Written Assignment

1 Optimization [14pt]

1.1 Mini-Batch Stochastic Gradient Descent (SGD)

1.1.1 Minimum Norm Solution [10pt]

Recall Question 1.3.2 from Homework 1. For an overparameterized linear model, gradient descent starting from zero initialization finds the unique minimum norm solution $\mathbf{w}^*$ such that $X\mathbf{w}^* = t$. Let $\mathbf{w}_0 = 0$, $d > n$. Assume mini-batch SGD also converges to a solution $\hat{\mathbf{w}}$ such that $X\hat{\mathbf{w}} = t$. Show that mini-batch SGD solution is identical to the minimum norm solution $\mathbf{w}^*$ obtained by gradient descent, i.e., $\hat{\mathbf{w}} = \mathbf{w}^*$.

Hint: Be more specific as to what other solutions? Or is x_j or B contained in span of X? Do the update steps of mini-batch SGD ever leave the span of X?

For some $\mathbf{x}_j$, the gradient is as follows,

$$\nabla_{\hat{\mathbf{w}}_t}\mathcal{L}(\mathbf{x}_j, \hat{\mathbf{w}}_t) = \nabla_{\hat{\mathbf{w}}_t}\left(\frac{1}{b}\|\hat{\mathbf{w}}_t^T\mathbf{x}_j - t\|_2^2\right) = \frac{2}{b}\cdot\mathbf{x}_j\underbrace{(\hat{\mathbf{w}}_t^T\mathbf{x}_j - t)}_{\in\mathbb{R}}$$

$$\nabla_{\hat{\mathbf{w}}_t}\mathcal{L}(\mathbf{x}_j, \hat{\mathbf{w}}_t) = c\cdot\mathbf{x}_j \quad \text{for } c\in\mathbb{R}$$

Since $w_0 = 0$, and all the updates are constant multiples of $\mathbf{x}$ which is a row in X, it means $\hat{\mathbf{w}} \in \text{span}(X)$. Equivalently,

$$\hat{\mathbf{w}} = X^T a \quad \text{for } a\in\mathbb{R}^n.$$

This was the same assumption we used in Question 1.3.2 from Homework 1 to obtain $\hat{\mathbf{w}} = X^T(XX^T)^{-1}\mathbf{t}$. Now to show that the solution is unique, lets take the linear regression solution w such that $Xw = t$. Examining the norms via inner product,

$$\langle\hat{\mathbf{w}} - w, \hat{\mathbf{w}}\rangle = (\hat{\mathbf{w}} - w)^T X^T a = (X\hat{\mathbf{w}} - Xw)^T a = (t-t)^T a = 0$$

The Pythagorean identity tells us that,

$$\langle\hat{\mathbf{w}} - w, \hat{\mathbf{w}}\rangle = 0 \implies \|\hat{w} - w\|^2 + \|\hat{w}\|^2 = \|w\|^2.$$

$$\implies \|\hat{\mathbf{w}}\|^2 \le \|w\|^2 \implies \|\hat{\mathbf{w}}\| \le \|w\|$$

Thus the mini-batch SGD solution is the minimum norm solution, as required.

1.2 Adaptive Methods

1.2.1 Minimum Norm Solution [4pt]

Consider the overparameterized linear model ($d > n$) for the loss function defined in Section 1. Assume the RMSProp optimizer converges to a solution. Provide a proof or counterexample for whether RMSProp always obtains the minimum norm solution. Hint: Compute a simple 2D case. Let $x_1 = [2, 1]$,$w_0 = [0, 0]$, $t = [2]$.

Computing the example from the hint using the following result gives the wolution $w = \begin{bmatrix}0.66 & 0.66\end{bmatrix}$ with a norm of 0.943.

```
import numpy as np
X, w, t = np.array([[2, 1]]), np.array([0.0, 0.0]), np.array([2])

lr, beta, epsilon = 0.01, 0.9, 1e-8
v = np.array([0.0, 0.0])
history = []

for i in range(200):
    error = X @ w - t
    grad = 2 * X.T @ error
    v = beta * v + (1 - beta) * (grad ** 2)
    w = w - (lr / (np.sqrt(v) + epsilon)) * grad
    loss = np.sum(error**2)
    history.append((i + 1, w.copy(), loss))
epoch, weights, loss = history[-1]
print(f'Epoch {epoch} Results:\nWeights = {weights}\nLoss = {loss}')
print('Norm:', np.sqrt(np.sum(weights**2)))
```

The pseudo-inverse, as derived in assignment 1 is as follows.

$$X^+ = X^T(XX^T)^{-1}$$

$$XX^T = \begin{bmatrix}2 & 1\end{bmatrix}\begin{bmatrix}2\\1\end{bmatrix} = 2\cdot 2 + 1\cdot 1 = 5 \qquad (XX^T)^{-1} = \frac{1}{5}$$

$$X^+ = X^T\cdot\frac{1}{5} = \begin{bmatrix}2\\1\end{bmatrix}\cdot\frac{1}{5} = \begin{bmatrix}0.4\\0.2\end{bmatrix}$$

$$w = X^+t = \begin{bmatrix}0.4\\0.2\end{bmatrix}\cdot 2 = \begin{bmatrix}0.8\\0.4\end{bmatrix}$$

Which has a norm of $\sqrt{0.8^2+0.4^2} = 0.894$, which is less than that of the RMSProp solution. Therefore, we have found a contradiction, so the RMSProp solution is not always the minimum norm solution.

2 Gradient-based Hyper-parameter Optimization [24pt]

2.1 Optimal Learning Rates

2.1.1 [8pt]

Write down the expression of $\mathbf{w}_1$ in terms of $\mathbf{w}_0$, η, $\mathbf{t}$ and X. Then use the expression to derive the loss $\mathcal{L}_1$ in terms of η. Hint: If the expression gets too messy, introduce a constant vector $\mathbf{a} = X\mathbf{w}_0 - \mathbf{t}$

Using the known loss graident,

$$\nabla_{\mathbf{w}}\mathcal{L} = \frac{2}{n}X^T(X\mathbf{w} - \mathbf{t}),$$

we can write the expression for $\mathbf{w}_1$ as follows.

$$\mathbf{w}_1 = \mathbf{w}_0 - \eta\cdot\frac{2}{n}X^T(X\mathbf{w}_0 - \mathbf{t})$$

Using the substitution,

$$\mathbf{w}_1 = \mathbf{w}_0 - \eta \cdot \frac{2}{n} X^T \mathbf{a}.$$

We can now expand the loss function to include η and write the loss in terms of η.

$$\begin{aligned}
\mathcal{L}_1 =& \frac{1}{n} \|X\mathbf{w}_1 - \mathbf{t}\|_2^2 \\
\mathcal{L}_1 =& \frac{1}{n} \|X\left(\mathbf{w}_0 - \eta \cdot \frac{2}{n} X^T \mathbf{a}\right) - \mathbf{t}\|_2^2 \\
\mathcal{L}_1 =& \frac{1}{n} \|X\mathbf{w}_0 - \eta \cdot \frac{2}{n} X^T \mathbf{a} - \mathbf{t}\|_2^2 \\
\mathcal{L}_1 =& \frac{1}{n} \|\mathbf{a} - \eta \cdot \frac{2}{n} XX^T \mathbf{a}\|_2^2
\end{aligned}$$

As required.

2.1.3 [4pt]

Write down the derivative of $\mathcal{L}_1$ w.r.t. η and use it to find the optimal learning rate η^* that minimizes the loss after one GD iteration. Show your work.

We can compute the loss w.r.t. η to compute the GD updated η^*.

$$\begin{aligned}
\nabla_\eta \mathcal{L}_1 =& \frac{\partial}{\partial \eta}\left(\frac{1}{n} \|\mathbf{a} - \eta \cdot \frac{2}{n} XX^T \mathbf{a}\|_2^2\right) \\
\nabla_\eta \mathcal{L}_1 =& \frac{1}{n}\frac{\partial}{\partial \eta}\left(\|\mathbf{a} - \eta \cdot \frac{2}{n} XX^T \mathbf{a}\|_2^2\right) \\
\nabla_\eta \mathcal{L}_1 =& \frac{2}{n}\left(\mathbf{a} - \eta \cdot \frac{2}{n} XX^T \mathbf{a}\right)\frac{\partial}{\partial \eta}\left(\mathbf{a} - \eta \cdot \frac{2}{n} XX^T \mathbf{a}\right) \\
\nabla_\eta \mathcal{L}_1 =& \frac{2}{n}\left(\mathbf{a} - \eta \cdot \frac{2}{n} XX^T \mathbf{a}\right)\left(-\frac{2}{n} XX^T \mathbf{a}\right) \\
\nabla_\eta \mathcal{L}_1 =& -\frac{4}{n^2}\mathbf{a}^T XX^T \mathbf{a} + \eta \cdot \frac{8}{n^3}\mathbf{a}^T XX^T XX^T \mathbf{a}
\end{aligned}$$

We can solve for η^* by setting $\nabla_\eta \mathcal{L}_1 = 0$ and solving for η.

$$\nabla_\eta \mathcal{L}_1 = 0 = -\frac{4}{n^2}\mathbf{a}^T XX^T \mathbf{a} + \eta \cdot \frac{8}{n^3}\mathbf{a}^T XX^T XX^T \mathbf{a}$$

$$\eta \cdot \frac{8}{n^3}\mathbf{a}^T XX^T XX^T \mathbf{a} = \frac{4}{n^2}\mathbf{a}^T XX^T \mathbf{a}$$

$$\eta \cdot \frac{2}{n}\mathbf{a}^T XX^T XX^T \mathbf{a} = \mathbf{a}^T XX^T \mathbf{a}$$

$$\eta (XX^T \mathbf{a})^2 = \frac{n}{2}(X^T \mathbf{a})^2$$

$$\eta = \frac{n}{2}\frac{(X^T \mathbf{a})^2}{(XX^T \mathbf{a})^2}$$

As required.

2.2 Weight decay and L2 regularization

2.2.1 [8pt]

Write down two expressions for $\mathbf{w}_1$ in terms of $\mathbf{w}_0$, η, $\mathbf{t}$, λ, $\tilde{\lambda}$, and X. The first one using $\tilde{\mathcal{L}}$, the second with $\mathcal{L}$ and weight decay.

The expressions for $\mathbf{w}_1$ will take the form

$$\mathbf{w}_1 = (1-\lambda)\mathbf{w}_0 - \eta\nabla_{\mathbf{w}_0}\mathcal{L}$$

and

$$\mathbf{w}_1 = \mathbf{w}_0 - \eta\nabla_{\mathbf{w}_0}\tilde{\mathcal{L}}.$$

So let us start by computing the loss gradients. For the weight decay case, the gradient is the same as we have seen many times before.

$$\nabla_{\mathbf{w}_0}\mathcal{L} = \frac{2}{n}X^T(X\mathbf{w}_0 - \mathbf{t})$$

In the regularized case, it is similar,

$$\begin{aligned}
\nabla_{\mathbf{w}_0}\tilde{\mathcal{L}} =& \frac{\partial}{\partial \mathbf{w}_0}\left(\frac{1}{n}\|X\mathbf{w}_0 - \mathbf{t}\|_2^2 + \tilde{\lambda}\|\mathbf{w}_0\|_2^2\right)\\
\nabla_{\mathbf{w}_0}\tilde{\mathcal{L}} =& \frac{\partial}{\partial \mathbf{w}_0}\left(\frac{1}{n}\|X\mathbf{w}_0 - \mathbf{t}\|_2^2\right) + \frac{\partial}{\partial \mathbf{w}_0}\left(\tilde{\lambda}\|\mathbf{w}_0\|_2^2\right)\\
\nabla_{\mathbf{w}_0}\tilde{\mathcal{L}} =& \frac{2}{n}X^T(X\mathbf{w}_0 - \mathbf{t}) + 2\tilde{\lambda}\mathbf{w}_0
\end{aligned}$$

Now, we can substitute these computations back into their update rules.
For weight decay,

$$\mathbf{w}_1 = (1-\lambda)\mathbf{w}_0 - \eta\cdot\frac{2}{n}X^T(X\mathbf{w}_0 - \mathbf{t}).$$

and for L2-regularization,

$$\begin{aligned}
\mathbf{w}_1 =& \mathbf{w}_0 - \eta\left(\frac{2}{n}X^T(X\mathbf{w}_0 - \mathbf{t}) + 2\tilde{\lambda}\mathbf{w}_0\right)\\
\mathbf{w}_1 =& (1-2\eta\tilde{\lambda})\mathbf{w}_0 - \eta\cdot\frac{2}{n}X^T(X\mathbf{w}_0 - \mathbf{t})
\end{aligned}$$

As required.

2.2.2 [4pt]

How can you express $\tilde{\lambda}$ (corresponding to L2 loss) so that it is equivalent to λ (corresponding to weight decay)?
Hint: Think about how you can express $\tilde{\lambda}$ in terms of λ and another hyperparameter.

The above expression brings us closer to this derivation, we must pick $\tilde{\lambda}$ to be equivalent to the weight decay coefficent.

$$\begin{aligned}
1-\lambda =& 1-2\eta\tilde{\lambda}\\
\lambda =& 2\eta\tilde{\lambda}\\
\tilde{\lambda} =& \frac{\lambda}{2\eta}
\end{aligned}$$

Thus we pick $\tilde{\lambda} = \frac{\lambda}{2\eta}$ so that it is equivalent to weight decay.

3 Trading off Resources in Neural Net Training [16pt]

3.1 Effect of batch size

3.1.1 Batch size vs. learning rate

(a) [4pt] As batch size increases, how do you expect the optimal learning rate to change? Briefly explain in 2-3 sentences. (Hint: Think about how the minibatch gradient noise change with B.)

For larger batches, the noise is smaller as it is has 0 mean and B increases. Thus the optimal learning rate can increase as it more accurately responding to the loss, allowing the optimizer to take more confident steps towards the minima. Starting from a lower batch size will see greater improvements.

3.1.2 Training steps vs. batch size

(a) [4pt]
For the three points (A, B, C) on Figure 1, which one has the most efficient batch size (in terms of best resource and training time trade-off)? Assume that you have access to scalable (but not free) compute such that minibatches are parallelized efficiently. Briefly explain in 1-2 sentences.

Point C is the most efficient. Increasing the batch size further from C does not yield a proportional decrease in required train steps. Decreasing the batch size requires more train steps in an almost one-to-one linear fashion which tracks but is not the most efficient.

(b) [4pt]

Point A:

- Regime: noise dominated
- Potential way to accelerate training: seek parallel compute

Point B:

- Regime: curvature dominated
- Potential way to accelerate training: use higher order optimizers

3.2 Model size, dataset size and compute

(a) [4pt] The best option would be to increase the model size. These scaling laws for language models indicate that, with additional compute resources, increasing the model size yields better test performance than merely training longer or increasing the batch size. Larger models have a greater capacity to capture complex patterns in data, which helps reduce test loss more effectively when trained from scratch with sufficient compute.

4 Pooling and Upsampling

4.2 [4pt]

Run main training loop of PoolUpsampleNet. This will train the CNN for a few epochs using the cross-entropy objective. It will generate some images showing the trained result at the end. Do these results look good to you? Why or why not?

The test loss is continually decreasing with train loss and the accuracy is at 41.3% and steadily increasing. This indicates that the model is still learning and is learning well, but is not yet converged and can be improved. So the results look good and demonstrate that training is going well, but we need to keep training in order to learn a more useful model.

4.3 [8pt]

Compute the number of weights, outputs, and connections in the model, as a function of **NIC**, **NF** and **NC**. Compute these values when each input dimension (width/height) is doubled. Report all 6 values.

Total number of weights:

$$NIC \cdot NF \cdot k^2 + NF + 2NF + 2NF^2 \cdot k^2 + 2NF + 4NF + 2NF^2 \cdot k^2$$

$$+NF + 2NF + NF \cdot NC \cdot k^2 + NC + 2NC + NC^2 \cdot k2 + NC$$

$$= k^2 \cdot (NIC \cdot NF) + 12NF + 4k^2 \cdot NF^2 + k^2 \cdot (NF \cdot NC) + 4NC + k^2 \cdot NC^2$$

$$\boxed{\text{num\_weights} = k^2 \cdot (NIC \cdot NF + 4NF^2 + NF \cdot NC + NC^2) + 12NF + 4NC}$$

Total number of outputs:

$$NF322 + NF \cdot 162 + NF \cdot 162 + 2NF \cdot 162 + 2NF \cdot 162 + NC \cdot 162 + NC \cdot 322 + NC \cdot 322 + NC \cdot 322$$

$$\boxed{\text{num\_outputs} = 2880NF + 3328NC}$$

Total number of connections:

$$82 + 2NF \cdot 82 + NF \cdot 82 + NF \cdot 162 + NFk2 \cdot NIC \cdot NF \cdot 322 + NF \cdot 322 + NF \cdot 162$$

$$+k^2 \cdot NF \cdot 2NF \cdot 162 + 2NF \cdot 162 + 2NF \cdot 82 + k^2 \cdot 2NF \cdot NF \cdot 82$$

$$+NF \cdot 162 + NF \cdot 162 + k^2 \cdot NF \cdot NC \cdot 162 + NC \cdot 322 + NC \cdot 322 + k2 \cdot NC2 \cdot 322$$

$$= 1024k^2 \cdot (NIC \cdot NF) + 2432NF + 640k^2 \cdot NF^2 + 256k^2 \cdot (NF \cdot NC) + 2048NC + 1024k^2 \cdot NC^2$$

$$\boxed{\text{num\_conns} = k^2 \cdot (1024(NIC \cdot NF) + 640NF^2 + 256(NF \cdot NC) + 1024NC^2) + 2432NF + 2048NC}$$

When each input dimension (width/height) is doubled, the number of weights remains the same due to convolution. The number of outputs and the number of connections will now be scaled by 4.
Total number of weights:

$$\boxed{\text{num\_weights} = k^2 \cdot (NIC \cdot NF + 4NF^2 + NF \cdot NC + NC^2) + 12NF + 4NC}$$

Total number of outputs:

$$\boxed{\text{num\_outputs} = 4\,(2880NF + 3328NC)}$$

Total number of connections:

$$\boxed{\text{num\_conns} = 4\left[k^2 \cdot (1024(NIC \cdot NF) + 640NF^2 + 256(NF \cdot NC) + 1024NC^2) + 2432NF + 2048NC\right]}$$

5 Strided and Transposed Dilated Convolutions

5.2 [4pt]

Train the model for at least 25 epochs using a batch size of 100 and a kernel size of 3. Plot thetraining curve, and include this plot in your write-up.

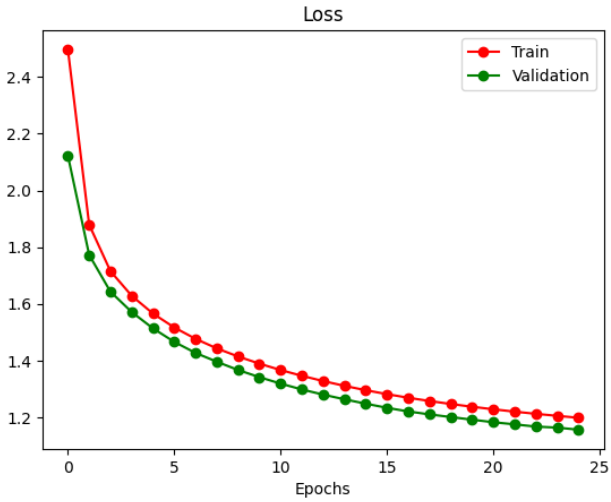

Figure 1: Train and Test Loss after 25 epochs with $k = 3$ and batch size of 100.

5.3 [4pt]

How do the results compare to Section 4? Does the `ConvTransposeNet` model result in lower validation loss than the `PoolUpsampleNet`? Why may this be the case?

The `ConvTransposeNet` performs better with a validation loss of 1.15 and validation accuracy of 54.8%. The `PoolUpsampleNet` is close behind with a validation loss of 1.58 and validation accuracy of 41.3%. One reason for this is the use of the `ConvTranspose2d` layers in the second one which upsample better than the interpolation of the first model. Another reason could be the use of stride as a replacement for maxpooling and this could possible perform a better implicit pooling.

5.4 [4pt]

How would the padding parameter passed to the first two nn.Conv2d layers, and the padding and output padding parameters passed to the nn.ConvTranspose2d layers, need to be modified if we were to use a kernel size of 4 or 5 (assuming we want to maintain the shapes of all tensors shown in Figure 4b)? Note: PyTorch documentation for nn.Conv2d and nn.ConvTranspose2d includes equations that can be used to calculate the shape of the output tensors given the parameters.

Using the equations in the documentation, we get the following.

- For $k = 4$: padding $= 1$ and output padding $= 0$
- For $k = 4$: padding $= 2$ and output padding $= 1$