Gradient-Based Learning

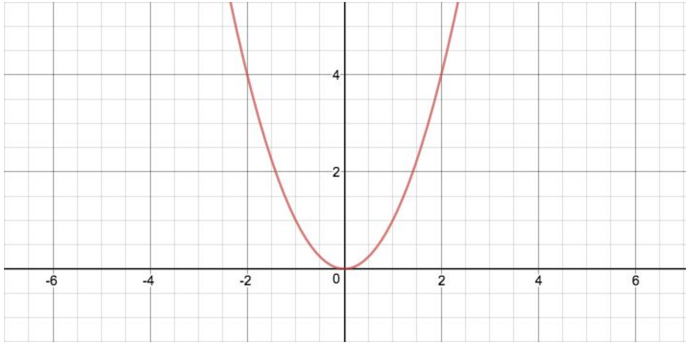
BOSTON UNIVERSITY
MACHINE INTELLIGENCE COMMUNITY

Justin Chen
Sept. 19, 2017

Univariate Functions

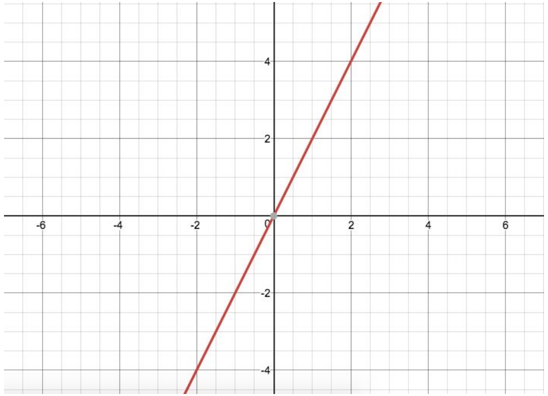
Univariate function

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



Derivative

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



Multivariate functions

Multivariate function

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

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

Multivariate derivative

Shows us slope of a function in all dimensions

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

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

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

MACHINE INTELLIGENCE COMMUNITY

Linear Equations in PYTORCH

Generate random matrix

```
>> import torch
>> r = torch.randn(2,3)

-0.2220  1.3369 -1.3627
 0.0863  0.8932 -0.6577
[torch.FloatTensor of size 2x3]
```

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

```
>> import torch.nn as nn
>> f = nn.Linear(2,1)
>> [x.data for x in a.parameters()]
[
 0.1685
-0.0136
[torch.FloatTensor of size 2x1]
,
-0.5899
-0.8569
[torch.FloatTensor of size 2]
]
```

weights

bias

MACHINE INTELLIGENCE COMMUNITY

Cost Functions

- Learning algorithms need to measure how wrong it is in order to improve the model
- **Cost function** measures **error** distance between **hypothesis** and **label** (correct answer)
- For example
  - Learning algorithm’s hypothesis: [0.1, 0.2, 0.3]
  - Correct answers: [0.5, 0.8, 0.9]
  - Errors: [0.4, 0.6, 0.6]
  - Least Square Error (LSE): $\frac{1}{2}[(0.1-0.5)^2+(0.2-0.8)^2+(0.3-0.9)^2]$

Different Cost Functions

- Common terms: **cost/ 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**

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

MACHINE INTELLIGENCE COMMUNITY

Loss Functions in PYTORCH

Negative Log Likelihood

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

```
>> import torch.nn.functional as F
>> from torch.autograd import Variable
>> input = Variable(torch.randn(3, 5))
>> target = autograd.Variable(torch.LongTensor([1, 0, 4]))
>> loss = F.nll_loss(input, target)
Variable containing:
 2.2282
[torch.FloatTensor of size 1]
```

MACHINE INTELLIGENCE COMMUNITY

Optimizing Cost Function

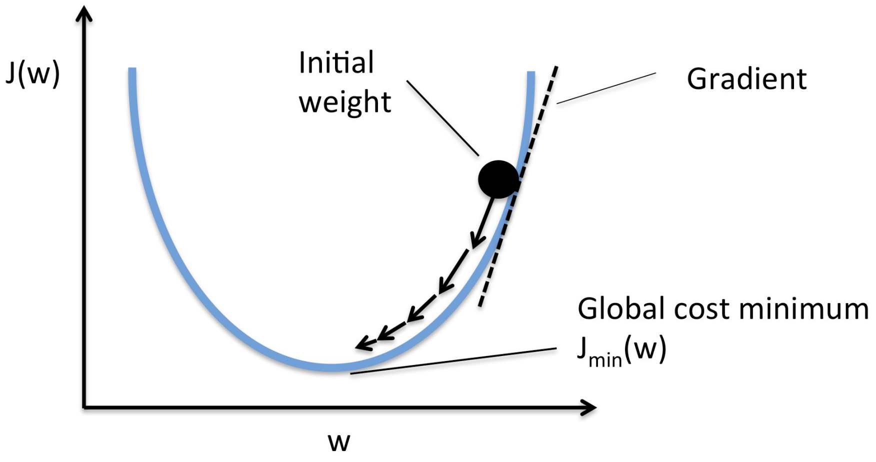
- Goal is to **minimize cost function** J
  - Compute derivative of J w.r.t. parameters θ

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

- Consider this simple cost function

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

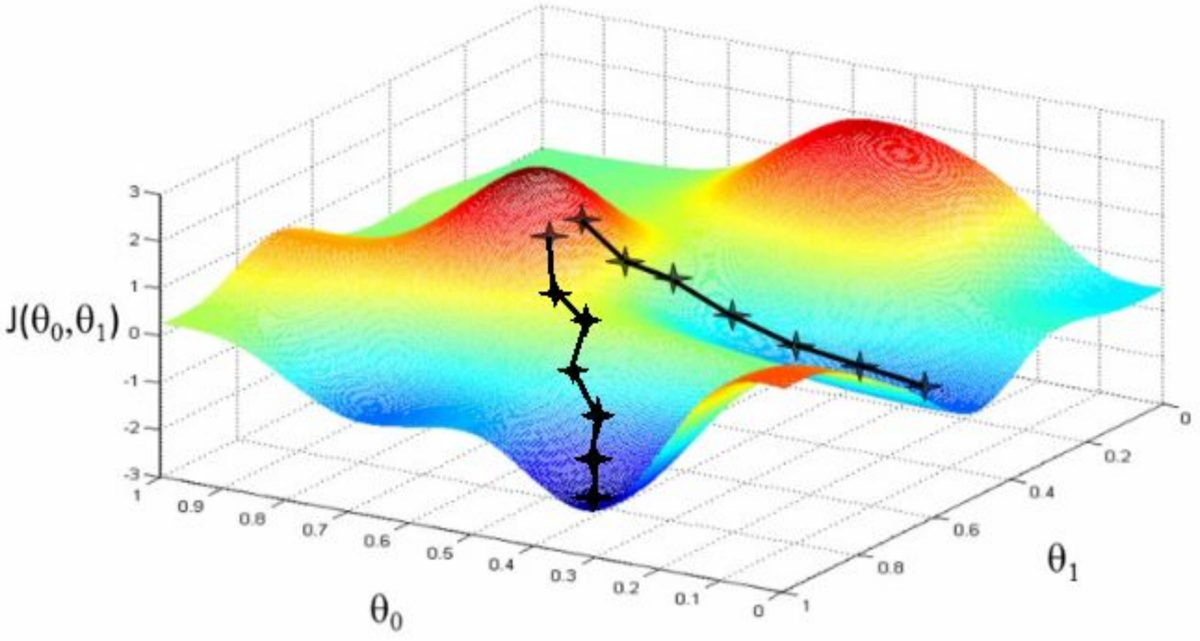
  - Solve derivative for 0
  - **Convex** functions have single **global minima**
  - Most **cost landscapes** are **non-convex** - contain many **local minima (exponentially many in the number of dimensions)**



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

MACHINE INTELLIGENCE COMMUNITY

Non-convex Optimization



MACHINE INTELLIGENCE COMMUNITY

Gradient Descent

Scalar learning rate

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

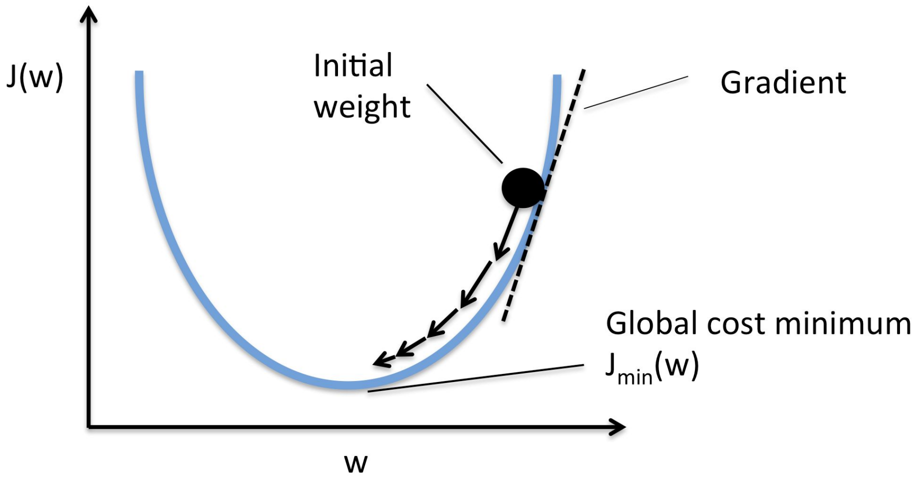
Individual weights

Cost/objective/loss function

Vector of weights

Gradient Descent

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



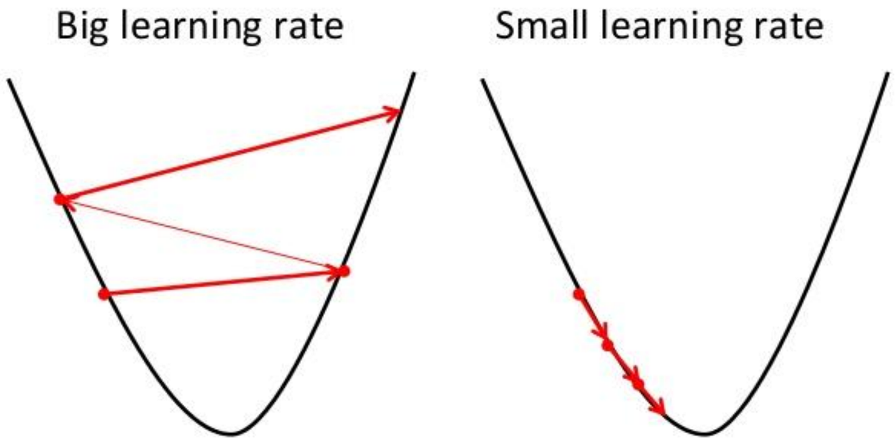
https://sebastianraschka.com/faq/docs/closed-form-vs-gd.html

MACHINE INTELLIGENCE COMMUNITY

Learning rate

- Gradient points in direct of steepest descent, but does not indicate magnitude of step
- Multiply the learning rate to slow down how fast our network tries to compensate for a given piece of data
- Typical learning rates to try:

  0.1, 0.01, 0.001, 0.0001



Robbins-Monro Conditions

Condition 1:

$$\sum_t \alpha_t = \infty$$

Condition 2:

$$\sum_t \alpha_t^2 < \infty$$

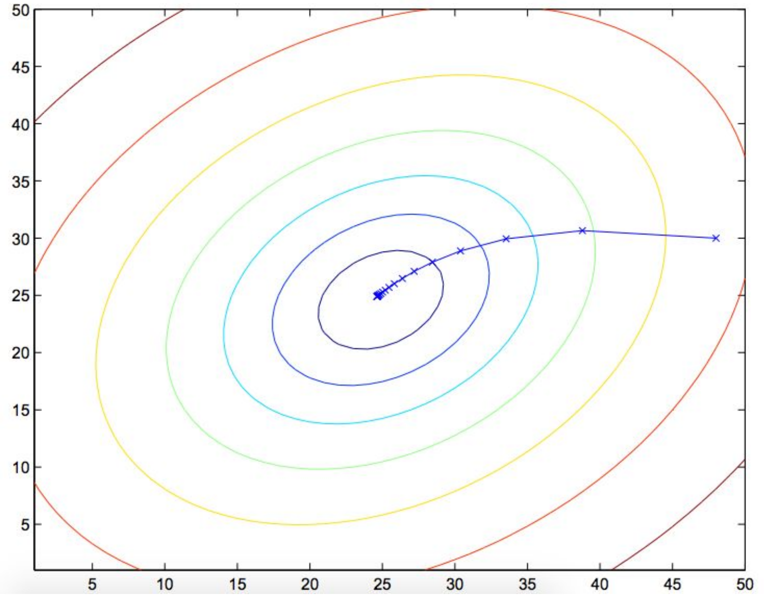
MACHINE INTELLIGENCE COMMUNITY

Batch Gradient Descent (GD)

$$Loop\,\{$$

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

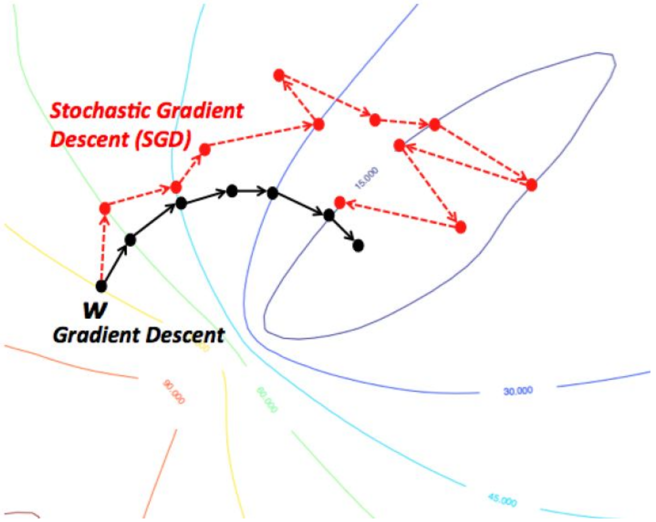
$$\}$$



MACHINE INTELLIGENCE COMMUNITY

Stochastic Gradient Descent

- Calculate the gradient using one 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”
- Works better when there are many minima in a complex “manifold”
- Makes for as noisier descent, which can be useful for
  - Noise can be reduced with mini-batch
    - Use 8 or 16 samples at a time
  - Noise can be useful for “saddle points”



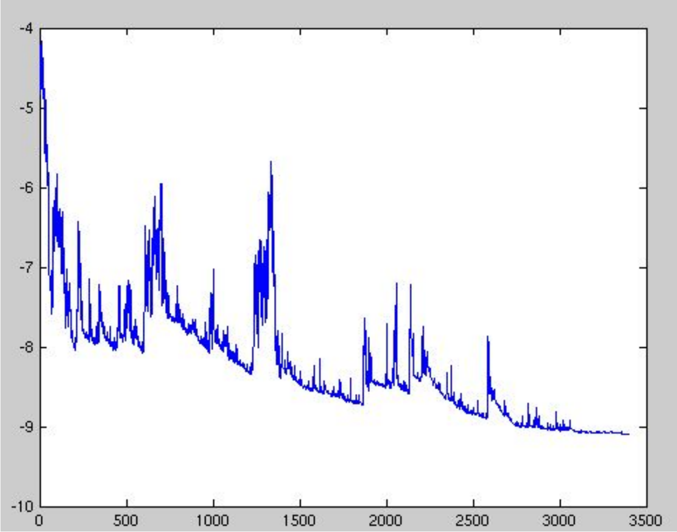
http://www.bogotobogo.com/python/scikit-learn/scikit-learn_batch-gradient-descent-versus-stochastic-gradient-descent.php

MACHINE INTELLIGENCE COMMUNITY

Stochastic Gradient Descent (SGD)

Size of dataset

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

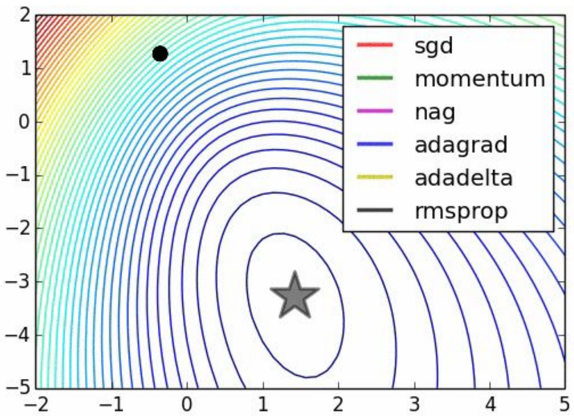


https://upload.wikimedia.org/wikipedia/commons/f/f3/Stogra.png

MACHINE INTELLIGENCE COMMUNITY

Different types of Gradient Descent

- The concept of gradient descent can be actualized in many different ways. Two popular, and simple, implementations of gradient descent are:
    - Batch gradient descent (which we have been describing so far)
    - Stochastic gradient descent
- A third variety is mini-batch gradient descent
    - Still simple, though more abstract in why it works
    - Between batch and gradient, select batches at a time (ex. 16 out of 100,000)
- For each method, there are multiple algorithms used to optimize the descent calculation



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

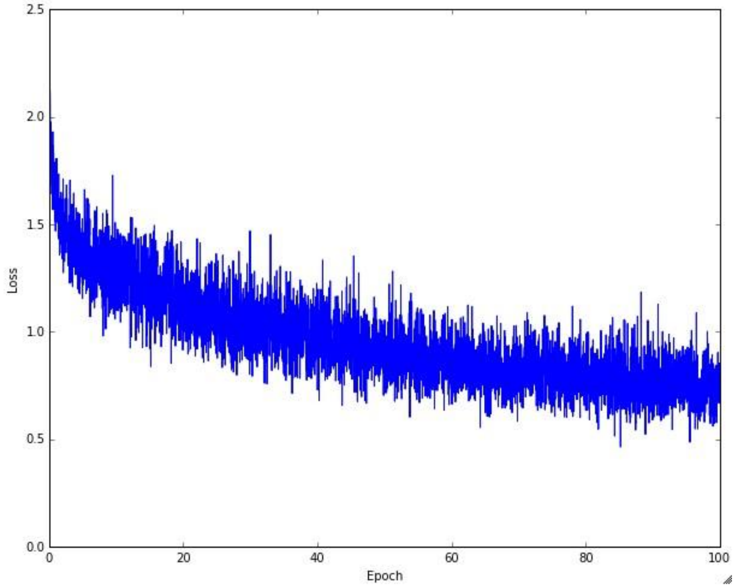
MACHINE INTELLIGENCE COMMUNITY

Mini-Batch SGD

Number of mini-batches

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

ith mini-batch

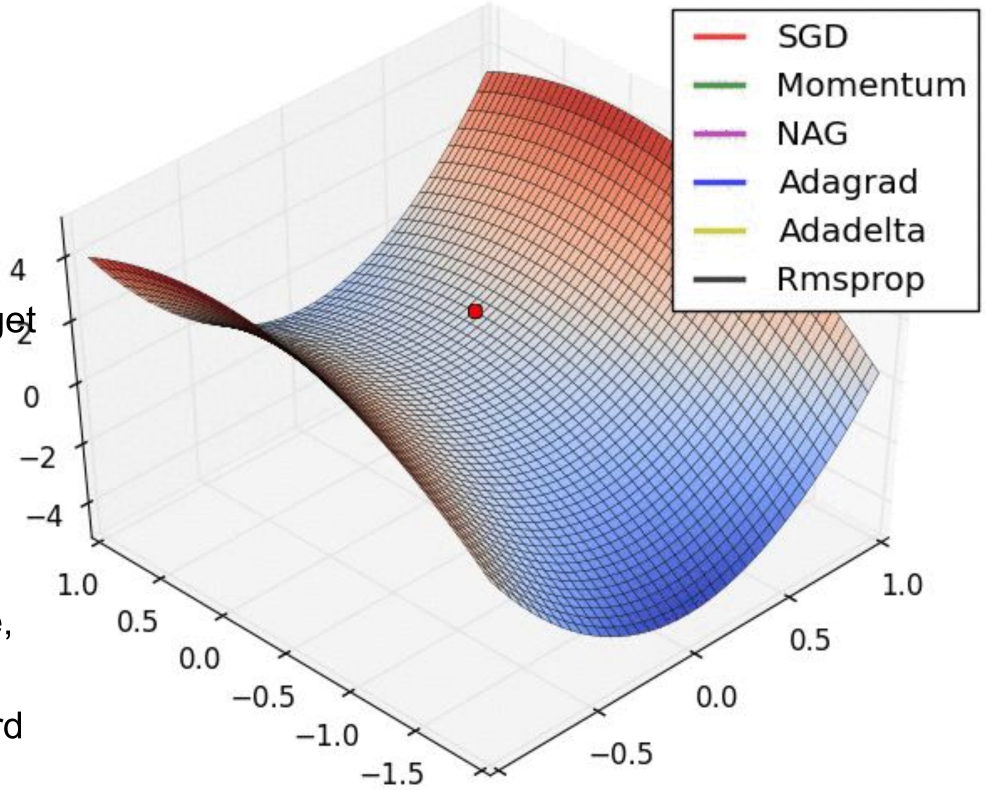


MACHINE INTELLIGENCE COMMUNITY

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

Saddle points

- Can be a big issue for gradient descent.
  - 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/

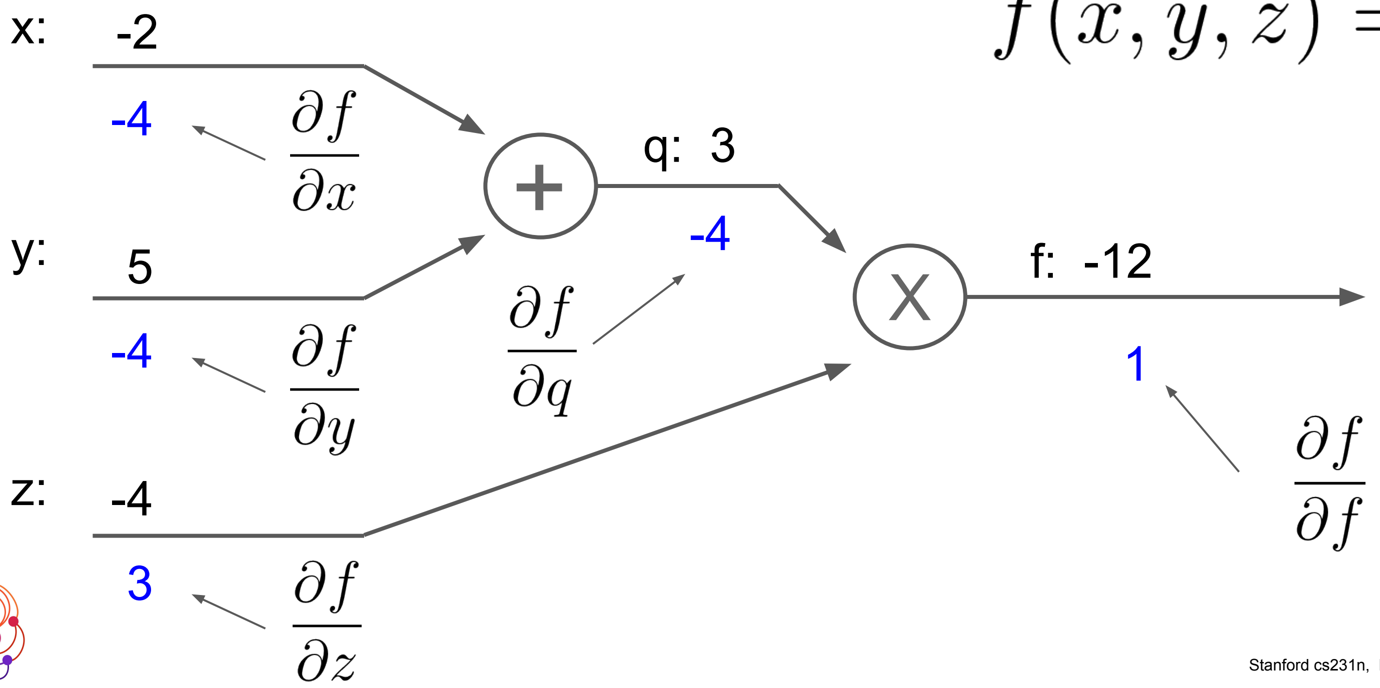
Backpropagation (BP)

- Method for calculating the error contribution of a single computation unit
    - We can use this information to update weights (parameters), and get closer to a model with less loss
- Neurons (computational units) that contributed more to the error will be changed more by BP
    - Scaled by the learning rate, how much should the weight be adjusted

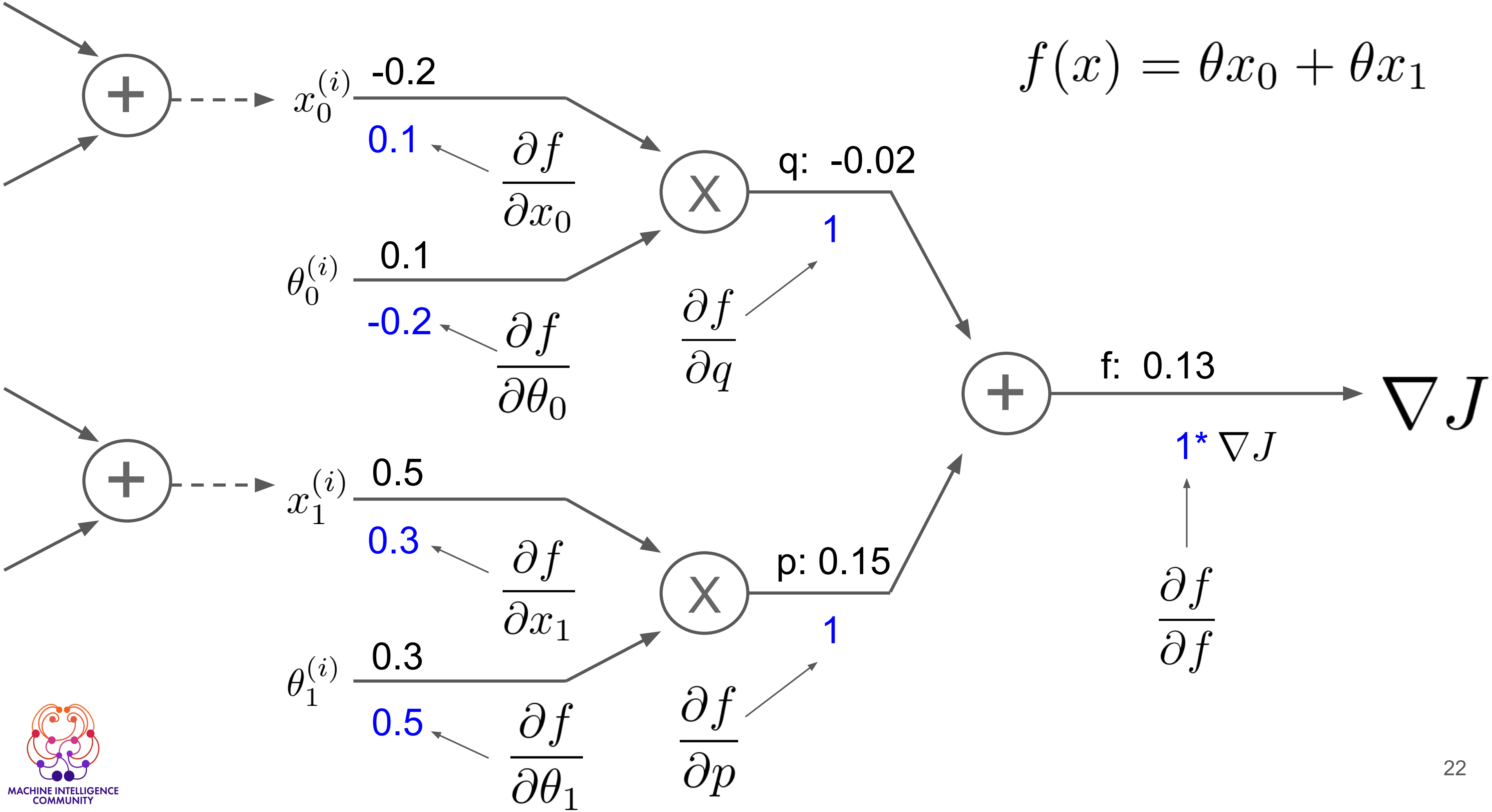
MACHINE INTELLIGENCE COMMUNITY

Derivatives of Computation Graph

$$f(x, y, z) = (x + y)z$$

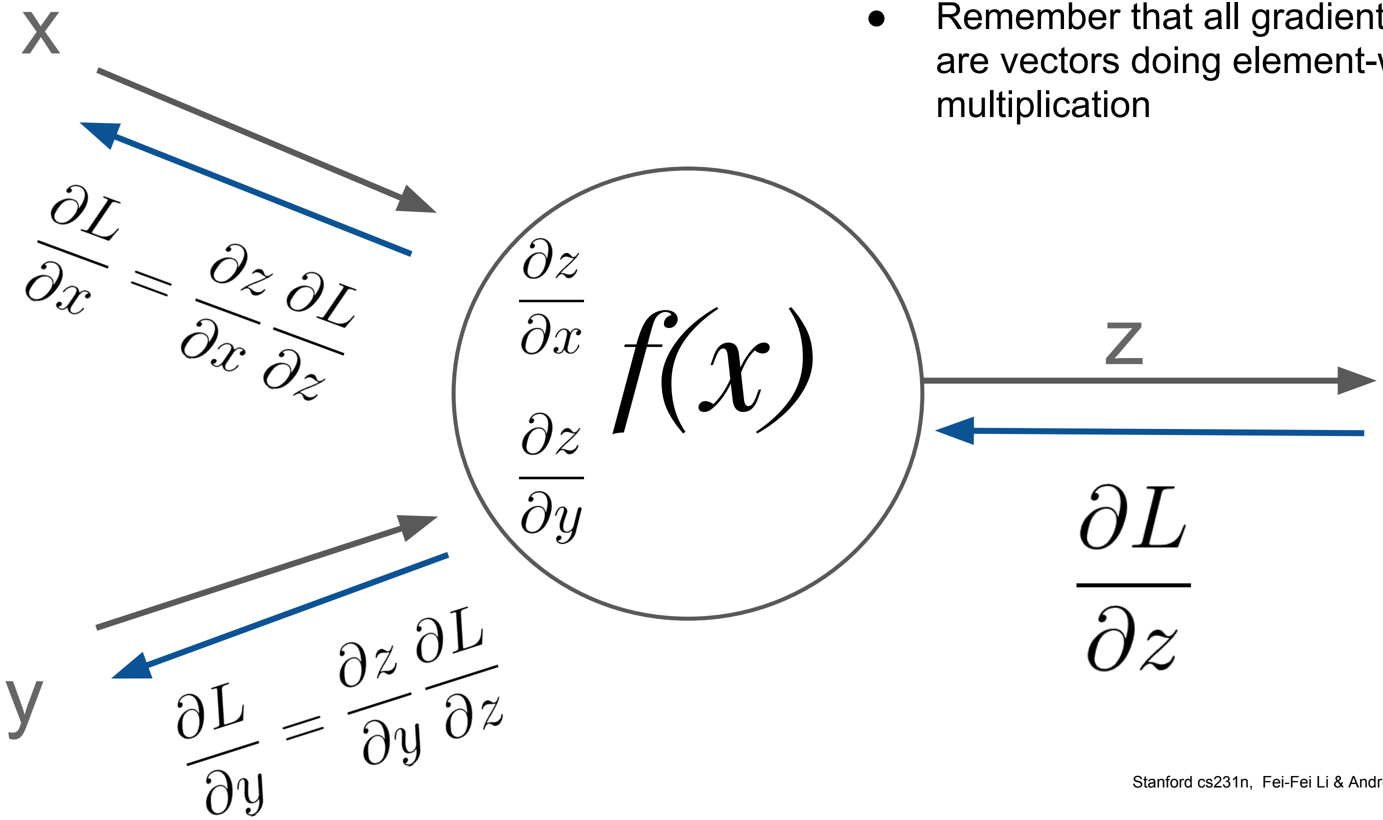


$$f(x) = \theta x_0 + \theta x_1$$



MACHINE INTELLIGENCE COMMUNITY

Local Gradient



- Remember that all gradients are vectors doing element-wise multiplication

MACHINE INTELLIGENCE COMMUNITY

Gradient Descent + Backpropagation

- Now, we can adjust a weight based off our gradient descent calculations!
- θ_j can be updated as its old weight minus the change in cost function in respect to itself

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

MACHINE INTELLIGENCE COMMUNITY

SGD + BP with PYTORCH

```
>> optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
>> optimizer.zero_grad()
>> loss = F.nll_loss(input, target)
>> loss.backward()
>> optimizer.step()
```

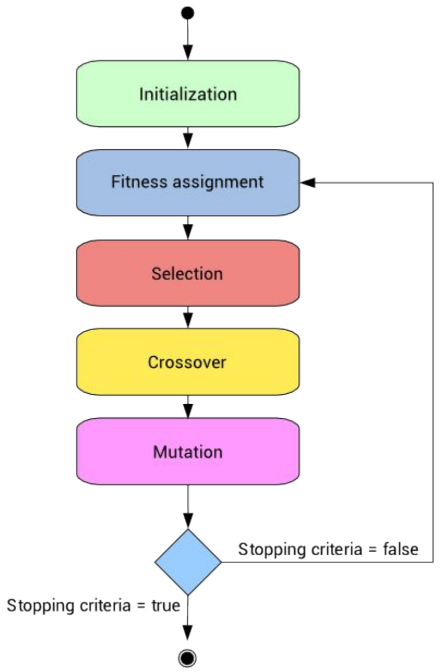
1. Define optimizer
2. **Clear local gradients**
3. Compute loss
4. Backpropagate error
5. Apply gradients to parameters

MACHINE INTELLIGENCE COMMUNITY

http://pytorch.org/docs/master/optim.html?highlight=sgd#torch.optim.SGD

Anything else besides Gradient Descent?

- Yes, but GD is fairly easy and effective
- Another cool option is using evolution models, or “genetic algorithms”
    - Make random mutations to your model to produce many new generation models
    - Test each one's’ “fitness”
    - Kill off the bad performers, evolve the good ones (feature selection)



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

References & Further Reading

[1] Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv preprint arXiv:1609.04747 (2016).

[2] Sun, Xu, et al. "meProp: Sparsified Back Propagation for Accelerated Deep Learning with Reduced Overfitting." arXiv preprint arXiv:1706.06197 (2017).

[3] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[4] Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." arXiv preprint arXiv:1212.5701 (2012).

[5] Du, Simon S., et al. "Gradient Descent Can Take Exponential Time to Escape Saddle Points." arXiv preprint 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.

Shoutout to our sponsors

Boston University Software Application and Innovation Lab

Boston University SPARK!

MACHINE INTELLIGENCE COMMUNITY

Upcoming Events

MIC Paper signup: https://goo.gl/iAm6TL
BUMIC Projects signup: https://goo.gl/GmP9oK

MIT MIC reading group:

Paper: Learning from Simulated and Unsupervised Images through Adversarial Training
Location: MIT 56-154 (building 56, room 154)
Date: 9.21.17 Time: 5 PM

BU MIC reading group:

Paper: SimRank Computation on Uncertain Graphs
Location: BU Hariri Seminar Room
Date: 9.22.17 Time: 7 PM

Next workshop:

Location: BU Hariri Seminar Room
Date: 9.26.17 Time: 7 PM

MACHINE INTELLIGENCE COMMUNITY