Python Machine Learning: The 10th Book Circle

Deep Neural Network Algorithm I: Multi-Layer Neural Network in General

Jianan Liu

Gothenburg

July, 3, 2017

Disclaimer

All opinions and statements in this presentation are mine and do not in any way represent the company
Any comment or correction of error is welcome, please contact me
chisyliu@hotmail.com

Acknowledgement

Acknowledgement

I would like to thank Tiancong Zheng from machine learning group in Zenuity for his recommendation of books regarding modern deep neural network. As an engineer who has statistical signal processing/wireless communication academic background, most of knowledge in neural network learned from university were from Simon Haykin's papers and book which are more related to classical feedforward neural network and statistical adaptive filter, the books Tiancong refereed are extremely good complementary resource for me to get deep insight of modern deep learning.

The 10th Book Circle of Python Machine Learning

In this presentation belongs to **algorithm** part of the book

- If you have time, please read the book first. This slide could be used as complementary resource for the book
- If you want to know more about Classical Feed-Forward Neural Network in General, I recommend reading Simon Haykin's book Neural Network and Learning Machine or Kalman Filtering and Neural Network
- We will try to go through basic mathematics behind Classical Multi-Layer Feed-Forward Neural Network.
- All of us need to debug the python code, in order to get practice of implementing machine learning algorithm
- A complete resources for deep neural network and machine learning could be found here ▸ Link

Data Set for Feedforward Neural Network

Data set used for feedforward neural network

Training data set (Y, X) for feedforward neural network

Suppose we have M samples, each sample has d features/dimensions and l output labels(So the input **X** is M by d matrix, label **Y** is M by l vector). Note we don't use l or L to denote the number of samples in feedforward neural network in this presentation just because we want to uniform the notations between the equations and figures we borrowed from other book/course

Overview

Outline for Section 1

Perceptron Model

Let's recall the first linear classifier: perceptron, which we learned in the 1st circle

Perceptron Model

- $z^{(l)} = \mathbf{x}^{(l)}\mathbf{w}$, $\mathbf{x}^{(l)}$ is 1 by D vector which represents D features/dimension for l^{th} training data sample and $\mathbf{w}$ is D by 1 vector which represents weight
- $y^{(l)} = 1$, if $z^{(l)} \geq$ threshold; $y^{(l)} =$ -1, otherwise. where $y^{(l)}$ is the decision output for l^{th} training data sample

Adaptive Algorithm for Percetron Model

```
1:  Initialize the all elements $w_d$ in weight $\mathbf{w}$ to 0
2:  for t in T do
3:      for l in L do
4:          computer $\hat{y}^{(l)} = \text{sgn}(\mathbf{x}^{(l)}\mathbf{w})$
5:          for d in D do
6:              computer $w_d(\text{t}+1) = w_d(\text{t}) + \eta(y^{(l)} - \hat{y}^{(l)})x_d^{(l)}$
7:              d = d + 1
8:          end for
9:          Return w if all training samples are correctly classified
10:         t = t + 1
11:     end for
12: end for
```

- $\hat{y}^{(l)}$ is predication of decision output of l^{th} sample
- $y^{(l)}$ is the real decision output of l^{th} sample training data
- T is the number of iteration(guarantee algorithm can stop)

Single Layer Neural Network(Adaptive Linear Neuron: Adaline)

Now let's recall the second linear classifier, Adaptive linear neuron(Adaline)

- In perceptron, the weight is updated according to $\Delta \mathbf{w} = \eta(y^{(l)} - \hat{y}^{(l)})\mathbf{x}^{(l)}$, where $\hat{y}^{(l)} = \text{sgn}(\mathbf{x}^{(l)}\mathbf{w})$. Which means we use the binary value $y^{(l)}$ as lable of training data directly to update weight of model
- But in adaptive linear neuron(adaline), we use $\hat{y}^{(l)} = \mathbf{x}^{(l)}\mathbf{w}$ to be as criteria for updating weight, which means instead of updating weight by applying the binary predication to compare with training data directly, we use the continues predication value $\hat{y}^{(l)}$ to calculate the difference/error by comparing with binary label $y^{(l)}$ and update weight of linear model $\mathbf{x}^{(l)}\mathbf{w}$, then apply the weight and map the result of new unknow data to binary value

Batch Gradient Descent Algorithm for Adaline

How to train the linear model $y^{(l)} = \mathbf{x}^{(l)}\mathbf{w}$ for all l in L sample?

Batch Gradient Descent Algorithm

- The objective cost function $\mathbf{J}(\mathbf{w}) = \Sigma_l (y^{(l)} - \mathbf{x}^{(l)}\mathbf{w})^2$ to be minimized, is convex
- The adaptive algorithm which updates the weights by taking a step away from the gradient $\bigtriangledown \mathbf{J}(\mathbf{w})$ of our cost function $\mathbf{J}(\mathbf{w})$: $\mathbf{w}(t+1) = \mathbf{w}(t) + (-1)\eta \bigtriangledown \mathbf{J}(\mathbf{w})$
- $\bigtriangledown \mathbf{J}(\mathbf{w})$ equals to compute the partial derivative of the cost function with respect to each weight w_d: $\bigtriangledown \mathbf{J}(\mathbf{w}) = \frac{\partial \mathbf{J}(\mathbf{w})}{\partial w_d} = 2(-\Sigma_l (y^{(l)} - \mathbf{x}^{(l)}\mathbf{w}) x_d^{(l)})$ for all w_d
- Compared with perceptron, BGD for adaline doesn't update the weight **w** every sample but count all L samples together to update weight once. That is reason it is **Batch** gradient descent

Batch Gradient Descent Algorithm for Adaline

```
1: Initialize the all elements $w_d$ in weight $\mathbf{w}$ to 0
2: for t in T do
3:     for d in D do
4:         computer $w_d(t+1) = w_d(t) + \eta\Sigma_l(y^{(l)} - \mathbf{x}^{(l)}\mathbf{w})x_d^{(l)}$
5:         d = d + 1
6:     end for
7:     t = t + 1
8: end for
```

- $y^{(l)}$ is the real continues intermediate output of l^{th} sample training data
- T is the number of iteration

Stochastic Gradient Descent Algorithm

In most of the scenario of supervised machine learning, the number of training data samples L is quite large, the computation time of updating weight will be too long for BGD cause it needs all samples to update the weight once. So we use Stochastic GD. The other reason is, Stochastic GD will somehow avoid falling into local optimum when applying on a non-convex problem

Stochastic Gradient Descent Algorithm

- $\bigtriangledown \mathbf{J}(\mathbf{w})$ in BGD is $\bigtriangledown \mathbf{J}(\mathbf{w}) = \frac{\partial \mathbf{J}(\mathbf{w})}{\partial w_d} = 2(-\Sigma_l (y^{(l)} - \mathbf{x}^{(l)}\mathbf{w}) x_d^{(l)})$ for all w_d, but now stead of computing $\bigtriangledown \mathbf{J}(\mathbf{w})$ by all L samples, we randomly update weight once per sample
- $\bigtriangledown \mathbf{J}(\mathbf{w})$ in SGD is $\bigtriangledown \mathbf{J}(\mathbf{w}) = \frac{\partial \mathbf{J}(\mathbf{w})}{\partial w_d} = 2(-(y^{(l)} - \mathbf{x}^{(l)}\mathbf{w}) x_d^{(l)})$ for all w_d, it is updated every sample

Stochastic Gradient Descent Algorithm for Adaline

```
1:  Initialize the all elements $w_d$ in weight $\mathbf{w}$ to 0
2:  for t in T do
3:      Shuffle(); // randomly sort the L samples once
4:      for l in L do
5:          for d in D do
6:              computer $w_d(t+1) = w_d(t) + \eta(t)(y^{(l)} - \mathbf{x}^{(l)}\mathbf{w})x_d^{(l)}$
7:              d = d + 1
8:          end for
9:          l = l + 1
10:     end for
11:     t = t + 1
12: end for
```

- $\eta(t)$ means η is a function of t

Stochastic Gradient Descent Algorithm for Adaline

Adaptive Learning Rate

The learning rate should also be adaptively changed, basically decreases over time(The closer to the global minimum the smaller each "step of learning" should be in order to ensure the (approximate)minimum value could be captured), a common method is define the learning rate to be inverse of number of iteration time

- $\eta(t) = \frac{a}{t+b}$, a and b are constants

Mini-Batch Gradient Descent: Hybrid Combination of Ideas from BGD and SGD

- We could also update weight once per patch, a patch has several samples(larger than 1 sample in SGD but smaller than all samples as in BGD)
- For each iteration, Mini-Batch Gradient Descent uses a patch(which contains "number of batch size" samples) to train target model. It will use all patches by iteration "number of batches" times. Note "patch size" x "number of patches" = total number of data samples in data set. All data samples in data set forms an epoch
- Mini-Batch Gradient Descent Algorithm is most common method to train deep neural network and we will see it a lot in future. We will analyze reason in the following section

Mini-Batch Gradient Descent for Adaline Model

```
 1: Initialize the all elements $w_d$ in weight $\mathbf{w}$ to 0
 2: for e in E do
 3:     for t in T do
 4:         for d in D do
 5:             computer
    $w_d(t+1) = w_d(t) + \eta\Sigma_{n=1}^{N}(y^{(l)} - \mathbf{x}^{(l)}\mathbf{w})x_d^{(l)}$
 6:             d = d + 1
 7:         end for
 8:         t = t + 1
 9:     end for
10: end for
```

- E is number of epoch(that is how many times we repeat on the whole data set)
- T is the number of iteration, and $T = L/N$. L is the total number of data samples in the data set and N is the batch size

Linear Aggregation of Multiple Perceptrons: Idea of One Layer Neural Network

Recall linear aggregation we introduced in 3rd book circle when we stepped from decision tree into random forest(Although actually we used bootstrap data samples method to replace aggregation of models in random forest)

-

Linear Aggregation of Multiple Perceptrons: Idea of One Layer Neural Network

What we could fetch by linear aggregating of multiple perceptrons?
The answer is enough perceptrons = smooth boundary
We could check the example from Neural Network part of Hsuan-Tien Lin's Machine Learning Technique Course

-

Multi-Layer of Multiple Perceptrons: Basic Idea Behind Multi-Layer Neural Network

-

Multi-Layer Neural Network: Other Activation Function to Replace of Sign Function in Preceptron

-

Advantage and Drawback of Feed-Forward Neural Network

-

Outline for Section 2

Objective Functions for Neural Network

Minimize the negative log of likelihood

- The most common used objective function for neural network is **maximize likelihood**, or more precisely minimize the negative log of likelihood

Objective Functions for Neural Network

Minimize cross entropy

- The other common used objective function for neural network is **minimize cross entropy**. It is quite popular cause this objective function could provide larger gradient even when combine with some kind of activation function which easily saturates(e.g. sigmoid function), it means it is easy to train/optimize

Objective Functions for Neural Network

Least square of output error

- Or, same as saw for classical machine learning technology, **least square of output error**. This is the worst(probable) objective function for training deep neural network cause it usually provides small gradient when combine with some kind of activation function which easily saturates(e.g. sigmoid function), hard to train/optimize

Activation Functions for Neuron on Output Layer

The common principle to choose activation function on output layer, is, using the function which could make the "combination of activation function and selected objective function" avoid saturating(saturating stands for "flat on the shape", it will make the gradient too small to train/optimize)

Linear activation function

-

Activation Functions for Neuron on Output Layer

Sigmoid activation function

-
- When using Sigmoid as activation function on the output layer, we usually choose minimize the negative log of likelihood as objective function to avoid saturating

Activation Functions for Neuron on Output Layer

Softmax activation function

-
- When using Softmax as activation function on the output layer, we usually choose minimize cross entropy as objective function to avoid saturating

Activation Functions for Neuron on Hidden Layer

Design the activation function for hidden layer is somehow still an open/research topic, the commonly used ones in the hidden layer neuron are:

Sigmoid activation function

-

ReLU activation function

-

Activation Functions for Neuron on Hidden Layer

Other possible activation functions for neuron on hidden layer include:

Tanh activation function

-

RBF activation function: Radial Basis Function based Neural Network

-

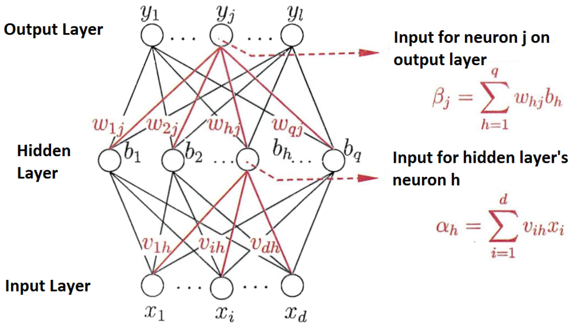
Backpropagation Algorithm for Multi-Layer Feed-Forward Neural Network

As we discussed, gradient descent based search algorithm is widely used to train neural network, backpropagation is the principle/method to realize the gradient descent for each neuron

- From this page we will see what is the backpropagation
- We will also discuss how to use backpropagation in general to implement the calculation of gradient for each neuron for computation(in practice)

Notations in Feedforward Neural Network

- Let's check how the backpropagation algorithm works for feedforward (fully connected) neural network
- I borrowed figure and example from Zhihua Zhou's machine learning book to represent structure of feedforward neural network, and we use the same the notations as in the figure



$$\beta_j = \sum_{h=1}^{q} w_{hj} b_h$$

$$\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$$

Notations and Variables

Following the figure on previous page and use sigmiod as activation function for all neurons, we have

- The input for hidden layer's h^{th} neuron: $\alpha_h = \sum_{i=1}^{d} v_{i,h} x_i$, where $v_{i,h}$ is the weight for neuron on hidden layer
- The output from hidden layer's h^{th} neuron: $b_h = f(\alpha_h - \gamma_h)$, where f(z) is sigmoid function
- The input for output layer's j^{th} neuron: $\beta_j = \sum_{h=1}^{q} w_{h,j} b_h$, where $w_{h,j}$ is the weight for neuron on output layer
- The output from output layer's j^{th} neuron: $y_j = f(\beta_j - \theta_j)$, where f(z) is sigmoid function

How to Calculate the Gradient for All Variables?

- Using the gradient descent searching algorithm in the same way as we used in Adaline, we have the **variable update rule for any variable** as below:
  $v = v + \Delta v$
- where $\Delta v = -\eta \frac{\partial E_k}{\partial v}$ stands for the reverse direction of gradient with learning rate
- where $E_k = \frac{1}{2} \sum_{j=1}^{j=l} (y_j^k - \hat{y}_j^k)^2$ stands for the sum squared error between label of training data y_j^k and the estimation/output from j^{th} neuron on output layer of feedforward neural network $\hat{y}_j^k$, for k^{th} training sample, which is the objective function we want to minimize(It is clearly a least square error method which stands for maximize likelihood)

How to Calculate the Gradient for All Variables?

- Note: Here we just assume gradient descent searching algorithm could be used to find the optimum for feedforward neural network, **however we could clearly see the feedforward neural network is NOT convex**! It means using gradient descent searching algorithm for feedforward neural network simply will lead to local minimum. How to solve this issue?
- We will discuss this issue later
- Currently let's just go ahead towards how to update variables

How to Calculate the Gradient for All Variables?

Totally in this 3 layers feedforward neural network, we have several variables to update: $w_{h,j}$, θ_j, $v_{i,h}$, γ_h
By following the variable update rule on the previous page, we firstly check how to update variable $w_{h,j}$ by chain rule

- $\Delta w_{h,j} = -\eta \frac{\partial E_k}{\partial w_{h,j}} = -\eta \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{h,j}}$
- obviously, $\frac{\partial \beta_j}{\partial w_{h,j}} = \frac{\partial(\sum_{h=1}^{q} w_{h,j} b_h)}{\partial w_{h,j}} = b_h$, so we have $\Delta w_{h,j} = -\eta \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot b_h$ until this step
- Notice $\hat{y}_j^k = f(\beta_j - \theta_j)$, and $f^{'}(z) = f(z)(1 - f(z))$ for sigmoid f(z)
- So $\frac{\partial \hat{y}_j^k}{\partial \beta_j} = \frac{\partial f(\beta_j - \theta_j)}{\partial \beta_j} = f(\beta_j - \theta_j)(1 - f(\beta_j - \theta_j))\frac{\partial(\beta_j - \theta_j)}{\partial \beta_j} = f(\beta_j - \theta_j)(1 - f(\beta_j - \theta_j)) = \hat{y}_j^k(1 - \hat{y}_j^k)$

How to Calculate the Gradient for All Variables?

- We have $\Delta w_{h,j} = -\eta \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot (\hat{y}_j^k(1 - \hat{y}_j^k)) \cdot b_h$ until this step
- The last, $\frac{\partial E_k}{\partial \hat{y}_j^k} = \frac{\partial(\frac{1}{2}\sum_{j=1}^{j=l}(y_j^k - \hat{y}_j^k)^2)}{\partial \hat{y}_j^k} = -(y_j^k - \hat{y}_j^k)$
- back to the original equation, finally we get $\Delta w_{h,j} = -\eta(-(y_j^k - \hat{y}_j^k)) \cdot (\hat{y}_j^k(1 - \hat{y}_j^k)) \cdot b_h = \eta \hat{y}_j^k(1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k) b_h = \eta g_j b_h$, where $g_j = \hat{y}_j^k(1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k)$

How to Calculate the Gradient for All Variables?

Let's to update variable θ_j by chain rule

- $\Delta\theta_j = -\eta\frac{\partial E_k}{\partial\theta_j} = -\eta\frac{\partial E_k}{\partial\hat{y}_j^k}\cdot\frac{\partial\hat{y}_j^k}{\partial\theta_j}$
- similarly, notice $\hat{y}_j^k = f(\beta_j - \theta_j)$, and $f^{'}(z) = f(z)(1-f(z))$ for sigmoid f(z)
- So $\frac{\partial\hat{y}_j^k}{\partial\theta_j} = \frac{\partial f(\beta_j-\theta_j)}{\partial\theta_j} = f(\beta_j-\theta_j)(1-f(\beta_j-\theta_j))\frac{\partial(\beta_j-\theta_j)}{\partial\theta_j} = f(\beta_j-\theta_j)(1-f(\beta_j-\theta_j))(-1) = -\hat{y}_j^k(1-\hat{y}_j^k)$
- The last, $\frac{\partial E_k}{\partial\hat{y}_j^k} = \frac{\partial(\frac{1}{2}\sum_{j=1}^{j=l}(y_j^k-\hat{y}_j^k)^2)}{\partial\hat{y}_j^k} = -(y_j^k-\hat{y}_j^k)$
- back to the original equation, finally we get $\Delta\theta_j = -\eta(-(y_j^k-\hat{y}_j^k))\cdot(-\hat{y}_j^k(1-\hat{y}_j^k)) = -\eta\hat{y}_j^k(1-\hat{y}_j^k)(y_j^k-\hat{y}_j^k) = -\eta g_j$, where $g_j = \hat{y}_j^k(1-\hat{y}_j^k)(y_j^k-\hat{y}_j^k)$

How to Calculate the Gradient for All Variables?

Here we go for how to update variable $v_{i,h}$ by chain rule

- $\Delta v_{i,h} = -\eta \frac{\partial E_k}{\partial v_{i,h}} = -\eta \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \cdot \frac{\partial \alpha_h}{\partial v_{i,h}}$
- Notice $b_h = f(\alpha_h - \gamma_h)$, and $f'(z) = f(z)(1 - f(z))$ for sigmoid f(z)
- Similarly, we well get the result by using the same procedure for each part in the equation(We skip the exact steps for derivation but give the result directly). It is $\Delta v_{i,h} = \eta b_h(1 - b_h) \sum_{j=1}^{j=l} w_{h,j} g_j x_i$, where $g_j = \hat{y}_j^k(1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k)$

How to Calculate the Gradient for All Variables?

The final variable, we see how to update variable γ_h by chain rule

- $\Delta\gamma_h = -\eta\frac{\partial E_k}{\partial v_{i,h}} = -\eta\frac{\partial E_k}{\partial \hat{y}_j^k}\cdot\frac{\partial \hat{y}_j^k}{\partial \beta_j}\cdot\frac{\partial \beta_j}{\partial b_h}\cdot\frac{\partial b_h}{\partial \gamma_h}$
- Notice $b_h = f(\alpha_h - \gamma_h)$, and $f^{'}(z) = f(z)(1 - f(z))$ for sigmoid f(z)
- Similarly, we well get the result by using the same procedure for each part in the equation(We skip the exact steps for derivation but give the result directly). It is $\Delta\gamma_h = -\eta b_h(1 - b_h)\sum_{j=1}^{j=l} w_{h,j}g_j$, where $g_j = \hat{y}_j^k(1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k)$

Backpropagation Algorithm for Feedforward Neural Network

After acquiring the way of updating 4 variables, we could give backpropagation algorithm

Update Variables of Backpropagation Algorithm

- $\Delta w_{h,j} = \eta g_j b_h$
- $\Delta \theta_j = -\eta g_j$
- $\Delta v_{i,h} = \eta b_h (1 - b_h) \sum_{j=1}^{j=l} w_{h,j} g_j x_i$
- $\Delta \gamma_h = -\eta b_h (1 - b_h) \sum_{j=1}^{j=l} w_{h,j} g_j$

where $g_j = \hat{y}_j^k (1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k)$

Backpropagation Algorithm for Feedforward Neural Network

```
1: Initialize all variables $w_{h,j}, \theta_j, v_{i,h}, \gamma_h$ with random value
   between [0, 1]
2: **for** t in T **do**
3:     **for** k in M **do**
4:         computing $\hat{y}_j^k$ by current values of $w_{h,j}, \theta_j, v_{i,h}, \gamma_h$
5:         update $w_{h,j}, \theta_j, v_{i,h}, \gamma_h$ by rule: $v = v + \Delta v$
6:         k = k + 1
7:     **end for**
8:     t = t + 1
9: **end for**
```

- $v = v + \Delta v$ is the update rule for all variables $w_{h,j}, \theta_j, v_{i,h}, \gamma_h$
- The 'update part' $\Delta w_{h,j}, \Delta\theta_j, \Delta v_{i,h}, \Delta\gamma_h$ are computed according to the results we had from previous page
- T is the number of iteration

Backpropagation Algorithm in General for Computation

-

Optimization for Neural Network: A Brief Discussion

-

Optimization for Neural Network: A Brief Discussion

-

Why Mini-Batch Gradient Descent Used for Deep Neural Network rather than BGD or SGD?

As we discussed at beginning part of this presentation, people usually use mini-batch gradient descent(Mini-GD) for training deep neural network rather than stochastic gradient descent(SGD) or batch gradient descent(BGD). So, at last of this sub section regarding optimization for neural network, we discuss the reason for doing this

- As we know, there is a parameter called "batch size" in mini-batch gradient descent, N. If $N = 1$ we will get SGD and $N = L$(L stands for the total number of training data samples in data set) we will get BGD
- For example we have 10000 data samples then we usually set batch size = 100 rather than 10000 or 1, why?

Why Mini-Batch Gradient Descent Used for Deep Neural Network rather than BGD or SGD?

- According to the definition of SGD, it is clear to see "SGD could update variables/parameters faster/more frequently"; However the speed of running over(using) all the data samples is pretty slow and it is hard to converge/less stable(Because it is NOT easy to find minimum due to only use one sample per iteration/update)
- On the other hand, BGD is faster on speed of running over(using) all the data samples compared with SGD(because implementing "matrix multiple a matrix contains more than one samples" is NOT time consuming than "matrix multiple vector" in GPU for neural network, so the more samples could be batched together to process the faster of running over all data samples) and easy to converge/more stable; However the slower to update variables/parameters

Why Mini-Batch Gradient Descent Used for Deep Neural Network rather than BGD or SGD?

- Another issue for BGD in neural network is, we really need some kind of random/stochastic in gradient descent to avoid falling into local minimum cause neural network model is NOT convex!! In case we use BGD for deep neural network, it will fall into local minimum extremely fast then can NOT train anymore
- From these advantages and drawbacks of SGD and BGD, we realize we want the stochastic to avoid falling into local minimum and update variables/parameters faster, at same time also want faster speed to look over all samples and better converge from "batch" of BGD. So the best choice is Mini-GD
- **We will discuss optimization methods for deep neural network in detail on the later book circle**

Outline for Section 3

Deep Feed-Forward Neural Network in General

According to the definition of feedforward network in general on page 167 of Ian Goodfellow, Yoshua Bengio, Aaron Courville's book Deep Learning. The general format of feedforward neural network is defined as:

General Format of Feedforward Neural Network

- We might have several functions called $f^{(1)}, f^{(2)}, ..., f^{(N)}$ which are connected in a chain to form $f(\mathbf{x}) = f^{(N)}(f^{(N-1)}(...f^{(2)}((f^{(1)}(\mathbf{x})))))$
- Where the $f^{(1)}$ is the first layer of network, $f^{(2)}$ is the second and so on, the last layer is called as output layer. The layers between first and output layer are called as hidden layers
- The overview length of the chain N is called as depth of the network. If $N = 3$, the network is a non-deep feedforward network. And in case of $N > 3$, the network is a deep feedforward network

How Deep it should be?

- As we discussed last page, if number of hidden layer is more than 1(sometimes more than 3) the neural network is so called "deep neural networks"
- For example "AlexNet" in 2012 has 8 layers to achieve error rate 16.4%; "GoogleNet" in 2014 has 22 layers to achieve error rate 6.7% and "ResidualNet" in 2015 has 152 layers(NOT fully connected) to achieve error rate 3.57% [Check detail in the Machine Learning Course by Hung-Yi Lee, National Taiwan University]

How Deep it should be?

- In principle, for arbitrary function, we always could find a large enough neural network(e.g. neural network has only one hidden layer and very large amount of neurons/units) to fit quite well. However, it is usually too hard to find such neural network and might to hard to train
- In common, the deeper neural network is, the less neurons/units we need for each hidden layer and better performance on generalization(Easy to avoid overfitting), but it is harder to train

How to Decide Structure/Architecture of Deep Neural Network?

Compared with classical learning technologies, e.g. SVM, logistic regression, etc. Deep neural network's hypothesis(mathematical model) is NOT given.

- In the other words, classical learning technologies usually have the mathematical framework of the model and we just need to train the model to decide what is values for variables/parameters. However for deep neural network, we have two things are unknown: the exact mathematical framework of the model and variables/parameters. We need to design/decide the structure(how it looks like) of the deep neural network first(to decide mathematical framework of model), then based on the decided structure to train to get the values of variables/parameters
- So the question is, how to decide/design the structure of deep neural network?

How to Decide Structure/Architecture of Deep Neural Network?

- The short answer in general is "try out several times"
- We need to divide data set into 3 parts, training data set, validation data set and prediction data set
- Design different structure of neural networks and use validation data set to see which one is better on error rate after training it by training data set, then apply the best one on prediction data set to check whether it works pretty
- Design structure of neural network needs "experience" + "intuitive" + "experiment"

Deep Learning v.s. Classical Learning

As we discussed, Deep neural network's hypothesis(mathematical model) is NOT given and we need to design/experiment bit.

- But deep neural network has an advantage which is unlike classical learning technologies, e.g. SVM, logistic regression, etc. That is, we do NOT need to do the feature extract/selection(feature engineering) before applying deep neural network. We could use the original data, e.g. a picture with 64x64 pixel to train the deep neural network directly without needing any pre-processing to extract feature of figure
- Classical learning needs feature extract/selection(feature engineering) but they do NOT need to design hypothesis(mathematical framework of model)

Deep Learning v.s. Classical Learning

- So deep learning converts the problem of feature extract/selection(feature engineering) from classical learning to problem of designing hypothesis(mathematical framework of model)
- Question: feature extract/selection or designing hypothesis, Which one is easier?
- For the problem which human can NOT explain how to do it logically, e.g. recognize mouse in the figures, designing hypothesis is easier than feature extract/selection. It means deep neural network is usually advanced in such task

What Tasks Deep Feedforward Learning Could Do? An Example

What does Feedforward Neural Network Do on Each Layer?

Actually the mathematics in each layer does, are amplify, reduce/shirk, translation and twist. Combining together all of these operations on one layer, the total behavior on one layer is trying mapping into a higher dimension space(same as Kernel SVM does) by using amplify, reduce/shirk, translation and twist. Here ▸ Link in Jianguo Yu's book we could find detailed discussion(in Chinese, sorry)

-

What does Feedforward Neural Network Do in Total?

Outline for Section 4

Regularization for Deep Feed-Forward Neural Network

-
-

Course and Tutorial References

- Geoff Hinton, Neural Networks for Machine Learning Course, University of Toronto
- Hung-Yi Lee, Machine Learning Course, National Taiwan University
- Hsuan-Tien Lin, Machine Learning Technique Course, National Taiwan University
- Jianguo Yu, Super Intelligence ▸ Link
- John A. Bullinaria, Introduction to Neural Computation/Introduction to Neural Networks Course, The University of Birmingham
- Metin Akay, Neural Network and Computation Course, Dartmouth College
- Shenlong Wang, Optimization for Machine Learning ▸ Link
- Jihun Hamm, Optimization for Machine Learning ▸ Link
- John Duchi, Introduction to Convex Optimization for Machine Learning ▸ Link

Book and Other References

- Sebastian Raschka, Python Machine Learning Learning
- Zhihua Zhou, Machine Learning(Chinese Version)
- Simon Haykin, Neural Network-A Comprehensive Foundation
- Simon Haykin, Neural Network and Learning Machine
- Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning ▸ Link
- Michael Nielsen, Neural Network and Deep Learning ▸ Link
- Nathan Srebro, Ambuj Tewari, Stochastic Optimization for Machine Learning
- Wikipedia: Artificial Neural Network ▸ Link
- Wikipedia: Feedforward Neural Network ▸ Link
- Wikipedia: Radial Basis Function Network ▸ Link

Question?