Regressão Logística e Redes Neurais

Naomi Takemoto
RA 184849
naomitkm1@gmail.com

Thiago Dong Chen
RA 187560
thiagodchen@gmail.com

I. Objetivos

Estudar diferentes métodos de classificação e técnicas para ajustá-las de modo a evitar *overfitting*.

II. Introdução

O *dataset Fashion-MNIST* conta com 70000 imagens de 28px por 28px em preto e branco (com um único canal de cores), 10000 dessas imagens foram separadas para teste e as demais alocadas para treinamento. Os dados estão separados em 10 classes disjuntas relacionadas a *labels* numeradas de 0 a 9, cada uma representando um tipo de vestimenta. 0 t-shirt/top, 1 calça, 2 pullover, 3 vestido, 4 casaco, 5 sandália, 6 camisa, 7 tênis, 8 mochila, 9 ankle boot.

III. Teoria

Dentre as técnicas utilizadas para tratar o problema de classificação estão:

A. Regressão Logística

A regressão logística é comumente utilizada para realizar a classificação binária, isto é quando se deseja saber se um elemento pertence (1) ou não (0) a uma dada classe. Nesse método, utiliza-se a função sigmoide que tem como mapeamento números reais no intervalo (0-1), os quais representam:

$$\hat{y} = P(y = 1|x) \tag{1}$$

Onde $\hat{y}$ é definido como probabilidade condicional de um elemento pertencer a uma classe dado x (no problema tratado neste estudo x é o vetor contendo os pixeis de uma imagem). Para o caso em que existem $k \geq 2$ classes, pode-se utilizar a estratégia *one versus all* que consiste em implementar k modelos de regressão logística, um para cada uma das classes a que os elementos do conjunto de dados podem pertencer. Depois calcula-se a probabilidade condicional de que o elemento pertença à classe $0, 1, 2...$, a predição relacionada ao elemento é então escolhida como sendo a classe que apresenta a maior probabilidade. A função de entropia cruzada da regressão logística é:

$$L(\hat{y}^{(i)}, y^{(i)}) = -y^{(i)} \log(\hat{y}^{(i)}) - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \tag{2}$$

Para a fórmula acima, o sobrescrito (i) indica o elemento i-ésimo elemento do conjunto de treinamento. O processo de regressão se dá pela minimização iterativa da função de custo que é uma média da entropia cruzada para m exemplos:

$$J = \frac{1}{m} \sum_{i}^{m} L(\hat{y}, y) \tag{3}$$

B. Regressão Multinomial Softmax

Essa técnica pode ser vista como uma generalização da regressão logística. Com o uso da função *Softmax*, tem-se que o *label* $y \in (1, ..k)$, onde k é o número de classes A função de entropia cruzada para a regressão *softmax* é [1]:

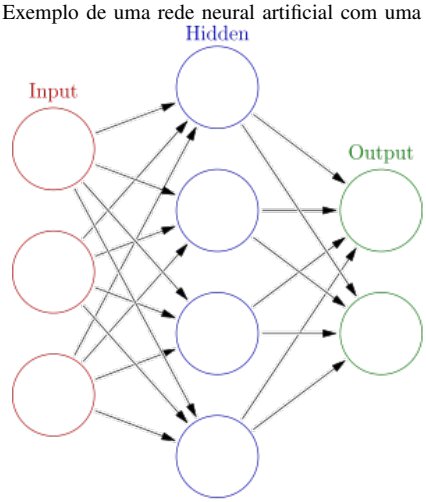
$$p_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \qquad L_i = -\log\left(p_{y_i}\right) \tag{4}$$

onde f é um vetor com as probabilidades de o elemento pertencer a cada classe, o custo L é calculado como uma "distância"da predição p ao *label*. De maneira análoga, a função de custo é também a média de L para todos os elementos do conjunto.

C. Redes Neurais

Essas redes são frequentemente usadas para tratar problemas supervisionados de classificação e não supervisionados de clusterização. Uma rede neural é formada por um conjunto de neurônios distribuídos em camadas(*layers*). Cada par de layers (l e $l-1$ por exemplo) podem ser vistas como se fossem uma regressão, sendo que o processo de minimização da função de custo é semelhante, baseado na descida de gradiente. Quanto mais "profunda"a rede, mais complexas são as funções que ela é capaz de aprender, mas com isso o treinamento também se torna mais difícil por conta de problemas como *gradient vanishing*. Quando esse fenômeno ocorre, a minimização do erro se torna lenta, assim como o aprendizado.

Figura 1. Exemplo de uma rede neural artificial com uma hidden layer



IV. Soluções Propostas

Para a classificação de imagens do conjunto de dados de Fashion-MNIST, foram propostos os seguintes métodos: regressão logística utilizando a estratégia *one-vs-all*, regressão logística com *softmax* e a utilização de redes neurais com uma e duas camadas escondidas. A implementação dessas últimas foi feita sem a utilização de *frameworks*, para explicitar os passos do treinamento (*forward propagation* e *backward propagation*). Com redes neurais é possível treinar modelos capazes de aprender funções mais complexas do que aquelas geradas pelas regressões descritas acima. No entanto, o treinamento dessas redes pode se tornar computacionalmente custoso na medida em que se aumentam o número de camadas e de neurônios. Uma estratégia adotada neste trabalho para a escolha do número de neurônios das camadas escondidas se baseou na fórmula: [3]

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o)} \tag{5}$$

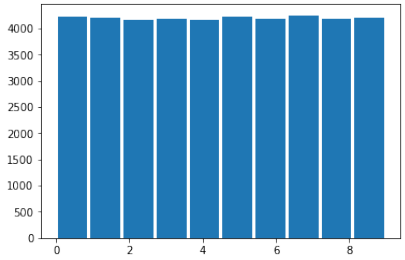
V. Experimentos e Discussões

Divisão do conjunto de dados

Quando se pretende abordar um problema com o uso de técnicas de *machine learning*, alguns cuidados quanto ao tratamento dos dados são necessários. O primeiro passo é dividir o *dataset* disponível tem três subconjuntos, treinamento, validação e teste.

O dado fornecido já está dividido em treinamento e teste. Para ter um conjunto de dados para escolher o melhor modelo, o conjunto de treinamento foi dividido em treinamento (70%) e validação (30%), utilizando a ferramenta *sklearn.model_selection.train_test_split*. Observando o a quantidade de amostras por classe, é possível observar que está balanceada por classe (ver Figura 2). Um outro pré processamento utilizado foi realização da operação de *scaling*, normalizando o valor de cada pixel pelo maior valor possível (255). Tal processo foi aplicado a todos os dados.

Figura 2. Histograma que mostra a frequência de amostras por classe



- **Regressão Logística**
- Para um número de épocas igual a 1000 (100 para cada classe), foi realizada uma regressão logística para cada classe $c \in (0, 1, 2..9)$, o objetivo era avaliar os elementos da entrada como pertencentes a c ou não pertencentes. Ao final do processo, cada imagem tinha 10 valores de probabilidade indicando a expectativa de ela ser uma instância de determinada classe. Na primeira parte do experimento, foi utilizada a técnica *K-Fold Cross Validation* com balanceamento de classes, implementada pela função StratifiedKFold, da bibiblioteca Scikit-Learn, sobre os dados do conjunto de treinamento e validação. O objetivo do uso dessa estratégia foi diminuir o tempo para a realização de experimentos, podendo-se também obter a informação de quão bem o modelo era capaz de generalizar. Como é possível percebera a partir da tabela

Tabela I
Porcentagem de inferências corretas em relação ao tamanho do *Fold* para 1000 épocas

Fold	***Acurácia Treino %***	***Acurácia validação %***
1	73.1	73.0
2	74.9	74.7
3	77.1	77.3

acima, o modelo conseguiu acertar cerca de 70% tanto em treino quanto em validação, ou seja ele é capaz de generalizar.

- **Regressão Multinomial**
- No experimento evolvendo a regressão multinomial foram usadas 100 épocas. A porcentagem de inferências corretas no conjunto de treinamento (42000 amostras) foi de 77.6 %. Já no conjunto de validação (18000 amostras) esse número caiu para 77.4%. Apesar de os valores estarem próximos, não caracterizando *overfitting*, foi implementada a regularização. a função de custo foi atualizada com o acréscimo do termo a seguir: [1]

$$\frac{1}{2}\lambda \sum_k \sum_l W_{kl}^2 \tag{6}$$

onde k a variável que itera sobre os exemplos e l sobre as *layers*. A derivada do parâmetro w deve ser acrescida de:

$$\frac{d}{dw}(\frac{1}{2}w^2) = \lambda w \tag{7}$$

Ao repetir o experimento com o uso desse recurso, foi necessário utilizar um *learning rate* 10x menor para evitar divergência. O resultado da minimização durante 100 épocas pode ser visto na figura 2, é possível observar que o custo atinge um valor mínimo entre 60 e 80 iterações, este caso justifica a utilização do *early stoping* a fim de evitar o crescimento do erro. Com 70 épocas, a porcentagem de inferências corretas foi de 66.2 % e no conjunto de validação foi de 66.5%.

Para se fazer uma análise mais precisa da acurácia do modelo, gerou-sea matriz de confusão (figura 3). No eixo vertical estão os *labels* (classes verdadeiras) e no eixo horizontal as classes preditas. Como é possivel obervar na matriz, a maior parte das predições foram corretas. No entanto, uma grande confusão ocorreu na predição dos elementos da classe 6 (camisa), que foram erroneamente colocados principalmente como 0 (t-shirt/top), 4(casaco) e 2 (pullover).

- **Rede Neurais**

Figura 3. Custo por época na regressão softmax com regularização

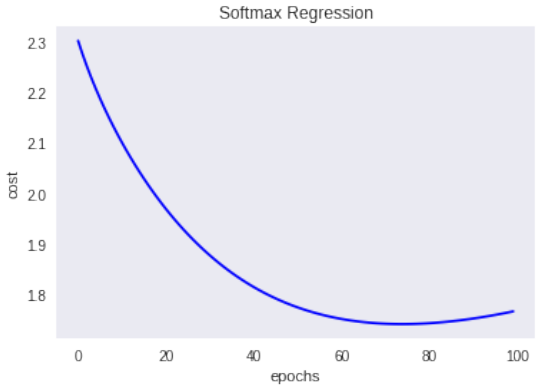


Figura 4. Matriz de confusão para a regressão multinomial *softmax* sem regularização

	0	1	2	3	4	5	6	7	8	9
0	1435	12	29	173	9	0	79	0	35	0
1	12	1668	32	67	11	0	7	0	1	0
2	25	3	1210	17	398	0	145	0	45	1
3	63	24	14	1570	63	0	75	0	7	0
4	3	10	221	131	1336	0	116	0	12	0
5	1	0	2	1	2	1143	0	425	15	181
6	420	10	266	86	301	1	669	0	55	0
7	0	0	0	0	0	36	0	1559	2	149
8	3	2	14	37	11	6	42	27	1677	1
9	0	0	0	1	1	16	1	101	1	1676

Na primeira tentativa de treinamento, após a primeira época do treinamento os pesos pararam de atualizar, utilizando apenas 16 neurônios na camada escondida. O fenômeno que pode ter acontecido *gradient vanishing*, que faz com que as derivadas parciais dos pesos sejam tão pequenos que a rede neural pare de treinar. A fim de solucionar isso, realizamos ajustes nos hiper-parâmetros (números de neurônios na camada escondida e o *learning rate*). Infelizmente, o erro não estava no ajuste nos hiper-parâmetros (ver Tabela II). A partir dos resultados da Tabela II, é notável que a rede não está "aprendendo", mesmo com os custos decrescendo. Os modelos de rede neural apresentados possuem acurácias que variam na faixa de 5%-20%, tanto no treinamento quanto validação. Isso alega que os modelos estão apenas classificando aleatoriamente as imagens inferidas, pois os resultados são próximos de 10%, que representa a esperança de acerto de um chute aleatório. A suspeita para a baixa porcentagem de acurácia, que seja um erro de implemetação da rede neural, especificamente no *backpropagation*.

Ao aumentar o número de camadas escondidas na rede neural não houve aumento na acurácia em nenhuma das funções de ativação. A acurácia continuou na faixa de 5%-20%, reforçando a hipótese que o erro está na codificação do algoritmo.

Após várias interpretações do algoritmo de *backpropagation*, percebemos que o algoritmo codificado não era o que se esperava, pois os gradientes dos outputs não estavam sendo considerados para os pesos entre o *input* e o *hidden layer*.

Depois de codificar o algoritmo corretamente, observamos que a acurácia subiu expressivamente, como pode-se observar na tabela III.

É importante ver que as redes neurais necessitaram mais épocas para que a a rede convirja para um ponto de mínimo em relação a regressão *softmax*. Necessitou aproximadamente 80 épocas para que a regressão atingisse um ponto de mínimo, já na rede neural, com 80 épocas, ainda estaria longe do ponto ótimo, ver Figuras 5 3 e Tabela III.

Observando a Tabela III, o aumento do número de camadas não necessariamente aumenta a acurácia. Ao aumentar o número de camadas, a quantidade de nodos na *hidden layer* que foi utilizado na rede neural pode não ser a mais adequada para o mesmo problema, ver Figura 7. Além disso, ao aumentar a quantidade de nodos de 20 para 128, houve aumento de 197s para 574s com a mesma quantidade de épocas (500). Isso é devido a necessidade de realizar maior quantidade de cálculos com matrizes, deixando a computação mais custosa. Portanto, deixar a rede neural mais profunda, em certos problemas pode não ser o mais adequado devido ao custo computacional em relação a acurácia adquirida pelo modelo.

Tabela II
INFLUÊNCIA DOS HIPER-PARÂMETROS NA REDE NEURAL DE UMA CAMADA ESCONDIDA

Eta	***Units na hidden layer***	***Acurácia Treinamento %***	***Acurácia validação %***
0.03	16	18.0	17.9
0.001	16	17.8	17.9
0.001	32	17.8	17.7
0.000001	128	6.2	6.3

Tabela III
INFLUÊNCIA DOS HIPER-PARÂMETROS E FUNÇÕES DE ATIVAÇÕES NA REDE NEURAL DE UMA OU DUAS CAMADAS ESCONDIDAS NA ACURÁCIA NO CONJUNTO DE VALIDAÇÃO

Números de hidden layers	***Units na hidden layer***	***Eta***	***Iterações***	***Acurácia Relu %***	***Acurácia Softmax %***
1	20	0.3	100	66.2	35.5
1	20	0.3	1000	85.1	64.3
2	20	0.3	1000	10.1	10.0
2	128	0.3	500	74.7	79.9

VI. O MELHOR MODELO

Unindo os conjuntos de treinamento e validação em um só, para realizar o treino do melhor modelo, conseguiu-se uma acurácia de 85,79%. Utilizando os pesos aprendidos para realizar a inferência, atingiu-se, no conjunto de teste que inicialmente havia sido isolado, uma acurácia de 85,65%. A pequena queda indica a não ocorrência de *overfitting*, portanto o modelo é capaz de generalizar. Analisando a matriz de

Figura 5. Custo por época na rede neural com 20 units na camada escondida com 1000 iterações

Figura 6. Custo por época na rede neural com 16 units na camada escondida com 100 iterações

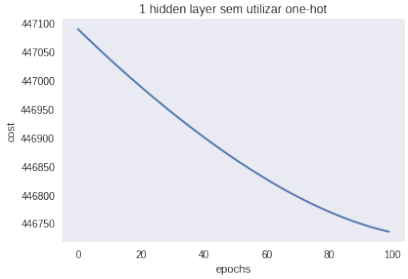


Figura 7. Custo por época na rede neural com duas camadas escondidas de 20 units cada com 1000 iterações

	0	1	2	3	4	5	6	7	8	9
0	796	2	6	22	2	0	158	0	13	1
1	1	967	5	17	5	0	5	0	0	0
2	10	1	693	7	161	0	119	0	9	0
3	39	17	11	855	39	0	36	0	3	0
4	0	0	39	21	833	0	105	0	2	0
5	3	1	0	0	0	918	1	46	7	24
6	113	4	74	15	75	0	704	0	14	1
7	0	0	0	0	0	40	0	915	0	45
8	3	0	3	2	7	5	17	6	956	1
9	0	0	0	0	0	19	0	34	1	946

Figura 8. Matriz de confusão para o melhor modelo, onde cada elemento da matriz C, C_{ij} pertence à classe i, mas a predição foi feita para j.

confusão, percebe-se que as classes 2(*pulover*) e 6(camisa), foram as que obtiveram o menor índice de previsões corretas, situação semelhante à encontrada na regressão multinomial com *softmax*. Além disso, não foi necessário a utilização da acurácia normalizada, pois tanto no conjunto de treinamento, quanto no conjunto de teste, as classes estavam balanceadas, ver Figuras 2 e 8.

VII. Principais Dificuldades

A principal dificuldade na implementação ocorreu no processo de *backward propagation* para as redes neurais. Além disso, para todos os modelos implementados, percebeu-se que fatores como a inicialização dos parâmetros é crítica, pois se forem utilizados valorem muito altos o processo se torna lento e por vezes a descida de gradiente diverge. A estabilidade numérica também foi um obstáculo para a implementação de algumas funções principalmente a sigmoide e a softmax que trabalham com exponenciais. Em muitos casos, ouve *overflow* ou *underflow*. Durante a execução do treinamento das redes neurais, percebeu-se que os seguintes agentes afetam a velocidade de execução do algoritmo:

1) Vetorização: fazer computação de valores de forma paralela aumenta a velocidade.
2) Função de ativação e inicialização: para parâmetros com valores altos, quando se usava sigmoide como ativação a execução ficava mais lenta em comparação com o uso da ReLu.

VIII. Conclusões e trabalhos futuros

Com o aumento do número de dados disponíveis para acesso, aliado à evolução do poder computacional na última década, as redes neurais ganharam destaque na área de inteligência artificial. Outro fator importante para a popularização dessa técnica, além dos resultados de alta acurácia em problemas de classificação é a existência de *frameworks* que automatizam e facilitam a implementação. Alguns dos principais *frameworks* são: Tensorflow, Torch, Keras, Caffe. Nesse sentido, uma maneira de otimizar o processo de se construir redes é utilizar as ferramentas citas acima, já que automatizam o *back propagation* e facilitam o processo de testar novas funções de ativação, variar o número de layers e neurônios por layer, etc. Uma outra estratégia para trabalhar com imagens é empregar redes neurais convolucionais, que possuem a vantagem de utilizar informações espaciais da imagem além de possuir um número menor de parâmetros que são compartilhados. Além disso, o treinamento dessas redes pode ainda ser otimizado se for realizado em GPU, a qual faz cálculos paralelos de forma mais eficiente que CPUs.

Referências

[1] "CS231n Convolutional Neural Networks for Visual Recognition". Disponível em: http://cs231n.github.io/neural-networks-case-study/. Acesso em: 02 out 2018.

[2] scikit-learn: machine learning in Python. Disponível em: ¡*http://scikit-learn.org/stable/*¿. Acesso em: 02 de outubro de 2018.

[3] model selection. Disponível em: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw. Acesso em 03 out 2018.

[4] Neural Network and Backpropagation explained in a simple way https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e. Acesso em 04 out 2018.

[5] Slides da prof. Sandra http://www.ic.unicamp.br/˜sandra/teaching/2018-2-mc886-mo444/. Acesso em 02 out 2018.