Chapter 6 Deep Feedforward Networks

Overview

- The goal of a feedforward network is to approximate some function
- Called **feedforward** cause no output will be the feedback of the input. If there are feedback, it's called **recurrent neural networks**
- Called networks since it's a composition of multiple simpler functions
- Feedforward network is inspired by brain. But it should better be regarded as a function approximation machine rather than a model of a brain machine

As an extension of linear models:

Use a non-linear transform ϕ to transform input features to a new feature space $\phi(\mathrm{x})$

Three methods to choose the mapping ϕ

1. Use very generic ϕ, such as the infinite-dimensional ϕ used in kernel machine based on RBF kernel
2. Use manually engineer ϕ
3. Learning a ϕ , from a given model families derived by human knowledge (Adopted by deep learning)

6.1 Example: Learning XOR

- Linear model cannot learn XOR function
- Feedforward network $\implies$ Affine transformation + Non-linear activation functions

> 这一章用一个双层的MLP 学习XOR 为例子说明的了什么是feedforward networks, 主要就是介绍了MLP 的组成部分，不了解的可以看一下

6.2 Gradient-based Learning

Overview

- Non-linearity of neural networks $\implies$ Non-convex cost functions
- Non-convex optimization $\implies$ No guarantee to converge to global minimal
- Not robust
- Sensitive to the initial choice of parameters
- Stochastic gradient descent

In order to use SGD for optimization, we first need to define the cost function and find its gradient.

6.2.1 Cost Functions

Review: The cost function is presented below:

$$J(\theta) = \mathbb{E}_{\mathbf{x},y\sim\hat{p}_{data}} L(\mathbf{x}, y, \theta) = \frac{1}{m}\sum_{i=1}^{m} L(\mathrm{x}^{(\mathrm{i})}, y^{(i)}, \theta)$$

Where $L((x), y, \theta)$ is example-wise loss function

Two kinds of interpretation about output

- The output reveal a conditional probability distribution $P(\mathbf{y}|\mathrm{x};\theta)$
- The output gives a single conditional statistics

> Cost function其实对于所有 parametric model 都是通用的
>
> 不同的Cost function 其实就是对Output layer产生的output不同角度的解读

6.2.1.1 Learning Conditional Distributions with Maximum Likelihood

> 这里把Output理解为对应一个Output distribution
>
> 例如，一个Bernoulli Distribution 可以用一个 [0, 1] 的scalar确定
>
> 一个 Multinoulli Distribution 如果有N种Class, 可以用$N-1$ 维的vector 唯一确定
>
> 一个Gaussian Distribution $\mathcal{N}(y;\mu,\sigma)$ 可以由 两个数 μ, σ 唯一确定

Cost function

如果是拟合的一个probability distribution, 很自然的可以用Cross-entropy 作为loss function。因为最小化cross-entropy 等价于最小化NLL(negative log-likelihood) 所以loss funtion就有了如下的形式

$$J(\theta) = \mathbb{E}_{x,y\sim\hat{p}_{data}}[-\log P_{model}(y;x)]$$

如果确定了我们要拟合的Probability distribution family（例如到底是Bernoulli, multinoulli, Gaussian 还是其他）也就确定了对应的Cost function的函数形式。所以选定要拟合的Model类型，或者说最后的probability distribution 的类型其实是在implicitly 引入 priori beliefs

例如，我们确定了需要拟合一个Gaussian distribution $P_{model}(y|x) = \mathcal{N}(y; f(x;\theta), I)$, 这里的意思是我们认为Output对应这个分布的均值，那么根据上面的Cost function, 最后得到的Cost function是

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

这个跟Chapter 5 Maximum likelihood estimation那部分对于基于MSE的Linear regression model和基于NLL的Maximum likelihood estimation的等价性的一样的

Advantages

- 确定了我们要拟合的$P(y|x)$的类型就确定了Cost function
- NLL 自带的 $\log$ 函数抵消了$\exp$ 在x轴左侧的saturation， saturation对于 gradient descent optimization有很大的危害

Disadvantages

- The cross-entropy loss usually do not have a minimum value

> 这里不太懂，为什么没有最小值，书里面也没有举例

6.2.1.2 Learning Conditional Statistics

> 对比前一个小结，这里就是把输出理解成一个单独的conditional probability
>
> 书中这部分的叙述我没看懂

6.2.2 Output Units

- Any kind of neural network unit that may be used as an output unit can also be used as a hidden unit
- The output unit provide some additional transformation from the features to complete the task that the network must perform
- In this section, we assume that the output of previous hidden layer is h

> 其实挺好理解的， 比如这个neural network的目的就是做一个Binary classification， 而且我们选定了Cost function是Learning Conditional Distribution with maximum likelihood 那么最后Output unit就需要得到一个scalar value

6.2.2.1 Linear Units for Gaussian Output Distributions

The output is simply $\hat{y} = W^\top h + b$, without non-linearity

The output $\hat{y}$ is interpreted as the mean of the Gaussian distribution

$$P(y|x) = \mathcal{N}(y; \hat{y}, I)$$

Advantage

- Linear output unit does not saturation

6.2.2.2 Sigmoid Units for Bernoulli Output Distributions

Output Requirement

To determine a Bernoulli distribution, a scalar value in $[0, 1]$ is required. Theoretically, any function that has range $[0, 1]$ can be used as the output layer here.

For example, the following function is a valid output unit function

$$f(h) = \max\{0, \min\{1, w^\top h + b\}\}$$

But this $f(h)$ has saturation problem, thus making it hard to optimize with gradient descent

Sigmoid output unit with NLL

An solution to the saturation problem is the combination of sigmoid function with NLL

If use MSE with sigmoid output , there also existed saturation problem

6.2.2.3 Softmax Units for Multinoulli Output Distribution

> Softmax 其实是generalized sigmoid
>
> 这一个section和前面一个Section 主要讲的是怎么处理saturation的问题， 要妥善的选择 Output unit 和对应的Cost function 才能避免saturation。

Output Requirement

For n classes classification problem, a $n-1$ dimensional array is required. The sum of the elements in the array should be within $[0,1]$

Saturation problem

The combination of NLL and Softmax can solve the saturation problem of softmax output unit since the log in NLL undoes the exponential in softmax.

If use MSE as cost function, there will be saturation problems

6.2.2.4 Other Output Types

> 这一个section 没怎么看懂，需要之后再看

6.3 Hidden Units

> Output units 其实是所有Machine learning 算法通用的，或者说是Task-specific的，不是feedforward network的特有属性。 Hidden units 才是 Feedforward neural network 区别的其他算法的地方。
>
> 这一个section主要讨论了
>
> - 有哪些hidden unit 的类型
> - 各自的优缺点是什么

- ReLU is an excellent choice for default hidden unit for feedforward neural network

- ReLU is not differentiable at all input point but it can works
    - The training algorithm do not usually arrive at a local minimum. In must case, we just arrive at a point that the training error is small enough
- Under most situation, hidden unit = affine transformation + non-linearity

A common format for hidden unit

$$h = g(w^\top x + b)$$

Where $g(z)$ is the non-linear function, which is used to distinguish different types of hidden units.

6.3.1 Rectified Linear Units and Their Generalizations

ReLU

$$g(z) = \max\{0, z\}$$
$$h = g(w^\top x + b)$$

- ReLU is a piece-wise linear function
- b will be initialized with small positive value

Advantages

- Easy to optimize since it's close to linear function
- Preserve gradient (The derivative remains large when the unit is active)
- The second order is 0 almost everywhere, no *second order effects*(What is this ????)

Disadvantages

- Cannot learn from example for which their activation is zero

Generalization of ReLU

The generalizations are based on $h = g(z, \alpha) = \max(0, z) + \alpha \min(0, z)$

- Absolute value rectification $g(z) = |z|$, where $\alpha = 1$
- Leaky ReLU, where α is a small positive value
- Parametric ReLU, where α is a trainable parameter
- Maxout units

> 这里的Maxout 其实是作者Goodfollow提出的， 介绍了很多好处和有点，有一点夹带私货的嫌疑

Catastrophic Forgetting

在持续性学习中，学习新的知识会很快遗忘旧知识的问题。 例如，一个neural network如果已经学会了怎么下象棋，然后再去训练怎么下五子棋，很快就会不知道怎么下象棋

作者提到Maxout可以解决catastrophic forgetting的问题，没有给出很具体的例子，存疑

6.3.2 Logistic Sigmoid and Hyperbolic Tangent

Logistic sigmoid

$$g(z) = \sigma(z)$$

Hyperbolic tangent

$$g(z) = \tanh(z)$$

These two activation functions are highly related cause $\tanh(z) = 2\sigma(2z) - 1$

Disadvantage

- Saturation of sigmoid and hyperbolic tangent make it hard to train
- Not a good choice for hidden unit

Note

- If we must use the squash hidden unit, Hyperbolic tangent is better than sigmoid since the hyperbolic tangent is close to identity function $g(z) = z$ when input is close to 0

6.3.3 Other Hidden Units

In general , many differentiable function can be used as activation function.

> 在对应的Chapter 6的代码中，我在MNIST上测试了Cosine 用作activation 的效果，并没有书中说的那么好当然，我只是试了一种Hyperparameter setting 可能跟这个有关

Other possible types of hidden untis

- Linear hidden unit, no activation function
    - effective to reduce the number of parameters in a network
- Softmax hidden unit
    - Can act as a switch
- Radial basis function (RBF unit)
- Softplus unit (Differentiable edition of ReLU)
- Hard $\tanh$

6.4 Architecture Design

- **Architecture** refers to the overall structure of the network: how many units it should have and how these units should be connected to each other.
- In the common case, the feedforward neural network is organized into layers in a chain structure

- In the chain-based architecture, the main architectural considerations are to choose the depth of the network and the width of each layer

6.4.1 Universal Approximation Properties and Depth

Universal Approximation theorem A feedforward network with a linear output layer and at least one hidden layer with any "squashing" or ReLU activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units

> Universal approximate theorem 其实就是再说，只有一层hidden layer的feedforward neural network其实就可以拟合任意的函数了，只要hidden unit size足够大。

Challenges

- The learning algorithm may not be able to find the value of the parameters that corresponding to the desired function
- The learning algorithm may lead to overfitting
- The number of hidden units required may grow exponentially with the input size

Summary of universal approximation theorem

A feedforward with single hidden layer is sufficient to represent any function but the layer may be infeasibly large and may fail to learn and generalize correctly.

> 这一节后面的部分介绍了一个关于input d, depth l , number of unit n 和number of linear regions carved out by this deep rectifier network的结论，但是我没看懂

Practical experience

Tend to use deep model

- Greater depth does seem to result in better generalization for a wide variety of tasks

6.4.2 Other Architectural Considerations

- No need to have chain-based architecture
  - Can have skip connection or some other kinds of connections
- How to connect a pair of layers to each other
  - Feedforward neural network use a matrix W
  - CNN use a sparse connection (Not fully connected)

6.5 Back-Propagation and Other Differentiation Algorithms

Forward propagation

From the input $\mathbf{x}$ to get the output y

Back propagation

Compute the gradient

Stochastic gradient descent

Use the gradient to learning the parameters

Why use back-propagation

- Compute the gradient of a complex function numerally is computation expensive
- Backprop use a simple and inexpensive procedure with additional storage cost

6.5.1 Computational Graphs

To precisely describe back-propagation algorithm, we need to have formal **computational graph** language

Variable

The variable may be scalar, vector, matrix, tensor or even a variable of another type

Operation

A operation is a simple function of one or more variable

Each node is a variable. One or several edges represent a operation

Then the neural network can be represented as a DAG

6.5.2 Chain Rule of Calculus

Omit

6.5.3 Recursively Applying the Chain Rule to Obtain Backprop

> 实际上这一节就表达了一个想法，在backpop的时候有两种选择

> 1. 不存储中间节点的Jacobian， 会有很多重复计算
> 2. 存储中间节点的Jacobian和对应的hidden state output， 会省去很多重复计算，但是有额外的存储开销

6.5.4 Back_propagation Computation in Fully-Connected MLP

Omit

6.5.5 Symbol-to-Symbol Derivative

Symbolic and numeric

Symbol $\implies$ variable that does not have specific value, like a, b, x, y

Numeric $\implies$ variable with specific value, like $x = [1, 2, 3]^\top$

Symbol-to-number differentiation

Take a computational graph and a input, return a set of numerical value describing the gradient at those input values

- used by Torch and caffe

Symbol-to-Symbol defferentiation

Take a computational graph and add additional nodes to the graph that provide a symbolic description of the desired derivatives

- used by Theano and Tensorflow

6.5.6 General Back-propagation

Omit

6.5.7 Example : Back-Propagation for MLP Training

Omit

6.5.8 Complication

Omit

6.5.9 Differentiation outside the Deep Learning Community

The back-propagation algorithm is a special case of a boarder class of techniques called **reverse mode accumulation**

6.5.10 Higher-order Derivatives

Krylov methods

Iterative approximation algorithms to compute the Hessian matrix

> Higher-order derivatives 到底在训练过程中有什么用呢?

6.6 Historical Notes

现在在deep learning 中应用很多的techniques其实上世纪80-90年代就提出来了，这一波机器学习或者说人工只能的复兴主要原因有两个

1. Larger dataset
2. More powerful computer

算法上最大的提升就是用 piece-wise linear activation function , 也就是ReLU