

Deep learning

Feed-forward deep networks

Hamid Beigy

Sharif university of technology

October 9, 2019

Table of contents

- 1 Introduction
- 2 Training feed-forward networks
- 3 Error back-propagation algorithm
- 4 Network configuration
- 5 Reading



Table of contents

1 Introduction

2 Training feed-forward networks

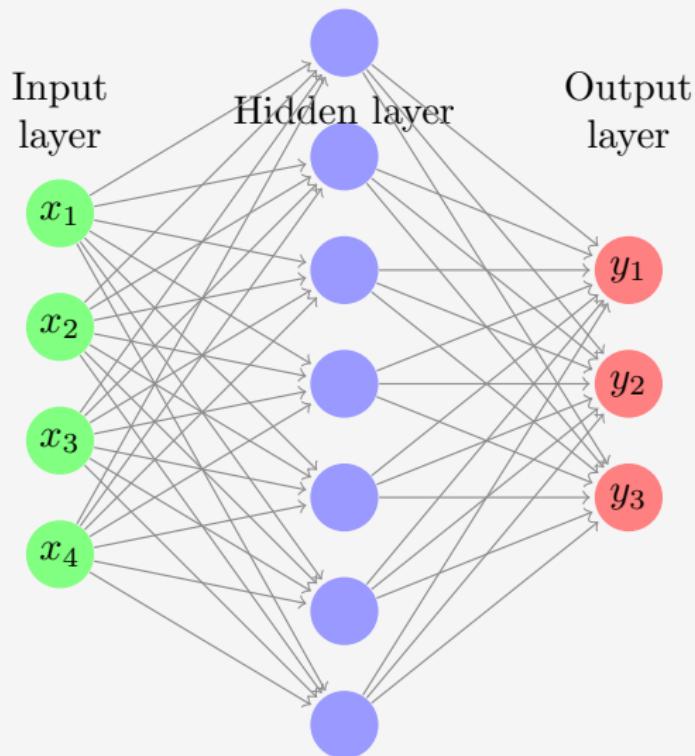
3 Error back-propagation algorithm

4 Network configuration

5 Reading

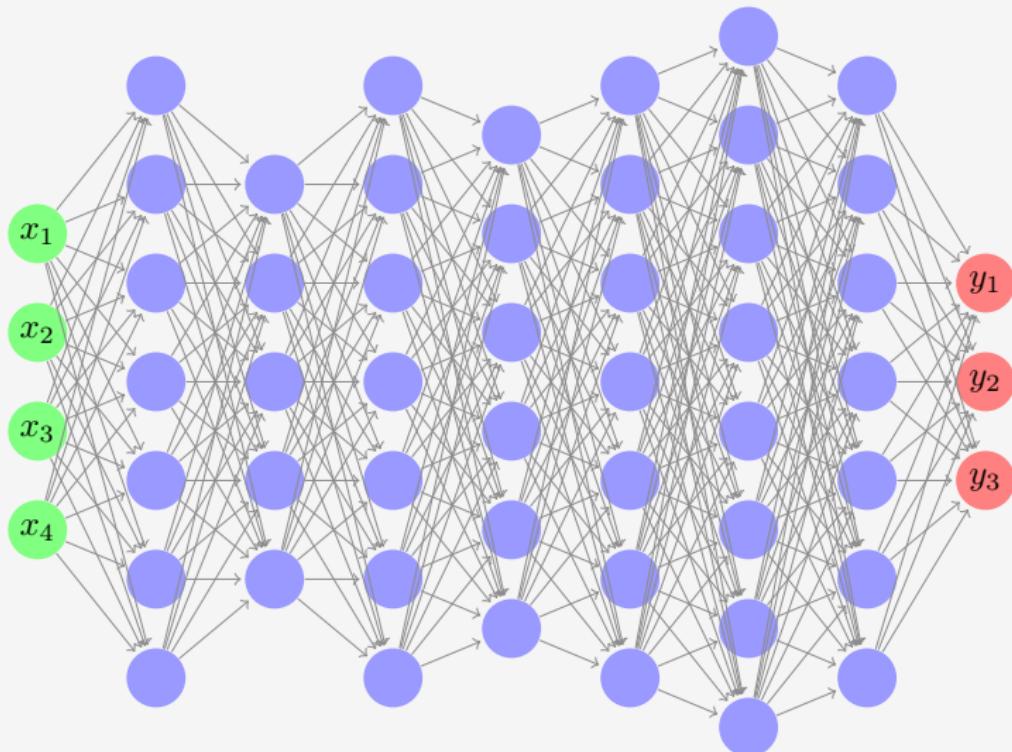


Deep feed-forward networks





Deep feed-forward networks





The optimal topology of the networks

- 1 What is **the topology of network** for the given problem?
- 2 Can we build a network to create every decision boundary?
- 3 Neural networks are **universal approximators**.
- 4 Can we build a network without local minimia in cost function?



Table of contents

1 Introduction

2 Training feed-forward networks

3 Error back-propagation algorithm

4 Network configuration

5 Reading



Training feed-forward networks

- 1 Specifying **the topology of network** and **the cost function**
 - #-layers
 - #-nodes in each layer
 - function of each node
 - activation of each node
- 2 We use **gradient decent algorithm** for training the network.
- 3 **But, we don't have the true output of each hidden unit.**
- 4 We back-propagate error from the output layer.



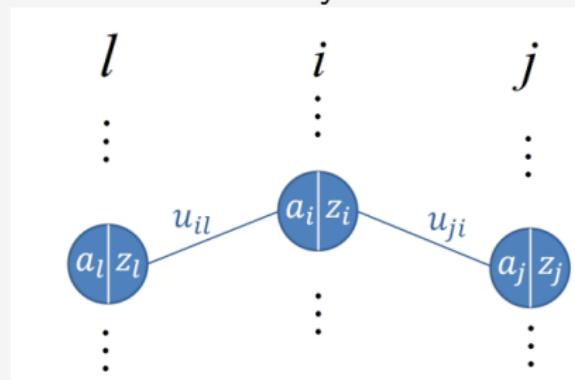
Table of contents

- 1 Introduction
- 2 Training feed-forward networks
- 3 Error back-propagation algorithm
- 4 Network configuration
- 5 Reading



Error back-propagation algorithm

1 Consider nodes from three hidden layers



2 where

$$a_i = \sum_l z_l u_{il}$$

$$z_i = \sigma(a_i)$$

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$



Error back-propagation algorithm

- 1 Take the derivative of error with respect to weight u_{il} , i.e. we need

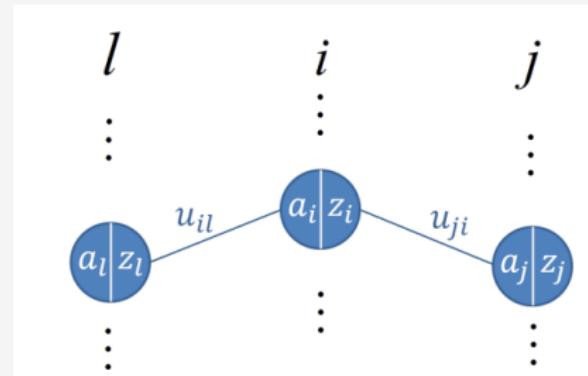
$$\frac{\partial(y - \hat{y})^2}{\partial u_{il}}$$

- 2 This equals to

$$\frac{\partial(y - \hat{y})^2}{\partial u_{il}} = \delta_i \times z_l$$

where

$$\delta_i = \frac{\partial(y - \hat{y})^2}{\partial a_i}$$





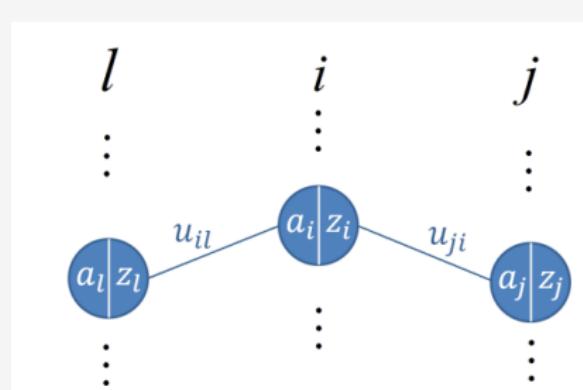
Error back-propagation algorithm

1 We need

$$\begin{aligned}\delta_i &= \frac{\partial(y - \hat{y})^2}{\partial a_i} \\ &= \sum_j \frac{\partial(y - \hat{y})^2}{\partial a_j} \times \frac{\partial a_j}{\partial a_i}\end{aligned}$$

where

$$\begin{aligned}\delta_i &= \sum_j \delta_j \times \frac{\partial a_j}{\partial z_i} \times \frac{\partial z_i}{\partial a_i} \\ &= \sum_j \delta_j \times u_{ji} \times \sigma'(a_i) \\ \delta_j &= \frac{\partial(y - \hat{y})^2}{\partial a_j}\end{aligned}$$





Error back-propagation algorithm

- 1 Assume that nodes use sigmoid activation function
- 2 We have

$$\sigma'(a) = \sigma(a)(1 - \sigma(a))$$

- 3 Recursive definition of δ_i equals to

$$\delta_i = \sigma'(a_i) \sum_j \delta_j u_{ji}$$

- 4 Now consider δ_k for output layer

$$\delta_k = \frac{\partial(y - \hat{y})^2}{\partial a_k}$$

where $a_k = \hat{y}$.

- 5 Assume linear activation function for output layer

$$\delta_k = \frac{\partial(y - \hat{y})^2}{\partial \hat{y}} = -2(y - \hat{y}).$$



Error back-propagation algorithm

- 1 After computing gradient of cost function with respect to a weight, we can update the weight using

$$u_{il} = u_{il} - \eta \frac{\partial(y - \hat{y})^2}{\partial u_{il}}$$

- 2 The weights are updated using error-back-propagation algorithm when the input sample x is fed to the network.

$$\delta_k = \frac{\partial(y - \hat{y})^2}{\partial a_k}$$



Error back-propagation algorithm

- 1 Back-propagation algorithm is done using the following steps
 - 1 Choose arbitrary random weights for the network (it is better to be close to zero).
 - 2 Apply x to the input layer and calculate the output of the input layer.
 - 3 Propagate the output of each hidden layer, one layer per time and calculate their outputs.
 - 4 When the output of the output layer is calculated, we calculate $\delta_k = -2(y_k - \hat{y}_k)$ for each output layer.
 - 5 Compute each δ_i , starting from $i = k - 1$ to the first hidden layer, where $\delta_i = \sigma'(a_i) \sum_j \delta_j u_{ji}$
 - 6 Compute $\frac{\partial(y - \hat{y})^2}{\partial u_{il}} = \delta_i z_l$ for all weights u_{il}
 - 7 Updates all weights u_{il} using the following rule
$$u_{il}^{(t+1)} = u_{il}^{(t)} - \eta \frac{\partial(y - \hat{y})^2}{\partial u_{il}}$$
- 8 Give the next input sample and follow the above steps until weights converged.



Table of contents

- 1 Introduction
- 2 Training feed-forward networks
- 3 Error back-propagation algorithm
- 4 Network configuration
- 5 Reading



Choosing the network configuration

- 1 Given a sample $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$, how choose the network configuration
 - 1 Choosing the number of layers
 - 2 Choosing the number of nodes in each layer
 - 3 Choosing the activation function for each nodes/ each layer
 - 4 Choosing the cost function
 - 5 Training



Learning conditional distributions

- 1 In some cases, the network defines a distribution $P(y|x; \theta)$.
- 2 We use the principle of maximum likelihood and the cost function is defined as

$$J(\theta) = -E_{(x,y) \sim D} \log P_{model}(y|x)$$

- 3 The specific form of the cost function changes from a model to a model, depending on the specific form of $\log P_{model}(y|x)$.
- 4 What is the activation function when $y \in \{0, 1\}$?
- 5 What is the activation function when $y \in \{0, 1, \dots, C\}$?



Learning conditional distributions

- 1 Instead of learning a full probability distribution $P(y|x; \theta)$, we often want to learn just one conditional statistic of y given x .
- 2 For example, we may have a predictor $f(x; \theta)$ that we wish to employ to predict the mean of y .
- 3 The cost function is defined as

$$J(f) = -E_{(x,y) \sim D} \|y - f(x)\|^2$$

- 4 Minimizing the above cost function yields,

$$f^*(x) = -E_{y \sim D(y|x)} [y]$$

- 5 Unfortunately, mean squared error often lead to poor results when used with gradient-based optimization.
- 6 Some output units that saturate produce very small gradients when combined with these cost functions.
- 7 This is one reason that the cross-entropy cost function is more popular than mean squared error.



Choosing activation functions

1 Output layer

- Linear activation function
- ReLU activation function
- Sigmoid activation function
- Softmax activation function

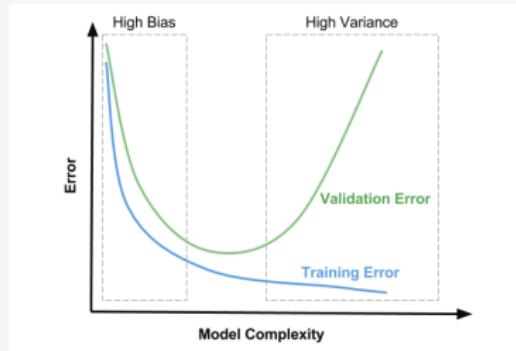
2 Hidden layers

- Linear activation function
- ReLU activation function
- Sigmoid activation function



Choosing the network topology

- 1 A simple approach for choosing the network topology is by trial and error.
- 2 We partition the available data into three parts : **training data**, **validation data**, and **test data**.
- 3 We choose a topology and train the network using the **training data**.
- 4 After training, we evaluate the trained network using **validation data**.





Stopping the training of the network

- 1 A simple approach for stopping criteria is to partition the available data into three parts : **training data**, **validation data**, and **test data**.
- 2 We train the network using the **training data** and in some epochs, we evaluate the trained network using **validation data**.

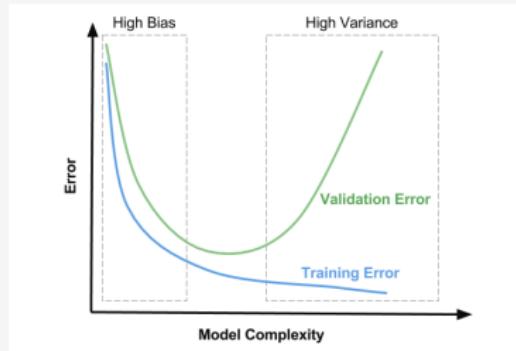




Table of contents

- 1 Introduction
- 2 Training feed-forward networks
- 3 Error back-propagation algorithm
- 4 Network configuration
- 5 Reading



Reading

Please read chapter 6 of Deep Learning Book.