FUNDAMENTALS OF MACHINE LEARNING IN DATA SCIENCE

CSIS 3290

KERAS AND TENSORFLOW

FATEMEH AHMADI

Neural Network with Keras and Tensorflow

```
In [5]: from numpy import loadtxt
        import pandas as pd
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense

In [15]: dataset = loadtxt('C:/Users/Paris/Desktop/pimaDiabetes.csv', delimiter=',')
         x=dataset[:,0:8]
         y=dataset[:,8]
```

We can split the array into two arrays by selecting subsets of columns using the standard NumPy slice operator or ":". You can select the first eight columns from index 0 to index 7 via the slice 0:8. We can then select the output column (the 9th variable) via index 8.

Sequential and Dense

We create a *Sequential model* and add layers one at a time until we are happy with our network architecture.

The first thing to get right is to ensure the input layer has the correct number of input features. This can be specified when creating the first layer with the **input_shape** argument and setting it to `(8,)` for presenting the eight input variables as a vector.

Fully connected layers are defined using the Dense class. You can specify the number of neurons or nodes in the layer as the first argument and the activation function using the **activation** argument.

Also, you will use the rectified linear unit activation function referred to as ReLU on the first two layers and the Sigmoid function in the output layer.

Neural Network with Keras and Tensorflow

```
In [17]: # define the keras model
model1 = Sequential()
model1.add(Dense(12, input_shape=(8,), activation='relu'))
model1.add(Dense(8, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))
```

```
In [18]: # compile the keras model
model1.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
In [20]: # fit the keras model on the dataset
model1.fit(x, y, epochs=150, batch_size=10)
```

Training occurs over epochs, and each epoch is split into batches.

- **Epoch**: One pass through all of the rows in the training dataset
- **Batch**: One or more samples considered by the model within an epoch before weights are updated

Loss Functions for Neural Networks

1. Binary Cross Entropy
2. Categorical Cross Entropy
3. Hinge Loss
4. Mean Square Error (MSE) / Quadratic Loss / L2 Loss
5. Mean Absolute Error / L1 Loss
6. Huber Loss / Smooth Mean Absolute Error
7. Log-Cosh Loss
8. Quantile Loss

Optimizers for Neural Networks

- ✓ Gradient Descent (GD)
- ✓ Stochastic Gradient Descent (SGD)
- ✓ Mini-Batch Gradient Descent (MBGD)
- ✓ Momentum
- ✓ Adagrad (Adaptive Gradient Descent)
- ✓ RMS Prop
- ✓ ADAM

https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0

Optimizers for Neural Networks

Optimizers are algorithms or methods used to minimize an error function(*loss function*) or to maximize the efficiency of production. Optimizers are mathematical functions which are dependent on model's learnable parameters i.e Weights & Biases. Optimizers help to know how to change weights and learning rate of neural network to reduce the losses.

Gradient Descent (GD):

$$W_{new} = W_{old} - \alpha * \frac{\partial(Loss)}{\partial(W_{old})}$$

https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0

Neural Network with Keras and Tensorflow

```
Epoch 1/150
77/77 [==============================] - 1s 987us/step - loss: 23.6431 - accuracy: 0.6510
Epoch 2/150
77/77 [==============================] - 0s 974us/step - loss: 12.0750 - accuracy: 0.6510
Epoch 3/150
77/77 [==============================] - 0s 1ms/step - loss: 5.4129 - accuracy: 0.5964
Epoch 4/150
77/77 [==============================] - 0s 1ms/step - loss: 2.2949 - accuracy: 0.4193
Epoch 5/150
77/77 [==============================] - 0s 1ms/step - loss: 1.5959 - accuracy: 0.4193
Epoch 6/150
77/77 [==============================] - 0s 1000us/step - loss: 1.3321 - accuracy: 0.4102
Epoch 7/150
77/77 [==============================] - 0s 1ms/step - loss: 1.0848 - accuracy: 0.4479
Epoch 8/150
77/77 [==============================] - 0s 974us/step - loss: 0.9425 - accuracy: 0.4557
Epoch 9/150
77/77 [==============================] - 0s 947us/step - loss: 0.8536 - accuracy: 0.5260
Epoch 10/150
```

Neural Network with Keras and Tensorflow

```
Epoch 142/150
77/77 [==============================] - 0s 1ms/step - loss: 0.5045 - accuracy: 0.7617
Epoch 143/150
77/77 [==============================] - 0s 1ms/step - loss: 0.5040 - accuracy: 0.7539
Epoch 144/150
77/77 [==============================] - 0s 1ms/step - loss: 0.5119 - accuracy: 0.7500
Epoch 145/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4983 - accuracy: 0.7630
Epoch 146/150
77/77 [==============================] - 0s 1ms/step - loss: 0.5098 - accuracy: 0.7487
Epoch 147/150
77/77 [==============================] - 0s 1ms/step - loss: 0.5038 - accuracy: 0.7591
Epoch 148/150
77/77 [==============================] - 0s 1ms/step - loss: 0.5068 - accuracy: 0.7565
Epoch 149/150
77/77 [==============================] - 0s 1ms/step - loss: 0.5123 - accuracy: 0.7396
Epoch 150/150
77/77 [==============================] - 0s 1ms/step - loss: 0.4968 - accuracy: 0.7578

Out[20]: <keras.callbacks.History at 0x2d6cc1b4fd0>
```

Neural Network with Keras and Tensorflow

Ideally, you would like the loss to go to zero and the accuracy to go to 1.0 (e.g., 100%). This is not possible for any but the most trivial machine learning problems. Instead, you will always have some error in your model. The goal is to choose a model configuration and training configuration that achieve the lowest loss and highest accuracy possible for a given dataset.

The **evaluate()** function will return a list with two values. The first will be the loss of the model on the dataset, and the second will be the accuracy of the model on the dataset. You are only interested in reporting the accuracy so ignore the loss value.

Making predictions is as easy as calling the **predict()** function on the model. You are using a sigmoid activation function on the output layer, so the predictions will be a probability in the range between 0 and 1. You can easily convert them into a crisp binary prediction for this classification task by rounding them.

Alternately, you can convert the probability into 0 or 1 to predict crisp classes directly.

Neural Network with Keras and Tensorflow

```
In [21]: # evaluate the keras model
         _, accuracy = model1.evaluate(x, y)
         print('Accuracy: %.2f' % (accuracy*100))

         24/24 [==============================] - 0s 870us/step - loss: 0.4977 - accuracy: 0.7591
         Accuracy: 75.91

In [22]: # make probability predictions with the model
         predictions = model1.predict(x)
         # round predictions
         rounded = [round(x[0]) for x in predictions]

         24/24 [==============================] - 0s 826us/step

In [23]: # make class predictions with the model
         predictions = (model1.predict(x) > 0.5).astype(int)

         24/24 [==============================] - 0s 783us/step
```

Neural Network with Keras and Tensorflow

```
In [25]: # summarize the first 5 cases
         for i in range(5):
          print('%s => %d (expected %d)' % (x[i].tolist(), predictions[i], y[i]))

[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0] => 1 (expected 1)
[1.0, 85.0, 66.0, 29.0, 0.0, 26.6, 0.351, 31.0] => 0 (expected 0)
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0] => 1 (expected 1)
[1.0, 89.0, 66.0, 23.0, 94.0, 28.1, 0.167, 21.0] => 0 (expected 0)
[0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0] => 1 (expected 1)

C:\Users\Paris\AppData\Local\Temp\ipykernel_8828\3023483958.py:3: DeprecationWarning: Conversion of an array with ndim > 0 to a
scalar is deprecated, and will error in future. Ensure you extract a single element from your array before performing this oper
ation. (Deprecated NumPy 1.25.)
  print('%s => %d (expected %d)' % (x[i].tolist(), predictions[i], y[i]))
```

https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/