DDA3020 Machine Learning: Lecture 09 Neural Networks

Baoyuan Wu
School of Data Science, CUHK-SZ

March 20/25, 2024

Outlines

1. Recall of previous classification models
2. Neural networks
   - Perceptron model
   - Multi-layer feedforward neural networks
3. Backpropagation
4. Deep neural networks

1. Recall of previous classification models
2. Neural networks
   - Perceptron model
   - Multi-layer feedforward neural networks
3. Backpropagation
4. Deep neural networks

Recall of previous classification models

Logistic regression:

- Hypothesis function:

$$h_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) \quad (1)$$

- Cost function:

$$Cost = -\log \sigma(y \cdot \mathbf{w}^\top \mathbf{x}) \quad (2)$$

- Learning algorithm:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} J(\mathbf{w}) \quad (3)$$

Support vector machine:

- Hypothesis function:

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b \quad (4)$$

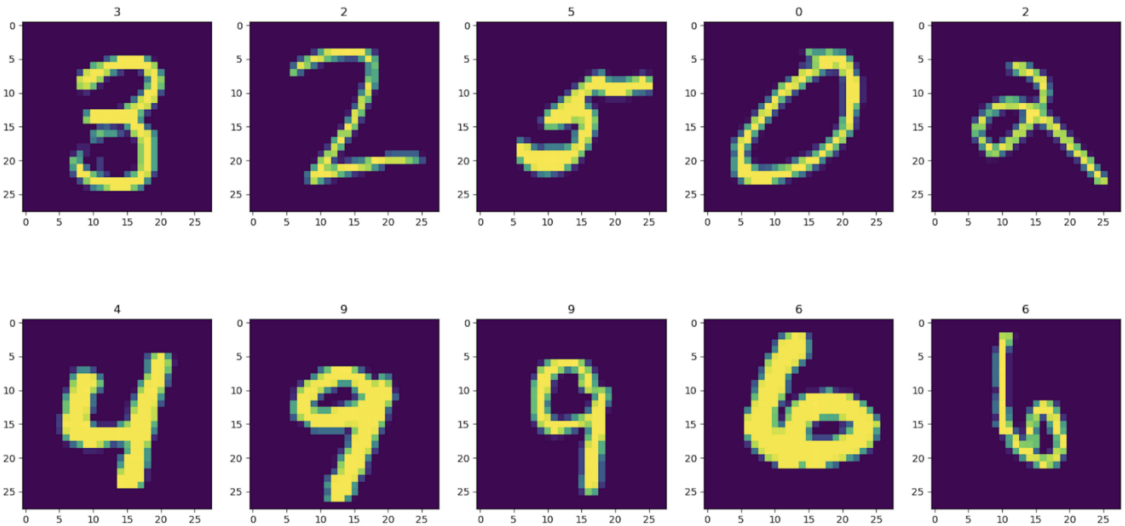
- Cost function:

$$Cost = \max(0, 1 - y \cdot (\mathbf{w}^\top \mathbf{x} + b)) \quad (5)$$

- Learning algorithm: Lagrange duality and KKT conditions

Recall of previous classification models

- In above two models, we mainly focus on designing the hypothesis function, cost function, learning algorithm, while defaultly assuming that the feature $\mathbf{x}$ is given as a vector
- However, for practical tasks, such as image classification or text classification, how to transform the image/text to vector $\mathbf{x}$?
- There are many studies on how to extract informative features from data, such as SIFT, HOG, optical flow features from image/video. All these features are **handcrafted features**, independent of the learning of classifier.
- We will introduce neural networks, which combines feature learning and classifier learning together.



1. Recall of previous classification models
2. Neural networks
   - Perceptron model
   - Multi-layer feedforward neural networks
3. Backpropagation
4. Deep neural networks

Neuron

- Our brain has $\sim 10^{11}$ neurons, each of which communicates (is connected) to $\sim 10^4$ other neurons

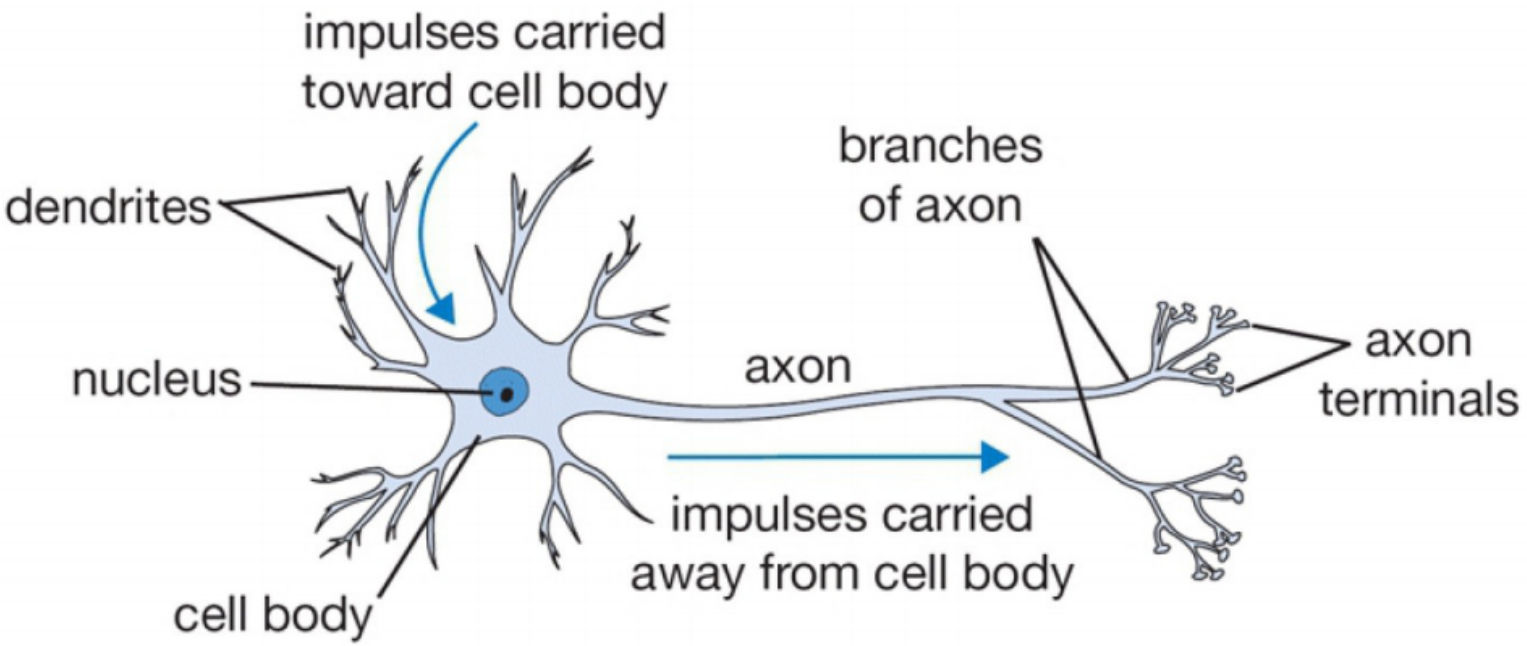
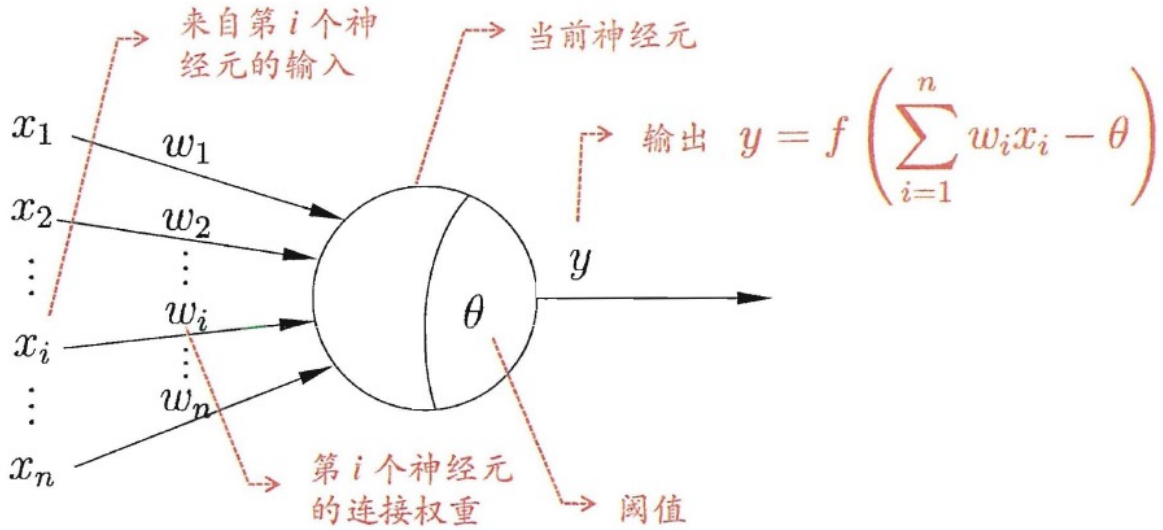


Figure: The basic computational unit of the brain: Neuron

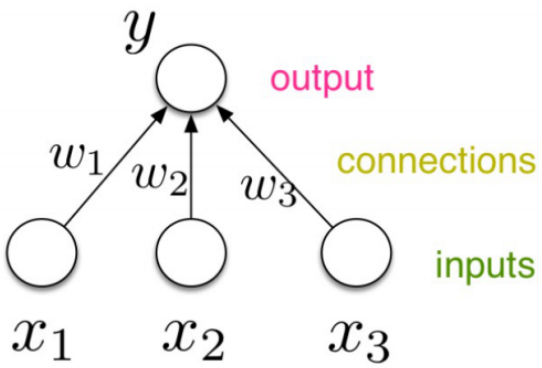
Neuron model

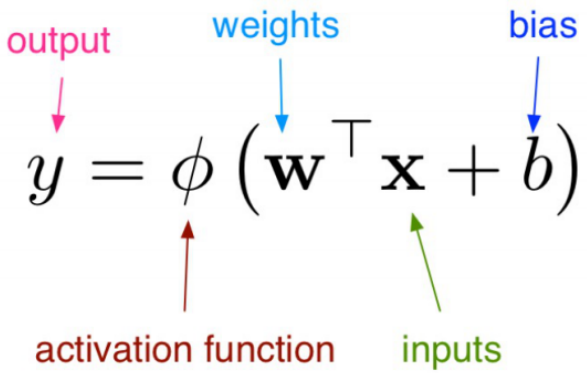
- In biological neuron model, the neurons are connected to others. There are positive and negative electric potentials in each neuron. If the number of positive electric potentials is over one threshold, then it is activated to send chemical substances to other connected neurons, leading to the change of the electric potentials of these neurons.
- Inspired by that phenomenon, in 1943, MaCulloch and Pitts proposed the M-P neuron model, as follows.



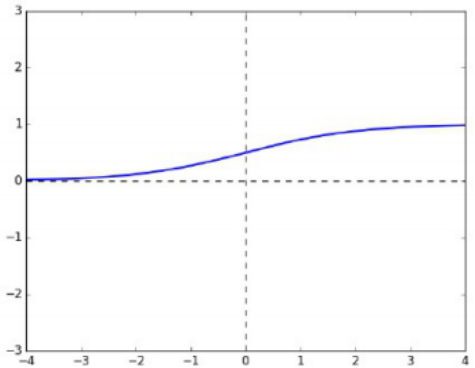
Neuron model

- For neural nets, we use a much simpler model neuron, or unit:



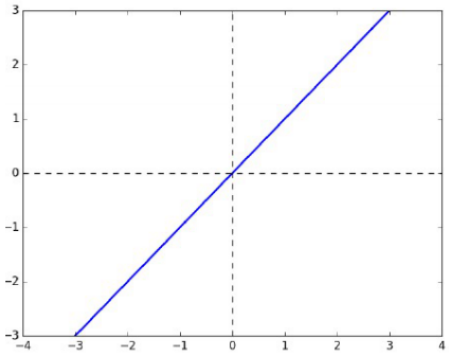


- Compare with logistic regression: $y = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)$

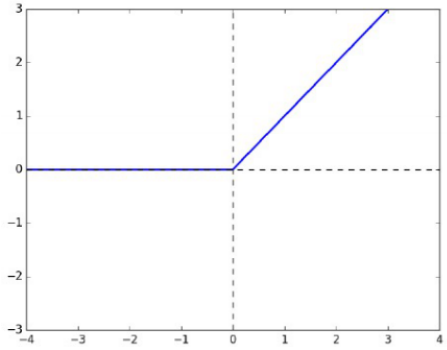


- By throwing together lots of these incredibly simplistic neuron-like processing units, we can do some powerful computations!

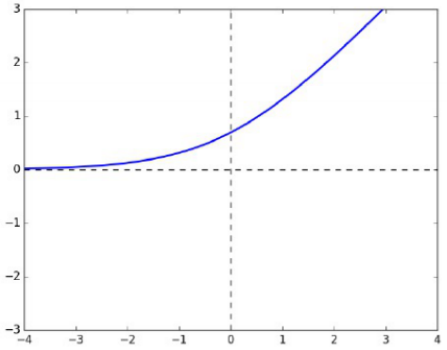
Activation functions



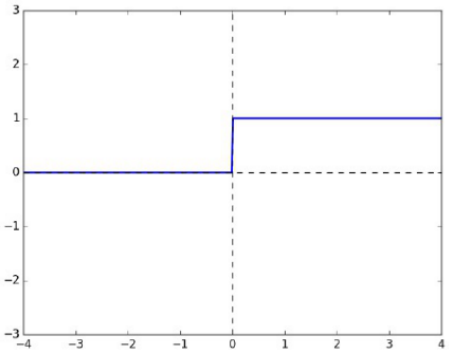
(a) Linear $y = z$



(b) Rectified Linear Unit(ReLU) $y = \max(0, z)$

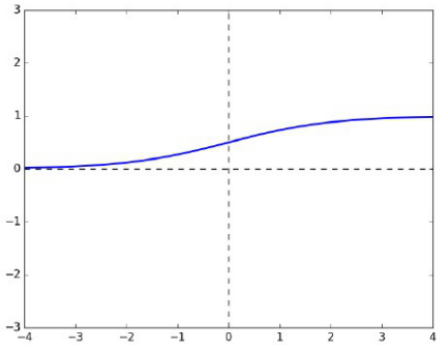


(c) Soft ReLU $y = \log 1 + e^z$

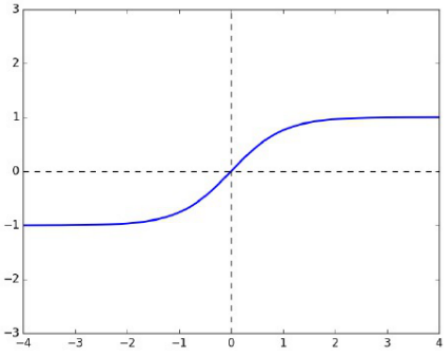


(d) Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$

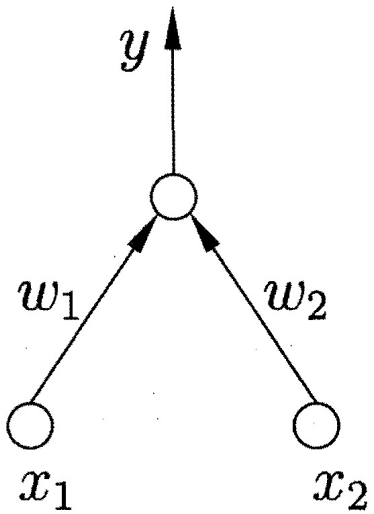


(e) Logistic $y = \frac{1}{1+e^{-z}}$



(f) Hyperbolic Tangent (tanh) $y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Perceptron model



- **Perceptron model**:
  - One input layer to receive input signals
  - One output layer including one M-P neuron, also called **threshold logic unit**
- **Formulation**: $y = f(\mathbf{w}^\top \mathbf{x} + b) = \text{Sgn}(\mathbf{w}^\top \mathbf{x} + b)$
- **Objective function**:

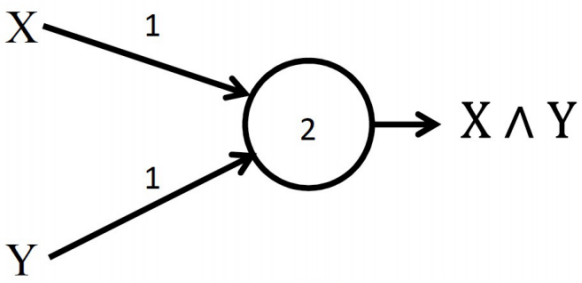
$$J(\mathbf{w}) = \frac{1}{2}(y - t)^2 = \frac{1}{2}(\text{Sgn}(\mathbf{w}^\top \mathbf{x} + b) - t)^2,$$

  where t denotes the ground-truth label.
- **Learning by gradient descent**:

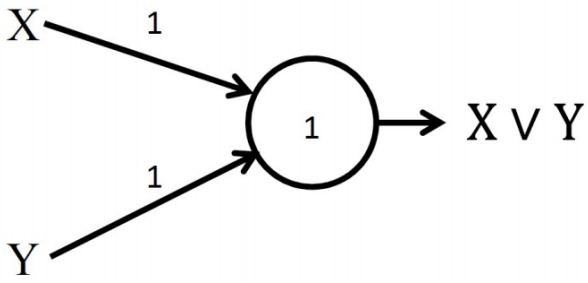
$$\mathbf{w} \leftarrow \mathbf{w} - \eta(y - t)\mathbf{x},$$

  where the gradient of $\text{Sgn}(\cdot)$ is approximated as 1.
- **Prediction** for a new data $\mathbf{x}_t$:
  - predict $y = +1$, if $\mathbf{w}^\top \mathbf{x} + b > 0$
  - predict $y = -1$, if $\mathbf{w}^\top \mathbf{x} + b \leq 0$

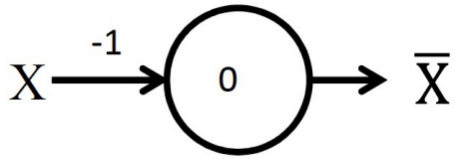
Basic logic problems



(a) AND



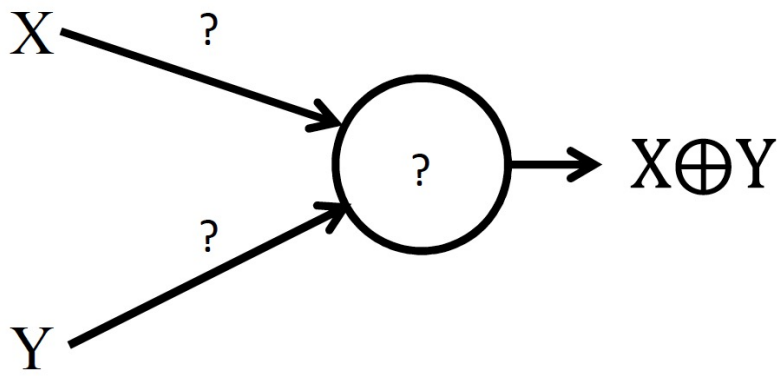
(a) OR



(C) NOT

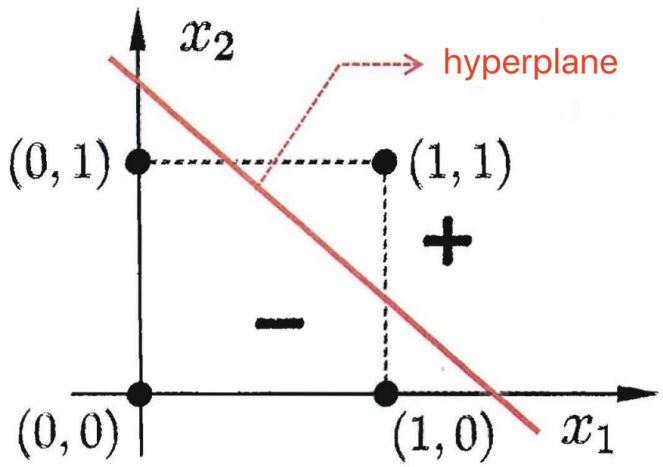
A perceptron can model any simple binary Boolean gate.

Basic logic problems

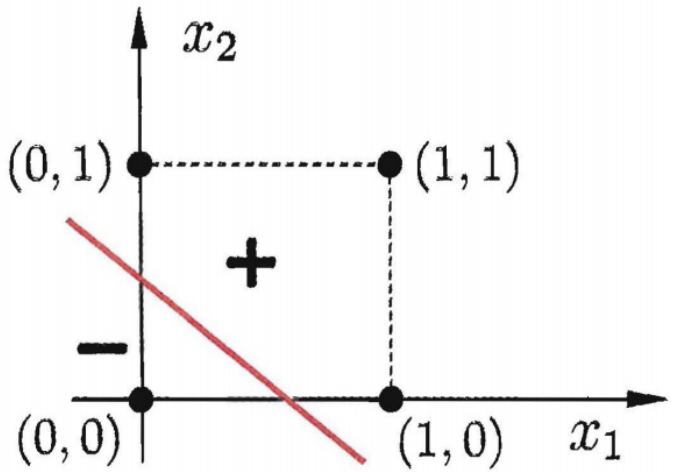


A perceptron cannot model XOR problem.

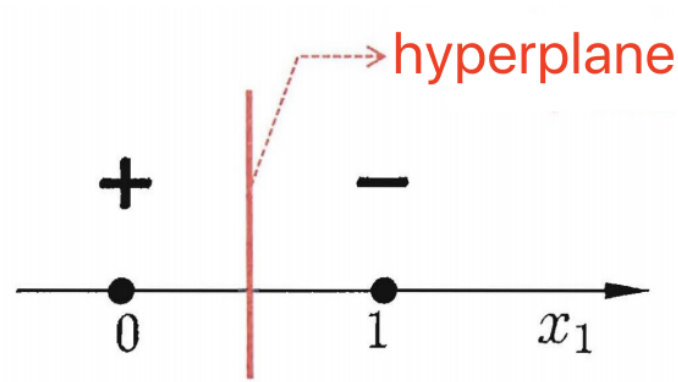
Basic logic problems



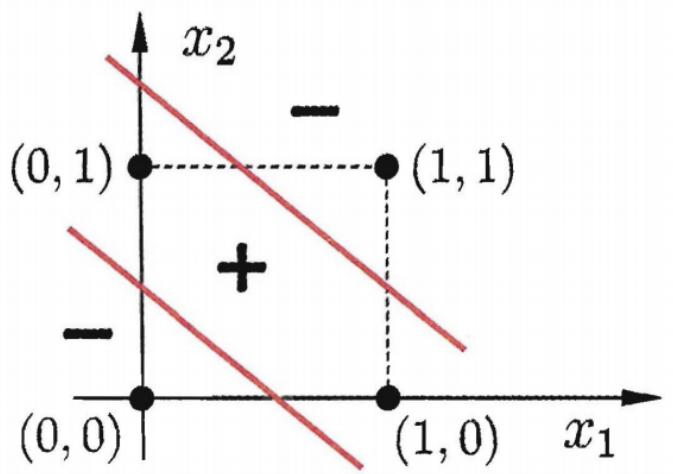
(a) 'And' Problem ($x_1 \wedge x_2$)



(b) 'Or' Problem ($x_1 \vee x_2$)



(c) 'Not' Problem ($\neg x_1$)

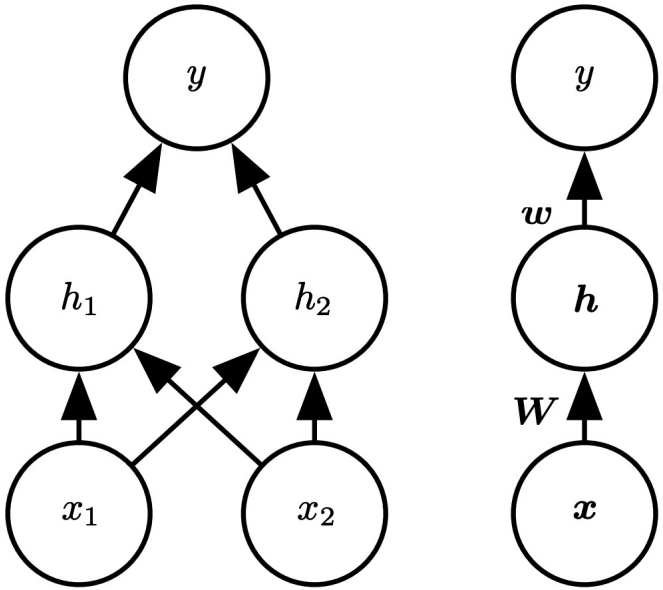


(d) 'XOR' Problem ($x_1 \bigoplus x_2$)

A perceptron/linear model cannot model XOR problem.

1. Recall of previous classification models
2. Neural networks
   - Perceptron model
   - Multi-layer feedforward neural networks
3. Backpropagation
4. Deep neural networks

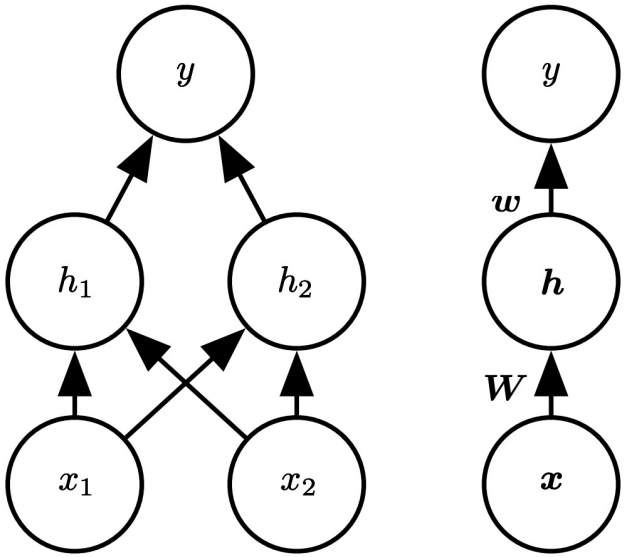
Multi-layer feedforward neural networks



- **Definition**:
  - Input layer, hidden layer(s), output layer
  - Only one direction from the input layer to the output layer
  - Fully connection between two layers
  - No connections among neurons in the same layer, no connections among neurons in non-adjacent layers (no skip connections)
- **Formulation**:

$$y = g_1(\mathbf{w}^\top \boldsymbol{h} + b)$$
$$\boldsymbol{h} = g_2(\mathbf{W}\mathbf{x} + c)$$

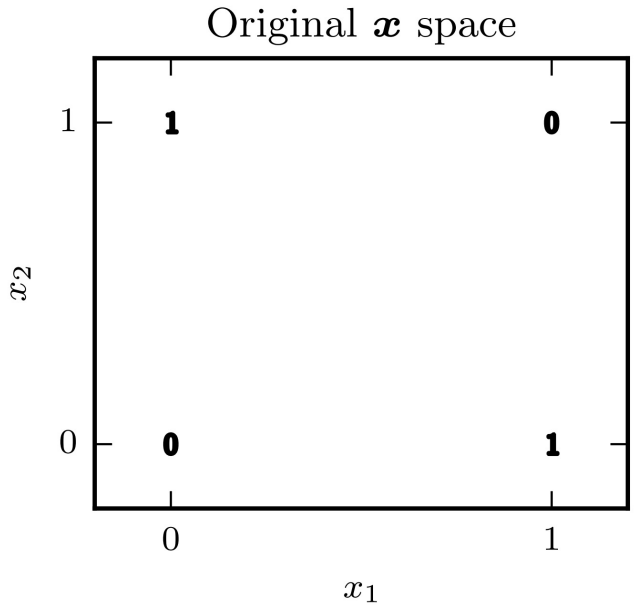
XOR problem



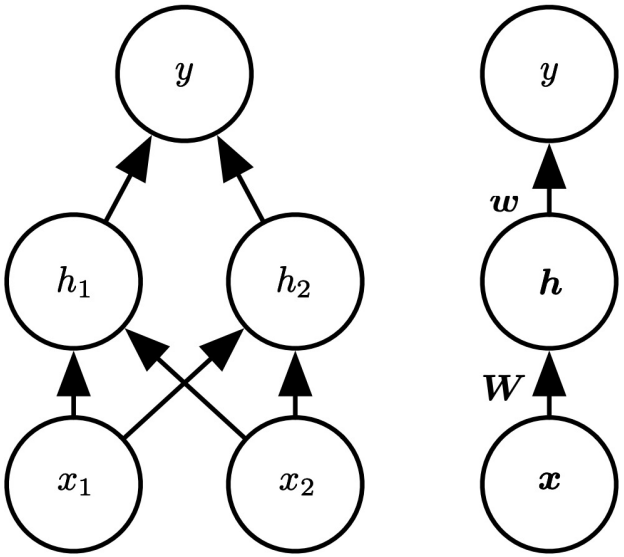
$$y = g_1(\mathbf{w}^\top \boldsymbol{h} + b) = \text{Sgn}\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}^\top \boldsymbol{h} - 0.5 \right)$$

$$\boldsymbol{h} = g_2(\mathbf{W}\mathbf{x} + c) = \text{Sgn}\left(\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \mathbf{x} - \begin{bmatrix} 0.5 \\ 1.5 \end{bmatrix} \right)$$

Try to transform the data in the original space to the hidden space of the neural network, and compute the output:



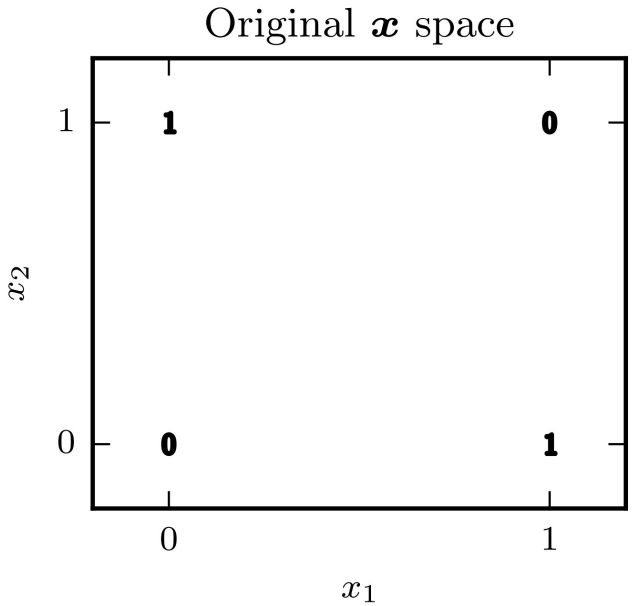
XOR problem



$$y = g_1(\mathbf{w}^\top \boldsymbol{h} + b) = \max\left(0, \begin{bmatrix} 1 \\ -2 \end{bmatrix}^\top \boldsymbol{h} - 0.5\right)$$

$$\boldsymbol{h} = g_2(\mathbf{W}\mathbf{x} + c) = \max\left(0, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}\right)$$

Try to transform the data in the original space to the hidden space of the neural network, and compute the output:



Multi-layer feedforward neural networks

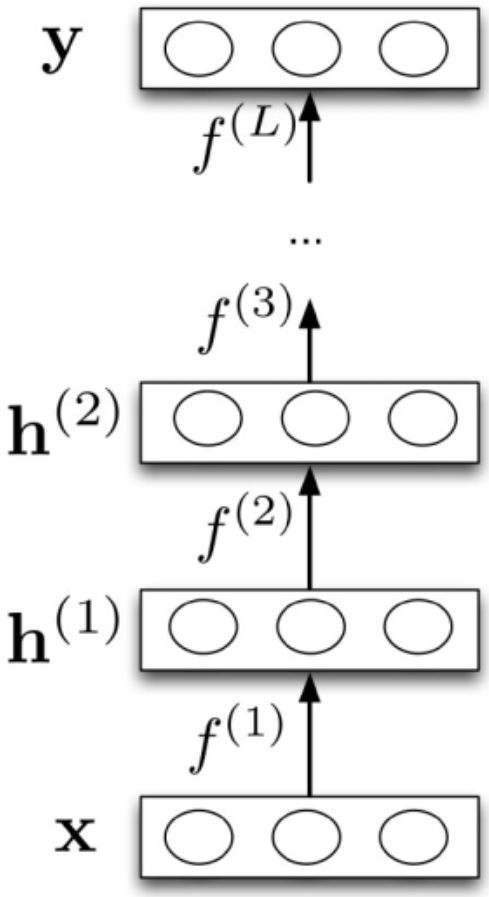
- Each layer computes a function, so the network computes a composition of functions:

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x})$$
$$\mathbf{h}^{(2)} = f^{(2)}(\mathbf{h}^{(1)})$$
$$\vdots$$
$$\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)})$$

- Or more simply:

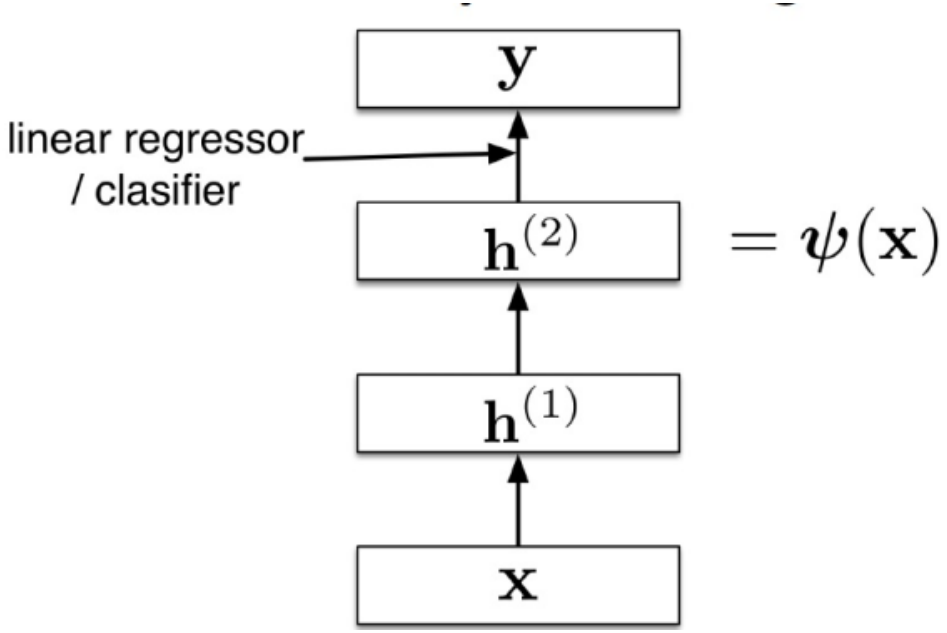
$$\mathbf{y} = f^{(L)} \circ \ldots \circ f^{(1)}(\mathbf{x})$$

- Neural nets provide modularity: we can implement each layer's computations as a black box.

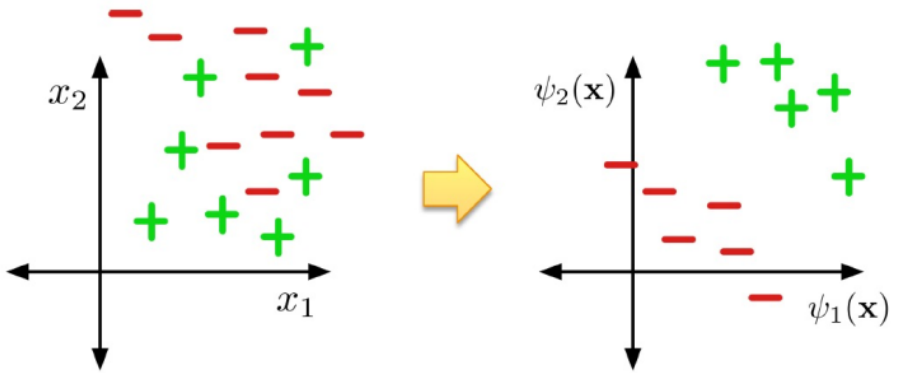


Multi-layer feedforward neural networks

- Neural nets can be viewed as a way of learning features:

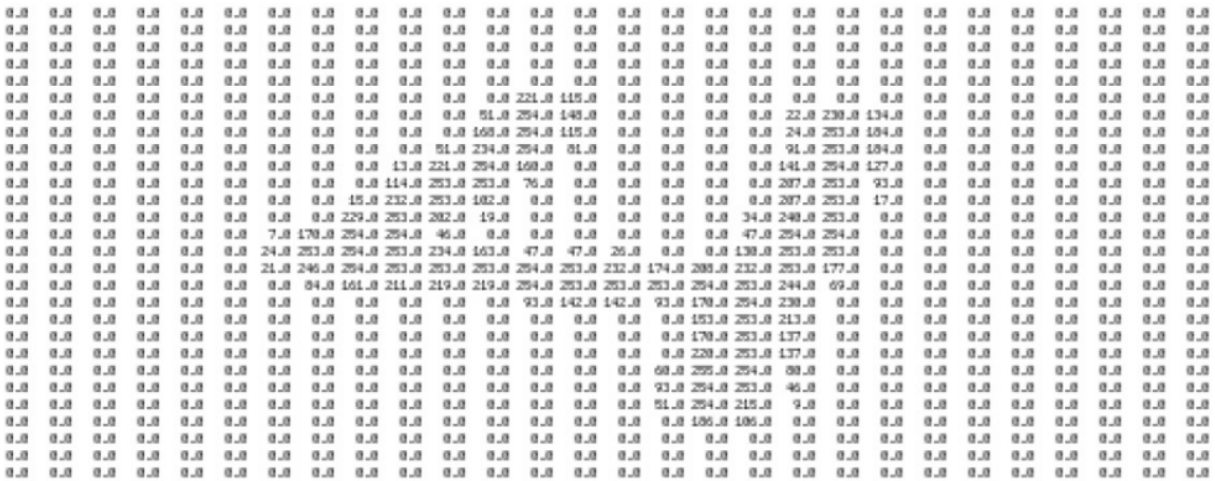


- The goal: non-linearly separable to linearly separable



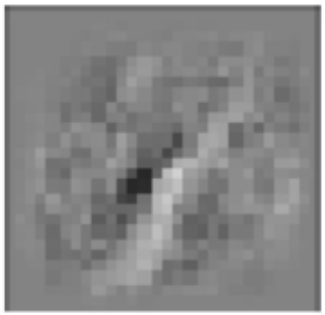
Multi-layer feedforward neural networks

Input representation of a digit: 784 dimensional vector.



Multi-layer feedforward neural networks

- Suppose we're trying to classify images of handwritten digits. Each image is represented as a vector of $28 \times 28 = 784$ pixel values.
- Each first-layer hidden unit computes $\sigma(\mathbf{w}_j^T \mathbf{x})$. It acts as a **feature detector**.
- We can visualize $\mathbf{w}_j$ by reshaping it into an image. Here's an example that responds to a digital stroke.



1 Recall of previous classification models

2 Neural networks
- Perceptron model
- Multi-layer feedforward neural networks

3 Backpropagation

4 Deep neural networks

Training Multi-layer feedforward neural networks

- Let's consider the gradient of the following loss function:

$$\mathcal{L} = \frac{1}{2}\big(\sigma(wx+b) - t\big)^2. \tag{6}$$

  For clarity, here w, x, b, t are scalars.
- Please compute the derivatives *w.r.t.* w and b, respectively.

$$\begin{aligned}
\mathcal{L} &= \frac{1}{2}(\sigma(wx+b)-t)^2 \\
\frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial}{\partial w}\left[\frac{1}{2}(\sigma(wx+b)-t)^2\right] \\
&= \frac{1}{2}\frac{\partial}{\partial w}(\sigma(wx+b)-t)^2 \\
&= (\sigma(wx+b)-t)\frac{\partial}{\partial w}(\sigma(wx+b)-t) \\
&= (\sigma(wx+b)-t)\sigma'(wx+b)\frac{\partial}{\partial w}(wx+b) \\
&= (\sigma(wx+b)-t)\sigma'(wx+b)x
\end{aligned}$$

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial}{\partial b}\left[\frac{1}{2}(\sigma(wx+b)-t)^2\right] \\
&= \frac{1}{2}\frac{\partial}{\partial b}(\sigma(wx+b)-t)^2 \\
&= (\sigma(wx+b)-t)\frac{\partial}{\partial b}(\sigma(wx+b)-t) \\
&= (\sigma(wx+b)-t)\sigma'(wx+b)\frac{\partial}{\partial b}(wx+b) \\
&= (\sigma(wx+b)-t)\sigma'(wx+b)
\end{aligned}$$

Backpropagation

- **Chain rule**: if $f(h)$ and $h(w)$ are univariate functions, then

$$\frac{\mathrm{d}f(h(w))}{\mathrm{d}w} = \frac{\mathrm{d}f}{\mathrm{d}h} \cdot \frac{\mathrm{d}h}{\mathrm{d}w}. \tag{7}$$

Backpropagation

- We write the above loss function again for comparison,

$$\mathcal{L} = \frac{1}{2}\big(\sigma(wx + b) - t\big)^2. \tag{8}$$

- We can rewrite it as follows:

$$\begin{aligned} z &= wx + b \\ y &= \sigma(z) \\ \mathcal{L} &= \frac{1}{2}(y - t)^2 \end{aligned}$$

Let's compute the loss derivatives.

Backpropagation

A more structured way to do it

Computing the loss:

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Computing the derivatives:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} = y - t$$

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\frac{\mathrm{d}y}{\mathrm{d}z} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\sigma'(z)$$

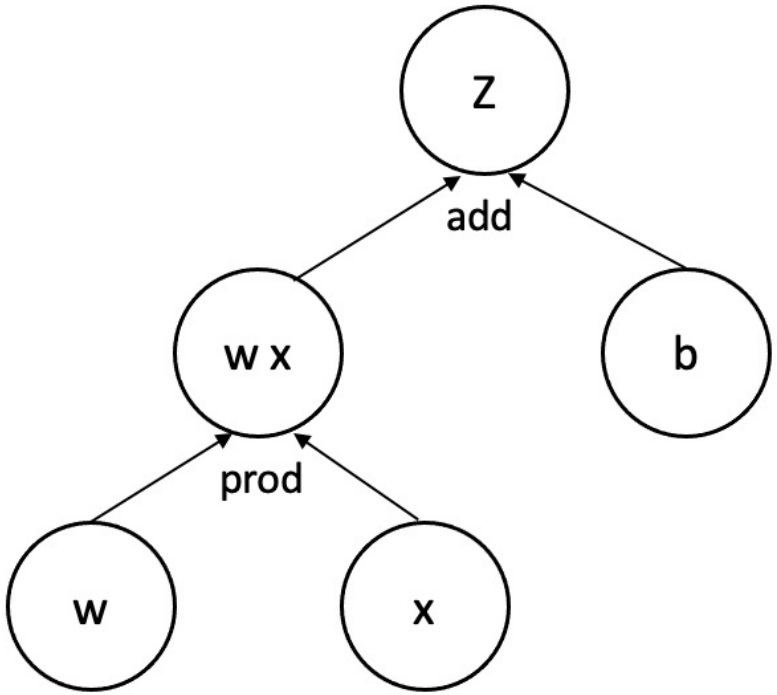
$$\frac{\partial\mathcal{L}}{\partial w} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}\frac{\partial z}{\partial w} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}x$$

$$\frac{\partial\mathcal{L}}{\partial b} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}\frac{\partial z}{\partial b} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}$$

Remember, the goal isn't to obtain closed-form solutions, but to be able to write a program that efficiently computes the derivatives.

Computational graph

- We can diagram out the computations using a computational graph (CG).
- The nodes represent all the inputs and computed quantities, and the edges represent which nodes are computed directly as a function of which other nodes.
- For example, $z = wx + b$ can be represented by the following CG

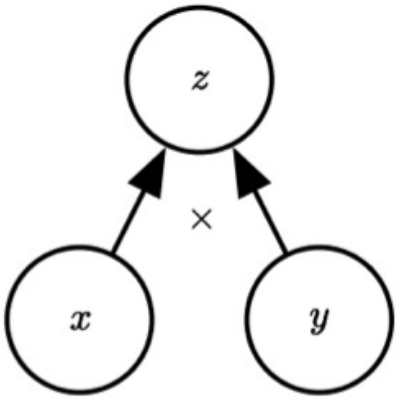


Computational graph of forward pass

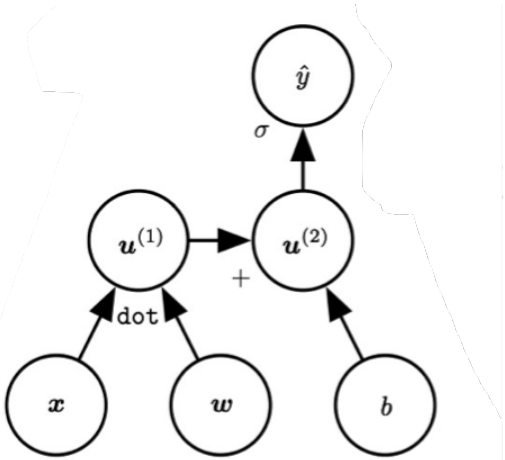
Please plot the corresponding CG.

$$\mathcal{L} = \frac{1}{2}\big(\sigma(wx + b) - t\big)^2. \tag{9}$$

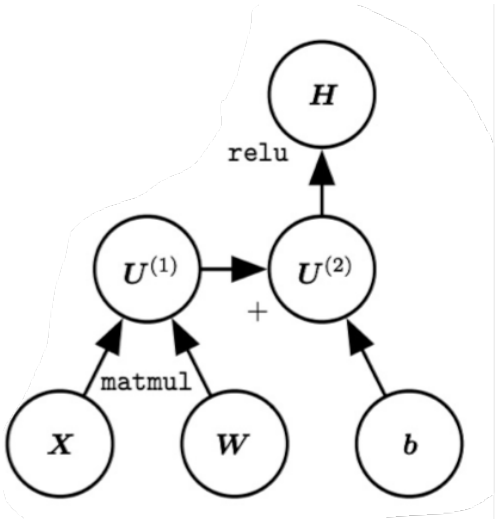
Computational graph of forward pass



(a) Multiplication



(b) Logistic Regression



(c) ReLU layer

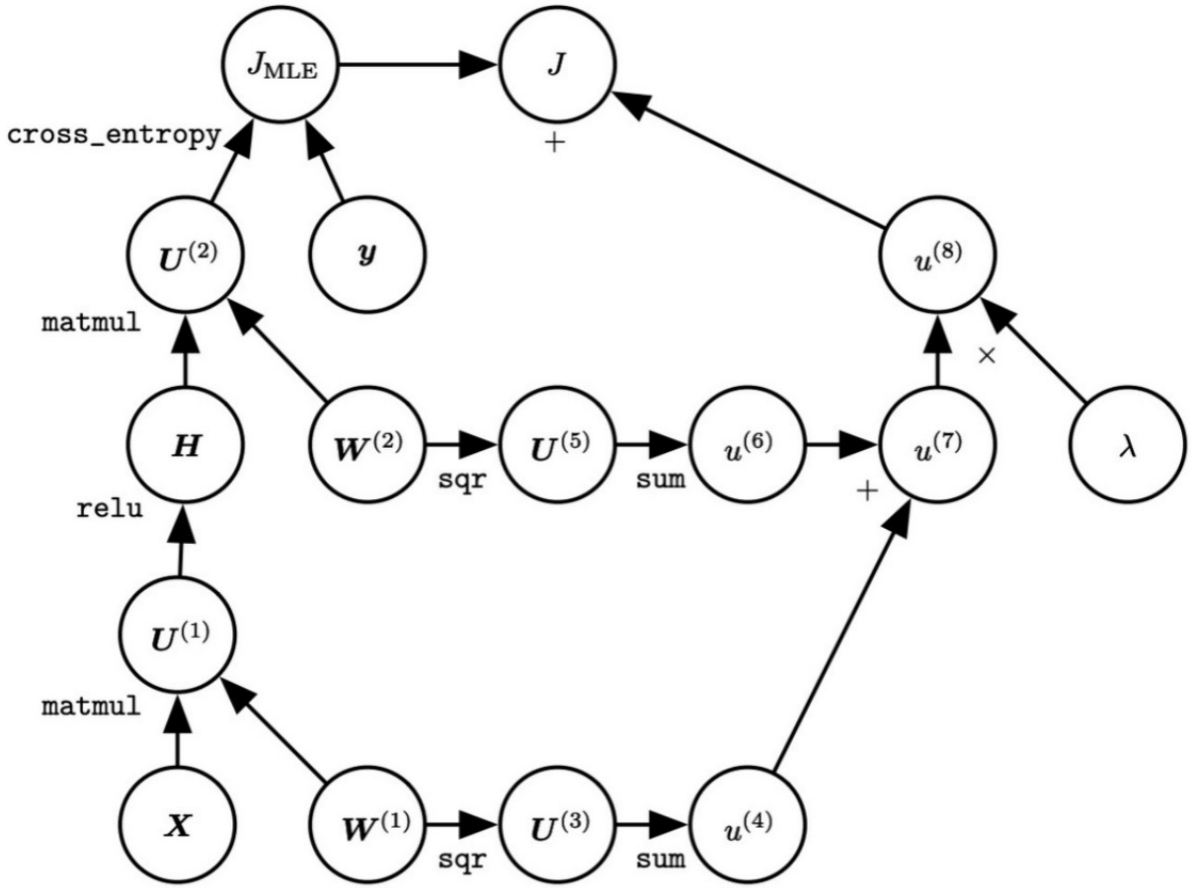
Computational graph of forward pass

- Consider the following problem

$$z = wx + b \tag{10}$$
$$y = \sigma(z) \tag{11}$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2 \tag{12}$$
$$\mathcal{R} = \frac{1}{2}w^2 \tag{13}$$
$$\mathcal{L}_{reg} = \mathcal{L} + \lambda\mathcal{R} \tag{14}$$

- Please plot its CG. Note that it involves two terms in the final loss.

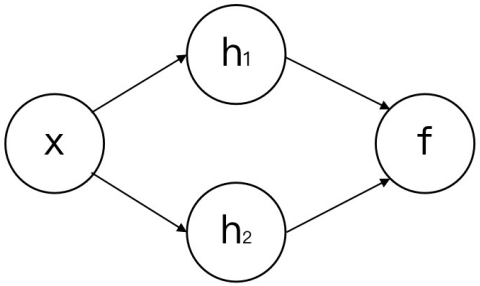
Computational graph of forward pass



Exercise 1: Please write the function according to this computational graph.

Computational graph of multivariate function

- Suppose we have a function $f(h_1, h_2)$ and functions $h_1(x)$ and $h_2(x)$. (All the variables here are scalar-valued.) Then



$$\frac{\mathrm{d}}{\mathrm{d}x} f(h_1(x), h_2(x)) = \frac{\partial f}{\partial h_1}\frac{\mathrm{d}h_1}{\mathrm{d}x} + \frac{\partial f}{\partial h_2}\frac{\mathrm{d}h_2}{\mathrm{d}x}$$

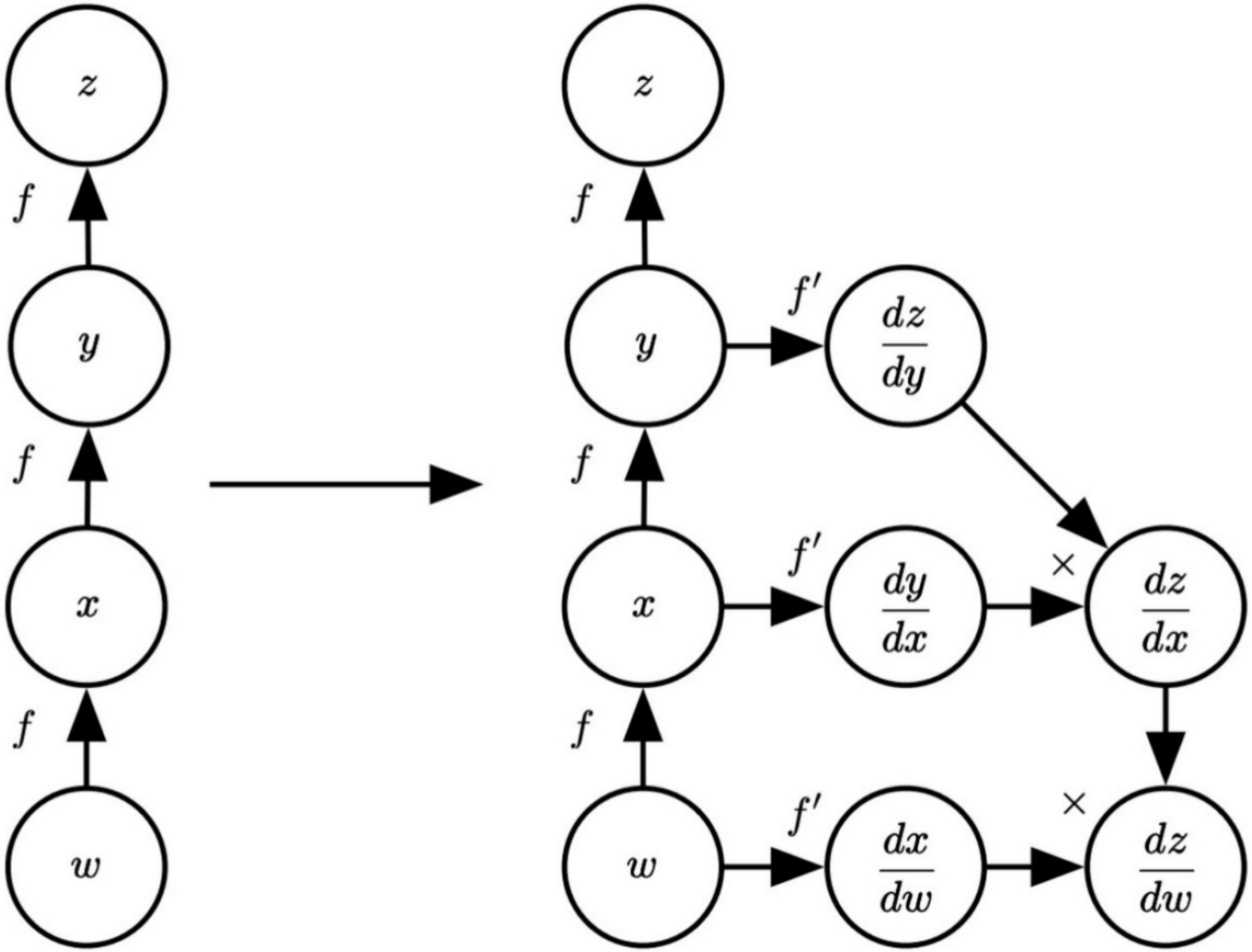
- Example:

$$\begin{aligned} f(h_1, h_2) &= h_2 + e^{h_1 h_2} \\ h_1(x) &= \cos x \\ h_2(x) &= x^2 \end{aligned}$$

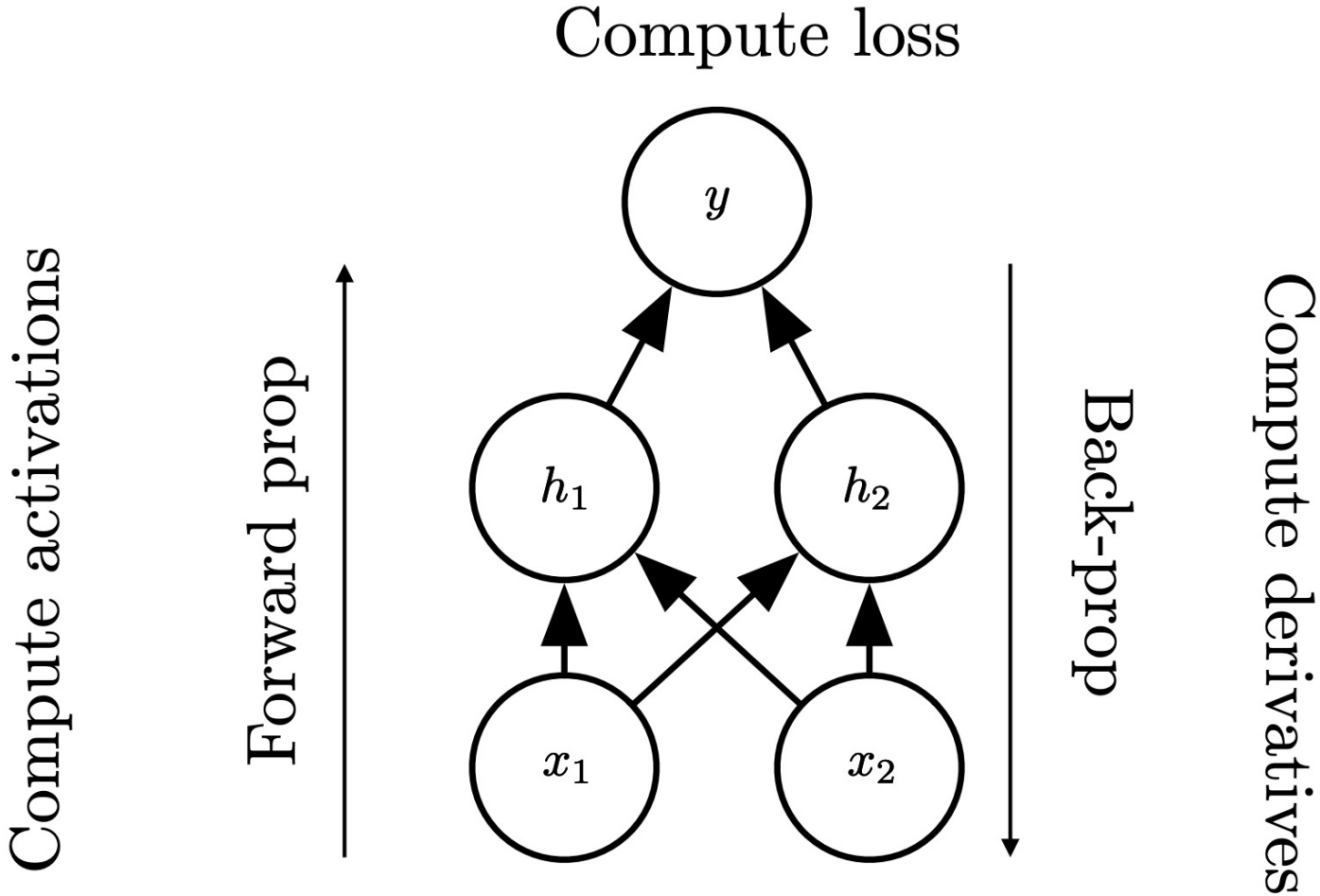
- Plug in to Chain Rule:

$$\begin{aligned} \frac{\mathrm{d}f}{\mathrm{d}x} &= \frac{\partial f}{\partial h_1}\frac{\mathrm{d}h_1}{\mathrm{d}x} + \frac{\partial f}{\partial h_2}\frac{\mathrm{d}h_2}{\mathrm{d}x} \\ &= \left(h_2 e^{h_1 h_2}\right) \cdot (-\sin x) + \left(1 + h_1 e^{h_1 h_2}\right) \cdot 2x \end{aligned}$$

Computational graph of backward pass



Forward and backward propagation



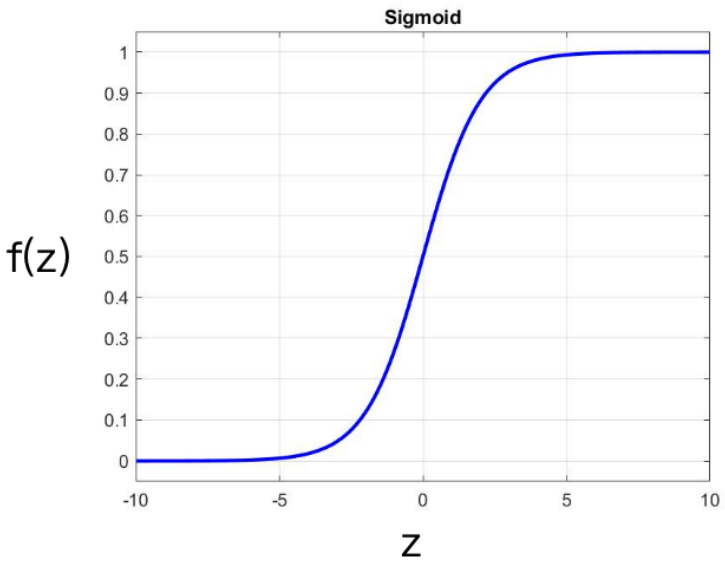
Backpropagation

- A single logistic neuron

$$z = f_w(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$$

$$y = f(z) = \frac{1}{1 + e^{-z}}$$

- These give a real-valued output that is a smooth and bounded function of their total input.



Backpropagation

- The derivatives of the logit, z, with respect to the inputs and the weights

$$z = \mathbf{x}^T \mathbf{w} = b + \sum_i w_i x_i$$

$$\frac{\partial z}{\partial w_i} = x_i$$

$$\frac{\partial z}{\partial x_i} = w_i$$

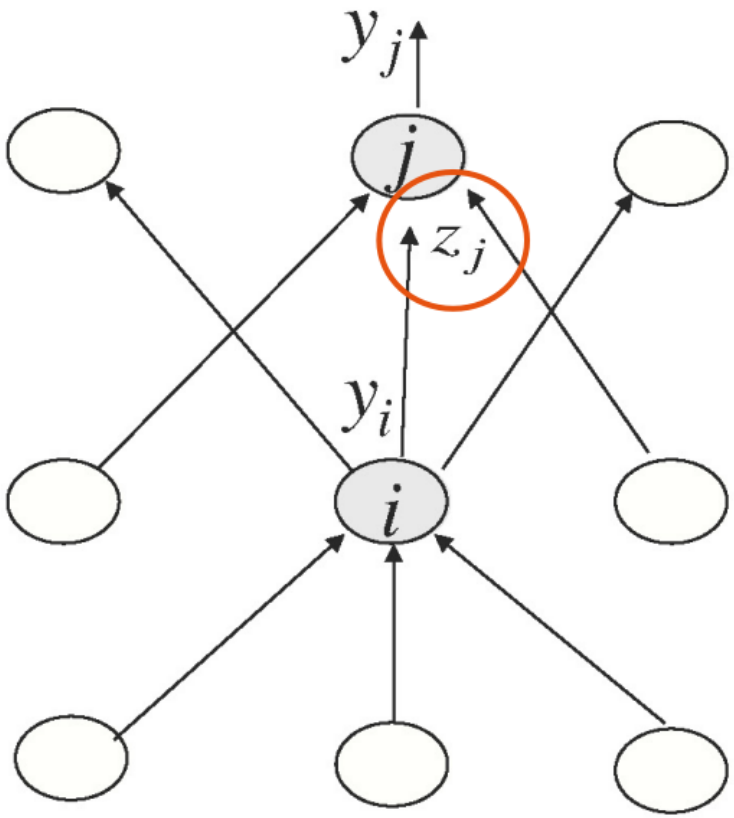
- The derivative of the output y with respect to the logit in terms of the output:

$$y = f(z) = \frac{1}{1 + e^{-z}}$$

$$\frac{\partial y}{\partial z} = y(1 - y)$$

Backpropagation

Backpropagating $\frac{\partial L}{\partial y}$ through multiple layers



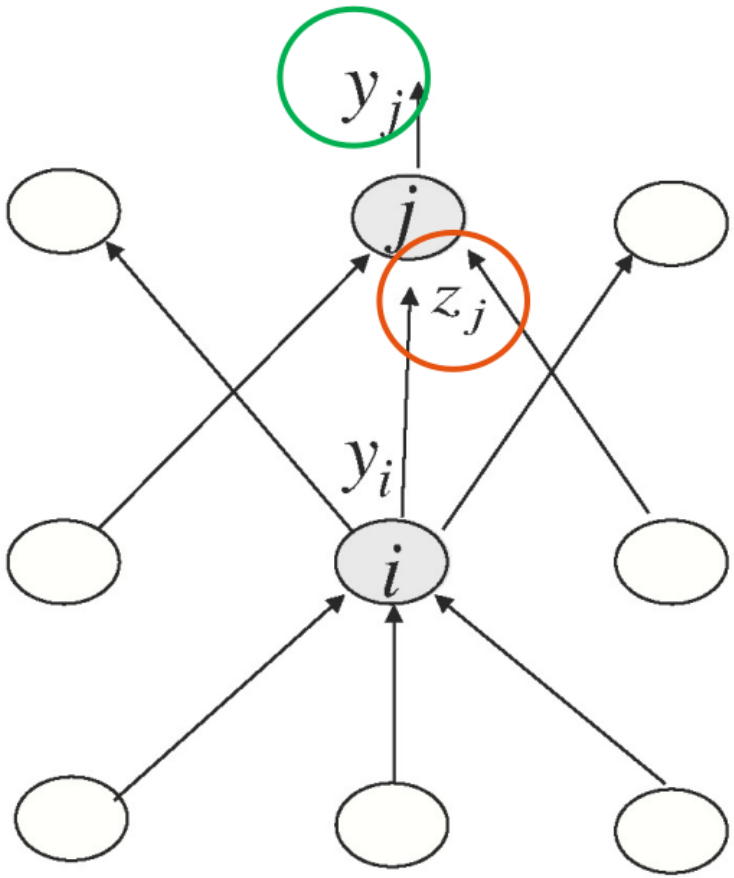
$$\frac{\partial L}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial L}{\partial y_j} = y_j\left(1-y_j\right)\frac{\partial L}{\partial y_j}$$

$$\frac{\partial L}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial L}{\partial z_j} = \sum_j w_{ij}\frac{\partial L}{\partial z_j}$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial L}{\partial z_j} = y_i\frac{\partial L}{\partial z_j}$$

Backpropagation

Backpropagating $\frac{\partial L}{\partial y}$ through multiple layers



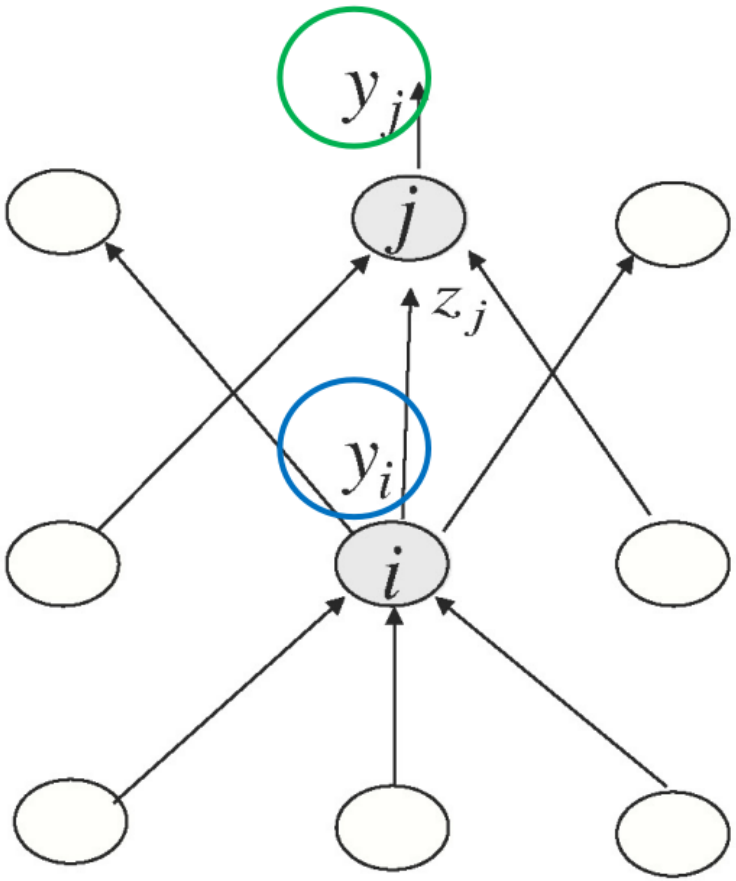
$$\frac{\partial L}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial L}{\partial y_j} = y_j\left(1-y_j\right)\frac{\partial L}{\partial y_j}$$

$$\frac{\partial L}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial L}{\partial z_j} = \sum_j w_{ij}\frac{\partial L}{\partial z_j}$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial L}{\partial z_j} = y_i\frac{\partial L}{\partial z_j}$$

Backpropagation

Backpropagating $\frac{\partial L}{\partial y}$ through multiple layers



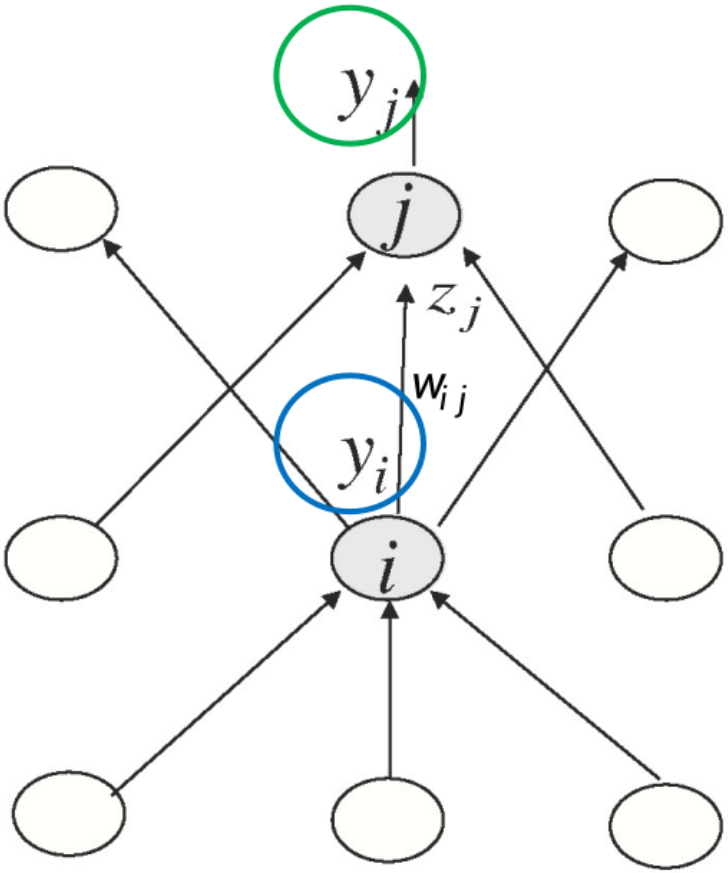
$$\frac{\partial L}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial L}{\partial y_j} = y_j\left(1-y_j\right)\frac{\partial L}{\partial y_j}$$

$$\frac{\partial L}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial L}{\partial z_j} = \sum_j w_{ij}\frac{\partial L}{\partial z_j}$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial L}{\partial z_j} = y_i\frac{\partial L}{\partial z_j}$$

Backpropagation

Backpropagating $\frac{\partial L}{\partial y}$ through multiple layers



$$\frac{\partial L}{\partial z_j} = \frac{dy_j}{dz_j}\frac{\partial L}{\partial y_j} = y_j\left(1-y_j\right)\frac{\partial L}{\partial y_j}$$

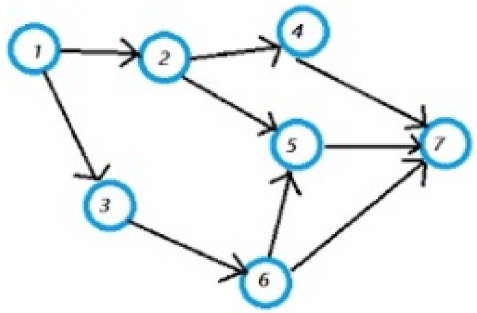
$$\frac{\partial L}{\partial y_i} = \sum_j \frac{dz_j}{dy_i}\frac{\partial L}{\partial z_j} = \sum_j w_{ij}\frac{\partial L}{\partial z_j}$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}}\frac{\partial L}{\partial z_j} = y_i\frac{\partial L}{\partial z_j}$$

- Recall the gradient descent algorithm $\mathbf{w} \leftarrow \mathbf{w} - \alpha\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$

Back propagation

- Let $v_1, ..., v_N$ denote the variables following the topological ordering of the computation graph(*i.e.*, parents come before children.)

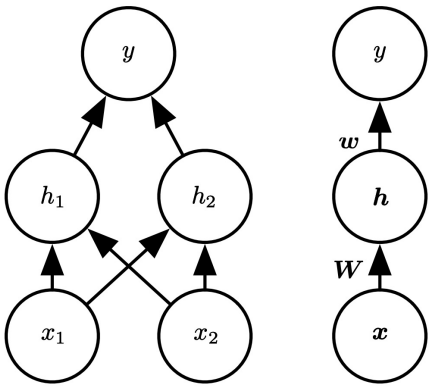


- We're trying to compute derivatives of some function (*e.g.*, loss function) *w.r.t.* $v_1, ..., v_N$.

forward pass
$$\left[\begin{array}{l} \text{For } i = 1, \ldots, N \\ \qquad \text{Compute } v_i \text{ as a function of } \mathrm{Pa}\,(v_i) \end{array}\right.$$

backward pass
$$\left[\begin{array}{l} \overline{v}_N = 1 \\ \text{For } i = N-1, \ldots, 1 \\ \qquad \overline{v}_i = \sum_{j \in \mathrm{Ch}(v_i)} \overline{v}_j \cdot \frac{\partial v_j}{\partial v_i} \end{array}\right.$$

Computational cost



The computational cost of backward pass is about two times of that of forward pass, and is square of the width of each hidden layer.

- **Basics**: $\mathbf{A}^{a\times b}\cdot\mathbf{B}^{b\times c}$ takes $O(a\cdot b\cdot c)$; $\frac{d\ \mathbf{a}^{d_1\times 1}}{d\ \mathbf{b}^{d_2\times 1}}$ takes $O(d_1\cdot d_2)$.
- **Forward pass** (calculating activation of each layer):

$$y = g_1(\mathbf{w}^\top \boldsymbol{h} + b),\ \ \boldsymbol{h} = g_2(\mathbf{W}\mathbf{x} + \mathbf{c})$$

where $\mathbf{W}\in\mathbb{R}^{m\times d}, \mathbf{x}\in\mathbb{R}^{d\times 1}, \boldsymbol{h}, \mathbf{w}\in\mathbb{R}^{m\times 1}$. The cost is

$$O_F = O(md + m).$$

Note: here the costs of g_1, g_2 are ignored, as they are small compared to the cost of their inside arguments.

- **Backward pass** (calculating parameter derivative of each layer): $\frac{dL}{d\mathbf{w}} = \frac{dL}{dy}\cdot\frac{dy}{d\mathbf{w}}$, $\frac{dL}{d\mathbf{W}} = \frac{dL}{dy}\cdot\frac{dy}{d\mathbf{h}}\cdot\frac{d\mathbf{h}}{d\mathbf{W}}$.
The computational cost is:

$$O_B = O_{\mathbf{w}}(1+m+m) + O_{\mathbf{W}}(m+m^2d+m+m^2d) = O(2(m^2d+2m))$$

Note: The cost consists of both each derivative and the product between derivatives.

Summary

What you should know about neural networks:

- **Perceptron model**: definition, property
- **Multi-layer feedforward neural network**: definition, property
- **Backdpropagation**: forward pass, back-ward pass
- **Computational graph (CG)**: transformation between network and CG, using CG to run backpropagation

1 Recall of previous classification models

2 Neural networks
- Perceptron model
- Multi-layer feedforward neural networks

3 Backpropagation

4 Deep neural networks

Expressive power of deep linear networks

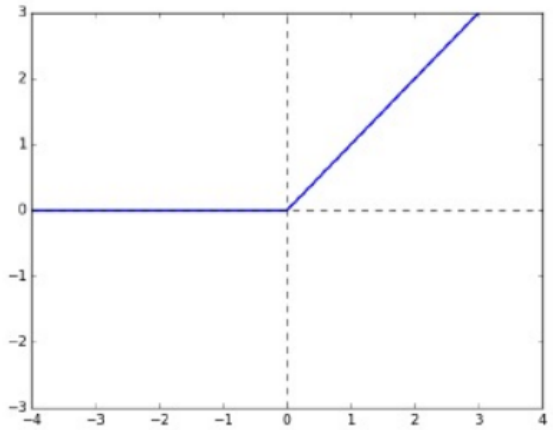
- We've seen that there are some functions that linear classifiers can't represent. Are deep networks any better?
- Any sequence of *linear* layers can be equivalently represented with a single layer.

$$\mathbf{y} = \underbrace{\mathbf{W}^{(3)}\mathbf{W}^{(2)}\mathbf{W}^{(1)}}_{\triangleq \mathbf{W}'}\mathbf{x}$$

Deep linear networks are not more expressive than linear regression.

Expressive power of one non-linear hidden layer network

- Multilayer feed-forward neural nets with nonlinear activation functions are universal function approximators: they can approximate any function arbitrary well [1].
- This has been shown for various activation functions (thresholds, logistic, ReLU, etc.)
  - Even though, ReLU is"almost" linear, it's nonlinear enough.



Reference: K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural Networks, vol. 2, no. 5, pp. 359–366, 1989.

Why we need deep neural networks

- Neural networks with one hidden layer is enough to represent (not learn) an approximation of any function to an arbitrary degree of accuracy

- So, why deep neural networks?
  - Shallow net may need (exponentially) more hidden neurons (*i.e.*, very wide hidden layer)
  - Shallow net may over-fit more

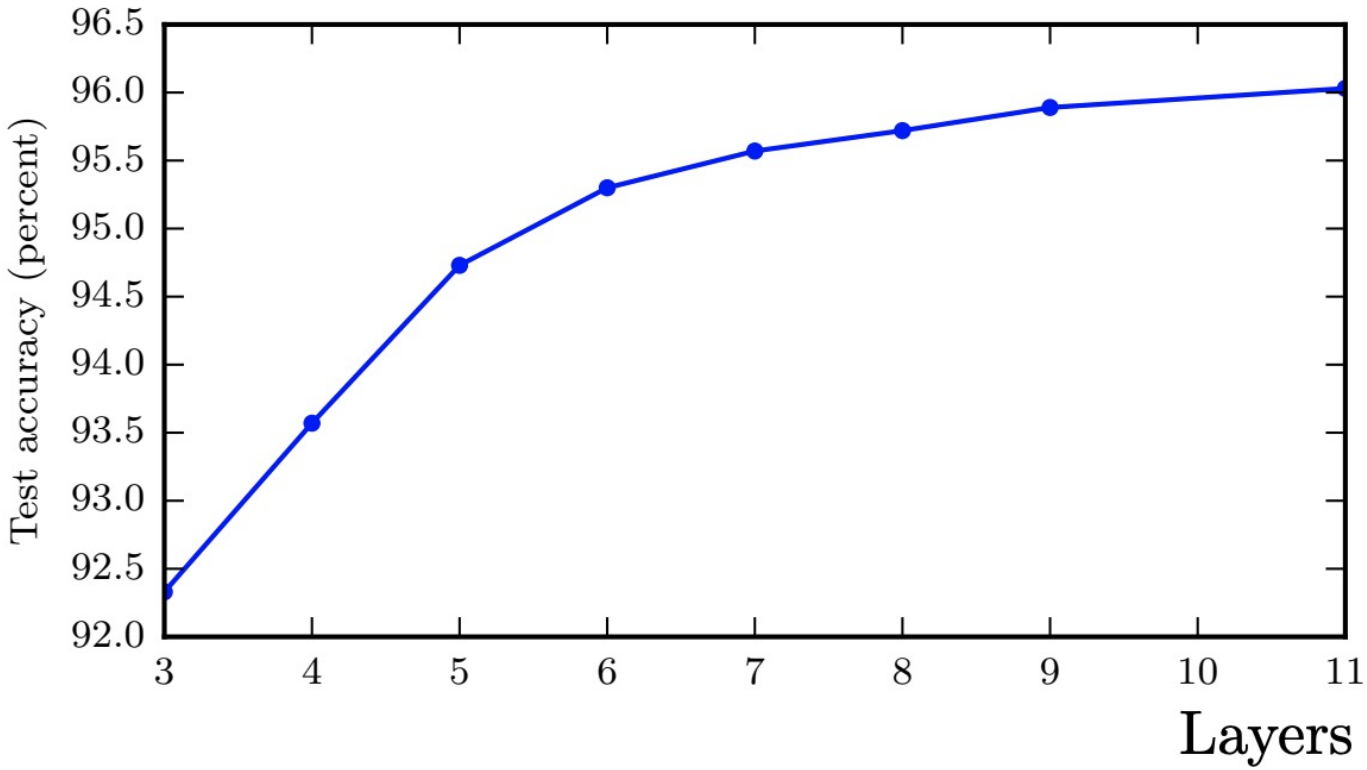
If you are interested in the details of above claims, please refer to the following videos and blog:

Video 1: `https://www.youtube.com/watch?v=KKT2VkTdFyc&list=PLJV_el3uVTsOh1F5e`
`index=1`

Video 2: `https://www.youtube.com/watch?v=FN8jclCrqY0&list=PLJV_el3uVTsOh1F5e`
`index=2`

Blog: `https://medium.com/@jacklindsai/why-is-deep-learning-deep-d4305e596b7`

Better Generalization with Greater Depth



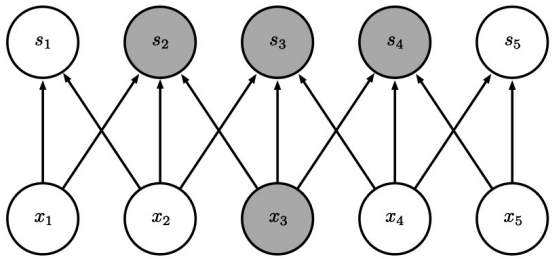
Motivation of convolutional neural networks (CNNs)

- Although deeper neural networks have larger representation capability and better generalization, it is difficult to extend the multi-layer feed-forward neural networks to very deep, since every layer is fully connected.
- For example, given a large image, the number of parameters could be very large. How to alleviate such limitation?
- People resort to two tricks:
  - Sparse connection
  - Shared parameters

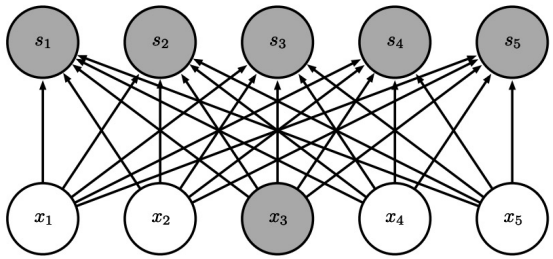
Sparse connection

- Each input neuron only connects to partial output neurons
- Each output neuron only connects to a few neighboring input neurons. And the range of input neurons is called receptive field.

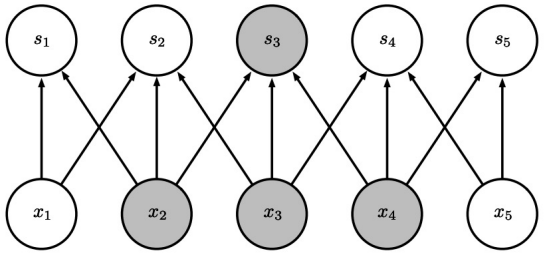
Sparse connections due to small convolution kernel



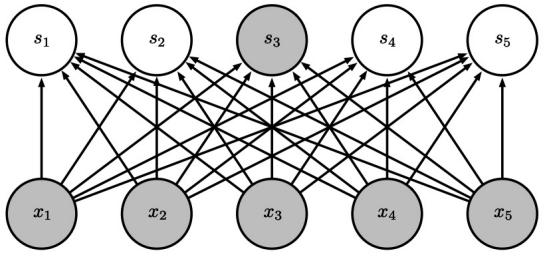
Dense connections



Sparse connections due to small convolution kernel

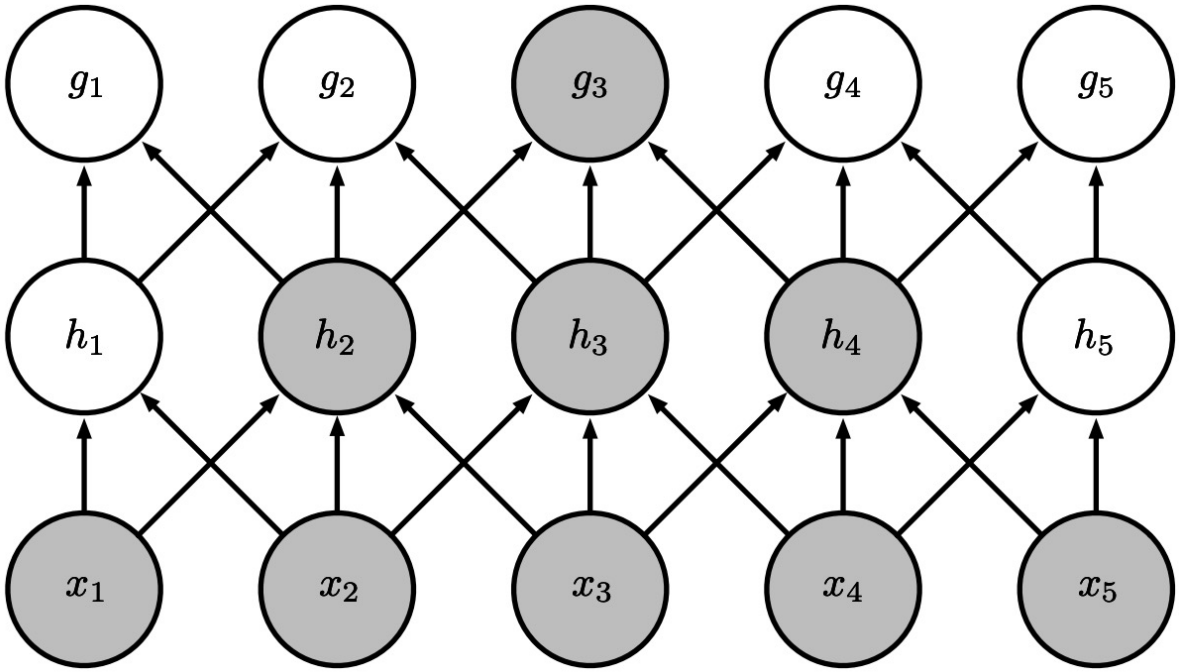


Dense connections



Growing receptive field

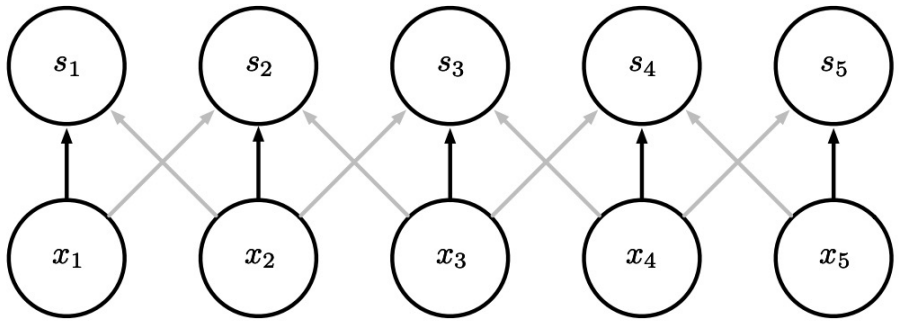
- Receptive field will increase along the layer goes deeper.



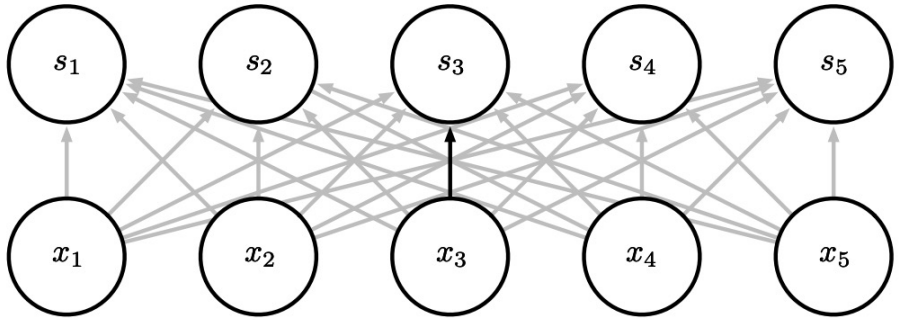
Shared parameters

- Parameters at different spatial locations are shared.
- Consequently, as shown in the following example, the number of parameters in the convolution filter is 3, while $5 \times 5 = 25$ in the fully connected layer.

Convolution shares the same parameters across all spatial locations

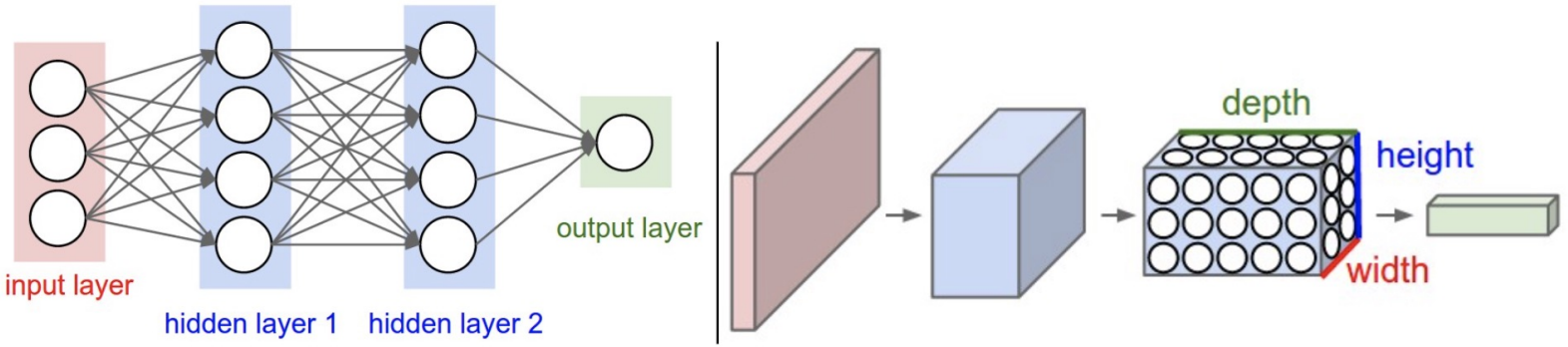


Traditional matrix multiplication does not share any parameters



Convolutional neural networks

In the next lecture, we will introduce more details about convolutional neural networks (CNNs).



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).