Machine Learning

Lecture 10: Neural Networks and Deep Learning

Feng Li

fli@sdu.edu.cn
https://funglee.github.io

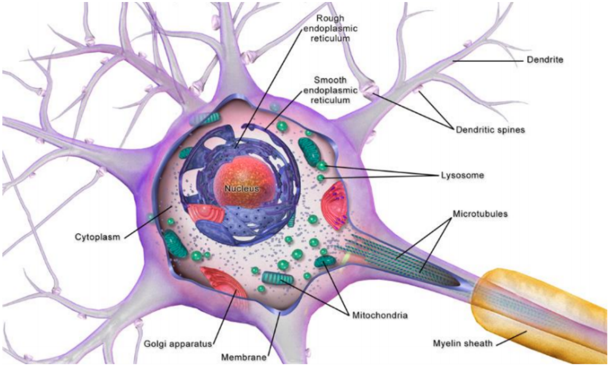
School of Computer Science and Technology
Shandong University

Fall 2018

Deep Feedforward Networks

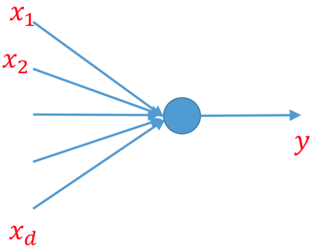
- Also called feedforward neural networks or multilayer perceptrons (MLPs)
- The goal is to approximate some function f^*
  - E.g., for a classifier, $y = f^*(x)$ maps an input x to a category y
- A feedforward network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that result in the best function approximations
- $f(x)$ is usually a highly non-linear function
- Feedforward networks are of extreme importance to machine learning practioners
  - The conventional neural networks (CNN) used for object recognition from photos are a specialized kind of feedforward network
  - It can be extended to recurrent neural networks (RNN) by involving feedback connections, which power many natural language applications

Neuron


Rough endoplasmic reticulum
Dendrite
Smooth endoplasmic reticulum
Dendritic spines
Lysosome
Nucleus
Microtubules
Cytoplasm
Mitochondria
Myelin sheath
Golgi apparatus
Membrane

Neuron (Contd.)

- Neuron activated when the correlation between the input and a pattern θ exceeds some threshold b
- $y = g(\theta^T x - b)$
- $g(\cdot)$ is called activation function
  - Sigmoid: $g(z) = 1/(1 + e^{-z})$
  - ReLU: $g(z) = \max(z, 0)$
  - Tanh: $g(z) = (e^z - e^{-z})/(e^z + e^{-z})$

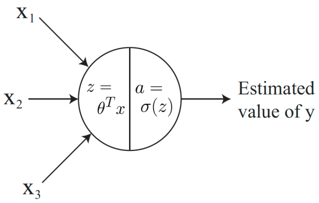


Neuron (Contd.)

- An example: logistic regression function

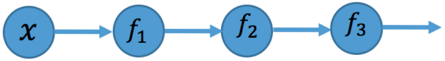
$$g(x) = \frac{1}{1 + \exp(-w^T x - b)}$$

- Break it into two computations
  - $z = w^T x + b$
  - $a = \sigma(z)$ where $\sigma(z) = 1/(1 + e^{-z})$



Neural Feedforward Networks

- In feedforward networks, information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y
- The model is associated with a directed acyclic graph describing how the functions are composed together
    - E.g., we use a chain to represent $f(x) = f_3(f_2(f_1(x)))$

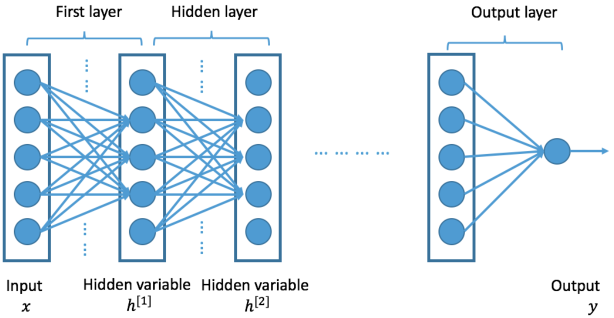


$$\begin{tikzcd} x \arrow[r] & f_1 \arrow[r] & f_2 \arrow[r] & f_3 \arrow[r] & {} \end{tikzcd}$$

    - If we take sigmod function as the activation function
        - $z_1 = w_1 x + b_1$ and $a_1 = \sigma(z_1)$
        - $z_2 = w_2 a_1 + b_2$ and $a_2 = \sigma(z_2)$
        - $z_3 = w_3 a_2 + b_3$ and $a_3 = \sigma(z_3)$

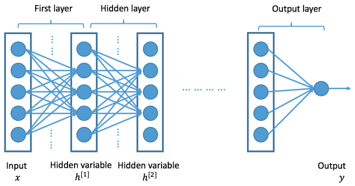
Neural Feedforward Networks (Contd.)

- The architecture of feedforward neural networks
  - Input layer, hidden layers (consisting of hidden units), and output layer



Neural Feedforward Networks (Contd.)

- We approximate $f^*(x)$ by learning $f(x)$ from the given training data
    - In the output layer, $f(x) \approx y$ for each training data, but the behavior of the other layers is not directly specified by the training data
    - Learning algorithm must decided how to use those intermediate layers such that right results can be obtained in the output layer, but the training data do not say what each individual layer should do
    - The only thing we must provide to the neural network is a suffcient number of training examples $(x^{(i)}, y^{(i)})$
    - It can be diffcult to understand the features a neural network has invented; therefore, people refer to neural networks as a black box



Gradient Descent (GD) Algorithm

- If the multi-variable cost (or loss) function $\mathcal{L}(\theta)$ is differentiable in a neighborhood of a point θ, then $\mathcal{L}(\theta)$ decreases fastest if one goes from θ in the direction of the negative gradient of $\mathcal{L}$ at θ
- Find a local minimum of a differentiable function using gradient descent

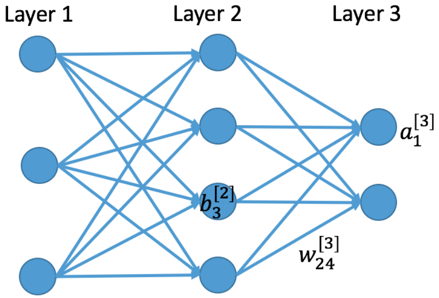
$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial \mathcal{L}(\theta)}{\partial \theta_j}, \quad \forall j$$

  where α is so-called learning rate
- Variations
  - Gradient ascent algorithm
  - Stochastic gradient descent/ascent
  - mini-batch gradient descent/ascent

Back-Propagation: Warm Up

- $w_{jk}^{[l]}$ is the weight from the k-th neuron in the $(l-1)$-th layer to the j-th neuron in the l-th layer
- $b_j^{[l]}$ is the bias of the j-th neuron in the l-th layer
- $a_j^{[l]}$ is the activation of the j-th neuron in the l-th layer

$$a_j^{[l]} = \sigma\left(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}\right)$$



Back-Propagation: Warm Up

- Vectorization

$$a^{[l]} = \sigma(w^{[l]} a^{[l-1]} + b^{[l]})$$

  where $\sigma(\cdot)$ is an element-wise function such that $\sigma(v)_j = \sigma(v_j)$
- Introduce an intermediate variable

$$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$

- We thus have

$$a^{[l]} = \sigma(w^{[l]} a^{[l-1]} + b^{[l]}) = \sigma(z^{[l]})$$

  - $z^{[l]}$ is the weighted input to the neurons in layer l
  - $z_j^{[l]} = \sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}$ is the weighted input to the activation function for neuron j in layer l

Back-Propagation: Warm Up

- Assumptions on the cost function
  - The loss function can be written as the average over the loss functions from individual training samples

$$\mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}_i$$

    - Backpropagation computes the gradients with respect to only one single training sample as given by $\partial \mathcal{L}_i / \partial w$ and $\partial \mathcal{L}_i / \partial b$
    - We then calculate $\partial \mathcal{L} / \partial w$ and $\partial \mathcal{L} / \partial b$ by averaging the gradients cross different training samples
  - The loss function can be written as a function of the output from the neural network
- Hadamard product: Elementwise product of two vectors $s \odot t$ such that $(s \odot t)_j = s_j t_j$

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 * 3 \\ 2 * 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

Fundamental Equations

- The error in the j-th neuron in the l-th layer:

$$\delta_j^{[l]} = \frac{\partial \mathcal{L}}{\partial z_j^{[l]}}$$

- An equation for the error in the output layer

$$\delta_j^{[L]} = \frac{\partial \mathcal{L}}{\partial a_j^{[L]}} \sigma'(z_j^{[L]})$$

  - $\partial \mathcal{L} / \partial a_j^{[L]}$ measures how fast the loss is changing as a function of the j-th output activation
  - $\sigma'(z_j^{[L]})$ measures how fast the activation function σ is changing at z_j^L
- A vectorized form

$$\delta^{[L]} = \bigtriangledown_a \mathcal{L} \odot \sigma'(z^{[L]})$$

Fundamental Equations

- An equation for the error δ^l in terms of the error in the next layer δ^{l+1}

$$\delta^{[l]} = ((w^{[l+1]})^T \delta^{[l+1]}) \odot \sigma'(z^{[l]})$$

- $(w^{[l+1]})^T \delta^{[l+1]}$ is to move the error backward through the networks, giving us some sort of measure of the error at the output of the l-th layer
- $((w^{[l+1]})^T \delta^{[l+1]}) \odot \sigma'(z^{[l]})$ is to move the error backward through the activation function in the layer l, giving us the error δ^l in the weight input to layer l
- So far, we can compute $\delta^{[l]}$ for any layer in the network

Fundamental Equations

- Proof: Rewrite $\delta_j^{[l]} = \partial\mathcal{L}/\partial z_j^{[l]}$ in terms of $\delta_k^{[l]} = \partial\mathcal{L}/\partial z_k^{[l+1]}$
  - By the chain rule,

$$\delta_j^{[l]} = \frac{\partial\mathcal{L}}{\partial z_j^{[l]}} = \sum_k \frac{\partial\mathcal{L}}{\partial z_k^{[l+1]}} \frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = \sum_k \frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} \delta_k^{[l+1]}$$

  - Since

$$z_k^{[l+1]} = \sum_j w_{kj}^{[l+1]} a_j^{[l]} + b_k^{[l+1]} = \sum_j w_{kj}^{[l+1]} \sigma(z_j^{[l]}) + b_k^{[l+1]} \ ,$$

    we have

$$\frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = w_{kj}^{[l+1]} \sigma'(z_j^{[l]})$$

  - Hence,

$$\delta_j^{[l]} = \sum_k w_{kj}^{[l+1]} \delta_k^{[l+1]} \sigma'(z_j^{[l]})$$

Fundamental Equations

- An equation for the rate of change of the loss with respect to any bias in the network

$$\frac{\partial \mathcal{L}}{\partial b_j^{[l]}} = \delta_j^{[l]} \Rightarrow \frac{\partial C}{\partial b} = \delta$$

- Proof

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b_j^{[l]}} &= \frac{\partial \mathcal{L}}{\partial z_j^{[l]}} \cdot \frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} \\
&= \delta_j^{[l]} \cdot \frac{\partial}{\partial b_j^{[l]}} \left(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]} \right) \\
&= \delta_j^{[l]}
\end{aligned}$$

Fundamental Equations

- An equation for the rate of change of the loss with respect to any weight in the network

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^{[l]}} = a_k^{[l-1]}\delta_j^{[l]}$$

- Proof

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_{jk}^{[l]}} &= \frac{\partial \mathcal{L}}{\partial z_j^{[l]}} \cdot \frac{\partial z_j^{[l]}}{\partial w_{jk}^{[l]}} \\
&= \delta_j^{[l]} \cdot \frac{\partial}{\partial w_{jk}^{[l]}} \left(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]} \right) \\
&= a_k^{[l-1]}\delta_j^{[l]}
\end{aligned}$$

Backpropagation Algorithm

- The backpropagation equations provides us with a way of computing the gradient of the cost function
    - Input: Set the corresponding activation $a^{[1]}$ for the input layer
    - Feedforward: For each $l = 2, 3, \cdots, L$, compute $z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}$ and $a^{[l]} = \sigma(z^{[l]})$
    - Output error $\delta^{[L]} = \bigtriangledown_a \mathcal{L} \odot \sigma'(z^{[L]})$
    - Backpropagate the error: For each $l = L-1, L-2, \cdots, 2$, compute $\delta^{[l]} = ((w^{[l+1]})^T \delta^{[l+1]}) \odot \sigma'(z^{[l]})$
    - Output: The gradient of the loss function is calculated by $\partial\mathcal{L}/\partial w_{jk}^{[l]} = a_k^{[l-1]}\delta_j^{[l]}$ and $\partial\mathcal{L}/\partial b_j^{[l]} = \delta_j^{[l]}$
- BP algorithm is usually combined with stochastic gradient descent algorithm or mini-batch gradient descent algorithm

Thanks!

Q & A