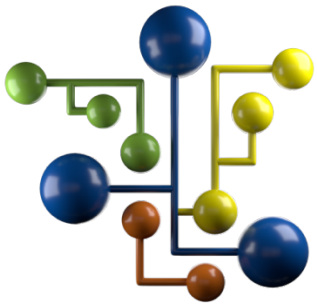


Data Science Academy

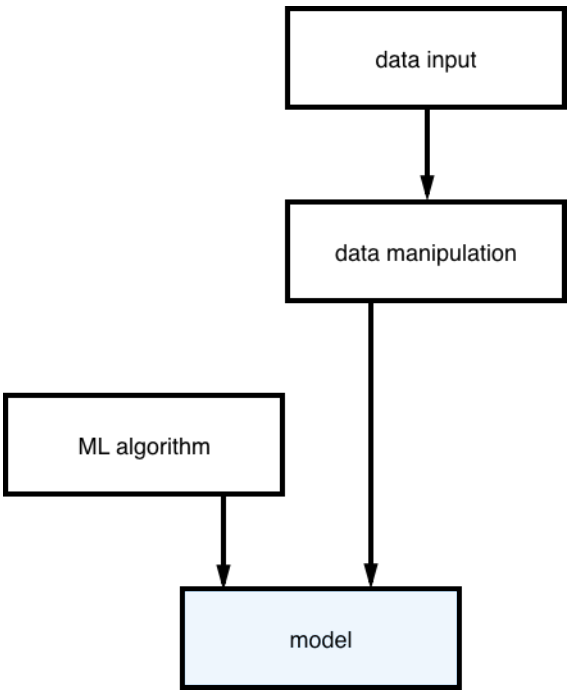
www.datascienceacademy.com.br

Deep Learning Frameworks

Como Ocorre o Processo de Aprendizagem de Uma Rede Neural?

Modelo Preditivo

A criação de um modelo preditivo, envolve basicamente dois componentes: algoritmo e dados.



Como você pode ver, o modelo consiste em um algoritmo de aprendizado de máquina "treinado" com dados. Quando você tem o modelo você obterá resultados, aplicando o modelo a novos conjuntos de dados.

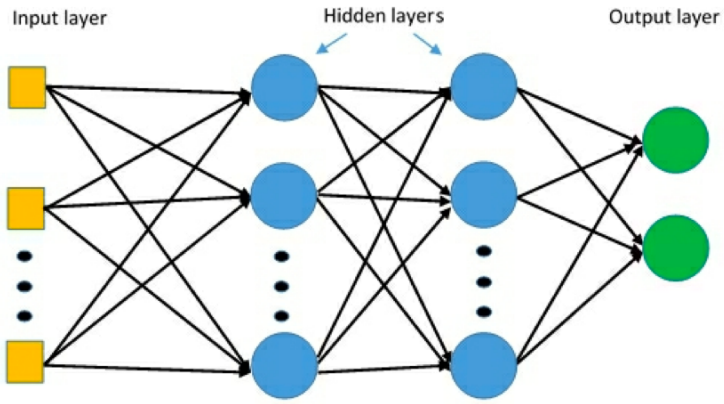
O objetivo do modelo que você criará é classificar as entradas em categorias, definimos que:

Entrada: variáveis preditores ou atributos
Resultado: categoria (variável target)

Temos um conjunto de dados de treinamento com todos os registros rotulados, indicando a qual categoria pertence. Na aprendizagem de máquina este tipo de tarefa é denominado aprendizagem supervisionada.

Redes Neurais

Uma rede neural é um modelo computacional (uma forma de descrever um sistema usando linguagem matemática e conceitos matemáticos). Esses sistemas são *self-learning* e treinados, em vez de explicitamente programados. As redes neurais são inspiradas pelo nosso sistema nervoso central. Eles têm nós conectados que são semelhantes aos nossos neurônios.



O Perceptron foi o primeiro algoritmo de rede neural. Este artigo: https://appliedgo.net/perceptron explica muito bem o funcionamento interno de um perceptron (a animação " Inside an artificial neuron" é incrível. Clique no botão Start).

Para entender como funciona uma rede neural vamos construir a arquitetura da rede neural com TensorFlow. A rede neural terá duas camadas ocultas (você tem que escolher quantas camadas ocultas a rede terá, faz parte do design da arquitetura). O trabalho de cada camada oculta é transformar as entradas em algo que a camada de saída pode usar.

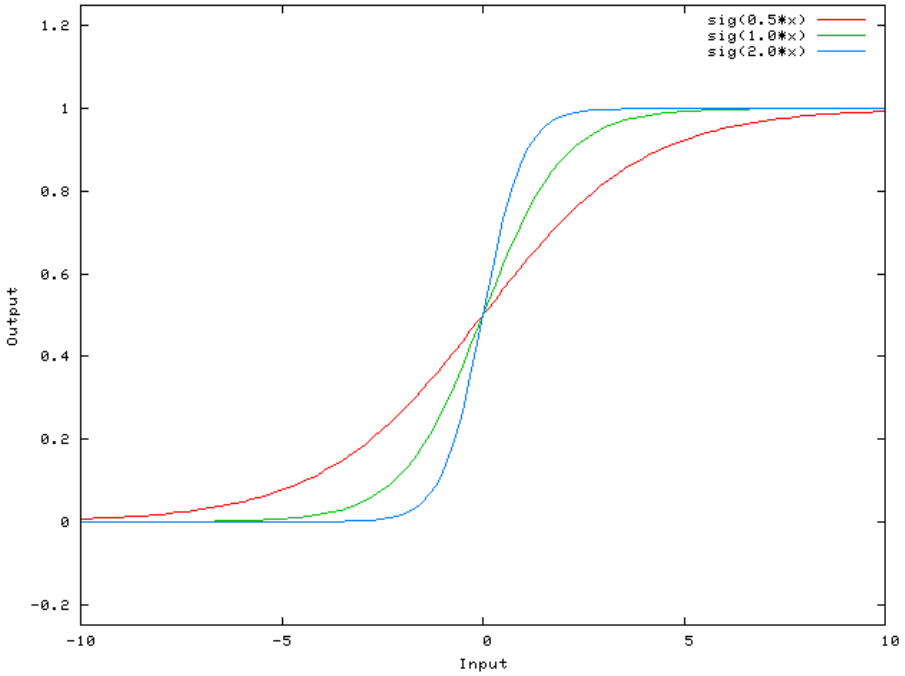
Você também precisa definir quantos nós a primeira camada oculta terá. Esses nós também são chamados de recursos ou neurônios, e na imagem acima eles são representados por cada círculo.

Na camada de entrada cada nó corresponde a uma palavra do conjunto de dados (veremos como isso funciona mais tarde).

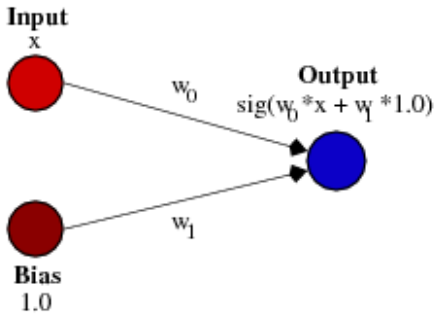
Como explicado aqui, cada nó (neurônio) é multiplicado por um peso. Cada nó tem um valor de peso e durante a fase de treinamento a rede neural ajusta esses valores para produzir uma saída correta. Além de multiplicar cada nó de entrada por um peso, a rede também adiciona um viés (bias). Em efeito, um valor de bias permite que você *desloque* a função de

ativação para a esquerda ou para a direita, o que pode ser crítico para uma aprendizagem bem-sucedida.

A saída da rede é calculada multiplicando a entrada (x) pelo peso (w0) e passando o resultado através de algum tipo de função de ativação (por exemplo, uma função sigmóide). Aqui está a função que a rede calcula, para vários valores de w0:



Mudando o peso w0 essencialmente muda a "inclinação" do sigmóide. Isso é útil, mas o que você quer é que a rede para saída 0 quando x é 2? Apenas mudar a inclinação do sigmóide não vai realmente funcionar - você quer ser capaz de mudar a curva inteira para a direita. Isso é exatamente o que o viés permite que você faça.



Uma maneira mais simples de entender o que é o viés: ele é de alguma forma semelhante à constante b de uma função linear Y = ax + b

Ele permite que você mova a linha para cima e para baixo para ajustar a previsão com os dados. Sem b a linha sempre passa pela origem (0, 0) e seu ajuste seria mais pobre e menos eficiente.
Em sua arquitetura depois de multiplicar as entradas pelos pesos e somar os valores para o viés, os dados também passam por uma função de ativação. Esta função de ativação define a saída final de cada nó. Uma analogia: imagine que cada nó é uma lâmpada, a função de ativação diz se a lâmpada acenderá ou não.

Existem muitos tipos de funções de ativação. Veja nesse link a lista de funções e seus comparativos: https://en.wikipedia.org/wiki/Activation_function

Neste exemplo, usaremos a ReLu (Rectified Linear Unit).

A segunda camada oculta faz exatamente o que a 1ª camada oculta faz, mas agora a entrada da segunda camada oculta é a saída da primeira.

E finalmente chegamos à última camada, a camada de saída (output_layer). Você usará a codificação one-hot para obter os resultados dessa camada. Nesta codificação apenas um bit tem o valor 1 e todos os outros têm um valor zero. Por exemplo, se quisermos codificar três categorias (esportes, espaço e computação gráfica):

Categoria	Valor
Negócios	001
Esportes	010
Lazer	100

Portanto, o número de nós de saída é o número de classes do conjunto de dados de entrada. Os valores da camada de saída também são multiplicados pelos pesos e também adicionamos o viés, mas agora a função de ativação é diferente.

Você quer rotular cada texto com uma categoria, e essas categorias são mutuamente exclusivas (um texto não pertence a duas categorias ao mesmo tempo). Para considerar isso, em vez de usar a função de ativação ReLu, você usará a função Softmax. Essa função transforma a saída de cada unidade em um valor entre 0 e 1 (uma distribuição de probabilidade) e também garante que a soma de todas as unidades seja igual a 1. Assim, a saída nos dirá a probabilidade de cada texto para cada categoria.

Como Ocorre a Aprendizagem da Rede Neural

Como vimos anteriormente, os valores de peso são atualizados enquanto a rede é treinada. Agora vamos ver como isso acontece no ambiente TensorFlow.

tf.Variable

Os pesos e vieses (bias) são armazenados em variáveis (tf.Variable). Essas variáveis mantêm o estado no grafo entre as chamadas a executar. Na aprendizagem de máquina nós começamos geralmente os valores do peso e do bias por uma distribuição normal.

Quando executamos a rede pela primeira vez (isto é, os valores de peso são os definidos pela distribuição normal), temos:

- input values: x
- weights: w
- bias: b
- output values: z
- expected values: y

Para saber se a rede está aprendendo ou não, você precisa comparar os valores de saída (z) com os valores esperados (y). E como calculamos essa diferença (perda)? Existem muitos métodos para fazer isso. Como estamos trabalhando com uma tarefa de classificação, a melhor medida para a perda é o erro de entropia cruzada.

Quando se utiliza uma rede neural para realizar a classificação e a previsão, é geralmente melhor usar o erro de entropia cruzada do que o erro de classificação e um pouco melhor usar o erro de entropia cruzada do que o erro quadrático médio para avaliar a qualidade da rede neural. A ideia básica é simples, mas há um monte de questões relacionadas que confundem muito a ideia principal. Primeiro, é importante deixar claro que estamos lidando apenas com uma rede neural que é usada para classificar dados, como prever a filiação de um partido político (democrata, republicano, outro) de dados independentes, como idade, sexo, renda anual, e assim por diante. Não estamos lidando com uma rede neural que faça regressão, onde o valor a ser previsto seja numérico, ou uma rede neural de séries temporais, ou qualquer outro tipo de rede neural.

Agora suponha que você tenha apenas três itens de dados de treinamento. Sua rede neural usa ativação de softmax para os neurônios de saída, de modo que existem três valores de saída que podem ser interpretados como probabilidades. Por exemplo, suponha que as saídas computadas da rede neural e os valores de destino (desejado) são os seguintes:

```
computed        | targets              | correct?
------------------------------------------------
0.3  0.3  0.4   | 0  0  1 (democrat)   | yes
0.3  0.4  0.3   | 0  1  0 (republican) | yes
0.1  0.2  0.7   | 1  0  0 (other)      | no
```

Esta rede neural tem erro de classificação de 1/3 = 0,33, ou, de forma equivalente, uma precisão de classificação de 2/3 = 0,67. Observe que a rede neural apenas obtém os dois primeiros itens de treinamento corretos e está longe no terceiro item de treinamento. Mas agora considere a seguinte rede neural:

```
computed        | targets              | correct?
------------------------------------------------
0.1  0.2  0.7   | 0  0  1 (democrat)   | yes
0.1  0.7  0.2   | 0  1  0 (republican) | yes
0.3  0.4  0.3   | 1  0  0 (other)      | no
```

Esta rede neural também tem um erro de classificação de 1/3 = 0,33. Mas esta rede neural é melhor do que a primeiro, porque acerta os dois primeiros itens de treinamento e erra por pouco o terceiro item de treinamento. Para resumir, erro de classificação é uma medida muito grosseira de erro.

Agora considere o erro de entropia cruzada. O erro de entropia cruzada para o primeiro item de treinamento na primeira rede neural acima é:

```
-( (ln(0.3)*0) + (ln(0.3)*0) + (ln(0.4)*1) ) = -ln(0.4)
```

Observe que no caso da classificação de redes neurais, a computação é um pouco estranha porque todos os termos, exceto um, desaparecerão. Assim, o erro médio de entropia cruzada (ACE – Average Cross Entropy) para a primeira rede neural é calculado como:

```
-(ln(0.4) + ln(0.4) + ln(0.1)) / 3 = 1.38
```

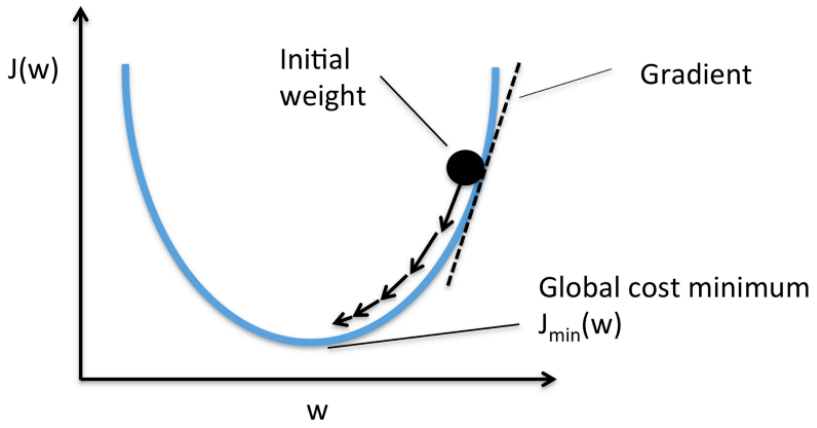
O erro médio de entropia cruzada para a segunda rede neural é:

```
-(ln(0.7) + ln(0.7) + ln(0.3)) / 3 = 0.64
```

Observe que o erro médio de entropia cruzada para a segunda rede neural descrita acima é menor do que o erro ACE para a primeira rede neural. A função ln() na entropia cruzada leva em conta a proximidade de uma previsão e é uma forma mais granular para calcular o erro.

Com TensorFlow você calculará o erro de entropia cruzada usando o método tf.nn.softmax_cross_entropy_with_logits() (aqui a função de ativação é a softmax) e calculará o erro médio (tf.reduced_mean()).

Você deseja encontrar os melhores valores para os pesos e bias, a fim de minimizar o erro de saída (a diferença entre o valor que obtivemos e o valor correto). Para fazer isso, você usará o método de descida do gradiente.



Existem também muitos algoritmos para calcular a descida do gradiente, e podemos usar o Adaptive Moment Estimation (Adam). Para usar este algoritmo no TensorFlow você precisa passar o valor learning_rate, que determina os passos incrementais dos valores para encontrar os melhores valores de peso.

Treinando o Modelo

Na terminologia da rede neural, um epoch = uma passagem para a frente (recebendo os valores de saída) e uma passagem para trás (atualizando os pesos) de todos os exemplos de treinamento.

Nesta rede neural calculamos duas coisas: o cálculo de perda e o passo de otimização

Agora você tem o modelo, treinado. Para testá-lo, você também precisará criar elementos de grafo. Vamos medir a precisão do modelo, então você precisa obter o índice do valor previsto e o índice do valor correto (pois estamos usando one-hot encoding. Mais sobre isso no próximo capítulo), verificar se eles são iguais e calcular a média para todos os dados de teste.

Obrigado

Equipe DSA

Referências:

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. Neural Networks : The Official Journal of the International Neural Network Society, 12(1), 145–151. http://doi.org/10.1016/S0893-6080(98)00116-6

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research, 12, 2121–2159. Retrieved from http://jmlr.org/papers/v12/duchi11a.html

Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V, … Ng, A. Y. (2012). Large Scale Distributed Deep Networks. NIPS 2012: Neural Information Processing Systems, 1–11. http://doi.org/10.1109/ICDAR.2011.95