Aditya Ganji

TE Comps A Div

60004190007

DJS SYNAPSE WEEKLY TASK - TASK 5

1.Write a paragraph on loss functions.

Loss Functions:

In a supervised deep learning context the loss function measures the quality of a particular set of parameters based on how well the output of the network agrees with the ground truth labels in the training data.

Loss function = cost function = objective function = error function

Loss functions for regression:

Regression involves predicting a specific value that is continuous in nature. Estimating the price of a house or predicting stock prices are examples of regression because one works towards building a model that would predict a real-valued quantity.

1.Mean Absolute Error (MAE):

Mean Absolute Error (also called L1 loss) is one of the most simple yet robust loss functions used for regression models.

Regression problems may have variables that are not strictly Gaussian in nature due to the presence of outliers (values that are very different from the rest of the data). Mean Absolute Error would be an ideal option in such cases because it does not take into account the direction of the outliers (unrealistically high positive or negative values).

As the name suggests, MAE takes the average sum of the absolute differences between the actual and the predicted values. For a data point x_i and its predicted value y_i, n being the total number of data points in the dataset, the mean absolute error is defined as:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

2.Mean Squared Error (MSE):

Mean Squared Error (also called L2 loss) is almost every data scientist's preference when it comes to loss functions for regression. This is because most variables can be modeled into a Gaussian distribution.

Mean Squared Error is the average of the squared differences between the actual and the predicted values. For a data point Y_i and its predicted value $\hat{Y}_i$, where n is the total number of data points in the dataset, the mean squared error is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

3.Mean Bias Error (MBE):

Mean Bias Error is used to calculate the average bias in the model. Bias, in a nutshell, is overestimating or underestimating a parameter. Corrective measures can be taken to reduce the bias post-evaluating the model using MBE.

Mean Bias Error takes the actual difference between the target and the predicted value, and not the absolute difference. One has to be cautious as the positive and the negative errors could cancel each other out, which is why it is one of the lesser-used loss functions.

The formula of Mean Bias Error is:

$$MBE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n}$$

Where y_i is the true value, $\hat{y}_i$ is the predicted value and 'n' is the total number of data points in the dataset.

4.Mean Squared Logarithmic Error (MSLE):

Sometimes, one may not want to penalize the model too much for predicting unscaled quantities directly. Relaxing the penalty on huge differences can be done with the help of Mean Squared Logarithmic Error.

Calculating the Mean Squared Logarithmic Error is the same as Mean Squared Error, except the natural logarithm of the predicted values is used rather than the actual values.

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}(\log(Y_i) - \log(\hat{Y}_i))^2$$

Where y_i is the true value, $\hat{y}_i$ is the predicted value and 'n' is the total number of data points in the dataset.

5.Huber Loss:

A comparison between L1 and L2 loss yields the following results:

1. L1 loss is more robust than its counterpart.

On taking a closer look at the formulas, one can observe that if the difference between the predicted and the actual value is high, L2 loss magnifies the effect when compared to L1. Since L2 succumbs to outliers, L1 loss function is the more robust loss function.

2. L1 loss is less stable than L2 loss.

Since L1 loss deals with the difference in distances, a small horizontal change can lead to the regression line jumping a large amount. Such an effect taking place across multiple iterations would lead to a significant change in the slope between iterations.

On the other hand, MSE ensures the regression line moves lightly for a small adjustment in the data point.

Huber Loss combines the robustness of L1 with the stability of L2, essentially the best of L1 and L2 losses. For huge errors, it is linear and for small errors, it is quadratic in nature.

Huber Loss is characterized by the parameter delta (δ). For a prediction f(x) of the data point y, with the characterizing parameter δ, Huber Loss is formulated as:

Quadratic

$$L_\delta = \begin{cases} \frac{1}{2}\left(y - f(x)\right)^2, & if\ |y - f(x)| \le \delta \\ \delta |y - f(x)| - \frac{1}{2}\delta^2, & otherwise \end{cases}$$

Linear

Loss functions for classification:

Classification problems involve predicting a discrete class output. It involves dividing the dataset into different and unique classes based on different parameters so that a new and unseen record can be put into one of the classes.

A mail can be classified as a spam or not a spam and a person's dietary preferences can be put in one of three categories - vegetarian, non-vegetarian and vegan. Let's take a look at loss functions that can be used for classification problems.

1.Binary Cross Entropy Loss:

This is the most common loss function used for classification problems that have two classes. The word "entropy", seemingly out-of-place, has a statistical interpretation.

Entropy is the measure of randomness in the information being processed, and cross entropy is a measure of the difference of the randomness between two random variables.

If the divergence of the predicted probability from the actual label increases, the cross-entropy loss increases. Going by this, predicting a probability of .011 when the actual observation label is 1 would result in a high loss value. In an ideal situation, a "perfect" model would have a log loss of 0. Looking at the loss function would make things even clearer -

$$J = -\sum_{i=1}^{N} y_i \log(h_\theta(x_i)) + (1 - y_i)\log(1 - h_\theta(x_i))$$

Where y_i is the true label and $h_\theta(x_i)$ is the predicted value post hypothesis.

Since binary classification means the classes take either 0 or 1, if $y_i = 0$, that term ceases to exist and if $y_i = 1$, the $(1-y_i)$ term becomes 0.

2.Categorical Cross Entropy Loss:

Categorical Cross Entropy loss is essentially Binary Cross Entropy Loss expanded to multiple classes. One requirement when categorical cross entropy loss function is used is that the labels should be one-hot encoded.

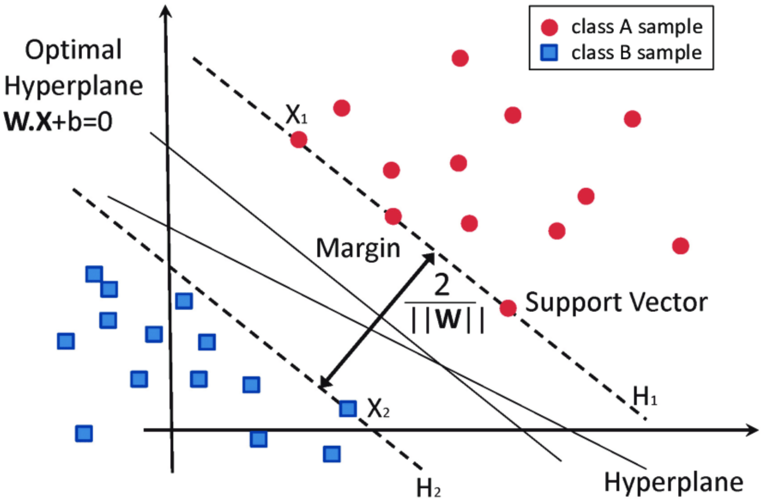
This way, only one element will be non-zero as other elements in the vector would be multiplied by zero. This property is extended to an activation function called softmax, more of which can be found in this article.

3.Hinge Loss:

Another commonly used loss function for classification is the hinge loss. Hinge loss is primarily developed for support vector machines for calculating the maximum margin from the hyperplane to the classes.

Loss functions penalize wrong predictions and does not do so for the right predictions. So, the score of the target label should be greater than the sum of all the incorrect labels by a margin of (at the least) one.

This margin is the maximum margin from the hyperplane to the data points, which is why hinge loss is preferred for SVMs. The following image clears the air on what a hyperplane and maximum margin is:



The mathematical formulation of hinge loss is as follows:

$$SVMLoss = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

Where s_j is the true value and s_{yi} is the predicted value.

Hinge Loss is also extended to Squared Hinge Loss Error and Categorical Hinge Loss Error.

4.Kullback Leibler Divergence Loss (KL Loss):

Kullback Leibler Divergence Loss is a measure of how a distribution varies from a reference distribution (or a baseline distribution). A Kullback Leibler Divergence Loss of zero means that both the probability distributions are identical.

The number of information lost in the predicted distribution is used as a measure. The KL Divergence of a distribution P(x) from Q(x) is given by:

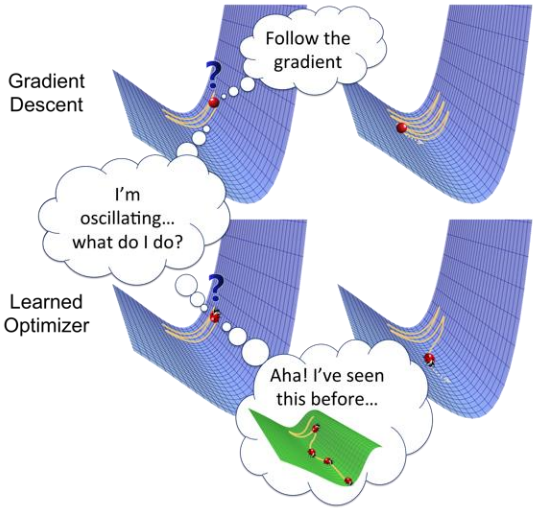
$$D_{KL}(P||Q) = \begin{cases} -\sum_{x} P(x).log\frac{Q(x)}{P(x)} = \sum_{x} P(x).log\frac{P(x)}{Q(x)}, & \textit{for discrete distributions} \\ -\int P(x).log\frac{Q(x)}{P(x)}.dx = \int P(x).log\frac{P(x)}{Q(x)}.dx, & \textit{for continuous distributions} \end{cases}$$

Although picking a loss function is not given much importance and overlooked, one must understand that there is no one-size-fits-all and choosing a loss function is as important as choosing the right machine learning model for the problem in hand.

2. Write a paragraph on different optimizers.

Optimizers:

Many people may be using optimizers while training the neural network without knowing that the method is known as optimization. Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.


Gradient Descent
Follow the gradient
I'm oscillating... what do I do?
Learned Optimizer
Aha! I've seen this before...

Optimizers help to get results faster:

Optimization algorithms or strategies are responsible for reducing the losses and to provide the most accurate results possible.

Different types of optimizers and their advantages:

1.Gradient Descent:

Gradient Descent is the most basic but most used optimization algorithm. It's used heavily in linear regression and classification algorithms. Backpropagation in neural networks also uses a gradient descent algorithm.

Gradient descent is a first-order optimization algorithm which is dependent on the first order derivative of a loss function. It calculates that which way the weights should be altered so that the function can reach a minima. Through backpropagation, the loss is transferred from one layer to another and the model's parameters also known as weights are modified depending on the losses so that the loss can be minimized.

algorithm: $\theta = \theta - \alpha \cdot \nabla J(\theta)$

Advantages:

1. Easy computation.
2. Easy to implement.
3. Easy to understand.

Disadvantages:

1. May trap at local minima.
2. Weights are changed after calculating gradient on the whole dataset. So, if the dataset is too large than this may take years to converge to the minima.
3. Requires large memory to calculate gradient on the whole dataset.

2.Stochastic Gradient Descent:

It's a variant of Gradient Descent. It tries to update the model's parameters more frequently. In this, the model parameters are altered after computation of loss on each training example. So, if the dataset contains 1000 rows SGD will update the model parameters 1000 times in one cycle of dataset instead of one time as in Gradient Descent.

$$\theta = \theta - \alpha \cdot \nabla J(\theta; x(i); y(i))$$

where $\{x(i), y(i)\}$ are the training examples.

As the model parameters are frequently updated parameters have high variance and fluctuations in loss functions at different intensities.

Advantages:

1. Frequent updates of model parameters hence, converges in less time.
2. Requires less memory as no need to store values of loss functions.
3. May get new minima's.

Disadvantages:

1. High variance in model parameters.
2. May shoot even after achieving global minima.
3. To get the same convergence as gradient descent needs to slowly reduce the value of learning rate.

3.Mini-Batch Gradient Descent:

It's best among all the variations of gradient descent algorithms. It is an improvement on both SGD and standard gradient descent. It updates the model parameters after every batch. So, the dataset is divided into various batches and after every batch, the parameters are updated.

$$\theta=\theta-\alpha\cdot\nabla J(\theta;\ B(i))$$

where {B(i)} are the batches of training examples.

Advantages:

1. Frequently updates the model parameters and also has less variance.
2. Requires medium amount of memory.

All types of Gradient Descent have some challenges:

1. Choosing an optimum value of the learning rate. If the learning rate is too small than gradient descent may take ages to converge.
2. Have a constant learning rate for all the parameters. There may be some parameters which we may not want to change at the same rate.
3. May get trapped at local minima.

4.Momentum:

Momentum was invented for reducing high variance in SGD and softens the convergence. It accelerates the convergence towards the relevant direction and reduces the fluctuation to the irrelevant direction. One more hyperparameter is used in this method known as momentum symbolized by 'γ'.

$$V(t)=\gamma V(t-1)+\alpha.\nabla J(\theta)$$

Now, the weights are updated by $\theta=\theta-V(t)$.

The momentum term γ is usually set to 0.9 or a similar value.

Advantages:

1. Reduces the oscillations and high variance of the parameters.

2. Converges faster than gradient descent.

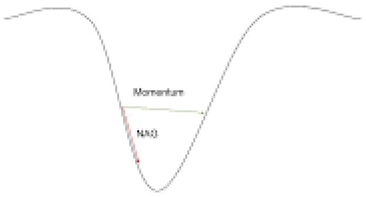
Disadvantages:

1. One more hyper-parameter is added which needs to be selected manually and accurately.

5.Nesterov Accelerated Gradient:

Momentum may be a good method but if the momentum is too high the algorithm may miss the local minima and may continue to rise up. So, to resolve this issue the NAG algorithm was developed. It is a look ahead method. We know we'll be using $\gamma V(t-1)$ for modifying the weights so, $\theta-\gamma V(t-1)$ approximately tells us the future location. Now, we'll calculate the cost based on this future parameter rather than the current one.

$V(t)=\gamma V(t-1)+\alpha.\ \nabla J(\ \theta-\gamma V(t-1)\ )$ and then update the parameters using $\theta=\theta-V(t)$.



NAG vs momentum at local minima

Advantages:

1. Does not miss the local minima.
2. Slows if minima's are occurring.

Disadvantages:

1. Still, the hyperparameter needs to be selected manually.

6.Adagrad:

One of the disadvantages of all the optimizers explained is that the learning rate is constant for all parameters and for each cycle. This optimizer changes the learning rate. It changes the learning rate 'η' for each parameter and at every time step 't'. It's a type second order optimization algorithm. It works on the derivative of an error function.

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}),$$

A derivative of loss function for given parameters at a given time t.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}.$$

Update parameters for given input i and at time/iteration t

η is a learning rate which is modified for given parameter θ(i) at a given time based on previous gradients calculated for given parameter θ(i).

We store the sum of the squares of the gradients w.r.t. θ(i) up to time step t, while ϵ is a smoothing term that avoids division by zero (usually on the order of 1e−8). Interestingly, without the square root operation, the algorithm performs much worse.

It makes big updates for less frequent parameters and a small step for frequent parameters.

Advantages:

1. Learning rate changes for each training parameter.
2. Don’t need to manually tune the learning rate.
3. Able to train on sparse data.

Disadvantages:

1. Computationally expensive as a need to calculate the second order derivative.
2. The learning rate is always decreasing results in slow training.

7.AdaDelta:

It is an extension of AdaGrad which tends to remove the *decaying learning Rate* problem of it. Instead of accumulating all previously squared gradients, *Adadelta* limits the window of accumulated past gradients to some fixed size w. In this exponentially moving average is used rather than the sum of all the gradients.

$$E[g^2](t)=\gamma.E[g^2](t-1)+(1-\gamma).g^2(t)$$

We set γ to a similar value as the momentum term, around 0.9.

$$\mathbb{E}[g^2]_t = \gamma \mathbb{E}[g^2]_{t-1} + (1-\gamma) g_t^2,$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\mathbb{E}[g^2]_t + \epsilon}} \cdot g_t.$$

Update the parameters

Advantages:

1. Now the learning rate does not decay and the training does not stop.

Disadvantages:

1. Computationally expensive.

8.Adam:

Adam (Adaptive Moment Estimation) works with momentums of first and second order. The intuition behind the Adam is that we don’t want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for a careful search. In addition to storing an exponentially decaying average of past squared gradients like AdaDelta, *Adam* also keeps an exponentially decaying average of past gradients M(t).

M(t) and V(t) are values of the first moment which is the *Mean* and the second moment which is the *uncentered variance* of the gradients respectively.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}.$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}.$$

First and second order of momentum

Here, we are taking mean of M(t) and V(t) so that E[m(t)] can be equal to E[g(t)] where, E[f(x)] is an expected value of f(x).

To update the parameter:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t.$$

Update the parameters

The values for β1 is 0.9 , 0.999 for β2, and (10 x exp(-8)) for ‘ϵ’.

Advantages:

1. The method is too fast and converges rapidly.
2. Rectifies vanishing learning rate, high variance.

Disadvantages:

1. Computationally costly.

Adam is the best optimizers. If one wants to train the neural network in less time and more efficiently than Adam is the optimizer. For sparse data use the optimizers with dynamic learning rate.If, want to use gradient descent algorithm than min-batch gradient descent is the best option.