week7_ipynb

February 17, 2021

1 Week 7: Neural Networks with TensorFlow

This week we will go over how to use TensorFlow to implement a neural network

```
[100]: import tensorflow as tf
       from tensorflow import keras
       from tensorflow.keras import layers
```

For now, we will be working with the Sequential neural network class in keras. A sequential neural net is one where each layer takes in a single set of inputs and outputs a single set of outputs. When our input is a vector (as has been the case so far) we can think of this as each layer taking in a single input matrix and outputing a single output matrix, where the input matrix has shape

$$(\text{number of neurons in past layer}) \times (\text{number of neurons in this layer})$$

and the output matrix has shape

$$(\text{number of neurons in this layer}) \times (\text{number of neurons in next layer})$$

Non zero entries in these matrices will indicate that the output from a neuron in a prior layer is being passed to a neuron in the next layer.

We can initialize this neural network using `model = keras.Sequential()`

```
[101]: model = keras.Sequential()
```

Now that we have initiallized our model as a sequential neural network, we want to start to add layers to this neural net. To initialize a dense layer we can use

`layers.Dense(num_neurons)`

A Dense layer means each neuron will take in all the inputs from the past layer (input matrix is fully non-zero). There are a few other options we can use when initializing a layer:

`use_bias = True/False`

This specifies whether the neurons in this layer will use a bias term. By default use_bias is set to be True.

`activation = "relu"/"sigmoid"/etc.`

This specifies what activation function is used in the layer. If this is left unspecified, then we will use a linear activation function $\pi(x) = x$.

We also may want to specify the shape of the initial input. We can do this using

`keras.Input(shape = ())`

In this particular case we will be using 4 features to clasify penguins, so we specify our input shape as (4,)

[102]:
```
input = keras.Input(shape = (4,))
layer1 = layers.Dense(2, use_bias = True, activation = "sigmoid")
layer2 = layers.Dense(10, use_bias = True, activation = "relu")
layer3 = layers.Dense(20, use_bias = False)
output = layers.Dense(1, use_bias = True, activation = "sigmoid")
```

Once we have set up our layers, we can add them to our neural network using `model.add()`

[103]:
```
model.add(input)
model.add(layer1)
model.add(layer2)
model.add(layer3)
```

We can remove the last layer by using `model.pop()`

[104]:
```
print(len(model.layers))
model.pop()
print(len(model.layers))
```

```
3
2
```

At this point we can add the output layer

[105]:
```
model.add(output)
```

Our model is now initialized. We can use the `model.summary()` model to take a look at the layers in our model

[106]:
```
model.summary()
```

```
Model: "sequential_5"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_42 (Dense)             (None, 2)                 10

_________________________________________________________________
dense_43 (Dense)             (None, 10)                30

_________________________________________________________________
dense_45 (Dense)             (None, 1)                 11
=================================================================
Total params: 51
Trainable params: 51
```

```
Non-trainable params: 0
_________________________________________________________________
```

and the `model.weights` attribute to see what (random) weights the model has been initialized with

[107]:
```python
model.weights
```

[107]:
```
[<tf.Variable 'dense_42/kernel:0' shape=(4, 2) dtype=float32, numpy=
 array([[-0.79385805, -0.17703986],
        [ 0.05690479, -0.1694057 ],
        [-0.20957494, -0.74130917],
        [-0.16002965, -0.71957016]], dtype=float32)>,
 <tf.Variable 'dense_42/bias:0' shape=(2,) dtype=float32, numpy=array([0., 0.],
dtype=float32)>,
 <tf.Variable 'dense_43/kernel:0' shape=(2, 10) dtype=float32, numpy=
 array([[ 0.2694075 ,  0.028027  ,  0.70394963, -0.4074887 ,  0.6567015 ,
          0.3653553 , -0.6868362 ,  0.5480557 ,  0.5594962 , -0.23371726],
        [ 0.05369288, -0.57631075,  0.4588763 ,  0.1184721 ,  0.33819538,
         -0.4287973 , -0.3378166 , -0.24377418,  0.47639102, -0.34292564]],
       dtype=float32)>,
 <tf.Variable 'dense_43/bias:0' shape=(10,) dtype=float32, numpy=array([0., 0.,
0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)>,
 <tf.Variable 'dense_45/kernel:0' shape=(10, 1) dtype=float32, numpy=
 array([[-0.34175256],
        [ 0.62870115],
        [ 0.6746159 ],
        [ 0.2061693 ],
        [ 0.677187  ],
        [-0.2911702 ],
        [ 0.10088718],
        [-0.09161514],
        [ 0.6591665 ],
        [-0.250164  ]], dtype=float32)>,
 <tf.Variable 'dense_45/bias:0' shape=(1,) dtype=float32, numpy=array([0.],
dtype=float32)>]
```

We can see what predictions our model would make given these weights by just pasing it a single 4 element vector.

[108]:
```python
import seaborn as sns
import numpy as np
penguins = sns.load_dataset('penguins').dropna()
penguins.head()
Y = 1*(penguins['species']=="Adelie")
X =␣
 ↪penguins[['bill_length_mm','bill_depth_mm','flipper_length_mm','body_mass_g']]
test = np.array(X[0:1])
test.shape
```

```
y = model(test)
print(y)
```

```
tf.Tensor([[0.5]], shape=(1, 1), dtype=float32)
```

Now, we want to move on to training our neural network using the data. We first split our data into testing and training.

[109]:
```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, Ytrain, Ytest = train_test_split(X,Y)
```

We first specify a training configuration using

`model.compile(optimizer, loss, metrics)`

The optimizer tells keras how to numerically compute the derivative at each step of the back-popogation. Examples include `keras.optimizers.SGD()`, `keras.optimizers.Adam()`, or `keras.optimizers.RMSprop()`. For now we use Adam, which implements a stochastic gradient descent estimating the first and second derivatives at each step of the back propogation.

The loss function tells us what loss function we want to minimize when doing back-propogation. Examples include `keras.losses.MeanSquaredError()`, `keras.losses.KLDivergence()`, etc. For this example we will use the MSE.

Finally, the metrics argument tells what metrics to use to describe our models performance on the training data. Examples include `keras.metrics.Accuracy()`, `keras.metrics.Crossentropy()`, etc. We can pass multiple metrics to this argument. For this example we will use the KL Divergence.

[136]:
```
model.compile(optimizer = keras.optimizers.Adam(), loss=keras.losses.
 ↪MeanSquaredError(), metrics=[keras.metrics.KLDivergence()])
```

Finally, we fit the model using

`model.fit(Xtrain, Ytrain, epochs)`

When training the model keras will try to split up the data into batches (randomly) and do back-progogation to compute the gradient on each subsample. Epochs tells the data how many times to iterate through and use the whole dataset and run back-propogation. For now we try 50 epochs.

[113]:
```
model.fit(Xtrain, Ytrain, epochs = 50)
```

```
Epoch 1/50
8/8 [==============================] - 0s 749us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3554
Epoch 2/50
8/8 [==============================] - 0s 797us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3556
Epoch 3/50
8/8 [==============================] - 0s 733us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3557
Epoch 4/50
8/8 [==============================] - 0s 750us/step - loss: 0.2451 - accuracy:
```

```
0.0000e+00 - kullback_leibler_divergence: 0.3560
Epoch 5/50
8/8 [==============================] - 0s 782us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3562
Epoch 6/50
8/8 [==============================] - 0s 842us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3564
Epoch 7/50
8/8 [==============================] - 0s 778us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3566
Epoch 8/50
8/8 [==============================] - 0s 839us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3567
Epoch 9/50
8/8 [==============================] - 0s 706us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3568
Epoch 10/50
8/8 [==============================] - 0s 725us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3569
Epoch 11/50
8/8 [==============================] - 0s 839us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3569
Epoch 12/50
8/8 [==============================] - 0s 750us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3571
Epoch 13/50
8/8 [==============================] - 0s 748us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3571
Epoch 14/50
8/8 [==============================] - 0s 675us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3572
Epoch 15/50
8/8 [==============================] - 0s 746us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3573
Epoch 16/50
8/8 [==============================] - 0s 798us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3573
Epoch 17/50
8/8 [==============================] - 0s 758us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3573
Epoch 18/50
8/8 [==============================] - 0s 728us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3575
Epoch 19/50
8/8 [==============================] - 0s 787us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3577
Epoch 20/50
8/8 [==============================] - 0s 664us/step - loss: 0.2451 - accuracy:
```

```
0.0000e+00 - kullback_leibler_divergence: 0.3578
Epoch 21/50
8/8 [==============================] - 0s 883us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3579
Epoch 22/50
8/8 [==============================] - 0s 719us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3578
Epoch 23/50
8/8 [==============================] - 0s 745us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3579
Epoch 24/50
8/8 [==============================] - 0s 838us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3581
Epoch 25/50
8/8 [==============================] - 0s 743us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3583
Epoch 26/50
8/8 [==============================] - 0s 913us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3584
Epoch 27/50
8/8 [==============================] - 0s 748us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3583
Epoch 28/50
8/8 [==============================] - 0s 776us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3584
Epoch 29/50
8/8 [==============================] - 0s 906us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3588
Epoch 30/50
8/8 [==============================] - 0s 731us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3592
Epoch 31/50
8/8 [==============================] - 0s 738us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3593
Epoch 32/50
8/8 [==============================] - 0s 794us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3594
Epoch 33/50
8/8 [==============================] - 0s 755us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3594
Epoch 34/50
8/8 [==============================] - 0s 831us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3593
Epoch 35/50
8/8 [==============================] - 0s 750us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3595
Epoch 36/50
8/8 [==============================] - 0s 827us/step - loss: 0.2451 - accuracy:
```

```
0.0000e+00 - kullback_leibler_divergence: 0.3596
Epoch 37/50
8/8 [==============================] - 0s 750us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3595
Epoch 38/50
8/8 [==============================] - 0s 715us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3596
Epoch 39/50
8/8 [==============================] - 0s 899us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3598
Epoch 40/50
8/8 [==============================] - 0s 689us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3598
Epoch 41/50
8/8 [==============================] - 0s 810us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3598
Epoch 42/50
8/8 [==============================] - 0s 757us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3599
Epoch 43/50
8/8 [==============================] - 0s 752us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3600
Epoch 44/50
8/8 [==============================] - 0s 777us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3602
Epoch 45/50
8/8 [==============================] - 0s 757us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3602
Epoch 46/50
8/8 [==============================] - 0s 790us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3602
Epoch 47/50
8/8 [==============================] - 0s 725us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3604
Epoch 48/50
8/8 [==============================] - 0s 766us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3605
Epoch 49/50
8/8 [==============================] - 0s 764us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3606
Epoch 50/50
8/8 [==============================] - 0s 788us/step - loss: 0.2451 - accuracy:
0.0000e+00 - kullback_leibler_divergence: 0.3605
```

```
[113]: <tensorflow.python.keras.callbacks.History at 0x7f86f7439cd0>
```

Using the predict method we can make predictions and validate this neural net using our testing data

[137]:
```
Yhat = 1*(model.predict(Xtest).flatten() >= 0.5)
print(np.mean(Yhat == Ytest))
```

```
0.5357142857142857
```

Our model preforms just ok. But, we did get it to work.

[ ]:
```
```