1. **Multiple choice (Shreyas)**
   Please pick the correct answers for each questions, note that each question can have one or more than one correct.

   (a) Consider Figure 1 plotting loss values as a function of the number of epochs, select the option that best describe the shaded regions in the plot, and the point where you would stop training to achieve the best generalization.

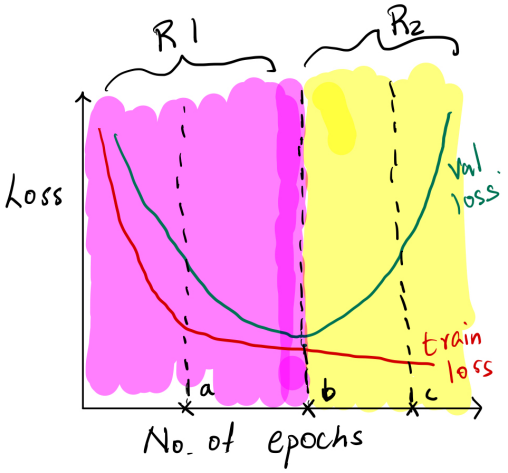


Figure 1

   i. R1: Overfitting, R2: Underfitting, stop at a.
   ii. R1: Overfitting, R2: Underfitting, stop at b.
   iii. R1: Underfitting, R2: Overfitting, stop at b.
   iv. R1: Underfitting, R2: Overfitting, stop at c.

   (b) When we minimize the negative log likelihood for a classification problem with c classes, which of the following are we inherently performing?

   i. Maximizing the likelihood of observing the training data.
   ii. Minimizing the Mean Squared Error.
   iii. Minimizing the Cross Entropy loss.

   (c) Mark all the correct choices regarding cross validation.

   i. A 5-fold cross-validation approach results in 5-different model instances being fitted.

ii. A 5-fold cross-validation approach results in 1 model instance being fitted over and over again 5 times.

iii. A 5-fold cross-validation approach results in 5-different model instances being fitted over and over again 5 times.

iv. None of the above.

(d) Which of the following are considered as hyperparameter choices while training a neural network.

i. Loss Function.

ii. Learning Rate.

iii. Number of Layers.

iv. Batch Size.

v. All of the above.

(e) Assuming Stochastic Gradient Descent (SGD) computes gradient using a single sample from the training data, which of the following statements are true.

i. Gradient computed using SGD will be noisier than gradient computed using Batch Gradient Descent.

ii. Empirically, SGD takes longer (in terms of clock time) to converge than Batch Gradient Descent.

iii. SGD usually avoids the trap of poor local minima.

iv. SGD is computationally more expensive than Batch Gradient Descent.

2. **Short answer (Kaifeng)**

(a) Please explain the difference between batchnormalization during training and testing.

(b) Your friend designed a novel activation function:

$$f(x) = x^3 \tag{1}$$

Please discuss if this is a good idea to use this activation in a neural network.

(c) Your friend is utilizing a Multi-layer Perceptron (MLP) for a deep learning task and is trying to increase the number of units within each layer to enhance the model's complexity. Please explain potential effect of this action on the model performance.

(d) Please explain the role of ℓ_1 regularization.

(e) Please explain the role of the bias correction step in the Adam optimizer.

3. **Backpropagation in parallel neural network (Tonmoy)**

A parallel neural network consists of twin networks which accept distinct inputs but share the same weights. The outputs of the twin networks are later processed by more hidden layers. Let's assume we have a parallel neural network with the following architecture:

$$
\begin{aligned}
h_p &= W_1 x_p^{(i)} + b_1 \\
z_1 &= ReLU(h_p) \\
h_q &= W_1 x_q^{(i)} + b_1 \\
z_2 &= ReLU(h_q) \\
z &= z_1 - z_2 \\
z_3 &= W_2 z + b_2 \\
\hat{y}^{(i)} &= \sigma(z_3) \\
L^{(i)} &= L_{CE}(y^{(i)}, \hat{y}^{(i)}) \\
L &= -\frac{1}{m} \sum_{i=1}^{m} L^{(i)}
\end{aligned}
$$

In the above architecture, ($x_p^{(i)}$,$x_q^{(i)}$) represent the pair of i^{th} input example and are each of shape D_x. $y^{(i)}$ represent the label of the i^{th} input example and is a scalar. We also assume z_1 and z_2 have shape of D_z.

(a) Draw the computational graph for the parallel neural network described above. You can start from $L^{(i)}$ as your output variable and then backtrack to the input variables $x_p^{(i)}$ and $x_q^{(i)}$.

(b) Compute $\nabla_{\hat{y}^{(i)}} L^{(i)}$ and denote it as $\delta_{\hat{y}^{(i)}}$. For all the following parts, you can refer to this computed gradient as $\delta_{\hat{y}^{(i)}}$.

(c) Compute $\nabla_{z_3} L^{(i)}$ and denote it as δ_{z_3}. For all the following parts, you can refer to this computed gradient as δ_{z_3}.

(d) Compute $\nabla_{b_2} L^{(i)}$ and denote it as δ_{b_2}. For all the following parts, you can refer to this computed gradient as δ_{b_2}.

(e) Compute $\nabla_{W_2} L^{(i)}$ and denote it as δ_{W_2}. For all the following parts, you can refer to this computed gradient as δ_{W_2}.

(f) Compute $\nabla_{z} L^{(i)}$ and denote it as δ_{z}. For all the following parts, you can refer to this computed gradient as δ_{z}.

(g) Compute $\nabla_{z_1} L^{(i)}$ and denote it as δ_{z_1}. For all the following parts, you can refer to this computed gradient as δ_{z_1}.

(h) Compute $\nabla_{z_2} L^{(i)}$ and denote it as δ_{z_2}. For all the following parts, you can refer to this computed gradient as δ_{z_2}.

(i) Compute $\nabla_{h_q} L^{(i)}$ and denote it as δ_{h_q}. For all the following parts, you can refer to this computed gradient as δ_{h_q}.

(j) Compute $\nabla_{h_p} L^{(i)}$ and denote it as δ_{h_p}. For all the following parts, you can refer to this computed gradient as δ_{h_p}.

(k) Compute $\nabla_{b_1} L^{(i)}$.

(l) Compute $\nabla_{W_1} L^{(i)}$.

4. **Regularization techniques (Yang)**

   (a) True or False: Regularization is intended to reduce training error but not validation error.

   (b) Consider a model $\tilde{\mathcal{L}}(\theta) = \mathcal{L}(\theta; \mathbf{X}, \mathbf{y}) + \alpha\Omega(\theta)$ where $\mathcal{L}(\theta; \mathbf{X}, \mathbf{y})$ is some loss function and $\Omega(\theta)$ is some norm penalty. What are the effects on the model when $\alpha = 0$ and $\alpha \to \infty$?

   (c) Mathematically show that ℓ_2 regularization shrinks the weight in gradient descent. Hint: start with $\tilde{\mathcal{L}}(\theta; \mathbf{X}, \mathbf{y}) = \mathcal{L}(\theta; \mathbf{X}, \mathbf{y}) + \frac{\alpha}{2}||\theta||_2^2$ and derive the gradient descent step for θ.

   (d) List two dataset augmentation techniques for image classification.

   (e) How did you implement dropout in homework 4? Please comment on both training and testing.

5. **Optimization techniques (Lahari)**

   (a) In lecture, we have learnt about Nesterov Momentum and it's update rule for parameters. The update rule for parameter θ is given by:

$$\begin{aligned} v_t &= \alpha v_{t-1} - \epsilon \nabla_\theta L(\theta_{t-1} + \alpha v_{t-1}) \\ \theta_t &= \theta_{t-1} + v_t \end{aligned} \tag{2}$$

   Prove that the update rule in (3) is equivalent to the update rule in (2)

$$\begin{aligned} v_{new} &= \alpha v_{old} - \epsilon \nabla_{\widetilde{\theta}_{old}} L(\widetilde{\theta}_{old}) \\ \widetilde{\theta}_{new} &= \widetilde{\theta}_{old} + v_{new} + \alpha(v_{new} - v_{old}) \end{aligned} \tag{3}$$

   Explain one advantage of using the update update rule in (3) over the update rule in (2).

   (b) Consider the two loss curves $L_1(x)$ and $L_2(x)$ shown in Figure 2. Which loss curve has a saddle point? Which loss curve has a poor local minima? In which of the loss curves, is an optimizer more likely to escape the trap of a saddle point or a poor local minima? And what property does the optimizer require for it to escape these traps in this case?

   (c) Consider the contour plot shown in Figure 3 , where the loss surface is plotted with respect to just 2 weights w1 and w2, where $w1, w2 \in R$ (scalars). Assume you are given a hypothetical scenario, where you start from point A and use vanilla gradient descent in many iterations to get to point B. During this process, we have started accumulating momentum based on the following equation,

$$\begin{aligned} g_t &= \nabla_{\theta_t} L(\theta_t) \\ v_t &= v_{t-1} - \epsilon g_t \end{aligned}$$

   Comment on which direction does the weight update occur if we use the following optimizers : vanilla gradient descent, gradient descent with momentum, gradient descent with Nesterov momentum, Adagrad.

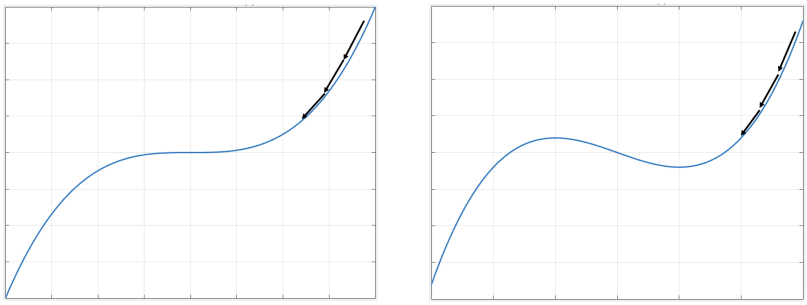


Figure 2: Loss curves $L_1(x)$ (Left), $L_2(x)$ (Right)

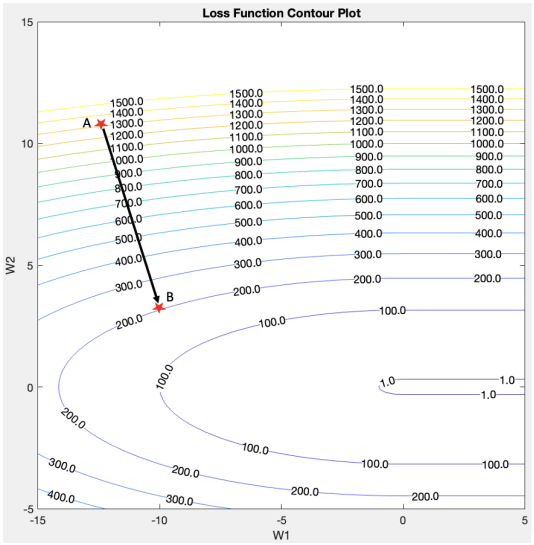


Figure 3: Contour plot of a Loss function $L(w1, w2)$

(d) In the Gradient descent + momentum scheme, find a general expression of v_t in terms of gradients $g_1, g_2, ..., g_t$ and ϵ(learning rate), considering an initial value of momentum $v_0 = 0$.

(e) Consider that the gradients $g_1, g_2, ..., g_t$ in part (d) are i.i.d. random variables with mean μ and variance σ. Find the expected value of weights θ_t at t = 3.

6. ℓ_∞ **regularization (Tonmoy)**

Let $x \in R^n$, then we define the ℓ_∞ norm and the Log-Sum-Exponent (LSE) of it as follows:

$$\|x\|_\infty = \max_i \ |x_i|$$

$$LSE(x) = \ln\left(\sum_{i=1}^{n} e^{|x_i|}\right)$$

(a) Show that the following inequality holds for $n \geq 1$,

$$\|x\|_\infty \leq LSE(x) \leq \|x\|_\infty + \ln(n) \tag{4}$$

(b) Is the lower bound in (4) strict for $n > 1$?

(c) Under what condition on x, will the upper bound in (4) be satisfied with equality.

(d) Use the result from (4) to show that the following inequality holds,

$$\|x\|_\infty \leq \frac{1}{t} LSE(tx) \leq \|x\|_\infty + \frac{\ln(n)}{t} \tag{5}$$

for some scaling constant $t > 0$.

a)

R1 → underfitting

R2 → overfitting

b → stopping point

Answer → (iii)

b) $\mathcal{L}(\theta) = \log \prod_{i=1}^{n} p(y_i = 1|\theta)^{y_i} p(y_i = 0|\theta)^{1-y_i}$

2 classes

labels $\rightarrow y_i \in \{0, 1\}$

BCE : $\mathcal{L}_2(\theta) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log p(y=1|\theta) + (1-y_i) \log p(y=0|\theta)$

i) True

iii) True.

Answer : i, iii

c)

5 fold cross validation fits 5 different models.

Answer : (i)

d) Anything that is not learnt through gradient descent is a hyperparameter.

Answer: (v) All the above.

c)

i) True, b'cos its a single sample estimate

ii) False, SGD will take more steps but converges faster in terms of clock time.

iii) True, every sample is unlikely to have the same minima of the batch.

iv) False, Batch GD requires loading the entire data.

Answers: i, iii //

Short Answers

(a) Please explain the difference between batchnormalization during training and testing.

Solution: During training, it first computes the mean and variance of the mini-batch data in the unit-wise. Then BatchNorm normalizes the layer's input based on the mean and variance. After the normalization, BatchNorm applies two learnable parameters for each unit: γ for scaling and β for shifting, which are learned during training and allow the network to adaptively adjust the output distribution. Meanwhile, it also keeps tracking the running averages for mean and variance through all batches.

During testing, BatchNorm uses the fixed accumulated running averages from training for normalizing the testing data. The learned scaling and shifting parameters γ and β are then applied to the normalized data.

(b) Your friend designed a novel activation function:

$$f(x) = x^3 \tag{1}$$

Please discuss if this is a good idea to use this activation in a neural network.

Solution: This activation function is nonlinear and differentiable everywhere, which satisfy some requirements of a good activation function. However, it is likely to cause exploding gradients as the gradient can be very large for inputs with large absolute values. Also, small inputs can lead to vanishing gradients, e.g. inputs that are close to zero.

(c) Your friend is utilizing a Multi-layer Perceptron (MLP) for a deep learning task and is trying to increase the number of units within each layer to enhance the model's complexity. Please discuss potential effect of this action on the model performance.

Solution: **(i)** If the model is originally underfitting on the training data, adding more units in layers allows the MLP to capture more complex patterns in the data, which can improve the model performance, e.g. decrease both training and testing error. **(ii)** On the other hand, it may also cause overfitting as increasing the model's capacity is likely to make the model sensitive to the training data. As a result, the training loss may still keep decreasing while the testing error increase.

(d) Please explain the role of ℓ_1 regularization.

Solution: ℓ_1 regularization introduces a penalty which is equal to the sum of absolute values of the model weights. It encourages the model to optimize some of the feature weights to zero. This property allows the model to learn a simpler and sparser patterns by pushing less important feature weights to zero, which can help prevent overfitting. Further, by observing the trained weights, we can implement feature selection to simplify the model and save computation resource.

(e) Please explain the role of the bias correction step in the Adam optimizer.

Solutions: Since Adam optimizer uses running averages to estimate the gradient and its square, these estimations are biased towards zero at the start of training because we initialize them to zero. Therefore, the optimizer is likely to take larger steps in the initial several updates of the model, leading to unstable training and slower convergence. The bias correction step adjusts these estimations to be more accurate.

After the early phase of training, the estimations tend to be accurate, so the bias correction factor will gradually approach 1, reducing the impact of bias correction.

4. Regularization techniques

– lecture 9

(a) False

(b) $\tilde{L}(\theta) = L(\theta; X, y) + \alpha \Omega(\theta)$ — loss fn.; reg. factor; norm penalty

$\alpha = 0$? $\tilde{L}(\theta) = L(\theta; X, y)$

$\alpha \to \infty$? $\tilde{L}(\theta) \to \alpha \Omega(\theta)$ — no learning of $L(\theta; X, y)$

(c) $\tilde{L}(\underline{\theta}; X, y) = L(\underline{\theta}; X, y) + \frac{\alpha}{2} \|\underline{\theta}\|_2^2$

$\underline{\theta}$ is a vector

$\nabla_{\underline{\theta}} \tilde{L}(\underline{\theta}; X, y) = \frac{\partial}{\partial \underline{\theta}} \tilde{L}(\underline{\theta}; X, y)$

$\frac{\alpha}{2} \|\underline{\theta}\|_2^2 = \frac{\alpha}{2} \underline{\theta}^T \underline{\theta}$

$\frac{\partial}{\partial \underline{\theta}} \frac{\alpha}{2} \underline{\theta}^T \underline{\theta} = \alpha \underline{\theta}$

$= \frac{\partial}{\partial \underline{\theta}} \left[L(\underline{\theta}; X, y) + \frac{\alpha}{2} \|\underline{\theta}\|_2^2 \right]$

$= \nabla_{\underline{\theta}} L(\underline{\theta}; X, y) + \alpha \underline{\theta}$

$\underline{\theta} \leftarrow \underline{\theta} - \varepsilon \nabla_{\underline{\theta}} \tilde{L}(\underline{\theta}; X, y)$

$\leftarrow \underline{\theta} - \varepsilon [\nabla_{\underline{\theta}} L(\underline{\theta}; X, y) + \alpha \underline{\theta}]$

$\leftarrow (1 - \varepsilon\alpha) \underline{\theta} - \varepsilon \nabla_{\underline{\theta}} L(\underline{\theta}; X, y)$

↑ weight decay

(d) flipping / mirroring → random crops, rotation, brightness adjustments

lens correction

pooling (CNN)

label smoothing

subsampling

color space trans.

(e) inverted dropout

training: generate binary mask

* values after affine transform

/ p ← the prob. of keeping

test: nothing

(Q5) (a) Given equation of nesterov momentum

$$\left.\begin{aligned} v_t &= \alpha v_{t-1} - \epsilon \nabla_\theta L(\theta_{t-1} + \alpha v_{t-1}) \\ \theta_t &= \theta_{t-1} + v_t \end{aligned}\right\} \longrightarrow (1)$$

To prove it is equivalent to:

$$\left.\begin{aligned} v_{new} &= \alpha v_{old} - \epsilon \nabla_{\tilde{\theta}} L(\tilde{\theta}_{old}) \\ \tilde{\theta}_{new} &= \tilde{\theta}_{old} + v_{new} + \alpha(v_{new} - v_{old}) \end{aligned}\right\} \longrightarrow (2)$$

$\Rightarrow$ Let us assume that new parameter $\tilde{\theta}$

$$\tilde{\theta} = \theta + \alpha v$$

This is: $\tilde{\theta}_{old} = \theta_{old} + \alpha v_{old}$ | $\theta_{old} = \tilde{\theta}_{old} - \alpha v_{old}$

$\tilde{\theta}_{new} = \theta_{new} + \alpha v_{new}$ | $\theta_{new} = \tilde{\theta}_{new} - \alpha v_{new}$

$\Rightarrow$ From (1), we have:

$$v_{new} = \alpha v_{old} - \epsilon \nabla_\theta L(\theta_{old} + \alpha v_{old})$$

$$\nabla_{\tilde{\theta}} L(\tilde{\theta}_{old}) = \frac{\partial L(\tilde{\theta}_{old})}{\partial \tilde{\theta}} = \frac{\partial \theta}{\partial \tilde{\theta}} \cdot \frac{\partial L(\tilde{\theta}_{old})}{\partial \theta}$$

$$= \frac{\partial \theta}{\partial \tilde{\theta}} \cdot \frac{\partial L(\theta_{old} + \alpha v_{old})}{\partial \theta} = \nabla_\theta L(\theta_{old} + \alpha v_{old})$$

$\left[\because \frac{\partial \theta}{\partial \tilde{\theta}} = \frac{\partial}{\partial \tilde{\theta}}(\tilde{\theta} + \alpha v) = I + 0 = I\right]$

So, we have, $\nabla_{\tilde{\theta}} L(\tilde{\theta}_{old}) = \nabla_{\theta} L(\theta_{old} + \alpha V_{old})$

$V_{new} = \alpha V_{old} - \epsilon \nabla_{\tilde{\theta}} L(\tilde{\theta}_{old})$ ⟶ This is (2) equation

⟹ From (1), we have: $\theta_{new} = \theta_{old} + V_{new}$

$\tilde{\theta}_{old} - \alpha V_{new} = \theta_{old} - \alpha V_{old} + V_{new}$

$\tilde{\theta}_{old} = \theta_{old} + V_{new} + \alpha(V_{new} - V_{old})$ ⟶ This is (2) equation

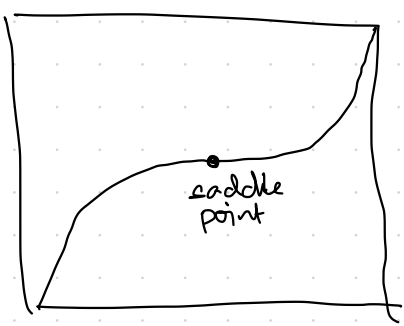
Hence, we have shown that both (1) & (2) are equivalent as we can get (2) from (1) by a change in variable.

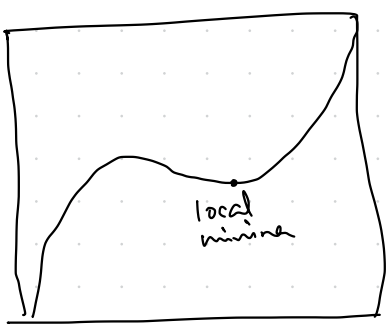
This representation helps us in implementation of nesterov momentum as this doesn't require us to calculate gradient at a different value of $\theta + \alpha V$.

[Also, both θ and $\tilde{\theta}$ start from the same value of parameter initialization as $\theta = \tilde{\theta}$ initially as $V = 0$ at start]

(b) $L_1(x)$ has a saddle point. The curve decreases on one side of the saddle and increases on other side of saddle point. The gradient at a saddle point is equal to zero.

$L_2(x)$ has a poor local minima. This is a local minima as it is a minimum value of the function in it's surrounding until a certain limit in all the directions. The gradient is zero at a local minima.

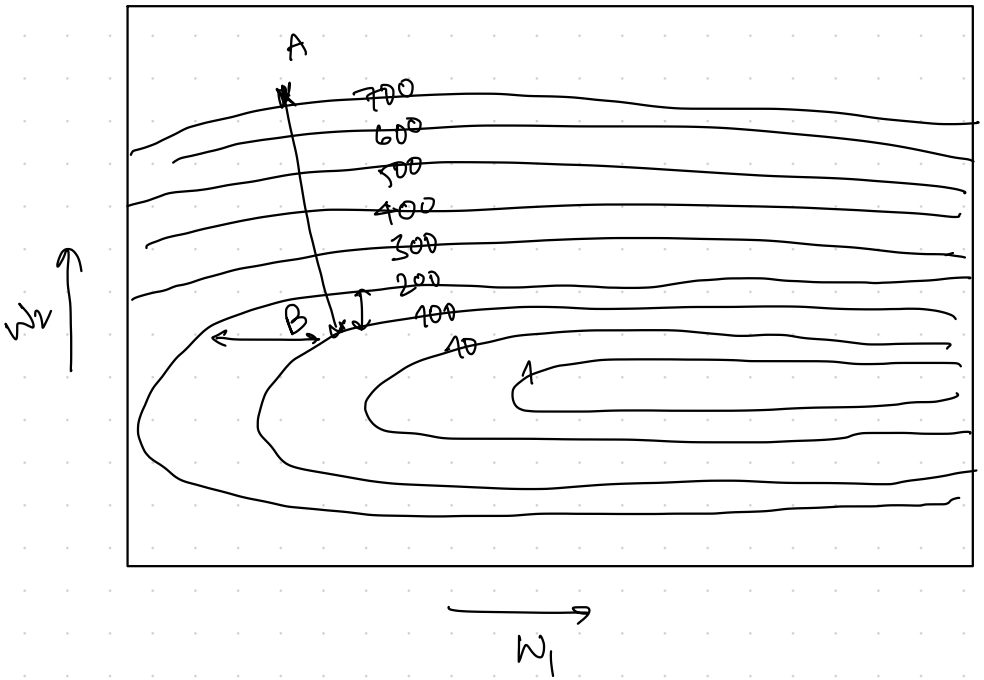




We need momentum to get out of saddle point or local minima because the gradient becomes zero at these points.

It is also more likely to escape the saddle point than the local minima because after crossing the local minima, the momentum decreases due to gradient being in opposite direction to the momentum accumulated down the slope.

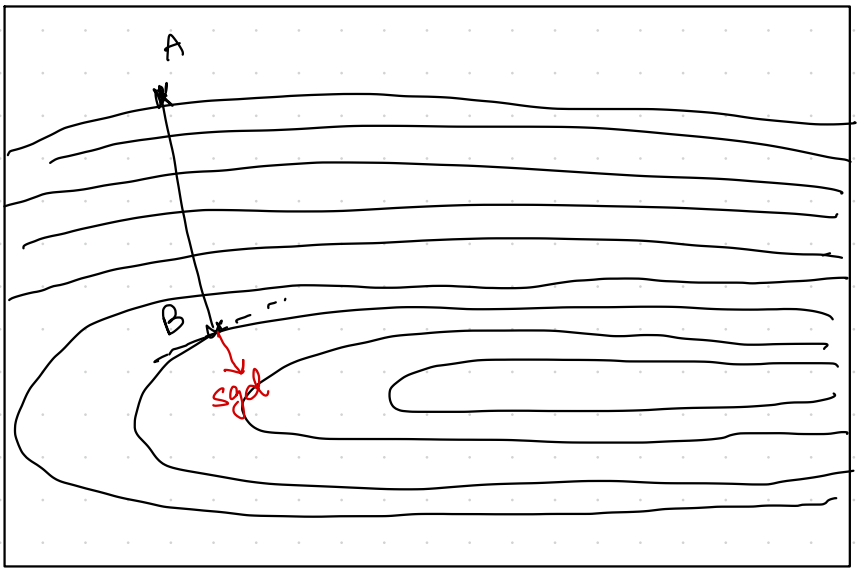
(c)



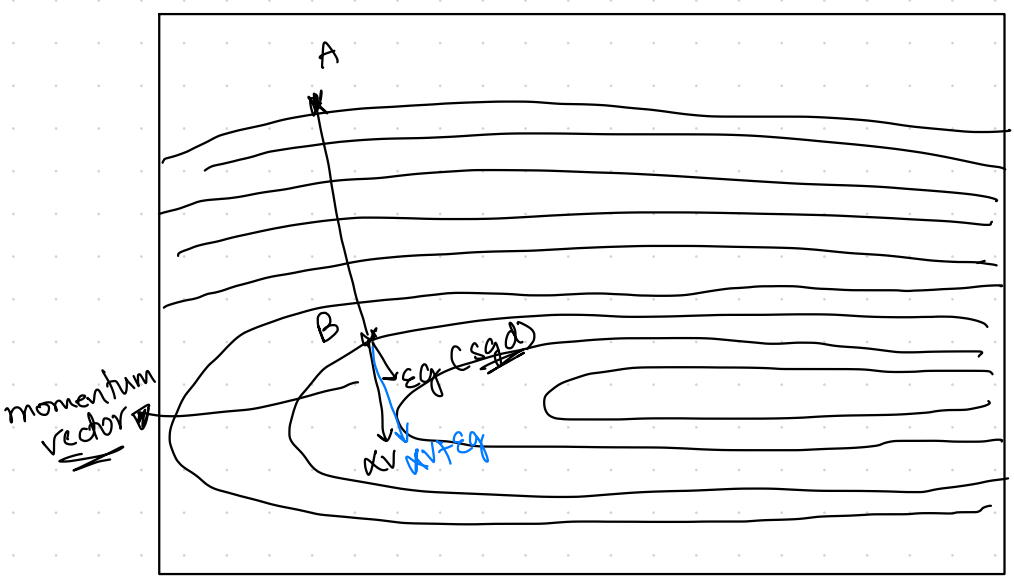
From the above contour plot, we can make some initial observations:

(1) The gradient along w_2 direction is higher than the gradient along w_1 direction.
This is because the contour lines along w_2 directions are very close to each other which indicates a steep curve in w_2 direction.
The contour lines along w_1 direction are further apart and hence has slower descent. (or low gradient)

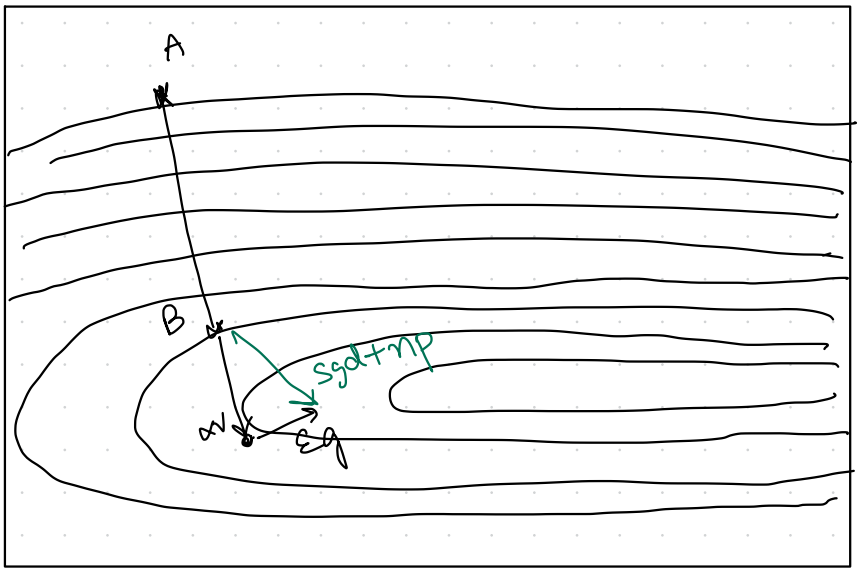
(2) Vanilla gradient descent is in the direction perpendicular to a contour line.



(3) sgd + momentum



(4) sgd + nesterov momentum.



In this, first we take a step along the momentum and then calculate the gradient at that point in graph.

(5) Adagrad.

The update equation:

$$a \leftarrow a + g \odot g \quad \text{[accumulates gradient]}$$

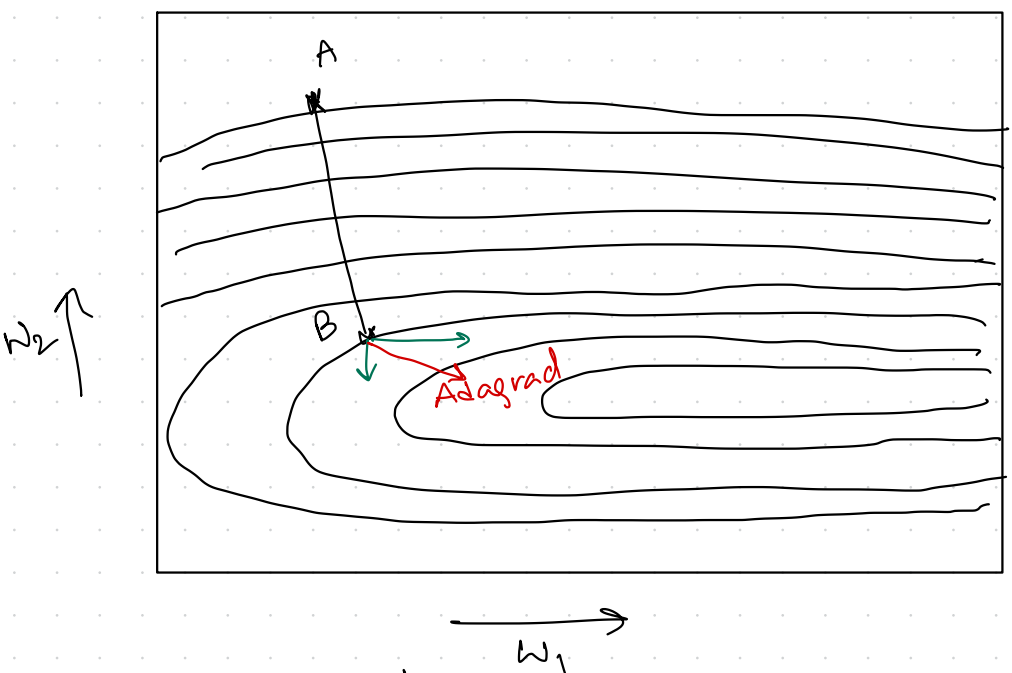
$$\theta \leftarrow \theta + \frac{\epsilon}{\sqrt{a+v}} \odot g$$

If gradient is accumulated more in a direction, then the step $\frac{\epsilon}{\sqrt{a+v}}$ is less in that direction.

If gradient accumulated is less in a direction, then the step $\frac{\epsilon}{\sqrt{a+v}}$ is more in that direction

From plot, we can see that when it traversed from A to B, the gradients accumulated along w_2 direction is more and gradient along w_1 direction is lesser than it.

Hence, the step along w_1 direction is more and step along w_2 direction is less.



Also, in this particular case we can see that Adagrad is converging faster to minima without much zigzagging.

(d) Sgd + momentum update:

$$v_t = \alpha v_{t-1} - \varepsilon g_t$$
$$\theta_t = \theta_{t-1} + v_t$$

$\rightarrow v_0 = 0$ [Given]

$v_1 = \alpha v_0 - \varepsilon g_1 = -\varepsilon g_1$

$v_2 = \alpha v_1 - \varepsilon g_2 = \alpha(-\varepsilon g_1) - \varepsilon g_2 = -\varepsilon(g_2 + \alpha g_1)$

$v_3 = \alpha v_2 - \varepsilon g_3 = \alpha(-\varepsilon(g_2 + \alpha g_1)) - \varepsilon g_3$

$= -\varepsilon(g_3 + \alpha g_2 + \alpha^2 g_1)$

⋮

From this, we can observe that

$\Rightarrow v_t = -\varepsilon(g_t + \alpha g_{t-1} + \alpha^2 g_{t-2} + \ldots + \alpha^{t-1} g_1)$

$E(v_t) = E[-\varepsilon(g_t + \alpha g_{t-1} + \alpha^2 g_{t-2} + \ldots + \alpha^{t-1} g_1)]$

$= -\varepsilon[E(g_t) + \alpha E(g_{t-1}) + \ldots + \alpha^{t-1} E(g_1)]$

(e) $\theta_0 = \theta_0$

$\theta_1 = \theta_0 + v_1$

$\theta_2 = \theta_1 + v_2 = \theta_0 + v_1 + v_2$

$\theta_3 = \theta_2 + v_3 = \theta_0 + v_1 + v_2 + v_3$

⋮

So, for general $\theta_t = \theta_0 + v_1 + v_2 + \ldots + v_t$

$$E(\theta_3) = E\left[\theta_0 + v_1 + v_2 + v_3\right]$$

$$= E(\theta_0) + E(v_1) + E(v_2) + E(v_3)$$

Given, $g_1, g_2 \ldots, g_t$ are i.i.d r.vs with mean μ and var σ^2.

$\Rightarrow$ From part (d), we saw

$$E(v_1) = E(-\varepsilon g_1) = -\varepsilon E(g_1) = -\varepsilon\mu$$

$$E(v_2) = E\left(-\varepsilon(g_2 + \alpha g_1)\right) = -\varepsilon\left[\mu + \alpha\mu\right]$$

$$= -\varepsilon\mu[1+\alpha]$$

$$E[v_3] = E\left(-\varepsilon(g_3 + \alpha g_2 + \alpha^2 g_1)\right)$$

$$= -\varepsilon\left[\mu + \alpha\mu + \alpha^2\mu\right]$$

$$= -\varepsilon\mu[1 + \alpha + \alpha^2]$$

$$E(\theta_3) = E(\theta) - \varepsilon\mu\left[1 + (1+\alpha) + (1+\alpha+\alpha^2)\right]$$

** We know that sum of 'n' terms in * geometric series with factor 'r' and first term a is

$$a + ar + \ldots + ar^{n-1} = \frac{a(1-r^n)}{1-r}$$

Alternately, $E(V_t) = -\varepsilon\left[\mu + \alpha\mu + \ldots + \alpha^{t-1}\mu\right]$

$$= -\varepsilon\mu\left[1 + \alpha + \ldots + \alpha^{t-1}\right]$$

$$= -\varepsilon\mu\left[\frac{1-\alpha^t}{1-\alpha}\right]$$

$$E(\theta_3) = E(\theta_0) + \sum_{t=1}^{3}\left\{-\varepsilon\mu\frac{[1-\alpha^t]}{1-\alpha}\right\}$$

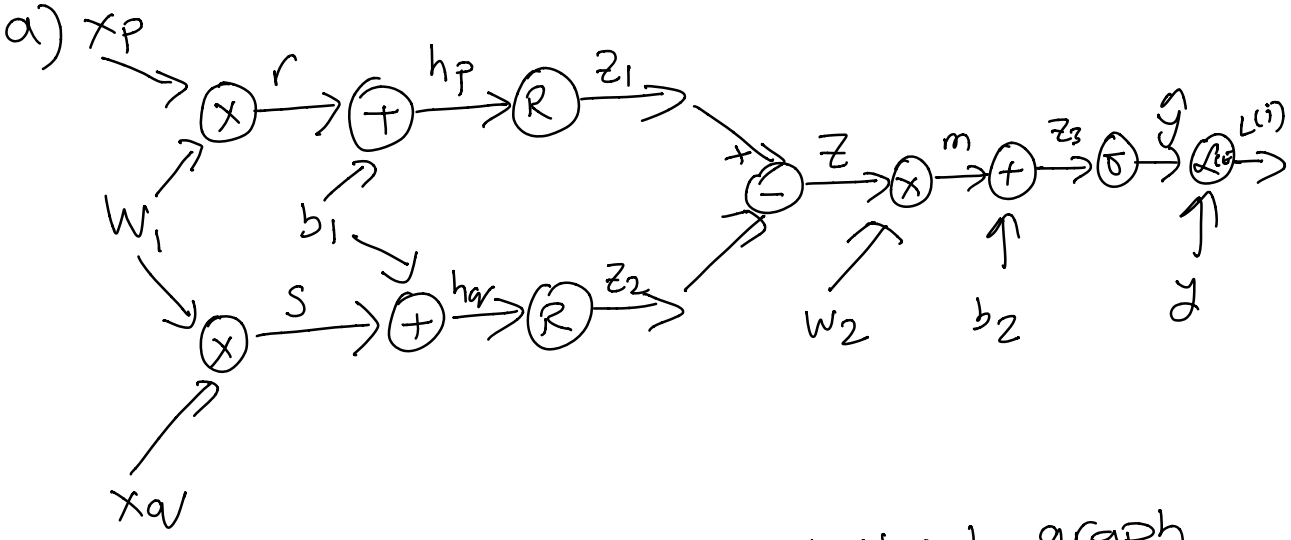
$$= E(\theta_0) - \frac{\varepsilon\mu}{1-\alpha}\left[\sum_{t=1}^{3}(1-\alpha^t)\right]$$

$$= E(\theta_0) - \frac{\varepsilon\mu}{1-\alpha}\left[3 - \sum_{t=1}^{3}\alpha^t\right]$$

$$= E(\theta_0) - \frac{\varepsilon\mu}{1-\alpha}\left[3 - \left(\frac{1-\alpha^3}{1-\alpha}\right)\right]$$

3

a)



We have drawn the computational graph above. Now, we will use backpropagation to compute the derivatives required for learning the parameters through gradient descent.

**Always write dimensions

$x_p^{(i)} \in \mathbb{R}^{D_x}$, $x_q^{(i)} \in \mathbb{R}^{D_x}$, $\hat{y}^{(i)} \in \mathbb{R}$

$W_1 \in \mathbb{R}^{D_z \times D_x}$, $b_1 \in \mathbb{R}^{D_z}$

$W_2 \in \mathbb{R}^{1 \times D_z}$, $b_2 \in \mathbb{R}$

b) From the computational graph,

$$L^{(i)} = LCE(y^{(i)}, \hat{y}^{(i)})$$

$$= y^{(i)} \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$$

Since all quantities are scalar, so by scalar derivatives,

$$\frac{\partial L^{(i)}}{\partial \hat{y}^{(i)}} = \frac{y^{(i)}}{\hat{y}^{(i)}} - \frac{1}{1-\hat{y}^{(i)}} + \frac{y^{(i)}}{1-\hat{y}^{(i)}}$$

$$\delta_{\hat{y}^{(i)}} = \frac{y^{(i)}}{\hat{y}^{(i)}} - \left\{ \frac{1-y^{(i)}}{1-\hat{y}^{(i)}} \right\}$$

c) From the computational graph,

$$\hat{y}^{(i)} = \sigma(z_3)$$

Since all quantities are scalar, so by scalar chain rule

$$\frac{dL^{(i)}}{dz_3} = \frac{d\hat{y}^{(i)}}{dz_3} \frac{dL^{(i)}}{d\hat{y}^{(i)}}$$

Now, from lecture we know

$$\frac{d}{dz_3}\sigma(z_3) = \sigma(z_3)[1-\sigma(z_3)]$$

(Quotient rule)

So,

$$\delta_{z_3} = \sigma(z_3)[1-\sigma(z_3)]\,\delta_{\hat{y}^{(i)}}$$

where $\sigma(z_3) = \frac{1}{1+e^{-z_3}}$

d) From the computational graph, backpropating through ⊕ gate

$$\frac{\partial L^{(i)}}{\partial b_2} = \frac{\partial L}{\partial z_3} = \delta_{z_3}$$

So, $\delta_{b_2} = \delta_{z_3}$

e) From the computational graph,

$$m = W_2 z$$

So, by chain rule

$$\frac{\partial L^{(i)}}{\partial W_2} = \frac{\partial m}{\partial W_2} \frac{\partial L^{(i)}}{\partial m}$$

$$\delta_{W_2} = \delta_{z_3} z^T$$

f) Using chain rule,

$$\frac{dL^{(i)}}{dz} = \frac{dm}{dz} \frac{dL^{(i)}}{dm}$$

$$\delta_z = \delta_{z_3} W_2^T$$

g) From the computational graph,

$$z = z_1 - z_2$$

Since we backpropagate through + terminal of $\ominus$ gate for z_1, so

$$\frac{dL^{(i)}}{dz_1} = \frac{dL^{(i)}}{dz}$$

$$\delta_{z_1} = \delta_z$$

h) Since we backpropagate through $-$ terminal of $\ominus$ gate for z_2, so

$$\frac{dL^{(i)}}{dz_2} = -\frac{dL^{(i)}}{dz}$$

$$\delta_{z_2} = -\delta_z$$

i) From the computational graph,

$$z_2 = \text{ReLU}(har)$$

From Discussion, we know that backpropagating through ReLU(·) leads to a hadamard product, so

$$\frac{dL^{(i)}}{dh_{a'}} = \mathbb{I}(h_{a'} > 0) \odot \frac{dL^{(i)}}{dz_2}$$

$$\delta_{h_{a'}} = \mathbb{I}(h_{a'} > 0) \odot \delta_{z_2}$$

j) Using the same logic as i) we have

$$\delta_{h_p} = \mathbb{I}(h_p > 0) \odot \delta_{z_1}$$

k) By law of total derivatives,

$$\frac{dL^{(i)}}{db_1} = \frac{dL^{(i)}}{dh_p} + \frac{dL^{(i)}}{dh_{a'}}$$

$$\delta_{b_1} = \delta_{h_p} + \delta_{h_{a'}}$$

1) Backprogating through (+)

gate,

$$\frac{dL^{(i)}}{dr} = \frac{dL^{(i)}}{dh_p}$$

$$\frac{dL^{(i)}}{ds} = \frac{dL^{(i)}}{dh_{av}}$$

By chain rule,

$$\left.\frac{dL^{(i)}}{dW_1}\right|_{Path\,1} = \frac{dr}{dW_1} \frac{dL^{(i)}}{dr}$$

By 3-D tensor derivative trick)

$$\left.\frac{dL^{(i)}}{dW_1}\right|_{Path\,1} = \frac{dL^{(i)}}{dh_p} X_p^T$$

By chain rule,

$$\frac{dL^{(i)}}{dW_1}\bigg|_{path\,2} = \frac{ds}{dW_1}\frac{dL^{(i)}}{ds}$$

By 3-D tensor derivative trick,

$$\frac{dL^{(i)}}{dW_1}\bigg|_{path\,2} = \frac{dL^{(i)}}{dh_a} X_a^T$$

Hence by law of total derivatives,

$$\frac{dL^{(i)}}{dW_1} = \frac{dL^{(i)}}{dW_1}\bigg|_{path\,1} + \frac{dL^{(i)}}{dW_1}\bigg|_{path\,2}$$

$$\delta_{W_1} = \delta_{h_p} X_p^T + \delta_{h_a} X_a^T$$

6

a) Suppose,

$$p = \max_i \{ |x_1|, |x_2|, \ldots, |x_n| \}$$

Then, we have the following lower bound

$$e^p \leq \sum_{i=1}^{n} e^{|x_i|}$$

We also have the following upper bound

$$\sum_{i=1}^{n} e^{|x_i|} \leq n e^p$$

Combining the upper and lower bounds we have

$$e^p \leq \sum_{i=1}^{n} e^{|x_i|} \leq n e^p$$

Taking natural logarithm of the above inequality we have

$$\ln(e^p) \leq \ln\left(\sum_{i=1}^{n} e^{|x_i|}\right) \leq \ln(n e^p)$$

$$\Rightarrow \quad p \ln e \leq \ln\left(\sum_{i=1}^{n} e^{|x_i|}\right) \leq \ln(n) + p \ln(e)$$

Since $p = \|x\|_\infty$, so we have the required inequality

$$\|x\|_\infty \leq LSE(x) \leq \|x\|_\infty + \ln(n)$$

b) Clearly for $n > 1$,

$$e^p < \sum_{i=1}^{n} e^{|x_i|}$$

Taking natural logarithm of both sides,

$$\ln(e^p) < \ln\left(\sum_{i=1}^{n} e^{|x_i|}\right)$$

$$p \ln(e) < \ln\left(\sum_{i=1}^{n} e^{|x_i|}\right)$$

$$\therefore \quad \|x\|_\infty < LSE(x)$$

c) Suppose,

$$|x_i| = |x_j|$$

for all $i, j \in n$. Then we have

$$\sum_{i=1}^{n} e^{|x_i|}$$

$$= \sum_{i=1}^{n} e^{p}$$

$$= n e^{p}$$

Hence, the upper bound will be satisfied with equality

$$LSE(x) = \|x\|_\infty + \ln(n)$$

d) Let $t>0$ be some scaling constant. Substituting X with tx in (2), we get

$$\|tx\|_\infty \leq LSE(tx) \leq \|tx\|_\infty + \ln(n)$$

Now,

$$\|tx\|_\infty = t\|x\|_\infty$$

So,

$$t\|x\|_\infty \leq LSE(tx) \leq t\|x\|_\infty + \ln(n)$$

Dividing by t, we get the required inequality,

$$\|x\|_\infty \leq \frac{1}{t} LSE(tx) \leq \|x\|_\infty + \frac{\ln(n)}{t}$$