The Art & Science of Training
Deep MLPs

M. R. Hasan

Deep Learning
Readings

• Geron: 11
The Art & Science of Training Deep MLPs

• We will discuss some key issues regarding the training of MLPs.

• Some issues concern the deep MLPs (many hidden layers and neurons) or more generally deep neural networks (DNNs).
Issues Regarding Training Deep MLPs

• We will discuss the following issues.
  - Feature scaling
  - Intractability of computation
  - Computational complexity
  - Overfitting due to complex architecture
  - Overfitting due to overtraining
  - Weight initialization
  - Vanishing gradient & exploding gradient problem
  - Achilles’ Heel of Stochastic Gradient Descent
  - Prediction invariance
  - Scalability with respect to the input size
Training Issues:
Achilles’ Heel of SGD
Can SGD take us to the elusive happiness?
Achilles’ Heel of SGD

• So far we have discussed several issues including the vanishing and exploding gradient problem that make training deep MLPs challenging.
• Now we will turn to the stochastic gradient descent (SGD) algorithm that could stall the training in deep neural networks (DNNs).
Achilles’ Heel of SGD

• The SGD optimization algorithm is based on the Stochastic Approximation method proposed in 1951 by Robbins and Monro.
• SGD minimizes the loss function in an iterative fashion.

A STOCHASTIC APPROXIMATION METHOD

By Herbert Robbins and Sutton Monro
University of North Carolina

1. Summary. Let $M(x)$ denote the expected value at level $x$ of the response to a certain experiment. $M(x)$ is assumed to be a monotone function of $x$ but is unknown to the experimenter, and it is desired to find the solution $x = \theta$ of the equation $M(x) = \alpha$, where $\alpha$ is a given constant. We give a method for making successive experiments at levels $x_1, x_2, \cdots$ in such a way that $x_n$ will tend to $\theta$ in probability.
Achilles’ Heel of SGD

- There are **two variants** of SGD.
  - Update weights based on a single data point.
  - Update weights based on a mini-batch of data points.
- The latter is known as mini-batch SGD.

When we refer to SGD in the context of deep neural networks, we mean the **mini-batch SGD**.
Achilles’ Heel of SGD

- Advantages of SGD:
  - *Faster* than the batch GD for each weight update.
  - Makes it possible to train on huge training sets.
  - Due to its inherent randomness, SGD is useful to escape from sub-optimal minima.
Achilles’ Heel of SGD

• However, SGD has an *Achilles’ heel*.
• It’s sloth!
• SGD takes **much longer time** (due to more iterations) to *converge* near the minima of the loss function, as compared to batch GD.
Achilles’ Heel of SGD

Why does the SGD takes longer time to converge?
Because of its oscillatory behavior.
But why does oscillation occur in SGD?
Achilles’ Heel of SGD

• In the SGD, at each update of the weight parameters, successive gradient values may be so different that large oscillations may occur.
• The loss gradient directions are erratic!
• This happens even on a smooth loss surface terrain.
Achilles’ Heel of SGD

• There are **two main contributing factors** to its slow convergence.
  - Nature of SGD algorithm
  - Topology of the cost function
• We describe these two causes.
Achilles’ Heel of SGD

• In SGD, at each update of the weight parameters, successive gradient values may be so different that large oscillations may occur.
• The cost gradients directions are erratic!
• This happens even on a smooth cost surface terrain.
Achilles’ Heel of SGD

• Thus, instead of gently decreasing until it reaches the minimum, the cost function will oscillate, decreasing only on average.

• Over time it will end up very close to the minimum, but once it gets there it will continue to bounce around around the minimum, never settling down.

Oscillation is the evil!
Achilles’ Heel of SGD

- What is the **root cause of the oscillations** in SGD?
- At each iteration SGD measures the gradient of \( J(w) \) w.r.t all components of \( w \).
- In other words, it measures the slopes along all dimensions.
- Then, it takes a step along the steepest direction.
- A single step size is chosen for **all components of \( w \)**.

\[
J(w) = \frac{1}{|X|} \sum_{x \in X} l(x, w)
\]
Achilles’ Heel of SGD

• The step size is determined by the scalar constant learning rate hyperparameter $\eta$.
• This hyperparameter is the root cause of the oscillatory evil!
• Typically the scalar constant $\eta$ is chosen by hand.
Achilles’ Heel of SGD

- A too small $\eta$ results in slow learning.
- A too large $\eta$ overshot the step and SGD movement diverges.

Finding the optimal $\eta$ is a challenging task!
Achilles’ Heel of SGD

• We can reduce the oscillation without explicitly finding the optimal scalar constant $\eta$.
• We could use a simple approach.
• As SGD nears a minima in the cost surface, we could decrease $\eta$.
• But this needs to be done gently.
Achilles’ Heel of SGD

• SGD starts out by **taking large steps** (i.e., large $\eta$) so that it can make quick progress and escape local minima.
• Then, we reduce $\eta$ to make the **steps smaller and smaller**.
• It will allow the algorithm to settle at the minimum.
• This process is known as **simulated annealing**.
• It is inspired from the process in metallurgy of annealing, where molten metal is slowly cooled down.
Achilles’ Heel of SGD

- For a detail discussion on the implementation and experimentation with various learning schedules in ANN see the Jupyter Notebook

For faster convergence the 1cycle learning schedule algorithm is quite effective as it can afford large learning rate without diverging.
Achilles’ Heel of SGD

- There are two limitations of using learning schedules with SGD.
  - The biggest limitation is that we need to define additional hyperparameters in advance.
  - The same learning rate is applied to all weight parameter updates.

A superior approach would be to find optimal learning rates for each dimension.
Achilles’ Heel of SGD

- An alternative to learning schedule is to fine optimal $\eta$.
- Let’s do it analytically.
- First, we expand the cost function $J(w)$ in a Taylor series about the current weight $w_c$:

$$J(w) = J(w_c) + (w - w_c) \frac{\partial J(w_c)}{\partial w} + \frac{1}{2} (w - w_c)^2 \frac{\partial^2 J(w_c)}{\partial w^2} + ...$$

Differentiating $J$ w.r.t. $w$:

$$\frac{\partial J(w)}{\partial w} = \frac{\partial J(w_c)}{\partial w} + (w - w_c) \frac{\partial^2 J(w_c)}{\partial w^2}$$
Achilles’ Heel of SGD

• At the minimum:

\[
\frac{\partial J(w)}{\partial w} = 0
\]

\[
\frac{\partial J(w_c)}{\partial w} + (w - w_c) \frac{\partial^2 J(w_c)}{\partial w^2} = 0
\]

\[
\Rightarrow w \frac{\partial^2 J(w_c)}{\partial w^2} = w_c \frac{\partial^2 J(w_c)}{\partial w^2} - \frac{\partial J(w_c)}{\partial w}
\]

\[
\Rightarrow w = w_c - \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1} \frac{\partial J(w_c)}{\partial w}
\]

Compare it with SGD update rule:

\[
w \leftarrow w - \eta \nabla_w J(w)
\]

Thus, optimal learning rate is:

\[
\eta_{opt} = \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1}
\]
Achilles’ Heel of SGD

- Observe that the $\eta_{opt}$ is no more a scalar constant.
- Instead it is a $d \times d$ matrix.
- It is computed by taking the inverse of the second-order derivative of the cost function w.r.t. the weight parameters.
- It is called the Hessian.
- The components of the Hessian are given by:

$$H_{ij} = \frac{\partial^2 J(w_c)}{\partial w_i \partial w_j}$$

$$\eta_{opt} = \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1}$$
Achilles’ Heel of SGD

• We have computed $\eta_{opt}$ by **ignoring the higher-order** terms in the Taylor series expansion of the cost function.
• Thus, $\eta_{opt}$ does not give the exact optimal learning rate.
• It may take multiple iterations to locate the minimum even when using $\eta_{opt}$.
• However, convergence can still be quite fast.

\[
\eta_{opt} = \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1}
\]

\[
J(w) = J(w_c) + (w - w_c) \frac{\partial J(w_c)}{\partial w} + \frac{1}{2} (w - w_c)^2 \frac{\partial^2 J(w_c)}{\partial w^2} + \ldots
\]
Achilles’ Heel of SGD

• We can modify the SGD update rule by replacing $\eta$ by the optimal learning rate matrix, which is the inverse of the second-order derivatives of $J(w)$:

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

$$w_{t+1} = w_t - H^{-1} g$$

This new update rule is based on the 2nd-order derivative of the cost function.

$$\eta_{opt} = \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1}$$

$$H_{ij} \equiv \frac{\partial^2 J(w_c)}{\partial w_i \partial w_j}$$

This new update rule is also known as Newton’s Algorithm.
Achilles’ Heel of SGD

• How does the Hessian (optimal learning rate matrix) help SGD to **converge faster**?

• Because the Hessian provides the **curvature of the cost space**.

• SGD can **move faster** by using the curvature of the space into account.

• The curvature along each dimension acts as a proxy of the **learning rate** for that dimension.

• Let’s explain this.

\[ \eta_{opt} = \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1} \]
Achilles’ Heel of SGD

- Consider an example of a curvature of a circular space.
- At point “a” the curvature tells us that the slope is steep.
- Thus, we would like to increase the learning rate to move faster along the ridge.
- On the other hand, at point “b” the region is almost flat, according to its curvature.
- Hence, we would want to reduce the learning rate to move slowly.

$$\eta_{opt} = \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1}$$
Achilles’ Heel of SGD

• The beauty and benefit of computing the Hessian of the cost function is that it can quantify the steepness/flatness of the cost space.

• For example, the curvature at point “a” is large, while at point “b” it is small.

\[
\eta_{opt} = \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1}
\]
Achilles’ Heel of SGD

• Thus, by measuring the curvature of a space at a point we could know **how fast/slow we need to move** toward the optimum.

• In other words, **curvature provides the exact step size** suitable for the slope of the ground beneath.

• The curvature is the optimal “learning rate”.

$$\eta_{opt} = \left(\frac{\partial^2 J(w_c)}{\partial w^2}\right)^{-1}$$
Achilles’ Heel of SGD

• In the first-order SGD we only use gradient with a hand-tuned step size (learning rate).
• We don’t know whether the step size is too small or large.
• In the 2\textsuperscript{nd}-order SGD, the curvature tells us whether a gradient step will cause as much of an improvement as we would expect based on the gradient alone.

\[ \eta_{opt} = \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1} \]

\[ w_{t+1} = w_t - \eta \nabla_w J(w_t) \]  \hspace{1cm} 1\textsuperscript{st}-order SGD

\[ w_{t+1} = w_t - H^{-1}g \]  \hspace{1cm} 2\textsuperscript{nd}-order SGD
Achilles’ Heel of SGD

• Example: the cost function is quadratic.
• The dashed line indicates the value of the cost function we would expect based on the gradient information alone as we make a gradient step downhill.

\[ \eta_{opt} = \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1} \]
• Left: curvature is negative. Thus, the cost function actually **decreases faster** than the gradient predicts. Step size (learning rate) should be larger.

• Middle: curvature is zero, the gradient predicts the decrease correctly.

• Right: curvature is positive, hence the function decreases slower than expected and **eventually begins to increase**. If we set the step size larger then it will overshoot.

The gradient alone **doesn’t have the information** to determine the step size, which we can get from Hessian.
Achilles’ Heel of SGD

- Hessian is a real and symmetric matrix.
- Two properties of Hessian are of interest to us.
  - Property 1: The eigenvalues measure the steepness of the cost function along the corresponding eigendirection.
  - Property 2: If the cost function can be approximated well by a quadratic function, then the optimal rate for the i-th weight $w_i$ would be the inverse of the eigenvalue:

\[
\eta_{opt} = \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1}
\]

\[
\eta_{opt,i} = \frac{1}{\lambda_i}
\]

Thus, by solving the Hessian eigen equation we can find the optimal learning rates for each dimension.
The main limitation of the Hessian is that it is **computationally expensive**.
- The space complexity of storing Hessian is $O(d^2)$.
- Moreover, the time-complexity of inverting the Hessian is $O(d^3)$.
- Also, **larger batch sizes**, e.g., 10,000, are required to minimize fluctuations in the estimates of $H^{-1}g$.

There are efficient quasi-Newton methods and **diagonal approximations** of computing Hessian.

$$\eta_{opt} = \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1}$$
Achilles’ Heel of SGD

- Although Hessian is expensive to compute, it contains \textbf{very useful information} about the structure of the cost space.
- This information is required for \textbf{per-dimension adaptation} of the learning rate.
- We will see that the \textbf{fast optimizers (e.g., Adam)} will derive their intuition from Hessian by trying to design efficient \textbf{pseudo-Hessian}.

\[ \eta_{opt} = \left( \frac{\partial^2 J(w_c)}{\partial w^2} \right)^{-1} \]
Summary:

We discussed an approach to find a suitable scalar constant learning rate by using learning schedule.

We found that the optimal learning rate is not a scalar constant, it is a matrix that is the 2\textsuperscript{nd} order derivative of the cost function known by Hessian.

Hessian provides the curvature information, using which SGD can move faster.

However, Hessian is expensive to compute.
Achilles’ Heel of SGD

• Next we turn to the **extrinsic cause** for the slow convergence of SGD.

• It is due to the **topology of the cost function space**.

• The cost function typically has a highly **nonlinear dependence** on the weights and bias parameters.
Due to the non-convex nature of the cost function, in practice, there will be many points in the weight space at which the gradient vanishes (or is numerically very small).
• Problematic regions in cost space:
  - Zero-gradient locations: local minima, saddle points, plateau
  - Pathological curvature
  - Cliff
  - Varying slope
• Local Minima: Many local minima might exist in low-dimensional cost space.
• **Rare in high-dimension!** (thus not much of a problem in DNN)
• Because the slope along all dimensions out of the local minima must be positive (i.e., should bend upward), which is rare.
Achilles’ Heel of SGD

• Saddle Point:
• DNN cost space is **proliferated with saddle points**.
• The expected ratio of the number of saddle points to local minima grows exponentially with d, i.e., number of weight parameters.
• Saddle Point:
• At the saddle point region some dimensions would have positive slope (bend upward), while some dimensions have negative slope (bend downward).
• Intuitively, this means that a saddle point acts as both a **local minima** for some neighbors and a **local maxima** for the others.

We see that the “saddle point” region looks like a **horse saddle**.
• Saddle Point:
• How can SGD identify and escape saddle points?
• It needs to **distinguish a local minimum from a saddle point**?
• But how?
• By computing the **Hessian** of the cost function.
• The point $x$ is a local minimum: When the Hessian has only positive eigenvalues.

• The point $x$ is a saddle point: When the Hessian has both positive and negative eigenvalues.

Thus, by computing the Hessian we can find a saddle point, and escape it by moving along the direction of the negative eigenvalue.
• Computing Hessian is infeasible.
• Also in some cases it doesn’t help, when we don’t have **well-behaved** saddle points.
• Example: monkey saddle point: at (0, 0) **curvature is 0**.
• In this case, neither the gradient nor the Hessian provide any information on which way is down.

Thus, using Hessian we are unable to escape **degenerate saddle points**.
Achilles’ Heel of SGD

- Saddle Point:
- Considering Hessian is expensive, we could use the following technique to avoid saddle points.

- Noisy SGD!
- When updating parameters, add a tiny amount of noise on top of the gradient to nudge SGD to a slightly lower point instead of getting stuck at the saddle point.
Achilles’ Heel of SGD

• Plateau:
  • **Wide regions** in higher-dimensional cost space at which the gradient (as well as the Hessian) at each dimension is zero or near-zero.
  • Near-zero updates for all weights.

We would be stuck on that plateau for a **long time** and training would be extremely slow.
Achilles’ Heel of SGD

- Plateau:
- For moving along plateau augment SGD with **momentum**.

Use **adaptive learning rate per dimension**, such that learning rate increases in regions where cost function slope is flat (i.e., on a plateau).

Plateau
• Pathological Curvature: A **ravine or a long narrow valley**.
• The minimum point in the cost space is at the bottom of the valley.
• To get there, SGD has to go through the ravine.

However, the varying curvature of this terrain makes SGD’s movement towards the minimum (direction of \( w_2 \)) is **slower**.
• Let’s see why the movement along $w_2$ is slower.
• Consider a point on the surface of the ridge.
• The gradient at the point can be decomposed into two components, one along direction $w_1$ and other along $w_2$.

The component of the gradient in direction of $w_1$ is **much larger** because of the curvature of the cost function.
• Hence the direction of the gradient is much more toward $w_1$, and not toward $w_2$ (along which the minima lies).
• Thus, the movement toward the minimum oscillates, which requires a larger number of iterations before convergence.

This scenario is “pathological” due to the mixture of low and high curvature directions.
To navigate through a pathological curvature faster, we need to **adapt the learning rate per dimension** by using second-order methods (i.e., Hessian).

In addition to this, **adding momentum** is useful.

These two techniques are combined in a **fast optimizer named “Adam”**, which is suitable for navigating through pathological curvature.
Achilles’ Heel of SGD

• Cliff:
• In some type of DNNs (e.g., in Recurrent Neural Networks) extremely steep regions in the cost space exist.
• Using metaphor, these are “cliffs” on a 2D cost space.

This type of **sharp nonlinearity** in cost space may arise due to the multiplication of several layers of weights.
Achilles’ Heel of SGD

• Cliff:
• These nonlinearities give rise to very high derivatives in some places.
• When the weight parameters get close to such a cliff region, a SGD update can shoot the parameters very far even with a modest step size.

Standard solution for avoiding cliffs is to perform gradient clipping.
Achilles’ Heel of SGD

• Varying Slopes
• There are regions on the cost space that has **steeper** slopes.
• In general, in DNN cost functions **some dimensions are flatter** where the movement should be slower, while **some dimensions are steeper** where movement should be faster.
• A constant scalar learning rate, as used by SGD, would surely **cause uneven progress and slow down** the training process.
Achilles’ Heel of SGD

• Varying Slopes
• We need to **adapt the learning rate per dimension**.
• Following optimization algorithms are used for this purpose.
  - AdaGrad
  - Adadelta
  - RMSProp
  - Adam
  - Nadam
  - AdaMax