Lecture Notes:- 15a

Roshan kumar[1] and Akansksha singh[2]

[1]MS SPCOM(20204417)
[2]PHD EE(20204262)

Abstract

Optimization schemes such as gradient descent and RMSprop have been discussed in this lecture along with hyperparameter tuning and ways to find the best learning rate and Data visualization.

1 Optimization Schemes

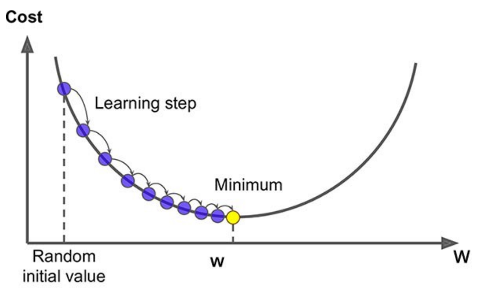
In optimization we train the model iteratively that results in a maximum and minimum function evaluation, on comparing the results in every iteration by changing the hyper-parameters in each step until we reach the optimum results we create an accurate model with less error rate. Gradient descent in one such optimization algorithm that we will study next.

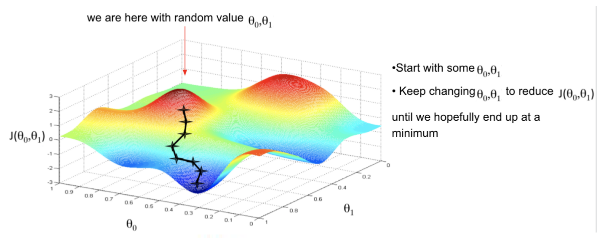
2 Gradient Descent

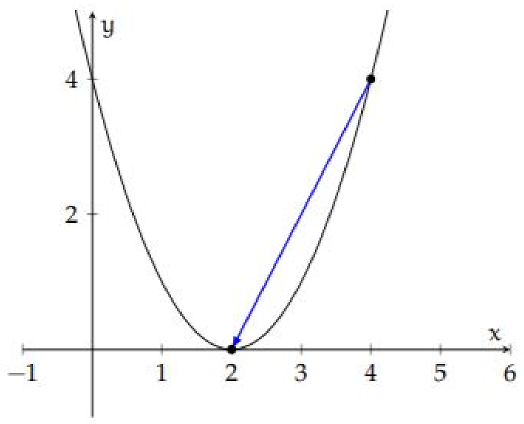
It is an optimizing algorithm used in Machine and deep Learning algorithm as to minimize the the objective function to find the optimal $\theta^* = \arg min_\theta J(\theta)$. One mostly used function in machine learning paradigm is to minimizes the loss function $J(\theta)$ dependent on the weights θ and the goal of gradient descent is to find the weight that minimizes the loss function through iteratively updating weight.To use gradient descent the function must be differentiable and if function is convex for any desired accuracy η there is some ϵ such that gradient descent will converge to θ within η of the optimum. The steps involved are as :

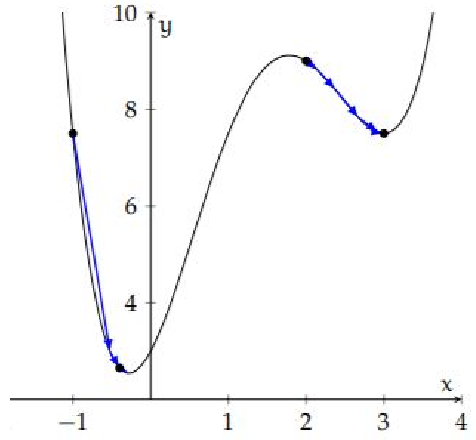
Gradient Descent$(\theta_{init}, \eta, f, f^{'}, \epsilon)$	
1-D parameter $\theta : 1 * 1$	N-D parameter $\theta : N * 1$
$\theta^{(0)} = \theta^{(init)}$:Randomly initialize values	$\theta^{(0)} = \theta^{(init)}$:Randomly initialize values
t=0	t=0
return	**return**
t=t+1	t=t+1
$\theta^{(t)} = \theta^{(t-1)} - \eta f^{'}\left(\theta^{(t-1)}\right)$:Update values	$\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta f(\theta^{(t-1)})$:Update values
until $\left\lvert f\left(\theta^{(t)}\right) - f\left(\theta^{(t-1)}\right)\right\rvert < \epsilon$	**until** $\left\lvert f\left(\theta^{(t)}\right) - f\left(\theta^{(t-1)}\right)\right\rvert < \epsilon$
return$\theta^{(t)}$	**return**$\theta^{(t)}$
$\eta : stepsize$	$\eta : stepsize$
	$\nabla_\theta f = \left[\frac{\partial f}{\partial \theta_1} \ldots\ldots \frac{\partial f}{\partial \theta_N}\right]^T$

The algorithm terminates when the change in the function f is sufficiently small,but their are other reasonable ways to decide to terminate such as.
• Stop after a fixed numbers of iteration T,i.e when t=T.
• Stop when the change in the value of the parameter is sufficiently small,i.e when $\left\lvert\theta^{(t)} - \theta^{(t-1)}\right\rvert < \epsilon$
• Stop when the derivative $f^{'}$at the latest value of Θ is sufficiently small, i.e. when $\left\lvert f^{'}\left(\theta^{(t)}\right)\right\rvert < \epsilon$









• If J is convex, for any desired accuracyϵ, there is some step size ηsuch that gradient descent will converge to within ϵ of the optimal θ.

•If J is non-convex, where gradient descent converges to depends on θ_{init}. When it reaches a value of θ where $f\prime(\theta) = 0$ and $f\prime\prime(\theta) > 0$, but it is not a minimum of the function, it is called a local minimum or local optimum θ.

2.1 Gradient Descent Strategies

2.1.1 Batch Gradient Descent

• (Vanilla gradient descent)batch gradient descent, computes the gradient of the cost function w.r.t. to the parameters for the entire training data-set:

$$\theta = \theta\eta - \nabla_\theta J(\theta) \tag{1}$$

•We have to calculate the gradients for the whole data-set to perform just one update.
• Batch gradient descent is slow and intractable for data-sets that do not fit in memory.
• It also does not allow us to update our model online, i.e. with new examples on-the-fly.

When we update the parameters in the direction of the gradients with the learning rate determining how big of an update to perform. Batch gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

2.1.2 Stochastic Gradient Descent

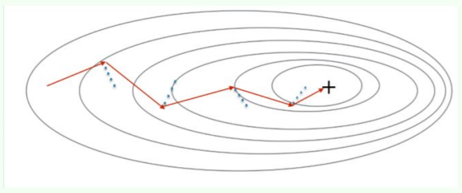
When the form of the gradient is a sum, rather than take one big(ish) step in the direction of the gradient, we can, instead, randomly select one term of the sum, and take a very The word “stochastic”means probabilistic,or random; so does “aleatoric,”. Look up aleatoric music,sometime.small step in that direction.would average out to the same direction as the big step if you were to stay in one place. Of course, you’re not staying in that place, so you move, in expectation, in the direction of the gradient.

```
$\theta^{(0)} = \theta^{(init)}$
for t= 1 : T
randomly select $i \in \{1, 2, 3..., n\}$
$\theta^{(t)} = \theta^{(t-1)} - \eta\nabla_\theta f_i(\theta^{(t-1)})$
return$\theta^{(\mathbf{t})}$
```

Most objective functions in machine learning can end up being written as a sum over data points, in which case, stochastic gradient descent (SGD) is implemented by picking a data point randomly out of the data set, computing the gradient as if there were only that one point in the data set, and taking a small step in the negative direction.

2.2 Momentum

It is an extension to the gradient descent optimization algorithm that allows the search to build inertia in a direction in the search space and overcome the oscillations of noisy gradients and coast across flat spots of the search space.In this we try to "average" recent gradient updates, so that if they have been bouncing back and forth in some direction, we take out that component of the motion. A problem with the gradient descent algorithm is that the progression of the search can bounce around the search space based on the gradient. For example, the search may progress downhill towards the minima, but during this progression, it may move in another direction, even uphill, depending on the gradient of specific points (sets of parameters) encountered during the search.This can slow down the progress of the search, especially for those optimization problems where the broader trend or shape of the search space is more useful than specific gradients along the way. Momentum involves adding an additional hyper-parameter that controls the amount of history (momentum) to include in the update equation, i.e. the step to a new point in the search space.



The red arrows show the update after one step of mini-batch gradient descent with momentum. The blue points show the direction of the gradient with respect to the mini-batch at each step. Momentum smooths the path taken towards the local minimum and leads to faster convergence.

$$\gamma \text{ : momentum constant}$$
$$\vartheta_t \text{ : is the new weight update done at iteration t}$$
$$\vartheta_0 = 0$$
$$\vartheta_t = \gamma\vartheta_{t-1} + \eta\nabla_\theta J(\theta_{t-1})$$
$$\theta_t = \theta_{t-1} - \vartheta_t \text{ :New update}$$

2.3 RMS PROP

In rprop one combines the idea of only using the sign of the gradient with the idea of adapting the step size separately for each weight.It is equivalent to using the gradient but also dividing by the size of the gradient. RMS prop is a mini-batch version of rprop ,where one keeps moving average of the squared gradient for each weight.And then we divide the gradient by square root the mean square,Which is why it's called RMSprop(root mean square).
$MeanSquare(\theta, t) = 0.9 MeanSquare(\theta, t1) + 0.1 \left(\frac{\partial E}{\partial \theta^{(t)}}\right)^2$
As one sees from the above equation we adapt learning rate by dividing by the root of squared gradient, but since we only have the estimate of the gradient on the current mini-batch, we need instead to use the moving average of it. Default value for the moving average parameter that one uses in projects is 0.9. RMSprop is good, fast and very popular optimizer to train the artificial neural netwrok. This implementation of RMS prop uses plain momentum, not Nesterov momentum.

2.4 Hyper-parameter Tuning

Parameters expressing important properties of the model such as its complexity or how fast it should learn and whose values are set before the beginning of learning process are Hyper-parameters.Examples of hyper-parameters are-Model architecture, layers in neural networks=,neurons per layer,etc.

Choosing a set of optimal hyper-parameters for a learning algorithm is called hyper-parameter tuning. The two methods involved in hyper-parameter tuning are- 1. Grid Search 2.Random Search

2.5 Need Of Hyper-parameter Tuning

A machine learning algorithm needs different constraints to identify the patterns present in the dataset. It might not be feasible to train a machine learning model with default parameters for all kinds of data present in the dataset.Thus, it is essential to select the best parameter for an algorithm as it determines its learning process and performance. With the help of hyperparameter tuning, we choose the best parameter for an algorithm so that model gives a good prediction and performs well enough to solve a problem.

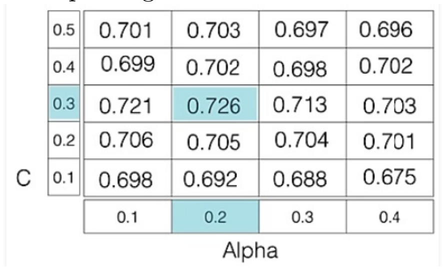
2.6 Methods Of Hyper-parameter Tuning

Being a traditional way to perform hyper-parameter optimization it works by searching exhaustively through a specified subset of hyper-parameters. Even though it definitely finds optimal combination of parameters supplied it can be computationally very expensive.

2.6.1 Grid Search

Being a traditional way to perform hyper-parameter optimization it works by searching exhaustively through a specified subset of hyper-parameters. Even though it definitely finds optimal combination of parameters supplied it can be computationally very expensive.

Example of grid search



C				
0.5	0.701	0.703	0.697	0.696
0.4	0.699	0.702	0.698	0.702
0.3	0.721	0.726	0.713	0.703
0.2	0.706	0.705	0.704	0.701
0.1	0.698	0.692	0.688	0.675
	0.1	0.2	0.3	0.4
	Alpha			

As shown in the image, for C = [0.1, 0.2, 0.3, 0.4, 0.5] and Alpha = [0.1, 0.2, 0.3, 0.4]. For a combination C=0.3 and Alpha=0.2, performance score comes out to be 0.726(Highest), therefore it is selected.

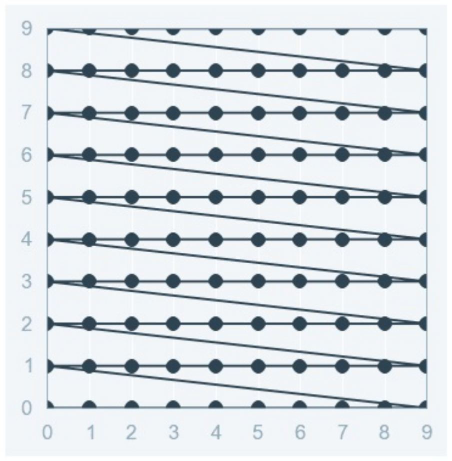
2.6.2 Random Search

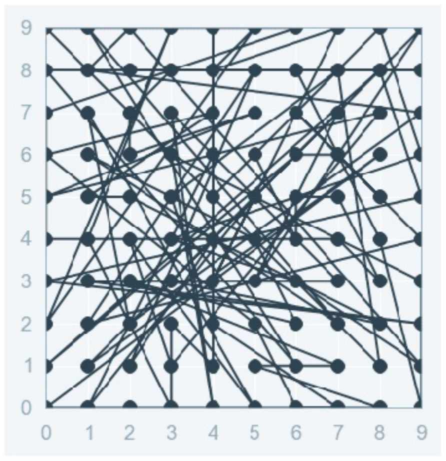
Instead of exhaustive search it performs random search on the specified subset of hyper-parameters. Thus, decreasing the processing time to provide a relatively good solution easily. They are useful for many ill-structured global optimization problems with continuous and/or discrete variables. In contrast to deterministic methods, which typically guarantee asymptotic convergence to the optimum, random search algorithms ensure convergence in probability. However there's no guarantee of finding an optimal combination of parameters.

GenericRandomSearchAlgorithm

Step 0. Initialize algorithm parameters 0, initial points X0 S and iteration index k = 0.
Step 1. Generate a collection of candidate points Vk+1 S according to a specific generator and associated sampling distribution.
Step 2. Update Xk+1 based on the candidate points Vk+1, previous iterates and algorithmic parameters. Also update algorithm parameters k+1.
Step 3. If a stopping criterion is met, stop. Otherwise increment k and return to Step 1

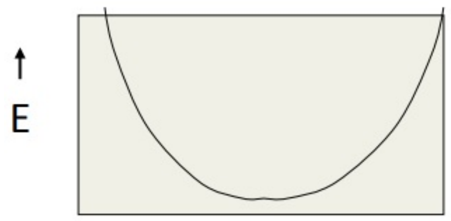
2.7 Visual description of Grid and Random search



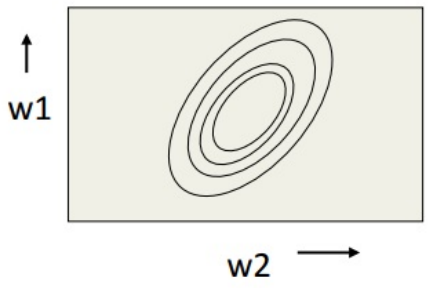


2.8 Learning rate

2.8.1 The error surface for a linear neuron



The error surface is in space represented by weights on x axis and error on y axis. It is a quadratic bowl for a linear neuron with squared error whose vertical cross-section are parabolas and horizontal



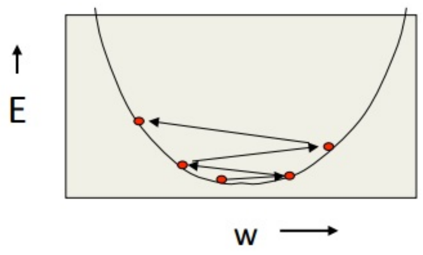
cross-section are ellipses.

2.8.2 Convergence speed of full batch learning when the error surface is a quadratic bowl

The error is reduced while going downhill,but the direction of steepest descent does not point at the minimum unless the ellipse is a circle. The direction in which we want to travel a small distance has greater gradient than in the direction of larger distance.

2.8.3 How the learning goes wrong

When the learning rate is big, the weights slosh to and fro across the ravine leading to diverging oscillation for too big learning rate. What we want is-
1) To move quickly in directions with small but consistent gradients.
2) To move slowly in directions with big but inconsistent gradients.
And towards the end of learning reduce the learning rate.This has analogy in Newtonian mechanics.



2.8.4 Ways to find best learning rate

Bayesian and heuristic approaches help in finding the best learning rate for a model. However, one has to be cautious to not optimize it on test set and not to overfit on training set. Thus, using a validation set. Here's an example. Which λ will you choose from the table?

Better learning rate

E$_v alidation$	Hyper-parameter	E$_t rain$	E$_t est$
0.49	λ1	0.3	0.5
0.61	λ2	0.4	0.6
0.51	λ3	0.2	0.52

Here we will choose λ1 instead of λ3 because of low validation error.
λ1,λ2,λ3 are choosen by any of the methods hyper-parameter tuning.

3 Validation

Proper validation techniques helps in understanding the model and estimate an unbiased generalization performance.It is used for efficient implementation :-

1. Use separate machine
2. Reduce data size
3. Reduce frequency of chunks

4 Stopping Criterion

1. When training error stops decreasing (threshold), but thresholding validation error is a better idea.
2. To monitor N epochs.

One should not judge on one epoch, waiting for few epochs to overcome plateaus in dataset is a better option.

5 Weight Initialization

5.0.1 Random Weights

$$h_{i1} = \sum w_{i1}, w_{i0}x_0$$

If

$$w_{i1} = w_{i1}^{'}$$

Then, hidden units having exactly the same bias and exactly the same incoming and outgoing weights will always get exactly the same gradient.

-Thus,never learning different features.

-The symmetry is broken by initializing the weights to have small random values.

If started with a very big learning rate ,the weights of each hidden unit becomes huge positive or negative and the error derivatives for them becomes small resulting in non-decremental errors,which is a plateau mistaken for local minima.

5.0.2 Fan-in

Fan-in means number of input to neurons. For a hidden unit with big fan-in small changes on many of its incoming weights causes the learning to overshoot. So, when the fan-in is big we want smaller incoming weights. Therefore,initializing the weights to be proportional to $\sqrt{fan-in}$. Learning rate can also be initialized in the same way.

6 Data Normalization

Normalization techniques decreases model's training time by a huge factor also gradient descent converges much faster with feature scaling than without it.

6.1 Advantages Of Normalization

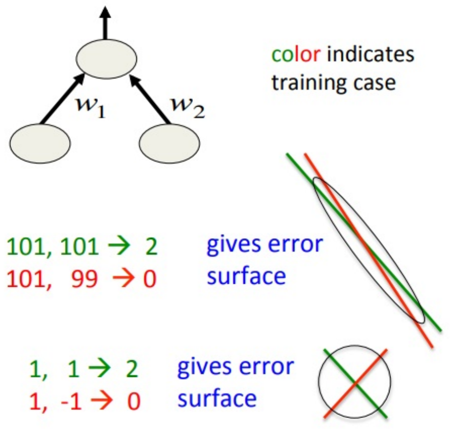
1. Since some of the features have higher numerical value ,to make the network unbiased(to higher value features) we normalize the features to bring parity in their contributions.

2.To improve the training,data normalization reduces the internal co-variate shift.

6.2 Ways of Normalizing

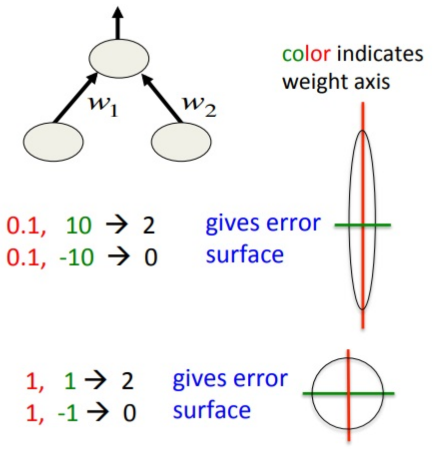
6.2.1 Shifting the inputs

1. It helps to transform each component of the input vector so that it has zero mean over the whole training set.



6.2.2 Scaling the inputs

1.It helps to transform each component of the input vector so that it has unit variance over the whole



training set.

7 Conclusion

In conclusion we have described optimization schemes such as gradient descent and hyper-parameter tuning ,momentum method,rms-prop and other ways. Optimizing techniques tie together the loss function and model parameters by updating the model in response to the output of the loss function. Overall they shape and mold the model into its most accurate possible form by dealing with the weights.