EE488 Special Topics in EE <Deep Learning and AlphaGo>

Sae-Young Chung

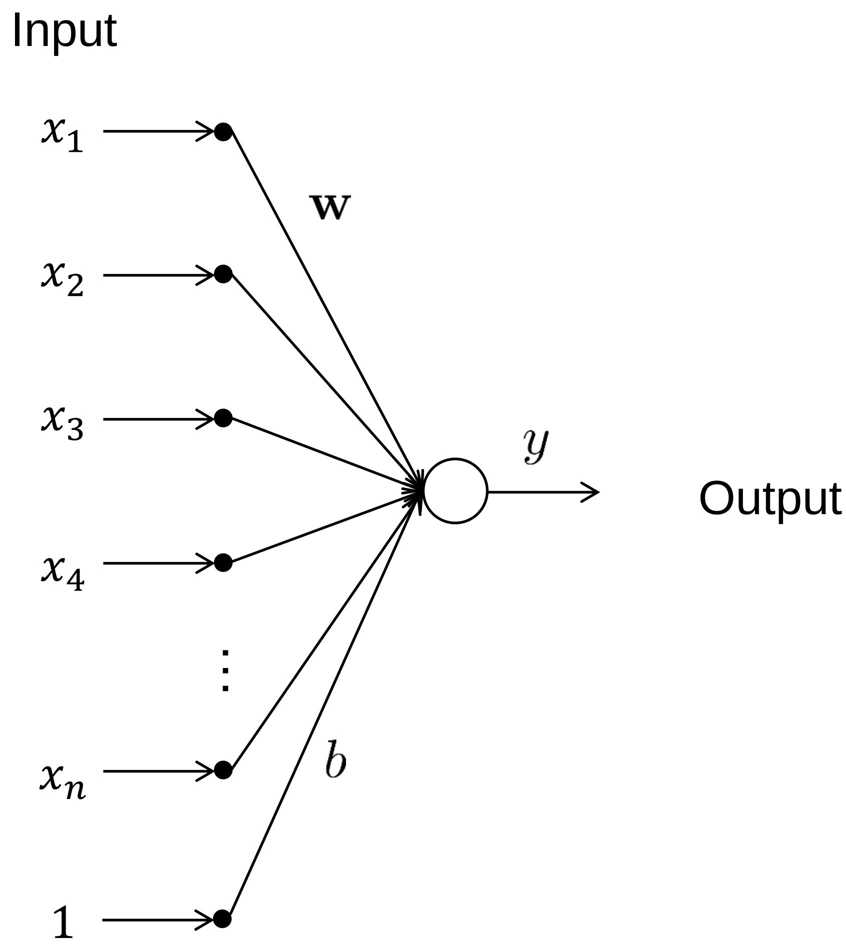
Lecture 6

September 25, 2017

Chap. 6 Deep Feedforward Networks

- Perceptron
- Multilayer perceptron
- XOR example
- Cost functions
- Output units
- Sigmoid output units
- Softmax output units
- Hidden units
- Back-propagation algorithm

Perceptron



Simple artificial neuron capable of learning to perform binary classification

$$y = g(\mathbf{w} \cdot \mathbf{x} + b)$$

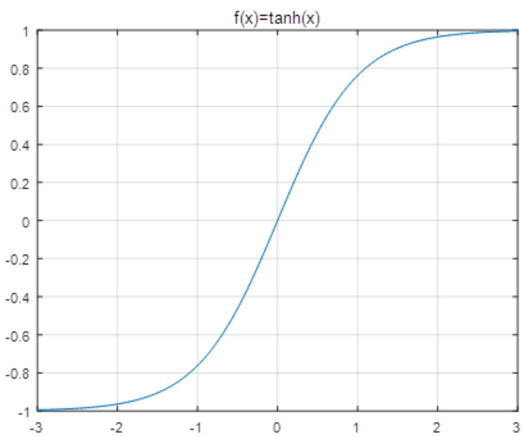
$$g(u) = \begin{cases} 1 & \text{if } u > 0 \\ 0 & \text{otherwise} \end{cases}$$

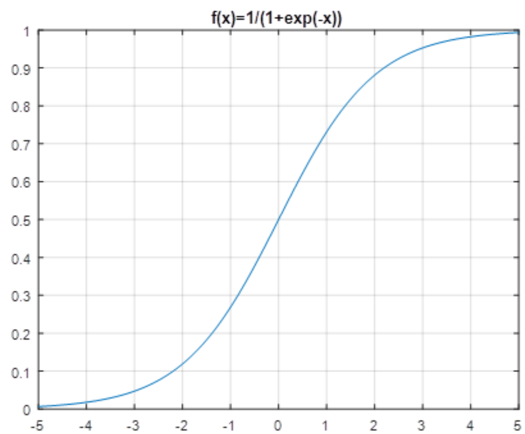
$$\theta = (\mathbf{w}, b)$$

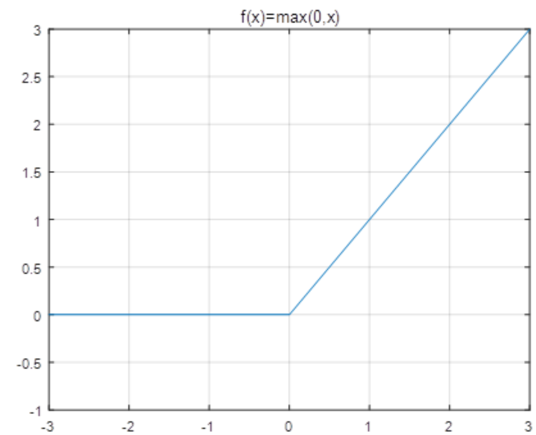
Rosenblatt, 1958

Activation Functions

- Activation function: $g(\cdot)$
  - Unit step: $g(x) = 1$ if $x = 0$ and $g(x) = 0$ otherwise (original perceptron)
  - Sigmoid: $g(x) = 1/(1 + e^{-x})$
  - tanh: $g(x) = \tanh x$
  - Rectified linear unit (ReLU): $g(x) = \max(0, x)$







Multi-layer Perceptron

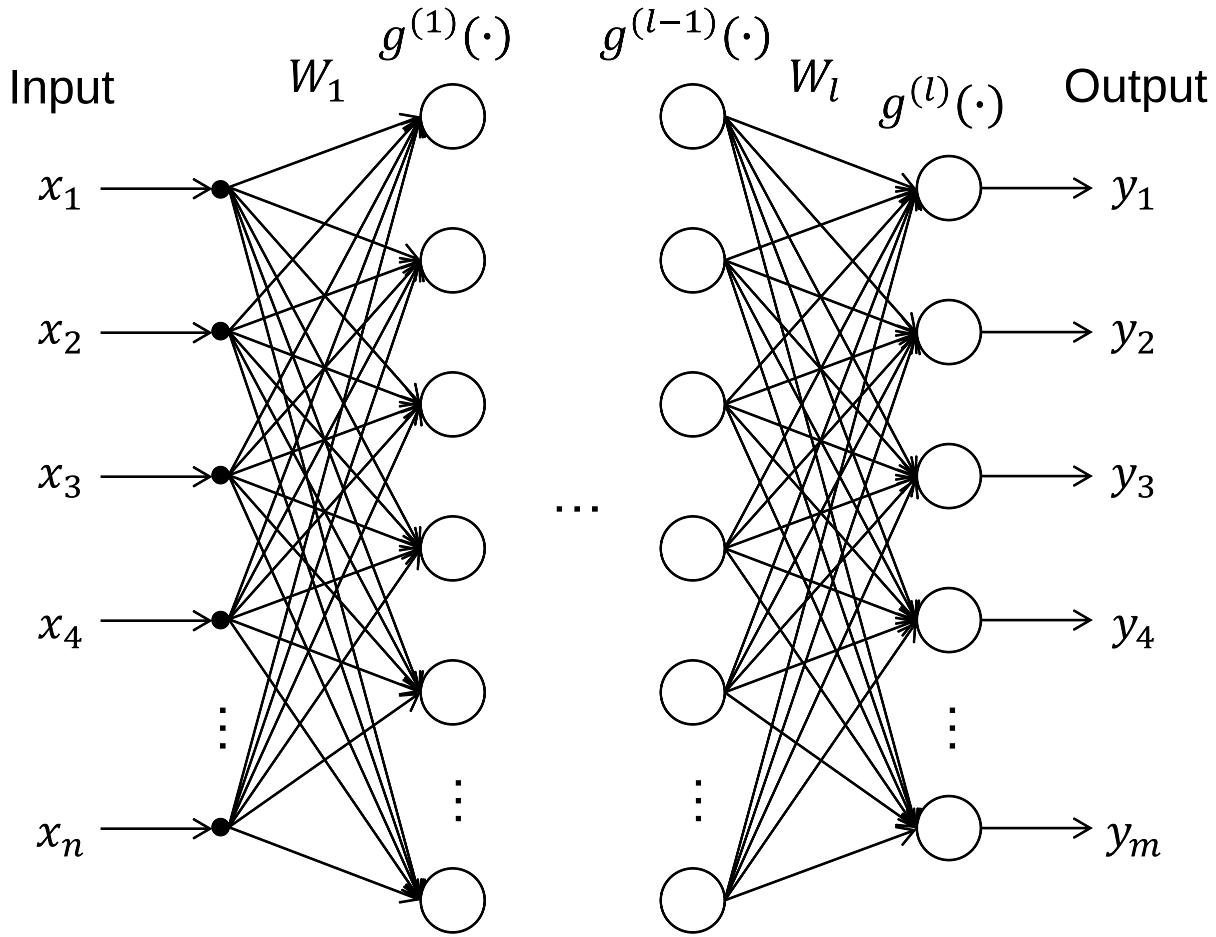
- Multilayer perceptron (MLP): a feedforward neural network having multiple layters
  - Each layer consists of multiple neurons (or units).
  - Original perceptron used step function as an activation function
  - We often use other nonlinear activation functions for MLPs
- Deep feedforward network: a feedforward neural network with many layers

$$\mathbf{y} = f(\mathbf{x}) = f^{(l)}(f^{(l-1)}(\dots(f^{(2)}(f^{(1)}(\mathbf{x}))\dots)$$

output layer 2nd layer 1st layer

- Last layer: output layer
- Layers $1 \sim (l-1)$: hidden layers (their outputs are hidden)

Multi-layer Perceptron



Multi-layer Perceptron

- Fully connected feedforward network

$$\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta}) = g^{(l)}(\mathbf{b}_l + W_l g^{(l-1)}(\mathbf{b}_{l-1} + W_{l-1} ... g^{(1)}(\mathbf{b}_1 + W_1 \mathbf{x})) \dots)$$

  - $\boldsymbol{\theta} = (\mathbf{b}_1, W_1, \dots, \mathbf{b}_l, W_l)$: parameters of the neural network

- Convolutional neural network uses convolutions in some layers
- Recurrent neural network: a neural network with recurrent connections, e.g.,
  - Hopfield network
  - LSTM (Long short-term memory)

XOR Example

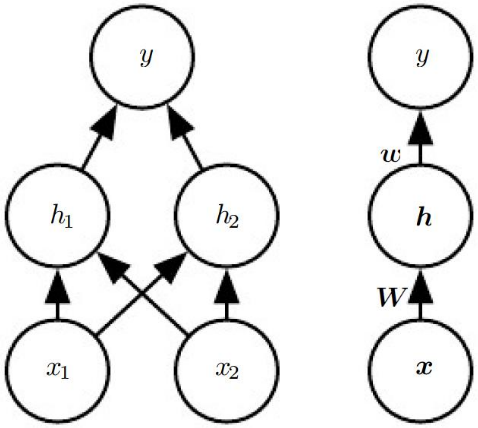


Fig. 6.2

- Having a single layer is not enough for solving the XOR problem
- Two layers with one output neuron and two neurons in the hidden layer

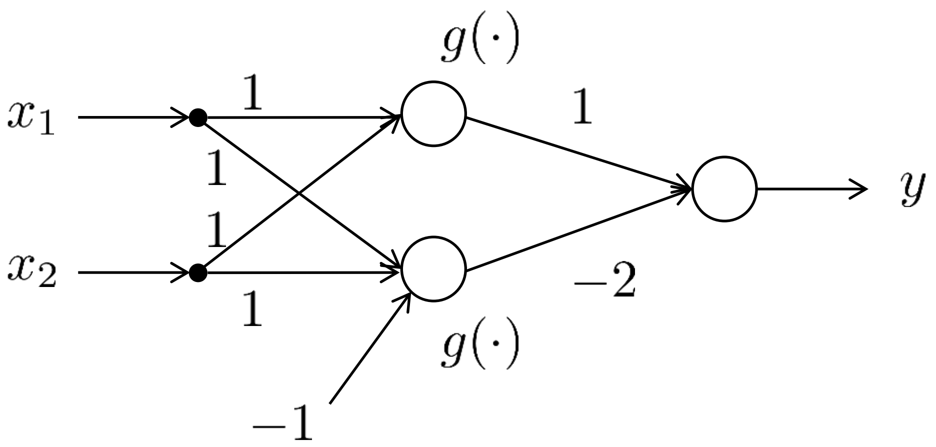
$$y = f^{(2)}(\mathbf{h}; \mathbf{w}, b) = \mathbf{w}^T \mathbf{h} + b$$
$$\mathbf{h} = f^{(1)}(\mathbf{x}; \mathbf{W}, \mathbf{c}) = g(\mathbf{W}^T \mathbf{x} + \mathbf{c})$$

- Let's assume $g(\cdot)$ is elementwise ReLU

XOR Example

- Let's assume

$$\mathbf{W} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}, \quad b = 0$$

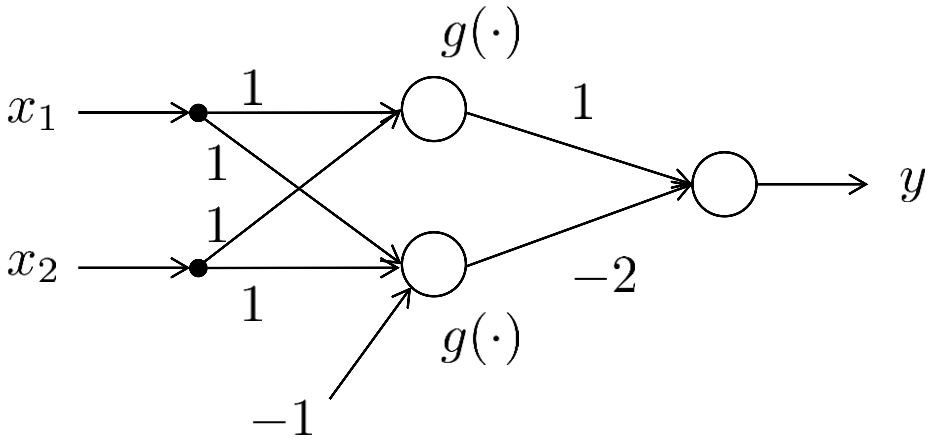


XOR Example

- XOR problem

$$\mathbf{X} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}, \quad \mathbf{XW} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{pmatrix}, \quad \mathbf{XW} + \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \mathbf{c}^T = \begin{pmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

$$g(\mathbf{XW} + \mathbf{c}) = \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$



XOR Example

- How to train the network given the following?

$$\mathbf{X} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

- We can use gradient descent to minimize the loss function, e.g.,

$$J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\mathbf{x} \in \mathbf{X}} (f^*(\mathbf{x}) - f(\mathbf{x}; \boldsymbol{\theta}))^2$$

$$\boldsymbol{\theta} = (\mathbf{W}, \mathbf{c}, \mathbf{w}, b)$$

- The following is a global minimum point

$$\mathbf{W} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}, \quad b = 0$$

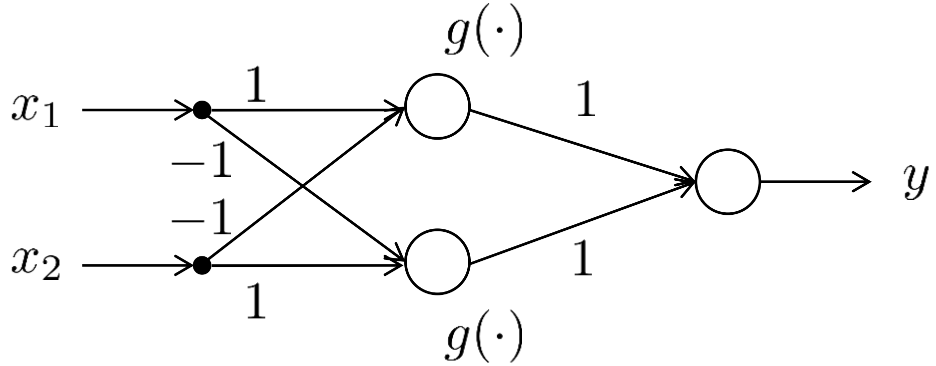
XOR Example

- Another global minimum

$$\mathbf{W} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b = 0$$

- Verification

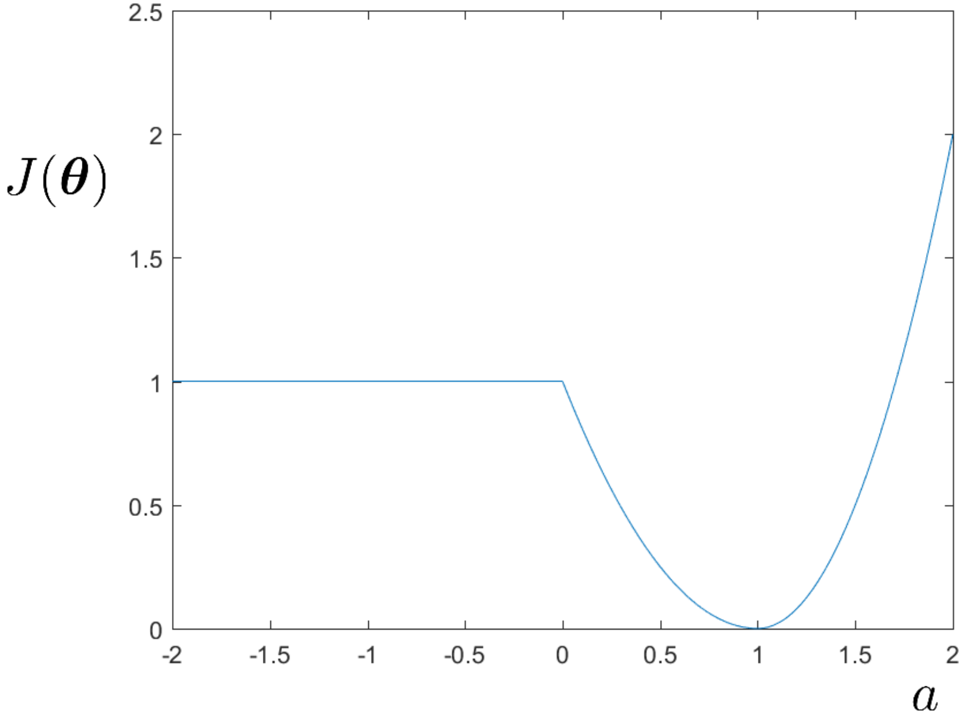
$$\mathbf{X} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}, \quad \mathbf{XW} = \begin{pmatrix} 0 & 0 \\ -1 & 1 \\ 1 & -1 \\ 0 & 0 \end{pmatrix}, \quad g(\mathbf{XW}) = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \mathbf{y} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$



XOR Example

-

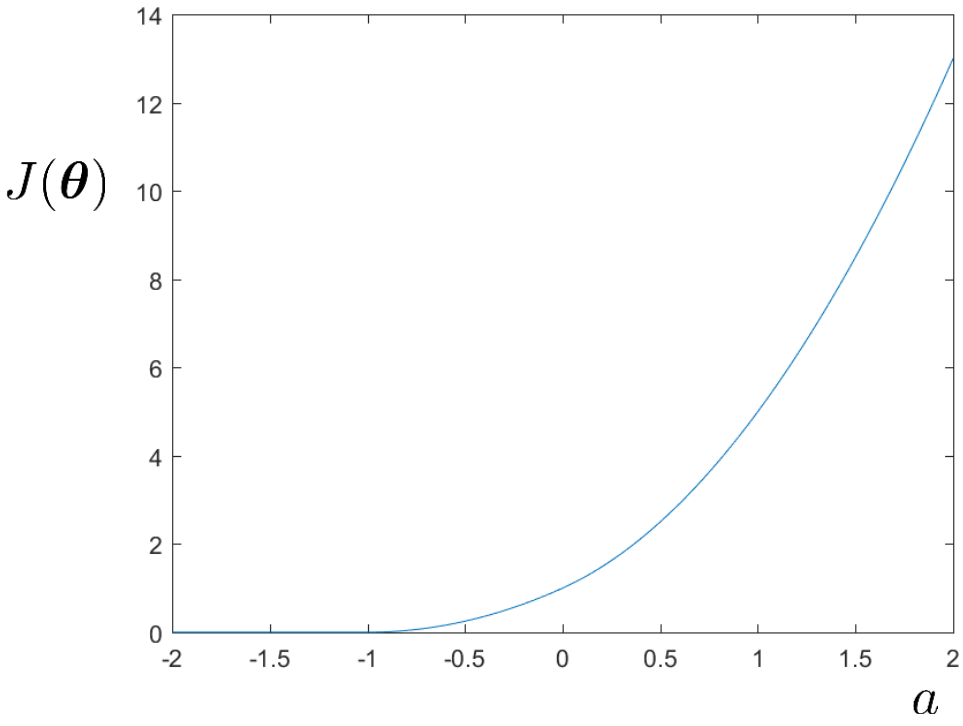
$$\mathbf{W} = \begin{pmatrix} a & -1 \\ -1 & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b = 0$$



XOR Example

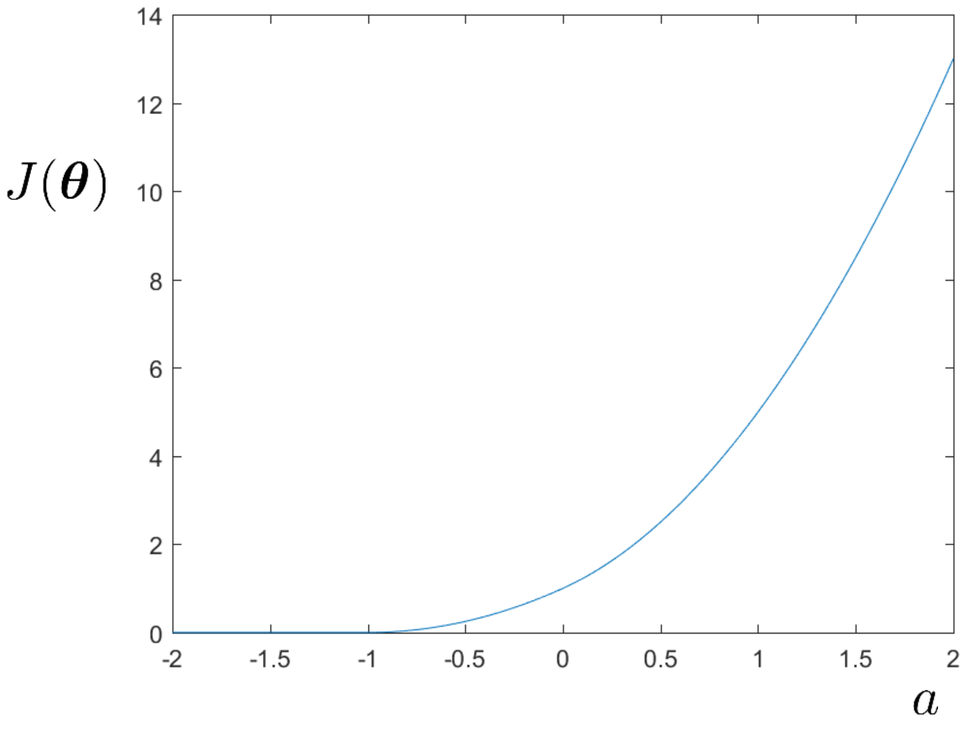
-

$$\mathbf{W} = \begin{pmatrix} 1 & a \\ -1 & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b = 0$$



XOR Example

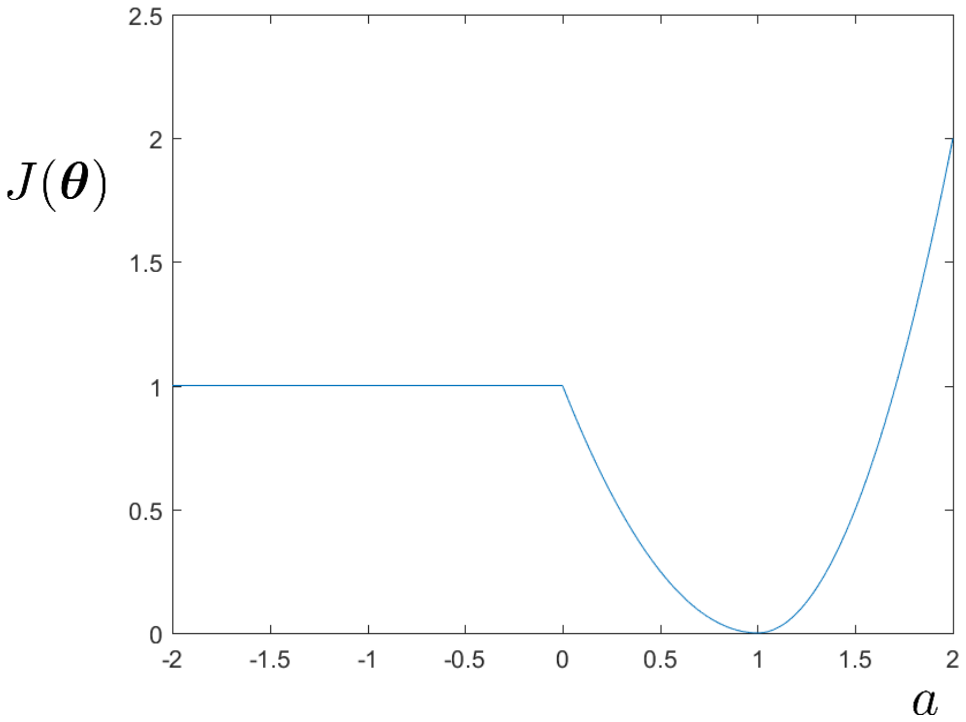
- $$\mathbf{W} = \begin{pmatrix} 1 & -1 \\ a & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b = 0$$



XOR Example

-

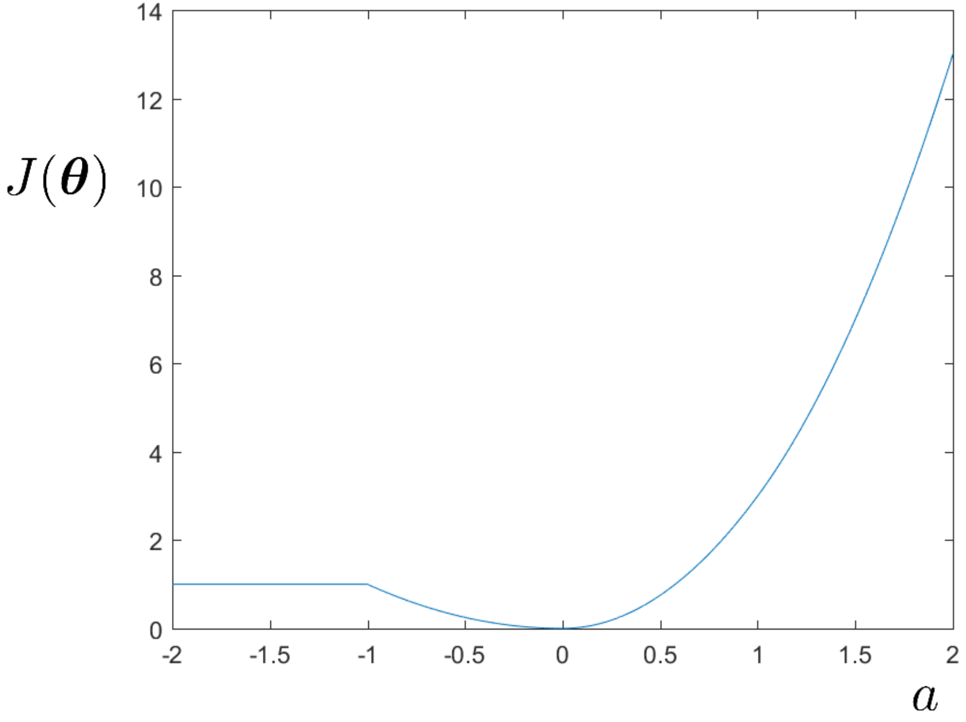
$$\mathbf{W} = \begin{pmatrix} 1 & -1 \\ -1 & a \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b = 0$$



XOR Example

-

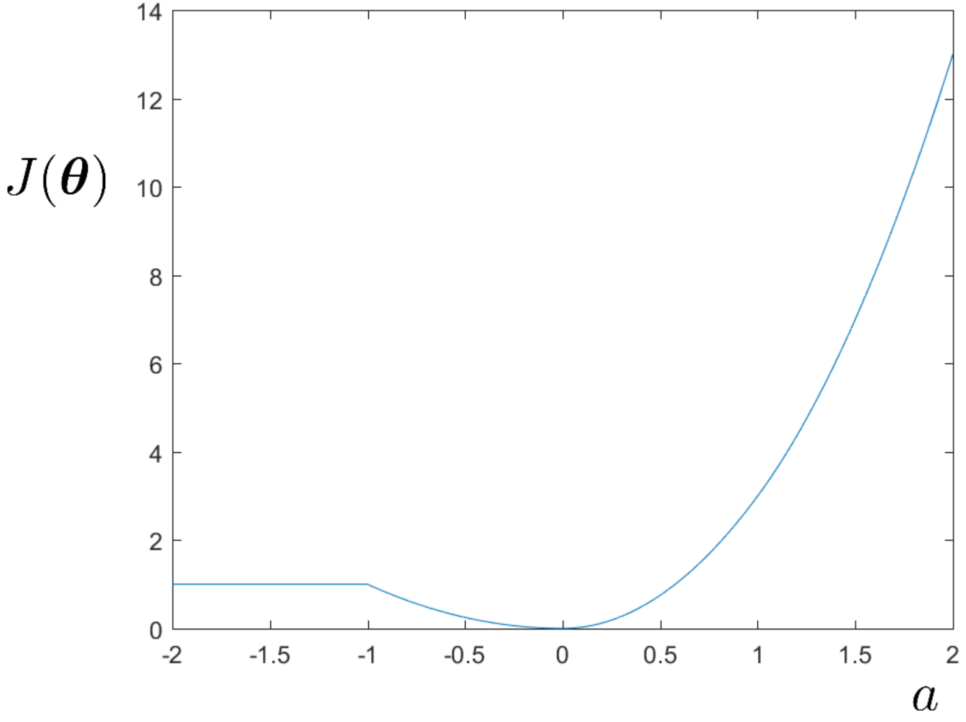
$$\mathbf{W} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} a \\ 0 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b = 0$$



XOR Example

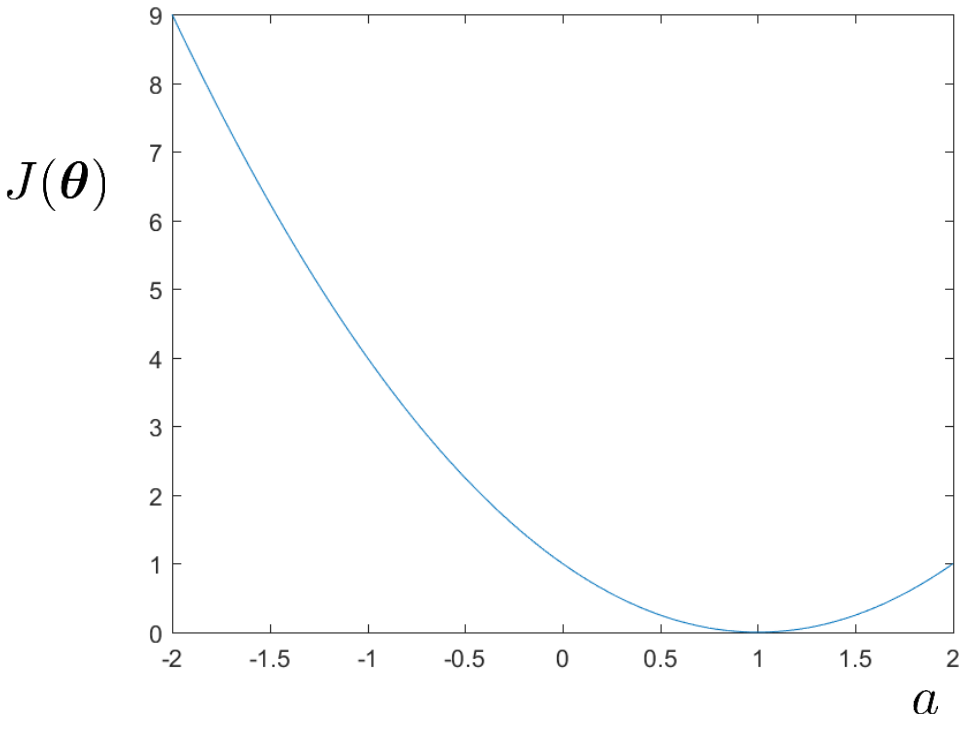
-

$$\mathbf{W} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ a \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b = 0$$



XOR Example

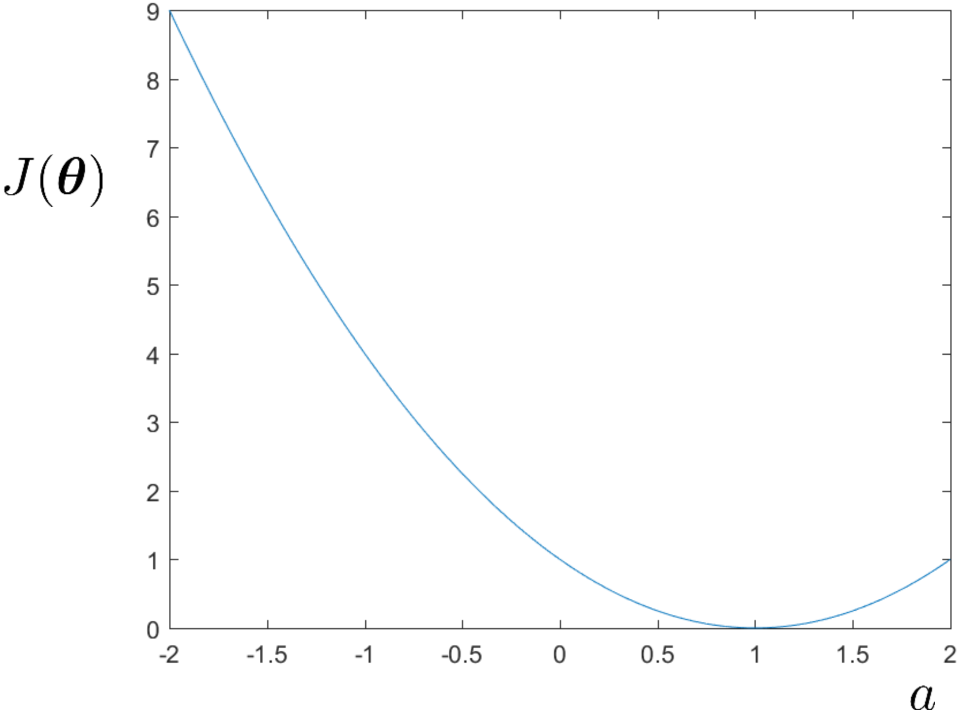
- $$\mathbf{W} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} a \\ 1 \end{pmatrix}, \quad b = 0$$



XOR Example

-

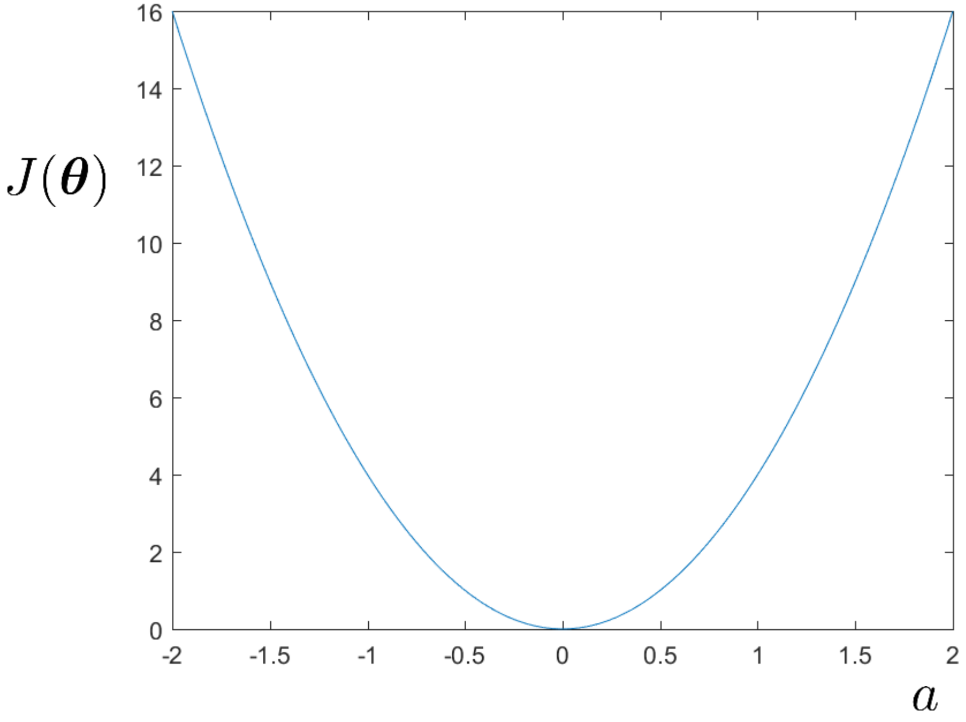
$$\mathbf{W} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ a \end{pmatrix}, \quad b = 0$$



XOR Example

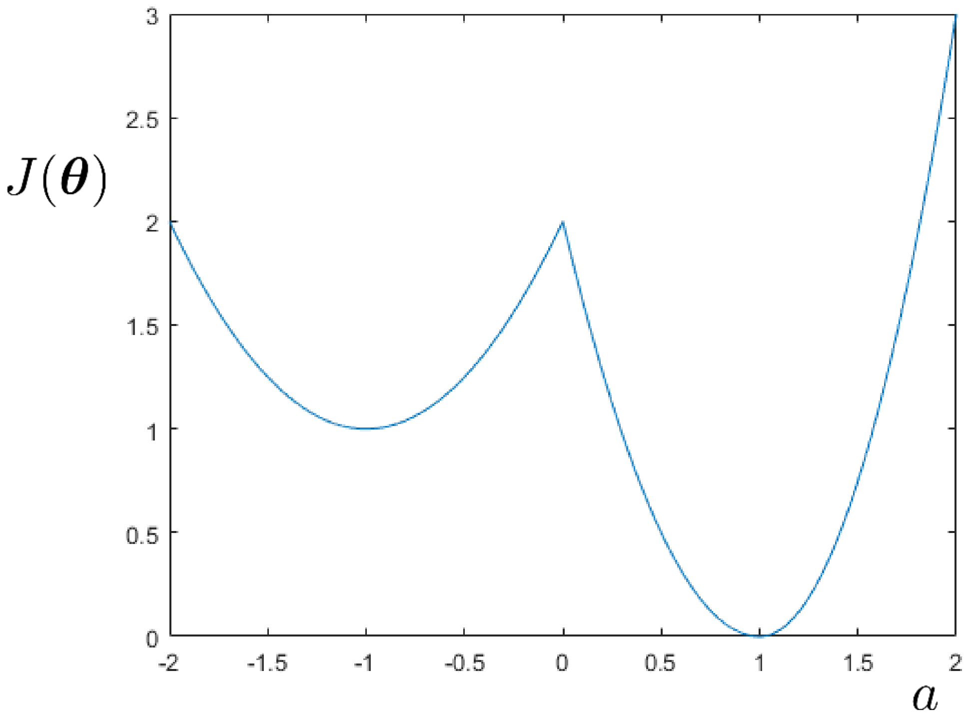
-

$$\mathbf{W} = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b = a$$



XOR Example

- $$\mathbf{W} = \begin{pmatrix} a & -a \\ -a & 1 \end{pmatrix}, \quad \mathbf{c} = \begin{pmatrix} 0 \\ a-1 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b = 0$$



Cost Functions

- Commonly used cost functions
  - Cross entropy between model distribution and training data

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\mathbf{y}|\mathbf{x})$$

  - If $p_{\text{model}}(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}; f(\mathbf{x};\boldsymbol{\theta}), I)$, then

$$J(\boldsymbol{\theta}) = \frac{1}{2}\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \|\mathbf{y} - f(\mathbf{x},\boldsymbol{\theta})\|^2 + \text{const}$$

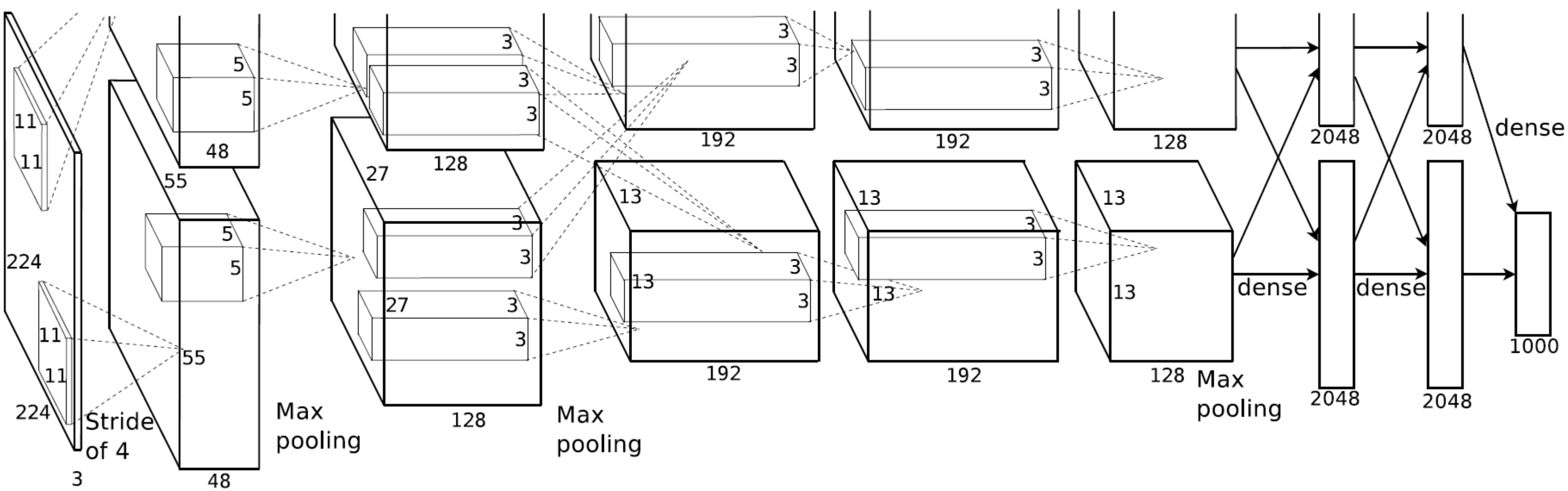
- Regularization term can be added

Output Units

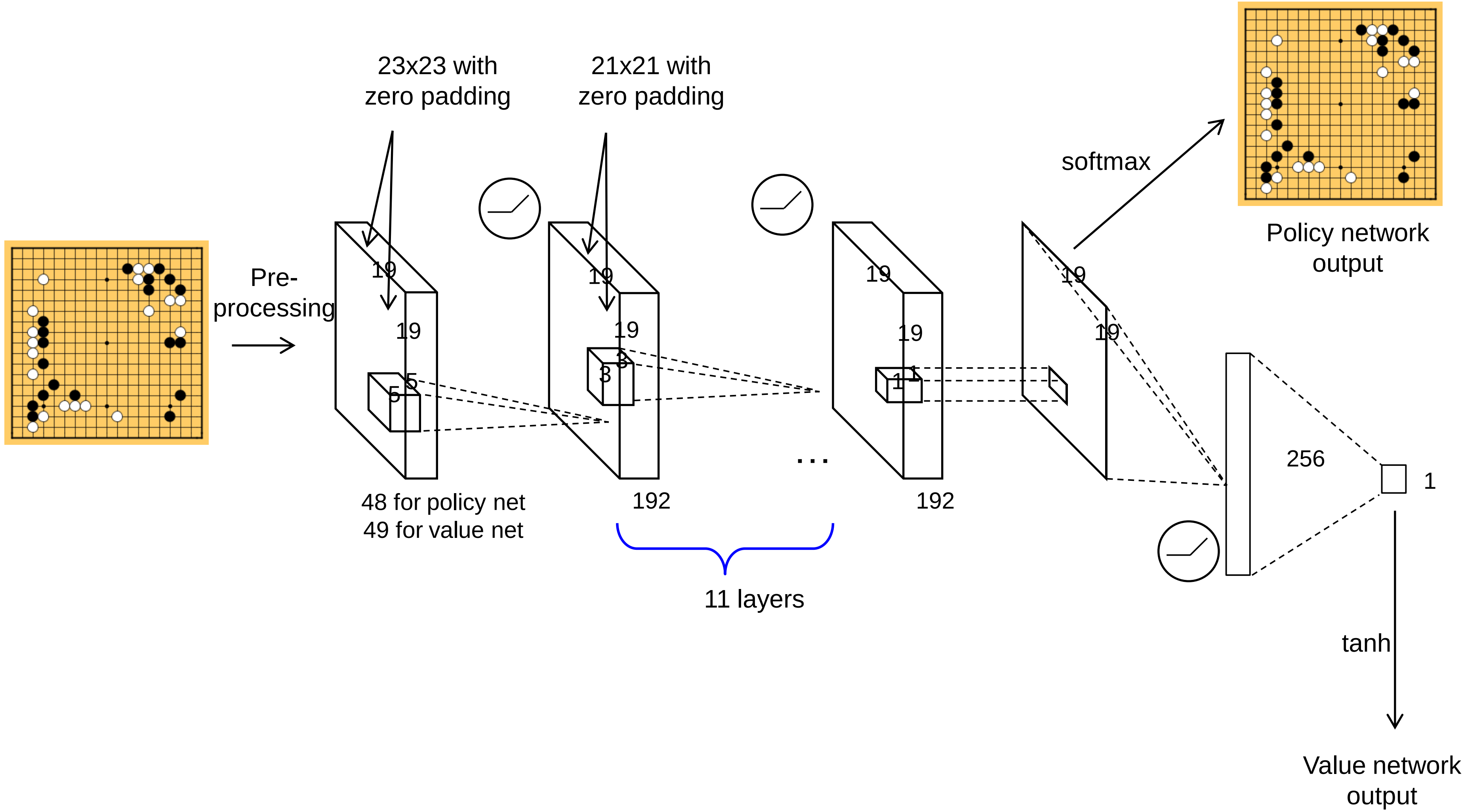
- Commonly used output units
  - Linear: useful for real outputs, e.g., Gaussian distribution model $p(\mathbf{y}|\mathbf{h}) = \mathcal{N}(\mathbf{y}; \hat{\mathbf{y}}, I)$, often used with MSE loss function (equivalent to NLL or cross-entropy loss function for Gaussian)
$$\hat{\mathbf{y}} = \mathbf{W}^T\mathbf{h} + \mathbf{b}$$
  - Sigmoid: useful for (soft) binary outputs, e.g., Bernoulli distribution model, often used with NLL loss function
$$\hat{\mathbf{y}} = \sigma(\mathbf{W}^T\mathbf{h} + \mathbf{b})$$
  - tanh: similar to sigmoid, but the range is between -1 and 1
  - Softmax: useful for (soft) n-ary outputs, e.g., multinoulli distribution model, often used with NLL loss function
$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{W}^T\mathbf{h} + \mathbf{b})$$

CNN for Image Classification

- Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton, “ImageNet classification with deep convolutional neural networks”, NIPS 2012



CNN in AlphaGo



Sigmoid Units

- Sigmoid unit

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{h} + b)$$

- What about the following?

$$\hat{y} = \max\left\{0, \min\left\{1, \mathbf{w}^T \mathbf{h} + b\right\}\right\}$$

  – It works too, but the gradient vanishes if $\mathbf{w}^T \mathbf{h} + b < 0$ or $\mathbf{w}^T \mathbf{h} + b > 1$.

  – Harder to train when gradient vanishes since the gradient does not give a direction.

Sigmoid Units

- Let $z = \mathbf{w}^T\mathbf{h} + b$ and $\hat{y} = \sigma(z)$. $\mathbf{h}$ is the output of the final hidden layer.
- Assume $p_{\text{model}}(y = 1|z) = \sigma(z)$ and $p_{\text{model}}(y = 0|z) = 1 - \sigma(z) = \sigma(-z)$, then $p_{\text{model}}(y|z) = \sigma((2y-1)z)$ for $y \in \{0, 1\}$.
- The ML minimizes the following NLL

$$
\begin{aligned}
-\sum_{i=1}^{m} \log p_{\text{model}}(y^{(i)}|\mathbf{x}^{(i)}; \boldsymbol{\theta}) &= -\sum_{i=1}^{m} \log \sigma((2y^{(i)} - 1)z^{(i)}) \\
&= \sum_{i=1}^{m} \log[1 + \exp((1 - 2y^{(i)})z^{(i)})] \\
&= \sum_{i=1}^{m} \zeta((1 - 2y^{(i)})z^{(i)})
\end{aligned}
$$

- i.e., it will try to make $z^{(i)}$ very positive if $y^{(i)} = 1$ and make $z^{(i)}$ very negative if $y^{(i)} = 0$.

Recap – Softplus function

- Softplus function

$$\zeta(x) = \log(1 + \exp(x))$$

Softened version of $\max(0, x)$

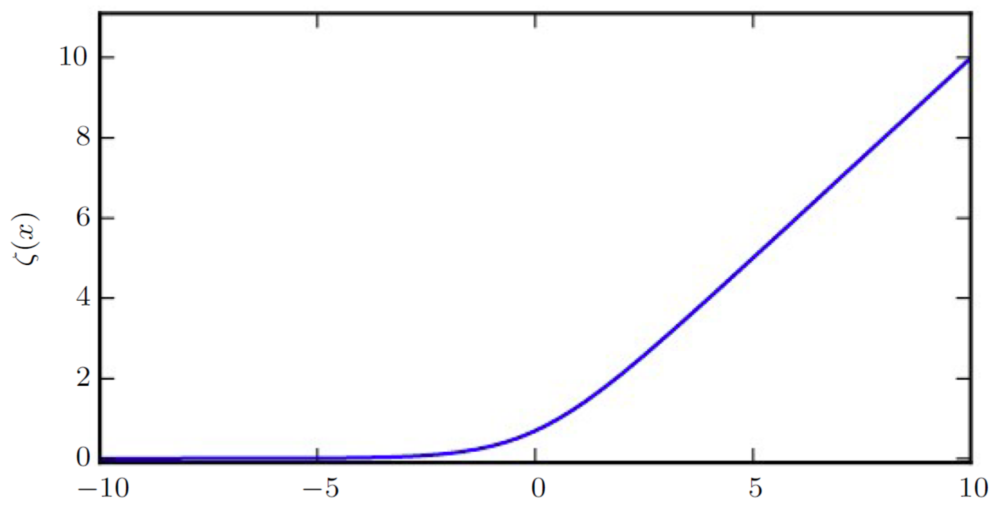


Figure 3.4: The softplus function.

Sigmoid Units

- The gradient of the cost function with respect to $z^{(i)}$ is almost constant if $z^{(i)}$ is in the wrong regime and will try to influence the GD algorithm to correct it.
- This means the sigmoid output goes well with NLL.
- Exp in sigmoid is "undone" by log in NLL.

Softmax Units

- Let $\mathbf{z} = \mathbf{W}^T\mathbf{h} + \mathbf{b}$ and $\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{z})$, where

$$\mathrm{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- Assume $p_{\text{model}}(y = k|\mathbf{z}) = \mathrm{softmax}(\mathbf{z})_k$, $1 \le k \le n$, then the ML minimizes the following NLL

$$\begin{aligned} -\sum_{i=1}^{m} \log p_{\text{model}}(y^{(i)}|\mathbf{x}^{(i)};\boldsymbol{\theta}) &= -\sum_{i=1}^{m} \log \frac{\exp\left(z_{y^{(i)}}^{(i)}\right)}{\sum_j \exp(z_j^{(i)})} \\ &= \sum_{i=1}^{m} -z_{y^{(i)}}^{(i)} + \log \sum_j \exp(z_j^{(i)}) \end{aligned}$$

i.e., it will try to make the $y^{(i)}$-th output in $\mathbf{z}^{(i)}$ the biggest.

- Note that $\sum_{i=1}^m -z_{y^{(i)}}^{(i)} + \log \sum_j \exp(z_j^{(i)}) \sim \sum_{i=1}^m -z_{y^{(i)}}^{(i)} + \max_j z_j^{(i)}$.

Softmax Units

- The gradient of the NLL cost function with respect to $\mathbf{z}^{(i)}$ is almost constant if $\mathbf{z}^{(i)}$ is in the wrong regime and will try to influence the GD algorithm to correct it.
- This means the softmax output goes well with NLL.
- Exp in softmax is "undone" by log in NLL.
- Note that $\text{softmax}(\mathbf{z}) = \text{softmax}(\mathbf{z} - c)$ and $\sum_j \text{softmax}(\mathbf{z})_j = 1$. This means you only need $n-1$ outputs rather than n outputs.
- In fact, if $n = 2$, then

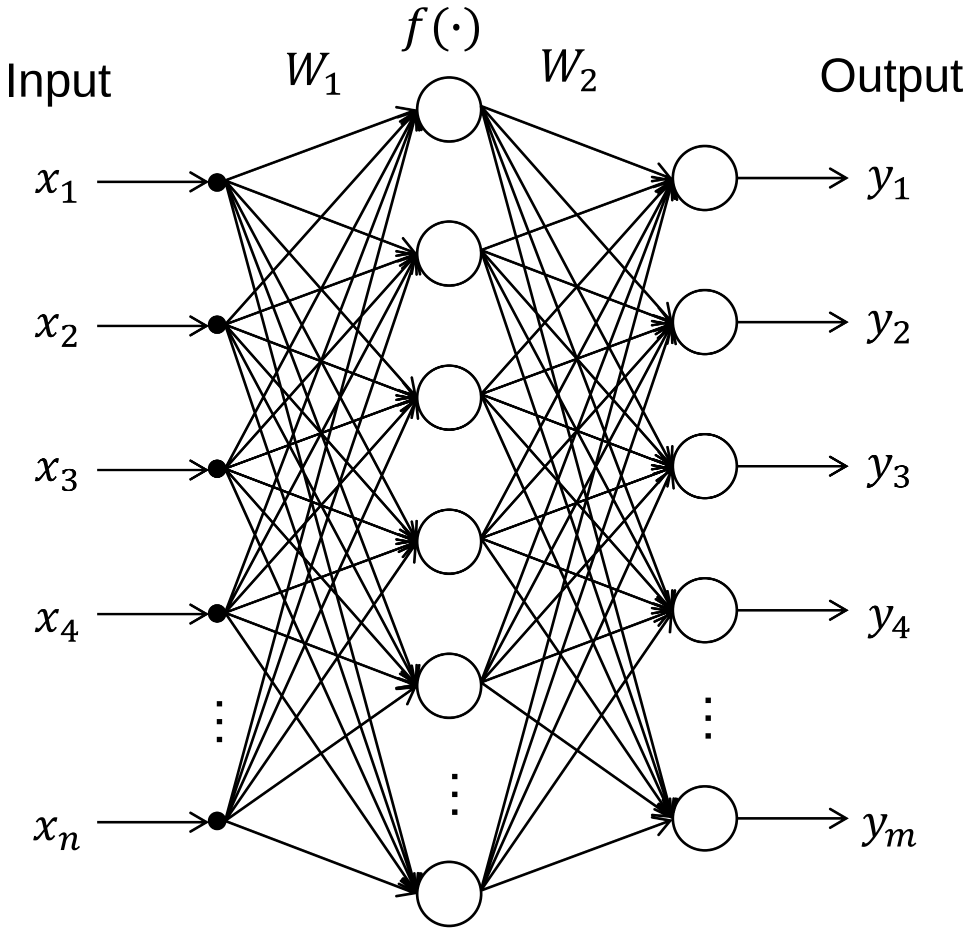
$$\text{softmax}(\mathbf{z})_1 = \frac{\exp(z_1)}{\exp(z_1) + \exp(z_2)} = \frac{1}{1 + \exp(z_2 - z_1)}$$
$$\text{softmax}(\mathbf{z})_2 = \frac{\exp(z_2)}{\exp(z_1) + \exp(z_2)} = \frac{1}{1 + \exp(z_1 - z_2)}$$

i.e., they are functions of $z_2 - z_1$ only and you only need one output, which is exactly what sigmoid does for the Bernoulli case.

Hidden Units

- Rectified linear unit (ReLU): $g(z) = \max\{0, x\}$
  - Default choice for feedforward networks (lower computational complexity than sigmoid)
  - Provides constant gradient in the regime it is active
  - Not differentiable only at $z = 0$, which is OK
  - Good to initialize biases to have small positive values (e.g., 0.1) so that ReLU is more likely to be active initially
- Maxout unit: $g(z)_i = \max_{(i-1)k+1 \le j \le ik} z_j$
- Sigmoid: $g(z) = \sigma(z)$
  - Not recommended for feedforward networks due to its saturation behavior, higher complexity than ReLU
  - Often used for recurrent networks and autoencoders
- Tanh: $g(z) = \tanh(z) = 2\sigma(2z) - 1$
- Linear units: $\mathbf{W}^T\mathbf{x} + \mathbf{b}$ or $\mathbf{V}^T\mathbf{U}^T\mathbf{x} + \mathbf{b}$ (can be useful if $\mathbf{U}$ and $\mathbf{V}$ have low-rank)

Universal Approximation Theorem



- Cybenko 1989
- A neural network with a single hidden layer with some non-linear activation function and a linear output layer can approximate any continuous function with arbitrary accuracy.

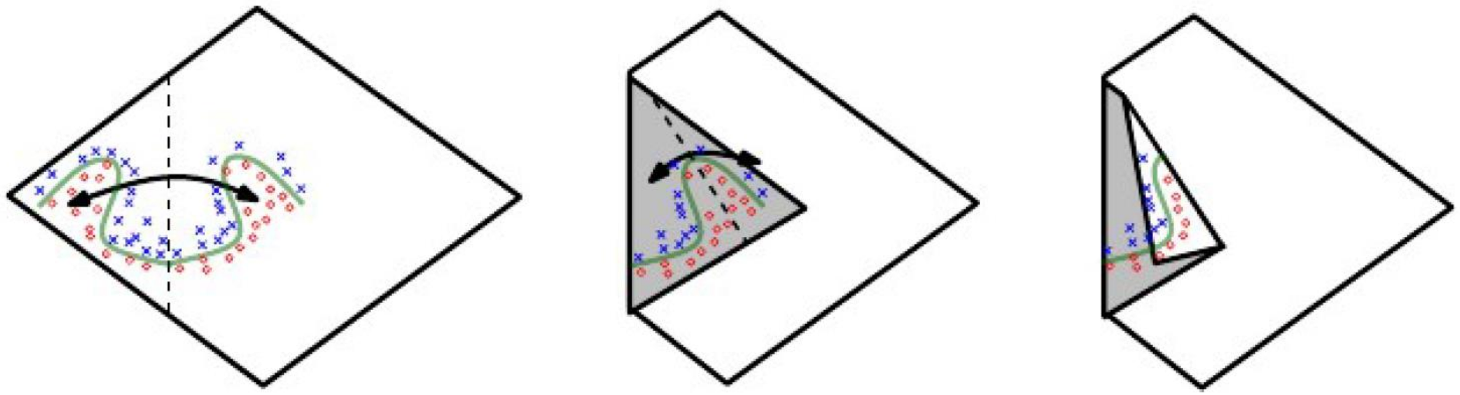


Fig. 6.5

Number of Layers

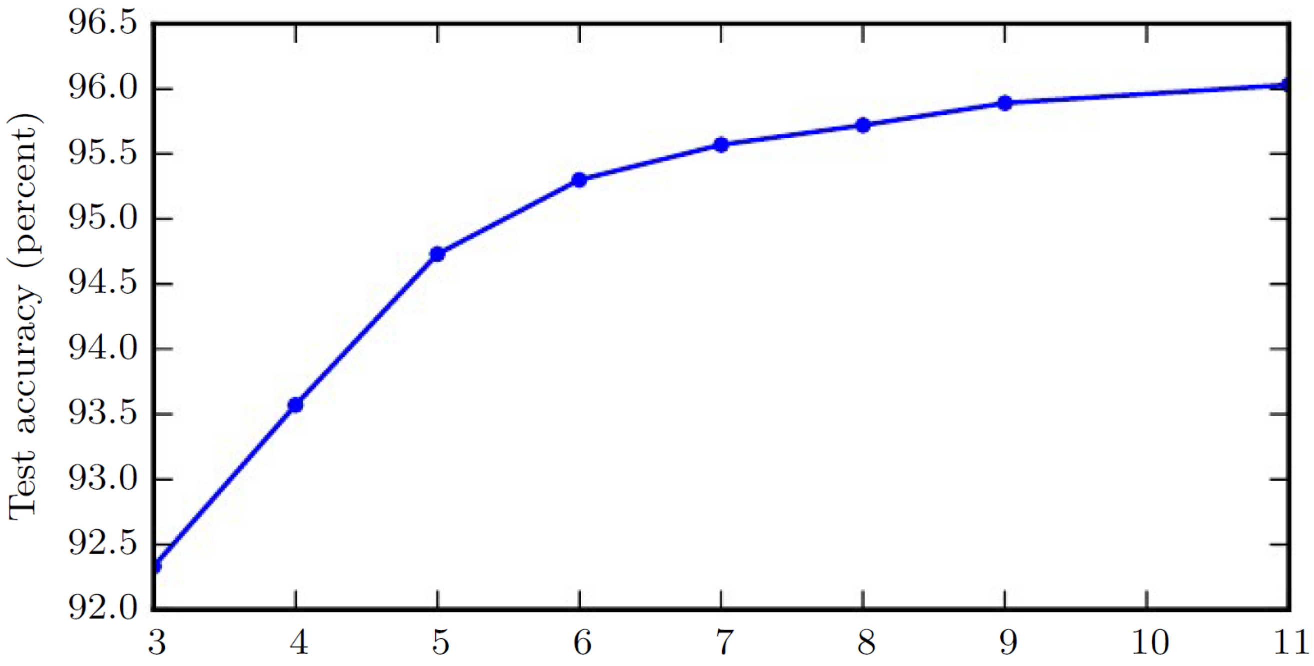


Fig. 6.6

Architectural Considerations

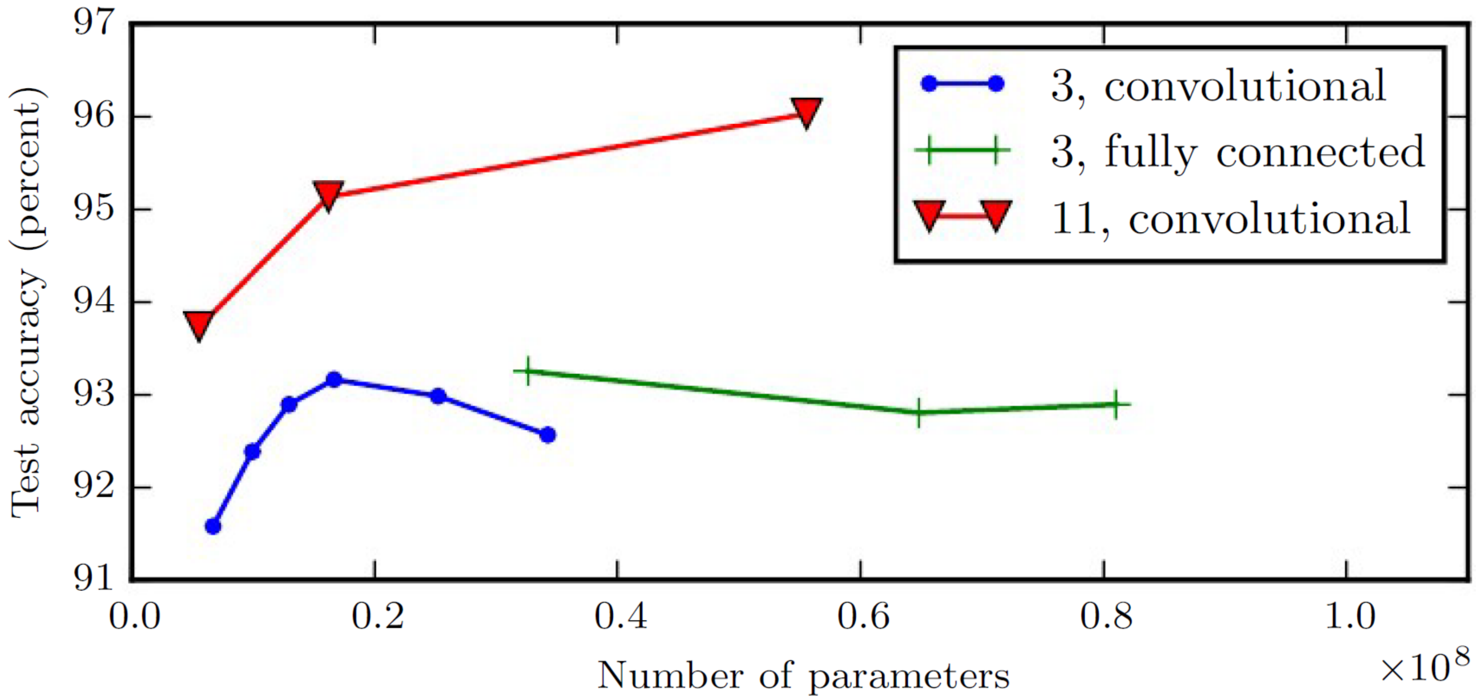


Fig. 6.7

Back-propagation Algorithm

- Forward propagation: $\mathbf{x} \to \hat{\mathbf{y}} \to J(\boldsymbol{\theta})$
- Backward propagation: propagation of gradients in the backward direction for efficient computation of gradients
- Back-propagation algorithm (or backprop) is not a learning algorithm. It is only for computing gradients. Actual learning is done by gradient descent, etc.

Chain Rule of Calculus

- Chain rule of calculus assuming $y = g(x)$ and $z = f(y) = f(g(x))$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

- If $g : \mathbb{R}^m \to \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$, then

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^T \nabla_{\mathbf{y}} z$$

$$\left. \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right|_{(i,j)} = \frac{\partial y_i}{\partial x_j} \quad n \times m \text{ Jacobian matrix of } g$$

- For tensors X and Y, we rearrange them as long vectors

Example 1

- Training set: $(\mathbf{x}, \mathbf{y})$
  - $\mathbf{x}$: $m \times 1$ vector of m inputs for training
  - $\mathbf{y}$: $m \times 1$ vector of m outputs for training
- Assume l layers with 1 neuron in each layer
  - First layer output: $\mathbf{h}_1 = \phi^{(1)}(\mathbf{x}w_1)$ (assume no bias for simplicity) (define also $\mathbf{h}_0 = \mathbf{x}$)
    * w_1: weight and bias of the first layer
    * $\phi^{(1)}(\cdot)$: activation function of the first layer
    * $\mathbf{h}_1$: $m \times 1$ vector containing m outputs of the first layer
  - Second layer output: $\mathbf{h}_2 = \phi^{(2)}(\mathbf{h}_1 w_2)$
    * w_2: weight and bias of the second layer
    * $\phi^{(2)}(\cdot)$: activation function of the second layer
    * $\mathbf{h}_2$: $m \times 1$ vector containing m outputs of the second layer
  - l-th layer output: $\mathbf{h}_l = \phi^{(l)}(\mathbf{h}_{l-1} w_l)$ (output layer)
- Cost

$$J(w_1, w_2, \ldots, w_l) = \frac{1}{2m}\|\mathbf{y} - \mathbf{h}_l\|^2$$

Example 1

- Goal: to calculate gradients $\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \ldots, \frac{\partial J}{\partial w_l}$

- Let's define $\mathbf{u}_k = \mathbf{h}_{k-1} w_k$ and $\mathbf{g}_k = \nabla_{\mathbf{u}_k} J$, $k = 1, \ldots, l$, then $\mathbf{g}_l = \frac{1}{m} \phi^{(l)'}(\mathbf{u}_l) \odot (\mathbf{h}_l - \mathbf{y})$ and $\mathbf{g}_{k-1} = \phi^{(k-1)'}(\mathbf{u}_{k-1}) \odot \mathbf{g}_k w_k$, $k = 2, \ldots, l$ since

$$g_{l,i} = \frac{\partial J}{\partial u_{l,i}} = \frac{\partial}{\partial u_{l,i}} \frac{1}{2m} \|\mathbf{y} - \phi^{(l)}(\mathbf{u}_l)\|^2 = \frac{\partial}{\partial u_{l,i}} \frac{1}{2m} (y_i - \phi^{(l)}(u_{l,i}))^2 = \frac{1}{m} \phi^{(l)'}(u_{l,i})(\phi^{(l)}(u_{l,i}) - y_i)$$

$$g_{l-1,i} = \frac{\partial J}{\partial u_{l-1,i}} = \sum_{i'} \frac{\partial u_{l,i'}}{\partial u_{l-1,i}} \frac{\partial J}{\partial u_{l,i'}} = \phi^{(l-1)'}(u_{l-1,i}) w_l g_{l,i}$$

$$\vdots$$

$$g_{1,i} = \phi^{(1)'}(u_{1,i}) w_2 g_{2,i}$$

- $\frac{\partial J}{\partial w_k} = \mathbf{h}_{k-1}^T \mathbf{g}_k$, $k = 1, \ldots, l$ since

$$\frac{\partial J}{\partial w_k} = \sum_{i'} \frac{\partial u_{k,i'}}{\partial w_k} \frac{\partial J}{\partial u_{k,i'}} = \sum_{i'} h_{k-1,i'} g_{k,i'} = \mathbf{h}_{k-1}^T \mathbf{g}_k$$

Training Algorithm for Example 1

Algorithm 1 Training with forward and backward propagations

function TRAININGALGORITHM(l, α, $\mathbf{x}$, $\mathbf{y}$, $MaxIter$)
    initialize $w[1], w[2], \ldots, w[l]$ randomly
    $\mathbf{h}[0] \leftarrow \mathbf{x}$
    for $iter = 1, 2, \ldots, MaxIter$ **do**
        for $k = 1, 2, \ldots, l$ **do** ▷ Forward propagation
            $\mathbf{u}[k] \leftarrow \mathbf{h}[k-1] * w[k]$
            $\mathbf{h}[k] \leftarrow \phi_k(\mathbf{u}[k])$
        $\mathbf{g}[l] \leftarrow \phi_l'(\mathbf{u}[l]) * (\mathbf{h}[l] - \mathbf{y})/m$
        for $k = l-1, l-2, \ldots, 1$ **do** ▷ Backward propagation
            $\mathbf{g}[k] \leftarrow \phi_k'(\mathbf{u}[k]) * \mathbf{g}[k+1] * w[k]$
        for $k = 1, 2, \ldots, l$ **do** ▷ Gradient descent
            $w[k] \leftarrow w[k] - \alpha \mathbf{h}[k-1]^{\mathsf{T}} * \mathbf{g}[k]$
    return $w[1], w[2], \ldots, w[l]$

Example 2

- Training set: $(\mathbf{X}, \mathbf{y})$
  - $\mathbf{X}$: $m \times n$ matrix of m training examples. The i-th row is the i-th example.
  - $\mathbf{y}$: $m \times 1$ vector of m labels whose i-th element is $y_i \in \{1, 2, \ldots, k\}$, $1 \leq i \leq m$, where k is the number of categories
- Assume 2 layers
  - Hidden layer output: $\mathbf{H} = \phi(\mathbf{X}\mathbf{W}^{(1)})$ (assume no bias for simplicity)
    * $\mathbf{W}^{(1)}$: $n \times n_1$ matrix of weights
    * n_1: number of neurons in the hidden layer
    * $\mathbf{H}$: $m \times n_1$ matrix
    * $\phi(\cdot)$: element-wise activation function
  - Output layer output: $\exp(\mathbf{H}\mathbf{W}^{(2)})$ (assume unnormalized softmax for simplicity)
    * $\mathbf{W}^{(2)}$: $n_1 \times k$ matrix of weights
    * There are k output neurons
- The total cost

$$J = J_{\text{MLE}} + \lambda \left(\sum_{i,j} \left(W_{i,j}^{(1)} \right)^2 + \sum_{i,j} \left(W_{i,j}^{(2)} \right)^2 \right)$$

Example 2

- Cross-entropy term

$$\begin{aligned}
J_{\text{MLE}} &= -\frac{1}{m}\sum_{i=1}^{m} \log p_{\text{model}}(y_i|\mathbf{X}_{i,\cdot};\mathbf{W}^{(1)},\mathbf{W}^{(2)}) \\
&= -\frac{1}{m}\sum_{i=1}^{m} \log \exp(\phi(\mathbf{X}_{i,\cdot}\mathbf{W}^{(1)})\mathbf{W}^{(2)}_{\cdot,y_i}) \\
&= -\frac{1}{m}\sum_{i=1}^{m} \log \exp(\phi(\mathbf{X}_{i,\cdot}\mathbf{W}^{(1)})\mathbf{W}^{(2)}\mathbf{Y}_{\cdot,i}) \\
&= -\frac{1}{m}\sum_{i=1}^{m} \phi(\mathbf{X}_{i,\cdot}\mathbf{W}^{(1)})\mathbf{W}^{(2)}\mathbf{Y}_{\cdot,i} \\
&= -\frac{1}{m}\operatorname{tr}(\phi(\mathbf{X}\mathbf{W}^{(1)})\mathbf{W}^{(2)}\mathbf{Y}) \\
&= -\frac{1}{m}\operatorname{tr}(\mathbf{H}\mathbf{W}^{(2)}\mathbf{Y})
\end{aligned}$$

where $\mathbf{Y}_{j,i} = 1$ if $y_i = j$ and 0 otherwise and $\operatorname{tr}(A)$ is the trace of matrix A.

Example 2

- Goal: to calculate gradients $\nabla_{\mathbf{W}^{(1)}} J$ and $\nabla_{\mathbf{W}^{(2)}} J$
- Note that

$$\nabla_{\mathbf{W}^{(1)}} J = \nabla_{\mathbf{W}^{(1)}} J_{\text{MLE}} + 2\lambda \mathbf{W}^{(1)}$$
$$\nabla_{\mathbf{W}^{(2)}} J = \nabla_{\mathbf{W}^{(2)}} J_{\text{MLE}} + 2\lambda \mathbf{W}^{(2)}$$

because $\frac{\partial \lambda \sum_{i',j'} \left(W_{i',j'}^{(1)}\right)^2}{\partial W_{i,j}^{(1)}} = \sum_{i',j'} 2\lambda W_{i',j'}^{(1)} \delta_{i,i'} \delta_{j,j'} = 2\lambda W_{i,j}^{(1)}$, where $\delta_{i,i'} = 1$ if $i = i'$ and 0 otherwise.

- Let's define $\mathbf{U}^{(2)} = \mathbf{H}\mathbf{W}^{(2)}$, $\mathbf{G}^{(2)} = \nabla_{\mathbf{U}^{(2)}} J_{\text{MLE}}$, i.e., $G_{i,j}^{(2)} = \frac{\partial J_{\text{MLE}}}{\partial U_{i,j}^{(2)}}$
- $\nabla_{\mathbf{W}^{(2)}} J_{\text{MLE}} = \mathbf{H}^T \mathbf{G}^{(2)}$ since

$$(\nabla_{\mathbf{W}^{(2)}} J_{\text{MLE}})|_{i,j} := \frac{\partial J_{\text{MLE}}}{\partial W_{i,j}^{(2)}} = \sum_{i',j'} \frac{\partial U_{i',j'}^{(2)}}{\partial W_{i,j}^{(2)}} \frac{\partial J_{\text{MLE}}}{\partial U_{i',j'}^{(2)}} = \sum_{i',j'} \frac{\partial \sum_k H_{i',k} W_{k,j'}^{(2)}}{\partial W_{i,j}^{(2)}} \frac{\partial J_{\text{MLE}}}{\partial U_{i',j'}^{(2)}}$$
$$= \sum_{i',j'} \sum_k H_{i',k} \delta_{k,i} \delta_{j',j} \frac{\partial J_{\text{MLE}}}{\partial U_{i',j'}^{(2)}} = \sum_{i'} H_{i',i} \frac{\partial J_{\text{MLE}}}{\partial U_{i',j}^{(2)}} = (\mathbf{H}^T \mathbf{G}^{(2)})|_{i,j}$$

Example 2

- Let's define $\mathbf{U}^{(1)} = \mathbf{X}\mathbf{W}^{(1)}$ and $\mathbf{G}^{(1)} = \nabla_{\mathbf{U}^{(1)}} J_{\text{MLE}}$, then we have $\mathbf{G}^{(1)} = \phi'(\mathbf{U}^{(1)}) \odot (\mathbf{G}^{(2)}\mathbf{W}^{(2)T})$ since

$$G^{(1)}_{i,j} = \frac{\partial J_{\text{MLE}}}{\partial U^{(1)}_{i,j}} = \sum_{i',j'} \frac{\partial J_{\text{MLE}}}{\partial U^{(2)}_{i',j'}} \frac{\partial U^{(2)}_{i',j'}}{\partial U^{(1)}_{i,j}} = \sum_{i',j'} \frac{\partial J_{\text{MLE}}}{\partial U^{(2)}_{i',j'}} \frac{\partial}{\partial U^{(1)}_{i,j}} \sum_{k'} \phi(U^{(1)}_{i',k'}) W^{(2)}_{k',j'}$$

$$= \sum_{i',j'} \frac{\partial J_{\text{MLE}}}{\partial U^{(2)}_{i',j'}} \sum_{k'} \phi'(U^{(1)}_{i',k'}) W^{(2)}_{k',j'} \delta_{i,i'} \delta_{j,k'} = \sum_{j'} \frac{\partial J_{\text{MLE}}}{\partial U^{(2)}_{i,j'}} \phi'(U^{(1)}_{i,j}) W^{(2)}_{j,j'} = \phi'(U^{(1)}_{i,j})(\mathbf{G}^{(2)}\mathbf{W}^{(2)T})|_{i,j}$$

- $\nabla_{\mathbf{W}^{(1)}} J_{\text{MLE}} = \mathbf{X}^T \mathbf{G}^{(1)}$ since

$$(\nabla_{\mathbf{W}^{(1)}} J_{\text{MLE}})|_{i,j} := \frac{\partial J_{\text{MLE}}}{\partial W^{(1)}_{i,j}} = \sum_{i',j'} \frac{\partial U^{(1)}_{i',j'}}{\partial W^{(1)}_{i,j}} \frac{\partial J_{\text{MLE}}}{\partial U^{(1)}_{i',j'}} = \sum_{i',j'} \frac{\partial \sum_k X_{i',k} W^{(1)}_{k,j'}}{\partial W^{(1)}_{i,j}} \frac{\partial J_{\text{MLE}}}{\partial U^{(1)}_{i',j'}}$$

$$= \sum_{i',j'} \sum_k X_{i',k} \delta_{k,i} \delta_{j',j} \frac{\partial J_{\text{MLE}}}{\partial U^{(1)}_{i',j'}} = \sum_{i'} X_{i',i} \frac{\partial J_{\text{MLE}}}{\partial U^{(1)}_{i',j}} = (\mathbf{X}^T \mathbf{G}^{(1)})|_{i,j}$$

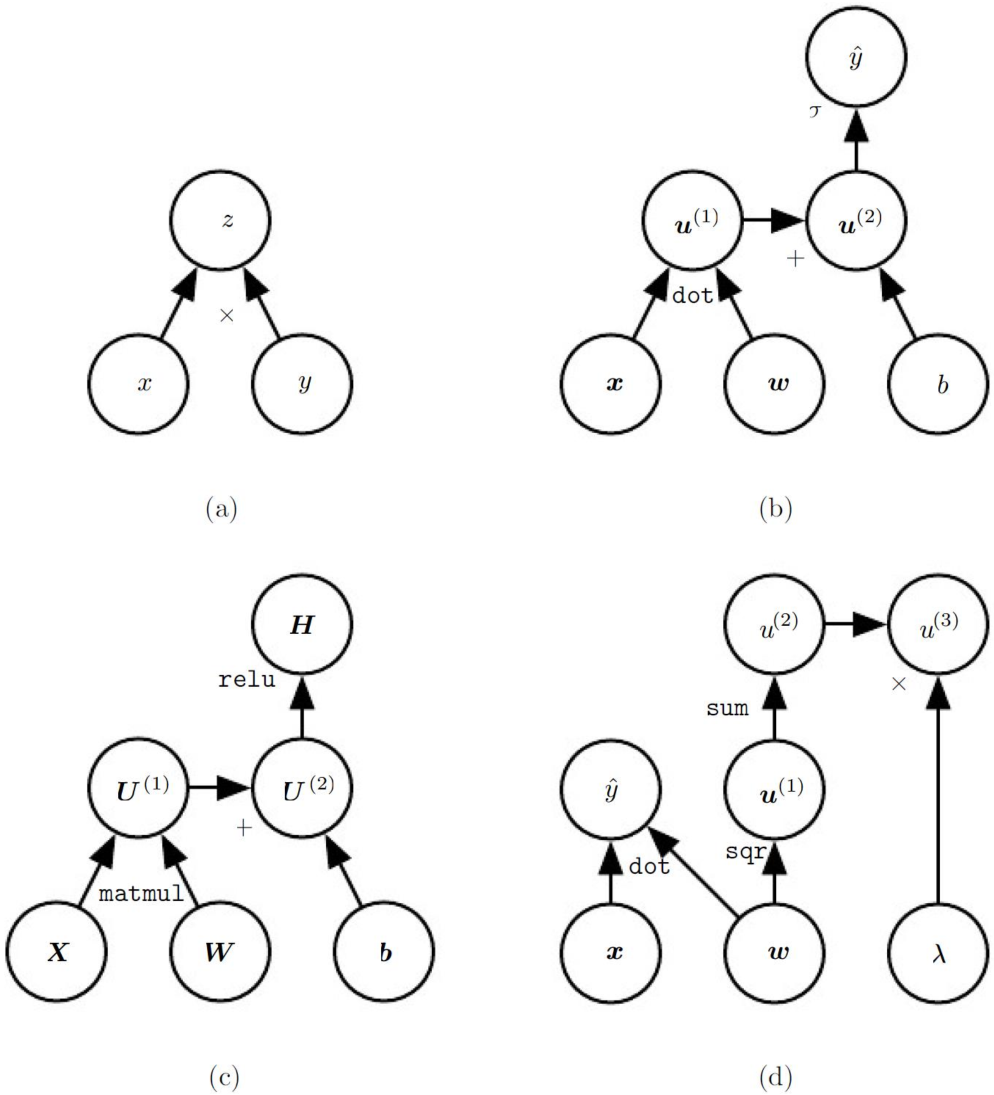


Fig. 6.8: Examples of computational graphs

Example

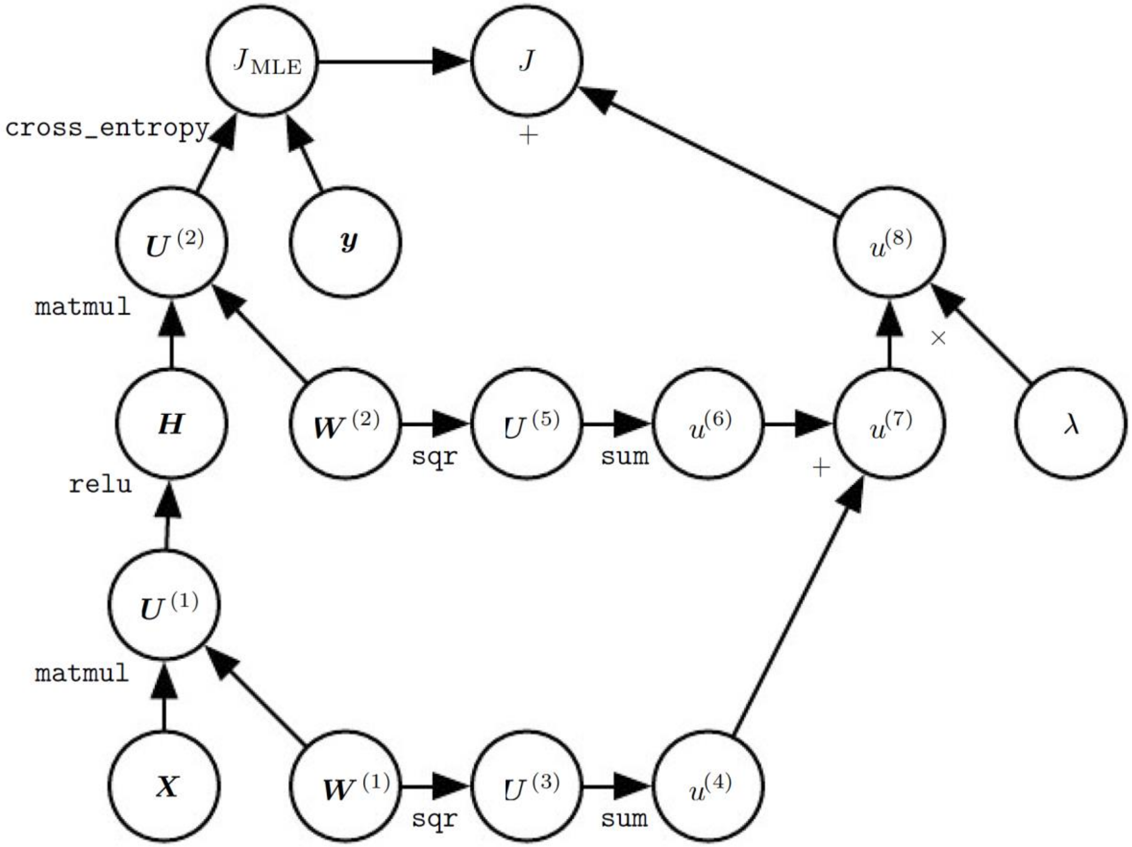


Fig. 6.11

Assignments, Mid-term Exam

- Reading assignment: Chapters 6 & 7 of DL book
- Holidays: October 2, 4, 9 (no class)
- Homework #2
  - To be uploaded on September 27
  - Due: October 11 (Wed) 1pm
  - No late submissions allowed for homeworks since solutions will be uploaded immediately after the deadline
- Mid-term exam
  - October 16 (Mon) 1pm – 3pm
  - Place: classroom