CE9010: Introduction to Data Science

Lecture 10: Neural Networks

Semester 2 2017/18

Xavier Bresson

School of Computer Science and Engineering
Data Science and AI Research Centre
Nanyang Technological University (NTU), Singapore

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Outline

- Motivation
- Brain inspiration
- Neurons and connections
- Neural networks with multiple layers
- Learned features
- Logical gates with neural networks
- Logistic regression loss
- Backpropagation
- Initialization
- Training neural networks
- Conclusion

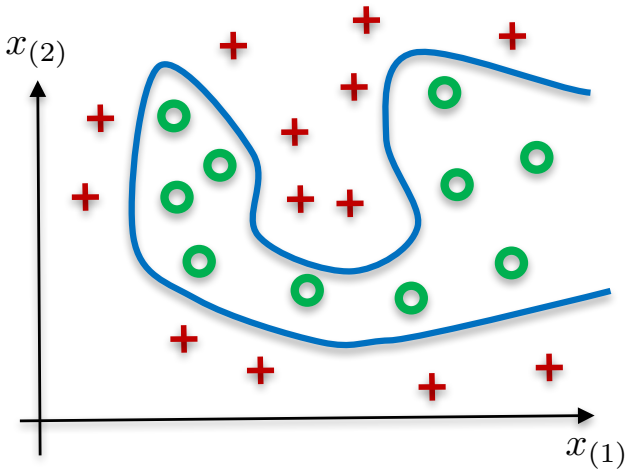
Outline

- **Motivation**
- Brain inspiration
- Neurons and connections
- Neural networks with multiple layers
- Learned features
- Logical gates with neural networks
- Logistic regression loss
- Backpropagation
- Initialization
- Training neural networks
- Conclusion

High learning capacity with polynomials

- Learning complex predictive functions:

Logistic regression: A higher order polynomial function is required to learn the statistics of these data:



$$p_w(x) = \sigma(w_0 + w_1 x_{(1)} + w_2 x_{(2)} + \ldots + w_{p^2} x_{(2)}^p)$$

How many terms for polynomials of degree 10 and (only) 2 features?

100

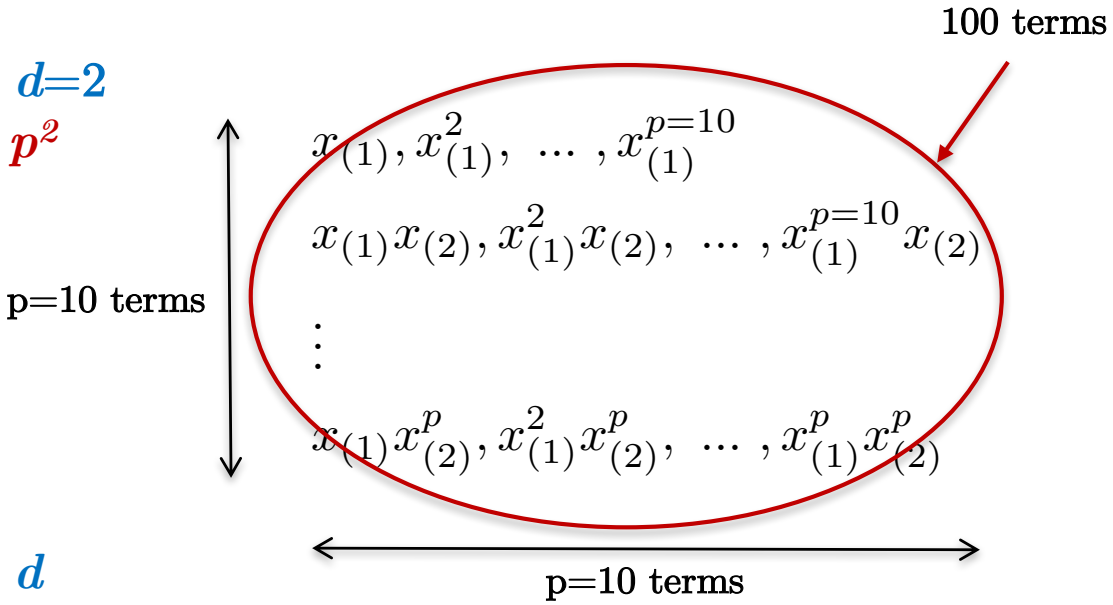
Exponential explosion

- Nb of polynomial terms for d=1 features and degree p=10: p

$$x_{(1)}, x_{(1)}^2, \ \dots\ , x_{(1)}^{p=10}$$

p=10 terms

- Nb of polynomial terms for d=2 features and degree p=10: p^2



- Nb of polynomial terms for d features and degree p: p^d

$\Rightarrow$ The number of polynomial grows exponentially with the number d of data features.

Smaller number of features

- Learning polynomial functions with a large number d of features is computationally expensive (intractable) as the number of parameters w to learn also grows exponentially with d.

- Data with high number of features are common.

  Example: Images have 1024 x 1024 pixels

  $\Rightarrow d=10^6$ (1M) features

  $\Rightarrow$ Nb of polynomial terms for degree 10 is $p^d=10^{1,000,000}$ terms!

- Potential solution: Select a smaller number of polynomial features but which ones? We would need to know before hand the geometry of the decision boundary, which is generally never the case $\Rightarrow$ Not a solution.

- Require a new approach to learn high capacity function for large number of data features $\Rightarrow$ Neural networks

Outline

- Motivation
- **Brain inspiration**
- Neurons and connections
- Neural networks with multiple layers
- Learned features
- Logical gates with neural networks
- Logistic regression loss
- Backpropagation
- Initialization
- Training neural networks
- Conclusion

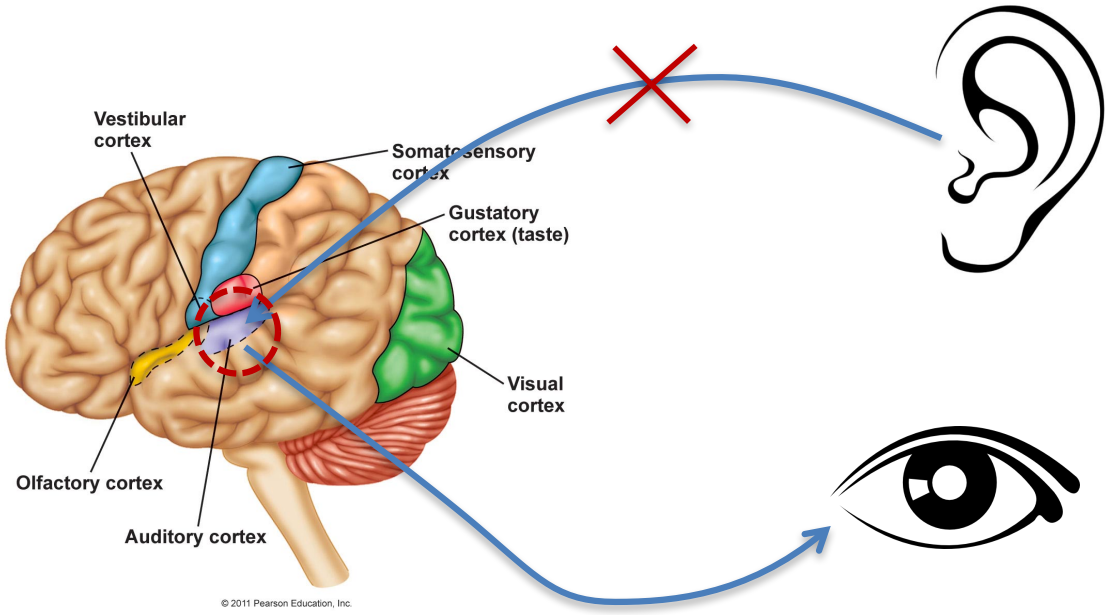
Neural networks

- Neural networks (NNs) are loosely inspired by the biological brain (very simplified).
- NNs started in the 50's with the Nobel prize of medicine of Hubel and Wiesel who decoded the first two layers of the cortex visual system. From this, computer scientists dreamed to give vision to machine by simulating it on computers, and more (mimic the human brain).
- Preliminary results were fine, but not better, slower and less math grounded than competitive techniques. People were more interested in e.g. SVM, random forest, LDA techniques from the 90's to 2012.
- After 2000, computers became more powerful (Moore's law) and emergence of big data (e.g. smartphone). In 2012, NNs were born again (and called deep learning) and overcome by a large margin existing techniques.
- NNs are the state-of-the art techniques for all learning problems.

The brain system

- Biological NNs form a powerful system to learn perceptual tasks: seeing, hearing, speaking, smelling, touching.
- Biological experiments were developed to demonstrate the impressive learning ability of the brain:
  - Neuroscientists surgically cut the connection between the auditory cortex and the ear, and re-routed the connection from the eye to the auditory cortex

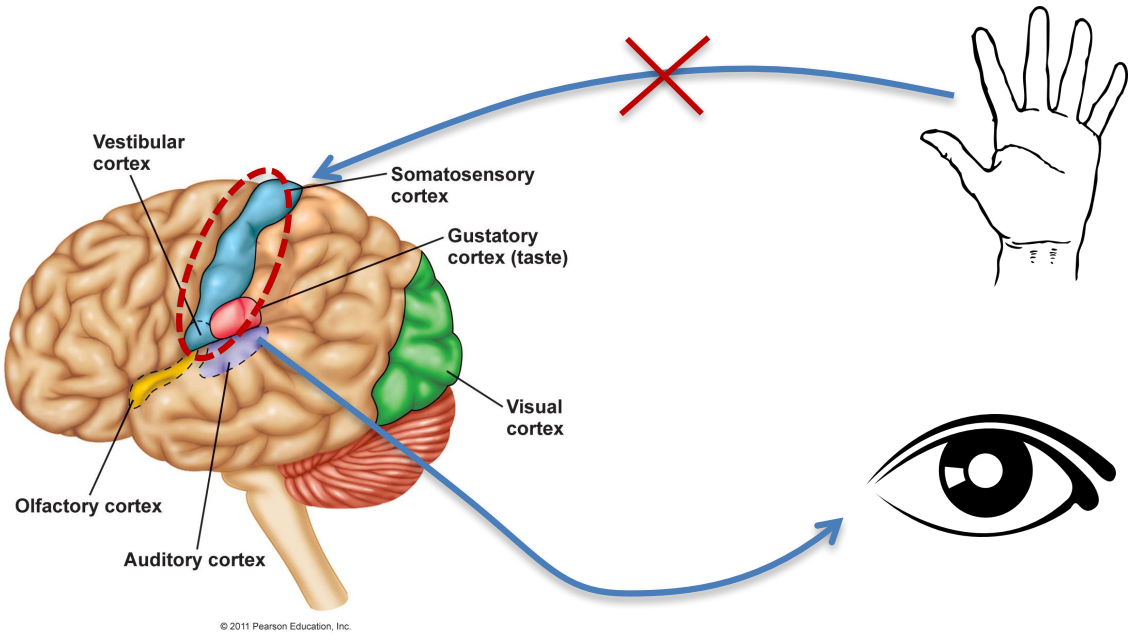
    $\Rightarrow$ The auditory cortex learns to see!



The brain system

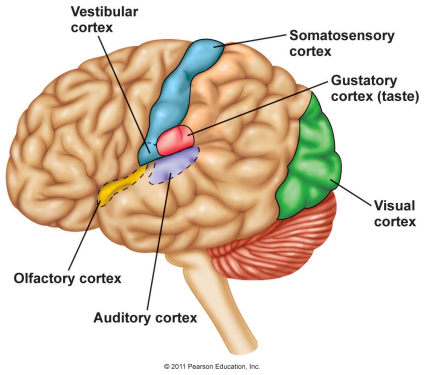
- A similar experiment: The connection between the somatosensory cortex and the sensory touch, and re-routed the connection from the eye to the somatosensory cortex

  $\Rightarrow$ The somatosensory cortex learns to see.



The brain "learning algorithm"

- This is a potential evidence that the brain has a generic "learning algorithm" as different parts of the brain can learn the same task.

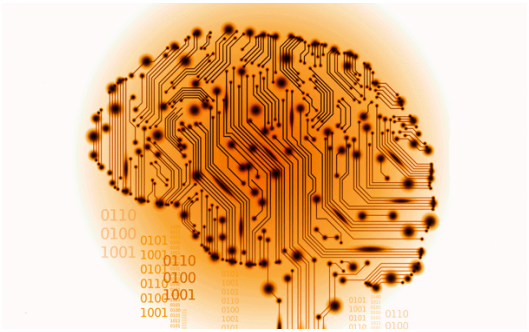
Data → 

→ Task

Biological brain

Proof of intelligent learning system (lower bound of intelligence)

Perceptual intelligence (vision is easy) vs. cognitive intelligence (math is hard)

Data →  → Task

Artificial brain

The path and goal of all learning systems today

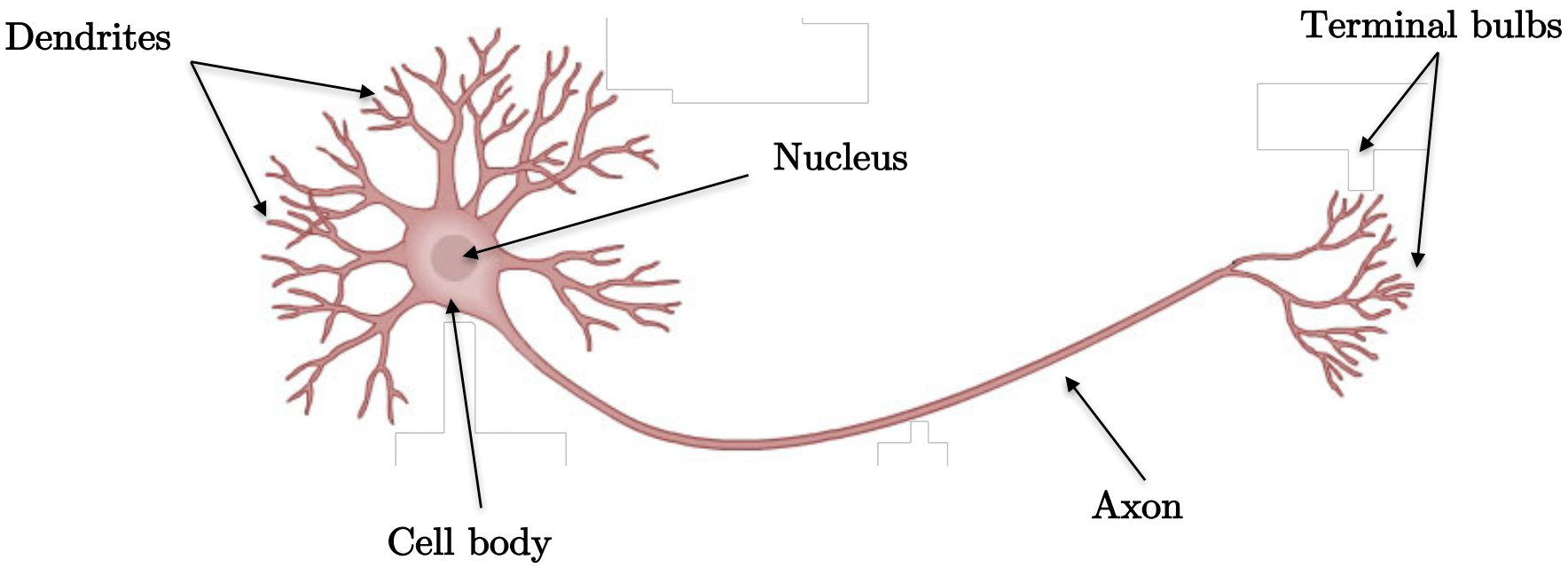
Outline

- Motivation
- Brain inspiration
- **Neurons and connections**
- Neural networks with multiple layers
- Learned features
- Logical gates with neural networks
- Logistic regression loss
- Backpropagation
- Initialization
- Training neural networks
- Conclusion

Neurons and axons

- A human brain has 10^{11} neurons and 10^{14} axons (connection between neurons).

- A basic neuron:

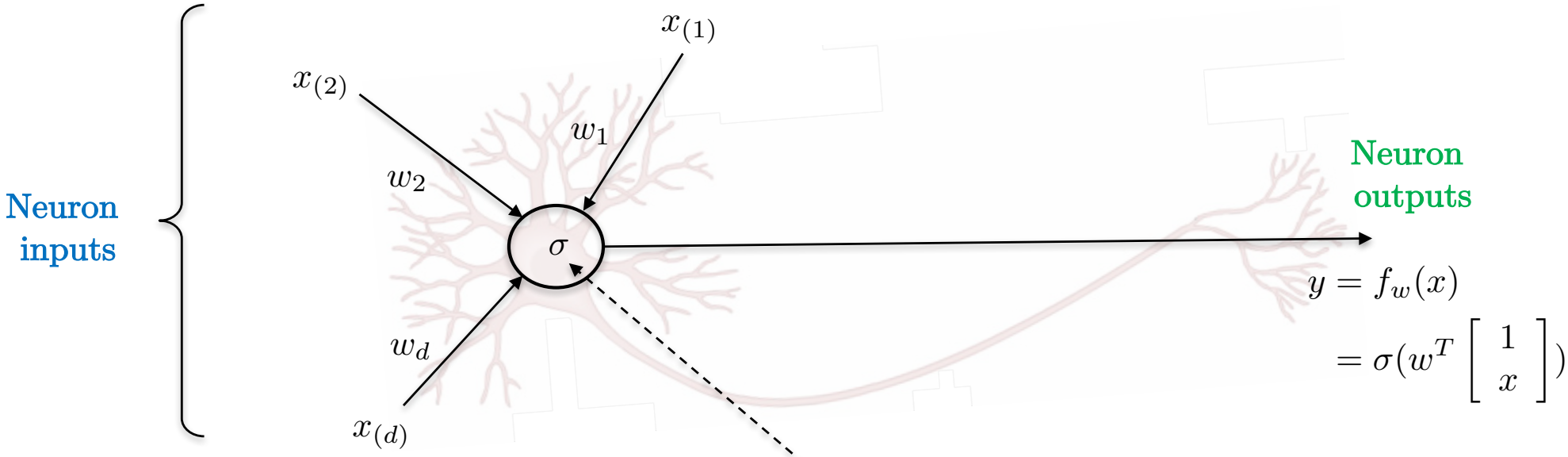


Neurons and axons

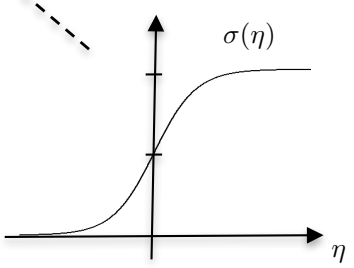
- Brain = architecture/group of connected neurons
  = neural networks

Neuron modeling

- (Non-)linear neural model:



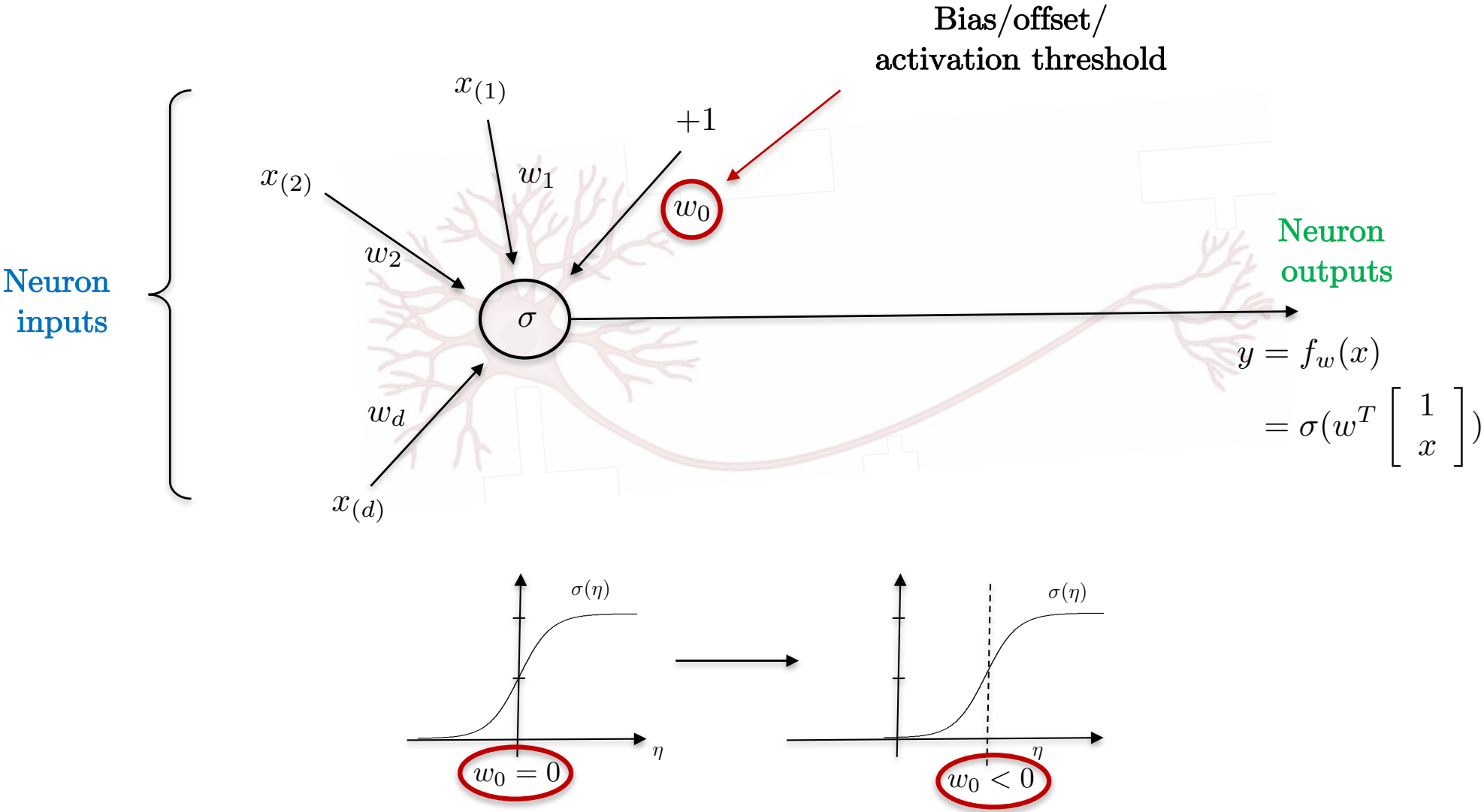
$$x = \begin{bmatrix} x_{(1)} \\ \vdots \\ x_{(d)} \end{bmatrix} \qquad w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$$



Activation function sigmoid σ

Bias unit

- (Non-)linear neural model:

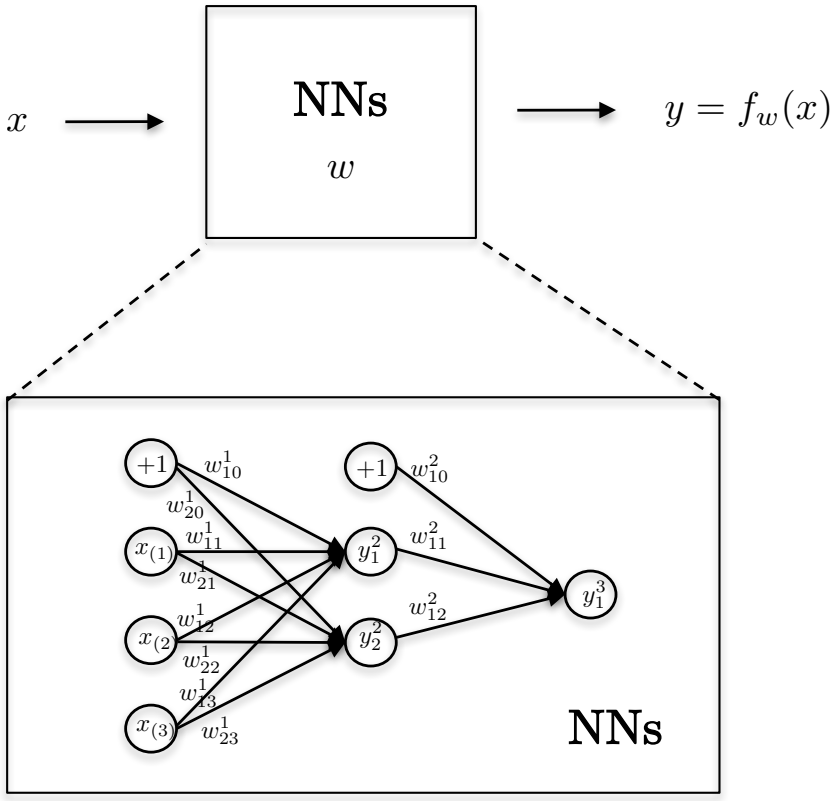


Outline

- Motivation
- Brain inspiration
- Neurons and connections
- **Neural networks with multiple layers**
- Learned features
- Logical gates with neural networks
- Logistic regression loss
- Backpropagation
- Initialization
- Training neural networks
- Conclusion

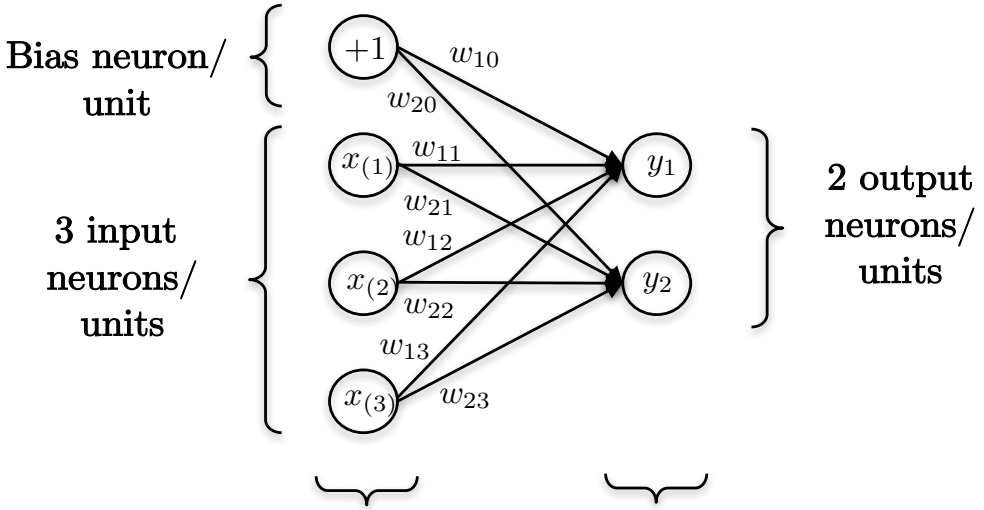
Neural network learning function

- Neural networks learn a mapping f_w from input data feature x to output vector y:



2-layer neural network

- One input layer and one output layer:



$$x = \begin{bmatrix} x_{(1)} \\ x_{(2)} \\ x_{(3)} \end{bmatrix}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \sigma(W \begin{bmatrix} 1 \\ x \end{bmatrix}) = f_w(x)$$

Output

Predictive function

2-layer neural network

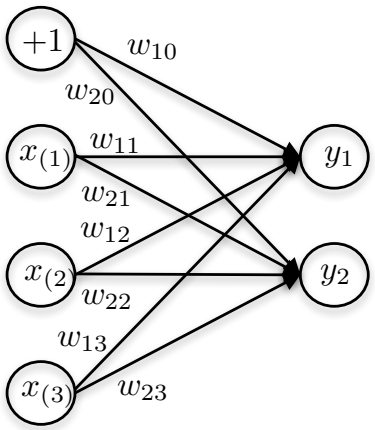
- Activation equations for the output layer:
  - Element/neuron-wise activation equations:

$$y_1 = \sigma(w_{10} + w_{11}x_{(1)} + w_{12}x_{(2)} + w_{13}x_{(3)})$$
$$y_2 = \sigma(w_{20} + w_{21}x_{(1)} + w_{22}x_{(2)} + w_{23}x_{(3)})$$

  - Vectorized activation equations:

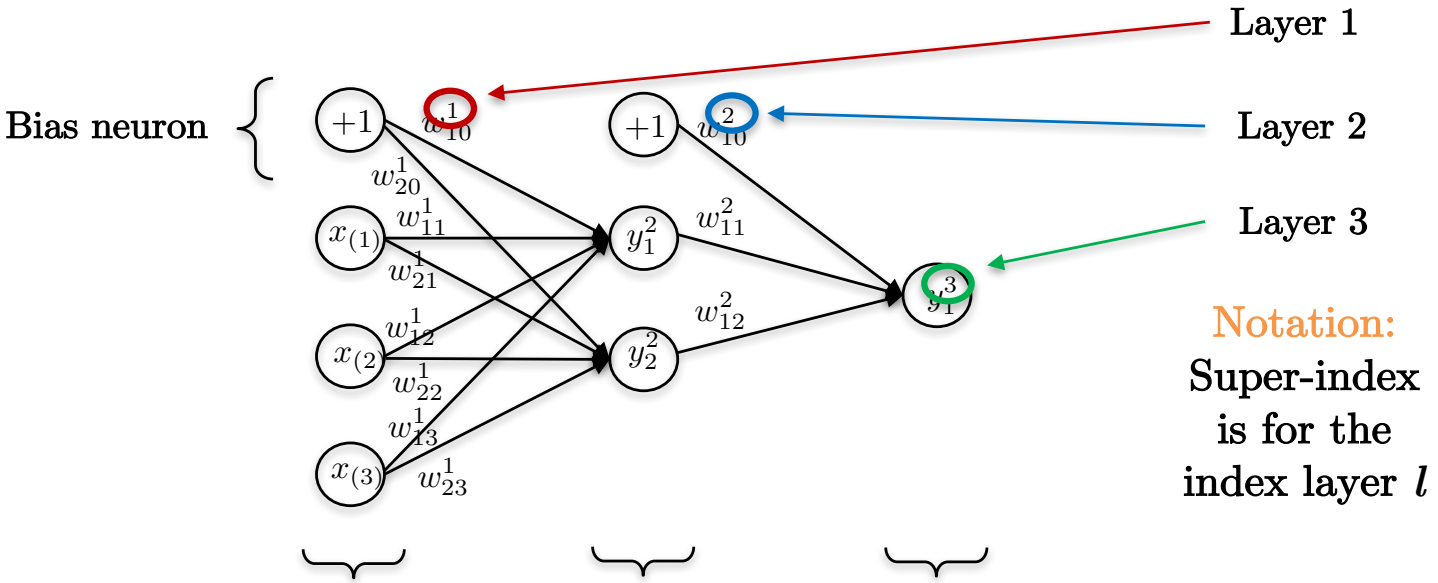
$$y = \sigma(W \begin{bmatrix} 1 \\ x \end{bmatrix}) \qquad W = \begin{bmatrix} w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \end{bmatrix}$$

$$x = \begin{bmatrix} x_{(1)} \\ x_{(2)} \\ x_{(3)} \end{bmatrix}$$



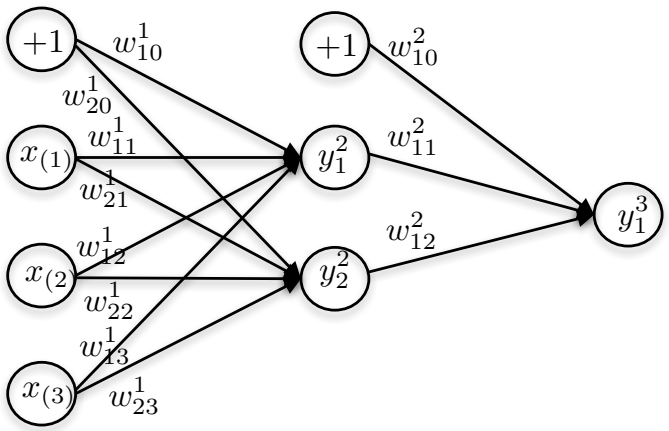
3-layer neural network

- One input layer, one hidden layer and one output layer:



3-layer neural network

- Activation equations for the hidden layer and the output layer:
  - Element/neuron-wise activation equations:
    - **Layer 2:** $y_1^2 = \sigma(w_{10}^1 + w_{11}^1 x_{(1)} + w_{12}^1 x_{(2)} + w_{13}^1 x_{(3)})$
      $y_2^2 = \sigma(w_{20}^1 + w_{21}^1 x_{(1)} + w_{22}^1 x_{(2)} + w_{23}^1 x_{(3)})$
    - **Layer 3:** $y_1^3 = \sigma(w_{10}^1 + w_{11}^1 y_1^2 + w_{12}^1 y_2^2)$
  - Vectorized activation equations:
    - **Layer 2:** $y^2 = \sigma(W^1 \begin{bmatrix} 1 \\ y^1 \end{bmatrix})$
    - **Layer 3:** $y^3 = \sigma(W^2 \begin{bmatrix} 1 \\ y^2 \end{bmatrix})$



$$W^1 = \begin{bmatrix} w_{10}^1 & w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{20}^1 & w_{21}^1 & w_{22}^1 & w_{23}^1 \end{bmatrix}$$

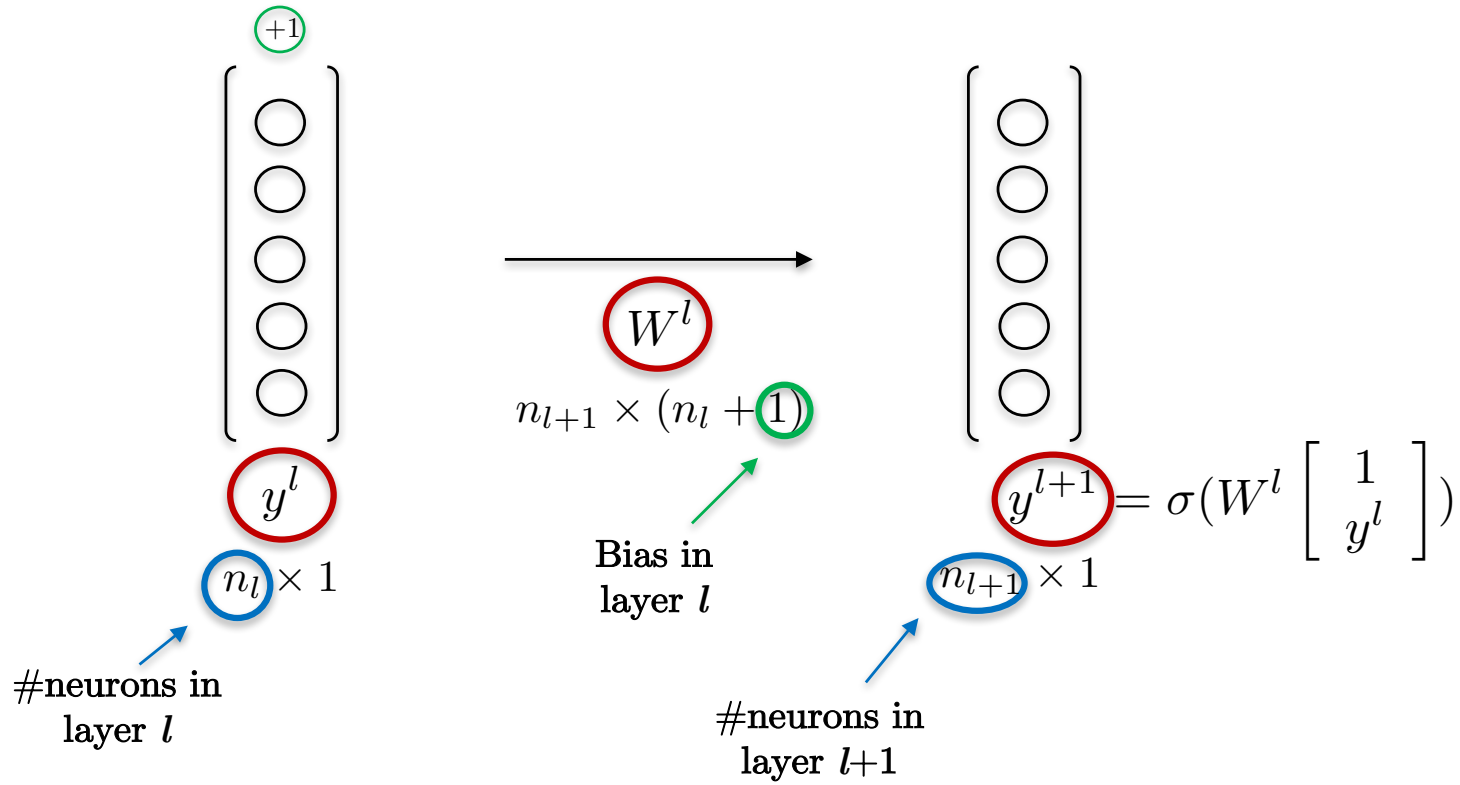
$$W^2 = \begin{bmatrix} w_{10}^2 & w_{11}^2 & w_{12}^2 \end{bmatrix}$$

Weight matrices

- **Notation:**

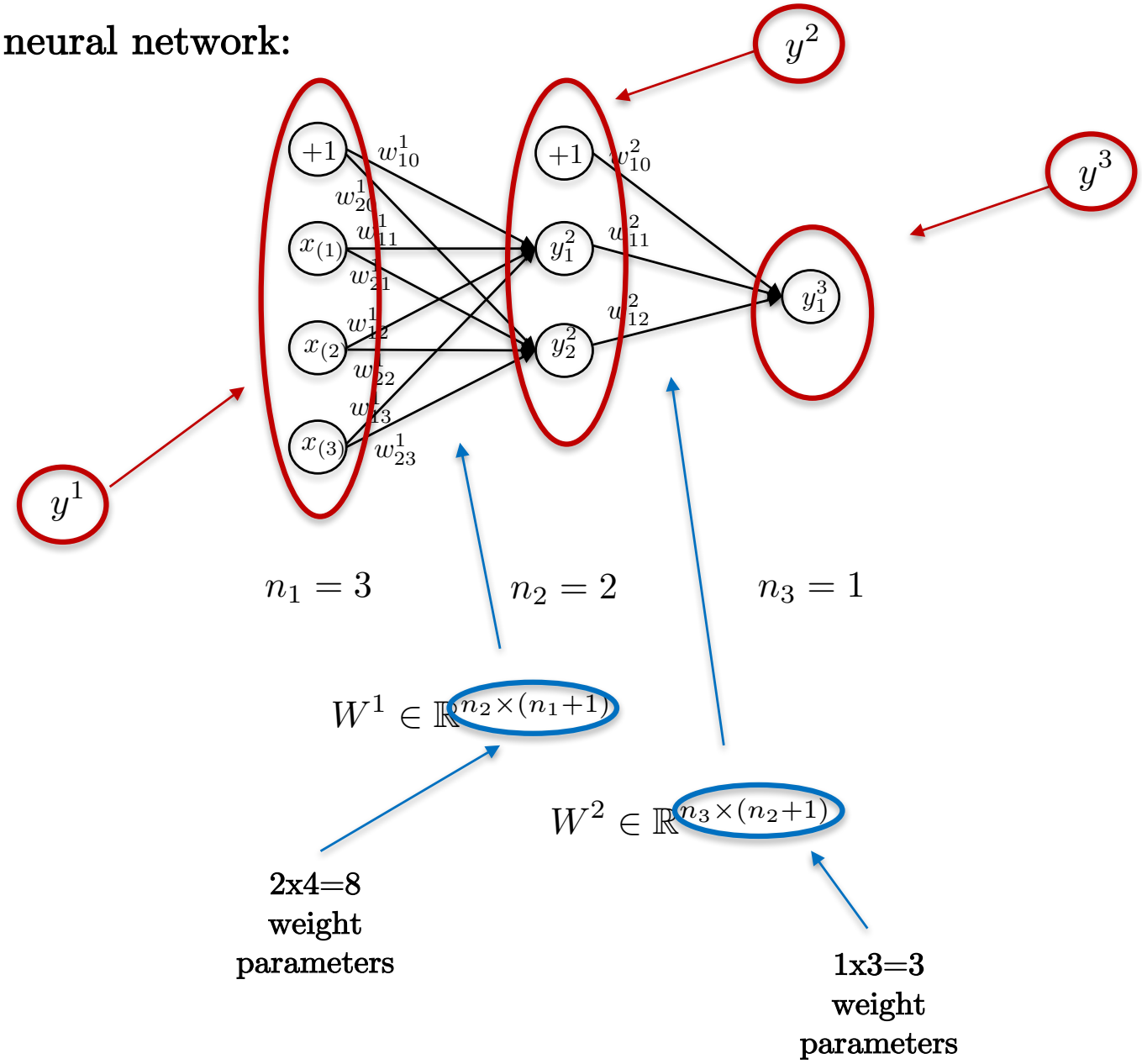
y_i^l : Activation of the i^{th} neuron at the l^{th} layer

W^l : Weight parameter matrix between layer l and layer l+1.
It defines the mapping from layer l to layer l+1:



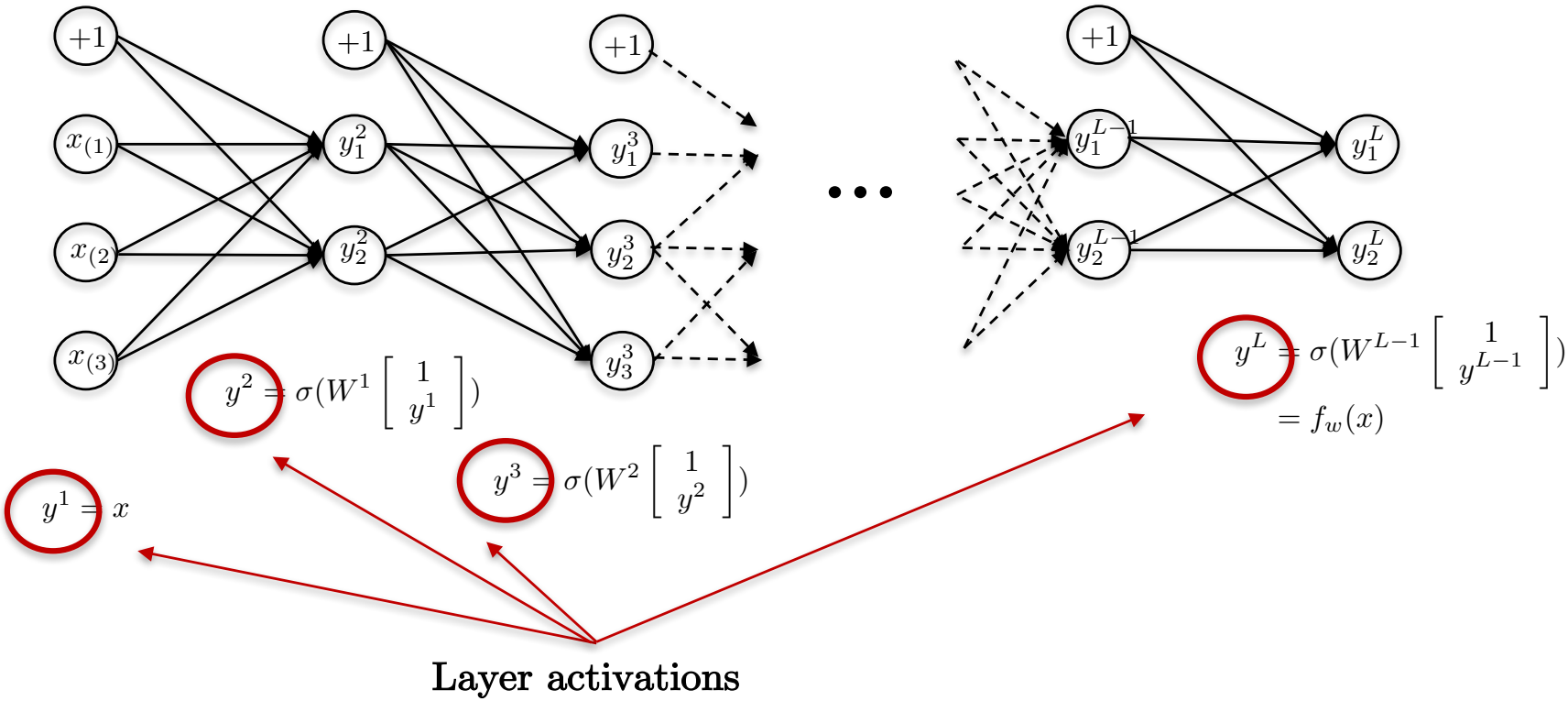
Example

- 3-layer neural network:



Deep neural networks

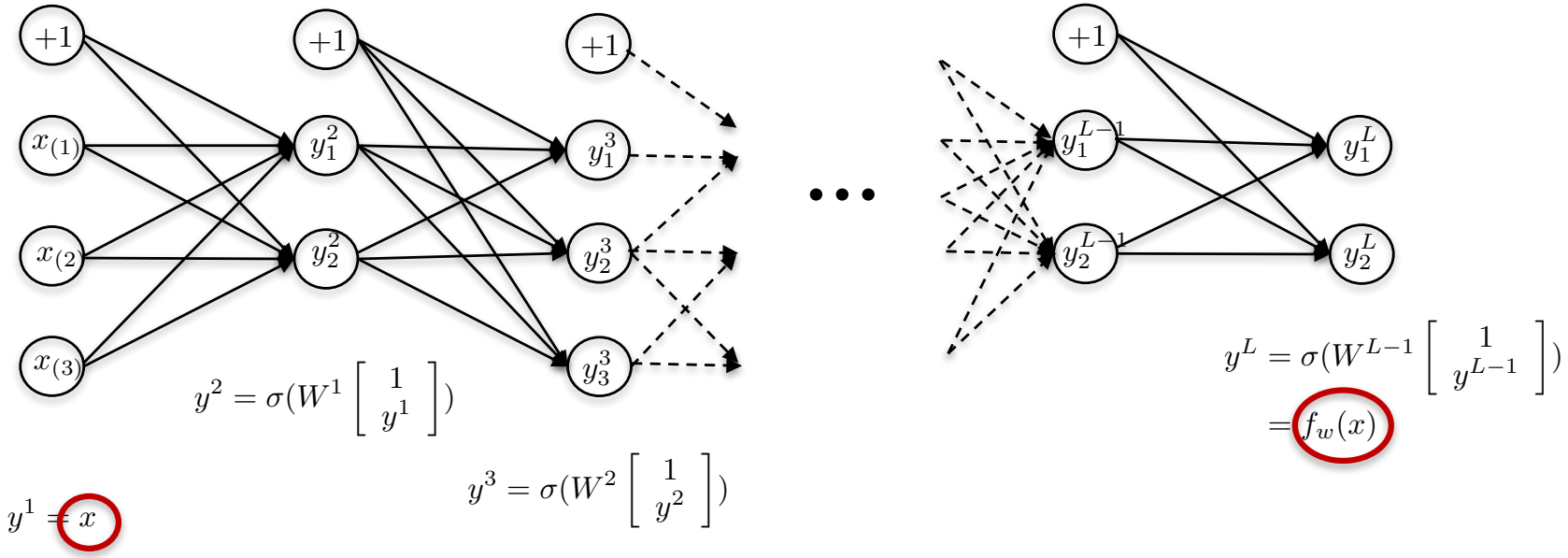
- Neural network with multiple layers (a.k.a. deep learning):



The parameters to learn are the weight matrices: $W^1, W^2, ..., W^L$

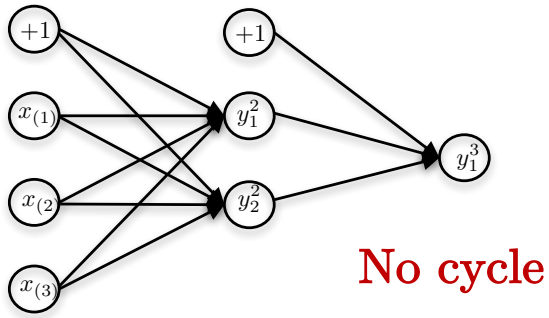
Forward pass

- The neural network activations y^l , l=1,2,...,L are computed from left to right, from input data x to output prediction $f_w(x)$. This is called the forward pass:

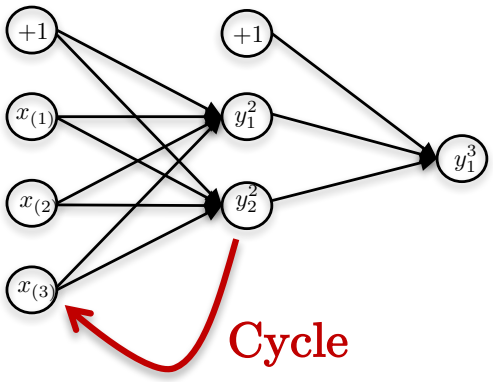


Feedforward neural networks

- The neural network architectures without loop connections (i.e. cycles) are called feedforward NNs. Examples are fully connected neural networks and convolutional neural networks.



- The neural network architectures with loop connections (i.e. cycles) are called recurrent NNs:

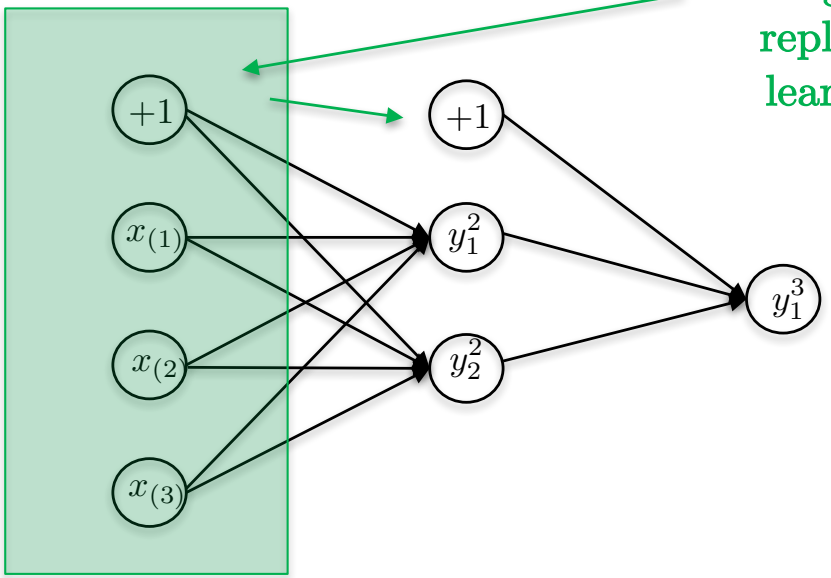


Outline

- Motivation
- Brain inspiration
- Neurons and connections
- Neural networks with multiple layers
- **Learned features**
- Logical gates with neural networks
- Logistic regression loss
- Backpropagation
- Initialization
- Training neural networks
- Conclusion

Hidden layers

- NNs learn new features with the hidden layers:



Original data features x are replaced by new features y^2 learned from training data.

Key property: The new features y^2 are learned from the training data $\Rightarrow$ NNs learn their own data features to solve the data analysis task (e.g. regression, classification) the best possible way.

Powerful paradigm called end-to-end learning systems.

Original/ raw features

Learned features

Predictive function

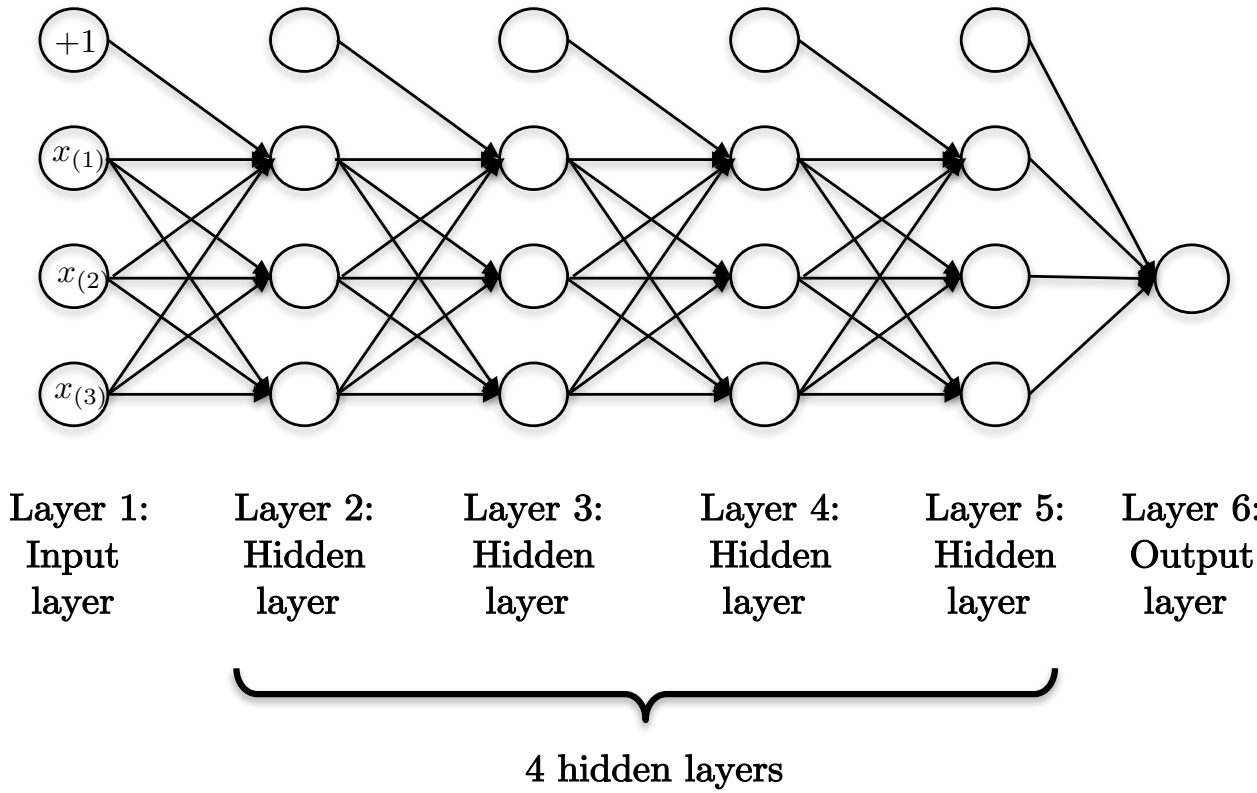
Input layer

Hidden layer

Output layer

Hidden layers

- The deeper the better: Deep neural networks are able to learn highly meaningful/abstract data features/patterns that can capture complex statistics of data.



Outline

- Motivation
- Brain inspiration
- Neurons and connections
- Neural networks with multiple layers
- Learned features
- **Logical gates with neural networks**
- Logistic regression loss
- Backpropagation
- Initialization
- Training neural networks
- Conclusion

AND function

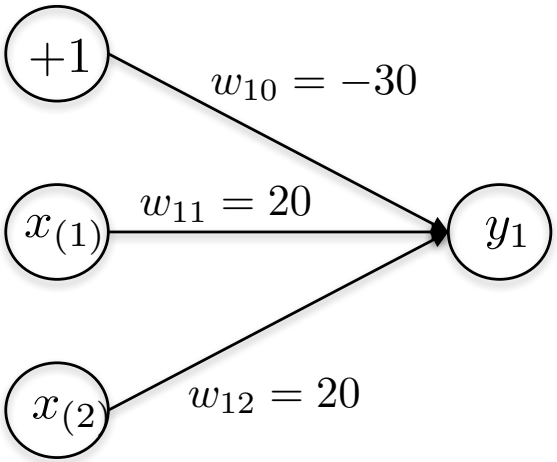
- Consider the AND logic function defined the following truth table:

$x_{(1)}$	$x_{(2)}$	AND
True	True	True
True	False	False
False	True	False
False	False	False

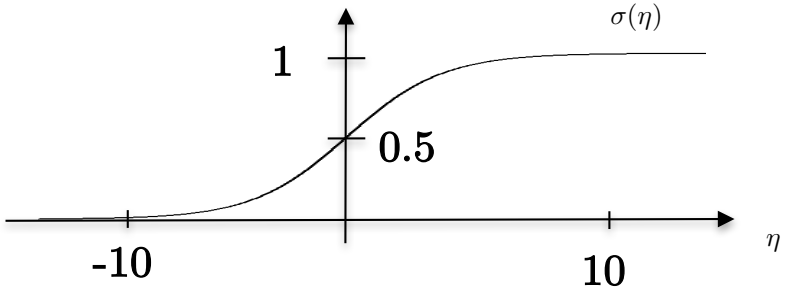
$$y = x_{(1)} \text{ AND } x_{(2)}$$

AND function with neural network

- We can define a neural network that encodes the AND logistic function:



$$y_1 = f_w(x) = \sigma(w_{10} + w_{11}x_{(1)} + w_{12}x_{(2)})$$
$$= \sigma(-30 + 20x_{(1)} + 20x_{(2)})$$



Convention:
True is 1.
False is 0.

$x_{(1)}$	$x_{(2)}$	AND
1	1	σ(10)≈1
1	0	σ(-10)≈0
0	1	σ(-10)≈0
0	0	σ(-30)≈0

OR function

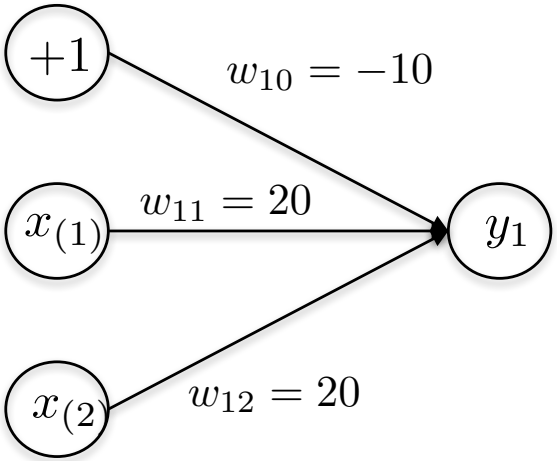
- Consider the OR logic function defined the following truth table:

$x_{(1)}$	$x_{(2)}$	OR
True	True	True
True	False	True
False	True	True
False	False	False

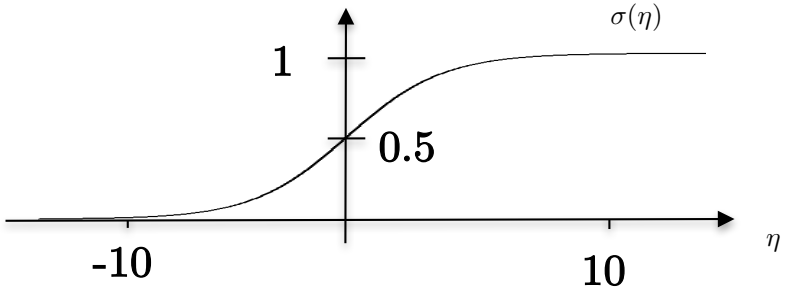
$$y = x_{(1)} \text{ OR } x_{(2)}$$

OR function with neural network

- We can define a neural network that encodes the OR logistic function:



$$y_1 = f_w(x) = \sigma(w_{10} + w_{11}x_{(1)} + w_{12}x_{(2)})$$
$$= \sigma(-10 + 20x_{(1)} + 20x_{(2)})$$



Convention:
True is 1.
False is 0.

$x_{(1)}$	$x_{(2)}$	OR
1	1	σ(30)≈1
1	0	σ(10)≈1
0	1	σ(10)≈1
0	0	σ(-10)≈0

Outline

- Motivation
- Brain inspiration
- Neurons and connections
- Neural networks with multiple layers
- Learned features
- Logical gates with neural networks
- **Logistic regression loss**
- Backpropagation
- Initialization
- Training neural networks
- Conclusion

Linear logistic regression

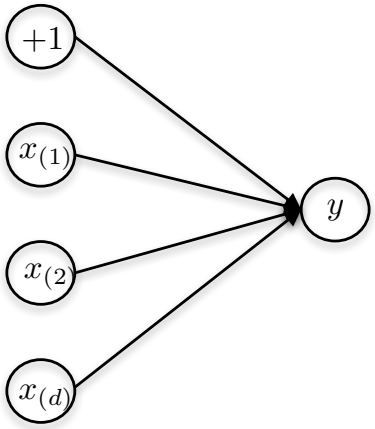
- **Reminder:** Logistic regression loss

$$L(w) = -\frac{1}{n}\Big[\sum_{i=1}^{n} \hat{y}_i \log p_w(x_i) + (1-\hat{y}_i)\log(1-p_w(x_i))\Big] + \frac{\lambda}{d}\sum_{j=1}^{d} w_j^2$$

Binary classification (K=2) : $p_w(x) \in \mathbb{R}$

Label probability : $\hat{y} \in \{0, 1\}$

- **Neural network of linear logistic regression:**



$$y = p_w(x) = \sigma(W \begin{bmatrix} 1 \\ x \end{bmatrix})$$

Logistic regression for neural networks

- **Multi-class logistic regression loss for NNs:**

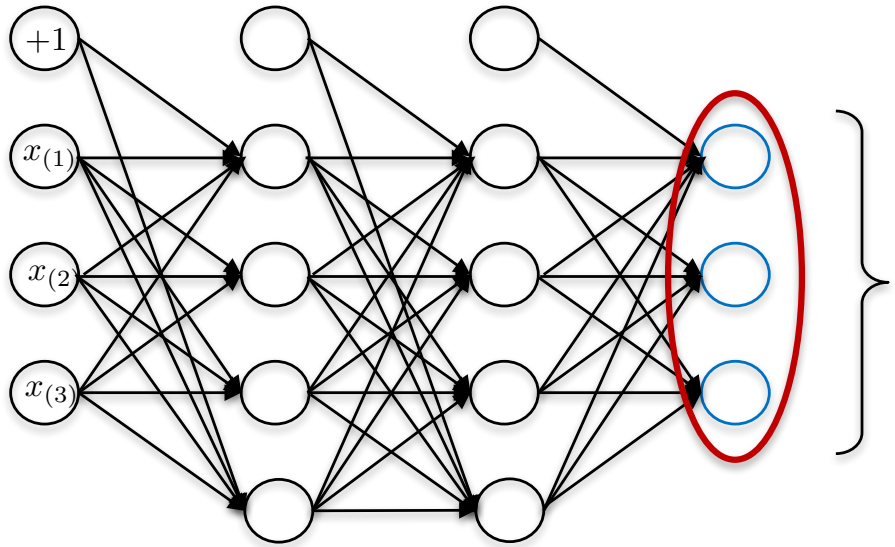
$$L(w) = \underbrace{-\frac{1}{n}\Big[\sum_{k=1}^{K}\sum_{i=1}^{n} \hat{y}_{i,k} \log p_w(x_i)_k + (1-\hat{y}_{i,k}) \log(1-p_w(x_i)_k)\Big]}_{\text{Measure of fitness between the data and the predictive NN model}} + \underbrace{\lambda \sum_{l=1}^{L-1}\sum_{i=1}^{n_{l+1}}\sum_{j=1}^{n_l+1}(W_{ij}^l)^2}_{\text{Sum over all weight parameters of the NN}}$$

Multi-class classification with K clusters:

$$p_w(x_i) = \begin{bmatrix} p_w(x_i)_1 \\ p_w(x_i)_2 \\ \vdots \\ p_w(x_i)_K \end{bmatrix} \in \mathbb{R}^K \qquad \hat{y}_i = \begin{bmatrix} \hat{y}_{i,1} \\ \hat{y}_{i,2} \\ \vdots \\ \hat{y}_{i,K} \end{bmatrix} \in \mathbb{R}^K$$

Probability vector for multi-class

- Output layer for the multi-class neural networks:

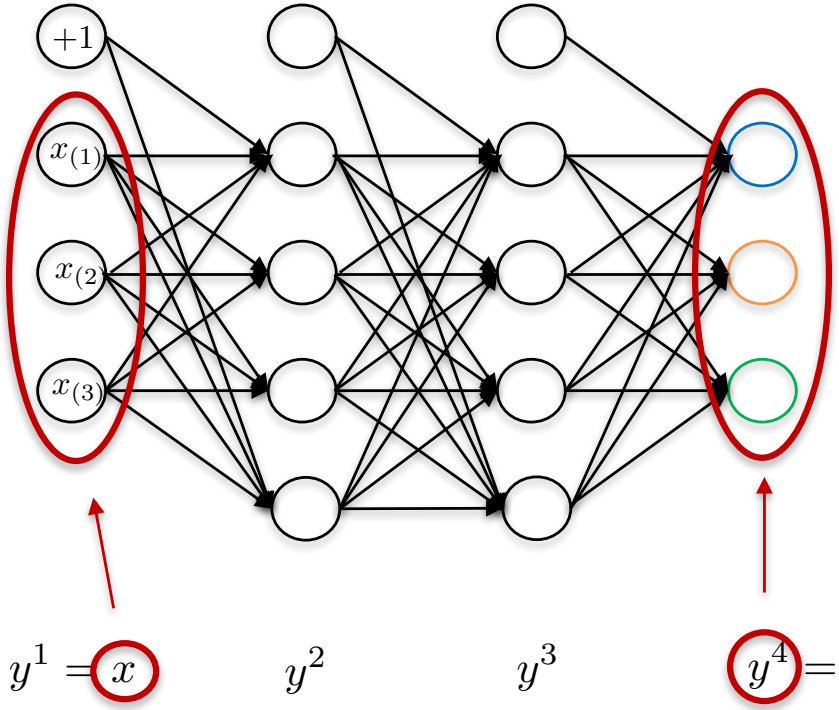


Output layer:
The number of output neurons is the number of classes, here K=3.

$y^1 = x$ y^2 y^3 $y^4 = p_w(x) \in \mathbb{R}^K$

Probability vector for multi-class

- For example, input x is an image and output y^4 is the probability for the image to belong to the class {cat,dog,car}.



Output neuron encoding the probability for class cat

Output neuron encoding the probability for class dog

Output neuron encoding the probability for class car

80% of chance that this image is a dog.

Probability vector for multi-class

- One-hot/Dirac representation of classes:

$$\hat{y}_i = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ if } x_i = \text{cat}$$

$$\hat{y}_i = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ if } x_i = \text{dog}$$

$$\hat{y}_i = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ if } x_i = \text{car}$$

This defines a 1-vs-all classification technique $\Rightarrow$
Each output neuron estimates the probability of the input to belong to one of the 3 classes.

In practice

- Classification task
  - Training set : $(x_1, \hat{y}_1), ..., (x_n, \hat{y}_n)$

$x \in \mathbb{R}^d$ $\quad$ $\hat{y} = \{1, 2, ..., K\}$

$\Downarrow$ One-hot representation of the class

Here $\hat{y}_2 = 2$

$$\hat{y}_{hot} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^K = \begin{cases} 1 & \text{if } \hat{y}_k = k \\ 0 & \text{otherwise} \end{cases}$$

Outline

- Motivation
- Brain inspiration
- Neurons and connections
- Neural networks with multiple layers
- Learned features
- Logical gates with neural networks
- Logistic regression loss
- **Backpropagation**
- Initialization
- Training neural networks
- Conclusion

Gradient descent

- Minimizing the loss function to learn the weight parameters $W^1, W^2, \ldots, W^L$ of the NNs:

$$\min_{w=(W^1,\ldots,W^L)} L(w)$$

$$L(w) = -\frac{1}{n}\Big[\sum_{k=1}^{K}\sum_{i=1}^{n} \hat{y}_{i,k} \log p_w(x_i)_k + (1-\hat{y}_{i,k}) \log(1 - p_w(x_i)_k))\Big] + \lambda \sum_{l=1}^{L-1}\sum_{i=1}^{n_{l+1}}\sum_{j=1}^{n_l+1} (W_{ij}^l)^2$$

- Gradient descent technique:

$$\text{Weight update :} \quad W^l \leftarrow W^l - \tau \underbrace{\frac{\partial L}{\partial W^l}}$$

Gradient of the loss w.r.t. weight parameter W^l

Notation:

$$\nabla_{W^l} = \frac{\partial L}{\partial W^l}$$

Backpropagation algorithm

Suppose we have no regularization: $\lambda = 0$

Suppose we have only 1 data: $n = 1$

- **Repeat until convergence:**
  - **Forward pass** (compute all activations):

For $l = 1, 2, ..., L$

$$y^{l+1} = \sigma\left(W^l \begin{bmatrix} 1 \\ y^l \end{bmatrix}\right)$$

$n_{l+1} \times 1$ $\quad$ $n_{l+1} \times (n_l + 1)$ $\quad$ $(n_l + 1) \times 1$

  - **Backward pass** (compute all gradients of weight parameters):

$$\delta^{l=L} = y^L - \hat{y}$$

$n_l \times 1$

No proof given!

For $l = L - 1, L - 2, ..., 1$

$$\nabla_{W^l} = \delta^{l+1} \begin{bmatrix} 1 \\ y^l \end{bmatrix}^T$$

$$W^l \leftarrow W^l - \tau \nabla_{W^l}$$

$n_{l+1} \times (n_l + 1)$

$$\delta^l = (\bar{W}^l)^T \delta^{l+1} \,.\, \sigma'\left(y^l\right)$$

$n_l \times 1$

Derivative of activation function σ

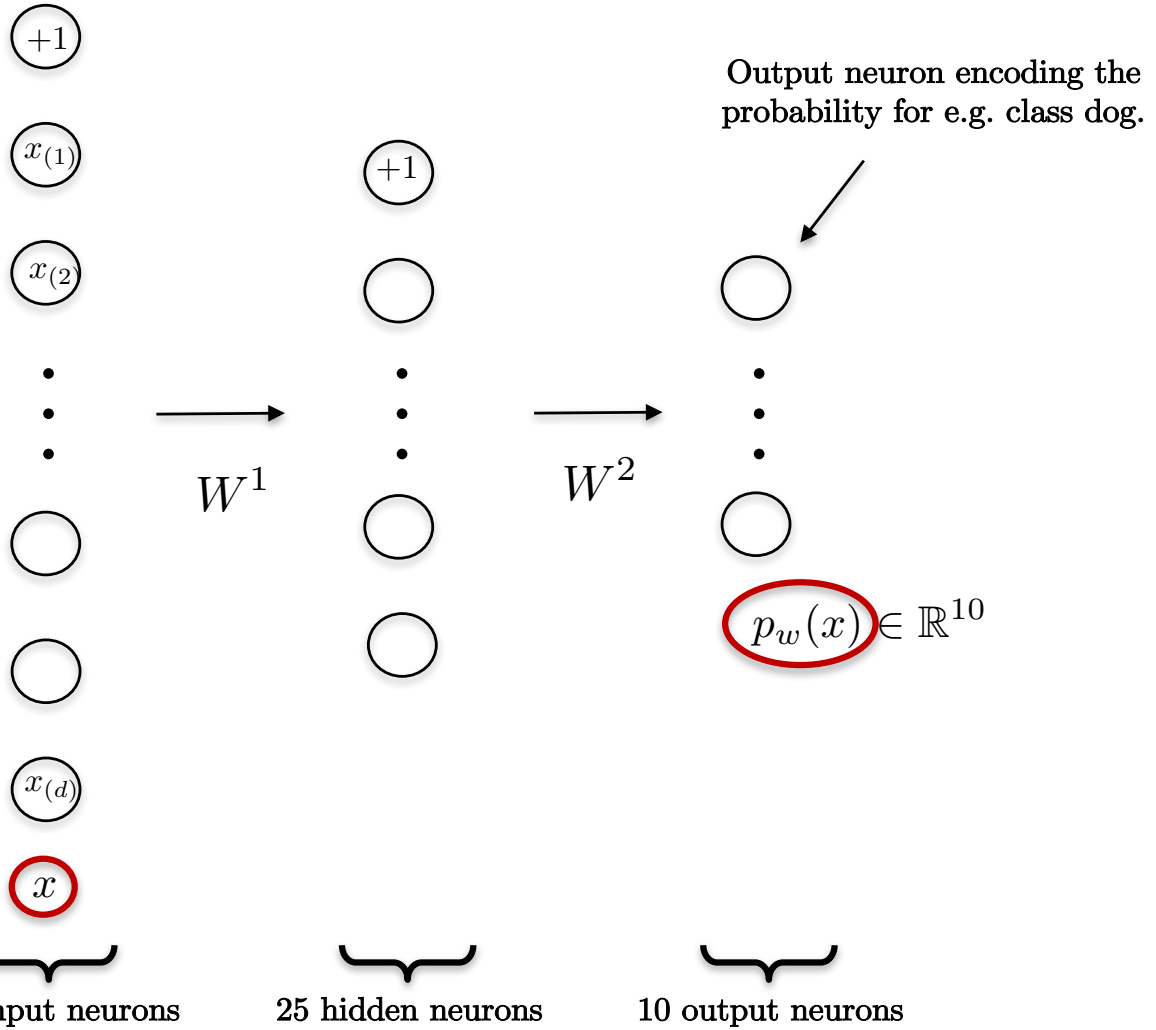
$$W^l = \left[\begin{array}{c|c} | & \\ W_0^l & \bar{W}^l \\ | & \end{array}\right]$$

$n_{l+1} \times 1$ $\quad$ $n_{l+1} \times n_l$

Bias vector

Example

- Three-layer neural network for classifying image data x into 10 classes: $n_1=d=400$, $n_2=25$, $n_3=K=10$.



Example

- **Forward equation:**

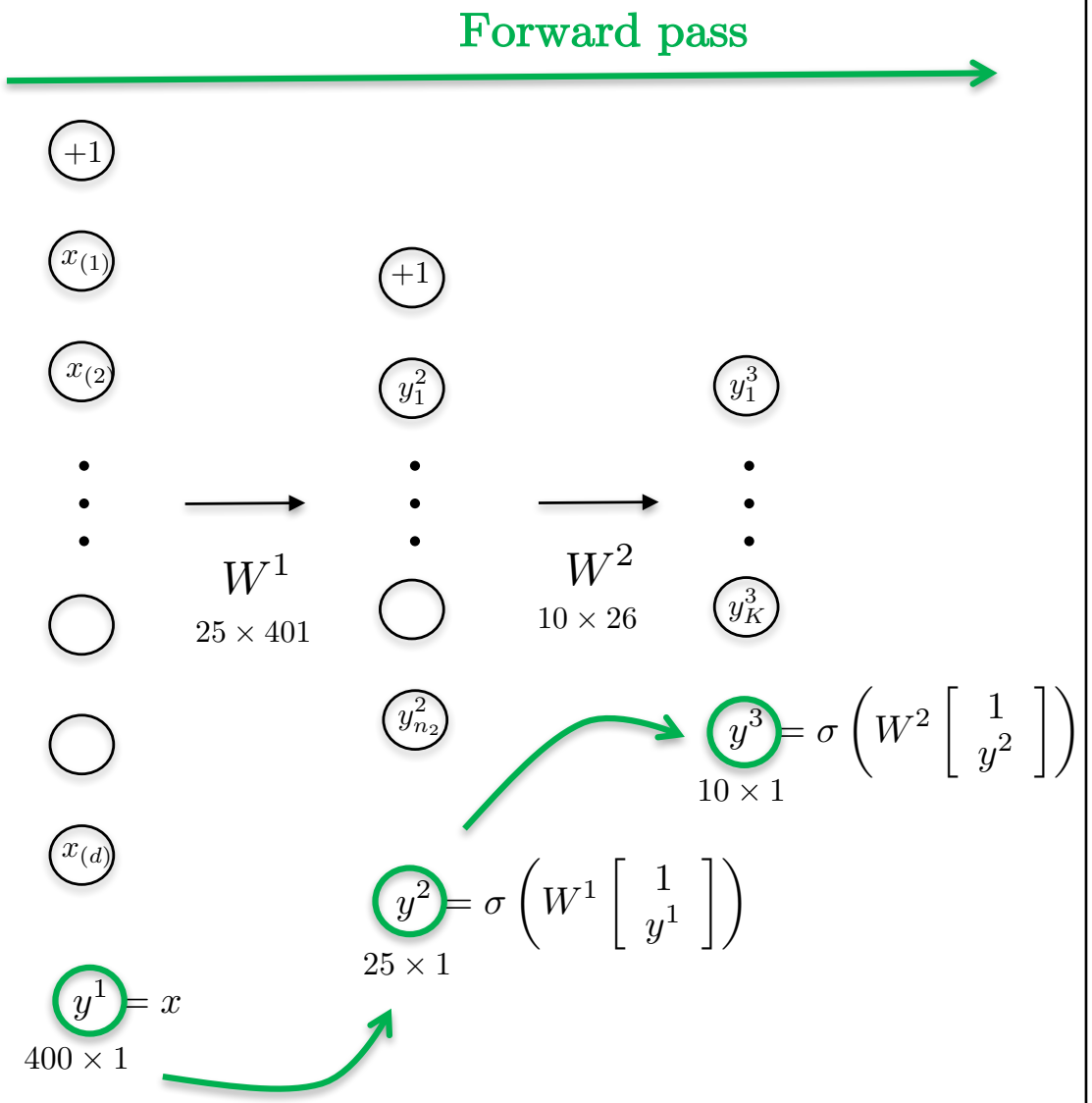
For $l = 1, 2, ..., L$

$$y^{l+1} = \sigma\left(W^l \begin{bmatrix} 1 \\ y^l \end{bmatrix}\right)$$

$n_{l+1} \times 1$ $\quad$ $(n_l + 1) \times 1$ $\quad$ $n_{l+1} \times (n_l + 1)$

- **Forward sequence:**

$$y^1 \rightarrow y^2 \rightarrow y^3$$



Example

- **Backward equation:**

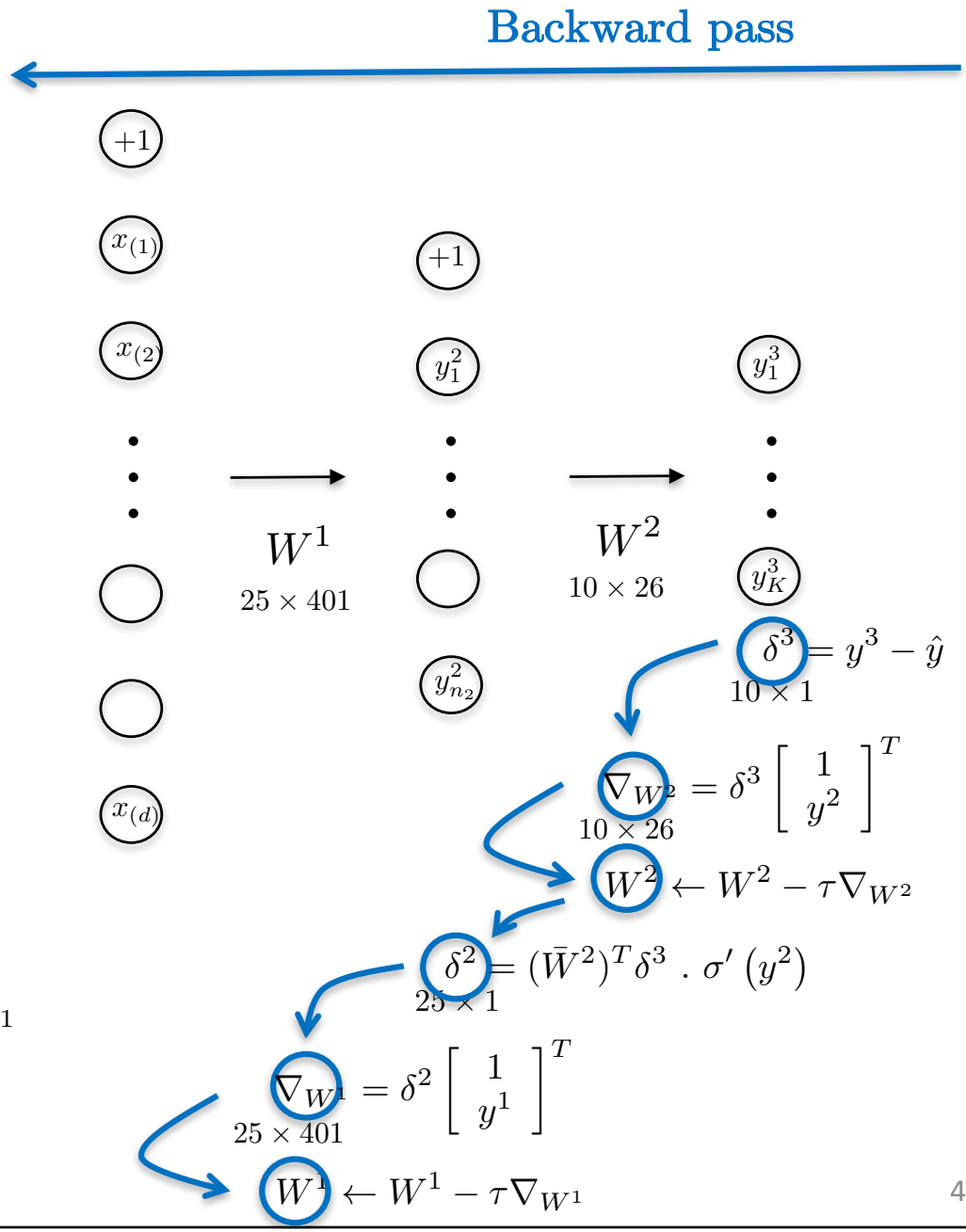
$$\delta^3 = y^3 - \hat{y}$$

For $l = 2, 1$

$$\nabla_{W^l} = \delta^{l+1} \begin{bmatrix} 1 \\ y^l \end{bmatrix}^T$$

$$W^l \leftarrow W^l - \tau \nabla_{W^l}$$

$$\delta^l = (\bar{W}^l)^T \delta^{l+1} \; . \; \sigma'(y^l)$$

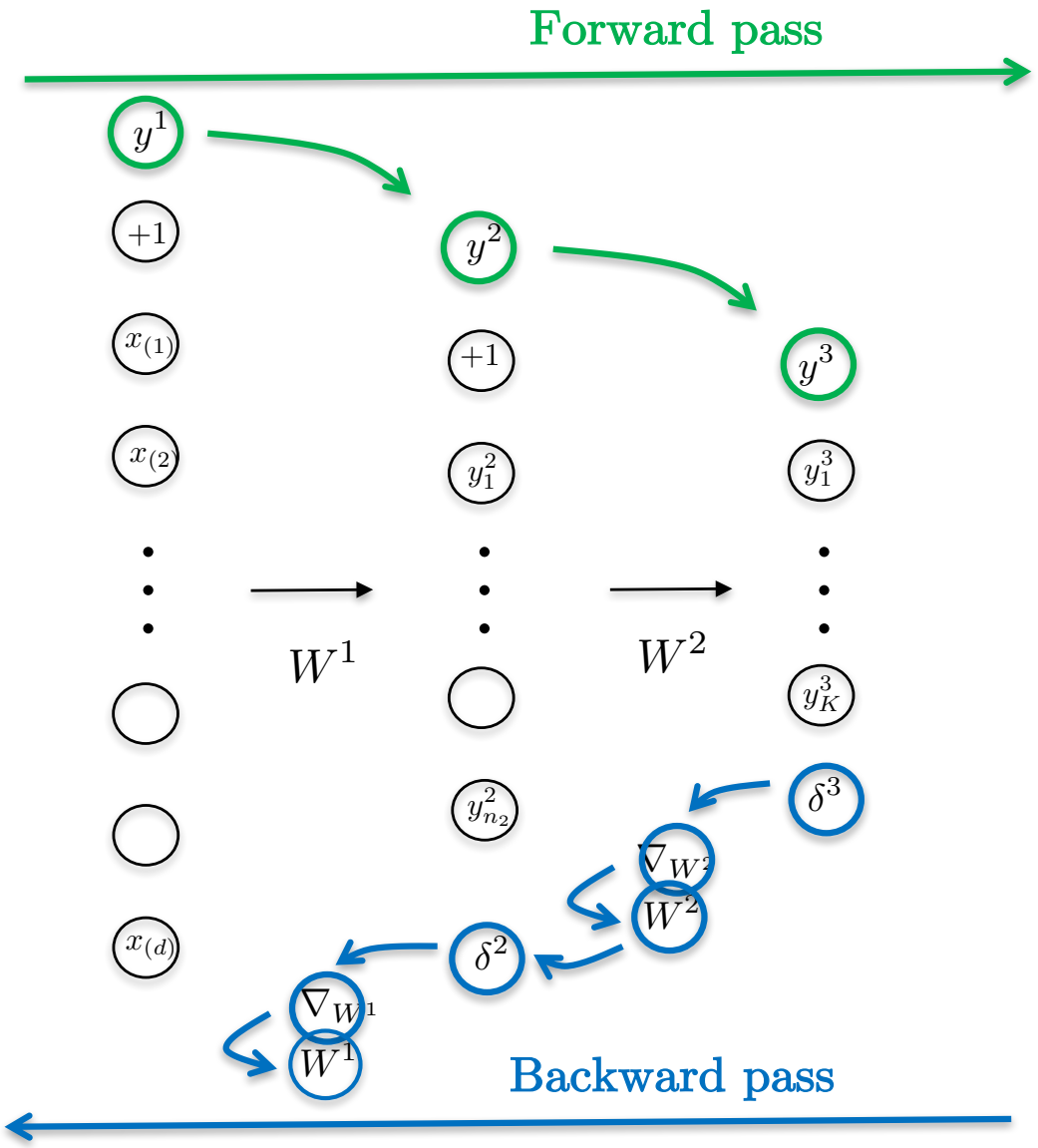


- **Backward sequence:**

$$\delta^3 \rightarrow \nabla_{W^2} \rightarrow W^2 \rightarrow \delta^2 \rightarrow \nabla_{W^1} \rightarrow W^1$$

Iterate

- Iterate forward pass and backward pass until convergence:



Regularization

- Simple modification of the backpropagation algorithm:
  - Forward pass (compute all activations):

For $l = 1, 2, ..., L$

$$y^{l+1} = \sigma\left(W^l \begin{bmatrix} 1 \\ y^l \end{bmatrix}\right)$$

  - Backward pass (compute all gradient of weight parameters):

$$\delta^{l=L} = y^L - \hat{y}$$

For $l = L-1, L-2, ..., 1$

$$\nabla_{W^l} = \delta^{l+1} \begin{bmatrix} 1 \\ y^l \end{bmatrix}^T + 2\lambda W^l$$

$$W^l \leftarrow W^l - \tau \nabla_{W^l}$$

$$\delta^l = (\bar{W}^l)^T \delta^{l+1} \,.\, \sigma'\left(y^l\right)$$

Regularization term

Intuition

- Backpropagation algorithm back-propagates the prediction error from the output/last layer to the input layer:

$$\delta^{l=L} = y^L - \hat{y}$$

prediction — label/ground truth

prediction error

- At each layer, the algorithm modifies the weight parameters to encourage or discourage (depending on the prediction error) the current values of the weights:

$$W^l \leftarrow W^l - \tau \nabla_{W^l}$$

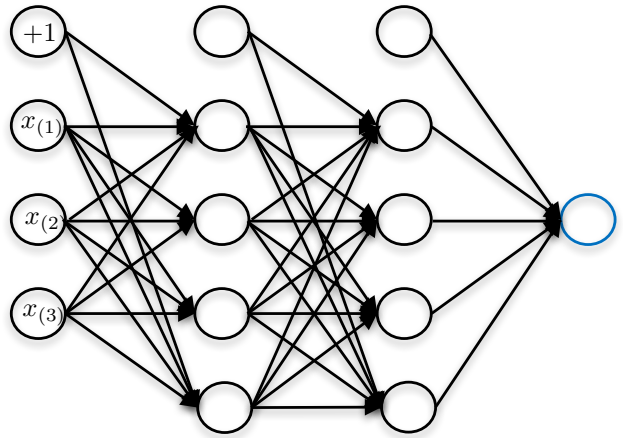
$$\nabla_{W^l} = \delta^{l+1} \begin{bmatrix} 1 \\ y^l \end{bmatrix}^T$$

$$\delta^l = (\bar{W}^l)^T \delta^{l+1} \cdot \sigma'(y^l)$$

- This approach is called pattern matching.

Regression with neural networks

- Neural networks can also be used for regression:



Output layer
There is only one output neuron.

$y^1 = x$ y^2 y^3 $y^4 = f_w(x) \in \mathbb{R}$

- The loss is the standard MSE loss:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n}(f_w(x_i) - y_i)^2 + \lambda \sum_{l=1}^{L-1}\sum_{i=1}^{n_{l+1}}\sum_{j=1}^{n_l+1}(W_{ij}^l)^2$$

- Backpropagation can also be applied to minimize the MSE loss.

Vectorized backpropagation

- The backpropagation algorithm can be vectorized:
  - Forward pass (compute all activations):

Number of data points

For $l = 1, 2, ..., L$

$n_{l+1} \times (n_l + 1)$ $\quad (n_l + 1) \times n$

$$y^{l+1} = \sigma\left(W^l \begin{bmatrix} 1 \\ y^l \end{bmatrix}\right)$$

$n_{l+1} \times n$

  - Backward pass (compute all gradients of weight parameters):

$$\delta^{l=L} = y^L - \hat{y}$$

$n_l \times n$

For $l = L-1, L-2, ..., 1$

$$\nabla_{W^l} = \frac{1}{n}\delta^{l+1}\begin{bmatrix} 1 \\ y^l \end{bmatrix}^T + 2\lambda W^l$$

$n_{l+1} \times (n_l + 1)$

$$W^l \leftarrow W^l - \tau \nabla_{W^l}$$

$$\delta^l = (\bar{W}^l)^T \delta^{l+1} \,.\, \sigma'\left(y^l\right)$$

$n_l \times n$

Derivative of activation function σ

$n_{l+1} \times n$ $\quad n_{l+1} \times n_l$

$$W^l = \left[\begin{array}{c|c} \vdots & \\ W_0^l & \bar{W}^l \\ \vdots & \end{array}\right]$$

Bias vector

Outline

- Motivation
- Brain inspiration
- Neurons and connections
- Neural networks with multiple layers
- Learned features
- Logical gates with neural networks
- Logistic regression loss
- Backpropagation
- **Initialization**
- Training neural networks
- Conclusion

Initialization

- It is not good to initialize all weight values to $W^l=0$.
- If $W^l=0$ then the forward pass would give the same value for all activations y^l, the same prediction error for all output neurons, and consequently the backpropagation algorithm would compute the same gradient for all weight parameters at each layer $l \Rightarrow$ All weight matrices W^l would be the same.

$$\delta^3 = y^3 - \hat{y}$$

For $l = 2, 1$

$$\nabla_{W^l} = \delta^{l+1} \begin{bmatrix} 1 \\ y^l \end{bmatrix}^T$$

$$W^l \leftarrow W^l - \tau \nabla_{W^l}$$

$$\delta^l = (\bar{W}^l)^T \delta^{l+1} \,.\, \sigma'(y^l)$$

- The issue is that all initial weights have the same value, so the solution is to break the symmetry.
- Random initialization: This also guarantees unit variance (no proof).

Uniform distribution

$$W^l = U\left[-\frac{2}{\sqrt{n_l}}, \frac{2}{\sqrt{n_l}}\right]$$

Outline

- Motivation
- Brain inspiration
- Neurons and connections
- Neural networks with multiple layers
- Learned features
- Logical gates with neural networks
- Logistic regression loss
- Backpropagation
- Initialization
- **Training neural networks**
- Conclusion

Training neural networks

- Back to the recipe for designing learning systems (Lecture 7):
  - Step 1: Pre-process data (zero-mean, unit variance)
  - Step 2: Choose first a small NN and increase the size if needed.
  - Step 3: Make sure you have enough learning capacity. Extract a sub-set of training data and over-fit them, i.e. $L_{\text{Train}} \approx 0$ (L_{Val} is high) by manually selecting the hyper-parameters.
  - Step 4: Add regularization (L2, dropout) and evaluate the generalization performance on the validation set. We should have $L_{\text{Val}} \searrow$ and $L_{\text{Train}} \nearrow$. The gap between L_{Val} and L_{Train} should be as small as possible.
  - Step 5: Use all training data and cross-validation (only k=1 fold) to estimate the parameters and the hyper-parameters (long running time). Ideally, $L_{\text{Val}} \approx L_{\text{Train}} \approx$ small value.

Modern backpropagation

- Backpropagation is the backbone of the learning algorithm.
- It is a simple algorithm (but it may not be easy to implement efficiently).
- Modern implementations are TensorFlow (Google) and PyTorch (Facebook) that perform automatically backpropagation and weight update - no need to implement backprog!
- TensorFlow and pyTorch are amazing libraries to build and train neural network architectures, and they are optimized for GPUs.

Outline

- Motivation
- Brain inspiration
- Neurons and connections
- Neural networks with multiple layers
- Learned features
- Logical gates with neural networks
- Logistic regression loss
- Backpropagation
- Initialization
- Training neural networks
- **Conclusion**

Conclusion

- Neural networks have become the state-of-the-art learning techniques since 2012.
- They provide impressive results in perceptual tasks (computer vision, speech processing, and natural language processing).
- This lecture has only covered fully connected neural networks, a.k.a. multi-layer perceptron (MLP).
- The 2 most successful neural network classes are:
  - Convolutional neural networks
  - Recurrent neural networks
- There exits a large and highly dynamic research area for designing neural network architectures.

Coding exercise

- [tutorial08.ipynb]

Tutorial 8: Neural networks

Objectives

- **Coding 3-layer neural network**
- **Implementing backpropagation**
- **Understanding bias vs. variance**

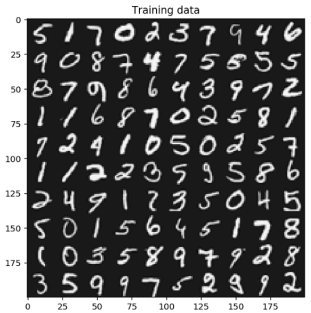
2. Visualize the datasets

Hint: You may use function `display_data`.

In [3]:
```
def display_data(X,width,height,nrows,ncols,title):

    big_picture = np.zeros((height*nrows,width*ncols))
    indices_to_display = random.sample(range(X.shape[1]), nrows*ncols)
    irow, icol = 0, 0
    for idx in indices_to_display:
        if icol == ncols:
            irow += 1
            icol  = 0
        iimg = X[:,idx].reshape(width,height).T
        big_picture[irow*height:irow*height+iimg.shape[0],icol*width:icol*width+iimg.shape[1]] = iimg
        icol += 1
    fig = plt.figure(figsize=(6,6))
    plt.title(title)
    img = scipy.misc.toimage( big_picture )
    plt.imshow(img,cmap = cm.Greys_r)


#YOUR CODE HERE
display_data(X_train,20,20,10,10,'Training data')
display_data(X_test,20,20,10,10,'Test data')
```



Questions?