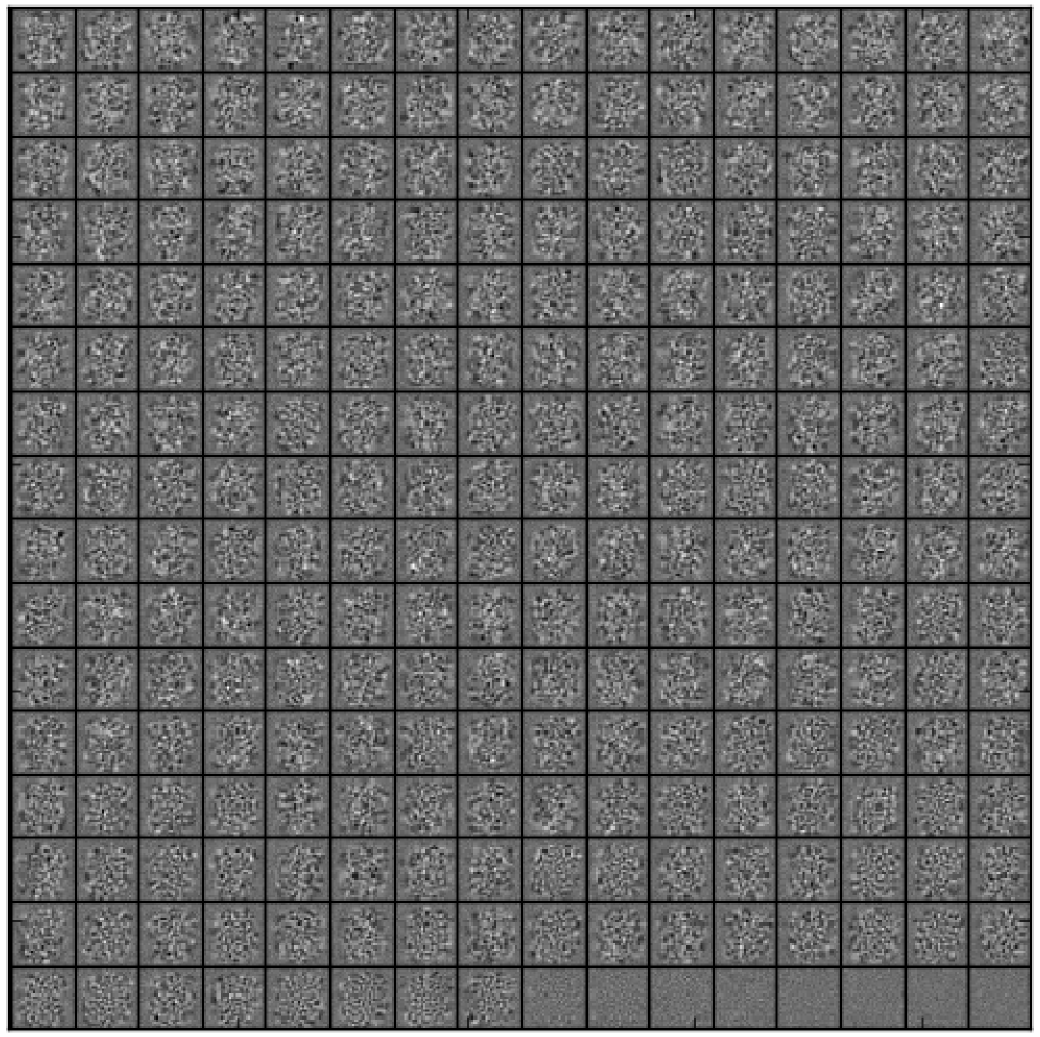
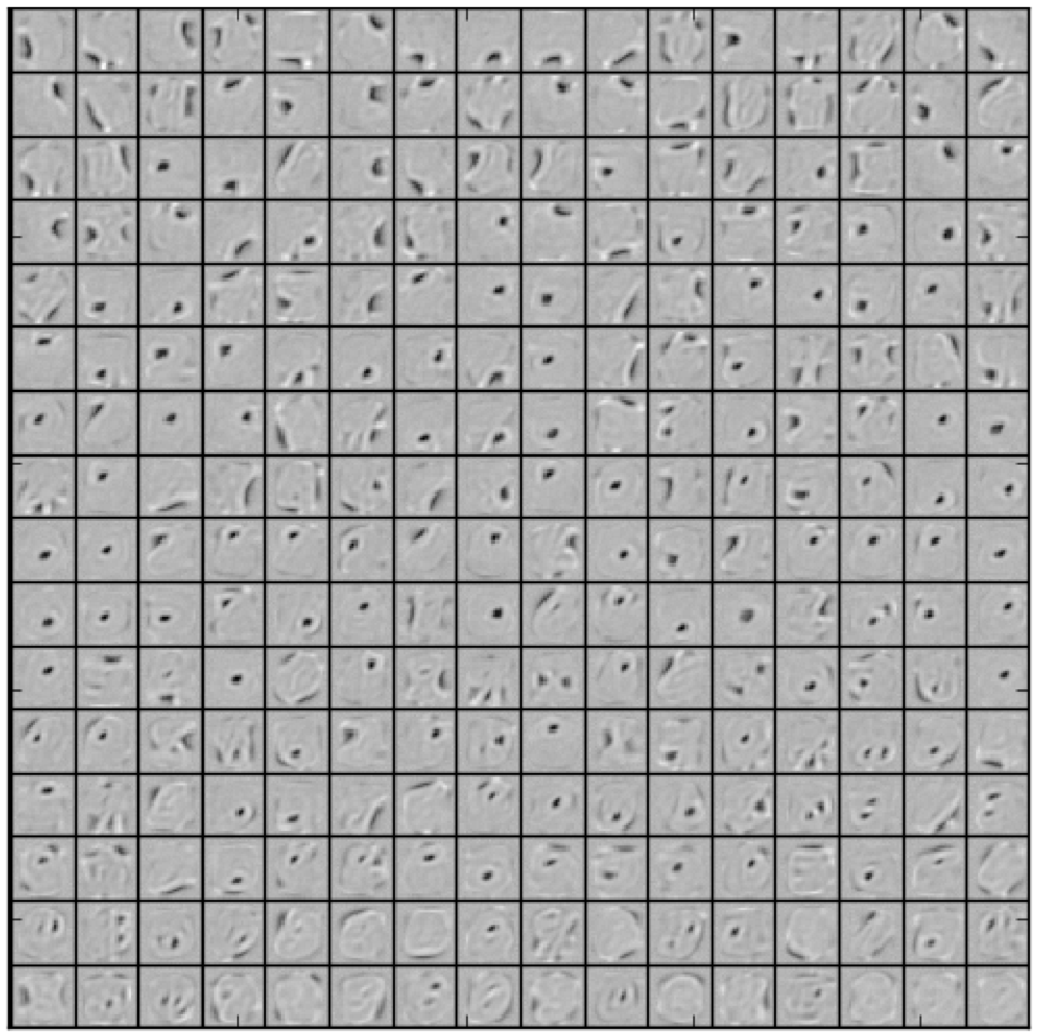
Effect on Features

- Experiment:
  - MNIST auto encoder with 1 fully-connected hidden layer of 256 units


(a) Without dropout


(b) Dropout with $p = 0.5$.

[Srivastava et al. (2014)]

- Hypothehis: dropout prevents co-adaptation (learns simpler and more robust features)

- Further interesting studies in the paper: effect on activation sparsity, connection to ridge regression, etc.

Deep Learning (BEV033DLE)
Lecture 8
Adaptive SGD Methods

Czech Technical University in Prague

Loss Landscape

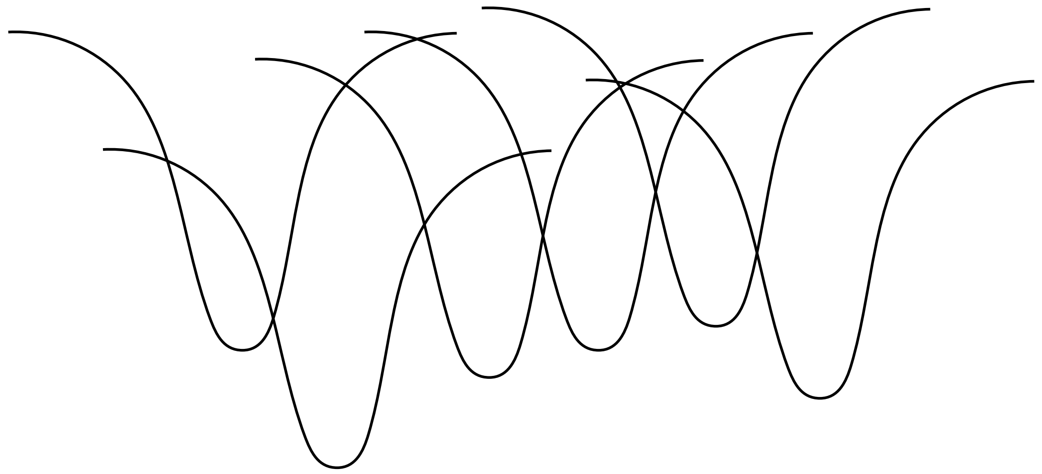
Local Minima

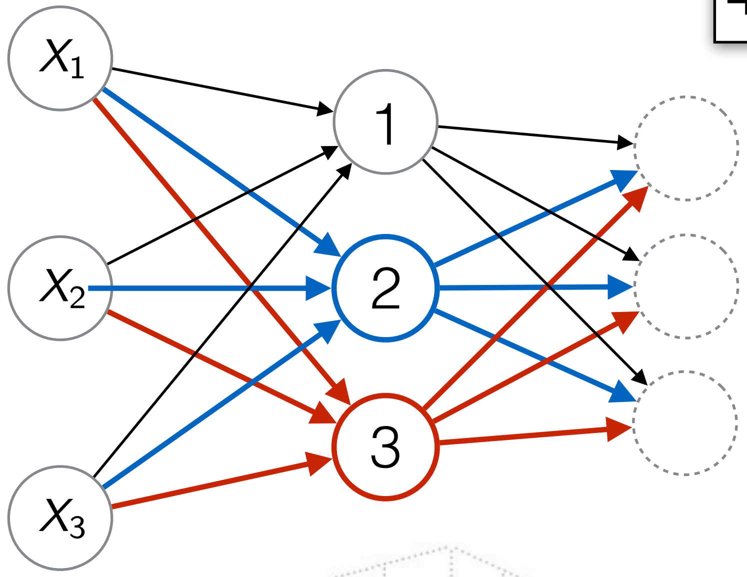
✦ There are several reasons for local minima

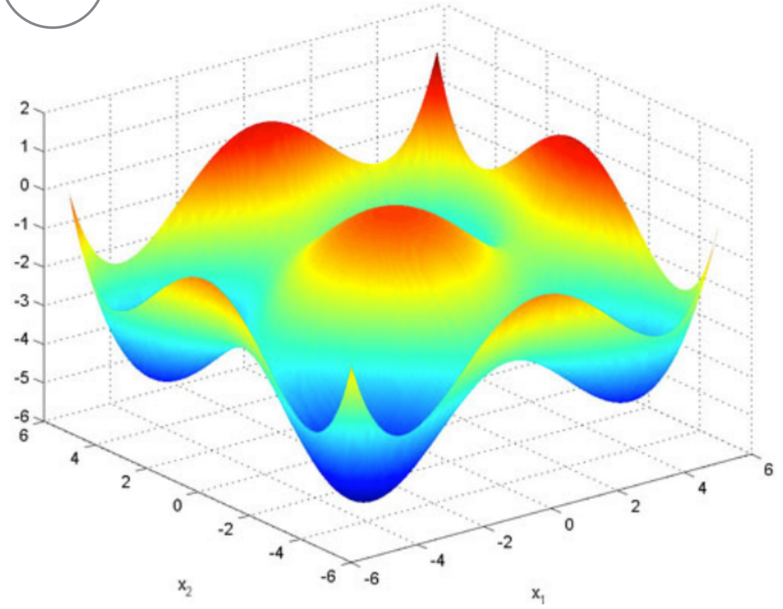
- **Symmetries** (Permutation invariances)
  - Fully connected layer with **n** hidden units: **n!** permutations
  - Convolutional layer with **c** channels: **c!** permutations
  - In a deep network many equivalent local minima, but all of them are equally good -- no need to avoid
- Loss function is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

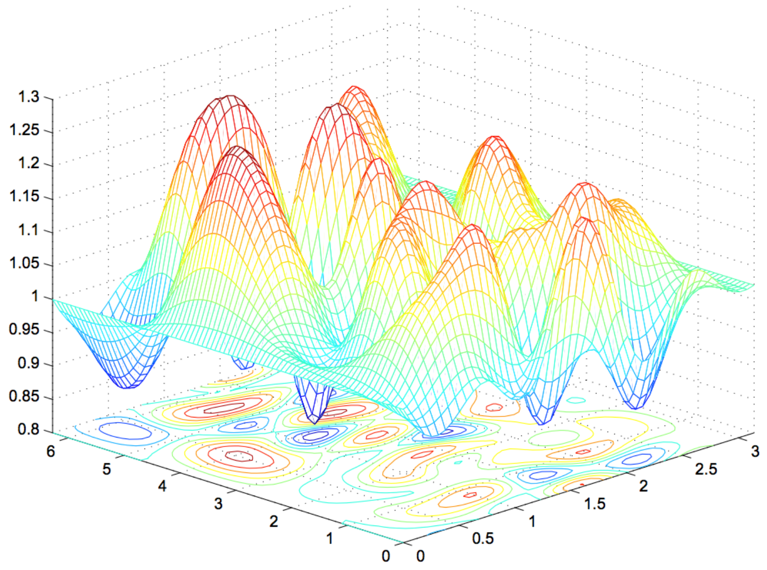
$$L(\theta) = \sum_i l(y_i, f(x_i;\theta))$$

often convex non-linear









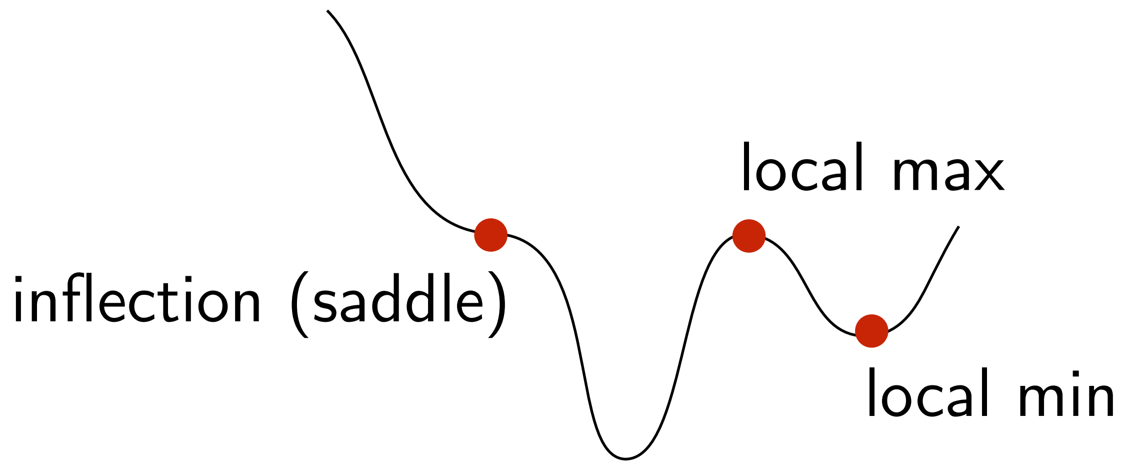
Stationary Points in High Dimensions

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

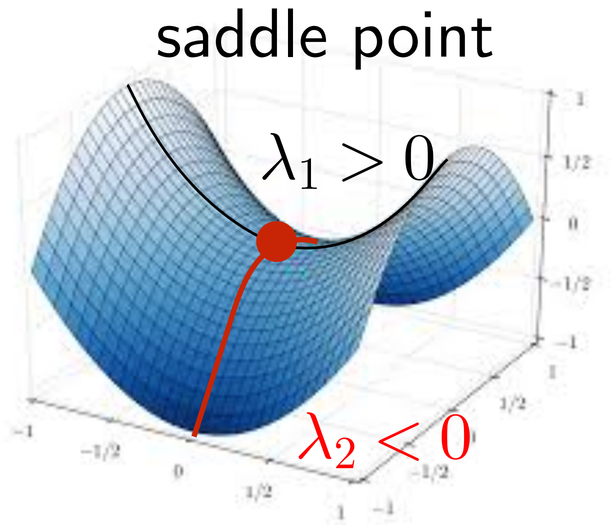
Stationary point: the gradient at x is zero

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

1D



2D



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

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

Index: α — the fraction of negative eigenvalues

$\alpha = 0 \Rightarrow$ local minimum

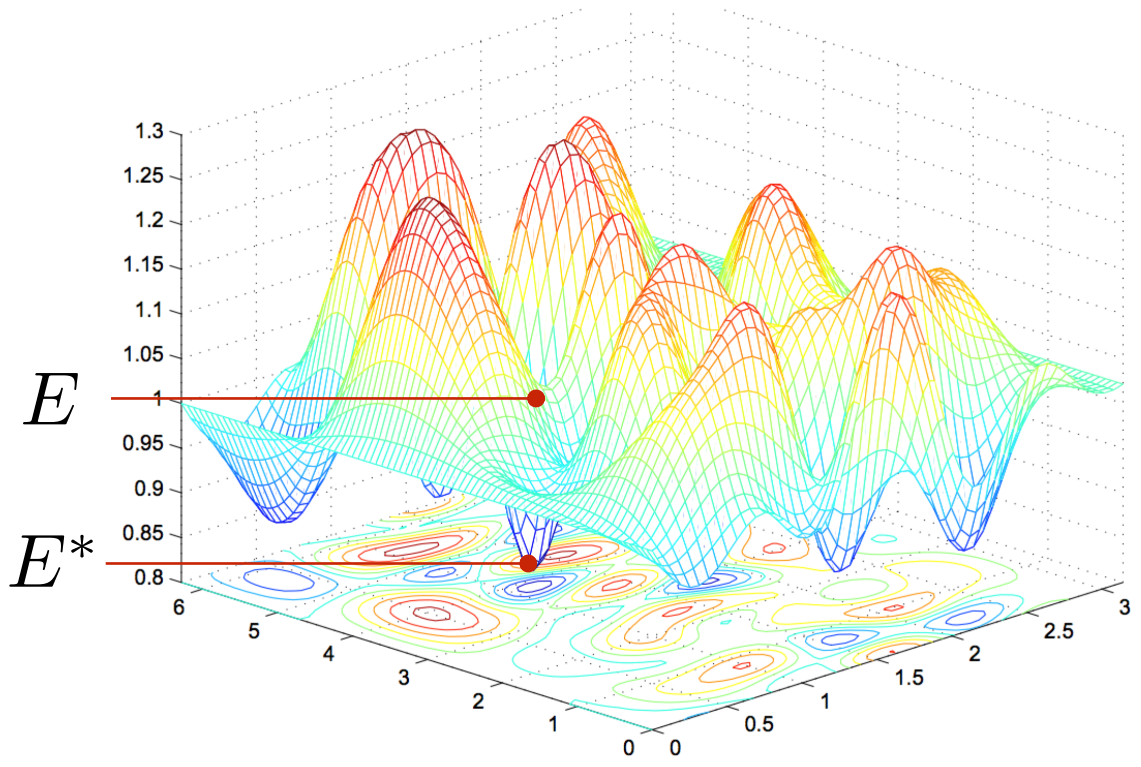
$\alpha = 1 \Rightarrow$ local maximum

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

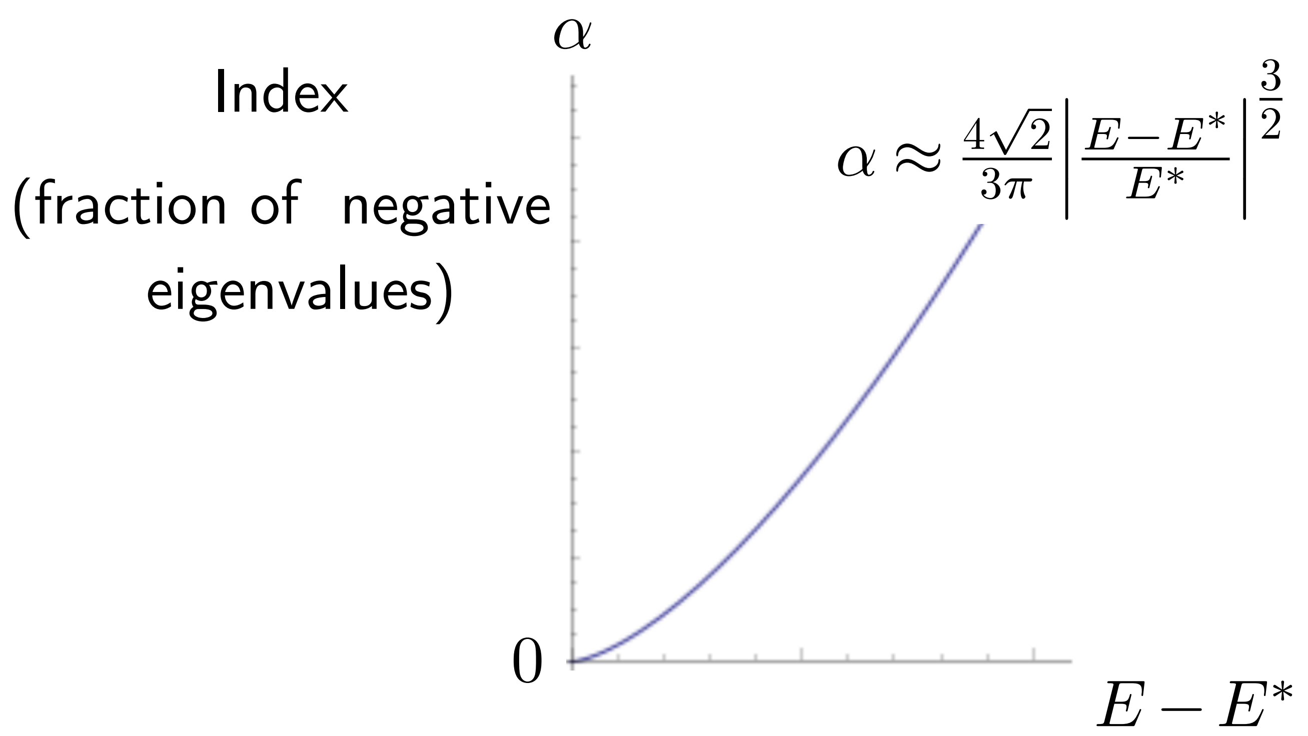
Stationary Points in High Dimensions

✦ Insights from Theoretical Physics --- **Gaussian Random Fields**:

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



Asymptotic relation for small alpha:

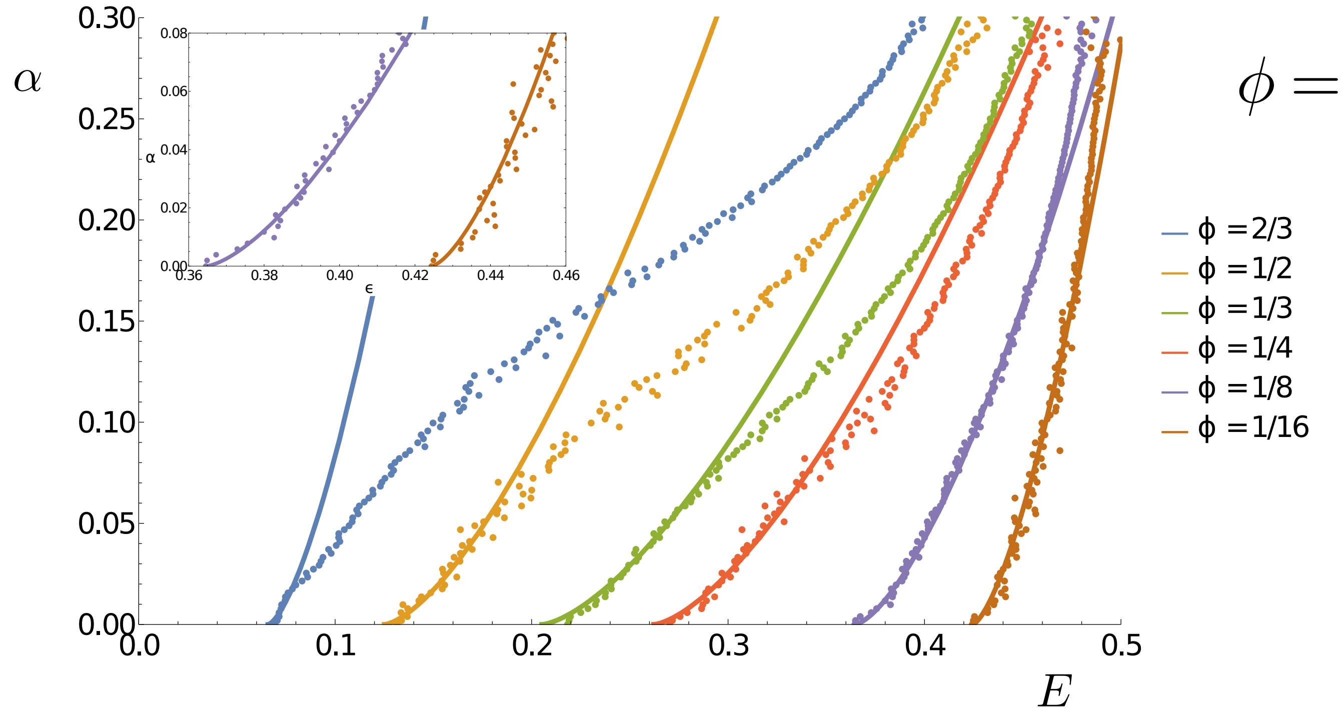


average energy of a st. point

[Bray & Dean (2007) The statistics of critical points of Gaussian fields on large-dimensional spaces]

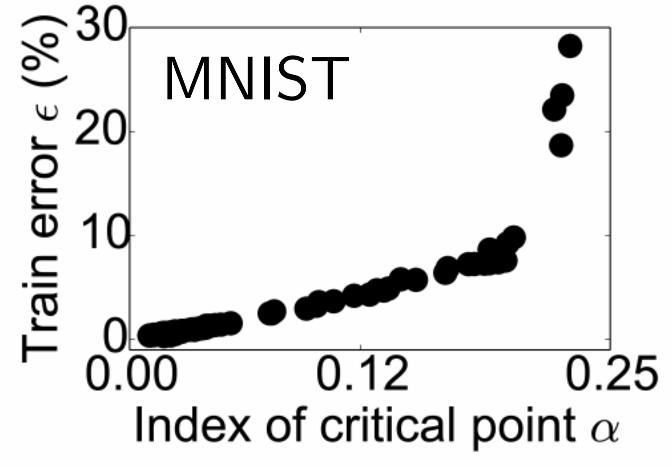
Stationary Points in High Dimensions

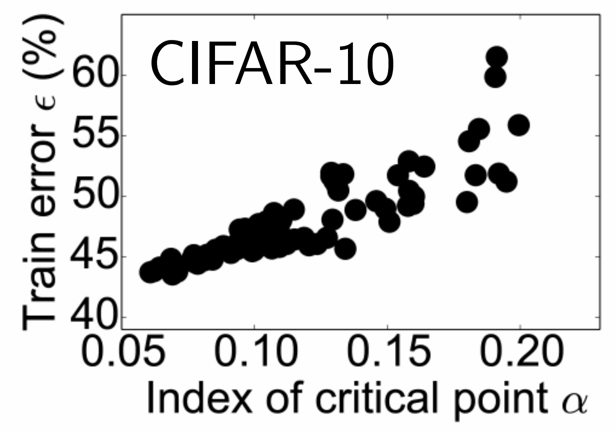
✦ Experimental Confirmations in Neural Networks



$$\phi = \frac{\#\text{parameters}}{\#\text{samples}}$$

[Pennington & Bahri 2017]





[Dauphin et. al. 2017]

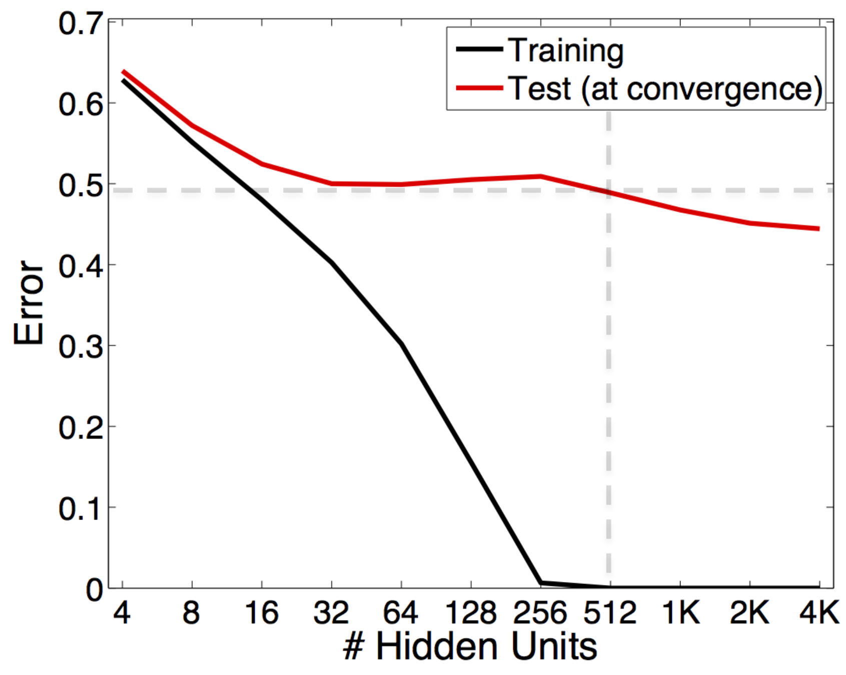
- 1 hidden layer
- good agreement for small alpha (as expected)

[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]

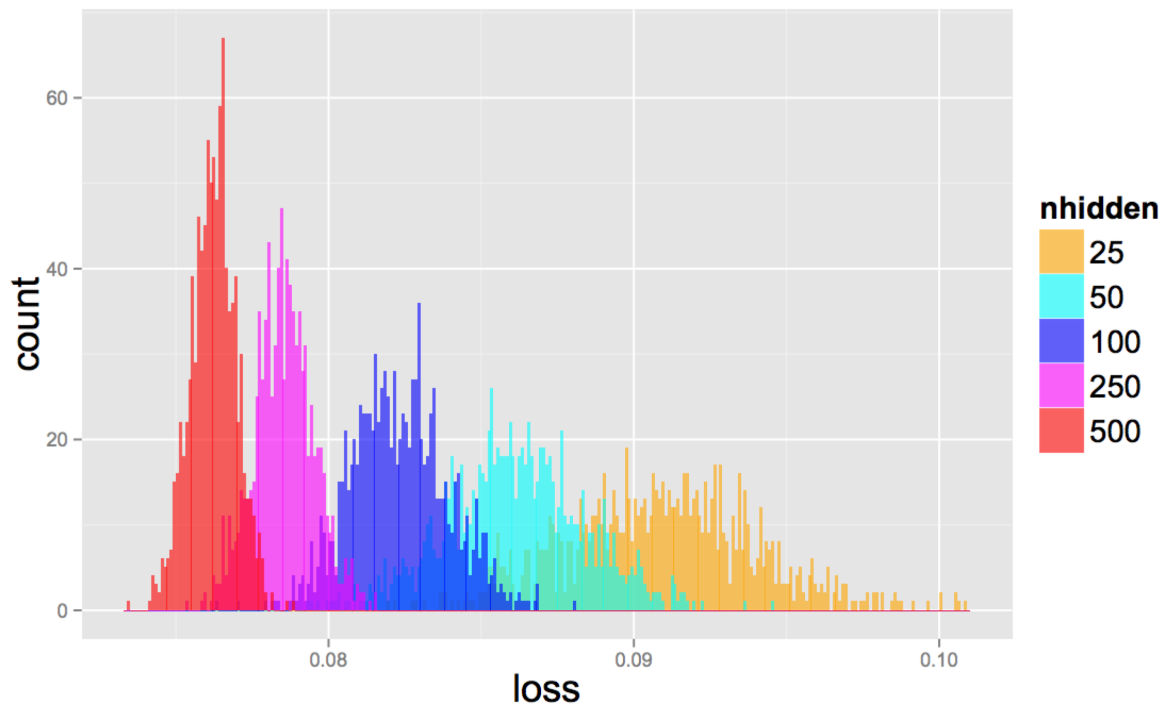
High Dimensionality Helps Optimization

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

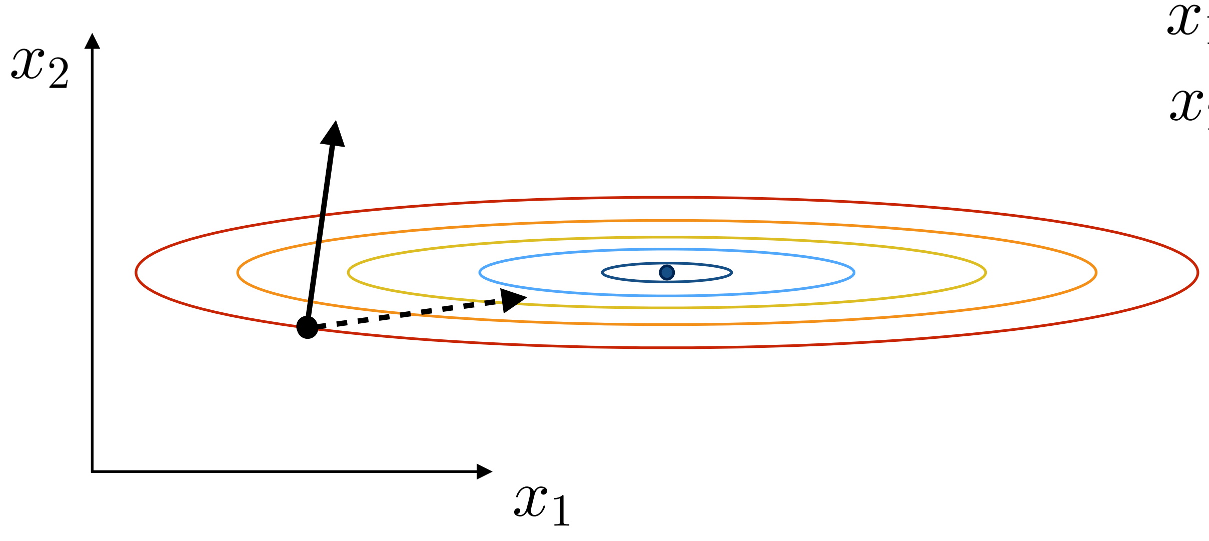
Problem: Gradient Descent Depends on Parameterization

Gradient Descent under Reparameterization

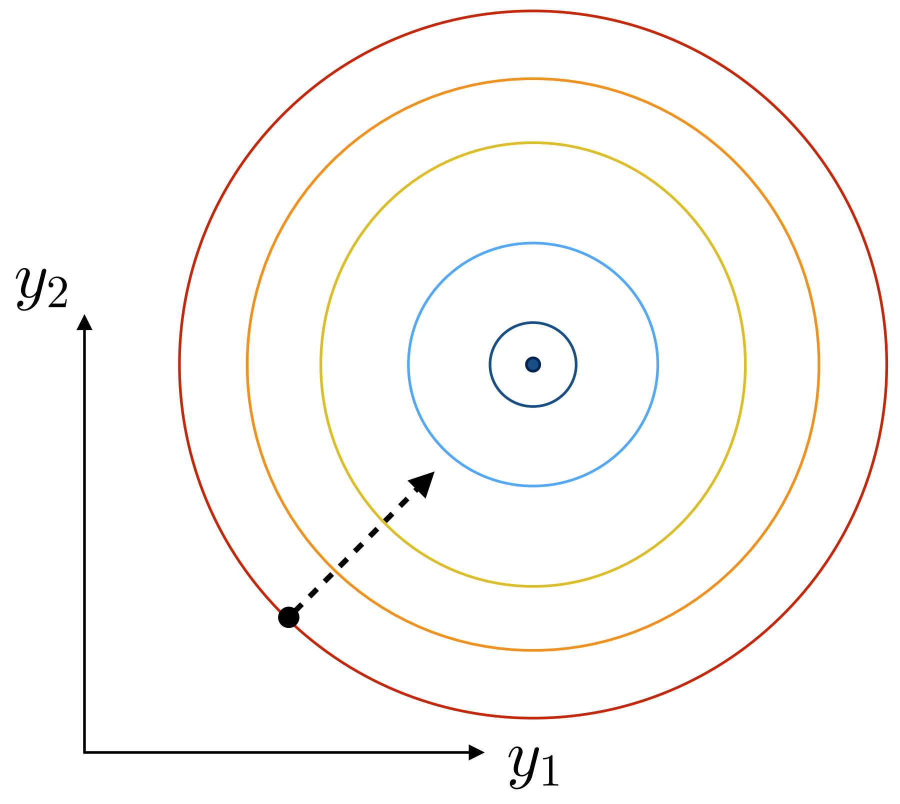
- Basic Example
  - Want to minimize $f(x)$
    By gradient descent: $x^{t+1} = x^t - \alpha f'(x^t)$, starting from x^0
  - Make a change of variables: $y = 2x$
    $y^0 = 2x^0$
    $g(y) = f(y/2)$
    $g'(y) = 1/2 f'(y/2) = 1/2 f'(x)$
  - Perform gradient descent on g:
    $y^{t+1} = y^t - \alpha g'(y)$
  - 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

Gradient Descent under Reparameterization

- Let $f\colon \mathbb{R}^n \to \mathbb{R}$ and its derivative $J(x) = \frac{\mathrm{d}f(x)}{dx}$.
  Gradient descent:
  - $x_{t+1} = x_t - \alpha J(x_t)$
- Make a substitution: $x = Ay$ (change of coordinate) and consider GD in y:
  - Problem in new coordinates: $\min\limits_{y\in\mathbb{R}^n} f(Ay)$
  - GD: $y_{t+1} = y_t - \alpha(J(Ay_t)A)^{\mathsf{T}}$
- Substitute back $y = A^{-1}x$:
  - $A^{-1}x_{t+1} = A^{-1}x_t - \alpha A^{\mathsf{T}} J^{\mathsf{T}}(x_t)$
  - Obtained: $x_{t+1} = x_t - \alpha(AA^{\mathsf{T}})J^{\mathsf{T}}(x_t)$



$x_1 = y_1$
$x_2 = 5y_2$



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

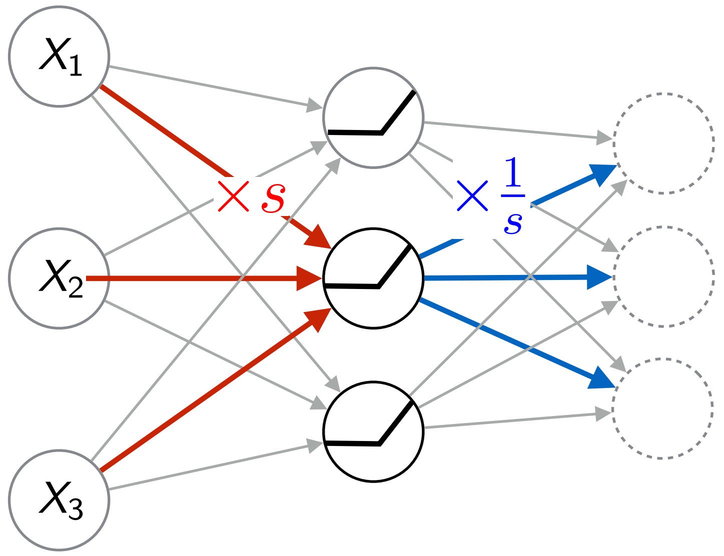
Gradient Descent under Reparameterization

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

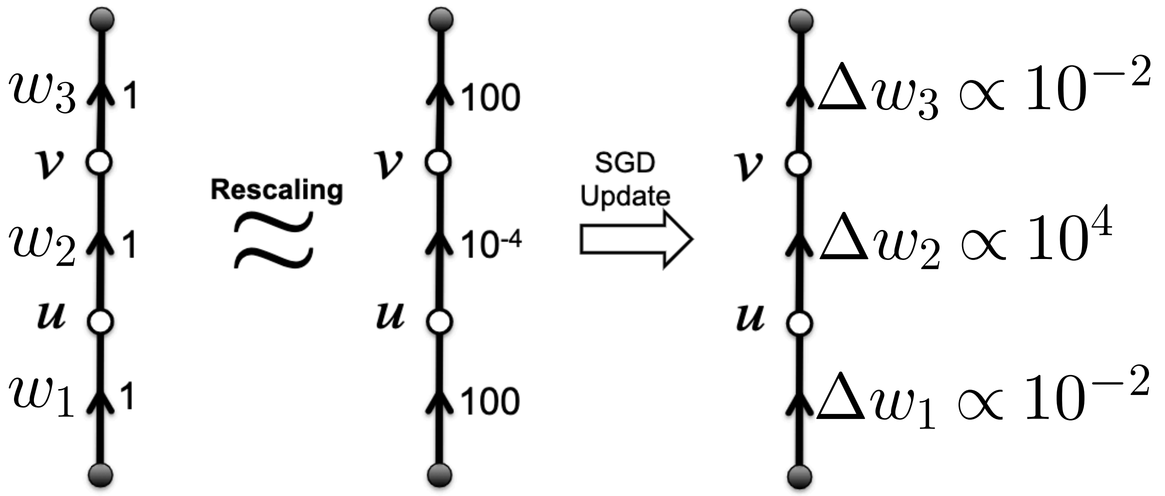
- ReLU units are *1-homogenous*:

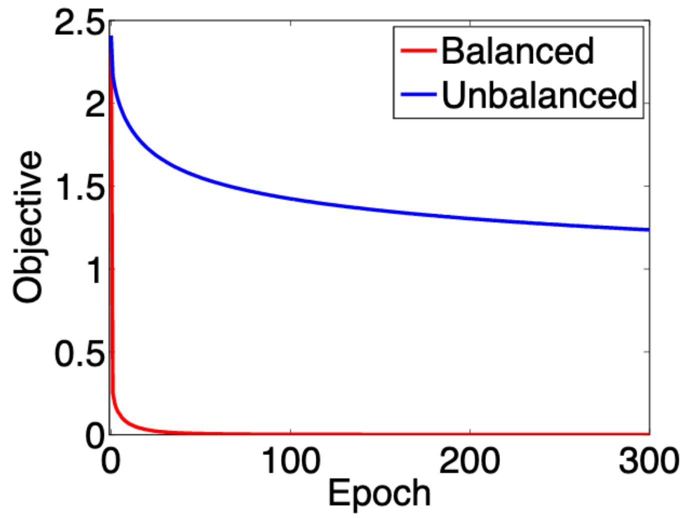
  for $s > 0$: $\text{ReLU}(sx) = \max(0, sx) = s\max(0, x)$
- Can rescale inputs and outputs of each unit (channels in conv networks)

$$f(Aw) = f(w), \text{ but } \frac{\partial f(Aw)}{\partial w} \neq \frac{\partial f(w)}{\partial w}$$



✦ Can lead to completely different SGD behavior


w_3 1
v
w_2 1
u
w_1 1
Rescaling
$\approx$
100
10^{-4}
100
SGD Update
$\Delta w_3 \propto 10^{-2}$
$\Delta w_2 \propto 10^{4}$
$\Delta w_1 \propto 10^{-2}$



(a) Training on MNIST

✦ Importance of weight initialization:

- controls forward statistics (prevent activations from saturating)
- controls effective local learning rate

✦ Another good example is BN: forward is invariant to weight scale, but backward is not

Approach 1: Steepest Descent in Invariance-Preserving Norm

Proximal Problem

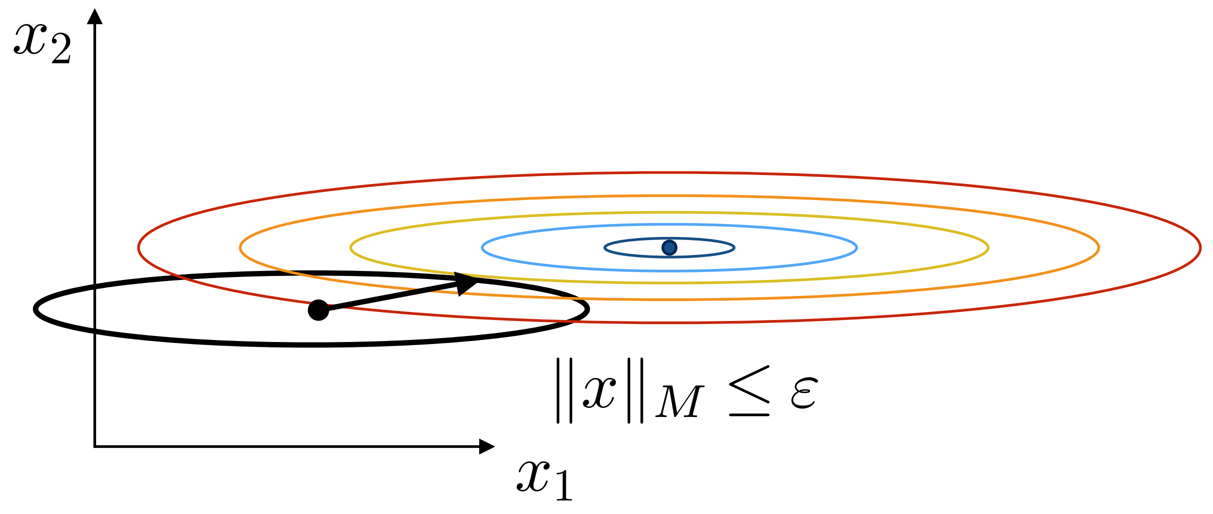
- Let's revisit how do we find the step Δx for SGD
  - Approximate: $f(x_0+\Delta x) \approx f(x_0) + J\Delta x$. This approximation is local.
- Find the step by solving **Proximal Problem**:

$$\boxed{\min_{\Delta x} \left(f(x_0) + J\Delta x + \tfrac{1}{2\alpha}\|\Delta x\|_2^2\right)}$$

$$0 = \frac{\partial}{\partial \Delta x} = J + \frac{1}{\alpha}\Delta x^{\mathsf{T}}$$

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

$$x_{t+1} = x_t - \alpha J(x_t)^{\mathsf{T}}$$ — common SGD



- p-norm SGD, $p>1$:

$$\min_{\Delta x} \left(f(x_0) + J\Delta x + \tfrac{1}{p\alpha}\|\Delta x\|_p^p\right)$$

$$\Delta x_i = -\alpha\, \mathrm{sign}(J_i)|J_i|^{\frac{1}{p-1}}$$

-- achieves different implicit regularization

- Machalanobis distance SGD:
  - $\min_{\Delta x} \left(f(x_0) + J\Delta x + \tfrac{1}{2\alpha}\|\Delta x\|_M^2\right)$
  - $\|\Delta x\|_M = (\Delta x^{\mathsf{T}} M \Delta x)^{\frac{1}{2}}$ – Mahalanobis distance

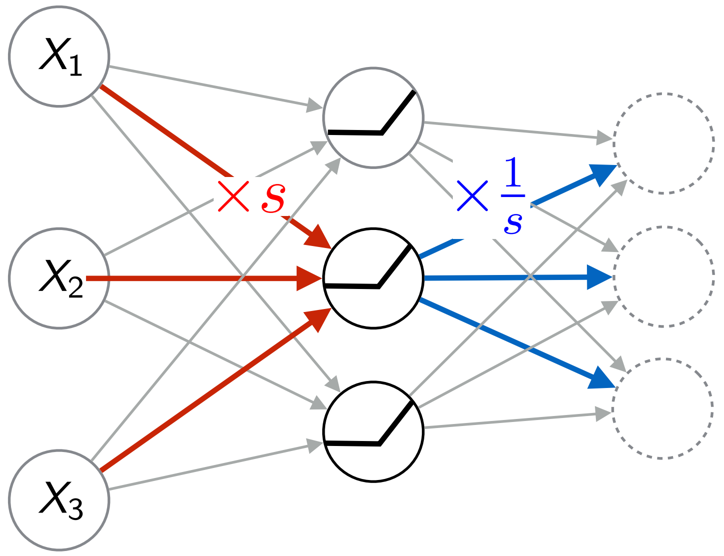
$$\Delta x = -\alpha M^{-1} J^{\mathsf{T}}$$

-- can compensate uneven curvature,

but how do we choose M?

Path-SGD

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



✦ Path-SGD considers metric invariant to equivalent transformations

Prox. problem:

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

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

- An efficient approximate solution is found

✦ Outcomes:

- Invariant (robust due to approximation) to all inner rescaling
- Specialized for ReLU networks
- Probably no substantial advantage in case the initialization is good

Approach 2: Normalize

Trust Region Problem

- Similar to proximal problem, but constrained optimization form:

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

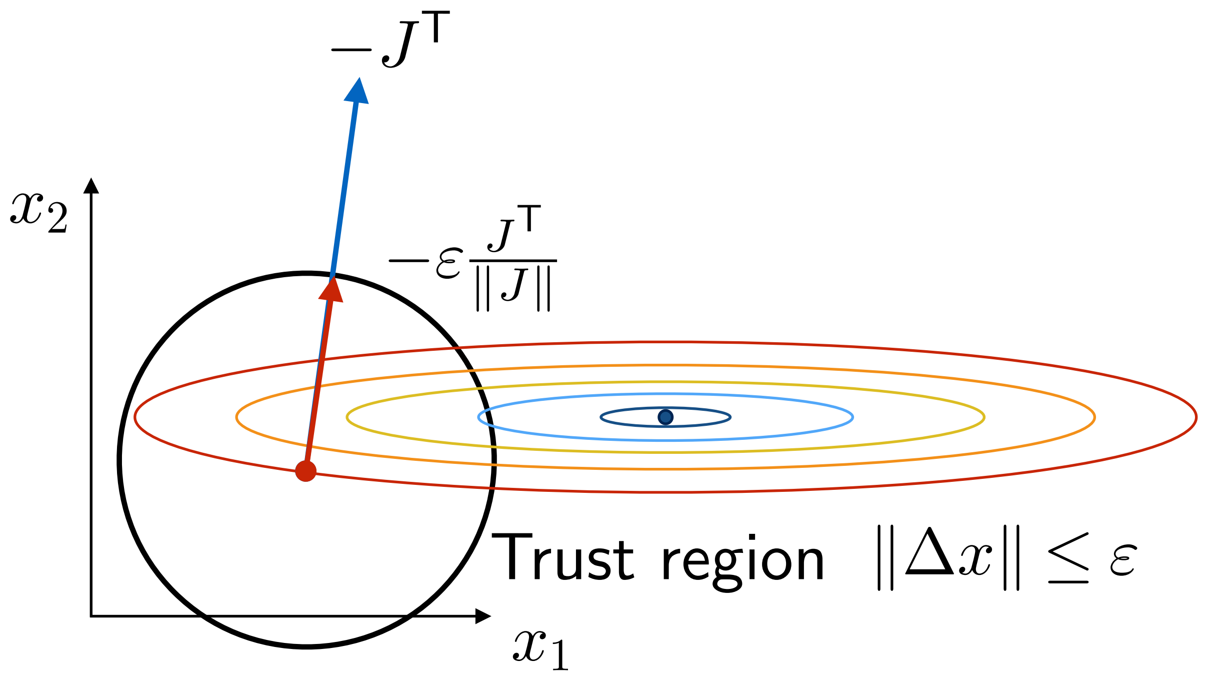
Equivalent to:

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

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

$$\|\Delta x^\mathsf{T}\|^2 = \varepsilon^2 \to \lambda = \frac{1}{2\varepsilon}\|J\|_2$$

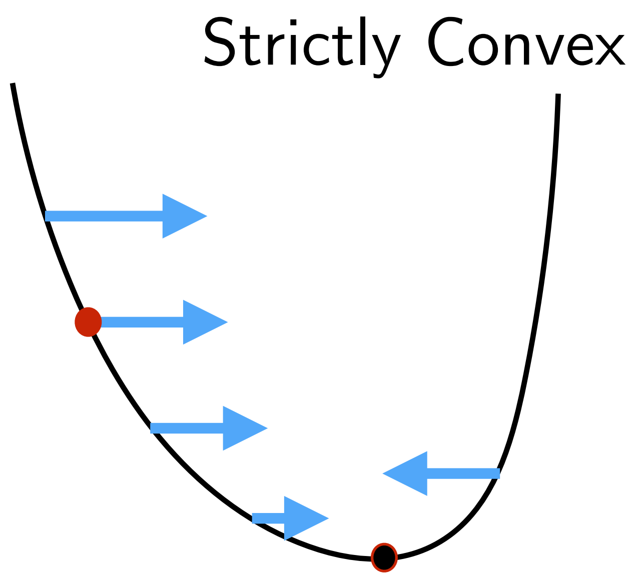
Trust region step: $\Delta x = -\varepsilon \frac{J^\mathsf{T}}{\|J\|_2}$


$-J^\mathsf{T}$
x_2
$-\varepsilon\frac{J^\mathsf{T}}{\|J\|}$
Trust region $\|\Delta x\| \leq \varepsilon$
x_1

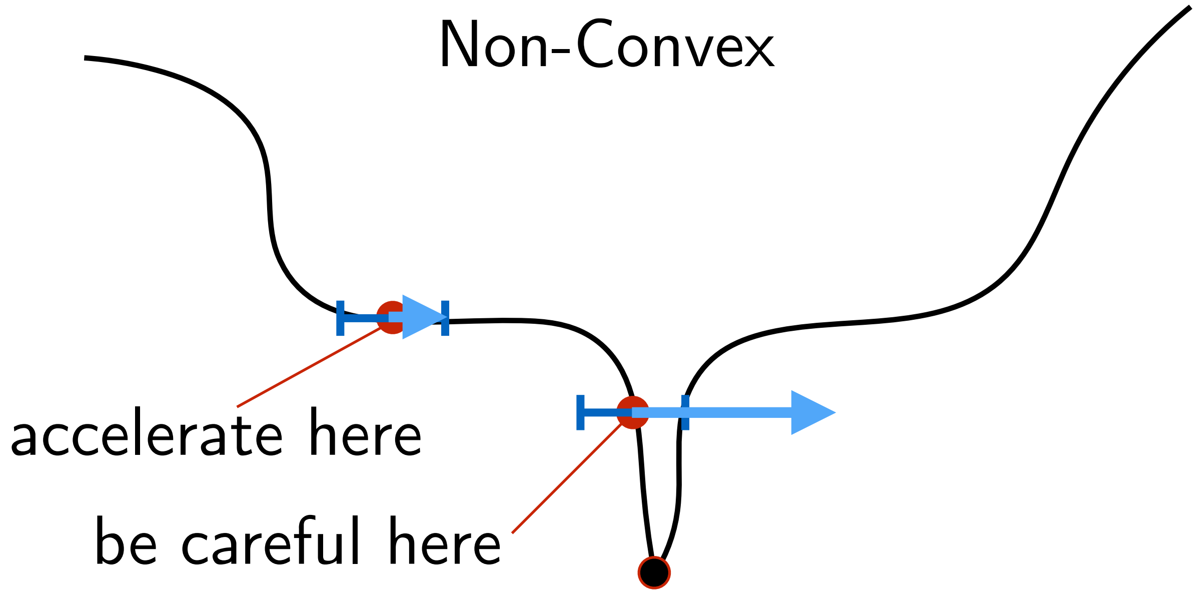
- We can choose the metric / trust region differently from Euclidean
- The step length is controlled explicitly and is invariant to gradient magnitude

Differences of Convex vs. Non-Convex

Why to step proportionally to the gradient:

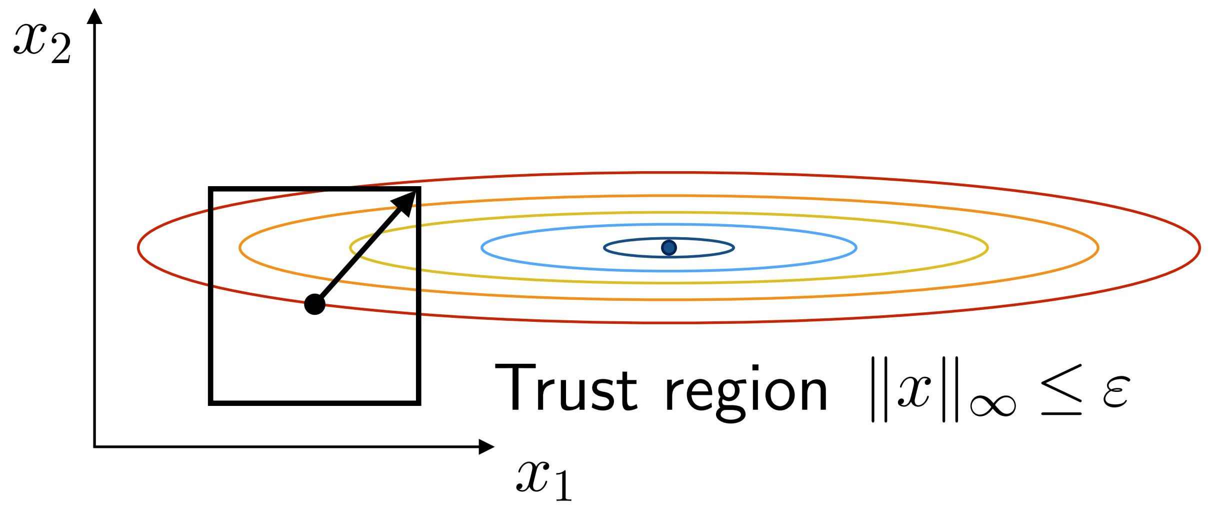


Why to normalize:



- 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

Box Trust Regions



- This time solve for step as:
  - $\min_{\|\Delta x_i\| \leq \varepsilon \ \forall i} \left(f(x_0) + J\Delta x \right)$

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

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

    $2\lambda_i \Delta x_i = -J_i$

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

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

Adaptive Methods

- **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**:

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

- **RMSProp**:

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

- **Adam**:

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

- 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
  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

Conclusions

✦ Two views:

- Proximal problem with a metric respecting some invariances --> path SGD, natural Gradient. Computation complexity vs approximation.
- Trust region problem: achieving invariance to local scaling via normalization.

✦ Practical adaptive methods:

- Proposed empirically, not optimal in some good sense. But achieve some desired properties like above, approximately.
- There is a space for alternative choices, like normalizing per layer / tensor of parameters seems like a good idea.