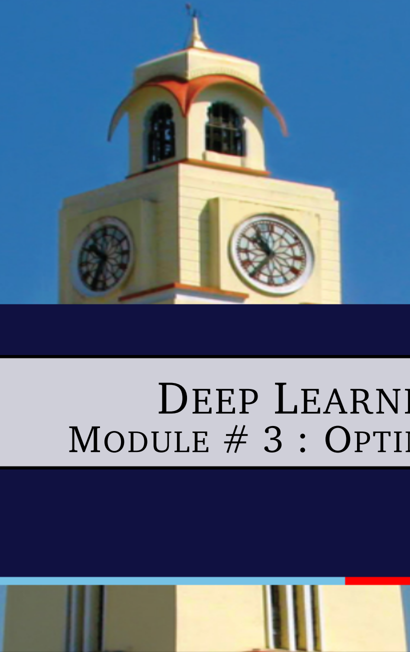


Deep Learning
Module # 3 : Optimization

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

DL Team, BITS Pilani

The instructor is gratefully acknowledging the authors who made their course materials freely available online.

What we learn ...

1. 3.1 Challenges in Neural Network Optimization - saddle points and plateau
2. 3.2 Non-convex optimization intuition
3. 3.3 Overview of optimization algorithms
4. 3.4 Momentum based algorithms
5. 3.5 Algorithms with Adaptive Learning Rates

Variance vs Bias

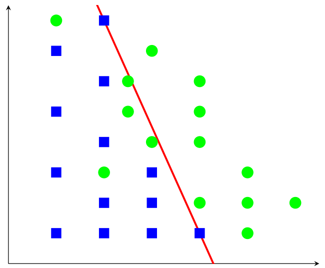


Figure: Underfitting; High Bias

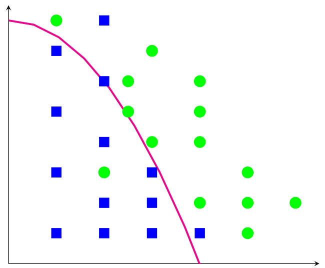


Figure: Low variance and bias

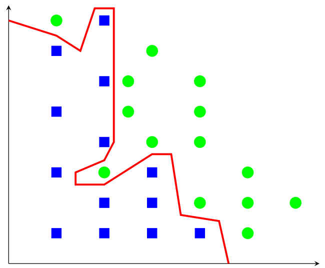
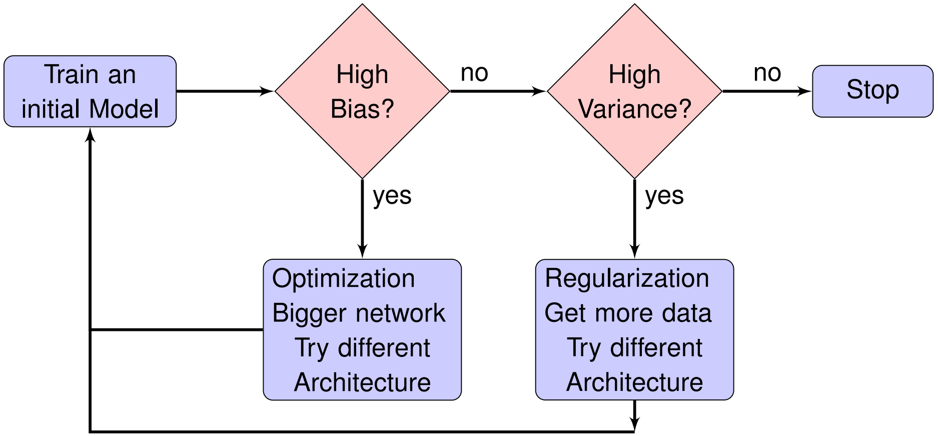


Figure: Overfitting; High variance

- Our aim is to achieve low variance and bias.

Deep Learning Recipe (Andrew Ng)



In This Segment

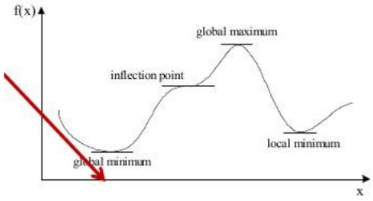
Optimization

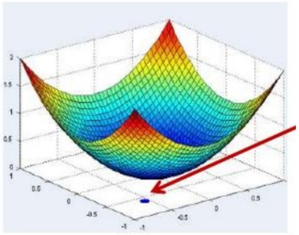
- Training of DNN is an iterative process.
- We may be training the DNN on a very large dataset or big data.
- The aim of DNN is improve the performance measure evaluated on the training set, i.e. **reduce the bias**.
- The performance measure P is optimized indirectly. Train the DNN to find the parameters θ that significantly reduce the cost function $J(\theta)$, so that P improves.

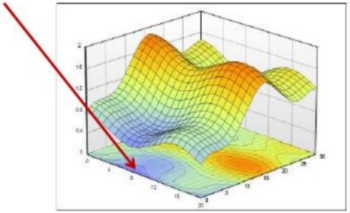
Optimization Algorithms

- Optimization algorithms train deep learning models.
- Optimization algorithms are the tools that allow
  - continue updating model parameters
  - to minimize the value of the loss function, as evaluated on the training set.
- In optimization, a loss function is often referred to as the **objective function** of the optimization problem.
- By tradition and convention most optimization algorithms are concerned with **minimization**.

Optimization







- General problem of optimization: find the value of x where $\mathbf{f}(x)$ is minimum

Why Optimization Algorithms?

- The **performance** of the optimization algorithm directly affects the model's training efficiency.
- Understanding the principles of different optimization algorithms and the role of their hyperparameters will enable us to **tune the hyperparameters in a targeted manner** to improve the performance of deep learning models.
- **The goal of optimization is to reduce the training error.** The goal of deep learning is to reduce the generalization error, this requires reduction in overfitting also.

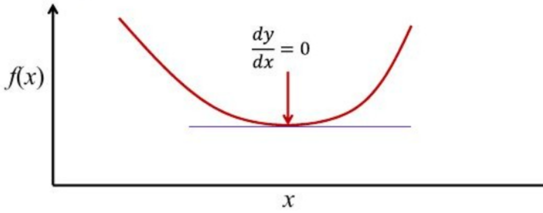
In This Segment

Challenges in Optimization

- Local minima
- Saddle points
- Vanishing gradients

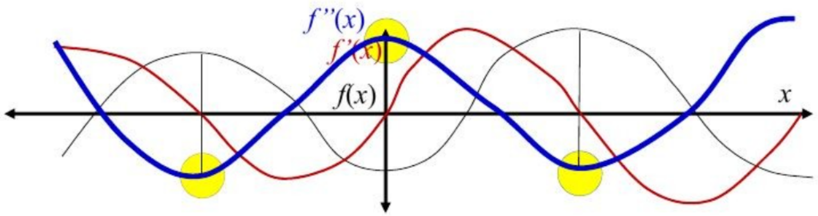
Finding Minimum of a Function

- Find the of x at which $f'(x) = 0$. i.e $\frac{df(x)}{dx} = 0$.
- Solution is at a turning point.
- A turning point is a value where the derivatives turn from positive to negative or vice versa.
- Turning points occur at minima, maxima or inflection (saddle) points.

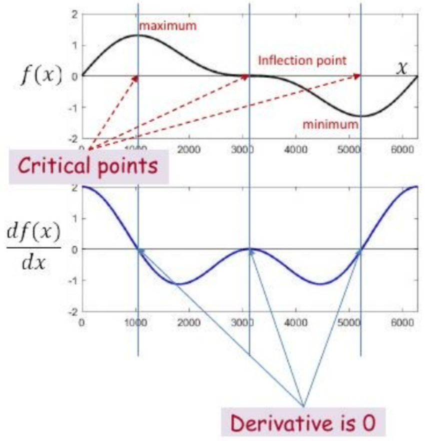


Derivatives of a Function

- Both maxima and minima are turning points.
- Both maxima and minima have zero derivatives.
- The second derivative $f''(x)$ is negative at maxima and positive at minima.

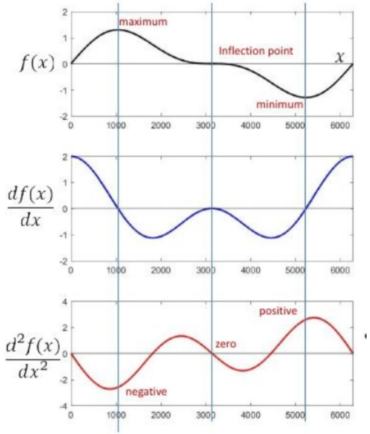


Derivatives of a Function



- All locations with zero derivative are **critical points.**
  - These can be local minima, local maxima or saddle points.

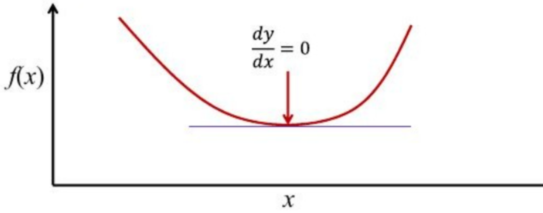
Derivatives of a Function



- All locations with zero derivative are **critical points.**
  - These can be local minima, local maxima or saddle points.
- The second derivative is
  - $\geq$ 0 at minima
  - $\leq$ 0 at maxima
  - $=$ 0 at saddle point
- It is more complicated for functions of multiple variables.

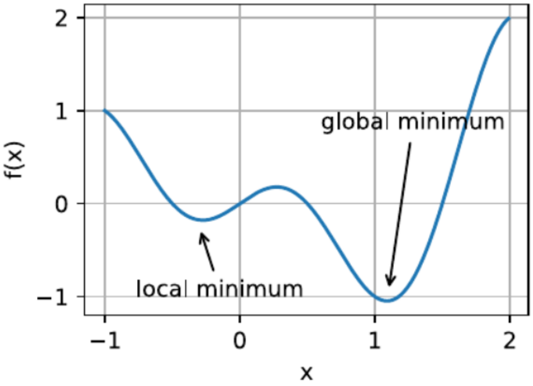
Finding Minimum of a Function

- Find the of x at which $f'(x) = 0$. i.e $\frac{df(x)}{dx} = 0$.
- Solution x_{soln} is at a turning point.
- Check the double derivative $f''(x_{soln})$ i.e $f''(x_{soln})\frac{df'(x)}{dx}$.
- If $f''(x_{soln})$ is positive, x_{soln} is minimum.
- If $f''(x_{soln})$ is negative, x_{soln} is maximum.
- If $f''(x_{soln})$ is zero, x_{soln} is saddle point.



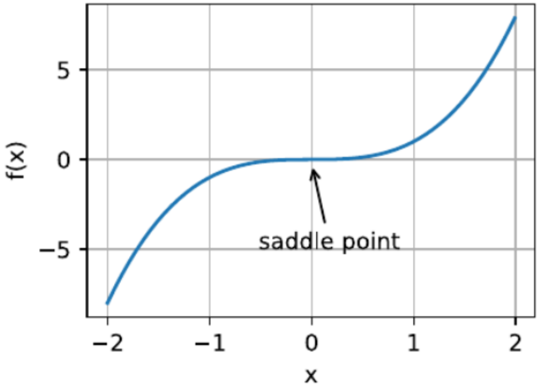
Local Minima

- For any objective function $f(x)$, if the value of $f(x)$ at x is smaller than the values of $f(x)$ at any other points in the vicinity of x, then $f(x)$ could be a local minimum.
- If the value of $f(x)$ at x is the minimum of the objective function over the entire domain, then $f(x)$ is the global minimum.
- In minibatch stochastic gradient descent, the natural variation of gradients over minibatches is able to dislodge the parameters from local minima.



Saddle Points

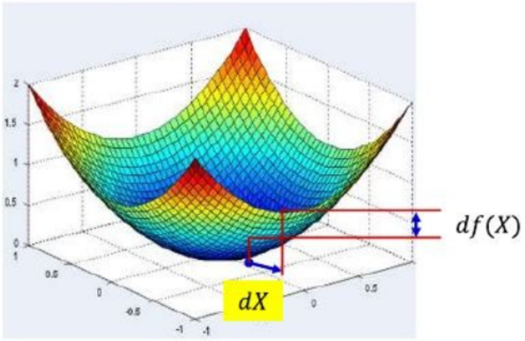
- In high dimensional space, the local points with gradients is closer to zero are known as saddle points.
- Saddle points tend to slow down the learning process.
- A saddle point is any location where all gradients of a function vanish but which is neither a global nor a local minimum.
  Eg: $f(x, y) = x^2 - y^2$
  saddle point at (0, 0); maximum wrt y and minimum wrt x



Gradient of a Scalar Function with Multiple Variables

- The gradient $\nabla f(X)$ of a mutli-variate input X is a multiplicative factor that gives us the change in $f(X)$ for tiny variations in X.

$$df(X) = \nabla f(X) d(X)$$



Gradient of a Scalar Function with Multiple Variables

- Consider $f(X) = f(x_1, x_2, \ldots, x_n))$
- The gradient is given by

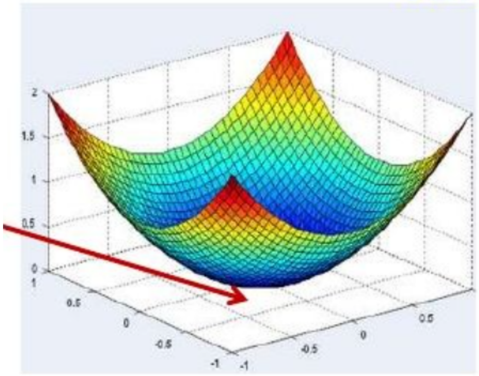
$$\nabla f(X) = \left[\frac{\partial f(X)}{\partial x_1}, \frac{\partial f(X)}{\partial x_2}, \ldots, \frac{\partial f(X)}{\partial x_n}\right]$$

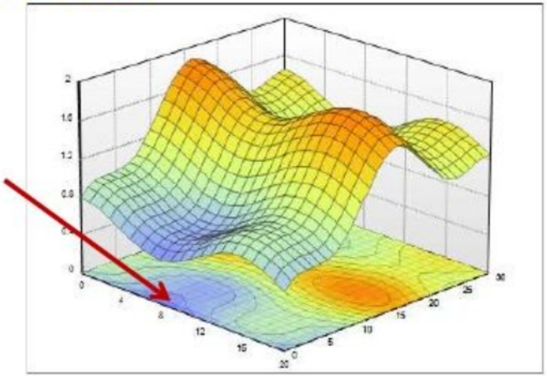
- The second derivative of the function is given by the Hessain.

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \ldots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \ldots & \frac{\partial^2 f}{\partial x_2 x_n} \\ \ldots & & & \\ \frac{\partial^2 f}{\partial x_n x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \ldots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

Gradient of a Scalar Function with Multiple Variables

- The optimum point is the turning point. The gradient is zero.





Solution of Unconstrained Minimization

- Solve for X where the gradient equation equal to zero.

$$\nabla f(X) = 0$$

- Compute the Hessian matrix $\nabla^2 f(X)$ at the candidate solution and verify that
- Local Minimum
  - Eigenvalues of Hessian matrix are all positive.
- Local Maximum
  - Eigenvalues of Hessian matrix are all negative.
- Saddle Point
  - Eigenvalues of Hessian matrix at the zero-gradient position are negative and positive.

Example

- Minimize

$$f(x_1, x_2, x_3) = x_1^2 + x_1(1 - x_2) + x_2^2 - x_2 x_3 + x_3^2 + x_3$$

- Gradient

$$\nabla f(X) = \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix}$$

Example

- Set Gradient = 0

$$\nabla f(X) = \begin{bmatrix} 2x_1 + 1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 + 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

- Solve the 3 equations in 3 unknowns.

$$X = (x_1, x_2, x_3) = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

Example

- Compute the Hessian

$$\nabla^2 f(X) = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

- Evaluate the eigenvalues of the Hessain matrix

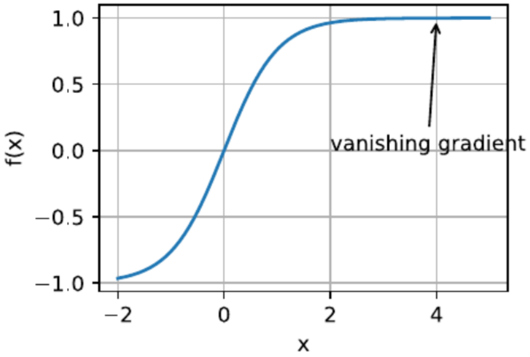
$$\lambda_1 = 3.414 \quad \lambda_2 = 0.586 \quad \lambda_3 = 2$$

- All Eigen vectors are positive.
- The obtained solution is minimum.

$$X = (x_1, x_2, x_3) = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

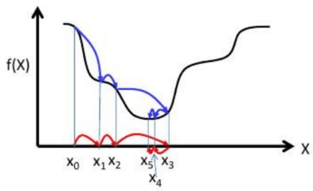
Vanishing Gradients

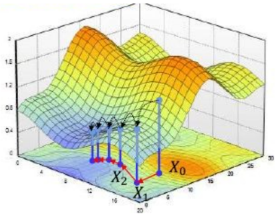
- Function $f(x) = tanh(x)$
- $f'(x) = 1 - tanh2(x)$
- $f'(4) = 0.0013$.
- Gradient of f is close to nil.
- Vanishing gradients can cause optimization to stall.
- Reparameterization of the problem helps.
- Good initialization of the parameters.



In This Segment

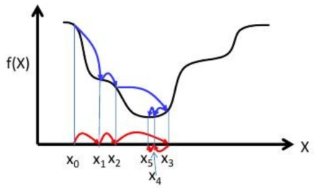
How to find Global Minima?

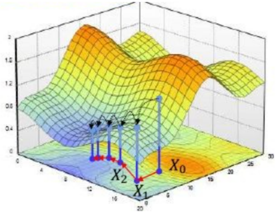




- It is not possible to solve $\nabla f(X) = 0$.
  - The function to minimize / maximize may have a intractable form
- In these situation, iterative solutions are used.
  - Begin with a guess for the optimal X and refine it iteratively until the correct value is obtained.

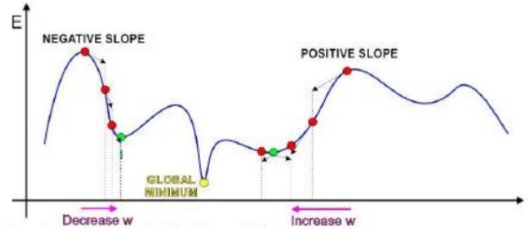
Find Global Minima Iteratively





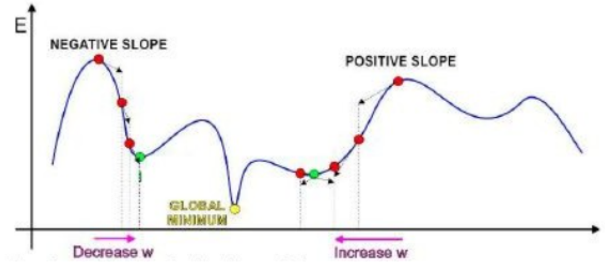
- Iterative solutions
  - Start with an initial guess X_0.
  - Update the guess towards a better value of $f(X)$.
  - Stop when $F(X)$ no longer decreases.
- Challenges
  - Which direction to step in?
  - How big the steps should be?

Gradient Descent



- Iterative Solution
  - Start at some initial guess.
  - Find the direction to shift this point to decrease error.
    - This can be found from the derivative of the function.
    - A positive derivative implies, move left to decrease error.
    - A negative derivative implies, move right to decrease error.
  - Shift the point in this direction.

Gradient Descent



Algorithm 1: Gradient Descent Algorithm

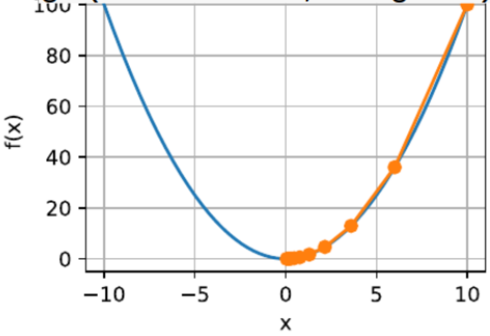
```
1 Initialize $x_0$
2 while $f'(x^k) \neq 0$ do
3     $x^{k+1} = x^k - step\_size f'(x^k)$
```

Gradient Descent

- First order Gradient Descent algos consider the first order derivatives to get the magnitude and direction of update.

```
def gd(eta, f_grad):
 x = 10.0
 results = [x]
 for i in range(10):
   x -= eta * f_grad(x)
   results.append(float(x))
 return results
```

```
gd(eta = 0.2, f_grad)
```



Numerical Example

Consider an error function

$$E(w_1, w_2) = 0.05 + \frac{(w_1 - 3)^2}{4} + \frac{(w_2 - 4)^2}{9} - \frac{(w_1 - 3)(w_2 - 4)}{6}$$

Different variants of gradient descent algorithm can be used to minimize this error function w.r.t (w_1, w_2). Assume $(w_1, w_2) = (1, 1)$ at time $(t - 1)$ and after update $(w_1, w_2) = (1.5, 2.0)$ at time (t). Assume $\alpha = 1.5, \beta = 0.6, \eta = 0.3$.

Numerical Example

Compute the value that minimizes (w_1, w_2). Compute the minimum possible value of error.

$$E(w_1, w_2) = 0.05 + \frac{(w_1 - 3)^2}{4} + \frac{(w_2 - 4)^2}{9} - \frac{(w_1 - 3)(w_2 - 4)}{6}$$

$$\frac{\partial E}{\partial w_1} = \frac{2(w_1 - 3)}{4} - \frac{(w_2 - 4)}{6} = 0$$

$$\frac{\partial E}{\partial w_2} = \frac{2(w_2 - 4)}{9} - \frac{(w_1 - 3)}{6} = 0$$

Solving $w_1 = 3$

$w_2 = 4$

Substituting $E = 0.05$

Numerical Example

Compute the value of (w_1, w_2) at time $(t+1)$ if standard gradient descent is used.

$$E(w_1, w_2) = 0.05 + \frac{(w_1-3)^2}{4} + \frac{(w_2-4)^2}{9} - \frac{(w_1-3)(w_2-4)}{6}$$

$$\frac{\partial E}{\partial w_1} = \frac{2(w_1-3)}{4} - \frac{(w_2-4)}{6}$$

$$\frac{\partial E}{\partial w_2} = \frac{2(w_2-4)}{9} - \frac{(w_1-3)}{6}$$

$$at \quad (t+1) \qquad w_{1(t+1)} = w_{1(t)} - \eta\frac{\partial E}{\partial w_1} = 1.5 - 0.3 * \left[\frac{(1.5-3)}{2} - \frac{(2-4)}{6}\right] = 1.625$$

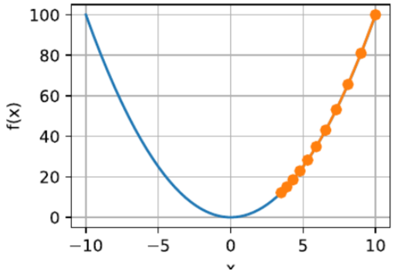
$$w_{2(t+1)} = w_{2(t)} - \eta\frac{\partial E}{\partial w_2} = 2 - 0.3 * \left[\frac{2(2-4)}{9} - \frac{(1.5-3)}{6}\right] = 2.05$$

Learning Rate

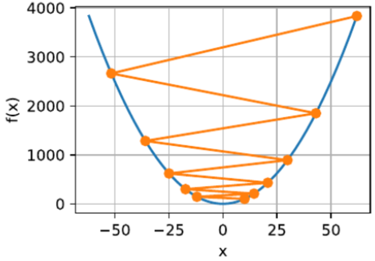
- The role of the learning rate is to moderate the degree to which weights are changed at each step.
- Learning rate η is set by the algorithm designer.
- If the learning rate that is too small, it will cause parameters to update very slowly, requiring more iterations to get a better solution.
- If the learning rate that is too large, the solution oscillates and in the worst case it might even diverge.

Gradient Descent

```
gd(eta = 0.05, f_grad)
```



```
gd(eta = 1.1, f_grad)
```



- Small learning rate
  - Slow Learning
  - Definitely converge
- Large learning rate
  - Oscillations
  - Worst case it might diverge

Example

Error surface is given by $E(x,y,z) = 3x^2 + 2y^2 + 4z^2 + 6$. Assume gradient descent is used to find the minimum of this error surface. What is the optimal learning rate that leads to fastest convergence to the global minimum?

$$\begin{aligned}
E(x,y,z) &= 3x^2 + 2y^2 + 4z^2 + 6 \\
\eta_{xopt} &= 1/6 \\
\eta_{yopt} &= 1/4 \\
\eta_{zopt} &= 1/8 \\
\text{optimal learning rate for convergence} &= \min[\eta_{xopt}, \eta_{yopt}, \eta_{zopt}] = 1/8 \\
\text{Largest learning rate for convergence} &= \min[2\eta_{xopt}, 2\eta_{yopt}, 2\eta_{zopt}] = 0.33 \\
\text{Learning rate for divergence} &> 2\eta_{opt} = 2 * 1/8 = 0.25
\end{aligned}$$

Stochastic Gradient Descent

- In deep learning, the objective function is the average of the loss functions for each example in the training dataset.
- Given a training dataset of m examples, let $f_i(\theta)$ is the loss function with respect to the training example of index i, where θ is the parameter vector.
- Computational cost of each iteration is O(1).

Stochastic Gradient Descent Algorithm

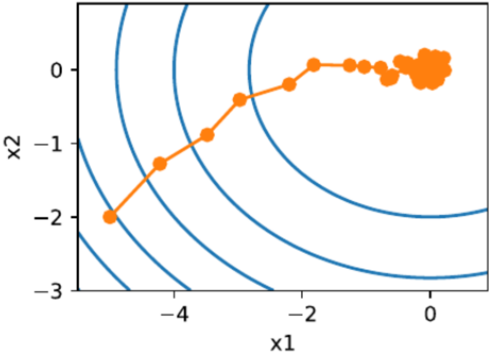
Algorithm 2: Stochastic Gradient Descent Algorithm

```
Data: Learning Rate $\eta$
Data: Initial Parameters $\theta$
1 while stopping criterion not met do
2     for each example i do
3         Perform Forward prop on $X^{\{i\}}$ to compute $\hat{y}$ and cost $J(\theta)$
4         Perform Backprop on $\left(X^{\{i\}}, Y^{\{i\}}\right)$ to compute gradient $\mathbf{g}$
5         Update parameters as $\theta \leftarrow \theta - \eta\mathbf{g}$
```

Stochastic Gradient Descent

- Trajectory of the variables in the stochastic gradient descent is much more noisy. This is due to the stochastic nature of the gradient. Even near the minimum, uncertainty is injected by the instantaneous gradients.

```
def sgd(x1, x2, s1, s2, f_grad):
 g1, g2 = f_grad(x1, x2)
 # Simulate noisy gradient
 g1 += np.random.normal(0.0, 1, (1,))
 g2 += np.random.normal(0.0, 1, (1,))
 eta_t = eta * lr()
 return (x1-eta_t*g1, x2-eta_t*g2, 0, 0)
```



Dynamic Learning Rate

- Replace η with a time-dependent learning rate $\eta(t)$ adds to the complexity of controlling convergence of an optimization algorithm.
- A few basic strategies that adjust η over time.
  - Piecewise constant
    - decrease the learning rate, e.g., whenever progress in optimization stalls.
    - This is a common strategy for training deep networks

$$\eta(t) = \eta_i \qquad \text{if} \quad t_i \leq t \leq t_{i+1}$$

  - Exponential decay
    - Leads to premature stopping before the algorithm has converged.

$$\eta(t) = \eta_0 e^{-\lambda t}$$

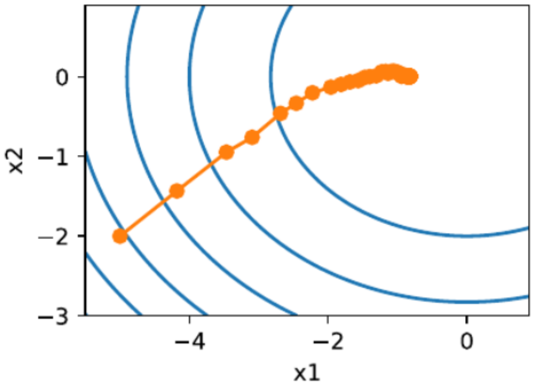
  - Polynomial decay

$$\eta(t) = \eta_0(\beta t + 1)^{-\alpha}$$

Exponential Decay

- Variance in the parameters is significantly reduced.
- The algorithm fails to converge at all.

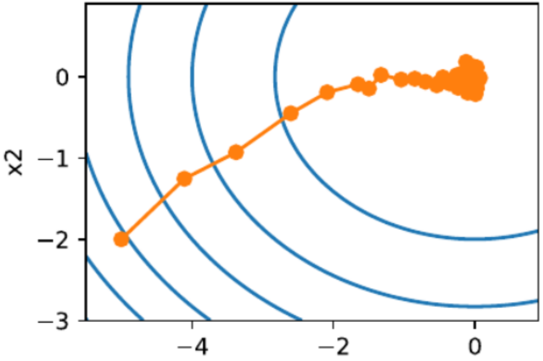
```
def exponential_lr():
 global t
 t += 1
 return math.exp(-0.1 * t)
```



Polynomial Decay

- Learning rate decays with the inverse square root of the number of steps.
- Convergence gets better after only 50 steps.

```
def polynomial_lr():
    global t
    t += 1
    return (1 + 0.1 * t) ** (-0.5)
```



Gap

- Gradient descent
  - Uses the full dataset to compute gradients and to update parameters, one pass at a time.
  - Gradient Descent is not particularly data efficient whenever data is very similar.
- Stochastic Gradient descent
  - Processes one observation at a time to make progress.
  - Stochastic Gradient Descent is not particularly computationally efficient since CPUs and GPUs cannot exploit the full power of vectorization.
  - For noisy gradients, choice of the learning rate is critical.
  - If we decrease it too rapidly, convergence stalls.
  - If we are too lenient, we fail to converge to a good enough solution since noise keeps on driving us away from optimality.
- Minibatch SGD
  - Accelerate computation, or better or computational efficiency.
  - A [illegible] dients reduce the amount of variance.

Minibatch Stochastic Gradient Descent

- Suppose we have 4 million examples in the training set.

$$X = [X^{(1)}, X^{(2)}, \ldots, X^{(m)}]$$
$$Y = [Y^{(1)}, Y^{(2)}, \ldots, Y^{(m)}]$$

- The gradient descent algorithm process the entire dataset of $m = 4$ million examples before proceeding to the next step. This is cumbersome and time consuming.
- Hence split the training dataset.

Minibatch Stochastic Gradient Descent

- Let the mini-batch size be 1000.
- Then we have 4000 mini-batches, each having 1000 examples.

$$
\begin{aligned}
X &= [X^{(1)}, X^{(2)}, \ldots, X^{(m)}] \\
&= [X^{(1)}, X^{(2)}, \ldots, X^{(1000)}, X^{(1001)}, \ldots, X^{(2000)}, \ldots, X^{(m)}] \\
&= X^{\{1\}}, X^{\{2\}}, \ldots, X^{\{m/b\}} \\
Y &= [Y^{(1)}, Y^{(2)}, \ldots, Y^{(m)}] \\
&= [Y^{(1)}, Y^{(2)}, \ldots, Y^{(1000)}, Y^{(1001)}, \ldots, Y^{(2000)}, \ldots, Y^{(m)}] \\
&= Y^{\{1\}}, Y^{\{2\}}, \ldots, Y^{\{m/b\}}
\end{aligned}
$$

- The minibatch t is denoted as $X^{\{t\}}, Y^{\{t\}}$
- The Gradient descent algorithm will be repeated for t batches.

Minibatch Stochastic Gradient Descent Algorithm

- In each iteration, we first randomly sample a minibatch B consisting of a fixed number of training examples.
- We then compute the derivative (gradient) of the average loss on the minibatch with regard to the model parameters.
- Finally, we multiply the gradient by a predetermined positive value η and subtract the resulting term from the current parameter values.

$$\theta \leftarrow \frac{\eta}{|B|} \sum_{i \in B} \partial_\theta loss^{(t)}(\theta)$$

- $|B|$ represents the number of examples in each minibatch (the batch size) and η denotes the learning rate.

Minibatch Stochastic Gradient Descent Algorithm

- Gradients at time *t* is calculated as

$$g_{t,t-1} = \partial_{theta} \frac{1}{|B|} \sum_{i \in B_t} f(x_i, \theta_{t-1}) = \frac{1}{|B|} \sum_{i \in B_t} h_{i,t-1}$$

- $|B|$ represents the number of examples in each minibatch (the batch size) and *eta* denotes the learning rate.
- $h_{i,t-1} = \partial_\theta f(x_i, \theta_{t-1})$ is the stochastic gradient descent for sample *i* using the weights updated at time $t-1$.

Minibatch Stochastic Gradient Descent Algorithm

Algorithm 3: Minibatch Stochastic Gradient Descent Algorithm

```
Data: Learning Rate $\eta$
Data: Initial Parameters $\theta$
1 while stopping criterion not met do
2     for t in range $(1, m/batch\_size)$ do
3         Perform Forward prop on $X^{\{t\}}$ to compute $\hat{y}$ and cost $J(\theta)$
4         Perform Backprop on $\left(X^{\{t\}}, Y^{\{t\}}\right)$ to compute gradient $\mathbf{g}$
5         Update parameters as $\theta \leftarrow \theta - \eta\mathbf{g}$
```

Minibatch Size

- Typical mini batch size is taken as some power of 2; say 32, 64, 128, 256.
- Better utilization of multicore architecture.
- Amount of memory scales with batch size.

Leaky Average in Minibatch SGD

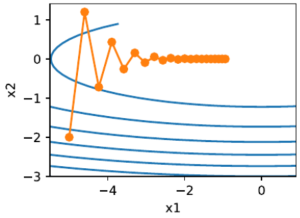
- Replace the gradient computation by a ”leaky average” for better variance reduction.

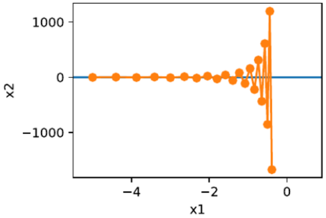
$$\beta \in (0, 1)$$

- This effectively replaces the instantaneous gradient by one that’s been averaged over multiple past gradients.
- v is called momentum.
- **Momentum accumulates past gradients.**
- Large β amounts to a long-range average and small β amounts to only a slight correction relative to a gradient method.
- The new gradient replacement no longer points into the direction of steepest descent on a particular instance any longer but rather in the direction of a weighted average of past gradients.

Momentum Method Example

- Consider a moderately distorted ellipsoid objective $f(x) = 0.1x_1^2 + 2x_2^2$
- f has its minimum at $(0, 0)$. This function is very flat in x_1 direction.



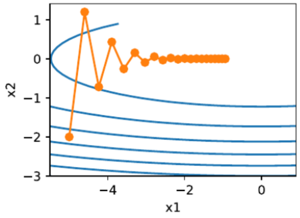


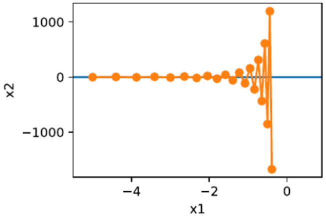
- For $\eta = 0.4$ Without momentum
- The gradient in the x_2 direction oscillates than in the horizontal x_1 direction.

- For $\eta = 0.6$ Without momentum
- Convergence in the x_1 direction improves but the overall solution quality is diverging.

Momentum Method Example

- Consider a moderately distorted ellipsoid objective $f(x) = 0.1x_1^2 + 2x_2^2$
- Apply momentum for $\eta = 0.6$





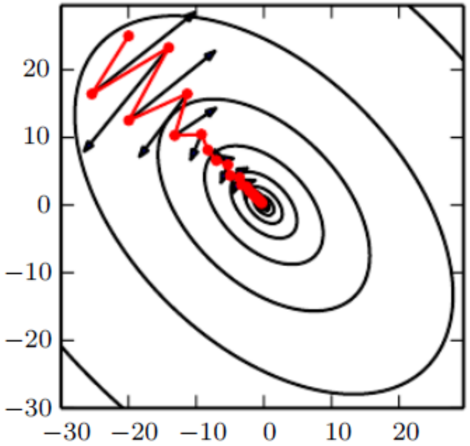
- For $\beta = 0.5$
- Converges well. Lesser oscillations. Larger steps in x_1 direction.

- For $\beta = 0.25$
- Reduced convergence. More oscillations. Larger magnitude of oscillations.

Momentum

- Compute an exponentially weighted average of the gradients and use it to update the parameters.
- The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction.
- The momentum introduces a velocity v that denotes the direction and speed at which the parameters change.
- The velocity is set to an exponentially decaying average of the negative gradient.
- The Momentum parameter $\beta \in [0, 1)$ is a hyperparameter and typically takes a value of 0.9.

SGD with Momentum



SGD with Momentum Algorithm

Algorithm 4: Stochastic Gradient Descent with Momentum

```
Data: Learning Rate $\eta$, Momentum Parameter $\beta$
Data: Initial Parameters $\theta$ , Inital velocities $v$
1 while stopping criterion not met do
2     for t in range $(1, m/batch\_size)$ do
3         Perform Forward prop on $X^{\{t\}}$ to compute $\hat{y}$ and cost $J(\theta)$
4         Perform Backprop on $\left(X^{\{t\}}, Y^{\{t\}}\right)$ to compute gradient $\mathbf{g}$
5         Update velocities as $v = \beta v + (1 - \beta)\mathbf{g}$
6         Update parameters as $\theta \leftarrow \theta - \eta v$
```

Momentum method: Summary

- Momentum replaces gradients with a leaky average over past gradients. This accelerates convergence significantly.
- Momentum prevents stalling of the optimization process that is much more likely to occur for stochastic gradient descent.
- The effective number of gradients is given by $1/(1-\beta)$ due to exponentiated down weighting of past data.
- Implementation is quite straightforward but it requires us to store an additional state vector (momentum v).

Numerical Example

Consider an error function

$$E(w_1, w_2) = 0.05 + \frac{(w_1 - 3)^2}{4} + \frac{(w_2 - 4)^2}{9} - \frac{(w_1 - 3)(w_2 - 4)}{6}$$

Different variants of gradient descent algorithm can be used to minimize this error function w.r.t (w_1, w_2). Assume $(w_1, w_2) = (1, 1)$ at time $(t - 1)$ and after update $(w_1, w_2) = (1.5, 2.0)$ at time (t). Assume $\alpha = 1.5, \beta = 0.6, \eta = 0.3$. Compute the value of (w_1, w_2) at time $(t + 1)$ if momentum is used.

$$\frac{\partial E}{\partial w_1} = \frac{2(w_1 - 3)}{4} - \frac{(w_2 - 4)}{6}$$
$$\frac{\partial E}{\partial w_2} = \frac{2(w_2 - 4)}{9} - \frac{(w_1 - 3)}{6}$$

Numerical Example

$$at \quad (t+1) \qquad w_{1(t+1)} = w_{1(t)} - \eta \frac{\partial E}{\partial w_1} + \beta \Delta w_1$$

$$= 1.5 - 0.3 * \left[\frac{(1.5-3)}{2} - \frac{(2-4)}{6} \right] + 0.9 * (1.5-1)$$

$$= 2.075$$

$$w_{2(t+1)} = w_{2(t)} - \eta \frac{\partial E}{\partial w_2} + \beta \Delta w_1$$

$$= 2 - 0.3 * \left[\frac{2(2-4)}{9} - \frac{(1.5-3)}{6} \right] + 0.9 * (2-1)$$

$$= 2.958$$

Adagrad

- Used for features that occur infrequently (sparse features).
- Adagrad uses aggregate of the squares of previously observed gradients.

Adagrad Algorithm

- Variable s_t to accumulate past gradient variance.
- Operation are applied coordinate wise. $\sqrt{1/v}$ has entries $\sqrt{1/v_1}$ and $u \cdot v$ has entries $u_i v_i$.
- η is the learning rate and ϵ is an additive constant that ensures that we do not divide by 0.
- Initialize $s_0 = 0$.

$$g_t = \partial_w loss(y_t, f(x_t, w))$$
$$s_t = s_{t-1} g_t^2$$
$$w_t = w_t - \frac{\eta}{\sqrt{s_t + \epsilon}} \cdot g_t$$

Adagrad: Summary

- Adagrad decreases the learning rate dynamically on a per-coordinate basis.
- It uses the magnitude of the gradient as a means of adjusting how quickly progress is achieved - coordinates with large gradients are compensated with a smaller learning rate.
- If the optimization problem has a rather uneven structure Adagrad can help mitigate the distortion.
- Adagrad is particularly effective for sparse features where the learning rate needs to decrease more slowly for infrequently occurring terms.
- On deep learning problems Adagrad can sometimes be too aggressive in reducing learning rates.

RMSProp (Root Mean Square Prop)

- Adapts the learning rates of all model parameters by scaling the gradients into an exponentially weighted moving average.
- Performs better with non-convex setting.
- Adagrad use learning rate that decreases at a predefined schedule of effectively $O(t^{-1/2})$.
- RMSProp algorithm decouples rate scheduling from coordinate-adaptive learning rates. This is essential for non-convex optimization.

RMSProp Algorithm

- Use leaky average to accumulate past gradient variance.
- Parameter $\gamma > 0$.
- The constant ϵ is set to 10^{-6} to ensure that we do not suffer from division by zero or overly large step sizes.

$$s_t = \gamma s_{t-1} + (1 - \gamma) g_t^2$$
$$w_t = w_t - \frac{\eta}{\sqrt{s_t + \epsilon}} \odot g_t$$

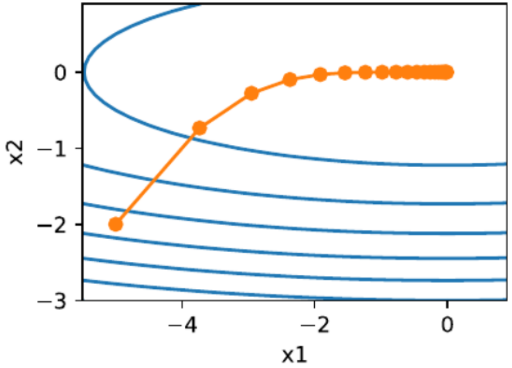
RMSProp Algorithm

Algorithm 5: RMSProp

```
Data: Learning Rate $\eta$, Decay Rate $\gamma$
Data: Small constant $\epsilon = 10^{-7}$ to stabilize division
Data: Initial Parameters $\theta$
1 Initialize accumulated gradient $s = 0$
2 while stopping criterion not met do
3     for $t$ in range $(1, m/batch\_size)$ do
4         Perform Forward prop on $X^{\{t\}}$ to compute $\hat{y}$ and cost $J(\theta)$
5         Perform Backprop on $\left(X^{\{t\}}, Y^{\{t\}}\right)$ to compute gradient $\mathbf{g}$
6         Compute accumulated squared gradient $s = \gamma s + (1 - \gamma)\mathbf{g}^2$
7         Update parameters as $\theta \leftarrow \theta - \frac{\eta}{\sqrt{s+\epsilon}} \odot \mathbf{g}$
```

RMSProp Example

```
def rmsprop_2d(x1, x2, s1, s2):
 g1, g2, eps = 0.2*x1, 4*x2, 1e-6
 s1 = gamma*s1 +(1-gamma)*g1**2
 s2 = gamma*s2 +(1-gamma)*g2**2
 x1 -= eta/math.sqrt(s1 + eps)*g1
 x2 -= eta/math.sqrt(s2 + eps)*g2
 return x1, x2, s1, s2
```



RMSProp: Summary

- RMSProp is very similar to Adagrad as both use the square of the gradient to scale coefficients.
- RMSProp shares with momentum the leaky averaging. However, RMSProp uses the technique to adjust the coefficient-wise preconditioner.
- The learning rate needs to be scheduled by the experimenter in practice.
- The coefficient γ determines how long the history is when adjusting the per-coordinate scale.

Review of algorithms learned so far

1. Stochastic gradient descent
   - more effective than Gradient Descent when solving optimization problems, e.g., due to its inherent resilience to redundant data.
2. Minibatch Stochastic gradient descent
   - affords significant additional efficiency arising from vectorization, using larger sets of observations in one minibatch. This is the key to efficient multi-machine, multi-GPU and overall parallel processing.
3. Momentum
   - added a mechanism for aggregating a history of past gradients to accelerate convergence.

Review of algorithms learned so far

1. Adagrad
   - used per-coordinate scaling to allow for a computationally efficient preconditioner.
2. RMSProp
   - decoupled per-coordinate scaling from a learning rate adjustment.

Adam (Adaptive Moments)

- Adam combines all these techniques into one efficient learning algorithm.
- Very effective and robust algorithm
- Adam can diverge due to poor variance control. (disadvantage)
- Adam uses exponential weighted moving averages (also known as leaky averaging) to obtain an estimate of both the momentum and also the second moment of the gradient.

Adam Algorithm

- State variables

$$v_t \leftarrow \beta_1 v_{t-1} + (1 - \beta_1) g_t$$
$$s_t \leftarrow \beta_2 s_{t-1} + (1 - \beta_2) g_t^2$$

- β_1 and β_2 are nonnegative weighting parameters. Common choices for them are $\beta_1 = 0.9$ and $\beta_2 = 0.999$. That is, the variance estimate moves much more slowly than the momentum term.
- Initialize $v_0 = s_0 = 0$.

Adam Algorithm

- Normalize the state variables

$$\hat{v}_t = \frac{v_t}{1-\beta_1^t}$$
$$\hat{s}_t = \frac{s_t}{1-\beta_2^t}$$

- Rescale the gradient

$$\hat{g}_t = \frac{\eta \hat{v}_t}{\sqrt{\hat{s}_t}+\epsilon}$$

- Compute updates

$$w_t \leftarrow w_{t-1} - \frac{\eta \hat{v}}{\sqrt{\hat{s}+\epsilon}}$$

Adam Algorithm

Algorithm 6: Adam

```
Data: Learning Rate $\eta = 0.001$, Momentum parameter $\beta_1 = 0.9$, Decay Rate
      $\beta_2 = 0.999$, Small constant $\epsilon = 10^{-7}$ to stabilize division
Data: Initial Parameters $\theta$; Initialize first and second moments $v = 0, s = 0$
1 while stopping criterion not met do
2     for t in range $(1, m/batch\_size)$ do
3         Perform Forward prop on $X^{\{t\}}$ to compute $\hat{y}$ and cost $J(\theta)$
4         Perform Backprop on $\left(X^{\{t\}}, Y^{\{t\}}\right)$ to compute gradient $\mathbf{g}$
5         Update first moment as $v \leftarrow \beta_1 v + (1 - \beta_1)\mathbf{g}$
6         Update second moment as $s = \beta_2 s + (1 - \beta_2)\mathbf{g}^2$
7         Perform bias correction as $\hat{v} = \frac{v}{1-\beta_1^t}$ and $\hat{s} = \frac{s}{1-\beta_2^t}$
8         Update parameters as $\theta \leftarrow \theta - \frac{\eta \hat{v}}{\sqrt{\hat{s}+\epsilon}}$
```

Adam: Summary

- Adam combines features of many optimization algorithms into a fairly robust update rule.
- Adam uses bias correction to adjust for a slow startup when estimating momentum and a second moment.
- For gradients with significant variance we may encounter issues with convergence. They can be amended by using larger minibatches or by switching to an improved estimate for state variables.
- Yogi algorithm offers such an alternative.

Numerical Example

The training data $(x_1, y_1) = (3.5, 0.5)$ for a single Sigmoid neuron and initial values of $w = -2.0, b = -2.0, \eta = 0.10, \beta_1 = 0.90, \beta_2 = 0.99, \epsilon = 1e-8, s_W = 0$ and $s_B = 0$ are provided. Showing all the calculations, find out the values of w, b, s_W, s_B, r_W, r_B after one iteration of Adam. Use BCE as loss function. Apply bias-correction.

Numerical Example

- Forward Pass and Calculate Loss

$$\hat{y} = \sigma(wx + b) = \sigma(-2.0 \times 3.5 - 2.0) \approx 0.0180$$
$$L = -[y \times \log(\hat{y}) + (1 - y) \times \log(1 - \hat{y})$$
$$L = -[0.5 \times \log(0.0180) + (1 - 0.5) \times \log(1 - 0.0180)] \approx 4.0076$$

- Calculate Gradients

$$\nabla_w L = (\hat{y} - y) \times x = (0.0180 - 0.5) \times 3.5 \approx -0.493$$
$$\nabla_b L = \hat{y} - y = 0.0180 - 0.5 \approx -0.482$$

- Update First Moments

$$s_W = \beta_1 \times s_W + (1 - \beta_1) \times \nabla_w L = 0.90 \times 0 - 0.10 \times (-0.493) \approx 0.04437$$
$$s_B = \beta_1 \times s_B + (1 - \beta_1) \times \nabla_b L = 0.90 \times 0 - 0.10 \times (-0.482) \approx 0.04382$$

Numerical Example

- Update Second Moments

$$r_W = \beta_2 \times r_W + (1-\beta_2) \times (\nabla_w L)^2$$
$$= 0.999 \times 0 - 0.001 \times (-0.493)^2 \approx 0.0001216$$
$$r_B = \beta_2 \times r_B + (1-\beta_2) \times (\nabla_b L)^2$$
$$= 0.999 \times 0 - 0.001 \times (-0.482)^2 \approx 0.0001168$$

- Corrected First Moments

$$\hat{s}_W = \frac{s_W}{1-\beta_1^1} \approx \frac{0.04437}{1-0.90^1} \approx 0.4437$$
$$\hat{s}_B = \frac{s_B}{1-\beta_1^1} \approx \frac{0.04382}{1-0.90^1} \approx 0.4382$$

Numerical Example

- Corrected Second Moments

$$\hat{r}_W = \frac{r_W}{1-\beta_2^1} \approx \frac{0.0001216}{1-0.999^1} \approx 0.0608$$
$$\hat{r}_B = \frac{r_B}{1-\beta_2^1} \approx \frac{0.0001168}{1-0.999^1} \approx 0.0584$$

- Update Weights and Biases

$$w = w - \frac{\eta}{\sqrt{\hat{r}_W} + \epsilon} \times \hat{s}_W = -2.0 - \frac{0.10}{\sqrt{0.0608} + 1e-8} \times 0.4437 \approx -2.081$$
$$b = b - \frac{\eta}{\sqrt{\hat{r}_B} + \epsilon} \times \hat{s}_B = -2.0 - \frac{0.10}{\sqrt{0.0584} + 1e-8} \times 0.4382 \approx -2.079$$

References

1. Dive into Deep Learning by Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola
   `https://d2l.ai/chapter_introduction/index.html`
2. Deep Learning by Ian Goodfellow, YoshuaBengio, Aaron Courville
   `https://www.deeplearningbook.org/`

Thank You!