How to train neural network

Yun-Hsiang Chan

June 2021

Neural Network Training Loop, See Figure 1.

I. Initialization of Parameters

- Idea 1: Constant Initialization
Result: identical gradients, identical neurons, bad!

- Idea 2: Random weights, to break symmetry
Too **large** of initialization $\Rightarrow$ **exploding gradient**
Too **small** of initialization $\Rightarrow$ **vanisihing gradient**

Two Popular schemes:
1. **Xavier** Init: For $Tanh$ activation

$$W^{[l]} \sim N(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}}), b^{[l]} = 0$$

or,

$$W^{[l]} \sim N(0, \frac{2}{n^{[l-1]} + n^{[l]}}), b^{[l]} = 0$$

2. **Kaiming He** Init: For **ReLU** activation

$$W^{[l]} \sim N(\mu = 0, \sigma^2 = \frac{2}{n^{[l-1]}}), b^{[l]} = 0$$

Note: $n^{[l]}$ is the number of neurons in a layer, l is layer

1.1 Xavier Initialization Intuition

1. **Variance** of activation approximately **constant** across every layer

$$Var(a^l) = Var(a^{l-1})$$

2. **Mean** of the activation and weights = **zero**

$$E[w^l] = 0, E[a^l] = 0$$

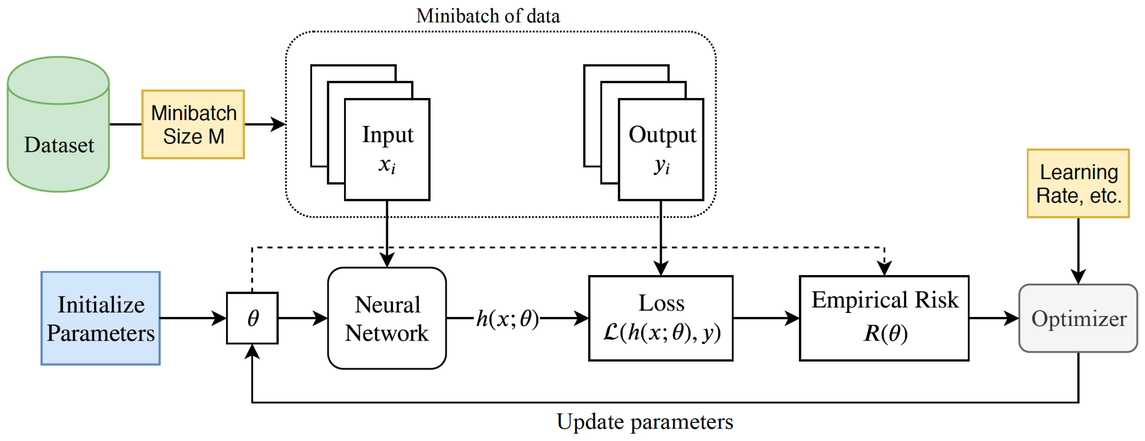

Figure 1: Neural Network Training Loop

II. Batch Normalization Layer

Can we compensate for bad initializations in some other ways?

BatchNorm's Idea
- Explicitly normalize the activations of each layer to be unit Gaussian
- Apply immediately after fully connected/con layers and before non-lineraities
- Learn an additional scale and shift and running statisitcs for test time
- Significantly speeds up training in practice
- Distribution after connected layer is much more stable with BN, reducing the "internal covariate shift", i.e. the change in the distribution of network activations due to the change in network parameters during training.

III. Optimization

Given a training set, we have prediction function and loss function. We want to find the parameters that minimize the empirical risk.

- Batch Grdient Desccent

- Stochastic Gradient Descent

- Gradient Descent with Momentum
Pros: acclerate learning by accumulating some "velocity/momentum" with the past gradients

- Nesterov Accerlated Gradient
Pros: look into the future to see how much momentum is required.

IV. Learning Rate Schedulers

We adjust the learning rate in our learning process.

- Adagrad
Intuition: It increases the learning rate for more sparse features and decreases the learning rate for less sparse ones, according to the history of the gradient.

- RMSprop/Adadelta
Intuition: Unlike Adagrad, the denominator places a significant weight on the most recent gradient. This also helps avoid decreasing learning rate too much.

- Adam
The most popular way, combining Ada and RMSprop.

V. Learning Rate Schedule

- A good learning rate should not be too big or too small.

Find a decay schedule for it.
- Step Decay
- Exponential Decay
- 1/t decay

VI. Batch Size

Typically, small batches are power of 2: $32, 64, 128, 256, 512$

Large Batch Size has:
- Fewer Frequency of updates
- More accurate gradient
- More paralleization efficiency
- May hurt generalization

VII. Hyperparameter Tuning

Grid search and Random search.