Student Name: Mohammad Mohammad Beigi
Student ID: 99102189
Subject: Deep Learning

Deep Learning - Dr. E. Fatemizadeh
Assignment 2

Question 1

a)

By increasing size of mini batch each update is based on a larger set of training examples. This can result in more accurate gradient estimates, leading to more meaningful steps in the parameter space. So when we increase mini batch we need less steps because we have more accurate steps. And when mini batch size increases more than a rational size, the updates accuracies decrease and number of steps to converge, increases as a result.

b)

In batch normalization (BN), noise is not intentionally injected into activation functions. The primary purpose of batch normalization is to improve the training of deep neural networks by normalizing the input of each layer during training. This normalization helps address issues such as internal covariate shift, which can slow down the training process.

Batch normalization helps mitigate the vanishing/exploding gradient problems and allows for more stable and faster training of deep networks. It doesn't introduce noise intentionally but can be seen as adding a controlled form of noise through the normalization process, which helps prevent the network from becoming too reliant on specific weight initializations.

c)

Using a fully connected network with sigmoid activation functions and initializing weights with large numbers may not be the best choice for training a neural network, especially for classification tasks. There are several reasons for this:

Vanishing Gradient Problem: Sigmoid activation functions squash input values to a range between 0 and 1. During backpropagation, the gradients can become very small, leading to slow or stalled learning. This is known as the vanishing gradient problem.

Initialization Issues: Initializing weights with large values can exacerbate the vanishing gradient problem and also lead to slow convergence during training. Weight initialization is crucial for proper training of neural networks.

Saturated Neurons: Sigmoid functions can saturate, meaning for very large positive or negative inputs, the output becomes very close to 0 or 1. When this happens, the gradient becomes close to zero, and the network has difficulty learning from those neurons.

d)

Between each pair of hidden layer we have 10^2 weights : 4×10^2

Between input layer and first hidden layer we have 20×10 weights.

Between last hidden layer and output layer we have 10×1 weights.

Also except the input layer we have a bias parameter for each neuron of all layers: $5 \times 10 + 1 = 51$

So totally we have $400 + 200 + 10 + 51 = 661$ learnable parameters.

e)

No. The performances are the same.

$$\text{For softmax we have: } W_s = [W_1, W_2], b = [b_1, b_2]$$

$$\frac{e^{W_1 x + b_1}}{e^{W_2 x + b_2} + e^{W_1 x + b_1}} = \frac{1}{e^{[W_2 - W_1]x + [b_2 - b_1]} + 1} = \frac{1}{e^{-([W_2 - W_1]x + [b_2 - b_1])} + 1} = \frac{1}{e^{[W_1 - W_2]x + [b_1 - b_2]} + 1}$$

So for we can set the following parameters to have same methods of classifying:

$$W_l = W_1 - W_2 \text{ and } b_l = b_1 - b_2$$

Question 2

a)

A committee of neural nets refers to a group of individual neural networks that work together to make predictions or classifications. Each neural network in the committee is trained on the same dataset but with different preprocessing techniques applied to the data. The goal is to create a committee where the errors of the individual networks are not correlated.

By combining the predictions of multiple neural networks in a committee, the overall performance of the model can be improved. The committee takes advantage of the fact that different neural networks may capture different aspects or patterns in the data, leading to complementary information. This can help to reduce the overall error rate and improve the accuracy of the model's predictions.

The committee can be formed using different fusion strategies, such as majority voting, averaging the class probabilities, or taking the median of the class probabilities. These strategies allow for the combination of the individual predictions from each neural network in the committee.

In summary, a committee of neural nets improves the function of the model by leveraging the complementary information captured by different neural networks, leading to improved accuracy and performance in making predictions or classifications.

b)

In the article, several preprocessing techniques were used to decorrelate the errors of the individual models in the committee of neural nets. These techniques include width normalization (WN) and deslanting of the training set.

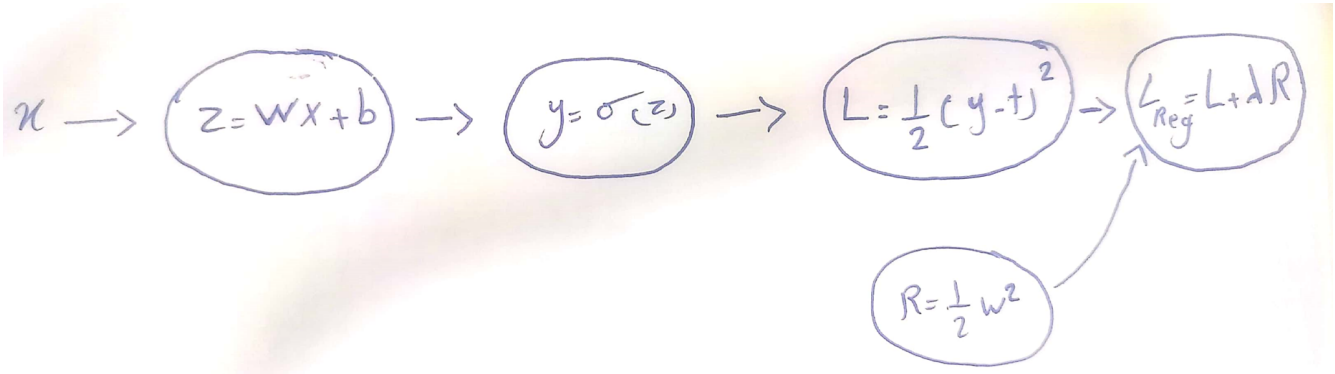
Width normalization involves normalizing the bounding box of the handwritten digits to a specific width. This helps to reduce variations in aspect ratio and ensures that all digits have a consistent width.

Deslanting refers to correcting the slant or tilt of the handwritten digits. By aligning the digits vertically, the preprocessing technique helps to remove any slant and make the digits more uniform.

The purpose of preprocessing the data for each individual model in a committee of neural nets is to ensure that the errors of the individual experts are not strongly correlated. By applying the explained techniques to the data prior to training, the individual predictors in the committee become less correlated. This helps to improve the overall recognition performance of the committee. Additionally, preprocessing the data allows for the introduction of desired invariance and implicit regularization, which can help prevent overfitting.

Question 3

a)



$$\frac{\partial L_{reg}}{\partial L} = 1$$

$$\frac{\partial L_{reg}}{\partial \lambda} = R = \frac{w^2}{2}$$

$$\frac{\partial L_{reg}}{\partial R} = \lambda$$

$$\frac{\partial L_{reg}}{\partial y} = 1 \times (y - t) = y - t = \sigma(wx + b) - t$$

$$\frac{\partial L_{reg}}{\partial w} = \lambda \times w + 1 \times (y - t) \times (\sigma(z))' \times x = \lambda w + (y - t)\sigma(z)(1 - \sigma(z))x = \lambda w + (y - t)y^2 e^{-z} x$$

$$= \lambda w + (\sigma(wx + b) - t)\sigma^2(wx + b)e^{-wx-b}x$$

$$\frac{\partial L_{reg}}{\partial z} = 1 \times (y - t) \times (\sigma(z))' = (\sigma(wx + b) - t)\sigma^2(wx + b)e^{-wx-b}$$

$$\frac{\partial L_{reg}}{\partial b} = 1 \times (y - t) \times (\sigma(z))' \times 1 = (\sigma(wx + b) - t)\sigma^2(wx + b)e^{-wx-b}$$

b)

Initializing the parameters of a neural network with random and small numbers is a common practice in deep learning. The rationale behind this approach is to break the symmetry among neurons in the network and prevent neurons from learning the same features. If all the neurons start with the same weights, they will always update in the same way during training, and the network won't be able to learn diverse features, limiting its representational capacity.

Here are some consequences of not initializing parameters properly:

1. Symmetry Issues: If all the neurons in a layer have the same initial weights, they will produce the same output during the forward pass and have the same gradients during the backward pass. This symmetry can prevent the model from learning meaningful representations.

2. Vanishing or Exploding Gradients: Poor parameter initialization can lead to the vanishing or exploding gradient problem. If weights are too small, the gradients during backpropagation can become very small, causing the model to learn very slowly or not at all. On the other hand, if weights are too large, gradients can explode, leading to numerical instability during training.

3. Convergence Issues: In some cases, improper initialization(big numbers) can prevent the model from converging to a good solution. The optimization process may get stuck in a poor local minimum, and the training process may become unstable.

4. Training Time and Resource Inefficiency: Poorly initialized parameters can result in a slower convergence rate, requiring more epochs for the model to reach a satisfactory performance level. This inefficiency can be problematic, especially when dealing with large datasets or complex models.

To address these issues, various initialization techniques have been proposed, such as Xavier/Glorot initialization or He initialization, which take into account the number of input and output units for each layer. These methods help mitigate problems associated with improper initialization and contribute to more stable and efficient training of neural networks.

c)

$$W^{(1)} = W^{(0)} - 0.1\frac{\partial L_{reg}}{\partial w} = W^{(0)} - 0.1\lambda W^{(0)} + (\sigma(W^{(0)}x + b^{(0)}) - t)\sigma^2(W^{(0)}x + b^{(0)})e^{-W^{(0)}x - b^{(0)}}x$$

$$b^{(1)} = b^{(0)} - 0.1\frac{\partial L_{reg}}{\partial b} = b^{(0)} - 0.1(\sigma(W^{(0)}x + b^{(0)}) - t)\sigma^2(W^{(0)}x + b^{(0)})e^{-W^{(0)}x - b^{(0)}}$$

Question 4

a)

Adam (short for Adaptive Moment Estimation) is a popular optimization algorithm for stochastic gradient descent (SGD), commonly used to update the weights of a neural network during training.

It combines ideas from two other optimization algorithms: RMSprop (Root Mean Square Propagation) and Momentum.

Now, what is moment ? N-th moment of a random variable is defined as the expected value of that variable to the power of n. More formally:

$$m_n = E[X^n]$$

To estimate the moments, Adam utilizes exponentially moving averages, computed on the gradient evaluated on a current mini-batch:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Where m and v are moving averages, g is gradient on current mini-batch, and betas — new introduced hyper-parameters of the algorithm. They have really good default values of 0.9 and 0.999 respectively. Almost no one ever changes these values. The vectors of moving averages are initialized with zeros at the first iteration.

Now we need to correct the estimator, so that the expected value is the one we want(we will explain it in next part). This step is usually referred to as bias correction. The final formulas for our estimator will be as follows:

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

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

The only thing left to do is to use those moving averages to scale learning rate individually for each parameter. The way it's done in Adam is very simple, to perform weight update we do the following:

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Where w is model weights, eta (look like the letter n) is the step size (it can depend on iteration).
Totally η, β_1 , β_2 , and ϵ are hyper parameters.

b)

Now, we will see that these do not hold true for the our moving averages. Because we initialize averages with zeros, the estimators are biased towards zero. Let's prove that for m (the proof for v would be analogous). To prove that we need to formula for m to the very first gradient. Let's try to unroll a couple values of m to see he pattern we're going to use:

$$m_0 = 0$$
$$m_1 = \beta_1 m_0 + (1-\beta_1) g_1 = (1-\beta_1) g_1$$
$$m_2 = \beta_1 m_1 + (1-\beta_1) g_2 = \beta_1 (1-\beta_1) g_1 + (1-\beta_1) g_2$$
$$m_3 = \beta_1 m_2 + (1-\beta_1) g_3 = \beta_1^2 (1-\beta_1) g_1 + \beta_1 (1-\beta_1) g_2 + (1-\beta_1) g_3$$

As you can see, the 'further' we go expanding the value of m, the less first values of gradients contribute to the overall value, as they get multiplied by smaller and smaller beta. Capturing this patter, we can rewrite the formula for our moving average:

$$m_t = (1-\beta_1) \sum_{i=0}^{t} \beta_1^{t-i} g_i$$

To calculate bias, we calculate expected value:

$$E[m_t] = E[(1-\beta_1) \sum_{i=1}^{t} \beta_1^{t-i} g_i]$$
$$= E[g_i](1-\beta_1) \sum_{i=1}^{t} \beta_1^{t-i} + \zeta$$
$$= E[g_i](1-\beta_1^t) + \zeta$$

Because the approximation is taking place, the error ζ emerge in the formula(ζ is a very small number). Since m and v are estimates of first and second moments, we want to have the following property:

$$E[m_t] = E[g_t]$$
$$E[v_t] = E[g_t^2]$$

If we use

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

We have:

$$E[\hat{m}_t] = E[\frac{m_t}{(1-\beta_1^t)}] = \frac{E[g_i](1-\beta_1^t)+\zeta}{(1-\beta_1^t)} = E[g_i] + \text{very small number}$$

Question 5

a)

$$w_{t+1} = w_t - \alpha\nabla_w(w_t^T H w_t) = w_t - 2\alpha H w_t = (I - 2\alpha H)w_t = (I - 2\alpha Q\Lambda Q^T)w_t$$

Now by applying the fact that $QQ^T = I$ we have:

$$w_{t+1} = (QQ^T - 2\alpha Q\Lambda Q^T)w_t = Q(I - 2\alpha\Lambda)Q^T w_t$$

b)

$w_1 = Q(I-2\alpha\Lambda)Q^T w_0 \Rightarrow w_2 = Q(I-2\alpha\Lambda)Q^T w_1 = Q(I-2\alpha\Lambda)Q^T Q(I-2\alpha\Lambda)Q^T w_0 = Q(I-2\alpha\Lambda)^2 Q^T w_0$

Similarly we have:

$$w_t = (Q(I - 2\alpha\Lambda)Q^T)^t w_0 = Q(I - 2\alpha\Lambda)^t Q^T w_0$$

c)

To check the conditions for convergence we should check the matrix $(I - 2\alpha\Lambda) = diag(1 - 2\alpha\lambda_i)$ elements :

$$|1 - 2\alpha\lambda_i| < 1 \Rightarrow -1 < 1 - 2\alpha\lambda_i < 1 \Rightarrow -2 < -2\alpha\lambda_i < 0$$

We know that H is a positive definite matrix. So its eigenvalues are positive and we have:

$$0 < \alpha < \frac{1}{\lambda_i}$$

d)

we have for the gradient and the Hessian

$$\begin{aligned} f &= \mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} \\ \nabla_{\mathbf{x}} f = \frac{\partial f}{\partial \mathbf{x}} &= (\mathbf{A} + \mathbf{A}^T)\mathbf{x} + \mathbf{b} \\ \frac{\partial^2 f}{\partial \mathbf{x}\partial \mathbf{x}^T} &= \mathbf{A} + \mathbf{A}^T \end{aligned}$$

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \boldsymbol{g} + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left.\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}, \boldsymbol{H} = \nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left.\frac{\partial^2 J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^2}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0}$$

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1}\boldsymbol{g}$$

In Newton's method we have:

$$w_{t+1} = w_t - (H + H^T)^{-1}(H + H^T)w_0 = w_t - w_0$$

So the algorithm will converge in one step.

e)

Both Newton's method and gradient descent are optimization algorithms commonly used in the context of training deep neural networks. However, they have distinct characteristics, advantages, and limitations. Let's compare Newton's method and gradient descent:

1. **Nature of Optimization:**

Gradient Descent:
- Iterative first-order optimization algorithm.
- Relies on the gradient (first derivative) of the loss function with respect to the parameters.

Newton's Method:
- Iterative second-order optimization algorithm.
- Utilizes both the gradient and the Hessian matrix (second derivative) of the loss function.

2. **Update Rule:**

Gradient Descent:
- Update is proportional to the negative of the gradient.
- The general update rule for parameter θ is $\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t)$, where η is the learning rate.

Newton's Method:
- Update is based on the inverse of the Hessian matrix.
- The update rule is $\theta_{t+1} = \theta_t - H^{-1}\nabla J(\theta_t)$, where H is the Hessian matrix.

3. **Convergence:**

Gradient Descent:
- May converge slowly, especially in regions with high curvature.
- Sensitive to the choice of learning rate.

Newton's Method:

- Can converge faster, especially in regions with complex curvature.
- May be computationally expensive due to the need to compute and invert the Hessian matrix.

4. **Computational Complexity:**
Gradient Descent:
- Generally less computationally intensive as it only involves computing the gradient.
- Well-suited for large datasets and high-dimensional parameter spaces.
Newton's Method:
- Involves computing the Hessian matrix and its inverse, which can be computationally expensive, especially for large neural networks.

5. **Memory Requirements:**
Gradient Descent:
- Typically requires less memory as it only involves storing the gradient.
Newton's Method:
- Requires additional memory to store the Hessian matrix and its inverse.

6. **Robustness:**
Gradient Descent:
- Generally more robust and applicable to a wide range of optimization problems.
Newton's Method:
- Can be sensitive to the initial guess, and the Hessian matrix must be positive definite for convergence.

7. **Regularization:**
Gradient Descent:
- More straightforward to incorporate regularization techniques.
Newton's Method:
- May require additional modifications for effective regularization.

In practice, gradient descent variants, such as stochastic gradient descent (SGD) and its extensions (e.g., Adam, RMSprop), are more commonly used for training deep neural networks due to their computational efficiency and scalability. Newton's method is often reserved for smaller-scale problems or problems where the benefits of second-order optimization outweigh the computational costs. Also by considering the characteristics such as Computational Complexity, Memory Requirements, Robustness, and Regularization, we can say that Gradient Decent is more likely to e used.

Question 6

a)

$$E[\frac{\partial J_1}{\partial w_k}] = \frac{\partial E[J_1]}{\partial w_k} =$$

$$\frac{1}{2}\frac{\partial\left(E[y_d^2] + E[(\sum_{i=1}^n w_i x_i)^2] + E[(\sum_{i=1}^n \delta_i x_i)^2] - 2E[y_d \sum_{i=1}^n w_i x_i] - 2E[y_d \sum_{i=1}^n \delta_i x_i] + 2E[\sum_{i=1}^n w_i x_i \sum_{i=1}^n \delta_i x_i]\right)}{\partial w_k}$$

$$= \frac{1}{2}\frac{\partial\left(y_d^2 + (\sum_{i=1}^n w_i x_i)^2 + E[(\sum_{i=1}^n \delta_i x_i)^2] - 2y_d \sum_{i=1}^n w_i x_i - 2y_d \sum_{i=1}^n E[\delta_i] x_i + 2\sum_{i=1}^n w_i x_i \sum_{i=1}^n E[\delta_i] x_i\right)}{\partial w_k}$$

By knowing the fact that

$$E[\sum_{i=1}^{n} \delta_i x_i] = \sum_{i=1}^{n} E[\delta_i] x_i = 0$$

We can conclude that

$$E[(\sum_{i=1}^{n} \delta_i x_i)^2] = Var(\sum_{i=1}^{n} \delta_i x_i) + (E[\sum_{i=1}^{n} \delta_i x_i])^2 = Var(\sum_{i=1}^{n} \delta_i x_i)$$

By independence of $\delta_i s$ we have:

$$E[(\sum_{i=1}^{n} \delta_i x_i)^2] = \sum_{i=1}^{n} Var(\delta_i) x_i^2 = \sum_{i=1}^{n} \alpha w_i^2 x_i^2$$

By applying the equality above and $E[\delta_i] = 0$ we have:

$$E[\frac{\partial J_1}{\partial w_k}] = \frac{1}{2} \frac{\partial \Big((y_d - \sum_{i=1}^{n} w_i x_i)^2 + \sum_{i=1}^{n} \alpha w_i^2 x_i^2 \Big)}{\partial w_k} = \frac{1}{2} \Big(-2x_k (y_d - \sum_{i=1}^{n} w_i x_i) + 2\alpha x_k^2 w_k \Big) = x_k (-y_d + \sum_{i=1}^{n} w_i x_i) + \alpha x_k^2 w_k$$

So:

$$E[\frac{\partial J_1}{\partial w_k}] = x_k (-y_d + \sum_{i=1}^{n} w_i x_i) + \alpha x_k^2 w_k$$

b)

The first term $\big(x_k(-y_d + \sum_{i=1}^{n} w_i x_i)\big)$ is same as the usual derivative of error that we see in typical problems.

But the second term is similar to a weighted L2 regularization with te small difference that each w_i will be updated with a multiplier which is its corresponding input(x_i^2).