Machine Learning Department
School of Computer Science
Carnegie Mellon University

Unconstrained Optimization

Matt Gormley

Lecture 12

Dec. 2, 2018

Reminders

- Homework C: Data Structures
  - – Out: Mon, Nov. 26
  - – Due: Mon, Dec. 3 at 11:59pm
- Quiz B: Computation; Programming & Efficiency
  - – Wed, Dec. 5, in-class
  - – Covers Lectures 7 – 12
- Homework D: Inference & Optimization
  - – Out: Mon, Dec. 3
  - – Due: Fri, Dec. 7 at 11:59pm

Q&A

GRADIENT DESCENT

Motivation: Gradient Descent

Cases to consider gradient descent:

1. What if we **can not** find a closed-form solution?
2. What if we **can**, but it's inefficient to compute?
3. What if we **can**, but it's numerically unstable to compute?

Motivation: Gradient Descent

To solve the Ordinary Least Squares problem we compute:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmin}} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(y^{(i)} - (\boldsymbol{\theta}^T\mathbf{x}^{(i)}))^2$$

$$= (\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{Y})$$

The resulting shape of the matrices:

$$\underbrace{(\underbrace{\mathbf{X}^T}_{M\times N}\ \underbrace{\mathbf{X}}_{N\times M})^{-1}}_{M\times M}\ \underbrace{(\underbrace{\mathbf{X}^T}_{M\times N}\ \underbrace{\mathbf{Y}}_{N\times 1})}_{M\times 1}$$

Background: Matrix Multiplication Given matrices $\mathbf{A}$ and $\mathbf{B}$

- If $\mathbf{A}$ is $q \times r$ and $\mathbf{B}$ is $r \times s$, computing $\mathbf{AB}$ takes $O(qrs)$
- If $\mathbf{A}$ and $\mathbf{B}$ are $q \times q$, computing $\mathbf{AB}$ takes $O(q^{2.373})$
- If $\mathbf{A}$ is $q \times q$, computing A^{-1} takes $O(q^{2.373})$.

Computational Complexity of OLS:

$\mathbf{X}^T\mathbf{X}$	$O(M^2N)$
$(\quad)^{-1}$	$O(M^{2.373})$
$\mathbf{X}^T\mathbf{Y}$	$O(MN)$
$(\quad)^{-1}(\quad)$	$O(M^2)$
total	$O(M^2N + M^{2.373})$

Linear in # of examples, N
Polynomial in # of features, M

Regularized Regret

Chalkboard

- Regularized loss minimization
- L1 vs. L2 regularizers
- Learning linear models by optimization
- Example loss functions:
  - zero-one
  - logistic (log-loss)
  - exponential
  - squared
  - hinge
  - perceptron

Losses for Linear Models

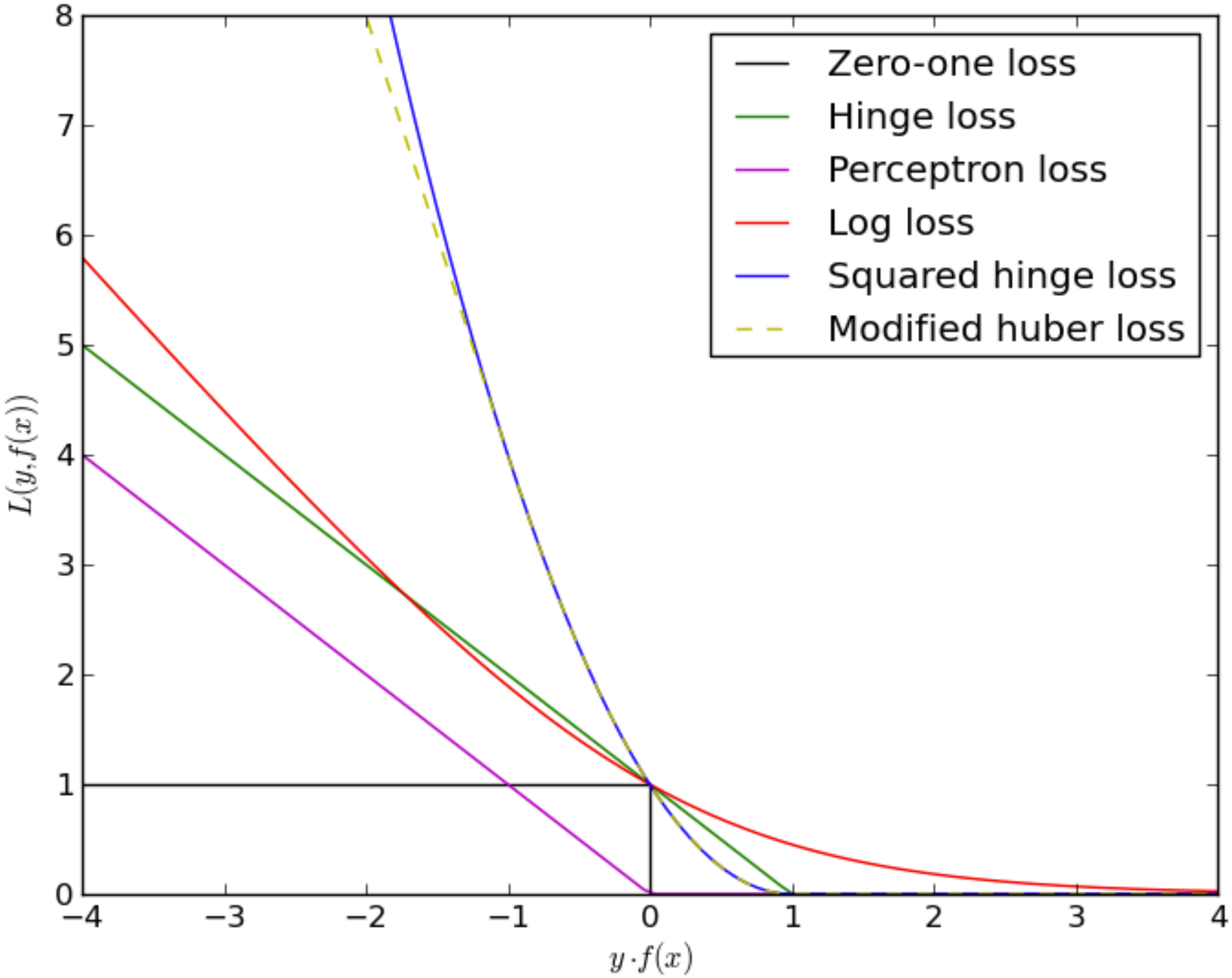
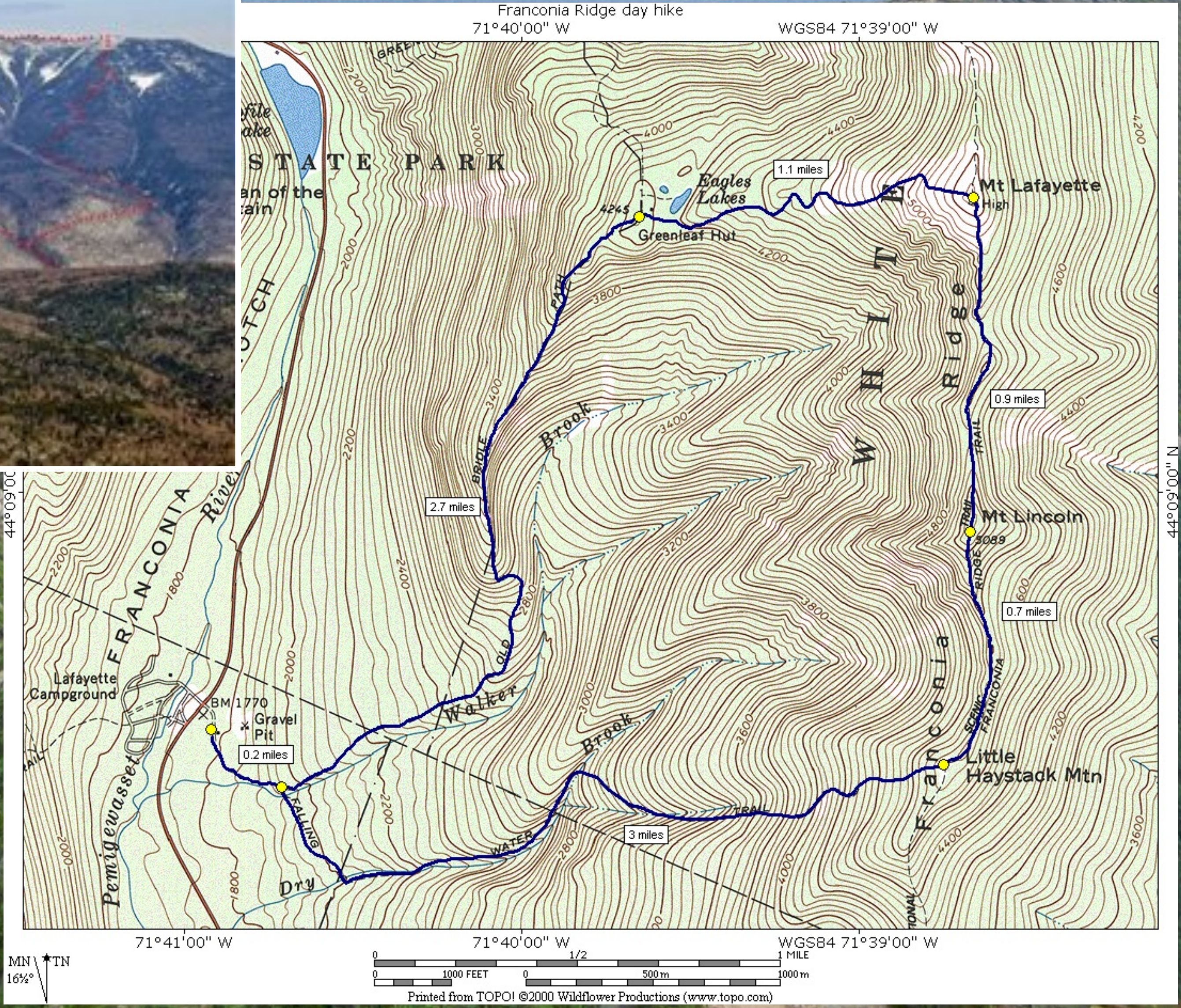


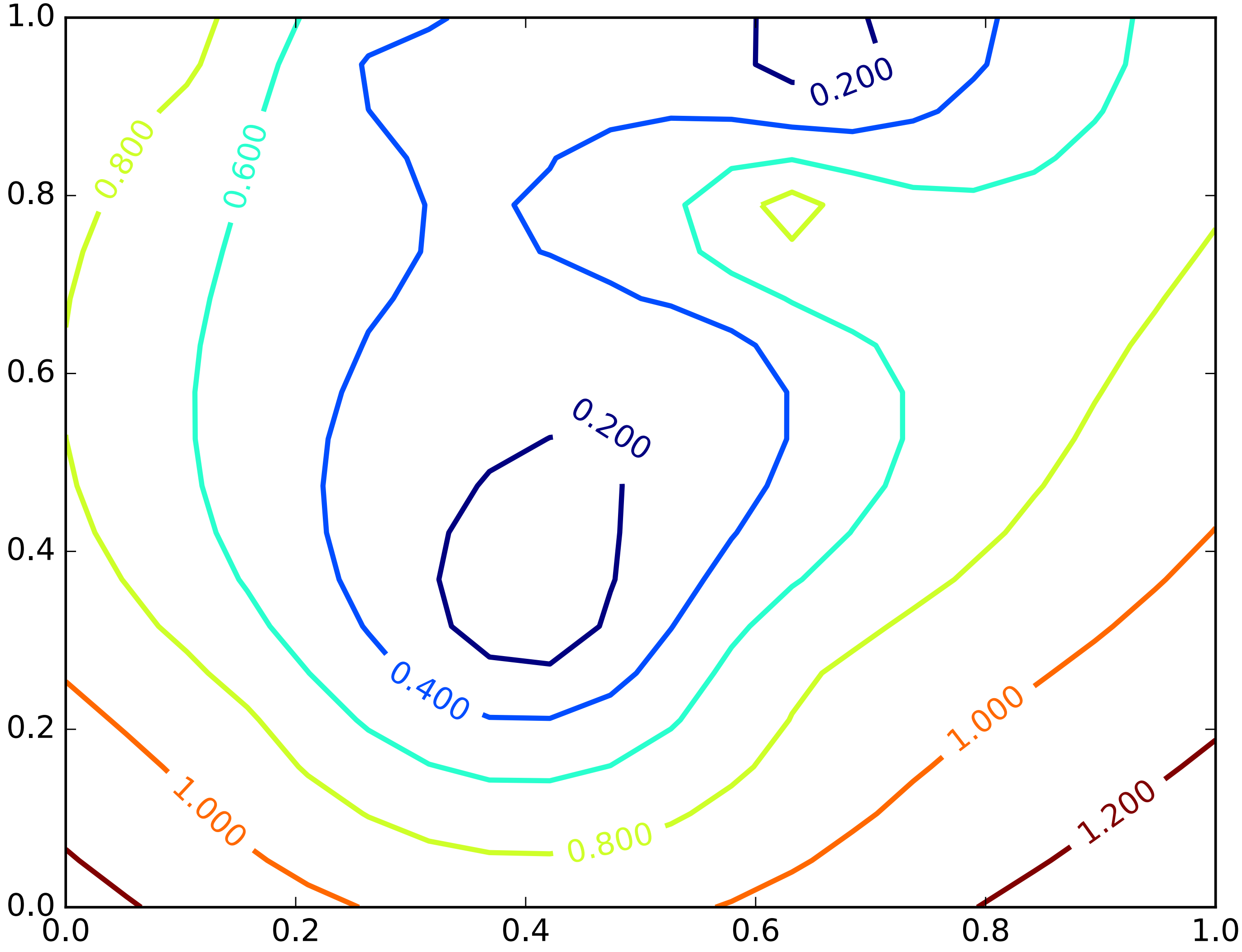
Figure from https://scikit-learn.org/stable/auto_examples/linear_model/plot_sgd_loss_functions.html

Topographical Maps

Topographical Maps

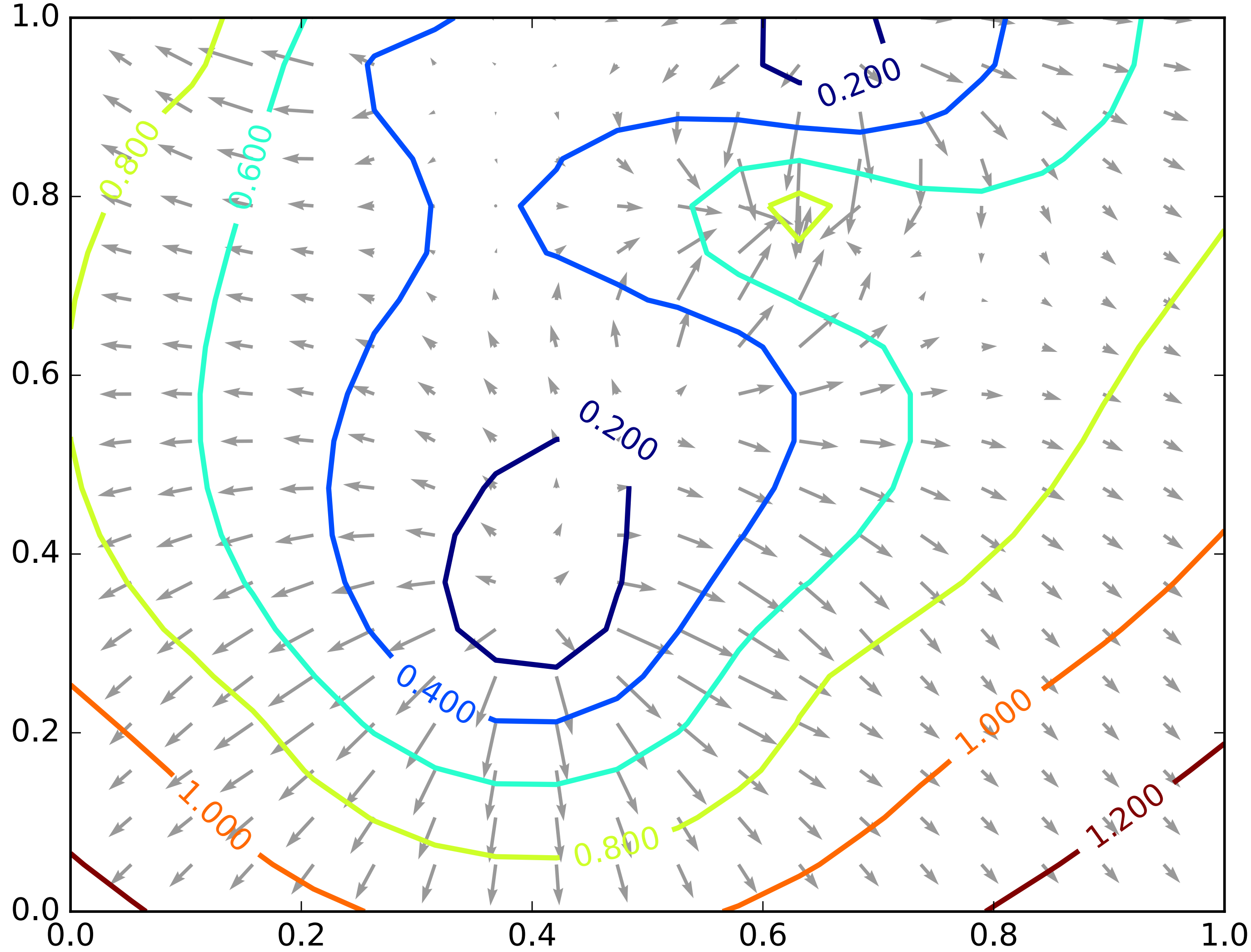


Gradients



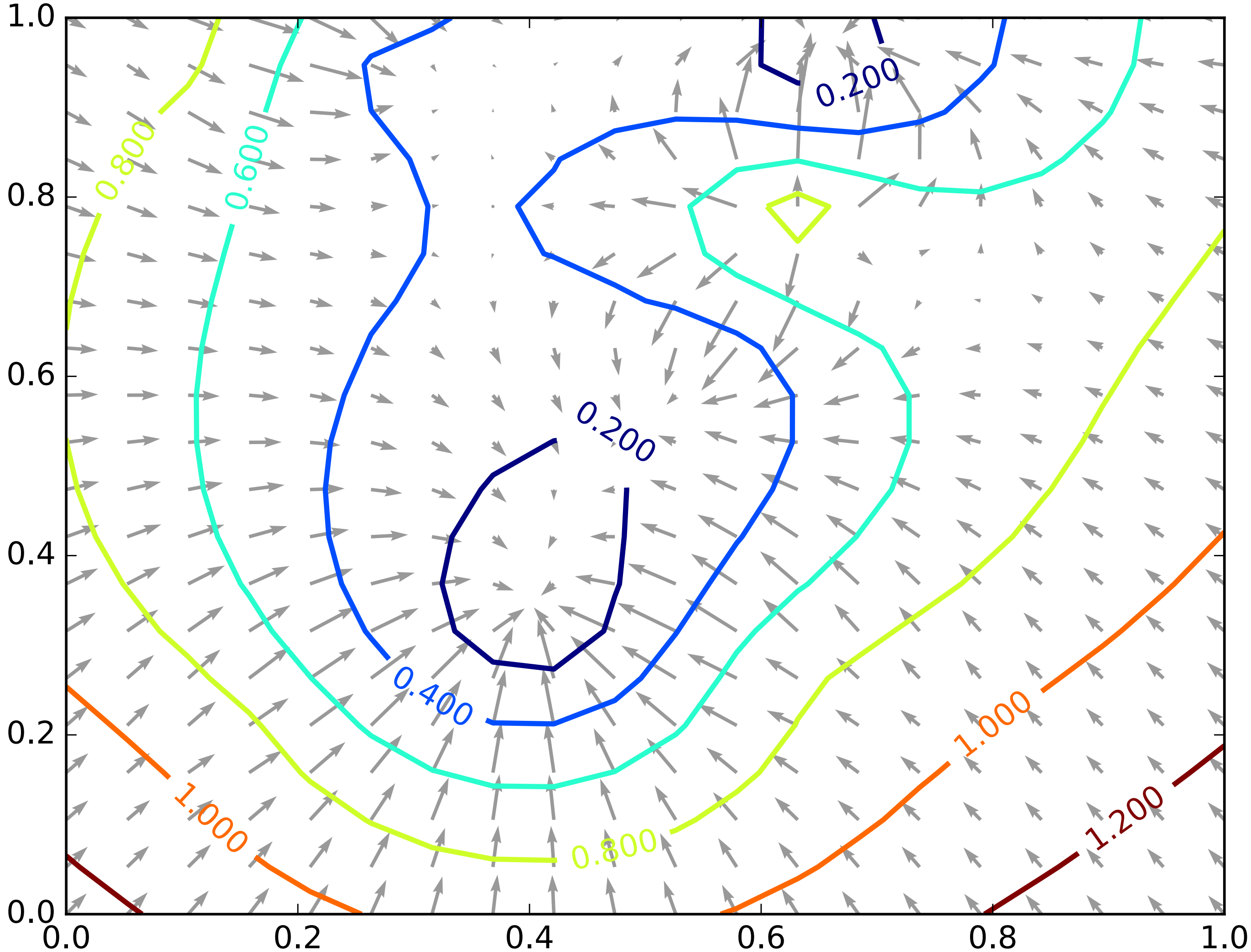
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0.200
0.200
0.400
0.600
0.800
0.800
1.000
1.000
1.200

Gradients



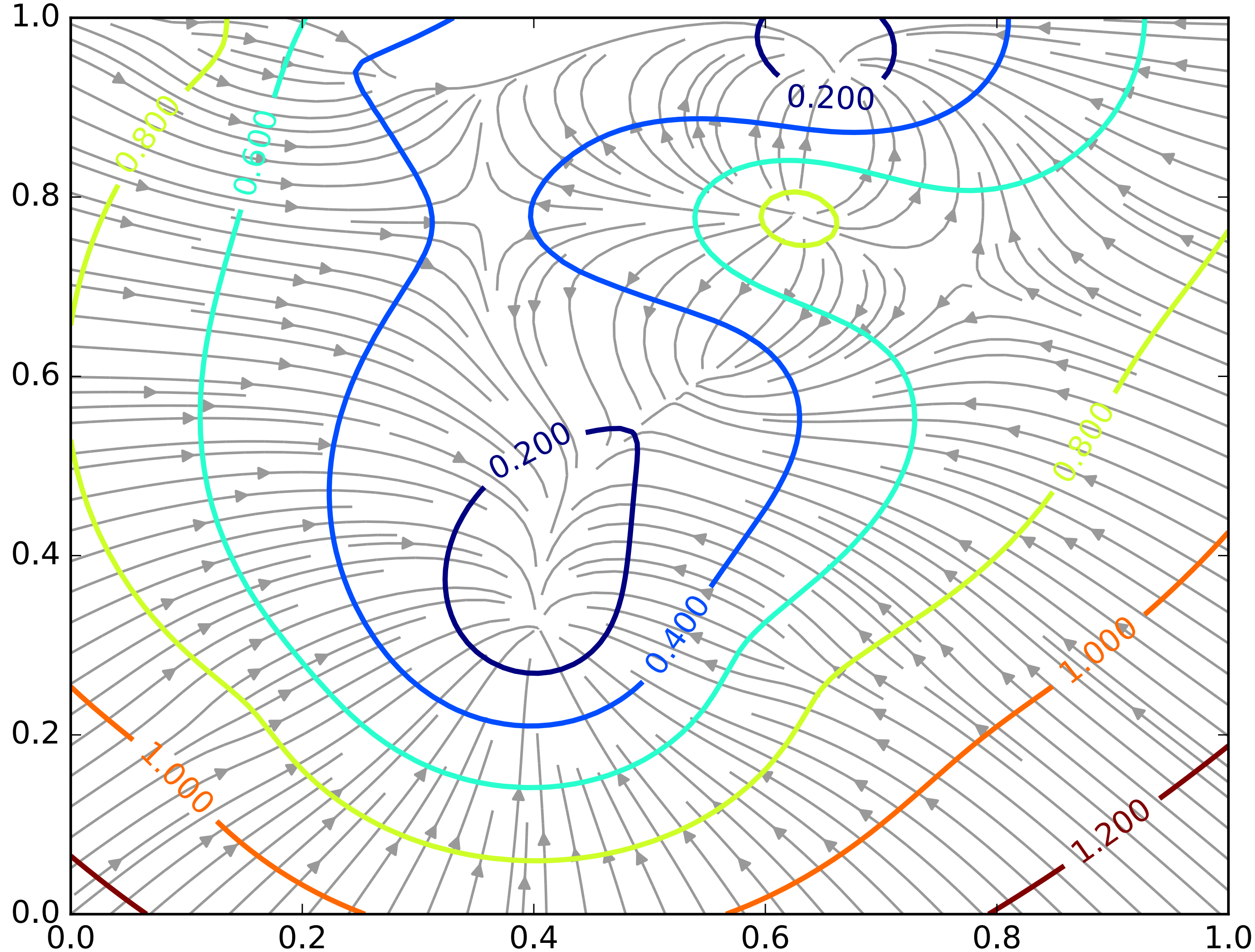
These are the **gradients** that
Gradient **Ascent** would follow.

(Negative) Gradients



These are the **negative** gradients that
Gradient **Descent** would follow.

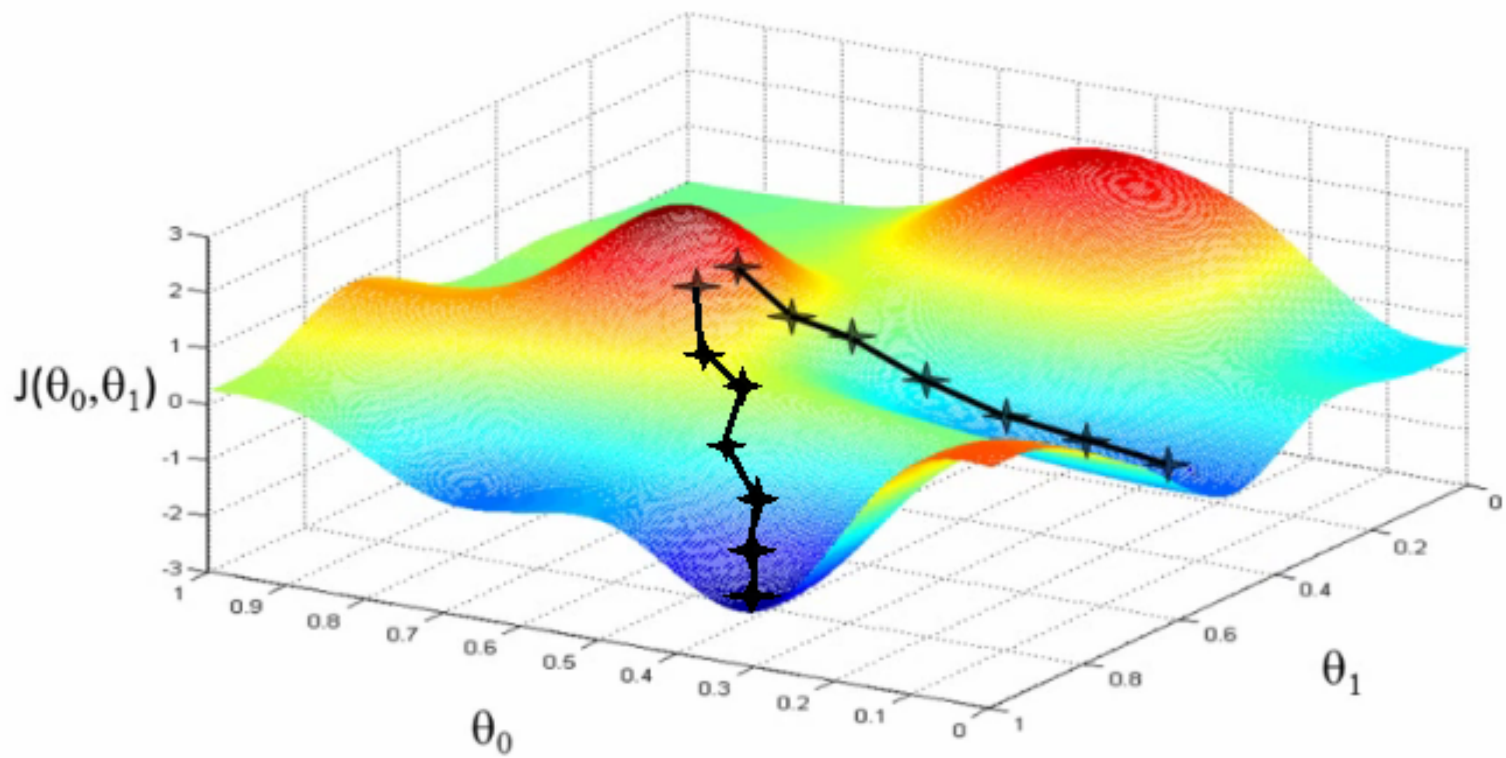
(Negative) Gradient *Paths*



Shown are the **paths** that Gradient Descent would follow if it were making **infinitesimally small steps.**

Pros and cons of gradient descent

- Simple and often quite effective on ML tasks
- Often very scalable
- Only applies to smooth functions (differentiable)
- Might find a local minimum, rather than a global one



Slide courtesy of William Cohen

Gradient Descent

Chalkboard

- Gradient Descent Algorithm
- Details: starting point, stopping criterion, line search

Gradient Descent

Algorithm 1 Gradient Descent

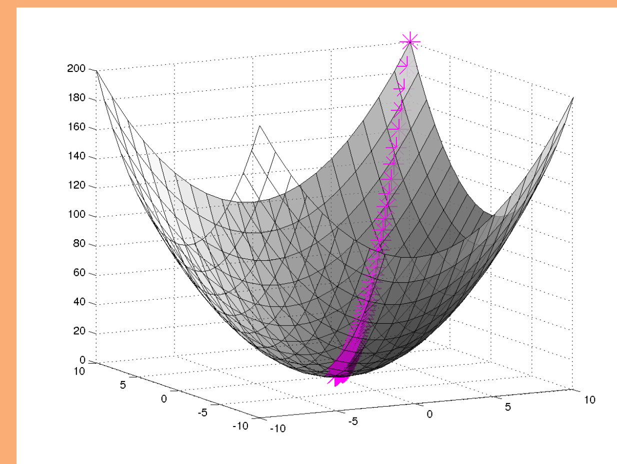
1: **procedure** $\text{GD}(\mathcal{D}, \boldsymbol{\theta}^{(0)})$

2: $\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$

3: $\quad$ **while** not converged **do**

4: $\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

5: $\quad$ **return** $\boldsymbol{\theta}$



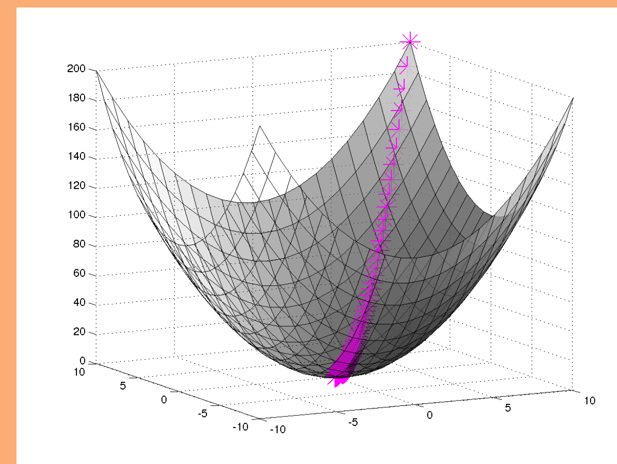
In order to apply GD to Linear Regression all we need is the **gradient** of the objective function (i.e. vector of partial derivatives).

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{d}{d\theta_1} J(\boldsymbol{\theta}) \\ \frac{d}{d\theta_2} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{d}{d\theta_M} J(\boldsymbol{\theta}) \end{bmatrix}$$

Gradient Descent

Algorithm 1 Gradient Descent

$$
\begin{array}{l}
\textbf{procedure } \mathrm{GD}(\mathcal{D}, \boldsymbol{\theta}^{(0)}) \\
\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)} \\
\quad \textbf{while } \text{not converged } \textbf{do} \\
\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \\
\quad \textbf{return } \boldsymbol{\theta}
\end{array}
$$



There are many possible ways to detect **convergence.** For example, we could check whether the L2 norm of the gradient is below some small tolerance.

$$||\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})||_2 \leq \epsilon$$

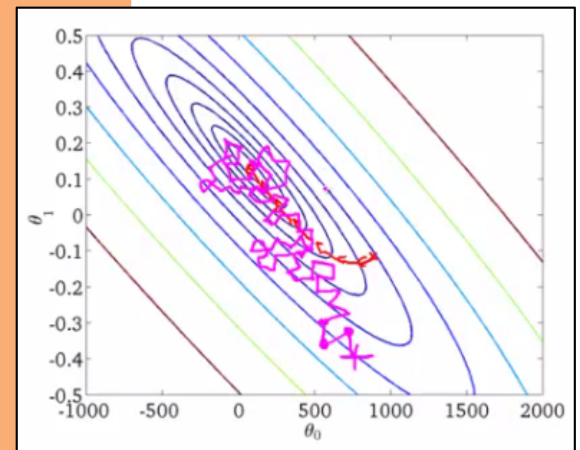
Alternatively we could check that the reduction in the objective function from one iteration to the next is small.

STOCHASTIC GRADIENT DESCENT

Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent (SGD)

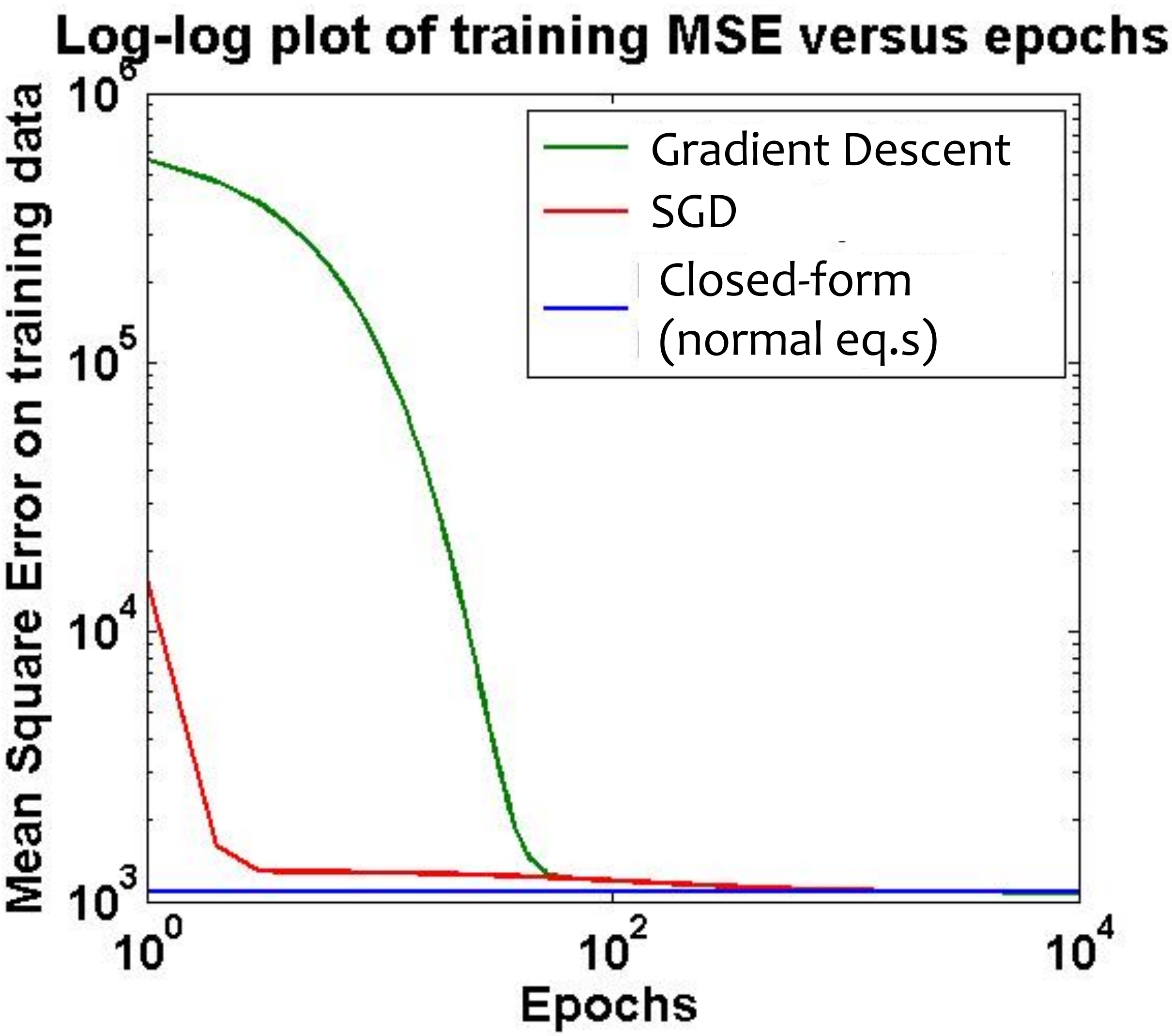
1: **procedure** SGD($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
2: $\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
3: $\quad$**while** not converged **do**
4: $\quad\quad$**for** $i \in \text{shuffle}(\{1, 2, \ldots, N\})$ **do**
5: $\quad\quad\quad\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda \nabla_{\boldsymbol{\theta}} J^{(i)}(\boldsymbol{\theta})$
6: $\quad$**return** $\boldsymbol{\theta}$



We need a per-example objective:

$$\text{Let } J(\boldsymbol{\theta}) = \sum_{i=1}^{N} J^{(i)}(\boldsymbol{\theta})$$

Convergence Curves

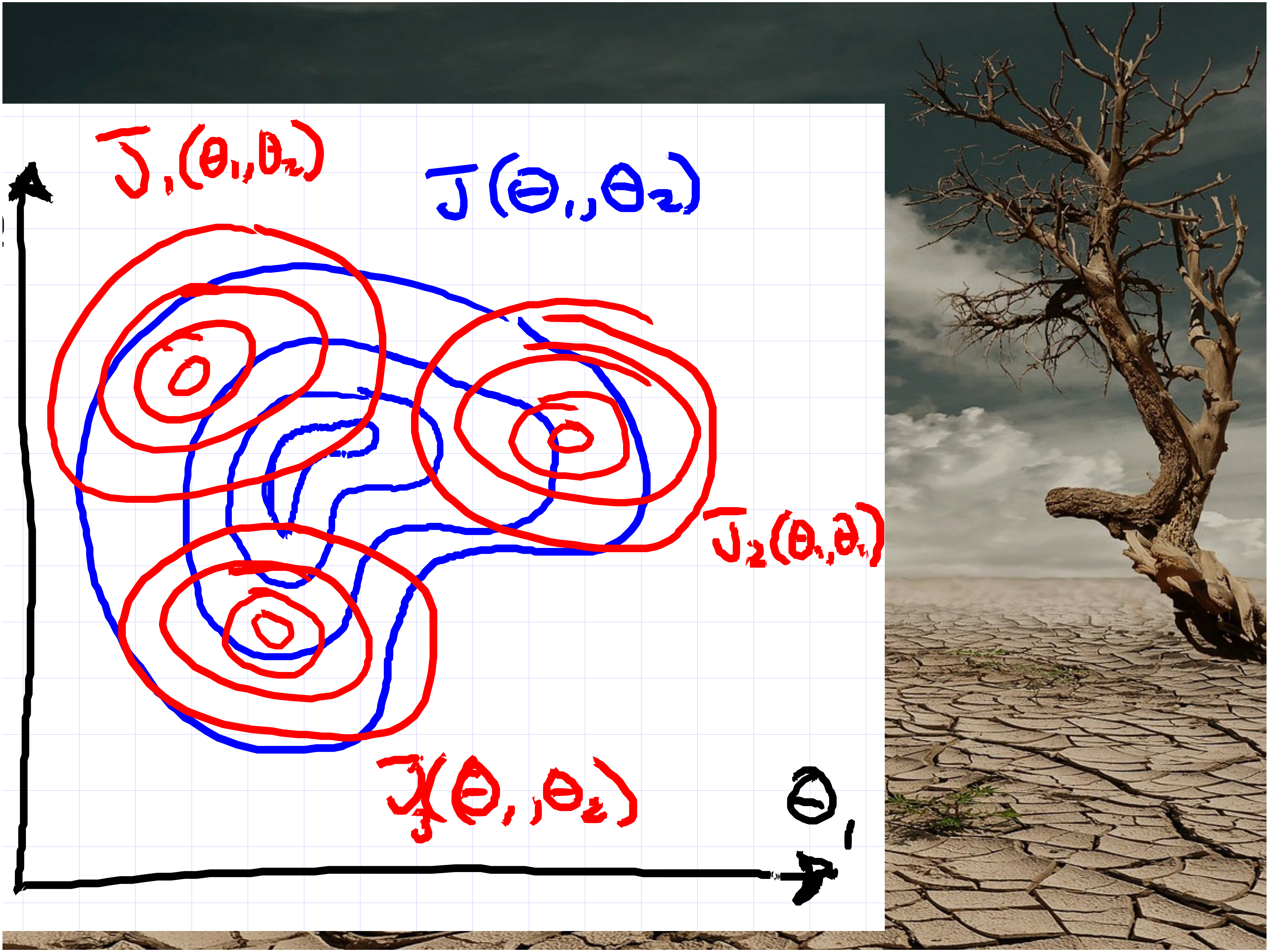


- *Def*: an **epoch** is a single pass through the training data

1. For GD, only **one update** per epoch
2. For SGD, ***N* updates** per epoch
   N = (# train examples)

- SGD reduces MSE much more rapidly than GD
- For GD / SGD, training MSE is initially large due to uninformed initialization

Figure adapted from Eric P. Xing


$J_1(\theta_1,\theta_2)$
$J(\theta_1,\theta_2)$
$J_2(\theta_1,\theta_2)$
$J_3(\theta_1,\theta_2)$
θ_1

Expectations of Gradients

$$\frac{dJ(\vec{\theta})}{d\theta_j} = \frac{1}{N}\sum_{i=1}^{N} \frac{d}{d\theta_j}\left(J_i(\vec{\theta})\right)$$

$$\nabla J(\vec{\theta}) = \begin{array}{c c} & \begin{bmatrix} \vdots \\ \underline{\qquad} \\ \vdots \end{bmatrix} \\ j^{th} & \end{array} = \frac{1}{N}\sum_{i=1}^{N} \nabla J_i(\vec{\theta})$$

Recall: for any discrete r.v. X

$$E_X[f(X)] \triangleq \sum_x P(X=x) f(x)$$

Q: What is the expected value of a randomly chosen $\nabla J_i(\theta)$?

Let $I \sim \text{Uniform}(\{1, \ldots, N\})$

$\Rightarrow P(I=i) = \frac{1}{N}$ if $i \in \{1 \ldots N\}$

$$E_I[\nabla J_I(\vec{\theta})] = \sum_{i=1}^{N} P(I=i)\nabla J_i(\vec{\theta})$$

$$= \frac{1}{N}\sum_{i=1}^{N} \nabla J_i(\vec{\theta})$$

$$= \nabla J(\vec{\theta})$$

Convergence of Optimizers

Convergence Analysis:

true unknown min

Def: convergence is when $J(\vec{\theta}) - J(\vec{\theta}^*) < \epsilon$

not covered

Methods	Steps to Converge	Computation per iteration
Newton's Method	$O(\ln \ln 1/\epsilon)$	$\nabla J(\theta)$ $\nabla^2 J(\theta) \leftarrow O(NM^2)$
GD	$O(\ln 1/\epsilon)$	$\nabla J(\theta) \leftarrow O(NM)$
SGD	$O(1/\epsilon)^*$	$\nabla J_i(\theta) \leftarrow O(M)$

* "almost sure" convergence

lots of caveats and conditions

way less computation

Takeaway: SGD has much slower asymptotic convergence. but is often faster in practice.

ADAGRAD

Comparison of Algorithms

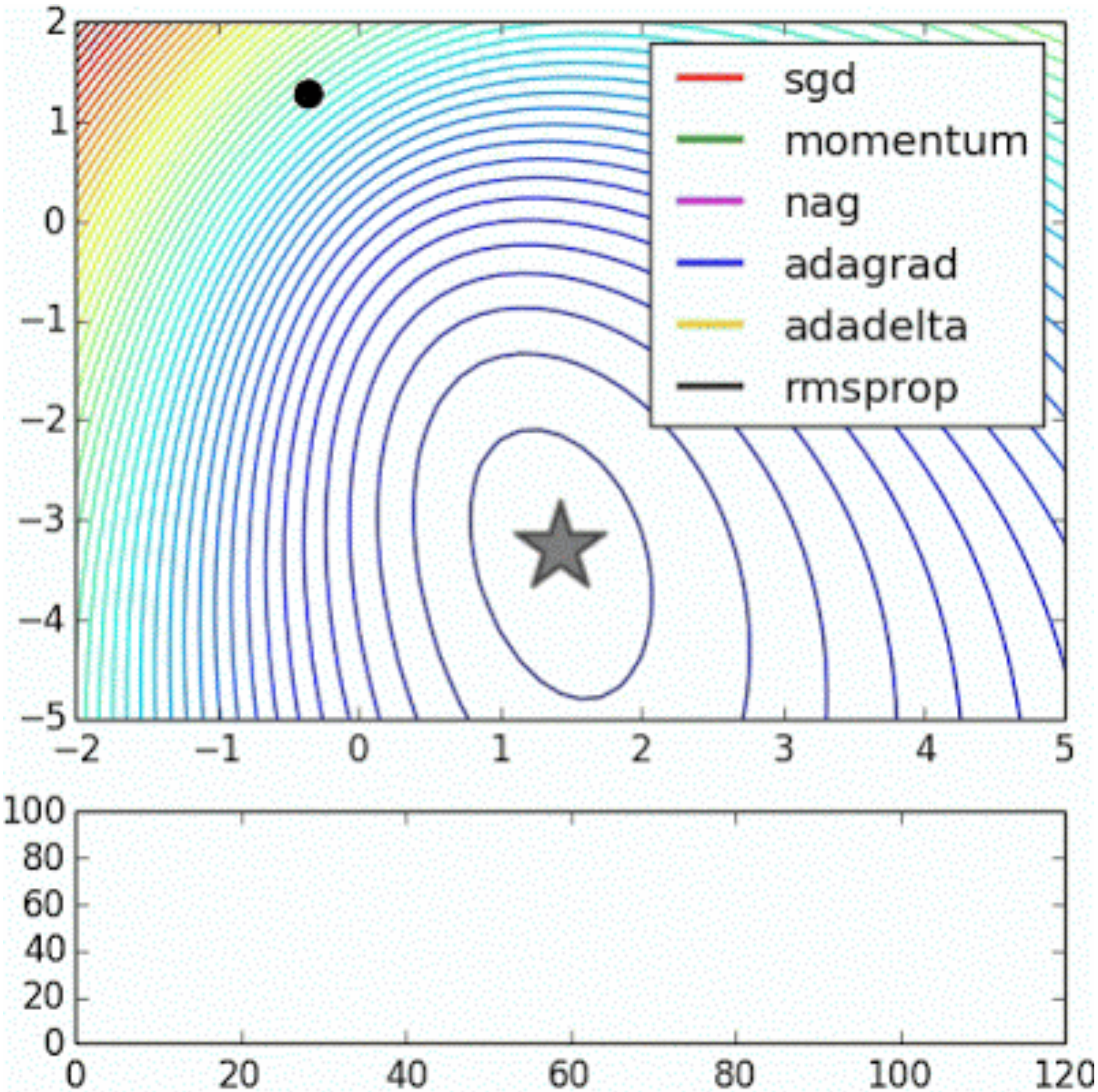


Figure from Alec Radford via http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Comparison of Algorithms

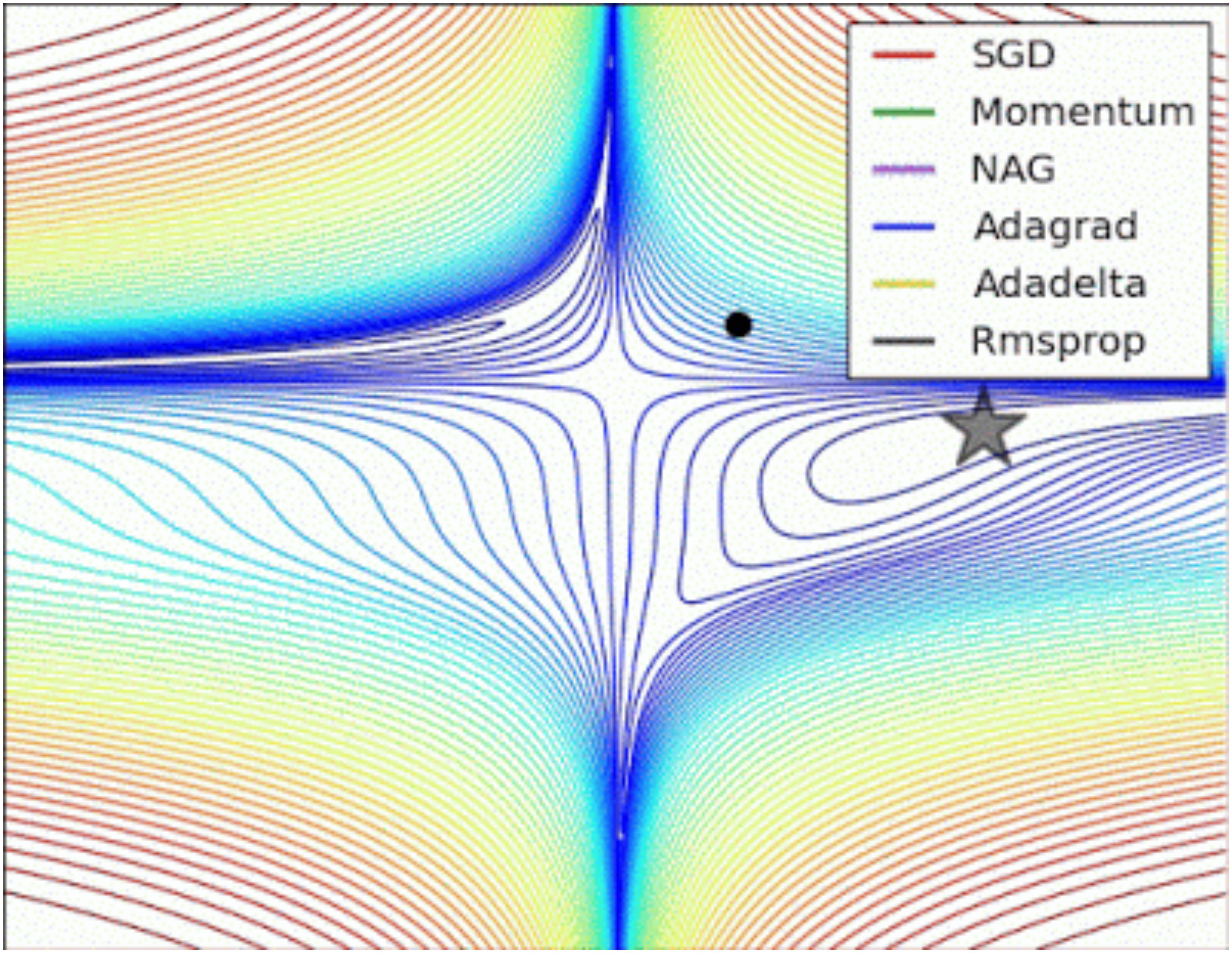


Figure from Alec Radford via http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Comparison of Algorithms

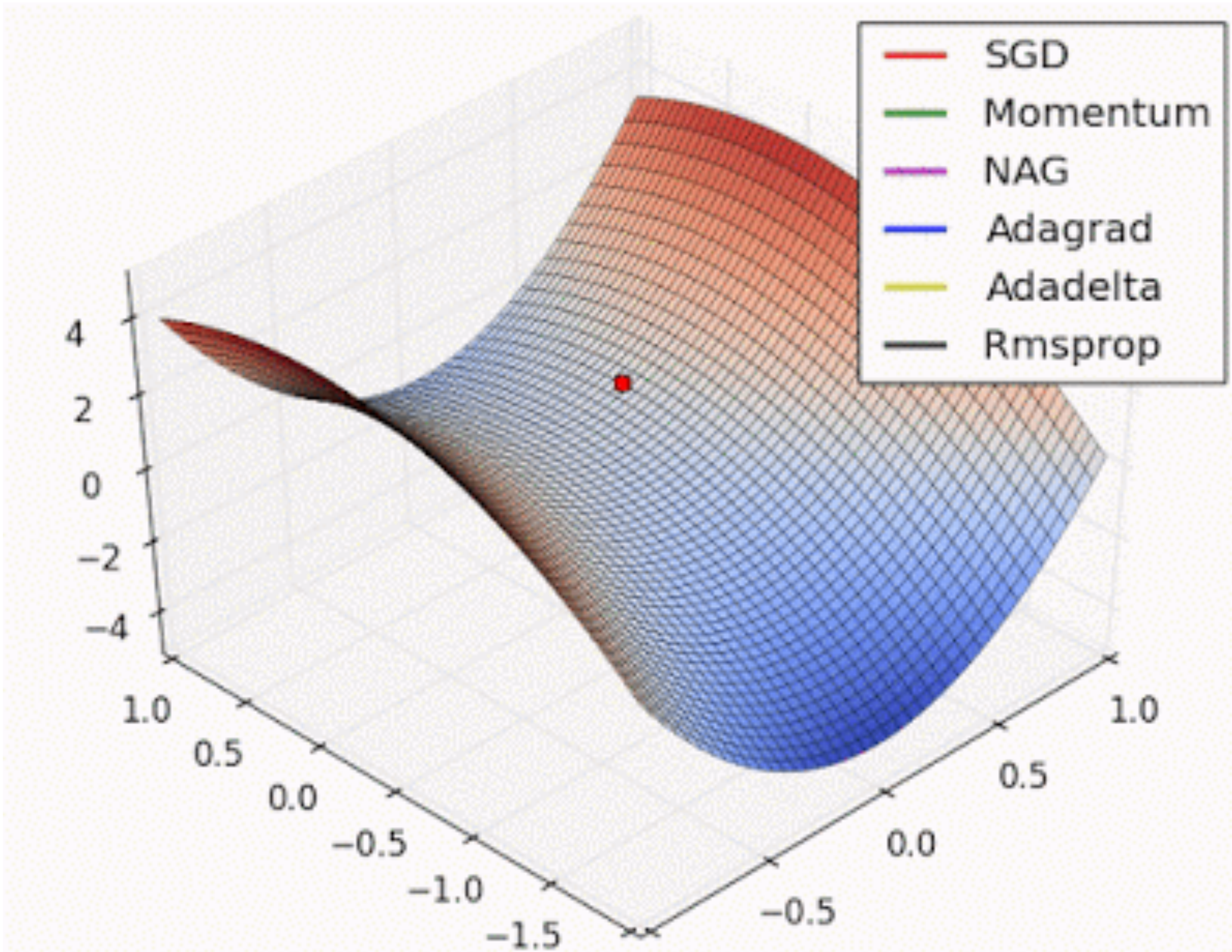


Figure from Alec Radford via http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Comparison of Algorithms

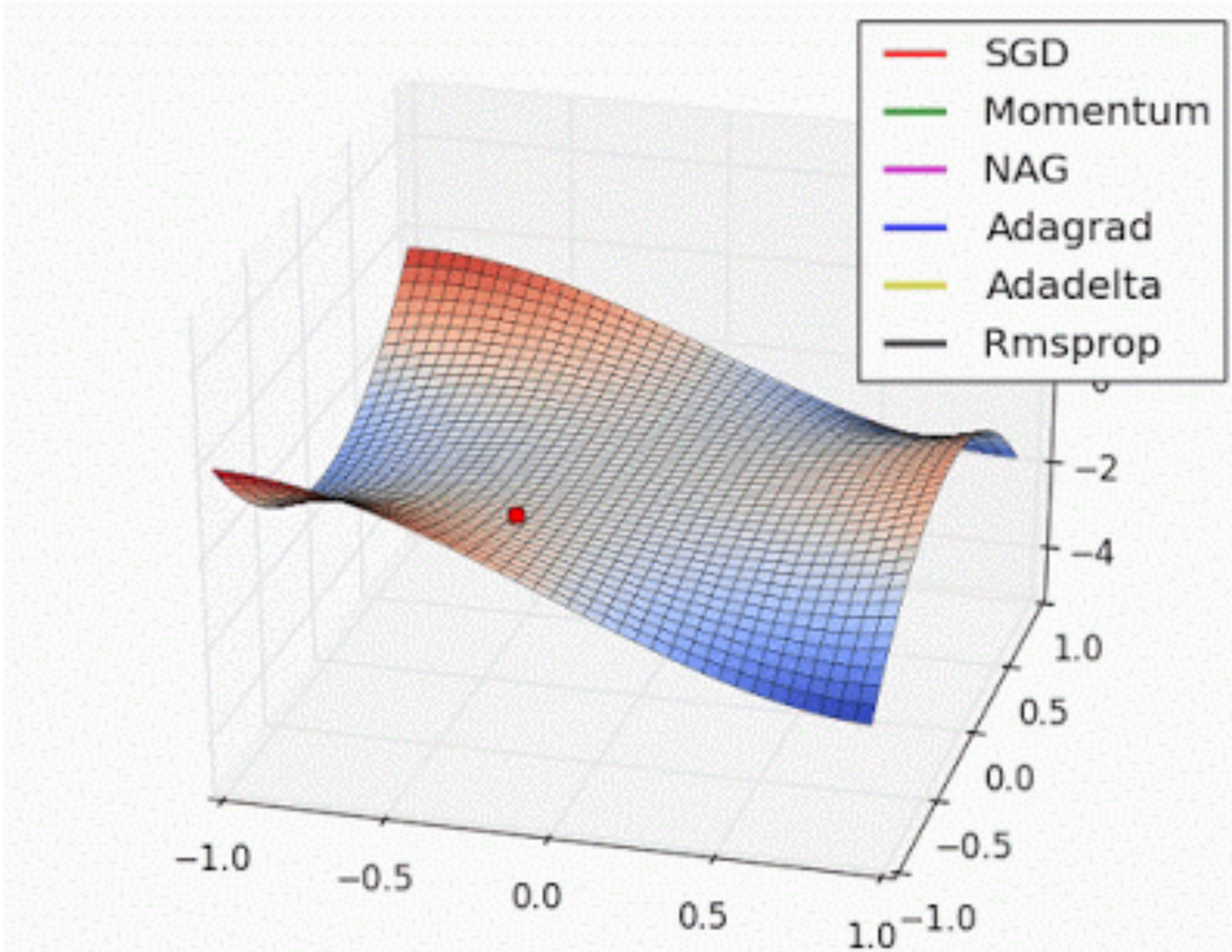


Figure from Alec Radford via http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Online Learning Algorithms

Algorithm Updates:

SGD: $w_{t+1} = w_t - \eta_t(f_t'(w_t) + r'(w_t))$

MD: $w_{t+1} = \arg\min_{w \in \Omega} \eta \langle f_t'(w_t), w - w_t \rangle + \eta r(w) + B_\psi(w, w_t)$

COMID: $w_{t+1} = \arg\min_{w \in \Omega} \eta \langle f_t'(w_t), w - w_t \rangle + \eta r(w) + B_\psi(w, w_t)$

RDA: $w_{t+1} = \arg\min_{w \in \Omega} \eta \langle \bar{g}_t, w \rangle + \eta r(w) + \frac{1}{t}\psi_t(w)$

AdaGrad-COMID: $w_{t+1} = \arg\min_{w \in \Omega} \eta \langle f_t'(w_t) - H_t w_t, w \rangle + \eta r(w) + \frac{1}{2} \langle w, H_t w \rangle$

AdaGrad-RDA: $w_{t+1} = \arg\min_{w \in \Omega} \eta \langle t\bar{g}_t, w \rangle + \eta r(w) + \frac{1}{2} \langle w, H_t w \rangle$

Online Learning Algorithms

Derived Algorithms:

ℓ_1**-regularization** For the regularizer $r(w) = \lambda||w||_1$, we have the following updates.

$$\text{RDA:} \quad w_{t+1,i} = \text{sign}(-\bar{g}_{t,i})\eta\sqrt{t}[|\bar{g}_{t,i}| - \lambda]_+$$

$$\text{AdaGrad-RDA:} \quad w_{t+1,i} = \text{sign}(-\bar{g}_{t,i})\frac{\eta t}{H_{t,ii}}[|\bar{g}_{t,i}| - \lambda]_+$$

$$\text{Fobos (COMID):} \quad w_{t+1,i} = \text{sign}(w_{t,i} - \eta_t g_{t,i}\,[|w_{t,i} - \eta_t g_{t,i}| - \eta_t\lambda]_+$$

$$\text{AdaGrad-COMID:} \quad w_{t+1,i} = \text{sign}\left(w_{t,i} - \frac{\eta}{H_{t,ii}}g_{t,i}\right)\left[\left|w_{t,i} - \frac{\eta}{H_{t,ii}}g_{t,i}\right| - \frac{\lambda\eta}{H_{t,ii}}\right]_+$$

where $[x]_+ = \max(0, x)$.

$g_{t,i}$ is shorthand for the ith element of $f_t'(\boldsymbol{\theta})$.

Online Learning Algorithms

Derived Algorithms:

AdaGrad-COMID

For the ℓ_2^2-regularizer, $r(\boldsymbol{\theta}) = \frac{\lambda}{2}||\boldsymbol{\theta}||_2^2$, we have the following update.

$$\theta_i^{(t+1)} = \frac{H_{t,i,i}\theta_i^{(t)} - \eta g_{t,i}}{\eta\lambda\delta + H_{t,i,i}}$$

where the hyperparameter δ helps deal with the initially noisy values in $H_{t,i,i}$ and typically takes a small positive value ≤ 1.

H_t is a diagonal matrix defined such that each

$$H_{t,i,i} = \delta + \sqrt{\sum_{s=1}^{t} (f_s'(\boldsymbol{\theta})_i)^2}$$

is a smoothed version of the square root of the sum of the squares of the ith element of

Online Learning Algorithms

Derived Algorithms:

AdaGrad-COMID

In the case of no regularizer (i.e. $r(\boldsymbol{\theta}) = 0$), we have the following update.

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\eta}{\sqrt{H_{t,i,i} + \delta}} g_{t,i}$$

H_t is a diagonal matrix defined such that each

$$H_{t,i,i} = \delta + \sqrt{\sum_{s=1}^{t} (f_s'(\boldsymbol{\theta})_i)^2}$$

is a smoothed version of the square root of the sum of the squares of the ith element of