Deep Feedforward Networks

Hankz Hankui Zhuo

May 10, 2019

http://xplan-lab.org

What's DFN

- Information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y
  - No feedback connections
- Extended to include feedback connections
  - Recurrent Neural Networks
- Also called feedforward neural networks
- Or multilayer perceptions (MLPs)

Structure

- Chain structures:
  - three layers: $f(x) = f^3(f^2(f^1(x)))$
  - "three" is the depth of the model
  - Output layer: the desired output is specified in the training data
  - Hidden layers: the desired output is not specified in the training data
  - Width of the model: the dimensionality of hidden layers

Mapping of input

- Linear model of input x
  - $y=w^{T}x$
- Nonlinear model
  - Introducing mapping ϕ
  - $y=w^{T}\phi(x)$
- How to choose ϕ?
  - 1. use a very generic ϕ:
    - kernel machines, e.g., RBF kernel
    - If ϕ (x) is of high enough dimension, we can always have enough capacity to fit the training set
  - 2. manually engineer ϕ

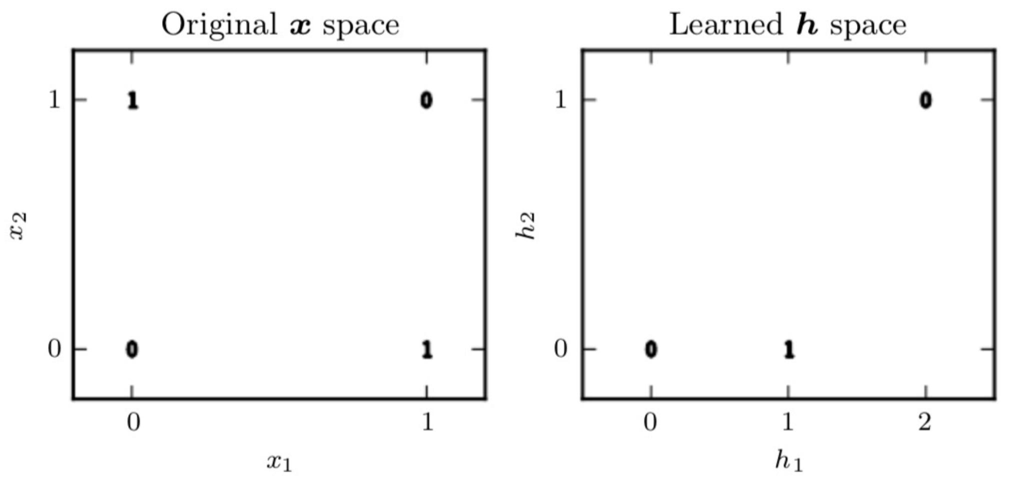
Mapping of input

- 3. The strategy of deep learning is to learn ϕ
  - $y=f(x;\theta,w)=\phi(x;\theta)^T w$
  - This approach can capture the benefit of the first approach by being highly generic
    - we do so by using a very broad family $\varphi(x;\theta)$.
  - Can also capture the benefit of the second approach
    - Human practitioners can encode their knowledge to help generalization by designing families $\varphi(x;\theta)$ that they expect will perform well.

Example: Learning XOR

- XOR is an operation on two binary values
- When exactly one of these binary values is equal to 1, the XOR function returns 1
- Training data with four points
  - $X=\{[0,0],[0,1],[1,0],[1,1]\}$
- MSE (mean squared error) loss function:
  - $J(\theta)=1/4\Sigma_x(f(x)-f(x;\theta))^2$
- Choose the form of our model $f(x;\theta)$
  - Linear model: $f(x;w,b)=x^Tw+b$
- We got $w=0$ and $b=1/2$, i.e., The linear model simply outputs 0.5 everywhere

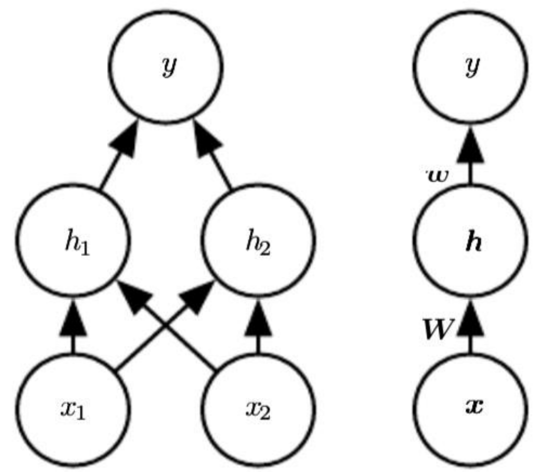
Why?



- 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.

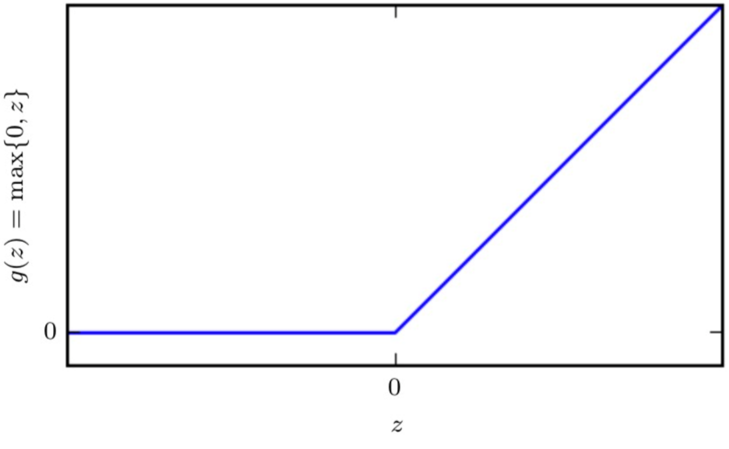
A simple feedforward network

- One hidden layer with two hidden units
- Hidden layer:
  - $h = f^1(x; W, c)$
- Output layer:
  - $y=f^2(h;w,b)$
- Complete model:
  - $f(x;W,c,w,b)=f^2(f^1(x))$
- If f^1 use linear model, then
  - $h=f^1(x)=W^T x$, $f^2(h)=h^T w$,
  - Then we have $y=f(x)=w^T W^T x$
  - Or just $f(x)=x^T w'$, where $w'=Ww$
  - We thus cannot solve the problem
- We need a nonlinear function
- Use a fixed nonlinear function called activation function
  - $h=g(W^T x+c)$
- g is typically chosen to be a function that is applied element-wise
  - $h_i=g(x^T W_{:,i}+c_i)$



Rectified Linear Unit (ReLU)

- The activation function g is defined by
  - $g(z)=\max\{0,z\}$
- i.e., the complete network is:
  - $f(x;W,c,w,b)=w^T\max\{0,W^Tx+c\}+b$



A solution to XOR

- A solution as show below:

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad \boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

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

- We can walk through the way the model processes based on f(x;W,c,w,b)=w^{T}max{0,W^{T}x+c}+b

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad \boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

add c | apply ReLU | multiply w

Gradient-Based Learning

- difference between the linear models and neural networks is
  - the nonlinearity of a neural network causes most interesting loss functions to become non-convex
  - This means that neural networks are usually trained by using iterative, gradient-based optimizers that merely drive the cost function to a very low value, rather than the linear equation solvers used to train linear regression models or the convex optimization algorithms with global convergence guarantees
  - Stochastic gradient descent applied to non-convex loss functions has no such convergence guarantee, and is sensitive to the values of the initial parameters.
  - For feedforward neural networks, it is important to initialize all weights to small random values.

Gradient-Based Learning

- The biases may be initialized to zero or to small positive values.
- For the moment, it suffices to understand that the training algorithm is almost always based on using the gradient to descend the cost function in one way or another.
- The specific algorithms are improvements and refinements on the ideas of gradient descent
- Computing the gradient is slightly more complicated for a neural network, but can still be done efficiently and exactly

Cost Function

- Cross-entropy between training data and the model distribution

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

- The specific form of the cost function changes from model to model, depending on the specific form of $\log p_{\text{model}}$
- For $p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; f(\boldsymbol{x};\boldsymbol{\theta}), \boldsymbol{I})$
- We have

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

Output Units

- suppose that the feedforward network provides a set of hidden features defined by
  - $h = f(x;\theta)$
- Linear Units for Gaussian Output Distributions

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

- Maximizing the log-likelihood is then equivalent to minimizing the mean squared error

Output Units (continued)

- Sigmoid Units for Bernoulli Output Distributions
- Suppose we were to use a linear unit, and threshold its value to obtain a valid probability
- A sigm $P(y=1 \mid \boldsymbol{x}) = \max\left\{0, \min\left\{1, \boldsymbol{w}^\top \boldsymbol{h} + b\right\}\right\}$
- logistic sigmoid

$$\hat{y} = \sigma\left(\boldsymbol{w}^\top \boldsymbol{h} + b\right)$$

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

Output Units (continued)

- The sigmoid output unit as having two components
- First, it uses a linear layer to compute $z=w^{T}h+b$
- Next, it uses the sigmoid activation function to convert z into a probability
- We omit the dependence on x for the moment to discuss how to define a probability distribution over y using the value z

Output Units (continued)

- omit the dependence on x for the moment to discuss how to define a probability distribution over y using the value z
- The sigmoid can be motivated by constructing an unnormalized probability distribution $\tilde{P}(y)$
- We then normalize to see that this yields a Bernoulli distribution controlled by a sigmoidal transformation of z
- The loss function for maximum likelihood learning of a Bernoulli parametrized by a sigmoid is

$$\begin{aligned}\log \tilde{P}(y) &= yz \\ \tilde{P}(y) &= \exp(yz) \\ P(y) &= \frac{\exp(yz)}{\sum_{y'=0}^{1} \exp(y'z)} \\ P(y) &= \sigma\left((2y-1)z\right).\end{aligned}$$

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

To be continued!