Please sign-in: goo.gl/4R9Zca

Gradient-Based Learning

BOSTON UNIVERSITY
MACHINE INTELLIGENCE COMMUNITY

Devin de Hueck, Justin Chen
Sept. 19, 2018

Goals of this Workshop

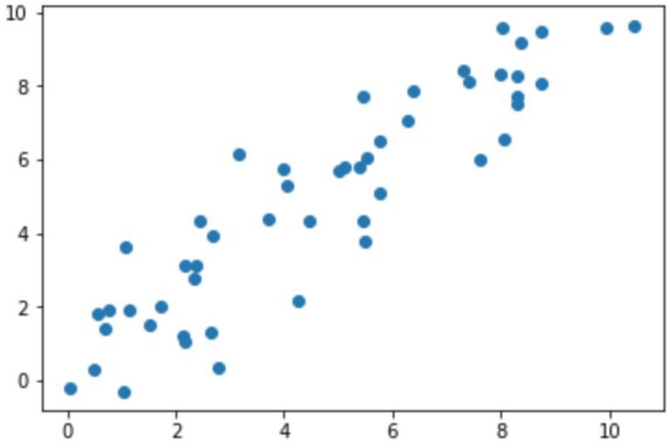
- Introduction to statistical learning
- Cost/error functions
- Gradient descent
- Variations of gradient descent

The learning problem

We have some data D

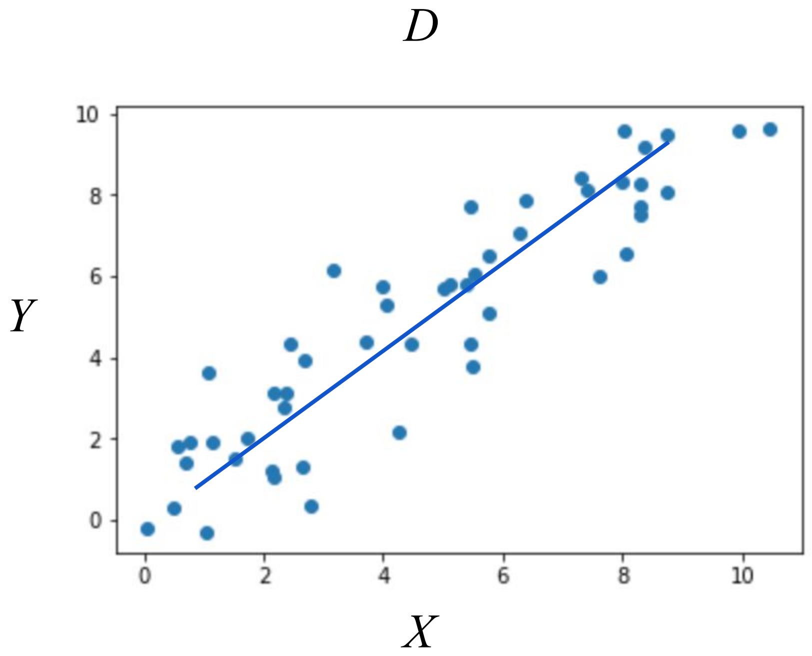
D

Y



X

Make an assumption about D



$$y = b + mx$$

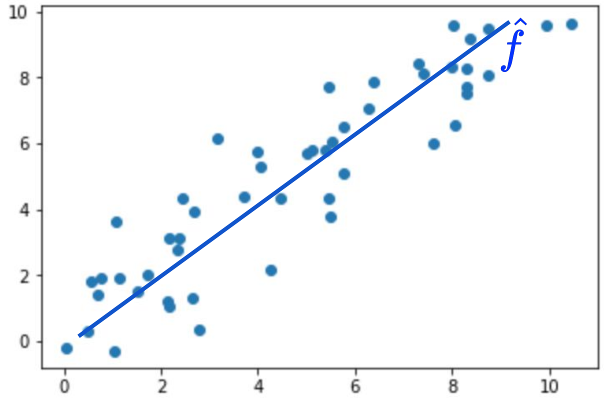
$$\Downarrow$$

$$\hat{f} = \theta_0 + \theta_1 x$$

What is learning?

The approximation of some unknown function f based on some data D.

D



Y

X

$$f : X \to Y$$

$$\hat{f} = \theta_0 + \theta_1 x$$

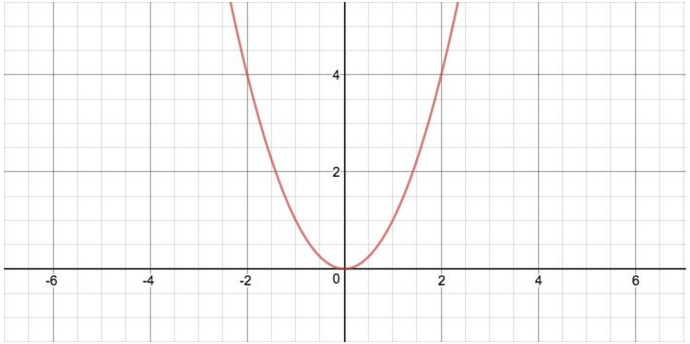
How do we set the parameters?
With the use of **gradients**.

What are gradients?

Univariate Functions

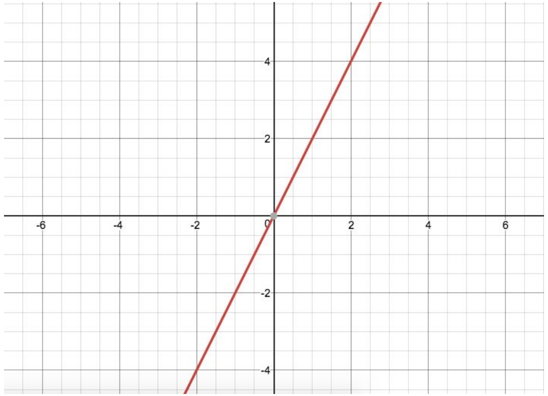
Univariate function

$$f(x) = x^2$$



Derivative

$$\frac{\delta f}{\delta x} = \text{2x}$$



Multivariate functions

Multivariate function

Function of more than one variable/**dimension**

$$f(\bar{x}) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 \ldots$$

Multivariate derivative

Shows us slope of a function in all dimensions

$$\nabla f = \begin{bmatrix} \frac{\delta f}{\delta x_0} \\ \frac{\delta f}{\delta x_1} \\ \vdots \\ \frac{\delta f}{\delta x_n} \end{bmatrix}$$

ex.

$$f(x_0, x_1) = 4x_0 + 8x_1$$

$$\nabla f = [4,8]$$

The gradient tells us how much the output of some function will change with respect to a parameter of that function.

How are gradients used to learn?

Cost Functions

- Learning algorithms need to measure how wrong it is in order to improve the model
- A **Cost function** measures the **error** distance between a **hypothesis** and a **label** (correct answer)
- For example
  - Learning algorithm's hypothesis: [0.1, 0.2, 0.3]
  - Correct answers: [0.5, 0.8, 0.2]
  - Errors: [-0.4, -0.6, 0.1]
  - Least Square Error (LSE): $\frac{1}{2}[(0.1-0.5)^2+(0.2-0.8)^2+(0.3-0.2)^2]$

Different Cost Functions

- Common terms: **cost/criterion/objective/loss function**
- Tailored for the model
- Common cost functions:

Regression: **Least Square Error**

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Classification: **Negative Log Likelihood a.k.a Cross Entropy**

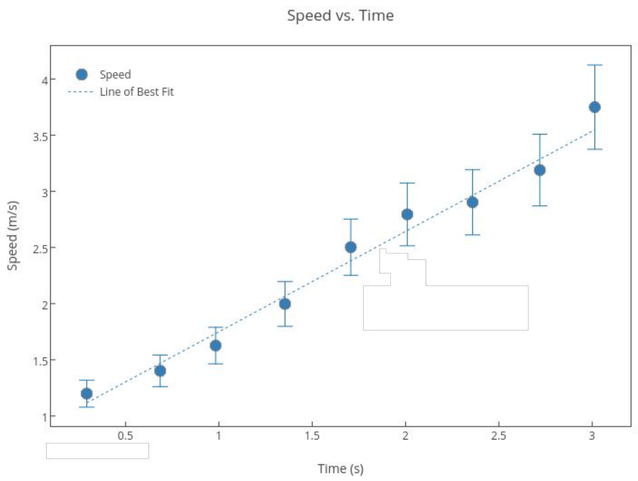
$$J(\theta) = \sum_{i=1}^{m}(y^{(i)}log(h_\theta(x^{(i)})) + (1 - y^{(i)})log(1 - h_\theta(x^{(i)})))$$

LSE Explained

$h_\theta(x^{(i)})$ - Your model's prediction

$y^{(i)}$ - The correct answer (label)

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$



https://plotlyblog.tumblr.com/post/84309369787/best-fit-lines-in-plotly

Optimizing Cost Function

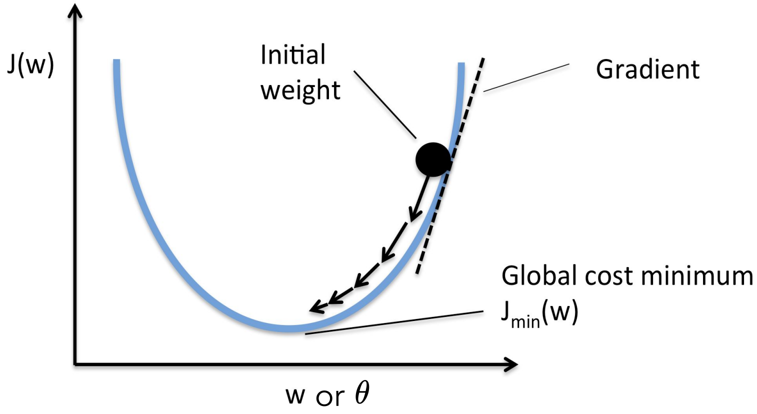
- Goal is to **minimize cost function** J
  - Compute derivative of J w.r.t. parameters θ

$$\nabla J = \frac{\delta J}{\delta \theta}$$

- Consider this simple cost function

$$f(x) = x^2 \quad \longrightarrow \quad \frac{\delta f}{\delta x} = 2\mathrm{x}$$

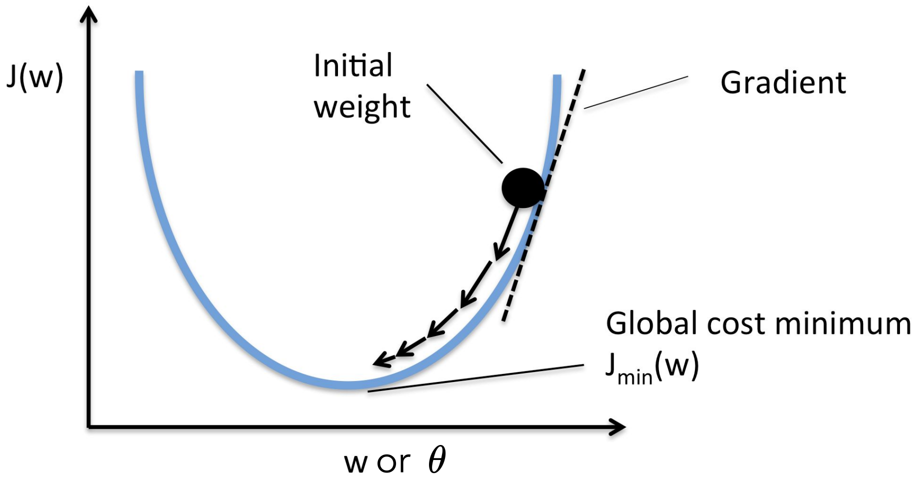
  - Solve derivative for 0
  - **Convex** functions have single **global minima**
  - Most **cost landscapes** are **non-convex** - contain many **local minima (exponentially many in the number of dimensions)**



https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html

Gradient Descent

- Assumption:
  - Inputs are sampled i.i.d.
- Use gradient to iteratively traverse parameter landscape
- Gradient is direction of steepest ascent



https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html

Gradient Descent

Scalar learning rate

$$\theta_j := \theta_j - \alpha \overbrace{\frac{\partial}{\partial \theta_j} J(\theta)}^{\text{Also written } \nabla J(\theta)}$$

Individual weights

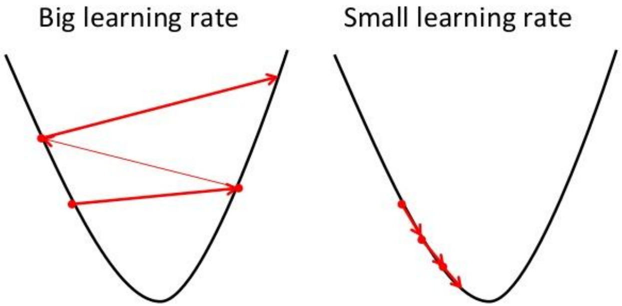
Vector of weights

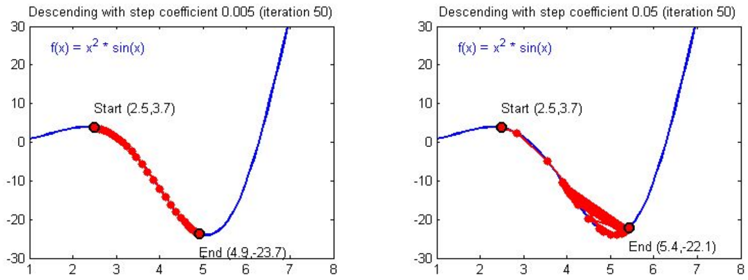
Cost/objective/loss function

Learning rate

- Gradient points in direct of steepest ascent, but we use the learning rate to control step size
- Typical learning rates to try:

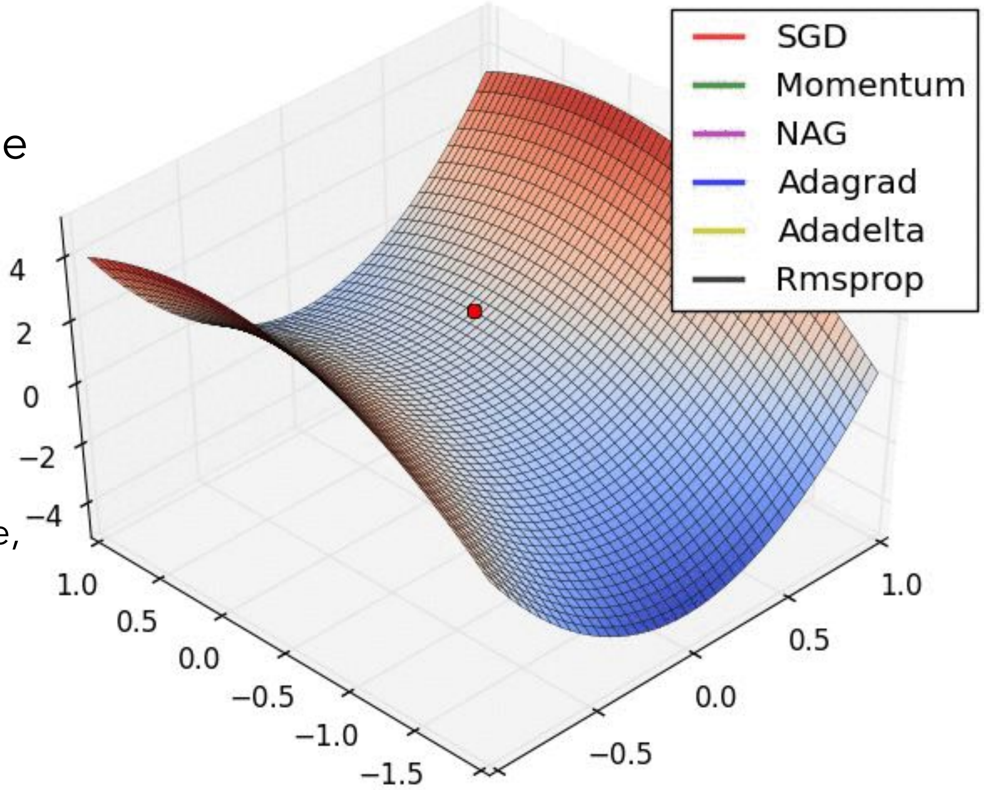
  0.1, 0.01, 0.001, 0.0001





Saddle points

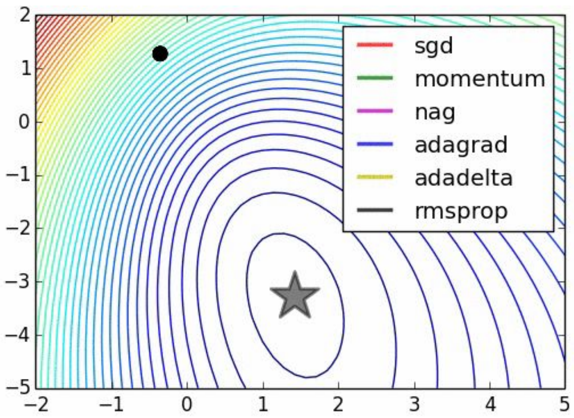
- If you are right on the saddle, the gradient does not always help you get off
- Can exist above one dimension
- Many solutions to this problem exist
  - A simple one is by just adding noise, you can force the algorithm to randomly “catch on” to the downward slope



http://cs231n.github.io/neural-networks-3/

Different types of Gradient Descent

- Batch/Vanilla gradient descent (which we have been describing so far)
- Stochastic gradient descent
- A third variety is mini-batch gradient descent
  - Between BGD and SGD
- Adagrad, Adam



http://blog.hackerearth.com/3-types-gradient-descent-algorithms-small-large-data-sets

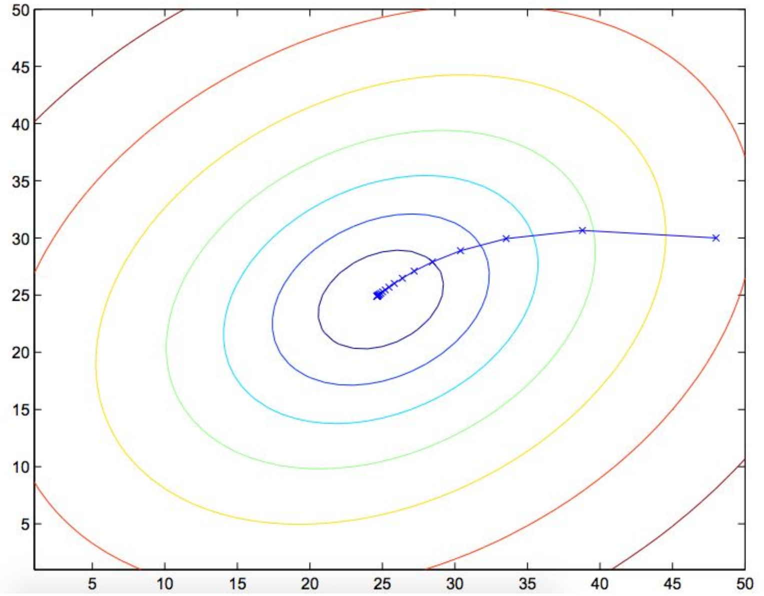
Batch/Vanilla Gradient Descent (GD)

Batch size of the entire dataset. Descend in the steepest direction given information from every training example in the dataset.

$$Loop\,\{$$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

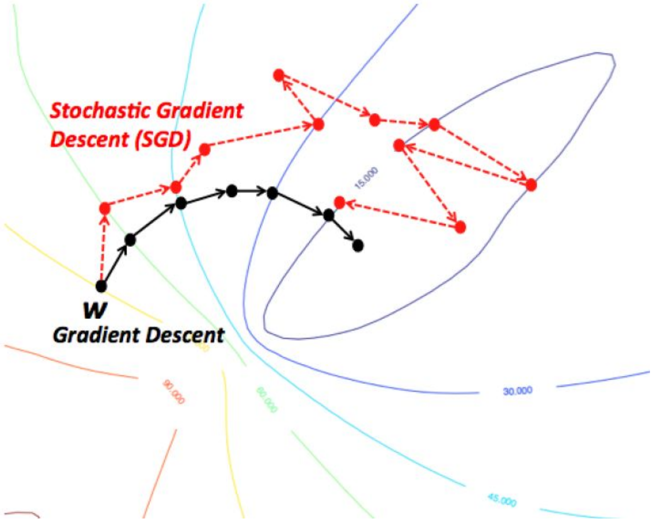
$$\}$$



Stanford CS229 Lecture Notes by Andrew Ng

Stochastic Gradient Descent

- Calculate the gradient using one random data sample at a time
  - Takes many iterations to go over entire data set
  - Each time the full data set is covered is an "epoch"
- Makes for as noisier descent, which can be useful at times
  - See saddle point

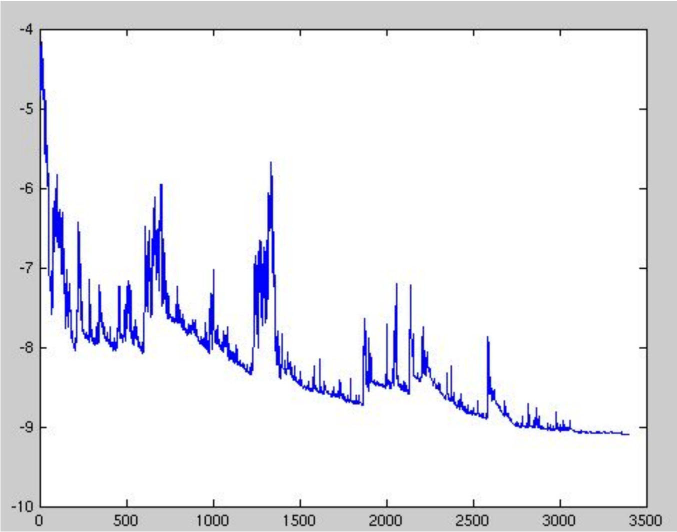


http://www.bogotobogo.com/python/scikit-learn/scikit-learn_batch-gradient-descent-versus-stochastic-gradient-descent.php

Stochastic Gradient Descent (SGD)

Size of dataset

$$
\begin{aligned}
&Loop\,\{\\
&\qquad for\ i = 1\ to\ m\ \{\\
&\qquad\qquad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)\\
&\qquad \}\\
&\}
\end{aligned}
$$



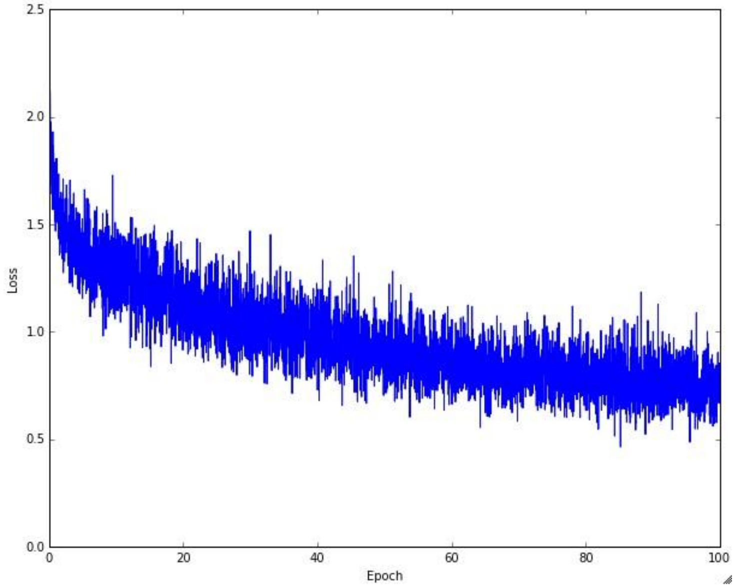
https://upload.wikimedia.org/wikipedia/commons/f/f3/Stogra.png

Mini-Batch SGD

Number of mini-batches

$$
\begin{aligned}
&Loop\,\{\\
&\qquad for\ i = 1\ to\ m\ \{\\
&\qquad\qquad \theta_j := \theta_j - \alpha \frac{1}{|b_i|} \sum_{x \in b_i} \frac{\partial}{\partial \theta_j} J(\theta_j, x)\\
&\qquad\}\\
&\}
\end{aligned}
$$

ith mini-batch

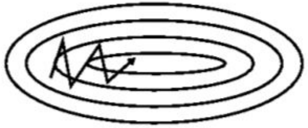


http://cs231n.github.io/neural-networks-3/

Momentum

- Helps SGD by reducing oscillation and focusing on the relevant direction to move
- Gives SGD a ‘short term memory’ by introducing a velocity term.
    - adds a fraction γ of the past time step’s update vector to the current update vector

(a) SGD without momentum



(b) SGD with momentum

Update Rule:

$$v_t := \gamma v_{t-1} + \nabla_\theta J(\theta)$$

$$\theta := \theta - v_t$$

γ - the momentum term, usually set to 0.9

Adagrad

- Adapts the learning rate to each parameter
  - *Large updates* for *infrequent* features
  - *Small updates* for *frequent* features
- Relies on accumulating information throughout training
  - Which will eventually bring the learning to effectively zero :(

Update Rule:

$$\theta_i := \theta_i - \frac{\alpha}{\sqrt{G_{ii} + \epsilon}} \cdot \nabla_\theta J(\theta_i)$$

G_{ii} - the sum of the squares of the gradients w.r.t θ_i at current step.

Adam - Adaptive Moment Estimation

- Comparative to a combination of Momentum and Adagrad
- Exponentially decaying gradient sum for update - akin to momentum
- Gradient squared sum exponentially decays - solves adagrad problem.
- Gaining in popularity - generalizes well

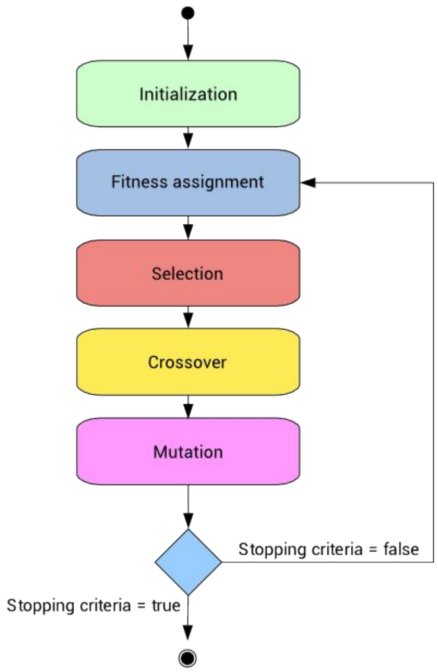
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Update Rule:

$$\theta_i := \theta_i - \frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

Anything else besides Gradient Based Learning?

- Yes, but GD is fairly easy and effective
- Another cool option is using evolutionary models - essentially a guided random search
- Need three things for evolution
  - Variation
  - Heritability
  - Selection



https://www.neuraldesigner.com/blog/genetic_algorithms_for_feature_selection

Download the Juypter Notebook:

https://goo.gl/WZDwgC

Gradient Descent for Linear Models

For the BUMIC gradient based learning workshop - fall 2018

Random Model

Let's generate some random data and make a random prediction for the best fit line. As you probably expecte

In [1]:

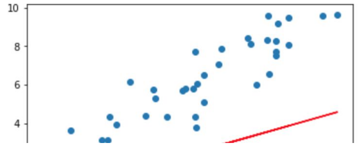
```
import matplotlib.pyplot as plt
import numpy as np

# Generate some random linear data
np.random.seed(42)
X = np.arange(0, 10, 0.2) + np.random.normal(size=50)
Y = np.arange(0, 10, 0.2) + np.random.normal(size=50)

# Hypothesize a line - just a random guess right now
h_slope, h_intercept = np.random.rand(), np.random.rand()

# Create a list of values in the hypothesized line
abline_values = [h_slope * x + h_intercept for x in X]

# View out hypothesis
plt.scatter(X, Y)
plt.plot(X, abline_values, 'r')
plt.show()
```



References & Further Reading

1. Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv:1609.04747 (2016).
2. Sun, Xu, et al. "meProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting." arXiv:1706.06197 (2017).
3. Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv:1412.6980 (2014).
4. Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." arXiv:1212.5701 (2012).
5. Du, Simon S., et al. "Gradient Descent Can Take Exponential Time to Escape Saddle Points." arXiv:1705.10412 (2017).
6. Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.
7. Bottou, Léon. "Curiously fast convergence of some stochastic gradient descent algorithms." Proceedings of the symposium on learning and data science, Paris. 2009.

Upcoming Events

- Introduction to Neural Networks
- First Hack Night on Sunday, September 30th
- First Paper Discussion within the next two weeks

Non-convex Optimization Issue