Implementing Multilayer Neural Network by Keras

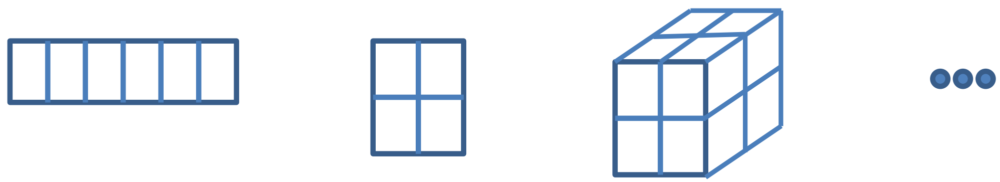
Kietikul Jearanaitanakij

Department of Computer Engineering, KMITL

Keras

- What is Tensor ?

  Tensor is the multidimensional (1d, 2d, 3d,..., Nd) data array.



- Keras: The Python Deep Learning library

  A high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK(Microsoft Cognitive Toolkit), or Theano.

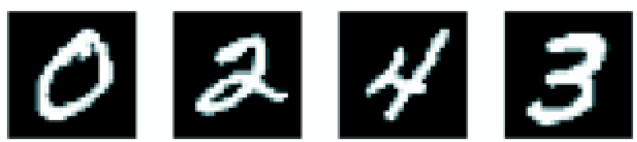
- You can run Keras on PC and Google Colab

Example 1

Content are taken from “Deep Learning with Python”, Francois Chollet

Classifying MNIST dataset by Multilayer NN

- A classic dataset in the machine-learning community.
- Grayscale images of handwritten digits (28 × 28 pixels) of 10 categories (0 through 9).



- The MNIST dataset comes preloaded in Keras, in the form of a set of four Numpy arrays.

Loading MNIST dataset

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Let’s look at the training data:

```
>>> train_images.shape
(60000, 28, 28)

>>> len(train_labels)
60000

>>> train_labels
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

And here’s the test data:

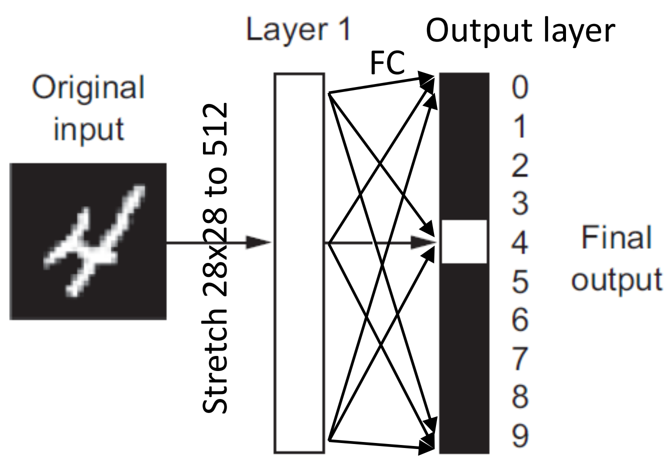
```
>>> test_images.shape
(10000, 28, 28)

>>> len(test_labels)
10000

>>> test_labels
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

Build Multilayer NN

```
from keras import models
from keras import layers
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```



https://colab.research.google.com/drive/1j_ZIamoQCKL4YBX_7TGENXxkTRQ2_L2I

Network Compilation

```
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Preparing the image data

```
#train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
#test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

Preparing the image labels

```
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Train the network

```
>>> network.fit(train_images, train_labels, epochs=5, batch_size=128)
Epoch 1/5
60000/60000 [==============================] - 9s - loss: 0.2524 - acc: 0.9273
Epoch 2/5
51328/60000 [========================>.....] - ETA: 1s - loss: 0.1035 - acc: 0.9692
```

We quickly reach an accuracy of 0.989 (98.9%) on the training data.
Now let’s check that the model performs well on the test set, too:

```
>>> test_loss, test_acc = network.evaluate(test_images, test_labels)
>>> print('test_acc:', test_acc)
test_acc: 0.9785
```

AUC - ROC Curve

- AUC - ROC curve is a performance measurement for classification problem at various thresholds settings.
- ROC is a probability curve.
- AUC represents degree or measure of separability.
- It tells how much model is capable of distinguishing between classes.
- Higher the AUC, better the model is at predicting True as True and False as False.
- Roc is defined in terms of true positive and false positive. Therefore, it is a good idea to understand the confusion matrix first.

Confusion Matrix

Reference: ttps://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/

- A confusion matrix is a table that is often used to describe the performance of a classification model.
- Let’s start with confusion matrix of a binary classifier.

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

- We are predicting the presence of a disease, for example, "yes" means they have the disease, and "no" means they don't have the disease.
- The classifier made a total of 165 predictions.
- Out of those 165 cases, the classifier predicted "yes" 110 times, and "no" 55 times.
- In reality, 105 patients in the sample have the disease, and 60 patients do not.

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Let's now define the most basic terms

- **true positives (TP) :** We predicted yes (they have the disease), and they do have the disease.
- **true negatives (TN) :** We predicted no, and they don't have the disease.
- **false positives (FP):** We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
- **false negatives (FN):** We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

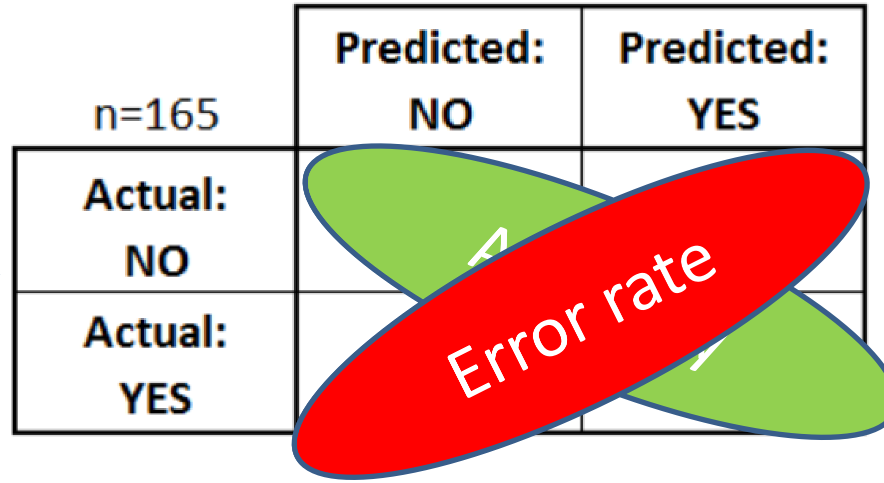
n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

n=165	Predicted: NO	Predicted: YES
Actual: NO	TN rate 50/60=0.83	FP rate 10/60=0.17
Actual: YES	FN rate 5/105=0.05	TP rate 100/105=0.95

Rates that are often computed from a confusion matrix.

- **true positives rate (TP rate) or "Sensitivity" or "Recall":** When it's actually yes, how often does it predict yes?
- **true negatives rate (TN rate) or "Specificity":** When it's actually no, how often does it predict no? Equivalent to 1 – FP rate.
- **false positives rate (FP rate):** When it's actually no, how often does it predict yes?
- **false negatives rate (FN rate):** When it's actually yes, how often does it predict no? Equivalent to 1 – TP rate.

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	



Rates that are often computed from a confusion matrix.

- **Accuracy:** Overall, how often is the classifier correct?

  (TP+TN)/total = (100+50)/165 = 0.91

- **Error (Misclassification) rate : Overall, how often is it wrong?**

  (FP+FN)/total = (10+5)/165 = 0.09

  Equivalent to 1 – Accuracy

- **Precision: When it predicts yes, how often is it correct?**

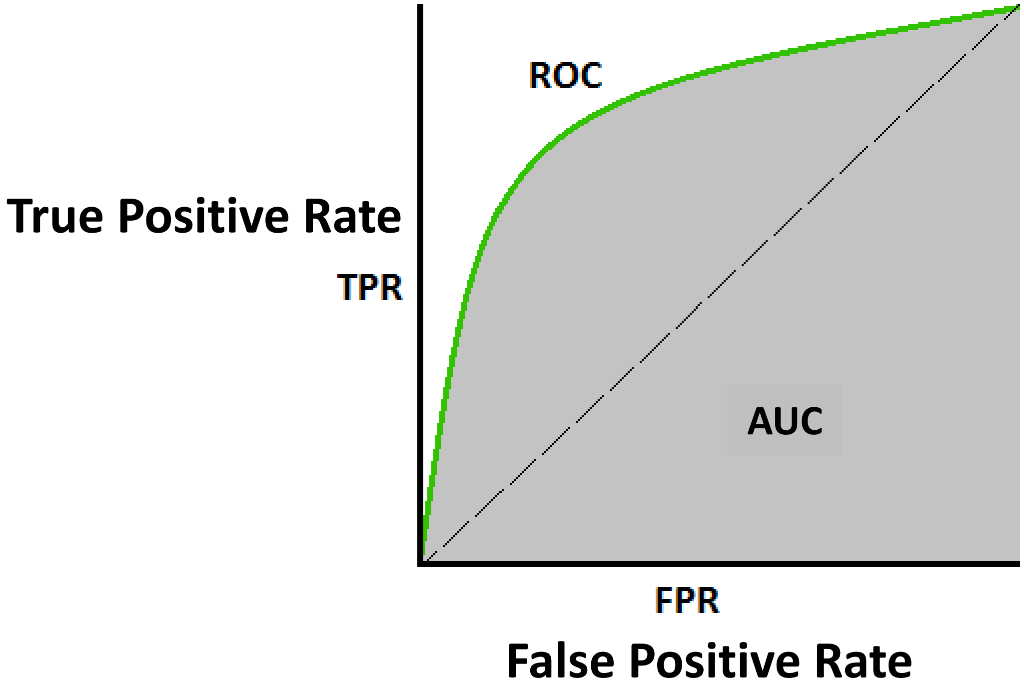
  TP/(TP+FP)= 100/110 = 0.91

- **Recall: Among all actual yes, how often does it predict yes?**

  TP/(TP+FN) = 100/105 = 0.95

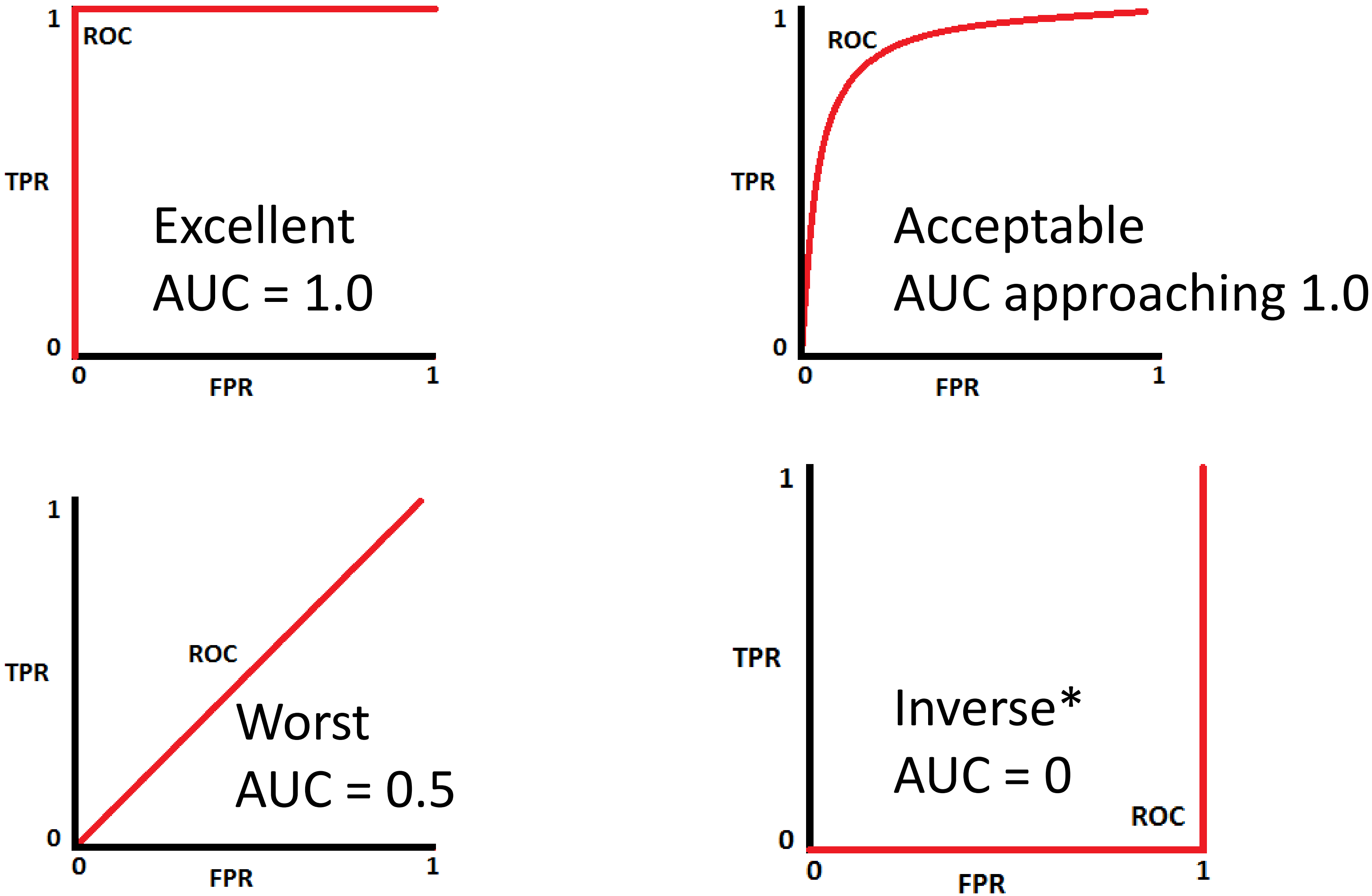
AUC - ROC Curve (continue)

- The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.



- An excellent model has AUC near to the 1 which means it has good measure of separability.
- A poor model has AUC near to the 0 which means it has worst measure of separability.
- And when AUC is 0.5, it means model has no class (useless) separation capacity.

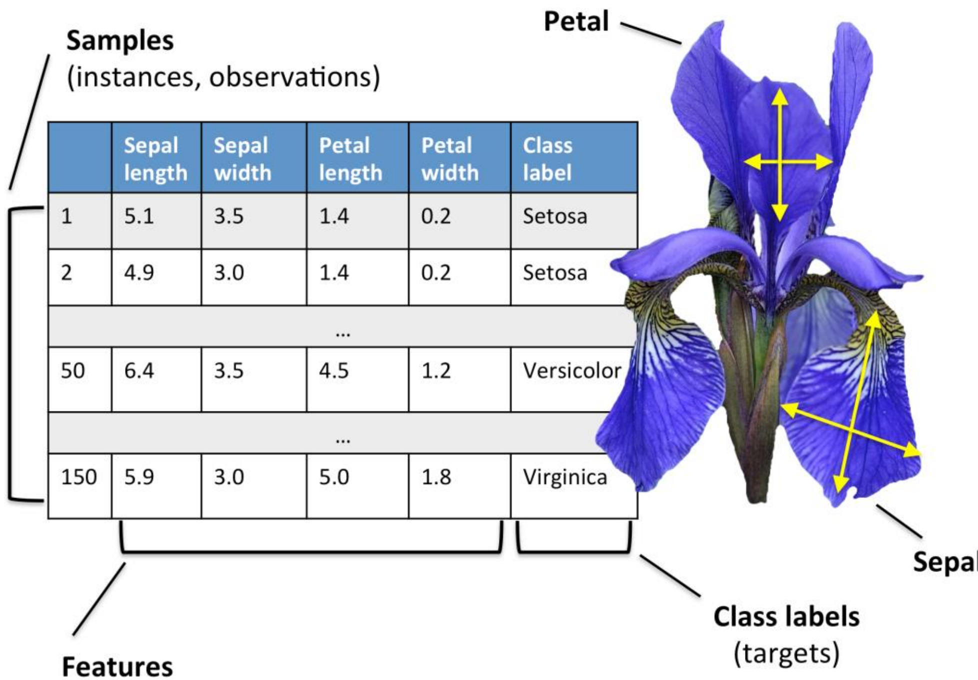
How to interpret AUC - ROC Curve



* Model is predicting negative class as a positive class and vice versa.

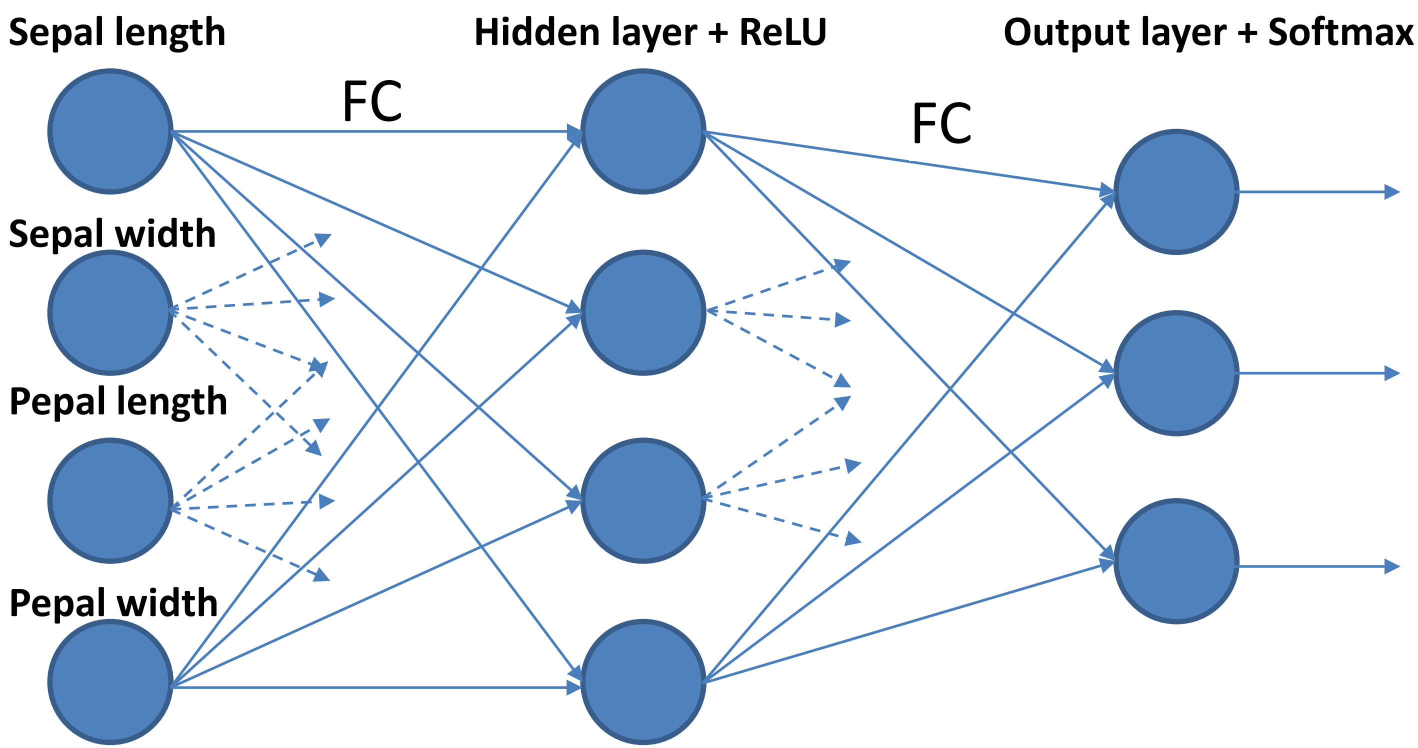
Example 2

- We will implement a neural network with two hidden layer for a classification problem on the iris dataset.
- Iris dataset is arguably the most classic dataset used in machine learning.
- It is a dataset that measures sepal length, sepal width, petal length, and petal width of three different types of iris flowers: Iris setosa, Iris virginica, and Iris versicolor.
- There are 150 measurements overall, 50 measurements of each species.



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Two-layer NN Architecture for IRIS



https://colab.research.google.com/drive/13hNT-C8sbBRLvP10sNK6atBrb451oCM

Example 3

Tic-Tac-Toe Endgame Data Set (from UCI repository)

- This database encodes the complete set of possible board configurations at the end of tic-tac-toe games, where "x" is assumed to have played first.
- The target concept is "win for x" (i.e., true when "x" has one of 8 possible ways to create a "three-in-a-row").
- Number of Instances: 958 (legal tic-tac-toe endgame boards).
- Number of Attributes: 9, each corresponding to one tic-tac-toe square

- Attribute Information: (x=player x has taken, o=player o has taken, b=blank).
- Example:

x x x x o o x o o positive

This pattern represents the following configuration.

x	x	x
x	o	o
x	o	o

x wins the game (positive)

- Another example

x o o x o x b o x negative

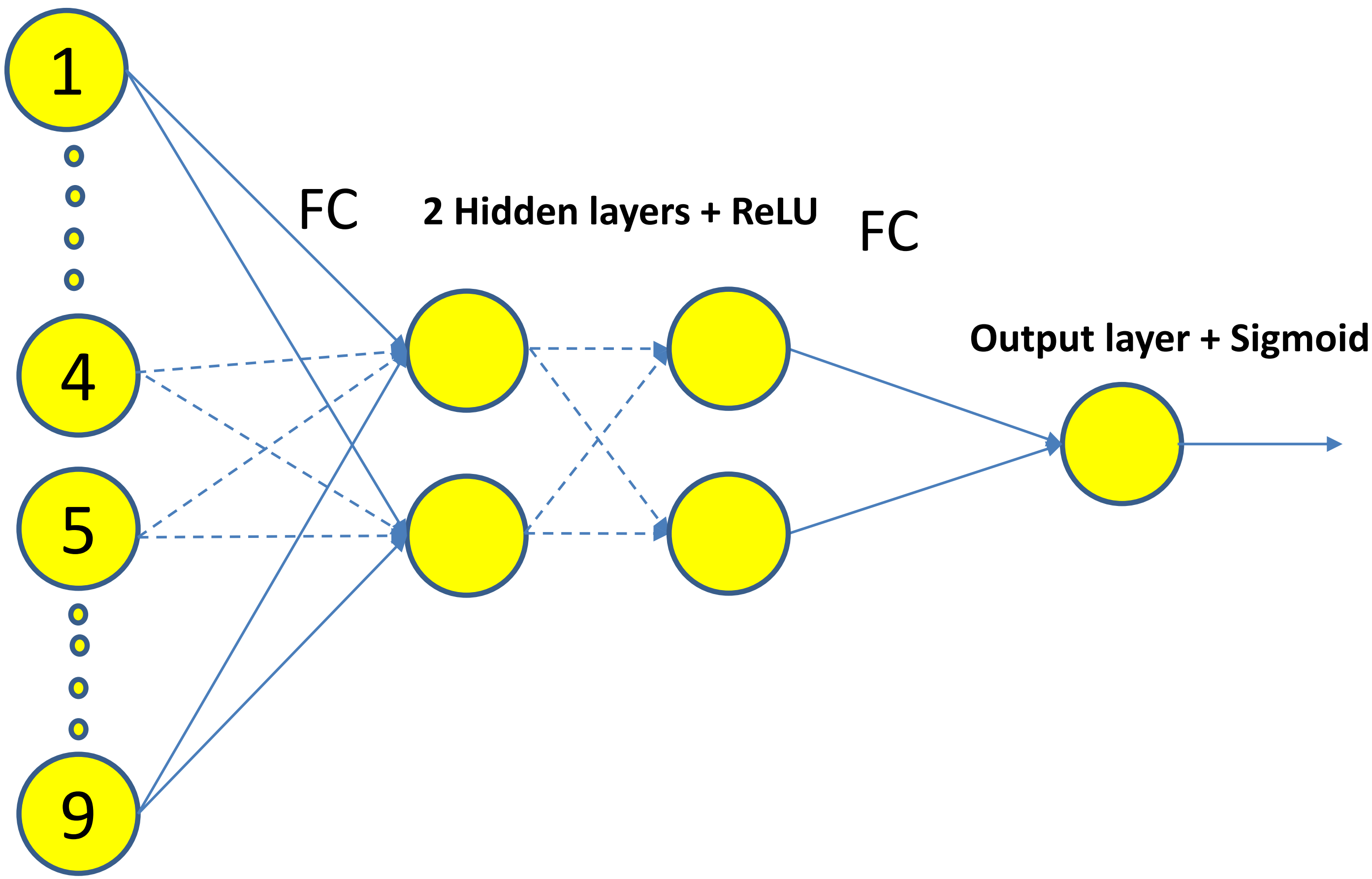
This pattern represents the following configuration.

x	o	o
x	o	x
b	o	x

o wins the game (negative)

- Class Distribution: 65.3% positive, 34.7% negative.

Three-layer NN Architecture for Tic-Tac-Toe



https://colab.research.google.com/drive/1iJyt7Tz54MqJHNJD_34of3wBKZdzT8zS

Assignment 1

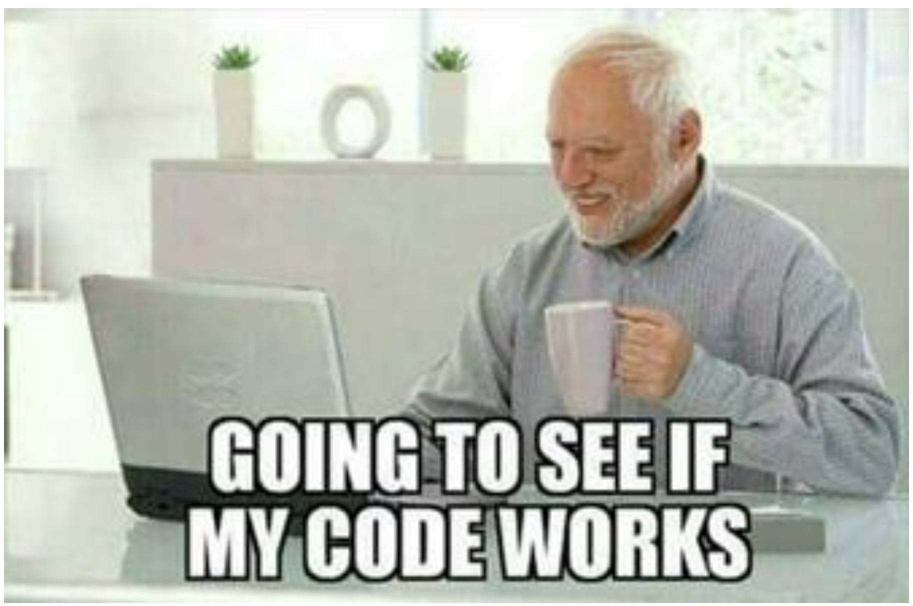
(Due date: 2 weeks from now)

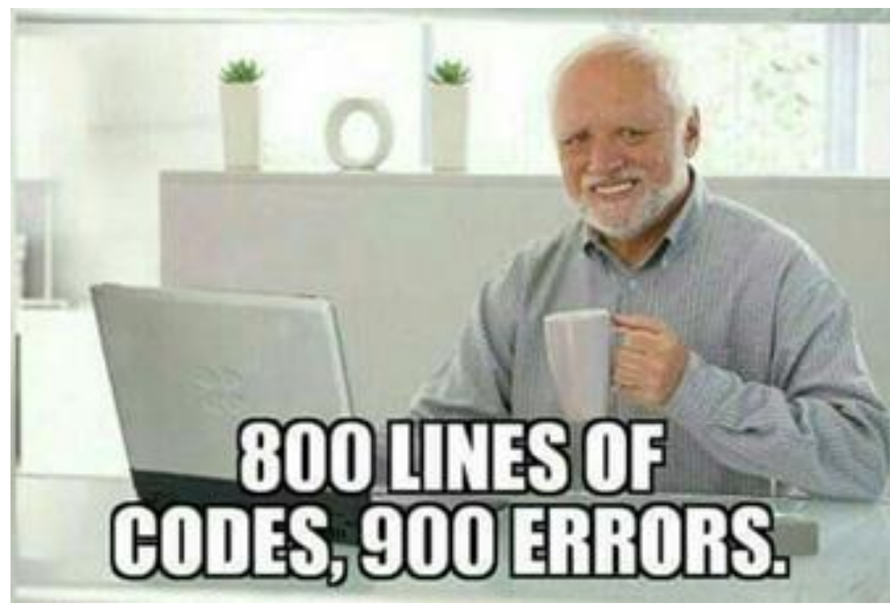
- Download **Tic-Tac-Toe** source code in python and the csv dataset.
- Modify the source code so that it has the best test accuracy. You can adjust the number of parameters , e.g., training epochs, hidden neurons, hidden layers, or change the learning rate, batch size, data preprocessing, etc. But DO NOT modify the dataset, test_size, random_state and stratify.
- Adjust hyperparameters in your network so that it achieves the highest test accuracy rate. Your score depends on the test accuracy.
- Things to turn in :
  - Demo your code. All members must show up and declare their responsibilities, including answer the questions.
  - Sketch your modified model architecture. Also list important settings that you've made to the original code.
  - Two graphs of training accuracy and training loss.
  - Average test accuracy rate, including min and max, over 10 runs.

Warning: Cheating will result in zero score.

Another warning: Start coding as soon as possible !

- You may spend so much time for searching the best setting of the network.
- Due to the limit of demo time, the training time for each run must be less than 3 minutes.
- Submit a list of members in your group TODAY.





- Next class
  - Loss function