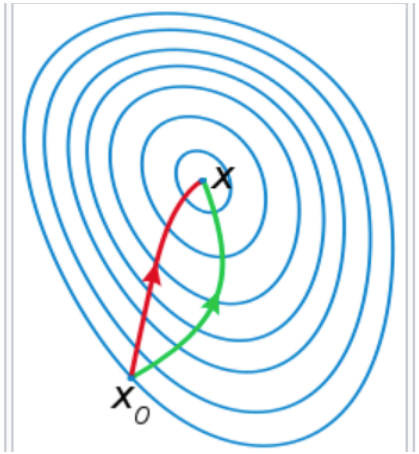
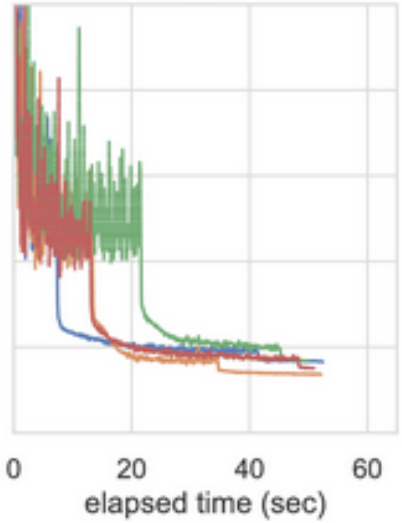
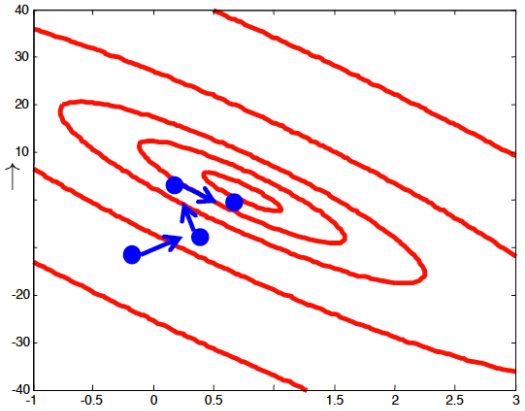
Tufts COMP 135: Introduction to Machine Learning

https://www.cs.tufts.edu/comp/135/2020f/

Stochastic Gradient Descent







Prof. Mike Hughes

Many slides attributable to:
Erik Sudderth (UCI), Emily Fox (UW),
Finale Doshi-Velez (Harvard)
James, Witten, Hastie, Tibshirani (ISL/ESL books)

Objectives Today (day 12)
Stochastic Gradient Descent

- Review: **Gradient Descent**
  - Repeatedly step downhill until converged
- Review: Training Neural Nets with Backprop
  - Backprop = chain rule plus dynamic programming
- L-BFGS : How to step in better direction?
- Stochastic Gradient Descent : How to go fast?

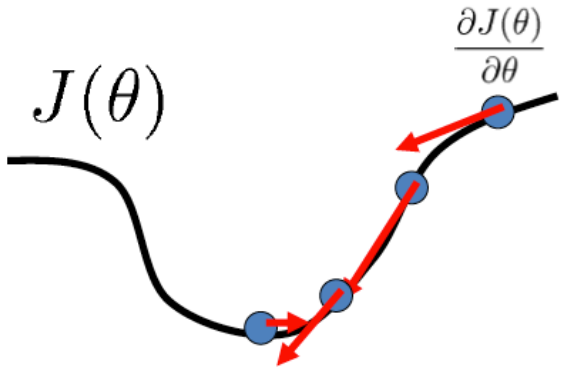
Review: Gradient Descent in 1D

input: initial $\theta \in \mathbb{R}$

input: step size $\alpha \in \mathbb{R}_+$

while not converged:

$$\theta \leftarrow \theta - \alpha \frac{d}{d\theta} J(\theta)$$



Q: Which direction to step?

A: Straight downhill
(steepest descent at current location)

Q: How far to step in that direction?

A: $\alpha \cdot \left|\left| \frac{\partial}{\partial \theta} J \right|\right|$

Step size parameter picked in advance, unaware of current location

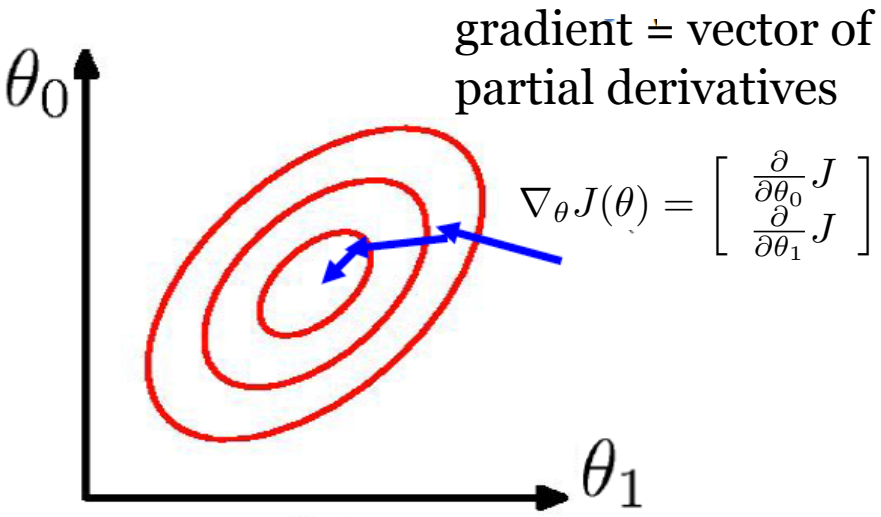
Review: Gradient Descent in 2D+

input: initial $\theta \in \mathbb{R}^D$

input: step size $\alpha \in \mathbb{R}_+$

while not converged:

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta)$$



Q: Which direction to step?

A: Straight downhill
(steepest descent at current location)

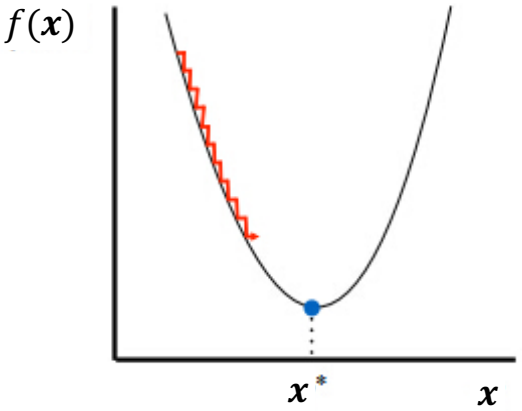
Q: How far to step in that direction?

A: $\alpha \cdot ||\nabla_\theta J(\theta)||$

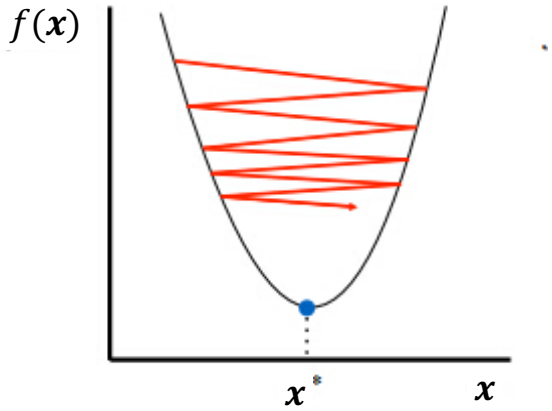
Step size parameter picked in advance, unaware of current location

Review: Step size matters

Even in one dimension, tough to select step size.



Too small: converge very slowly



Too big: overshoot and even diverge

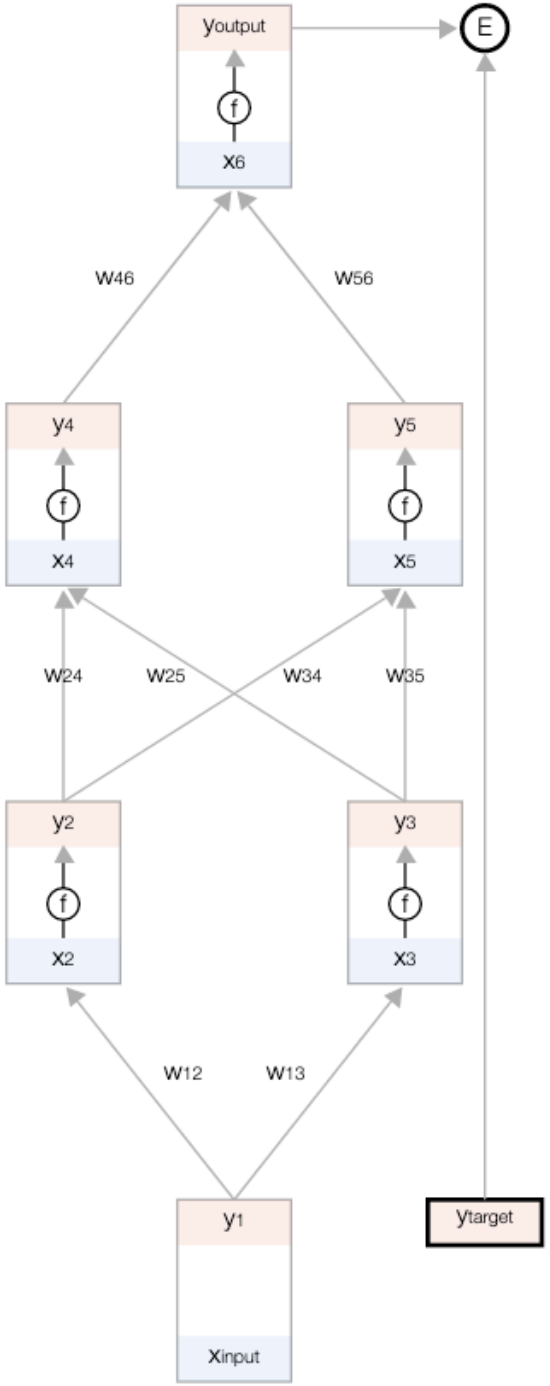
Recommendations

- Try multiple values
- Might need different sizes at different locations

Review: Neural Net as computational graph

2 directions of propagation

Forward: compute loss
Backward: compute grad



Review: Training Neural Nets

Training Objective:

$$\min_{w} \sum_{n=1}^{N} E(y_n, \hat{y}(x_n, w))$$

Gradient Descent Algorithm:

```
w = initialize_weights_at_random_guess(random_state=0)
while not converged:
    total_grad_wrt_w = zeros_like(w)
    for n in 1, 2, … N:
        loss[n], grad_wrt_w[n] = forward_and_backward_prop(x[n], y[n], w)
        total_grad_wrt_w += grad_wrt_w[n]

    w = w - alpha * total_grad_wrt_w
```

$$w_{ij} = w_{ij} - \alpha \frac{dE}{dw_{ij}}$$

How to pick step size reliably? How to go fast on big datasets?

Step size strategy: Slow decay

input: initial $\theta \in \mathbb{R}$

input: initial step size $\alpha_0 \in \mathbb{R}_+$

while not converged:

$$\theta \leftarrow \theta - \alpha_t \nabla_\theta J(\theta)$$

$$\alpha_t \leftarrow \text{decay}(\alpha_0, t)$$

t : number of steps

$$t \leftarrow t + 1$$

Linear decay

$$\frac{\alpha_0}{kt}$$

Exponential decay

$$\alpha_0 e^{-kt}$$

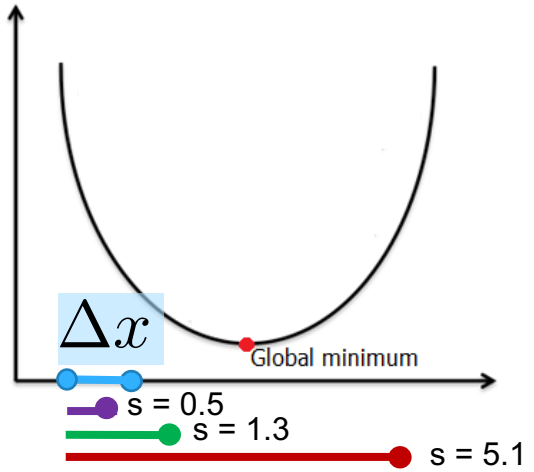
Often helpful, requires tuning and hard to get right!

Q: How far to step? A: Line search

Find good step size for current location

Goal: $\min_x f(x)$

Step Direction: $\Delta x = -\nabla_x f(x)$



Possible step lengths

Search for the best scalar s >= 0, such that:

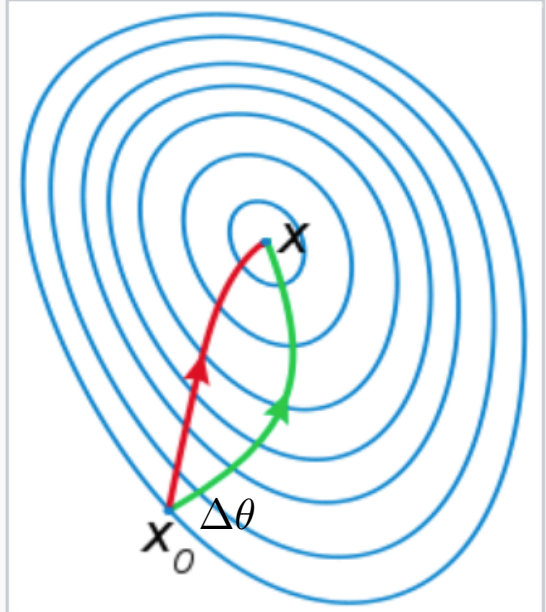
$$s^* = \arg\min_{s \geq 0} f(x + s\Delta x)$$

In Python code: **scipy.optimize.line_search**

Can be expensive, but often worth it

Q: Better direction to step than straight downhill?

A: Yes. Modify direction using **second-order derivative**.



A comparison of gradient descent (green) and Newton's method (red) for minimizing a function (with small step sizes). Newton's method uses curvature information (i.e. the second derivative) to take a more direct route.

$$\min_{\theta} J(\theta)$$

	1st order only decent direction	Using 2nd order Newton descent direction
1-D	$\Delta\theta = -J'(\theta)$	$\Delta\theta = -\frac{1}{J''(\theta)} J'(\theta)$
2-D+	$\Delta\theta = -\nabla_\theta J(\theta)$	$\Delta\theta = -H(\theta)^{-1} \nabla_\theta J(\theta)$

Hessian matrix for J
H is a D x D matrix
All second-order partial derivatives

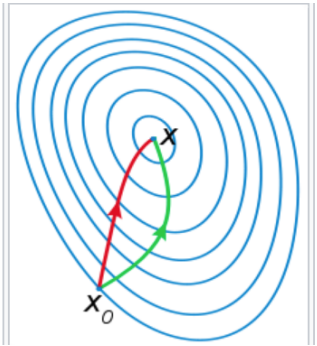
L-BFGS: Smarter Gradient Descent

scipy.optimize.fmin_l_bfgs_b

L-BFGS : Limited Memory Broyden–Fletcher–Goldfarb–Shanno (BFGS)

Approximate second order method

- Computes first-order gradient vector exactly on provided training dataset
- Computes efficient approximation of Hessian via recent history of steps



Q: Which direction to step?

A: Downhill, adjusted by curvature at current location

Q: How far to step in that direction?

A: Efficient line search
Step size adjusted to current location (as implemented in SciPy)

Objectives Today (day 12)
Stochastic Gradient Descent

- Review: **Gradient Descent**
  - Repeatedly step downhill until converged
- Review: Training Neural Nets with Backprop
  - Backprop = chain rule plus dynamic programming
- L-BFGS : How to step in better direction?
- Stochastic Gradient Descent : How to go **fast**?

Stochastic Estimate of Loss Function

- Standard "full-dataset" objective

$$\mathcal{L}(w) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n(x_n, y_n, w)$$

- Rewrite as an "expected value"

$$\mathcal{L}(w) = \mathbb{E}_{x_i, y_i \sim \text{Unif}(\{x_n, y_n\}_{n=1}^N)} \left[\mathcal{L}_i(x_i, y_i, w) \right]$$

Empirical distribution over our N training examples

Each index *i* selected with probability 1/N

Stochastic Estimate of Loss Function

- Standard "full-dataset" objective

$$\mathcal{L}(w) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n(x_n, y_n, w)$$

- Rewrite as an "expected value"

$$\mathcal{L}(w) = \mathbb{E}_{x_i, y_i \sim \text{Unif}(\{x_n, y_n\}_{n=1}^N)} \left[\mathcal{L}_i(x_i, y_i, w) \right]$$

- Approximate with one randomly-drawn sample

$$\mathcal{L}(w) \approx \mathcal{L}_i(x_i, y_i, w) \qquad x_i, y_i \sim \text{Unif}(\{x_n, y_n\}_{n=1}^N)$$

Each index *i* selected with probability 1/N

Stochastic Estimate of Gradient

- Standard "full-dataset" gradient

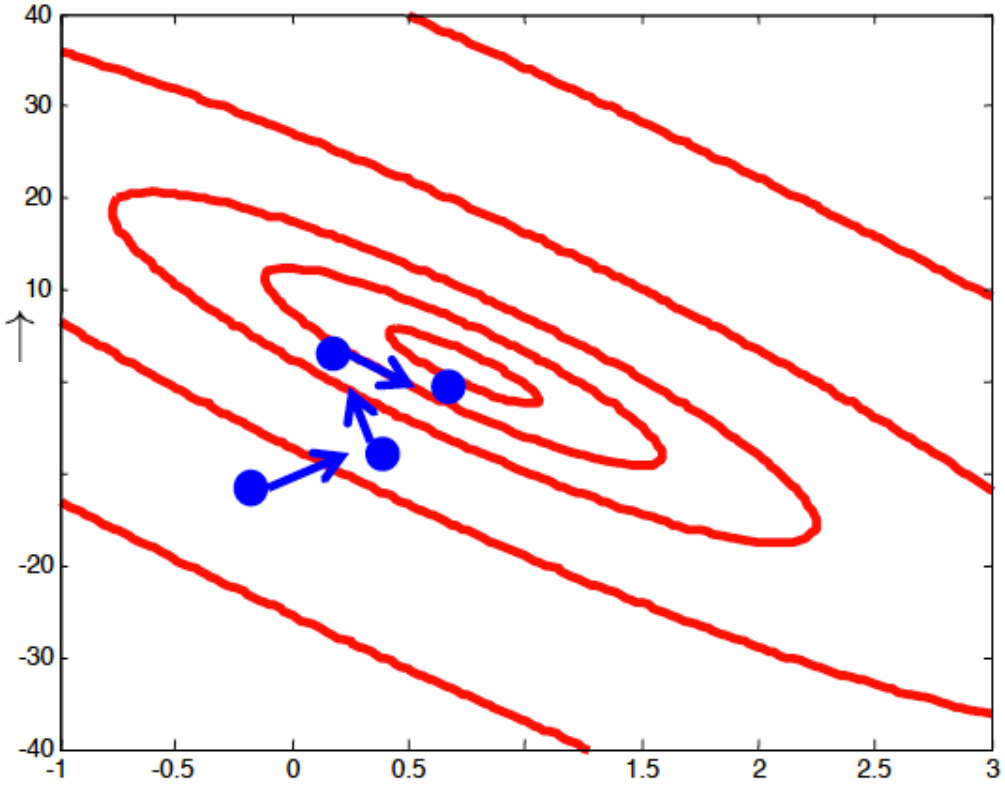
$$\nabla_w \mathcal{L}(w) = \frac{1}{N} \sum_{n=1}^{N} \nabla_w \mathcal{L}_n(x_n, y_n, w)$$

- Approximate with one randomly-drawn sample

$$\nabla_w \mathcal{L}(w) \approx \nabla_w \mathcal{L}_i(x_i, y_i, w) \qquad x_i, y_i \sim \text{Unif}(\{x_n, y_n\}_{n=1}^{N})$$

Each index *i* selected with probability 1/N

Gradient Descent using Noisy Estimates of the "True" Gradient



Intuition

As long as each noisy step takes us in a **direction that is correct on average**, we will over many steps make progress in minimizing the loss.

Formal guarantees

Our Monte Carlo estimate of gradient is unbiased, so its expected value is exactly equal to the true whole-dataset gradient

Stochastic gradient descent (SGD)

using one example at a time

input: initial $w \in \mathbb{R}$

input: step size $\alpha \in \mathbb{R}_+$

while not converged:

$$\{x_i, y_i\} \sim \text{Unif}(\{x_n, y_n\}_{n=1}^N)$$

$$w \leftarrow w - \alpha \nabla_w \mathcal{L}(x_i, y_i, w)$$

Should we only use one example i to estimate gradient?

SGD with minibatches of size B

input: initial $w \in \mathbb{R}$

input: step size $\alpha \in \mathbb{R}_+$

while not converged:

$$\{x_b, y_b\}_{b=1}^{B} \sim \text{Unif}(\{x_n, y_n\}_{n=1}^{N}, \quad \text{size} = B, \quad \text{replace} = \text{False})$$

$$w \leftarrow w - \alpha \cdot \frac{1}{B} \sum_{i=1}^{B} \nabla_w \mathcal{L}(x_i, y_i, w)$$

B = 1 recovers previous slide. B = N recovers standard GD.
In between: **trade off** *quality of estimate* with *cost of estimate*

Objectives Today (day 12)
Stochastic Gradient Descent

- Review: Gradient Descent
  - Repeatedly step downhill until converged
- Review: Training Neural Nets with Backprop
  - Backprop = chain rule plus dynamic programming
- Line Search: How to take step of good size?
- L-BFGS : How to step in better direction?
- Stochastic Gradient Descent : How to go **fast**?

Breakout to Lab

Warning: Notation can be confusing

- Alpha in these slides refers to step size (aka learning rate)
- In sklearn's MLPClassifier, alpha refers to a *different* hyperparameter: the scalar strength of a small L2 penalty on the magnitudes of the weights
- To set step size in sklearn:

```
learning_rate_init=0.5
```