CS577 - Deep Learning

Assignment 3

Theoretical Questions

Loss

1) Equations for L1, L2, Huber loss and Logcosh loss

L1: $L_i(\Theta) = \sum_{j=1}^{k} |\hat{y}_j^i - y_j^i|$

L2: $L_i(\Theta) = \sum_{j=1}^{k} (\hat{y}_j^{(i)} - y_j^{(i)})^2$

Huber:

$$\rho_\delta(d) = \begin{cases} \frac{1}{2} d^2 ; & |d| \leq \delta \text{ (threshold)} \\ \delta(d - \frac{1}{2}\delta) ; & \text{otherwise} \end{cases}$$

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

ρ_δ ↓ Huber fn.

Logcosh:

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

$$\log(\cosh(d)) = \begin{cases} d^2/2, & \text{if } d \text{ is small} \\ |d| - \log 2, & \text{otherwise.} \end{cases}$$

Comparison:

L1 loss is a common loss that determines the distance between the actual and predicted value.

It finds the difference among them.

L2 loss determines the square of difference between the actual and predicted value.

Huber loss determines this difference and multiplies it quadratically if it is less than a random threshold (or) multiplies them linearly.

Logcosh performs in a similar manner but reduces sensitivity towards outliers.

2) Cross entropy:

Cross entropy loss is given by

$$y_j^{(i)} = P(y=j \mid x^{(i)}) \quad j \in [1, K].$$

$$\text{Likelihood} = L(\Theta) = \prod_{j=1}^{m} \prod_{j=1}^{k} \left(P(y=j \mid x^{(i)}; y_{(j)})\right)$$

Maximizing by log likelihood:

Log Likelihood $\Rightarrow \log(L(\theta))$

$$= -\sum_{j=1}^{m} \sum_{i=i}^{k} y^{(i)} \log\left(P(y=j \mid x^{(j)})\right)$$

The worst case cross entropy occurs for random class assignment

$L(\theta) = \log(k)$ ← bad loss value

$P(y=j \mid x^{(i)}) = 1/k$

3) Softmax loss fn

Softmax loss fn is similar to cross entropy.

It occurs when the last layer is a softmax activation.

Softmax fn $\Rightarrow$ $L(\theta) = \prod_{j=1}^{m} \prod_{j=1}^{k} P(y=j \mid x^{(i)})^{y^{(i)}}$

4) Kullback - Leibler loss fn

This fn is used to find similarities between distributions

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

Let there be two distributions $p(x)$ & $q(x)$ and input sample be $\{x_i\}_1^m$ that are Identically and Independently distributed.

Likelihood dist $\Rightarrow \frac{p(x_1 \ldots x_m)}{q(x_1 \ldots x_m)}$

$$= \prod_{i=1}^{m} \frac{P(x_i)}{q(x_i)} \begin{cases} > 1 : P(x) \Rightarrow \text{better} \\ < 1 : q(x) \Rightarrow \text{better} \end{cases}$$

Similarly the log likelihood is found and solved as

$$\propto \sum_{i=1}^{m} P(x_i) \log\left(\frac{P(x_i)}{q(x_i)}\right) = KL(p \| q).$$

KL = cross entropy:

$$KL(p\|q) = \sum_{i=1}^{m} P(x_i) \log\left(\frac{P(x_i)}{q(x_i)}\right)$$

$$= \underbrace{\sum_{i=1}^{m} p(x_i) \log(P(x_i))}_{-\text{entropy}} - \underbrace{\sum_{i=1}^{m} P(x_i) \log(q(x_i))}_{\text{cross entropy}}$$

If the entropy value remains constant then KL loss = cross entropy.

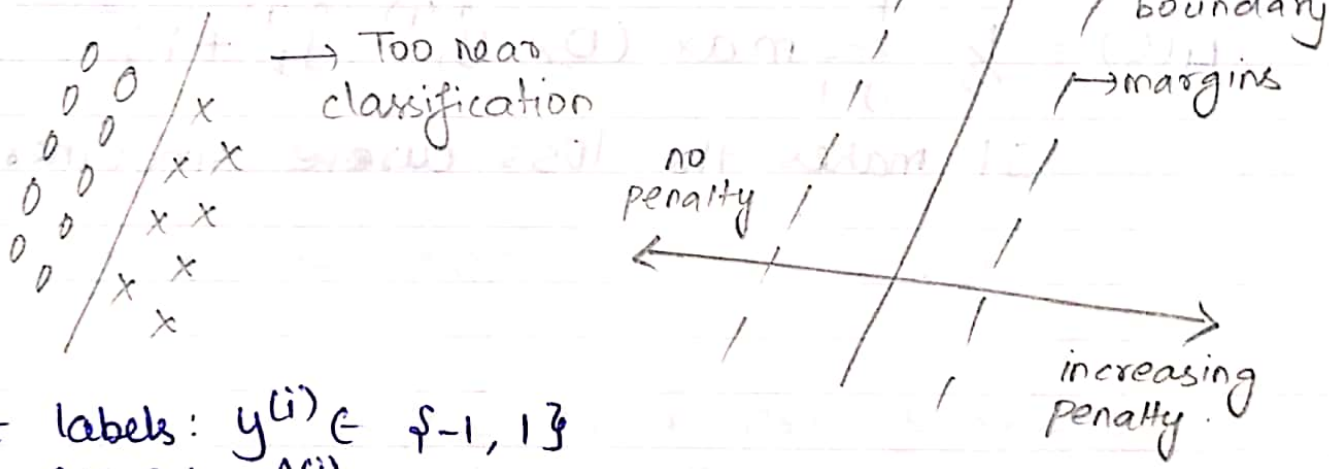
5) Hinge loss:

Hinge loss determines the distance of the distributions from ~~margin~~ decision boundary.

For any given classification, hinge loss adds margins between the values and the decision boundary.

Elements that are classified beyond margins have no penalty but if they are below the margins, penalty is added in proportational to its distance to the decision boundary.

Thus the ultimate aim is to make a far classification.



let labels: $y^{(i)} \in \{-1, 1\}$

score: $\hat{y}^{(i)}$

Margin len = 1

Hinge loss $L_i(\theta) = \max\{0, 1 - y^{(i)}\hat{y}^{(i)}\}$

If there are k classes:

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

Thus Hinge loss strongly enforces on discrimination.

Bad Hinge loss:

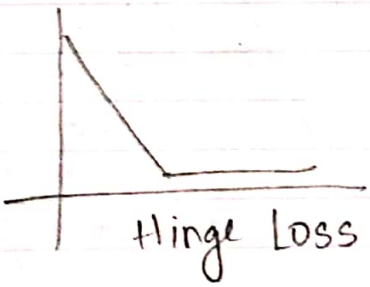
$$y_j^{(i)} \approx 0 \Rightarrow L_i = \max(0, -0 - 0 + 1) + \cdots \max(0, 0 - 0 + 1)$$
$$= k - 1$$

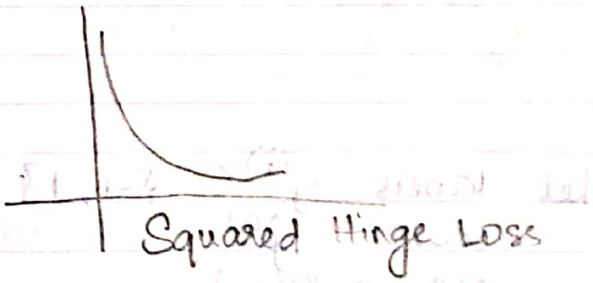
This occurs for random scoring.

Squared Hinge Loss:

$$L_i(\theta) = \frac{1}{2} \sum_{j=1}^{k} \max(0, \hat{y}_j^{(i)} - \hat{y}_t^{(i)} + 1)^2.$$

It makes the loss curve smoother.





6) Given: x^1 x^2 x^3

$y^1 = 1$ $y^2 = 2$ $y^3 = 3$.

$\hat{y}^{(1)} = 0.5 \quad 0.4 \quad 0.3$ $\hat{y}^{(2)} = 1.3 \quad 0.8 \quad -0.6$.

$\hat{y}^{(3)} = 1.4, -0.4, 2.7$.

Sol:

	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
	$y^1 = 1$	$y^2 = 2$	$y^3 = 3$

$L_1 = \max(0, 1.3 - 0.5 + 1) + \max(0, 1.4 - 0.5 + 1)$

$= 1.8 + 1.9 = 3.7$

$L_2 = \max(0, 0.4 - 0.8 + 1) + \max(0, -0.4 - 0.8 + 1)$

$= 0.6 + 0 = 0.6$

$L_3 = \max(0, 0.3 - 2.7 + 1) + \max(0, -0.6 - 2.7 + 1)$

$= 0 + 0 = 0$.

The classification made on class 3 is best.

7) Regularization:

Regularization term is added to loss fn for the purpose of weight decay.

Thus when weights are lowered, a simple network is formed.

It also makes the network stable by reducing the co-eff.

$$L1\ reg = R(\theta) = \sum_{\theta_{ij}} |\theta_{ij}|$$

$$L2\ reg = R(\theta) = \sum_{\theta_{ij}} (\theta_{ij})^2.$$

The overall loss fn is

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

where λ is the weight of the regularization term $R(\theta)$.

It is a hyperparameter and is used to give importance to regularization.

It is necessary to choose an optimal value of λ as higher λ reduces focus on data loss and lower value can degrade the $R(\theta)$ importance.

L1 regularization concentrates on weights but L2 spreads the weights.

8) L1 and L2 in gradients of a n/w:

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

Gradient of loss fn is

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

For L2 $R(\theta) = \sum_i \theta_i^2$

$$\frac{\partial}{\partial \theta} R(\theta) = 2\theta.$$

Thus $2\lambda\theta$ will be subtracted during gradient descent.

For L1 $R(\theta) = \sum_i |\theta_i|$

$$\frac{\partial}{\partial \theta} = \text{sign}(\theta).$$

Thus $\lambda \text{sign}(\theta)$ will be subtracted during grad. descent

9). Kernal, Activity, Bias Regularization:

$$\hat{y} = \sigma(\underset{\text{activity}}{\omega x} + \underset{\text{bias}}{b})$$

Kernal reg. reduces ω. It is preferred mostly for weight decay.

Bias reg. reduces b. when outputs are smaller it is prefered.

Activity reg. reduces both b & ω as well as o/p $\hat{y}$. Since $\hat{y}$ depends on i/p it is not mostly prefered and is chose when o/p is close to 0.

Optimizers:

1) Advantages of back propagation over numerical computing of gradients.

The process of minimizing the loss also known as Optimization is usually done in two methods.

i) Numerically computing gradients:

The gradients of every element of the network is computed as

$$\frac{\partial f}{\partial x_i}(X) = \lim_{h \to 0} \frac{f(x_1, \ldots X+h \ldots x_n) - f(x_1 \ldots x_n)}{h}$$

This method is easy to compile but is very slow. This method can be used for verification (as) as a gradient check on simplified n/w.

2) Back propagation:

Determines simple derivatives of the elements of network and is propagated back.

This causes a weight decay and yields a stable and simple n/w.

2) Difference b/w GD & SGD:

Gradient descent and Stochastic Gradient Descent both use gradients to optimize the network.

In classical gradient descent, ~~there is no~~ gradients are not updated in epoch (or) in batches but in stochastic they are updated in batches.

Also in GD the gradients are calculated from all data points but in SGD it is calculated only from the point it is used in the n/w.

Since SGD computes only the necessary gradient it converges faster but the error rate is not as low as GD.

3) Trade off in selecting batch size of SGD:

As minibatches are ~~to~~ used in SGD it is necessary to select an accurate mini batch size.

This is because it is necessary for the minibatches to cover the input samples with less noise in all the iterations.

Problems of SGD

1) What is the learning rate?

2) What happens if it becomes less/more sensitive to a single parameter?

3) How to avoid getting stuck at local minimum (or) saddle points?

4) Minibatch grad. estimates are noisy.

4) SGD with momentum:

It addresses the problems local minimum and noise.

It smoothes out the changes to gradients using momentum.

$$\begin{cases} v^{(i+1)} \leftarrow \rho v^{(i)} + \nabla L(\theta^{(i)}) \\ \theta^{(i+1)} \leftarrow \theta^{(i)} - \eta v^{(i+1)} \end{cases}$$

$\nabla L(\theta^{(i)})$ - contemporary as it is based on current batch.

$v^{(i)}$ - Stable as it is an average

In ordinary SGD $\rho = 0$.

In SGD + momentum $\rho > 0$

Thus it takes steps in direction of velocity instead of gradient.

Also parameters θ are updated with old param & the learning rate x velocity.

Thus

1) There is a velocity at every saddle pt.

2) Noisy grad. are moved out by averages.

3) Poor conditioning (high sensitivity at some direction) smoothened by averaging precise grad.

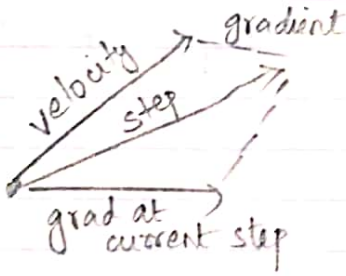
5. NAG vs Simple momentum:

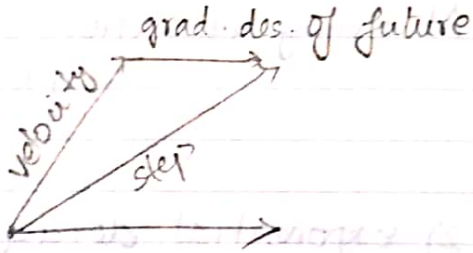
NAG momentum:

$$\begin{cases} v^{(i+1)} \leftarrow \rho v^{(i)} - \eta \nabla L(\theta^{(i)} + \rho v^{(i)}) \\ \theta^{(i+1)} \leftarrow \theta^{(i)} + v^{(i+1)} \end{cases}$$

where $\rho v^{(i)}$ is the corrected grad. descent.

Thus NAG determines the corrected grad. descent at future whereas SGD + momentum includes the current step grad.





$$\tilde{\theta}^{(i)} = \theta^{i} + \rho v^{(i)}$$

$$\theta^{(i+1)} \leftarrow \theta^{i} + v^{(i+1)}$$

$$\tilde{\theta}^{(i+1)} - \rho v^{(i+1)} \leftarrow \tilde{\theta}^{(i)} - \rho v^{(i)} + v^{(i+1)}$$

$$\tilde{\theta}^{(i+1)} \leftarrow \tilde{\theta}^{(i)} - \rho v^{(i)} + (1+\rho) v^{(i+1)}$$

$$\tilde{\theta}^{(i+1)} \leftarrow \tilde{\theta}^{(i)} + v^{(i+1)} + \rho \underbrace{(v^{(i+1)} - v^{(i)})}_{\text{acceleration}}$$

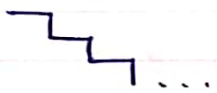
6) Strategies for learning rate deccay:

The learning rate of the optimizers are not fixed. The learning rate should be as low as possible to get an optimized network. This is because the loss becomes high as the iterations go for a high learning rate.

Strategies are:

1) Step decay: let η be learning rate.

At every iteration

$\eta \leftarrow \eta/2 \rightarrow$ 

2) Exponential decay:

$$\eta = \eta \cdot e^{-k/t}$$

where $k \Rightarrow$ decay rate
$t \Rightarrow$ iteration index.

3) Fractional decay:

$$\eta = \eta_0 / (1+kt).$$

7) Learning rate using Newton's Method:

The learning rate is computed by Newton's method as:

A learning rate η is initally fixed.

Find x such that $F(x) = 0$.

Find the update Δx such that $f(x_0 + \Delta x) = 0$. where x_0 is a random guess

Use taylor series of expansion

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

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

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

Continue till $f(x_0 + \Delta x) \neq 0$

In the fn replace $f(x_0)$ by $\nabla J(\theta)$.

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

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

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

$$\Delta\theta = H^{-1}(\nabla J(\theta_0))$$

The Hessian matrix is given by

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

$$= \nabla(\nabla J(\theta_0))$$

8) Condition no.

The sensitivity is delimited using the singular values of Hessian matrix.

The ratio of the singular value to the smallest singular value is known as conditional no.

$$C.N = \frac{SV_1}{SV_m}$$

$SV_1 \leftarrow$ largest SV

$SV_m \leftarrow$ smallest SV.

When conditional no. is high problem is more.

9) Adagrad approx. inverse of Hessian:

Computing Hessian is difficult as it is expensive & noisy.

Replacing it with a different pre condition in adagrad.

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

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

Comp. the inverse:

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

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

For j^{th} comp:

$$\theta_j^{(i+1)} \leftarrow \theta_j^{(i)} - \eta \frac{1}{\sqrt{\sum_{j=1} (\partial J/\partial \theta_j)^2}} \frac{\partial J}{\partial \theta_j}(\theta^{(i)})$$

For each direction element wise scaling is used.

$$\begin{cases} s_j^{(i+1)} = s_j^{(i)} + \| \nabla_j L(\theta^{(i)}) \|^2 \\ \theta_j^{(i+1)} \leftarrow \theta_j^{(i)} - \eta \nabla_j L(\theta^{(i)}) \cdot \frac{1}{\sqrt{s_j^{(i+1)} + \varepsilon}} \end{cases}$$

10. Problem with Adagrad

In adagrad, since elementwise normalization is done, the step size becomes smaller at every iteration.

RMSprop is thus used where a decay factor is added to the new gradients when grad. are summed.

$$\begin{cases} S_j^{(i+1)} = \gamma S_j^{(i)} + (1-\gamma) \| \nabla_j L(\theta^{(i)}) \|^2 \\ \theta_j^{(i+1)} \leftarrow \theta_j^{(i)} - \eta \nabla_j L(\theta^{(i)}) \cdot 1/S_j^{(i+1)} + \varepsilon \end{cases}$$

11. Adam:

Adam combines the properties of Adagrad and RMSprop.

The first moment is

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

Second moment

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

$$\theta^{i+1} = \theta^i - \eta m_1^{(i+1)} \cdot \frac{1}{\sqrt{m_2^{(i+1)} + \varepsilon}}$$

Since moments are initialized by zero, when divided by second step, a large step is obtained

Thus a bias correction term is used so that initial moments are zero.

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

where they are unbiased estimates of m_1, m_2

12. Grad. desc. with line search:

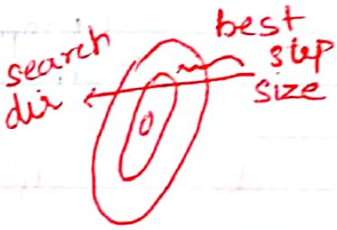
Idea is to find best step size.

Given direction

$$v = \nabla f(x)$$

Best step size:

$$\eta^* = \underset{\eta}{\operatorname{argmin}}\ f(x+\eta v)$$



Grad. desc.

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

η^* is solved by simple line search.

Bracketing:

Given bracket: $[a, b, c]$

$$x = \text{or } \frac{b+c}{2}$$

if $f(x) \leq f(b) \Rightarrow [b, x, c]$

if $f(x) \geq f(b) \Rightarrow [a, b, x]$

Smaller and smaller brackets are continued.

13. Quasi-Newton method:

In this method, the Hessian inverse is approximated using grad. descent. This req. large no. of examples.

Alg:

Compute $\Delta\theta = -(H^{(i-1)})^{-1} \nabla J(\theta^{(i-1)})$

Determined step size η^*

Compute param. updates $\theta^i = \theta^{i-1} + \eta^* \Delta\theta$.

Compute update H approx H^k

Instead of H^k it is easy to compute $(H^k)^{-1}$

BFGS update:

$$\Delta\theta = \theta^i - \theta^{i-1}$$

$$\Delta J = \nabla J(\theta^{(i)}) - \nabla J(\theta^{(i-1)})$$

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

Inverse update: H^i is also done as.

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

This inverse costs $O(n^2)$.

BFGS vs Adam:

BFGS does not work well with minibatches unlike Adam due to inaccuracies.

Regularization:

1. Weight deccay:

At each iteration of weight deccay a coeff $p \in [0,1]$ is multiplied to the weight.

Since this value is <1, it deccays the weight of a parameter and acts like a regularizing term.

2. Early stopping:

Stopping when validation error increases instead of training error reduction.

The search is limited by limiting the weight. Thus it acts like a L2 reg. that adds penalty to weights.

Strategies:

1) Reuse validation data: Retrain on all data using no. of iterations determined from validation

Problem: Is no. of iterations correct?

2) Continue training from prev. data weight on entire data $\Rightarrow$ when val. loss $>$ training loss.

Problem: Is loss value correct?

3) Data augmentation:

It refers to adding synthetic data to increase variability to training.
This gives better generalization.
Extra data (or) feature (interpolation b/w samples (or) noise) is added to improve efficiency.
This is done in Image classification.

4) Dropout:

At each stage, units with prob. (1-p) are dropped in fully connected n/w where p is a hyperparameter.
Removed nodes are reinstalled with original weights.

Procedure:

1) Do not drop out i/p nodes ⇒ loss of data.
2) Do not drop out during testing.
   ↳ expect o/p from all units.
   ↳ high total sum of all o/p.

Adv:

1) Reduced node interaction
2) Overfitting
3) Increase training speed
4) Reduce dependency on single node
5) Dist. features across multiple nodes

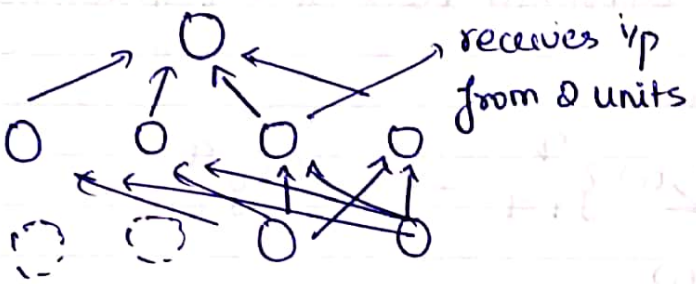
Qus:

Longer training speed.

5) Approximating dropout during testing in dropout

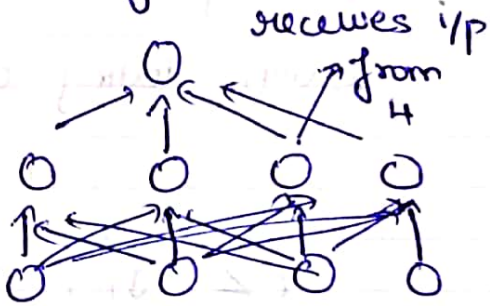
This is done by multiplying p(prob. of dropped units)

Training:



Multiply by 2(p)

Testing



Multiply by ½(½p)

This is equal to computing expected value for 2^n

$$\hat{y} = \underset{\downarrow}{E_D}[f(x, D)] = \int p(D) f(X, D) dD$$

mask for all n nodes.

6) Batch normalization:

In this method, all O/p from hidden layers are normalized.

$$\hat{z}_j^{(i)} = \frac{z_j^{(i)} - \mu_j}{\sigma_j}$$

$$\mu_j = \frac{1}{q} \sum_{i=1}^{m} z_j^{(i)}$$

$$\sigma_j = \left(\frac{1}{q} \sum_{i=1}^{m} (z_j^{(i)} - \mu_j)^2 \right)^{1/2}$$

↓

O/p of j^{th} unit of i^{th} batch.

Adv:

1) Make sure activations are not saturated
2) Computed for each batch in training.
3) Normalization is differentiable.

BN is done during training (or) testing. During training randomness is added during randomness.

During testing o/p is approximated and randomized.

7) Scale & Shift param:

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

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

↳ Learned

Cancelling: if BN not req:

$\gamma_j = \sigma_j$

$\beta_j = \mu_j$

$$\Rightarrow z_j^{(i)} = \sigma_j \hat{z}_j^{(i)} + \mu_j$$

$$= \frac{\sigma_j z_j^{(i)} - \mu_j + \mu_j}{\sigma_j}$$

$$= z_j^{(i)}$$

Uses:

Batches are randomized during training
Randomness added
Reduces overfitting.

Param. initialization is very important. It is initially any value but by regularization wt is moved to zero.

8) Ensemble classifier:

Train multiple indep. models.
Finally choose the best model by majority vote (or) avg. during testing.
This reduces overfitting as each incompletely trained model is combined to form a completely trained model.

Strategies for EC:

Change data
Change parameter
Record multiple parameter of model (diff. learning rate).
Tune different models differently.