CS551: Introduction to Deep Learning
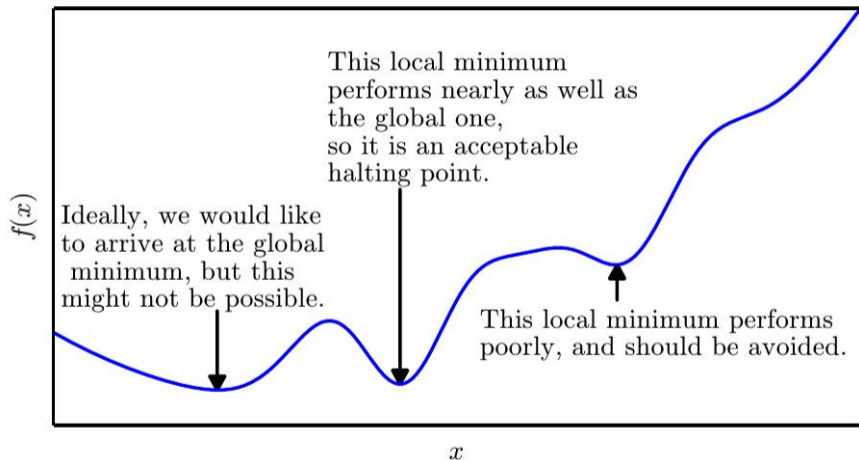
Optimization

Arijit Mondal

Dept. of Computer Science & Engineering

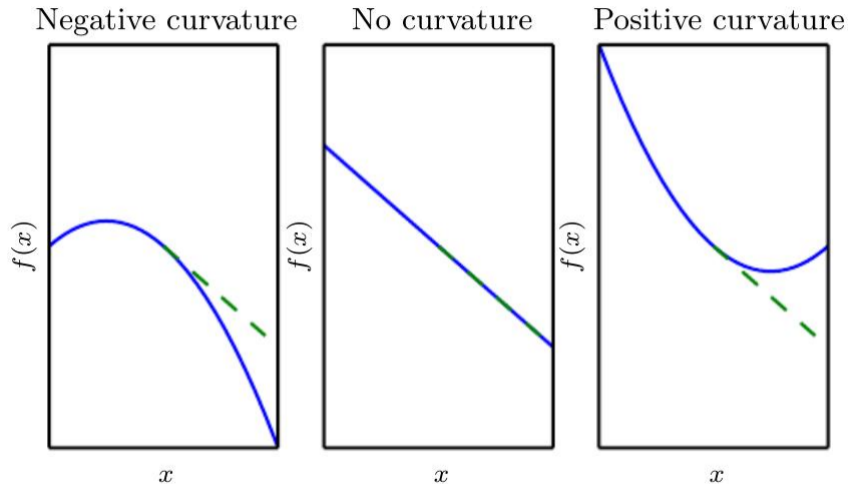**Indian Institute of Technology Patna**

arijit@iitp.ac.in

Minimization of cost function

Approximate minimization

Image source: Deep Learning Book

Curvature

Negative curvature   No curvature   Positive curvature

Image source: Deep Learning Book

Problem of optimization

- Differs from traditional pure optimization problem

- Performance of a task is optimized indirectly

- We optimize $J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathrm{x},y)\sim\hat{p}_{\mathrm{data}}} L(f(\mathrm{x}, \boldsymbol{\theta}), y)$ where $\hat{p}$ is the empirical distribution

- We would like to optimize $J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\mathrm{x},y)\sim p_{\mathrm{data}}} L(f(\mathrm{x}, \boldsymbol{\theta}), y)$ where p is the data generating distribution

  - Also known as risk

- We hope minimizing J will minimize J^*

CS551

4

Empirical risk minimization

- Target is to reduce risk

- If the true distribution is known, risk minimization is an optimization problem

- When $p_{\text{data}}(\mathsf{x}, y)$ is unknown, it becomes machine learning problem

- Simplest way to convert machine learning problem to optimization problem is to minimize expected cost of training set

- We minimize empirical risk

$$\mathbb{E}_{(\mathsf{x},y)\sim\hat{p}_{\text{data}}}[L(f(\mathsf{x}, \boldsymbol{\theta}), y)] = \frac{1}{m}\sum_i L(f(x^{(i)}, \boldsymbol{\theta}), y^{(i)})$$

  - We can hope empirical risk minimizes the risk as well
    - Empirical risk minimization is prone to overfitting
    - Gradient based solution approach may lead to problem with 0-1 loss cost function

Surrogate loss function

- Loss function may not be optimized efficiently
  - Exact minimization of 0-1 loss is typically intractable
- Surrogate loss function is used
  - Proxy function for the actual loss function
  - Negative log likelihood of correct class used as surrogate function
- There are cases when surrogate loss function results in better learning
  - 0-1 loss of test set often continues to decrease for a long time after training set 0-1 loss has reached to 0
- A training algorithm does not halt at local minima usually
  - Tries to minimize surrogate loss function but halts when validation loss starts to increase
- Training function can halt when surrogate function has huge derivative

Batch

- Objective function usually decomposes as a sum over training example

- Typically in machine learning update of parameters is done based on an expected value of the cost function estimated using only a subset of the terms of full cost function

- Maximum likelihood problem $\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}(\mathbf{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$

- Maximizing this sum is equivalent to maximizing the expectation over empirical distribution $J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y) \sim \hat{p}_{\text{data}}} \log p_{\text{model}}(\mathbf{x}, y, \boldsymbol{\theta})$

Batch (contd.)

- Common gradient is given by $\nabla_{\boldsymbol{\theta}} = \mathbb{E}_{(x,y) \sim \hat{p}_{\text{data}}} \nabla_{\boldsymbol{\theta}} \log p_{\text{model}}(x, y, \boldsymbol{\theta})$
  - It becomes expensive as we need to compute for all examples
  - Random sample is chosen, then average of the same is taken
  - Standard error in mean is $\frac{\sigma}{\sqrt{n}}$ where σ is the true standard deviation
  - Redundancy in training examples is an issue
- Optimization algorithm that uses entire training set is called batch of deterministic gradient descent
- Optimization algorithm that uses single example at a time is known as stochastic gradient descent or online method

Minibatch

- Larger batch provides more accurate estimate of the gradient but with lesser than linear returns

- Multicore architecture are usually underutilized by small batches

- If all examples are to be processed parallely then the amount of memory scales with batch size

- Sometime, better run time is observed with specific size of the array

- Small batch can add regularization effect due to noise they add in learning process

- Methods that update the parameters based on g only are usually robust and can handle small batch size ~ 100

- With Hessian matrix batch size becomes $\sim 10,000$ (Require to minimize $H^{-1}g$)

- SGD minimizes generalization error on minibatches drawn from a stream of data

Local quadratic approximation

- Taylor expansion of $J(w)$ around $\hat{w}$ in weight space is given by

$$J(w) = J(\hat{w}) + (w - \hat{w})^{\mathsf{T}} g + \frac{1}{2}(w - \hat{w})^{\mathsf{T}} H(w - \hat{w})$$

where $g = \nabla J(w)|_{w=\hat{w}}$ and $H = \nabla\nabla J(w)|_{w=\hat{w}}$

Local quadratic approximation

- Taylor expansion of $J(\mathsf{w})$ around $\hat{\mathsf{w}}$ in weight space is given by

$$J(\mathsf{w}) = J(\hat{\mathsf{w}}) + (\mathsf{w} - \hat{\mathsf{w}})^\mathsf{T}\mathsf{g} + \frac{1}{2}(\mathsf{w} - \hat{\mathsf{w}})^\mathsf{T} H(\mathsf{w} - \hat{\mathsf{w}})$$

where $\mathsf{g} = \nabla J(\mathsf{w})|_{\mathsf{w}=\hat{\mathsf{w}}}$ and $H = \nabla\nabla J(\mathsf{w})|_{\mathsf{w}=\hat{\mathsf{w}}}$

- Gradient can be approximated as

Local quadratic approximation

- Taylor expansion of $J(w)$ around $\hat{w}$ in weight space is given by

$$J(w) = J(\hat{w}) + (w - \hat{w})^\mathsf{T} g + \frac{1}{2}(w - \hat{w})^\mathsf{T} H(w - \hat{w})$$

  where $g = \nabla J(w)|_{w=\hat{w}}$ and $H = \nabla\nabla J(w)|_{w=\hat{w}}$

- Gradient can be approximated as $\nabla J(w) = g + H(w - \hat{w})$

Local quadratic approximation

- Taylor expansion of $J(\mathrm{w})$ around $\hat{\mathrm{w}}$ in weight space is given by

  $$J(\mathrm{w}) = J(\hat{\mathrm{w}}) + (\mathrm{w} - \hat{\mathrm{w}})^{\mathsf{T}} \mathrm{g} + \frac{1}{2}(\mathrm{w} - \hat{\mathrm{w}})^{\mathsf{T}} H(\mathrm{w} - \hat{\mathrm{w}})$$

  where $\mathrm{g} = \nabla J(\mathrm{w})|_{\mathrm{w}=\hat{\mathrm{w}}}$ and $H = \nabla\nabla J(\mathrm{w})|_{\mathrm{w}=\hat{\mathrm{w}}}$

- Gradient can be approximated as $\nabla J(\mathrm{w}) = \mathrm{g} + H(\mathrm{w} - \hat{\mathrm{w}})$

- In case w^* is a minimum point, then the expansion can be written as

  $$J(\mathrm{w}) = J(\mathrm{w}^*) + \frac{1}{2}(\mathrm{w} - \mathrm{w}^*)^{\mathsf{T}} H(\mathrm{w} - \mathrm{w}^*), \ H \text{ needs to be evaluated at } \mathrm{w}^*$$

- Consider the eigenvalue equation for the Hessian matrix ie. $H\mathrm{u}_i = \lambda_i \mathrm{u}_i$

Local quadratic approximation

- Taylor expansion of $J(w)$ around $\hat{w}$ in weight space is given by

  $$J(w) = J(\hat{w}) + (w - \hat{w})^\mathsf{T} g + \frac{1}{2}(w - \hat{w})^\mathsf{T} H(w - \hat{w})$$

  where $g = \nabla J(w)|_{w = \hat{w}}$ and $H = \nabla\nabla J(w)|_{w = \hat{w}}$

- Gradient can be approximated as $\nabla J(w) = g + H(w - \hat{w})$

- In case w^* is a minimum point, then the expansion can be written as

  $$J(w) = J(w^*) + \frac{1}{2}(w - w^*)^\mathsf{T} H(w - w^*), \; H \text{ needs to be evaluated at } w^*$$

- Consider the eigenvalue equation for the Hessian matrix ie. $Hu_i = \lambda_i u_i$

- The eigenvectors u_i form a complete orthonormal set so that $u_i u_j = 0$ when $i \neq j$ and 1 when $i = j$

Local quadratic approximation

- Taylor expansion of $J(\mathrm{w})$ around $\hat{\mathrm{w}}$ in weight space is given by

$$J(\mathrm{w}) = J(\hat{\mathrm{w}}) + (\mathrm{w} - \hat{\mathrm{w}})^{\mathsf{T}}\mathrm{g} + \frac{1}{2}(\mathrm{w} - \hat{\mathrm{w}})^{\mathsf{T}} H (\mathrm{w} - \hat{\mathrm{w}})$$

where $\mathrm{g} = \nabla J(\mathrm{w})|_{\mathrm{w}=\hat{\mathrm{w}}}$ and $H = \nabla\nabla J(\mathrm{w})|_{\mathrm{w}=\hat{\mathrm{w}}}$

- Gradient can be approximated as $\nabla J(\mathrm{w}) = \mathrm{g} + H(\mathrm{w} - \hat{\mathrm{w}})$

- In case w^* is a minimum point, then the expansion can be written as

$$J(\mathrm{w}) = J(\mathrm{w}^*) + \frac{1}{2}(\mathrm{w} - \mathrm{w}^*)^{\mathsf{T}} H (\mathrm{w} - \mathrm{w}^*), \; H \text{ needs to be evaluated at } \mathrm{w}^*$$

- Consider the eigenvalue equation for the Hessian matrix ie. $H\mathrm{u}_i = \lambda_i \mathrm{u}_i$

- The eigenvectors u_i form a complete orthonormal set so that $\mathrm{u}_i\mathrm{u}_j = 0$ when $i \neq j$ and 1 when $i = j$

- Treating eigenvectors as basis, we can write $\mathrm{w} - \mathrm{w}^* = \sum_i \alpha_i \mathrm{u}_i$. Now we have

Local quadratic approximation

- Taylor expansion of $J(w)$ around $\hat{w}$ in weight space is given by

$$J(w) = J(\hat{w}) + (w - \hat{w})^\mathsf{T} g + \frac{1}{2}(w - \hat{w})^\mathsf{T} H(w - \hat{w})$$

  where $g = \nabla J(w)|_{w=\hat{w}}$ and $H = \nabla\nabla J(w)|_{w=\hat{w}}$

- Gradient can be approximated as $\nabla J(w) = g + H(w - \hat{w})$

- In case w^* is a minimum point, then the expansion can be written as

$$J(w) = J(w^*) + \frac{1}{2}(w - w^*)^\mathsf{T} H(w - w^*),\ H \text{ needs to be evaluated at } w^*$$

- Consider the eigenvalue equation for the Hessian matrix ie. $H u_i = \lambda_i u_i$

- The eigenvectors u_i form a complete orthonormal set so that $u_i u_j = 0$ when $i \neq j$ and 1 when $i = j$

- Treating eigenvectors as basis, we can write $w - w^* = \sum_i \alpha_i u_i$. Now we have

$$J(w) = J(w^*) + \frac{1}{2}\sum_i \lambda_i \alpha_i^2$$

Issues in optimization

- Ill conditioning
- Local minima
- Plateaus
- Saddle points
- Flat region

- Cliffs
- Exploding gradients
- Vanishing gradients
- Long term dependencies
- Inexact gradients

Ill conditioning

- Ill conditioning of Hessian matrix
  - Common problem in most of the numerical optimization
  - The ratio of smallest to largest eigen value determines the condition number
  - We have the following

$$f(\mathbf{x}) = f(\mathbf{x}^{(0)}) + (\mathbf{x} - \mathbf{x}^{(0)})^T \mathbf{g} + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(0)})^T \mathbf{H}(\mathbf{x} - \mathbf{x}^{(0)})$$

$$f(\mathbf{x} - \epsilon \mathbf{g}) = f(\mathbf{x}^{(0)}) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2}\epsilon \mathbf{g}^T \mathbf{H} \epsilon \mathbf{g}$$

  - It becomes a problem when $\frac{1}{2}\epsilon^2 \mathbf{g}^T \mathbf{H} \mathbf{g} - \epsilon \mathbf{g}^T \mathbf{g} > 0$
  - In many cases gradient norm does not shrink much during learning and $\mathbf{g}^T \mathbf{H} \mathbf{g}$ grows more rapidly
  - Makes the learning process slow

Local minima

- For convex optimization problem local minima is often acceptable
- For nonconvex function like neural network many local minima are possible
  - This is not a major problem

Local minima (contd.)

- Neural network and any models with multiple equivalently parameterized latent variables results in local minima
  - This is due to model identifiability
  - Model is identifiable if sufficiently large training set can rule out all but one setting of model parameters
  - Model with latent variables are often not identifiable as exchanging of two variables does not change the model
    - m layers with n unit each can result in $(n!)^m$ arrangements
    - This non-identifiability is known as weight space symmetry
  - Neural network has other non-identifiability scenario
    - ReLU or MaxOut — weight is scaled by α and output is scaled by $\frac{1}{\alpha}$

Local minima (contd.)

- Model identifiability issues mean that there can be uncountably infinite number of local minima

- Non-identifiability results in local minima and are equivalent to each other in cost function

- Local minima can be problematic if they have high cost compared to global minima

Other issues

- Saddle points
  - Gradient is 0 but some have higher and some have lower value around the point
  - Hessian matrix has both positive and negative eigen value
- In high dimension local minima are rare, saddle points are common
  - For a function $f : \mathbb{R}^n \to \mathbb{R}$, the expected ratio of number of saddle points to local minima grows exponentially with n
    - Eigenvalue of Hessian matrix
- Cliffs - uses gradient clipping
- Long term dependency - mostly applicable for RNN
  - $w^t = V \text{diag}(\boldsymbol{\lambda})^t V^{-1}$
  - vanishing and exploding gradient
- Inexact gradients — bias in estimation of gradient

Stochastic gradient descent

- Inputs — Learning rate (ϵ_k), weight parameters ($\boldsymbol{\theta}$)
- Algorithm for SGD:

while stopping criteria not met

    Sample a minibatch $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ with labels $\{y^{(i)}\}$

    Estimate of gradient $\hat{g} = \dfrac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(x^{(i)}, \boldsymbol{\theta}), y^{(i)})$

    Update parameters $\boldsymbol{\theta} = \boldsymbol{\theta} - \epsilon_k \hat{g}$

end while

CS551

Convergence

- Gradient can be approximated as $\nabla J(w) = g + H(w - \hat{w})$
- In the vicinity of w^*,

  $\nabla J(w) =$

Convergence

- Gradient can be approximated as $\nabla J(w) = g + H(w - \hat{w})$
- In the vicinity of w^*,

  $\nabla J(w) = H(w - w^*) =$

Convergence

- Gradient can be approximated as $\nabla J(\mathsf{w}) = \mathsf{g} + H(\mathsf{w} - \hat{\mathsf{w}})$

- In the vicinity of w^*,

$$\nabla J(\mathsf{w}) = H(\mathsf{w} - \mathsf{w}^*) = \sum_i \alpha_i \lambda_i \mathsf{u}_i$$

Convergence

- Gradient can be approximated as $\nabla J(\mathbf{w}) = \mathbf{g} + H(\mathbf{w} - \hat{\mathbf{w}})$

- In the vicinity of $\mathbf{w}^*$,

$$\nabla J(\mathbf{w}) = H(\mathbf{w} - \mathbf{w}^*) = \sum_i \alpha_i \lambda_i \mathbf{u}_i$$

- We can express the change in weight vector in terms of corresponding changes in coefficients $\{\alpha_i\}$

$$\Delta \mathbf{w} = \sum_i \Delta \alpha_i \mathbf{u}_i$$

Convergence

- Gradient can be approximated as $\nabla J(\mathrm{w}) = \mathrm{g} + H(\mathrm{w} - \hat{\mathrm{w}})$

- In the vicinity of w^*,
$$\nabla J(\mathrm{w}) = H(\mathrm{w} - \mathrm{w}^*) = \sum_i \alpha_i \lambda_i \mathrm{u}_i$$

- We can express the change in weight vector in terms of corresponding changes in coefficients $\{\alpha_i\}$
$$\Delta \mathrm{w} = \sum_i \Delta \alpha_i \mathrm{u}_i$$

- Using gradient descent and above expressions,
$$\Delta \alpha_i = -\eta \lambda_i \alpha_i$$

Convergence

- Gradient can be approximated as $\nabla J(w) = g + H(w - \hat{w})$

- In the vicinity of w^*,
$$\nabla J(w) = H(w - w^*) = \sum_i \alpha_i \lambda_i u_i$$

- We can express the change in weight vector in terms of corresponding changes in coefficients $\{\alpha_i\}$
$$\Delta w = \sum_i \Delta \alpha_i u_i$$

- Using gradient descent and above expressions,
$$\Delta \alpha_i = -\eta \lambda_i \alpha_i$$

- Hence, we can say, $\alpha_i^{\mathsf{new}} = (1 - \eta \lambda_i) \alpha_i^{\mathsf{old}} \Rightarrow \alpha_i^{\tau} = (1 - \eta \lambda_i)^{\tau} \alpha_i^{(0)}$

Stochastic gradient descent

- Learning rate is a crucial parameter
- Learning rate ϵ_k is used in the kth iteration
- Gradient does not vanishes even when we reach minima as minibatch can introduce noise
- True gradient becomes small and then 0 when batch gradient descent is used
- Sufficient condition on learning rate for convergence of SGD

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

- Common way is to decay the learning rate $\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$ with $\alpha = \frac{k}{\tau}$
- Choosing learning rate is an art than science!
  - Typically ϵ_τ is 1% of ϵ_0
- SGD usually performs well for most of the cases
- For large task set SGD may converge within the fixed tolerance of final error before it has processed all training examples
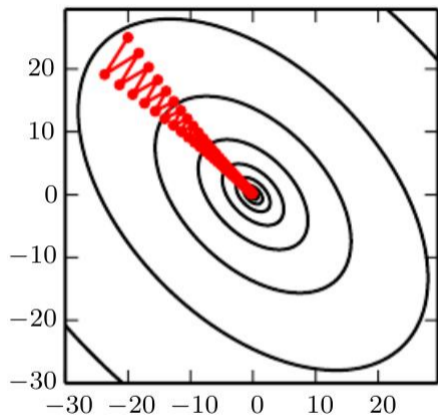
Momentum

- SGD is the most popular. However, learning may be slow sometime
- Idea is to accelerate learning especially in high curvature, small but consistent gradients
- Accumulates an exponential decaying moving average of past gradients and continue to move in that direction
- Introduces a parameter v that play the role of velocity
  - The velocity is set to an exponentially decaying average of negative gradients
- Update is given by

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

  - α — hyperparameter, denotes the decay rate

Momentum

Image source: Deep Learning Book
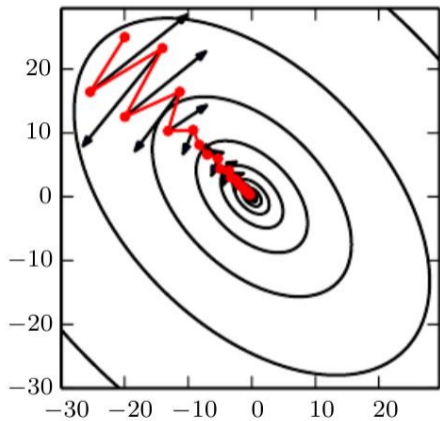
SGD with momentum

- Inputs — Learning rate (ϵ), weight parameters ($\boldsymbol{\theta}$), momentum parameter (α), initial velocity (v)

- Algorithm:

  while stopping criteria not met

      Sample a minibatch from set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ with labels $\{y^{(i)}\}$

      Estimate of gradient: $g = \dfrac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(x^{(i)}, \boldsymbol{\theta}), y^{(i)})$

      Update of velocity: $v = \alpha v - \epsilon g$

      Update parameters: $\boldsymbol{\theta} = \boldsymbol{\theta} + v$

  end while

Momentum

- The step size depends on how large and how aligned a sequence gradients are

- Largest when many successive gradients are in same direction

- If it observes g always, then it will accelerate in $-g$ with terminal velocity $\dfrac{\epsilon|g|}{1-\alpha}$

- Typical values for α is 0.5, 0.9, 0.99. However this parameter can be adapted.

Nesternov momentum

- Inputs — Learning rate (ϵ), weight parameters ($\boldsymbol{\theta}$), momentum parameter (α), initial velocity (v)

- Algorithm:

  while stopping criteria not met

      Sample a minibatch from set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ with labels $\{y^{(i)}\}$

      Interim update: $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + \alpha v$

      Gradient at interim point: $g = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(x^{(i)}, \tilde{\boldsymbol{\theta}}), y^{(i)})$

      Update of velocity: $v = \alpha v - \epsilon g$

      Update parameters: $\boldsymbol{\theta} = \boldsymbol{\theta} + v$

  end while

Parameter initialization

- Training algorithms are iterative in nature

- Require to specify initial point

- Training deep model is difficult task and affected by initial choice
  - Convergence
  - Computation time
  - Numerical instability

- Need to break symmetry while initializing the parameters

Adaptive learning rate

- Learning rate can affect the performance of the model
- Cost may be sensitive in one direction and insensitive in the other directions
- If partial derivative of loss with respect to model remains the same sign then the learning rate should decrease
  - Applicable for full batch optimization

AdaGrad

- Adapts the learning rate of all parameters by scaling them inversely proportional to the square root of the sum of all historical squared values of the gradient
  - Parameters with largest partial derivative of the loss will have rapid decrease in learning rate and vice-versa
  - Net effect is greater progress
- It performs well on some models

Steps for AdaGrad

- Inputs — Global learning rate (ϵ), weight parameters ($\boldsymbol{\theta}$), small constant (δ), gradient accumulation (r)

- Algorithm:

  while stopping criteria not met

      Sample a minibatch from set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ with labels $\{y^{(i)}\}$

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

      Accumulated squared gradient: $r = r + g \odot g$

      Update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$

      Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

  end while

RMSProp

- Gradient is accumulated using an exponentially weighted moving average
  - Usually, AdaGrad converges rapidly in case of convex function
  - AdaGrad reduces the learning rate based on entire history
- RMSProp tries to discard history from extreme past
- This can be combined with momentum

Steps for RMSProp

- Inputs — Global learning rate (ϵ), weight parameters ($\boldsymbol{\theta}$), small constant (δ), gradient accumulation (r), decay rate (ρ)

- Algorithm:

    while stopping criteria not met

        Sample a minibatch from set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ with labels $\{y^{(i)}\}$

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

        Accumulated squared gradient: $r = \rho r + (1 - \rho) g \odot g$

        Update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + r}} \odot g$

        Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

    end while

Steps for RMSProp with Nesternov

- Inputs — Global learning rate (ϵ), weight parameters ($\boldsymbol{\theta}$), small constant (δ), gradient accumulation (r), decay rate (ρ), initial velocity (v), momentum coefficient (α)

- Algorithm:

  while stopping criteria not met

      Sample a minibatch from set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ with labels $\{y^{(i)}\}$

      Interim update: $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta} + \alpha v$

      Gradient: $g = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(x^{(i)}, \tilde{\boldsymbol{\theta}}), y^{(i)})$

      Accumulated squared gradient: $r = \rho r + (1 - \rho)g \odot g$

      Update of velocity: $v = \alpha v - \frac{\epsilon}{\sqrt{r}} \odot g$

      Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + v$

  end while

Approximate 2nd order method

- Taking 2nd order term to train deep neural network
- The cost function at $\boldsymbol{\theta}$ near the point $\boldsymbol{\theta}_0$ is given by

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \mathsf{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- Solution for critical point provides $\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \mathsf{H}^{-1}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$
  - If the function is quadratic then it jumps to minimum
  - If the surface is not quadratic but H is positive definite then this approach is also applicable
- This approach is known as Newton's method

Steps for Newton's method

- Inputs — Initial parameters ($\boldsymbol{\theta}_0$)
- Algorithm:

  while stopping criteria not met

      Sample a minibatch from set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ with labels $\{y^{(i)}\}$

      Compute gradient: $g = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(x^{(i)}, \boldsymbol{\theta}), y^{(i)})$

      Compute Hessian: $H = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}}^2 L(f(x^{(i)}, \boldsymbol{\theta}), y^{(i)})$

      Compute inverse Hessian: H^{-1}

      Compute update: $\Delta \boldsymbol{\theta} = -H^{-1} g$

      Apply update: $\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

  end while

Batch normalization

- Reduces internal covariate shift
- Issues with deep neural network
  - Vanishing gradients
    - Use smaller learning rate
    - Use proper initialization
    - Use ReLU or MaxOut which does not saturate
- This approach provides inputs that has zero mean and unit variance to every layer of input in neural network

Batch normalization transformation

- Applying to activation x over a mini-batch

- Input — values of x over a minibatch $\mathcal{B} = \{x_{1\ldots m}\}$, parameters to be learned — γ, β

- Output — $\{y_i = \mathrm{BN}_{\gamma, \beta}(x_i)\}$

  - Minibatch mean: $\mu_{\mathcal{B}} = \dfrac{1}{m} \sum_{i=1}^{m} x_i$

  - Minibatch variance: $\sigma_{\mathcal{B}}^2 = \dfrac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$

  - Normalize: $\hat{x}_i = \dfrac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$

  - Scale and shift: $y_i = \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma, \beta}(x_i)$

Reference: Batch normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, S Ioffe, C Szegedy, 2015

Training & inference using batch-norm

- Input — Network N with trainable parameters $\boldsymbol{\theta}$, subset of activations $\{x^{(k)}\}_{k=1}^{K}$, Output — Batch-normalized network for inference $N_{\text{BN}}^{\text{inf}}$

- Steps:
  - Training BN network: $N_{\text{BN}}^{\text{tr}} = N$
  - **for** $k = 1, \ldots, K$
    - Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)}, \beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}} = N$
    - Modify each layer in $N_{\text{BN}}^{\text{tr}} = N$ with input $x^{(k)}$ to take $y^{(k)}$ instead
  - Train $N_{\text{BN}}^{\text{tr}}$ and optimize $\boldsymbol{\theta} \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
  - $N_{\text{BN}}^{\text{inf}} = N_{\text{BN}}^{\text{tr}}$
  - **for** $k = 1, \ldots, K$
    - Process multiple training minibatches and determine $\mathbb{E}[x] = \mathbb{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$ and $V[x] = \frac{m}{m-1}\mathbb{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$
    - In $N_{\text{BN}}^{\text{inf}}$ replace the transform $y = \text{BN}_{\gamma, \beta}(x)$ with $y = \frac{\gamma}{\sqrt{V[x]+\epsilon}}x + (\beta - \frac{\gamma\mathbb{E}[x]}{\sqrt{V[x]+\epsilon}})$

Exercise

- Consider the function $f(x, y) = \frac{1}{2}(x^2 + by^2)$ where $0 < b \leq 1$. We apply gradient descent with exact line search method. Here the step size (α) is computed as follows $\alpha = \arg\min_\alpha f(x - \alpha \nabla_x f(x))$. Let us assume that we start from $(x_0, y_0) = (b, 1)$. Find the value of (x_k, y_k). Can you find any interesting property of two consecutive gradients?