TENSORFLOW 2.0: APRENDIZAGEM PROFUNDA E INTELIGÊNCIA ARTIFICIAL

ESCRITO POR: NAILSON CHAGAS

Index

Sobre essas anotações

Minhas anotações do curso: https://udemy.com/course/deep-learning-tensorflow-2

Exemplos práticos: https://github.com/NailsonChagas/Tensorflow-2.0-Deep-Learning-and-Artificial-Intelligence

SEÇÃO 1

O que é Aprendizado de Máquina?

Aprendizagem de máquina é um subcampo da Engenharia e da ciência da computação que evoluiu do estudo de reconhecimento de padrões e da teoria do aprendizado computacional em inteligência artificial. Em 1959, Arthur Samuel definiu aprendizado de máquina como o "campo de estudo que dá aos computadores a habilidade de aprender sem serem explicitamente programados"

1.1 Aprendizado de Máquina é apenas um problema geométrico

- Um estatístico diria: Aprendizado de Máquina é apenas um ajuste de curva glorificado.
- Tanto regressão quanto classificação são exemplos de aprendizado supervisionado .

2

SEÇÃO

Teoria de Classificação Linear

Obs: Para facilitar o exemplo lida só com calssificação linear binária.

A linha que separa um conjunto de pontos em um plano cartesiano pode ser escrita através da equação: (1) $y = m * x + b$

Também podendo ser escrita como sendo: (2) $w_1 * x_1 + w_2 * x_2 + b = 0$, com x_1 sendo o eixo horizontal e x_2 sendo o eixo vertical.
Como podemos usar essa linha para classificar pontos?

$$(3)\ a = w_1 * x_1 + w_2 * x_2 + b$$

Usando a equação (3), podemos tomar uma decisão:

- Se $a \geq 0 \implies$ Classe 1
- Se $a < 0 \implies$ Classe 0

Matemáticamente podemos encapsular esse processo de decisão em uma função de Heaviside (função degrau: assume valor 0 ou 1):

$$(4)\ \hat{y} = u(a), a = w_1 * x_1 + w_2 * x_2 + b$$

Como em Deep Learn nós preferimos utilizar funções mais suaves e diferenciaveis, é utilizada uma função Sigmoide (função logística: assume valores entre 0 e 1):

$$(5)\ \hat{y} = \sigma(a), a = w_1 * x_1 + w_2 * x_2 + b$$

Com $\hat{y}$ da equação (5) podendo ser interpretado como a probabilidade de que:

$$p(y = 1|x) = \sigma(w_1 * x_1 + w_2 * x_2 + b),\ ou\ seja, y = 1\ dado\ x$$

Com o resultado da probabilidade podemos:

- Se $p(y = 1|x) \geq 0 \implies$ prever 1, se não 0

Quando aplicamos uma função Sigmoide em cima de uma função linear, nós chamamos o modelo de Regressão Logística, e o argumento de uma função Sigmoide é chamado de ativação.

Usando o que sabemos até agora, pode-se notar um problema de notação: e se tivermos mais de duas entradas $(w_1 * x_1, w_2 * x_2, ..., w_n * x_n)$? R: Sem problemas, considerando w como um vetor de pesos e x como um vetor de caracteristicas :

$$(6)\ p(y = 1|x) = \sigma(w^T * x + b) = \sigma(\sum_{d=1}^{D} w_d * x_d + b)$$

Mas como podemos implementar, o que vimos até agora no Tensorflow? A expressão $w^T * x + b$ é implementada através da função:

```
tf.keras.layers.Dense(output_size)
```

Para podermos implementar o modelo de Regressão Logística utilizaremos dois layers:

```
"""
 Input(shape=(D,)), basicamente falando para o keras o tamanho do vetor de entrada x
 Dense(1, activation='sigmoid') especifica o tamanho de saida e função de ativação
"""
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(D,)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

#configura o modelo para treino
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

r = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=100 #numero de iterações para calculo dos pesos (vetor w)
)

"""
 Visualiar a função de perda e outras métricas ao passar das iterações
 Com isso podemos avaliar se precisamos de mais epochs
 ou alterar outros hiper parametros
"""
plt.plot(r.history['loss'], label='loss')
plt.plot(r.history['val_loss'], label='val_loss')
```

Para olhar o código funcionado, usar o script classification na pasta ML and Neurons.

3

SEÇÃO

Teoria de Regressão Linear

Obs: Para facilitar o exemplo lida só com regressão linear.

Em estatística regressão linear é uma equação para se estimar a condicional (valor esperado) de uma variável y, dados os valores de algumas outras variáveis x. A regressão, em geral, tem como objetivo tratar de um valor que não se consegue estimar inicialmente. A regressão linear é chamada "linear" porque se considera que a relação da resposta às variáveis é uma função linear de alguns parâmetros.

- Equação da reta: $\hat{y} = \sum_{d=1}^{D} w_d * x_d + b = w^T * x + b$

Diferente da classificação, a regressão, devido a sua saída poder assumir "qualquer" valor, não necessita de uma função de ativação (ex: sigmoide)

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Input(shape=(D,)),
    tf.keras.layers.Dense(1, activation=None)
])

model.compile(
    #escolhido no lugar do adam pois esse performou melhor nesse caso
    optimizer = tf.keras.optimizers.SGD(0.001, 0.9),
    #como a saida não é binária (mse = mean square erro)
    loss = 'mse'
)
```

Em regressão o uso da acurácia não faz sentido pois ela quase sempre seria 0. No caso da regressão faz mais sentido utilizar a medida estatística R^2 (coeficiente de determinação)

4

SEÇÃO

O Neurônio

Chamamos regressão logistica de neurônio, com um neurônio sendo a base fundamental de uma rede neural.

$$\hat{y} = \sigma(\sum_{d=1}^{D} w_d * x_d + b)$$

Com cada x_d sendo uma característica, é possível afirmar que seu peso w_d diz o quão importante essa característica é para predizer a saída.

4.1 Como isso é relacionado com um neurônio?

Os neurônios têm um papel essencial na determinação do funcionamento e comportamento do corpo humano e do raciocínio. Eles são formados pelos dendritos, que são um conjunto de terminais de entrada, pelo corpo central, e pelos axônios que são longos terminais de saída. Neurônios se comunicam através de sinapses. Sinapse é a região onde dois neurônios entram em contato e através da qual os impulsos nervosos são transmitidos entre eles. Os impulsos recebidos por um neurônio A, em um determinado momento, são processados, e atingindo um dado limiar de ação, o neurônio A dispara, produzindo uma substância neuro transmissora que flui do corpo celular para o axônio, que pode estar conectado a um dendrito de um outro neurônio B. O neuro transmissor pode diminuir ou aumentar a polaridade da membrana pós-sináptica, inibindo ou excitando a geração dos pulsos no neurônio B. Este processo depende de vários fatores, como a geometria da sinapse e o tipo de neuro transmissor. Tendo essas inforamações, é possível fazer algumas suposições básicas:

- A atividade de um neurônio é um processo tudo ou nada (binário).

- Um certo número fixo (>1) de entradas devem ser excitadas dentro de um período de adição latente para excitar um neurônio.
- Único atraso significativo é o atraso sináptico.
- A atividade de qualquer sinapse inibitória previne absolutamente a excitação do neurônio.
- A estrutura das interconexões não muda com o tempo.

Através dessas suposições, pode-se representar um neoronio artificial através do algoritmo de Regressão Logistica:

$$\hat{y} = \sigma(\sum_{d=1}^{D} w_d * x_d + b)$$

Com os n elementos do vetor x sendo as entradas (dendritos) do neuronio, $\hat{y}$ sendo o terminal de saída (axônio), os elementos do vetor w descrevendo o comportamentos das sinapses, $w_d * x_d$ o efeito da sinapse e b representando o bias (Tendência).

4.2 Rede neural artificial

Uma rede neural artificial é composta por várias unidades de processamento (neurônios), cujo funcionamento é bastante simples. Essas unidades, geralmente são conectadas por canais de comunicação que estão associados a determinado peso. As unidades fazem operações apenas sobre seus dados locais, que são entradas recebidas pelas suas conexões. O comportamento inteligente de uma Rede Neural Artificial vem das interações entre as unidades de processamento da rede. A maioria dos modelos de redes neurais possui alguma regra de treinamento, onde os pesos de suas conexões são ajustados de acordo com os padrões apresentados. Em outras palavras, elas aprendem através de exemplos. Arquiteturas neurais são tipicamente organizadas em camadas, com unidades que podem estar conectadas às unidades da camada posterior. Usualmente as camadas são classificadas em três grupos:

- Camada de Entrada: onde os padrões são apresentados à rede.
- Camadas Intermediárias ou Escondidas: Onde é feita a maior parte do processamento, através das conexões ponderadas; podem ser consideradas como extratoras de características.
- Camada de Saída: onde o resultado final é concluído e apresentado.

4.3 Como um modelo aprende?

Uma rede neural aprende calculando os pesos de cada característica, em cada neurônio. O Tensorflow ira fazer todas as diferenciações para calcular os pesos com menores custos, sendo preciso selecionar apenas a taxa de aprendizado. Obs: Para escolher a taxa de aprendizado, normalmente, tentar potencias de 10 (0.1, 0.01, 0.001, ...)

5

SEÇÃO

Feedforward Neural Network

Feedforward Neural Network (FFNN) ou Rede Neural de Alimentação Direta, é o tipo mais básico de rede neural e forma a base de outros tipos de redes neurais como:

1. Convolutional Neural Network (CNN)
2. Recurrent Neural Network (RNN)

Em uma FFNN o sinal apenas vai da esquerda para a direita, ou seja, os neurônios da camada anterior sempre passam as informações para os neurônios da próxima camada sem que esses tenham como enviar sinais à camada anterior.

5.1 Arquitetura do Modelo

A rede neural é dividida em n camadas, com cada camada possuindo n neurônios, com todos os neurônios possuindo estrutura uniforme (ou seja, todos neurônios são iguais em todas as camadas). A mesma entrada pode ser usada em múltiplos neurônios de uma mesma camada, com cada um calculado algo diferente. Pode se considerar que uma rede neural possui duas dimensões, sua largura (quantos neurônios há por camada) e profundidade (quantas camadas existem na rede). Chamando a saída de um neurônio de z_j em uma camada com M neurônios, pode-se representar uma camada de uma rede neural através da expressão:

$$z_j = \sigma(w_j^T * x + b_j), para\, j = 1\ até\ M$$

Ou vetorizando:

$$z = \sigma(W^T * x + b)$$

Onde z é um vetor de tamanho M representando as saídas de uma camada, x é um vetor de tamanho D representando as entradas dos neurônios na camada, W é uma matriz de tamanho DxM reapresentando os pesos de cada entrada em cada neurônio e b um vetor de tamanho M representando o bias. Tendo a representação vetorizada, cada as camadas de uma rede com L camadas, podem ser escritas como:

$$z^{(1)} = \sigma(W^{(1)T} * x + b^{(1)})$$

$$z^{(2)} = \sigma(W^{(2)T} * z^{(1)} + b^{(2)})$$

$$z^{(3)} = \sigma(W^{(2)T} * z^{(2)} + b^{(3)})$$

...

$$p(y = 1|x) = \sigma(W^{(L)T} * z^{(L-1)} + b^{(L)})$$

Para uma rede neural usada para regressão, a única coisa que mudaria é que na última camada não haverá a função sigmóide:

$$\hat{y} = W^{(L)T} * z^{(L-1)} + b^{(L)}$$

5.2 A Geometria

Por que usar redes neurais ao invés de usar Regressão Logística e Linear? Nem todos os problemas do mundo podem ser representados bidimensionalmente e/ou linearmente, as vezes algo que queremos pode ser apenas representado através de curvas extremamente complexas.

5.3 Funções de Ativação

5.3.1 Sigmóide

$$\sigma(a) = \frac{1}{1 + exp(-a)}$$

- Mapeia a saída para valores entre 0 e 1 → Imita um neurônio biológico
- Faz com que a rede neural não seja linear
- Problema: A sigmóide foi usada em todos os exemplos até aqui, porém como é preferido que as entradas dos neurônios sejam centradas ao redor de 0 e tenham aproximadamente a mesma distância do centro (exemplo de como não deve ser: uma entrada tem valor de 1 até 5 milhões e outra de 0 até 0.0000001), a sigmóide não é mais tão utilizada (seu centro é 0.5)

5.3.2 Tangente Hiperbólica

$$\tanh(a) = \frac{exp(2a) - 1}{exp(2a) + 1}$$

- Mapeia a saída para valores entre -1 e 1
- Possui o mesmo formato de uma sigmóide, porém seu centro é em 0
- Problema de padronização: Apesar de ser um pouco melhor que a sigmóide, ela ainda é problematica (ver The Vanishing Gradient Problem, sigmóide tem o mesmo problema).

5.3.3 ReLU

- Escolha padrão
- Não possui o problema apresentado por The Vanishing Gradient Problem

5.3.4 ELU

$$f(x) = \begin{cases} x \; se \; x \geq 1 \\ \alpha(exp(x) - 1) \end{cases}$$

Autores dizem que acelera o aprendizado e leva a melhor precisão

5.3.5 LReLU

5.3.6 Softplus

$$f(x) = log(1 + e^x)$$

5.4 Classificação Multiclasse

Para classificação binária é utilizado ReLUs nas camadas intermediárias (hidden layers) e a sigmóide na saída. Para a classificação de K classes é utilizada a função Softmax na saída.

```
tf.keras.layers.Dense(K, activation='softmax')
```

5.5 O que Redes Neurais podem fazer?

Tarefa	Função de ativação comumente usada na camada de saída
Regressão	Nenhuma
Classificação Binária	Sigmóide
Classificação Multiclasse	Softmax

5.6 Imagens