Optimization for Training Deep Models

Dr. Josif Grabocka

ISMLL, University of Hildesheim

Deep Learning

Outline

Non-convex Optimization

- Finding the parameters that yield the minimum cost

$$\underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(f(x,\theta), y) \tag{1}$$

- The cost functions of neural networks are highly non-convex

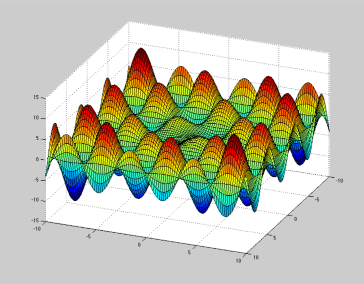

Figure 1: A non-convex function has multiple local optima, Source: `imgur.com`

Saddle Points

- In addition to local minima, cost functions include saddle points

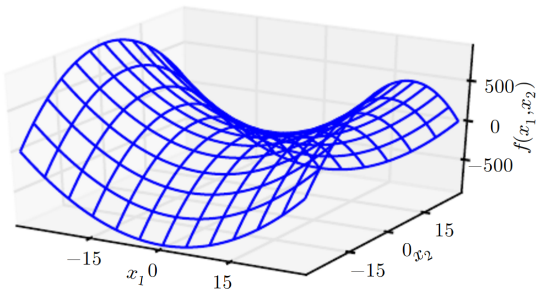

Figure 2: Saddle points, Source: Goodfellow et al., 2016

- Gradients are very small around a saddle point

Gradients

- Cost function surface is unknown
- We only know back-prop gradients of parameters θ:
  - Online w.r.t. a single instance $(x^{(i)}, y^{(i)})$:

$$\hat{g} = \nabla_\theta \mathcal{L}(f(x^{(i)}, \theta), y^{(i)})$$

  - Batch of full training set $\mathcal{D}^{Train} := \left\{(x^{(i)}, y^{(i)}), \ldots, (x^{(N)}, y^{(N)})\right\}$:

$$g = \frac{1}{N} \nabla_\theta \sum_{i=1}^{N} \mathcal{L}(f(x^{(i)}, \theta), y^{(i)})$$

  - Mini-batch $\left\{(x^{(i)}, y^{(i)}), \ldots, (x^{(m)}, y^{(m)})\right\} \subset \mathcal{D}^{Train}$ for $m << N$:

$$\hat{g} = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} \mathcal{L}(f(x^{(i)}, \theta), y^{(i)})$$

Stochastic Gradient Descent and Learning Rates

- We previously mentioned gradient descent $\theta \leftarrow \theta - \epsilon \hat{g}$
- What is the learning rate/step size $\epsilon = ?$

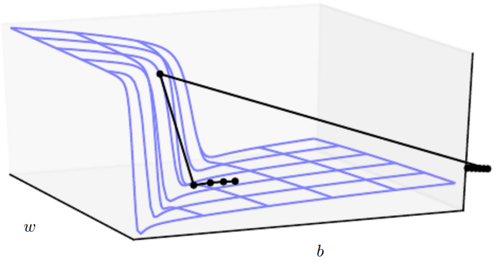

Figure 3: Cliffs and Exploding Gradients, Source: Goodfellow et al., 2016

- **This lecture discusses methods that help converge at a minimum cost with the smallest amount of steps and time**

Outline

Introduction

Parameter Initializations

SGD and Momentum

Adaptive Learning Rates

Parameter Initializations

- Most approaches are simple and heuristic
- It is important to break the symmetry between hidden units

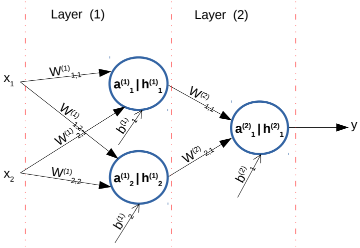

Figure 4: What if initially $W^{(1)}_{1,1} = W^{(1)}_{1,2}$, $W^{(1)}_{2,2} = W^{(1)}_{2,2}$ and $b^{(1)}_1 = b^{(1)}_2$?

- Biases set to heuristically-chosen constant and weights randomly

Parameter Initializations (II)

- Weights:
  - Gaussian Noise $W_{i,j}^{(\ell)} \sim \mathcal{N}(0, \sigma^2)$
  - Commonly used heuristic is the Normalized Initialization (Xavier):

$$W_{i,j}^{(\ell)} \sim \mathcal{U}\left(-\sqrt{\left(\frac{6}{N_\ell + N_{\ell+1}}\right)}, +\sqrt{\left(\frac{6}{N_\ell + N_{\ell+1}}\right)}\right)$$

- Biases:
  - Output layer: $\underset{b^{(L)}}{\operatorname{argmin}} \; \mathcal{L}(h^{(L)}, \bar{y})$
  - Hidden layer: $b^{(\ell)} := c$, when for ReLU set $c > 0$ to avoid the "Dead ReLU" phenomenon

Outline

Stochastic Gradient Descent (SGD)

Algorithm 1: SGD of the k-th epoch

Require: Learning rate ϵ_k, Initial parameters θ

1: **while** *stopping criterion not met* **do**

2: Sample mini-batch: $\left\{\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$

3: Estimate gradient: $\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} \mathcal{L}(f(x^{(i)}; \theta), y^{(i)})$

4: Apply gradient: $\theta \leftarrow \theta - \epsilon\, \hat{g}$

- Converges if $\sum_{k=1}^{\infty} \epsilon_k = \infty$ and $\sum_{k=1}^{\infty} \epsilon_k^2 < \infty$
- In practice, it is common to decay the learning rate:

$$\epsilon_k = \begin{cases} \left(1 - \frac{k}{\tau}\right)\epsilon_0 + \frac{k}{\tau}\epsilon_\tau & \text{if } k < \tau \\ \epsilon_\tau & \text{if } k \geq \tau \end{cases}, \quad \text{where } \epsilon_0 >> \epsilon_\tau$$

Momentum

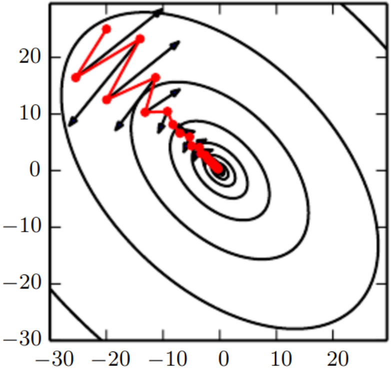

Figure 5: A quadratic loss with a poor conditioned Hessian; Black arrows: Gradient descent steps; Red line: Momentum correction, Source: Goodfellow et al., 2016

Momentum (II)

- Introduce a velocity term v per each parameter
- Accumulate gradient steps in the velocity term
- Update velocity in a moving average style with $\alpha \in [0, 1)$

$$v \leftarrow \alpha v - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(f(x^{(i)}; \theta), y^{(i)} \right) \right)$$

- Ultimately: $\theta \leftarrow \theta + v$
- The step depends on the alignment of a sequence of gradients

Momentum (III)

Algorithm 2: SGD of the k-th epoch with Momentum

Require: Learning rate ϵ_k, Initial parameters θ, Initial velocity v

1: **while** *stopping criterion not met* **do**

2: Sample mini-batch: $\left\{\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$

3: Estimate gradient: $\hat{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_{i=1}^{m} \mathcal{L}(f(x^{(i)}; \theta), y^{(i)})$

4: Update velocity: $v \leftarrow \alpha v - \epsilon \hat{g}$

5: Apply gradient: $\theta \leftarrow \theta + v$

- Commonly $\alpha \in \{0.5, 0.9, 0.99\}$
- Note: It might be useful to think momentum in terms of Newton dynamics $f(t) = \frac{\partial^2}{\partial t^2}\theta(t)$, $f(t) = \frac{\partial}{\partial t}v(t)$, $v(t) = \frac{\partial}{\partial t}\theta(t)$
  - The force pushing weights downwards is $-\nabla_\theta J(\theta)$.

Nesterov Momentum

- Nesterov momentum adds a correction (look-ahead) factor to the standard velocity update

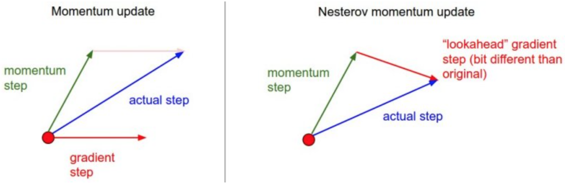

Figure 6: Nesterov momentum with a correction factor

- Computes gradient at the updated weights:

$$v \leftarrow \alpha v - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} \mathcal{L}\left(f(x^{(i)}; \underline{\theta + \alpha v}), y^{(i)} \right) \right)$$

Nesterov Momentum (II)

Algorithm 3: SGD of the k-th epoch with Nesterov Momentum

Require: Learning rate ϵ_k, Initial parameters θ, Initial velocity v

```
while stopping criterion not met do
    Sample mini-batch: {(x^(1), y^(1)), ..., (x^(m), y^(m))}
    Apply interim update: θ̃ ← θ + αv
    Estimate gradient: ĝ ← (1/m) ∇_θ̃ Σ_{i=1}^{m} L(f(x^(i); θ̃), y^(i))
    Update velocity: v ← αv − εĝ
    Apply gradient: θ ← θ + v
```

while *stopping criterion not met* **do**

Sample mini-batch: $\left\{\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$

Apply interim update: $\tilde{\theta} \leftarrow \theta + \alpha v$

Estimate gradient: $\hat{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_{i=1}^{m} \mathcal{L}(f(x^{(i)}; \tilde{\theta}), y^{(i)})$

Update velocity: $v \leftarrow \alpha v - \epsilon \hat{g}$

Apply gradient: $\theta \leftarrow \theta + v$

- Nesterov momentum is currently the most widely used momentum in Deep Learning libraries and is provided by Tensorflow and Theano

Outline

Adagrad

- Strongly decrease learning rate for high gradients
- Rapid progress in gently sloped directions

Algorithm 4: Adagrad learning rate adaptation

Require: Learning rate ϵ_k, Initial parameters θ, Small constant δ

1: **while** *stopping criterion not met* **do**

2: Sample mini-batch: $\left\{\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$

3: Estimate gradient: $\hat{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_{i=1}^{m} \mathcal{L}(f(x^{(i)};\theta), y^{(i)})$

4: Accumulate gradients $\boxed{r \leftarrow r + \hat{g} \odot \hat{g}}$

5: Apply gradient: $\theta \leftarrow \theta - \boxed{\frac{\epsilon}{\delta + \sqrt{r}} \odot \hat{g}}$

RMSProp (Root-mean square propagation)

- As $\sqrt{r}$ increases in Adagrad $\frac{\epsilon}{\sqrt{r}}$ becomes too small
- RMSProp introduces an exponentially decaying average of the gradient history

Algorithm 5: RMSProp learning rate adaptation

Require: Learning rate ϵ_k, Initial parameters θ, Small constant δ, Decay ρ

1: **while** *stopping criterion not met* **do**

2: Sample mini-batch: $\left\{\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$

3: Estimate gradient: $\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} \mathcal{L}(f(x^{(i)}; \theta), y^{(i)})$

4: Accumulate gradients $\boxed{r \leftarrow \rho r + (1-\rho)\hat{g} \odot \hat{g}}$

5: Apply gradient: $\theta \leftarrow \theta - \frac{\epsilon}{\delta + \sqrt{r}} \odot \hat{g}$

RMSProp with Nesterov Momentum

- Adaptive learning rate methods can be combined with momentum

Algorithm 6: RMSProp with Nesterov Momentum

Require: Learning rate ϵ_k, Initial parameters θ, Small constant δ, Decay ρ, Momentum α

1: **while** *stopping criterion not met* **do**

2: Sample mini-batch: $\left\{\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$

3: Apply interim update: $\tilde{\theta} \leftarrow \theta + \alpha v$

4: Estimate gradient: $\hat{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_{i=1}^{m} \mathcal{L}(f(x^{(i)}; \tilde{\theta}), y^{(i)})$

5: Accumulate gradients $r \leftarrow \rho r + (1-\rho)\hat{g} \odot \hat{g}$

6: Update velocity: $v \leftarrow \alpha v - \boxed{\frac{\epsilon}{\delta + \sqrt{r}}} \odot \hat{g}$

7: Update parameter $\theta \leftarrow \theta + v$

Adam (Adaptive Moment)

Algorithm 7: Adam (Suggested $\epsilon = 10^{-3}, \rho_1 = 0.9, \rho_2 = 0.99, \delta = 10^{-8}$)

Require: Learning rate ϵ_k, Initial parameters θ, Small constant δ, Decay rates ρ_1, ρ_2

1: $s = 0,\ r = 0$
2: $t = 0$
3: **while** *stopping criterion not met* **do**
4: Sample mini-batch: $\left\{\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}$
5: Estimate gradient: $\hat{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_{i=1}^{m} \mathcal{L}(f(x^{(i)};\theta), y^{(i)})$
6: $t \leftarrow t + 1$
7: Update gradient accumulator $s \leftarrow \rho_1 s + (1-\rho_1)\hat{g}$
8: Correct gradient accumulator $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$
9: Update squared gradient accumulator $r \leftarrow \rho_2 r + (1-\rho_2)\hat{g} \odot \hat{g}$
10: Correct squared gradient accumulator $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$
11: Update $\theta \leftarrow \theta - \epsilon \frac{\hat{s}}{\delta + \hat{r}}$

Comparing Various Optimization Approaches

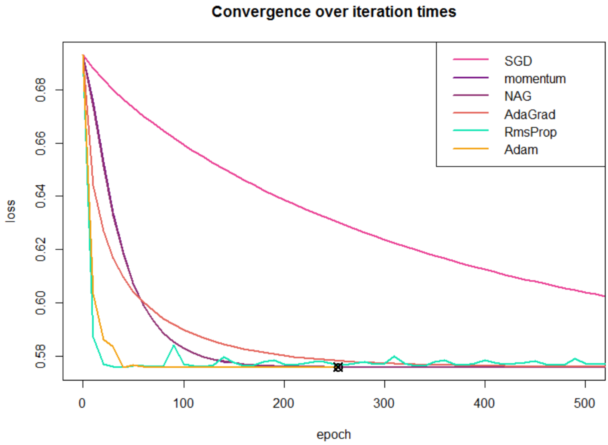

Figure 7: Optimizing a logistic regression model, Source: `gmo.jp`

Illustrations of Performance

Two illustrations (Source: cs.stanford.edu)

- http://cs231n.github.io/assets/nn3/opt1.gif
- http://cs231n.github.io/assets/nn3/opt2.gif