Deep Learning (BEV033DLE)
Lecture 9
Adaptive SGD Methods

Alexander Shekhovtsov

Czech Technical University in Prague

- Geometry of Neural Network Loss Surfaces
  - Local Minima and Saddle Points in nD
  - Parameter redundancy helps optimization
- Adaptive Methods
  - Change of Coordinates, Preconditioning, Trust Region
  - Equivalent reparameterizations
  - Adam
- Handling simple constraints - Mirror Descent

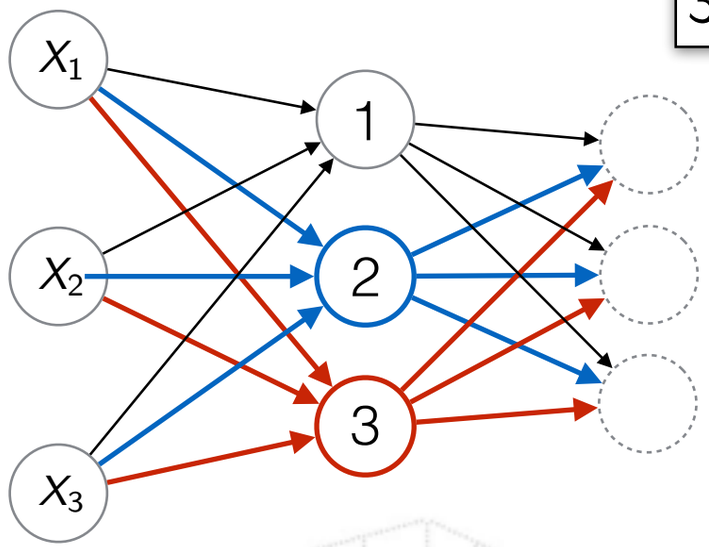
Loss Landscape

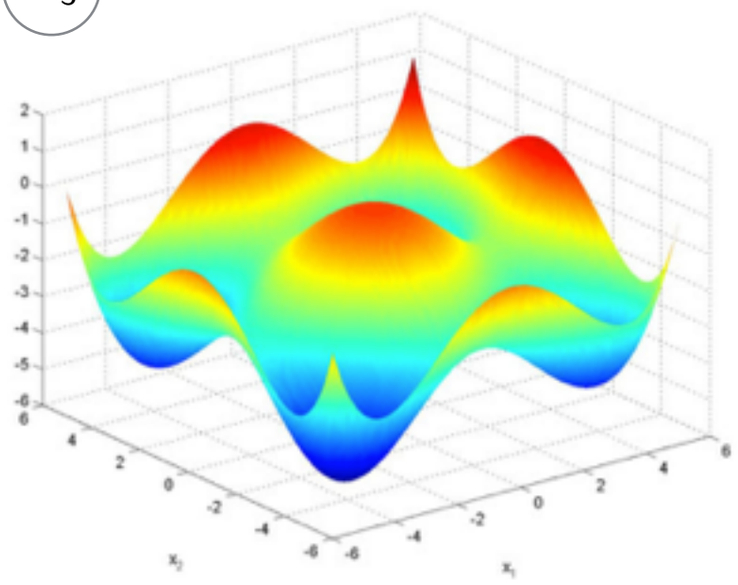
Local Minima

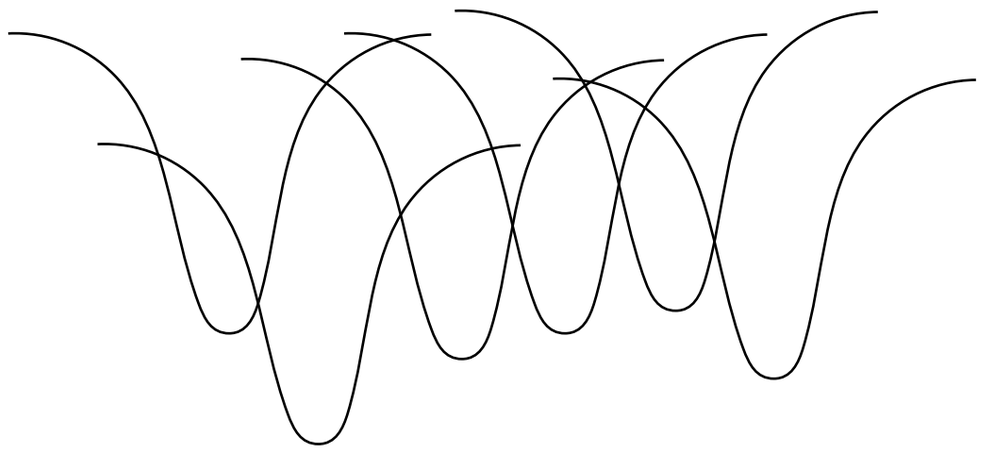
- There are several reasons for local minima
  - Permutation invariances (symmetries)
    - fully connected with n hidden units: n! permutations
    - convolutional with c channels: c! permutations
    - total number of local minima is the product of these
  - But all these are equally good for us - not a problem
  - Loss function is a sum of many terms:

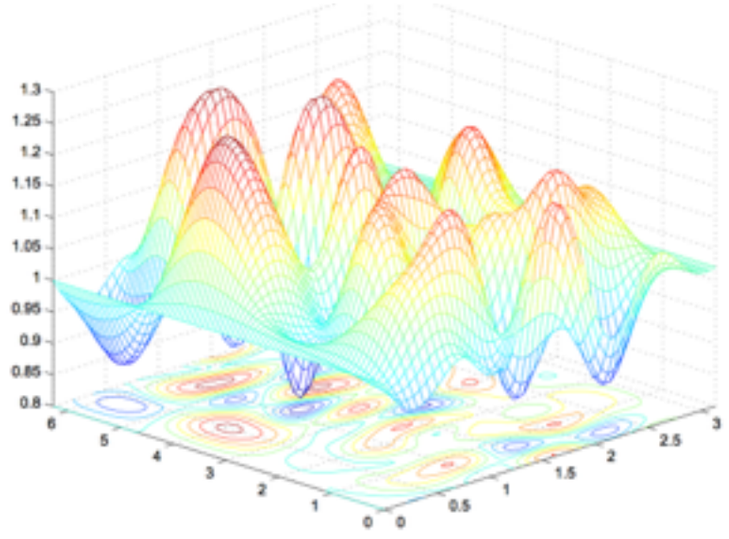
$$L(\theta) = \sum_i l(y_i, f(x_i; \theta))$$

often convex non-linear





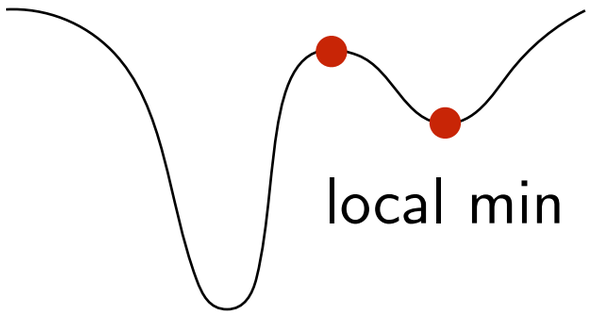




Local Minima in High Dimension

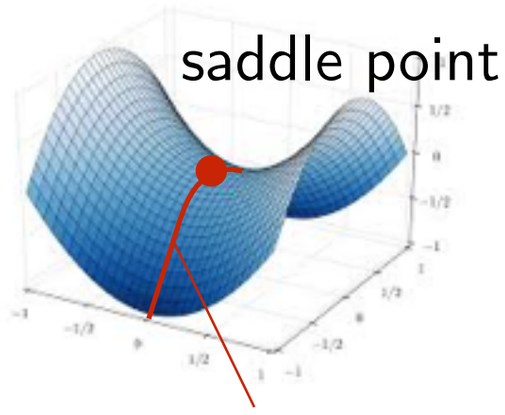
1D

local max



local min

2D



local min in one dimension still can descent

nD

$$f(x+\Delta x) \approx f(x) + J\Delta x + \Delta x^{\mathsf{T}} H \Delta x$$

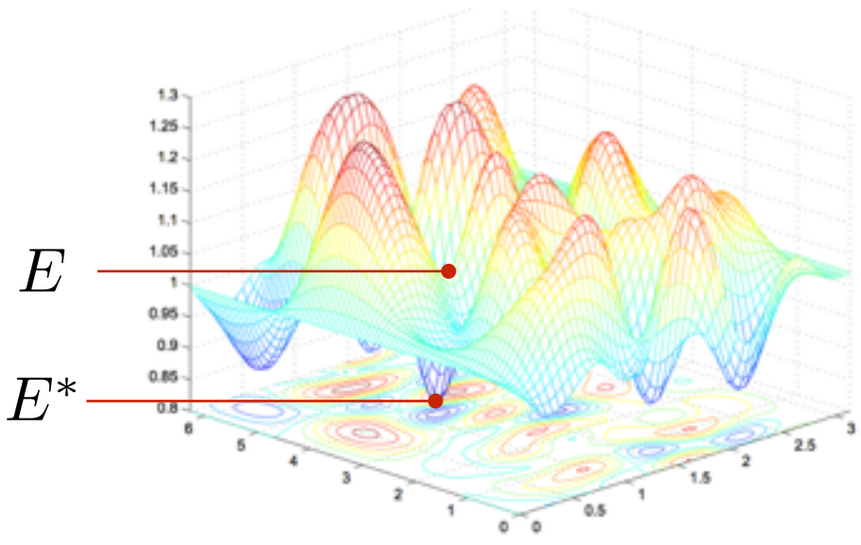
Eigenvalues of H: $\lambda_1, \ldots \lambda_2$

Stationary point: gradient is zero

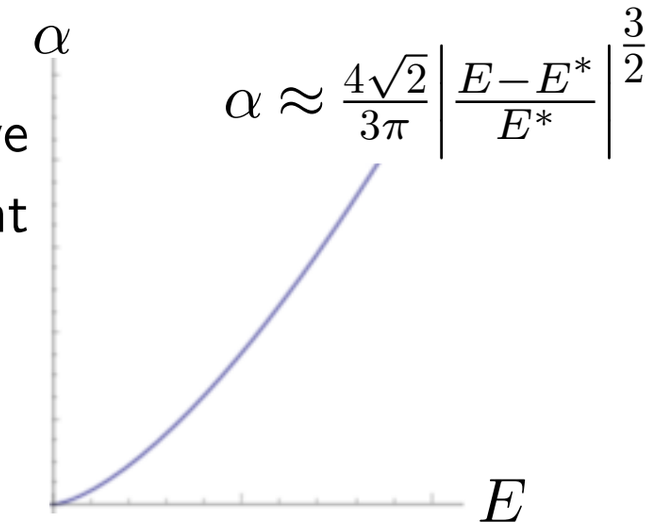
Saddle point: st. point and a fraction α of eigenvalues is negative

Local min: st. point and all eigenvalues are positive ($\alpha = 0$)

- Gaussian Random Fields [Ray & Dean 2007]:
  - local minima are exponentially more rare than saddle points
  - they become likely at lower energies (loss values)



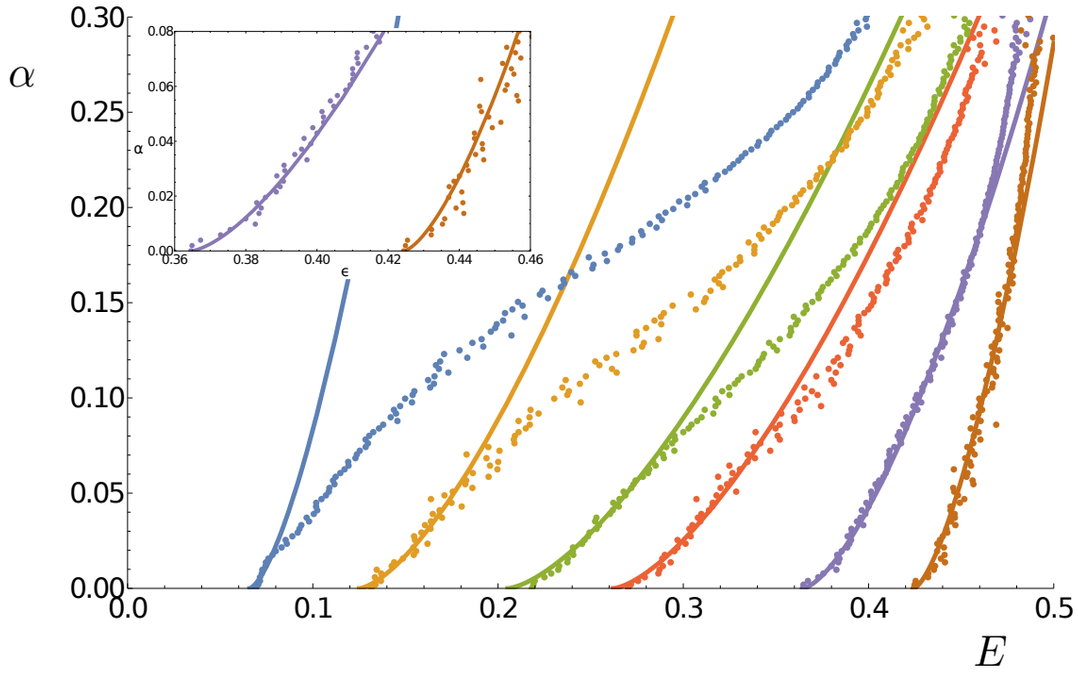
fraction of negative eigenvalues at st. point



average energy of st. point

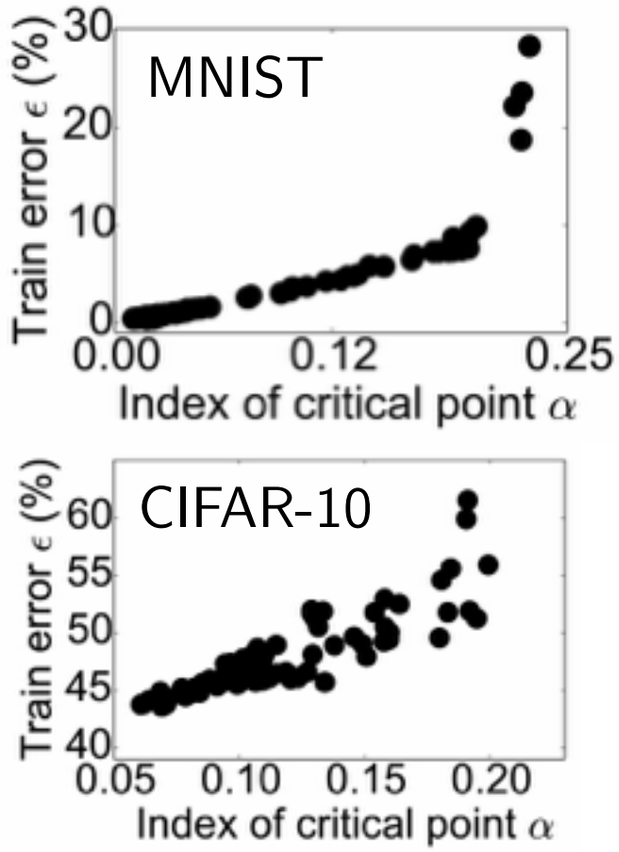
Local Minima in High Dimension

- Experiments for neural networks are in a good agreement with the above theory



[Pennington & Bahri 2017]

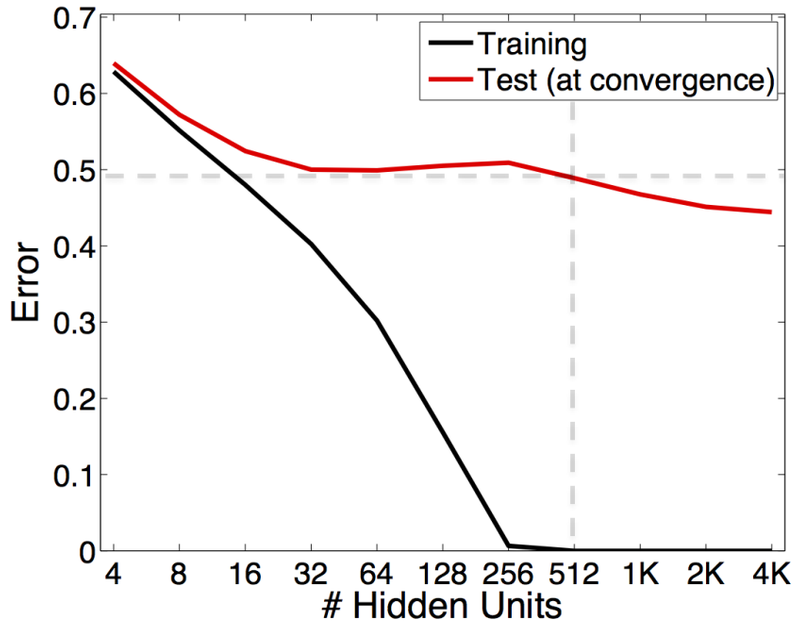
(1 hidden layer, good agreement for small alpha)



[Dauphin et. al. 2017]

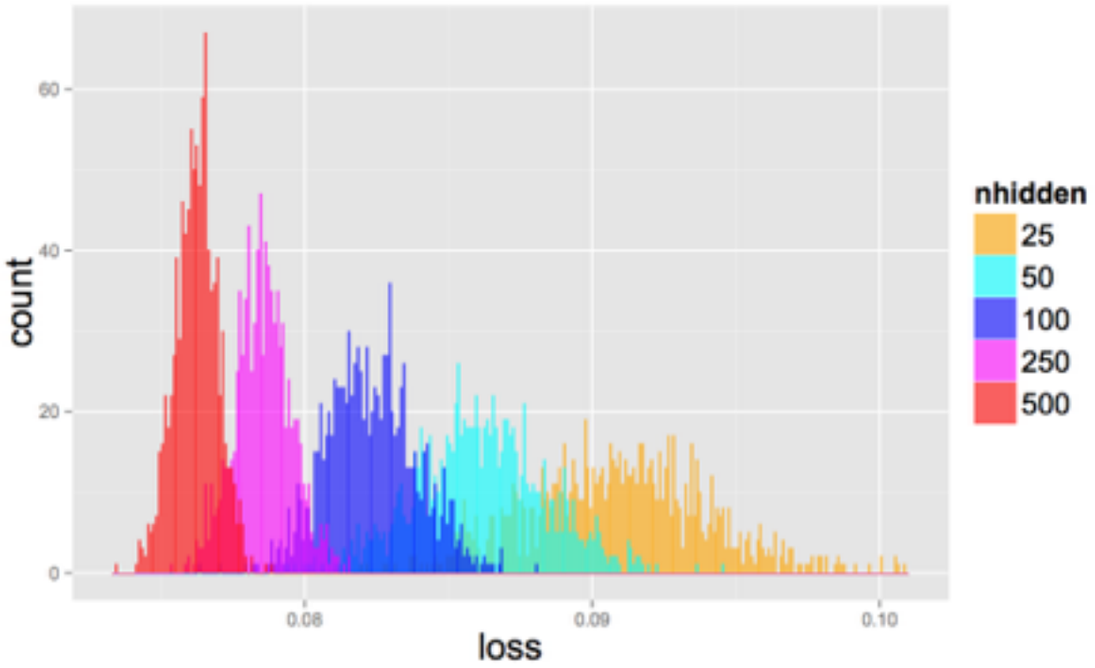
High Dimensionality Helps Optimization

Achieve 0 training error
with sufficiently large networks



[Neyshabur (2015)]

Hist of SGD trials



[Choromanska et al. (2015)]

✦ Summary:

- Local minima are rare and appear to be good enough
  (note, we just waved an NP-hard non-convex optimization problem)
- But we need (highly) overparametrized models to have this easy training
  (and hope that they will still generalize well)
- Maybe, optimization should worry a bit about efficiency around saddle points

Adaptive Methods

Need for Adaptive Methods

- In a deep model 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 Step Depends on the Choice of Coordinates
  - It is not necessarily the best direction for a step
- Many adaptive methods have emerged:

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

Common Adaptive Methods

- **Adagrad**:

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

- **RMSProp**:

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

- **Adam**:

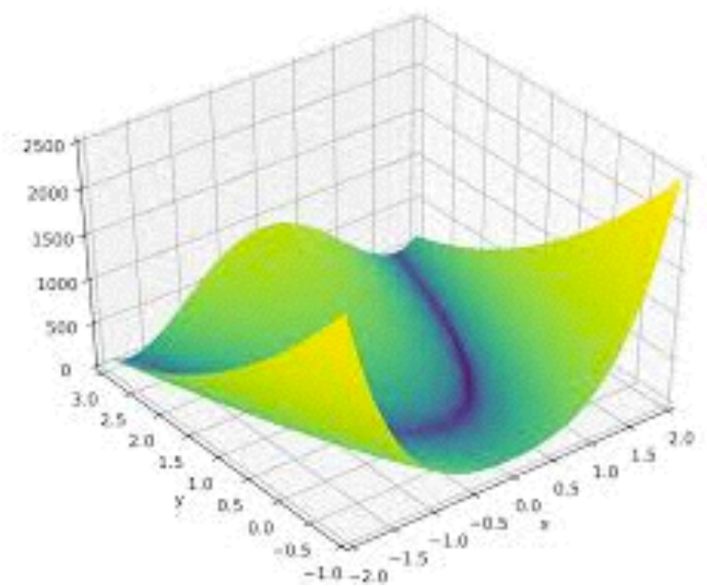
$$\theta_{t+1,i} = \theta_{t,i} - \varepsilon \frac{\mathsf{EWA}_{\beta_1}\left(\tilde{g}_{1:t,i}\right)}{\sqrt{\mathsf{EWA}_{\beta_2}\left(\tilde{g}^2_{1:t,i}\right)}}$$

- 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 by running averages used

✦ While they do work better for functions with valleys, explaining them as second order methods (dealing with curvature) has quite some gaps

✦ This lecture:

- consider some general useful optimization ideas
- that (hopefully) will provide insights for this design as well



Proximal Problem and Trust Region

- How did we find the steepest descent direction?

  Recall from backprop lecture:

  - Linearize: $f(x_0+\Delta x) \approx f(x_0) + J\Delta x$
  - Find the best improvement per length $\|\Delta x\|$

- Solve the step **proximal problem**:

  - $\min_{\|\Delta x\| \le \varepsilon} \left(f(x_0) + J\Delta x\right)$ for given ε

    Equivalent to:

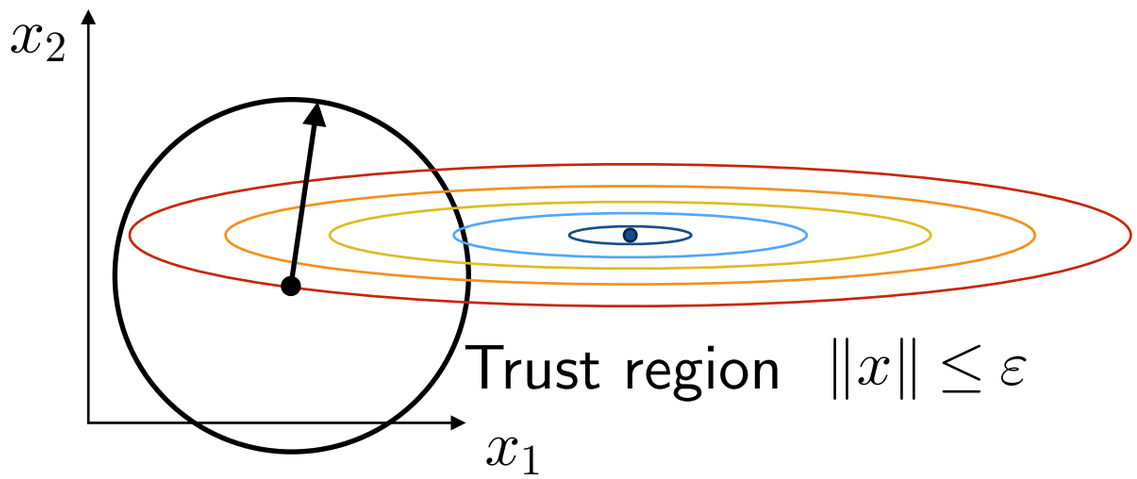
    $$\max_{\lambda} \min_{\Delta x} \left(J\Delta x + \lambda(\|\Delta x\|^2 - \varepsilon^2)\right)$$

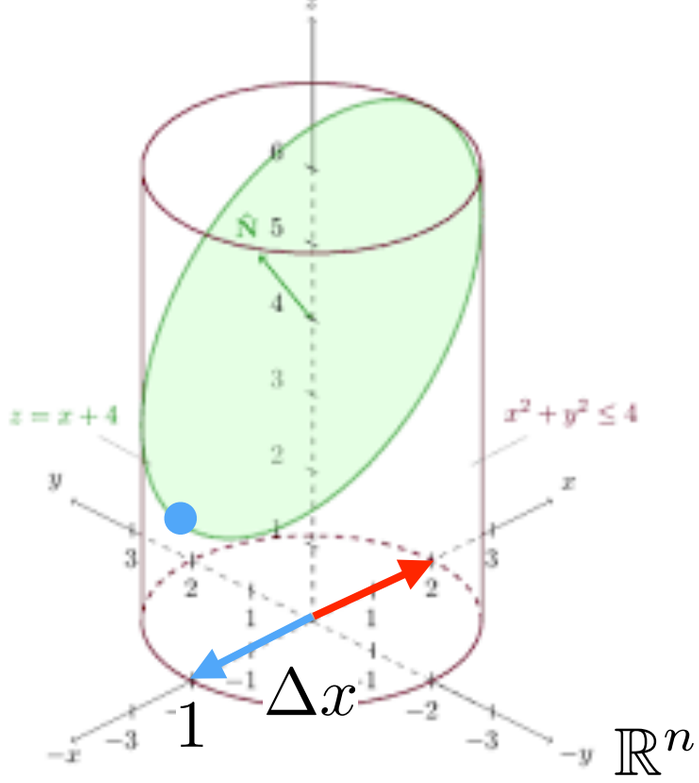
    $$2\lambda \Delta x^\mathsf{T} = -J$$

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

    $$\|\Delta x^\mathsf{T}\|^2 = \varepsilon^2 \rightarrow \lambda = \frac{1}{2\varepsilon}\|\nabla f(x)\|$$

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

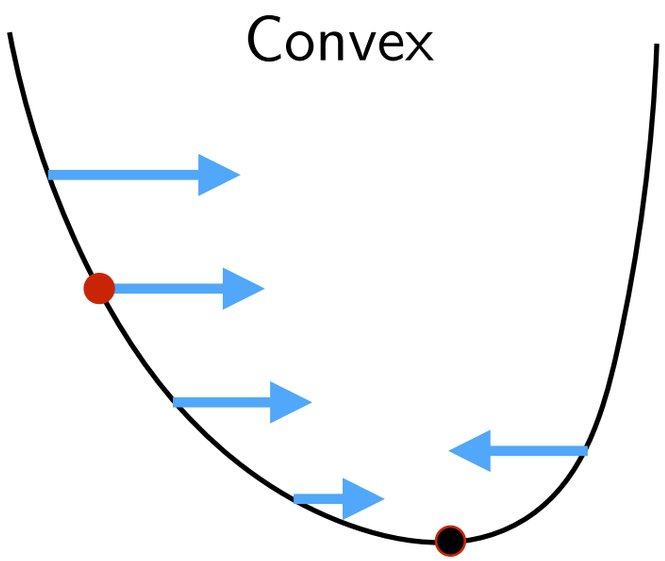


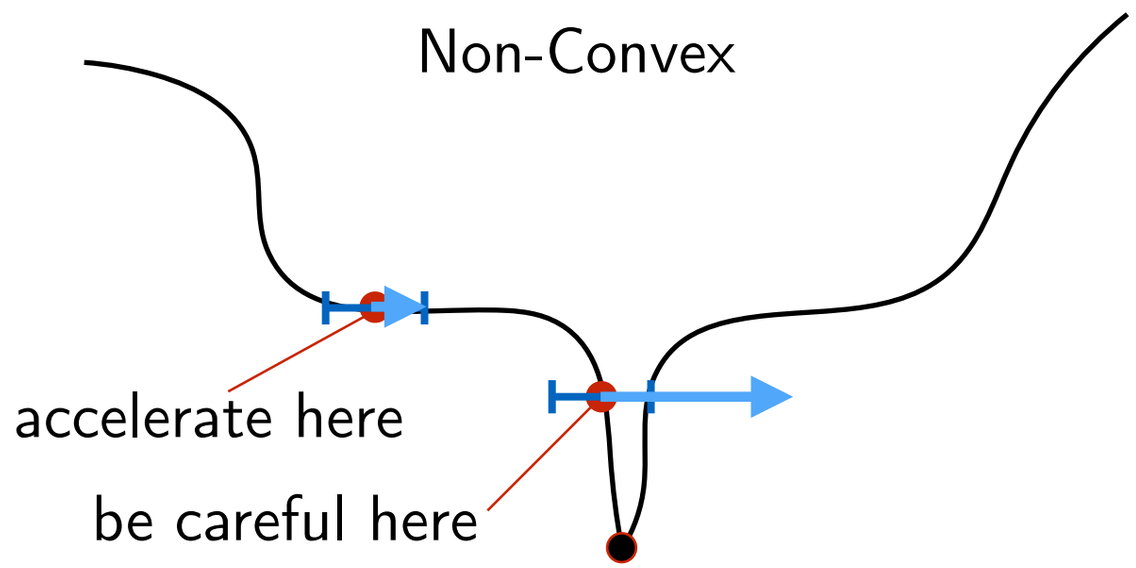


- Generates two kinds of algorithms:
  - using only step direction
  - using the normalized trust region step
- We can choose trust regions differently

Differences of Convex vs. Non-Convex

Step size proportional to the gradient?





- No other stationary points (saddle points, local minima) than the global minimum
- The further we are from the optimum, the larger is the gradient:
  - $\|\nabla f(x)\|^2 \geq \mu(f(x) - f^*)$
  - $\|\nabla f(x)\| \geq \mu|x - x^*|$
  - Makes sense so **step proportional to gradient**!
- Minus gradient points towards the optimum:
  - $\langle -\nabla f(x), x^* - x \rangle \geq f - f^* + \tilde{\mu}\|x - x^*\|^2$
  - Optimization need not be monotone in f

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

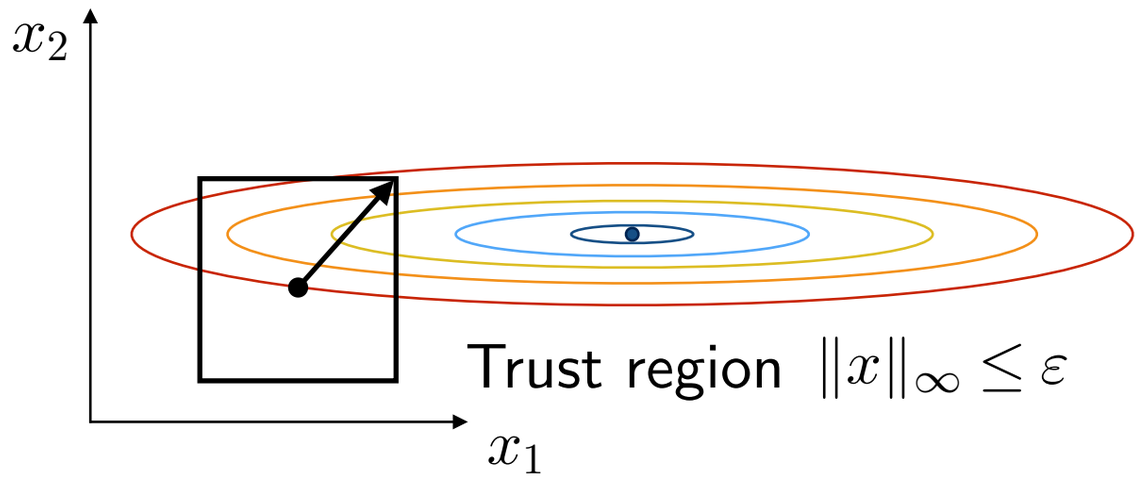
Box Trust Regions

- This time solve for step as:
  - $\min_{\|\Delta x_i\| \le \varepsilon \ \forall i} \left(f(x_0) + J\Delta x\right)$
  - In overparametrized models expect many parameters to have independent effect

    Equivalent to:

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

    $2\lambda_i \Delta x_i = -J_i$



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

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

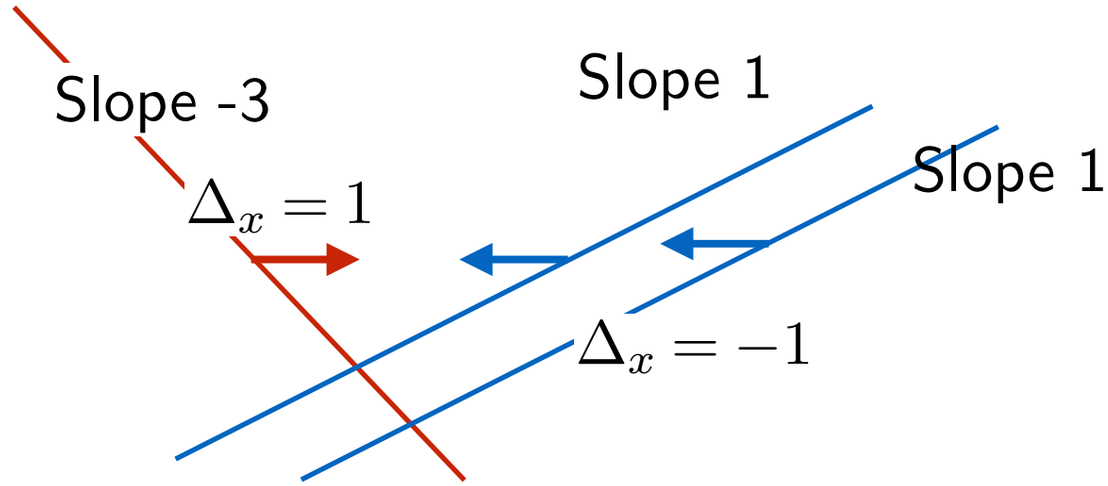
Non-Convex Stochastic

- Trust region steps: $\Delta x = -\frac{\nabla f(x)}{\|\nabla f(x)\|}$
- 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 change the stochastic gradient "too much nonlinearly"

Non-Convex Stochastic

- Solution: use running averages to approximate the expectation form:

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

  Also note that $\|\mathbb{E}[\nabla f]\| = \sqrt{(E[\nabla f])^2} \leq \sqrt{(E[(\nabla f)^2])}$ may be interpreted as a more robust setting

- **Adagrad**:

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

- **RMSProp**:

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

- **Adam**:

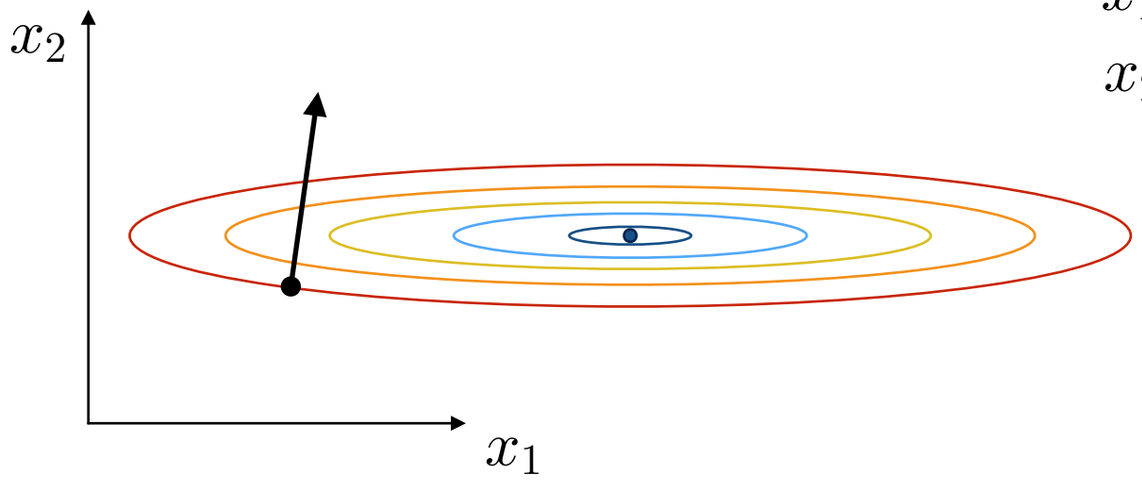
$$\theta_{t+1,i} = \theta_{t,i} - \varepsilon \frac{\mathsf{EWA}_{\beta_1}\left(\tilde{g}_{1:t,i}\right)}{\sqrt{\mathsf{EWA}_{\beta_2}\left(\tilde{g}^2_{1:t,i}\right)}}$$

- 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
  For sparse gradients, $t\,\mathsf{Mean}(\tilde{g}^2_{1:t,i})$ could grow much slower than t and achieve a speed-up compared to SGD
- 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 non-linear effect smooth enough

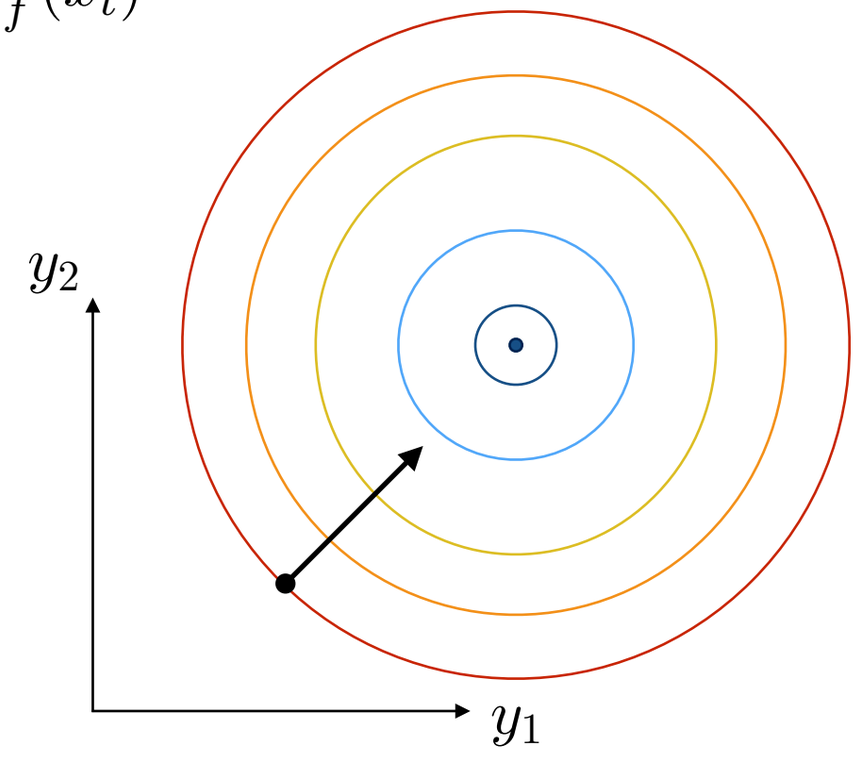
Change of Coordinates

Gradient Depends on the Choice of Coordinates

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



$$x_1 = y_1$$
$$x_2 = 5y_2$$



- Similar for non-linear change of coordinates, e.g. normalization ($\star$)

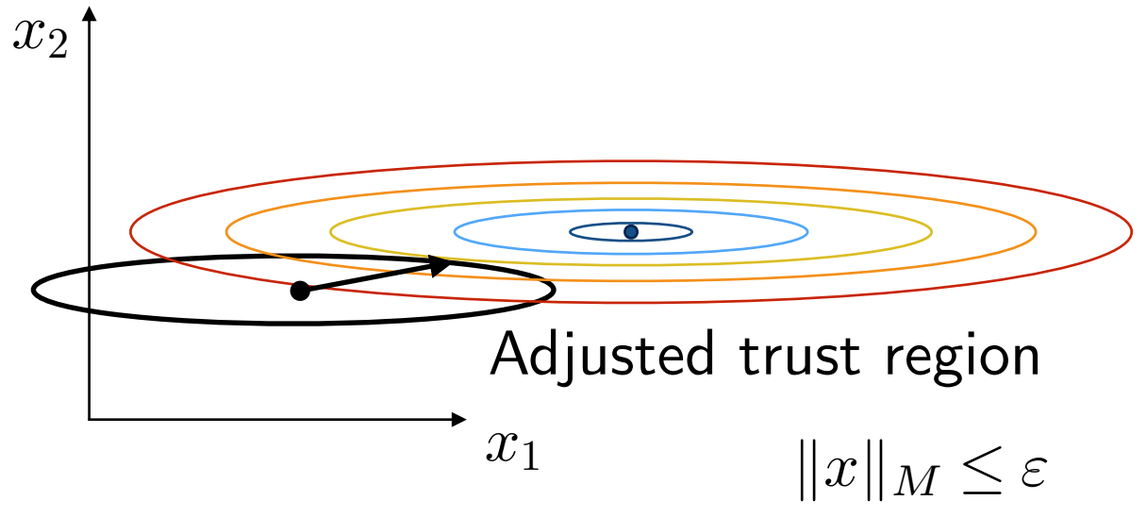
Proximal Problem View

- Adjust the trust region for sensitivity in different parameters:
  - $\min_{\|\Delta x\|_M \leq \varepsilon} \left(f(x_0) + J\Delta x\right)$ for given ε
  - $\|\Delta x\|_M = (\Delta x^\mathsf{T} M \Delta x)^{\frac{1}{2}}$ – Mahalanobis distance

  Equivalent to:

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

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

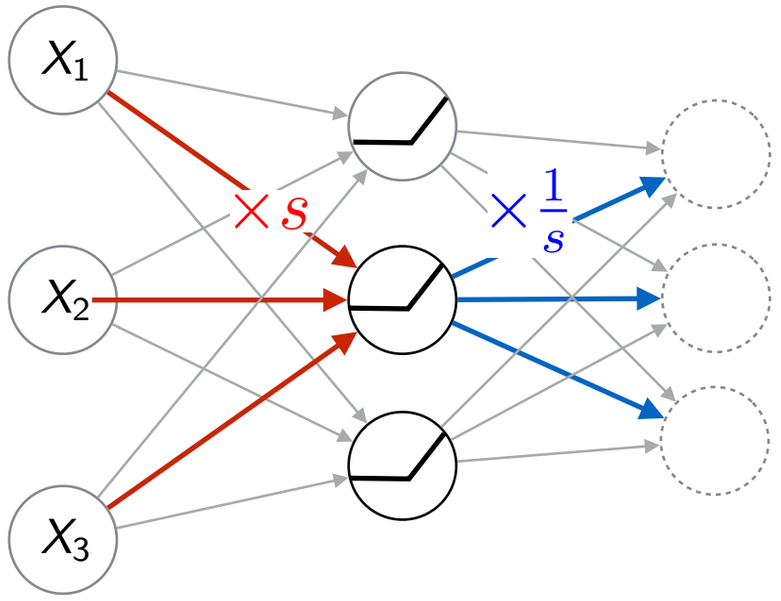


- Intuitive way to understand preconditioners
  - Can associate sensitivity with curvature → Second Order (Newton) Methods
  - Can associate sensitivity with some statistics of gradient oscillations,
    e.g. Adagrad: $M = \mathrm{Diag}\left(\sqrt{\mathrm{Mean}(g_{1:t}^2)}\right)$
  - Can use other metrics → Path SGD, Mirror Descent

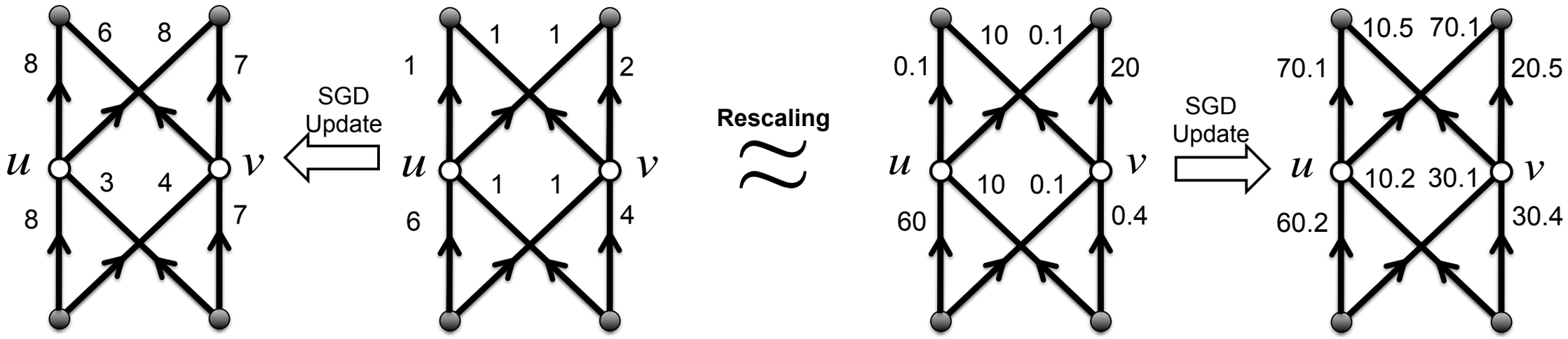
Equivalent Reparametrizations

- In ReLU networks 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)



- !Equivalent points but not equivalent SGD updates:



[Neyshabur et al. (2015) **Path-SGD**: Path-Normalized Optimization in Deep Neural Networks]

- Normally SGD works ok thanks to random initialization
- Path SGD uses metric not sensitive to such transformations

Mirror Descent

How to Handle Simple Constraints?

- **Example**: Need a parameter that models variance σ^2 of some distribution inside NN
  - Must be $\sigma^2 > 0$
  - But do not know the scale, e.g. $\sigma^2 \in [10^{-4}, 10^4]$

  Option 1: projected GD

  Parametrize as $\sigma^2 = y$

  Projecting to $y = 0$ results in invalid variance

  Cannot recover small σ^2 more accurately than the step size

  May never make enough steps to find big σ^2

  Option 2: Parametrize as $\sigma^2 = e^y$, $y \in \mathbb{R}$

  May overflow for large y

  Gradients grow unbounded

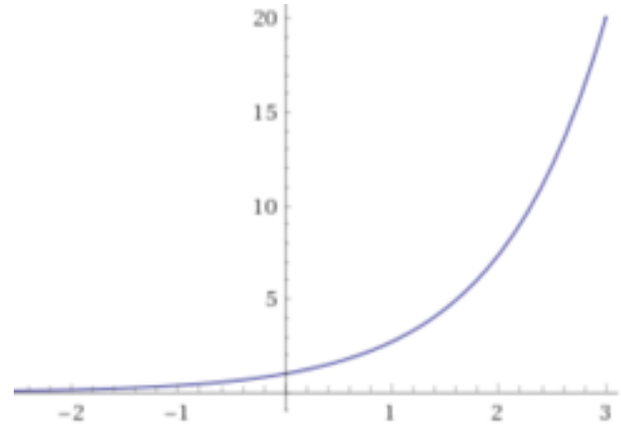
  If stepped to small values of y accidentally, gradients vanish

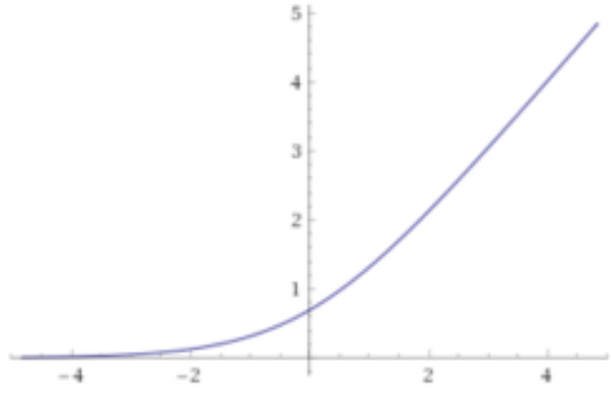
  Option 3: Parametrize as $\sigma^2 = \log(1 + e^y)$, $y \in \mathbb{R}$

  Gradients bounded

  May vanish if we step to $y \ll 0$

  May never get to high range values





(All options work to some extend, in particular Option 3 is often used in literature with variational Bayesian methods)

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 $x >= 0$

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

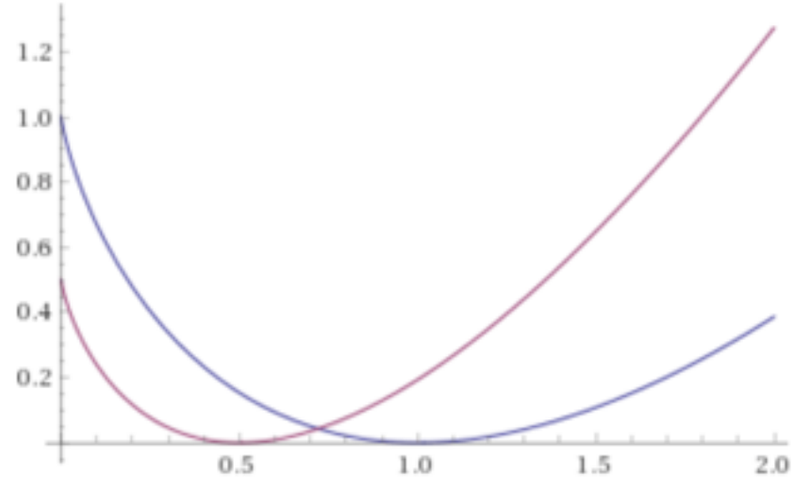
  Update: $\log x_{t+1} = \log x_t - \frac{1}{\lambda} \nabla_x f(x_t)$

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

  Can implement as:

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

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

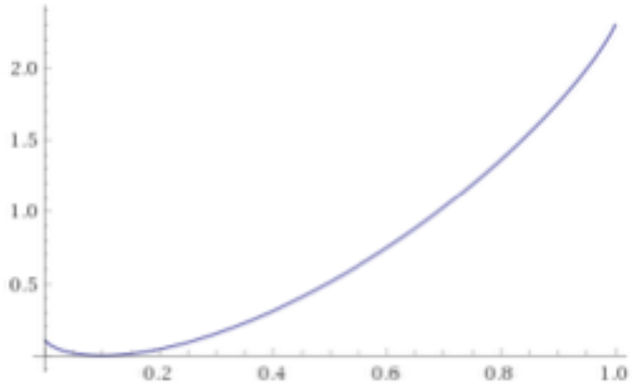


Mirror Descent

- 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 $x \in (0, 1)$

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

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

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

Mirror Descent

- Constraint $x_i \geq 0$, $\sum_i x_i = 1$ – simplex

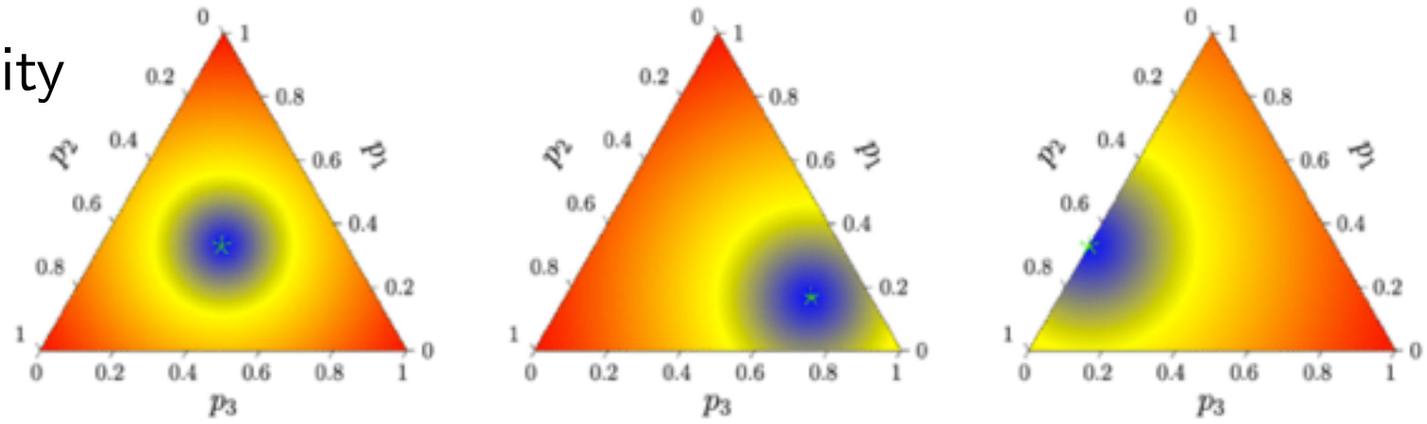
  $D(x, x^0) = \sum_i x_i \log \frac{x_i}{x_i^0}$ (KL divergence)

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

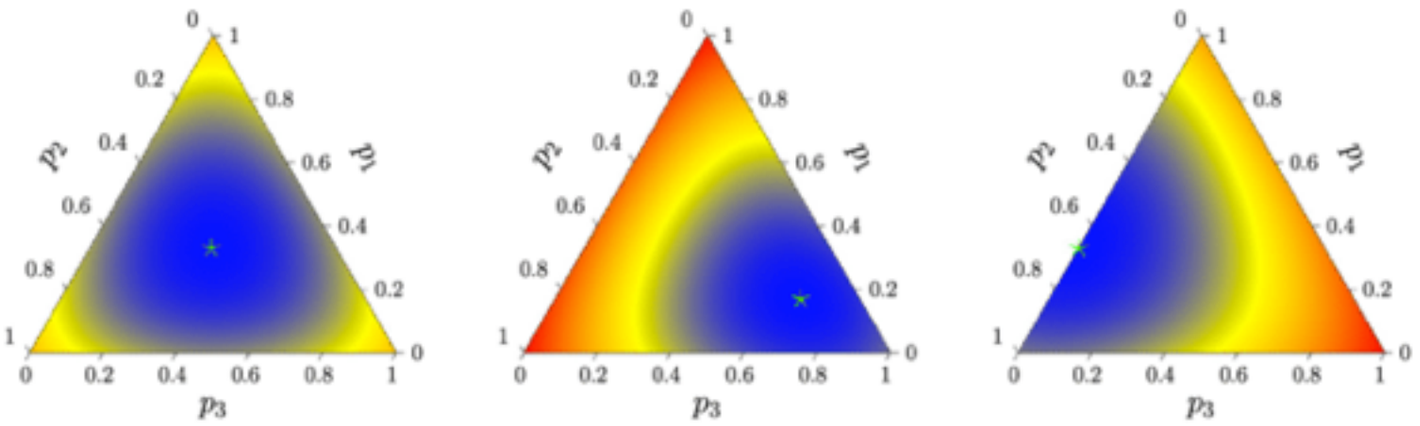
  $x_{t+1} = \mathrm{softmax}(y_t + 1)$

  - Can substitute and get update of x directly $\rightarrow$ **exponentiated GD** ($\star$)

Quadratic fidelity



KL fidelity



- Convergence in stochastic non-convex setting?
- At least we clearly see it averages gradients in the "mirror" space. Works in practice.