Problem Set 4

Nima Leclerc (nleclerc@seas.upenn.edu, PennID = nleclerc)
(Collaborated with Abhinav R.)
ESE 546 (Principles of Deep Learning)
School of Engineering and Applied Science
University of Pennsylvania
Total time = 8 hours

November 19, 2020

Problem 1:

Solution:

(a) See attached Jupyter notebook. The training and test set were partitioned, images were downsampled from size 28x28 to 14x14, then normalized.

(b) Provided the loss function for logistic regression,

$$l(w) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + e^{-y_i(w^T x_i + w_0)}) + \frac{\lambda}{2}(||w||_2^2 + w_0^2)$$

Here, $w \in \mathcal{R}^{c\times d}$, $x \in \mathcal{R}^{n\times d}$, and $y \in \mathcal{R}^{n\times c}$, and λ is a regularization constant. We need to compute its gradient $\nabla_w l(w)$ to perform gradient descent for training using the following update rule,

$$w_{t+1} = w_t - \eta \nabla_w l(w)$$

with η as the learning rate. The analytical expression for $\nabla_w l(w)$ comes out to,

$$\nabla_w l(w) = -\frac{1}{n}\sum_{i=1}^{n} \frac{y_i x_i e^{-y_i(w^T x_i + w_0)}}{1 + e^{-y_i(w^T x_i + w_0)}} + \lambda w$$

This learning objective is trained on the MNIST dataset and the implementation details can be found in the attached Jupyter notebook. The weights are initialized at random before training starts and w_0 is set to 0. Using gradient descent, the model is optimized for different values of regularization λ, with a learning rate of $\alpha = 0.01$, and over 50 time steps. The loss function is plotted (semi-log scale) for $\lambda = 1, 10, 20, 30, 40, 50, 70$ in Figure 1. From this, we see a clear trend that $\lambda = 10$ is the optimal regularization. Beyond $\lambda = 10$, the error (loss) begins to saturate. Beyond $\lambda = 70$, bias dominates error due to over-regularization. Hence, at the optimal regularization $\lambda = 10$, the condition number can be extracted from the slope within the linear regime (in log scale). This slope $m = -\kappa^{-1}$, with κ as the condition number. From this, we extract a slope of $m = -0.325$. Hence, the approximate condition number comes out to $\kappa = 3.12$.

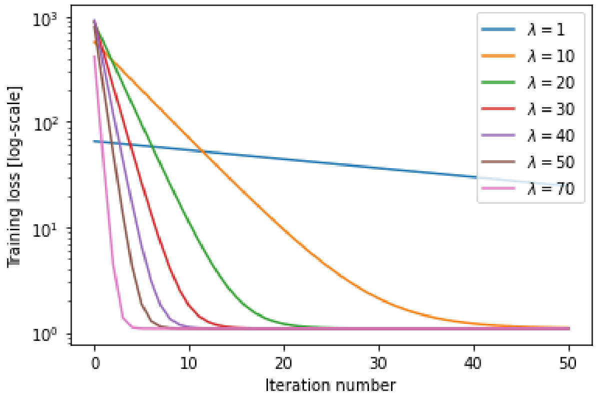


Figure 1: Training loss of logistic regression, using gradient descent with varying regularization λ.

(c) We can now obtain the Hessian of the loss function, that we will use to find the condition number κ. Evaluating for one training example, $(\frac{\partial^2 l}{\partial w^2})_i$. Here,

$$(\frac{\partial^2 l}{\partial w^2})_i = y_i x_i \frac{\partial}{\partial w}(\frac{e^{-y_i(w^T x_i + w_0)}}{1 + e^{-y_i(w^T x_i + w_0)}}) + \lambda$$

$$= -y_i^2 x_i(\frac{e^{-y_i(w^T x_i + w_0)}}{(1 + e^{-y_i(w^T x_i + w_0)})^2}) + \lambda$$

Hence, the Hessian comes out to

$$\nabla^2_w l(w) = \frac{1}{n}\sum_{i=1}^{n} y_i^2 x_i^T x_i (\frac{e^{-y_i(w^T x_i + w_0)}}{(1+e^{-y_i(w^T x_i + w_0)})^2}) + \lambda$$

$$= X^T A X + \lambda I$$

with $\sigma(w^T x) = (1 + e^{-(w^T x)})^{-1}$,$A_{ij} = \sigma(y_i w^T x_i)(1 - \sigma(y_j w^T x_j))$. Our condition for Strong convexity is that $\nabla^2 l \geq mI$ with m as the Strong-convexity parameter and I as the identity matrix. This translates into proving that $X^T AX + \lambda I$ is positive semidefinite. A is indeed positive-semidefinite since it is a diagonal matrix with elements $A_{ii} \geq 0$ (given by the form of the matrix elements). It's a diagonal matrix, hence it is symmetric. Since $\lambda \geq 0$ always, we can conclude that $\nabla^2 l$ is positive semidefinite. Hence, l is a strongly-convex function. The strong-convexity parameter is given by m. For i as the index of A, we can see that $0 < A_i i \leq \frac{1}{4}$. Hence, the upper bound here is $\frac{1}{4}$. From this, we get the strong convexity parameter

$$m \leq \frac{1}{4}\lambda_{min}(X^T X) + \lambda$$

Hence, our strong-convexity parameter is $m = \frac{1}{4}\lambda_{min}(X^T X) + \lambda$ and λ_{min} is the smallest eigenvalue of A. We can take $\lambda_{min} = 0$. Hence, in practice we can take the strong-convexity $m = \lambda$.

(d) Since this is a strongly-convex function, we can pick the momentum to be (provided the condition number κ),

$$\rho = \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}$$

Now in implementing Nesterov's optimizer, we will use the following update steps (using the same gradients derived in part (b)),

$$u^t = (1+\rho)w^t - \rho w^{t-1}$$

$$w^{t+1} = u^t - \eta \nabla l(u^t)$$

The implementation of this is found in the attached Jupyter notebook. We can vary the momentum during training to look at qualitative behavior in the loss over each time-step, as well as to see how the training is influenced by the condition number. This condition number can be validated by extracting the slope m of the loss in log-scale, since $m = -\kappa^{-1/2}$. Here, the model is trained with Nesterov momentum $\rho = 0.75, 0.80, 0.85, 0.9, 0.95$. Using our expression for strongly convex functions, these correspond to κ values $\kappa = 49, 81, 152, 361, 1521$. Plotted in Figure 2 is the training loss for each value of ρ. We see a trend that for smaller ρ (corresponding to smaller condition number), the loss function converges more quickly in contrast to larger ρ. In the linear regime for each loss, the slope is extracted to find κ (so we can verify). From each, we find slopes $m = -0.36, -0.33, -0.28, -0.23, -0.18$. These correspond to condition numbers, $\kappa = 7.63, 9.13, 12.10, 18.23, 29.99$. Clearly, the κs extracted from fitting are greatly in disagreement with those anticipated from our relationship, but qualitative the trend is still consistent.

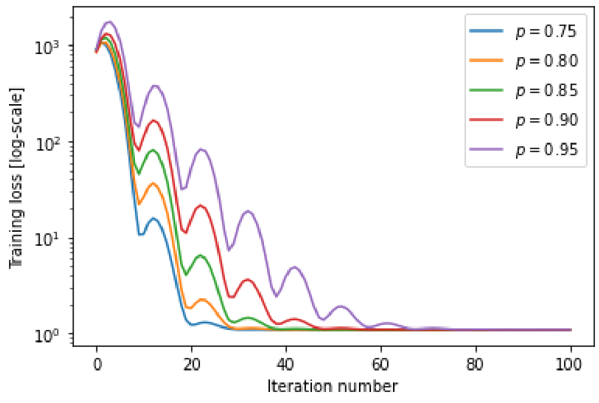


Figure 2: Training loss of logistic regression, using gradient descent with Nesterov's momentum over varying momentum.

(e) Now, both Stochastic Gradient Descent (SGD) and SGD with Nestereov's momentum are implemented. The loss functions (with $\alpha = 0.01, \rho = 0.75, \lambda = 10$) are plotted in Figure 3. We see a clear trend, that the addition of Nesterov momentum improves convergence while training. Figure 3 shows the this trend, comparing gradient descent (GD), GD with Nesterov's momentum, SGD, and SGD with Nesterov's momentum. SGD and GD have similar convergence. Convergence of SGD with Nesterov is faster than normal SGD, clearly.

.

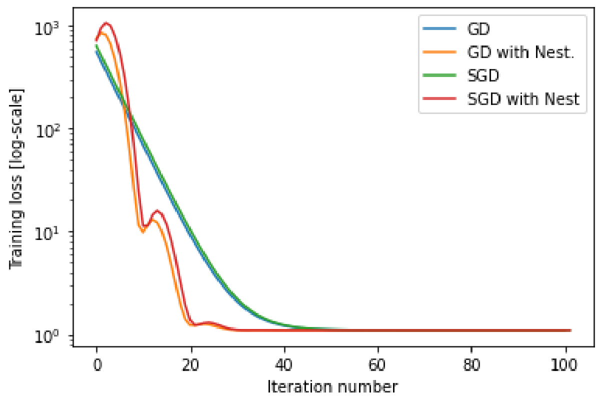


Figure 3: Training loss of logistic regression using GD, SGD, GD with Nest., and SGD with Nest.

Problem 2:

Solution:
(a) The cross entropy loss starts around 2.3 since it is a logit and there are 10 classes. Hence, $l \sim \ln 10 \approx 2.3$

(b) The learning rate finder is implemented on the allCNN network, trained on the CIFAR-10 dataset using the following update over 100 iterations (with $\eta(0) = 10^{-5}$) .

$$\eta(t+1) = 1.15\eta(t)$$

From this, the training loss is plotted as a function of learning rate in log-scale and shown in Figure 4. From this, we see a clear trend that the loss starts off high at very small learning rates, then decreases rapidly. The loss fluctuates throughout since a stochastic algorithm is used. As the learning rate is increased, the loss reaches a minimum (global) at $\eta = 0.088$. This minimum will correspond to our optimal learning rate that we will call $\eta^\star$. Hence, we find $\eta^\star = 0.088$. In particular, this is the maximum learning rate that can be used during training before the loss increases dramatically.

(c) We cannot use the $\eta^\star$ directly for training, so we take $\eta_{max} = \eta^\star/10 = 0.0088$. This is now used in training, using cosine annealing with warmup

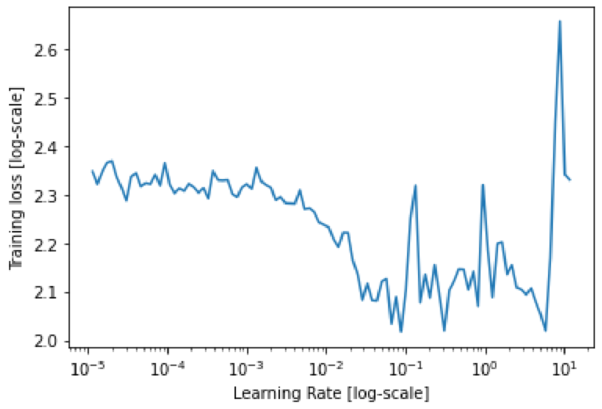
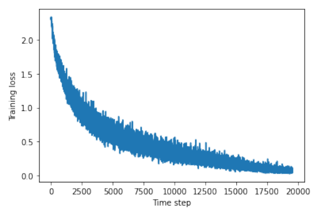


Figure 4: Learning rate finder for All-CNN network.

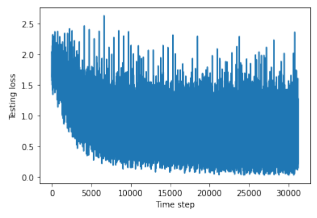
steps. In particular, we use the following scheduler.

$$\eta(t) = 10^{-4} + \frac{t}{T_0}\eta_{max}(t \leq T_0), \eta_{max}\cos(\frac{\pi}{2}(\frac{t-T_0}{T-T_0}) + 10^{-6}, T_0 \leq t \leq T)$$

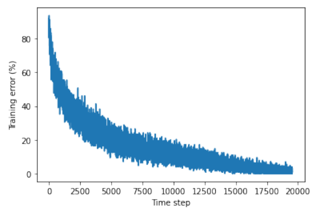
From this, the training/test loss, training/test error, and learning rate over iteration number are plotted in Figure 5 ((a) - (e)). The model is trained over 50 epochs.
(d) Now using, each setting η_{max} and $\rho = 0.9$ (plotted in Figure 6)), $5\eta_{max}$ and $\rho = 0.5$ (plotted in Figure 7)), and η_{max} and $\rho = 0.5$ (plotted in Figure 8)), we train the model over 50 epochs. In each case, we find a final validation error of (1) 8.5 %, (2) 8.2 %, and (3) 7.4 %.



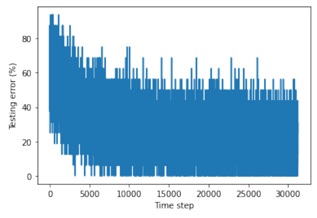
(a) Training loss.



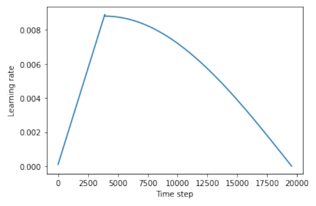
(b) Testing loss



(c) Training loss

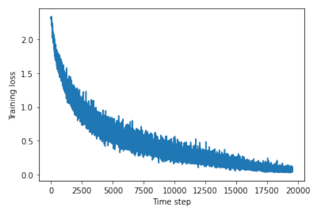


(d) Test error.

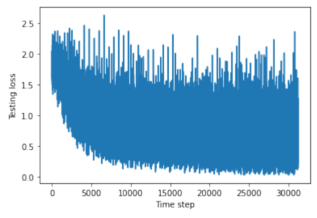


(e) Learning rate schedule.

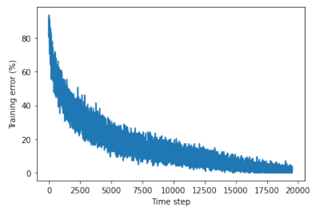
Figure 5: Training behavior of allCNN on CIFAR-10 dataset, with cosine annealing.



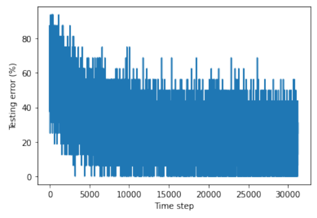
(a) Training loss.



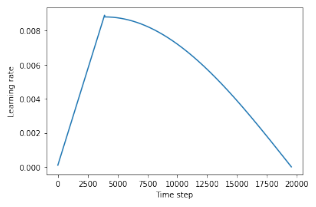
(b) Testing loss



(c) Training loss

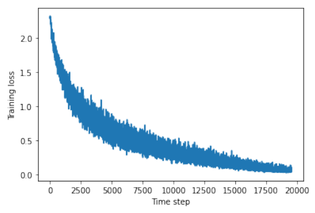


(d) Test error.

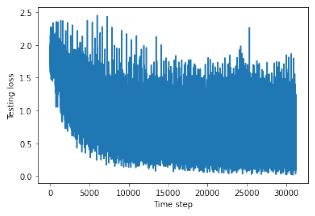


(e) Learning rate schedule.

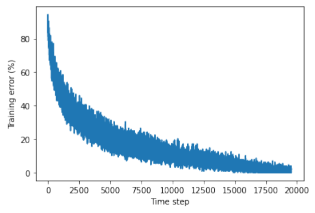
Figure 6: Training behavior of allCNN on CIFAR-10 dataset, with cosine annealing.



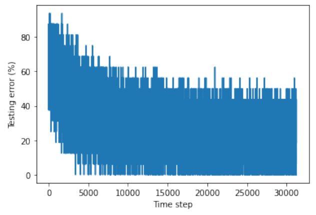
(a) Training loss.



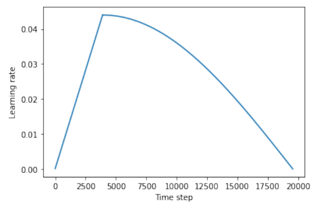
(b) Testing loss



(c) Training loss

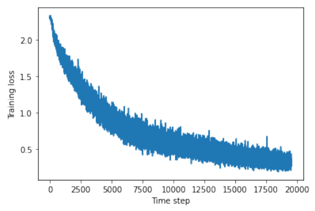


(d) Test error.

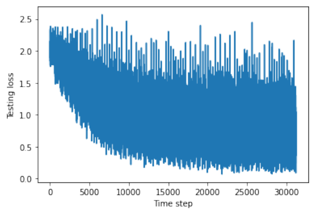


(e) Learning rate schedule.

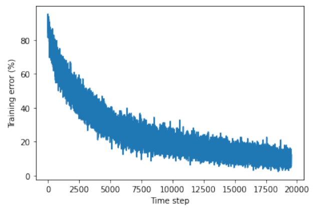
Figure 7: Training behavior of allCNN on CIFAR-10 dataset, with cosine annealing.



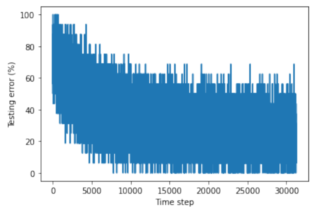
(a) Training loss.



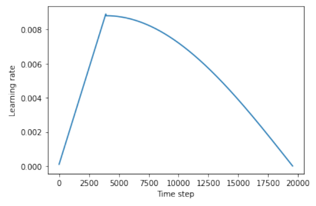
(b) Testing loss



(c) Training loss



(d) Test error.



(e) Learning rate schedule.

Figure 8: Training behavior of allCNN on CIFAR-10 dataset, with cosine annealing.