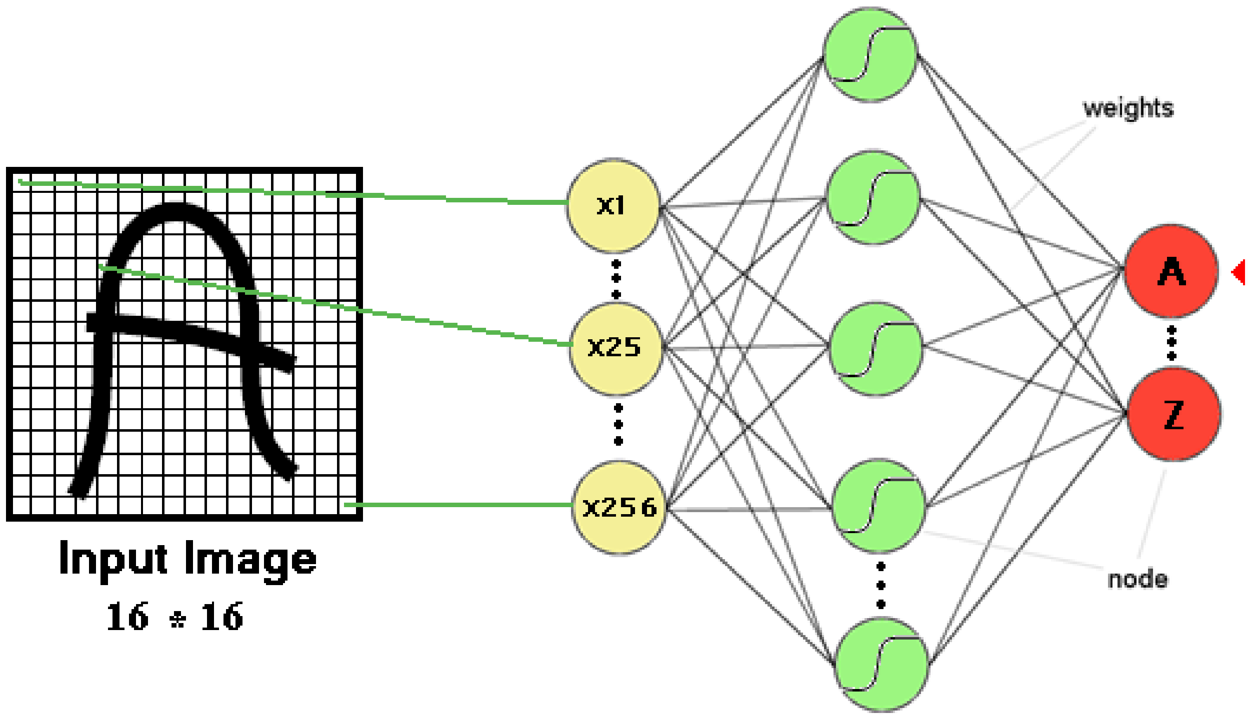
Neural Network

COEN140

Santa Clara University

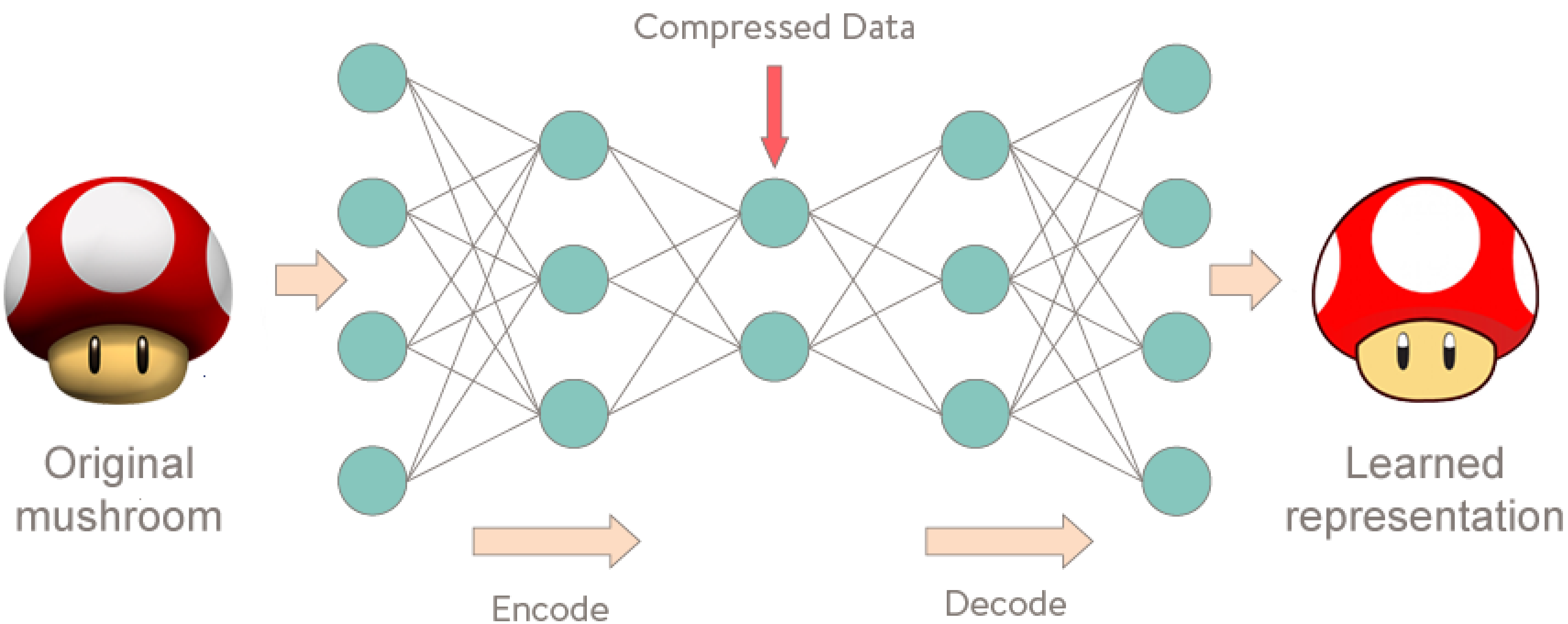
Neural Network

- Classification Tasks
  - Text classification
  - Image classification



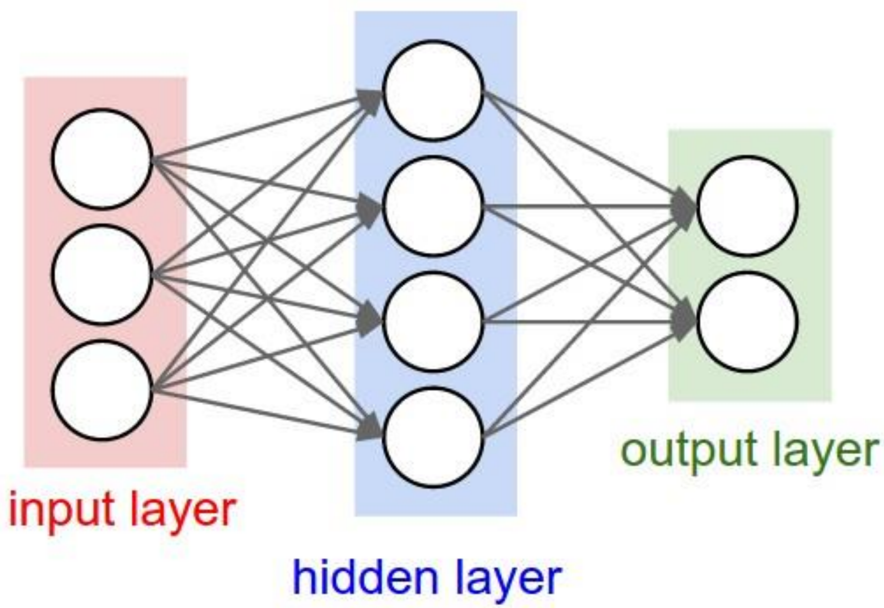
Neural Network

- Image compression (encoding) and decompression (decoding)



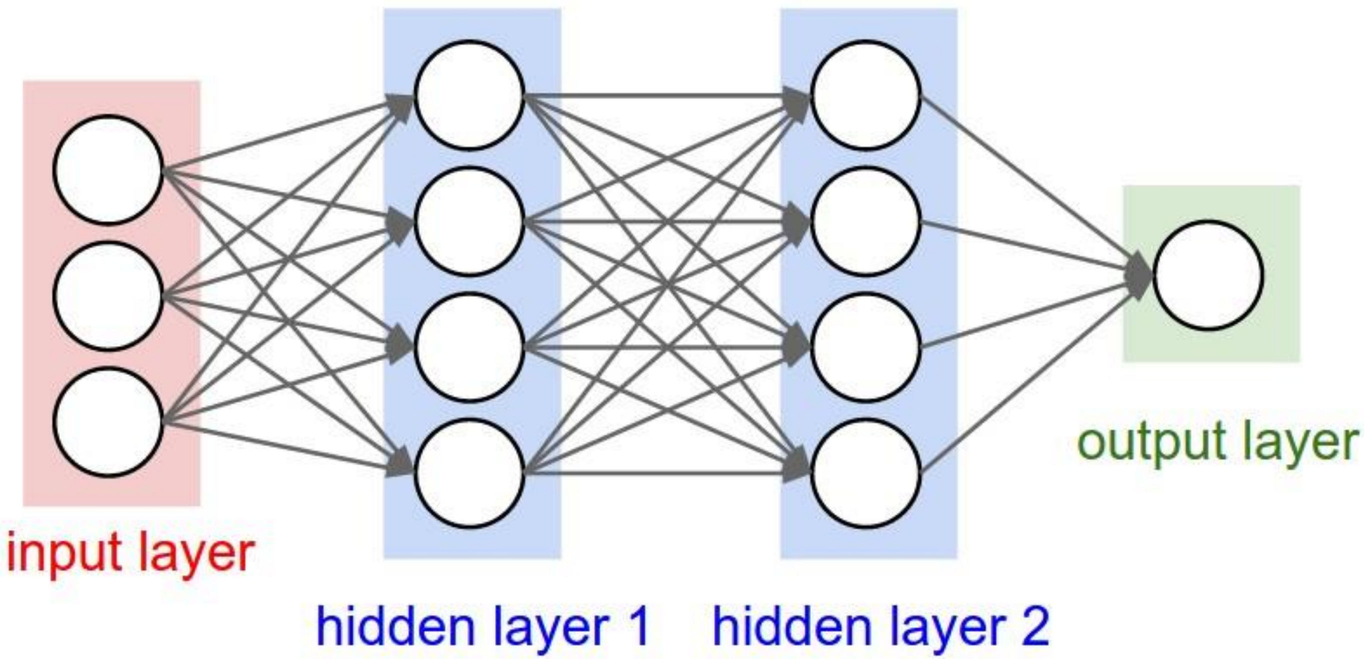
Neural Network

- A network with one hidden layer of four neurons



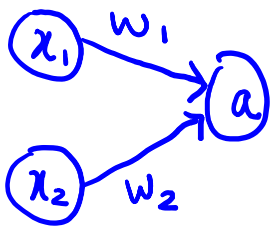
Neural Network

- A network with two layers of hidden units



Example: One-Layer Network

$\vec{x} = [x_1 \quad x_2]^T$



$y = h(a)$

$a = w_1 x_1 + w_2 x_2$

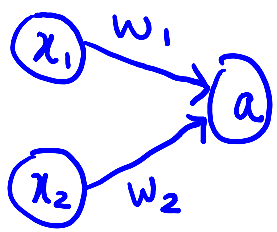
Let $y = \sigma(a) = \dfrac{1}{1 + e^{-a}} = p(C_1 \mid \vec{x})$

Question: How many parameters does the network have?

Answer: 2

Example: One-Layer Network

$\vec{x} = [x_1 \quad x_2]^T$



$y = h(a)$.

$a = w_1 x_1 + w_2 x_2$

Let $y = \sigma(a) = \dfrac{1}{1+e^{-a}} = p(C_1 | \vec{x})$

$x_1 = 2$, $x_2 = 4$

$w_1 = 0.5$, $w_2 = 0.2$

what is $P(C_1 | \vec{x})$?

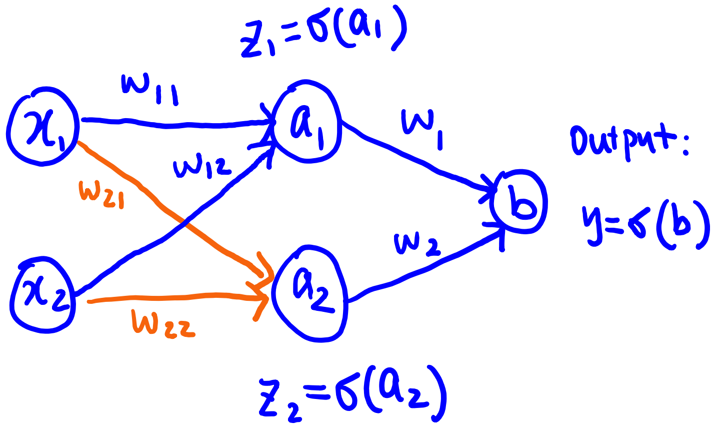
$a = 0.5 \times 2 + 0.2 \times 4 = 1 + 0.8 = 1.8$

$y = \sigma(1.8) = \dfrac{1}{1+e^{-1.8}} = 0.86$

$P(C_1 | \vec{x}) = 0.86$

Example: Two-Layer Network

$\vec{x} = [x_1 \quad x_2]^T$



$a_1 = w_{11}x_1 + w_{12}x_2$

$a_2 = w_{21}x_1 + w_{22}x_2$

$z_1 = \sigma(a_1), \quad z_2 = \sigma(a_2)$

$b = w_1 z_1 + w_2 z_2$

$y = \sigma(b)$

Question: How many parameters does the network have?

Answer: 6

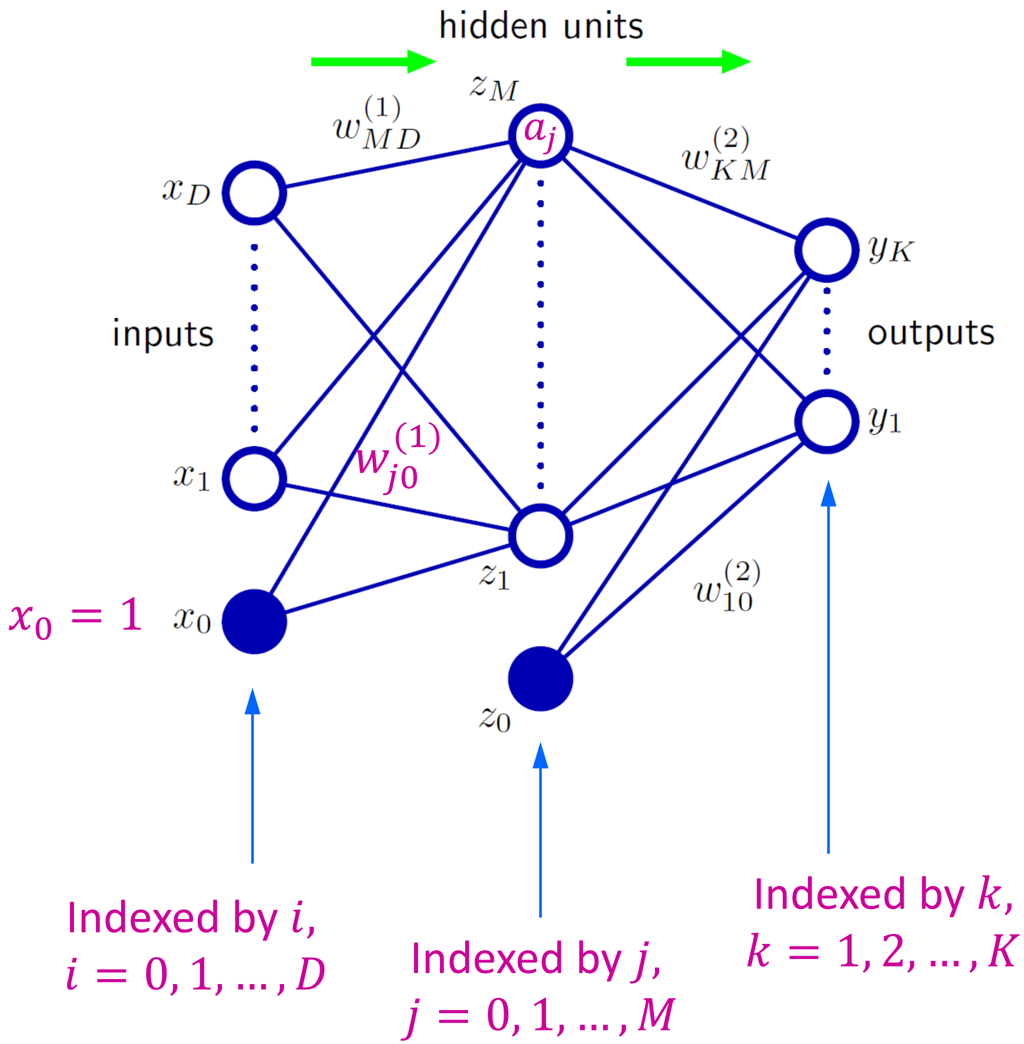
$\vec{w} = [w_{11} \ w_{12} \ w_{21} \ w_{22} \ w_1 \ w_2]^T$

Forward Pass

- What we have just seen is called: Forward Pass
  - Pass the input through the network, layer by layer
  - Obtain the network output

  - Need to know the “weights” on the links

Forward Pass



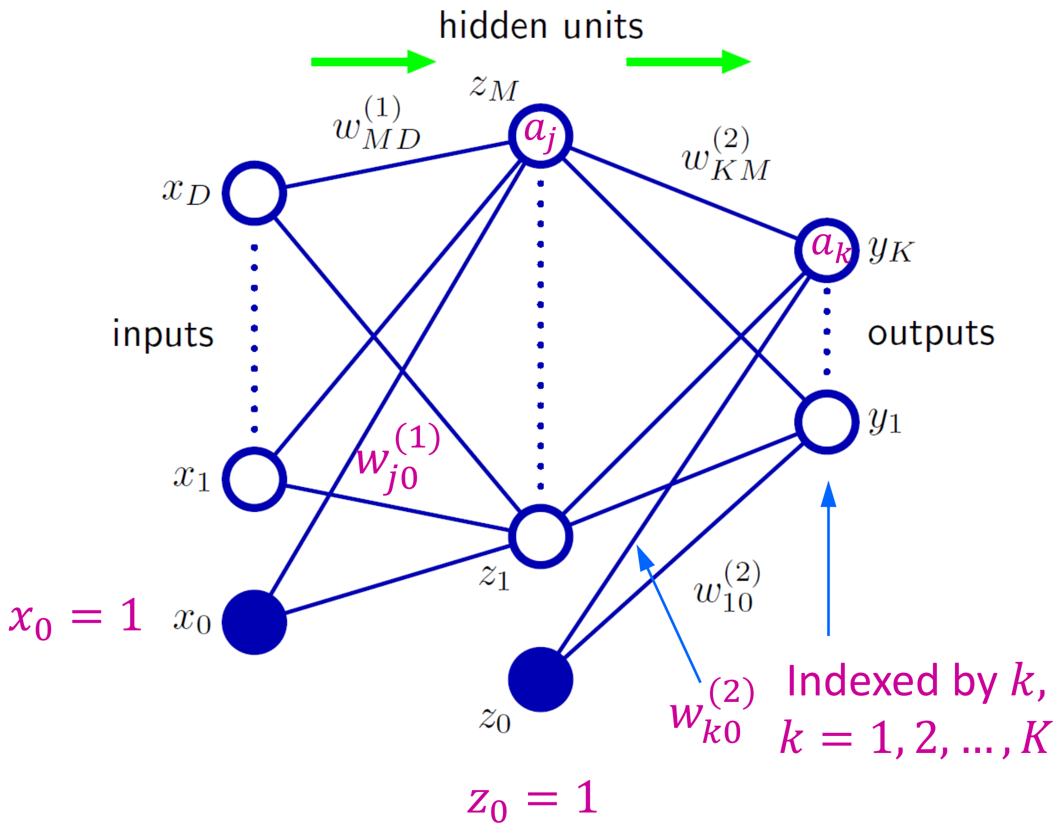
First Layer:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

The non-linear activation function: $h(\cdot)$

$$z_j = h(a_j)$$

Forward Pass



Second Layer:

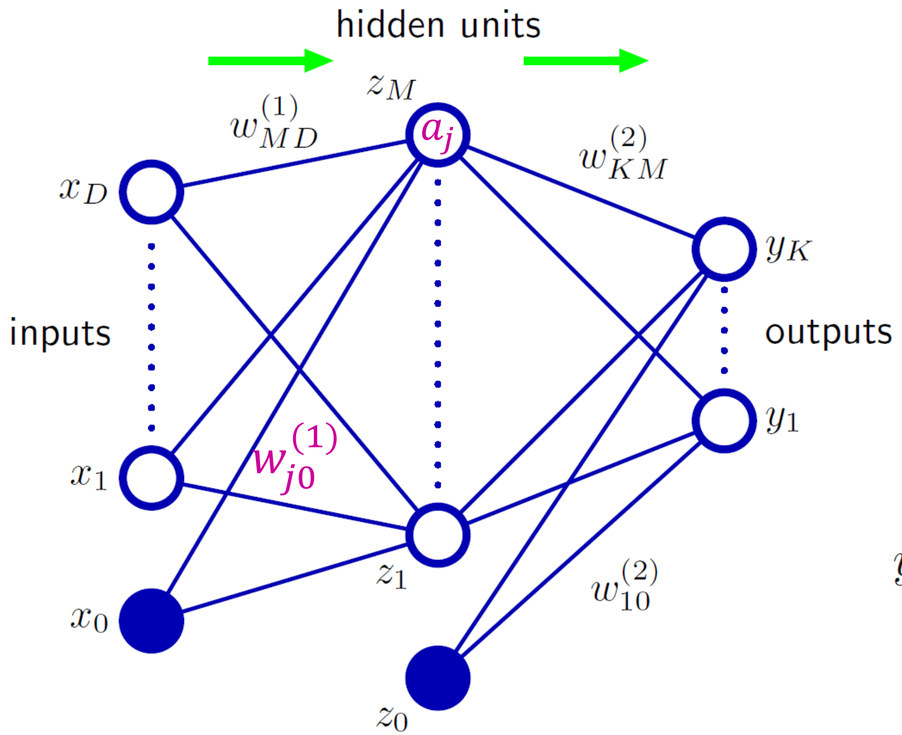
$$a_k = \sum_{j=1}^{M} w^{(2)}_{kj} z_j + w^{(2)}_{k0}$$

$$y_k = \sigma(a_k)$$

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

Can also be another type of activation function

Forward Pass



Include the bias in the summation:

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

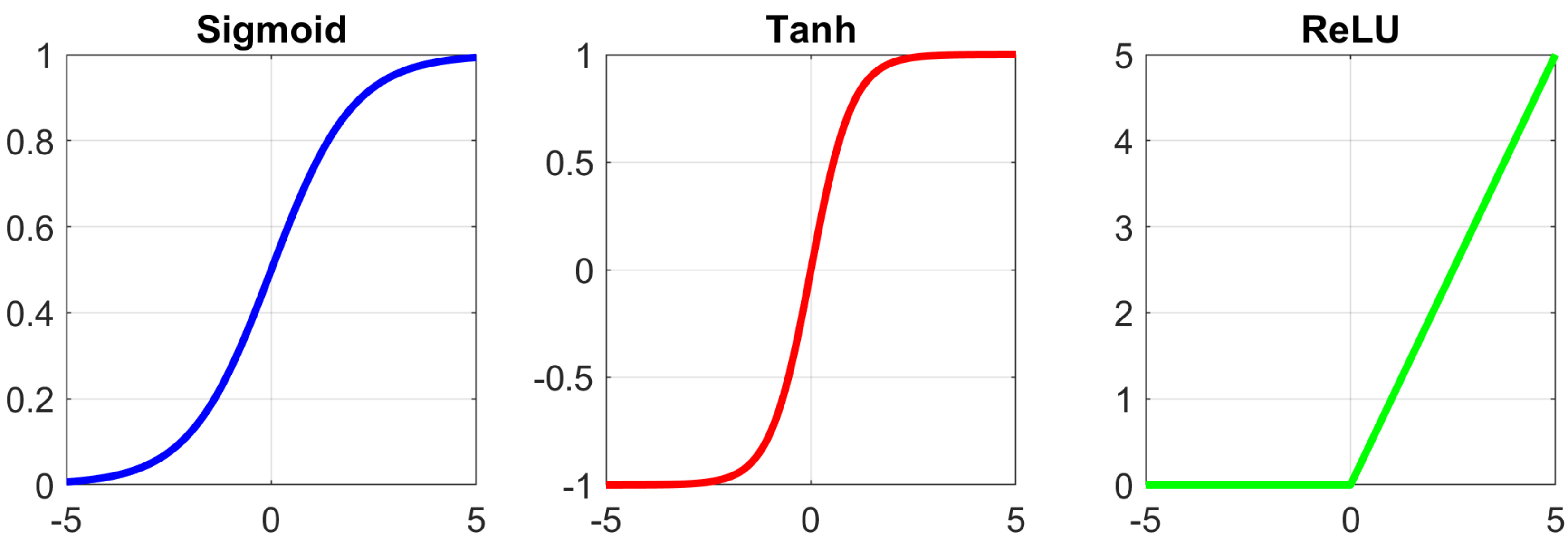
$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

The overall network function:

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

Activation Function: non-linear

- Sigmoid: $\sigma(a) = \frac{1}{1+\exp(-a)}$, represents a probability
- Tanh: $\tanh(a) = \frac{\exp(a)-\exp(-a)}{\exp(a)+\exp(-a)} = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$
- ReLU (Rectified Linear Unit): $\mathrm{ReLU}(a) = \max(0, a)$



Activation Function: non-linear

- Softmax function: $\frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)}$
- Outputs the probability of the data sample belonging to the k-th class, $k = 1, 2, \dots, K$

- $\sum_{k=1}^{K} \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)}$

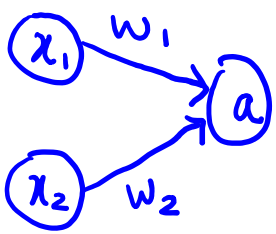
$$= \frac{\exp(a_1)}{\sum_{j=1}^{K} \exp(a_j)} + \frac{\exp(a_2)}{\sum_{j=1}^{K} \exp(a_j)} + \cdots + \frac{\exp(a_K)}{\sum_{j=1}^{K} \exp\left(a_j\right)}$$

$$= 1$$

Example: Regression

- How to calculate the weights?

$$\vec{x} = [x_1 \quad x_2]^T$$



$$y = \sigma(a)$$

$$a = w_1 x_1 + w_2 x_2$$

target output: $t \in [0, 1]$

Error function:

$$E(w_1, w_2) = \frac{1}{2}(y-t)^2$$

$$\min_{w_1, w_2} E(w_1, w_2)$$

gradient: $\frac{dE}{dw_1} = \frac{dE}{dy} \cdot \frac{dy}{da} \cdot \frac{da}{dw_1}$

$$\frac{dE}{dy} = \frac{1}{2} \cdot 2 \cdot (y-t) = y - t$$

$$\frac{dy}{da} = y(1-y)$$

$$\frac{da}{dw_1} = x_1$$

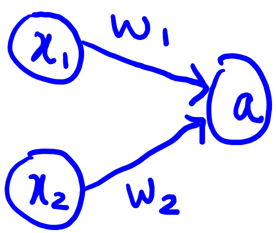
$$\frac{dE}{dw_1} = (y-t) \cdot y \cdot (1-y) \cdot x_1$$

$$\frac{dE}{dw_2} = (y-t) \cdot y \cdot (1-y) \cdot x_2$$

Example: Regression

$\vec{x} = [x_1 \quad x_2]^T$

$\min_{w_1, w_2} E(w_1, w_2)$



$y = \sigma(a)$

$\frac{dE}{dw_1} = (y-t)\cdot y \cdot (1-y)\cdot x_1$

$\frac{dE}{dw_2} = (y-t)\cdot y \cdot (1-y)\cdot x_2$

$a = w_1 x_1 + w_2 x_2$

gradient descent:

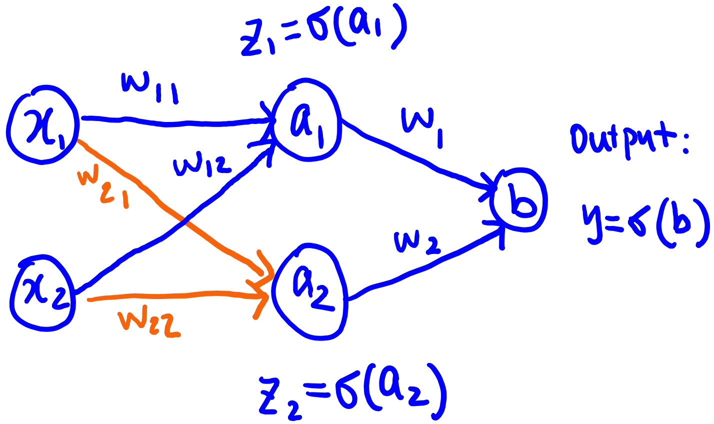
$w_1^{(\tau)} = w_1^{(\tau-1)} - \eta \cdot (y-t)\cdot y \cdot (1-y)\cdot x_1$

$w_2^{(\tau)} = w_2^{(\tau-1)} - \eta \cdot (y-t)\cdot y \cdot (1-y)\cdot x_2$

where $y = \sigma(a) = \sigma\left(w_1^{(\tau-1)} x_1 + w_2^{(\tau-1)} x_2\right)$

Example: Regression

$\vec{x} = [x_1 \quad x_2]^T$



$y = \sigma(b)$

$b = w_1 z_1 + w_2 z_2$

$z_1 = \sigma(a_1), \quad z_2 = \sigma(a_2)$

$a_1 = w_{11} x_1 + w_{12} x_2$

$a_2 = w_{21} x_1 + w_{22} x_2$

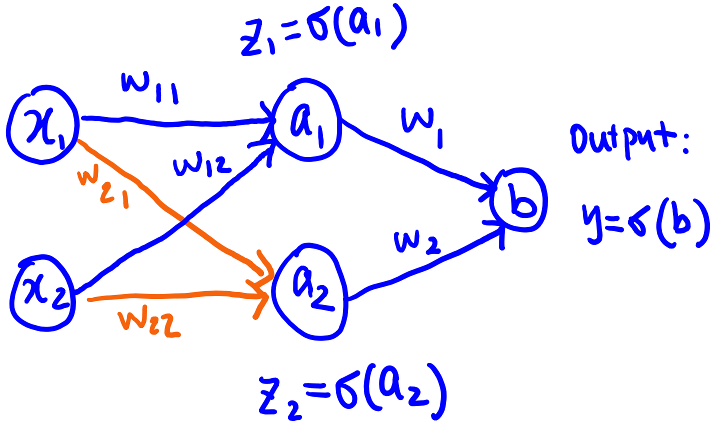
Question: How many parameters do we need to calculate?

Answer: 6

$\vec{w} = [w_{11} \; w_{12} \; w_{21} \; w_{22} \; w_1 \; w_2]^T$

Example: Regression

$\vec{x} = [x_1 \quad x_2]^T$



$y = \sigma(b)$

$b = w_1 z_1 + w_2 z_2$

$z_1 = \sigma(a_1), \quad z_2 = \sigma(a_2)$

$a_1 = w_{11} x_1 + w_{12} x_2$

$a_2 = w_{21} x_1 + w_{22} x_2$

$\min_{\vec{w}} E(\vec{w}), \quad E(\vec{w}) = \frac{1}{2}(y - t)^2$

gradient descent

$$\frac{dE}{dw_1} = \frac{dE}{dy} \cdot \frac{dy}{db} \cdot \frac{db}{dw_1}$$

$$\frac{dE}{dy} = y - t$$

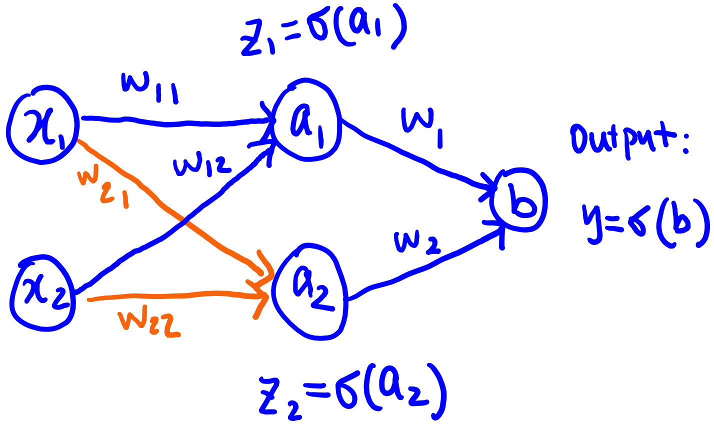
$$\frac{dy}{db} = y \cdot (1 - y)$$

$$\frac{db}{dw_1} = z_1$$

$$\boxed{\begin{aligned} \frac{dE}{dw_1} &= (y-t) \cdot y \cdot (1-y) \cdot z_1 \\ \frac{dE}{dw_2} &= (y-t) \cdot y \cdot (1-y) \cdot z_2 \end{aligned}}$$

Example: Regression

$\vec{x} = [x_1 \quad x_2]^T$



$y = \sigma(b)$

$b = w_1 z_1 + w_2 z_2$

$z_1 = \sigma(a_1), \quad z_2 = \sigma(a_2)$

$a_1 = w_{11} x_1 + w_{12} x_2$

$a_2 = w_{21} x_1 + w_{22} x_2$

$\min_{\vec{w}} E(\vec{w}), \quad E(\vec{w}) = \frac{1}{2}(y - t)^2$

gradient descent

$$\frac{dE}{dw_{11}} = \frac{dE}{dy} \cdot \frac{dy}{db} \cdot \frac{db}{dz_1} \cdot \frac{dz_1}{da_1} \cdot \frac{da_1}{dw_{11}}$$

$$\frac{dE}{dy} = y - t, \quad \frac{dy}{db} = y \cdot (1-y), \quad \frac{db}{dz_1} = w_1$$

$$\frac{dz_1}{da_1} = z_1(1-z_1), \quad \frac{da_1}{dw_{11}} = x_1$$

$$\frac{dE}{dw_{11}} = (y-t) \cdot y \cdot (1-y) \cdot w_1 \cdot z_1 \cdot (1-z_1) \cdot x_1$$

$$\frac{dE}{dw_{12}} = (y-t) \cdot y \cdot (1-y) \cdot w_1 \cdot z_1 \cdot (1-z_1) \cdot x_2$$

Backpropagation

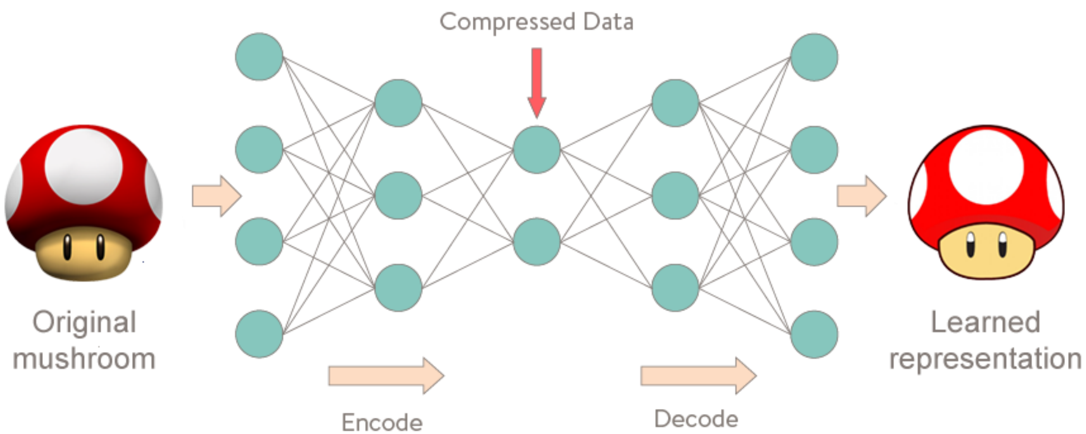
- What we have just seen is called: backpropagation
  - Propagation the (regression or classification) error from the output of the network to previous layers
  - to calculate the gradients of the error function with respect to the weights
  - and to update the weights by gradient descent

Summary

- Forward Pass
  - From input to output
  - Pass the input through the network, using the already known weights (parameters), and obtain the network output
- Backward Pass
  - From output to previous layers
  - Propagate the error from the network output to previous layers
  - Update the weights (parameters)

Regression problem

- For example:
  - Image compression and decompression



  - Input: $\mathbf{x}_n \in \mathbb{R}^D$;
  - Target output:
  - $\mathbf{t}_n = [t_{n1}, t_{n2}, \dots, t_{nD}]^T = \mathbf{x}_n = [x_{n1}, x_{n2}, \dots, x_{nD}]^T \in \mathbb{R}^D$
  - Actual network output: $\mathbf{y}_n = [y_{n1}, y_{n2}, \dots, y_{nD}]^T \in \mathbb{R}^D$

Regression problem

- For example:
  - Image compression and decompression
- Sum-squared-error error function
  - N data samples

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n$$

where $E_n = \frac{1}{2}\sum_{k=1}^{D}(y_{nk}(\mathbf{x}_n, \mathbf{w}) - t_{nk})^2$

Binary Classification

- Given a set of training samples $\mathbf{x}_n \in \mathbb{R}^D$ and the corresponding target vectors $t_n, n = 1, 2, \ldots, N$.
- Binary classification problem: $t_n \in \{0,1\}$
  - $t_n = 1$: $\mathbf{x}_n$ belongs to class-1
  - $t_n = 0$: $\mathbf{x}_n$ belongs to class-0

Binary Classification

- Cross-entropy error function
  - $E(\mathbf{w}) = \sum_{n=1}^{N} E_n$
- $E_n = -[t_n \ln y_n + (1 - t_n)\ln(1 - y_n)]$ is the error function for the n-th training sample
  - $t_n \in \{0,1\}$ is the class label for the n-th training sample
  - y_n: the predicted class probability for the n-th training sample
  - $y_n = y_n(\mathbf{x}_n, \mathbf{w})$: a function of the input $\mathbf{x}_n$ and the weights $\mathbf{w}$

K-class classification

- Given a set of training samples $\mathbf{x}_n \in \mathbb{R}^D$ and the corresponding target vectors $\mathbf{t}_n, n = 1, 2, \dots, N$.
- For example
  - K-class classification problem: $\mathbf{t}_n \in \{0,1\}^K$
    - This is called: 1-of-K coding
    - $t_{nk} = 1$: $\mathbf{x}_n$ belongs to class-k
    - $t_{nk} = 0$: $\mathbf{x}_n$ does not belong to class-k

    - Example: $K = 5$, 5 classes
    - $\mathbf{t}_n = [0,0,0,1,0]^T$: $\mathbf{x}_n$ belongs to class-4

K-class classification

- Cross-entropy error function

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n$$

where $E_n = -\sum_{k=1}^{K} t_{nk} \ln y_{nk}$

  – t_{nk} is the one-of-K coding class label for the n-th training sample
  – $y_{nk} = y_{nk}(\mathbf{x}_n, \mathbf{w}), k = 1, \dots, K$, is the predicted probability of the n-th training sample belonging to class-k

Network Training

- Find weights: $\mathbf{w}^* = arg \min_{\mathbf{w}} E(\mathbf{w})$
- Gradient descent:

$$\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \eta \nabla_{\mathbf{w}} E(\mathbf{w}^{(\tau-1)})$$

- where $\eta > 0$ is the learning rate
- τ: the iteration index
- The above is the general training method for
  - Regression
  - Binary classification
  - Multi-class classification

Mini-batch Gradient Descent

- In each iteration, compute the gradient based on a small set of training samples
- $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \eta \sum_{n \in \mathcal{N}_i} \nabla_{\mathbf{w}} E_n\left(\mathbf{w}^{(\tau-1)}\right)$
- $\mathcal{N}_i$: the set of indices for data samples in the i-th mini-batch

Concepts

- Epoch: the number of rounds to go through all training samples. Each round is an epoch.
- Shuffle: before starting each epoch, the training samples are shuffled, therefore the mini-batches in different epochs are different.

- Notations:
  - τ: iteration index (for weight update)
  - n: training sample index
  - i: mini-batch index

Training Procedure

- 10,000 training samples, Batch size = 200;
- 10,000/200 =50 mini-batches, Epoch=10
- Step 1: initialize $\mathbf{w}^{(0)}$
- Step 2: for epoch=1, 2, …, 10, do

  shuffle the training samples

  for i =1:50

    1. forward pass of the i-th mini-batch, using the old $\mathbf{w}^{(\tau-1)}$: $\mathbf{x}_n \rightarrow \mathbf{z}_n \rightarrow \mathbf{y}_n$ for the n-th sample in the i-th mini-batch

    2. backward pass: backpropagate the gradients

    $\sum_{n\in\mathcal{N}_i} \nabla_{\mathbf{w}} E_n\left(\mathbf{w}^{(\tau-1)}\right)$

    3. update $\mathbf{w}$ by $\mathbf{w}^{(\tau)} = \mathbf{w}^{(\tau-1)} - \eta \sum_{n\in\mathcal{N}_i} \nabla_{\mathbf{w}} E_n\left(\mathbf{w}^{(\tau-1)}\right)$

  end

end

What's the total number of iterations?

How many times are the weights updated?

Summary

- Network Training
  - Use the training set
  - Calculate the network weights
  - Use both forward and backward pass
- Testing
  - Also called “inference”
  - After the model is trained (weights are found), use the test set to evaluate the model performance
  - Only use the forward pass

Diabetes: binary classification

- Dataset Description
  - The Pima Indian Diabetes Dataset consists of information on 768 female patients (268 tested_positive instances and 500 tested_negative instances) coming from a population near Phoenix, Arizona, USA. Tested_positive and tested_negative indicates whether the patient is **diabetic** or not, respectively.
- 8 features
  - Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age.
- Labels
  - 1: tested_positive
  - 0: tested_negative

Diabetes: binary classification

- Objective
  - Build a 3-layer neural network to predict whether a patient is diabetic or not.

Diabetes: binary classification

```
# Create first network with Keras
from keras.models import Sequential
from keras.layers import Dense
import numpy
from sklearn.model_selection import train_test_split
from sklearn import metrics
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
X_train, X_test, y_train, y_test = train_test_split(X, Y, \
                                                    stratify=Y, random_state=42,test_size=0.25)
```

Diabetes: binary classification

```
# 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'))
```

Output dimension

- Dense layer: fully connected layer
- Input_dim: input dimension

Diabetes: binary classification

```
# 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=2)
# calculate predictions
predictions = model.predict(X_test)
# round predictions
y_test_hat = [round(x[0]) for x in predictions]
```

- Binary_crossentropy: the cost function (or error function) for binary classification problem. We need to minimize this function to find the weights
- optimizer: the method/solver used to minimize the cost function

Diabetes: binary classification

```
num_correct = 0
for i in range(len(y_test)):
    if y_test_hat[i]==y_test[i]:
        num_correct +=1

Accuracy_rate = num_correct/len(y_test)
print("Accuracy Rate = ", Accuracy_rate)
```

Diabetes: binary classification

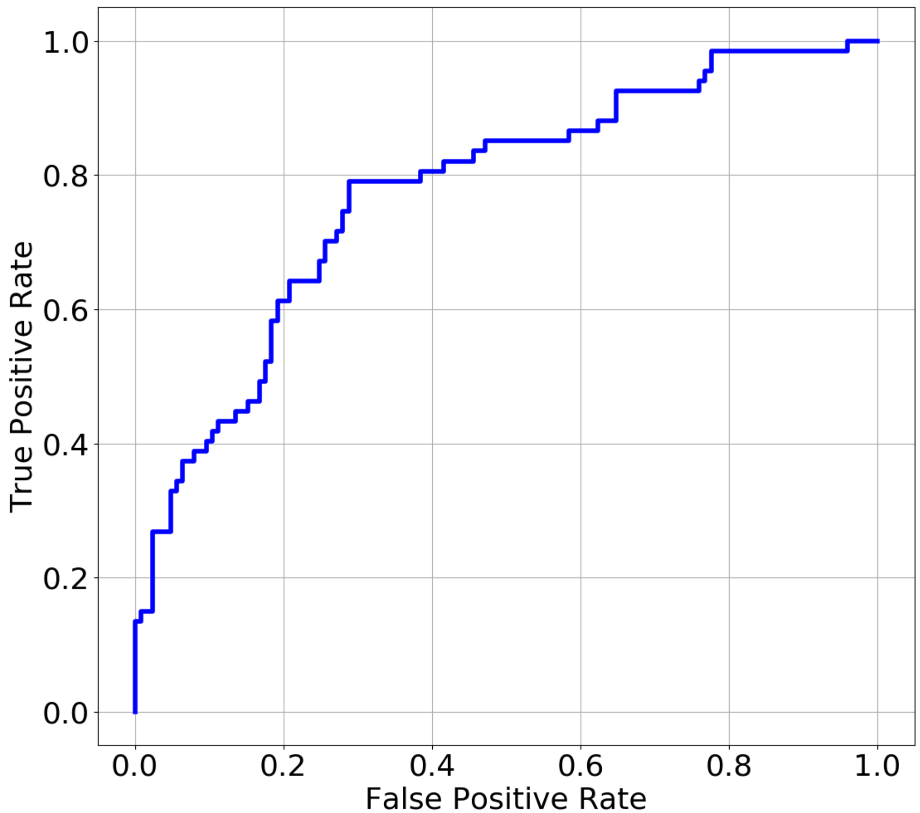
```
fpr, tpr, thresholds = metrics.roc_curve(y_test, predictions, pos_label = 1)
import matplotlib.pyplot as plt
plt.plot(fpr,tpr, 'b-',linewidth=4)
plt.ylabel('True Positive Rate',fontsize = 26)
plt.xlabel('False Positive Rate',fontsize = 26)

plt.tick_params(labelsize=26)
plt.grid(True)
plt.show()
```

- Plot the ROC curve

Diabetes: binary classification

- ROC curve: the receiver operating characteristic curve
- For binary classification, plot the false positive rate vs the true positive rate (obtained by setting different threshold)



Diabetes: binary classification

```python
from keras.models import Sequential
from keras.layers import Dense
import numpy
from sklearn.model_selection import train_test_split
from sklearn import metrics
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")
# split into input (X) and output (Y) variables
X = dataset[:,0:8]
Y = dataset[:,8]
X_train, X_test, y_train, y_test = train_test_split(X, Y, \
                                                    stratify=Y, random_state=42,test_size=0.25)
# 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=2)
# calculate predictions
predictions = model.predict(X_test)
# round predictions
y_test_hat = [round(x[0]) for x in predictions]

num_correct = 0
for i in range(len(y_test)):
    if y_test_hat[i]==y_test[i]:
        num_correct +=1

Accuracy_rate = num_correct/len(y_test)
print("Accuracy Rate = ", Accuracy_rate)

fpr, tpr, thresholds = metrics.roc_curve(y_test, predictions, pos_label = 1)
import matplotlib.pyplot as plt
plt.plot(fpr,tpr, 'b-',linewidth=4)
plt.ylabel('True Positive Rate',fontsize = 26)
plt.xlabel('False Positive Rate',fontsize = 26)

plt.tick_params(labelsize=26)
```

Usage

Here you can get
either on the Edit

Help can also be
parenthesis next t
Preferences > He

Help | Variable explorer | File explorer

IPython console

Console 1/A

```
 - 0s - loss: 0.5181 - acc: 0.7535
Epoch 143/150
 - 0s - loss: 0.4907 - acc: 0.7639
Epoch 144/150
 - 0s - loss: 0.4774 - acc: 0.7917
Epoch 145/150
 - 0s - loss: 0.4807 - acc: 0.7847
Epoch 146/150
 - 0s - loss: 0.4763 - acc: 0.7795
Epoch 147/150
 - 0s - loss: 0.4733 - acc: 0.7847
Epoch 148/150
 - 0s - loss: 0.4809 - acc: 0.7830
Epoch 149/150
 - 0s - loss: 0.4936 - acc: 0.7726
Epoch 150/150
 - 0s - loss: 0.4819 - acc: 0.7778
Accuracy Rate =  0.7135416666666666

In [9]:
```

Choice of Output Activation and Error Function

- Binary classification
  - Sigmoid activation
  - Cross-entropy error function
- Multi-class classification
  - Softmax activation function
  - Cross-entropy error function
- Regression problem
  - For example, image compression and decompression
  - (optional) Activation function
    - If input pixels are in [0,255]: can use ReLU
    - If input pixels are in [0,1]: can use Sigmoid
    - If input pixels are in [-1,1]: can use tanh
  - Sum-squared-error cost function