Gradients and Optimization

Deep Neural Networks
Session 09
Pramod Sharma
pramod.sharma@prasami.com

2

Agenda

- Loss / Cost Optimization
- Stochastic Gradient Descent (SGD) and others
- Momentum Learning Rates
- Adaptive Learning Rates

11/23/2024

pra-sâmi

3

Optimization Problem

- Neural Network: we solve it as a optimization problem
- Classification problem : predicting the probability of an instance belonging to each class
- Regression problem : predicting the actual value of an instance
- Gradient is actually Error Gradient
- Model is estimating weights to map inputs with the target
- Loss function : ℓ (a, y), a, in turn, is a function of W and b
- Major component comes from weights of different layers
- Compute gradient $\frac{\partial J}{\partial W}, \cdot \frac{\partial J}{\partial b}$
  - Update weights W = W - α . $\frac{\partial J}{\partial W}$. X
  - Similarly b = b - α . $\frac{\partial J}{\partial b}$

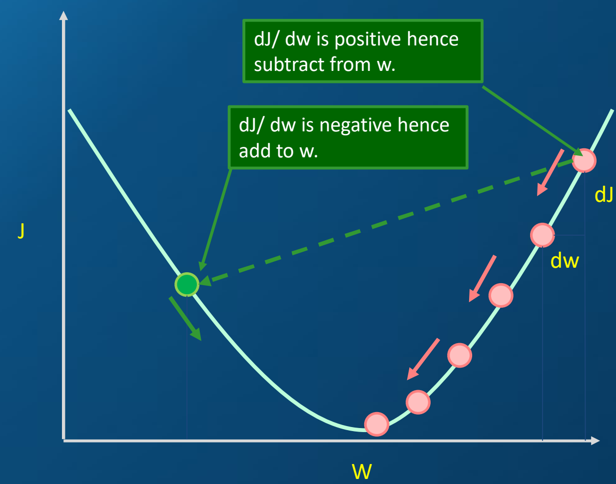
    Where α is defined as learning rate

4

Loss / Cost Optimization

5

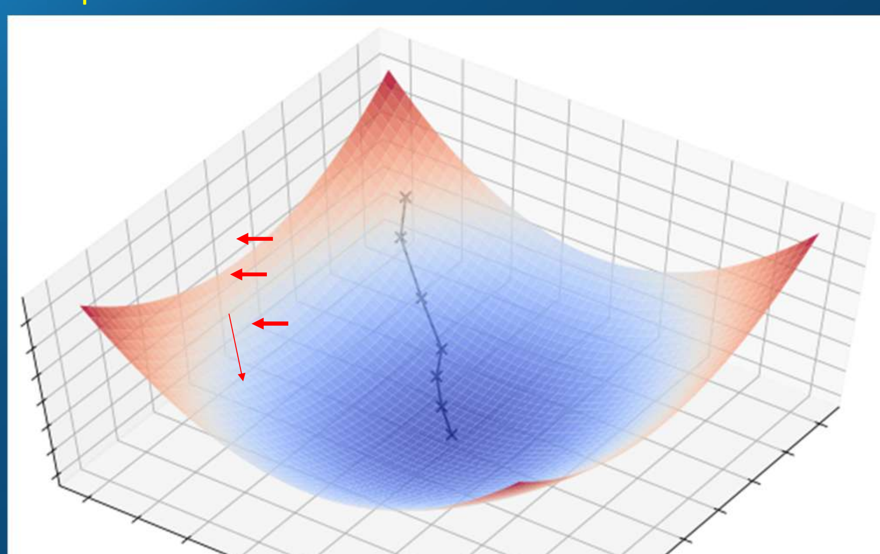
Optimization Problem



11/23/2024

6

Loss / Cost Optimization

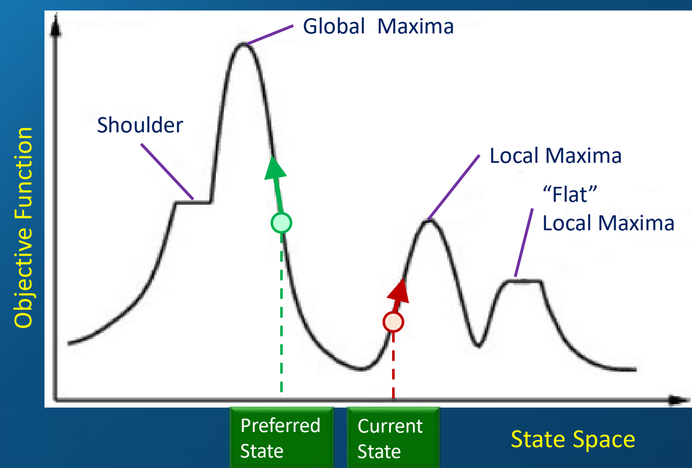


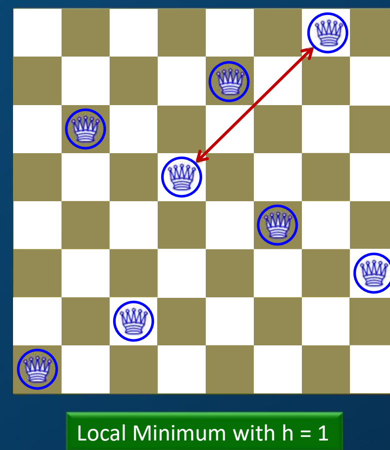
11/23/2024

7

State Space Landscape

- Problem: depending on initial state, can get stuck in local maxima/minima

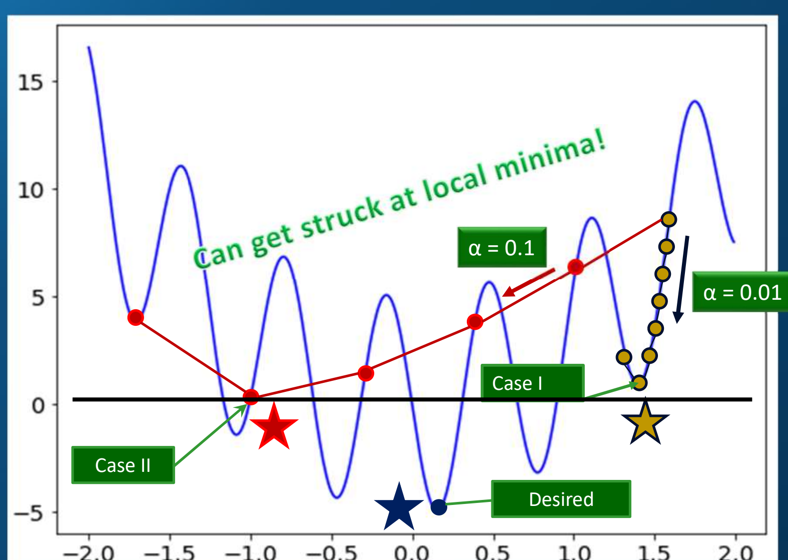




pra-sâmi

8

Learning Rate : Difficult to assess what's doing to work?



pra-sâmi

Gradient – Tough Terrain

9

11/23/2024

pra-sâmi

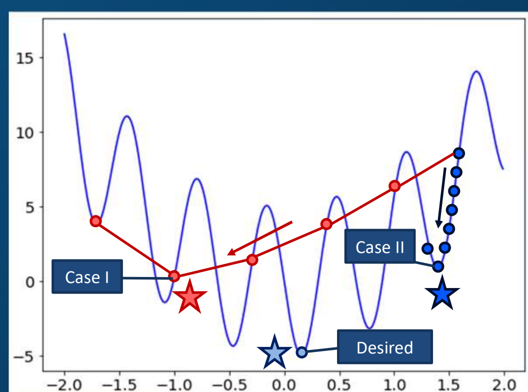
Learning Rate : Tough Terrain

10

Finding most optimal gradient descent can be difficult

Q1: How to select right learning rate?

- Too fast and you can miss minima!
- Too slow, you can be struck at local minima!
- Need to look for learning rate converges smoothly and avoids local minima! => "***Momentum***"



11/23/2024

pra-sâmi

11

Learning Rate : Tough Terrain

Question: How to select right learning rate?

- Need a learning that 'adapts' to the terrain
- No Fixed learning rate
- Learning to change as per the change in gradient
- Popular algorithms
  - SGD
  - Adam
  - Adadelta
  - Adagrad
  - RMSProp

pra-sâmi

12

Gradient Descent

- Gradient descent is one of the most popular algorithms to perform optimization
  - Most common way to optimize neural networks
- Every state-of-the-art Deep Learning library contains implementations of these algorithms
- Used as black-box optimizers
- Practical explanations of their strengths and weaknesses are hard to come by
- Makes sense to understand the implementations under-the-hood
- We optimize cost function J (W, b)
- Learning rate α decides our step size (Go down the slope… till you reach the valley!)

pra-sâmi

13

Stochastic Gradient Descent (SGD) and Others

11/23/2024

pra-sâmi

14 Stochastic Gradient Descent (SGD)

- **Stochastic** refers to a randomly determined process
- In theory it performs parameter update for each of training example
- Gradient computations are tough on computing resources
- Imagine how many calculations will be needed to cover all data points and all batches
- In this method, we pick one point randomly out of the “batch” and compute the loss for the same
- You will see wide fluctuations in the objective function
- The data set is processed in batches to parallelize the computations
- Enables it to jump over local minima with of hope of finding global minima

Our very first model…

11/23/2024

pra-sâmi

15

Batch Gradient Descent

- Stochastic Gradient Descent is computationally expensive
- There is a possibility that it can make a noisy gradient descent where values are jumping around uncontrollably
  - It was so noisy in some cases that we needed to tweak a bit in our implementation
- So we changed, it to batch gradient descent which performs model updates at the end of each training epoch

Epoch

Dictionary : "A long period of time, especially one in which there are new developments and great change"
ML: One cycle through the entire training dataset

11/23/2024

pra-sâmi

16

Batch Gradient Descent

- Vanilla gradient descent, computes the gradient of the cost function w. r. t. for the entire training dataset:

$$\frac{\partial J}{\partial W} = \frac{1}{m} * \left(\Sigma \frac{\partial \ell(a, y)}{\partial w1} \right) \text{ and } W = W - \alpha . \frac{\partial J}{\partial W}$$

- Gradients for the whole dataset to perform one update, batch gradient
- Most deep learning libraries provide automatic differentiation that efficiently computes the gradient
- Update our parameters in the direction of the gradients with the learning rate
- For non–convex surfaces, it converges to local minima
- We have coded in recent examples

11/23/2024

pra-sâmi

17

Batch Gradient Descent

- Pros:
  - Fewer updates, computationally lightweight
  - Fewer updates may result in more steady error gradient and stable convergence
  - Calculation of errors and weight calculation are separate. Hence, Easier to implement parallel processing
- Cons:
  - More stable gradient may result in premature convergence
  - Need additional step of collecting errors across all training examples
  - Model updates and training speed may become very slow for large dataset

pra-sâmi

18

Batch vs. Mini-batch

Index	X1	X2	y
0	0.871	0.64	0
1	0.987	0.633	0
2	0.52	0.405	0
…	…	…	…
127	0.857	0.44	0
128	0.154	0.161	0
129	0.642	0.722	0
…	…	…	…
256	0.825	0.844	1
257	0.763	0.244	1
258	0.562	0.225	0
…	…	…	…
383	0.22	0.573	0
384	0.953	0.797	0
385	0.118	0.62	1
…	…	…	…
…	…	…	…
1152	0.695	0.215	0

Split entire data in mini batches of 128 rows.

Batch	Index	X1	X2	y
Batch 1	0	0.660	0.775	1
	1	0.343	0.605	1
	2	0.771	0.958	0
	…	…	…	…
	127	0.291	0.231	0
Batch 2	128	0.886	0.255	1
	129	0.630	0.873	1
	…	…	…	…
	256	0.260	0.880	0
Batch 3	257	0.002	0.263	1
	258	0.055	0.351	1
	…	…	…	…
	383	0.444	0.986	1
∘ ∘ ∘ ∘ ∘	384	0.997	0.020	0
	385	0.807	0.789	0
	…	…	…	…
	…	…	…	…
	1152	0.695	0.215	0

pra-sâmi

19

Mini-Batch Gradient Descent

- Instead using batch of entire dataset, create mini batches
- Good balance between Stochastic Gradient Descent and Batch Gradient Descent.

- Pros:
  - Frequent model update
  - Maintains advantage of batched updates
  - Mini-batch prevents need to process entire training data in one go
- Cons:
  - Error information must be accumulated across mini-batches of training examples like batch gradient descent.

20

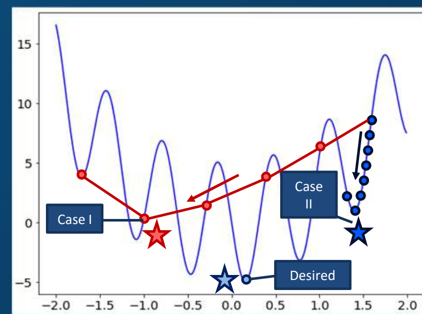
Difference

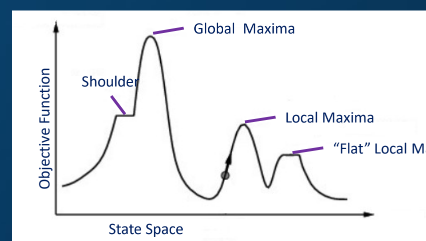
- Batch Gradient Descent : Use all 'm' examples then update once…
- Stochastic Gradient Descent : update of each example
- Mini Batch Gradient Descent : Good balance between the two…

21

Overall Challenges – Batch / Stochastic

- What's proper learning rate ???
  - Too small → painfully slow convergence
  - Too large → loss function to fluctuate around the minimum or even to diverge
- Use Learning rate schedules
  - Adjust the learning rate during training by reducing at certain interval
  - Have to be defined in advance and are thus unable to adapt to a dataset's characteristics
- While minimizing highly non-convex error functions, how to avoid getting trapped in local minima.
- What about plateau?





11/23/2024

pra-sâmi

22

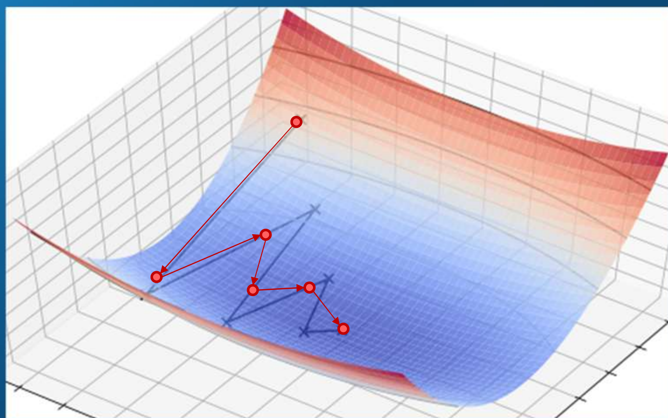
Momentum Learning Rates

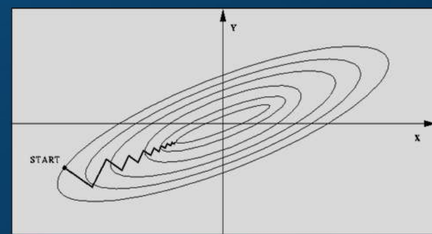
11/23/2024

pra-sâmi

23

Momentum Gradient Descent

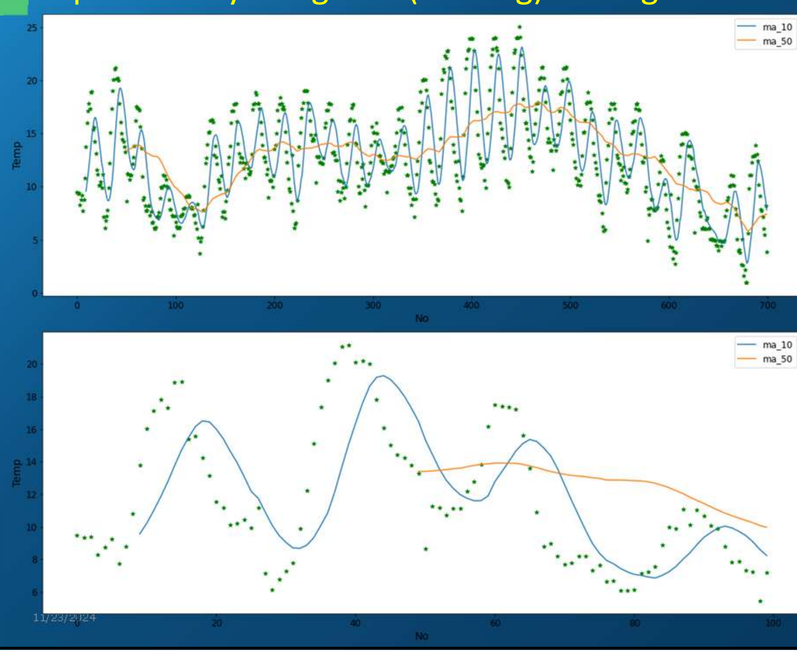




Approximate weighted averages???

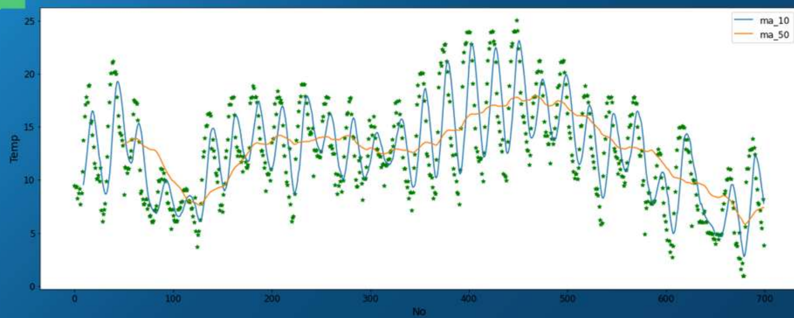
24

Exponentially Weighted (Moving) Average

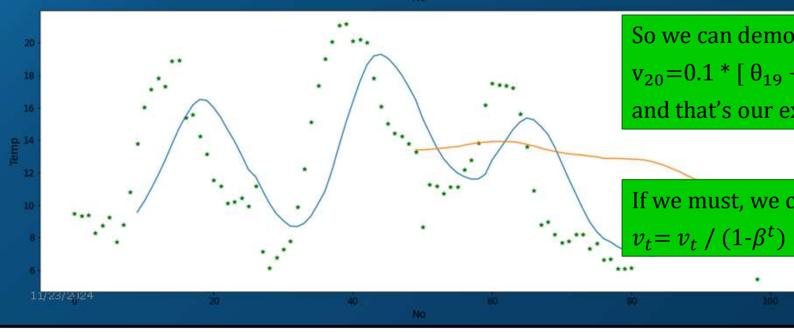


25

Exponentially Weighted (Moving) Average



$v_t = \beta\ v_{t-1} + (1+\beta) * \theta_{t-1}$



So we can demonstrate that

$v_{20} = 0.1 * [\ \theta_{19} + 0.9 * \theta_{19} + 0.9^2\ \theta_{18} + 0.9^3\ \theta_{17} + ...)\ )$

and that's our exponentially weighted moving average….

If we must, we can correct the bias using following value

$v_t = v_t\ /\ (1\text{-}\beta^t)$

11/23/2024

pra-sâmi

26

Gradient Descent with Momentum

- SGD has trouble navigating ravines
  - Areas where the surface curves much more steeply in one dimension than in another
- SGD **oscillates** across the slopes of the ravine, progress towards bottom is very slow.
- Momentum accelerate SGD in the relevant direction and **dampens oscillations**
- Momentum Gradient Descent updates as follows:

  $V_t = \beta * V_{t-1} + (1\text{-}\ \beta) * \frac{\partial J}{\partial W}$

  and $W = W - \alpha * V_t$
- Congratulations!!! β is another parameter you can tune.

11/23/2024

pra-sâmi

27

Momentum Gradient Descent

- The momentum term β is usually set to 0.9
- It accumulates momentum as it rolls downhill, becoming faster and faster
- The momentum term increases for dimensions whose successive gradients point in the same directions
- Reduces updates for dimensions whose successive gradients change directions.
- As a result, we gain faster convergence and reduced oscillation.

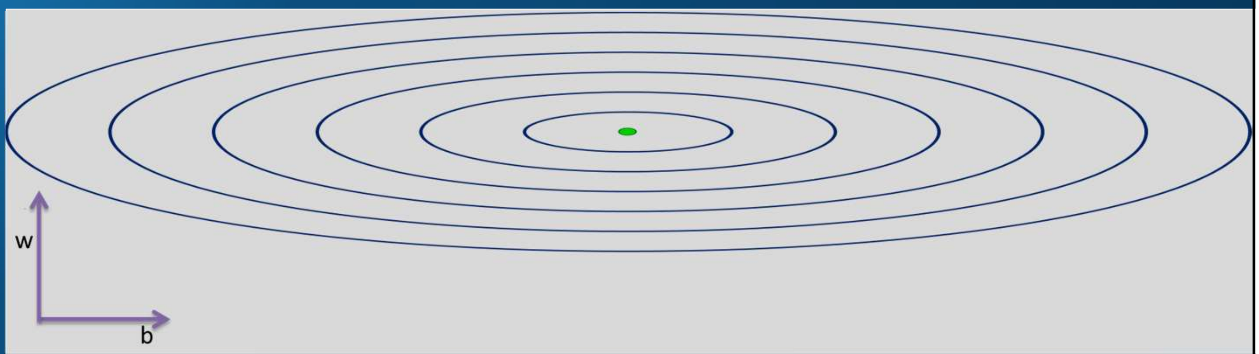
11/23/2024

pra-sâmi

28

Gradient Descent Example

- Compute dW and db
- Compute V_{dw} and V_{db}
- Update $V_{dw} = \beta V_{dw} + (1 - \beta)\, dw$ and $V_{db} = \beta V_{db} + (1 - \beta)\, db$
- Update $W = W - \alpha * V_{dw}$ and $b = b - \alpha * V_{db}$



11/23/2024

pra-sâmi

29

Adaptive Learning Rates

pra-sâmi

30

AdaGrad

- SGD needs:
  - Starting point to be selected
  - Constant learning rate
- AdaGrad tries to overcome these issues.
  - Adaptively scaled learning rate for each dimension
  - It is cumulating squares of terms in the denominator.
- In some cases learning rate can become infinitesimally small.

Previously, we performed an update for all parameters θ at once as every parameter θ_i used the same learning rate η. As Adagrad uses a different learning rate for every parameter θ_i at every time step t, we first show Adagrad's per-parameter update, which we then vectorize. For brevity, we set $g_{t,i}$ to be the gradient of the objective function w.r.t. to the parameter θ_i at time step t:

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i}) \tag{6}$$

The SGD update for every parameter θ_i at each time step t then becomes:

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i} \tag{7}$$

In its update rule, Adagrad modifies the general learning rate η at each time step t for every parameter θ_i based on the past gradients that have been computed for θ_i:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \tag{8}$$

$G_t \in \mathbb{R}^{d \times d}$ here is a diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients w.r.t. θ_i up to time step t[11], while ϵ is a smoothing term that avoids division by zero (usually on the order of $1e-8$). Interestingly, without the square root operation, the algorithm performs much worse.

From Original Paper

pra-sâmi

31

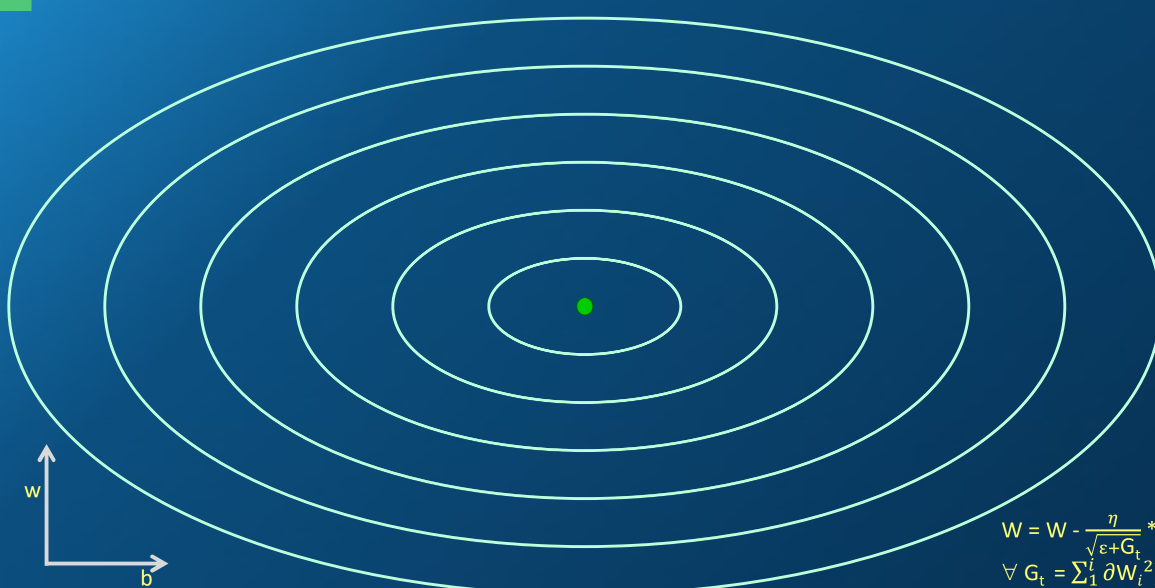
AdaGrad

- Improved the robustness of SGD
- Being used for training large-scale neural nets
  - At Google, to train GloVe word embeddings, as infrequent words require much larger updates than frequent ones.
- Previously, we performed an update for all Weights W at once as every parameter w_i used the same learning rate η (**read** α).
- As Adagrad uses a different learning rate for every weight w_i at every time step t,
- $\partial w_{t,i}$ to be the gradient of the objective function w.r.t. to the parameter w_i at time step t:
  - $W = W - \frac{\eta}{\sqrt{\varepsilon + G_t}} * \partial W$
  - Where G_t is the sum of the element wise multiplication of the gradients until time-step t, $= \sum_1^i \partial W_i^{\,2}$
- Out-of-box libraries available
- Some libraries use diagonal matrix instead of using full matrix

pra-sâmi

32

AdaGrad



$W = W - \frac{\eta}{\sqrt{\varepsilon + G_t}} * \partial W$

$\forall\, G_t = \sum_1^i \partial W_i^{\,2})$

pra-sâmi

33

Adadelta

- Derived from AdaGrad
- Aimed at reducing aggressive, monotonically decreasing learning rate
- Instead of considering entire history of time steps**, use a window**.
- Exponential decay (**Exponential Moving Average**) is also considered
- Benefits:
  - No manual adjustment of a learning rate after initial selection.
  - Insensitive to hyperparameters.
  - Separate dynamic learning rate per-dimension.
  - Minimal computation over gradient descent.
  - Robust to large gradients, noise and architecture choice.
  - Applicable in both local or distributed environments.
- All libraries have built in functions for Adadelta

11/23/2024

pra-sâmi

34

Resilient Back Propagation - Rprop

- Each weight and bias has a different, variable, implied learning rate
- Each weight has a delta value that increases when the gradient doesn't change sign (meaning it's a step in the correct direction) or decreases when the gradient does change sign
- It's not commonly used as its implementation is clumsy and not many out of box solution are available.

11/23/2024

pra-sâmi

35

RMSProp

RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in Lecture 6e of his Coursera Class[12].

RMSprop and Adadelta have both been developed independently around the same time stemming from the need to resolve Adagrad's radically diminishing learning rates. RMSprop in fact is identical to the first update vector of Adadelta that we derived above:

Added exponential moving average on AdaGrad

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \qquad (18)$$

RMSprop as well divides the learning rate by an exponentially decaying average of squared gradients. Hinton suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001.

Read β

Read α

11/23/2024

pra-sâmi

36

RMSProp

- Gradient for different weights are different
- Combines the idea of only using the sign of the gradient with the idea of adapting the step size separately for each weight
- Keep moving averages of squared gradients for each weight
- Then divide the gradient by square root the mean square above
- In Momentum, the gradient descent was modified by its exponential moving average
- RMSProp updates by taking RMS values of the gradient:
  - $v_t^2 = \beta_2 * v_{t-1}^2 + (1-\beta_2) * (\frac{\partial J}{\partial W})^2$ (element wise square)
  - and W = W - $\frac{\eta}{\sqrt{v_t^2+\varepsilon}} * \frac{\partial J}{\partial W}$ (*read α for η*)

11/23/2024

pra-sâmi

37

Adaptive Moment Estimation (Adam)

- Adam is another method that computes adaptive learning rates for each parameter.
- SGD was too simplistic, Adam is an improvement
- Adam was presented by Diederik Kingma (OpenAI) and Jimmy Ba (University of Toronto) in their 2015 ICLR paper (poster) titled “Adam: A Method for Stochastic Optimization“
  - https://arxiv.org/abs/1412.6980
- Its name Adam is derived from **Adaptive Moment Estimation**
- Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training.
- Adam is adopted from two other methods
  - Adaptive Gradient Algorithm (AdaGrad): works well with sparse gradients
  - Root Mean Square Propagation (RMSProp): works well in on-line and non-stationary settings
- Also storing an exponentially decaying average of past squared gradients v_t

pra-sâmi

38

Adaptive Moment Estimation (Adam)

- Parameters:
- α = 0.001,
- β_1 = 0.9,
- β_2 = 0.999 and
- ε = 10^{-8}
- Refer Paper cited for details of algorithm

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Diederik P. Kingma*
University of Amsterdam, OpenAI
dpkingma@openai.com

Jimmy Lei Ba∗
University of Toronto
jimmy@psi.utoronto.ca

pra-sâmi

39

Which one to use???

- For sparse data, use one of the adaptive learning-rate methods
- No need to tune the learning rate,
- Generally work with the default values
- RMSprop:
  - is an extension of Adagrad that deals with its radically diminishing learning rates.
  - Identical to Adadelta, except that Adadelta uses the RMS of parameter updates
- Adam,
  - adds bias-correction and momentum to RMSprop.
- RMSprop, Adadelta, and Adam are very similar algorithms. Pick any and then try others
- Adam slightly outperform RMSprop towards the end of optimization as gradients become sparser
- Adam may appear to be the best overall choice
- Many recent papers use vanilla SGD without momentum and a simple learning rate annealing schedule
- SGD usually achieves a minimum, but it might take significantly longer than with some of the optimizers,
- For fast convergence and train a deep or complex neural network, you should choose one of the adaptive learning rate methods

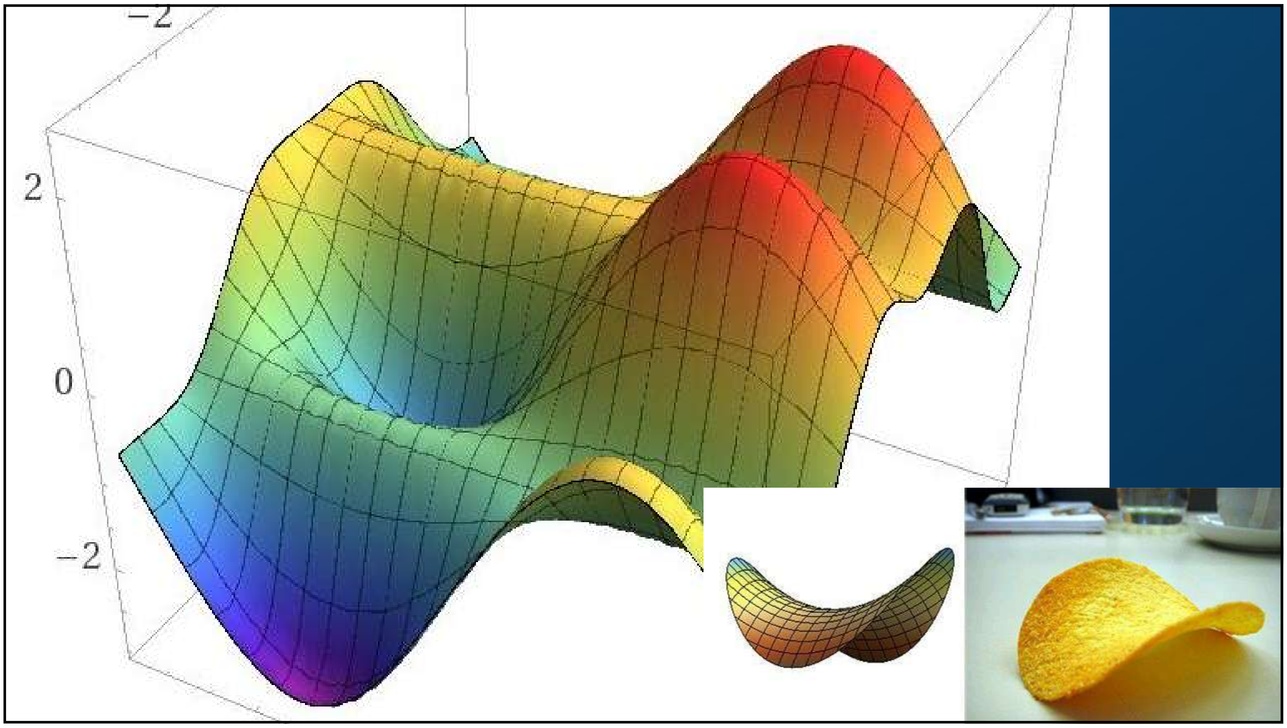
11/23/2024

40

Learning Rate Decay

- Idea is to take advantage of large steps in the beginning and then reduce your step size
  - With some threshold at a minimum value
- If you continue with the same step size,
  - You might keep hoping around
- A few recommended techniques:
  - $\alpha = \frac{1}{1 - decay\ rate * epoch\ number} * \alpha_0$
  - Or $\alpha = (0.98)^{epochnum} * \alpha_0$
  - Or may be stepped; reduce by 0.1 * α_{t-1} after 100 epochs

11/23/2024



42

Reflect…

- What is Gradient Descent used for?
  - A. Clustering
  - B. Regression
  - C. Dimensionality Reduction
  - D. Image Classification
- Answer: B. Regression
- Which of the following best describes Gradient Descent?
  - A. An optimization algorithm used to minimize a function by iteratively moving in the direction of steepest descent.
  - B. An algorithm for finding the maximum of a function.
  - C. A clustering algorithm based on distance between data points.
  - D. A classification algorithm for non-linearly separable data.
- Answer: A. An optimization algorithm used to minimize a function by iteratively moving in the direction of steepest descent.
- In Gradient Descent, what does the "gradient" represent?
  - A. The direction of steepest ascent of the function.
  - B. The rate of change of the function at a point.
  - C. The distance between data points.
  - D. The probability of occurrence of a data point.
- Answer: B. The rate of change of the function at a point.
- What is the role of the learning rate in Gradient Descent?
  - A. It determines the number of iterations.
  - B. It specifies the initial position of the algorithm.
  - C. It controls the size of the steps taken during optimization.
  - D. It sets the threshold for convergence.
- Answer: C. It controls the size of the steps taken during optimization.

11/23/2024

pra-sâmi

43

Reflect…

- Which variant of Gradient Descent updates the parameters after evaluating the gradient over the entire dataset?
  - A. Stochastic Gradient Descent (SGD.
  - B. Mini-batch Gradient Descent
  - C. Batch Gradient Descent
  - D. Momentum-based Gradient Descent
- Answer: C. Batch Gradient Descent
- Which statement best describes the trade-offs between different variants of Gradient Descent?
  - A. Batch Gradient Descent is faster than Stochastic Gradient Descent.
  - B. Stochastic Gradient Descent guarantees convergence to the global minimum.
  - C. Mini-batch Gradient Descent balances the efficiency of batch GD and the stochastic nature of SGD.
  - D. Momentum-based Gradient Descent is less prone to getting stuck in local minima.
- Answer: C. Mini-batch Gradient Descent balances the efficiency of batch GD and the stochastic nature of SGD.
- What is a potential issue with a high learning rate in Gradient Descent?
  - A. Slow convergence
  - B. Overshooting the minimum
  - C. Premature convergence to a local minimum
  - D. Increased computational complexity
- Answer: B. Overshooting the minimum
- Which of the following statements is true regarding the convergence of Gradient Descent?
  - A. Gradient Descent always converges to the global minimum.
  - B. Gradient Descent may converge to a local minimum depending on the initialization and learning rate.
  - C. Gradient Descent always converges, regardless of the function being optimized.
  - D. Gradient Descent converges faster with a smaller number of iterations.
- Answer: B. Gradient Descent may converge to a local minimum depending on the initialization and learning rate.

11/23/2024

pra-sâmi

44

Reflect…

- Which technique is often used to mitigate the issue of oscillations around the minimum in Gradient Descent?
  - A. Decreasing the learning rate over time
  - B. Increasing the learning rate over time
  - C. Using a larger batch size
  - D. Adding momentum to the updates
- Answer: D. Adding momentum to the updates
- Which of the following is NOT a common variant of Gradient Descent?
  - A. Adagrad
  - B. K-means
  - C. RMSprop
  - D. Adam
- Answer: B. K-means

11/23/2024

pra-sâmi

45

Next Session

- L-1, L-2
- Dropout
- Early Stopping
- Augmentation

11/23/2024

pra-sâmi

46

Over to Jupyter Notebook

11/23/2024

pra-sâmi

47

11/23/2024

pra-sâmi

ADDITIONAL MATERIAL

pra-sâmi

What’s Next

49

Augmented Random Search

Horia Mania Aurelia Guy Benjamin Recht
Department of Electrical Engineering and Computer Science
University of California, Berkeley
March 20, 2018

50

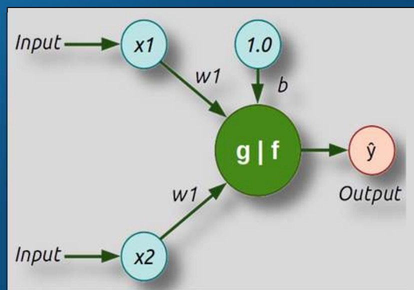
Augmented Random Search

- It's a Shallow Learning Algorithm
- Using Random Noise
- Exploits Generic Evolution Theory

51

Perceptron

- ❑ Welcome back our perceptron



11/23/2024

52

Perceptron

x1, w1, x2, w2, b → z = w1*x1+w2*x2 + b → $\hat{y} = a = \sigma(z)$ → $\ell(a, y)$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z}$$

$= \{ - y / a + (1 - y) / (1 – a) \} * a * (1-a)$

$= a - y$

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z).(1- \sigma(z))$$

$$\frac{\partial a}{\partial z} = a.(1- a)$$

$$\frac{\partial L}{\partial a} = -y/a + (1-y) / (1-a)$$

$z = W * X + b$

$\hat{y} = a = \sigma(z)$

$\sigma(z) = \frac{1}{1+e^{-z}}$

$\ell(a, y) = -y * \log(a) + (1-y) * \log(1-a)$

For binary classification:

$\ell(a,y) = -y * \log(a)$

$\Longrightarrow$

$$\frac{\partial \ell}{\partial w1} = x1 \cdot \frac{\partial L}{\partial z} = x1\ (a-y)$$

$$\frac{\partial \ell}{\partial w2} = x2 \cdot \frac{\partial L}{\partial z} = x2\ (a-y)$$

$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial z} = (a-y)$$

$\Longrightarrow$

$w1 = w1 - \alpha * \frac{\partial L}{\partial w1} = w1 – \alpha * x1 * (a-y)$

$w2 = w2 - \alpha * \frac{\partial L}{\partial w2} = w2 - \alpha * x2 * (a-y)$

$b = b - \alpha * \frac{\partial L}{\partial b} = b - \alpha * (a-y)$

Where α is learning rate. The cost function is

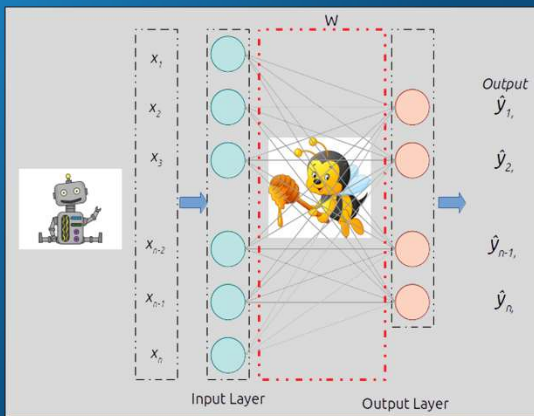
$J(W, b) = \frac{1}{m} * (\Sigma\ \ell(a, y))$

Hence $\frac{\partial J}{\partial w1} = \frac{1}{m} * (\Sigma \frac{\partial \ell(a, y)}{\partial w1})$

11/23/2024

53

Perceptron



Shape of Weights W (inputs, outputs)

- Earlier we were using very elaborate calculations to estimate direction of gradient descent.....
- Why???
- Can't we just calculate in both directions and see what's better?
- Not a bad suggestion... lets workout a procedure....
- Add random noise to weights W
- Run a trial run
- Move in the direction of improvement

54

Method of Finite Difference

- Generate small random numbers (δW)
- Shape of δW will be same as that of W
- Create W_1 and W_2 as follows
  - W_1 = W + δW
  - W_2 = W – δW
- Test both version and record Loss from each G_p, G_n
- Update Weights as follows
- W = W + α $(G_p - G_n)$. δW
- Keep repeating till it converges to a Minima (We are minimizing the cost!)
- Can easily be implemented for gain in a similar fashion

55

Learning Rate Decay

- Slowly reduce learning rate.
- As mentioned before mini-batch gradient descent won't reach the optimum point (converge). But by making the learning rate decay with iterations it will be much closer to it because the steps (and possible oscillations) near the optimum are smaller.
- One equations is
  - learning_rate = (1 / (1 + decay_rate * epoch_num)) * learning_rate_0
  - epoch_num is over all data (not a single mini-batch)
- Other learning rate decay methods (continuous):
  - learning_rate = ($0.95^{epoch\ num}$) * learning_rate_0
  - learning_rate = ($k\ /\ sqrt(epoch\_num)$) * learning_rate_0
- Some people perform learning rate decay discretely - repeatedly decrease after some number of epochs
- Some people are making changes to the learning rate manually
  - 'decay_rate' is another hyperparameter
- Learning rate decay has less priority… last thing to tune in your network

11/23/2024

pra-sâmi