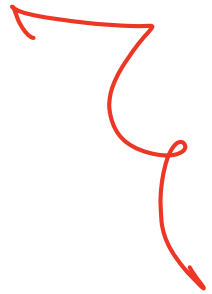
COMPSCI 689
Lecture 12: Optimization for Neural Networks

Brendan O'Connor

College of Information and Computer Sciences
University of Massachusetts Amherst



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

Outline

Neural Network Optimization Challenges

- Unlike GLMs and SVMs, deep neural network models have many local optima.

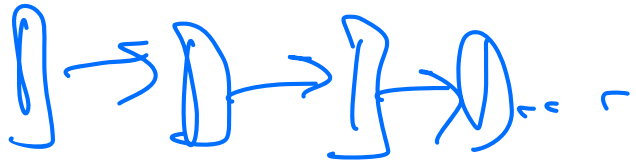
Neural Network Optimization Challenges

- Unlike GLMs and SVMs, deep 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.

Neural Network Optimization Challenges

- Unlike GLMs and SVMs, deep 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.

Deep feedforward net



$$f(W^4 f(W^3 f(W^2 f(W^1 x))))$$

Neural Network Optimization Challenges

- Unlike GLMs and SVMs, deep 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 more specialized algorithms and model components.

Outline

The ERM Gradient Approximation

- In ERM, the goal is to minimize empirical risk:

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

The ERM Gradient Approximation

- In ERM, the goal is to minimize empirical risk:

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

- Recall that the empirical risk $R_{\mathcal{D}}(\theta)$ is an unbiased estimate of the expected risk (loss) under $p_*(\mathbf{x}, y)$ when the data are sampled from $p_*(\mathbf{x}, y)$.

The ERM Gradient Approximation

- In ERM, the goal is to minimize empirical risk:

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

- Recall that the empirical risk $R_{\mathcal{D}}(\theta)$ is an unbiased estimate of the expected risk (loss) under $p_*(\mathbf{x}, y)$ when the data are sampled from $p_*(\mathbf{x}, y)$.

- Similarly, the gradient of the empirical risk $g_{\mathcal{D}}(\theta)$ is an unbiased estimate of the gradient of the expected risk.

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

The ERM Gradient Approximation

- In ERM, the goal is to minimize empirical risk:

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

- Recall that the empirical risk $R_{\mathcal{D}}(\theta)$ is an unbiased estimate of the expected risk (loss) under $p_*(\mathbf{x}, y)$ when the data are sampled from $p_*(\mathbf{x}, y)$.
- Similarly, the gradient of the empirical risk $g_{\mathcal{D}}(\theta)$ is an unbiased estimate of the gradient of the expected risk.

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

- Importantly, this is true for any $N > 0$.

ERM Gradient Properties

- Under independent sampling, the weak law of large numbers shows that the standard deviation of the gradient vector elements scales with $1/\sqrt{m}$ where $m \leq N$ is the number of samples used to compute the gradient estimate.

$X_i = \nabla_\theta R(y_i, f(x_i; \theta))$ same $\sigma^2 = \mathrm{Var} X_i \ \forall i$

$\sigma \underbrace{\mathrm{Var}\left[\frac{1}{m}\sum_{i=1}^{m} X_i\right]} = \frac{1}{m^2}\sum_i \mathrm{Var} X_i$

$= \frac{m\sigma^2}{m^2} = \frac{\sigma^2}{m}$

ERM Gradient Properties

- Under independent sampling, the weak law of large numbers shows that the standard deviation of the gradient vector elements scales with $1/\sqrt{m}$ where $m \leq N$ 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 m increases.

ERM Gradient Properties

- Under independent sampling, the weak law of large numbers shows that the standard deviation of the gradient vector elements scales with $1/\sqrt{m}$ where $m \leq N$ 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 m increases.
- For example, if we use $m = 100$ data cases, the standard deviation would be proportional to $1/10$. However, if you do 10 times more computation ($m = 1000$), 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 $\mathcal{D}'$ of the available data: $g_{\mathcal{D}'}(\theta) \approx g_{\mathcal{D}}(\theta)$ for $\mathcal{D}' \subset \mathcal{D}$.

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 $\mathcal{D}'$ 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.

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 $\mathcal{D}'$ 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.

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 $g_{\mathcal{D}'}(\theta)$.

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:

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

$\alpha \sum_k \frac{1}{k} \to \infty$ $\alpha \sum_k \frac{1}{k^2} \to$ finite

Step Size Rules

- A step size proportional to $1/k$ will satisfy this step size condition.

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 sometimes used, even through they violate summability:

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

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 sometimes used, even through they violate 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.

Forming Mini-Batches

- While the algorithm works if you draw a random sample on every iteration, it is more common to first randomly shuffle the data, then break it into N/m contiguous blocks of size $m < N$.

Forming Mini-Batches

- While the algorithm works if you draw a random sample on every iteration, it is more common to first randomly shuffle the data, then break it into N/m contiguous blocks of size $m < N$.
- We then loop through the blocks in order. This version of the algorithm has identical properties to when new random samples are drawn.

Forming Mini-Batches

- While the algorithm works if you draw a random sample on every iteration, it is more common to first randomly shuffle the data, then break it into N/m contiguous blocks of size $m < N$.
- We then loop through the blocks in order. This version of the algorithm has identical properties to when new random samples are drawn.
- In the literature, an “epoch" of training refers to one complete pass through all of the N/m batches of data. At the end of an epoch, all data cases have been used exactly one time.

Forming Mini-Batches

- While the algorithm works if you draw a random sample on every iteration, it is more common to first randomly shuffle the data, then break it into N/m contiguous blocks of size $m < N$.
- We then loop through the blocks in order. This version of the algorithm has identical properties to when new random samples are drawn.
- In the literature, an “epoch" of training refers to one complete pass through all of the N/m batches of data. At the end of an epoch, all data cases have been used exactly one time.
- Learning time is often described in terms of epochs instead of iterations.

Outline

Conditioning

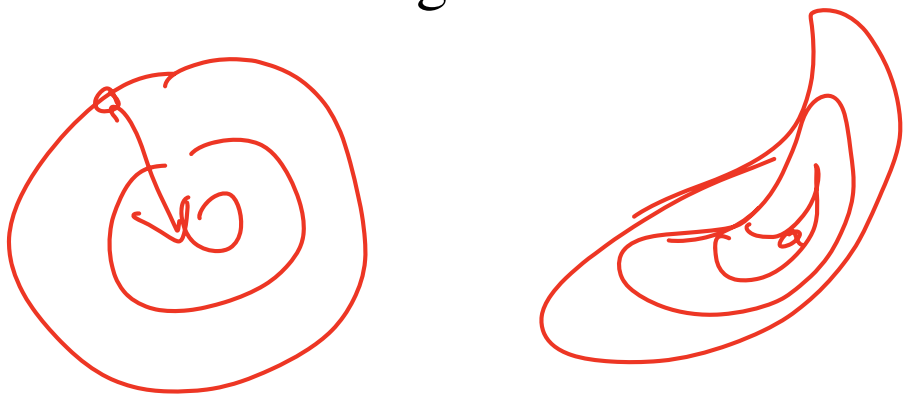
- The local curvature of the objective function $R_{\mathcal{D}}(\theta)$ is given by the Hessian matrix of $R_{\mathcal{D}}(\theta)$.

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.

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.



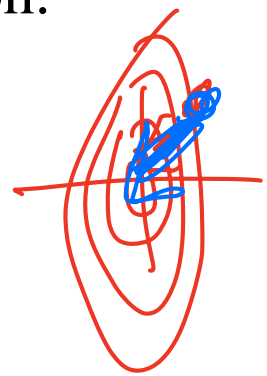
Conditioning

- 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 magnitude eigenvalue of $H_{\mathcal{D}}(\theta)$ and λ_{min} is the minimum magnitude 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.



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.

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 $g_{\mathcal{D}}(\theta) \approx 0$. Empirically, this problem also occurs in neural network optimization.

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 $g_{\mathcal{D}}(\theta) \approx 0$. Empirically, this problem also occurs in neural network optimization.
- Newton's method (locally) 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

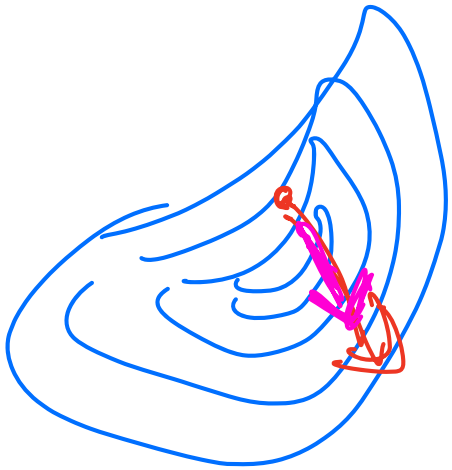
- One commonly used heuristic for dealing with large curvature directions is gradient clipping.

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.

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

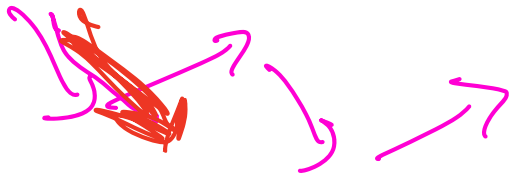
- Momentum is an approach for dealing with differential curvature by taking steps in an accumulated gradient.

Momentum

- Momentum is an approach for dealing with differential curvature by taking steps in an accumulated gradient.
- When the gradient vector consistently has a component in a given direction, the magnitude of that component is amplified over iterations.

Momentum

- Momentum is an approach for dealing with differential curvature by taking steps in an accumulated gradient.
- When the gradient vector consistently has a component in a given direction, the magnitude of that component is amplified over iterations.
- 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 $\boldsymbol{y}^{(i)}$.

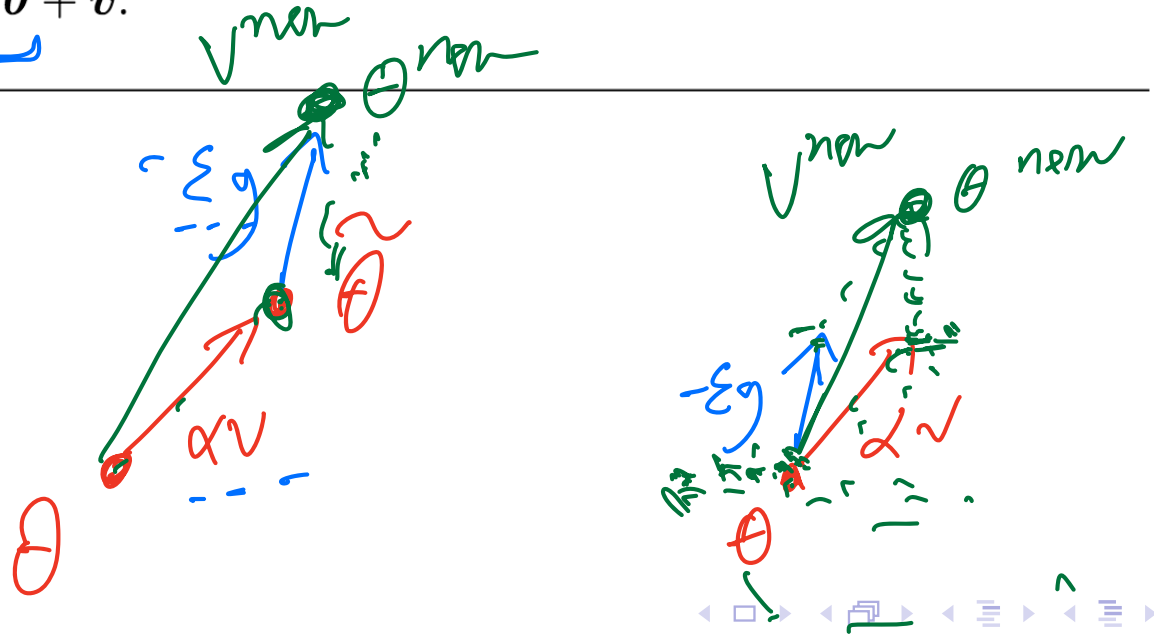
Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$.

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



Outline

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.

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.

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

Outline

Regularization

- Early stopping is a (very) heuristic method for regularization that uses a validation set to halt learning when overfitting is detected.

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.

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 heuristic method for regularization based on randomly zeroing the output of a randomly selected collection of inputs/hidden units on each iteration.

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 heuristic method for regularization based on randomly zeroing the output of a randomly selected collection of inputs/hidden units on each iteration.
- Empirically, Dropout seems to give a consistent improvement when training deep networks, but it is still not completely understood.

Outline

Batch Normalization

- Normalization methods aim to modify learning dynamics by partially decoupling the effect of parameter updates across different parts of the network.

Batch Normalization

- Normalization methods aim to modify learning dynamics by partially decoupling the effect of parameter updates across different parts of the network.
- This can help to speed up learning for models by partially mitigating the effect of higher order interaction between parameter values during learning.

Batch Normalization

- Normalization methods aim to modify learning dynamics by partially decoupling the effect of parameter updates across different parts of the network.
- This can help to speed up learning for models by partially mitigating the effect of higher order interaction between parameter values during learning.
- This often allows for larger learning rates and enables faster model learning.

Batch Normalization

- Suppose $\mathbf{h}_n \in \mathbb{R}^K$ is the vector of hidden unit values for layer l and data case n. The batch normalization transformation is:

$$\mathbf{h}'_{kn} = \beta_k + \gamma_k \left(\frac{\mathbf{h}_{kn} - \mu_k}{\sqrt{\epsilon + \sigma_k^2}} \right)$$

Batch Normalization

- Suppose $\mathbf{h}_n \in \mathbb{R}^K$ is the vector of hidden unit values for layer l and data case n. The batch normalization transformation is:

$$\mathbf{h}'_{kn} = \beta_k + \gamma_k \left(\frac{\mathbf{h}_{kn} - \mu_k}{\sqrt{\epsilon + \sigma_k^2}} \right)$$

- In this expression μ_k is the mean of $\mathbf{h}_{kn}$ computed over a minibatch of m data cases, and σ_k^2 is the corresponding variance of $\mathbf{h}_{kn}$ computed over the same minibatch of m data cases.

Batch Normalization

- Suppose $\mathbf{h}_n \in \mathbb{R}^K$ is the vector of hidden unit values for layer l and data case n. The batch normalization transformation is:

$$\mathbf{h}'_{kn} = \beta_k + \gamma_k \left(\frac{\mathbf{h}_{kn} - \mu_k}{\sqrt{\epsilon + \sigma_k^2}} \right)$$

- In this expression μ_k is the mean of $\mathbf{h}_{kn}$ computed over a minibatch of m data cases, and σ_k^2 is the corresponding variance of $\mathbf{h}_{kn}$ computed over the same minibatch of m data cases.

- This transformation sets the mean and variance of hidden unit k, to γ_k and β_k respectively. Both parameters are learned.

Batch Normalization

- Suppose $\mathbf{h}_n \in \mathbb{R}^K$ is the vector of hidden unit values for layer l and data case n. The batch normalization transformation is:

$$\mathbf{h}'_{kn} = \beta_k + \gamma_k \left(\frac{\mathbf{h}_{kn} - \mu_k}{\sqrt{\epsilon + \sigma_k^2}} \right)$$

- In this expression μ_k is the mean of $\mathbf{h}_{kn}$ computed over a minibatch of m data cases, and σ_k^2 is the corresponding variance of $\mathbf{h}_{kn}$ computed over the same minibatch of m data cases.
- This transformation sets the mean and variance of hidden unit k, to γ_k and β_k respectively. Both parameters are learned.
- This means that instead of the hidden unit statistics being a complex function of the parameters in earlier parts of the model, they depend only on the local parameters γ_k and β_k.

Batch Normalization

- Batch normalization thus provides robustness to changes in the inputs to a given layer that are caused by changes in layers below it during learning, allowing it to provide more stable inputs to the next layer.

Batch Normalization

- Batch normalization thus provides robustness to changes in the inputs to a given layer that are caused by changes in layers below it during learning, allowing it to provide more stable inputs to the next layer.
- While there are still higher-order interactions across layers and there may still be ill-conditioning for the weights within a layer, this type of normalization has been shown to be very helpful for some learning problems.

Batch Normalization

- Batch normalization thus provides robustness to changes in the inputs to a given layer that are caused by changes in layers below it during learning, allowing it to provide more stable inputs to the next layer.
- While there are still higher-order interactions across layers and there may still be ill-conditioning for the weights within a layer, this type of normalization has been shown to be very helpful for some learning problems.
- Other techniques like weight normalization and layer normalization attempt to improve on the properties of batch normalization such as limits on the batch size.