A20442409



# CS-577 ASSIGNMENT - 3

## THEORITICAL QUESTIONS:

### Loss

1. Equations for $L_1$, $L_2$ and huber loss, cosh loss

$L_1$: $L_1(\theta) = \sum_{j=1}^{k} | \hat{y}_j^{(i)} - y_j^{(i)} |$

$L_2$: $L_2(\theta) = \sum_{j=1}^{k} ( \hat{y}_j^{(i)} - y_j^{(i)} )^2$

Huber loss: $\rho(d) = \begin{cases} \frac{1}{2} d^2 & \text{if } |d| \leq \delta \\ \delta(d - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$

$\downarrow$

$L_i(\theta) = \sum_{j=1}^{k} \rho_\delta ( \hat{y}^{(i)} - y^{(i)} )$

log cosh loss: $L_i(\theta) = \sum_{j=1}^{k} \log ( \cosh ( \hat{y}^{(i)} - y^{(i)} ) )$

$\log(\cosh(d)) \approx \begin{cases} \frac{d^2}{2} & \text{if } d \text{ is small} \\ |d| - \log 2 & \text{otherwise} \end{cases}$

$L_1$ loss is generally used in regression problems where we are certain that there aren't many outliers. This loss function is not sensitive to outliers. Median is its optimal predicted values.

$L_2$ loss provides the sum of squared errors and is affected by large error values. This loss is comparitively more sensitive to outliers than the $L_1$ loss and mean is the optimal prediction in this loss.

Huber loss is used as a loss function for robust regression. This loss handles outliers the best. The curve is more flatter compared to $L_2$ loss.

log(cosh) loss: A smoother loss function compared to $L_2$ used for regression problems. The curve behaves differently for small ~~delta~~ d & large d, representing inliers and outliers respectively.

2. Maximizing the ~~log~~ likelihood function:

$$L(\theta) = \prod_{i=1}^{m} \prod_{j=1}^{k} \left( p(y=j|x^{(i)}) \right)^{y_j^{(i)}}$$

Minimizing the log likelihood of":

$$\ell(\theta) = -\log L(\theta) = -\sum_{i=1}^{m} \sum_{j=1}^{k} \log \left( p(y=j|x^{(i)}) \right)$$

$$= -\sum_{i=1}^{m} \sum_{j=1}^{k} y_j^{(i)} \log(\hat{y}_j^{(i)})$$

For a random assignment of k class $p(y=j|x^{(i)}) = \frac{1}{k} = \log(k)$. If the cross entropy loss becomes greater than $\log(k)$ this loss is considered to be the worst loss

3. Softmax loss $\ell(\theta) = -\sum_{i=1}^{m} \sum_{j=1}^{k} y_j^{(i)} \log(\hat{y}_j^{(i)})$.

When multiclass classification needs to be performed we use • softmax loss to perform the updation of weights. Gradients of the loss are calculated in order to manipulate the initial weights to achieve multiclass classification.

4. Kullback - Liebler loss :

$$L(\theta) = -\sum_{i=1}^{m}\sum_{j=1}^{k} y_j^{(i)} \log\left(\frac{y_j^{(i)}}{\hat{y}_j^{(i)}}\right)$$

log likelyhood loss ↑

K.L. loss $L_j(\theta) = \underbrace{\sum_{j=1}^{k}\left(y_j^{(i)} \log y_j^{(i)}\right)}_{\text{- entropy}} - \underbrace{\sum_{j=1}^{k}\left(y_j^{(i)} \log \hat{y}_j^{(i)}\right)}_{\text{cross entropy.}}$

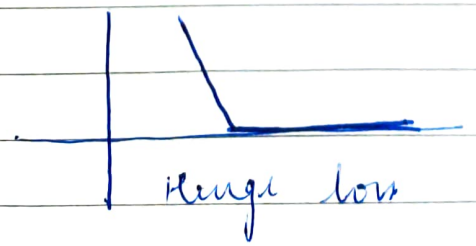
Given identically independent samples (IID) $\{x_i\}_{i=1}^{n}$ and distributions $P(x)$ and $q(x)$, $P(x)$ is the true distribution of labels and $q(x)$ being the predicted distributions, Kullback - Leibler loss measures the similarity of these distributions.

log likelyhood for the entire distribution being

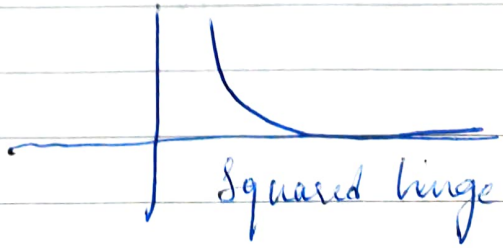
$$\sum_{i=1}^{m} \log\left(\frac{p(x_i)}{q(x_i)}\right) \begin{cases} >0 & p(x) \text{ is better method} \\ <0 & q(x) \text{ is better method.} \end{cases}$$

5. Hinge loss : $L_i(\theta) = \underbrace{\sum_{j \neq t} \max\left(0, \hat{y}_j^{(i)} - \hat{y}_t^{(i)} + 1\right)}_{\text{sum of incorrect class labels.}}$

Squared hinge loss : $L_i(\theta) = \frac{1}{2}\sum_{i=1}^{k} \max\left(0, \hat{y}_i^{(i)} - y_t^{(i)} + 1\right)^2$



Hinge loss



Squared hinge loss.

Both of the above terms add a penalty to the terms that are misclassified. Here the main idea is to add the terms that are incorrectly classified. The term $\hat{y}_j^{(i)} - \hat{y}_t^{(i)} + 1$ will be positive if there is incorrect classification and negative when there is correct classification.

6.

| | $x^{(1)}$ | $x^{(2)}$ | $x^{(3)}$ |
|---|---|---|---|
| $\hat{y}^{(1)}$ | 0.5 | 0.4 | 0.3 |
| $\hat{y}^{(2)}$ | 1.3 | 0.8 | -0.6 |
| $\hat{y}^{(3)}$ | 1.4 | -0.4 | 2.7 |

$$L_i(\theta) = \sum_{j \neq t} \max(0, \hat{y}_j^{(i)} - \hat{y}_t^{(i)} + 1)$$

$$L_1 = \max(0, (1.3 - 0.5) + 1) + \max(0, (0.4 - 0.5 + 1))$$
$$L_1 = 1.8 + 1.9 = 3.7$$
$$L_2 = \max(0, (0.4 - 0.8) + 1) + \max(0, (0.9 - 0.8) + 1)$$
$$L_2 = 0.6 + 0 = 0.6$$
$$L_3 = \max(0, (0.3 - 2.7) + 1) + \max(0, (-0.6 - 2.7 + 1))$$
$$L_3 = 0 + 0 = 0.$$

from $L_1$ and $L_2$ the predictions are better than $L_3$ since the values are much higher than zero.

7. Regularization is performed on a model to yield a simpler model than the original model, this is done because simpler explainations are better the coefficients are made smaller to obtain a model that generalizes better.

$$L(\theta) = \frac{1}{m} \sum_{i=1}^{m} L_i(f(x_i, w), y_i) + \lambda R(\theta)$$

loss Regularization

$L_1$ regularization $R(\theta) = \sum_{i,j} |\theta_{ij}|$.

$L_2$ regularization $R(\theta) = \sum_{i,j} \theta_{ij}^2$

Adding a regularization term to loss results in weight decay.
$L_1$ regularization makes the weights sparse, i.e, concentrate the weight, $L_2$ regularization the weights smaller while spreading.

$\lambda$ should be chosen within a range that its not too high or low, which might lead to the model that is too general for use.

8. we have the gradient terms

$$\frac{\partial L}{\partial \theta} = \frac{\partial}{\partial \theta} \frac{1}{m} \sum_{i=1}^{m} L_i(f(x^{(i)}, \theta), y^{(i)}) + \lambda \frac{\partial}{\partial \theta} R(\theta)$$

L2: Regularization: $R(\theta) = \sum \theta_i^2 : \theta^T\theta \quad \frac{\partial}{\partial \theta} R(\theta) = 2\theta$

during gradient decent subtract $\lambda\theta$ (weight decay).

L1 regularization : $R(\theta) = \sum_i |\theta_i| \Rightarrow \frac{\partial}{\partial \theta} R(\theta) = \lambda \text{sign}(\theta)$
During gradient descent subtract $\lambda \text{sign}(\theta)$.
(add or subtract depending on the sign).

9. We have the predicted output equation
$\hat{y} = \sigma(wx + b)$

Kernal regularization involves regularizing the weights wherein the weights are shrunk
Bias regularization involves in regularizing the bias term where the bias term is shrunk
Activity regularization involves in regularizing both weights and the bias term which in turn reduces $\hat{y}$ as $\hat{y}$ is a combination of both w and b.

## Optimization

1. Numerical computation of gradients is not a feasible option when we deal with non linear systems. During which, we use the backpropagation method to find the gradients using chain rule. Using back propogation we can multiply a chain of partial derivatives to obtain the gradients in a faster way compared to numerical computation of gradients.
Numerical method can be efficient while calculating the gradients of a linear system and an activation function so complex that backpropogation is difficult.

2. Gradient decent involves in calculating the optimal weights for the model designed. The update information of the weights is given by the equation

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta \nabla L(\theta^{(i)})$$

here the gradient of the loss is calculated over the entire dataset. We stop updating the weights when the difference in old and new loss is lower than a threshold.

Stocastic gradient involves in calculating the weights in a similar way as we do in the gradient decent method only difference being the loss captured will be calculated using a randomly selected data point. The update equation is as follows:

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta (x^{(i)T}\theta - y^{(i)}) x^{(i)}$$

Stocastic gradient decent converges quickly as compared with gradient decent method because in SGD loss is calculated over one point and weights are upgraded instead of computing it over all datapoints.

3. While performing SGD, for updation of weights if all the data points are considered then

the resulting update of weight would be more accurate when done with a single data point But performing gradient decent on a single datapoint is faster computationally when compared to performing it over the entire dataset which leads to the concussion, larger the batch size time taken to compute increases with increase in accuracy of the updated weight.
4 main problems with SGD are:
① what should be the learning rate ?
② what happens if loss is more sensitive to one param?
③ How to avoid getting stuck at a local minima ?
④ Minibatch gradient estimates are noisy.

4. Momentum fixes poor conditioning by averaging the gradients over previous gradients :
As far as local minima is concerned (saddle points), momentum reduces the gradient decent with a velocity so that there is no stop in calculating the gradients.
SGD with momentum smoothens changes to the gradients so that the resulting output is not noisy.

5. In momentum the gradient is computed at the old location whereas in NAG the gradient are computed by predicting the values of the gradient.

while computing the value for $\theta^{(i+1)}$ we get

$$\tilde{\theta}^{(i+1)} \leftarrow \tilde{\theta}^{(i)} + V^{(i+1)} + \underbrace{\beta(V^{(i+1)} - V^{(i)})}_{\text{Acceleration}}$$

There is an acceleration term added to the momentum equation which leads to the term Nesterov accelerated Gradient (NAG).

NAG converges faster than SGD + momentum ~~and~~

6. with better results for parameter update.

6. Strategies for learning rate decay.

(i) Step decay:
every k iterations reduce the learning rate by 50% $\eta \rightarrow \eta/2$ over k iterations.

(ii) Exponential decay
decrease the learning rate exponentially over time $\eta = \eta_0 e^{-kt}$.

(iii) fractional decay:
Reduce the ~~decay~~ learning rate using the following equation $\eta = \eta_0 / (1 + kt)$.

7. Newtons method is as follows:
Find x such that $f(x) = 0$ by taking an initial guess of x as $x_0$.
Find the update such that $f(x_0 + \Delta x) = 0$.
Expanding the above equation using taylor

Series expansion

$f(x_0 + \Delta x) = f(x_0) + \Delta x f'(x_0) + \cdots = 0$

$\Rightarrow f(x_0) + \Delta x f'(x_0) = 0 \Rightarrow \Delta x = -\frac{f(x_0)}{f'(x_0)}$ and

continue until $f(x_0 + \Delta x) \neq 0$.

We have the objective to compute $\nabla J(\theta) = 0$ where $\nabla J(\theta)$ is the loss

$\Rightarrow f(x_0 + \Delta x) = f(x_0) + \Delta x^T \nabla f(x) + \cdots = 0$

$\Rightarrow f(x_0) + \Delta x^T \nabla f(x_0) = 0$

$\Rightarrow \nabla J(\theta) + \nabla(\nabla J(\theta_0)) \Delta\theta = 0$

The term $\nabla(\nabla J(\theta_0))$ is known as the hessian matrix. denoted by $H$.

$$H = \begin{bmatrix} \partial J^2/\partial\theta_1\partial\theta_1 & \cdots & \partial J^2/\partial\theta_1\partial\theta_n \\ \vdots & \ddots & \vdots \\ \partial J^2/\partial\theta_n\partial\theta_1 & \cdots & \partial J^2/\partial\theta_n\partial\theta_n \end{bmatrix}$$

$\Rightarrow \nabla J(\theta_0) + H\Delta\theta = 0$

$\Rightarrow \Delta\theta = -H^{-1} \nabla J(\theta_n)$.

As per newton's method

$\theta^{(i+1)} \leftarrow \theta^{(i)} - H^{-1} \nabla J(\theta^{(i)})$

here we use $H^{-1}$ as the learning rate. Hessian matrix is a matrix comprising of all the gradients. We compute $H^{-1}$ to find the learning rate $\eta$.

8. Condition numbers are generally used to describe the sensitivity of the parameters using the singular values of the hessian matrix. Condition number is the ratio of the largest singular values in the hessian matrix to the smallest singular value.

$$\text{condition number} = \frac{S_v \text{ largest}}{S_v \text{ smallest}}$$

In case of poor conditioning the condition number increases

9. In Adagrad we replace $H$ with different precond-itions. This is done as follows

$$B^{(i)} = \text{diag}\left( \sum_{j=1}^{i} \nabla J(\theta^{(i)}) \, \nabla J(\theta^{(j)})^T \right)^{1/2}$$

$$B^{(i)^{-1}} = \begin{bmatrix} \frac{1}{\sqrt{\sum \left(\frac{\partial J}{\partial \theta_1}\right)^2}} & & \\ & \ddots & \\ & & \frac{1}{\sqrt{\sum \left(\frac{\partial J}{\partial \theta_n}\right)^2}} \end{bmatrix}$$

$$\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta \, B^{(i)^{-1}} \nabla J(\theta^{(i)})$$

we we approximate the hessian matrix as follows $\frac{\partial^2 J}{\partial \theta^2} = \sum \left(\frac{\partial J}{\partial \theta_1}\right)^2$

- In adagrad we use elementwise scaling of gradients based on history of gradients due to keeping a large squared sum of past gradients in the denominator in small ~~learning~~ steps.

This problem is fixed by using a decay factor (eq. 0.9) while adding new gradients to the gradient sum.

The $t^{th}$ update looks as follows

$$v_k^{(t+1)} = \left(\rho \sum_{j=1}^{t} \left(g_k^{(j)}\right)^2 + (1-\rho)\left(g_k^{(t)}\right)^2\right)$$

$\uparrow$ $k^{th}$ component of $\nabla J(\theta^{(t)})$

- Adam combines RMS prop with momentum in the following way.

and stand

Let denote the first momentum velocity with exponential:

$$m_1^{(t+1)} = \beta_1 \cdot m_1^{(t)} + (1-\beta_1)\nabla L(\theta^{(t)})$$

$$m_2^{(t+1)} = \beta_2 \cdot m_2^{(t)} + (1-\beta_2)\nabla L(\theta^{(t)}) \odot \nabla L(\theta^{(t)})$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta\, m_1^{(t+1)} \cdot \frac{1}{\sqrt{m_2^{(t+1)}} + \epsilon}$$

Because the momentums are initialized to zero which is dividing by the second momentum we get a large step

$$\hat{m}_1^{(t+1)} = \frac{m_1^{(t)}}{1-\beta_1^t} \qquad \hat{m}_2^{(t+1)} = \frac{m_2^{(t)}}{1-\beta_2^t}$$

Then $\theta^{(t+1)} = \theta^{(t)} - \eta\, \hat{m}_1^{(t+1)} \cdot \Gamma$

12. Gradient descent with line search involves in finding the best step size instead of fixing the first step size.

For a given direction $u = \nabla f(x)$ best step size is given by $\eta^* = \underset{\eta}{\operatorname{argmin}} f(x + \eta u)$ and perform gradient descent $\theta^{(i+1)} \leftarrow \theta^{(i)} - \eta^{(i)} \nabla f(\theta^{(i)})$. Then solve for $\eta^*$ to find explicit computation of the gradient descent or the simple line search.

Bracketing is performed by grouping points into a bracket like $[a, b, c]$ $x = (b+c)/2$
if $f(x) \leq f(b) \Rightarrow [b, x, c]$
else if $f(x) \geq f(b) \Rightarrow [a, b, x]$.
Continue this process to produce smaller and smaller brackets until the brackets are small enough.
An alternate to bracketing is to perform a successive line search: Start with $\theta_0$ and direction set $\{u^{(i)}\}$. Compute $\eta^{(i)} = \underset{\eta}{\operatorname{argmin}} f(\theta^{(i)} + \eta u^{(i)})$
$\theta^{(i+1)} \leftarrow \theta^{(i)} + \eta^{(i)} u^{(i)}$.
The above equation is not as efficient as gradient descent because the step calculations are not as accurate but GD with line search is less heavy computationally and gives faster results.

13. Quasi Newton methods approximate the hessian matrix inverse using gradient evaluations. Although this method is efficient these methods requires large number of datapoints.

Compute quasi newton direction
$\Delta\theta = (-H^{(i-1)})^{-1} \nabla J(\theta^{(i-1)})$
Determine step size $\eta^*$ with bracketing line search and then compute parameter update with $\theta^{(i)} = \theta^{(i-1)} + \eta^* \Delta\theta$
Finally compute updated hessian approximate $H^{(k)}$.

For BFGS update $\Delta\theta = \theta^{(i)} - \theta^{(i-1)}$
$\nabla J = \nabla J(\theta^{(i)}) - \nabla J(\theta^{(i-1)})$
inverse update

$$(H^{(i)})^{-1} = \left(I - \frac{\Delta\theta \Delta J^T}{\Delta J^T \Delta\theta}\right)\left(H^{(i-1)}\right)^{-1}\left(I - \frac{\Delta J \Delta\theta^T}{\Delta J^T \Delta\theta}\right) + \frac{\Delta\theta \Delta\theta^T}{\Delta J^T \Delta\theta}$$

BFGS is comparitively faster than the newton algorithm but less accurate
BFGS is comparitively slower than ADAM but produces more accurate results.

## Regularization:

1. For weight decay we multiply each coefficient by $\gamma \in [0, 1]$. As iterations progress weights that are not reinforced decay to 0 which is equivalent to adding regularization term to the loss function.

2. Early stopping is done by stopping when validation error increases instead of training error stops decreasing. This can be interpreted as $L_2$ ~~loss~~ regularization on weights in loss.

Strategies:

① Reuse validation data - I

Retrain on all data using the number of iterations determined through validation

We need more data so that we can confidently say that the number of iterations computed is the correct number of iterations

② Reuse validation data II

Retrain on all data from previous weights until validation loss is bigger than training loss. The idea here is to check if both training and validation loss converges and hence checking if the loss obtained is correct.

3. To prevent overfitting of data, synthetic data is added to increase variability in training leading to better generalization.

The following tasks can be performed:

① Augment by interpolating b/w examples or by adding noise.

② Augment in feature or data domain.

(3) Augment by ~~interpolating~~ transforming data using crop / rotate / scale / change intensity of images

(4) Popular in image classification to introduce scale / illumination / rotation invariance.

4. At each training stage dropout layers are added to fully connected layers with a probability of $(1-p)$ where 'p' is the hyperparameter.

The removed nodes are reinitialized with the original weights in the subsequent stage

Advantages: Reduce node interactions, node ~~fatt~~ overfitting, reduce dependency on single node, distribute features across multiple nodes and ~~increase~~ ~~training~~ ~~speed~~.

Disadvantages: longer training due to dropout

5. Multiplying the output at each node by p is equivalent to computing expected value for $2^n$ dropped out networks

$$\hat{y} = E_D[f(x, D)] = \int P(D) f(x, D) \, dD$$

During testing all nodes are utilized and hence leads to a higher values. To approximate this, the o/p is multiplied by 0 or $(1-p)$.

6 Batch normalization normalizes the output of the previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

During training of deep neural networks one or many weights value can become high which might affect the flow in the network this is where Batch normalization comes into picture. Here when each batch is normalized at each layer this imbalance in weights are normalized to obtain a stable network.

Normalizing data induces randomness to the model.

Advantages:
1. Activations are not saturated.
2. normalizes values compared for each batch
3. Suitable for back propagation

Diss The model generated can loose information of some key info, the model can get too general.

7. Saturation is required during learning for which we scale and shift after normalization.

$$\{z^{(i)}\}_{i=1}^{q} \longrightarrow \{\hat{z}^{(i)}\}_{i=1}^{q} \longrightarrow \{\tilde{z}^{(i)}\}_{i=1}^{q}$$

$$\tilde{z}_j^{(i)} = \gamma_j \hat{z}_j^{(i)} + \beta_j$$

$\gamma_j$ and $\beta_j$ are first learnt then the network can learn to cancel BN if there is no need for it.

During training we do Batch normalization because batches are random, the added randomness reduces overfitting.

8. Ensemble classifiers train multiple independent models and uses majority vote or average during testing.

Different models generated from ensembling overfit in different ways and hence cancel out each others method of overfitting when a collective decision is made.

Different models can be obtained by
- changing data
- changing parameters.

→ Record multiple snapshots of the model during training.