1. Load Data

```
# The first step is to define the functions and classes we intend to use in this tutori
# We will use the NumPy library to load our dataset
from numpy import loadtxt
```

```
?loadtxt  # Information about function
```

We can now load our dataset.

In this Keras tutorial, we are going to use the **Pima Indians onset of diabetes** dataset. This is a sta
Machine Learning repository. It describes patient medical record data for Pima Indians and wheth
years.

As such, it is a binary classification problem (onset of diabetes as 1 or not as 0). All of the input va
numerical. This makes it easy to use directly with neural networks that expect numerical input and
network in Keras.

Link: www.nrvis.com/data/mldata/pima-indians-diabetes.csv

We can now load the file as a matrix of numbers using the NumPy function loadtxt(). There are eig
(the last column). We will be learning a model to map rows of input variables (X) to an output vari

The variables can be summarized as follows:

Input Variables (X):

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)

Output Variables (y):

1. Class variable (0 or 1)

```
# load the dataset
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=',')
dataset.shape
```

```
(768, 9)
```

```
dataset[0], dataset[1]
```

```
(array([  6.   , 148.   ,  72.   ,  35.   ,   0.   ,  33.6  ,   0.627,
         50.   ,   1.   ]),
 array([ 1.   , 85.   , 66.   , 29.   ,  0.   , 26.6  ,  0.351, 31.   ,
         0.   ]))
```

```
# split into input (X) and output (y) variables
# We can split the array into two arrays by selecting subsets of columns using the star
# We can select the first 8 columns from index 0 to index 7 via the slice 0:8. We can t
X = dataset[:,0:8]
y = dataset[:,8]
X.shape, y.shape
```

```
((768, 8), (768,))
```

```
x_train = X[:600]
x_test = X[600:]
y_train = y[:600]
y_test = y[600:]
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

```
((600, 8), (168, 8), (600,), (168,))
```

2. Define Keras Model

Models in Keras are defined as a sequence of layers.

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

The first thing to get right is to ensure the input layer has the right number of input features. This c with the input_dim argument and setting it to 8 for the 8 input variables.

How do we know the number of layers and their types?

This is a very hard question. There are heuristics that we can use and often the best network struc error experimentation (I explain more about this here). Generally, you need a network large enough

In this example, we will use a fully-connected network structure with three layers.

Fully connected layers are defined using the Dense class. We can specify the number of neurons c specify the activation function using the activation argument.

We will use the rectified linear unit activation function referred to as ReLU on the first two layers a

It used to be the case that Sigmoid and Tanh activation functions were preferred for all layers. The using the ReLU activation function. We use a sigmoid on the output layer to ensure our network ou either a probability of class 1 or snap to a hard classification of either class with a default thresho

```
# We can piece it all together by adding each layer:
#
# * The model expects rows of data with 8 variables (the input_dim=8 argument)
# * The first hidden layer has 12 nodes and uses the relu activation function.
```

```
# * The second hidden layer has 8 nodes and uses the relu activation function.
# * The output layer has one node and uses the sigmoid activation function
```

```
from keras.models import Sequential
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the %t

```
# define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl
```

```
# Note, the most confusing thing here is that the shape of the input to the model lis c
# the first hidden layer. This means that the line of code that adds the first Dense la
# defining the input or visible layer and the first hidden layer.
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 12)                108       
_________________________________________________________________
dense_2 (Dense)              (None, 8)                 104       
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9         
=================================================================
Total params: 221
Trainable params: 221
Non-trainable params: 0
_________________________________________________________________
```

3. Compile Keras Model

Now that the model is defined, we can compile it.

Compiling the model uses the efficient numerical libraries under the covers (the so-called backen
backend automatically chooses the best way to represent the network for training and making pre

CPU or GPU or even distributed.

When compiling, we must specify some additional properties required when training the network. the best set of weights to map inputs to outputs in our dataset.

We must specify the loss function to use to evaluate a set of weights, the optimizer is used to sea and any optional metrics we would like to collect and report during training.

In this case, we will use cross entropy as the loss argument. This loss is for a binary classificatior "binary_crossentropy". You can learn more about choosing loss functions based on your problem https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-

We will define the optimizer as the efficient stochastic gradient descent algorithm "adam". This is because it automatically tunes itself and gives good results in a wide range of problems. To learn gradient descent see the post: https://machinelearningmastery.com/adam-optimization-algorithm

Finally, because it is a classification problem, we will collect and report the classification accuracy

```
1 # compile the keras model
2 model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimizers.py:79

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

4. Fit Keras Model

We have defined our model and compiled it ready for efficient computation.

Now it is time to execute the model on some data.

We can train or fit our model on our loaded data by calling the fit() function on the model.

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 cons weights are updated. One epoch is comprised of one or more batches, based on the chosen batch For more on the difference between epochs and batches, see the post: https://machinelearningma an-epoch/

The training process will run for a fixed number of iterations through the dataset called epochs, th argument. We must also set the number of dataset rows that are considered before the model we the batch size and set using the batch_size argument.

For this problem, we will run for a small number of epochs (150) and use a relatively small batch s involve (150/10) 15 updates to the model weights.

These configurations can be chosen experimentally by trial and error. We want to train the model enough) mapping of rows of input data to the output classification. The model will always have so after some point for a given model configuration. This is called model convergence.

```
1 # fit the keras model on the dataset
2 hisory = model.fit(x_train, y_train, epochs=10, batch_size=10)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

Epoch 1/10
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorfl

600/600 [==============================] - 1s 2ms/step - loss: 10.2939 - acc: 0.3467
Epoch 2/10
600/600 [==============================] - 0s 465us/step - loss: 10.0822 - acc: 0.346
Epoch 3/10
600/600 [==============================] - 0s 438us/step - loss: 2.5702 - acc: 0.4217
Epoch 4/10
600/600 [==============================] - 0s 456us/step - loss: 0.7294 - acc: 0.6033
Epoch 5/10
600/600 [==============================] - 0s 481us/step - loss: 0.6833 - acc: 0.6450
Epoch 6/10
600/600 [==============================] - 0s 471us/step - loss: 0.6663 - acc: 0.6517
Epoch 7/10
600/600 [==============================] - 0s 442us/step - loss: 0.6571 - acc: 0.6583
Epoch 8/10
600/600 [==============================] - 0s 448us/step - loss: 0.6547 - acc: 0.6550
Epoch 9/10
600/600 [==============================] - 0s 475us/step - loss: 0.6483 - acc: 0.6567
Epoch 10/10
600/600 [==============================] - 0s 488us/step - loss: 0.6448 - acc: 0.6617
```

5. Evaluate Keras Model

We have trained our neural network on the entire dataset and we can evaluate the performance of

This will only give us an idea of how well we have modeled the dataset (e.g. train accuracy), but no perform on new data. We have done this for simplicity, but ideally, you could separate your data in evaluation of your model.

You can evaluate your model on your training dataset using the evaluate() function on your model to train the model.

This will generate a prediction for each input and output pair and collect scores, including the ave configured, such as accuracy.

The evaluate() function will return a list with two values. The first will be the loss of the model on t accuracy of the model on the dataset. We are only interested in reporting the accuracy, so we will

```
1 model.evaluate(x_test, y_test)
```

```
168/168 [==============================] - 0s 284us/step
[0.6328285648709252, 0.6726190476190477]
```

7. Make Predictions

After I train my model, how can I use it to make predictions on new data?

Great question.

We can adapt the above example and use it to generate predictions on the training dataset, preter before.

Making predictions is as easy as calling the predict() function on the model. We are using a sigmo the predictions will be a probability in the range between 0 and 1. We can easily convert them into task by rounding them.

1. Number of times pregnant: **2**
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test: **160**
3. Diastolic blood pressure (mm Hg): **55**
4. Triceps skin fold thickness (mm): **30**
5. 2-Hour serum insulin (mu U/ml): **0**
6. Body mass index (weight in kg/(height in m)^2): **25**
7. Diabetes pedigree function: **0.7**
8. Age (years): **35**

```
1 import numpy as np
2 model.predict(np.array([[2, 160, 55, 30, 0, 25, 0.7, 35]]))
```

```
array([[0.41580808]], dtype=float32)
```