L9: Neural Network

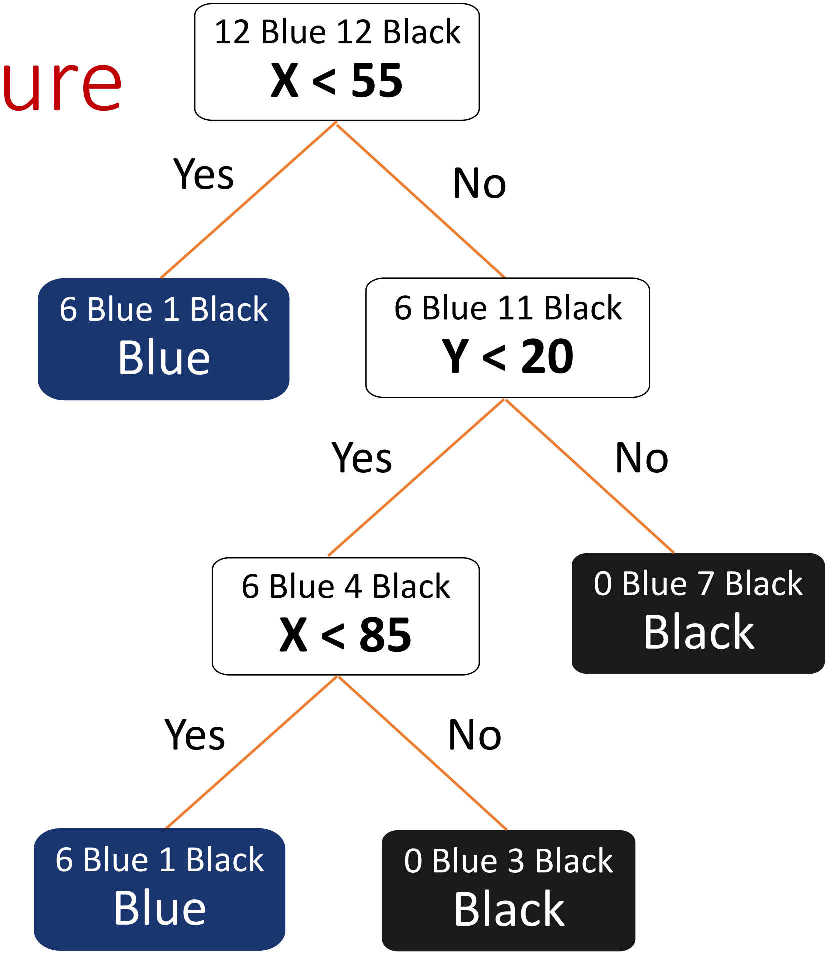
Shan Wang

Lingnan College, Sun Yat-sen University

2020 Data Mining and Machine Learning LN3119
https://wangshan731.github.io/DM-ML/

Last lecture

- Decision Tree Model
- Strategy & Algorithm
    - ID.3: information gain
    - C4.5: information gain ratio
    - CART
        - Classification: Gini index
        - Regression: Squared error
- Regularization: pruning
- Random forest: bagging
- Application: Supreme Court Decisions



Information gain: with the X, how much the uncertainty of Y decreases

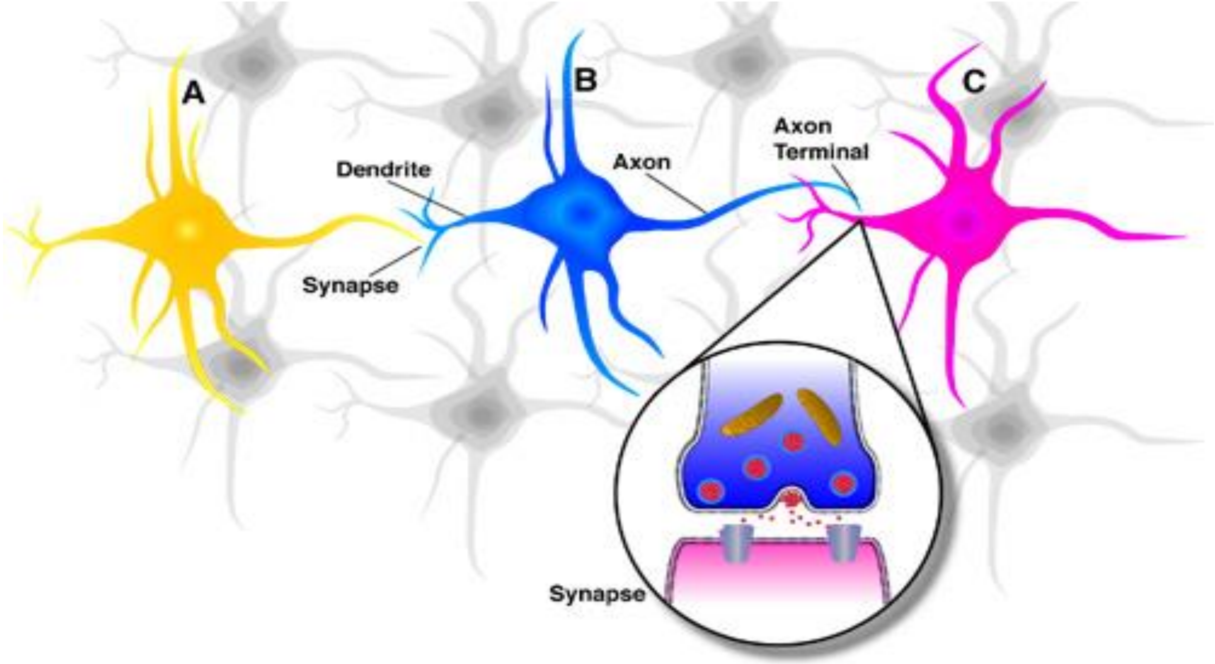
Classification Problem Revisit

$$y = \begin{cases} +1, & \text{if } f_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq 0 \\ -1, & \text{if } f_{\boldsymbol{\theta}}(\boldsymbol{x}) < 0 \end{cases}$$

- Logistic regression
  - $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{e^{\boldsymbol{\theta}' \boldsymbol{x}}}{1+e^{\boldsymbol{\theta}' \boldsymbol{x}}} - h$
- SVM
  - $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}' \phi(\boldsymbol{x}) + \theta_0$
- Decision tree
  - $f_{\boldsymbol{\theta}}(\boldsymbol{x})$: a list of rules
- What is the $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ in our brain?

Neurons in the Brain

- The brain is composed of neurons
  - A neuron receives inputs from other neurons via synapses (突触)
  - Inputs are approximately summed
  - When the input exceeds a threshold the neuron sends an electrical spike that from the body, down the axon(轴突), to the next neuron(s) as output



Course outline

- Supervised learning
  - Linear regression
  - Logistic regression
  - SVM and kernel
  - Tree models
- Unsupervised learning
  - Clustering
  - PCA
  - EM
- Deep learning
  - Neural networks
  - Convolutional NN
  - Recurrent NN
- Reinforcement learning
  - MDP
  - ADP
  - Deep Q-Network

This lecture

- One-layer Perceptron
- Multi-layer Perceptron
  - Model
  - Strategy
  - Algorithm
- Overfitting
- Application: Toyota Corolla

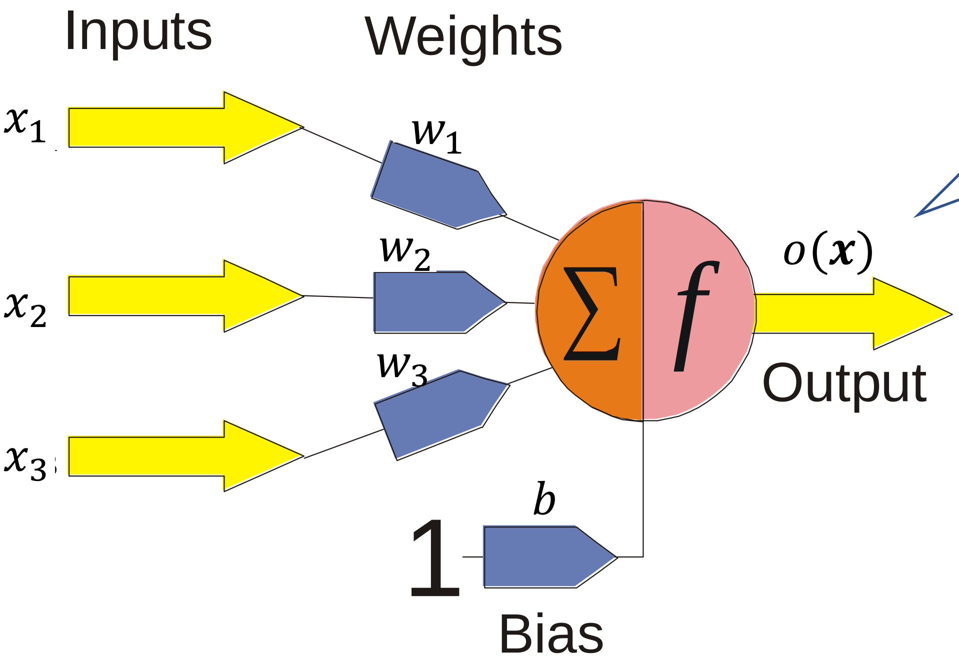
Reference: VE 445, Shuai LI (SJTU)

Perceptron

One-layer Neural Network

One-layer Perceptron

- The output is a function of the weighted sum of inputs

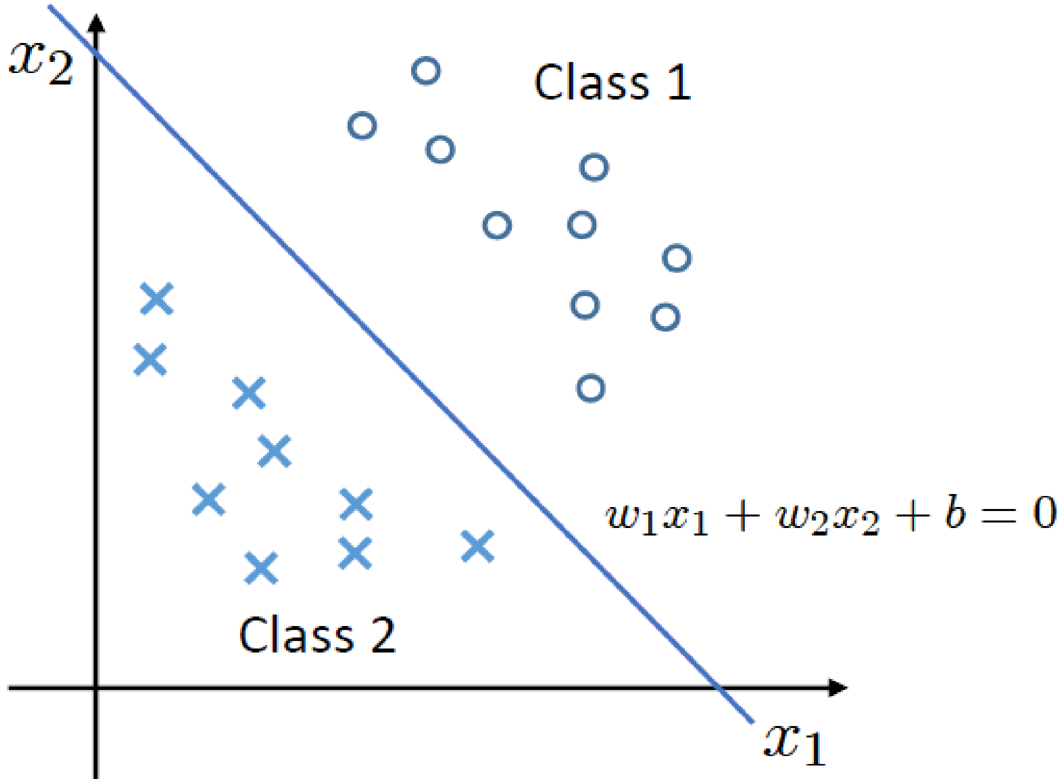


Only input layer and output layer

$$o(\boldsymbol{x}) = f(x_1w_1 + x_2w_2 + x_3w_3 + b)$$

One-layer Perceptron (cont.)

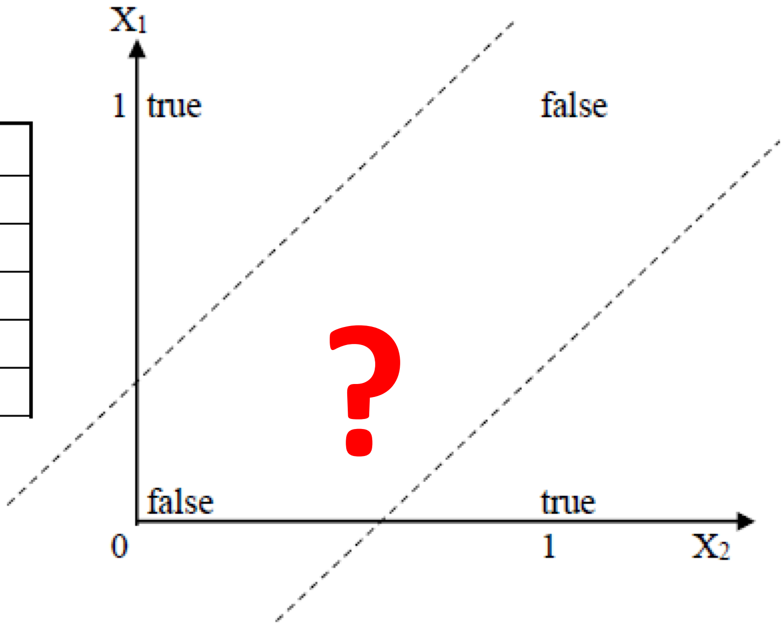
- A one-layer perceptron is very similar to Logistic regression and SVM!



Limitation of One-layer Perceptron

- XOR problem cannot be done by one-layer perceptron

Input x		Output y
X_1	X_2	X_1 XOR X_2
0	0	0
0	1	1
1	0	1
1	1	0

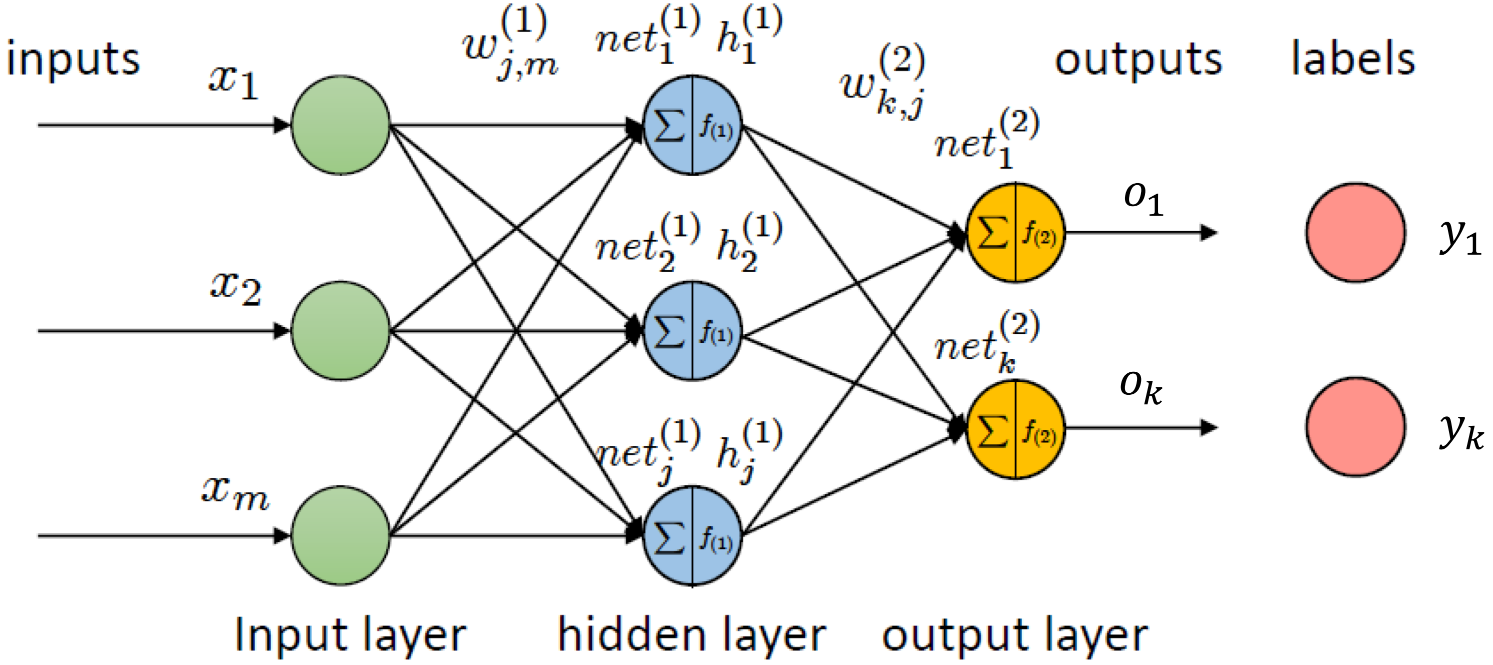


Solution: add layers, between input layer and output layer!

Multi-layer Perceptron

Model

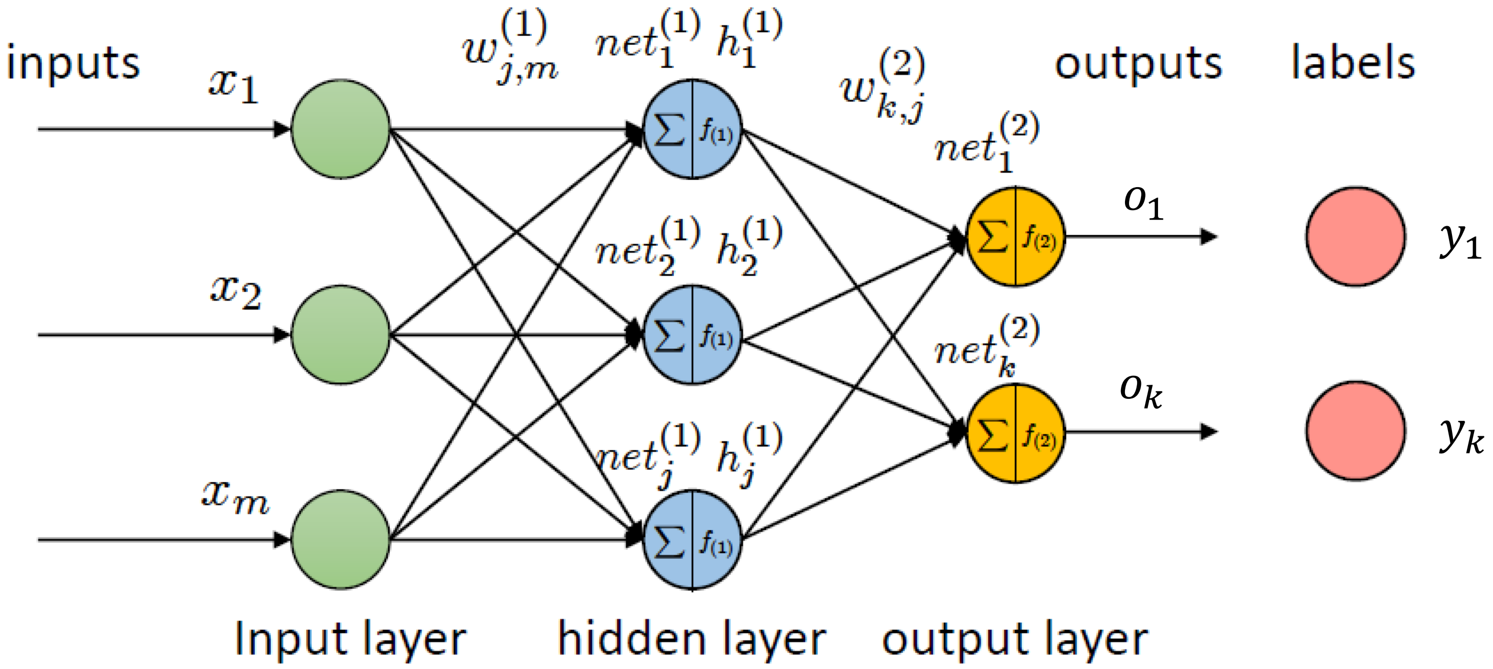
Multi-layer Perceptron - Model



Two-layer feedforward neural network

- Multi-layer Perceptron Structure:
  - Input layer
    - Representing variables: $(x_1, x_2, \dots, x_M)$
  - Hidden layer
  - Output layer
    - Classification problem: multiple neurons, representing labels: $(y_1, y_2, \dots, y_K)$
    - Regression problem: single neuron

Multi-layer Perceptron - Model

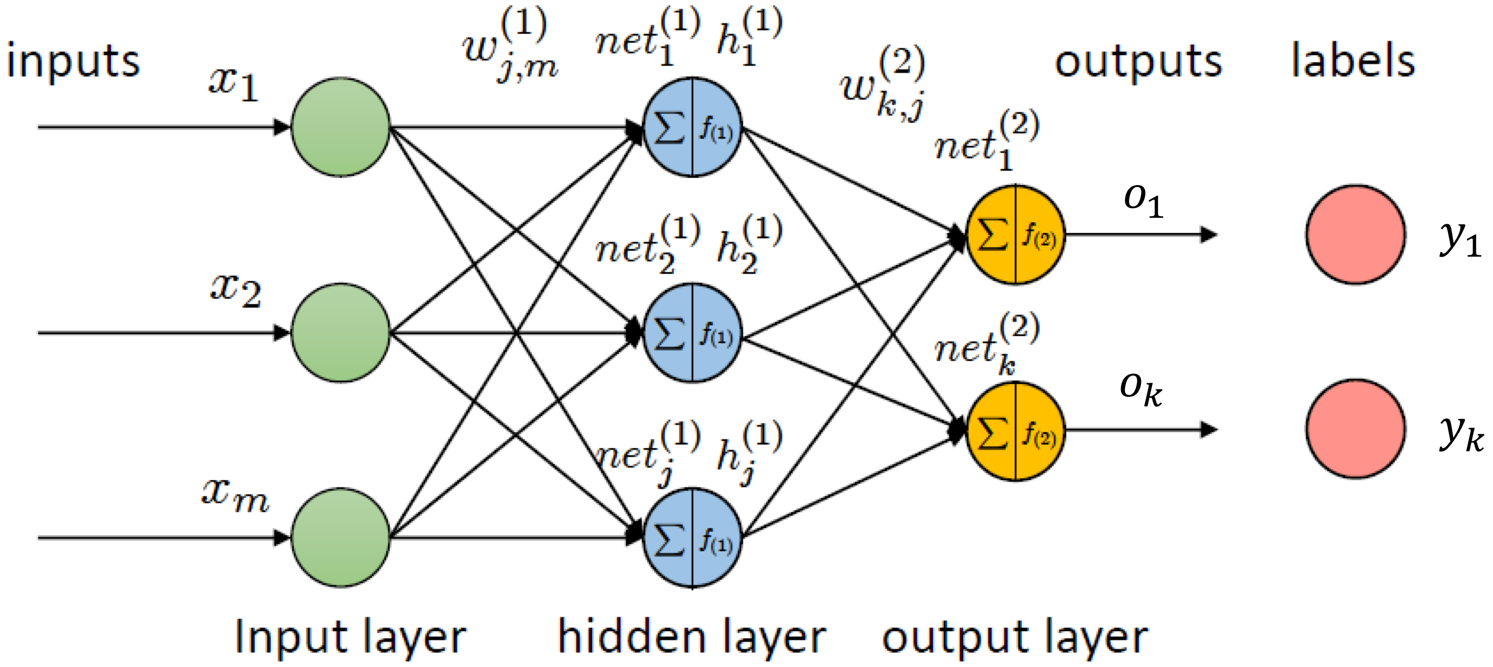


Two-layer feedforward neural network

- Hidden layer
  - Each neuron is a function
  - The outputs are the inputs for next layer
- The link
  - $w_{j,m}^{(1)}$: weight from input m to neuron j
  - $w_{k,j}^{(2)}$: weight from neuron j to output k

Key issue:
Design the structure!
How many layers?
How many nodes?

Multi-layer Perceptron - Model

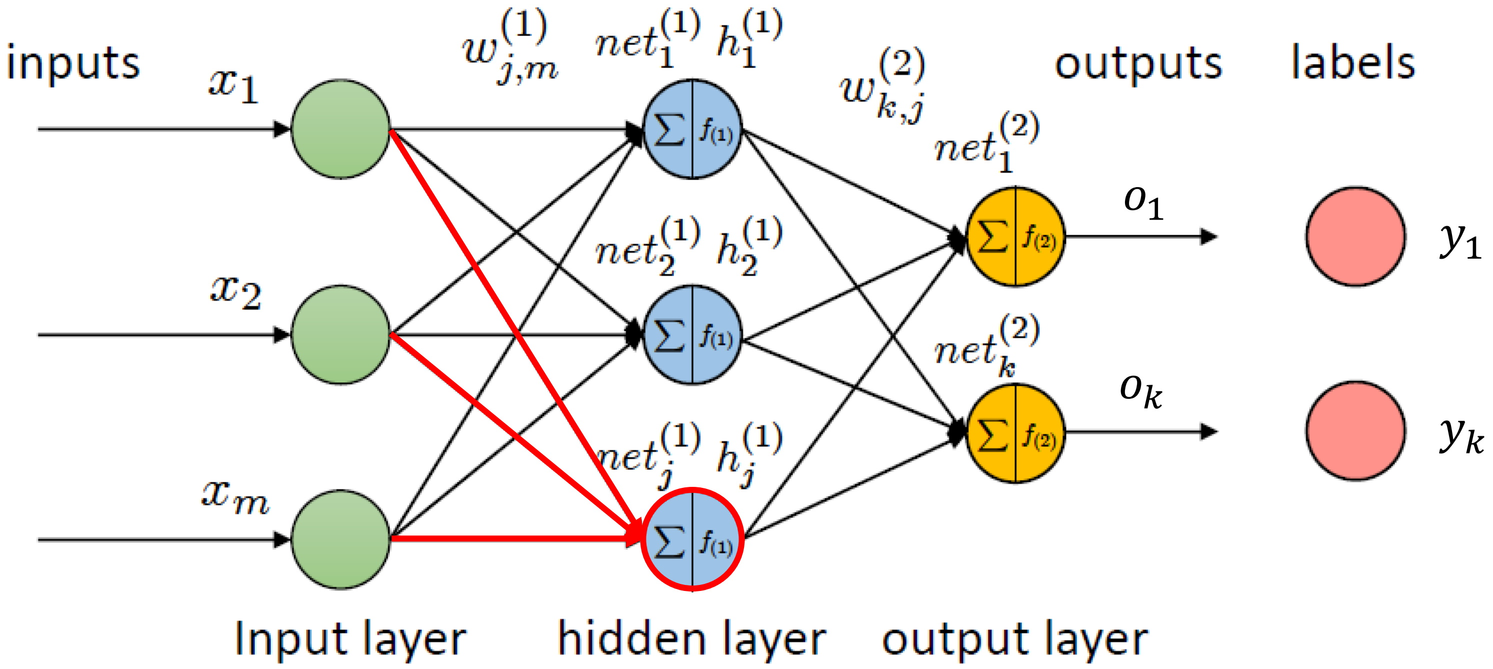


Two-layer feedforward neural network

Making prediction:

$$\boldsymbol{x} = (x_1, \dots, x_M) \longrightarrow h_j^{(1)} \longrightarrow o_k$$

Multi-layer Perceptron - Model



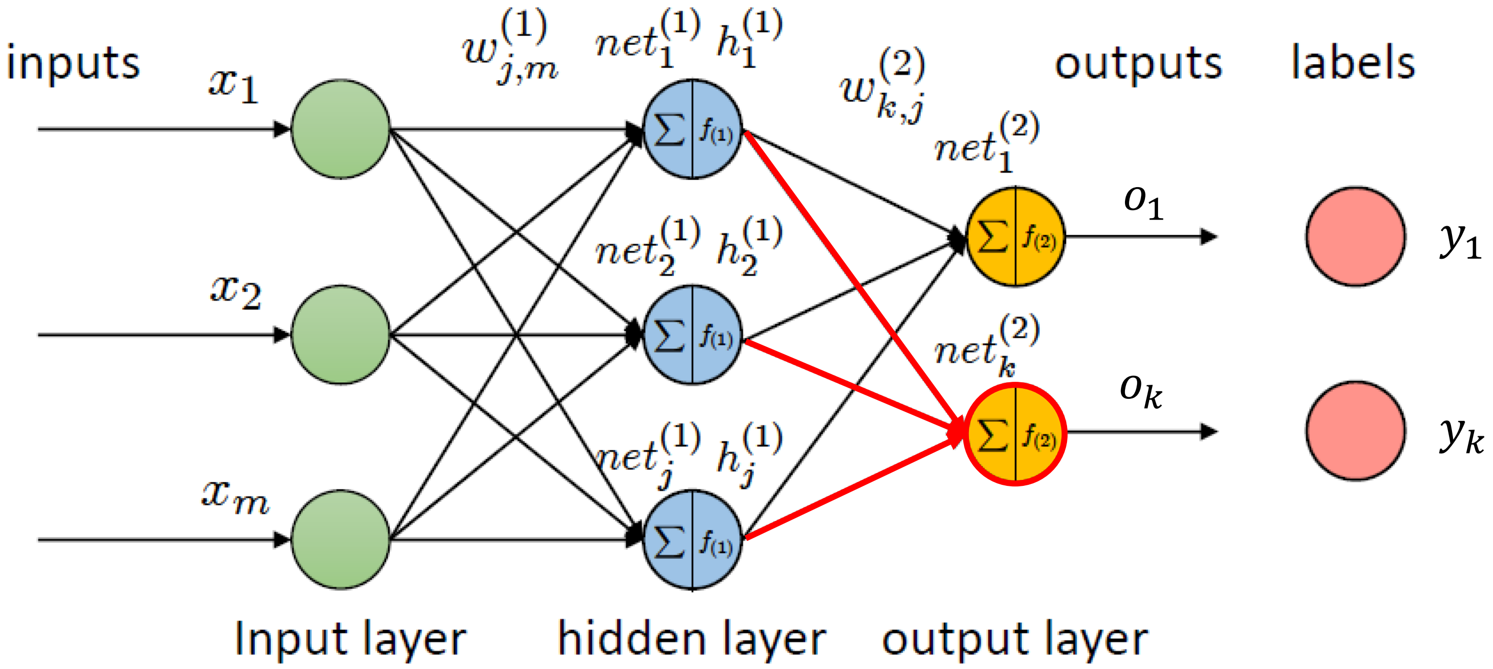
Two-layer feedforward neural network

Making prediction:

$$\boldsymbol{x} = (x_1, \dots, x_M) \xrightarrow{h_j^{(1)}=f_{(1)}\left(net_j^{(1)}\right)=f_{(1)}(\sum_m w_{j,m}^{(1)} x_m)} h_j^{(1)} \longrightarrow o_k$$

where $$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

Multi-layer Perceptron - Model



Two-layer feedforward neural network

Making prediction:

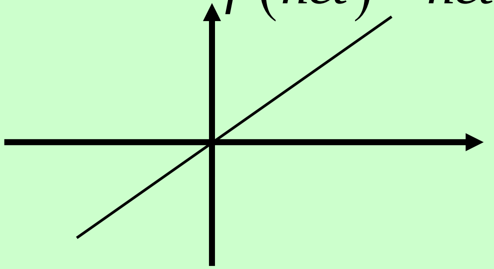
$$\boldsymbol{x} = (x_1, \dots, x_M) \xrightarrow{h_j^{(1)} = f_{(1)}\left(net_j^{(1)}\right) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m)} h_j^{(1)} \xrightarrow{o_k = f_{(2)}\left(net_k^{(2)}\right) = f_{(2)}(\sum_m w_{k,j}^{(2)} h_j^{(1)})} o_k$$

where $\quad net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m \qquad net_k^{(2)} = \sum_m w_{k,j}^{(2)} h_j^{(1)}$

Activation Function

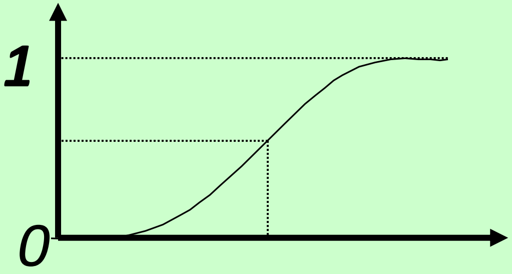
Linear activation

$$f\left(net\right)=net$$



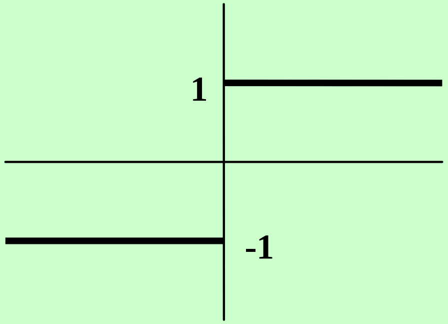
Sigmoid activation

$$f\left(net\right)=\frac{1}{1+e^{-net}}$$



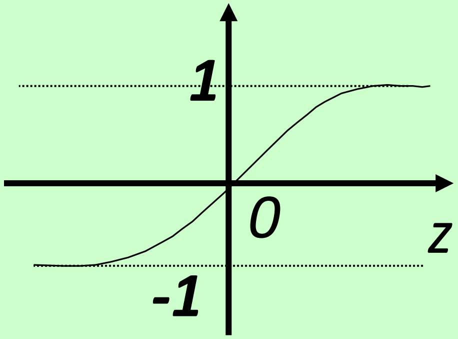
Threshold activation

$$f\left(net\right)=\text{sign}(net)=\begin{cases}1, & if \quad net\geq 0,\\ -1, & if \quad net<0.\end{cases}$$

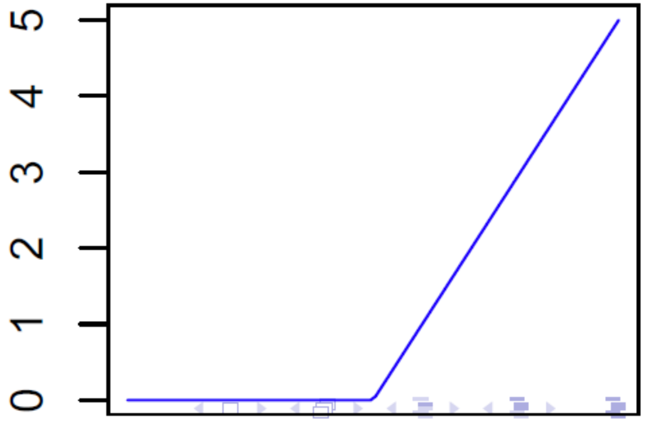


Hyperbolic tangent activation

$$f\left(net\right)=tanh\left(net\right)=\frac{1-e^{-2net}}{1+e^{-2net}}$$

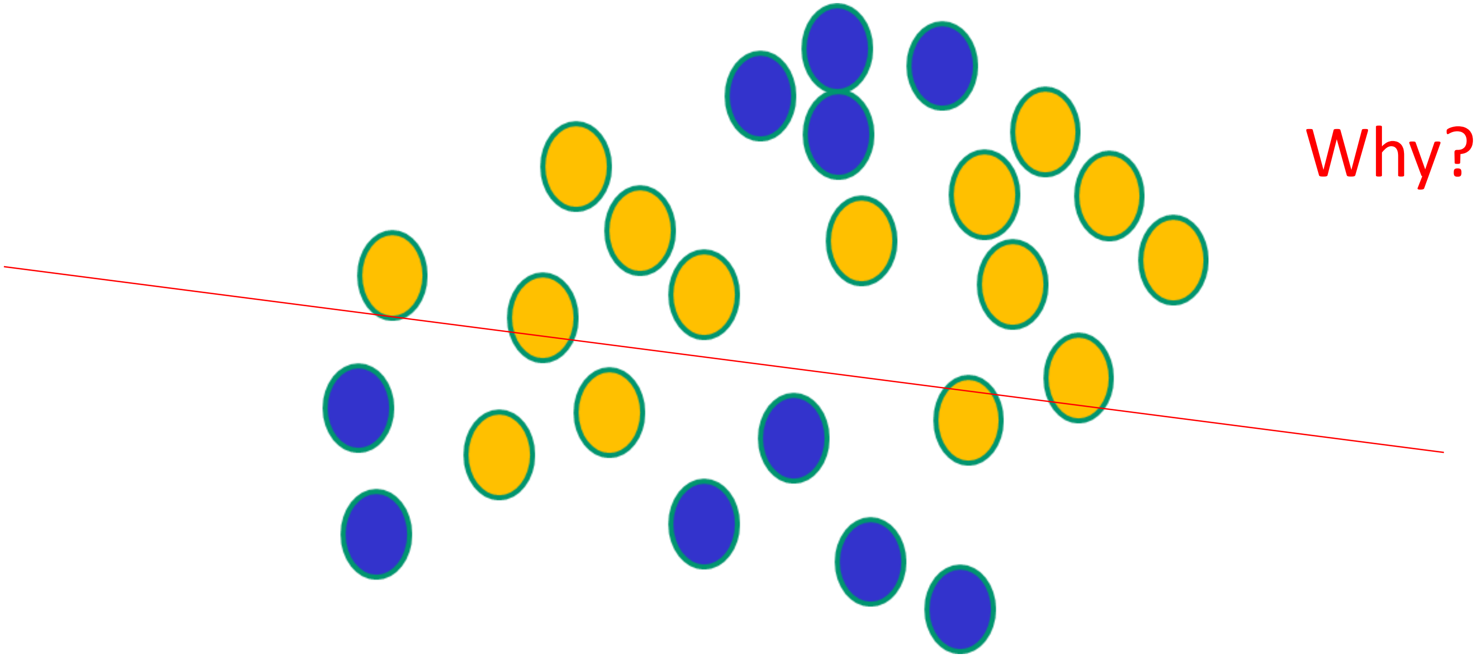


rectified linear unit (ReLU)



Activation Function (cont.)

- If $f(x)$ is linear, the NN can **only** draw straight decision boundaries (even if there are many layers of units)



Why?

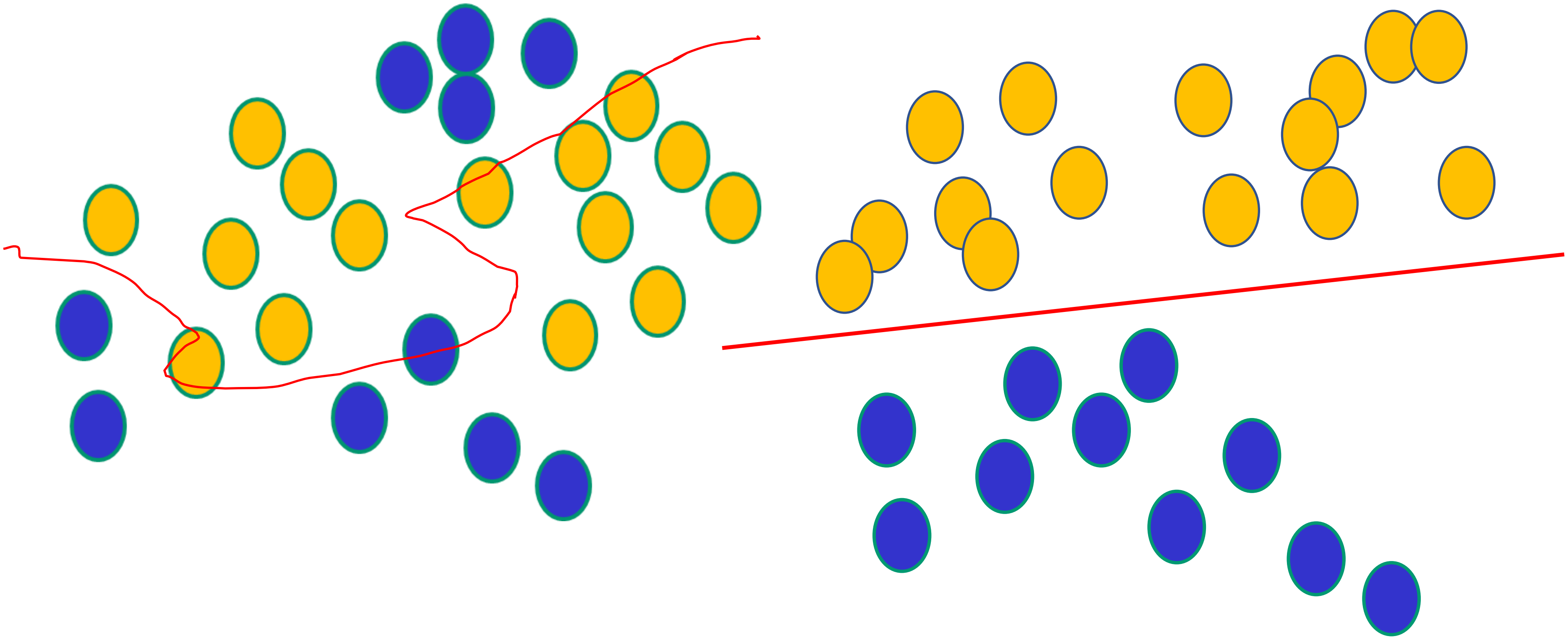
Activation Function (cont.)

- If $f(x)$ is non-linear, a network with 1 hidden layer can, in theory, learn perfectly **any** classification problem.
  - A set of weights exists that can produce the targets from the inputs.
  - The problem is finding them.

Compared with SVM

NNs use nonlinear $f(x)$ so they can draw complex boundaries, but keep the data unchanged

SVMs only draw straight lines, but they transform the data first in a way that makes that OK



Multi-layer Perceptron

Strategy

Minimize Empirical Error

- For regression problem

$$\min \frac{1}{2N} \sum_{n=1}^{N} (o^n - y^n)^2$$

  - o^n is the output of the input $\boldsymbol{x}^n$ of sample $(\boldsymbol{x}^n, y^n)$
  - y^n is the true value of the input $\boldsymbol{x}^n$

- For classification problem

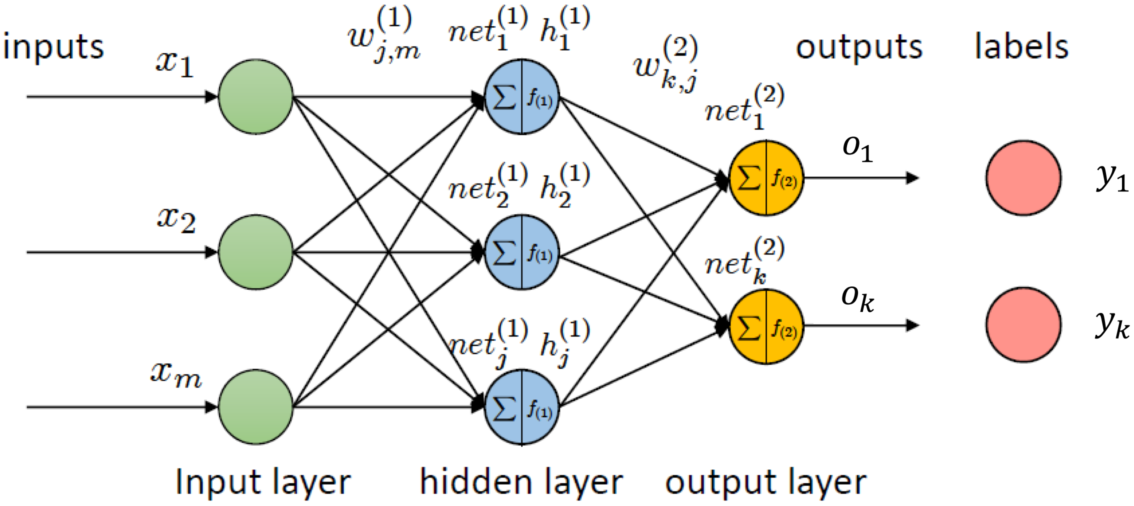
$$\min \frac{1}{2N} \sum_{n=1}^{N} \sum_{k=1}^{K} (o_k^n - y_k^n)^2$$

  - o_k^n is the output for label k of the input $\boldsymbol{x}^n$ of sample $(\boldsymbol{x}^n, \boldsymbol{y}^n)$
  - $\boldsymbol{y}^n$ is the true label of the input $\boldsymbol{x}^n$: $(0,0,1,\ldots,0)$

Multi-layer Perceptron

Algorithm

Forward Propagation of Information



Two-layer feedforward neural network

The error:

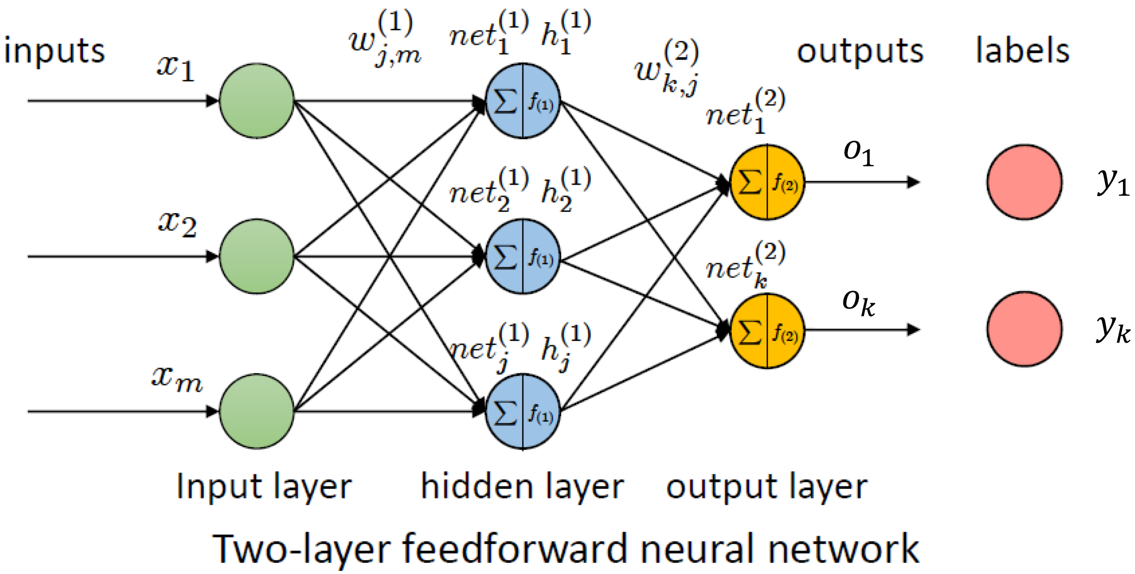
$$\varepsilon(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{K}(o_k - y_k)^2$$

Forward Propagation of Information:

$$\boldsymbol{x} = (x_1, \dots, x_M) \xrightarrow{h_j^{(1)}=f_{(1)}\left(net_j^{(1)}\right)=f_{(1)}(\sum_m w_{j,m}^{(1)} x_m)} h_j^{(1)} \xrightarrow{o_k=f_{(2)}\left(net_k^{(2)}\right)=f_{(2)}(\sum_m w_{k,j}^{(2)} h_j^{(1)})} o_k$$

where $net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ $net_k^{(2)} = \sum_m w_{k,j}^{(2)} h_j^{(1)}$

Backward Propagation of Errors



Two-layer feedforward neural network

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

$$h_j^{(1)} = f_{(1)}\left(net_j^{(1)}\right)$$

$$net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

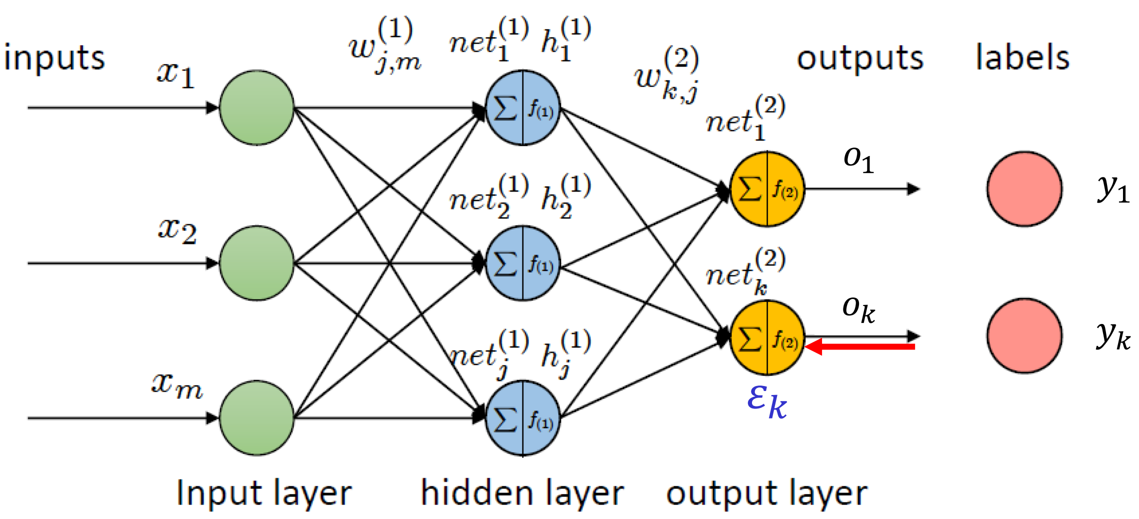
$$o_k = f_{(2)}\left(net_k^{(2)}\right)$$

$$\varepsilon(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{K}(o_k - y_k)^2$$

Backward Propagation of Errors:

$$\varepsilon_m \longleftarrow \varepsilon_j \longleftarrow \varepsilon_k \longleftarrow \varepsilon(\boldsymbol{w})$$

Backward Propagation of Errors



Two-layer feedforward neural network

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

$$h_j^{(1)} = f_{(1)}\left(net_j^{(1)}\right)$$

$$net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

$$o_k = f_{(2)}\left(net_k^{(2)}\right)$$

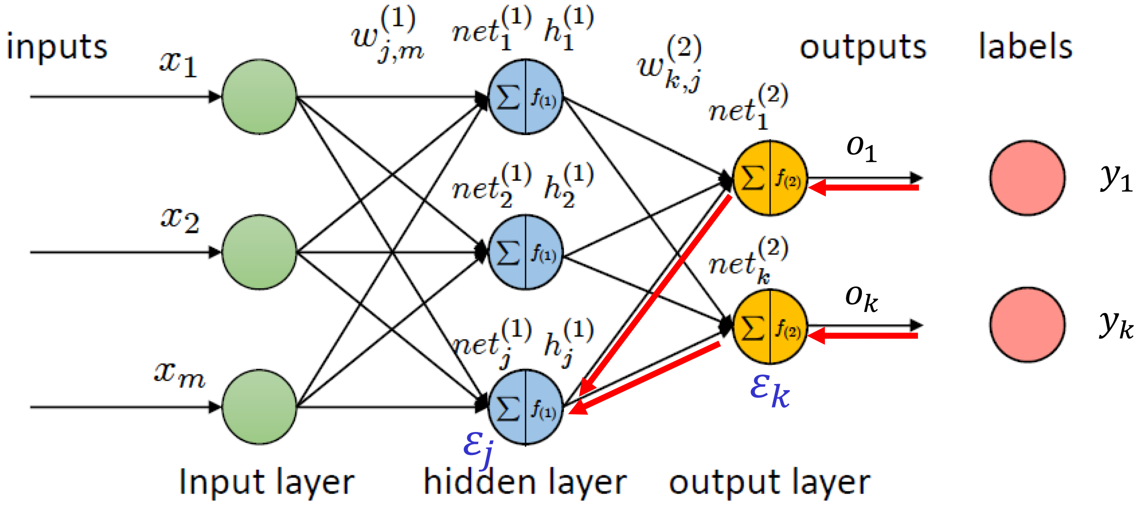
$$\varepsilon(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{K}(o_k - y_k)^2$$

Backward Propagation of Errors:

$$\begin{tikzcd}
\varepsilon_m & \varepsilon_j \arrow[l] & \varepsilon_k \arrow[l] & \varepsilon(\boldsymbol{w}) \arrow[l, "\frac{\partial \varepsilon(\boldsymbol{w})}{\partial net_k^{(2)}} = (o_k - y_k) f_{(2)}^{\prime k}"']
\end{tikzcd}$$

$$\frac{\partial \varepsilon(\boldsymbol{w})}{\partial net_k^{(2)}} = \frac{\partial \frac{1}{2}\sum_{k=1}^{K}\left(f_{(2)}(net_k^{(2)}) - d_k\right)^2}{\partial net_k^{(2)}} = \overset{\begin{gathered}\boxed{\varepsilon_k}\\ \downarrow\end{gathered}}{\boxed{(o_k - y_k) f_{(2)}^{\prime k}}}$$

Backward Propagation of Errors



Two-layer feedforward neural network

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

$$h_j^{(1)} = f_{(1)}\left(net_j^{(1)}\right)$$

$$net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

$$o_k = f_{(2)}\left(net_k^{(2)}\right)$$

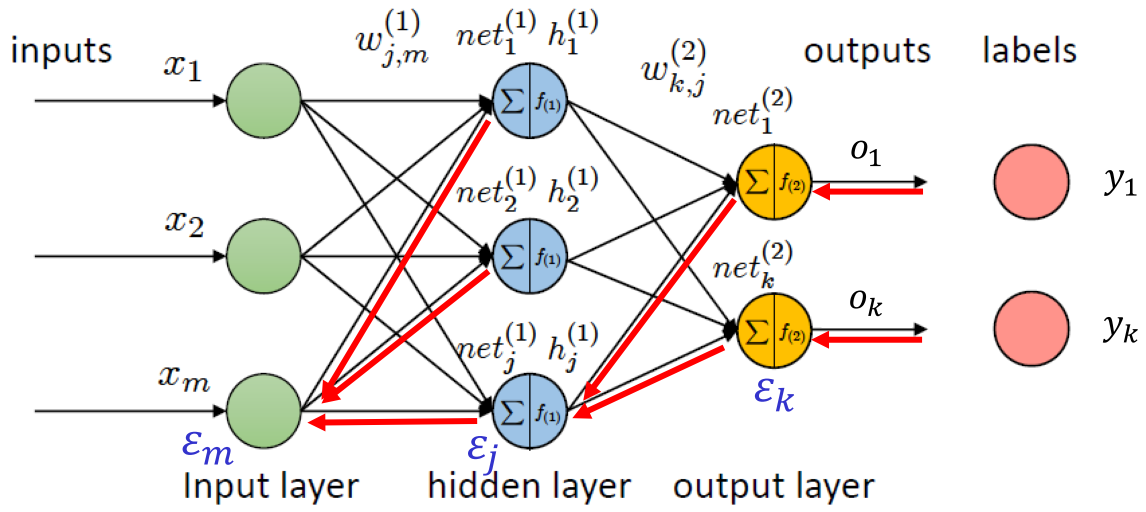
$$\varepsilon(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{K}(o_k - y_k)^2$$

Backward Propagation of Errors:

$$\varepsilon_m \longleftarrow \varepsilon_j \xleftarrow{\frac{\partial \varepsilon(\mathbf{w})}{\partial net_j^{(1)}} = \sum_{k=1}^{K} \varepsilon_k w_{k,j}^{(2)} f'^{j}_{(1)}} \varepsilon_k \xleftarrow{\frac{\partial \varepsilon(\mathbf{w})}{\partial net_k^{(2)}} = (o_k - y_k) f'^{k}_{(2)}} \varepsilon(\mathbf{w})$$

$$\frac{\partial \varepsilon(\mathbf{w})}{\partial net_j^{(1)}} = \sum_{k=1}^{K} \frac{\partial \varepsilon(\mathbf{w})}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial net_j^{(1)}} = \sum_{k=1}^{K} \varepsilon_k \frac{\partial \sum_j w_{k,j}^{(2)} f_{(1)}\left(net_j^{(1)}\right)}{\partial net_j^{(1)}} = \overset{\boxed{\varepsilon_j}\downarrow}{\boxed{\sum_{k=1}^{K} \varepsilon_k w_{k,j}^{(2)} f'^{j}_{(1)}}}$$

Backward Propagation of Errors



Two-layer feedforward neural network

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

$$h_j^{(1)} = f_{(1)}\left(net_j^{(1)}\right)$$

$$net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

$$o_k = f_{(2)}\left(net_k^{(2)}\right)$$

$$\varepsilon(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{K}(o_k - y_k)^2$$

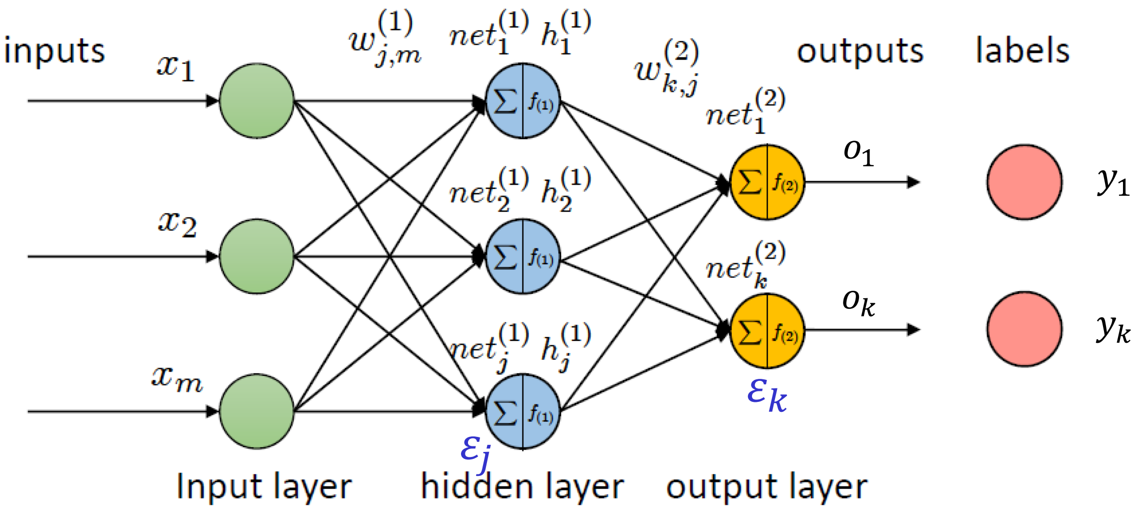
Backward Propagation of Errors:

$$\begin{tikzcd} \varepsilon_m & \varepsilon_j \arrow[l, "\frac{\partial\varepsilon(\mathbf{w})}{\partial x_m}=\sum_{j=1}^{J}\varepsilon_j w_{j,m}^{(1)}"'] & \varepsilon_k \arrow[l, "\frac{\partial\varepsilon(\mathbf{w})}{\partial net_j^{(1)}}=\sum_{k=1}^{K}\varepsilon_k w_{k,j}^{(2)} f_{(1)}^{\prime j}"'] & \varepsilon(\mathbf{w}) \arrow[l, "\frac{\partial\varepsilon(\mathbf{w})}{\partial net_k^{(2)}}=(o_k-y_k) f_{(2)}^{\prime k}"'] \end{tikzcd}$$

$$\boxed{\varepsilon_m}$$

$$\frac{\partial\varepsilon(\mathbf{w})}{\partial x_m} = \sum_{j=1}^{J}\frac{\partial\varepsilon(\mathbf{w})}{\partial net_j^{(1)}}\frac{\partial net_j^{(1)}}{\partial x_m} = \sum_{j=1}^{J}\varepsilon_j\frac{\partial\sum_m w_{j,m}^{(1)}x_m}{\partial x_m} = \boxed{\sum_{j=1}^{J}\varepsilon_j w_{j,m}^{(1)}}$$

Backward Propagation of Errors



Two-layer feedforward neural network

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

$$h_j^{(1)} = f_{(1)}\left(net_j^{(1)}\right)$$

$$net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

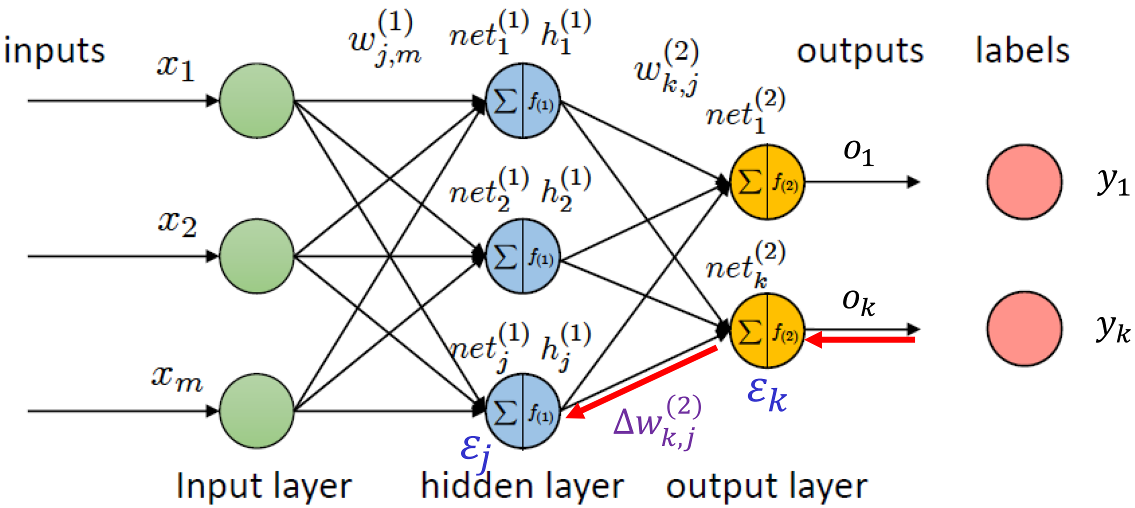
$$o_k = f_{(2)}\left(net_k^{(2)}\right)$$

$$\varepsilon(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{K}(o_k - y_k)^2$$

Backprop to learn the parameters:

$$w_{j,m}^{(1)} \longleftarrow w_{k,j}^{(2)} \longleftarrow \varepsilon(\boldsymbol{w})$$

Backward Propagation of Errors



Two-layer feedforward neural network

$$net^{(1)}_j = \sum_m w^{(1)}_{j,m} x_m$$

$$h^{(1)}_j = f_{(1)}\left(net^{(1)}_j\right)$$

$$net^{(2)}_k = \sum_j w^{(2)}_{k,j} h^{(1)}_j$$

$$o_k = f_{(2)}\left(net^{(2)}_k\right)$$

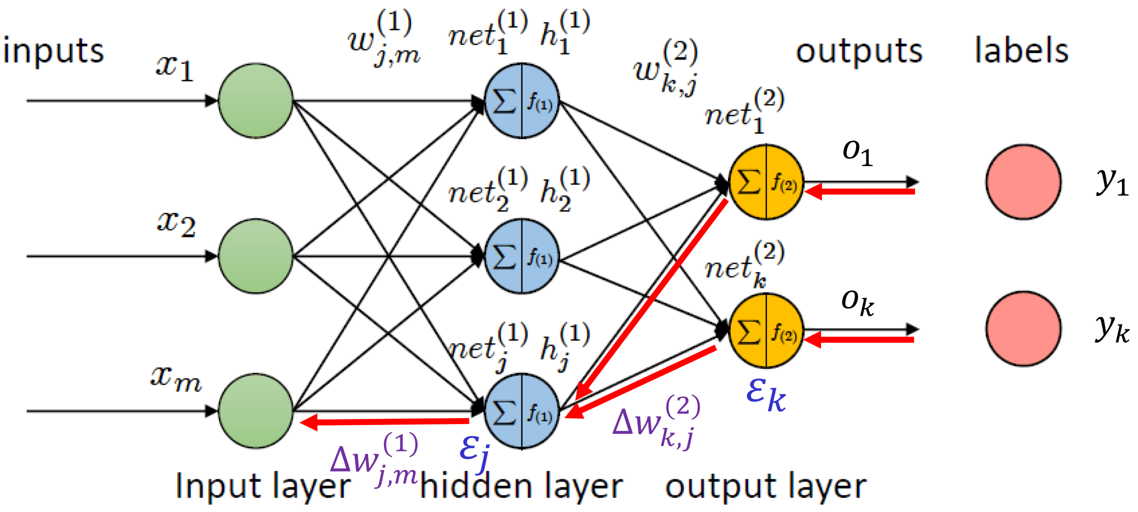
$$\varepsilon(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{K}(o_k - y_k)^2$$

Backprop to learn the parameters:

$$\begin{tikzcd} w^{(1)}_{j,m} & w^{(2)}_{k,j} \arrow[l] & \varepsilon(\mathbf{w}) \arrow[l, "\frac{\partial \varepsilon(\mathbf{w})}{\partial w^{(2)}_{k,j}} = \varepsilon_k h^{(1)}_j"'] \end{tikzcd}$$

$$\frac{\partial \varepsilon(\mathbf{w})}{\partial w^{(2)}_{k,j}} = \frac{\partial \varepsilon(\mathbf{w})}{\partial net^{(2)}_k} \frac{\partial net^{(2)}_k}{\partial w^{(2)}_{k,j}} = \boxed{\varepsilon_k h^{(1)}_j} \leftarrow \boxed{\Delta w^{(2)}_{k,j}}$$

Backward Propagation of Errors



Two-layer feedforward neural network

$$net^{(1)}_j = \sum_m w^{(1)}_{j,m} x_m$$

$$h^{(1)}_j = f_{(1)}\left(net^{(1)}_j\right)$$

$$net^{(2)}_k = \sum_j w^{(2)}_{k,j} h^{(1)}_j$$

$$o_k = f_{(2)}\left(net^{(2)}_k\right)$$

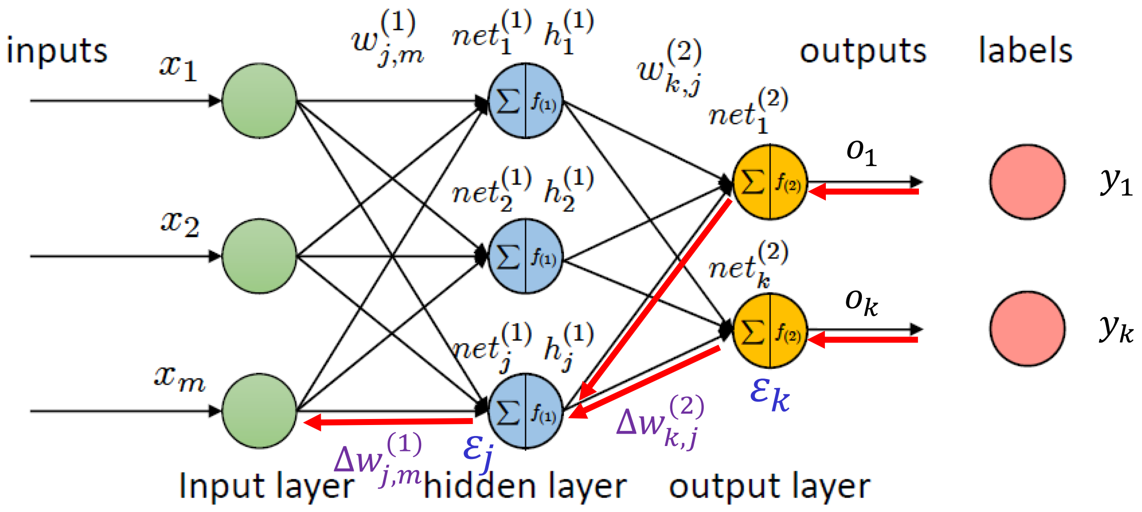
$$\varepsilon(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{K}(o_k - y_k)^2$$

Backprop to learn the parameters:

$$w^{(1)}_{j,m} \xleftarrow{\frac{\partial \varepsilon(\mathbf{w})}{\partial w^{(1)}_{j,m}}=\varepsilon_j x_m} w^{(2)}_{k,j} \xleftarrow{\frac{\partial \varepsilon(\mathbf{w})}{\partial w^{(2)}_{k,j}}=\varepsilon_k h^{(1)}_j} \varepsilon(\mathbf{w})$$

$$\frac{\partial \varepsilon(\mathbf{w})}{\partial w^{(1)}_{j,m}} = \frac{\partial \varepsilon(\mathbf{w})}{\partial net^{(1)}_j}\frac{\partial net^{(1)}_j}{\partial w^{(1)}_{j,m}} = \boxed{\varepsilon_j x_m} \xleftarrow{\quad} \boxed{\Delta w^{(1)}_{j,m}}$$

Backpropagation (BP) Algorithm



Two-layer feedforward neural network

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$$

$$h_j^{(1)} = f_{(1)}\left(net_j^{(1)}\right)$$

$$net_k^{(2)} = \sum_j w_{k,j}^{(2)} h_j^{(1)}$$

$$o_k = f_{(2)}\left(net_k^{(2)}\right)$$

$$\varepsilon(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{K}(o_k - y_k)^2$$

Update the parameters:

Error of k

$$w_{k,j}^{(2)\prime} = w_{k,j}^{(2)} - \eta\Delta w_{k,j}^{(2)} = w_{k,j}^{(2)} - \boxed{\eta}\boxed{\varepsilon_k}\boxed{h_j^{(1)}}$$

Output of j

Learning rate

$$w_{j,m}^{(1)\prime} = w_{j,m}^{(1)} - \eta\Delta w_{j,m}^{(1)} = w_{j,m}^{(1)} - \boxed{\eta}\boxed{\varepsilon_j}\boxed{x_m}$$

Output of m

Error of j

Chain Rule

- Update the weight on the link from node j to i

$$w_{i,j}{}' = w_{i,j} - \eta \varepsilon_i h_j$$

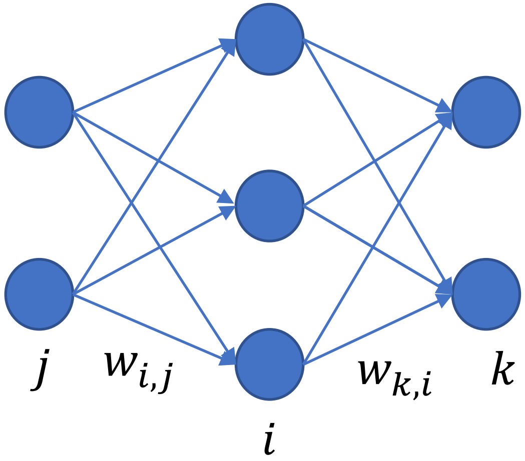
  - h_j is the output of node j

- If node i is in output layer

$$\varepsilon_i = (o_i - y_i) f'^{i}$$

- If node i is in hidden layer

$$\varepsilon_i = \sum_{k=1}^{K} \varepsilon_k \, w_{k,i} f'^{i}$$

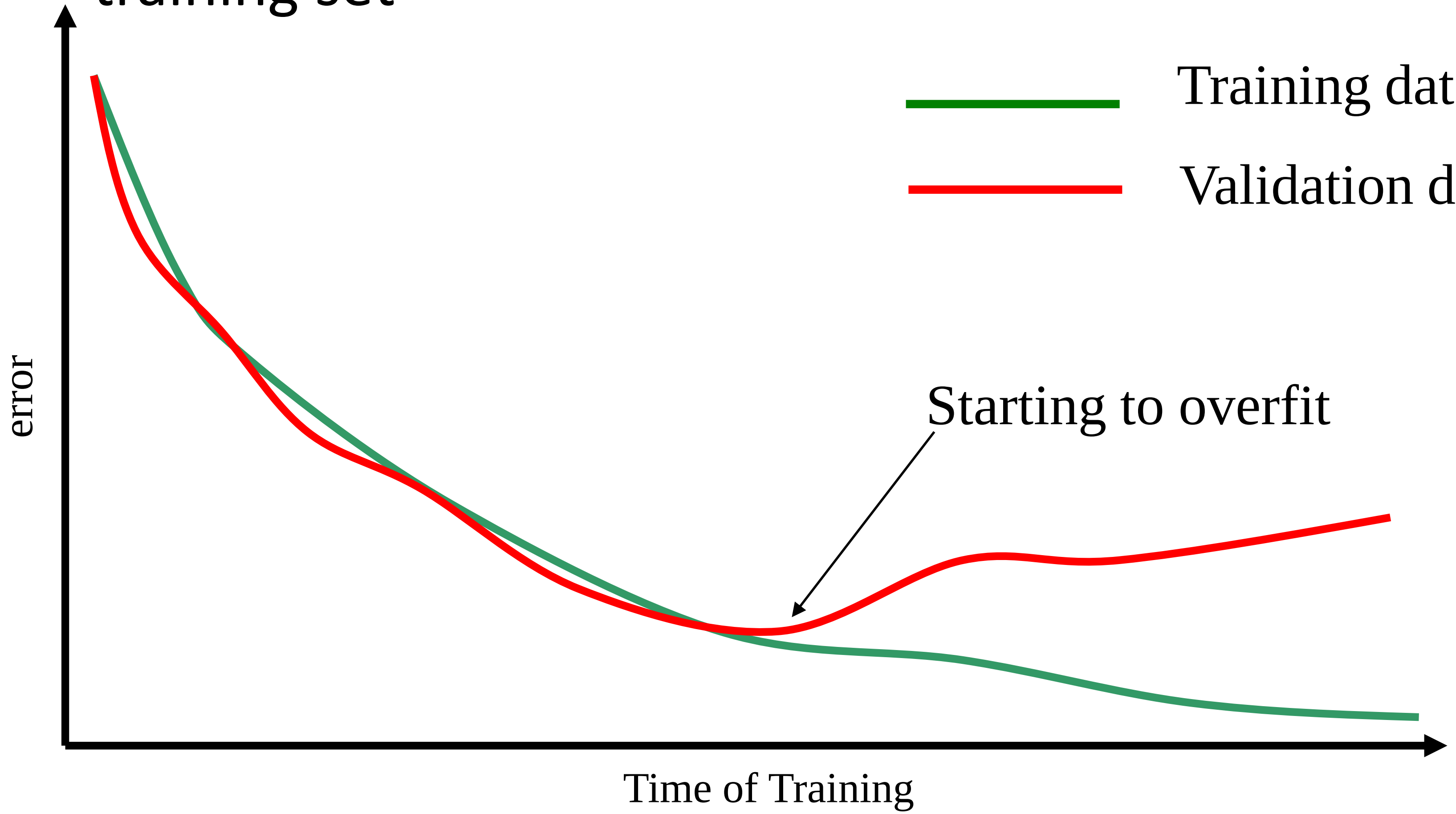


Repeating updates, after a number of iterations, we will get in the (local) optima

Overfitting

Avoiding Overfitting

- A typical curve showing performance during training
- But here is performance on unseen data, **not** in the training set



Avoiding Overfitting

- Use as much training data as possible, and ensure as much diversity as possible in the data
  - This cuts down on the potential existence of features that might be discriminative in the training data, but are otherwise spurious

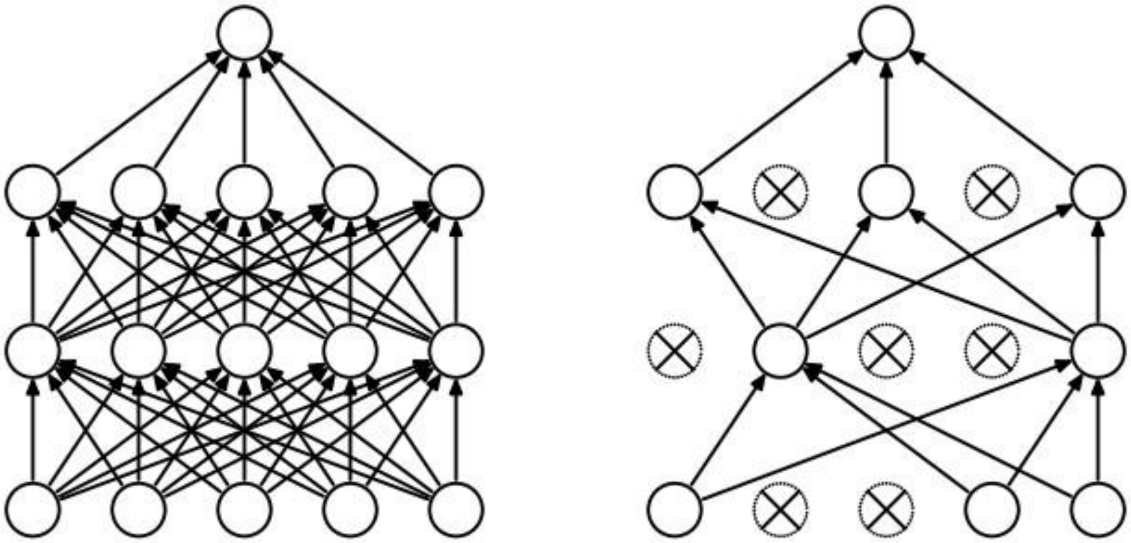
- Jittering (adding noise)
  - During training, every time an input pattern is presented, it is randomly perturbed
  - The idea of this is that spurious features will be `washed out’ by the noise, but valid discriminatory features will remain
  - The problem with this approach is how to correctly choose the level of noise

Avoiding Overfitting (cont.)

- Early stopping
  - During training, keep track of the network's performance on a separate *validation set* of data
  - At the point where error continues to improve on the training set, but starts to get worse on the validation set, that is when training should be stopped, since it is starting to overfit on the training data
  - The problem here is that this point is far from always clear cut

Avoiding Overfitting (cont.)

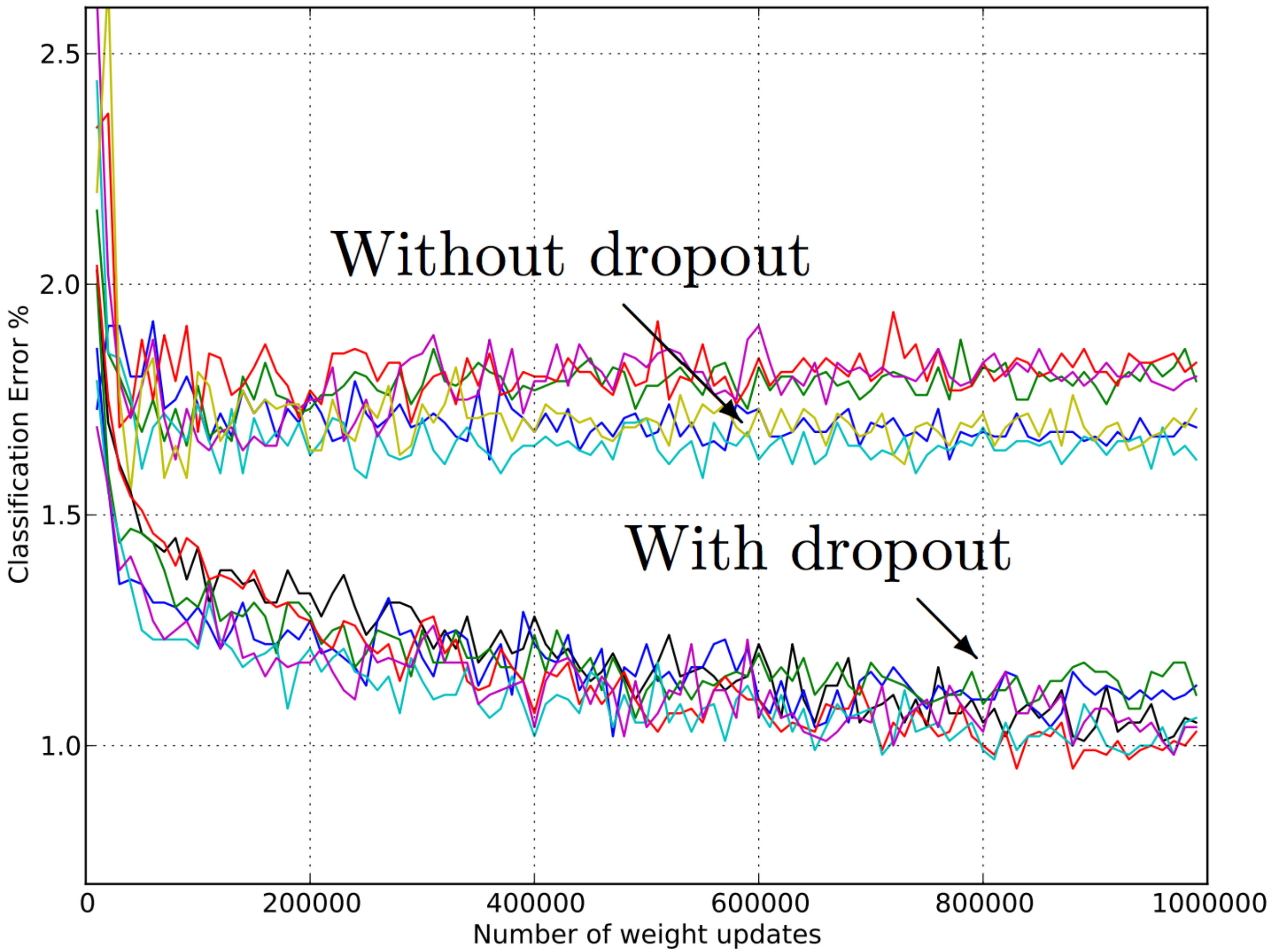
- Regularization
  - Limit the summation of the weights
- Dropout
  - In training, for each sample, dropout a neuron with some probability



(a) Standard Neural Net

(b) After applying dropout.

Dropout on an image classification task



NN Playground

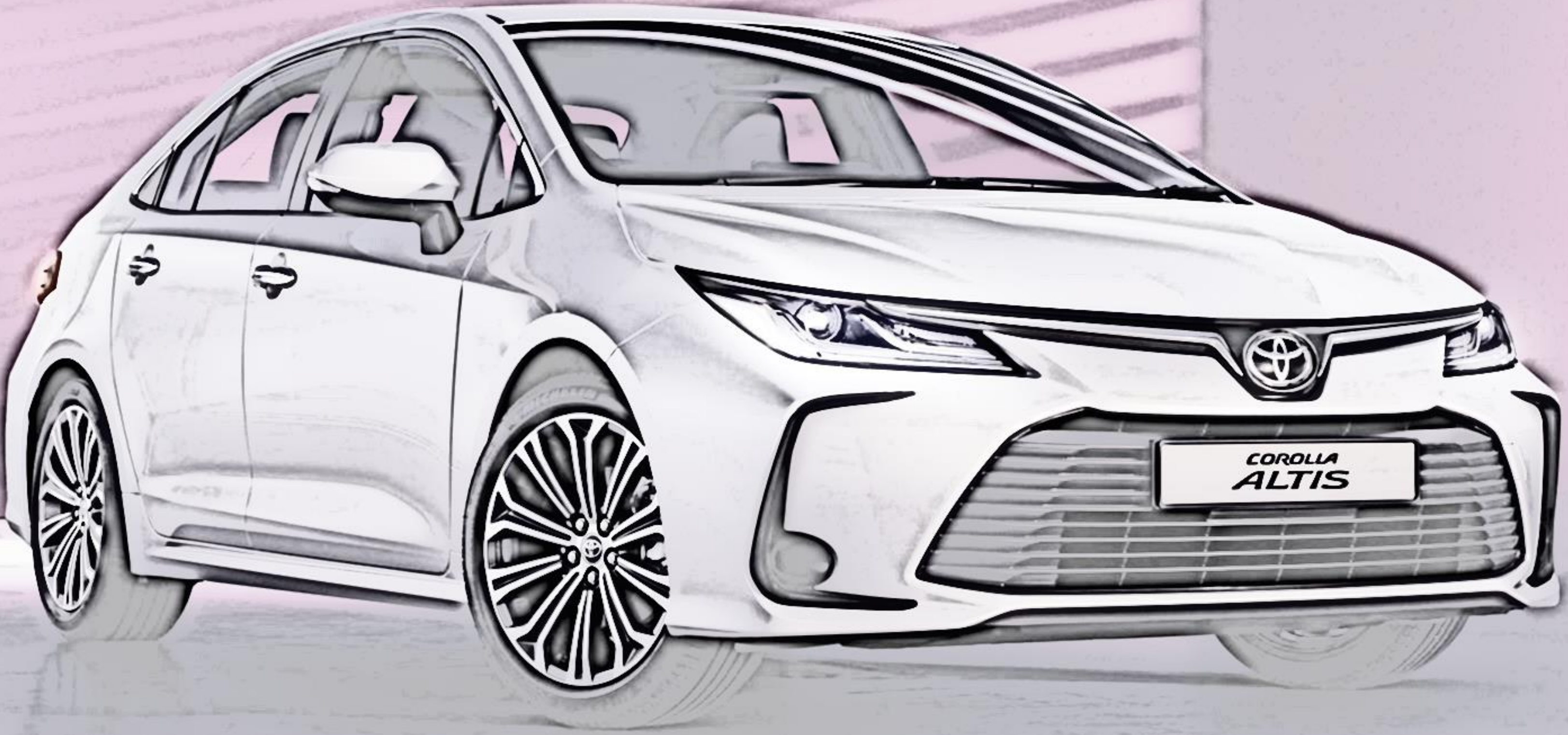
- playground.tensorflow.org

Application

Toyota Corolla

Toyota Corolla

- A line of subcompact and compact cars
- Introduced in 1966
- Best selling nameplate in the world



Application

Used Car Valuation

- If you have a car to sell or want to buy a used car
- Knowing how to calculate the estimated value of the car is paramount in making sure that you get the best value for your money

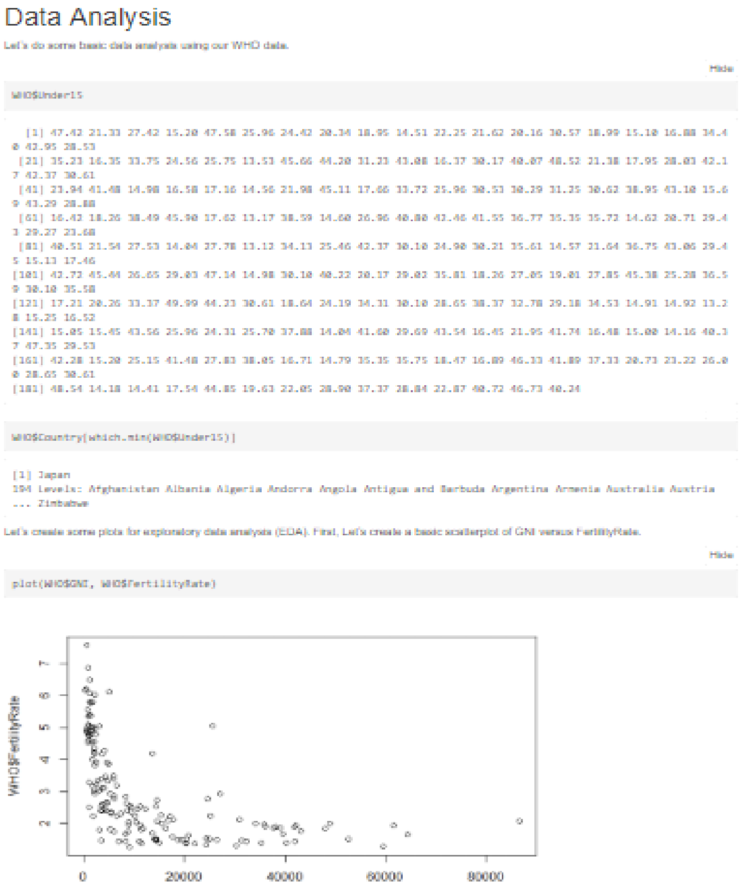
Data

- On used Toyota Corolla
- 1436 record
- 38 attributes
  - Price
  - Age
  - KM
  - HP
  - …

Goal:
Predict the price of a used Toyota Corolla

Method:
Neural Network

Let’s get our hands dirty!



Data Analysis

Let's do some basic data analysis using our WHO data.

```
WHO$Under15
```

```
  [1] 47.42 21.33 27.42 15.20 47.58 25.96 24.42 20.34 18.95 14.51 22.25 21.62 20.16 30.57 18.99 15.10 16.88 34.4
0 42.95 28.53
 [21] 35.23 16.35 33.75 24.56 25.75 13.53 45.66 44.20 31.23 43.08 16.37 30.17 40.07 48.52 21.38 17.95 28.03 42.1
7 42.37 30.61
 [41] 23.94 41.48 14.98 16.58 17.16 14.56 21.98 45.11 17.66 33.72 25.96 30.53 30.29 31.25 30.62 38.95 43.10 15.6
9 43.29 28.88
 [61] 16.43 18.26 38.49 45.90 17.62 13.17 38.59 14.60 26.96 40.80 42.46 41.55 36.77 35.35 35.73 14.62 20.71 29.4
3 29.27 23.68
 [81] 40.51 21.54 27.53 14.04 27.78 13.12 34.13 25.46 42.37 30.10 24.90 30.21 35.61 14.57 21.64 36.75 43.06 29.4
5 15.13 17.46
[101] 42.73 45.44 26.65 29.03 47.14 14.98 30.10 40.22 20.17 39.02 35.81 18.26 27.05 19.01 27.85 45.38 25.28 36.5
9 30.10 35.58
[121] 17.21 20.26 33.37 49.99 44.23 30.61 18.64 24.19 34.31 30.10 28.65 38.37 32.78 29.18 34.53 14.91 14.92 13.2
8 15.25 16.52
[141] 15.05 15.45 43.56 25.96 24.31 25.70 37.88 14.04 41.60 29.69 43.54 16.45 21.95 41.74 16.48 15.00 14.16 40.3
7 47.35 29.53
[161] 42.28 15.20 25.15 41.48 27.83 38.05 16.71 14.79 35.35 15.75 18.47 16.89 46.33 41.89 37.33 20.73 23.22 26.0
0 28.65 30.61
[181] 48.54 14.18 14.41 17.54 44.85 19.63 22.05 28.90 37.37 28.84 22.87 40.72 46.73 40.24
```

```
WHO$Country[which.min(WHO$Under15)]
```

```
[1] Japan
194 Levels: Afghanistan Albania Algeria Andorra Angola Antigua and Barbuda Argentina Armenia Australia Austria
... Zimbabwe
```

Let's create some plots for exploratory data analysis (EDA). First, Let's create a basic scatterplot of GNI versus FertilityRate.

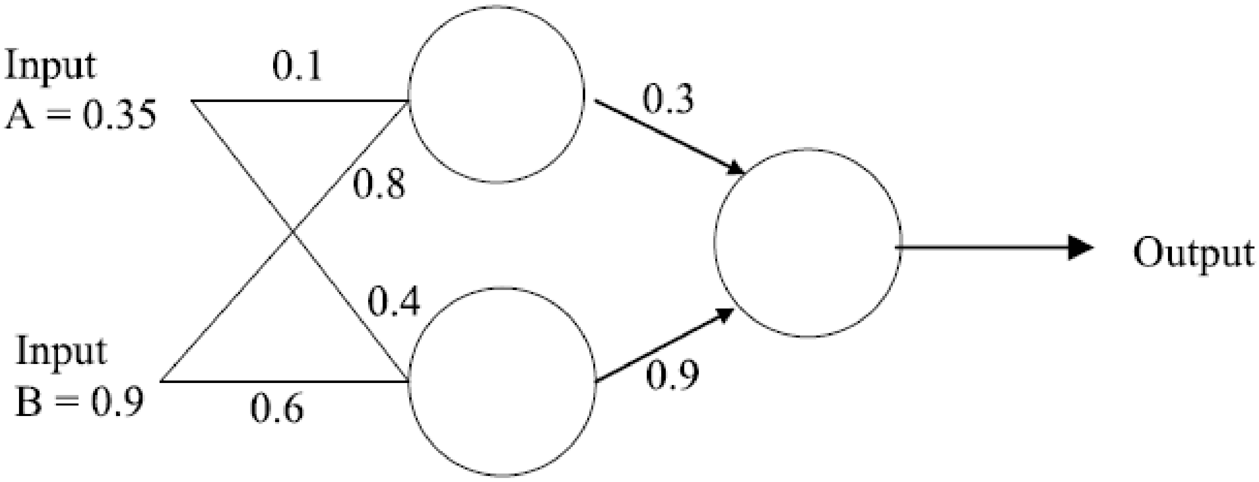
```
plot(WHO$GNI, WHO$FertilityRate)
```

Lecture 9 wrap-up

- ✓One-layer Perceptron
- ✓Multi-layer Perceptron
  - ✓Model
  - ✓Strategy
  - ✓Algorithm
- ✓Overfitting
- ✓Application: Toyota Corolla

Assignment 9

- Consider the simple network below:



- Assume that the neurons have a Sigmoid activation function and
  - 1. Perform a forward pass on the network
  - 2. Perform a reverse pass (training) once (target = 0.5)
  - 3. Perform a further forward pass and comment on the result

- i.e.,
  - 1. Calculate the input and output for each neuron
  - 2. Calculate the new weights for each layer after once training
  - 3. Calculate the new error and compare

- Send your answer to TA (any form, e.g., word, pdf, photo …)

- Due: June 5, 11pm
- TA: Mr. Xiong, xiongym3@mail2.sysu.edu.cn

Next lecture

- Supervised learning
  - Linear regression
  - Logistic regression
  - SVM and kernel
  - Tree models
- Deep learning
  - Neural networks
  - Convolutional NN
  - Recurrent NN
- Unsupervised learning
  - Clustering
  - PCA
  - EM
- Reinforcement learning
  - MDP
  - ADP
  - Deep Q-Network

Questions?

Shan Wang (王杉)

https://wangshan731.github.io/