Deep Learning

Chapter 6

Deep Feedforward Networks

Plan

- Introduction
- An Example: XOR
- Gradient Based Learning

Introduction

- Goal of the network is to approximate f*, for some function f*.

- Feedforward Network because no feedback connection from output.

- Network with multiple Layers.

- Depth is the number of layers.

Training Neural Network

- f* (x) - output for input
- We try to match f(x) to f* (x)
- Training Data consists of nosy, approximate examples of f* (x)
- x accompanied with y ≈ f* (x)

Why Feedforward Network??

- Linear Models are appealing as they can be fit efficiently

- But model capacity is limited to linear functions

- To extend linear models to represent nonlinear functions of x, we can apply the linear model not to x itself but to a transformed input $\varphi(x)$, where φ is a nonlinear transformation

- We can think of φ as providing a set of features describing x

How to choose the mapping φ??

- Option 1: use a very generic ϕ, such as the infinite-dimensional ϕ that is implicitly used by kernel machines based on the RBF kernel

- Problem: Generalization to the test set often remains poor

How to choose the mapping φ??

- Option 2: manually engineer ϕ

- Problem: this approach requires decades of human effort for each separate task .

How to choose the mapping φ??

- Option 2: Deep Learning

- In this case ϕ is represented in hidden layer.

An Example: XOR

- The XOR function ("exclusive or") is an operation on two binary values, x_1 and x_2. When exactly one of these binary values is equal to 1, the XOR function returns 1. Otherwise, it returns 0

- The XOR function provides the target function $y = f^*(x)$ that we want to learn. Our model provides a function $y = f(x;\theta)$ and our learning algorithm will adapt the parameters θ to make f as similar as possible to f^*.

- We want our network to perform correctly on the four points

$$\mathbb{X} = \{[0,0]^\top, [0,1]^\top, [1,0]^\top, \text{and } [1,1]^\top\}$$

Learning XOR

Evaluated on our whole training set, the MSE loss function is

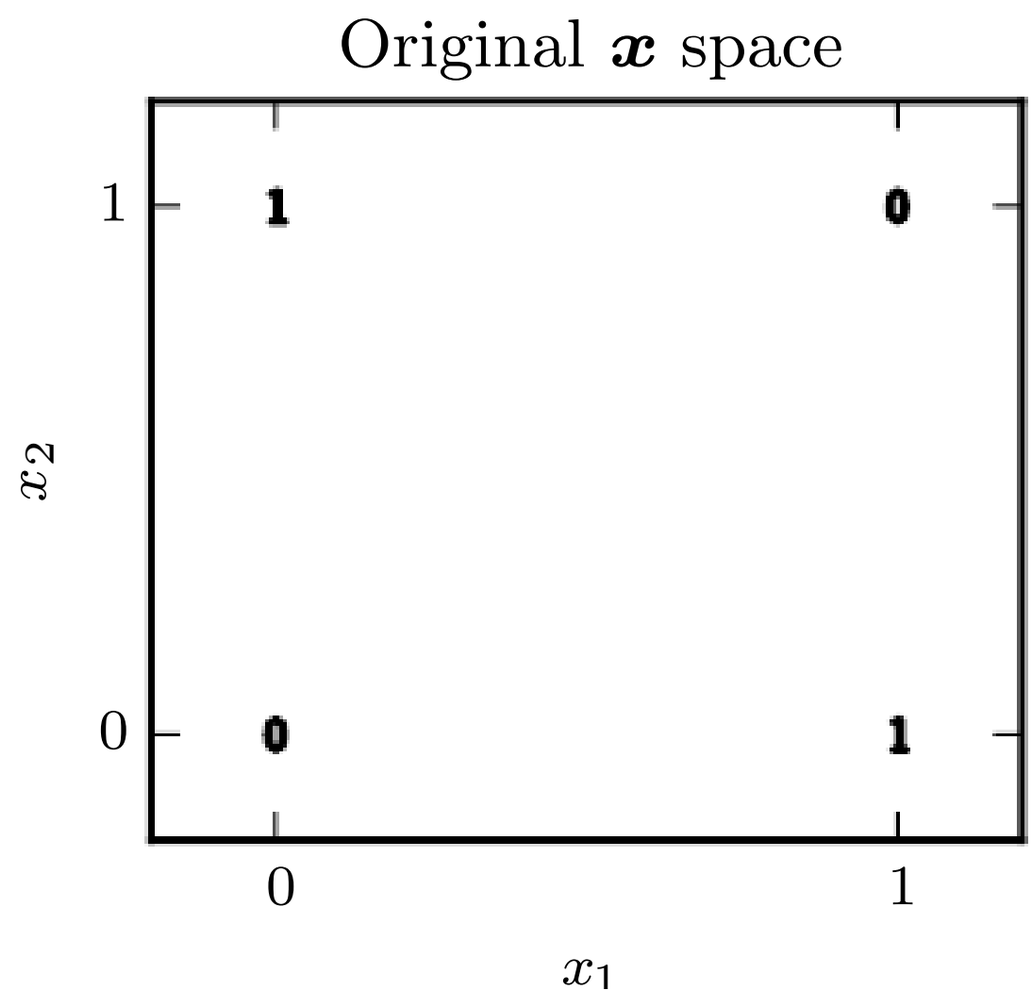
$$J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\boldsymbol{x} \in \mathbb{X}} \left(f^*(\boldsymbol{x}) - f(\boldsymbol{x}; \boldsymbol{\theta}) \right)^2 . \tag{6.1}$$

Now we must choose the form of our model, $f(\boldsymbol{x}; \boldsymbol{\theta})$. Suppose that we choose a linear model, with $\boldsymbol{\theta}$ consisting of $\boldsymbol{w}$ and b. Our model is defined to be

$$f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^\top \boldsymbol{w} + b. \tag{6.2}$$

We can minimize $J(\boldsymbol{\theta})$ in closed form with respect to $\boldsymbol{w}$ and b using the normal equations.

After solving the normal equations, we obtain w = 0 and b = 1/2. The linear model simply outputs 0.5 everywhere. Why does this happen?



When $x_1 = 0$, the model's output must increase as x_2 increases. When $x_1 = 1$, the model's output must decrease as x_2 increases. A linear model must apply a fixed coefficient w_2 to x_2. The linear model therefore cannot use the value of x_1 to change the coefficient on x_2 and cannot solve this problem

Feedforward Network for Learning XOR

$$\mathbf{h} = \mathbf{g}(\mathbf{W}^\top \mathbf{x} + \mathbf{c}) \quad \text{where} \quad g(z) = \max\{0, z\}$$

and finally

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

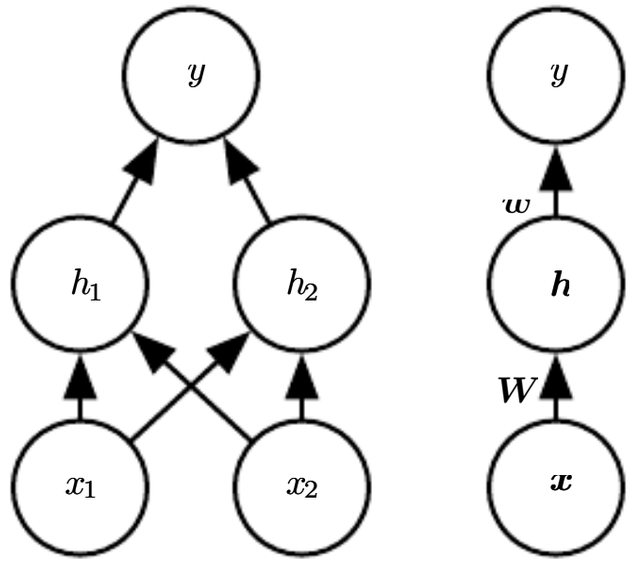


Figure 6.2: An example of a feedforward network, drawn in two different styles. Specifically, this is the feedforward network we use to solve the XOR example. It has a single hidden layer containing two units. *(Left)*In this style, we draw every unit as a node in the graph. This style is very explicit and unambiguous but for networks larger than this example it can consume too much space. *(Right)*In this style, we draw a node in the graph for each entire vector representing a layer's activations. This style is much more compact. Sometimes we annotate the edges in this graph with the name of the parameters that describe the relationship between two layers. Here, we indicate that a matrix $\boldsymbol{W}$ describes the mapping from $\boldsymbol{x}$ to $\boldsymbol{h}$, and a vector $\boldsymbol{w}$ describes the mapping from $\boldsymbol{h}$ to y. We typically omit the intercept parameters associated with each layer when labeling this kind of drawing.

Solution!!

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

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \qquad \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

b=0

$$\begin{bmatrix} 174 \\ \end{bmatrix}$$

Gradient Based Learning

- Neural network causes most interesting loss functions to become non-convex

- Neural networks are usually trained by using iterative, gradient-based optimizers

- Convex optimization algorithms with global convergence guarantees, whereas stochastic gradient descent applied to non-convex loss functions has no such convergence guarantee

- For Feedforward networks training algorithms are improvements and refinements on the ideas of gradient descent

Cost Functions

- Most popular Cost Function comes from MLE
- Alternatively Cost Function is the cross-entropy between the training data and the model distribution
- $J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x})$
- If $p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; f(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{I})$ then,

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

- Advantage of MLE: Specifying a model p(y | x) automatically determines a cost function log p(y | x)
- The negative log-likelihood helps to avoid the saturation problem for many models

Learning Conditional Statistics

- Instead of learning a full probability distribution $p(y \mid x; \theta)$ we often want to learn just one conditional statistic of y given x

- Neural Networks to represent any function f from a wide class of functions. Hence cost function is a functional

- Hence $f^* = \arg\min_f \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{data}}} ||\boldsymbol{y} - f(\boldsymbol{x})||^2$ yields,

$$f^*(\boldsymbol{x}) = \mathbb{E}_{\mathbf{y}\sim p_{\text{data}}(\boldsymbol{y}|\boldsymbol{x})}[\boldsymbol{y}]$$

- Also $f^* = \arg\min_f \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{data}}} ||\boldsymbol{y} - f(\boldsymbol{x})||_1$ yields the median

Linear Output Units for Gaussian output Distributions

- Given features h, a layer of linear output units produces a vector $\hat{\boldsymbol{y}} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$

- Linear output layers are often used to produce the mean of a conditional Gaussian distribution:

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$$

- MLE framework makes it straightforward to learn the covariance of the Gaussian

- The covariance must be constrained to be a positive definite matrix for all inputs which is difficult with linear output layer

Sigmoid Output Units for Bernoulli Output Distributions

- Suppose we were to use a linear unit, and threshold its value to obtain a valid probability:

$$P(y = 1 \mid \boldsymbol{x}) = \max\left\{0, \min\left\{1, \boldsymbol{w}^\top \boldsymbol{h} + b\right\}\right\}$$

- Problem: Any time that $\boldsymbol{w}^\top \boldsymbol{h} + b$ strayed outside the unit interval, the gradient of the output of the model with respect to its parameters would be 0

- Instead use sigmoid unit: $\sigma(w^\top b + h)$

Sigmoid to Normalize

$$\log \tilde{P}(y) = yz$$

$$\tilde{P}(y) = \exp(yz)$$

$$P(y) = \frac{\exp(yz)}{\sum_{y'=0}^{1} \exp(y'z)}$$

$$P(y) = \sigma\left((2y-1)z\right).$$

Importance of Sigmoid

- The cost function used with maximum likelihood is $-\log P(y \mid x)$, the log in the cost function undoes the exp of the sigmoid

- Without this effect, the saturation of the sigmoid could prevent gradient-based learning from making good progress

- Loss Function:

$$
\begin{aligned}
J(\boldsymbol{\theta}) &= -\log P(y \mid \boldsymbol{x}) \\
&= -\log \sigma\left((2y-1)z\right) \\
&= \zeta\left((1-2y)z\right).
\end{aligned}
$$

- Saturation thus occurs only when the model already has the right answer—when $y = 1$ and z is very positive, or $y = 0$ and z is very negative

- When we use other loss functions, such as mean squared error, the loss can saturate anytime $\sigma(z)$ saturates

Other Output Units

- Softmax Units for Multinoulli Output Distributions

- We may wish to learn the variance of a conditional Gaussian for y, given x. there is a closed form expression

- Alternative approach is to simply include the variance as one of the properties of the distribution $p(y \mid x)$ that is controlled by $\omega = f(x; \theta)$.

- In the simple case where the standard deviation does not depend on the input, we can make a new parameter in the network that is copied directly into ω

Other Output Units

- We often want to perform multimodal regression, that is, to predict real values that come from a conditional distribution p(y | x) that can have several different peaks in y space for the same value of x

- In this case, a Gaussian mixture is a natural representation for the output

- A Gaussian mixture output with n components is defined by the conditional probability distribution:

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \sum_{i=1}^{n} p(\mathrm{c} = i \mid \boldsymbol{x}) \mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}^{(i)}(\boldsymbol{x}), \boldsymbol{\Sigma}^{(i)}(\boldsymbol{x}))$$