

Deep Learning (BEV033DLE)

Lecture 8

Adaptive SGD Methods

Czech Technical University in Prague

◆ Geometry of Neural Network Loss Surfaces

- Local minima and saddle points in nD
- Parameter redundancy helps optimization

◆ Understanding Adaptive Methods

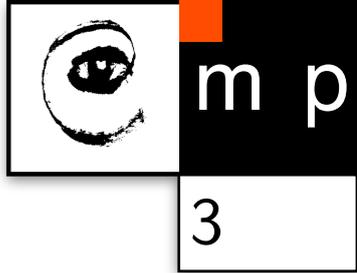
- Proximal Problems, Convex vs non-convex, Stochastic optimization
- Adam, RMSprop, Adargad

◆ Changing the Space Metric

- Change of Coordinates, Preconditioning, Equivalent reparameterizations, Constraints

Loss Landscape

Local Minima

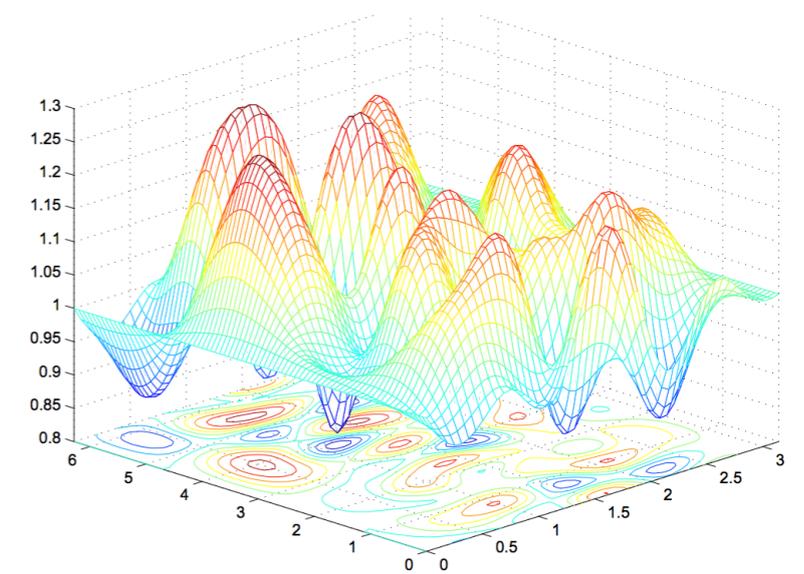
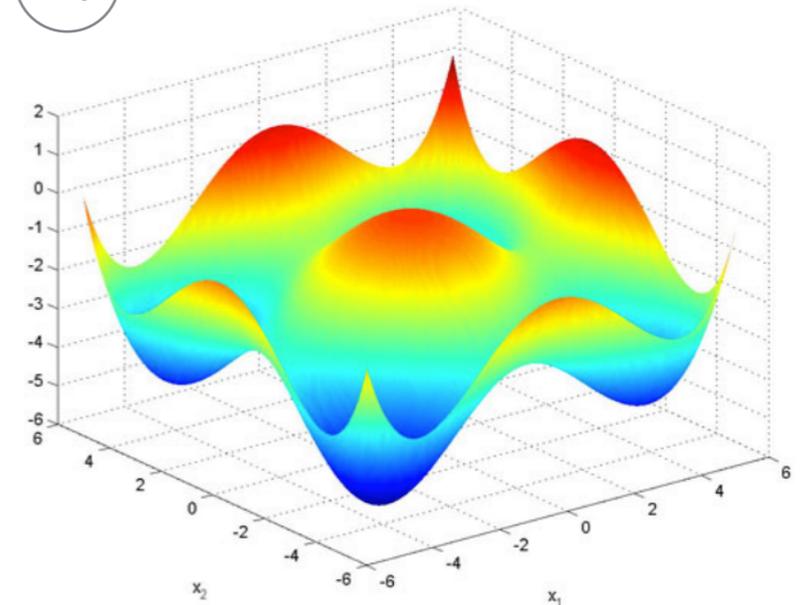
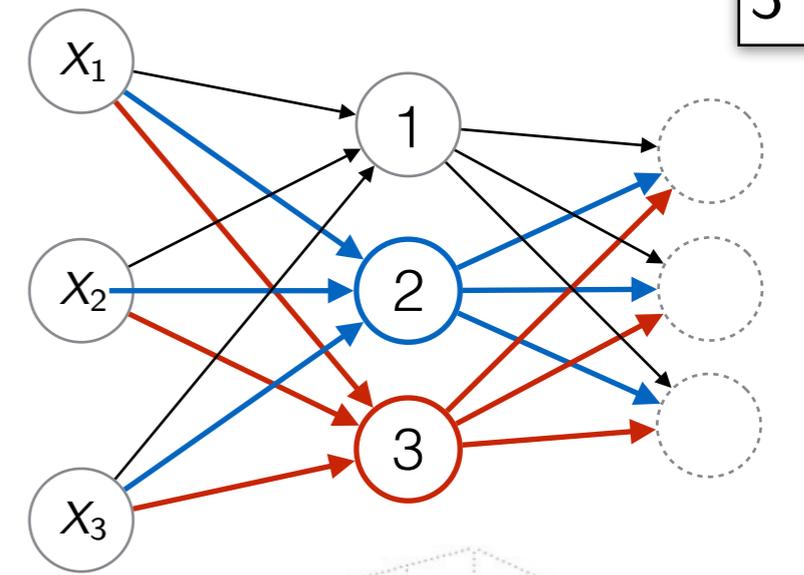
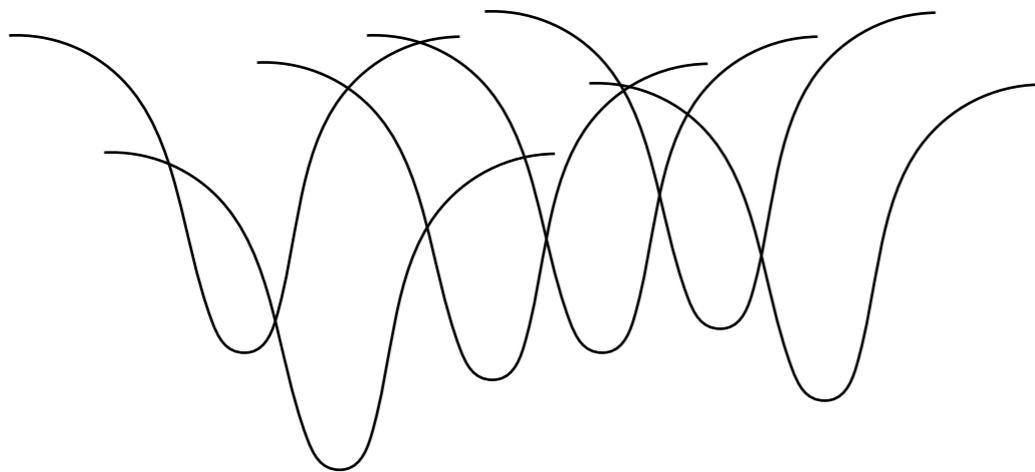


- ◆ There are several reasons for local minima
 - **Symmetries** (Permutation invariances)
 - Fully connected layer with **n** hidden units:
n! permutations
 - Convolutional layer with **c** channels:
c! permutations
 - In a deep network many equivalent local minima, but all of them are equally good -- no need to avoid
 - Loss function is a **sum of many non-convex terms**:

$$L(\theta) = \sum_i l(y_i, f(x_i; \theta))$$

often convex

non-linear



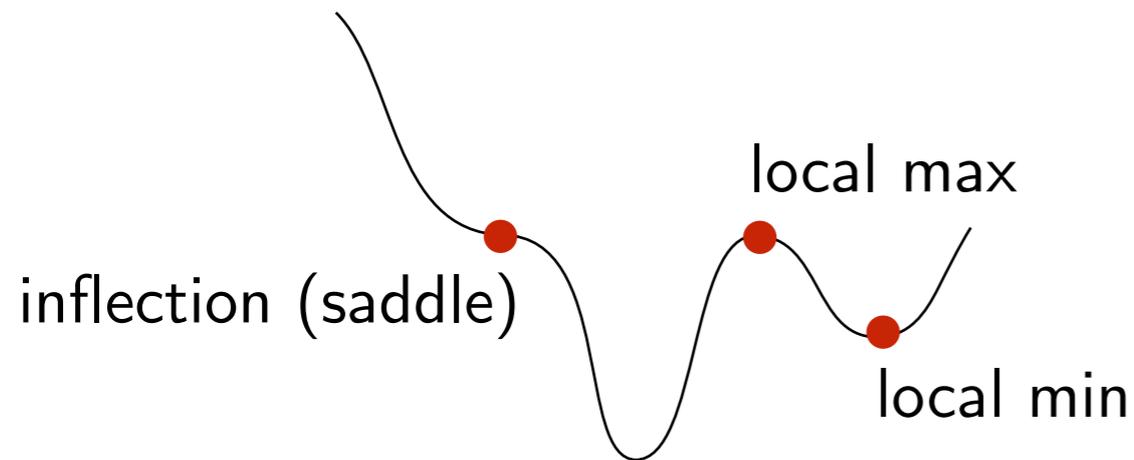
Stationary Points in High Dimensions

Let $f(x): \mathbb{R}^n \rightarrow \mathbb{R}$ – differentiable,

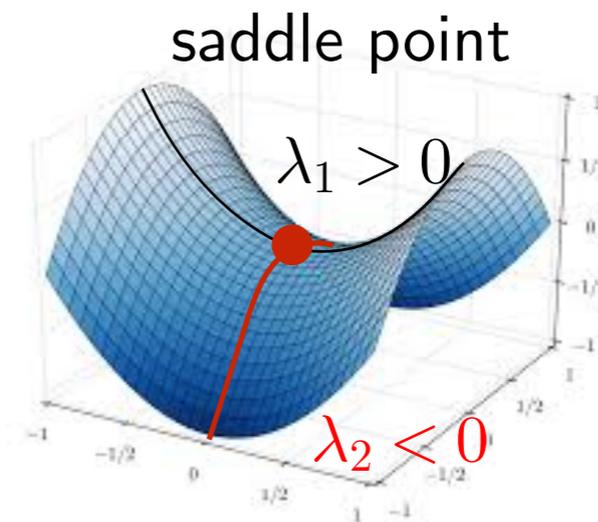
Stationary point: the gradient at x is zero

Saddle point: the gradient at x is zero but not a local extremum

1D



2D



Let $f(x + \Delta x) \approx f(x) + J\Delta x + \Delta x^T H \Delta x$

Let H have eigenvalues $\lambda_1, \dots, \lambda_n$

Index: α — the fraction of negative eigenvalues

$\alpha = 0 \Rightarrow$ local minimum

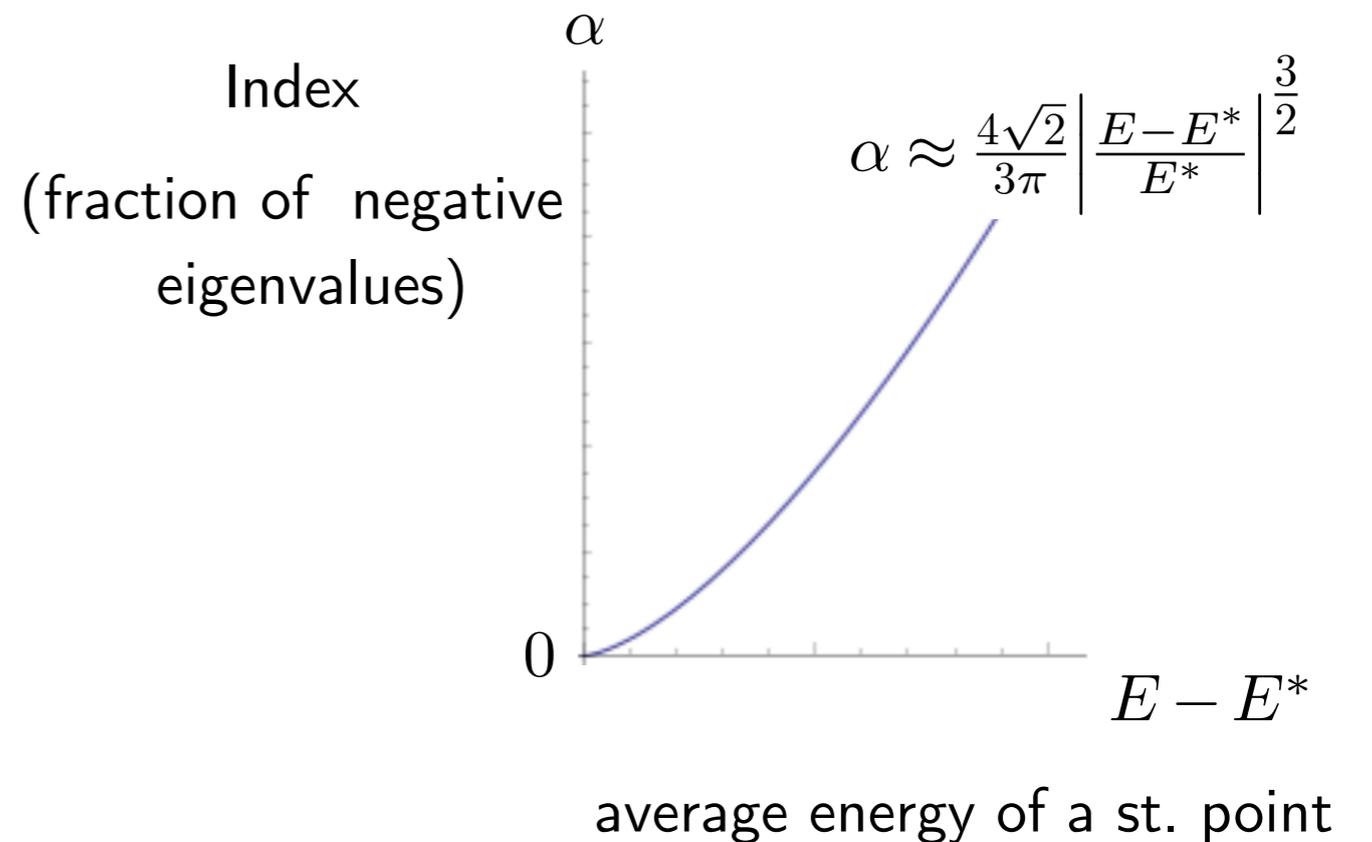
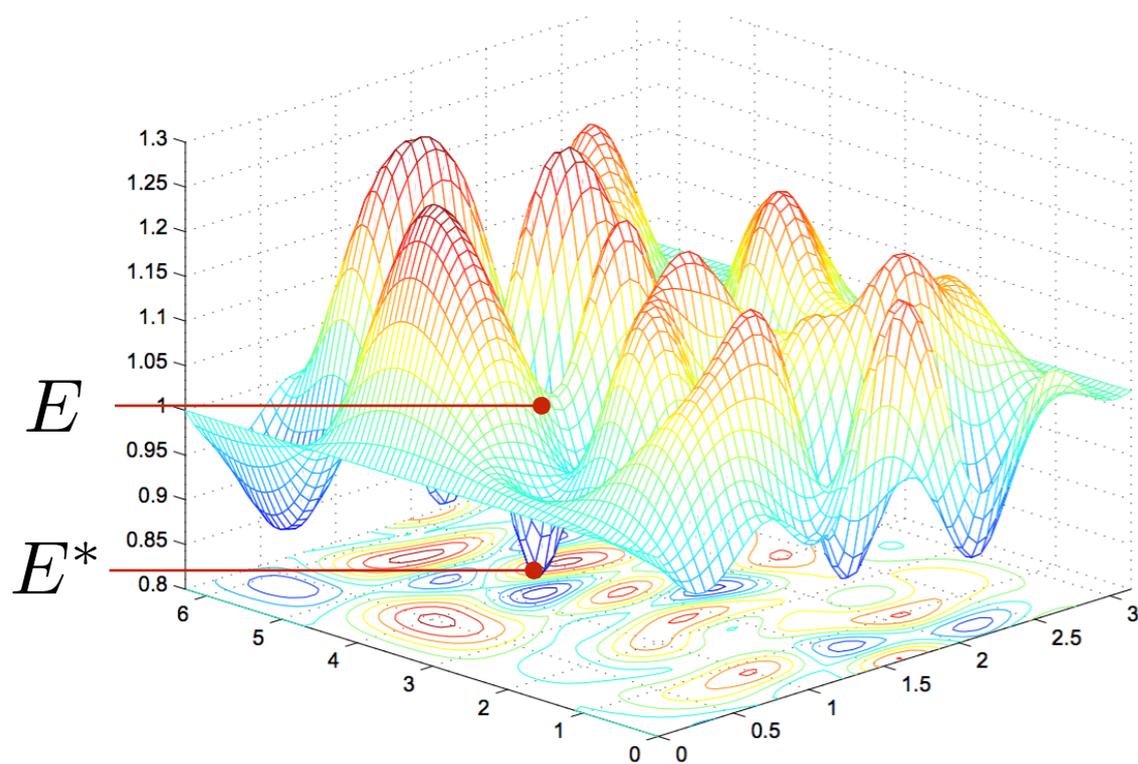
$\alpha = 1 \Rightarrow$ local maximum

$0 < \alpha < 1 \Rightarrow$ saddle point

Stationary Points in High Dimensions

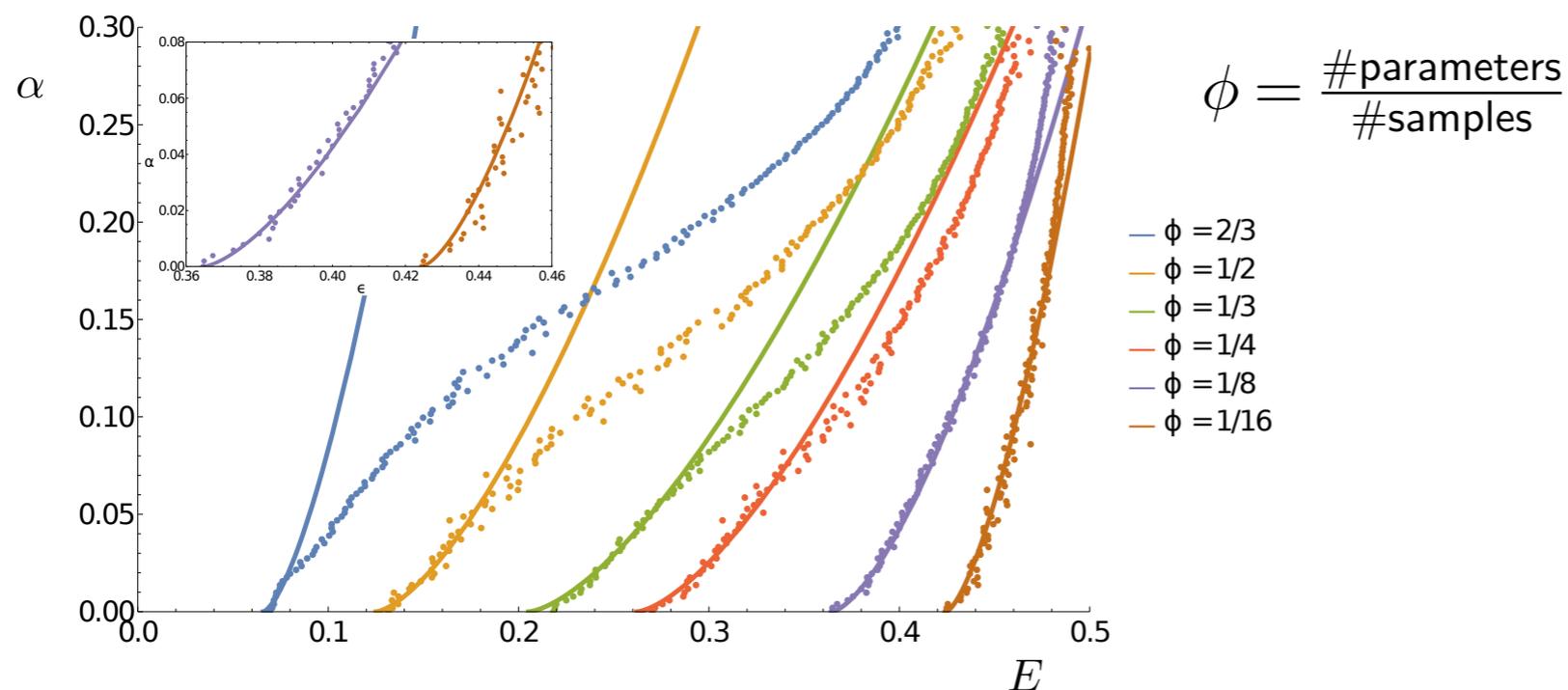
◆ Insights from Theoretical Physics --- Gaussian Random Fields:

- local minima are exponentially more rare than saddle points
- they become likely at lower energies (loss values)



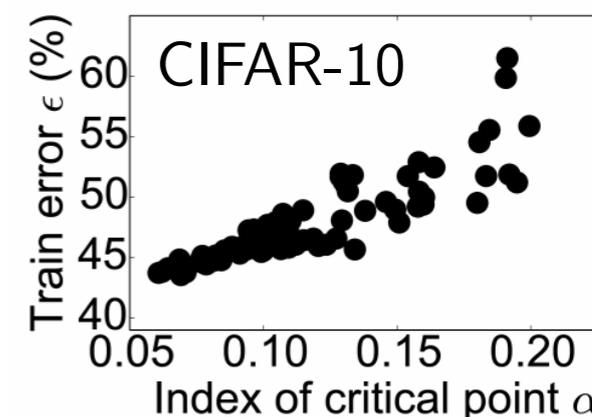
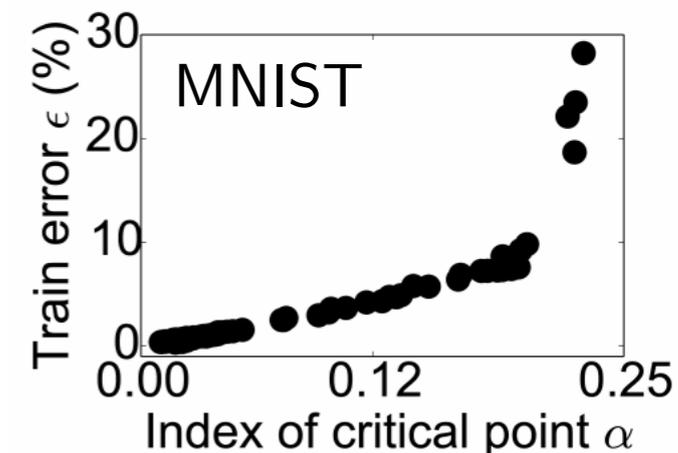
Stationary Points in High Dimensions

Experimental Confirmations in Neural Networks



[Pennington & Bahri 2017]

- 1 hidden layer
- good agreement for small alpha (as expected)

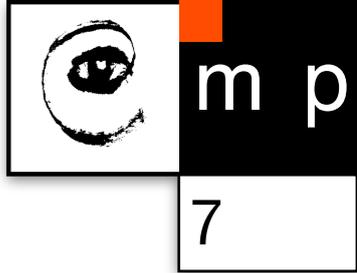


[Dauphin et. al. 2017]

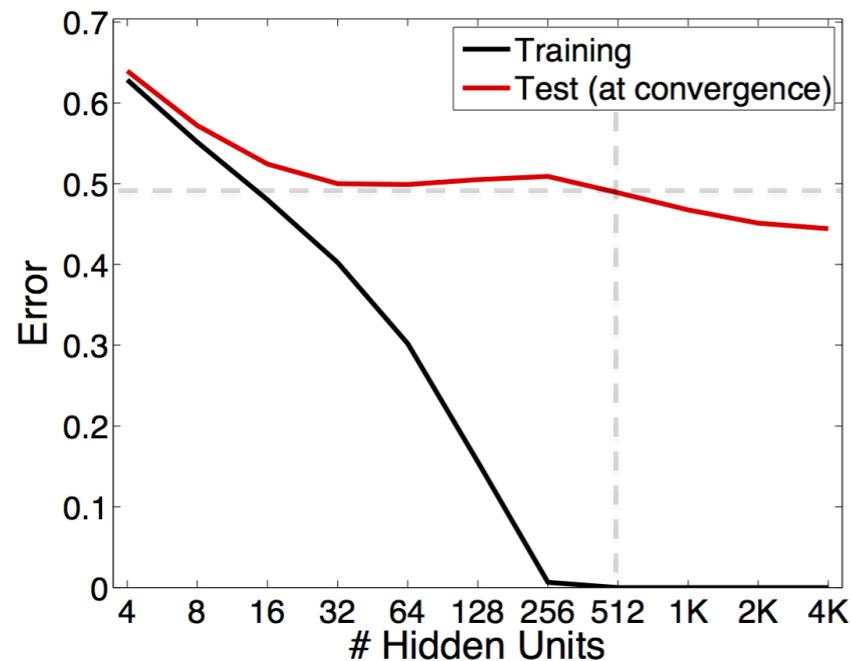
[Pennington & Bahri (2017) Geometry of Neural Network Loss Surfaces via Random Matrix Theory]

[Dauphin et. al. (2017) Identifying and attacking the saddle point problem in high-dimensional non-convex optimization]

High Dimensionality Helps Optimization

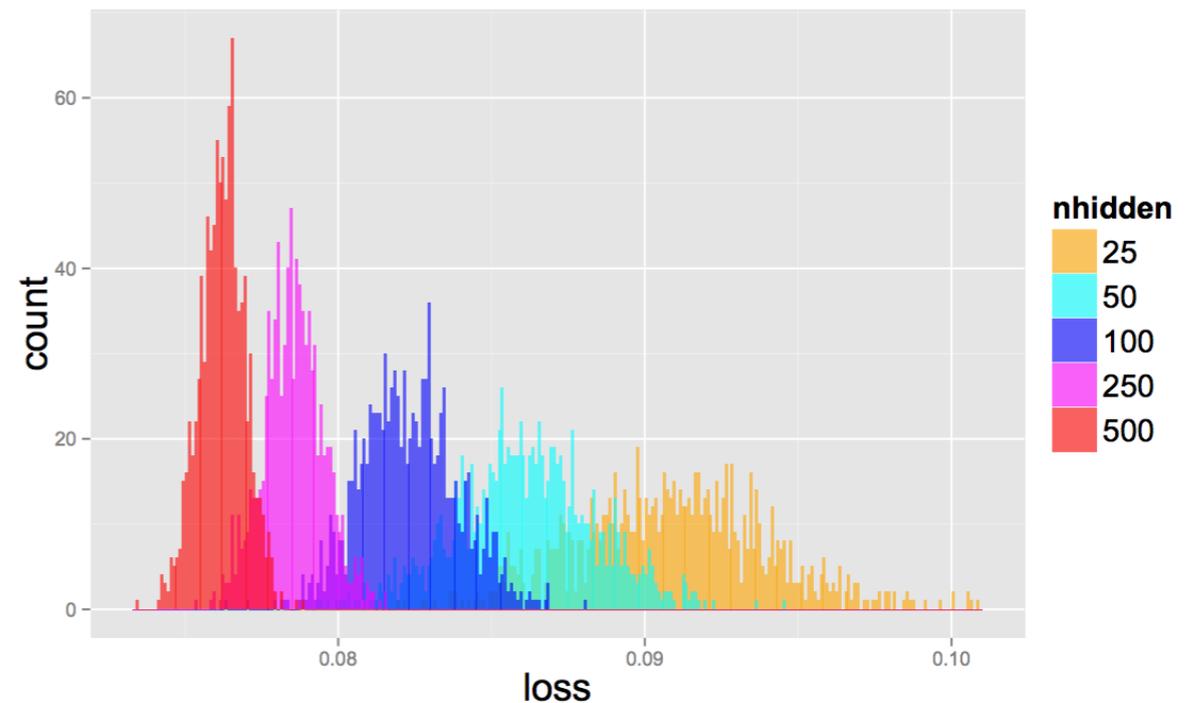


Achieve 0 training error
with sufficiently large networks



[Neyshabur (2015)]

Histogram of SGD trials (MNIST)



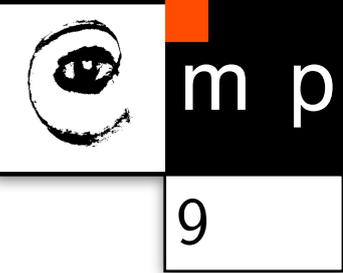
[Choromanska et al. (2015):
The Loss Surfaces of Multilayer Networks]

◆ Summary:

- Local minima are rare and appear to be good enough
- But we need (highly) over-parameterized models to have this easy training
- We hope that over-parameterized models will still generalize well
- Maybe, optimization should worry a bit about efficiency around saddle points

Adaptive Methods

Gradient Descent under Reparameterization



◆ Basic Example

- Want to minimize $f(x)$

By gradient descent: $x^{t+1} = x^t - \alpha f'(x^t)$, starting from x^0

- Make a change of variables: $y = 2x$

$$y^0 = 2x^0$$

$$g(y) = f(y/2)$$

$$g'(y) = 1/2 f'(y/2) = 1/2 f'(x)$$

- Perform gradient descent on g :

$$y^{t+1} = y^t - \alpha g'(y)$$

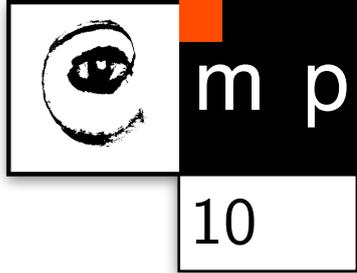
- Express back in x :

$$2x^{t+1} = 2x^t - \alpha \frac{1}{2} f'(x^t)$$

$$x^{t+1} = x^t - \alpha \frac{1}{4} f'(x^t).$$

- ◆ Substitution preserved the forward pass (equivalent initialization, same output)
- ◆ Substitution resulted in a different gradient
- ◆ We have many parameters, whose scales are chosen by architecture design and initialization

Need for Adaptive Methods



- ✦ In deep models we have:
 - **different kinds** of parameters: weights, biases, normalization parameters
 - located in **different layers**
- Some parameters may be more sensitive than other
- Some directions in the parameter space may be more sensitive (e.g. due to high curvature)
- ✦ Gradient Depends on the Choice of Coordinates
 - It is not necessarily the best direction for a step
- ✦ Many adaptive methods have emerged:

RMSProp	PAdam	AdamHD	...
Adagrad	NAdam	AmsGrad	...
Adam	AdamW	AdaDelta	...
BAdam	AdamX	Yogi	...
VAdam	Adamax

◆ Adagrad:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\varepsilon}{\sqrt{t}} \frac{\tilde{g}_{t,i}}{\sqrt{\text{Mean}(\tilde{g}_{1:t,i}^2)}}$$

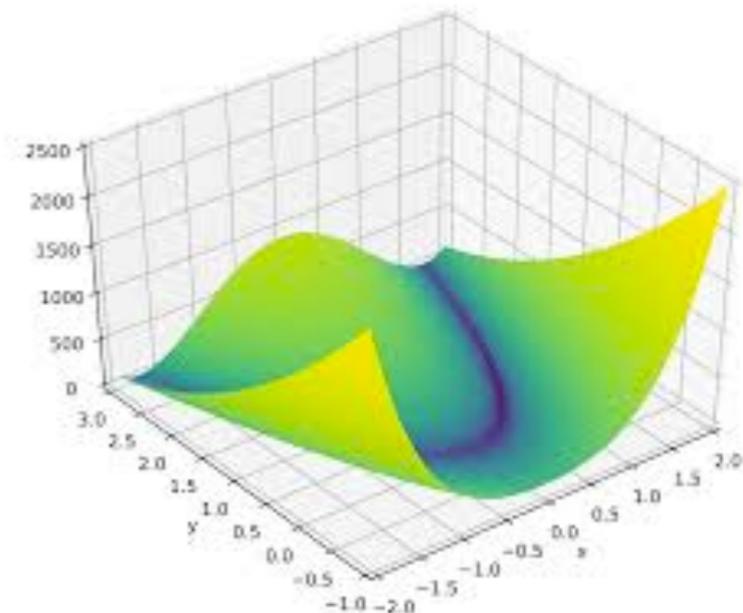
◆ RMSProp:

$$\theta_{t+1,i} = \theta_{t,i} - \varepsilon \frac{\tilde{g}_{t,i}}{\sqrt{\text{EWA}(\tilde{g}_{1:t,i}^2)}}$$

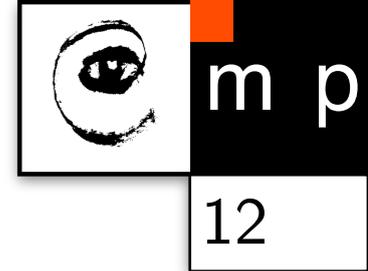
◆ Adam:

$$\theta_{t+1,i} = \theta_{t,i} - \varepsilon \frac{\text{EWA}_{\beta_1}(\tilde{g}_{1:t,i})}{\sqrt{\text{EWA}_{\beta_2}(\tilde{g}_{1:t,i}^2)}}$$

- All updates work per coordinate i independently
 - $\tilde{g}_{1:t,i}$ denotes the sequence of all past gradients
 - They are adaptive because each coordinate is rescaled differently
 - Mostly differ in the running averages used
- ◆ Often presented as approximate second order methods dealing with curvature -- has quite some gaps
- ◆ This lecture:
- discuss few general useful optimization ideas
 - that (hopefully) will provide insights for this design as well



Proximal Problem and Trust Region Problem



- ◆ Let's revisit how do we find the step Δx for SGD
 - Approximate: $f(x_0 + \Delta x) \approx f(x_0) + J\Delta x$. This approximation is local.
- ◆ Find the step by solving **Proximal Problem**:

$$\min_{\Delta x} \left(f(x_0) + J\Delta x \right) + \frac{\alpha}{2} \|\Delta x\|_2^2$$

$$\Delta x = -\alpha J^\top$$

- ◆ Alternatively, find the step by solving **Trust Region Problem**:

$$\min_{\|\Delta x\|_2 \leq \varepsilon} \left(f(x_0) + J\Delta x \right)$$

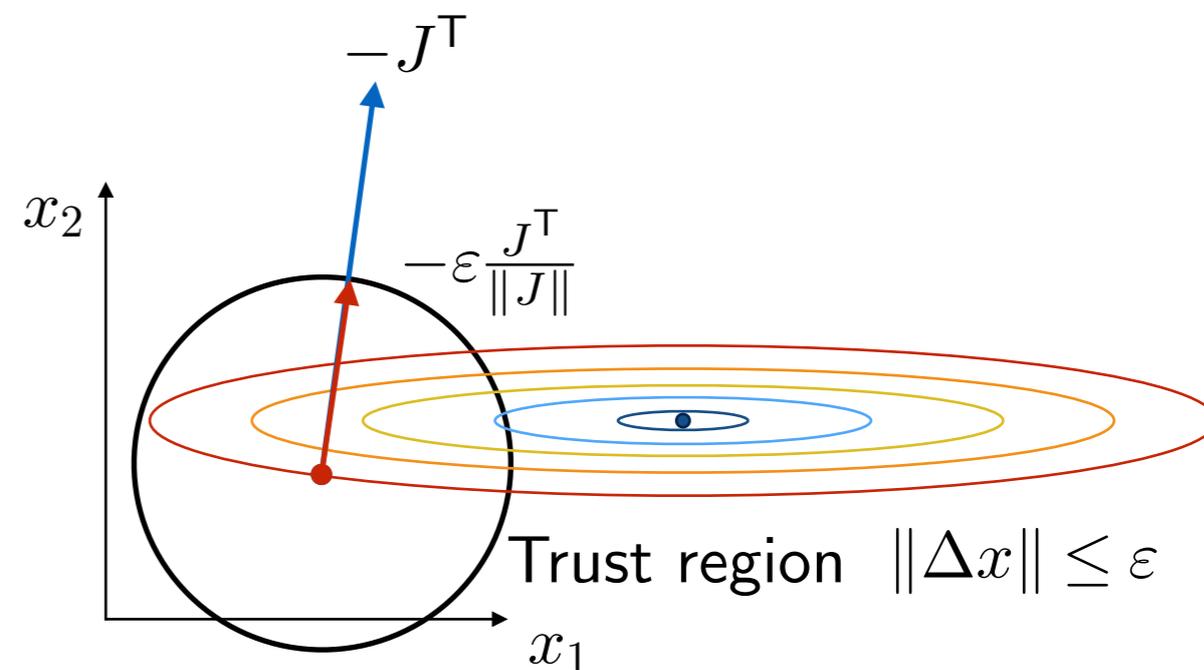
Equivalent to:

$$\max_{\lambda \geq 0} \min_{\Delta x} \left(J\Delta x + \lambda (\|\Delta x\|_2^2 - \varepsilon^2) \right)$$

$$\text{Step direction: } \Delta x = -\frac{1}{2\lambda} J^\top$$

$$\|\Delta x^\top\|^2 = \varepsilon^2 \rightarrow \lambda = \frac{1}{2\varepsilon} \|J\|_2$$

$$\text{Trust region step: } \Delta x = -\varepsilon \frac{J^\top}{\|J\|_2}$$



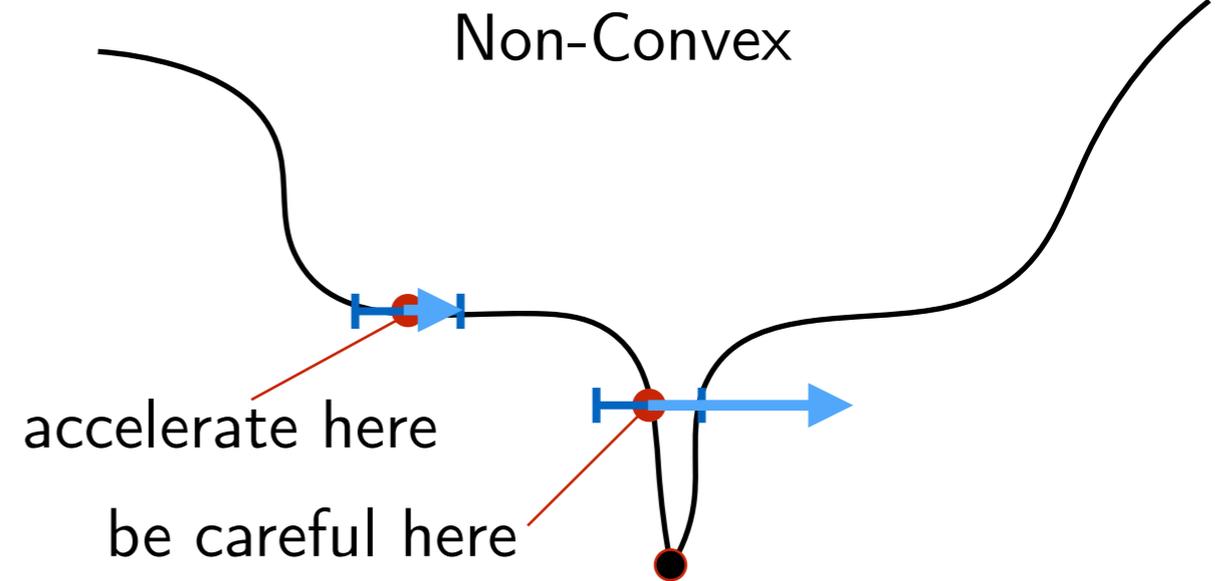
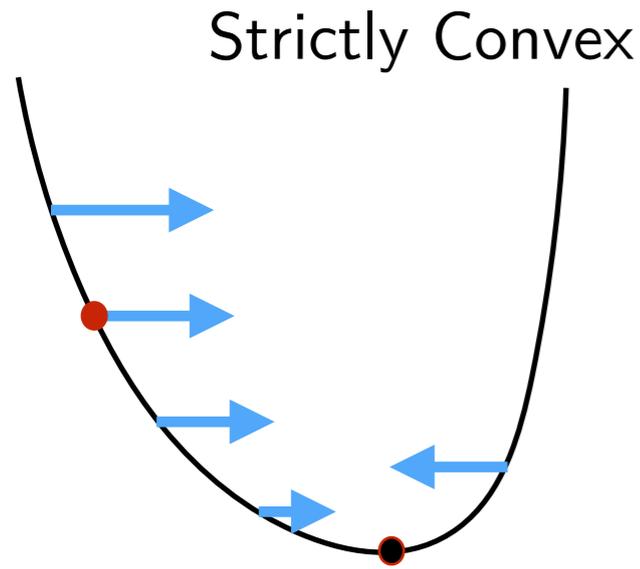
- ◆ We can choose the metric / trust region differently from Euclidean

Differences of Convex vs. Non-Convex



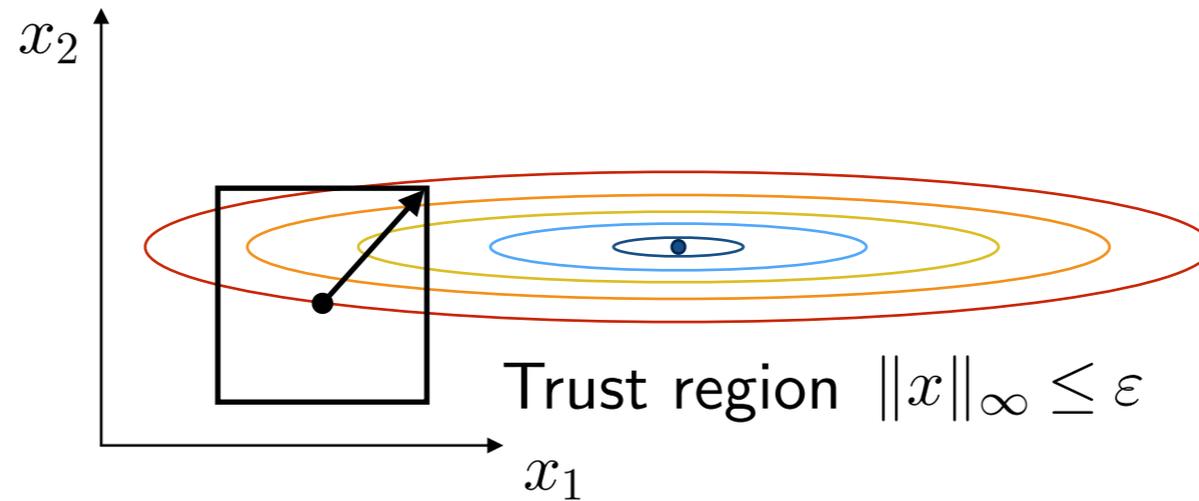
Why to step proportionally to the gradient:

Why to normalize:



- ◆ No other stationary points than global minima
- ◆ **The further we are from the optimum, the larger is the gradient:** $\exists \mu > 0$
 - $\|\nabla f(x)\|^2 \geq \mu(f(x) - f^*)$
 - $\|\nabla f(x)\| \geq \mu|x - x^*|$
- ◆ Negative gradient points towards the optimum:
 - $\langle -\nabla f, x^* - x \rangle \geq f - f^* + \tilde{\mu}\|x - x^*\|^2$
 - Optimization need not be monotone in f

- ◆ Gradient carries no global information
 - Need bigger steps where gradient and curvature are low
 - Need smaller steps when gradient and curvature are high
- ◆ Makes sense to use **trust region steps**:
 - $\Delta x = -\frac{\nabla f}{\|\nabla f\|}$
 - If the trust region is ok, should guarantee a steady progress



◆ This time solve for step as:

- $\min_{\|\Delta x_i\| \leq \varepsilon \forall i} (f(x_0) + J\Delta x)$

(In overparametrized models expect many parameters to have independent effect)

- Equivalent to:

$$\max_{\lambda \geq 0} \min_{\Delta x} \left(J\Delta x + \sum_i \lambda_i (\|\Delta x_i\|^2 - \varepsilon^2) \right)$$

$$2\lambda_i \Delta x_i = -J_i$$

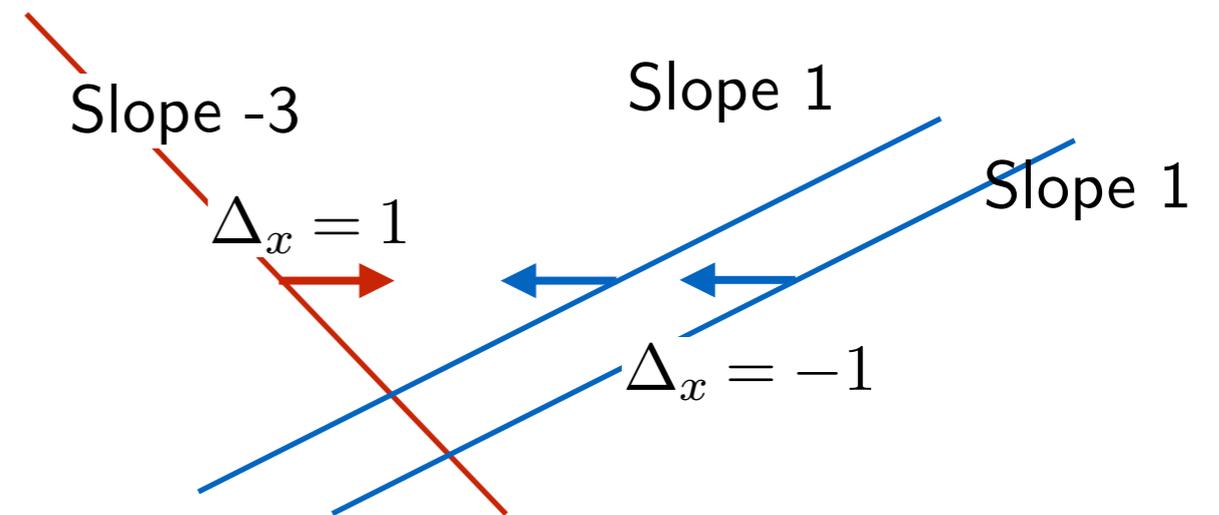
Step direction: $\Delta x_i = -\frac{1}{2\lambda_i} (\nabla f(x))_i$

Trust region step: $\Delta x_i = -\varepsilon \frac{(\nabla f(x))_i}{|(\nabla f(x))_i|}$

- ◆ Trust region steps: $\Delta x_i = -\varepsilon \frac{(\nabla f(x))_i}{|(\nabla f(x))_i|}$
- ◆ Problem: breaks in the stochastic setting
- ◆ Example

$f(x) = (-3x) + (x) + (x + 1)$, chose 1 summand at a time with equal probability

If we normalize stochastic gradients,
will move in the wrong direction!



- ◆ Want the steps to follow the descent direction on average
 - Cannot adjust the stochastic gradient “too much nonlinearly”
 - This example was used to show that Adam may fail to converge to a stationary point and motivated theoretical improvements

- ◆ **Practical Solution:** approximate expectations with running averages:

$$\Delta x = -\varepsilon \frac{\mathbb{E}[\nabla f]}{\|\mathbb{E}[\nabla f]\|}$$

Furhter approximate $\|\mathbb{E}[\nabla f]\| = \sqrt{(\mathbb{E}[\nabla f])^2} \leq \sqrt{(\mathbb{E}[(\nabla f)^2])}$

- ◆ **Adagrad:**

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\varepsilon}{\sqrt{t}} \frac{\tilde{g}_{t,i}}{\sqrt{\text{Mean}(\tilde{g}_{1:t,i}^2)}}$$

- ◆ **RMSProp:**

$$\theta_{t+1,i} = \theta_{t,i} - \varepsilon \frac{\tilde{g}_{t,i}}{\sqrt{\text{EWA}(\tilde{g}_{1:t,i}^2)}}$$

- ◆ **Adam:**

$$\theta_{t+1,i} = \theta_{t,i} - \varepsilon \frac{\text{EWA}_{\beta_1}(\tilde{g}_{1:t,i})}{\sqrt{\text{EWA}_{\beta_2}(\tilde{g}_{1:t,i}^2)}}$$

- In Adagrad:

$\frac{1}{\sqrt{t}}$ guarantees convergence. Other methods would also need this in theory but are typically presented and used with constant ε

The flat average appears not very practical

- In Adam:

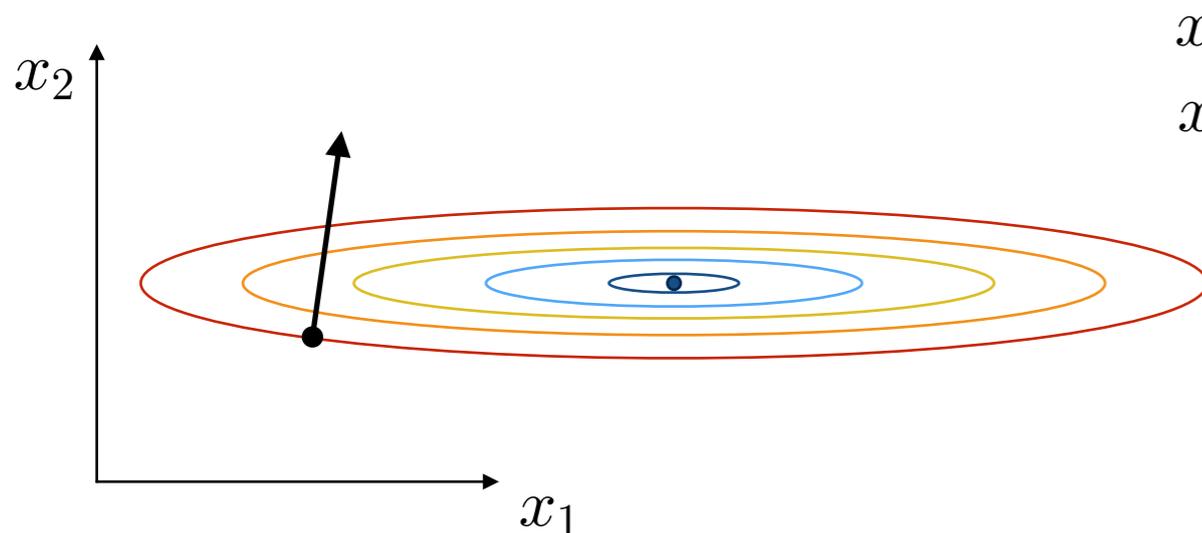
EWA with $\beta_1 = 0.9$ works as common momentum (20 batches averaging)

EWA with $\beta_2 = 0.999$ (2000 batches averaging) makes the normalization smooth enough

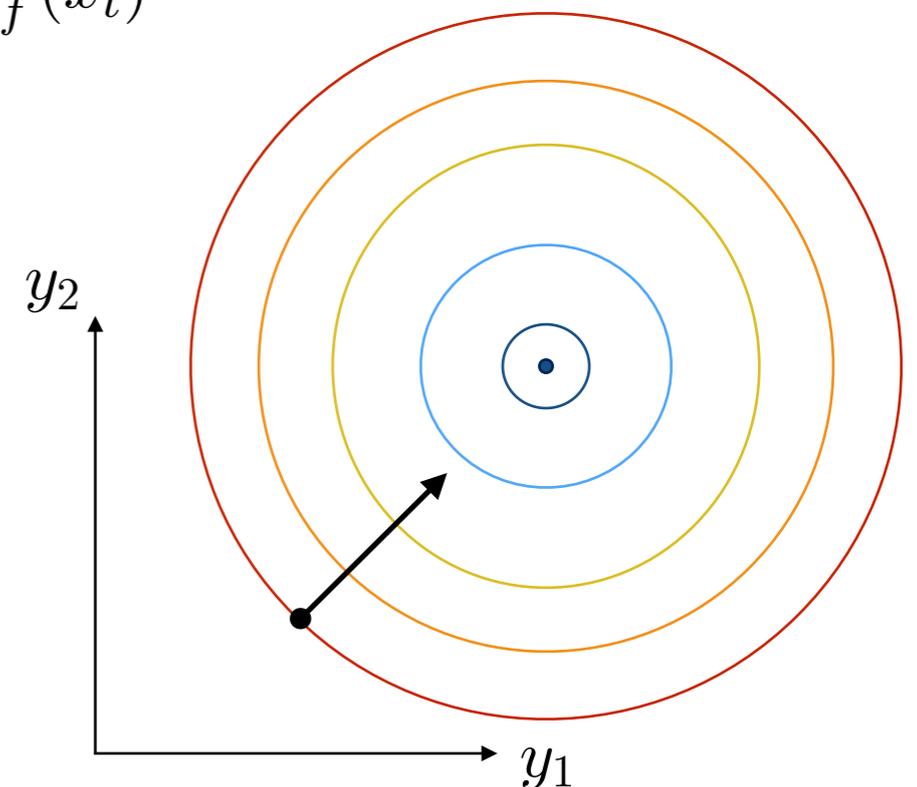
Changing the Metric

General Change of Coordinates

- ◆ Consider the simple gradient descent for a function $f: \mathbb{R}^n \rightarrow \mathbb{R}$:
 - $\min_{x \in \mathbb{R}^n} f(x)$
 - $x_{t+1} = x_t - \alpha J_f^\top(x)$
- ◆ Make a substitution: $x = Ay$ (change of coordinate) and write GD in y :
 - $\min_{y \in \mathbb{R}^n} f(Ay)$
 - $y_{t+1} = y_t - \alpha A^\top J_f^\top(Ay_t)$
- ◆ Substitute back $y = A^{-1}x$:
 - $A^{-1}x_{t+1} = A^{-1}x_t - \alpha A^\top J_f^\top(x_t)$
 - Obtained **preconditioned** GD: $x_{t+1} = x_t - \alpha(AA^\top)J_f^\top(x_t)$
 - $P = AA^\top$ – positive semidefinite
 - $P\nabla f(x)$ – is a descent direction



$$\begin{aligned}x_1 &= y_1 \\x_2 &= 5y_2\end{aligned}$$

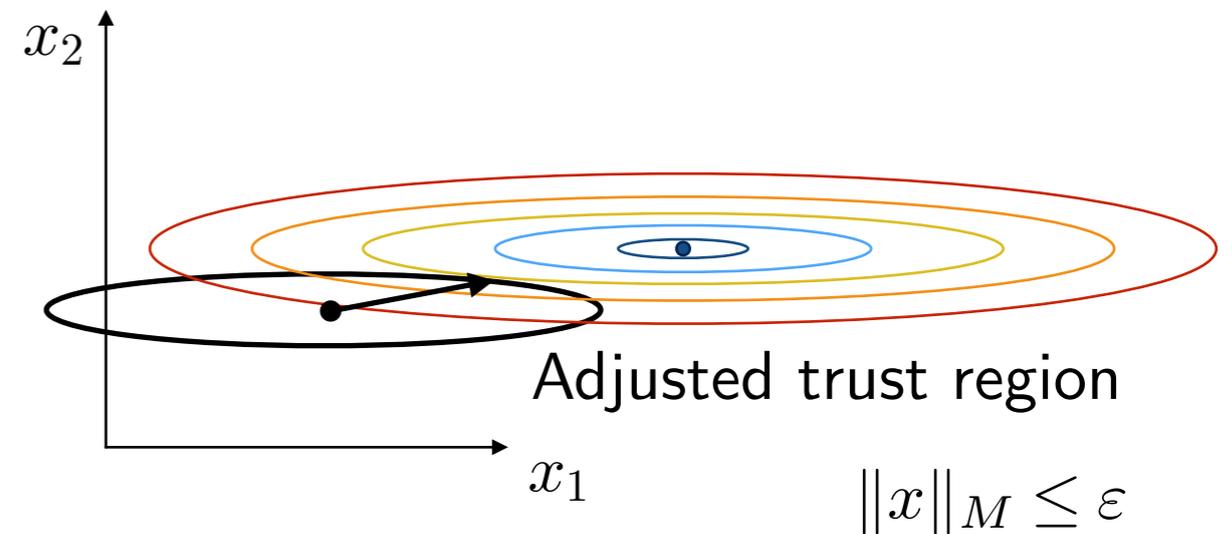


- ◆ Similar for non-linear change of coordinates, e.g. normalization

◆ Adjust the proximal problem for sensitivity in different parameters:

- $\min_{\Delta x} (f(x_0) + J\Delta x) + \frac{\alpha}{2} \|\Delta x\|_M$
- $\|\Delta x\|_M = (\Delta x^\top M \Delta x)^{\frac{1}{2}}$ – Mahalanobis distance

Optimal step: $\Delta x = -\frac{1}{2\lambda} M^{-1} \nabla f(x)$



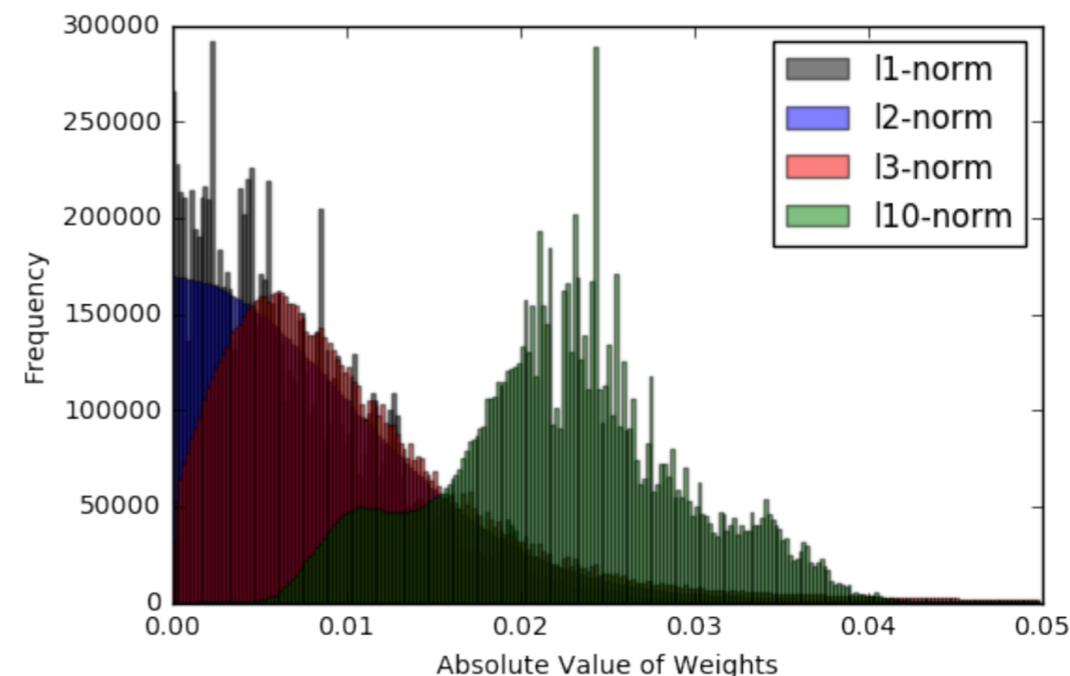
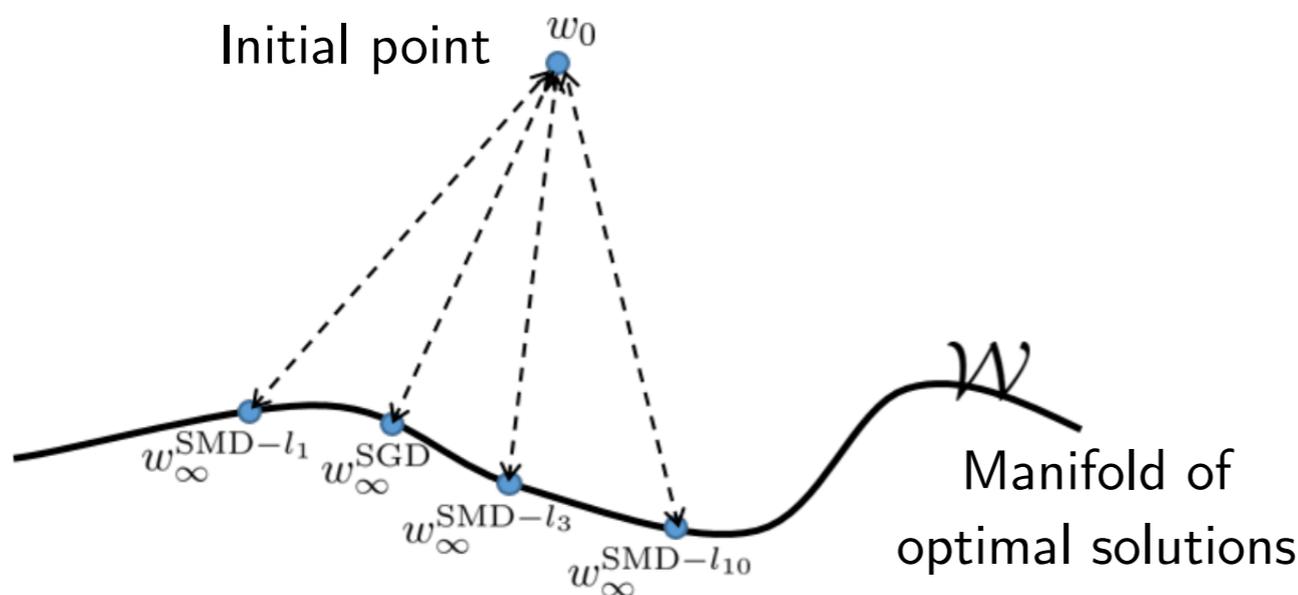
◆ Equivalent to change of coordinates/ preconditioning

- More intuitive, when setting M for each step differently
- If we set M to (approximate) Hessian \rightarrow Newton Methods
- Can set M using statistics of gradient oscillations (directions in which the gradient changes fast need a slower learning rate), e.g.

Adagrad: $M = \text{Diag}\left(\sqrt{\text{Mean}(g_{1:t}^2)}\right)$

Implicit Regularization by SGD / SMD

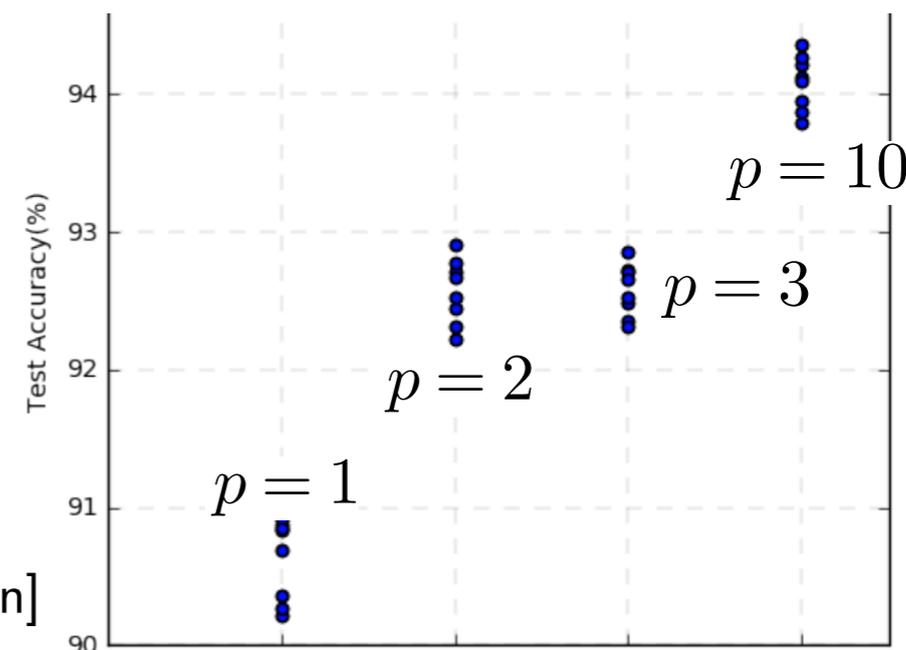
- ◆ Consider step proximal problem: $\min_x \langle \nabla f(x_0), x - x_0 \rangle + \lambda \|x - x_0\|_p^p$
 - i.e., p -norm stochastic mirror descent
- ◆ Using different p leads to solutions with different properties



- Iterates tend to $\operatorname{argmin}_{w \in \mathcal{W}} \|w - w_0\|_p^p$, the closest point in the respective norm

	SMD 1-norm	SMD 2-norm (SGD)	SMD 3-norm	SMD 10-norm
1-norm BD	141	9.19×10^3	4.1×10^4	2.34×10^5
2-norm BD	3.15×10^3	562	1.24×10^3	6.89×10^3
3-norm BD	4.31×10^4	107	53.5	1.85×10^2
10-norm BD	6.83×10^{13}	972	7.91×10^{-5}	2.72×10^{-8}

- Different sparsity and generalization

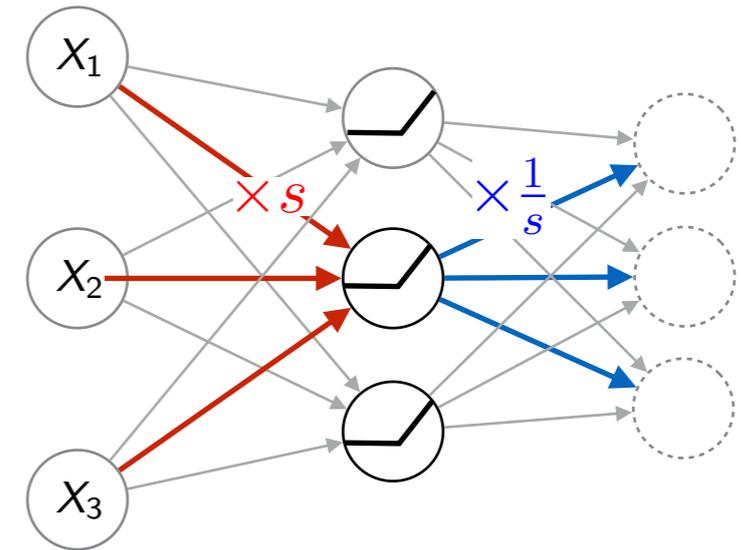


[Azizan et al. (2019) Stochastic Mirror Descent on Overparameterized Nonlinear Models: Convergence, Implicit Regularization, and Generalization]

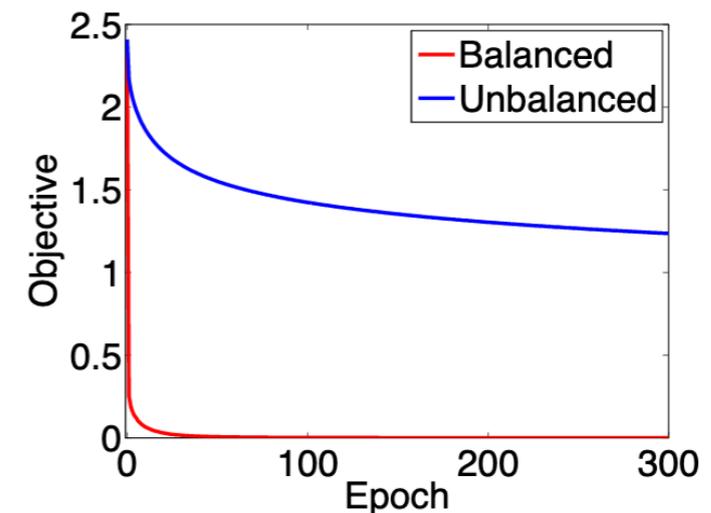
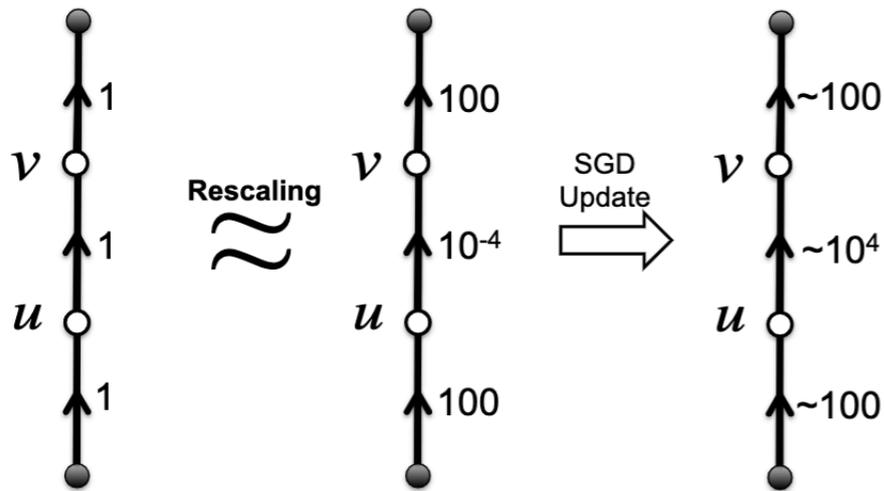
✦ In ReLU networks we can rescale the weights without affecting the output:

- ReLU units are *1-homogenous*:
for $s > 0$: $\text{ReLU}(sx) = \max(0, sx) = s \max(0, x)$
- Can rescale inputs and outputs of each unit (channels in conv networks)

$$f(Ay) = f(y), \text{ but } J_f(Ay) \neq J_f(y)$$



✦ Can lead to completely different SGD behavior



(a) Training on MNIST

✦ Path-SGD considers metric invariant to equivalent transformations.

$$\text{Prox. problem: } \arg \min_w \eta \langle \nabla L(w^{(t)}), w \rangle + \left(\sum_{v_{in}[i] \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \dots \xrightarrow{e_d} v_{out}[j]} \left(\prod_{k=1}^d w_{e_k} - \prod_{k=1}^d w_{e_k}^{(t)} \right)^p \right)^{2/p}$$

Mirror Descent

◆ Mirror Descent (**MD**)

- General step proximal problem:

$$\min_x \langle \nabla f(x_0), x - x_0 \rangle + \lambda D(x, x_0)$$

where D is Bregman divergence. Properties:

$$D(x, x_0) \geq 0$$

$$D(x, x_0) = 0 \text{ iff } x = x_0$$

$$D(x, x_0) \text{ is convex in } x$$

- We will consider algorithms using unnormalized steps (not solving for λ).
- Generalizes cases considered so far:

$$D = \|x - x_0\|^2 \text{ — (steepest) SGD}$$

$$D = \|x - x_0\|_M^2 \text{ — preconditioned SGD}$$

Constrained Optimization with Mirror Descent



◆ Let us use a proximal problem with an appropriate trust region

◆ Mirror Descent (**MD**)

- Use step proximal problem: $\min_x \langle \nabla f(x_0), x - x_0 \rangle + \lambda D(x, x_0)$

with a suitable divergence D

(recall previous choices $D = \|x - x_0\|^2$, $D = \|x - x_0\|_M^2$)

- Very elegant solutions in simple cases

◆ Example: constrained parameter $x > 0$

$$D(x, x_0) = x \log \frac{x}{x_0} - x + x_0 \text{ (Generalized KL divergence)}$$

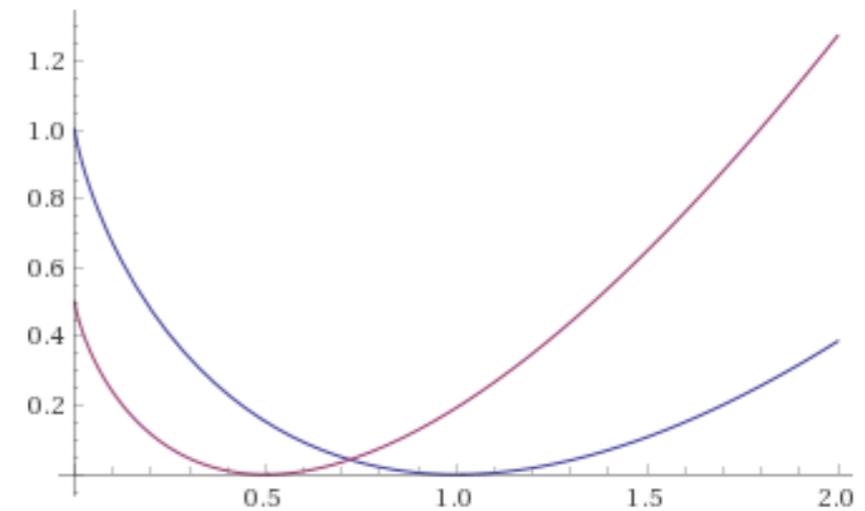
$$\text{Update: } \log x_{t+1} = \log x_t - \frac{1}{\lambda} \nabla_x f(x_t)$$

Note: **gradient in x** is added to **$\log x$**

Can implement as:

$$y_{t+1} = y_t - \frac{1}{\lambda} \nabla_x f(x_t)$$

$$x_{t+1} = e^{y_{t+1}}$$



Constrained Optimization with Mirror Descent



◆ Let us use a proximal problem with an appropriate trust region

◆ Mirror Descent (**MD**)

• Use step proximal problem: $\min_x \langle \nabla f(x_0), x - x_0 \rangle + \lambda D(x, x_0)$

with a suitable divergence D

(recall previous choices $D = \|x - x_0\|^2$, $D = \|x - x_0\|_M^2$)

• Very elegant solutions in simple cases

◆ Constraint $x \in (0, 1)$

$$D(x, x_0) = x \log \frac{x}{x_0} + (1 - x) \log \frac{1-x}{1-x_0} \text{ (KL divergence)}$$

$$y_{t+1} = y_t - \frac{1}{\lambda} \nabla_x f(x_t)$$

$$x_{t+1} = \mathcal{S}(y_{t+1}) = \frac{1}{1 + e^{-y_{t+1}}}$$

