Lecture 2: Optimization for Deep Learning

Supervised learning

$$\{x_1\ x_2\ x_3\ \ldots\ x_M\} \subset \mathbb{R}^N$$

$$\{y_1\ y_2\ y_3\ \ldots\ y_M\} \subset \mathcal{Y}$$

$$f: \mathbb{R}^N \longrightarrow \mathcal{Y}$$

"Textbook" instance: binary linear classifier

$$\mathcal{Y} = \{-1, 1\}$$

$$f(x) = \operatorname{sgn} w^T x$$

In practice, *f* would almost always be from some parameterized space

Minimizing empirical error

Quantity we may care about:

$$E(f) = \int_{x \sim P(x)} [f(x) + y(x)] \, dx$$

In practice, we can assess only *empirical* estimate:

$$E(f) = \sum_{i=1}^{N} [f(x_i) + y_i]$$

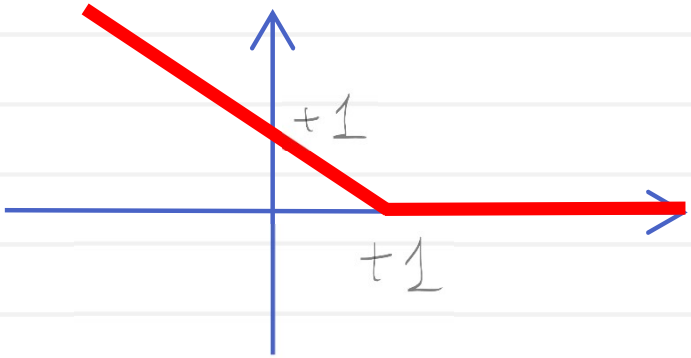
E.g. for binary linear classifier:

$$E(\omega) = \sum_{i}^{N} [\text{sgn } y_i \, \omega^T x_i = -1]$$

Classification problem: hinge loss

Zero-one loss is "non-optimizable".

Most popular relaxation:



$$E(\omega) = \sum_{i}^{N} [\operatorname{sgn} y_i \omega^T x_i = -1]$$

$$E(\omega) = \sum_{i=1}^{N} \max(0, 1 - y_i \omega^T x_i)$$

$$\frac{dE}{d\omega} = \sum_{i=1}^{N} [y_i \omega^T x_i < 1] y_i x_i$$

Classification problem: logistic loss

Logistic function maps R to [0;1]: $\sigma(t) = \frac{1}{1 - e^{-t}}$

Can treat logistic as probability:

$$P(y(x) = y_i \mid \omega) = \frac{1}{1 + e^{-y_i \omega^T x_i}} = \sigma(y_i \omega^T x_i)$$

Define loss as –log likelihood (*ML estimation*):

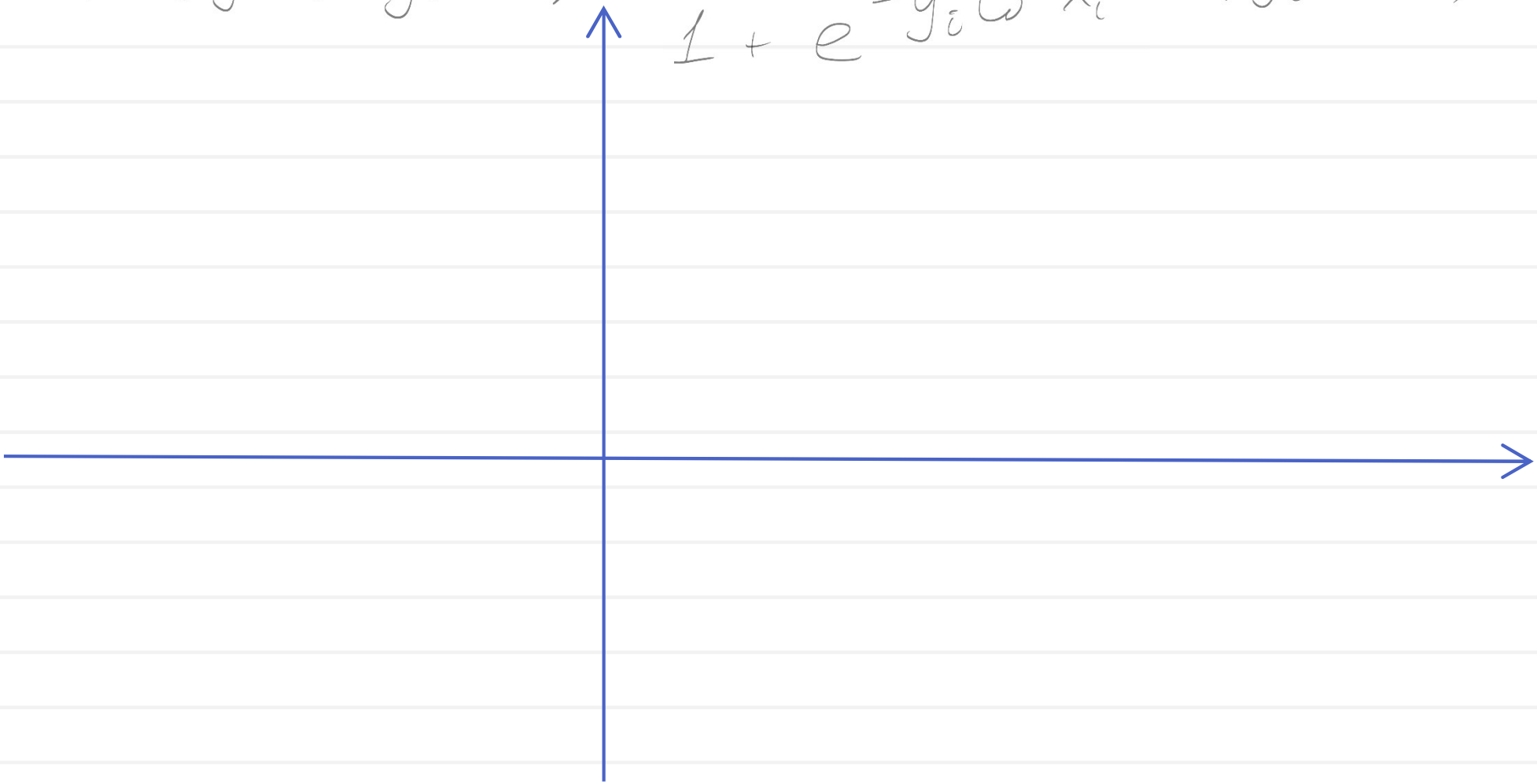
$$E(\omega) = -\sum_{i=1}^{N} \log P(y(x) = y_i \mid \omega) =$$

$$= \sum_{i=1}^{N} \log\left(1 + e^{-y_i \omega^T x_i}\right)$$

$$\frac{dE}{d\omega} = \sum_{i=1}^{N} \left(\sigma(y_i \omega^T x_i) - 1\right) y_i x_i$$

Loss within the logistic regression

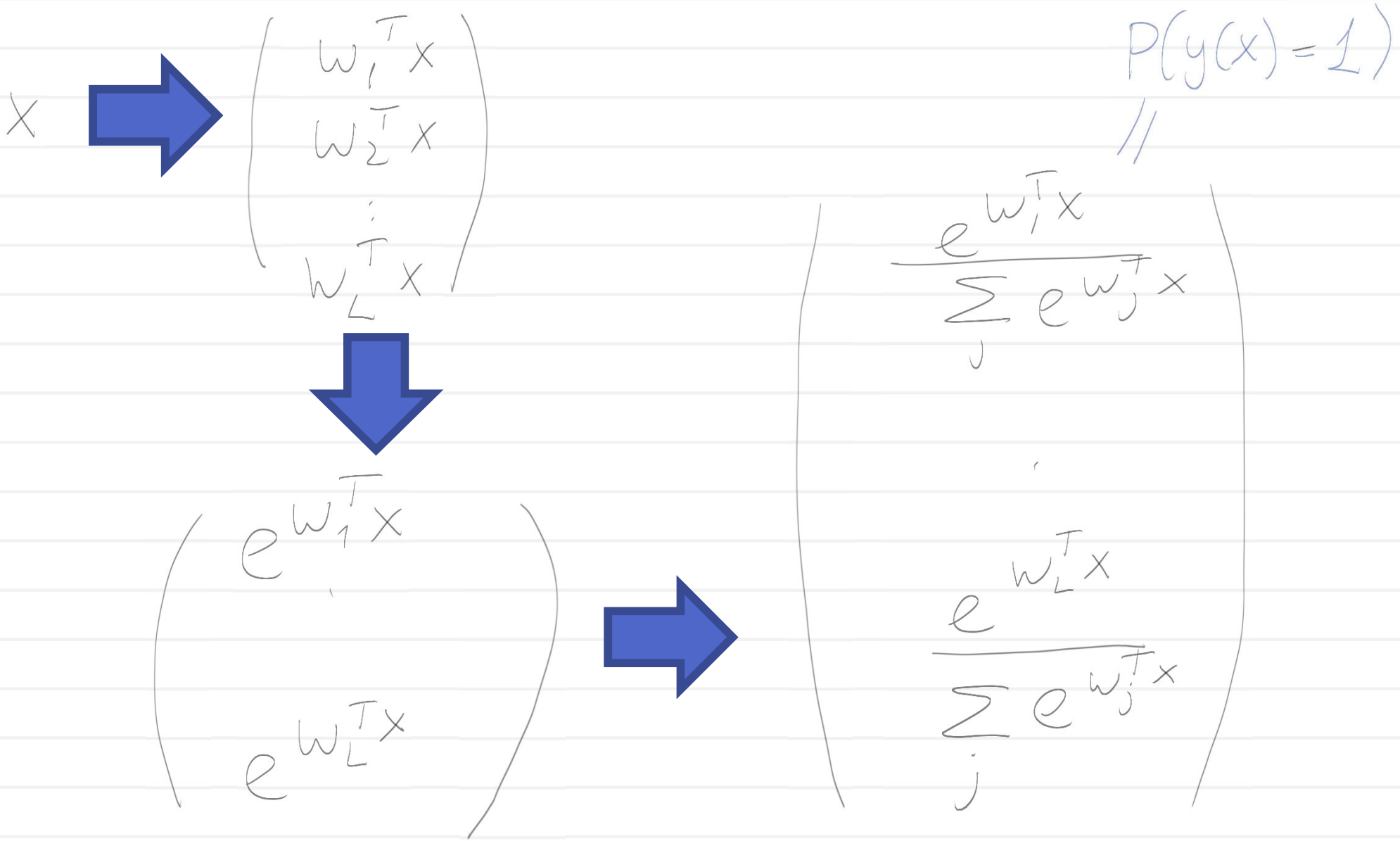
$$P(y(x)=y_i \mid \omega) = \frac{1}{1+e^{-y_i \omega^T x_i}} = \sigma(y_i \omega^T x_i)$$



Logistic loss is also convex!

Multinomial logistic regression

Softmax (generalizes logistic):



Multinomial logistic regression

Multinomial log loss (generalizes logistic loss):

$$E(w) = -\sum_{i=1}^{N} \log P(y(x)=y_i \mid w) =$$

$$-\sum_{i=1}^{N} w_{y_i}^T x_i + \log \sum_{j=1}^{L} e^{w_j^T x_i}$$

(Part of the) gradient over w_j:

$$\frac{dE}{dw_j} = -\sum_{i=1}^{N} x_i \left([y_i = j] - P(y(x_i)=j \mid w) \right)$$

Overfitting in supervised learning

Informally speaking, supervised learning overfits while unsupervised does not (why?)

5 strategies to avoid overfitting (aka *regularize learning*):

- Pick a very simple model
- Stop optimization early
- Add noise to training data
- Impose smoothness
- Combining multiple classifiers

Smoothness: L2-penalty

$$E(f) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x_i), y_i) + \lambda R(f)$$

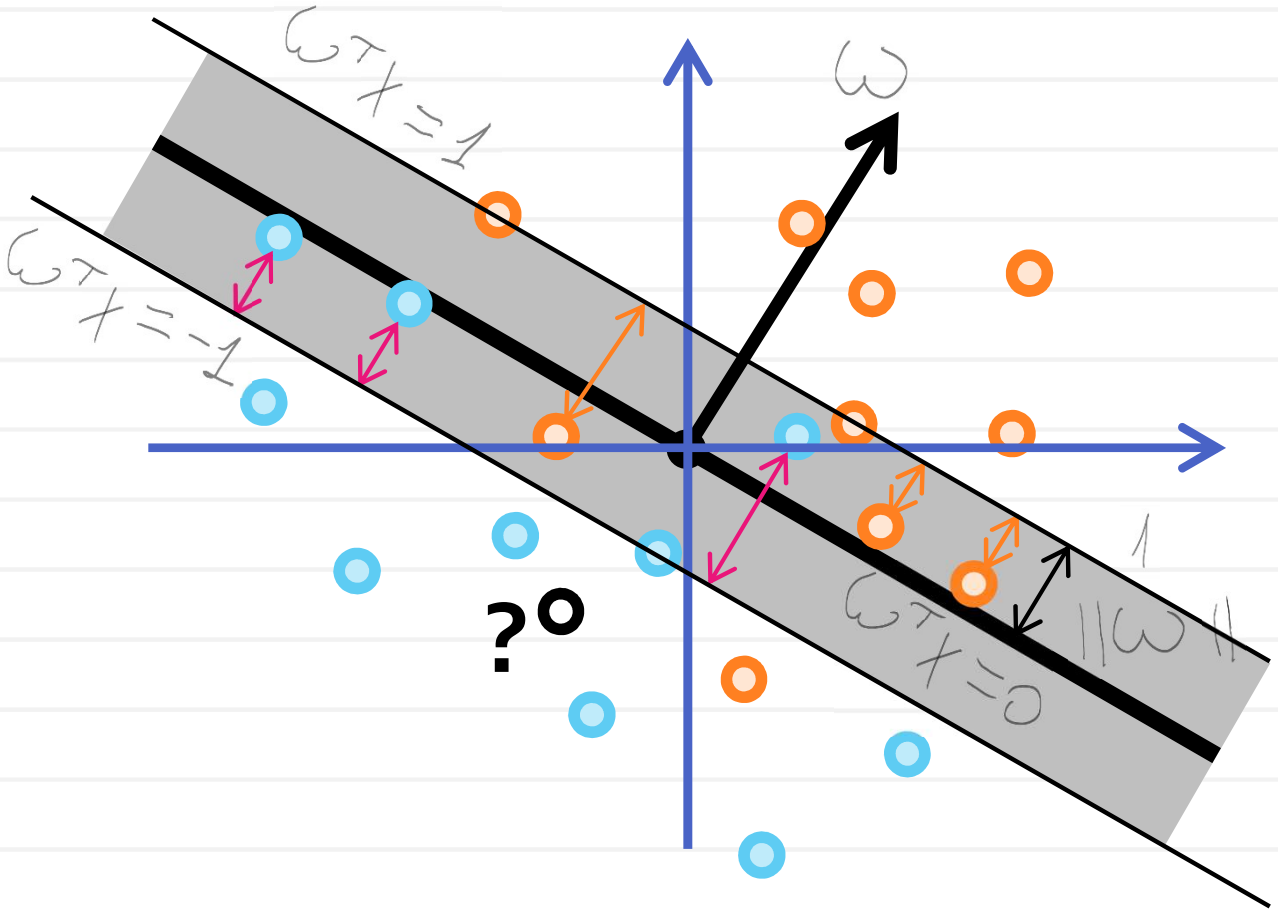
$$f(x) = \omega^T x$$

A natural way to measure the smoothness of a linear function: look at the (square of the) magnitude of its slope:

$$R(f) = \frac{1}{2} \|\omega\|_2^2$$

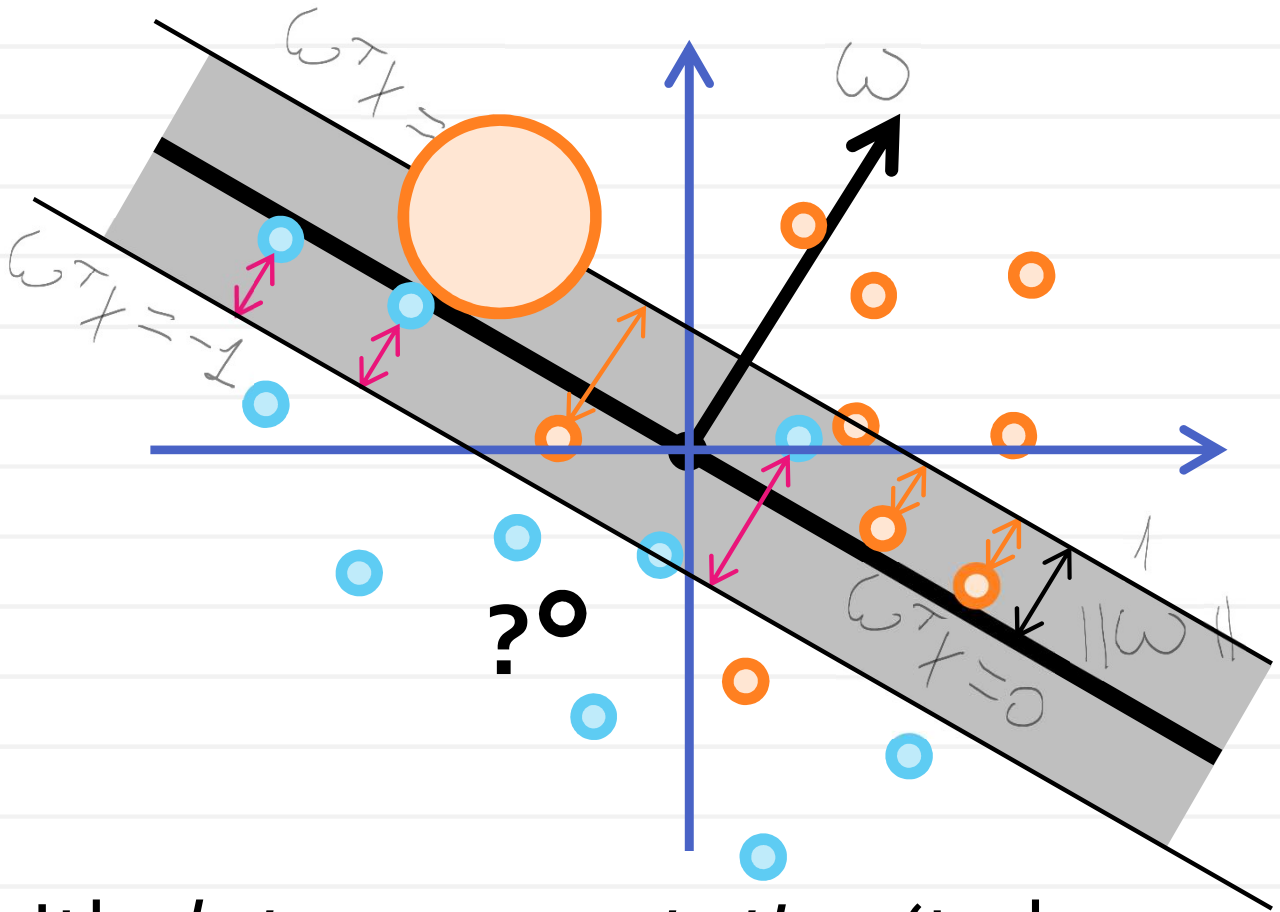
Smoothness -> margin

$$\min_{w} \sum_{i=1}^{N} h(y_i \omega^T x) + \frac{\lambda}{2} \|w\|_2^2$$



Regularizing by randomness

Many systems regularize by adding noise to training data. This is related to margin maximization:



This also borders with *data augmentation* (to be discussed later in the class)

A zoo of supervised learning tasks

- Classification (binary, multi-label)
- Regression
- Ranking
- *Structured prediction*
- …

all can be handled via different loss functions.

Example: ridge regression

$$\{x_1\ x_2\ x_3\ \ldots\ x_M\} \subset \mathbb{R}^N$$

$$\{y_1\ y_2\ y_3\ \ldots\ y_M\} \subset \mathbb{R}$$

$$\min_w \sum_{i=1}^{N} (w^T x_i - y_i)^2 + \frac{\lambda}{2} \|w\|_2^2$$

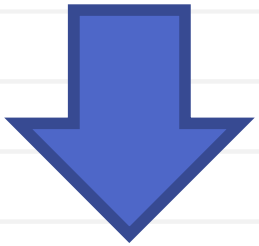
Closed form solution:

$$w = \left(XX^T + \lambda I\right)^{-1} X\, y$$

Other combinations of loss/regularizer possible (e.g. quadratic loss + L1-regularization = LASSO)

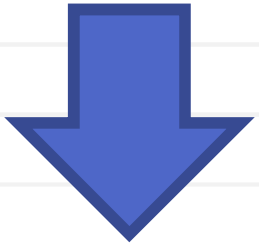
Supervised learning approach

Problem + Loss



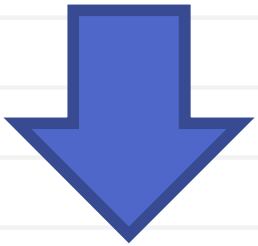
Approximation error

Parameterized predictor p(w)



Estimation error

E(w) = (loss on the training set)+regularizer



Optimization error

Result

Approximation error

Bayes-optimal classifier/predictor over all predictors:

$$f_0 = \arg\min_{f \in \mathcal{F}_0} \int_{(x,y)\sim P(x,y)} \ell(x, y, f)$$

The class of predictors is usually restricted (e.g. to some parametric family):

$$\tilde{f} = \arg\min_{f \in \tilde{\mathcal{F}}} \int_{(x,y)\sim P(x,y)} \ell(x, y, f)$$

$\|f_0 - \tilde{f}\|$ = approximation error

Estimation error

$$\tilde{f} = \arg\min_{f \in \mathcal{F}} \int_{(x,y)\sim P(x,y)} l(x, y, f)$$

Cannot integrate, have a finite sample of data:

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} l(x, y, f) + \lambda R(f)$$

$\|\tilde{f} - \hat{f}\|$ = estimation error

Optimization error

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} l(x, y, f) + \lambda R(f)$$

Have a limited amount of time to optimize:

$$\hat{f}^{(t)} = \arg\min_{f \in \mathcal{F}}{}^{(t)} \frac{1}{N} \sum_{i=1}^{N} l(x, y, f) + \lambda R(f)$$

$\| \hat{f} - \hat{f}^{(t)} \|$ = optimization error

Optimization and supervised ML

$$f_0 = \arg\min_{f \in \mathcal{F}_0} \int \ell(x, y, f) \quad (x, y) \sim P(x, y)$$

Approximation error

$$\tilde{f} = \arg\min_{f \in \mathcal{F}} \int \ell(x, y, f) \quad (x, y) \sim P(x, y)$$

Estimation error

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \ell(x, y, f) + \lambda R(f)$$

Optimization error

$$\hat{f}^{(t)} = \arg\min^{(t)}_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \ell(x, y, f) + \lambda R(f)$$

[Bottou & Bousquet NIPS 2007]

Overall error

$$\|f_0 - \hat{f}^{(t)}\| \leq$$

total error

$$\|f_0 - \tilde{f}\| +$$

approximation error

$$\|\tilde{f} - \hat{f}\| +$$

estimation error

$$\|\hat{f} - \hat{f}^{(t)}\|$$

optimization error

Various design choices trade-off different errors e.g.:

- Parameterization
- Time for data vs. optimization

[Bottou & Bousquet NIPS 2007]

Small-scale learning vs Large-scale learning

$$\|f_0 - \hat{f}^{(t)}\| \leq \|f_0 - \tilde{f}\| + \|\tilde{f} - \hat{f}\| + \|\hat{f} - \hat{f}^{(t)}\|$$

Small-scale learning:
we are data-bound,
estimation error
dominates

estimation error

Large-scale learning:
we are time-bound,
optimization error is a
bigger issue

optimization error

[Bottou & Bousquet NIPS 2007]

Small scale setting: traditional optimization

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} l(x, y, f) + \lambda R(f)$$

- Data are few, we can look through it at each optimization iteration
- Use adapted versions of standard optimization methods (gradient descent, quasi-Newton, quadratic programming,…)

Example: from SVM to quadratic program

$$w = \arg\min_{w} \frac{1}{N} \sum_{i=1}^{N} h(x_i, y_i, w) + \frac{\lambda}{2} \|w\|_2^2$$

Hinge loss is often implemented using "slack variables":

$$\min_{\omega, \xi} \frac{\lambda}{2} \|\omega\|^2 + \frac{1}{N} \sum_{i=1}^{N} \xi_i$$

$$\text{s.t.} \quad \xi_i \geq 1 \quad y_i \omega^T x_i$$

$$\xi_i \geq 0$$

Large-scale learning

$$E(w) = \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, y_i, w) + \lambda R(w)$$

$$\frac{dE}{dw} = \frac{1}{N} \sum_{i=1}^{N} \frac{d\ell(x_i, y_i, w)}{dw} + \lambda \frac{dR}{dw}$$

- Evaluating gradient is very expensive
- It will only be good for one (small) step

Stochastic gradient descent (SGD) idea:

- Evaluate a coarse approximation to grad
- Make "quick" steps

Stochastic gradient descent (SGD)

Gradient:

$$\frac{dE}{dw} = \frac{1}{N}\sum_{i=1}^{N}\frac{d\ell(x_i, y_i, w)}{dw} + \lambda\frac{dR}{dw}$$

Stochastic gradient:

$$\frac{dE^i}{dw} = \frac{d\ell(x_i, y_i, w)}{dw} + \lambda\frac{dR}{dw}$$

$$\frac{dE}{dw} = \frac{1}{N}\sum_{i=1}^{N}\frac{dE^i}{dw}$$

Stochastic gradient is an unbiased estimate of gradient

Stochastic gradient descent (SGD)

SGD:

$$v[t] = -\alpha[t]\,\nabla(E, w[t])$$
$$w[t+1] = w[t] + v[t]$$

where $\nabla(E, w[t]) = \left.\frac{dE^{c(t)}}{dw}\right|_{w[t]}$

- i(t) usually follow random permutations of training data
- One sweep over training data is called an **epoch**

Stochastic gradient descent (SGD)

SGD:

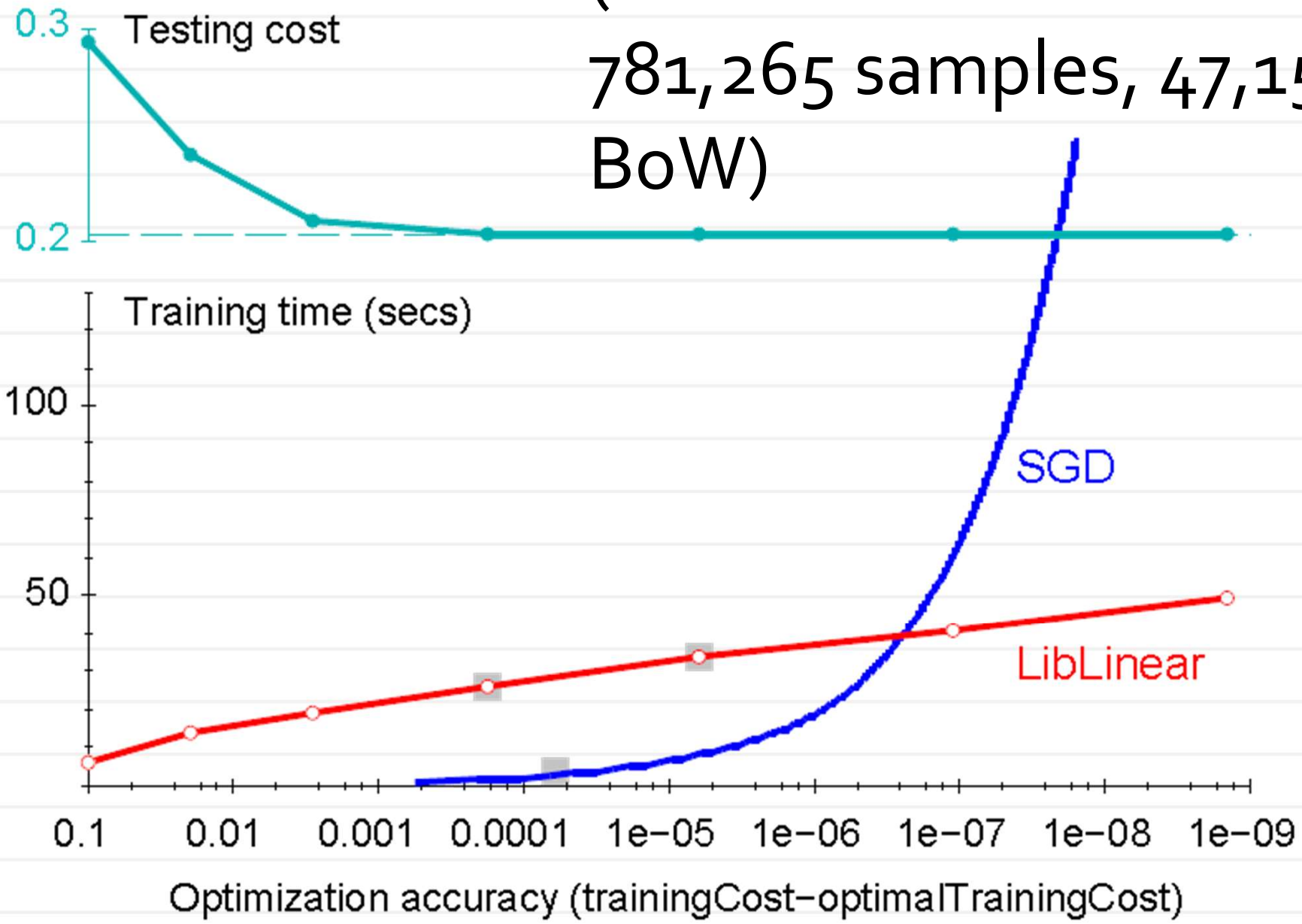
$$v[t] = -\alpha[t] \nabla(E, w[t])$$
$$w[t+1] = w[t] + v[t]$$

- One sweep over training data is called an **epoch**
- Popular choices for schedule $\alpha[t]$:
  - constant, e.g. $\alpha[t] = 0.0001$
  - piecewise constant, e.g. $\alpha[t]$ is decreased tenfold every N epochs
  - harmonic, e.g. $\alpha[t] = 0.001 / ([t/N]+10)$

The efficiency of SGD

slide credit: L.Bottou

(Document classification : 781,265 samples, 47,152-dim BoW)



Batch SGD

Gradient:

$$\frac{dE}{dw} = \frac{1}{N} \sum_{i=1}^{N} \frac{d\ell(x_i, y_i, w)}{dw} + \lambda \frac{dR}{dw}$$

Batch (aka mini-batch):

$$\{b_1, b_2 \dots b_{N_b}\} \subset 1 \dots N$$

Batch stochastic gradient:

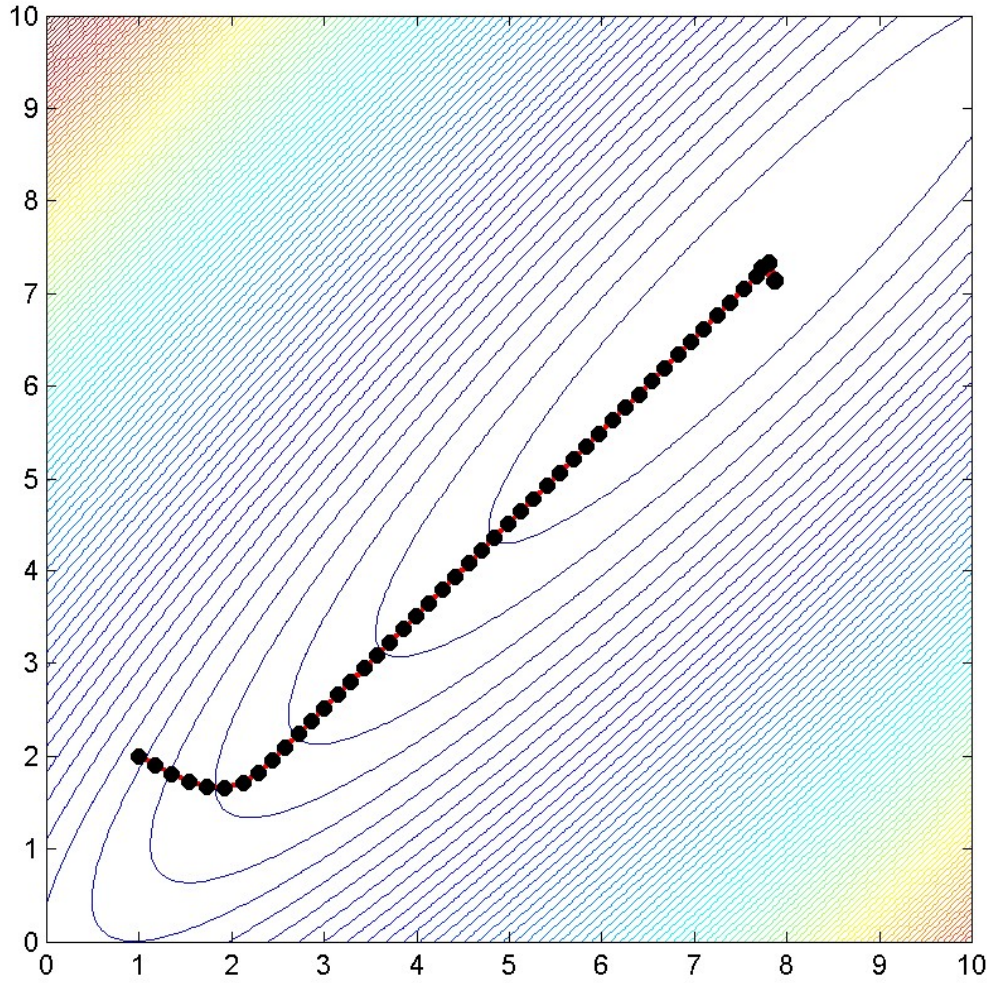
$$\frac{dE}{dw} = \frac{1}{N_b} \sum_{i=1}^{N_b} \frac{d\ell(x_{b(i)}, y_{b(i)}, w)}{dw} + \lambda \frac{dR}{dw}$$

Why batching?

$$\frac{dE}{dw} = \frac{1}{N_b} \sum_{i=1}^{N_b} \frac{d\ell(x_{b(i)}, y_{b(i)}, w)}{dw} + \lambda \frac{dR}{dw}$$

- "Less stochastic" approximation, more stable convergence (questionable)
- **Main reason:** all modern architectures have parallelism, hence computing mini-batch grad is often as cheap as a single stochastic grad

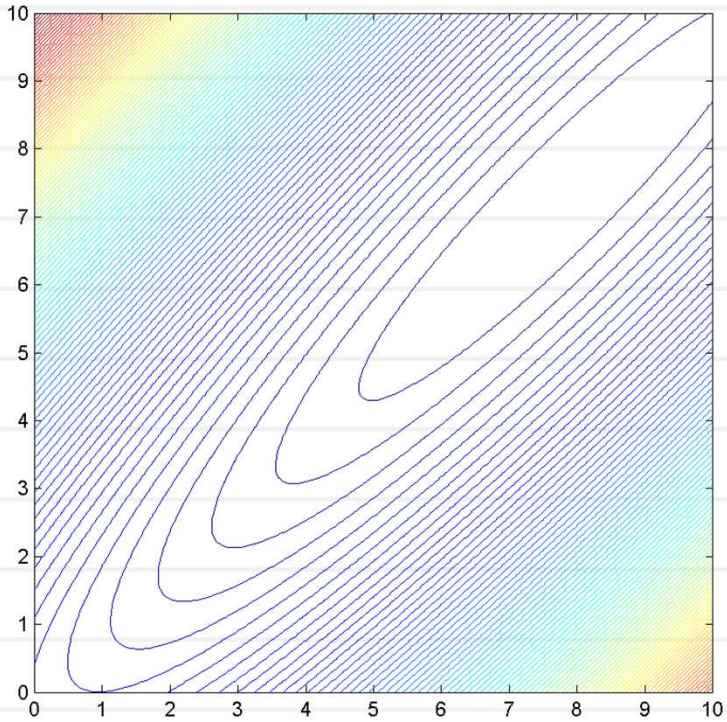
SGD inherits gradient descent problems

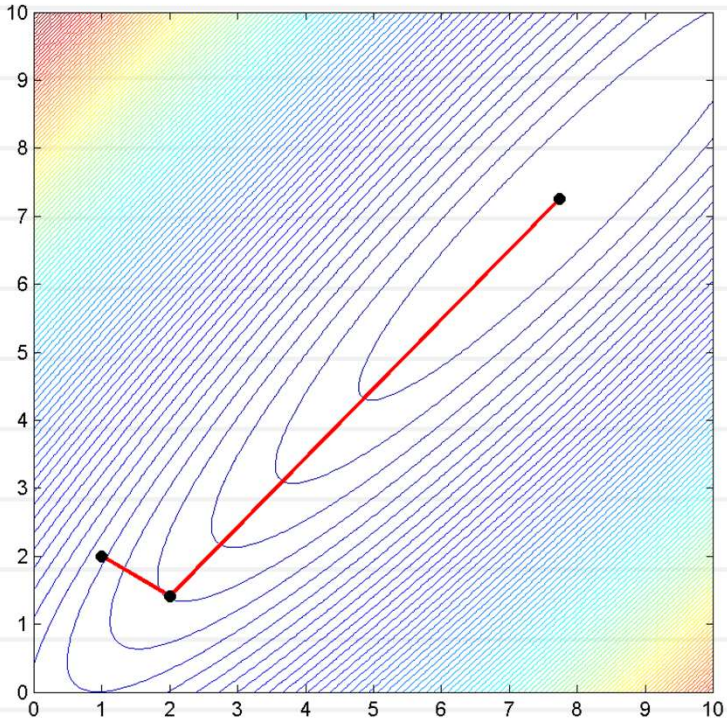


- Gradient descent is very poor "in ravines"
- SGD is no better

Better optimization methods

- Second order methods (Newton, Quasi-Newton)
- Krylov subspace methods, in particular *conjugate gradients*

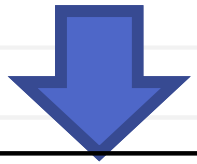




Improving SGD using momentum

- Conjugate gradients use a combination of the current gradient and previous direction for the next step
- Similar idea for SGD (*momentum*):

$$v[t] = -\alpha[t]\,\nabla(E, w[t])$$
$$w[t+1] = w[t] + v[t]$$



$$v[t+1] = \mu\, v[t] - \alpha[t]\,\nabla(E, w[t])$$
$$w[t+1] = w[t] + v[t+1]$$

Typical $\mu = 0.9$

Exponentially decaying running average

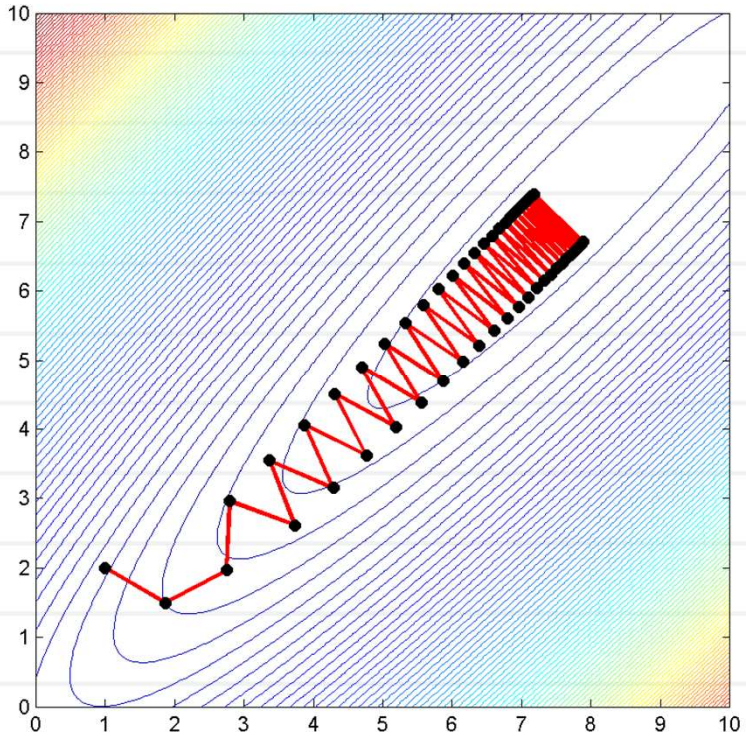
$$v[t+1] = \mu\, v[t] - \alpha[t]\, \nabla\,(\, E,\, w[t])$$
$$w[t+1] = w[t] + v[t+1]$$

$$v[t+1] = \mu\, v[t] - \alpha[t]\, \nabla\,(\, E,\, w[t]) =$$
$$= \mu^2\, v[t-1] - \mu\alpha[t-1]\, \nabla\,(\, E,\, w[t-1])$$
$$- \alpha[t]\, \nabla\,(\, f,\, w[t]) =$$
$$= \mu^3\, v[t-2] - \mu^2\, \alpha[t-2]\, \nabla\,(\, E,\, w[t-2])$$
$$- \mu\alpha[t-1]\, \nabla\,(E,\, w[t-1]) - \alpha[t]\, \nabla\,(\, E,\, w[t]) =$$
$$= \mu^{k+1}\, v[t-k] + \sum_{i=0}^{k} \mu^i\, \alpha[t-i]\, \nabla\,(E,\, w[t-i])$$

Momentum: why it works

$$v[t+1] = \sum_{i=0}^{k} \mu^i \alpha[t-i] \nabla(E, w[t-i])$$

- Smoothes out noise in SGD (~bigger batches)
- **Smoothes out oscilations inherent to gradient descent**
- Escapes local minima



Nesterov accelerated gradient

$$v[t+1] = \mu\, v[t] - \alpha[t]\, \nabla(E, w[t])$$
$$w[t+1] = w[t] + v[t+1]$$

Before we even compute the gradient, we have a good approximation where we will end up: $w[t+1] \approx w[t] + \mu\, v[t]$

Let us use this knowledge:

$$v[t+1] = \mu\, v[t] - \alpha[t]\, \nabla(E, w[t] + \mu\, v[t])$$
$$w[t+1] = w[t] + v[t+1]$$

(Computing the gradient at a more relevant spot)

Second-order methods

- Exponential smoothing helps, but still not optimal if large anisotropy exists
- Classic (Newton) solution: estimate the Hessian and make the update
  $v[t+1] = - H[t]^{-1} \nabla (E, w[t])$ (the lower the curvature the faster we go)
- Quasi-Newton methods: estimate some approximation to Hessian based on observed gradients
- Quasi-Newton can be used in batch mode, but same batch should be used over several iterations (why?)

Adagrad method [Duchi et al. 2011]

Adagrad idea: scale updates along different dimensions according to accumulated gradient magnitude

$$g[t+1] = g[t] + \nabla(E, w[t]) \odot \nabla(E, w[t])$$

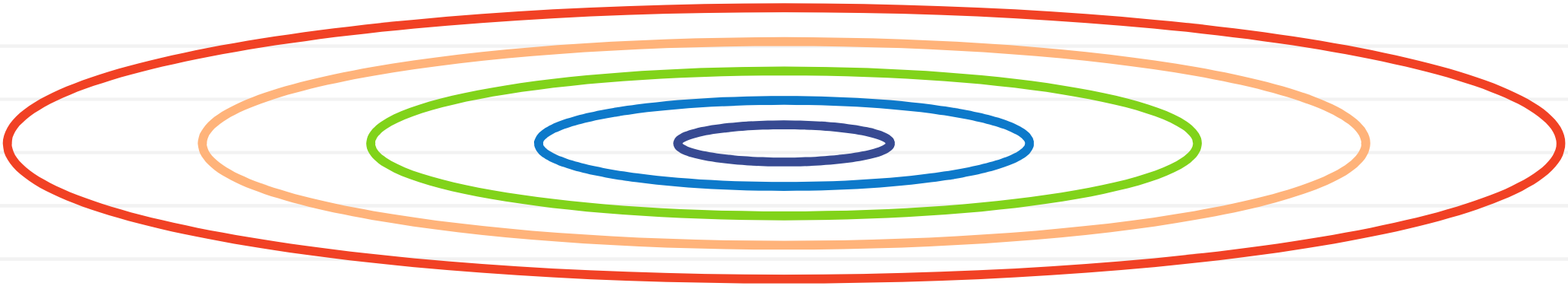
$$w[t+1] = w[t] - \frac{\alpha}{\sqrt{g[t] + \varepsilon}} \odot \nabla(E, w[t])$$

Note: step lengths automatically decrease (perhaps too quickly).

Adagrad method [Duchi et al. 2011]

$$g[t+1] = g[t] + \nabla(E, w[t]) \odot \nabla(E, w[t])$$

$$w[t+1] = w[t] - \frac{\alpha}{\sqrt{g[t] + \varepsilon}} \odot \nabla(E, w[t])$$



Adagrad in this case: find out that "vertical" derivatives are bigger, then make "vertical" steps smaller than "horizontal"

RMSPROP method [Hinton 2012]

Same as Adagrad, but replace accumulation of squared gradient with running averaging:

$$g[t+1] = \mu\, g[t] + (1-\mu)\nabla(E, w[t]) \odot \nabla(E, w[t])$$

$$w[t+1] = w[t] - \frac{\alpha[t]}{\sqrt{g[t] + \varepsilon}} \odot \nabla(E, w[t])$$

Units of measurements

- Let our coordinates be measured in meters. What is the unit of measurement for gradients? Assume unitless function…
- (Stochastic) gradient descent is inconsistent.
- Newton method is consistent.

Adadelta method [Zeiler 2012]

$$g[t+1] = \mu\, g[t] + (1-\mu)\, \nabla(E, w[t]) \odot \nabla(E, w[t])$$

$$w[t+1] = w[t] - \frac{\sqrt{d[t] + \varepsilon}}{\sqrt{g[t] + \varepsilon}} \odot \nabla(E, w[t])$$

$$d[t+1] = \mu\, d[t] + (1-\mu)\, (w[t+1]-w[t]) \odot (w[t+1]-w[t])$$

- No step length parameter (good!)
- Correct units within the updates

ADAM method [Kingma & Ba 2015]

ADAM = "ADAptive Moment Estimation"

$$v[t+1] = \beta\, v[t] + (1-\beta)\, \nabla(E, w[t])$$

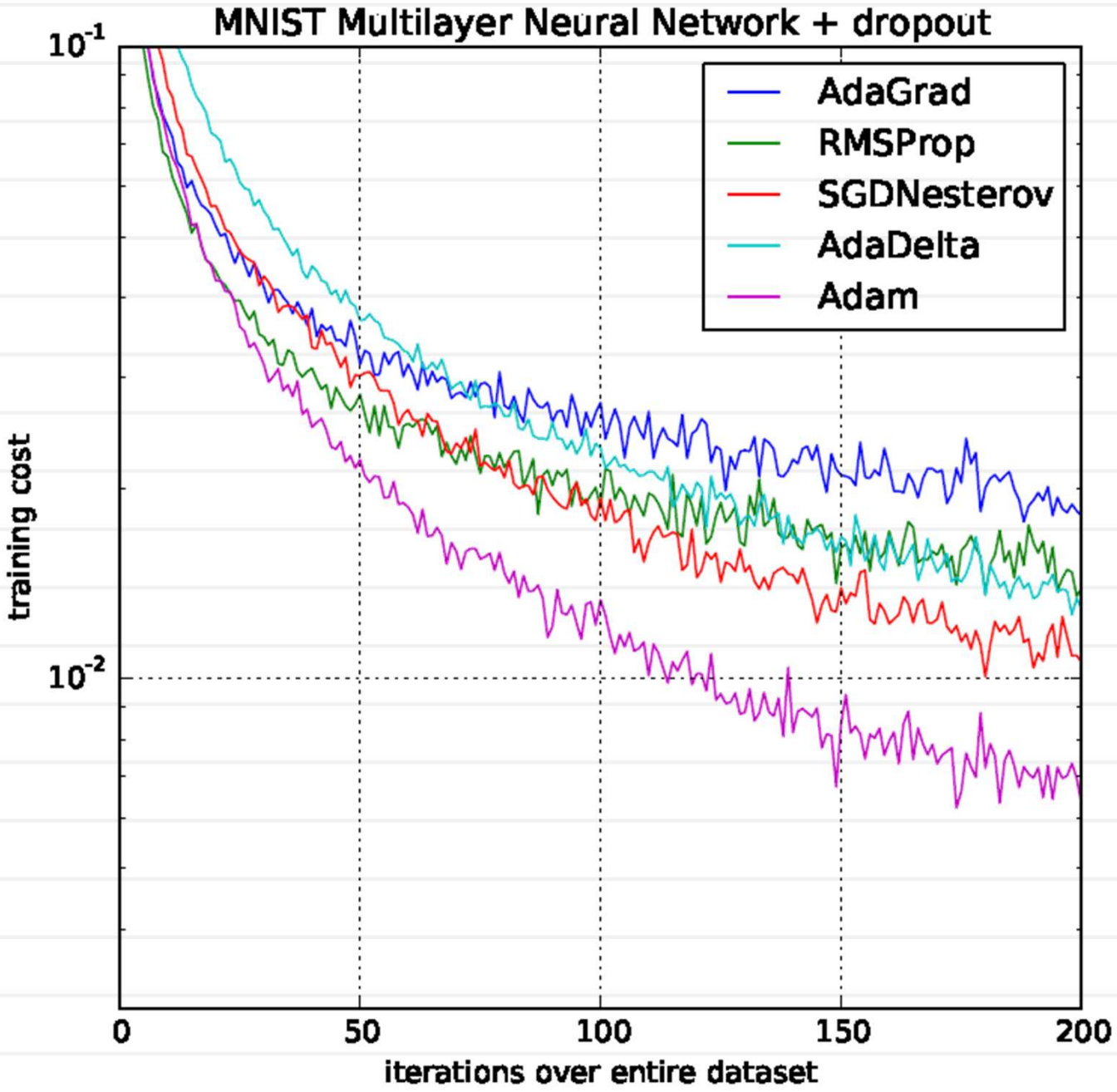
$$g[t+1] = \mu\, g[t] + (1-\mu)\nabla(E, w[t]) \odot \nabla(E, w[t])$$

$$w[t+1] = w[t] - \alpha \frac{1}{\sqrt{g[t+1]+\varepsilon}} \odot v[t+1] \frac{1}{1-\beta^t}$$

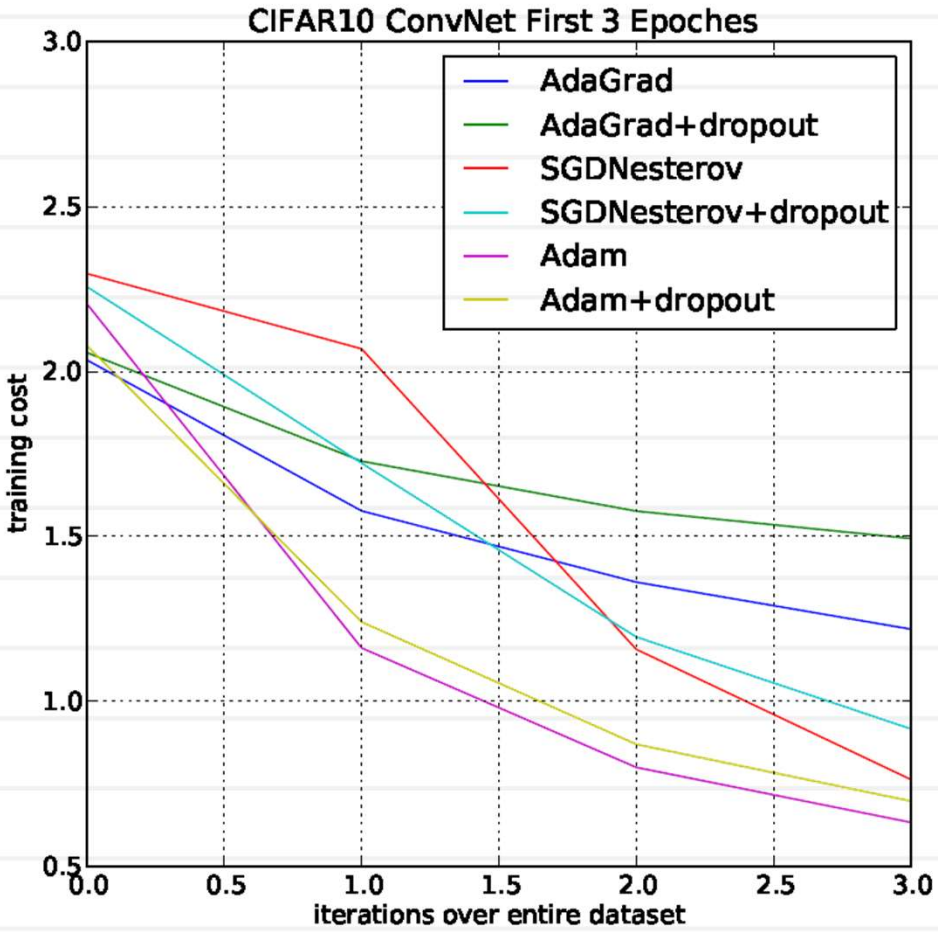
$$\frac{1}{1-\mu^t}$$

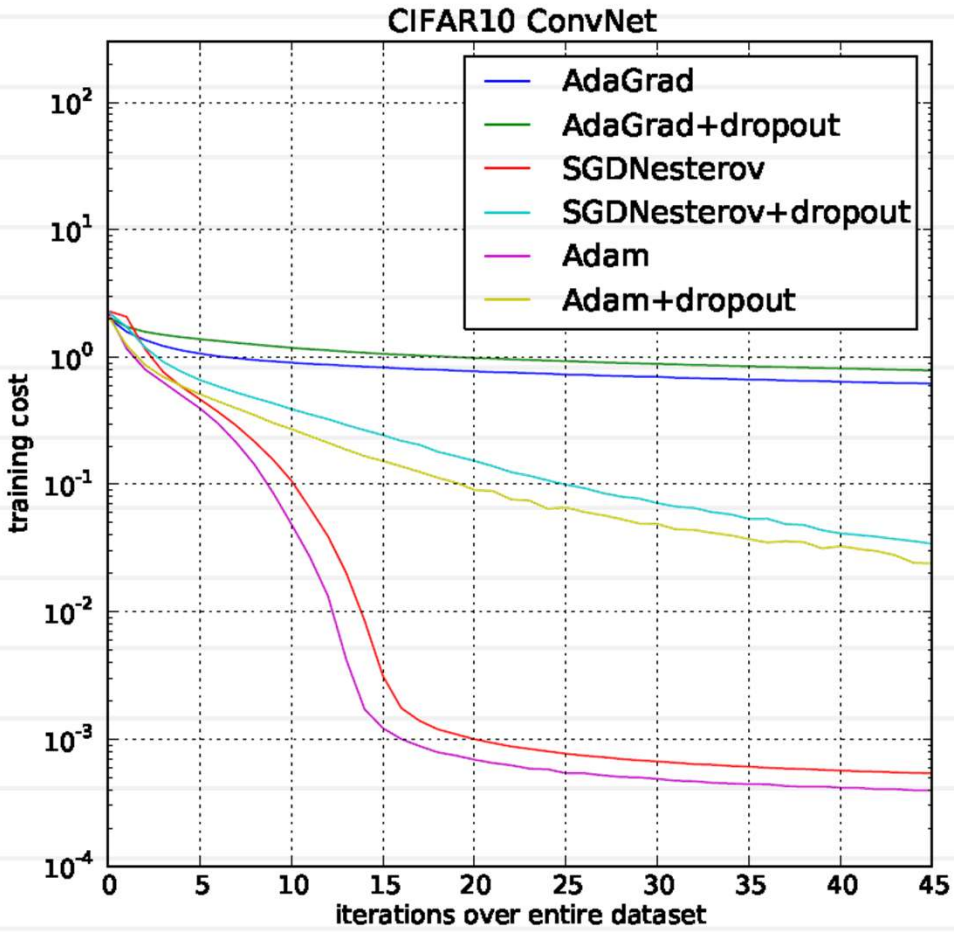
Recommended values: $\beta = 0.9$, $\mu = 0.999$, $\alpha = 0.001$, $\varepsilon = 10^{-8}$

ADAM method [Kingma & Ba 2015]



ADAM method [Kingma & Ba 2015]





Comparison: logistic regression

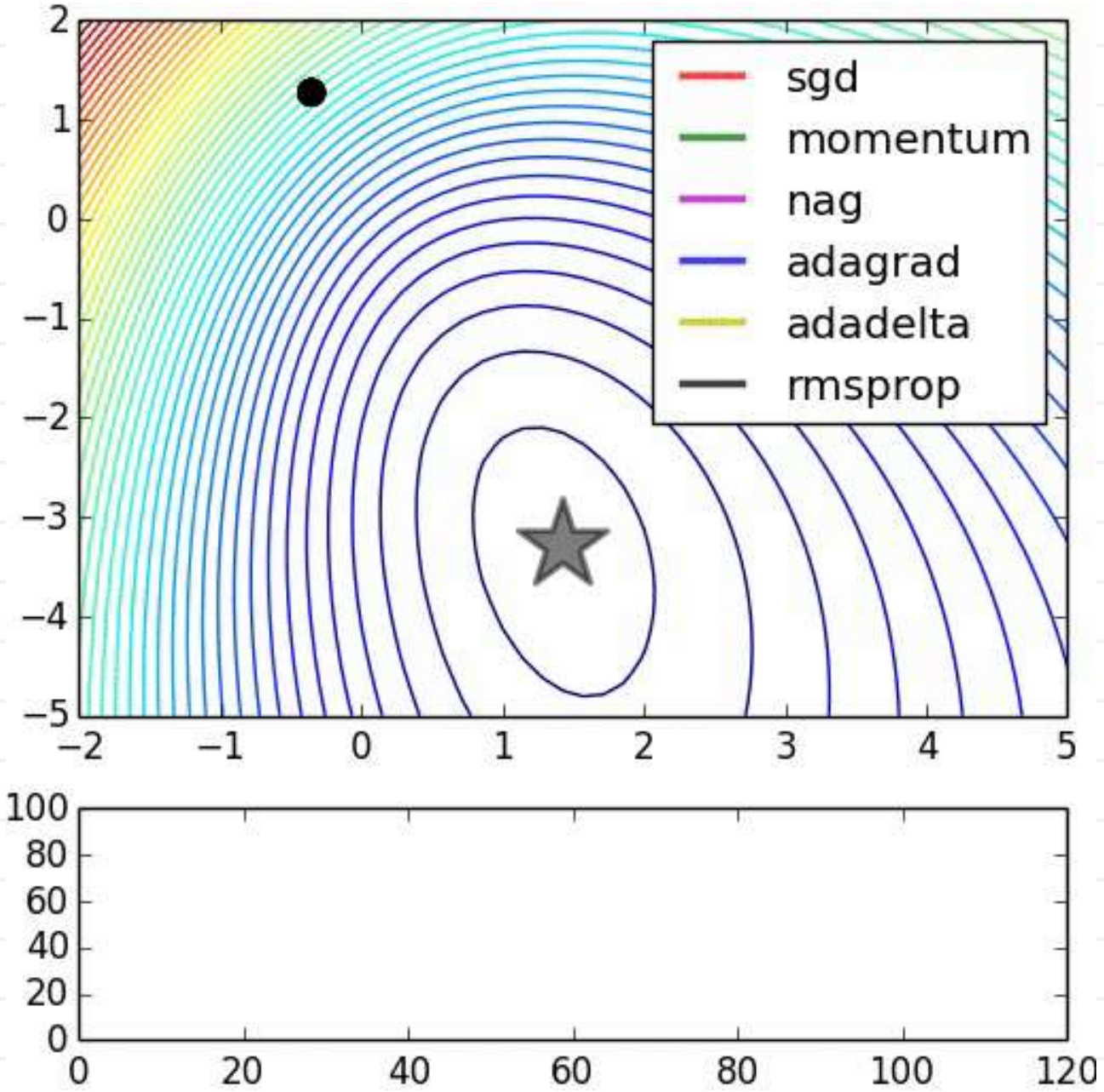


Image credit: Alec Redford

Further comparison

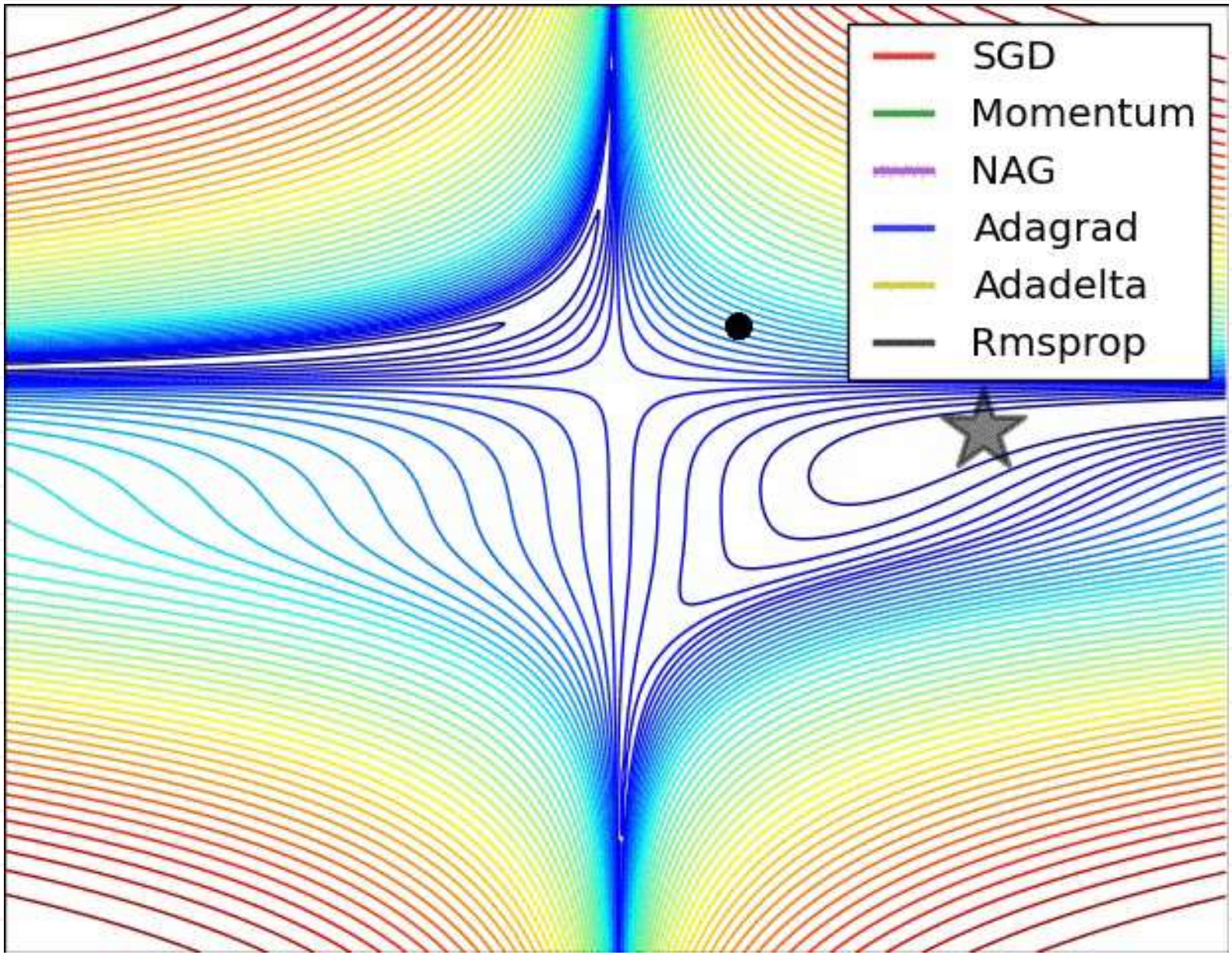


Image credit: Alec Redford

Further comparison: escaping from a saddle

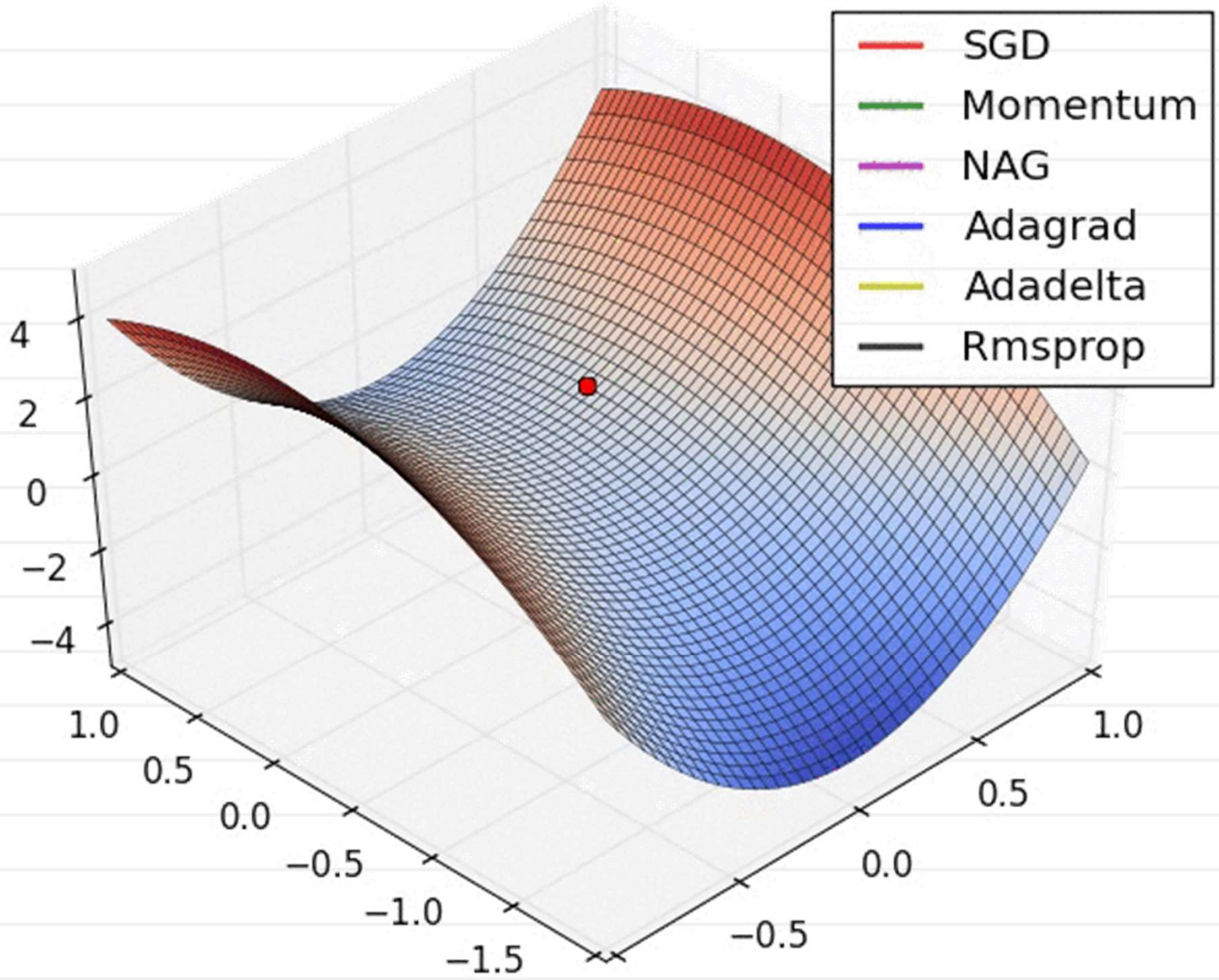


Image credit: Alec Redford

Recap

- Supervised learning: the most popular tool in machine learning
- Much more flexible than classification/regression
- Different types of error: Bayes, Approximation, Estimation, Optimization
- SGD optimization is used in large-scale setting
- Advanced SGD methods use running averages to smooth and rescale SGD steps

Bibliography

Léon Bottou, Olivier Bousquet:
The Tradeoffs of Large Scale Learning. NIPS 2007: 161-168

Nesterov, Yurii. "A method of solving a convex programming problem with convergence rate O (1/k2)." Soviet Mathematics Doklady. Vol. 27. No. 2. 1983.

John C. Duchi, Elad Hazan, Yoram Singer:
Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research 12: 2121-2159 (2011)

Matthew D. Zeiler:
ADADELTA: An Adaptive Learning Rate Method. CoRR abs/1212.5701

Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." ICLR 2015