Relatório Trabalho 2 - Regressão Logística e Redes Neurais

Gabriel Marinho da Costa
RA 172269
g172269@dac.unicamp.br

Giovanni Bertão
RA 173325
g173325@dac.unicamp.br

I. Introdução

Tendo em vista a crescente busca pela digitalização e automatização dos processos, atualmente temos uma grande quantia de dados disponíveis para se trabalhar. Nas últimas décadas tem-se investido muito tempo, dinheiro e esforços em diversas técnicas, dentre elas estão as técnicas de classificação de dados, o que nos permite criar subconjuntos de um determinado conjunto de dados, onde cada subconjunto possui características em comum. Duas das mais conhecidas técnicas são: regressão logística e redes neurais. Dentre os tipos de regressão logística temos um baseado na função sigmoide, onde conseguimos fazer uma classificação mais simples, dividindo os dados em apenas duas classes, e outro um pouco mais sofisticado, baseado na função *softmax*, com o qual conseguimos separar em diversas classes. Em ambos temos uma probabilidade de um determinado dado pertencer à uma classe e, baseando-se nessa probabilidade, o conjunto de dados é classificado. Uma rede neural é um técnica muito mais complexa, ela é inspirada no cérebro humano, a ideia basicamente é que há vários neurônios conectados, desta forma há uma camada de neurônios de entrada para os dados, uma ou mais camadas de neurônios subsequentes, onde os dados serão processados, e uma camada de neurônios de saída, onde o resultado é entregue. Pode-se usar redes neurais para diversas coisas, inclusive classificação de dados, que será o intuito desta técnica nas seções seguintes.

Sendo assim, este trabalho trata-se da aplicação de regressão logística multinomial e redes neurais no conjunto de imagens *"CINIC-10"* [1], com o intuito de classificá-las corretamente em 10 classes diferentes.

II. Atividades

Dado que o problema se trata de uma classificação de imagens, vimos em aula [2] que regressão logística e redes neurais podem ser usadas com este propósito, além disso também aprendemos como implementá-las. Há indicações da professora de que redes neurais funcionam muito bem quando se trata de um conjunto de imagens. Com base nos slides vistos em aula e em uma implementação na internet [3], conseguimos construir nossa própria implementação do zero. Desta forma, testamos a implementação da regressão logística multinomial já no conjunto de imagens, mas utilizando 100, 1000, 5000 e 10000 imagens. A partir disso foi possível verificar se a implementação estava correta, a *Loss* ao longo das épocas e ter uma noção em relação à acurácia do modelo de acordo com o número de épocas e a taxa de aprendizado, assim ganhamos confiança para prosseguir.

Foram implementados duas arquiteturas de redes neurais, uma com uma camada escondida e outra com duas camadas escondidas. As redes possuem 3072 neurônios na camada de entrada e 10 neurônios na camada de saída. A quantidade de neurônios nas camadas escondidas e totalmente parametrizável. Nas redes, com exceção da ultima camada, é utilizado como função de ativação a função *ReLu*. Na ultima camada foi implementado a função *softmax*. Além da arquitetura geral, foram implementados três modos de descida de gradiente: Modo Vanilla, Modo Mini-Batch e Modo Estocástico. Além disso, afim de otimizar os resultados foram implementados os otimizadores *Momentum* e *ADAGRAD*.

III. Dataset

O conjunto de imagens original foi reduzido e transformado em matrizes, sendo assim temos uma parte para treino, que consiste em 80 mil linhas, sendo que cada linha possui um vetor de 3072 valores entre 0 e 255, que representam os pixels da imagem, e também dois outros conjuntos, validação e teste, que estão no mesmo formato, mas possuem apenas 10 mil linhas. Para cada um destes conjuntos temos um vetor correspondente às classes das imagens, que possui, no caso do conjunto de treino, 80 mil linhas, e nos conjuntos de validação e teste, 10 mil linhas, cada uma com um único valor, que representa a classe daquela imagem, ou seja, o resultado que desejamos obter após a aplicação das técnicas.

IV. Solução Proposta

Para realizar a classificação das imagens, primeiramente iremos utilizar a técnica de regressão logística multinomial, que consiste em um algoritmo de descida de gradiente que utiliza como hipótese a função *softmax*. Após isto, utilizaremos uma rede neural para realizar a classificação afim de obter um melhor resultado, nela será utilizada a função *softmax* também. Como de costume, normalizamos os dados para evitar problemas com escala e eventuais problemas em cálculos. Também transformamos a matriz de classes em uma matriz de vetores utilizando One Hot Encoding. A seguir detalhamos as técnicas empregadas na composição da solução:

A. Hipótese

Na regressão logística multinomial, a hipótese é uma função do produto escalar de dois vetores: o vetor x, que contém os valores das *features*, no caso, os pixels da imagem, e o vetor θ, que contém o valor dos parâmetros associados a cada *feature*. Dessa forma, nossa hipótese é expressa pela Equação 2. Logo, teremos 3072 *features*, também é necessário adicionar o valor 1 na primeira posição de cada vetor da matriz, sendo assim a dimensão do nosso vetor θ é de 3073x10, já que temos 10 classes diferentes.

$$g(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} \tag{1}$$

$$h_\theta(x) = g(x \cdot \theta) \tag{2}$$

B. Função de Custo

Para aplicar a descida do gradiente na regressão logística, é necessário definir uma função de custo, ou função *Loss*, que será responsável por atualizar os valores de θ a cada iteração esperando que a função de custo convirja para o mínimo global do modelo. Para tanto, a função de custo $J(\theta)$ utilizada neste trabalho pode ser verificada na Equação 3.

$$J(\theta) = \frac{-1}{m}\sum_{i=1}^{m}(y_i \cdot log(h_\theta(x_i)) + (1-y_i)\cdot(1-log(h_\theta(x_i)))) \quad (3)$$

C. Função Gradiente

Utilizamos os valores do gradiente da função de custo J para permitir que o valor de θ seja atualizado a cada iteração. O gradiente nada mais é que um vetor de cada derivada parcial de J em relação a cada um dos parâmetros de θ. A Equação 4 apresenta o gradiente utilizado e que pode ser sintetizada na relação da Equação 5.

$$\nabla J(\theta) = \left(\frac{\partial J(\theta)}{\partial \theta_1}, ..., \frac{\partial J(\theta)}{\partial \theta_{3072}}\right) \quad (4)$$

$$\nabla J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)\cdot x \quad (5)$$

D. Taxa de aprendizado

A taxa de aprendizado é fundamental na atualização dos parâmetros, a escolha de uma boa taxa de aprendizado permite que o algoritmo consiga chegar em um ponto de mínimo local ou global sem demorar muito e ao mesmo tempo sem passar reto deste ponto. Taxas relativamente pequenas fazem com que o algoritmo demore para encontrar tal ponto, ao passo que taxas muito altas fazem com que o algoritmo perca este ponto facilmente.

E. Descida do Gradiente

Apresentadas as equações anteriores, O Algoritmo 1 apresenta o processo de descida do gradiente. No Algoritmo 1 generalizamos a condição de parada. Tal condição pode ser adotada como: "Enquanto não convergiu", "Repita X épocas", "Enquanto o erro médio for maior do que Y", etc. Como já esperávamos não alcançar um resultado tão satisfatório com a técnica de regressão logística, não faria sentido colocar como critério de parada um número suficientemente pequeno para o erro médio, portanto utilizamos como condição o número de épocas, que foi de 5000.

Algoritmo 1 Descida do Gradiente

while condição **do**
  $\theta = \theta - \alpha\nabla J(\theta)$
end while

F. Descida do Gradiente em Mini-Batch

Esse modo de execução, permite executar a descida do gradiente em um conjunto menor de dados. Uma iteração é realizada ao processar um conjunto fixo (reduzido) de dados. O vetor de pesos é atualizado a cada iteração. Depois que todos os dados do dataset (inteiro) foram utilizados, consideramos uma época. Dados os pesos gerados em cada época podemos coletar o valor da *Loss* e compor o gráfico *Loss* vs época.

G. Descida do Gradiente Estocástico (SGD)

Este terceiro modo de execução seleciona em cada época um dos exemplares do dataset de forma aleatória e realizada o processo de descida do gradiente com ele. Assumir um único exemplo para um época pode parecer ingenuo. Contudo, as propriedades de escolhas estocásticas permitem uma taxa de convergência mais rápida sem perda de precisão.

H. Função ReLu

A função ReLu (Rectified Linear Unit) foi utilizado, em nosso trabalho, como a função de ativação das camadas da rede neural. A Equação 6 ilustra a função e a Equação 7 a derivada da mesma.

$$f(x) = max(0, x) \quad (6)$$

$$f'(x) = \begin{cases} 1, \text{se x} > 0 \\ 0, c.c. \end{cases} \quad (7)$$

I. Rede Neural

Uma rede neural pode ser interpretada com uma cadeia de regressões logísticas. Temos um conjunto de nós de entrada que é utilizado na função de hipótese. O resultado da hipótese é processado por uma função de ativação, como por exemplo a ReLu. O resultado da função de ativação pode ser visto como um conjunto de entrada para outra camada. Dessa forma, é possível estender o procedimento para qualquer número de camadas. Cada uma das camadas, que não as de entrada e de saída, são chamadas de camadas escondidas. Por fim, a função de ativação da penúltima camada gera os valores da camada de saída que serão comparados com os valores esperados por meio da função de custo — *Loss*. Assim como na regressão logística, ligando cada uma das camadas, temos um conjunto de pesos. Esses pesos são atualizados com base no gradiente da função de custo e o peso específico. $\frac{\partial J}{\partial \Theta}$. O processo de atualização utiliza o processo de descida do gradiente: Inicialmente os valores de cada neurônio é calculado, tal processo é conhecido como *feed-forwarding*. Uma vez na ultima camada o erro acometido é propagado para as camadas iniciais utilizando o processo de *backpropagation*. Por fim, os pesos entre as camadas são atualizados. Esse processo se repete para um número pré-determinado de épocas. O interessante é que podemos aplicar os diferentes modos de descida do gradiente.

J. Otimizadores

O processo de execução da rede neural consiste em encontrar o mínimo global ao minimizar a função de custo. Esse processo pode ser custoso em relação a recursos de tempo e de espaço. Para tanto, existem métodos que otimizam o processo de aprendizado da rede neural. Em nosso trabalho, aplicamos os seguintes dois métodos:

1) Momentum: Esse otimizador consiste na utilização de um vetor (velocidade) que é atualizada a cada época com base no produto do seu antigo valor com uma constante γ e no gradiente do custo em relação aos pesos. A Equação 8 9 mostra o processo de atualização dos pesos utilizando esse otimizador.

$$v_t = \gamma v_{t-1} + \alpha\frac{\partial J}{\partial \Theta} \quad (8)$$

$$\theta = \theta - v_t \quad (9)$$

2) ADAGRAD: O otimizador anterior trabalha diretamente com os pesos da rede neural. Contudo, existem outras formas de otimizar o processo de aprendizado de uma rede neural. O ADAGRAD busca otimizar esse processo ao modificar a taxa de aprendizado. O processo consiste em definir um vetor de G, inicialmente zero, que é atualizado a cada iteração com o quadrado do gradiente dos pesos. Em seguida, G é utilizado para atualizar os pesos da rede. A Equação 10 e 11 ilustram esse procedimento.

$$G = G + \left(\frac{\partial J}{\partial \Theta}\right)^2 \tag{10}$$

$$\theta = \theta - \frac{\alpha}{\sqrt{||G|| + \epsilon}} \cdot \frac{\partial J}{\partial \Theta} \tag{11}$$

V. Experimentação e Discussão

A. Regressão Logística Multinomial

A primeira execução levou em consideração os seguintes parâmetros:

- θ: Vetor 3073x1 gerado de forma aleatória;
- Condição de parada: Repetição de 5.000 épocas;
- Taxa de aprendizado: 0.0001;

Podemos ver que o erro no conjunto de treino caiu, mesmo não alcançando um número baixo, entretanto o erro no conjunto de validação começou a crescer. A acurácia no conjunto de treino foi de 0.3998125, e no conjunto de validação, 0.3281. O desempenho pode ser conferido na Figura 1.

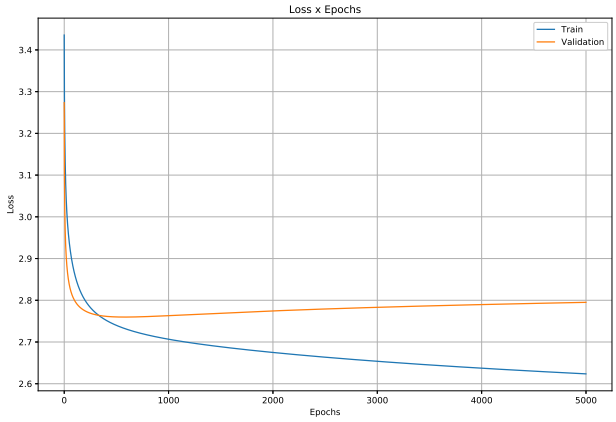

Figura 1. Relação entre a *Loss* e cada época iterada para a execução inicial

A seguir abaixamos um pouco mais a taxa de aprendizado, para 0.000001, e chegamos ao resultado da Figura 2. Dessa vez o erro no conjunto de validação permaneceu menor do que o erro no conjunto de treino, apesar da acurácia ter sido um pouco pior, 0.3151125 no conjunto de treino, e 0.3059 no de validação, os números permaneceram bastante próximos.

Para efeitos de comparação, nesta mesma configuração, testamos a taxa de aprendizado 0.1, o que resultou em um erro muito maior e que variava muito ao longo das épocas, ou seja, o algoritmo acaba saindo dos pontos de mínimos locais sempre que está chegando á algum, indício de que a taxa de aprendizado está alta demais. Também testamos uma taxa de aprendizado de 0.0000000001, onde o algoritmo estava convergindo, mas muito lentamente, desta forma teríamos que aumentar muito o número de épocas e investir mais tempo do que o necessário na execução, sinal de que a taxa de aprendizado estava baixa demais.

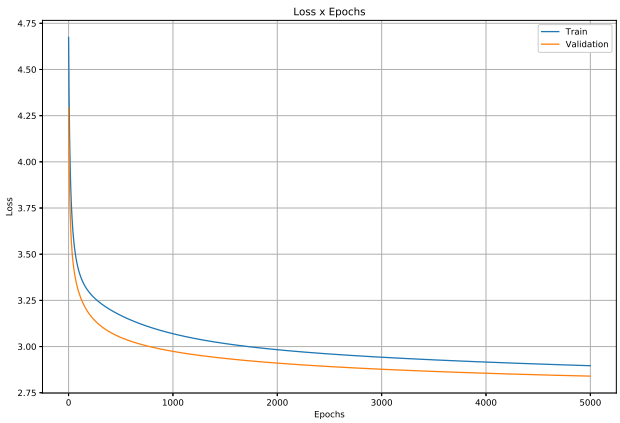

Figura 2. Relação entre a *Loss* e cada época iterada para a segunda execução

B. Rede Neural

Inicialmente montamos uma rede com uma única camada escondida, Vanilla e sem otimizadores. Executamos o conjunto de treino do *dataset* na rede e verificamos que a mesma apresentava uma acúracia de 0%. A Figura 5 mostra o gráfico da *Loss* vs. épocas observado, do mesmo concluímos que ele se manteve constante, ou seja, a rede não estava aprendendo. Tentamos aumentar o número de nós na camada intermediária, contudo, tivemos um aumento de tempo usado para treinar e os resultados continuavam os mesmos. Outra alteração foi a redução no número de exemplares. Foi decidido que de fato a rede não estava aprendendo, decidimos testar a rede com um conjunto fictício de dados — *ToyDataset* — que descreve a função lógica **and**. Rodamos 200 épocas, com 20 nós na camada escondida e uma learning rate de 0.01. A acurácia continuou como 0%. Contudo, ao observar o gráfico da Figura **??** observamos que aparentemente a rede estava aprendendo. Decidimos então executar o dataset em questão usando uma quantidade reduzida de exemplares. Reduzimos de 80k exemplares para 80, setamos o número de épocas para 90, 100 camadas escondidas e um learning rate de 0.001. Desse experimento, obtivemos a Figura **??**, que mostra a rede aprendeu algo. Contudo, a acurácia se manteve em 0$, tal resultado pode ser induzido da Figura **??** visto que o erro é menor que 5000. Com base, no resultado anterior decidimos utilizar a função de mini-batch, que usa uma quantidade reduzida de exemplares.Continuamos, com a mesma acurácia e o mesmo problema. Pensamos que talvez nossa rede só funcione com conjunto pequenos de exemplares. Pensamos, então, em aplicar a modo estocástico que sorteia somente um dos exemplares e realiza o processo de descida do gradiente. Novamente, obtivemos 0% de acurácia, mesmo no conjunto de treino. Tentamos utilizar os otimizadores *Momentum* e *ADAGRAD*. O processo levou mais tempo, cerca de 1 hora, e mesmo assim, continuamos com acurácia de 0% e o gráfico da nossa *Loss* continuou constante. Com base nesses resultados, geramos a hipótese de que talvez o nosso modelo seja muito fraco em relação ao conjunto de dados. Então, decidimos montar uma rede com duas camadas escondidas. Contudo, sem importar o como definíamos a densidade dos nós nessas camadas, o resultado era sempre o mesmo: acurácia 0%.

VI. Desempenho no Conjunto de Teste

Utilizando a configuração inicial da regressão logística multinomial, obtivemos uma acurácia no conjunto de teste de 0.3178. A

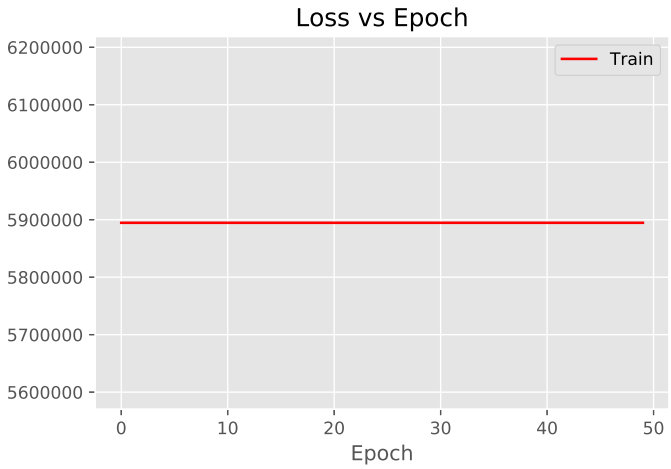

Figura 3. Relação entre a *Loss* e cada época iterada para a rede neural Vanilla. A rede não está aprendendo.

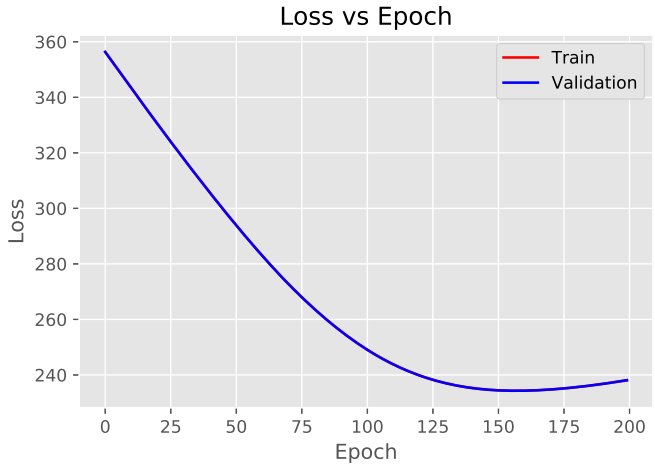

Figura 4. Relação entre a *Loss* e cada época iterada para a rede neural Vanilla do conjunto *Toy*. Talvez a rede esteja aprendendo.

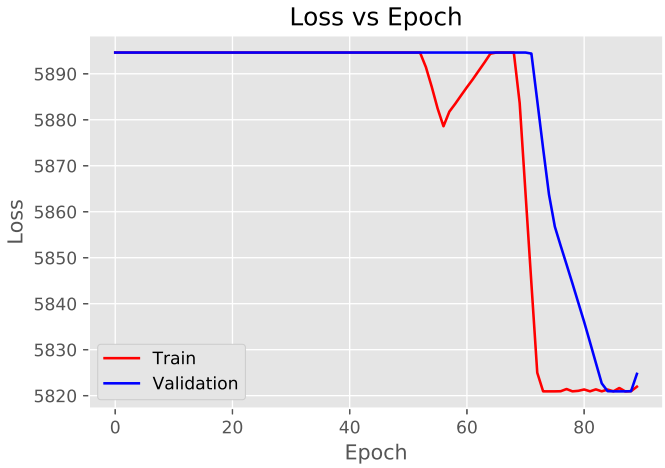

Figura 5. Relação entre a *Loss* e cada época iterada para a rede neural Vanilla em um conjunto reduzido de exemplares. Parece que a rede esta funcionando.

matriz de confusão da Figura 6 mostra os resultados da classificação no conjunto de teste e a confusão entre as classes que o algoritmo fez.

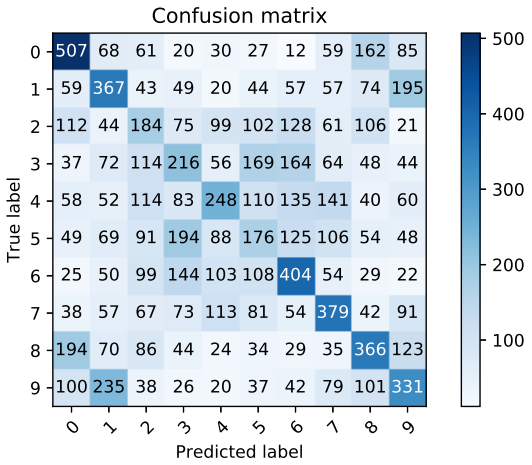

Figura 6. Matriz de confusão para a execução no conjunto de teste.

VII. Conclusão e Trabalhos Futuros

Com base nos resultados obtidos através dos experimentos e como o nosso modelo utilizado na rede neural não conseguiu produzir resultados satisfatórios. Elegemos como melhor modelo o produzido pela regressão logística. O resultado utilizando nosso melhor modelo não teve uma acurácia boa, foi de apenas trinta por cento no conjunto de teste, mas podemos concluir que tiramos um bom resultado em função das limitações deste modelo e conseguimos usá-lo corretamente. Daqui pra frente nos resta aperfeiçoar a implementação da rede neural, pois sabemos que ela certamente nos proporcionaria uma acurácia bem maior no quesito de classificação de imagens.

Referências

[1] L. Darlow, "Cinic-10." https://github.com/BayesWatch/cinic-10.
[2] S. Avila, "Logistic regression." http://www.ic.unicamp.br/~sandra/pdf/class/2019-2/mc886/2019-08-21-MC886-Logistic-Regression.pdf.
[3] A. Juliani, "Softmax regression." https://medium.com/@awjuliani/simple-softmax-in-python-tutorial-d6b4c4ed5c16.