COMPSCI 689
Lecture 13: Optimization for Neural Networks

Benjamin M. Marlin

College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).

Neural Network Optimization Challenges

- Unlike GLMs and SVMs, multi-layer sigmoid neural network models have many local optima.
- They can also have other types of complex features including many saddle points, large nearly flat regions, and highly ill conditioned curvature.
- The computational cost of computing objective functions and gradients can also be very high.
- All of these factors combine to make neural network learning quite difficult for large, deep models, resulting in the need for specialized algorithms and model components.

The ERM Approximation

- Recall that the problem that generated the ERM framework was:

$$\min_{\theta} R_{p_*}(\theta) = \min_{\theta} \mathbb{E}_{p_*(\mathbf{x},y)}[L(y, f_\theta(\mathbf{x}))]$$

- Given a data set $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1:N}$ of size N, we can form an empirical distribution $p_{\mathcal{D}}(\mathbf{x}, y)$. Under the approximation $p_{\mathcal{D}} \approx p_*$, we obtain the empirical risk approximation:

$$\min_{\theta} R_{p_*}(\theta) \approx \min_{\theta} R_{\mathcal{D}}(\theta) = \min_{\theta} \frac{1}{N} \sum_{n=1}^{N} L(y_n, f_\theta(\mathbf{x}_n))$$

The ERM Gradient Approximation

- During learning, we would really like to optimize the risk $R_{p_*}(\theta)$, but under ERM we optimize $R_{\mathcal{D}}(\theta)$ instead.
- This means that when using gradient-based optimization algorithms, instead of using $g_{p_*}(\theta) = \nabla_\theta R_{p_*}(\theta)$ as the gradient, we use:

$$g_{\mathcal{D}}(\theta) = \nabla_\theta R_{\mathcal{D}}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta L(y_n, f_\theta(\mathbf{x}_n))$$

ERM Gradient Properties

- It is easy to show that if a data set is sampled independently from p_*, then $g_{\mathcal{D}}(\theta)$ is an unbiased estimator of $g_{p_*}(\theta)$ for any N. In other words: $\mathbb{E}_{p_*(\mathcal{D})}[g_{\mathcal{D}}(\theta) - g_{p_*}(\theta)] = 0$
- Under independent sampling, the weak law of large numbers shows that the standard deviation of the gradient vector elements scales with $1/\sqrt{N}$ where N is the number of samples used to compute the gradient estimate.
- This indicates that there is a diminishing return on the computation required to form a gradient estimate as N increases.
- For example, if we use $N = 100$ data cases, the standard deviation would be proportional to $1/10$. However, if you do 10 times more computation ($N = 1000$), the standard deviation would be proportional to about $1/30$. A reduction of only a factor of 3.

Stochastic Gradient Descent

- These results suggest that a good strategy for reducing the computational cost of optimizing $R_{\mathcal{D}}(\theta)$ is to approximate the required gradients using a random sub-sample of the available data: $g_{\mathcal{D}'}(\theta) \approx g_{\mathcal{D}}(\theta)$ for $\mathcal{D}' \subset \mathcal{D}$.
- In the literature, this is known as a mini-batch algorithm. More formally, such methods are referred to as stochastic approximation algorithms because they draw a new subset of data from $\mathcal{D}$ at random on every iteration.
- When these ideas are combined with gradient descent, the result is a method called *stochastic gradient descent* or SGD.

The SGD Algorithm

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$

Require: Initial parameter $\boldsymbol{\theta}$

$k \leftarrow 1$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$

$k \leftarrow k + 1$

end while

Convergence of SGD

- Since the gradient $g_{\mathcal{D}'}(\theta)$ is stochastic with respect to $g_{\mathcal{D}}(\theta)$, $g_{\mathcal{D}'}(\theta)$ is not guaranteed to converge to zero as the number of SGD iterations goes to infinity for any constant step size ϵ.
- Instead, the theory of stochastic approximation requires that the sequences of step sizes goes to zero to offset the inherent noise in $g_{\mathcal{D}'}(\theta)$.
- It is sufficient for the sequence of step sizes to satisfy:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \quad \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

Step Size Rules

- A step size proportional to $1/k$ will satisfy this step size condition.
- In practice, more complex rules like the following are often used, even through they violate the square summability:

$$\epsilon_k = \begin{cases} (1 - \frac{k}{t})\alpha + \frac{k}{t}\beta & k \leq t \\ \beta & k > t \end{cases}$$

- This is typically used with $\beta \approx 0.01\alpha$ and $t \approx 100(N/M)$ where N is the size of the data set and M is the batch size.

Conditioning

- The local curvature of the objective function $R_{\mathcal{D}}(\theta)$ is given by the Hessian matrix of $R_{\mathcal{D}}(\theta)$.
- Let $H_{\mathcal{D}}(\theta)$ be the hessian matrix of $R_{\mathcal{D}}(\theta)$ at θ.
- A significant factor in the speed of a gradient-based optimization method is the conditioning of the Hessian matrix.
- The condition number of the hessian is given below where λ_{max} is the maximum eigenvalue of $H_{\mathcal{D}}(\theta)$ and λ_{min} is the minimum eigenvalue of $H_{\mathcal{D}}(\theta)$:

$$\kappa(H_{\mathcal{D}}(\theta)) = \frac{|\lambda_{max}|}{|\lambda_{min}|}$$

Conditioning

- A large condition number means that the objective function increases very rapidly in one direction and very slowly in another orthogonal direction.
- Empirically, deep neural networks suffer from ill-conditioning during optimization, often exhibiting large differences in curvature in different directions.
- A different, but equally difficult problem occurs when the objective function is almost flat so that $g_{\mathcal{D}}(\theta) \approx 0$ and $H_{\mathcal{D}}(\theta) \approx 0$. Empirically, this problem also occurs in neural network optimization.
- Newton's method exactly removes these curvature problems, but it can't be used when the Hessian has negative eigenvalues, which also often occurs with neural networks!

Gradient Clipping

- One commonly used heuristic for dealing with large curvature directions is gradient clipping.
- This method computes the gradient as usual, and then clips the gradient magnitude using a re-normalization step if it exceeds a given threshold.
- This stops SGD from taking overly large step in regions where the curvature is very large.

Momentum

- Momentum is an approach for dealing with differential curvature by taking steps in an accumulated gradient.
- When the gradient has a persistent sign in a given direction, the magnitude of that direction is amplified over time.
- The optimization dynamics mimic a ball with mass rolling down the objective function surface (subject to drag).

SGD with Momentum

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
end while

SGD with Nesterov Momentum

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ, momentum parameter α

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.

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

Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$.

Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$.

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.

end while

Adaptive Step Sizes

- Another way to deal with ill-conditioned problems is to define a different step size for every parameter and to adapt these step sizes over time.
- Current methods do this by re-scaling the individual gradient components using an accumulated gradient magnitude. This amplifies persistently small gradient values and damps persistently large gradient values.
- Examples of this type of method include RMSProp and Adam. These methods can also be combined with momentum.

RMSProp with Nesterov Momentum

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$
  Initialize accumulation variable $\boldsymbol{r} = \mathbf{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$.
    Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$.
  end while

Regularization

- Early stopping is a (very) heuristic method for regularization that uses a validation set to halt learning when overfitting is detected.
- Weight decay is the term often used in the neural network literature for a standard two norm squared regularization on the weights.
- Dropout is a relatively recent heuristic method for regularization based on randomly zeroing the output of a randomly selected collection of inputs/hidden units on each iteration.
- Empirically, it seems to give a consistent improvement when training deep networks, but it is still not completely understood.