Projeto 2 - MC886

Victor Ferreira Ferrari
RA 187890
vferrari@mpc.com.br

Vinícius Couto Espindola
RA 188115
vinicius.c.e@hotmail.com

I. Introdução

O trabalho tem como objetivo principal a exploração de técnicas para resolver problemas de classificação, como regressão logística e redes neurais densas. Utilizando uma base de dados de imagens, visamos reconhecer os *tradeoffs* e evitar *overfitting* de diferentes modelos. Uma base de dados com imagens torna o problema mais difícil, seja pela grande quantidade de *features*, ou pela pequena quantidade de informações que cada *feature* representa, sendo este um único pixel.

II. Condições e Conjunto de Dados

Na implementação do modelo, foram utilizadas diversas bibliotecas externas da linguagem *Python*, sendo essas `NumPy`, para operações vetorizadas, `MatPlotLib`, para criação de gráficos para visualização de resultados ou comparações. Além disso, bibliotecas padrão da linguagem também foram utilizadas.

O conjunto de dados a serem explorados consiste de uma série de imagens coloridas 32x32 separadas em 10 classes (avião, automóvel, pássaro, gato, veado, cachorro, sapo, cavalo, navio, caminhão). Os dados utilizados são um subconjunto de 100000 imagens do *CINIC-10*, que consiste de 270000 imagens.

Os conjuntos de treino, validação e teste já estão separados, com 80000 imagens para treinamento, 10000 para validação e 10000 para teste, que deve ser utilizado apenas para avaliação final, quando o melhor modelo estiver totalmente definido.

III. Implementação

Os conjuntos de treino e validação podem ser lidos de um arquivo de extensão `.npz` com o auxílio da biblioteca `NumPy`. Para a *feature scaling* dos dados testamos três métodos: *min-max normalization*, *mean normalization* e *standardization*. Nota-se que as estatísticas de normalização calculadas no conjunto de treino se mantêm a todos os outros, para preservar a normalização.

A inicialização dos pesos de ambas as técnicas utilizadas é aleatória, e como técnica foi escolhida a chamada *Xavier Initialization*.

Procurando garantir que o modelo final é o melhor encontrado, o conjunto de coeficientes que fornece o menor custo de validação é o retornado ao final do treinamento. Para essa verificação e visualização do andamento, os valores de custo para conjuntos de treino e validação são calculados e salvos a cada época.

A. Regressão Logística Multinomial

A regressão logística multinomial foi feita com a função **Softmax** como função logística, que converte um vetor de ”pontuações” por classe em um vetor de probabilidades, no qual cada posição indica a probabilidade daquele exemplo pertencer àquela classe.

Foi implementado *gradient descent* para regressão, e o tipo escolhido foi **Batch Gradient Descent (BGD)**. Ressalta-se que todos os modelos foram vetorizados ao máximo por questões de otimização. O único critério de parada é limite de épocas. No futuro, pode-se implementar um critério de *early stopping* baseado, por exemplo, na acurácia do modelo no conjunto de validação a cada iteração ou época.

B. Rede Neural Densa

No escopo do trabalho, foi implementado apenas redes com camadas densas que são propagadas pela função sigmoide, uma vez que implementar e testar múltiplos tipos de camadas e funções de ativação seria inviável. O programa consiste de três classes: *Network*, *Meta* e *Optimizer*.

A classe Network é a representação da arquitetura da rede, que consiste essencialmente das matrizes de pesos da rede. Sua inicialização apresenta três principais parâmetros: indicativo de função de custo (*sigmoide* ou *softmax*), o parâmetro de regularização e a arquitetura. A seleção da função de custo define, além do custo em si, a derivada a ser calculada na última camada da rede. A arquitetura (definida por uma lista de inteiros) pode assumir qualquer combinação de número de camadas e nós por camadas. Para o manuseio destes dados, diversas funções foram implementadas dentro da classe Network, as quais executam tarefas como *forward* e *backward propagation*, cálculo do custo a partir de um conjunto de amostras, função de treino da rede que executa o *gradient descent* (GD) dentre outras que variam de salvar a rede em um arquivo *.npz* à calcular a acurácia da rede.

A fim de facilitar a seleção de otimizadores, criou-se a classe *Optimizer*. Esta é instânciada durante o treinamento da rede neural e define o processamento pelo qual os gradientes calculados no *backpropagation* serão otimizados. Três métodos foram implementados. O método *Vanilla* é apenas a mutiplicação do gradiente pela *learning rate*, enquanto o *Adadelta* e o *Adam* [1] utilizam métodos mais sofisticados. Ao ser inicializada, a classe *Optimizer* define parâmetros para cada um dos métodos de otimização.

Para realizar o monitoramento do treino, foi criada a classe *Meta*. Esta classe também é instanciada durante o treino da

rede e abrange uma série de metadados relacionados ao treino da rede neural: lista de índices de amostras randomizados a cada época para aleatorizar os *batches*, contagem de épocas completadas e dados destes (*loss*), o tempo que cada época leva para ser completada, número de *batches* completados, dentre outros. Os dados coletados em um objeto *Meta* são essenciais para a análise dos modelos e dos métodos de treino.

O treinamento da rede neural é realizado pela função *train* dentro da classe *Network*, que realiza o *gradient descent* de três possíveis maneiras: *Batch* (BGD), *Mini Batch* (MBGD) e *Stochastic* (SGD). A quantia de 80k amostras tornou tanto o BGD quanto o SGD inviáveis, pois ambos são lentos para este caso (o BGD para atualizar o gradiente e o SGD para finalizar uma época), o que levou o MBGD a ser o escolhido para treinar as redes. A dinâmica de treino decorre de forma que um laço é executado até que o limite de épocas ou de tempo seja alcançado. Neste laço, primeiramente, seleciona-se um conjunto de amostras para compor o *batch* atual. em seguida, é chamada a função *back_prop* que retorna os gradientes dos pesos. Resta, então, processar os gradientes. Esta tarefa é delegada para a função *optimize* da classe *Optimizer*, que processa os gradientes de acordo com a otimização selecionada. O **Adadelta** possibilita a geração de *learning rates* adaptativas que dependem dos gradientes já calculados considerando um decaimento, e que são individuais para cada parâmetro da rede, todavia, é computacionalmente caro por necessitar uma quantia relativamente grande de cálculos por iteração. Para evitar ter que salvar os *W* gradientes, a somatória do quadrado dos gradientes é aproximada pelo decaimento da somatória dos quadrados de todos os gradientes calculados desde o início do treino. Assim, utilizamos a raiz do decaimento dos quadrados dos deltas dividida pela raiz do decaimento dos quadrados dos gradientes como *learning rate* de cada parâmetro à cada iteração. O **Adadelta** não necessita que seja definido um *learning rate*, mas requer o valor do decaimento como parâmetro.

Outra opção é o **Adam**, que é um dos métodos mais utilizados de otimização de GD. Ele pode ser descrito como uma combinação das vantagens de dois outros otimizadores: *AdaGrad* e *RMSProp*. Este consiste em manter um *learning rate* por parâmetro, baseado em magnitudes de gradientes recentes. O algoritmo mantém médias móveis exponenciais do gradiente e do gradiente quadrado, controladas por parâmetros de decaimento β_1 e β_2 [1]. Por fim a *optimize* retorna os deltas que devem ser somados aos pesos da rede a fim de minimizar o custo.

Ressalta-se alguns detalhes sobre a implementação. No caso dos resultados esperados, utilizou-se uma função para transformar o vetor de N resultados para *one-hot encoding* em uma matriz *10xM*. A validação da implementação da rede foi realizada utilizando pequenas redes para resolver operações lógicas (*and, or, xor & xnor*), tanto para a *sigmoide* quanto para a *softmax*. No caso da *softmax*, a última camada apresentava dois nós: um para classe `verdadeiro` e outro para a classe `falso`.

C. Métricas de Avaliação

Diversas métricas de avaliação são comuns para problemas de classificação. As principais métricas utilizadas são **acurácia**, **precisão**, **revocação** e **F1-Score**. Para certos problemas, alguma dessas métricas pode ter um peso maior que as outras, porém no problema apresentado a acurácia, que é a métrica mais simples, já avalia de maneira decente a solução.

Uma estrutura muito utilizada para visualização e cálculo dessas métricas é a **matriz de confusão** do modelo. A matriz indica, para cada classe alvo, qual a classificação do modelo. A partir dela, pode-se calcular a acurácia pela razão entre o **traço** da matriz e a soma de todos os elementos da mesma. As outras métricas também podem ser calculadas a partir da matriz de confusão, mas suas definições não escalam para múltiplas classes, então essas métricas são definidas **por classe**.

IV. Soluções Propostas

A abordagem inicial para resolver o problema foi uma regressão logística multinomial. Porém, ela não foi a melhor escolha para o *dataset*, uma vez que esta é equivalente a uma rede neural de duas camadas, ou seja, a complexidade dos modelos que esta pode representar é limitada.

Com uma rede de várias camadas podemos modelar conjuntos de dados mais complexos, mas para que isso seja possível necessitamos decidir uma arquitetura que consiga represetar os dados adequadamente (sem *over* ou *underfitting*) e que não seja muito demorada para treinar. Então arquiteturas de diferentes dimensões e complexidades foram testadas, e no final foi escolhida uma rede com **duas camadas escondidas** de tamanhos 256 e 128.

A função de ativação escolhida para a rede neural foi a **sigmóide**, por familiaridade com seu uso e implementação. Foi cogitada a utilização da função *softmax* na última camada, mas a ideia não retornou os melhores resultados.

Em relação ao preprocessamento da entrada, optamos pela normalização dos dados para que estes se comportem melhor com uso da sigmóide e também tentamos filtrar os canais de cores das imagens convertendo-as para imagens em *grayscale*, possibilitando a otimização do tempo de treino de redes.

Os hiperparâmetros da rede neural e da regressão logística foram definidos empiricamente, em sua maioria, após extensivos períodos de teste. As exceções são os otimizadores, cujos parâmetros possuem sugestão na literatura. Para o **Adadelta**, o único parâmetro é o decaimento. Este foi configurado para 0.99 seguindo a literatura. Para o **Adam**, a literatura sugere que β_1 seja 0.9, β_2 seja 0.999, e o *learning rate* seja 0.001.

Outra característica que afetou o resultado de maneira não-desprezível foi o tipo de normalização. Dos três tipos de normalização implementados, o que forneceu o melhor resultado foi a *padronização* (ou *standardization*, reduzindo o custo final e aumentando levemente a acurácia.

V. Experimentos and Discussões

A primeira abordagem ao problema foi com a regressão logística multinomial, com *learning rate* de 0.01 e limite de 300 épocas. Os resultados estão a seguir:

- Custo Mínimo de Validação: 1,9569
- Acurácia no Conjunto de Treino: 30,9%
- Acurácia no Conjunto de Validação: 29,8%
- F1-Score **Médio** no Conjunto de Validação: 0.2887

Não houve *overfitting*, mas percebe-se que o resultado não é satisfatório, com uma baixa taxa de acertos. Por isso, os experimentos seguintes foram realizados com uma rede neural densa.

Inicialmente com a rede neural, definimos a função de custo a ser utilizada. Comparando a primeira arquitetura treinada com ambas as funções de custo obtivemos que, utilizando a função *softmax* na última camada, tivemos uma convergência mais lenta do que se utilizarmos a sigmoide, como está demostrado na Figura 1. Isso pode ser justificado pelo fato que, por estarmos utilizando a sigmoide nas camadas ocultas, o erro de ativação dos nós da última camada é relativamente pequeno pois os expoentes utilizados na *softmax* são entre 0 e 1. Caso a função de ativação `ReLu` fosse utilizada, possivelmente teríamos resultados mais satisfatórios com a *softmax*.

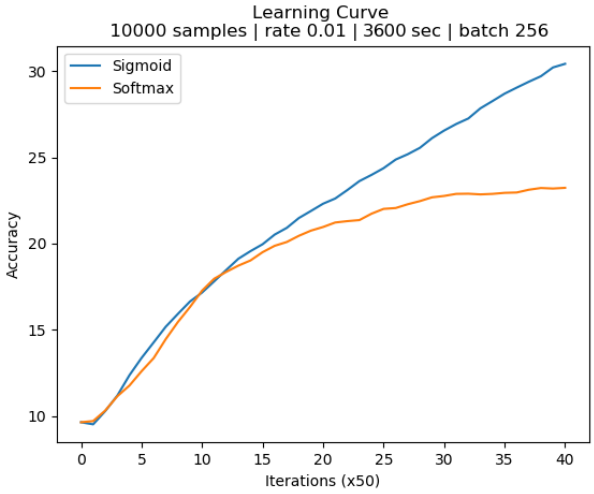

Figure 1. Curvas de aprendizado de uma rede neural densa com custo final dado pelas funções *sigmoide* e *softmax*

Testando cinco arquiteturas de rede distintas, as arquiteturas 2, 3 e 4 demonstraram instabilidade e não apresentaram uma boa taxa de convergência. Já as arquiteturas 1 e 5 convergiram de um modo melhor, diferindo na velocidade de treinamento, número de iterações e estabilidade. Tais observações seencontram na Figura 2. A arquitetura escolhida como final foi a identificada como 5, definida abaixo.

Arquiteturas (nós por camada):

1) L1:3072, L2:3072, L3:10
2) L1:3072, L2:1563, L3:768, L4:384, L5:192, L6:10
3) L1:3072, L2:1024, L3:342, L4:114, L5:40, L6:10
4) L1:3072, L1:768, L2:192, L3:48, L4:10
5) L1:3072, L2:256, L3:128, L4:10

Após definida uma arquitetura e a função de custo, foi treinada a primeira rede neural focada em melhorar os resultados da regressão logística. Dentre os hiperparâmetros

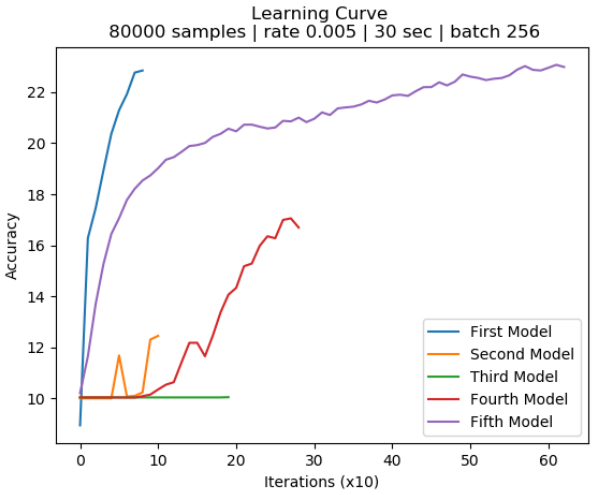

Figure 2. Curvas de aprendizado de uma rede neural densa a partir cinco arquiteturas (identficadas de 1 a 5) sobre as mesmas condições.

utilizados, temos que o tamanho do mini-batch foi de 256 exemplos, com limite de 500 épocas e 300 segundos. *Learning rate* utilizado foi 0.01. Os resultados estão a seguir:

- Custo Mínimo de Validação: 2,7832
- Acurácia no Conjunto de Treino: 33,3%
- Acurácia no Conjunto de Validação: 33,0%
- F1-Score **Médio** no Conjunto de Validação: 0.3166

Percebe-se que os resultados obtidos foram perto de 10% melhores que os vistos na regressão logística, mas ainda há necessidade de encontrar resultados melhores. É possível notar também que não houve *overfitting*.

Procurando melhorar os resultados, foram testados otimizadores na rede. Os selecionados para implementação foram o **Adadelta** e o **Adam**. As três redes foram testadas com os mesmos hiperparâmetros, modificados em relação ao teste anterior. Foram testadas as redes por 200 segundos, com limite de 50 épocas, *batches* de 1024 exemplos e *learning rate* de 0.001. Os parâmetros dos otimizadores já foram descritos.

Comparando a acurácia dos métodos no conjunto de validação, temos que o **Adam** forneceu o melhor resultado nos testes preliminares, com 40%, seguido pelo **Adadelta**, com 37,4%, e por último o método *Vanilla* (chamado assim quando não possui otimizador), com 21%.

A partir da Figura 3, observa-se que os otimizadores aceleram siginificativamente o aprendizado da rede neural, e assim descarta-se o método *Vanilla*. Quanto ao **Adam**, apesar de apresentar os melhores resultados no menor tempo, também é perceptível que o modelo sofreu *overfitting*.

Assim, foi encontrado o melhor modelo dentre os testados: uma rede neural densa com otimizador Adam, tamanho de batch igual a 1024 exemplos e *learning rate* de 0.001. Como tentativa de eliminação de *overfitting*, um parâmetro de regularização L2 de 0.002 foi passado para a rede. Como a regularização diminui a velocidade do treinamento, a rede

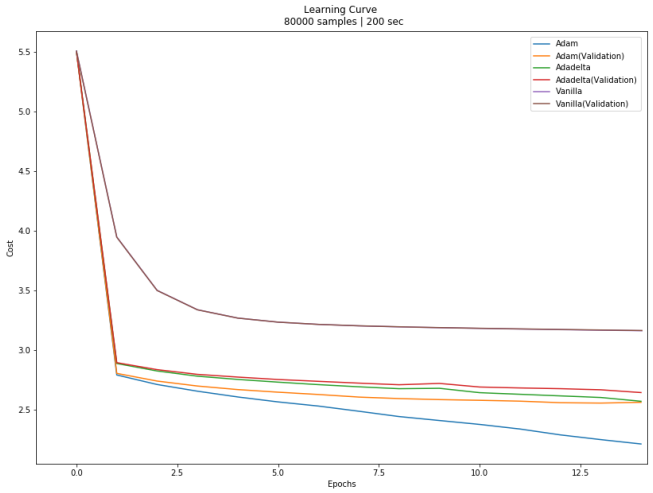

Figure 3. Curvas de aprendizado de treino e validação para modelos com otimizadores Adadelta, Adam e sem otimizadores (*Vanilla*), com os mesmos hiperparâmetros (batch size 1024, time limit 200s, *learning rate* 0,001).

Figure 4. Matriz de Confusão do melhor modelo de rede neural densa encontrado, com otimizador "Adam"

foi treinada com limite de 500 épocas, por 900 segundos (15 minutos). Os parâmetros de decaimentos são os mesmos da literatura, descritos anteriormente.

Os resultados dessa rede foram:

- Custo Mínimo de Validação: 2,4758
- Acurácia no Conjunto de Treino: 47,3%
- Acurácia no Conjunto de Validação: 42,3%
- F1-Score **Médio** no Conjunto de Validação: 0.4167

Então, foi confirmado que a rede descrita é a que fornece melhores resultados dentre as testadas. Com o modelo final, foi criada a matriz de confusão no conjunto de **teste** do modelo, para avaliar a generalização. A matriz pode ser vista na Figura 4.

Os resultados da rede no conjunto de teste foram:

- Acurácia: 41,5%
- Precisão **Média**: 40,7%
- Revocação **Média**: 41,4%
- F1-Score **Médio**: 0.4075

Também foram realizados testes com as imagens convertidas para *grayscale*, todavia, apesar do tempos de treino serem melhores, a remoção das cores causava perda significativa de informação, o que fazia com que, mesmo treinando mais rápido, os resultados eram piores do que com as imagens coloridas.

VI. Conclusões e Próximos Passos

A primeira conclusão deste projeto é que alguns problemas mais difíceis requerem uso de técnicas mais sofisticadas para serem resolvidos por Aprendizado de Máquina. A regressão logística multinomial não retornou bons resultados para o *dataset* CINIC-10, e mesmo com melhorias o resultado não ficará bom pois a técnica é muito simples para resolver bem o problema.

Embora os resultados com rede neural densa tenham sido melhores que os da regressão logística, a melhoria inicialmente não foi muito grande, sendo necessária a implementação de otimizadores para que uma grande diferença pudesse ser notada. Ainda assim, essa implementação trouxe outros problemas, como *overfitting* e portanto uma menor estabilidade graças a regularização.

O modelo final, embora não tenha produzido bons resultados, generalizou bem, com 41% de acurácia no conjunto de teste sendo que a acurácia de validação foi 42%. Isso é um indicativo de que a regularização funcionou, e não houve muito *overfitting*.

Assim, próximos passos para "atacar" o problema envolveriam explorar outros otimizadores, como a combinação de Adam e Nesterov, chamada de **Nadam** [1], ou outras funções de ativação, como *tangente hiperbólica* ou `ReLu`. Assim, além de possivelmente melhorar o desempenho da rede, pode também funcionar melhor em conjunto com a função *Softmax*. Além disso, como regularização L2 não é muito efetiva com otimizador `Adam` utilizado [2], outro tipo de regularização pode ser implementada. Além disso, os hiperparâmetros encontrados podem não ser ótimos, então mais testes com hiperparâmetros diferentes, principalmente ao modificar outras partes, podem melhorar o desempenho.

Por fim, percebe-se que certos problemas são mais difíceis de se encontrar uma boa resposta, por uma dificuldade do *dataset* ou do problema em si.

References

[1] S. Ruder, "An overview of gradient descent optimization algorithms", Outubro de 2016, [online] Disponível em: https://arxiv.org/abs/1609.04747.

[2] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization", Janeiro de 2019, [online] Disponível em: https://arxiv.org/abs/1711.05101v3