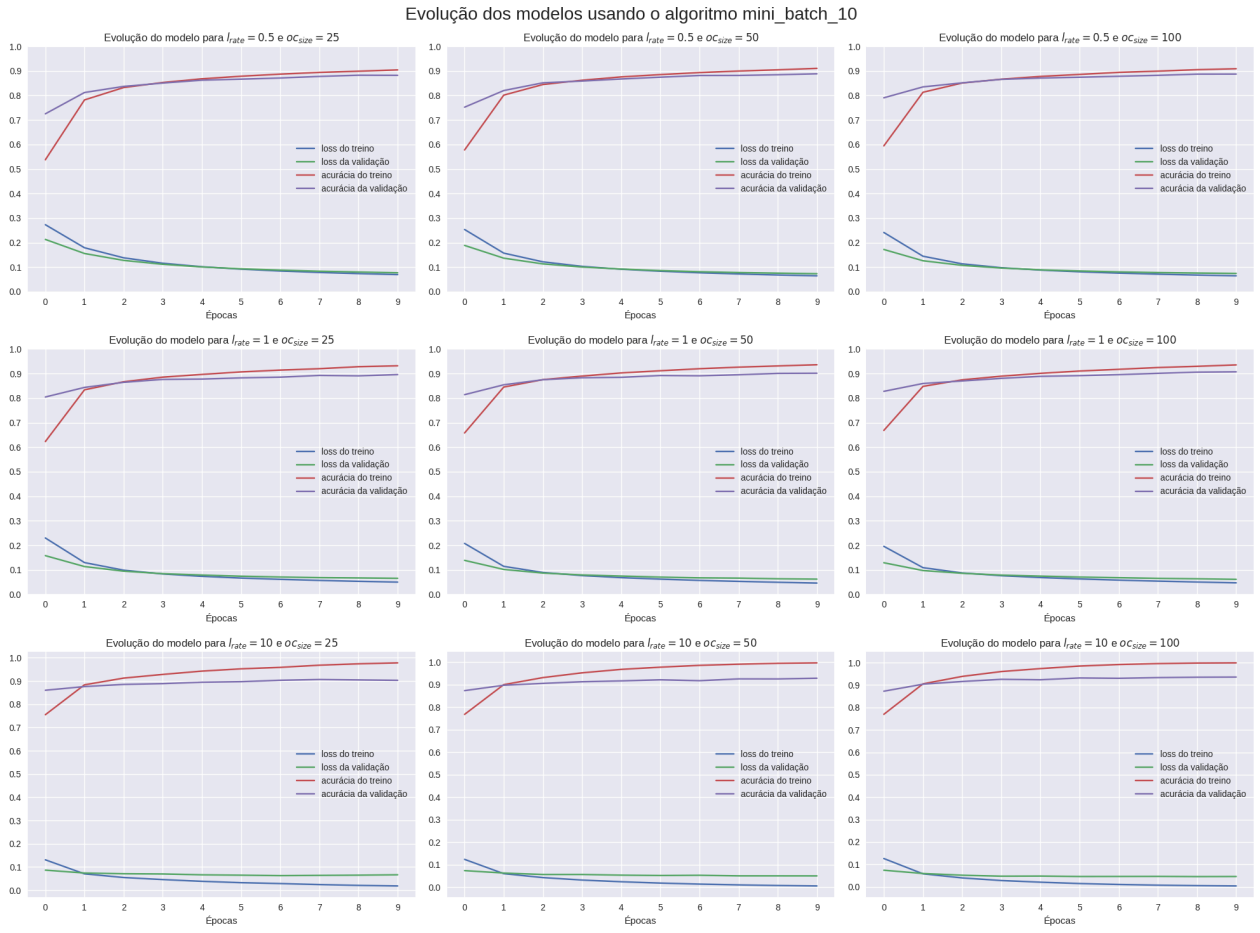


Neste caso, os dados começam a oscilar um pouco mais, mas ainda continuam bem estáveis. Parte disso ocorre pelo fato de que foram documentados apenas os resultados do fim de cada época, e não o de cada passo, então ocorre uma ilusão quanto à estabilidade dos resultados ao longo do treinamento. Isso se torna ainda mais evidente para os próximos algoritmos, que foram rodados com menos épocas. É evidente que, nesse caso, o algoritmo

convergiu muito mais rápido. A acurácia da validação ficou acima de 90 em todos os casos, mesmo com menos épocas.

Com relação ao número de nós na camada oculta, pouca diferença é visível neste caso também. Camadas maiores parecem aumentar a acurácia da validação muito irrisoriamente em alguns casos, mas em nenhum grau significativo. No entanto, as diferenças causadas pela variação da taxa de aprendizado foram bem maiores nesse caso que no GD. Nesse caso, quanto maior a taxa de aprendizado, mais o erro empírico se descolou da predição do erro esperado, ao ponto em que para taxa de aprendizado máxima as variações convergiram para acurácia de treino de 100%, embora não houvesse aumento da acurácia da validação. Possivelmente, a alta taxa de aprendizado, junto à rapidez com que o algoritmo convergiu, tenham feito com que o gradiente estourasse em torno do mínimo local e nunca o alcançasse, estagnado em pontos que não aumentam a acurácia real do modelo mesmo que a do teste estivesse em 100%.

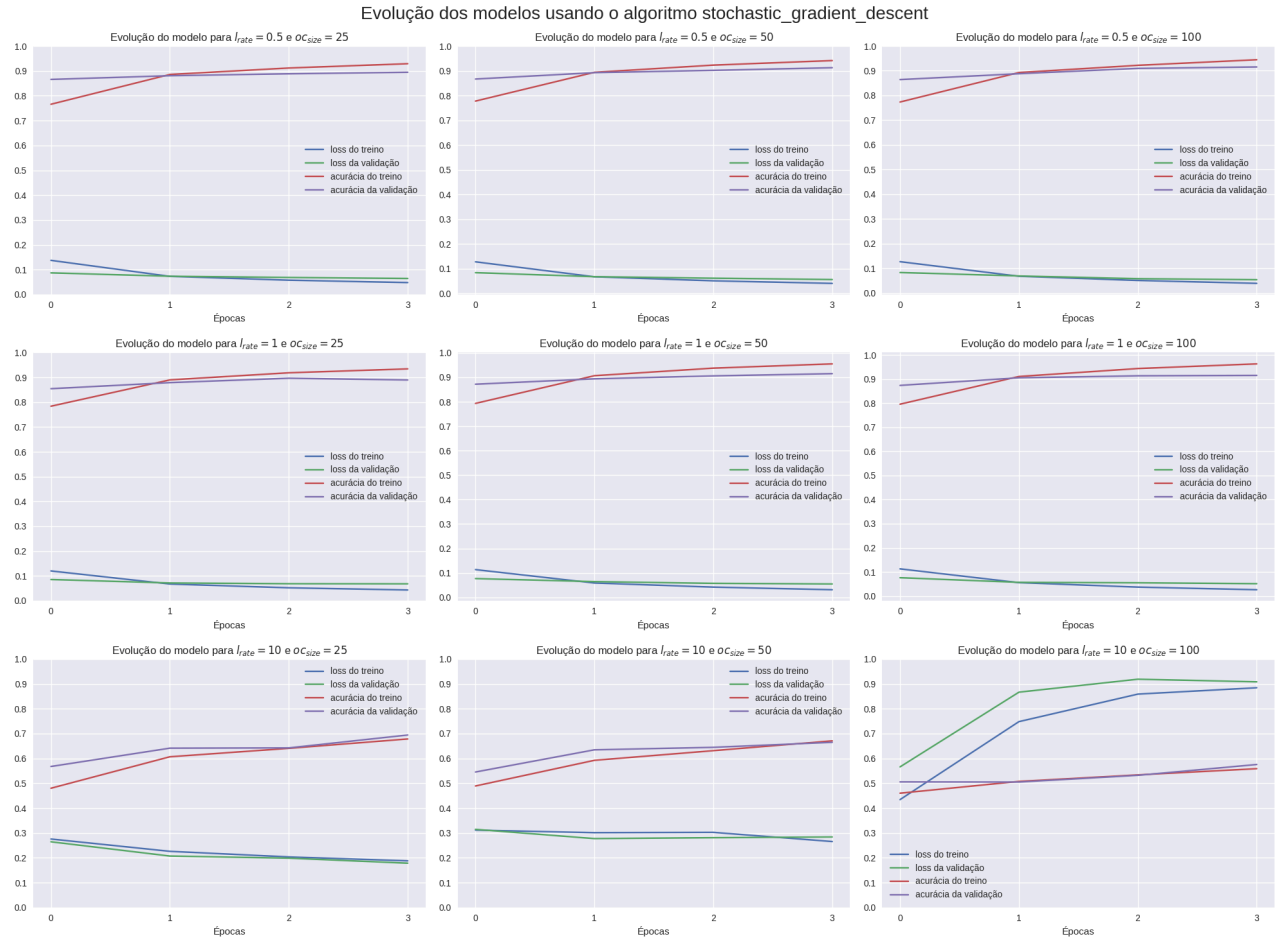
Vejamos agora os resultados do mini_batch_10:



É importante notar que este algoritmo atingiu resultados muito parecidos ao anterior, porém em um quinto da quantidade de épocas. Embora as acurácias máximas em todos os casos sejam muito próximas ao caso anterior, elas ainda foram ligeiramente melhores, principalmente utilizando número máximo de nós na camada oculta e máximo de taxa de aprendizado, alcançando cerca de 94% de acurácia na validação, a maior entre todos os modelos testados.

As variações entre o número de nós na camada oculta e a taxa de aprendizado também têm comportamento muito parecido ao caso anterior. No entanto, nesse caso o erro empírico descolou menos da previsão do erro esperado, muito provavelmente porque a quantidade reduzida de épocas impediu que o erro empírico estourasse para o mínimo.

Por fim, vejamos os resultados do stochastic_gradient_descent:



O primeiro ponto a se observar é a velocidade com que este algoritmo convergiu. Na primeira época, a maior parte das variações tinha a acurácia de validação perto de 90%, e em quatro épocas a maioria das variações estavam acima de 90% em acurácia de validação.

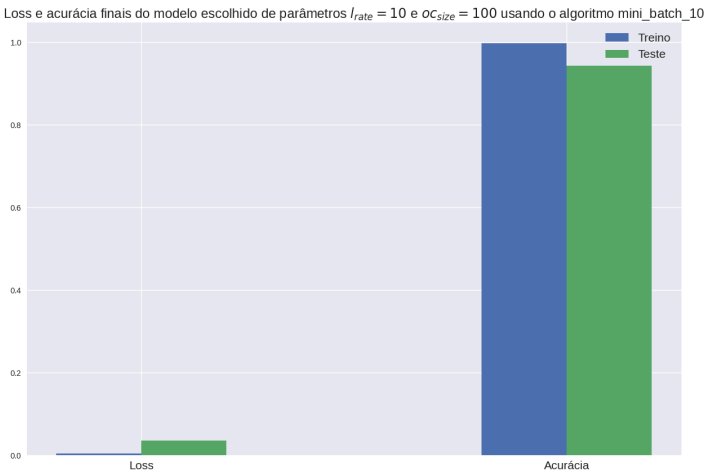
Este caso, no entanto, mostra valores bem peculiares para a taxa de aprendizado máxima. Em todos os casos, a acurácia dos modelos não superou 70%, diminuindo quanto maior era a camada oculta. Para além disso, com 100 nós na camada oculta, a função de perda pela primeira vez apresentou crescimento ao invés de diminuição. Neste caso, muito provavelmente a alta velocidade de convergência associada à alta taxa de aprendizado fizeram com que o modelo estourasse para mais longe do mínimo local buscado, aumentando a loss. Isso se torna ainda mais plausível quando levamos em consideração o fato de que o SGD é o algoritmo que converge com maior velocidade, embora com menor precisão.

Por fim, o modelo com melhores resultados foi o que usou como parâmetros a taxa de aprendizado 10, a camada oculta de tamanho 100 e o algoritmo de cálculo de gradiente mini batch com batches de tamanho 10. Com isso, podemos calcular a loss e a acurácia reais usando o conjunto de teste, guardado desde o início, para os quais obtemos os resultados absolutos:

```
Loss do Treino:      0.004158034920692444
Loss do Teste:       0.03535422310233116

Acurácia do Treino:  0.9982222318649292
Acurácia do Teste:   0.9440000057220459
```

E de forma comparativa:



Um ótimo resultado de 94.4% de acurácia para uma arquitetura simples de rede.

5. Conclusão

Este trabalho permitiu a consolidação do aprendizado sobre redes neurais, e a observação da forma como seu aprendizado é afetado por diferentes parâmetros. Por meio dele, foi possível observar, na prática, os benefícios e malefícios de cada parâmetro em cada quantidade, por meio de dados concretos que permitissem compará-los. Também foi possível aprender sobre o Tensorflow, um framework robusto para gerenciamento de redes neurais e deep learning, e de uso prático em outras áreas de pesquisa e trabalho. Por meio dos dados, podemos concluir, por fim, que dentro das opções de calibragem do modelo é possível resolver o problema de classificação das imagens do MNIST com acurácia de pelo menos 94,4%, o que é um resultado muito promissor considerando a simplicidade da arquitetura da rede.

Bibliografia

VELOSO, Adriano. *Slides da disciplina Aprendizado de Máquina. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. 2023.*

Module: tf | TensorFlow v2.12.0. Disponível em: <https://www.tensorflow.org/api_docs/python/tf>. Acesso em: 25 maio. 2023.