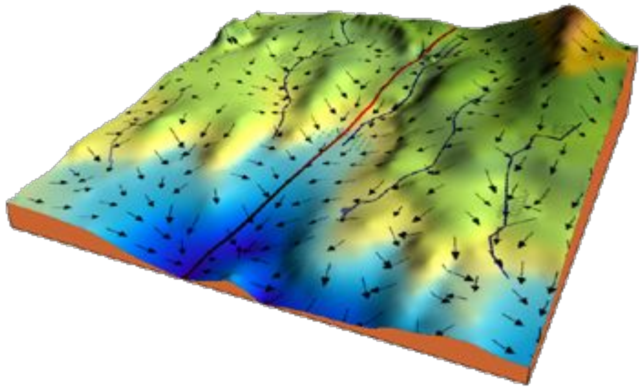
Recitation 2 – 9/9/2022 Gradient Descent and Optimization

TA: KAZEM MEIDANI

Gradient Descent - Basics

- Optimization algorithm used to minimize a function by iteratively moving in the direction of the steepest **descent**, found by taking the negative of the **gradient**.
- Used in ML to update parameters of a model.
    - Coefficients in linear regression.
    - Weights in neural networks.



Gradient Descent - Tunable Components

Learning Rate controls size of the steps taken during gradient descent.

- Can be adjusted during the process.
- Affects how much the value of the parameters are changed at each step.

Learning Rate Size	Changes to Parameters	Pros	Cons
Large	Large	Covers more ground (less iterations to convergence)	Risk overshooting the minimum
Small	Small	More precise	Covers less ground (more iterations to convergence)

Gradient Descent - Tunable Components

Cost Function (Loss Function) tells us how good our model is at making predictions for a given set of parameters.

- Minimizing the cost function minimizes the error between our model and the given data points.
- Slope of curve tells us how to update our parameters to make them more accurate.
- One commonly used cost function is Mean Squared Error.

$$MSE = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

i: *index of sample*
m: *total number of samples in dataset*
$\hat{y}$: *predicted value*
y: *expected (actual) value*

Cost Functions - Regression Losses

Mean Absolute Error (L1 Loss): average of sum of absolute error between predictions and actual observations.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|\hat{y}^{(i)} - y^{(i)}\right|$$

i: *index of sample*
n: *total number of samples in dataset*
$\hat{y}$: *predicted value*
y: *expected (actual) value*

Mean Squared Error (L2 Loss): average squared error between predictions and actual observations.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)})^2$$

i: *index of sample*
n: *total number of samples in dataset*
$\hat{y}$: *predicted value*
y: *expected (actual) value*

https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23

Cost Functions - Classification Losses

Cross Entropy Loss / Negative Log Likelihood: measures the difference between two probability distributions.

$$CrossEntropyLoss = -(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$$

$i: index\ of\ sample$
$\hat{y}: predicted\ value$
$y: expected\ (actual)\ value$

Hinge Loss / Multi-Class SVM Loss: measures the distance of a misclassified point from a decision margin. Error is 0 for correctly classified points.

$$SVMLoss = \max(0, 1 - y_i f(x_i))$$

$y_i: correct\ sign\ (\pm 1)\ of\ the\ data\ point$
$f(x_i): classifier\ output$

Gradient Descent - Linear Regression

1. Cost function (here we use Mean Squared Error)

$$f(m, b) = \frac{1}{N} \sum_{i=1}^{n} (y_i - (mx_i + b))^2$$

2. Calculate the gradient

$$f'(m, b) = \begin{bmatrix} \frac{df}{dm} \\ \frac{df}{db} \end{bmatrix} = \begin{bmatrix} \frac{1}{N} \sum -2x_i(y_i - (mx_i + b)) \\ \frac{1}{N} \sum -2(y_i - (mx_i + b)) \end{bmatrix}$$

3. Update the weights

$$m_i = m_{i-1} - \alpha \frac{\partial f}{\partial m}$$

$$b_i = b_{i-1} - \alpha \frac{\partial f}{\partial b}$$

$i: iteration$

$\alpha: learning\ rate$

4. Repeat steps 2 & 3 until convergence or for N iterations

Convergence: difference between weight values at iteration ***i*** and iteration ***i-1*** are within a specified tolerance, meaning the gradient curve is flat.

Gradient Descent - Logistic Regression

1. **Cost Function** (Log Loss)

$$Loss = -(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$$

i: *index of sample*
$\hat{y}$: *predicted value*
y: *expected (actual) value*

2. **Activation Function** (Sigmoid)

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} \qquad z = wx$$

3. Calculate the derivative of the **cost function (C)** with respect to the **weights (w)**

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w} \qquad => \frac{\partial C}{\partial w} = (\hat{y} - y)x$$

4. Update the **weights** (Batch Gradient Descent)

$$w := w - \alpha \frac{1}{N} \sum_{i}^{N} (\hat{y} - y)x$$

5. Repeat steps 3 & 4 until convergence.

$$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial \hat{y}}.\frac{\partial \hat{y}}{\partial z}.\frac{\partial z}{\partial w}$$

$$\frac{\partial C}{\partial \hat{y}} = \frac{\partial(-y\log(\hat{y}) - (1-y)\log(1-\hat{y}))}{\partial \hat{y}}$$
$$\Rightarrow \frac{\partial C}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{(1-y)}{(1-\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial(1/(1+e^{-z}))}{\partial z}$$
$$\Rightarrow \frac{\partial \hat{y}}{\partial z} = \frac{e^{-z}}{(1+e^{-z})^2}$$
$$\Rightarrow \frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$$

$$\frac{\partial z}{\partial w} = \frac{\partial(wx)}{\partial w} = x$$

$$\frac{\partial C}{\partial w} = (-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}).\hat{y}(y-\hat{y}).x$$

$$=> \frac{\partial C}{\partial w} = (\hat{y}-y)x$$

$Gradient\ Descent$

$$w := w - \alpha\frac{1}{N}\sum_{i}^{N}\frac{\partial C}{\partial w}$$
$$w := w - \alpha\frac{1}{N}\sum_{i}^{N}(\hat{y}-y)x$$

$Stochastic\ Gradient\ Descent$

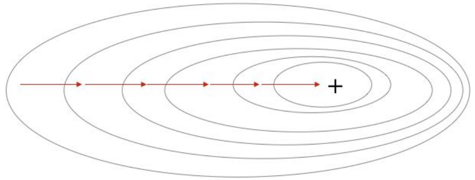
$$w := w - \alpha\frac{\partial C}{\partial w}$$
$$w := w - \alpha(\hat{y}-y)x$$

$Mini-Batch\ Gradient\ Descent$

$$w := w - \alpha\frac{1}{M}\sum_{i}^{M}\frac{\partial C}{\partial w}$$
$$w := w - \alpha\frac{1}{M}\sum_{i}^{M}(\hat{y}-y)x$$

M < N
Per epoch N/M steps

Batch



The weights of the network are updated after the entire dataset (batch) is processed.

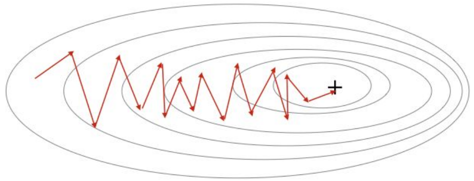
Advantages:

+ Smoother updates as the gradients average over the batch.

Disadvantages:

- Requires huge memory to store the gradients of entire batch.
- Long time before weights are updated.

Stochastic



The weights of the network are updated for each sample at every step. Thus weights change very quickly.

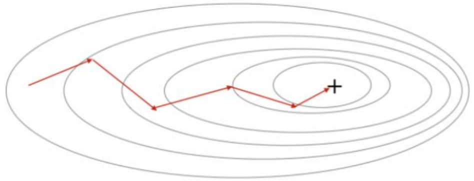
Advantages:

+ As the updates fluctuate a lot, it can possibly hop out of local minima.
+ Low memory usage and faster computation as only one sample is used at every update.

Disadvantages:

- The weights fluctuate around the optima and may take a long time to converge.
- Loses the advantage of vectorized operations (matrix and vector multiplications).

Mini-Batch



Tradeoff between batch and stochastic gradient descent. Uses a subset of the dataset to update weights at each step.

Advantages:

+ Faster and less memory intensive than batch gradient descent.
+ Smoother than stochastic gradient descent.

Disadvantages:

- Very large mini-batch sizes (~ batch) reduce the noise and can overfit to sharp saddle points.

Momentum Gradient Descent

Momentum descent exponentially average over past gradients. Helps in smoothing out the gradients not in the direction of the maximum descent and retains gradient in the direction of maximum descent.

$$v_{dW} = \beta v_{dW} + (1-\beta)\frac{\partial \mathcal{J}}{\partial W}$$

$$v_{dW}^{corrected} = \frac{v_{dW}}{1-(\beta)^t}$$

$$W = W - \alpha v_{dW}^{corrected}$$

Higher the value of β, higher is the weightage on previous gradients.

RMS Prop

As the weights are updated their gradient values can change drastically and hence it is a good idea to normalize the gradients. But since the different weights are on different scales, we cannot normalize gradients of all weights to the same value and hence we normalize each gradient by their moving average of squared gradients.

$$s_{dW} = \beta s_{dW} + (1-\beta)\left(\frac{\partial \mathcal{J}}{\partial W}\right)^2$$

$$s_{dW}^{corrected} = \frac{s_{dW}}{1-(\beta\ )^t}$$

$$W = W - \alpha \frac{\frac{\partial \mathcal{J}}{\partial W}}{\sqrt{s_{dW}^{corrected}} + \varepsilon}$$

Adam - Adaptive Momentum Estimation

It combines the best of Momentum and RMSProp. Most commonly used optimizer in present ML applications.

$$v_{dW} = \beta_1 v_{dW} + (1 - \beta_1)\frac{\partial \mathcal{J}}{\partial W}$$

$$s_{dW} = \beta_2 s_{dW} + (1 - \beta_2)(\frac{\partial \mathcal{J}}{\partial W})^2$$

$$v_{dW}^{corrected} = \frac{v_{dW}}{1 - (\beta_1)^t}$$

$$s_{dW}^{corrected} = \frac{s_{dW}}{1 - (\beta_1)^t}$$

$$W = W - \alpha \frac{v_{dW}^{corrected}}{\sqrt{s_{dW}^{corrected}} + \varepsilon}$$

Other Optimizers

- Adagrad
- Adadelta
- Nesterov Momentum