Artificial Neural Networks: Module 3 - Coding

Artificial Intelligence

Zool Hilmi Ismail
Tokyo City University

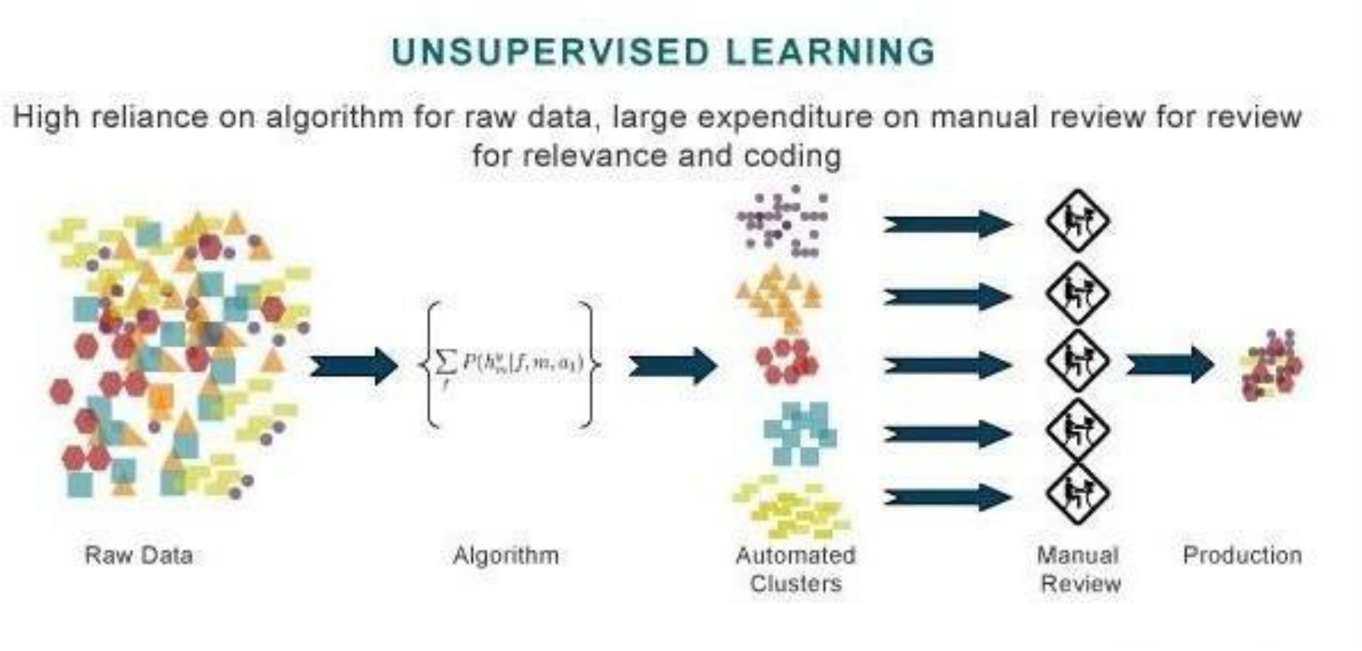
TYPE OF LEARNING

Unsupervised Learning

Semi-supervised Learning

Reinforcement Learning

Supervised Learning



UNSUPERVISED LEARNING

High reliance on algorithm for raw data, large expenditure on manual review for review for relevance and coding

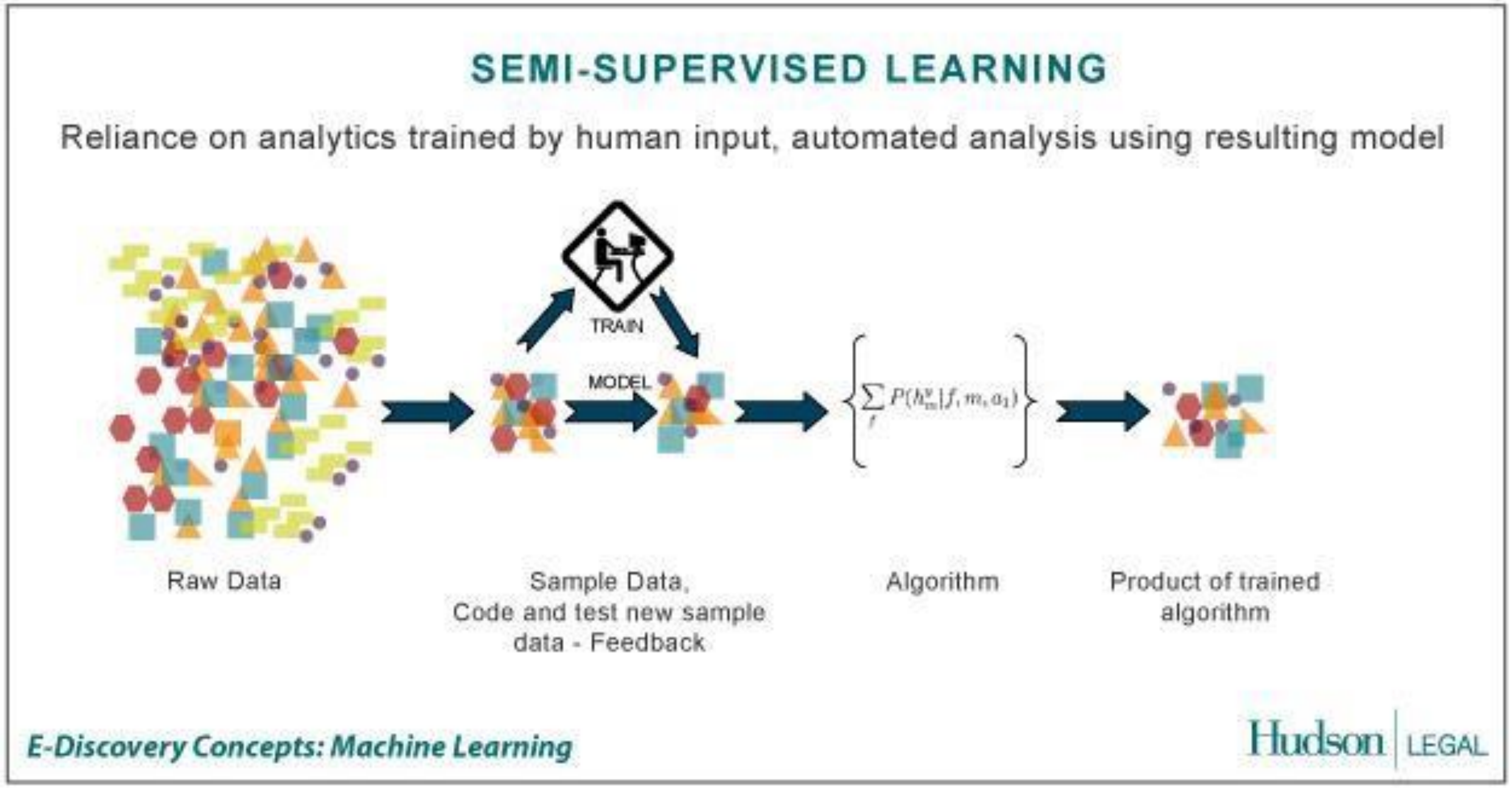
TYPE OF LEARNING

Unsupervised Learning

Semi-supervised Learning

Reinforcement Learning

Supervised Learning



SEMI-SUPERVISED LEARNING

Reliance on analytics trained by human input, automated analysis using resulting model

TRAIN
MODEL
$\sum_f P(h_m^v|f, m, a_1)$
Raw Data
Sample Data, Code and test new sample data - Feedback
Algorithm
Product of trained algorithm

E-Discovery Concepts: Machine Learning

Hudson | LEGAL

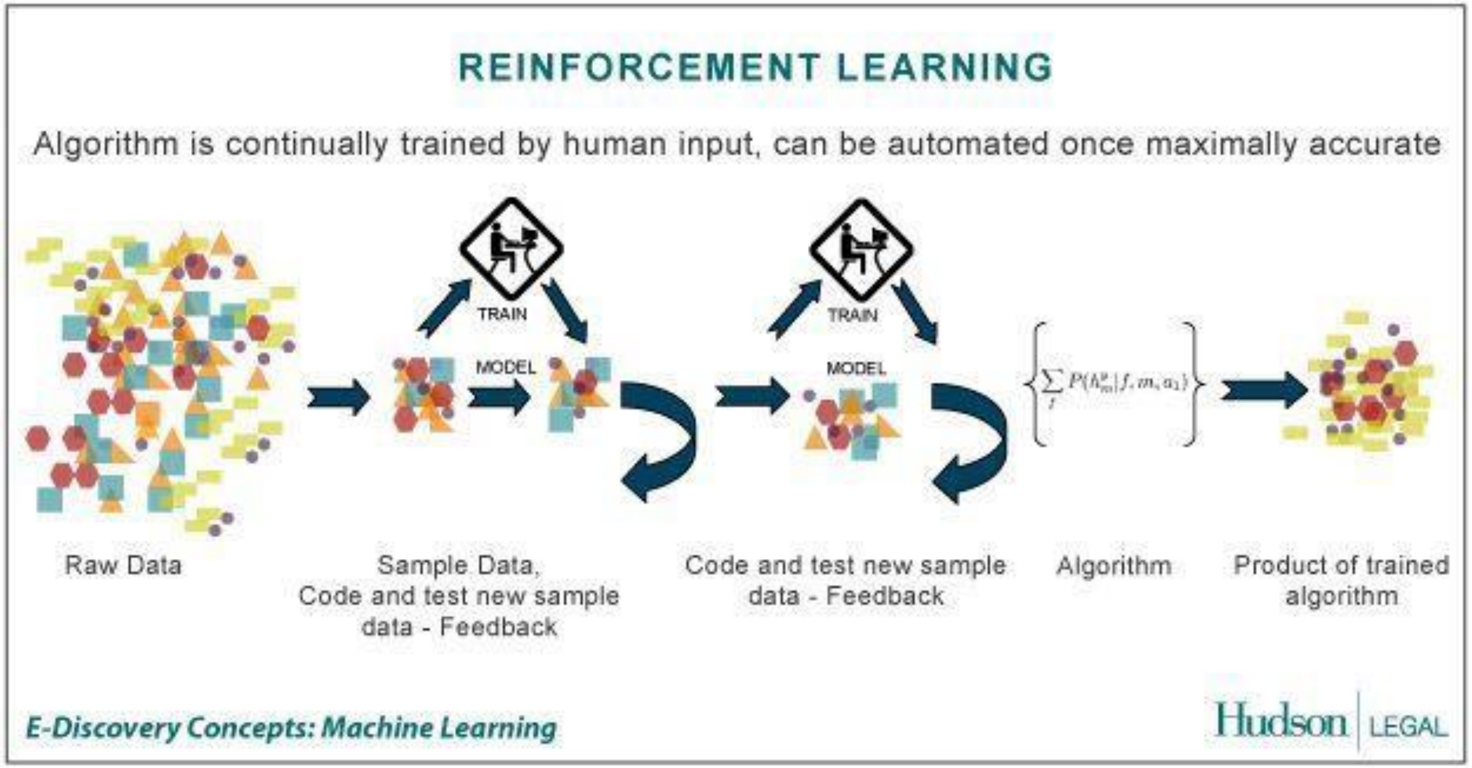
TYPE OF LEARNING

Unsupervised Learning

Semi-supervised Learning

Reinforcement Learning

Supervised Learning



REINFORCEMENT LEARNING

Algorithm is continually trained by human input, can be automated once maximally accurate

TRAIN
MODEL
TRAIN
MODEL
$\left\{\sum_{f} P(h_{m}^{p} | f, m, a_1)\right\}$
Raw Data
Sample Data, Code and test new sample data - Feedback
Code and test new sample data - Feedback
Algorithm
Product of trained algorithm
E-Discovery Concepts: Machine Learning
Hudson LEGAL

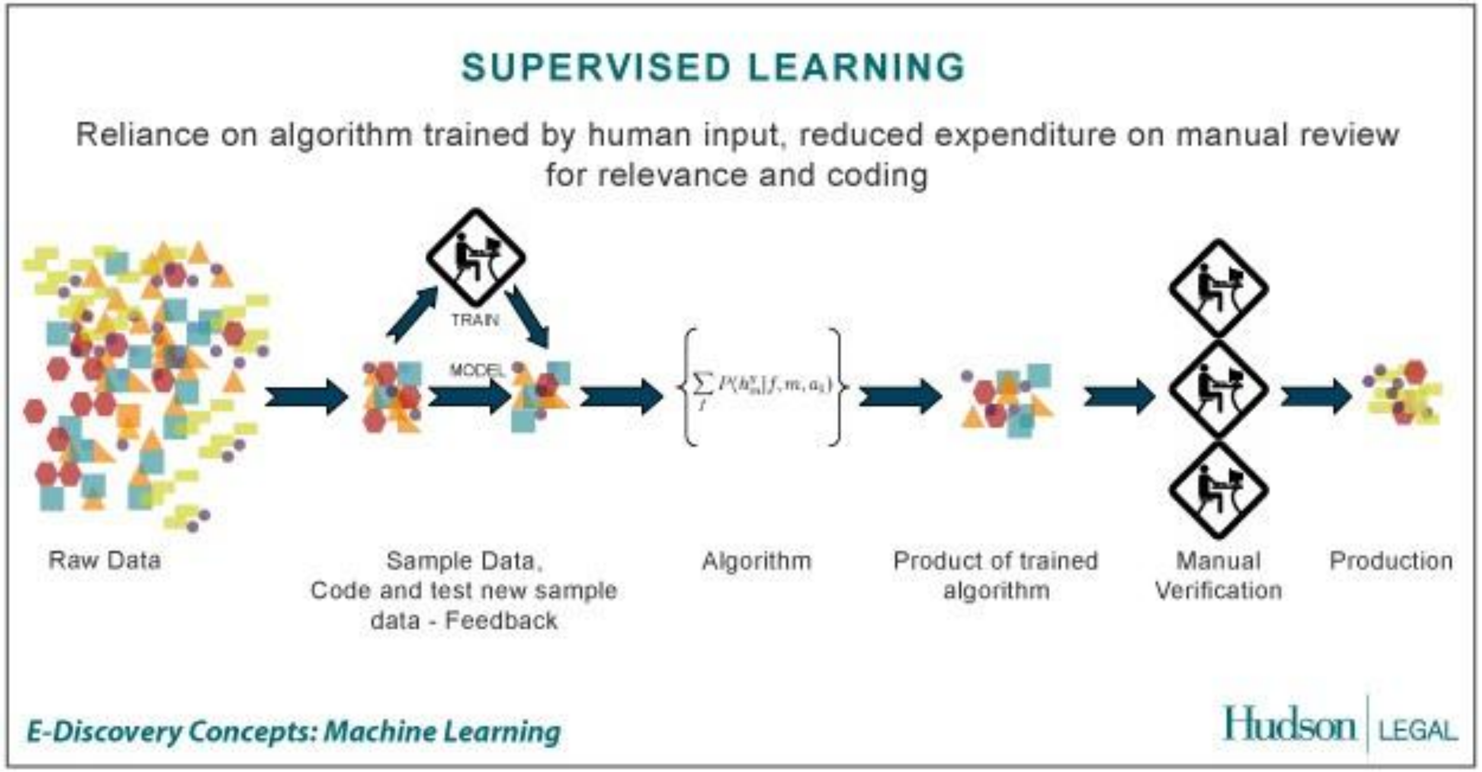
TYPE OF LEARNING

Unsupervised Learning

Semi-supervised Learning

Reinforcement Learning

Supervised Learning



SUPERVISED LEARNING

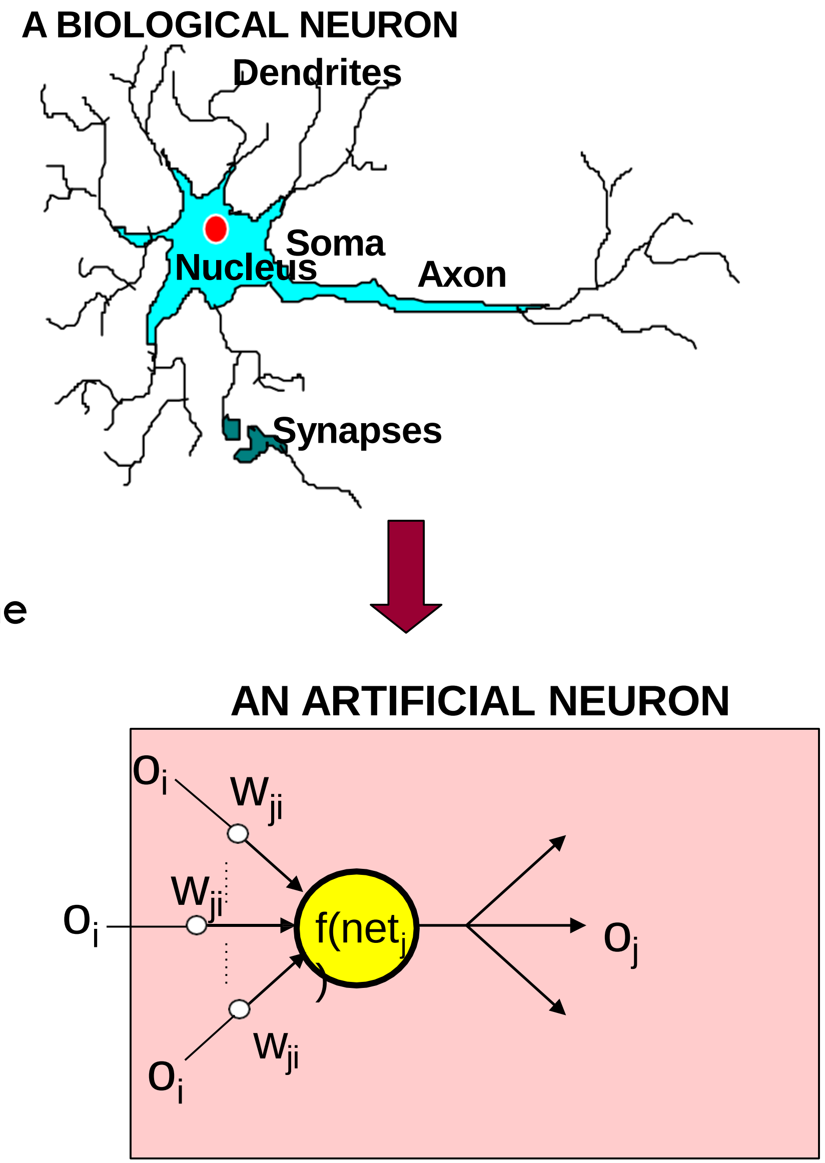
Reliance on algorithm trained by human input, reduced expenditure on manual review for relevance and coding

TRAIN
MODEL
$\sum_f P(h^y_m|f, m, a_k)$
Raw Data
Sample Data, Code and test new sample data - Feedback
Algorithm
Product of trained algorithm
Manual Verification
Production

E-Discovery Concepts: Machine Learning

- The biological neuron has lent insights into the design of the artificial neuron.
- The McCulloch-Pitts neuron which is the first artificial neuron was formulated in 1943

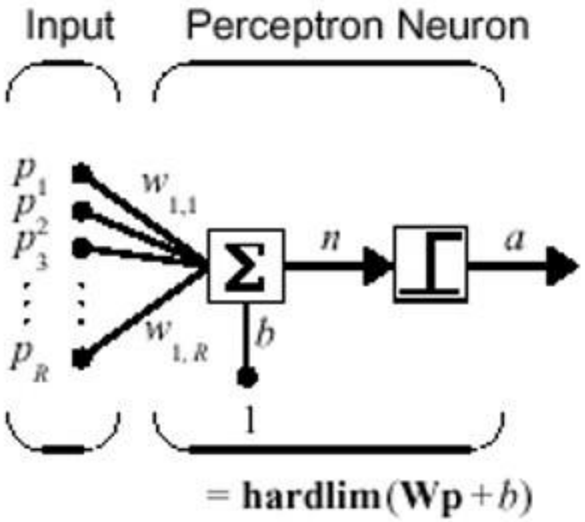
- Soma: also called the cell body which supports functions in the neuron.
- Axon: part of the neuron which carries the fired signal out of the neuron.
- Dendrites: very dense fiber-type of structure that are designed to receive incoming signals.
- Synapses: primary gateways where neurons communicate with each other. There is evidence that the chemical reactions at these synapses are altered when learning takes place.

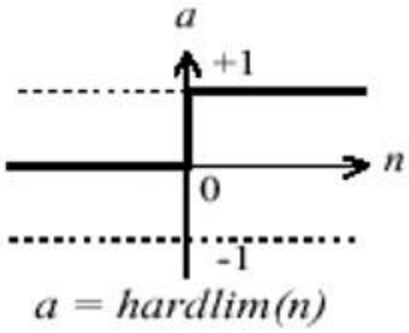


perceptron

Perceptrons

Neuron Model

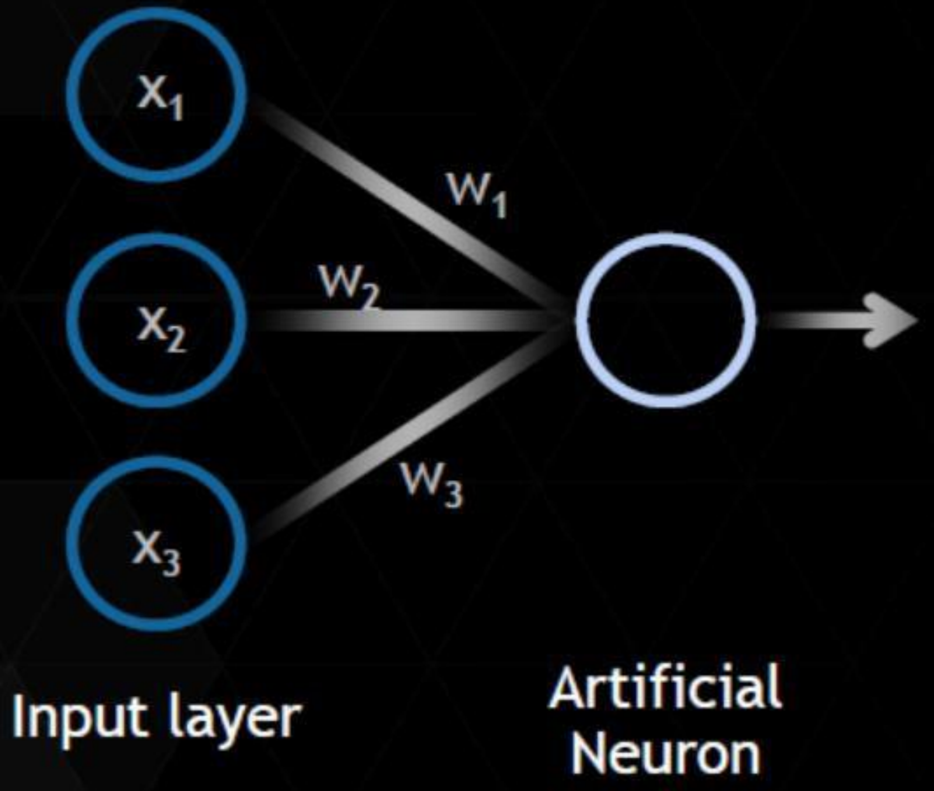




The perceptron neuron produces a 1 if the net input into the transfer function is equal to or greater than 0, otherwise it produces a 0.

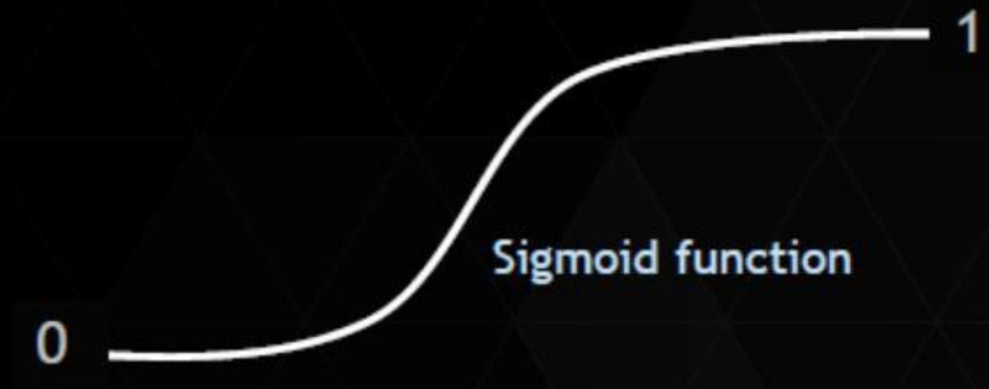
WHAT'S IN A NEURON?

Artificial neuron is modeled as a "Logistic Unit".

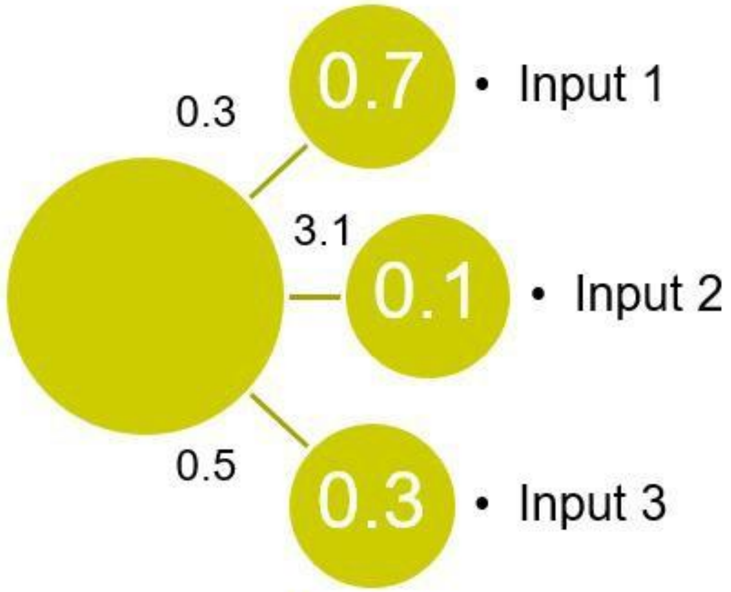


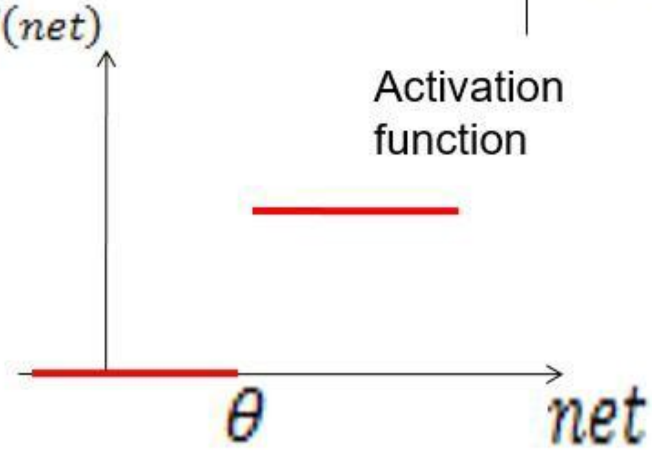
$$z = x_1 w_1 + x_2 w_2 + x_3 w_3$$

$$\text{Activation} = \frac{1}{1 + e^{-z}}$$



Combining input signals





Activation function

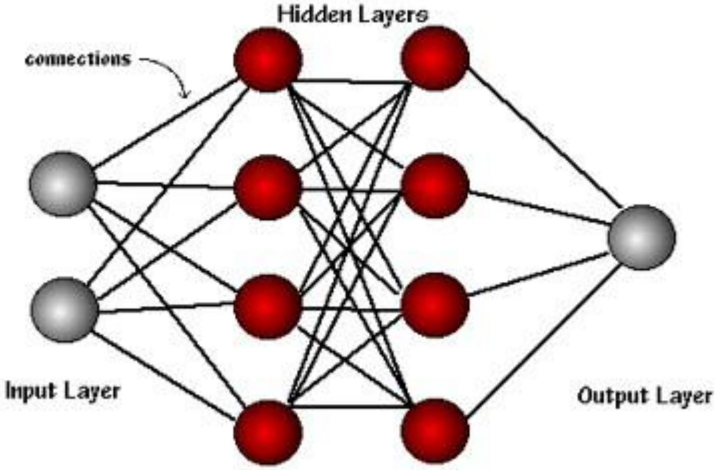
$$f(net) = \begin{cases} 1 \; if \; net \geq \theta \\ 0 \; if \; net < \theta \end{cases}$$

$$net_j = \sum_{i=1}^{n} x_i w_{ij}$$

$$net_j = (0.7 * 0.3) + (0.1 * 3.1) + (0.3 * 0.5) = 0.67$$

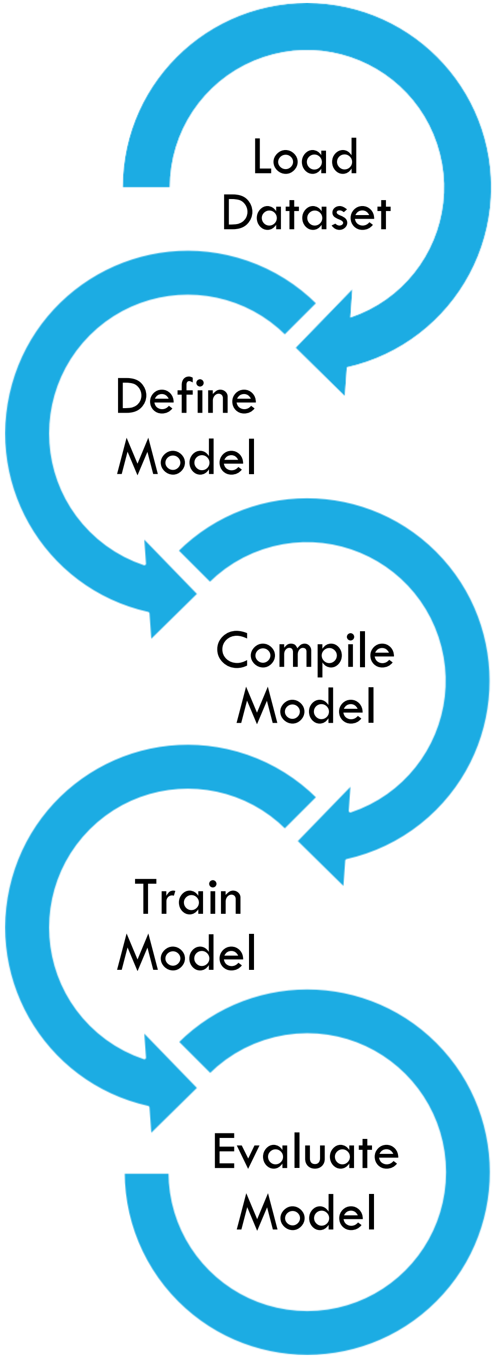
Feedforword NNs

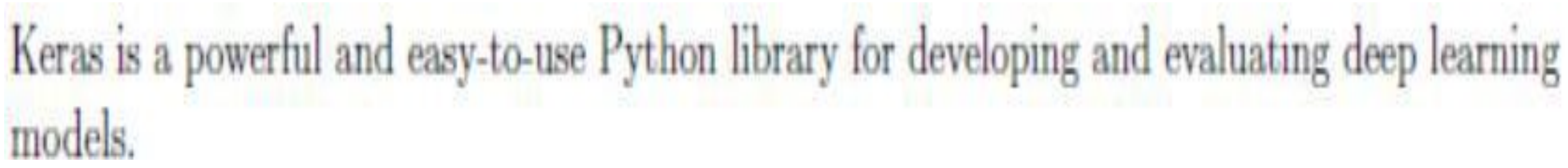
- The basic structure off a feedforward Neural Network



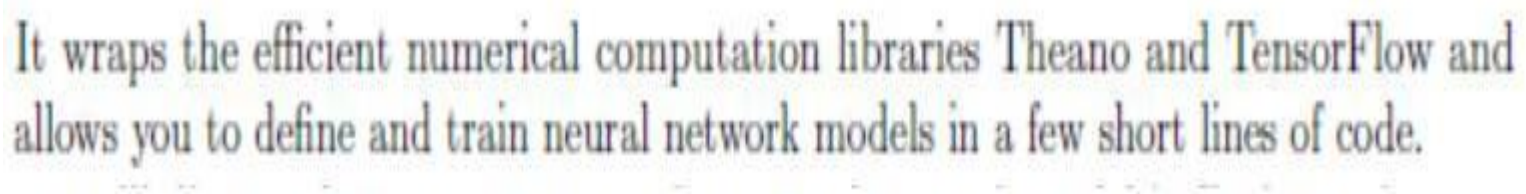
- The learning rule modifies the weights according to the input patterns that it is presented with. In a sense, ANNs learn by example as do their biological counterparts.
- When the desired output are known we have supervised learning or learning with a teacher.

DEVELOP ANN WITH KERAS



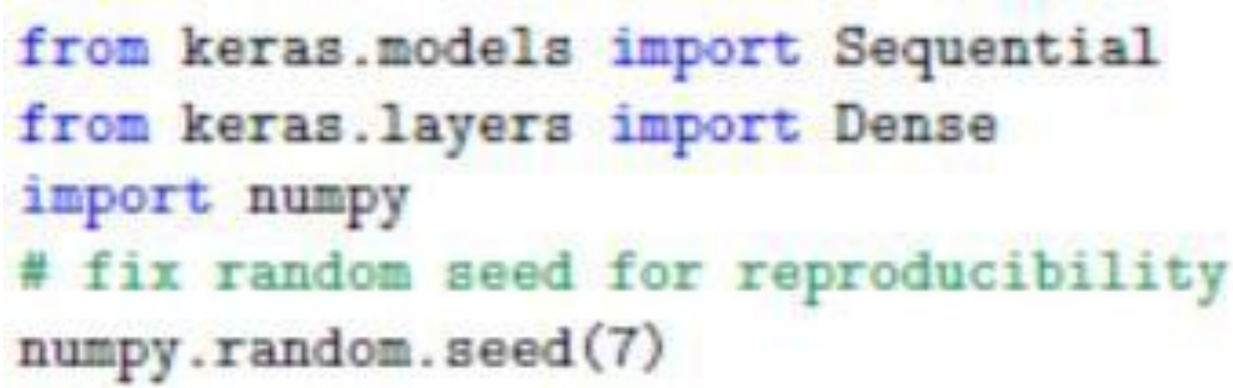


Keras is a powerful and easy-to-use Python library for developing and evaluating deep learning models.



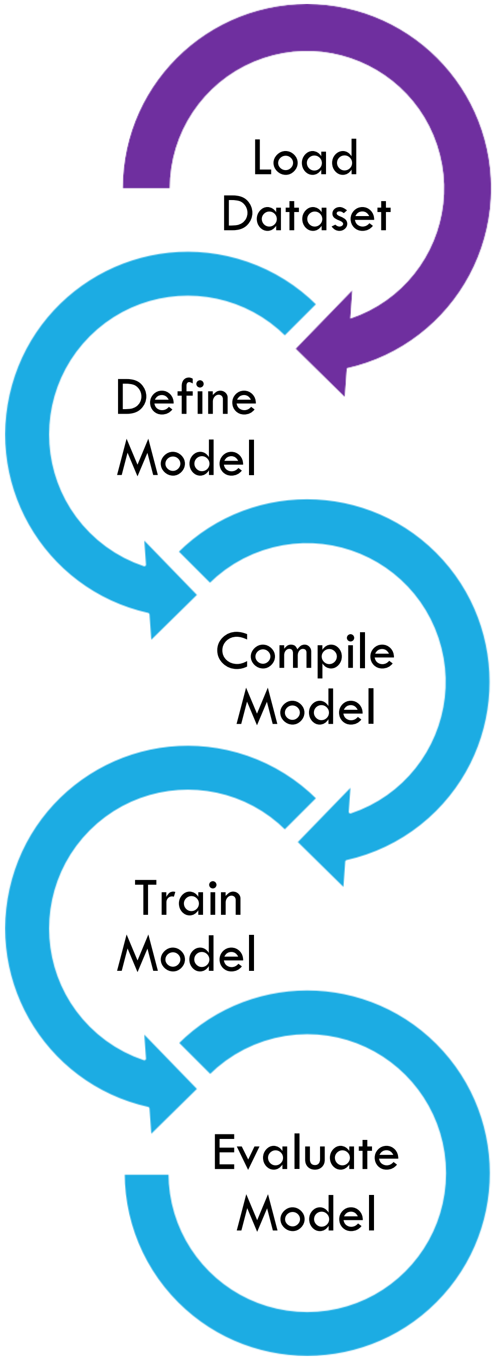
It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in a few short lines of code.

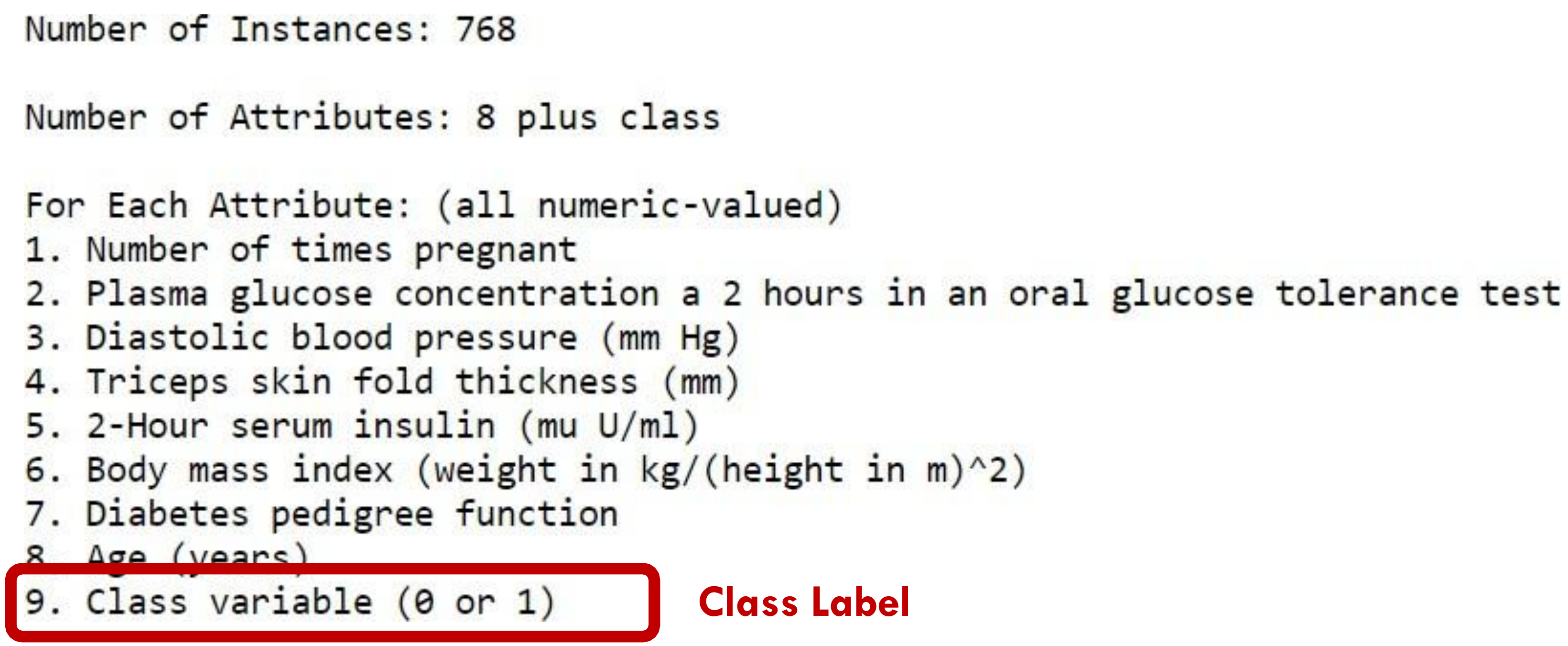
Prior to all these 5 steps, packages have to be imported



```
from keras.models import Sequential
from keras.layers import Dense
import numpy
# fix random seed for reproducibility
numpy.random.seed(7)
```

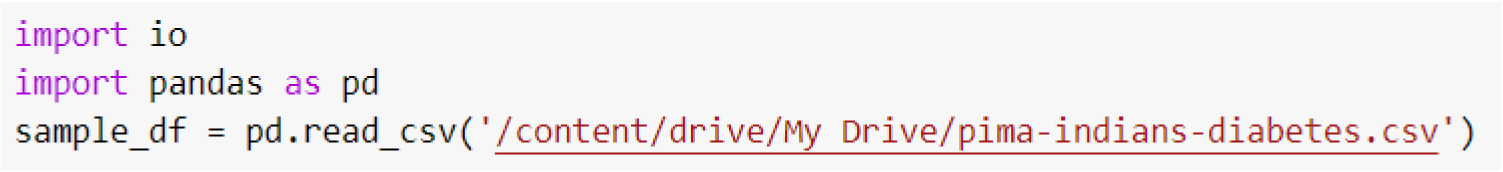
DEVELOP ANN WITH KERAS



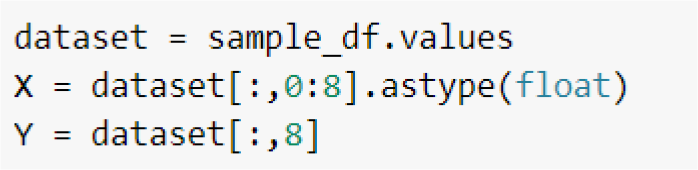


```
Number of Instances: 768

Number of Attributes: 8 plus class

For Each Attribute: (all numeric-valued)
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)
9. Class variable (0 or 1)
```

Class Label

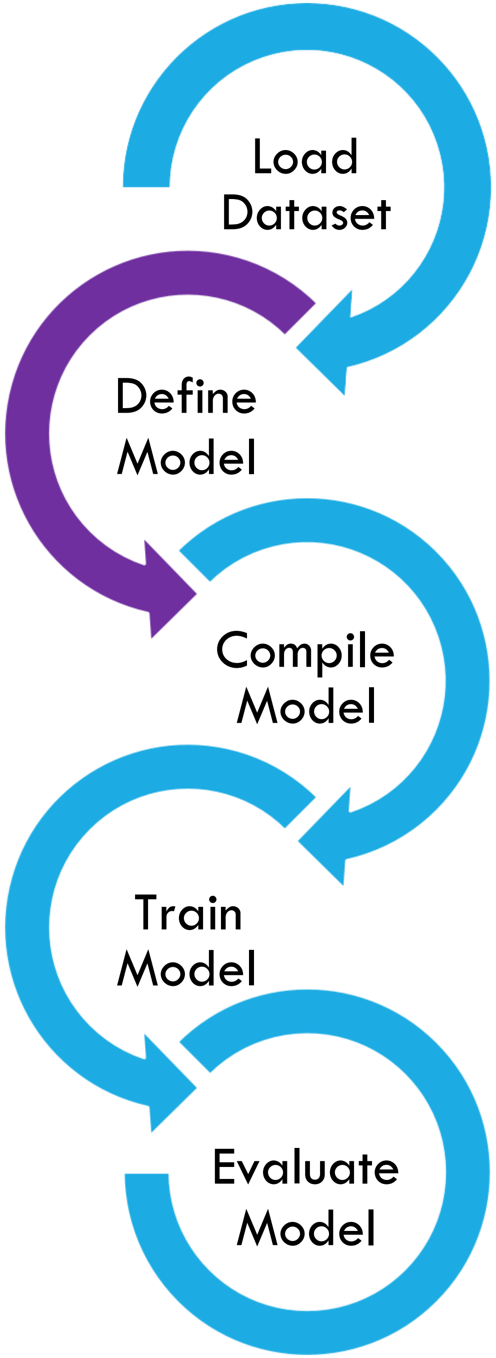


```python
import io
import pandas as pd
sample_df = pd.read_csv('/content/drive/My Drive/pima-indians-diabetes.csv')
```

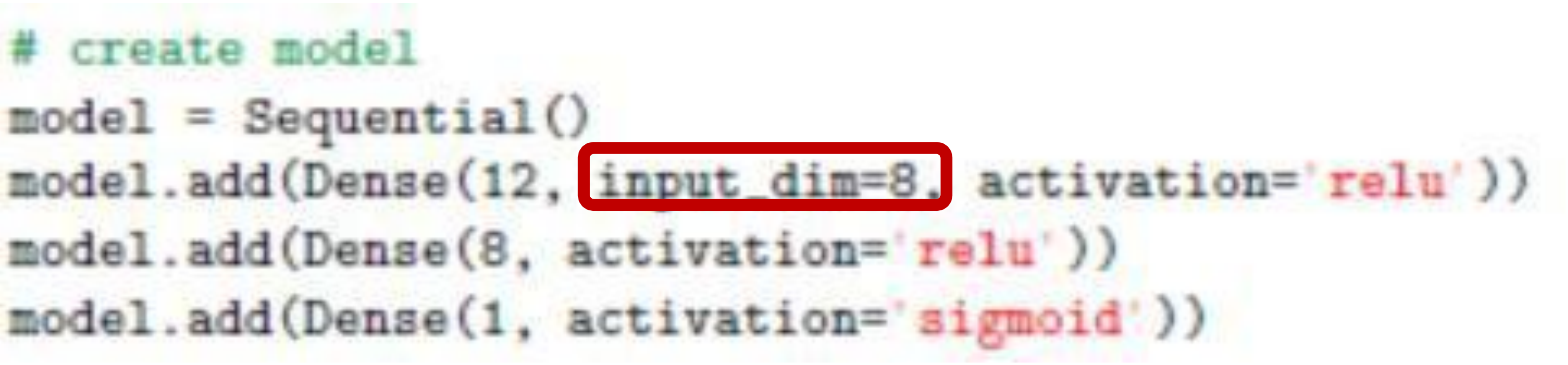


```python
dataset = sample_df.values
X = dataset[:,0:8].astype(float)
Y = dataset[:,8]
```

DEVELOP ANN WITH KERAS

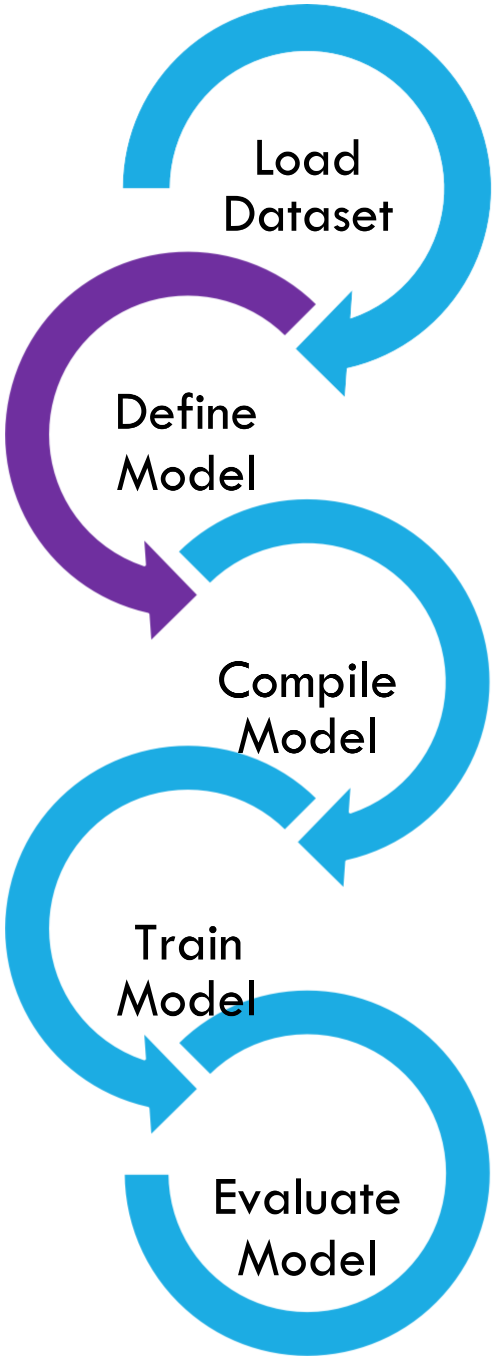


- Models in Keras are defined as sequence of layers
- Conventional ANN is defined as fully connected network
- Dense Class is imported to define fully connected network

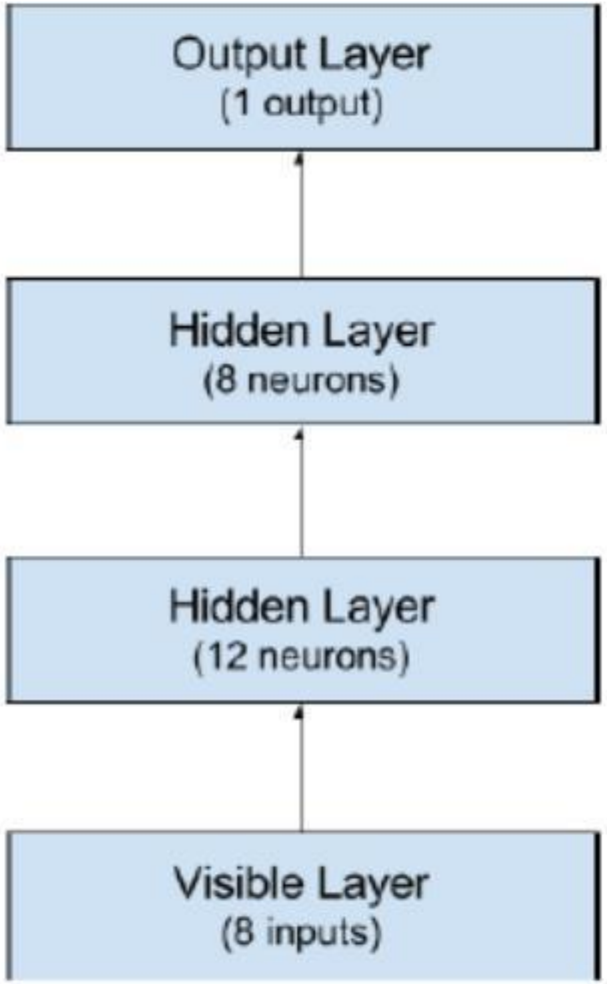


```
# create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

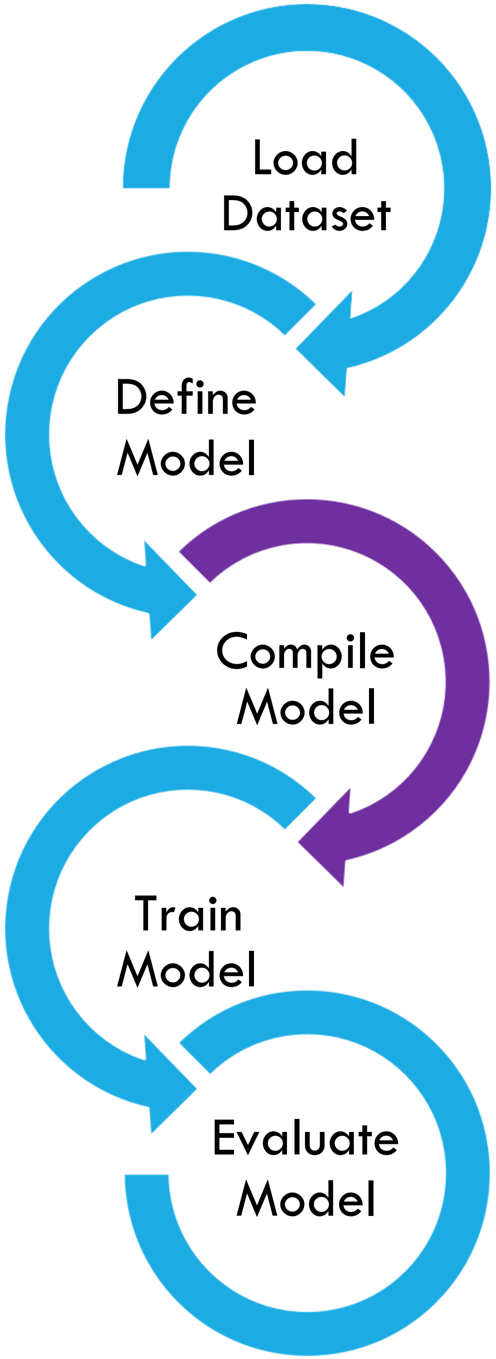
DEVELOP ANN WITH KERAS



- **The defined model**



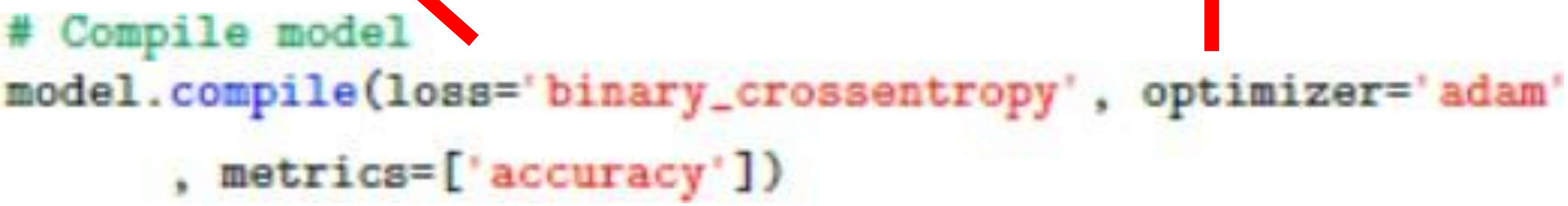
DEVELOP ANN WITH KERAS



- Models in Keras are defined as sequence of layers

Optimizer – gradient descent logarithmic

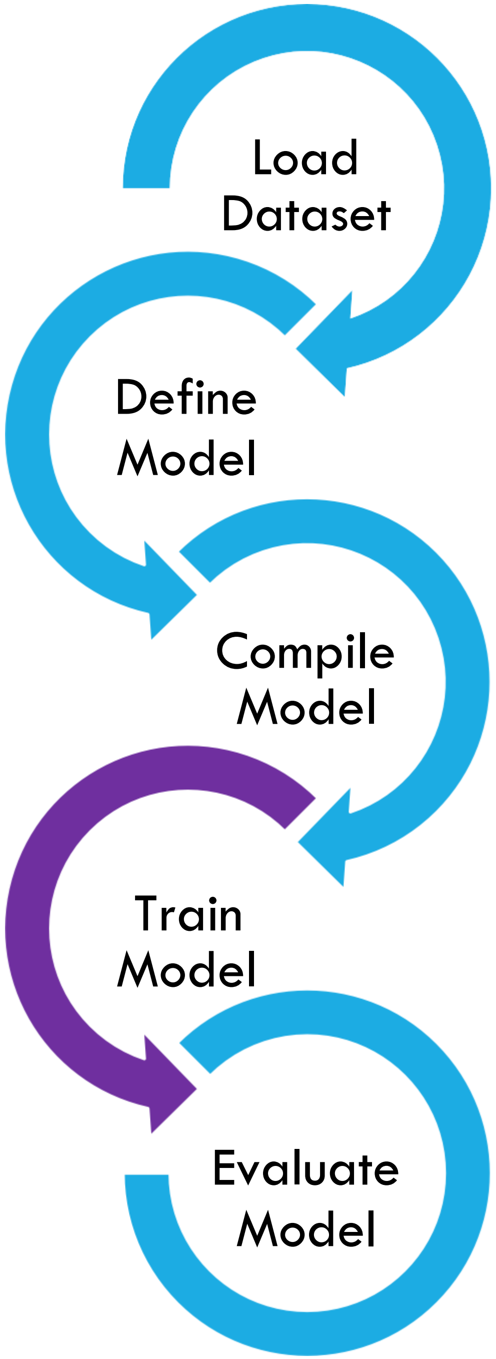
Loss function – logarithmic loss for binary



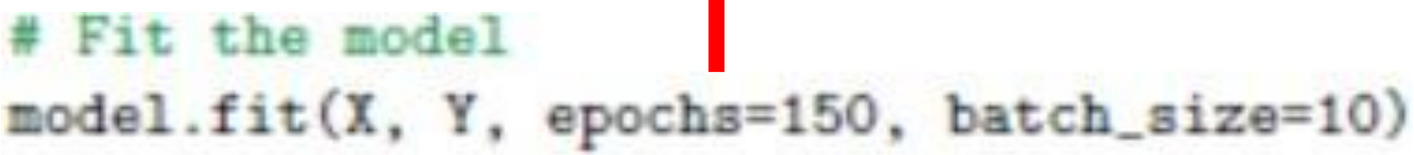
```
# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam'
        , metrics=['accuracy'])
```

Performance – classification accuracy

DEVELOP ANN WITH KERAS

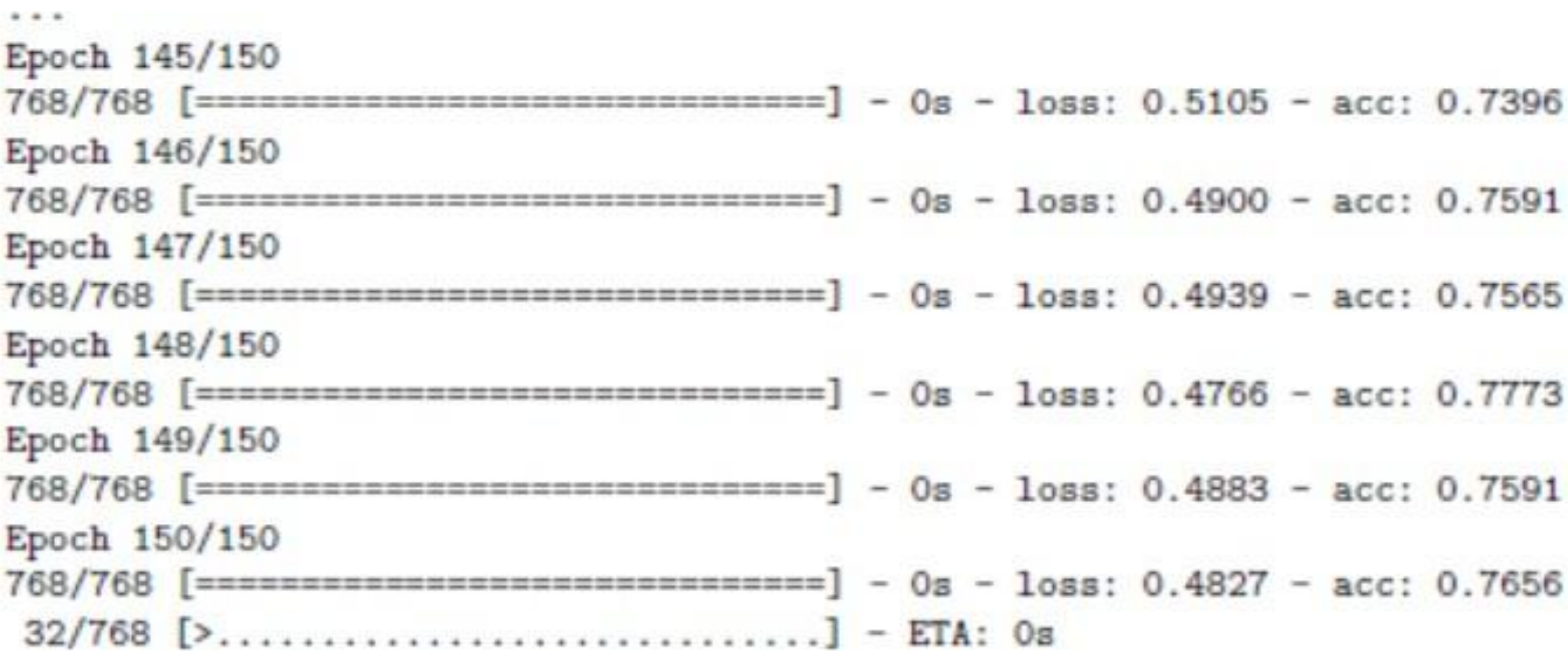


Epoch – number of iterations



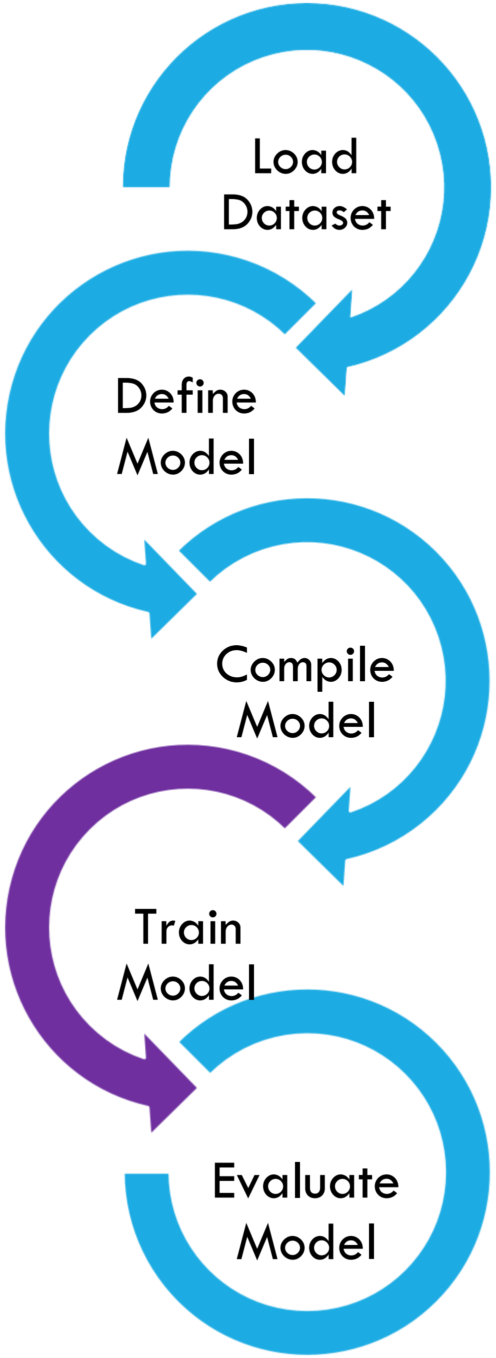
```
# Fit the model
model.fit(X, Y, epochs=150, batch_size=10)
```

Batch size – number of instances

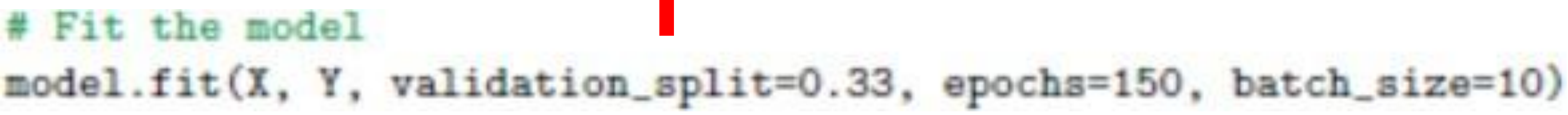


```
...
Epoch 145/150
768/768 [==============================] - 0s - loss: 0.5105 - acc: 0.7396
Epoch 146/150
768/768 [==============================] - 0s - loss: 0.4900 - acc: 0.7591
Epoch 147/150
768/768 [==============================] - 0s - loss: 0.4939 - acc: 0.7565
Epoch 148/150
768/768 [==============================] - 0s - loss: 0.4766 - acc: 0.7773
Epoch 149/150
768/768 [==============================] - 0s - loss: 0.4883 - acc: 0.7591
Epoch 150/150
768/768 [==============================] - 0s - loss: 0.4827 - acc: 0.7656
 32/768 [>.............................] - ETA: 0s
```

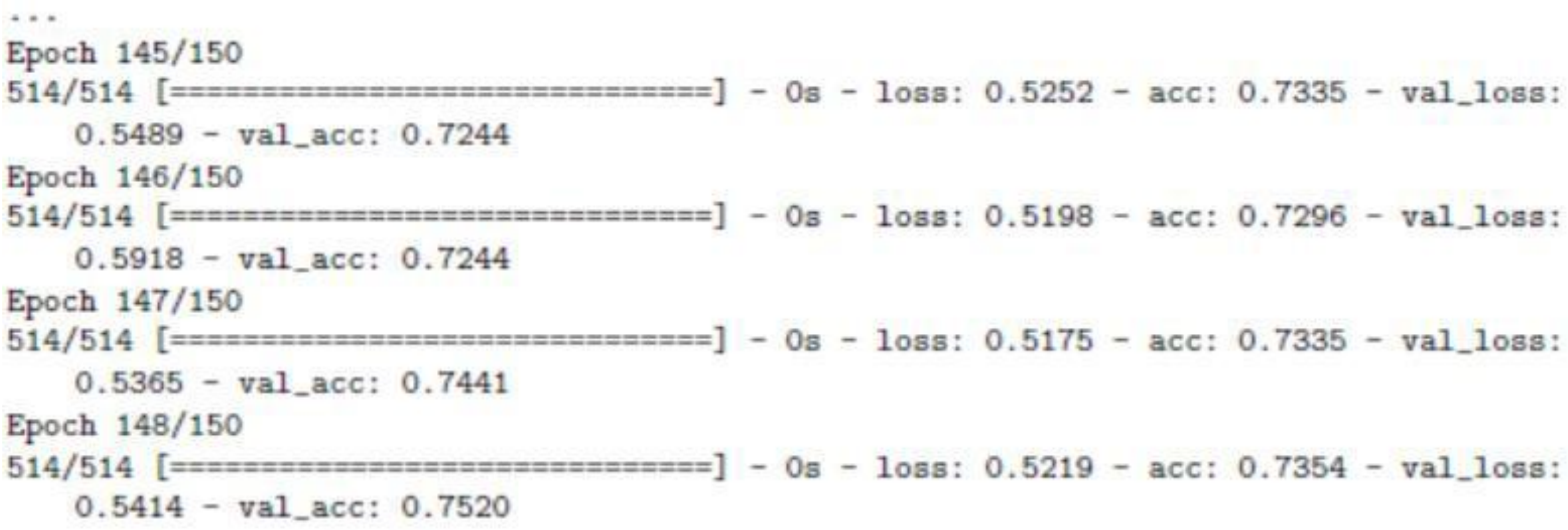
DEVELOP ANN WITH KERAS



Automatic Validation Dataset – percentage

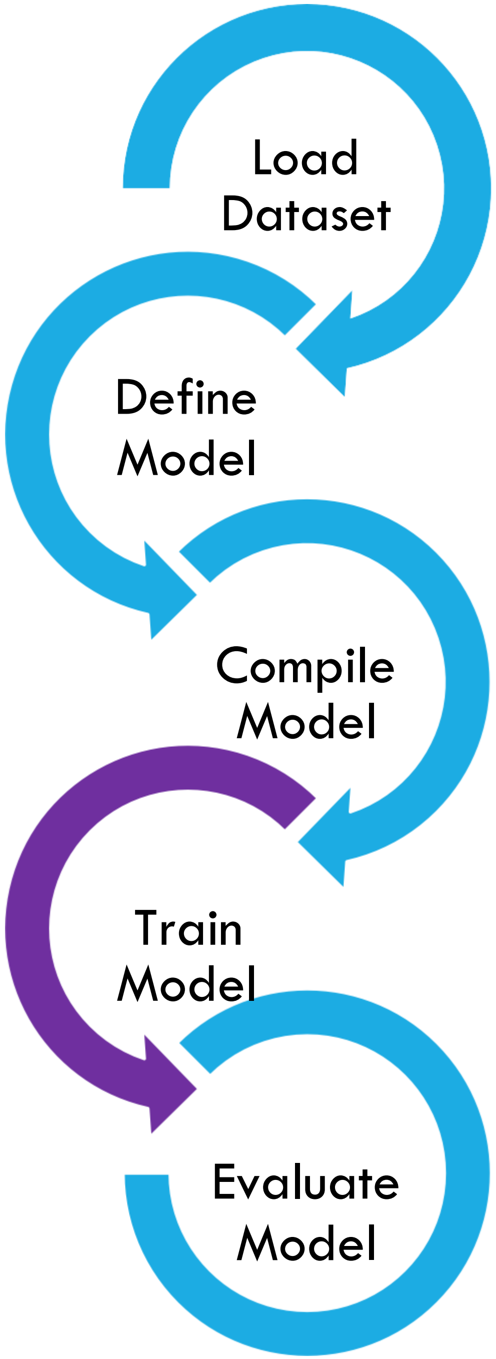


```
# Fit the model
model.fit(X, Y, validation_split=0.33, epochs=150, batch_size=10)
```

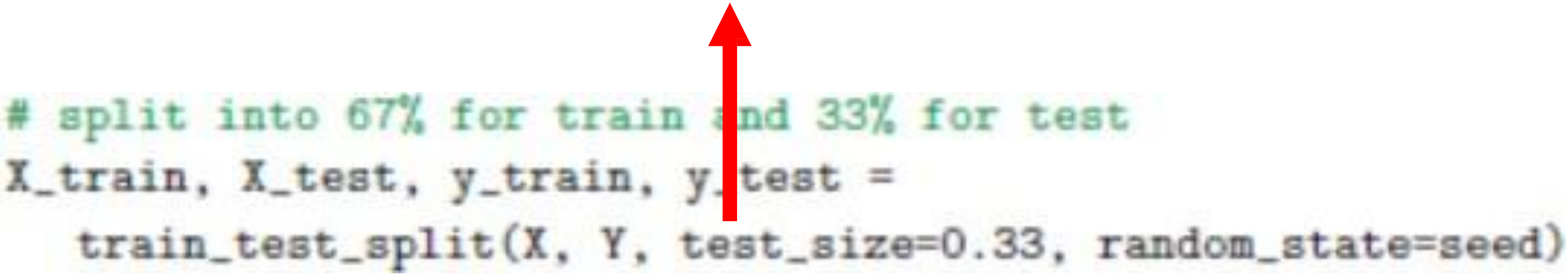


```
...
Epoch 145/150
514/514 [==============================] - 0s - loss: 0.5252 - acc: 0.7335 - val_loss:
    0.5489 - val_acc: 0.7244
Epoch 146/150
514/514 [==============================] - 0s - loss: 0.5198 - acc: 0.7296 - val_loss:
    0.5918 - val_acc: 0.7244
Epoch 147/150
514/514 [==============================] - 0s - loss: 0.5175 - acc: 0.7335 - val_loss:
    0.5365 - val_acc: 0.7441
Epoch 148/150
514/514 [==============================] - 0s - loss: 0.5219 - acc: 0.7354 - val_loss:
    0.5414 - val_acc: 0.7520
```

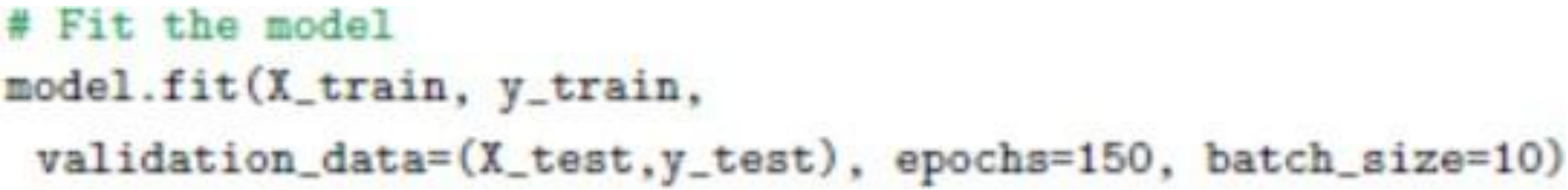
DEVELOP ANN WITH KERAS



Manual Validation Dataset – percentage

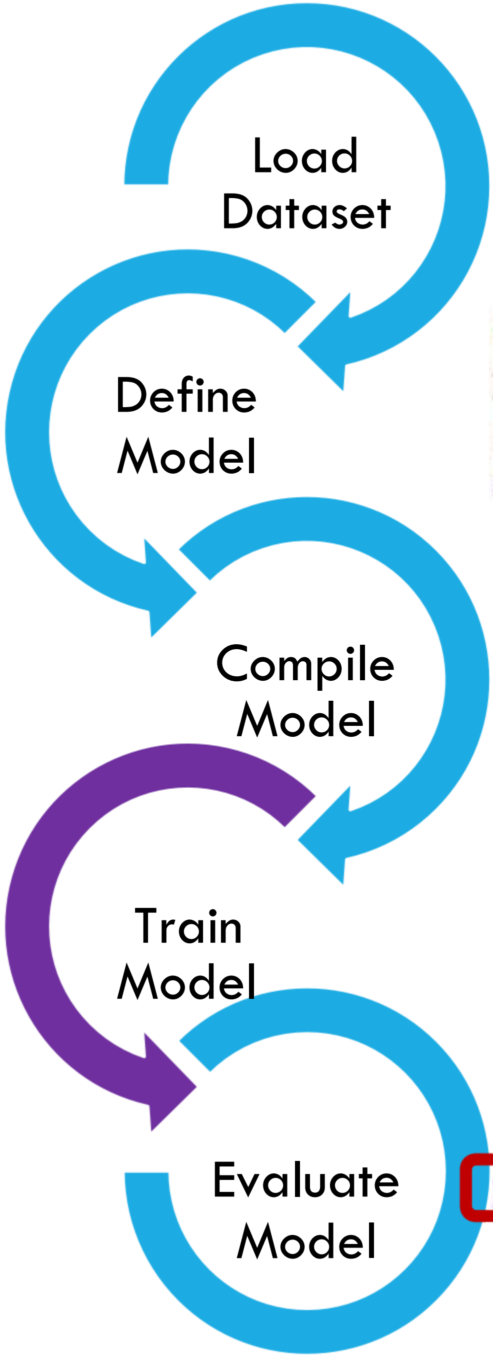


```
# split into 67% for train and 33% for test
X_train, X_test, y_train, y_test =
   train_test_split(X, Y, test_size=0.33, random_state=seed)
```



```
# Fit the model
model.fit(X_train, y_train,
 validation_data=(X_test,y_test), epochs=150, batch_size=10)
```

DEVELOP ANN WITH KERAS



Load Dataset

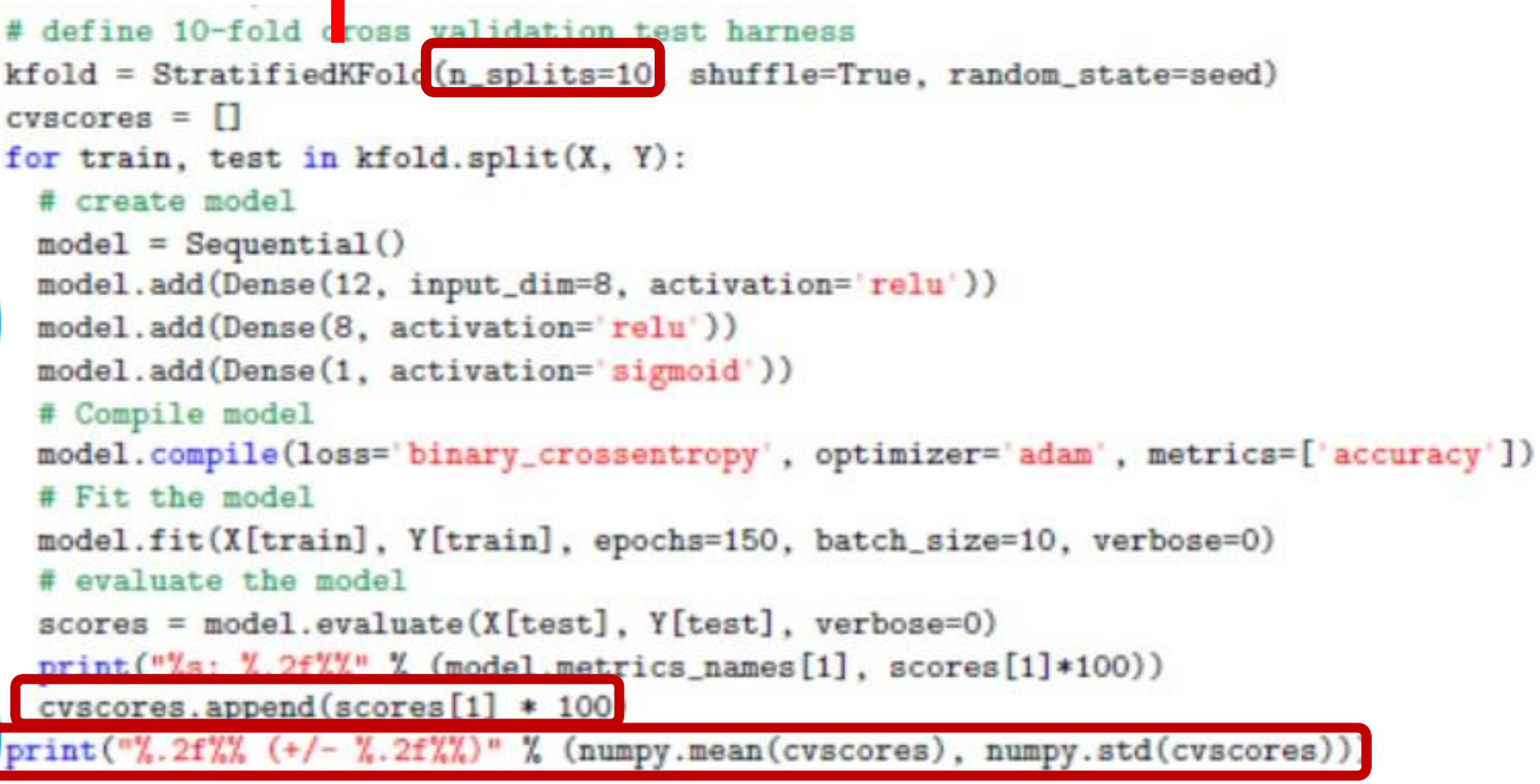
Define Model

Compile Model

Train Model

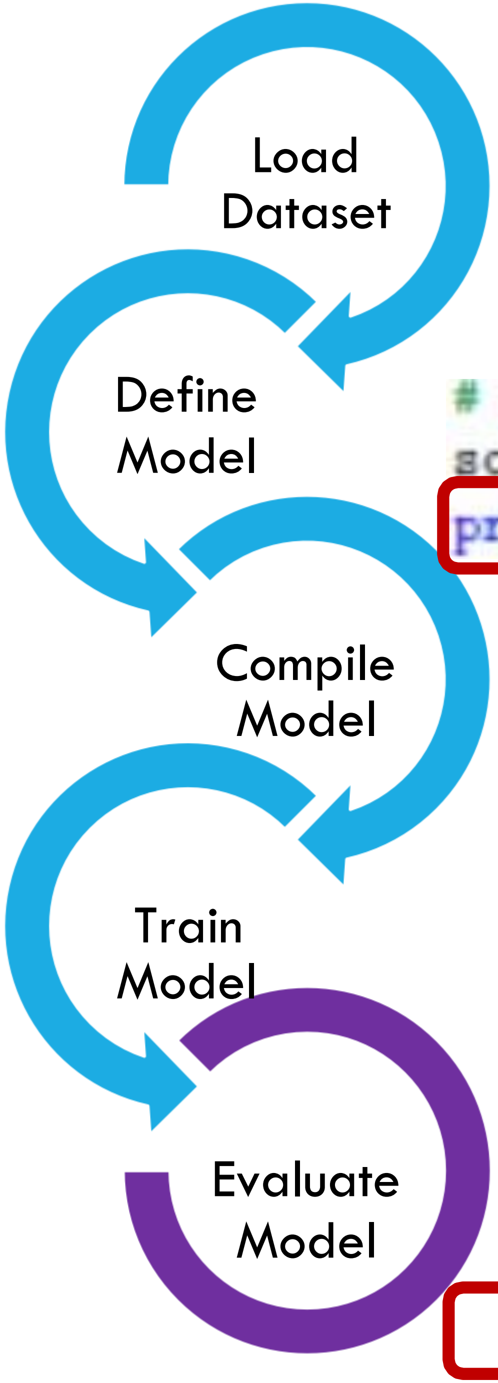
Evaluate Model

***K*-fold cross validation**



```
# define 10-fold cross validation test harness
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
cvscores = []
for train, test in kfold.split(X, Y):
  # create model
  model = Sequential()
  model.add(Dense(12, input_dim=8, activation='relu'))
  model.add(Dense(8, activation='relu'))
  model.add(Dense(1, activation='sigmoid'))
  # Compile model
  model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
  # Fit the model
  model.fit(X[train], Y[train], epochs=150, batch_size=10, verbose=0)
  # evaluate the model
  scores = model.evaluate(X[test], Y[test], verbose=0)
  print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
  cvscores.append(scores[1] * 100)
print("%.2f%% (+/- %.2f%%)" % (numpy.mean(cvscores), numpy.std(cvscores)))
```

DEVELOP ANN WITH KERAS


Load Dataset

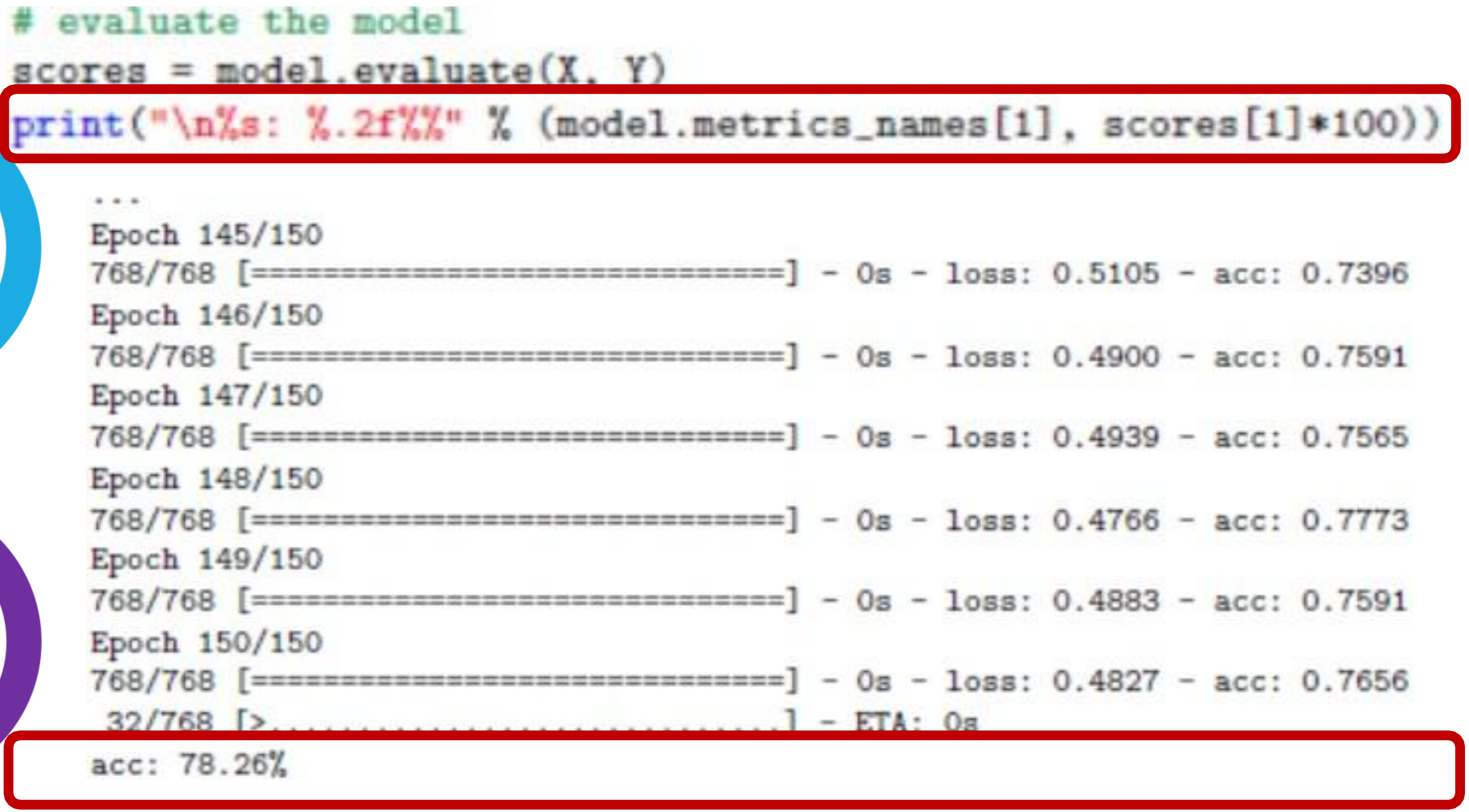
Define Model

Compile Model

Train Model

Evaluate Model

With the same dataset as training, the evaluation will give the training performance – generalization of model cannot be evaluated



```
# evaluate the model
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
...
Epoch 145/150
768/768 [==============================] - 0s - loss: 0.5105 - acc: 0.7396
Epoch 146/150
768/768 [==============================] - 0s - loss: 0.4900 - acc: 0.7591
Epoch 147/150
768/768 [==============================] - 0s - loss: 0.4939 - acc: 0.7565
Epoch 148/150
768/768 [==============================] - 0s - loss: 0.4766 - acc: 0.7773
Epoch 149/150
768/768 [==============================] - 0s - loss: 0.4883 - acc: 0.7591
Epoch 150/150
768/768 [==============================] - 0s - loss: 0.4827 - acc: 0.7656
 32/768 [>.............................] - ETA: 0s
acc: 78.26%
```

MNIST DATASET

http://yann.lecun.com/exdb/mnist/

MNIST has 60,000 images in its training set and 10,000 in its test set.

MNIST derives from NIST, and stands for “Mixed National Institute of Standards and Technology.”

Each image in the MNIST database is a 28x28 pixel cell, and each cell is contained within a bounding box

EXERCISE

Practice using IRIS Dataset

Iris setosa

Iris virginica

Iris versicolor

Iris Mythica