Improve Deep Learning Neural Network - Course 2 of Deep Learning - Andrew Ng

Contents

1 Introduction **1**

2 Week 1 **1**
- 2.1 Train/Dev/Test sets . . . 1
- 2.2 Basic recipe for bias and variance problem . . . 1
- 2.3 Regularization . . . 2
- 2.4 Dropout(be regularization) . . . 2
- 2.5 Other regularization . . . 2
- 2.6 Vanishing and exploding . . . 2

3 Week 2 **3**
- 3.1 Mini batch . . . 3
- 3.2 Batch and stochastic gradient . . . 3
- 3.3 Exponentially Weighted Averages and Gradient with Momentum . . . 3
  - 3.3.1 Exponentially Weighted Averages . . . 3
  - 3.3.2 Momentum . . . 3

4 Week 3 **4**
- 4.1 Hyperparameter tuning . . . 4
- 4.2 Batch Normalization . . . 5

1 Introduction

- Content is gotten by course 2 of Deep Learning (coursera) of Andrew Ng to review knowledge

2 Week 1

2.1 Train/Dev/Test sets

- Train set: to train models
- Dev set: to fine tune hyperparameters
- Test set: to evaluate, measure model's performance
- Note: make sure dev set and test set come from same distribution

2.2 Basic recipe for bias and variance problem

- We all know the trade-off between bias and variance. During training process, we always find a way to balance them
- This is a recipe, you can refer:
  - Firstly, we should make a model become high variance by using a deep(large) neural network

- Secondly, we can use apply mothods to decrease variance: get more data, use regularization.

2.3 Regularization

- Simply add the sum of all elements of W to the loss function
- Ex:

$$J = loss(ypred, ytrue) + \frac{\lambda}{2m}\sum_{l=1}^{L} FrobeniusNorm(W)$$

  - When backpropagation

$$dW[l] = backpro + \frac{\lambda}{m}W[l]$$

2.4 Dropout(be regularization)

- There are many type of dropout. Common: Inverted Dropout. "Inverted" maybe in this technique have a step inverted ?? :D
- "keep_prob": probability that a given unit will be kept, Otherwise, it will be eliminated
- Ex: Suppose we want to apply dropout into hidden layer 3. This is a way:
  - ex: keep_prob = 0.8
  - d3 = np.random.rand(a3.shape[0],a3.shape[1]) < keep_prob
    - np.random.rand will give number ranging [0,1]. If the number < 0.8, it returns 1 and once multi that unit will be kept
  - a3 = np.multiply(a3,d3)
  - a3 /= keep_prob
    - Divide by keep_prob to get the expected value of a3 even if some of its units are eliminated. Because of this, it is called "inverted".
- Note that during testing, process will not apply dropout
- Why does it work? Let take the course :D

2.5 Other regularization

- Data augmentation

_ Early stopping

- When training a model, we should try to optimize loss function as small as possible. Then, it can bring over-fitting and we will apply regularization methods. But early stopping do not follow that pipeline, that is a downside of early stopping.

2.6 Vanishing and exploding

- To avoid this problem, we should carefully initialize W
- There is a way to address it which is to set variance of W to be 1 over n.

```
W[l] = np.random.randn(shape) * np.sqrt(1/n)
```

3 Week 2

3.1 Mini batch

- ...
- 1 epoch pass through training set

3.2 Batch and stochastic gradient

- Batch gradient descent: size = m (whole training set)
- Stochastic gradient descent: size = 1
- There are some factor to choose whether batch and stochastic. But I think here are important things to make you give decision:
  - Update: Stochastic helps you to update more frequent then adapt more faster than batch which has to train whole training set before it updates
  - Vectorization: However when you train one example per time, you can not make the most of GPU's performance. That means you can not train in parallel
- It makes sense to choose reasonably the size for the mini-batch. In practice, people usually choose the size of the mini-batch to be the space of the disk(2,4,8,16,64,...2^i)

3.3 Exponentially Weighted Averages and Gradient with Momentum

3.3.1 Exponentially Weighted Averages

3.3.2 Momentum

- Based on EWA idea.
- Understood simply that we need to use average of some gradients to update parameters

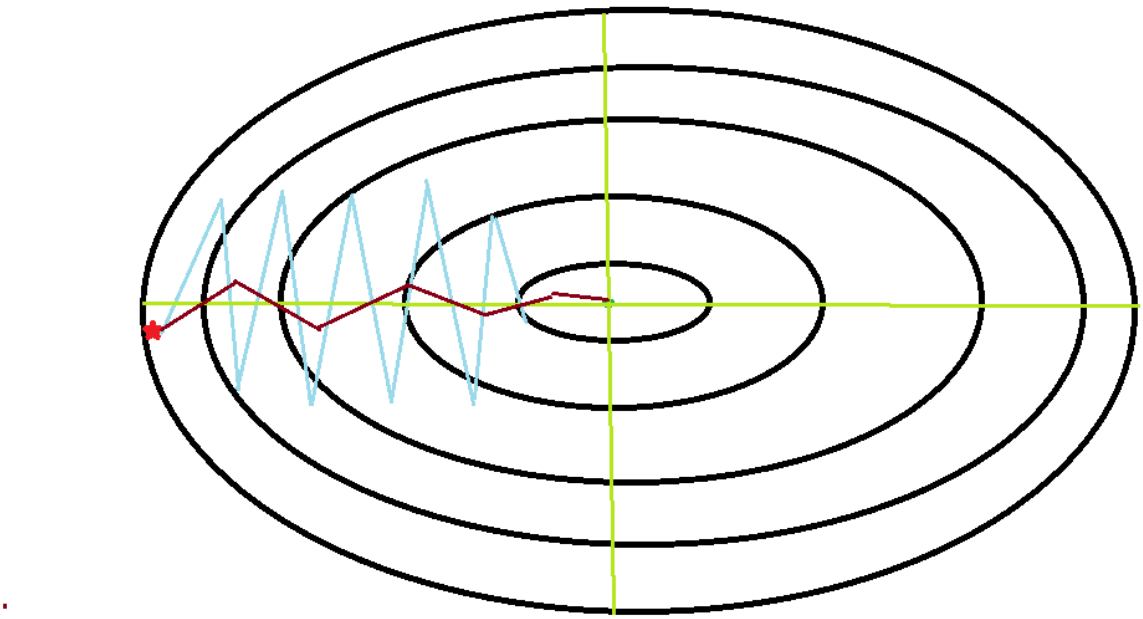

Figure 1: Gradient

- I am not sure why it is useful. Looking at a picture, we all know that, in the center, we have a minimum point where both the vertical and horizontal gradients are zero. So I put O(0,0) in there. Looking at the gradients, that's not good because we don't want it to move vertically that much. It's best to move straight to the minimum by moving horizontally a lot. When we get the average of some of these gradients, the vertical will cancel each other out because the vertical gradient is sometimes negative and sometimes positive. So the vertical average will be small and the horizontal average will be the opposite because it only moves in one direction to the minimum point.
- Hopefully, after applying momentum we will obtain red line instead of blue line
- Implement:

  +V_dW = βV_dw + (1-β)dW

  +V_db = βV_db + (1-β)db

  +W = W - αV_dW , b = b - αV_db

  - with $\frac{1}{1-\beta}$: gradients recently

4 Week 3

4.1 Hyperparameter tuning

- There are many hyperparameters to tune: the number of layer, the number of unit each layer, learning_rate, mini_batch size, beta(momentum,adam,..), epsilon(adam,..)
- Try random values, do not use a grid
- Coarse to fine: understood that initially we explore the large region and then pick a smaller region to continually explore
- Scale for hyperparameters
  - with L(the number of layer) : might be reasonable when we sampling uniformly at random to tune
    - ex: we can search with L = 1 , 2 , 3, 4
  - But with learning_ratem, it might be wrong.

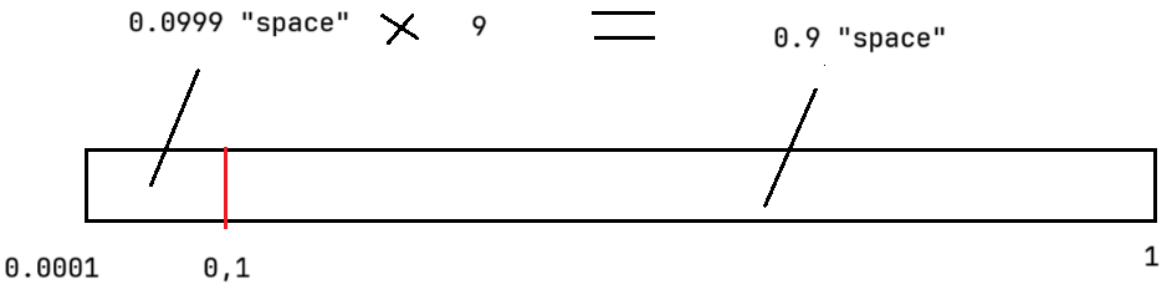

Figure 2: learning_rate_space

- Normally, searching for learning in the range [0.001:0.1] can be useful. But when we apply uniform sampling as shown in the figure, it will search for learning_rate in the range [0.1:1] more than 9 times. It is not reasonable
- Instead of that, we can use log scale

```
r = -4 * np.random.rand() ( r in range [0:-4])
learning_rate = 10 ** r
```

- There are two way to training model: babysiting one model and training models in parallel

4.2 Batch Normalization

- How to implement:
  - $\mu = \frac{1}{m} * \sum_i z_i$
  - $\sigma^2 = \frac{1}{m} * \sum_i (z_i - \mu)^2$
  - $z_i norm = \frac{z_i - \mu}{\sqrt{(\sigma^2 + \epsilon)}}$
  - Now, we normalize z with mean 0 and variance 1. But we do not want to be z close to 0 because it makes gradient slow
  - $z_i = \gamma z_i norm + \beta$, with γ and β are learnable parameters and it decide mean and variance of z
- Why it work?: it brings output with stable distribution make algorithms faster and stable
- In test time, we do not have example to compute μ and σ. Hence, we use exponentially weighted average with mini batch during training