ECE 449/590 – OOP and Machine Learning
Lecture 16 Deep Feedforward Networks

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

October 19, 2022

Outline

Reading Assignment

- This lecture: Deep Learning 6
- Next lecture: Deep Learning 9

Outline

Deep Feedforward Networks

- A.k.a. feedforward neural networks, or multilayer perceptrons (MLPs).
- To approximate a classifier $y = f^*(\boldsymbol{x})$ as a parametric vector-valued function $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$.
- Feedforward: no feedback in f (DAG, combinational) when calculating $\boldsymbol{y}$ from $\boldsymbol{x}$.
- Networks: composition of multiple layers (depth) of functions
  - E.g. $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$ where $f^{(1)}$ is the first layer, $f^{(2)}$ is the second layer, etc.
  - Output layer: the final layer generating the desired output.
  - Hidden layers: the rest of layers whose outputs are decided by the learning algorithm from the training set.
- Neural: functions are loosely inspired by neuroscience.

Learning XOR

- $\boldsymbol{x} = (x_1, x_2)^\top$ where both x_1 and x_2 are boolean.
- Approximate $f^*(\boldsymbol{x}) = x_1$ XOR x_2 with $f(\boldsymbol{x}; \boldsymbol{\theta})$.
- Consider the MSE loss function:

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

- With a single layer: $f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{w}^\top \boldsymbol{x} + b$ for $\boldsymbol{\theta} = (\boldsymbol{w}, b)$.
  - Minimizing $J(\boldsymbol{\theta})$ gives $\boldsymbol{w} = 0$ and $b = \frac{1}{2}$
  - Not a good solution as f is $\frac{1}{2}$ everywhere.

Learning XOR with a Hidden Layer

Network Diagrams

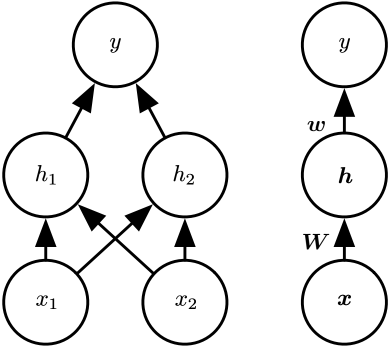


Figure 6.2

(Goodfellow 2017)

Learning XOR with a Hidden Layer

- How about using a hidden layer?
  - (Vector-valued) $\boldsymbol{h} = f^{(1)}(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c})$, and $y = f^{(2)}(\boldsymbol{h}; \boldsymbol{w}, b)$
  - Now $\boldsymbol{\theta} = (\boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b)$ and $f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = f^{(2)}(f^{(1)}(\boldsymbol{x}))$
- If $f^{(1)}$ is a linear function of $\boldsymbol{x}$, then f remains a linear function of $\boldsymbol{x}$.
  - E.g. if $\boldsymbol{h} = \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}$ and $y = \boldsymbol{w}^\top \boldsymbol{h} + b$, then $y = \boldsymbol{w}^\top(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}) + b = (\boldsymbol{w}^\top \boldsymbol{W}^\top)\boldsymbol{x} + (\boldsymbol{w}^\top \boldsymbol{c} + b)$.
  - $f^{(1)}$ need to be nonlinear.
- Activation function g: a fixed nonlinear function
  - $\boldsymbol{h} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c})$: apply g element-wise to outputs from a linear function to convert it into a nonlinear function.

ReLU (Rectified Linear Unit)

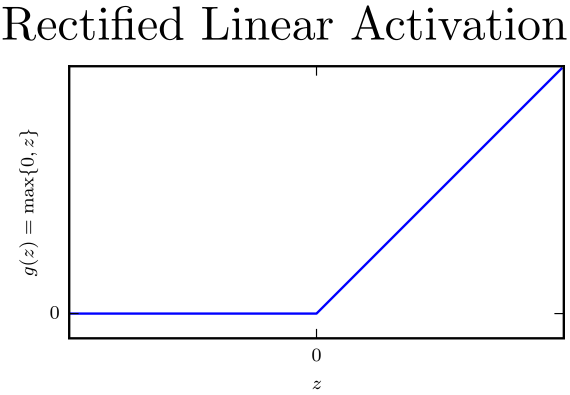


Figure 6.3

(Goodfellow 2017)

- $g(z) = \max\{0, z\}$

Solving XOR

$$f(\boldsymbol{x};\boldsymbol{W},\boldsymbol{c},\boldsymbol{w},b)=\boldsymbol{w}^\top \max\{0,\boldsymbol{W}^\top\boldsymbol{x}+\boldsymbol{c}\}+b. \tag{6.3}$$

$$\boldsymbol{W}=\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \tag{6.4}$$

$$\boldsymbol{c}=\begin{bmatrix} 0 \\ -1 \end{bmatrix}, \tag{6.5}$$

$$\boldsymbol{w}=\begin{bmatrix} 1 \\ -2 \end{bmatrix}, \tag{6.6}$$

(Goodfellow 2017)

Solving XOR

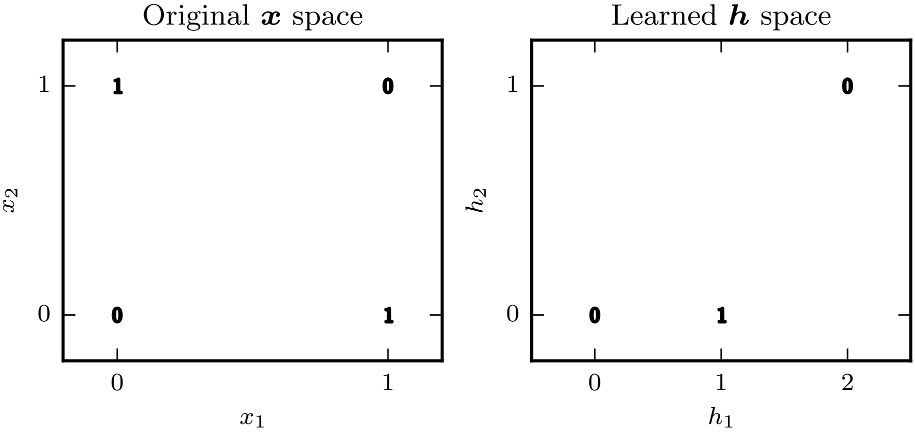


Figure 6.1

(Goodfellow 2017)

Outline

Deep Feedforward Networks

Cost Functions

Architecture Design

Gradient-Based Learning

- Like other machine learning algorithms, training neural network models requires to solve optimization problems.
  - Maximize the performance measure P.
  - Or minimize the loss function.
- For linear models, the optimization problems are usually to convex.
  - A loss function will have the global minimum.
  - Many times a closed-form solution can be derived.
- With neural network models, the loss functions usually have many local minimals.
  - Hard to find the global minimum.
  - Surprisingly, empirical evidences show it is not necessary to locate the global minimums for neural network models to perform well.
  - With proper choice of loss functions and models, use gradient descent for optimization.

Output Units and Classification Problems

- Conditional cross-entropy is widely used as the loss function for supervised learning.

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

.

- For a classification problem with m classes, p_{model} should be a vector-valued function whose output dimension is m.
  - Each dimension corresponds to a class y with value from $[0,1]$.
  - All values add up to 1.
- With neural network models, such requirements only apply to the output layer.
  - Hidden layers can output arbitrarily large or small values.
- Output units: maps output from the last hidden layer $\boldsymbol{h} = f(\boldsymbol{x};\boldsymbol{\theta})$ into the distribution p_{model}.
  - Usually a fixed vector-valued function applied to a linear transformation $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$ of $\boldsymbol{h}$.

Sigmoid Units for Bernoulli Output Distributions

- There are $m = 2$ classes and it is OK to just output $p_{model}(y = 1|\boldsymbol{x})$ within $[0, 1]$.
  - $z = \boldsymbol{w}^\top \boldsymbol{h} + b$
- Choice 1: $p_{model}(y = 1|\boldsymbol{x}) = \max\{0, \min\{1, z\}\}$
  - Does not work well for gradient descent for training.
- Choice 2: $p_{model}(y = 1|\boldsymbol{x}) = \text{sigmoid}(z) = \frac{e^z}{e^z+1}$
  - Intuitively, a weight of 0 is assigned to the case $y = 0$ and a weight of z is assigned to the case $y = 1$.
  - If $z < 0$ then $y = 0$ is more probable, and $y = 1$ is more probable if $z > 0$.
  - Quantitatively, the ratio of $P(y = 0)$ to $P(y = 1)$ is e^0 to e^z.
  - Works well with gradient descent for training.

Softmax Units for Multinoulli Output Distributions

- For $m > 2$ classes, let them be $1, 2, \ldots, m$.
  - It is convenient for $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$ to have a dimension of m.
- Softmax: $p_{model}(y = i|\boldsymbol{x}) = \text{softmax}(\boldsymbol{z})_i$ where

$$\text{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^m e^{z_j}}$$

.

  - The ratio of $P(y = 1)$ to $P(y = 2)$ to $\ldots$ $P(y = m)$ is e^{z_1} to e^{z_2} to $\ldots$ e^{z_m}.
  - For prediction, there is no need to compute softmax as

$$\arg\max_y \text{softmax}(\boldsymbol{z})_y = \arg\max_y z_y$$

.

- For the cross-entropy loss function,

$$-\log p_{model}(y|\boldsymbol{x};\boldsymbol{\theta}) = -\log \text{softmax}(\boldsymbol{z}(\boldsymbol{x};\boldsymbol{\theta}))_y = -z_y + \log \sum_{j=1}^m e^{z_j}$$

.

  - Intuitively, one need to increase z_y relatively to the rest elements of $\boldsymbol{z}$ in order to minimize the loss function.

Outline

Deep Feedforward Networks

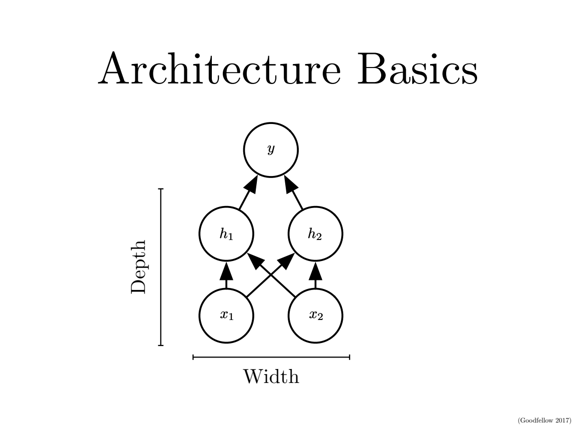
Cost Functions

Architecture Design

Neural Networks as Chained Layers

- Multiple layers of neural networks are usually arranged into a chain structure.
  - First layer: $\boldsymbol{h}^{(1)} = g^{(1)}(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)})$.
  - Second layer: $\boldsymbol{h}^{(2)} = g^{(2)}(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)})$.
  - etc.
- Can such neural network approximate any function?
- Universal approximation theorem: Yes for most functions we are interested into.
  - Even with a single hidden layer and proper activation function.
  - But whether such approximation can be found by training is a different question.

Hyperparameters



- Number of layers.
- Width of each layer, i.e. dimension of $\boldsymbol{h}^{(1)}$, $\boldsymbol{h}^{(2)}$, etc.

Why deeper?

- Shallower neural networks may need (exponentially) more width.
  - A nice analogue is the SOP/POS forms vs multi-level logics for digital designs.
- More width leads to more weights to be learned – larger model capacity.
- Shallower neural networks tend to overfit more!

Better Generalization with Greater Depth

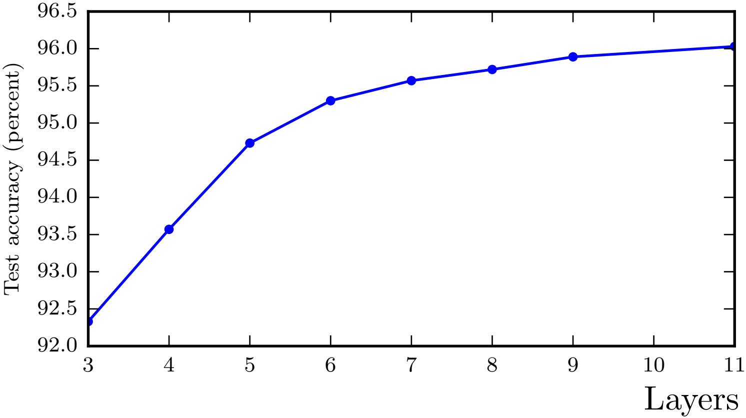


Figure 6.6

(Goodfellow 2017)

Large, Shallow Models Overfit More

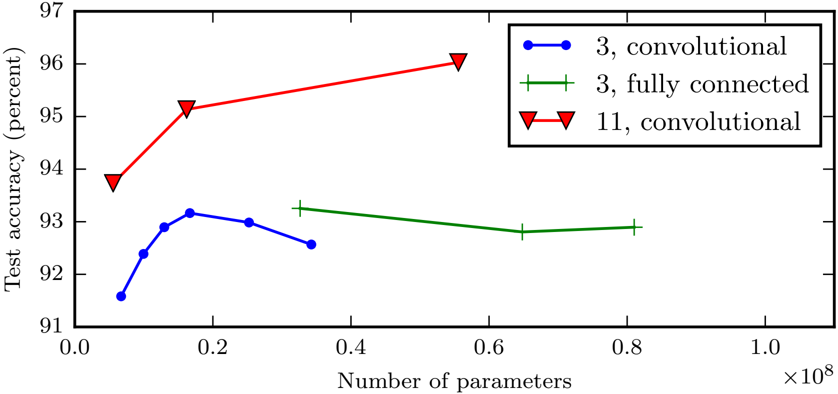


Figure 6.7

(Goodfellow 2017)

Summary

- Feedforward neural networks consists of multiple layers of computations without feedbacks.
- Activation functions, e.g. ReLU, are used to introduce nonlinear behaviors after each linear transformation.
- Output units, e.g. softmax, are used to map arbitrary values into probability distributions so that cross-entropy can be used as a loss function.
- Deeper networks tend to overfit less.
  - But could be harder to train for other reasons.