CS 282 Deep Neural Networks **Spring 2023**

Lecture 7: Survey of Problems and Architectures I

Lecturer: Anant Sahai *Scribe: Hansung Kim, Kornrapat Pongmala*

1 Optimization in Deep Learning Context

Optimization in deep learning context can be very different from the traditional context. In traditional optimization, the objective function is exactly what we would want to optimize over. In deep learning, however, often times we are optimizing over a differentiable surrogate loss on the training data, hoping that the model will learn meaningful representation of the data and generalize well to the true loss on test data. Moreover, as we are iteratively optimizing the parameters in deep learning, wall-clock time to reach convergence matters. Because of optimization algorithms can change what and how fast we converge to, it is an important topic to study and keep in mind when training a model. In the following subsections, we will go over important topics to consider in regard to optimizer and meaningful representation. Note that, like previously discussed, non-linear deep learning systems can be difficult to understand, but will try our best using tools from linear systems.

1.1 Learning Rate, Singular Values, and Convergence

Let's consider gradient descent on the ordinary least squares (OLS) problem.

$$\min_w \|Xw - y\|^2 \tag{7.1}$$

$$w_{t+1} = w_t - \eta(\nabla_w \|Xw - y\|^2) \tag{7.2}$$

We have seen earlier that the problem can be converted into the SVD coordinate form where it decouples into independent linear equations. The convergence rate of each parameter in the SVD coordinates ($\hat{w}[i] = (V^T w)[i]$) is $|1 - 2\eta\sigma_i^2|$ where σ_i is the corresponding singular value.

Now, we attempt to visualize the how different magnitude of η affect the convergence behavior of the system. To do so, we can plot $1 - 2\eta\sigma_i^2$ for each σ_i on a unit circle. To move the points around along the real axis, we can simply tune the learning rate. In the figure below, we can see how different magnitudes of learning rates affect the location of $1 - 2\eta\sigma_i^2$ along the real axis. The closer a point is to the origin, the faster it will converge. If $|1 - 2\eta\sigma_i^2| < 1$, then the system converges to a solution. If any $1 - 2\eta\sigma_i^2$ lies on the border or outside of the unit circle, the system diverges.

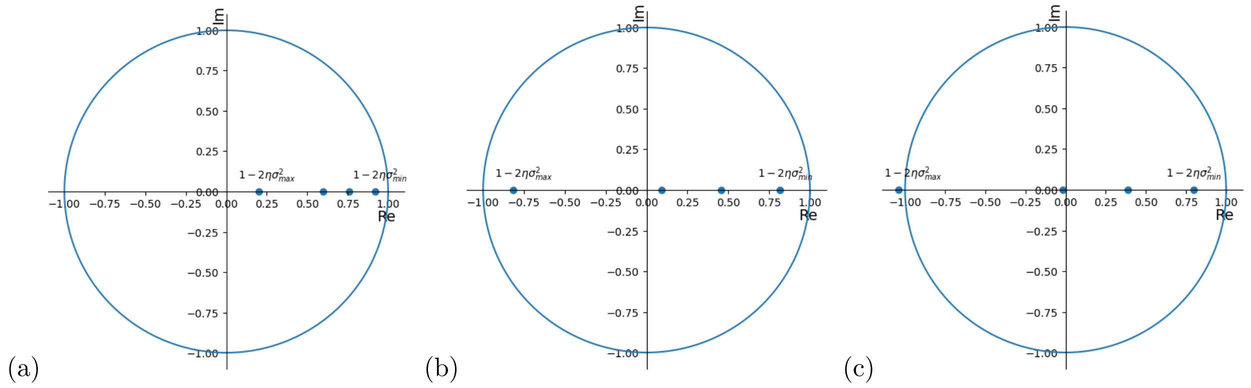

Figure 7.1: (a) Low learning rate (b) Medium learning rate (c) Too high learning rate[1]

Because smaller magnitude of $|1-2\eta\sigma_i^2|$ converges faster, different values of learning rate can yield different models that prioritize converging on direction corresponding to different singular values, some of which may generalize better to test data. Because the order of convergence of $\hat{w}[i]$ is dependent on η, hyperparameter search and model checkpointing during training are important.

1.2 Underdetermined Nature of Deep Learning

So far, we have discussed the "path" gradient descent takes to convergence (which $\hat{w}[i]$ to converge first depending on η). Not only that the choice of optimization algorithm and hyperparameter could affect the "path" towards convergence, but they also affect the solution we converge to. This is particularly important as we want to converge to a solution that better generalizes to unseen data.

In modern deep learning models, it is often the case where we have millions or billions of parameters and much fewer data points. Analogously, if the OLS has larger number of parameters than data points, i.e. X is a wide matrix, we know that there exists infinitely many solutions 7.3. So, which solution will different optimization algorithms converge to?

$$w^* \in \{X^T(XX^T)^{-1}y + v | v \in Null(X)\} \tag{7.3}$$

For underdetermined OLS problems, gradient descent with parameters initialized at 0 converges to the min-norm solution. This is because gradient descent update equation can be rewritten to show that each update only allows the weights to move within $Col(X^T)$ 7.4 and the min-norm solution is the only solution within $Col(X^T)$.

$$w_{t+1} = w_t - \eta(\nabla_w \|Xw-y\|^2) = w_t - 2\eta X^T(Xw-y) \tag{7.4}$$

A similar argument could be made for stochastic gradient descent.

[1]Shown in class, digitized by scribers

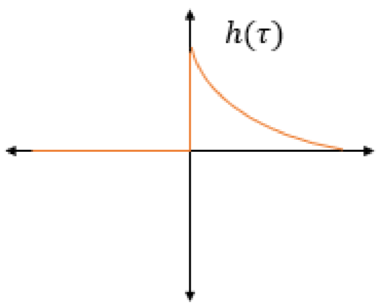

Figure 7.2: Impulse response for a first-order system[2]

2 Speeding Up Convergence with Momentum

2.1 Current Limitations

In gradient descent, too low of a learning rate means lower singular values will take too long to converge. One way to speed up convergence is to increase the learning rate. However, learning rate is bounded by the largest singular value. Too high of a learning rate could make $1 - 2\eta\sigma_{max}^2 \leq -1$, resulting in the weights oscillating too much and diverging.

2.2 Dampening Oscillation and Connection to Low Pass Filters

In order to alleviate the aforementioned limitations, we can attempt to dampen the oscillations by applying the low pass filter on the gradients. In circuit analogy, the gradient is the input voltage and by adding a grounded capacitor, the fluctuation or oscillation in the input voltage can be dampened. This can be expressed as a continuous-time first-order differential equation. The solution to the first-order differential equation to find the time-varying explicit function of the state (voltage, current) of the system involves a convolution integral:

$$\int_{-\infty}^{t} u(\tau)h(t-\tau)d\tau \tag{7.5}$$

The convolution integral can be interpreted as a "fancy averaging" where $h(t-\tau)$ is the impulse response to the input $u(\tau)$. The convolution integral can be thought of as a weighted average with exponentially decaying weights with emphasis on the $h(\tau)$ at the current time as shown in the graphical representation of $h(\tau)$ in Fig. 7.2. The discrete-time counterpart of the convolution integral is as follows:

$$\sum_{\tau=-\infty}^{t} u(\tau)\beta(1-\beta)^{(t-\tau)} \tag{7.6}$$

This solution corresponds to the following difference equations in discrete time:

$$\begin{aligned} \vec{a}_{t+1} &= (1-\beta)\vec{a}_t + \beta\vec{u}(t) \\ \vec{w}_{t+1} &= \vec{w}_t - \eta\vec{a}_{t+1} \end{aligned} \tag{7.7}$$

[2]Shown in class, digitized by scribers

where $\vec{u}_t = \nabla_w l(\vec{w}_t)$ and β is a tuning parameter. Here, $\vec{a}_{t+1}$ is the smoothed version of the input $\vec{u}_t$.

2.3 Gradient Descent with Momentum

The formulation in 7.7 is exactly how gradient descent with momentum works. In the context of gradient descent, $\vec{a}_t$ is the dampened or smoothed gradient. In essence, instead of taking a step in the negative gradient at the time, we want to take a step in the direction of the weighted average of the past gradients. $\beta \approx 0$ would cause the running average of the gradient $\vec{a}_t$ to decay slower because higher weights are placed on past gradients.

Alternatively, using $\vec{u}_t = \nabla_w l(\vec{w}_t - \eta(1-\beta)\vec{a}_t)$ instead of $\nabla_w l(\vec{w}_t)$ in a vanilla gradient descent update law is slightly better as now we are taking the gradient with respect to where the updated weights would have landed as opposed to the where current weights are.

2.3.1 Circuit Analogy for Gradient Descent & Gradient Descent with Momentum(Optional)

Now, we provide a detailed circuit analogy to gradient descent to better understand the internal workings of gradient descent. Consider the circuit shown in Fig. 7.3. When the training of the neural network begins, the current source turns on, providing constant current to the circuit. The current which is analogous to training data excites the modes which correspond to different singular values of the system. The inductor represents what the parameters (weights and biases) learn in gradient descent. So at convergence, the parameters perfectly fit training data, and in the circuit, the current through the inductor is equal to that of a current source. However, at the beginning of the training process, there is no current through the inductor as the parameters have not been learned. Thus, there is a residual in the current of the current source and the current through the inductor. This residual passes the current through the resistor connected in parallel, generating voltage. The voltage is analogous to the gradient and it exponentially allows more current through the inductor, approaching convergence.

Now, consider the circuit shown in Fig. 7.4. By adding the capacitor in parallel to the inductor and the resistor, the voltage or the gradient is dampened or averaged. This accelerates the convergence rate by smoothing out the gradient decay rate. Although it is physically impossible for the continuous-time first-order system represented by this RLC circuit to be unstable, the discrete-time system considered in the gradient descent algorithm can become unstable. Nevertheless, the circuit analogy is a useful tool to gain insight into how the gradient descent mechanism works. For more information on how and why momentum really works, please refer to `https://distill.pub/2017/momentum/`.

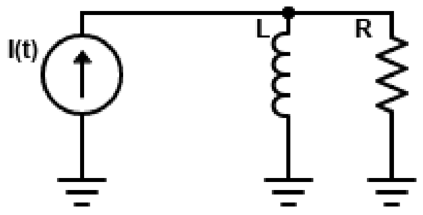

Figure 7.3: Circuit diagram analogous to vanilla gradient descent[3]

[3]Shown in class, digitized by scribers

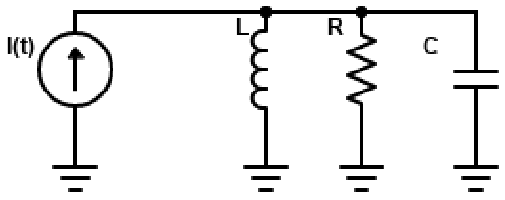

Figure 7.4: Circuit diagram analogous to gradient descent with momentum[4]

2.3.2 Adaptive Methods:

Adaptive methods are motivated by online learning in which data is partially available at a given time. Even in gradient descent with momentum, the largest singular value limits the learning rate for stability. One popular variation of gradient descent with momentum is Adam. In Adam, individual adaptive learning rates for different parameters are computed from estimates of the first and second moments of the gradients (**?**). However, in online learning, the algorithm doesn't know what the modes are because it only sees partial data at a given time. So how do we choose the best learning rates for each mode and determine the modes themselves without accessing the whole dataset? At first glance, this seems to be a fatal flaw in implementing adaptive learning rates for different parameters.

However, there is a way to circumvent this fatal flaw. We follow the spirit of deep learning which is "look where the light is". Hence, the name "Drunk Lamp Post Principle". The main idea is to use different learning rates for different parameters or weights. We do not know the relevant true modes of the data, but we use what we know: the parameters or weights.

For instance, it is desirable to have a big learning rate if the gradient is small and have small learning rate if the gradient is large. To do this, the sizes of gradients must be tracked during learning. Therefore, we add another state variable, $\vec{v}_t$, which we update every iteration. The parameter update laws become the following

$$\begin{aligned}
\vec{a}_{t+1} &= (1-\beta)\vec{a}_t + \beta\nabla l(\vec{w}_t) \\
\vec{v}_{t+1} &= (1-\beta')\vec{v}_t - \beta' \begin{bmatrix} (\frac{\partial}{\partial w[1]} l(\vec{w}_t))^2 \\ \vdots \\ (\frac{\partial}{\partial w[n]} l(\vec{w}_t))^2 \end{bmatrix} \\
\vec{w}_{t+1}[i] &= \vec{w}_t[i] - \eta \frac{\vec{a}_{t+1}[i]}{\sqrt{\vec{v}_{t+1}[i]} + \varepsilon}
\end{aligned} \tag{7.8}$$

where $\vec{w}_t[i]$ denotes the i-th element of $\vec{w}_t \in \mathbb{R}^n\ \forall t$. $\varepsilon \in \mathbb{R}_{++}$ is a small offset to the denominator to avoid the singularity. By doing this, different step sizes are achieved for different parameters by tracking three state variables in online learning. Note that a small $\vec{v}_t$ is undesirable in this adaptive optimization method. To avoid this, normalize $\vec{v}_t$ appropriately. The update equation for $\vec{w}_{t+1}$ includes the $\vec{a}_{t+1}$ term with the $t+1$ subscript. This is because we want to depend on the gradient at the current time (t) instead of (t-1). In gradient descent based methods, the main question is where do we want to evaluate the gradient? The standard approach in vanilla gradient descent is to evaluate it at the current time step, $\vec{w}_t$. In gradient

[4]Shown in class, digitized by scribers

descent with momentum, we have prior knowledge on what direction we want to take a step from the moving average. $\vec{a}_t$. We update this term by a weighted sum between prior knowledge and current gradient. Then, we use this updated direction to take a step in the $\vec{w}_{t+1}$ update. Also, we want $\beta' << \beta$ so $\vec{v}$ is more smooth and steady. Too much variability in the $\vec{v}$ term can destabilize the weight update. Therefore, a small β' ensures that $\vec{v}$ decays slowly.

3 What we wish this lecture also had to make things clearer

Circuit analogy can confuse people without much electrical engineering background.

References

[1] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.