CSE 547 - Assignment 2

Philip Pham

May 3, 2018

Problem 0

List of collaborators: I have not collaborated with anyone.

List of acknowledgements: None.

Certify that you have read the instructions: I have read and understood these policies.

Problem 1: Generalization, Streaming, and SGD

In class, we examined using Stochastic Gradient Descent (SGD) for empirical loss minimization, where we have an N sized training set $\mathcal{T}$. The empirical loss considered was:

$$F(w) = \frac{1}{N} \sum_{(x,y)\in\mathcal{T}} l\left(w, (x,y)\right). \tag{1}$$

Here, gradient descent for the function F is the algorithm:

1. Initialize at some point $w^{(0)}$.
2. Sample (x, y) uniformly at random from the set $\mathcal{T}$.
3. Update the parameters:

$$w^{(k+1)} = w^{(k)} - \eta_k \cdot \nabla l\left(w^{(k)}, (x,y)\right), \tag{2}$$

   and go back to 2.

We provided guarantees assuming that F was smooth and the gradients in our training set were uniformly bounded, $\|\nabla l\left(w, (x,y)\right)\| \leq B$.

However, in practice, we care about generalization, that is, statements on how well we do on the underlying distribution. Define:

$$\mathcal{L}(w) = \mathbb{E}_{(x,y)\in\mathcal{D}} l\left(w, (x,y)\right), \tag{3}$$

where $\mathcal{D}$ is the underlying distribution.

Suppose we sought a point where $\|\nabla\mathcal{L}\|^2$ was small. Obtaining this quantity to be small even in expectation would be acceptable for this problem Assume that $\mathcal{L}$ is smooth and that the gradients are uniformly bounded, $\|\nabla l\left(w, (x,y)\right)\| \leq B$ for all parameters and all possible points (x, y) (under $\mathcal{D}$).

1. Assume we have sampling access to our underlying distribution $\mathcal{D}$. Explain how we can make $\|\nabla\mathcal{L}(w)\|^2$ small in expection. What can you guarantee if you obtain m samples and how would you do this?

Solution

If we sample from $\mathcal{D}$ (as opposed to from $\mathcal{T}$ in Step 2 in the original algorithm), then we can guarantee that $\|\nabla\mathcal{L}(w)\|^2$ is small in expectation. The proof is identical to that of Theorem 1.2 from Lecture 6 in class but we replace $\mathcal{T}$ with $\mathcal{D}$. Thus, each gradient becomes a sample of $\nabla\mathcal{L}(w)$.

2. Suppose we contruct an N sized training set $\mathcal{T}$, where each point is sampled under $\mathcal{D}$; then we construct the empirical loss funciton $F(w)$; then we run SGD on F for K steps (suppose $K \geq N$). Is there an argument on this procedure that implies something non-trivial (and technically correct) about $\|\nabla\mathcal{L}(w)\|^2$, even in expectation?

Solution

If we let $K = N$, and sample each point from $\mathcal{T}$ exactly once, we can get the guarantee

$$\min_{k<K} \mathbb{E}\left[\left\|\nabla\mathcal{L}\left(w^{(k)}\right)\right\|_2^2\right] = B\sqrt{\frac{2\left(F\left(w^{(0)}\right) - F_*\right)L}{K}} \tag{4}$$

as in Theorem 1.2 from Lecture 6. This works because using each sample from $\mathcal{T}$ exactly once is the same as sampling from $\mathcal{D}$ K times. However, this falls apart as soon as $K > N$ because now our population is $\mathcal{T}$, not $\mathcal{D}$, so we can no longer make any guarantees about $\|\nabla\mathcal{L}(w)\|^2$.

Problem 2: GD versus Adaptive GD

Let us compare gradient descent (GD), with a constant stepsize η, to Adagrad on two simple one-dimensional objective functions. Adagrad will use the stepsize at iteration k where:

$$\eta_k = \frac{C}{\sqrt{\sum_{j=0}^{k}\|\nabla F\left(w^{(j)}\right)\|^2}}. \tag{5}$$

1. Consider $F(w) = \frac{1}{2}w^2$. Let us start at $w_0 = 1$. For this problem a constant stepsize certainly makes sense for GD. Let $\eta = 3/4$ and $C = 3/4$.

   Analytically, characterize the convergence rates of GD and Adagrad with these parameter settings.

Solution

For GD, we can explicitly, write down the formula for $w^{(k)}$. Since $F'(w) = w$, $w^{(k+1)} = (1-C)w^{(k)}$, so $w^{(k)} = (1-C)^k = (1/4)^k$. So, $\nabla F\left(w^{(k)}\right) = (1/4)^k$.

The global minimum occurs at $w^* = 0$. Since $\eta_k \leq C$ with equality only when $k = 0$, Adagrad converges strictly slower.

However, it does no converge much slower. We can show that $w^{(k)} \leq (1/3)^k$. We show this by strong induction. $w^{(0)} = 0$ satisifies this. Now, we have that

$$\begin{aligned} w^{(k+1)} &= w^{(k)} \left(1 - \frac{3/4}{\sqrt{\sum_{i=0}^{k} \left(w^{(i)}\right)^2}}\right) \\ &\leq w^{(k)} \left(1 - \frac{3/4}{\sqrt{\sum_{i=0}^{k} \left(\frac{1}{3}\right)^2}}\right) \\ &\leq w^{(k)} \left(1 - \frac{3/4}{\sqrt{9/8}}\right) \approx 0.2928932 w^{(k)} \\ &\leq \left(\frac{1}{3}\right)^{k+1}. \end{aligned}$$

Thus, we have that $(1/4)^k \leq w^{(k)} \leq (1/3)^k$, so the two optimization procedures do not differ substantially. In this case, GD and Adagrad differ by at most a constant factor of 3/4.

2. Consider the non-smooth, convex function $F(w) = |w|$. Let us start with $w_0 = 1$. Use $\eta = 3/4$ and $C = 3/4$ as before.

   Analytically, characterize the convergence rates of GD and Adagrad with these parameter settings.

Solution

We have that

$$F'(w) = \begin{cases} 1, & w > 0; \\ -1, & w < 0; \end{cases} \tag{6}$$

and $F'(w)$ is not defined at $w = 0$.

In this case, GD would not converge. The sequence would look like

$$\begin{aligned} w^{(0)} &= 1 \\ w^{(1)} &= 1/4 \\ w^{(2)} &= -1/2 \\ w^{(3)} &= 1/4 \\ w^{(4)} &= -1/2 \\ w^{(5)} &= 1/4 \\ &\vdots \end{aligned}$$

Adagrad on the other hand is $O\left(1/\sqrt{T}\right)$, where T is the number of steps that we have taken. To see this, note that $\|\nabla F(w)\|_2^2 = 1$ We have two cases: (2a) $w^{(k)} > 0$ and (2b) $w^{(k)} < 0$.

(a) In this case, we'd have that

$$w^{(k+1)} = w^{(k)} - \frac{3}{4\sqrt{k+2}}.$$

(b) Here,

$$w^{(k+1)} = w^{(k)} + \frac{3}{4\sqrt{k+2}}.$$

Suppose that $0 < w^{(k)} < (3/4)/\sqrt{k+1}$. Certainly, this is true for $k = 1$, where $w^{(1)} = 1/4$. Then, we have that

$$-\frac{3}{4\sqrt{k+2}} < w^{(k+1)} < \frac{3}{4\sqrt{k+1}} - \frac{3}{4\sqrt{k+2}} < \frac{3}{4\sqrt{k+2}}. \tag{7}$$

We can similarly show the same inequality holds if $-(3/4)/\sqrt{k+1} < w^{(k)} < 0$. Thus, we have that $\left|w^{(k)}\right| < (3/4)/\sqrt{k+1}$ for all $k \geq 1$, so Adagrad converges at the rate $O\left(1/\sqrt{T}\right)$.

Problem 3: Understanding Non-Convexity

Newton's Method

The least squares problem can be written as:

$$\min_w L(w), \text{ where } L(w) \coloneqq \frac{1}{2N} \|Xw - Y\|^2, \tag{8}$$

where X is our $N \times d$ matrix and Y is our $N \times 1$ output vector. If we define:

$$\Sigma \coloneqq \frac{1}{N} X^\intercal X, \; u = \frac{1}{N} X^\intercal Y, \tag{9}$$

then, this expression can be written as

$$L(w) = \frac{1}{2} w^\intercal \Sigma w - u^\intercal w + \frac{1}{2N} \|Y\|^2. \tag{10}$$

Note that Σ is a positive-definite matrix. Assume that Σ is full rank.

Newton's method is typically thought of a being better than gradient descent due to it using second order information. At some w_0, the method first constructs a second-order Taylor's approximation around w_0, and then it makes the new iterate to be the point at which the gradient of this approximation is 0, that is,

$$w \leftarrow w - \left[\nabla^2 L(w)\right]^{-1} \nabla L(w). \tag{11}$$

1. Starting at some w_0 write out one step of the Newton's method in terms of Σ and u. If you take one step, what point do you get to?

Solution

Since Σ is positive-definite, we can write $\Sigma = PD^2P^\intercal$, where D is diagonal. Using that $w^\intercal \Sigma w = (DP^\intercal w)^\intercal (DP^\intercal w)$ and the chain rule, we have that

$$\begin{aligned}
\nabla L(w) &= D\left(L(w)\right)^\intercal \\
&= \left(\frac{1}{2}\left(2w^\intercal PD\right)\left(DP^\intercal\right) - u^\intercal\right)^\intercal \\
&= \left(w^\intercal \Sigma - u^\intercal\right)^\intercal \\
&= \Sigma w - u. \qquad (12)
\end{aligned}$$

Calculating the derivative again, we have obtain

$$\nabla^2 L\left(w\right) = \Sigma. \qquad (13)$$

Subtituting Equations 12 and 13 into Equation 11, we obtain

$$w_1 = w_0 - \Sigma^{-1}\left(\Sigma w_0 - u\right) = \boxed{\Sigma^{-1}u.} \qquad (14)$$

2. Comment on the loss of this point and how it compares to the minimal function value?

Solution

Equation 14 expands to

$$w_1 = \Sigma^{-1}u = \left(X^\intercal X\right)^{-1} X^\intercal Y, \qquad (15)$$

which is the solution to the least squares problem, so $L(w_1)$ is the global minimum function value.

Non-convex Case

Consider the objective function:

$$F(w) = \frac{1}{2}w^\intercal Aw - b^\intercal w + c, \qquad (16)$$

where A is a symmetric matrix, b is a vector, and c is a scalar. Our goal is to minimize the objective function. Assume that Σ is full rank.

Assume that A is still symmetric, yet suppose now that there is at least one negative eigenvalue. Hence, the problem is non-convex. Again, suppose you start at w_0 and take a step of Newton's method. Let this new point be w_1.

1. What is the point w_1 and the object value $F\left(w_1\right)$.

Solution

The calcuation for the gradient is similar to that in Equation 12:

$$\nabla F(w) = Aw - b. \tag{17}$$

It's notable that D will now have at least one complex entry on its diagonal. However, we only D for an intermediate step. The final solution will still contain real values.

Then, the Hessian is constant-valued:

$$\nabla^2 F(w) = A. \tag{18}$$

Substituting Equations 17 and 18 into Equation 11, we get that

$$w_1 = w_0 - A^{-1}\left(Aw_0 - b\right) = \boxed{A^{-1}b.} \tag{19}$$

The objective value is

$$\begin{aligned} F\left(w_1\right) &= \frac{1}{2}\left(b^\intercal A^{-1}\right) A\left(A^{-1}b\right) - b^\intercal\left(A^{-1}b\right) + c \\ &= \boxed{-\frac{1}{2}b^\intercal A^{-1}b + c.} \end{aligned} \tag{20}$$

2. What is the gradient at w_1?

Solution

Substituting Equation 19 into Equation 12, we get that

$$\nabla F(w_1) = Aw_1 - b = \boxed{0.} \tag{21}$$

3. What is the minimal object value of $F\left(\cdot\right)$? Did we achieve it?

Solution

The minimal objective value is $-\infty$. We did not achieve it, for Equation 20 is finite.

To see that the minimal objective value is $-\infty$. Let v be an eigenvector of A associated with eigenvalue $\lambda < 0$. Then, we have

$$\begin{aligned} F\left(v\right) &= \frac{1}{2}v^\intercal Av - b^\intercal v + c \\ &= \frac{\lambda}{2}\|v\|^2 - b^\intercal v + c. \end{aligned}$$

$|b^\intercal v| \leq \|b\|\|v\|$, so by increasing the magnitude of v, we can push the function value towards infinity.

4. Suppose you now are at w_1. In terms of the eigendecomposition $A = UDU^\intercal$, what are the directions of movement which *strictly* decrease the objective value from w_1? Why?

Solution

Let v be an eigenvector with a negative eigenvalue. Then, for any $\epsilon > 0$, we have that

$$\nabla F(w_1 + \epsilon v) = A(w_1 + \epsilon v) - b = \epsilon \lambda v,$$

where $\lambda < 0$ is the eigenvalue associated with v.

By definition, $D_v F(w_1 + \epsilon v) = \nabla F(w_1 + \epsilon v) \cdot v = \epsilon \lambda \|v\|^2 < 0$, so F is always decreasing in the direction v relative to w_1. This will be true for any v that is a linear combination of eigenvectors with negative eigenvalues.

Problem 4: Empirical Optimization

We will now consider the multi-label classification problem. In the multi-label problem, there are multiple labels that could be "on" for each input x. You will use either the square loss or the binary logistic loss and consider training two models, namely (i) a linear model and (ii) a multi-layer perceptron (MLP) with a number of hidden nodes that you will tune.

You will try out three methods in each of the following: (1) SGD with a mini-batch size that you tune. You will use the same minibatch size for the other algorithms; (2) try out PolyakâĂŹs "heavy ball method" (aka momentum) or NesterovâĂŹs accelerated gradient descent (NAG); and (3) either Adagrad or Adam. You must tune all the parameters of these methods.

The dataset contains 18 total categories with a number of categories for each supercategory (vehicle or animal). In the dataset provided, each image contains objects of a single supercategory, say vehicle, and potentially multiple objects from the supercategory, such as car, boat, etc. In this exercise we shall build a classifier that learns to identify *all the categories of objects* present in each image, by optimizing either a square loss or a logistic loss objective. For the purposes of learning these classifiers, we shall use the dataset and features from the first homework. We shall also provide a larger version of this dataset since we need to train more parameters for this model.

The object function we choos to optimize is

$$L(w) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} l\left(y_{ij}, f_{ij}(w)\right), \tag{22}$$

where $f_{ij}(w) = w_j^\intercal x_i$ and $w_j \in \mathbb{R}^d$ is the jth column of $w \in \mathbb{R}^d \times \mathbb{R}^k$. Here, w is the linear model we wish to optimize over and $\lambda > 0$ is the strength of l_2 regularization. here l is the loss function:

- $l(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$ is the square error loss.
- $l(y, \hat{y}) = y \log(1 + \exp(-\hat{y})) + (1 - y)\log(1 + \exp(\hat{y}))$ is the logistic loss where the true label $y \in \{0, 1\}$.

Notice that we encode y_i as binary vector of length $k = 18$ (the number of categories) where a 1 indicates the presence of a category and 0 indicates the absence.

Determine which loss function works better for a linear classifier and use that loss throughout the question.

Figure 1: Some overfiiting likely occurred since training loss is significantly less that validation loss. Increasing λ made the model perform worse, however.

When using MLP, $f_{ij}(w) = \left\langle w_j^{(2)}, \text{relu}\left(w^{(1)}x_i\right)\right\rangle$, where $w^{(1)} \in \mathbb{R}^h \times \mathbb{R}^d$ are the weights in the first layer and h is the number of hidden nodes. Again $w_j^{(2)} \in \mathbb{R}^h$ is the jth column of $w^{(2)} \in \mathbb{R}^h \times \mathbb{R}^k$, the weights of the second layer.

SGD and Linear Regression

Now consider running stochastic gradient descent on $L(w)$.

1. What mini-batch size do you use? What stepsize did you use? What value of λ did you use? Specify your stepsize scheme if you chose to decay your stepsize. Which loss function did you find works better?

   Solution

   I used a mini-batch size of 8. My learning rate was 10^{-4}. I used $\lambda = 0.7$. I found logistic loss to work better.

2. Report your training and validation loss.

   Solution

   The results for the linear model can be seen in Figure 1 and Table 1. Training loss was 0.122861 and validation loss was 0.199776.

   The results for the MLP model can be seen in Figure 2 and Table 2. Training loss was 0.157085 and validation loss was 0.190817. I used $\lambda = 0.05$ as my regularization

	Average Precision Score	Loss
Training	0.871461	0.122861
Validation	0.563140	0.199776
Test	0.636070	0.177882

Table 1: Best average precision score and loss when training the linear model with SGD.

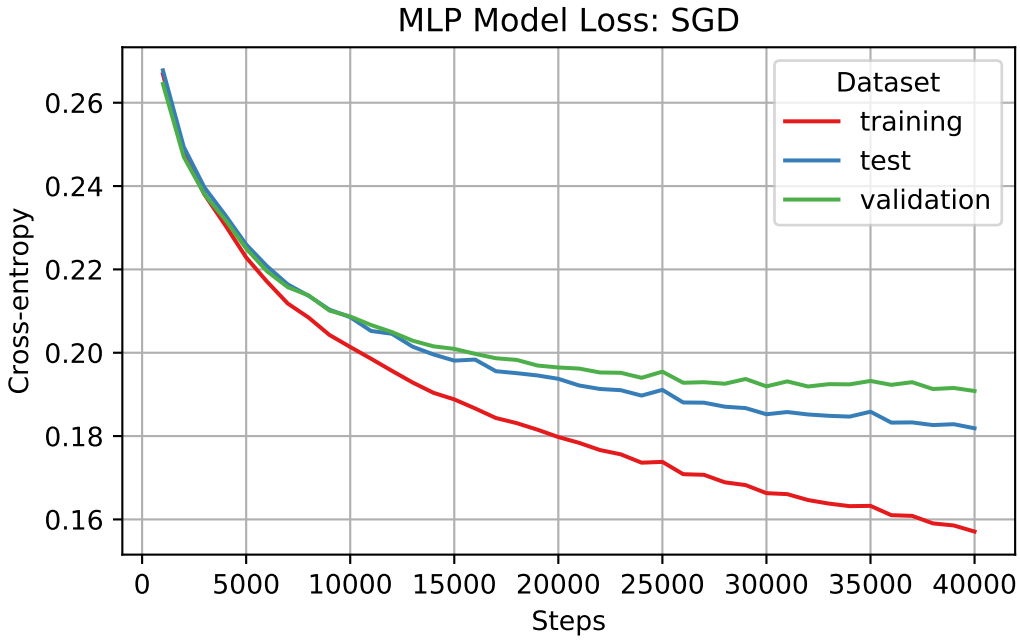

Figure 2: The model appears to not have converged yet.

parameter and 1 layer of 256 hidden units. The learning rate was 0.0005. Validation loss is better than the linear model, but test loss is worst.

3. Compute the average precision score in the same manner, and plot these quantities.

Solution

The results are in Tables 1 and 2. See Figures 3 and 4

4. Report the overall average precision score on the test set. Also report the average precision score for each of the 18 classes separately. Comment on how the AP scores

	Average Precision Score	Loss
Training	0.708156	0.157085
Validation	0.565152	0.190817
Test	0.596515	0.181875

Table 2: Best average precision score and loss when training the MLP model with SGD.

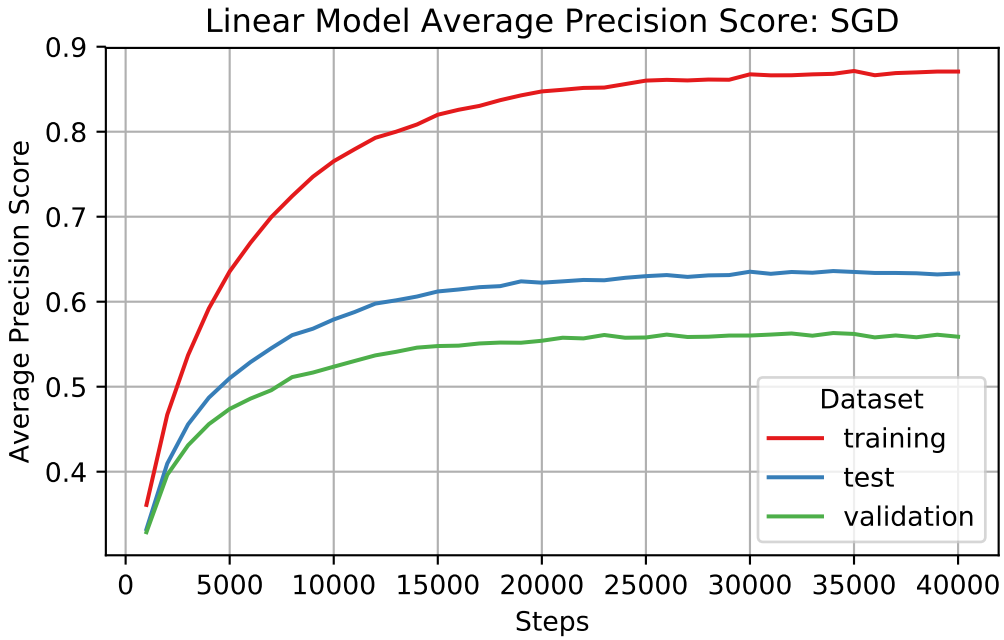

Figure 3: Average precision score by the number of steps for the linear model.

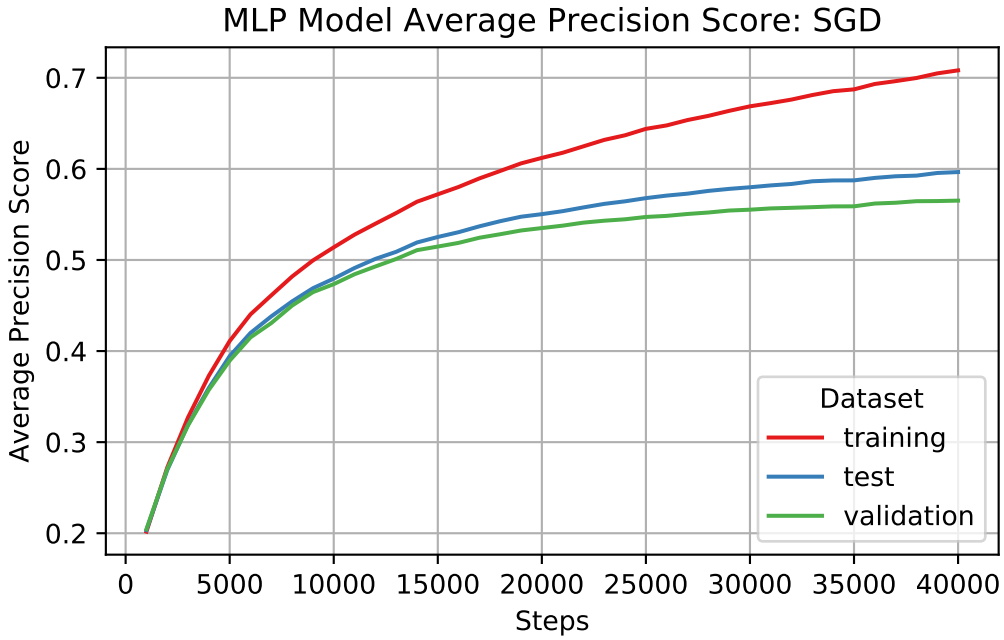

Figure 4: Average precision score by the number of steps for the multi-layer perceptron model.

	Average Precision Score	Loss
Training	0.759774	0.170063
Validation	0.521453	0.262688
Test	0.581486	0.232664

Table 3: Best average precision score and loss when training the linear model with SGD and Nesterov's momentum.

	Average Precision Score	Loss
Training	0.793293	0.141046
Validation	0.587959	0.188455
Test	0.629526	0.177171

Table 4: Best average precision score and loss when training the MLP model with SGD and Nesterov's momentum.

relate to the class imbalance.

Solution

Sorry, I ran out of time and didn't get this far :(.

Nesterov's method

I used Nesterov's method with a momentum parameter of 0.9. All other parameter were the same as last section. The results can be seen in Tables 3 and 4. The precision for the MLP model is notably better with an average precision score of 0.629526 on the test set.

The progress of the training runs can be seen in Figures 5, 6, 7, and 8. This method trained the fastest out of the three that I tried. In particular the linear model converged very quickly.

Adagrad

For the linear model, I used the same learning rate of 10^{-4}. The results were very similar to the previous results. The results can be seen in Tables 5.

For the MLP model, I had trouble getting convergence. I used Adagrad method with a learning rate of 0.0005. Judging from the results, this was likely, too small. Making it larger didn't seem to helper, either. Perhaps, step size decays, too, quickly. All other parameters were the same as last section. The results can be found in 6. The precision for the MLP model is notably better with an average precision score of 0.629526 on the test set.

The progress of the training runs can be seen in Figures 9, 10, 11, and 12.

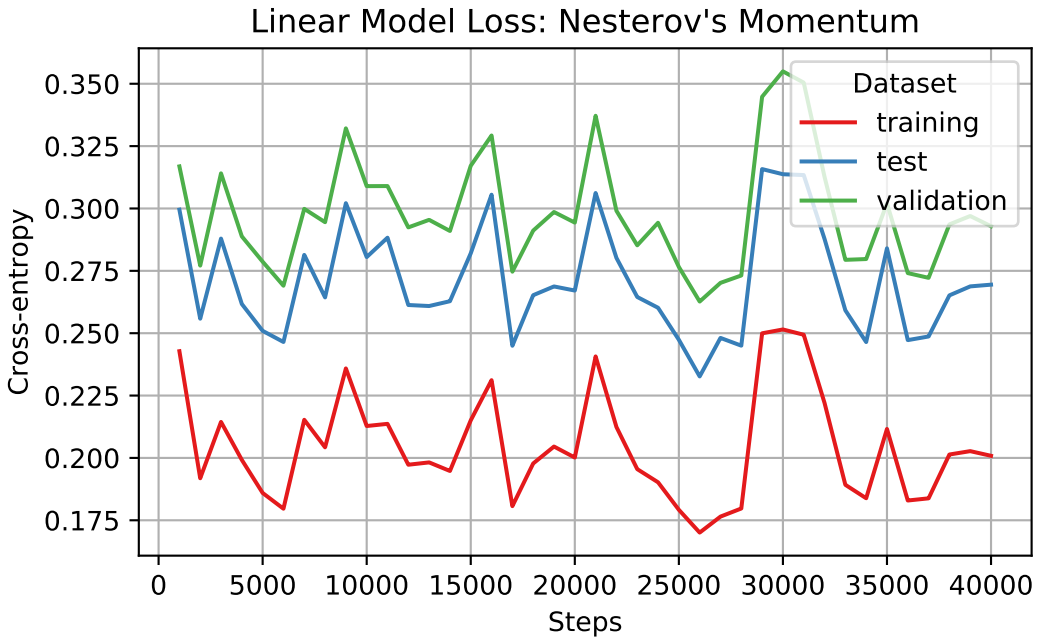

Figure 5: Evaluations on the datasets while training the linear model with Nesterov's momentum.

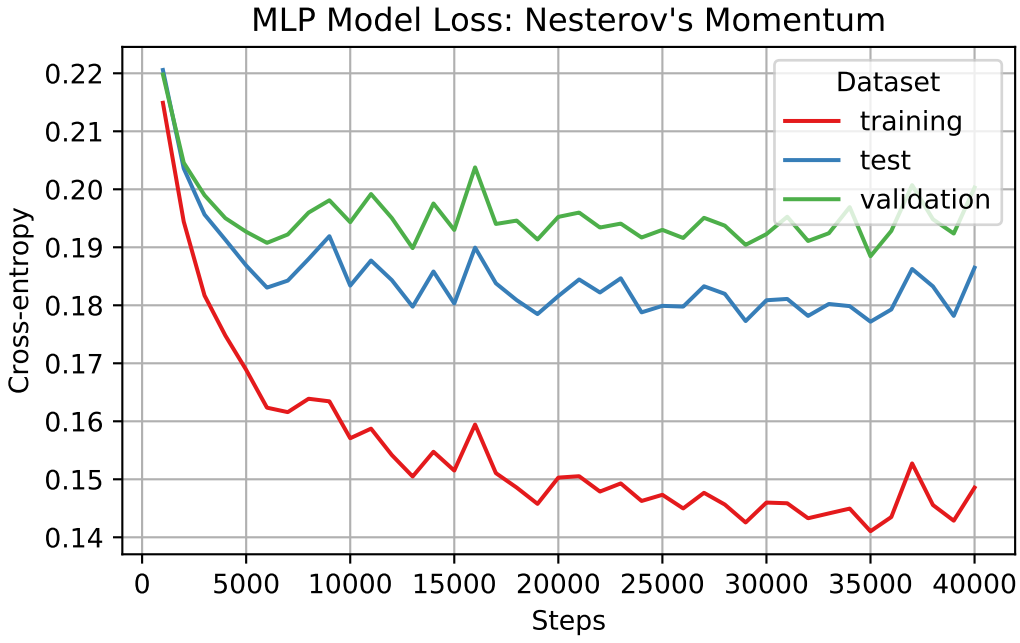

Figure 6: Evaluations on the datasets while training the MLP model with Nesterov's momentum.

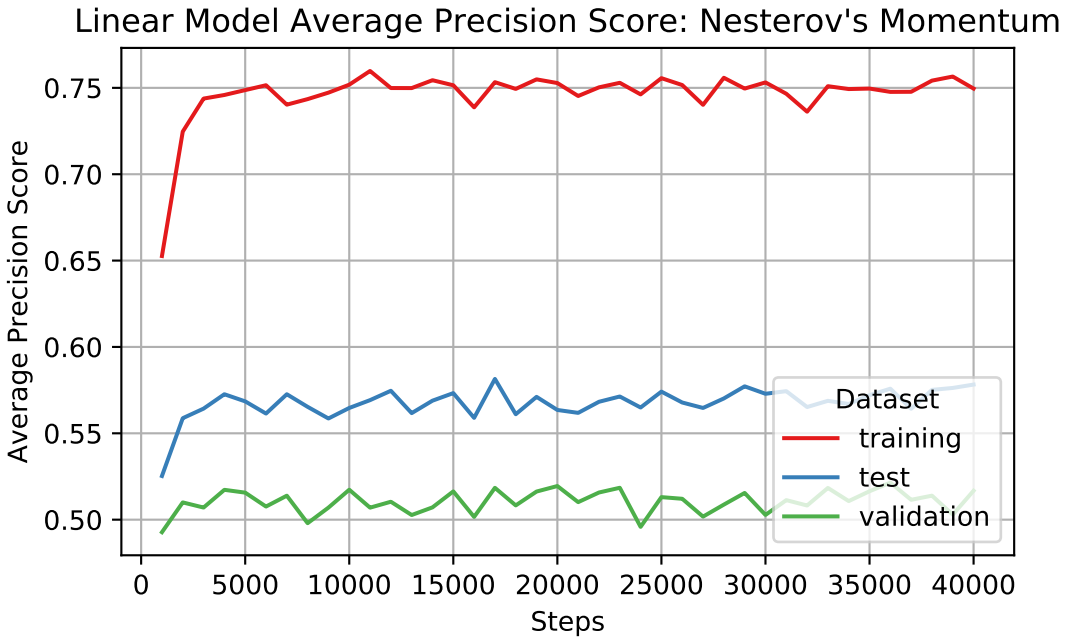

Figure 7: Average precision score by the number of steps for the linear model.

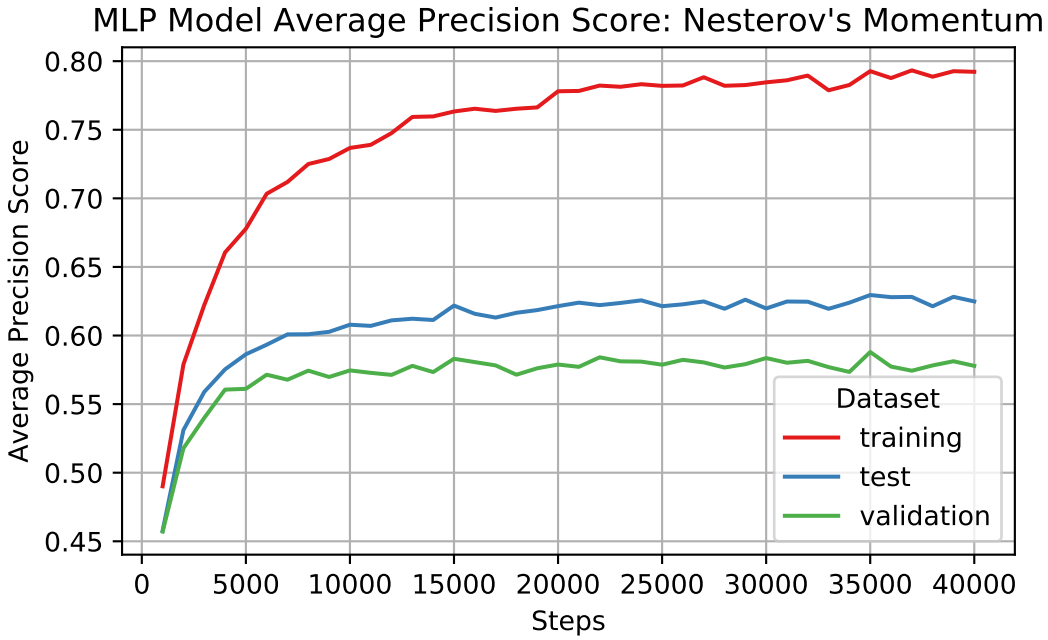

Figure 8: Average precision score by the number of steps for the multi-layer perceptron model.

	Average Precision Score	Loss
Training	0.820442	0.136633
Validation	0.542389	0.204523
Test	0.609165	0.184826

Table 5: Best average precision score and loss when training the linear model with SGD and Adagrad's momentum.

	Average Precision Score	Loss
Training	0.605249	0.186454
Validation	0.531518	0.199406
Test	0.546195	0.196277

Table 6: Best average precision score and loss when training the MLP model with SGD and Adagrad's momentum.

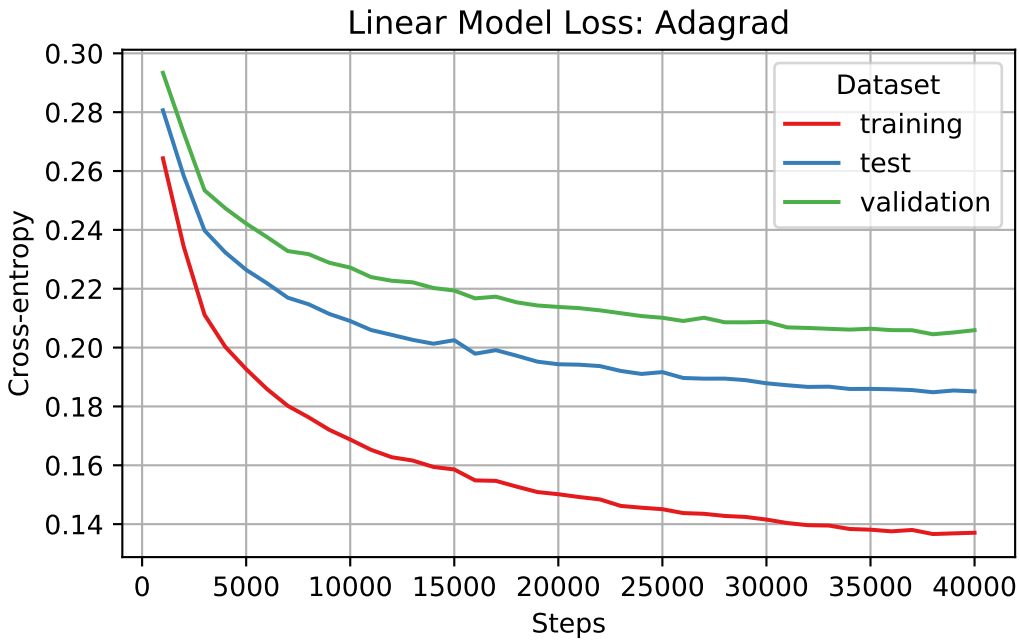

Figure 9: Evaluations on the datasets while training the linear model with Adagrad's momentum.

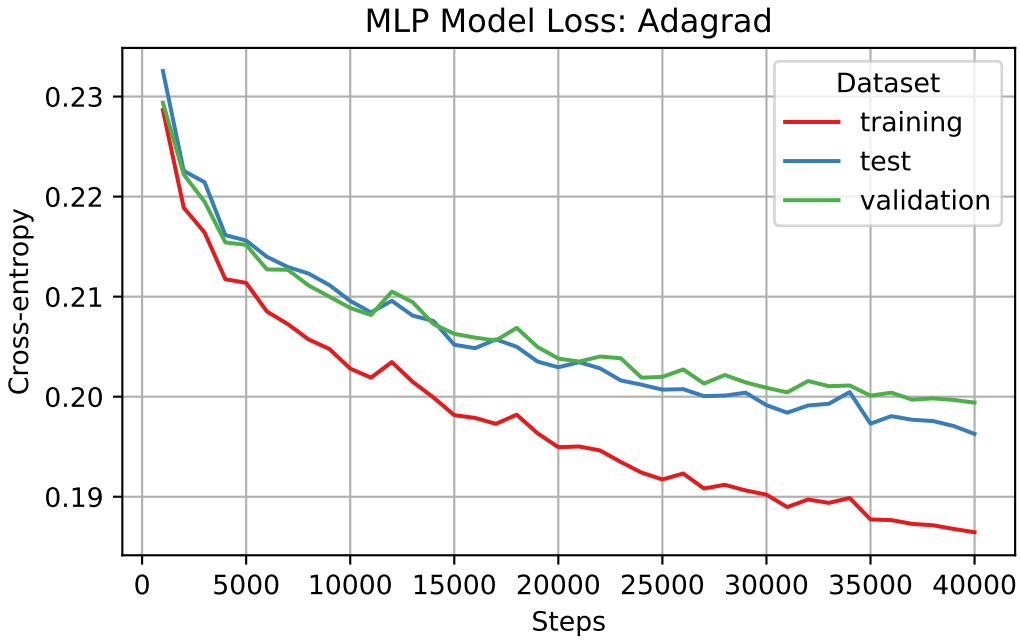

Figure 10: Evaluations on the datasets while training the MLP model with Adagrad's momentum.

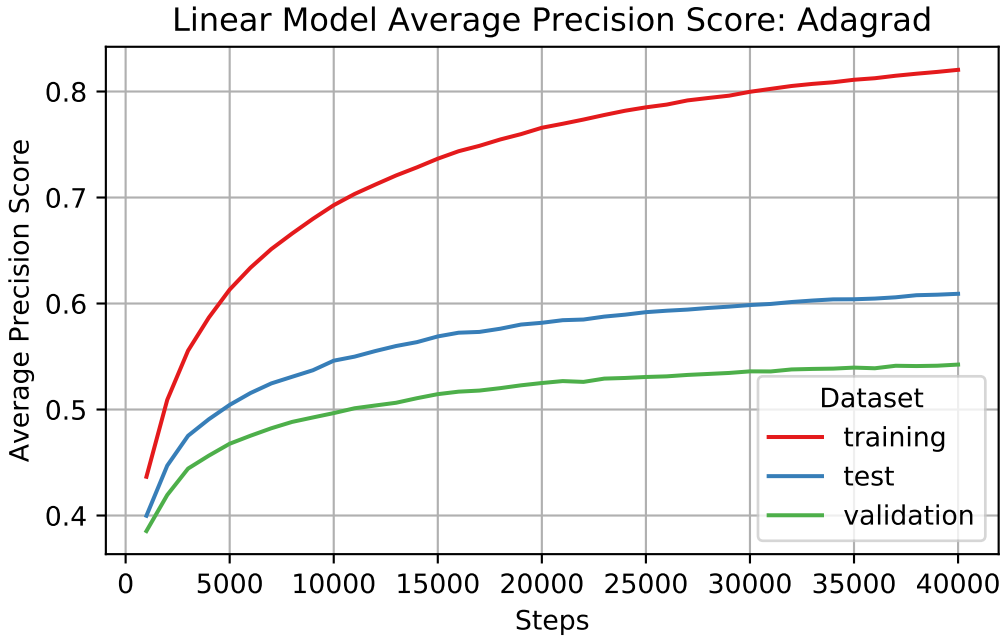

Figure 11: Average precision score by the number of steps for the linear model.

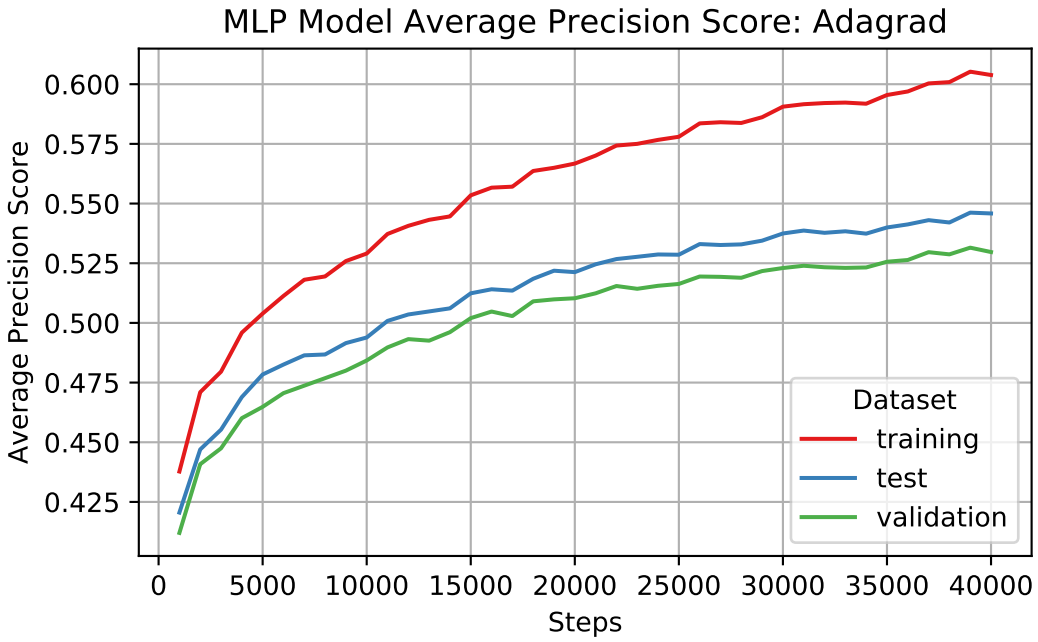

Figure 12: Average precision score by the number of steps for the multi-layer perceptron model.