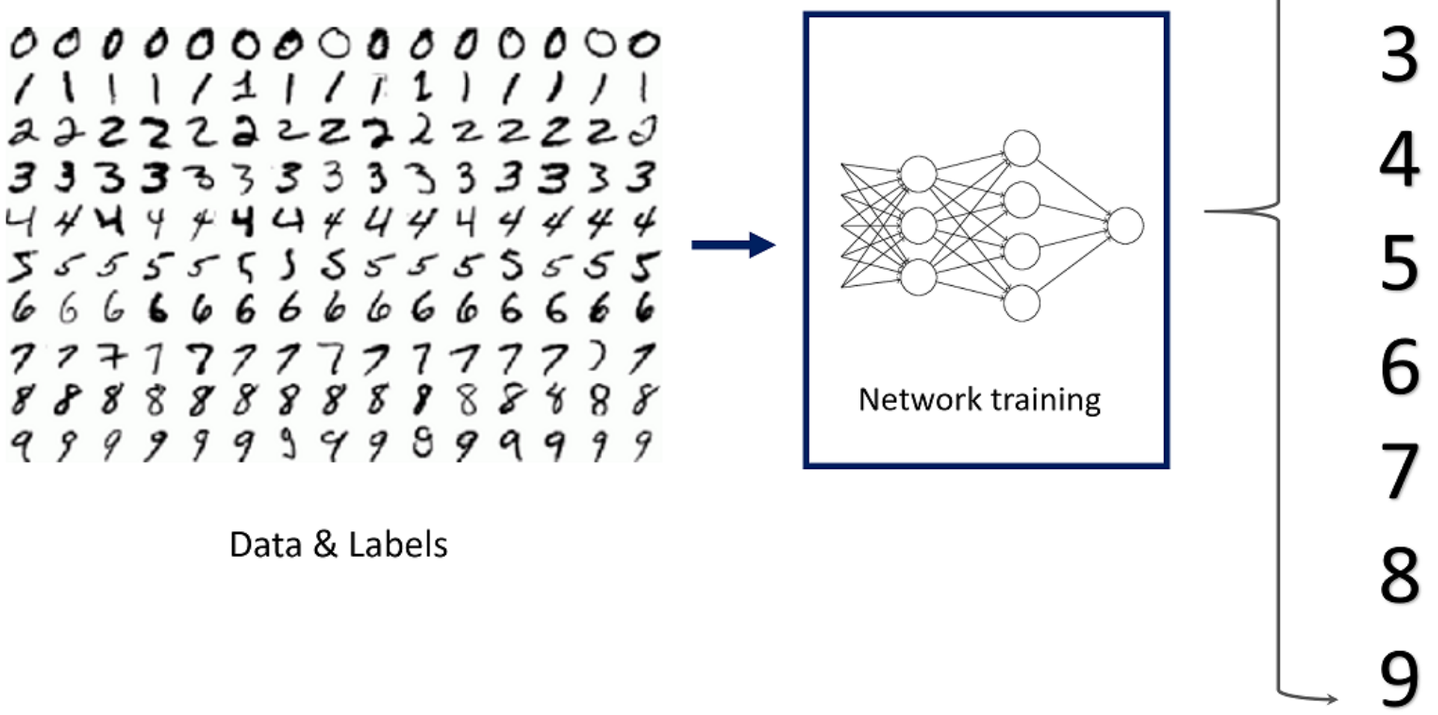
연세대학교
YONSEI UNIVERSITY

Classifying MNIST digits using MLP

TA. Hwanmoo Yong

Machine **L**earning &
Control **S**ystems Lab.

MNIST



Modified National Institute of Standards and Technology database

- 60,000 images for the training and 10,000 images for the test.
- Size : 28x28 pixels
- Only 1 channel (Gray scale)

Preparing Data

연세대학교 YONSEI UNIVERSITY

1. Load

- Use easy loading function in Keras library
- Need to create custom loading function when trying to load your own dataset.

2. Reshape

- Change size/shape of the dataset array

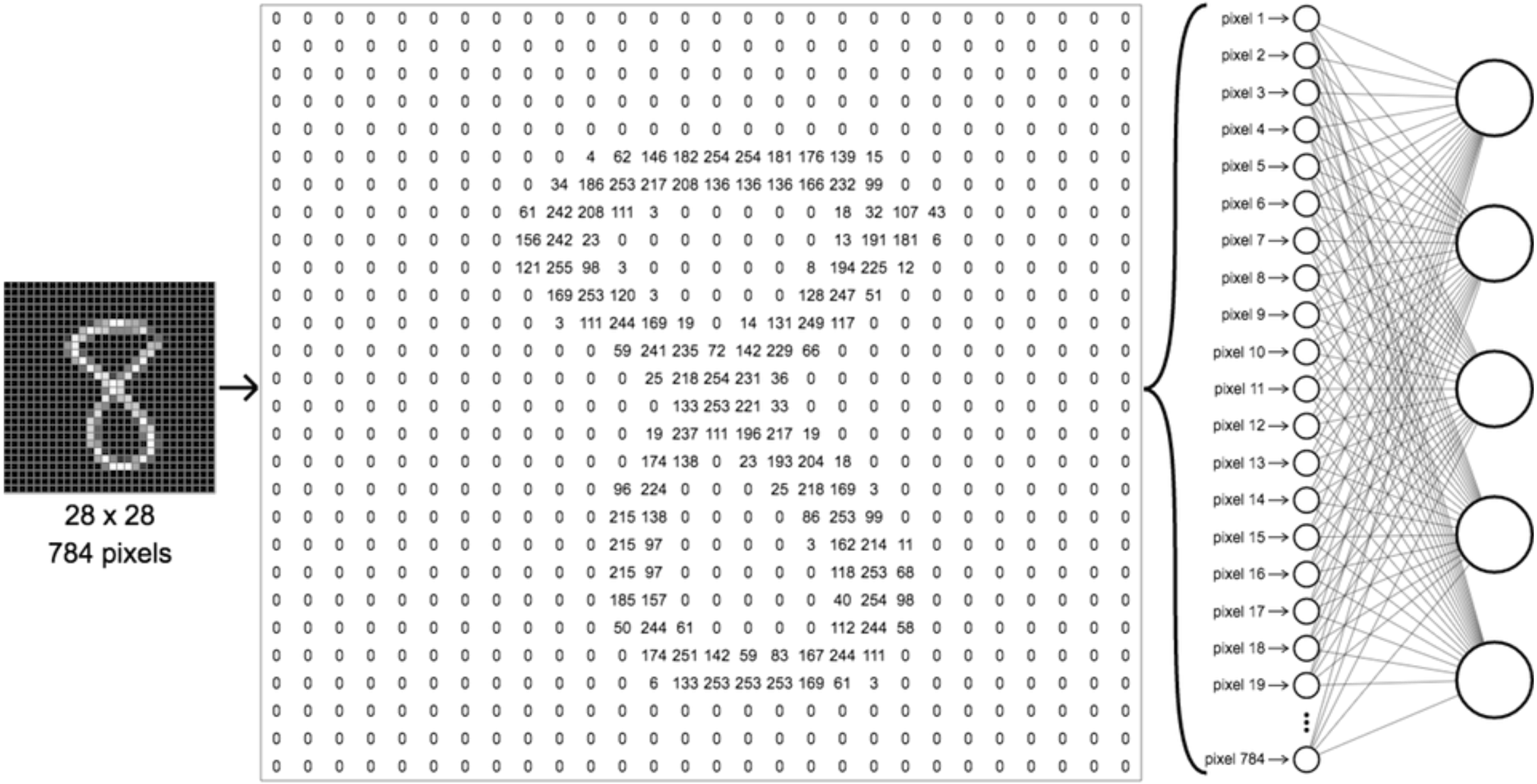
3. Normalize

- Standardize features by removing the mean and scaling to unit variance
- Divide by the maximum value

4. Convert to one-hot encoding/vector

- Use keras.utils.to_categorical function
- Need when training a neural network which classifies given data

Reshape

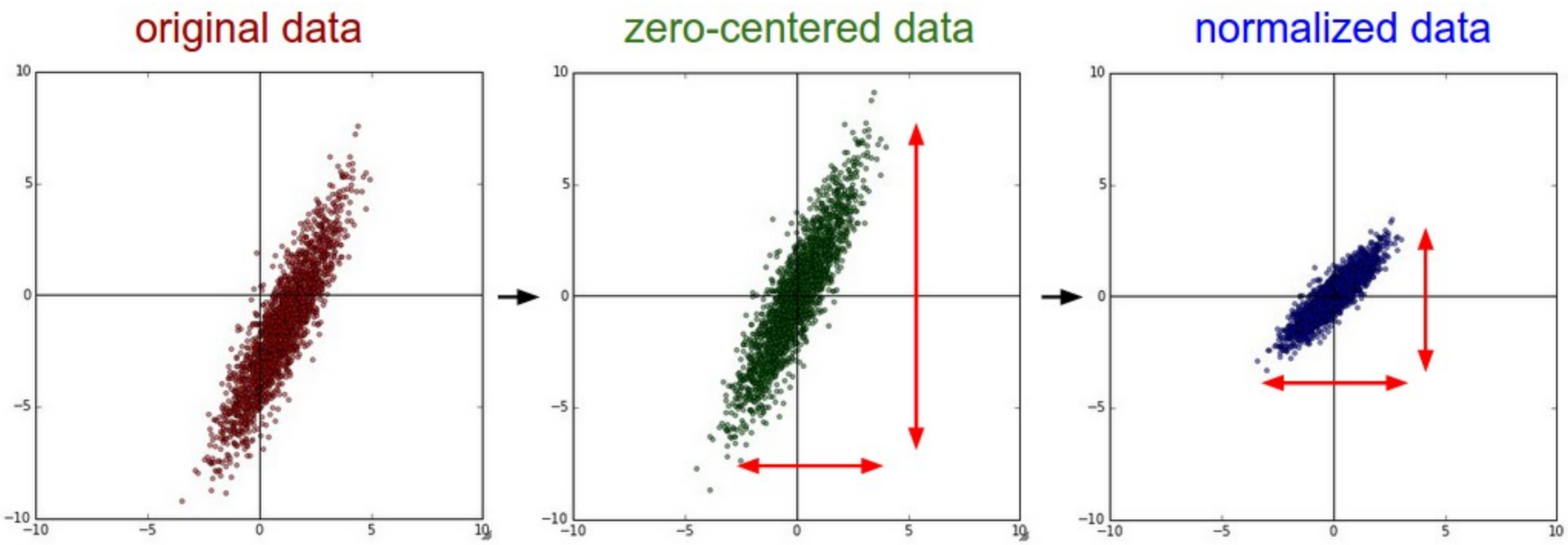


- Reshape 28x28 2-d vector to 1-d vector (size 764)

```
13  x_train = x_train.reshape(60000, 784)
14  x_test = x_test.reshape(10000, 784)
```

Normalize

- required when features have different ranges.

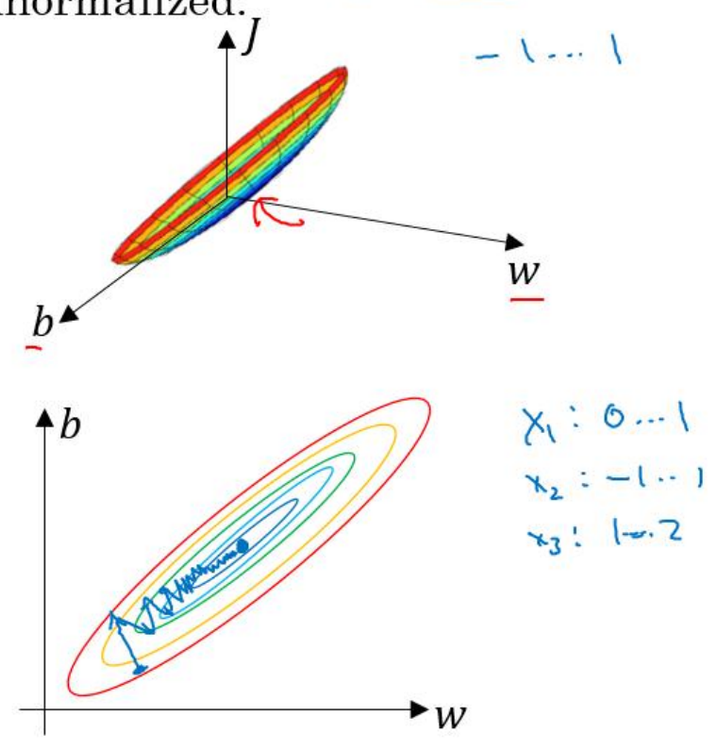


Common data preprocessing pipeline. Left: Original toy, 2-dimensional input data. Middle: The data is zero-centered by subtracting the mean in each dimension. The data cloud is now centered around the origin. Right: Each dimension is additionally **scaled by its standard deviation**. The red lines indicate the extent of the data - they are of unequal length in the middle, but of equal length on the right.

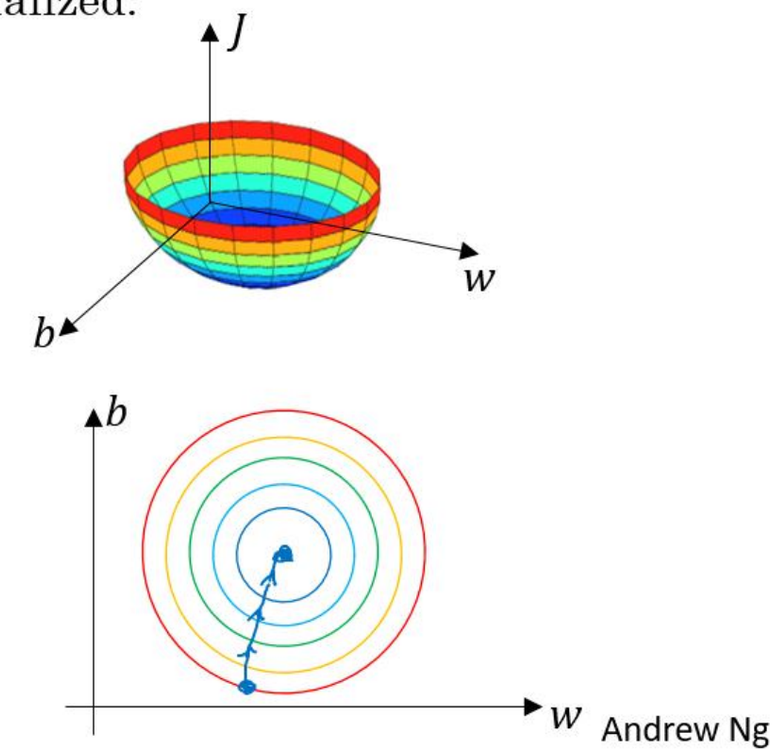
Why normalize inputs?

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right)$$

Unnormalized:



Normalized:



Andrew Ng

```
15  x_train = x_train.astype('float32')
16  x_test = x_test.astype('float32')
17  x_train /= 255
18  x_test /= 255
```

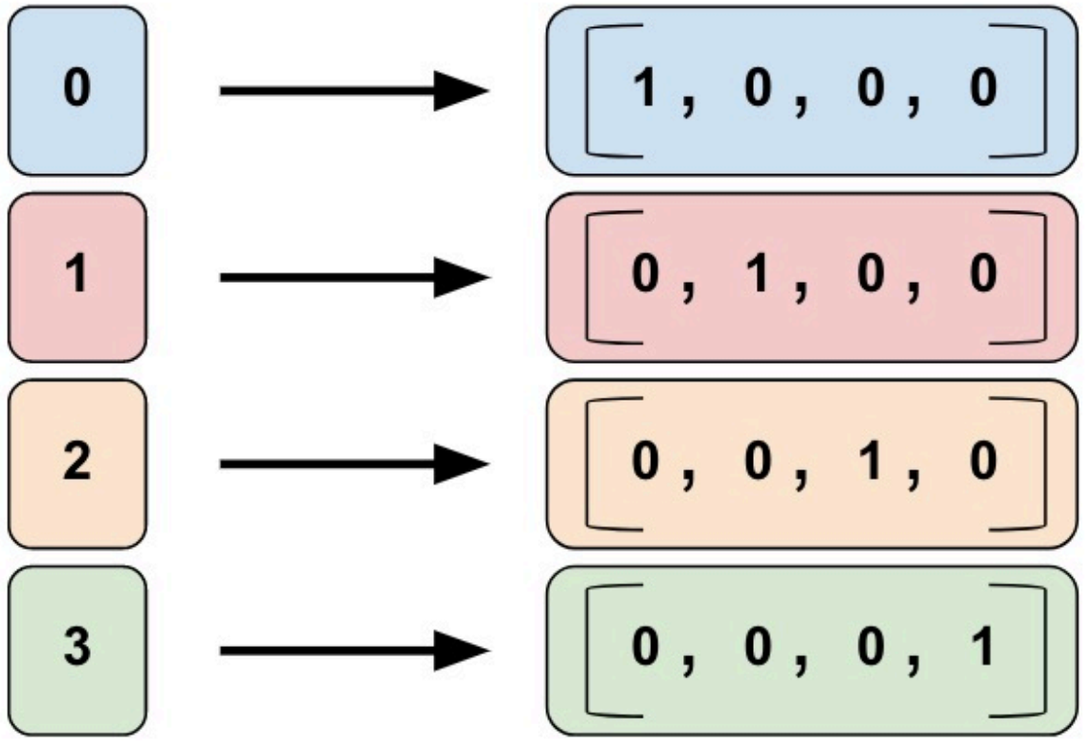
```
#Select numerical columns which needs to be normalized


train_norm = x_train[x_train.columns[0:10]]
test_norm = x_test[x_test.columns[0:10]]


# Normalize Training Data


std_scale = preprocessing.StandardScaler().fit(train_norm)
x_train_norm = std_scale.transform(train_norm)
```

Convert to one-hot encoding/vector




```
22  # convert class vectors to binary class matrices
23  y_train = keras.utils.to_categorical(y_train, num_classes)
24  y_test = keras.utils.to_categorical(y_test, num_classes)
```

Creating a model

연세대학교 YONSEI UNIVERSITY

1. Sequential (keras)

- Easy way of creating a network.
- Convenient, but hard to use when creating complex networks.

2. Dense Layer

- Fully connected layer.
- Consists of weights and biases

3. Activation Functions

- Defines the output of the node
- ReLU, tanh, sigmoid

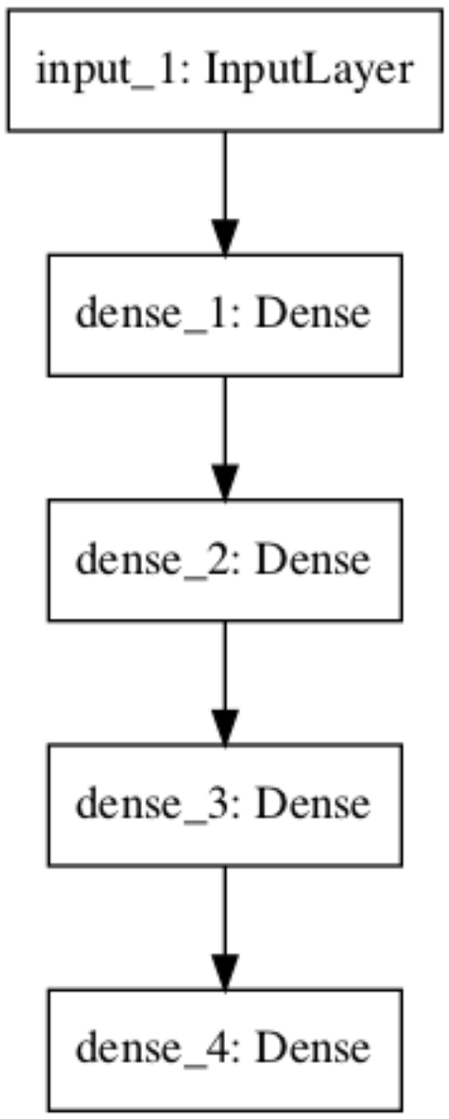
4. Dropout Layer

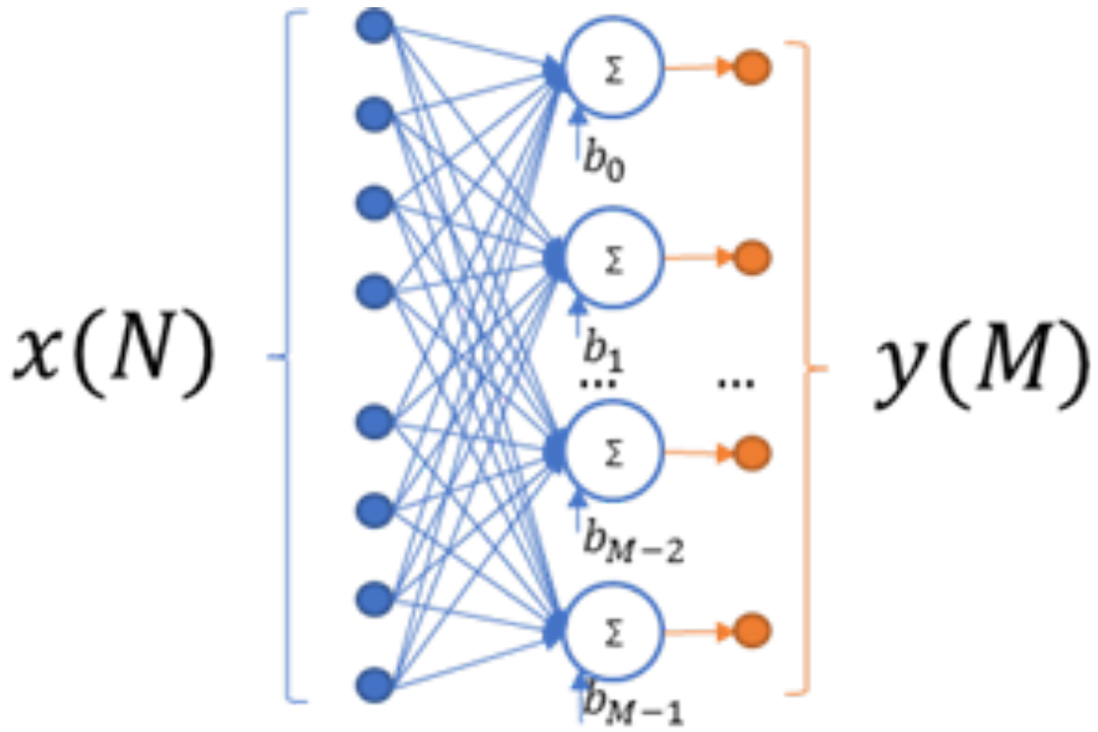
- Prevents overfitting
- Randomly removes the connections between weights

4. Softmax

- Converts the scores to the probabilities
- Usually locates at the end of the network

Dense Layer





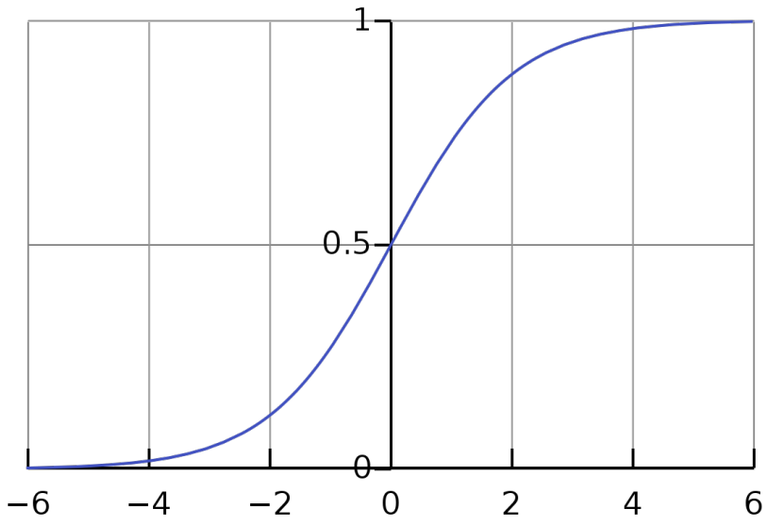
$$y_i = x_j W_{i,j} + b_i$$

Where:

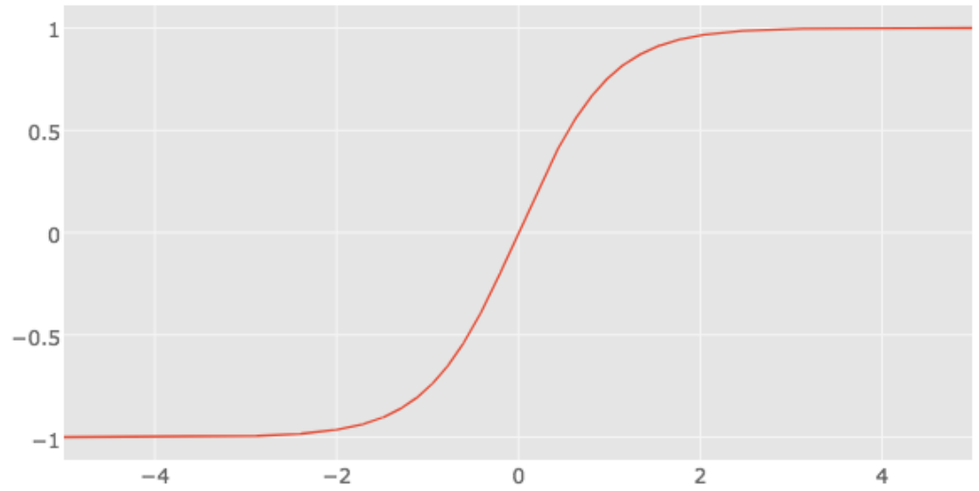
- x_j - j_{th} value in input tensor.
- y_i - i_{th} value in output tensor.
- W_{ij} - weight of j_{th} input element for i_{th} neuron.
- b_j - bias for i_{th} neuron.

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

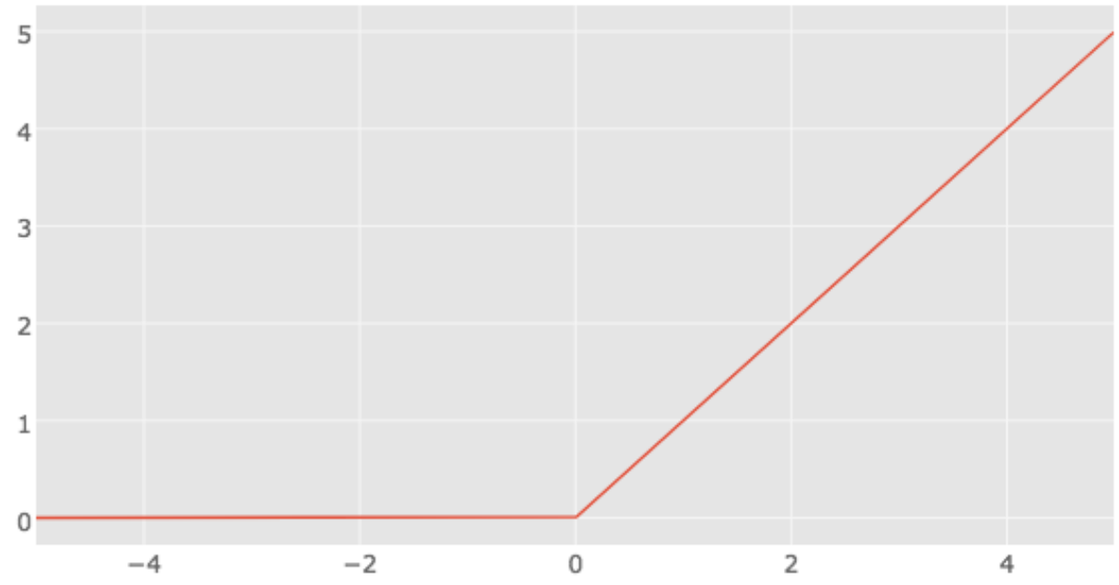
Activation Functions



Sigmoid



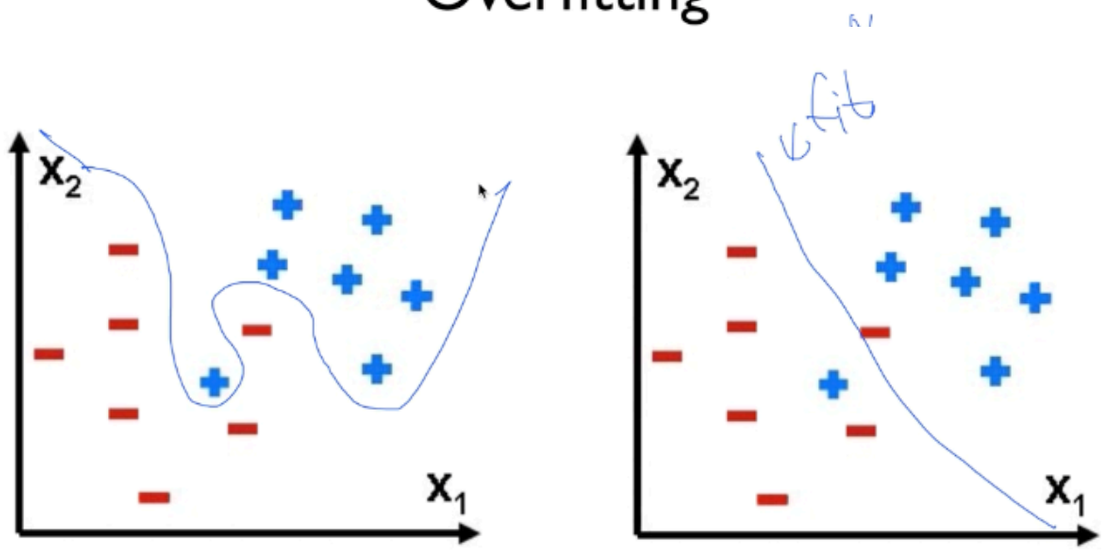
Tanh

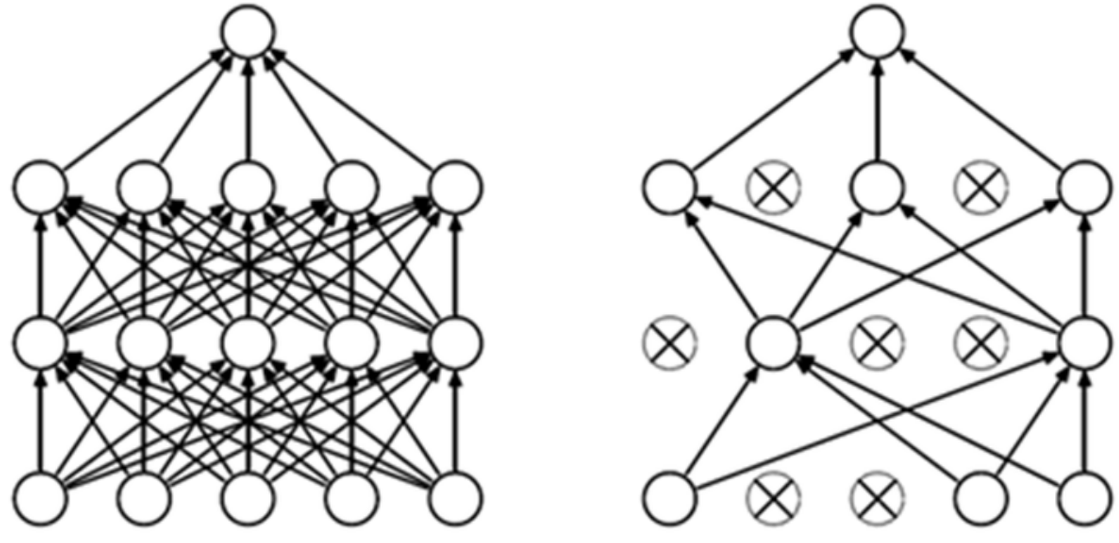


ReLU (Rectified Linear Unit)

- Defines the output strength of the signal to the following layer
- Select proper and better activation functions by trial and errors

Dropout Layer

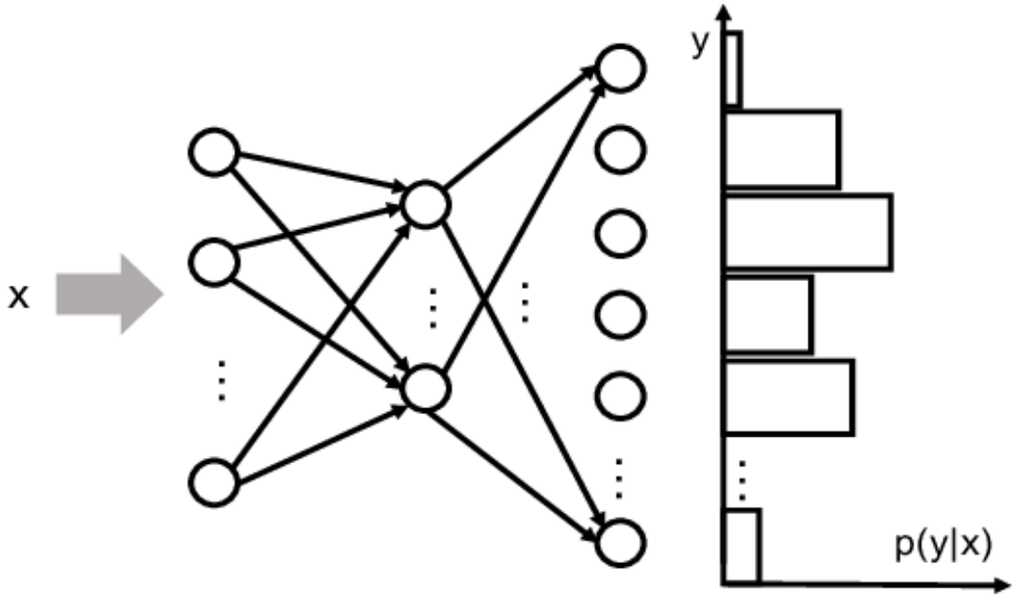



(a) Standard Neural Net

(b) After applying dropout.

- Prevents overfitting when the network has many hidden layers
- Skips / removes the connections between weights

Softmax



$$P(y=j \mid \Theta^{(i)}) = \frac{e^{\Theta^{(i)}}}{\sum_{j=0}^{k} e^{\Theta_k^{(i)}}}$$

Softmax function

where $\Theta = w_0x_0 + w_1x_1 + \ldots + w_kx_k = \sum_{i=0}^{k} w_ix_i = w^Tx$

- Converts the scores to the probabilities
- We select the index of the largest probability as a classification result

Training a network

1. Compile

- Sets loss/optimizer
- Use ‘categorical_crossentropy’ in classification project.

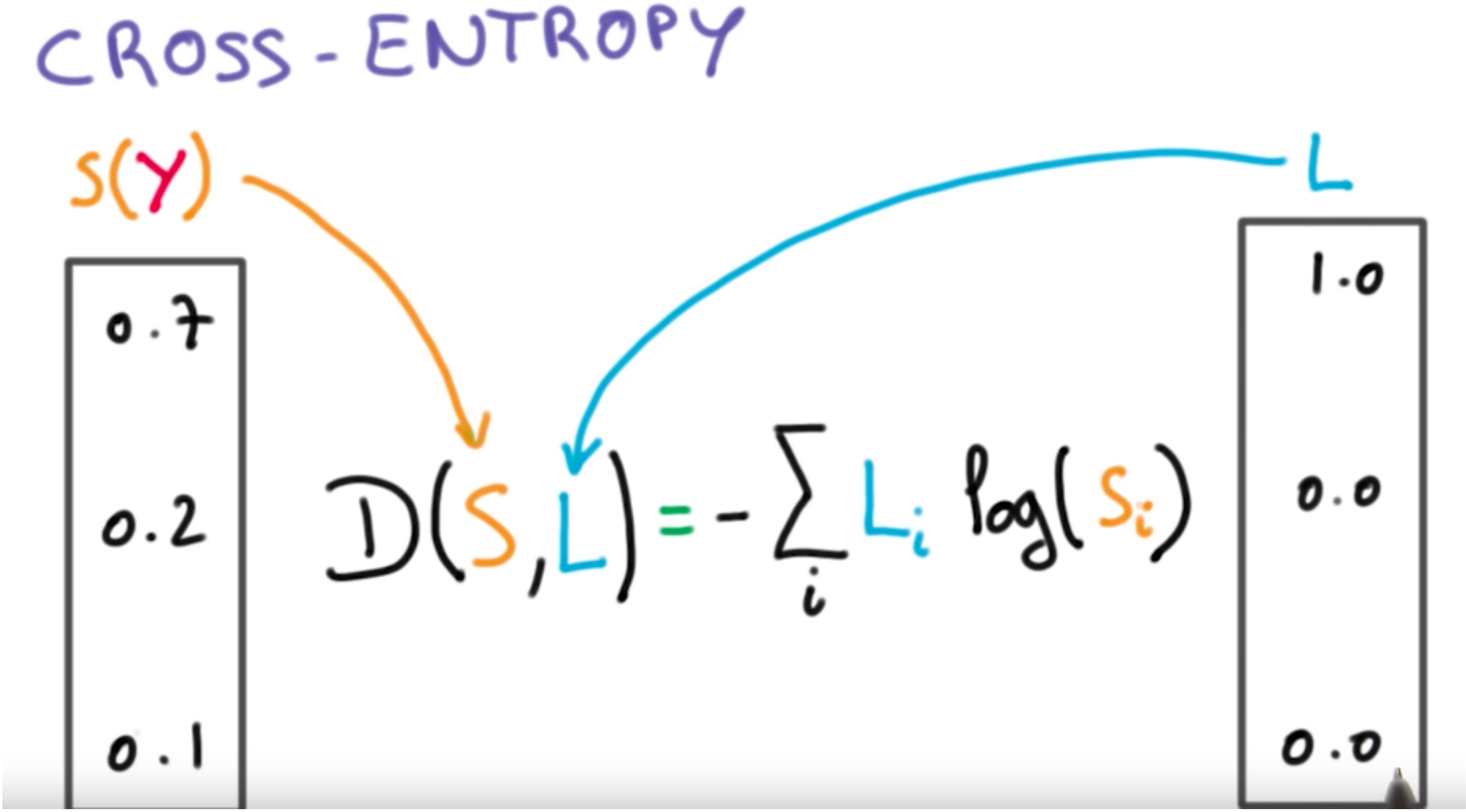
2. Fit

- Actual training part.
- Need to put train, validation data and other parameters for the training

3. Evaluate

- Gives the loss and the accuracy of the classification results with given dataset
- Use ‘predict’ instead when the actual classification result is needed.

Loss Function



- Calculate how the predictions of the network are different from the real values
- ie. Cross entropy, Mean squared error, Mean absolute error

Training a network

```
38  model.compile(loss='categorical_crossentropy',
39                optimizer=RMSprop(),
40                metrics=['accuracy'])
41
42  history = model.fit(x_train, y_train,
43                      batch_size=batch_size,
44                      epochs=epochs,
45                      verbose=1,
46                      validation_data=(x_test, y_test))
47  score = model.evaluate(x_test, y_test, verbose=0)
48  print('Test loss:', score[0])
49  print('Test accuracy:', score[1])
```