TP555 - AI/ML

Lista de Exercícios #12

Redes Neurais Artificiais (Parte 2)

1. Por que a função de ativação logística foi um ingrediente chave no treinamento das primeiras redes MLPs?

2. Utilizando o exemplo activation_functions.ipynb como base, cite três funções de ativação diferentes das que vimos. Plote a função e sua respectiva derivada.

3. Desenhe uma rede MLP que calcule $A \oplus B$, onde $\oplus$ representa a operação lógica XOR. (**Dica**: $A \oplus B = (A\&!B) \mid (!A\&B)$)

4. Suponha que você tenha uma rede MLP composta por uma camada de entrada com 10 neurônios, seguida por uma camada oculta com 50 neurônios e, finalmente, uma camada de saída com 3 neurônios. Todos os neurônios usam a função de ativação ReLU. Sendo assim, responda:

   a. Qual é a dimensão da matriz de entrada X?

   b. Qual a dimensão do vetor de pesos, W_h, da camada oculta e de seu vetor de bias b_h?

   c. Qual é a dimensão do vetor de pesos, W_o, da camada de saída e de seu vetor de bias b_o?

   d. Qual é a dimensão da matriz de saída, Y, da rede?

   e. Escreva a equação que calcula a matriz de saída da rede, Y, como uma função de X, W_h, b_h, W_o, e b_o.

5. Quantos neurônios você precisa na camada de saída de uma rede MLP se você deseja classificar emails em spam ou ham? Qual função de ativação você deve usar na camada de saída? Se você deseja classificar a base de dados MNIST (imagens de dígitos escritos à mão), quantos neurônios você precisa na camada de saída usando qual tipo de função de ativação?

6. Liste todos os hiperparâmetros que você pode ajustar em uma rede MLP? Caso você perceba que a rede MLP está sobreajustando, como você pode modificar esses hiperparâmetros para tentar resolver o problema?

7. Baseando-se no exemplo `MLPWithTensorFlowLowLevelAPI.ipynb`, utilize objetos da classe `Dense()` do módulo `tf.keras.layers` ao invés da função `neuron_layer()`. Após treinar o modelo, você percebeu alguma diferença na performance do seu modelo? Se sim, qual foi esta diferença?

   (**Dica**: A documentação da classe `Dense` pode ser encontrada no seguinte link: `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense`)

   (**Dica**: Lembre que `Dense` é uma classe que precisa ser instanciada. Apenas após a criação do objeto da classe `Dense` é que se deve passar a entrada para esta camada de nós.)

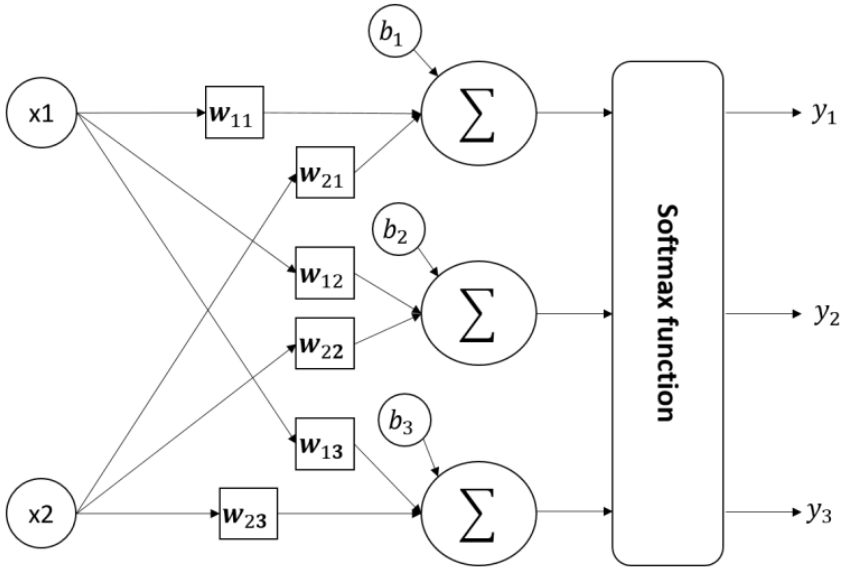
8. Baseando-se no código do exercício anterior, implemente a técnica do *early stopping* de forma que o modelo sempre armazene os pesos que resultem no menor erro. Além disso, crie um contador que conte o número de épocas sem progresso, ou seja, o número de vezes em que o erro da época não diminuiu com relação ao menor erro obtido até o momento. Com base nesse contador, faça com que o treinamento se encerre caso o contador atinja 50 épocas sem progresso.

   (**Dica**: Uma maneira de se evitar que um modelo sobreajuste é interromper o treinamento assim que o erro de validação/teste atinja o mínimo. Outra maneira válida é armazenar os pesos do modelo que resultem no menor erro de validação durante o treinamento através de um pré-determinado número de épocas. Essa abordagem é chamada de *early stopping*.)

9. Baseando-se no código do exercício anterior, utilize o otimizador que implementa o algoritmo *momentum* ao invés do gradiente descendente e treine uma rede MLP que obtenha uma precisão maior do que 98%. Use o trecho de código abaixo.

```
optimizer = tf.train.MomentumOptimizer(learning_rate=0.01,
   momentum=0.9)
```

10. Utilizando o TensorFlow, implemente o modelo mostrado na figura abaixo, para classificar 3 classes diferentes.



Se nós modelarmos o classificador da figura em formato matricial, então, nós temos a seguinte equação:

$$y = f(X \cdot W + b),$$

onde:

- X é a matriz de entrada com dimensão $N \times 2$.
- W é a matriz de coeficientes com dimensão 2×3.
- b é o vetor de *bias* para a primeira camada, com dimensão 3.

- y é o vetor de saída do classificador.
- $f(.)$ é a função *softmax*.
- N é o número de exemplos.

Crie a base de dados das 3 classes com o código abaixo:

```
from sklearn.datasets import make_blobs
# Create a 3-class dataset for classification
N = 1000
centers = [[-5, 0], [0, 1.5], [5, -1]]
X_, y_ = make_blobs(n_samples=N, centers=centers, random_state=42)
```

Em seguida, faça o seguinte:

a. Plote um gráfico mostrando as diferentes classes. (**Dica**: use cores ou marcadores diferentes para cada classe.)

b. Crie um grafo utilizando o `GradientDescentOptimizer` para classificar os dados. (**Dica**: verifique a documentação deste otimizador no seguinte link: GradientDescentOptimizer)

c. Imprima e salve o erro e a precisão a cada 100 épocas.

d. Treine o modelo. (**Dica**: Use 20000 épocas ou treine até que a precisão seja de 100%.)

e. Plote um gráfico mostrando o erro e a precisão versus o número de épocas.

f. Plote um gráfico com as fronteiras de decisão.

g. Crie um segundo grafo utilizando o `AdamOptimizer` para classificar os dados. (**Dica**: Não se esqueça de resetar o grafo com `tf.reset_default_graph()`.) (**Dica:** verifique a documentação deste otimizador no seguinte link: https://www.tensorflow.org/api_docs/python/tf/compat/v1/train/AdamOptimizer)

h. Imprima e salve o erro e a precisão a cada 100 épocas.

i. Treine o modelo. (**Dica**: Use 20000 épocas ou treine até que a precisão seja de 100%.)

j. Plote um gráfico mostrando o erro e a precisão versus o número de épocas.

k. Plote um gráfico com as fronteiras de decisão.

l. Baseado nos valores de erro e precisão impressos e plotados nas figuras anteriores, qual dos 2 otimizadores tem melhor performance? (**Dica**: Qual dos 2 converge mais rapidamente?)

Observação:Use a função `tf.nn.sparse_softmax_cross_entropy_with_logits()` para calcular o erro do modelo e assim conseguir treiná-lo, ou seja, encontrar os pesos que minimizem o erro. A função `tf.nn.sparse_softmax_cross_entropy_with_logits()` é equivalente a aplicar a função de ativação *softmax* e, em seguida, calcular a entropia cruzada, mas é mais eficiente e cuida adequadamente de casos incomuns, por exemplo, onde os *logits* são iguais a 0. A documentação dessa função pode ser encontrada em: https://www.tensorflow.org/api_docs/python/tf/nn/sparse_softmax_cross_entropy_with_logits.

Referências:
[1] 'Active Learning on MNIST — Saving on Labeling', https://towardsdatascience.com/active-learning-on-mnist-saving-on-labeling-f3971994c7ba
[2] 'Computing Costs on a Softmax Output Layer' https://riptutorial.com/tensorflow/example/17628/computing-costs-on-a-softmax-output-layer