# Strojové učení II

## Deep Feedforward Networks

Institute of Information Theory and Automation of the AS CR

# Cost Function

- Training data: $(x, y) \sim \hat{p}(x, y)$
- NN represents a function $f(x; \theta)$
- NN outputs $f(x; \theta)$ are not direct predictions of y
- You define $q(y|x; \theta)$
- ML principle give us the cost function as $-\log q(y|x; \theta)$
- NN provides parameters (e.g. mean) of $q(y|x; \theta)$

# Conditional Log-likelihood

- Empirical distribution of training data $(x, y) \sim \hat{p}(x, y)$

$$(\mathbb{X}, \mathbb{Y}) = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$$

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} q(\mathbb{Y}|\mathbb{X}, \boldsymbol{\theta})$$

$$= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log q(y^{(i)}|x^{(i)}, \boldsymbol{\theta})$$

$$\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{(x,y)\sim\hat{p}(x,y)} \log q(y|x, \boldsymbol{\theta})$$

# Example – Gaussian Model

- Training data distribution: $p(x, y) = p(y|x)p(x)$
- Estimator of y: $f(x; \theta)$
- Model distribution: $q(y|x) \equiv N(y|f(x; \theta), \sigma^2)$

$$\hat{\theta} = \arg\min_{\theta} \mathbb{E}_{p(x,y)}[-\log q(y|f(x; \theta))]$$

$$L(\theta) = \int \left[ \int (y - f(x; \theta))^2 p(y|x) dy \right] p(x) dx + c$$

$$\frac{\partial L}{\partial \theta} \propto \mathbb{E}_{p(x)} \left[ \left( \mathbb{E}_{p(y|x)}[y] - f(x; \theta) \right) \times \frac{\partial f}{\partial \theta} \right] = 0$$

$$f(x; \hat{\theta}) = \mathbb{E}_{p(y|x)}[y]$$

# Linear Regression

- Training data: $p(x,y) \rightarrow \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$
- Linear estimator of y: $f(x) = wx + b \qquad \theta = \{w, b\}$
- Model distribution: $q(y|x) \equiv N(y|f(x;\theta), \sigma^2)$

$$\hat{\theta} = \arg\min_{\theta} \mathbb{E}_{p(x,y)}[-\log q(y|f(x;\theta))]$$

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{2\sigma^2} \sum_i (y^{(i)} - f(x^{(i)};\theta))^2 + \mathrm{c}$$

$$\arg\min_{w,b} \frac{1}{2\sigma^2} \sum_i (y^{(i)} - wx^{(i)} - b)^2 + \mathrm{c}$$

- Closed-form solution

# Example – Laplace Model

- Training data distribution: $p(x, y) = p(y|x)p(x)$
- Estimator of y: $f(x; \theta)$
- Model distribution: $q(y|x) \equiv \mathrm{Laplace}(y|f(x;\theta), \gamma)$

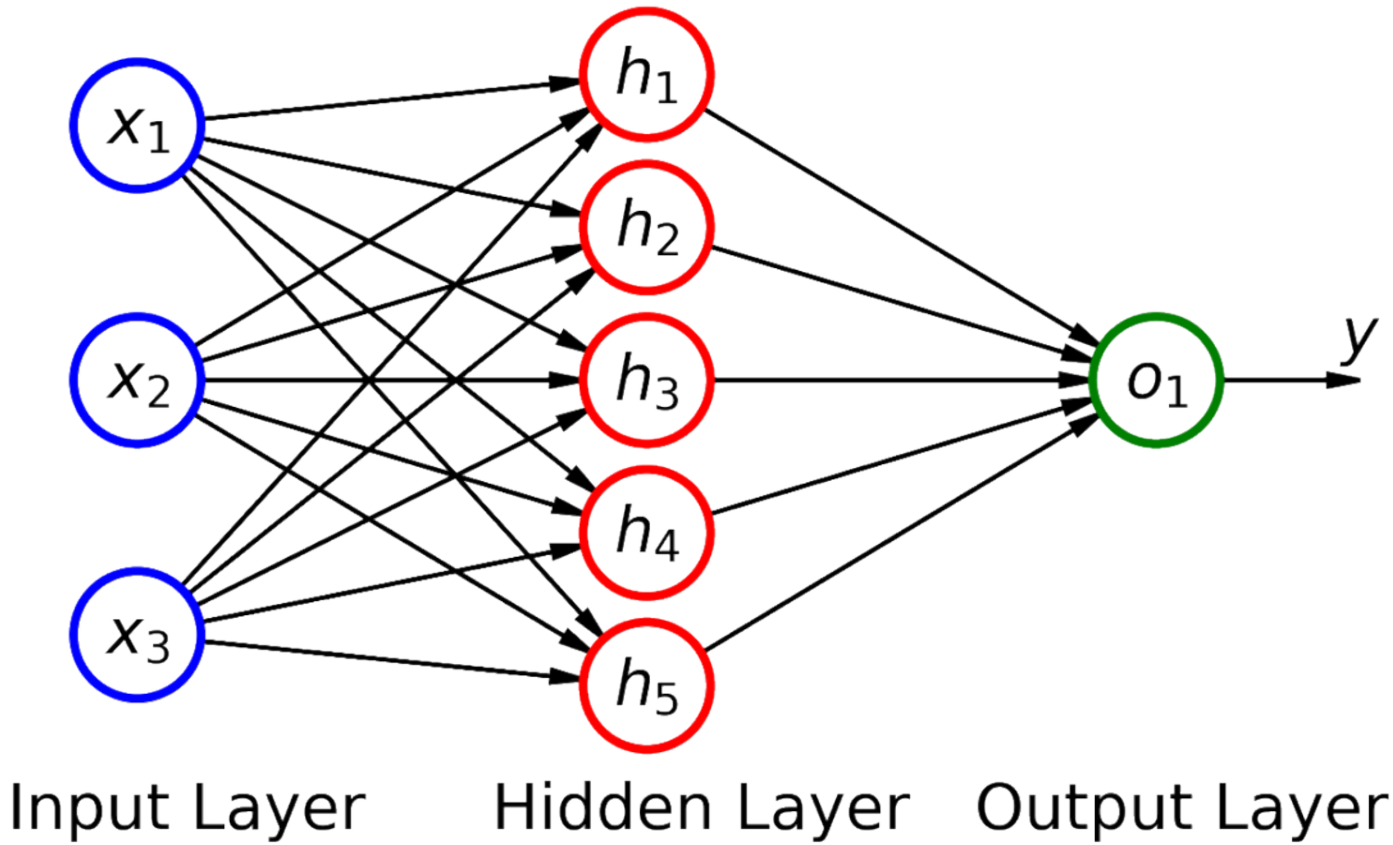
$$\hat{\theta} = \arg\min_{\theta} \mathbb{E}_{p(x,y)}[-\log q(y|f(x;\theta))]$$

$$L(\theta) = \int \left[ \int |y - f(x;\theta)|\, p(y|x) dy \right] p(x) dx + c$$

$$\frac{\partial L}{\partial \theta} \propto \mathbb{E}_{p(x)} \left[ \int \mathrm{sign}(y - f(x;\theta)) p(y|x) dy \times \frac{\partial f}{\partial \theta} \right] = 0$$

$f(x;\hat{\theta}) = m$ … median: $P(y \leq m|x) = P(y \geq m|x)$

# Feedforward Network



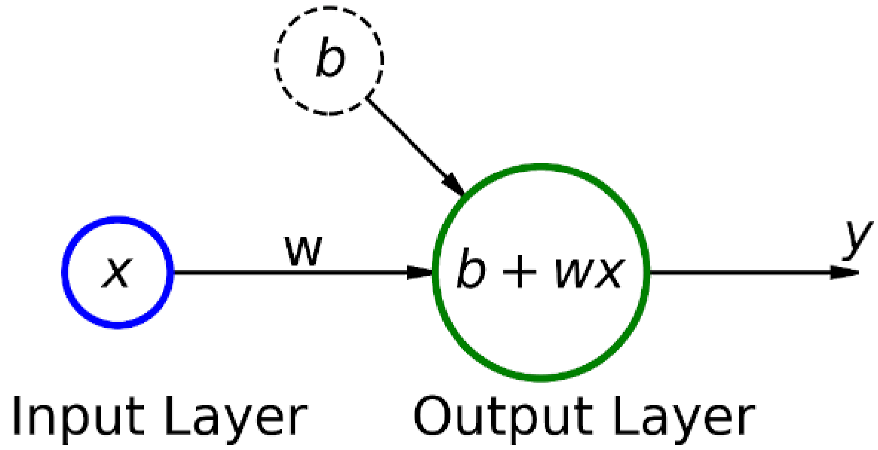
# Output Units

- Linear
- Sigmoid
- Softmax
- Softplus

# Linear Unit

- Affine transformation with no nonlinearity

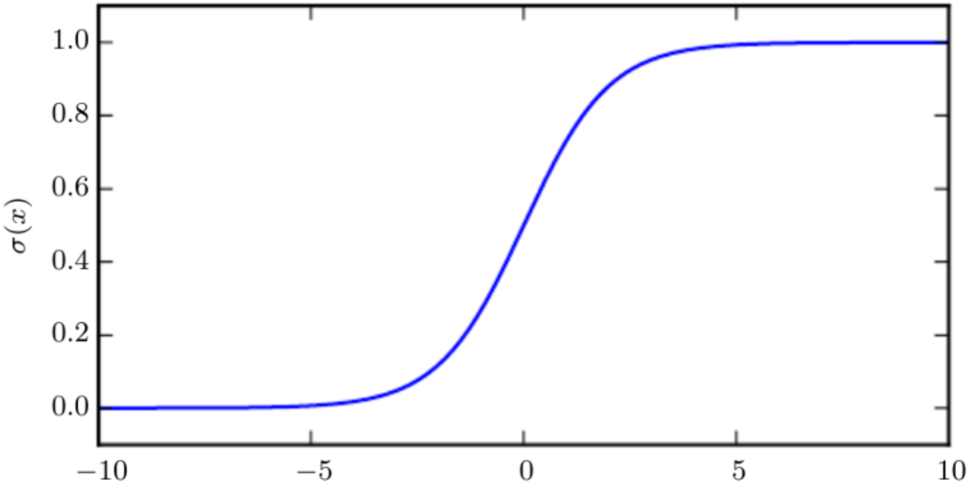
$$y = Wx + b$$




- e.g. predict mean of a Gaussian distribution

# Sigmoid

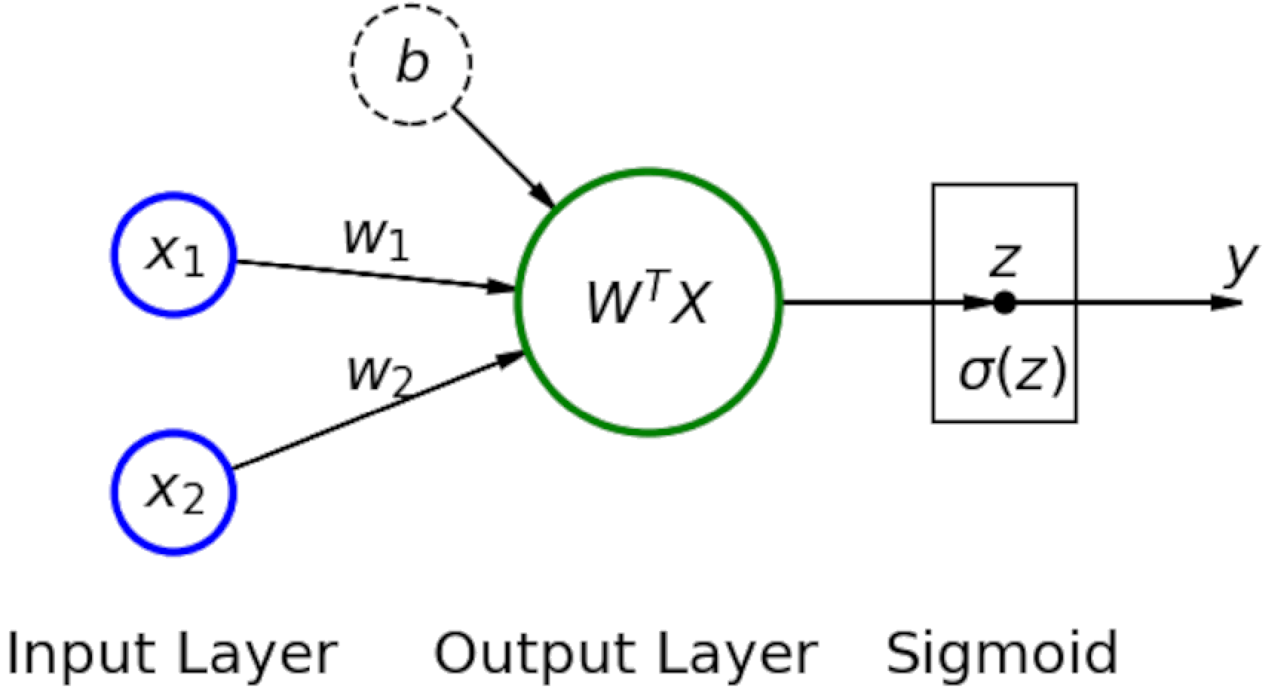
- Values in range (0,1)




$$\sigma(x) = \frac{1}{1+e^{-x}}$$

# Sigmoid output unit

- Predicting probability
- Ideal for Bernoulli output distributions



# Softmax

- Generalization of sigmoid
- Multi-class classification

$$z = Wx + b$$

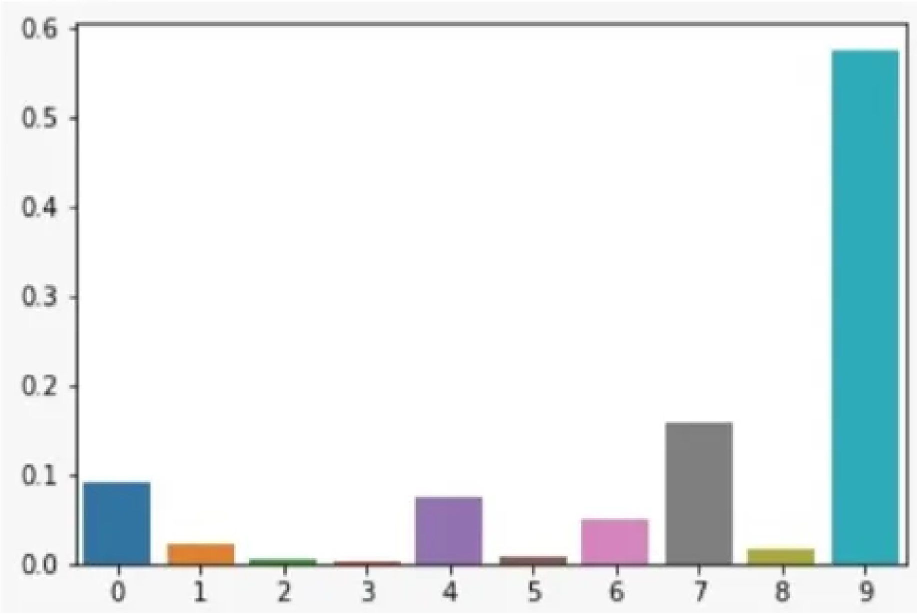
$$\mathrm{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

$$0 < \mathrm{softmax}(\mathbf{z})_i < 1 \qquad \sum_i \mathrm{softmat}(\mathbf{z})_i = 1$$

# Softmax

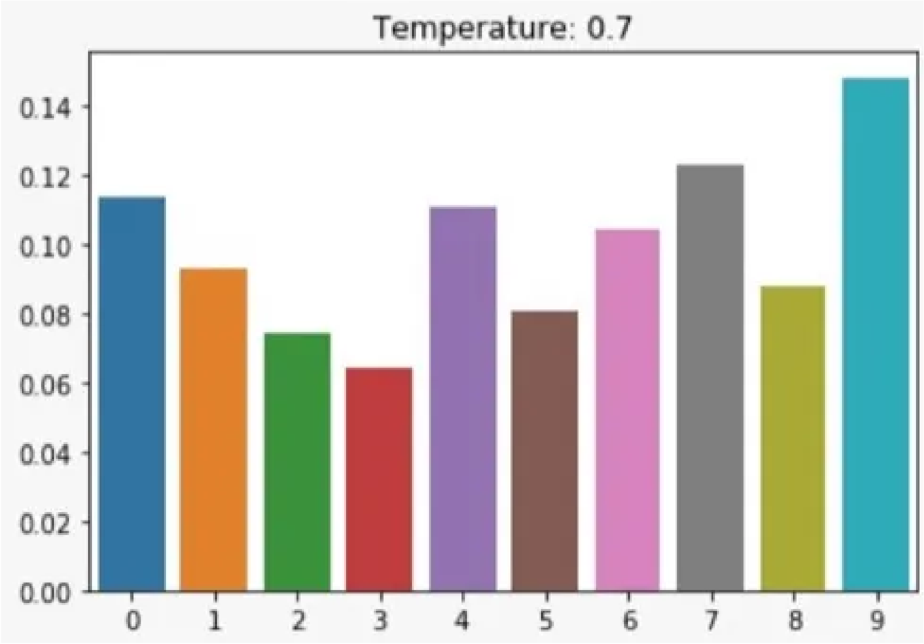
Softmax without temperature (T=1)

$$\frac{\exp(z_i)}{\sum_j \exp(z_j)}$$




Softmax with temperature

$$\frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$




Low entropy

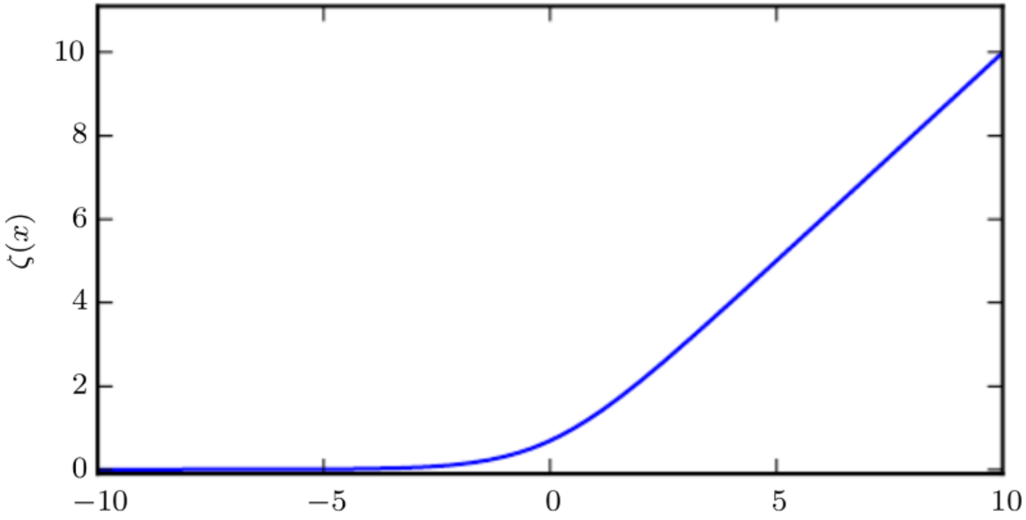
Increase in entropy

with increase in T

High entropy

# Softplus

- Non-negative values




$$\zeta(x) = \int_{-\infty}^{x} \sigma(y)dy$$

# Hidden Units

- Linear units with non-linear activation functions
- Activation functions:
  - Logistic Sigmoid
  - Hyperbolic Tangent

  old days

  - Rectification

  current

# Linear Layers

- Multiple linear layers <=> single linear layer

$$y_1 = W_1 x + b_1$$

$$y_2 = W_2 y_1 + b_2$$

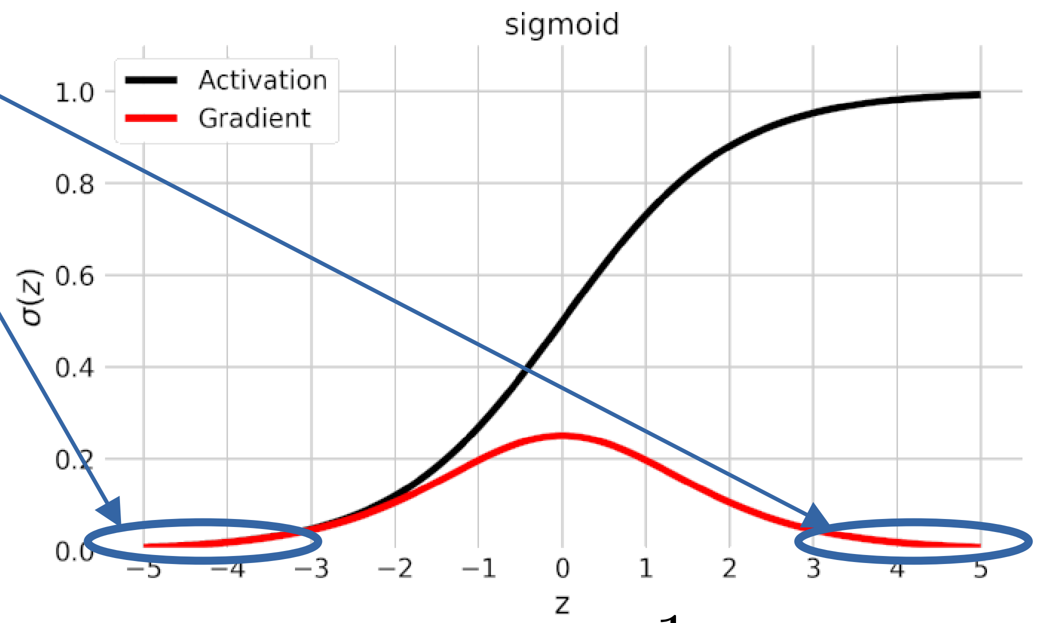
$$y_N = W_N y_{N-1} + b_N = W_N(W_{N-1}(\ldots) + b_{N-1}) + b_N$$

$$y_N = Wx + b$$

- We need a non-linear step in hidden units to go beyond!

# Sigmoid

Region of saturation



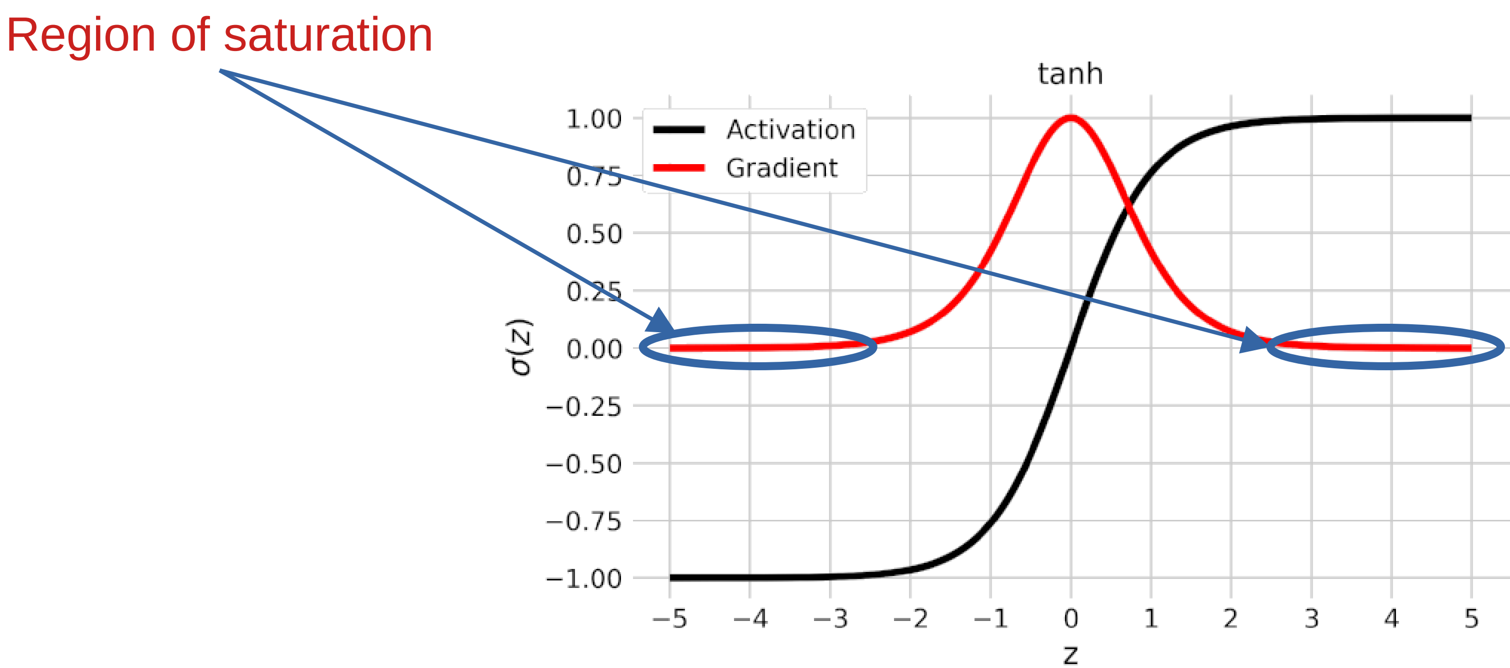

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

- Cons: $\sigma(0) \neq 0$ shifted

  $\sigma'(0) = 0.25 < 1$

  vanishing gradient for small and large inputs

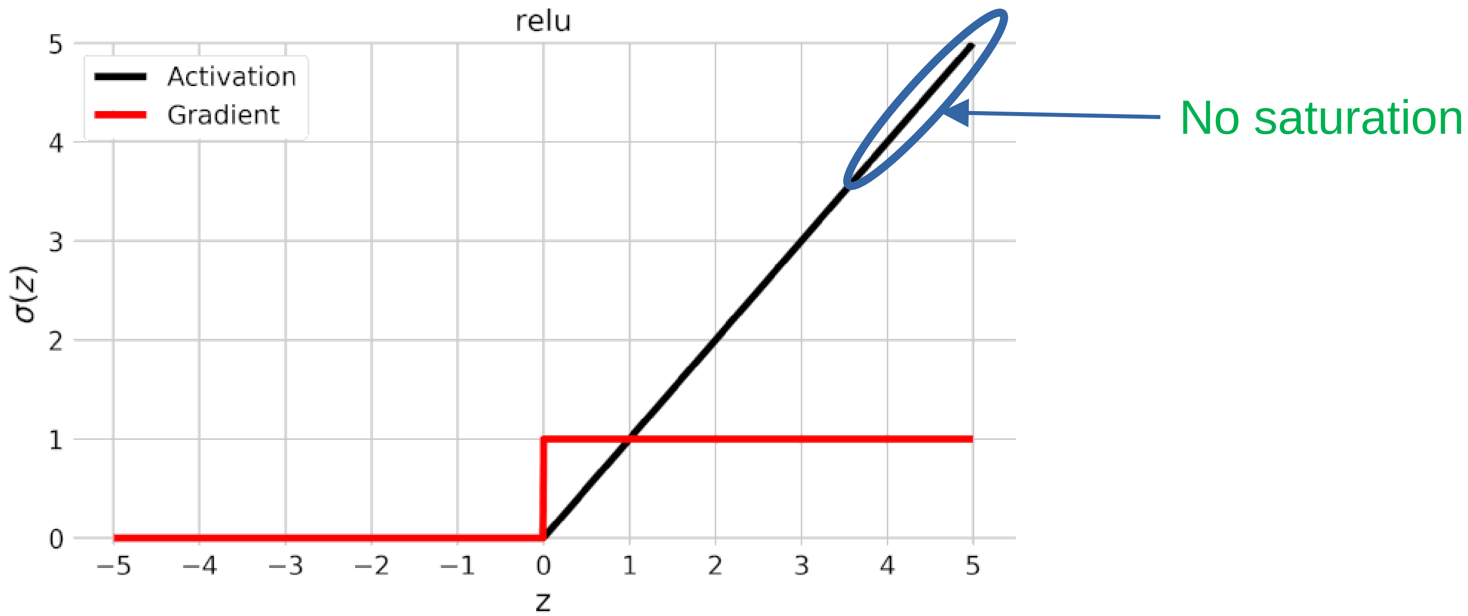
# Hyperbolic Tangent




$$\sigma(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- Pros: $\sigma(0) = 0$ symmetric
  $\sigma'(0) = 1$
- Cons: vanishing gradient for small and large inputs

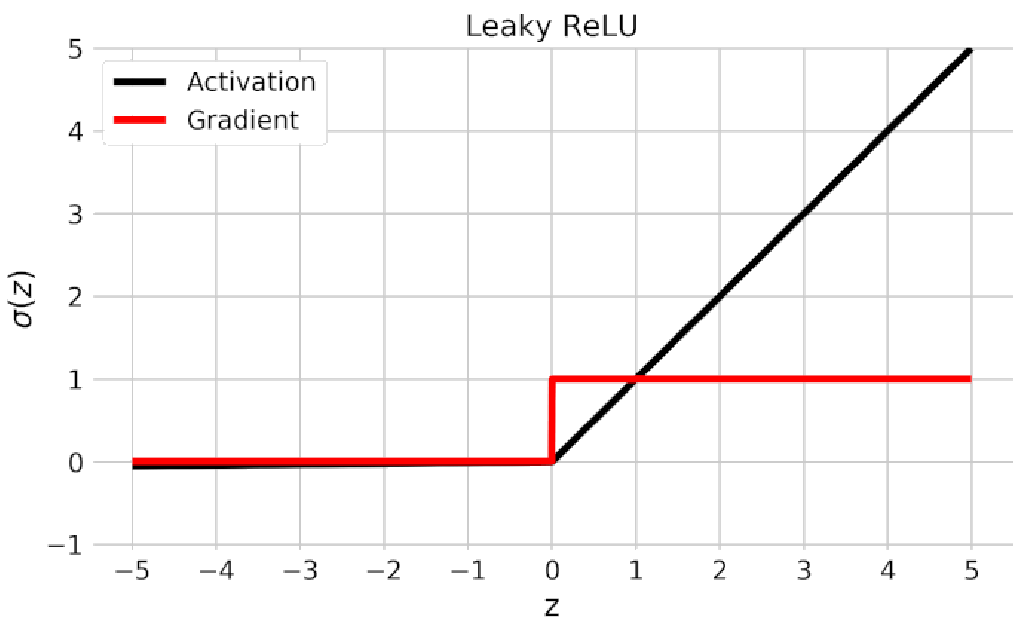
# ReLU




$$\sigma(z) = \max\{0, z\}$$

- Pros: derivatives in the positive domain are all 1
- Cons: output is not symmetric (internal covariate shift)
  “dead neuron” problem

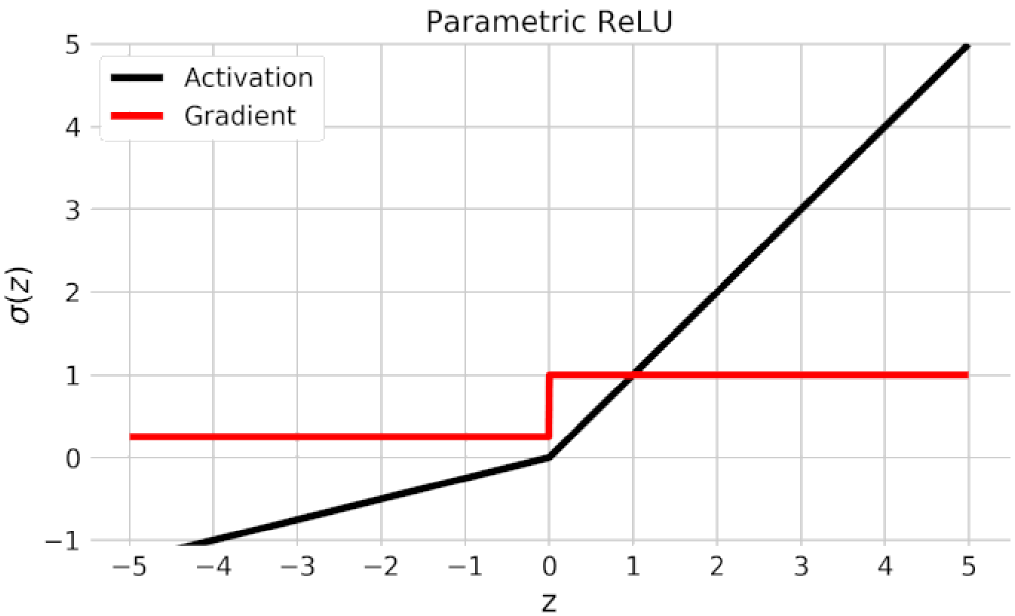
# Leaky ReLU




$$\sigma(z) = \max\{0, z\} + 0.01 \min\{0, z\}$$

- Pros: same as ReLU but solves "dead neuron" problem

# Parametric ReLU

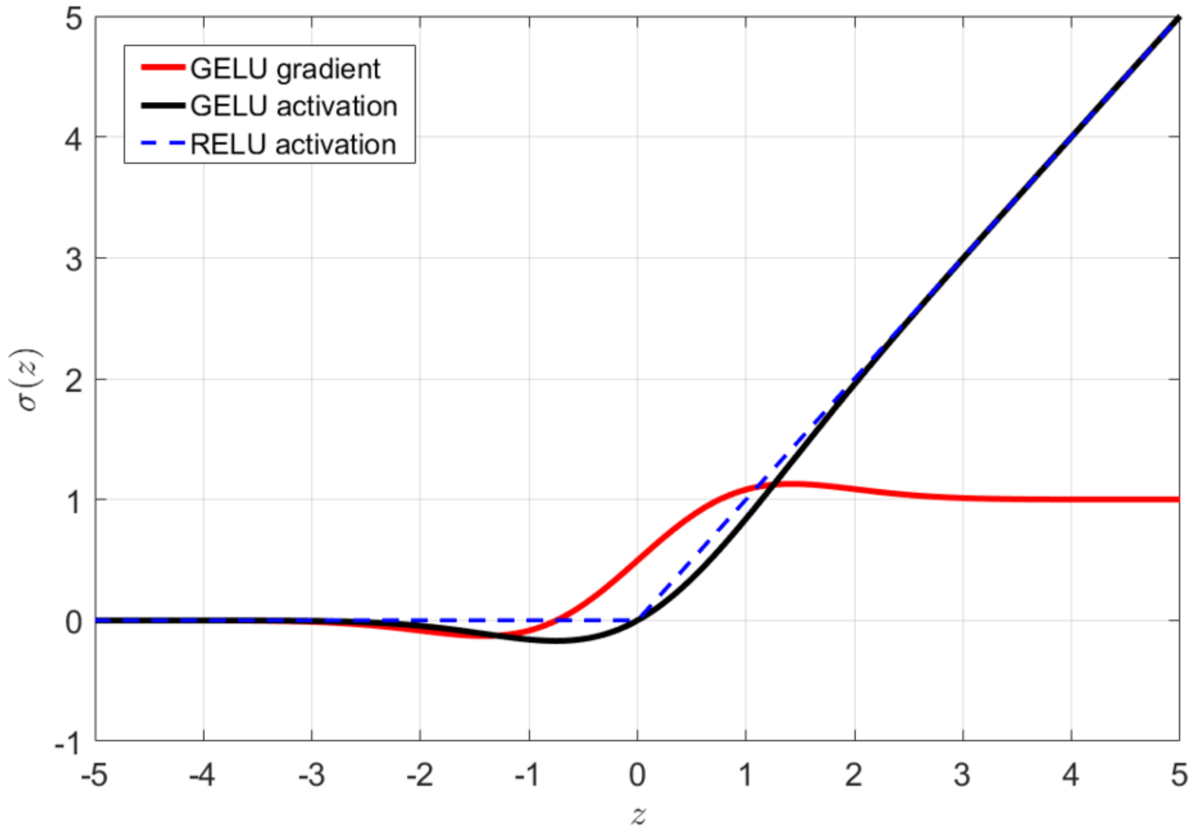



$$\sigma(z) = \max\{0, z\} + a \min\{0, z\}$$

- Pros: same as Leaky ReLU but parameterizes the leakage

# GELU

- Gaussian Error Linear Units



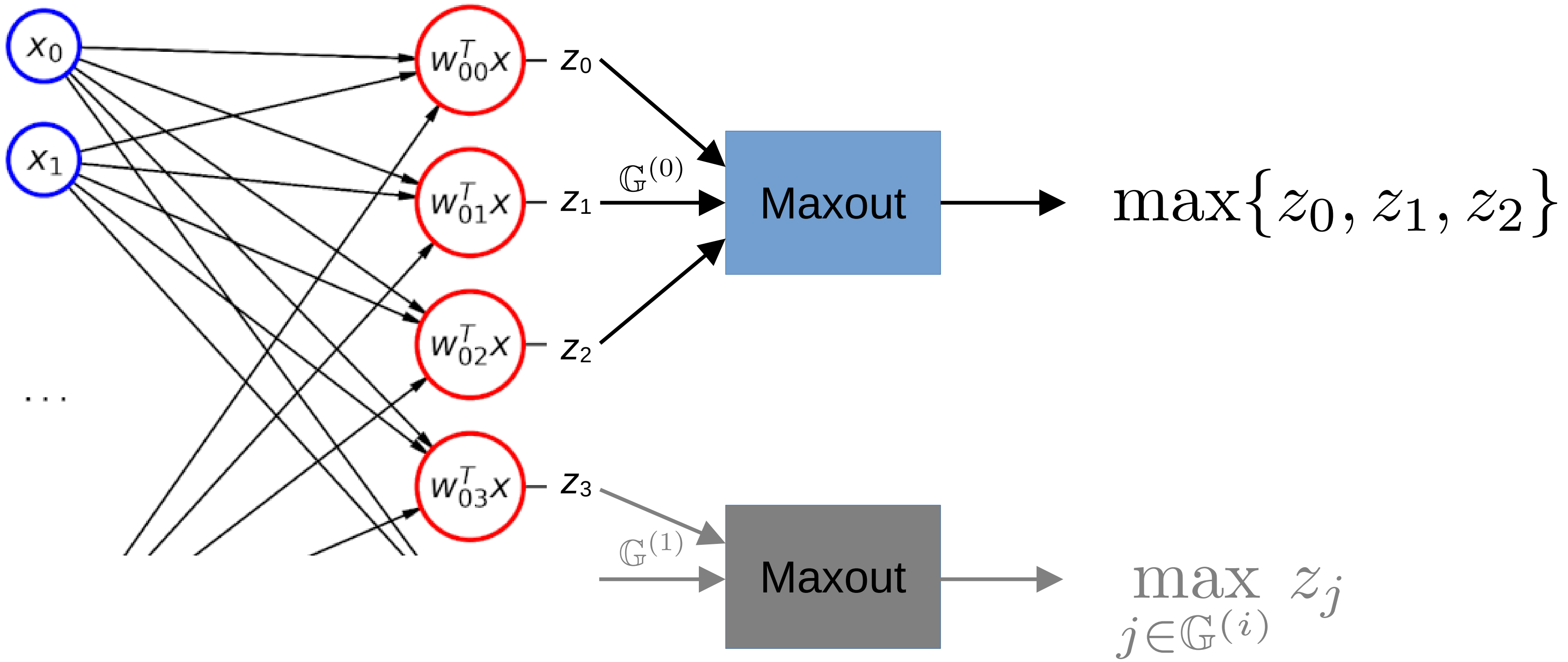

$$\sigma(z) = z\mathrm{P}(Z \leq z) \qquad Z \sim \mathcal{N}(0,1)$$

- Used in Transformers
- Similar: Swish, Mish

Hedrycks, Gimpel: GAUSSIAN ERROR LINEAR UNITS

# Maxout Units

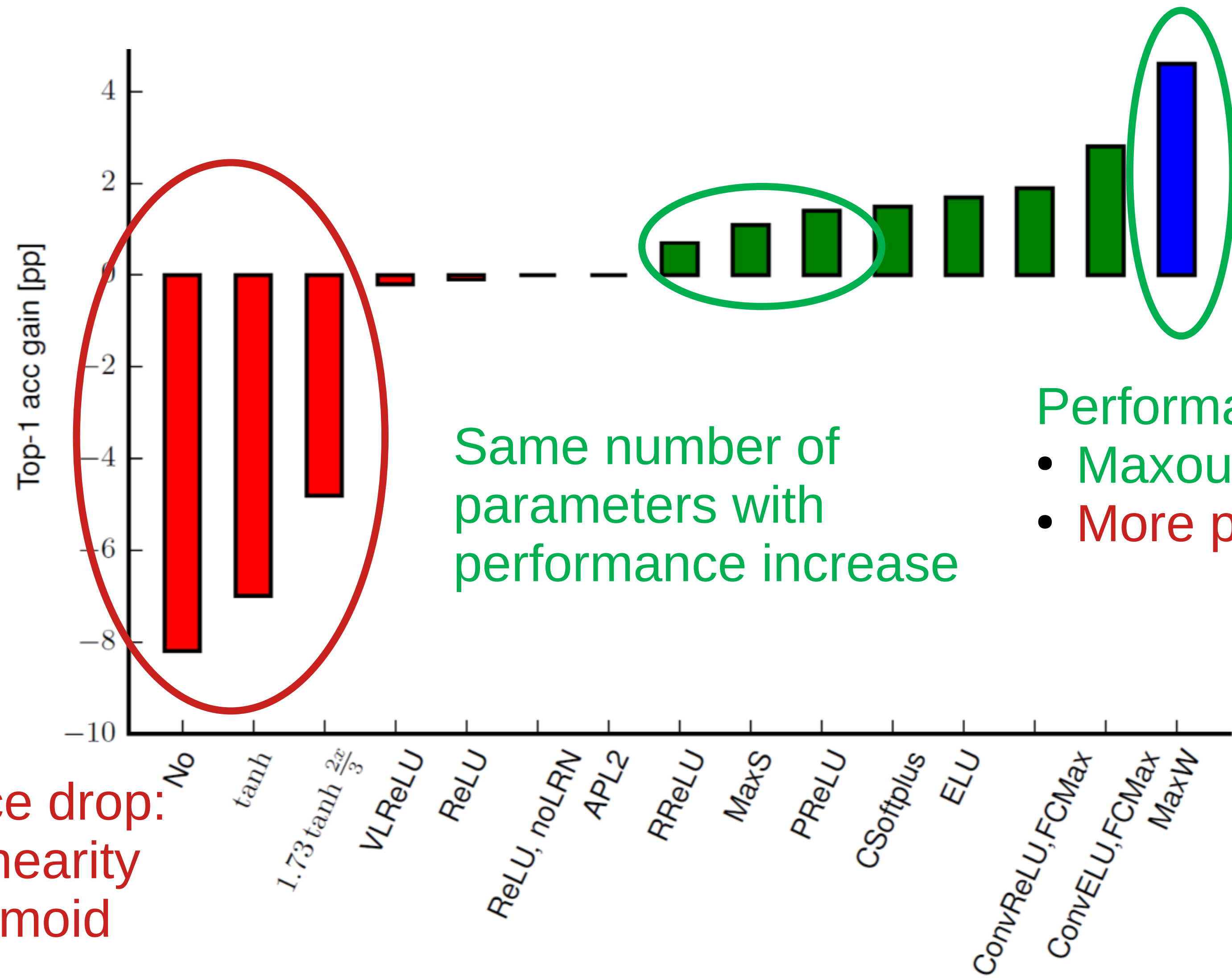
- Generalization of rectified linear units




- Can learn piecewise linear convex functions

# Quantitative Evaluation

- Classification performance vs. baseline with ReLU




Same number of parameters with performance increase

Performance increase
- Maxout
- More parameters

Performance drop:
- No non-linearity
- Tanh / sigmoid

Mishkin et.al., Systematic evaluation of CNN advances on the ImageNet, CoRR 2016

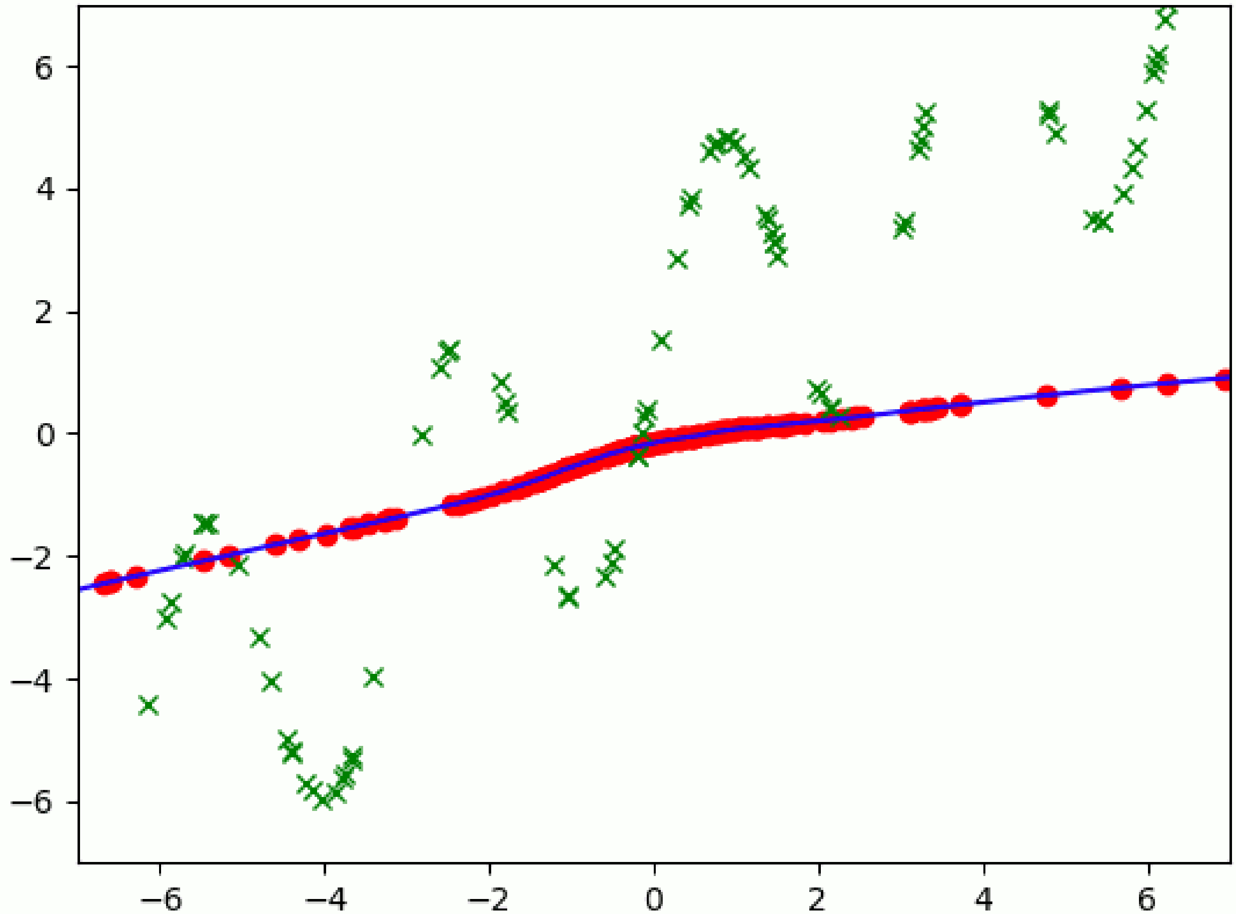
# Non-linearity conclusion

- Saturating non-linearities should be avoided
- ReLU is a standard choice for deep architectures
- Limited gains can be achieved with different types of non-linearities

# Universal Approximator

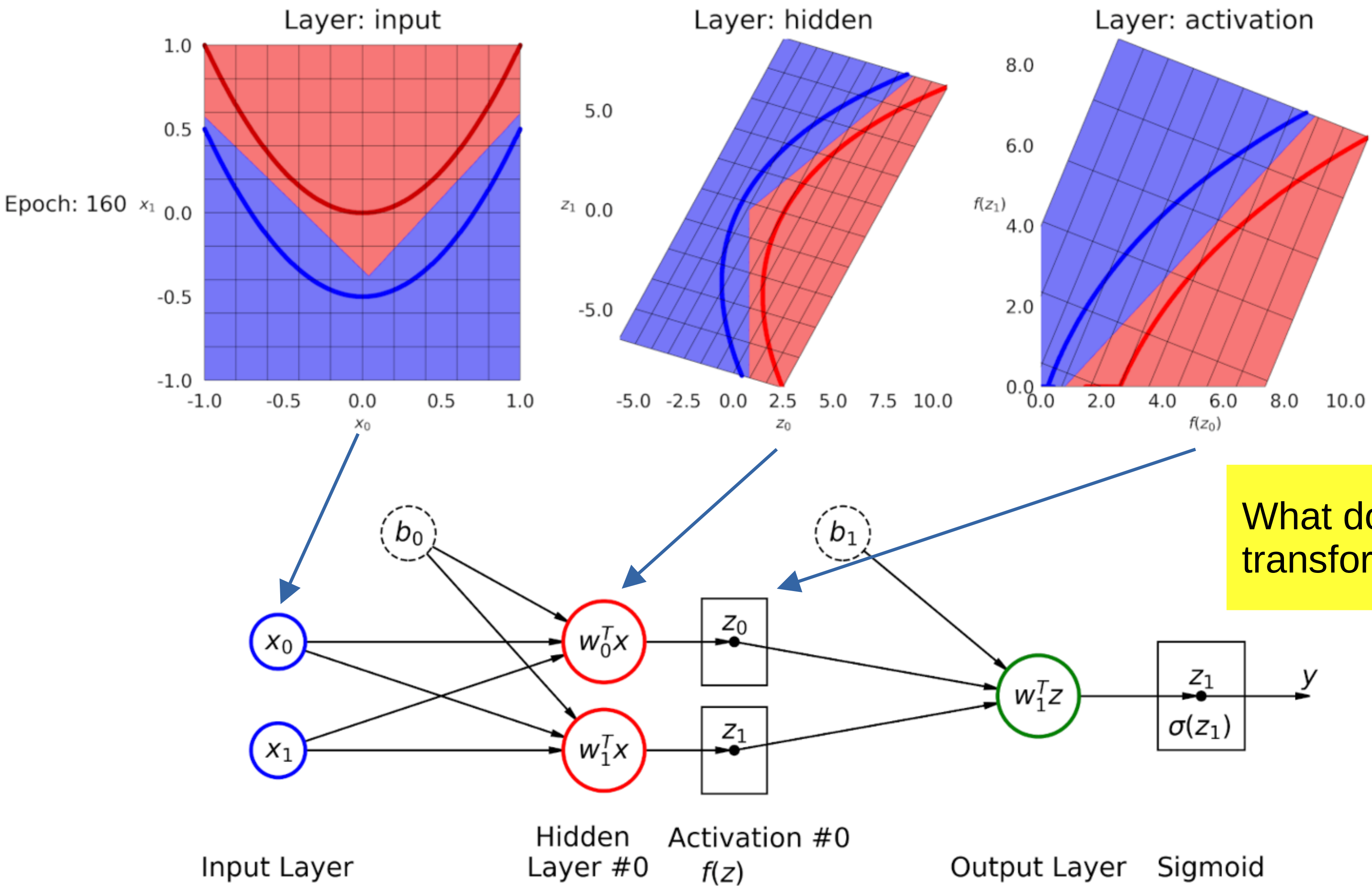
- Hornik [1989], Cybenko [1989]

+Proves existence
- Does not say how to learn
- No bounds for the capacity



# Effect of Activation Functions

- ReLU

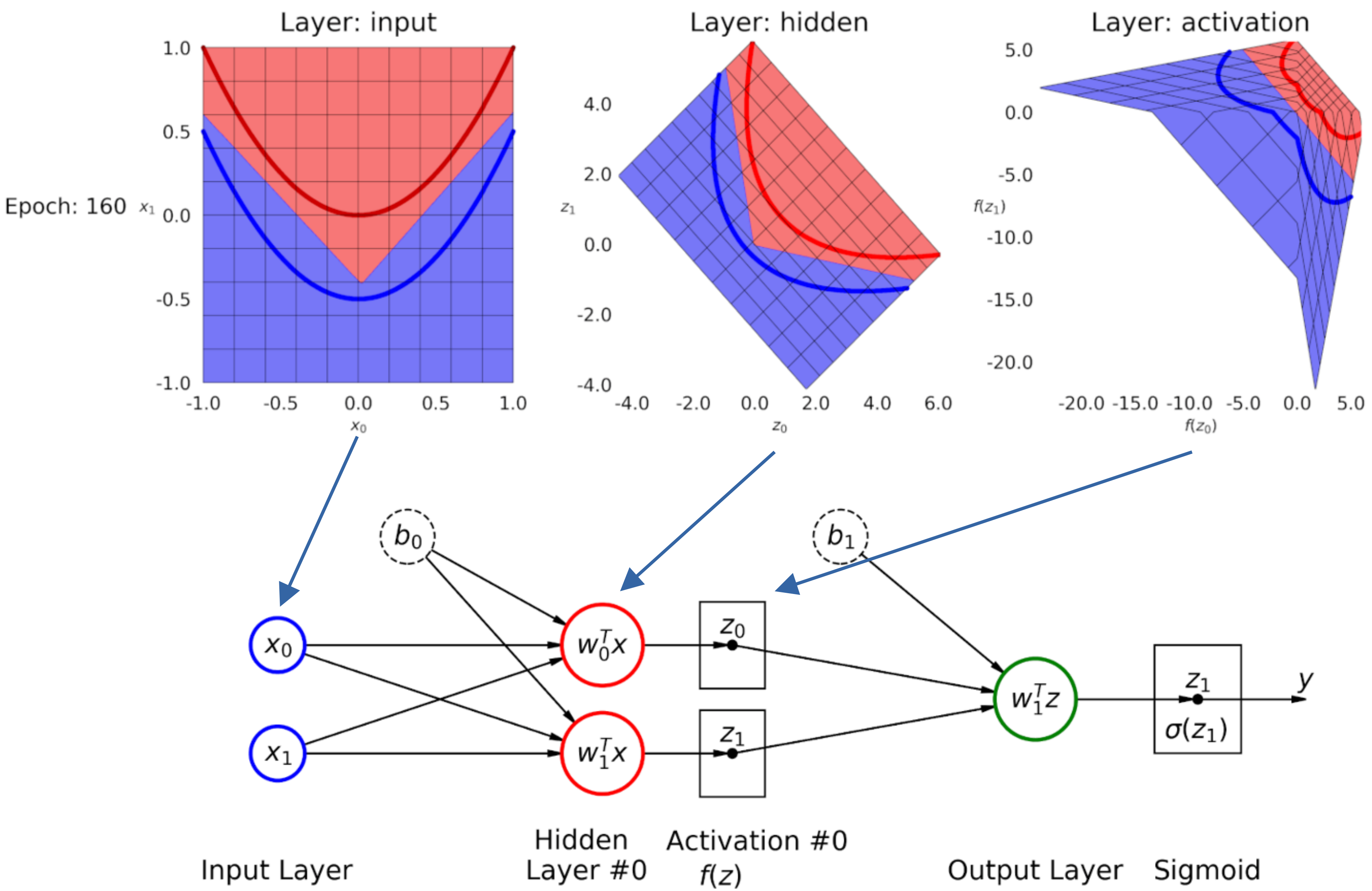



What does an affine transform preserve?

Godoy, Deep Learning with Pytorch, 2021

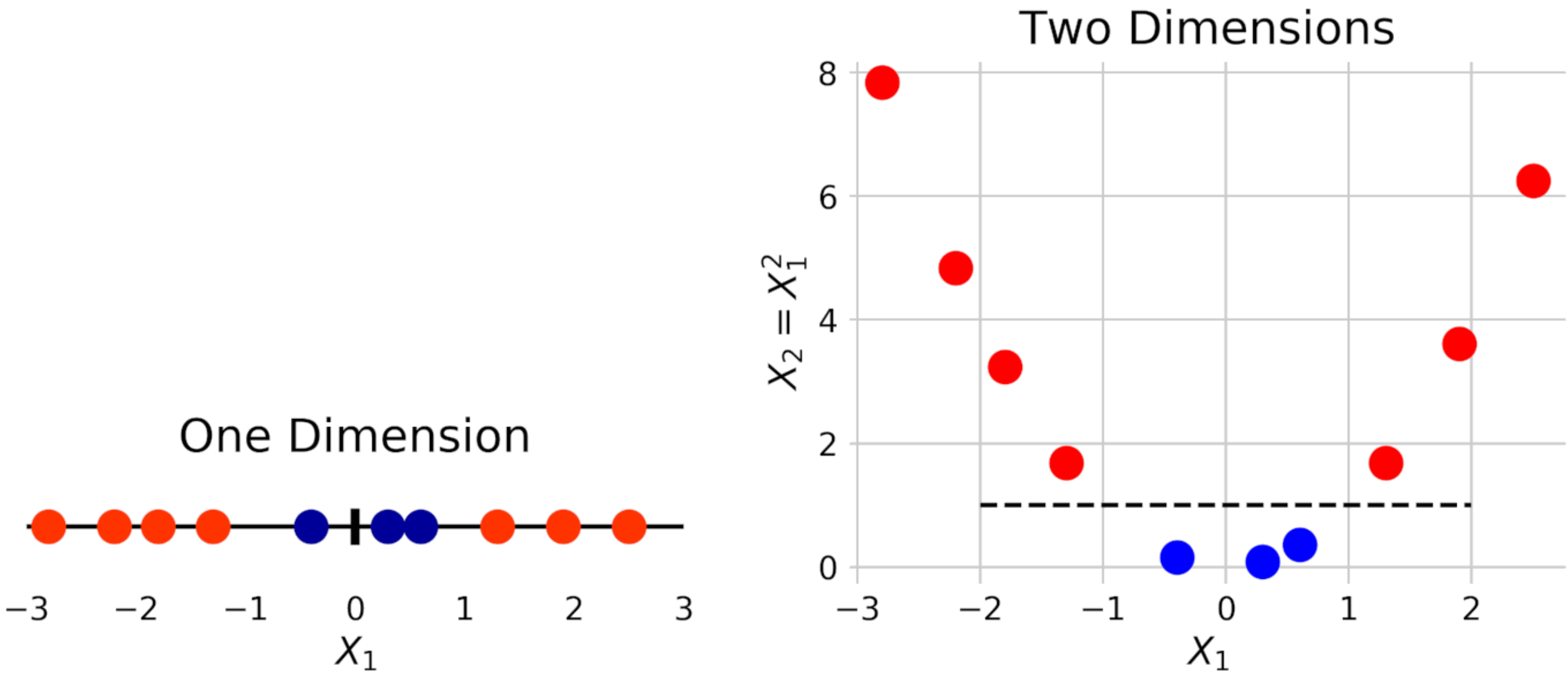
# Effect of Activation Functions

- PReLU




Godoy, Deep Learning with Pytorch, 2021

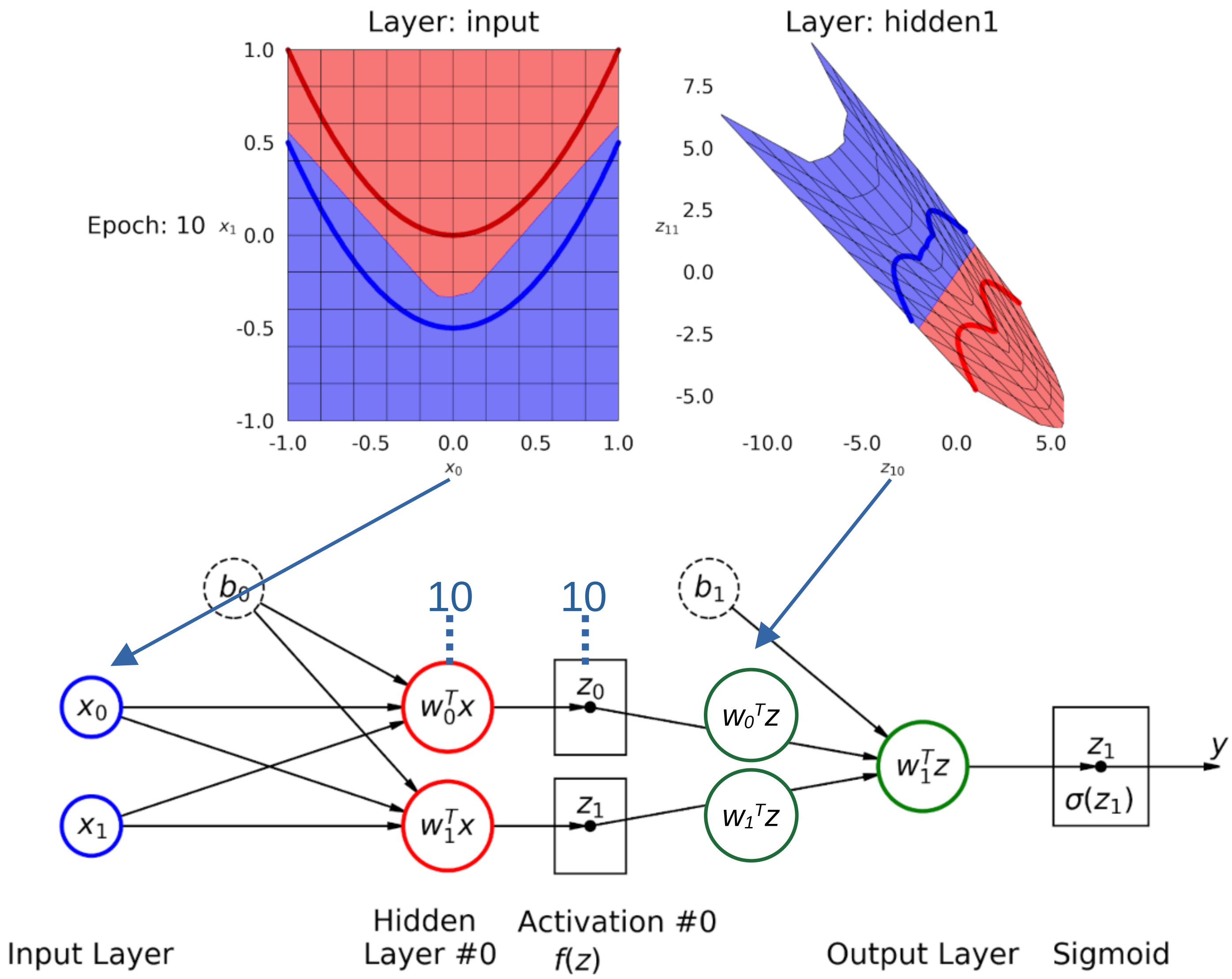
# Adding Dimensions




Godoy, Deep Learning with Pytorch, 2021

# Effect of Activation Functions

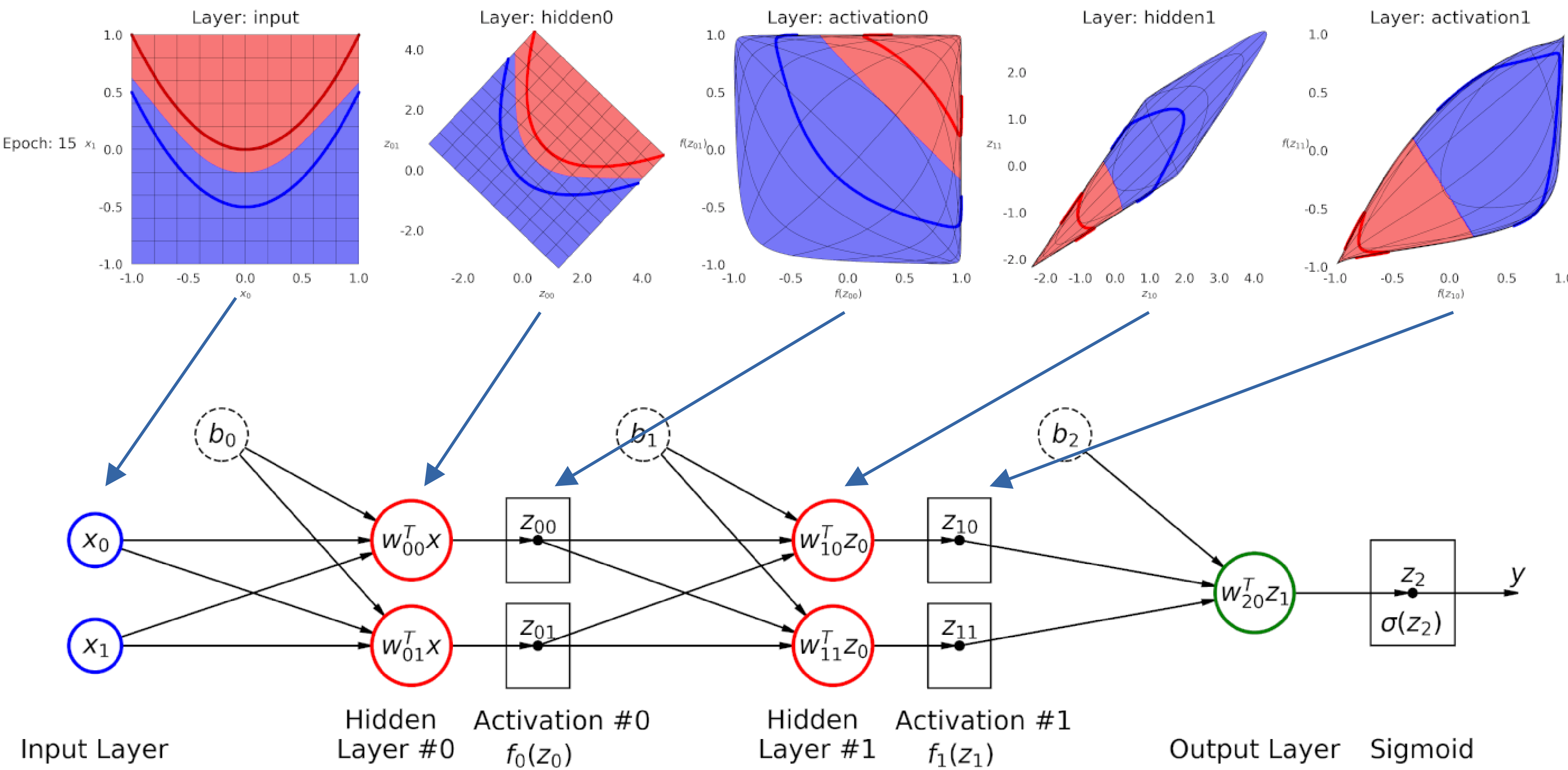
- More hidden units – only 10 epochs to learn




Godoy, Deep Learning with Pytorch, 2021

# Effect of Activation Functions

- More layers – only 15 epochs to learn




Godoy, Deep Learning with Pytorch, 2021

# Training Networks

- Find network parameters $\hat{\theta}$ (unit weights, biases, …) such that $\hat{\theta} = \arg\min_{\theta} J(\theta)$

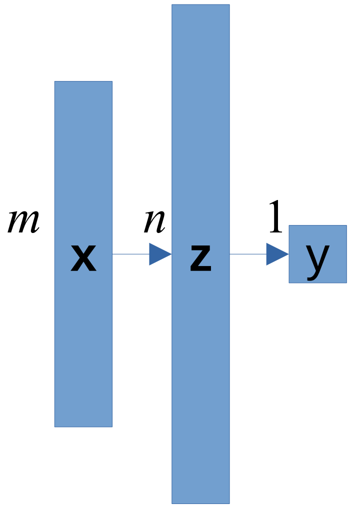
$$J(\theta) = \sum_{(x,y)\sim\hat{p}(x,y)} -\log q(y|f(x;\theta))$$

- Gradient descent is practically the only used minimization strategy
- Fast way to find gradients ==> back-propagation

# Chain Rule N-D

- N-D function composition: $y = g(f(\mathbf{x}))$

$$y = g(\mathbf{z}) : \mathbb{R}^n \to \mathbb{R}$$
$$\mathbf{z} = f(\mathbf{x}) : \mathbb{R}^m \to \mathbb{R}^n$$




- Jacobian:

$$\nabla_{\mathbf{x}} f = \begin{bmatrix} \frac{dz_1}{dx_1} & \frac{dz_1}{dx_2} & \cdots & \frac{dz_1}{dx_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{dz_n}{dx_1} & \frac{dz_n}{dx_2} & \cdots & \frac{dz_n}{dx_m} \end{bmatrix}$$

$$\nabla_{\mathbf{z}} g = \begin{bmatrix} \frac{dy}{dz_1} & \frac{dy}{dz_2} & \cdots & \frac{dy}{dz_n} \end{bmatrix}$$

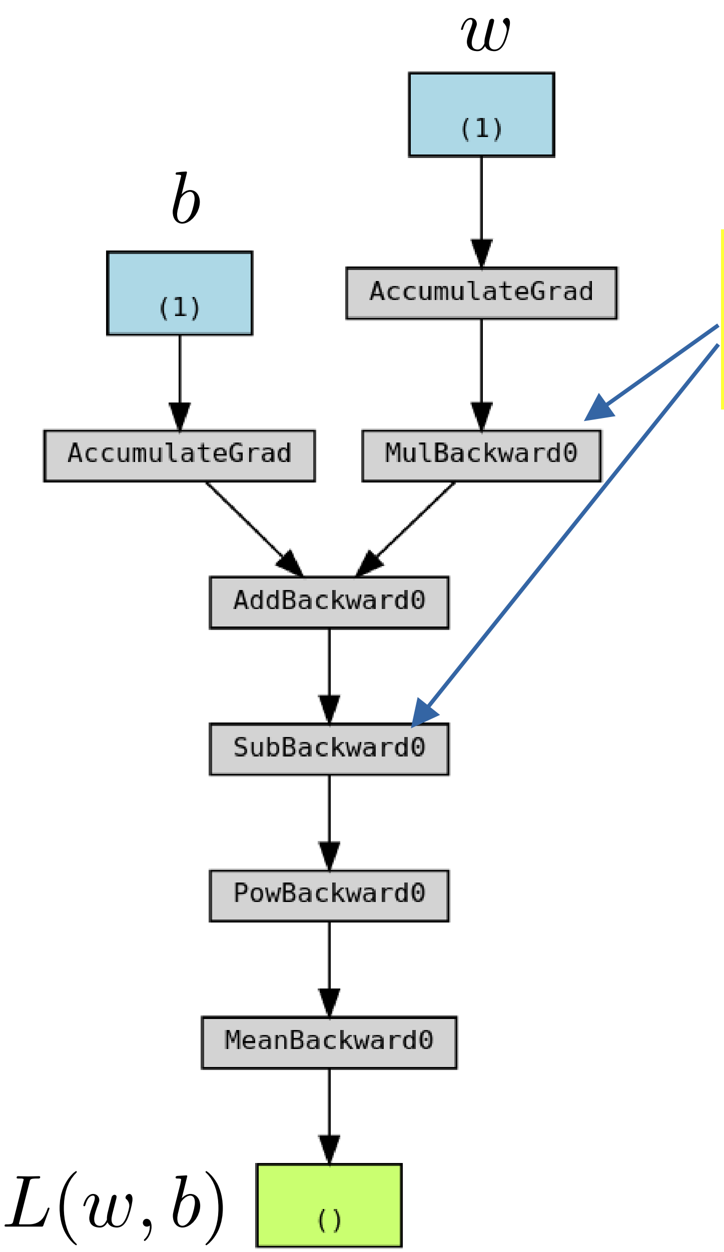
$$\frac{dy}{d\mathbf{x}} = \nabla_{\mathbf{z}} g . \nabla_{\mathbf{x}} f$$

# General Chain Rule

$$z^{(P)}(\dots z^{(p)}(z^{(1)}(z^{(0)}))\dots)$$

$$\frac{dz^{(P)}}{dz^{(0)}} = \prod_{p=0}^{P-1} \nabla_{z^{(P-p-1)}} z^{(P-p)}$$

# Computational Graph




Why are **x**, **y** missing in the graph?

$$L(w, b) = ||(w\mathbf{x} + b) - \mathbf{y})||^2$$

$$\mathbf{z}^{(0)} = w\mathbf{x}$$

$$\mathbf{z}^{(1)} = \mathbf{z}^{(0)} + b$$

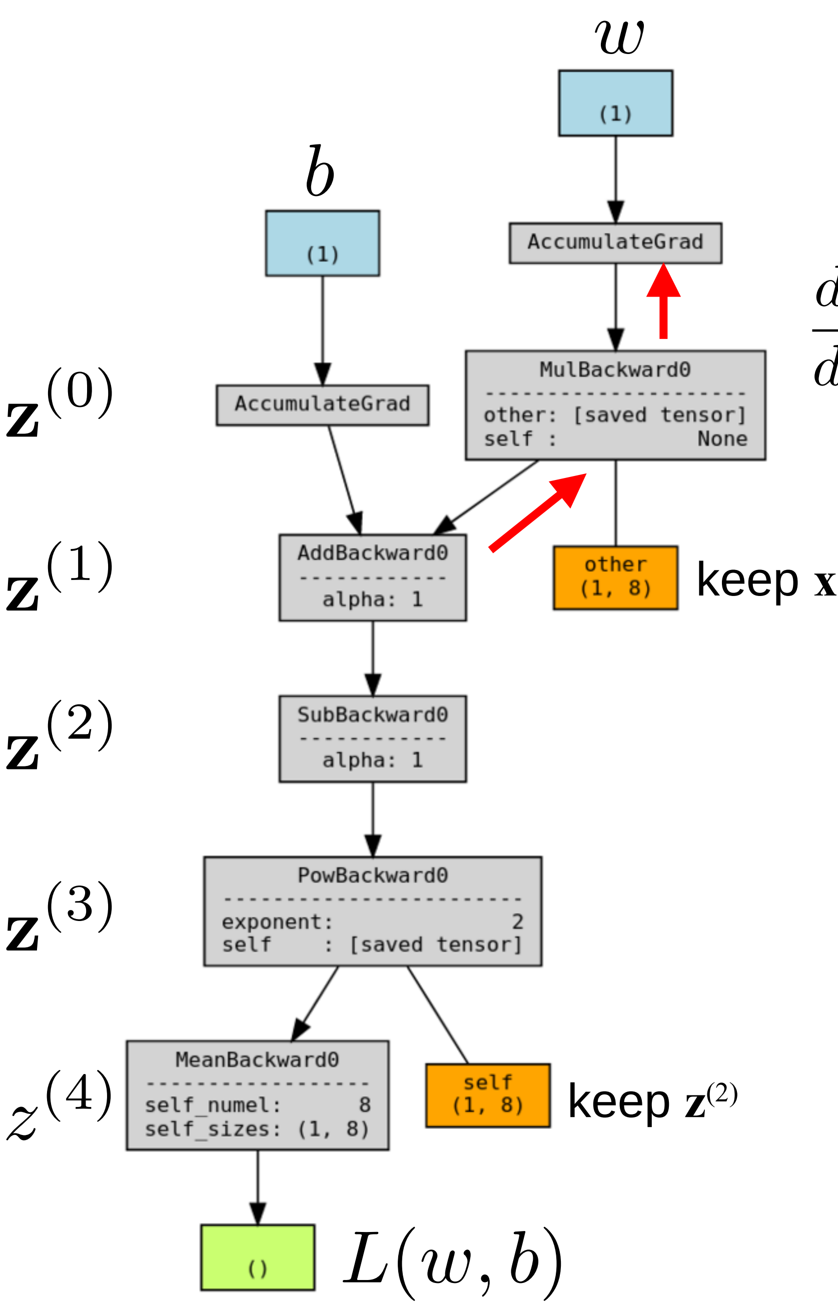
$$\mathbf{z}^{(2)} = \mathbf{z}^{(1)} - \mathbf{y}$$

$$z_i^{(3)} = [z_i^{(2)}]^2$$

$$z^{(4)} = \sum_i z_i^{(3)}$$

# Back-Propagation

$$\mathbf{z}^{(0)} = w\mathbf{x}$$
$$\mathbf{z}^{(1)} = \mathbf{z}^{(0)} + b$$
$$\mathbf{z}^{(2)} = \mathbf{z}^{(1)} - \mathbf{y}$$
$$z_i^{(3)} = [z_i^{(2)}]^2$$
$$z^{(4)} = \sum_i z_i^{(3)}$$




$$\frac{dL}{dw} = \nabla_{\mathbf{z}^{(3)}} z^{(4)} . \nabla_{\mathbf{z}^{(2)}} \mathbf{z}^{(3)} . \nabla_{\mathbf{z}^{(1)}} \mathbf{z}^{(2)} . \nabla_{\mathbf{z}^{(0)}} \mathbf{z}^{(1)} . \nabla_{w} \mathbf{z}^{(0)}$$

$$\nabla_w \mathbf{z}^{(0)} = \mathbf{x}$$

$$\nabla_{\mathbf{z}^{(0)}} \mathbf{z}^{(1)} = \mathbf{I}$$

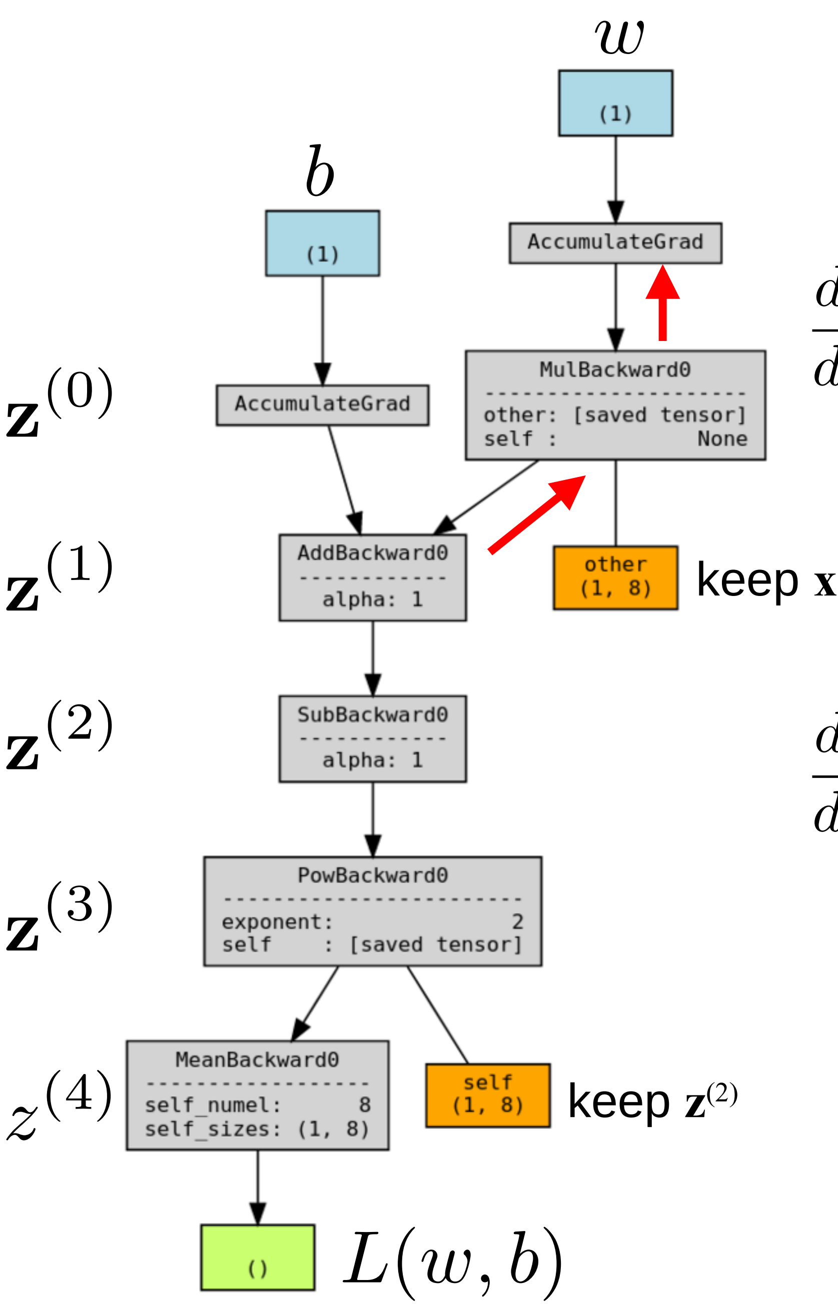
$$\nabla_{\mathbf{z}^{(1)}} \mathbf{z}^{(2)} = \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix} = \mathbf{I}$$

$$\nabla_{\mathbf{z}^{(2)}} \mathbf{z}^{(3)} = \begin{bmatrix} 2z_1^{(2)} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 2z_n^{(2)} \end{bmatrix} = \mathrm{diag}(2\mathbf{z}^{(2)})$$

$$\nabla_{\mathbf{z}^{(3)}} z^{(4)} = [1, 1, \dots, 1] = \mathbf{1}^T$$

# Back-Propagation

$$
\begin{aligned}
\mathbf{z}^{(0)} &= w\mathbf{x} \\
\mathbf{z}^{(1)} &= \mathbf{z}^{(0)} + b \\
\mathbf{z}^{(2)} &= \mathbf{z}^{(1)} - \mathbf{y} \\
z_i^{(3)} &= [z_i^{(2)}]^2 \\
z^{(4)} &= \sum_i z_i^{(3)}
\end{aligned}
$$

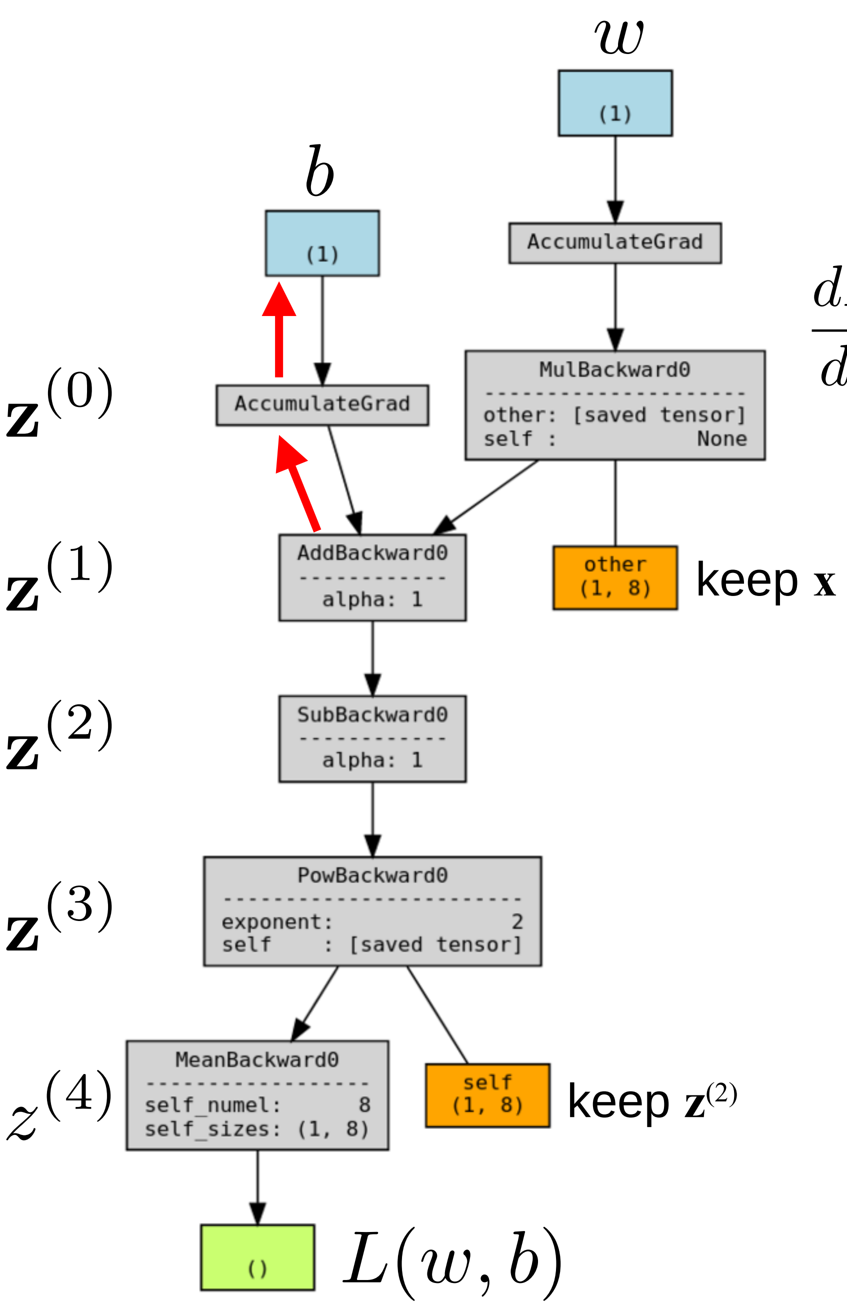



$$\frac{dL}{dw} = \nabla_{\mathbf{z}^{(3)}} z^{(4)} . \nabla_{\mathbf{z}^{(2)}} \mathbf{z}^{(3)} . \nabla_{\mathbf{z}^{(1)}} \mathbf{z}^{(2)} . \nabla_{\mathbf{z}^{(0)}} \mathbf{z}^{(1)} . \nabla_w \mathbf{z}^{(0)}$$

$$\frac{dL}{dw} = \sum_i 2 z_i^{(2)} x_i$$

# Back-Propagation

$$\mathbf{z}^{(0)} = w\mathbf{x}$$
$$\mathbf{z}^{(1)} = \mathbf{z}^{(0)} + b$$
$$\mathbf{z}^{(2)} = \mathbf{z}^{(1)} - \mathbf{y}$$
$$z_i^{(3)} = [z_i^{(2)}]^2$$
$$z^{(4)} = \sum_i z_i^{(3)}$$




$$\frac{dL}{db} = \nabla_{\mathbf{z}^{(3)}} z^{(4)} . \nabla_{\mathbf{z}^{(2)}} \mathbf{z}^{(3)} . \nabla_{\mathbf{z}^{(1)}} \mathbf{z}^{(2)} . \nabla_b \mathbf{z}^{(1)}$$

$$\nabla_b \mathbf{z}^{(1)} = \mathbf{1}$$

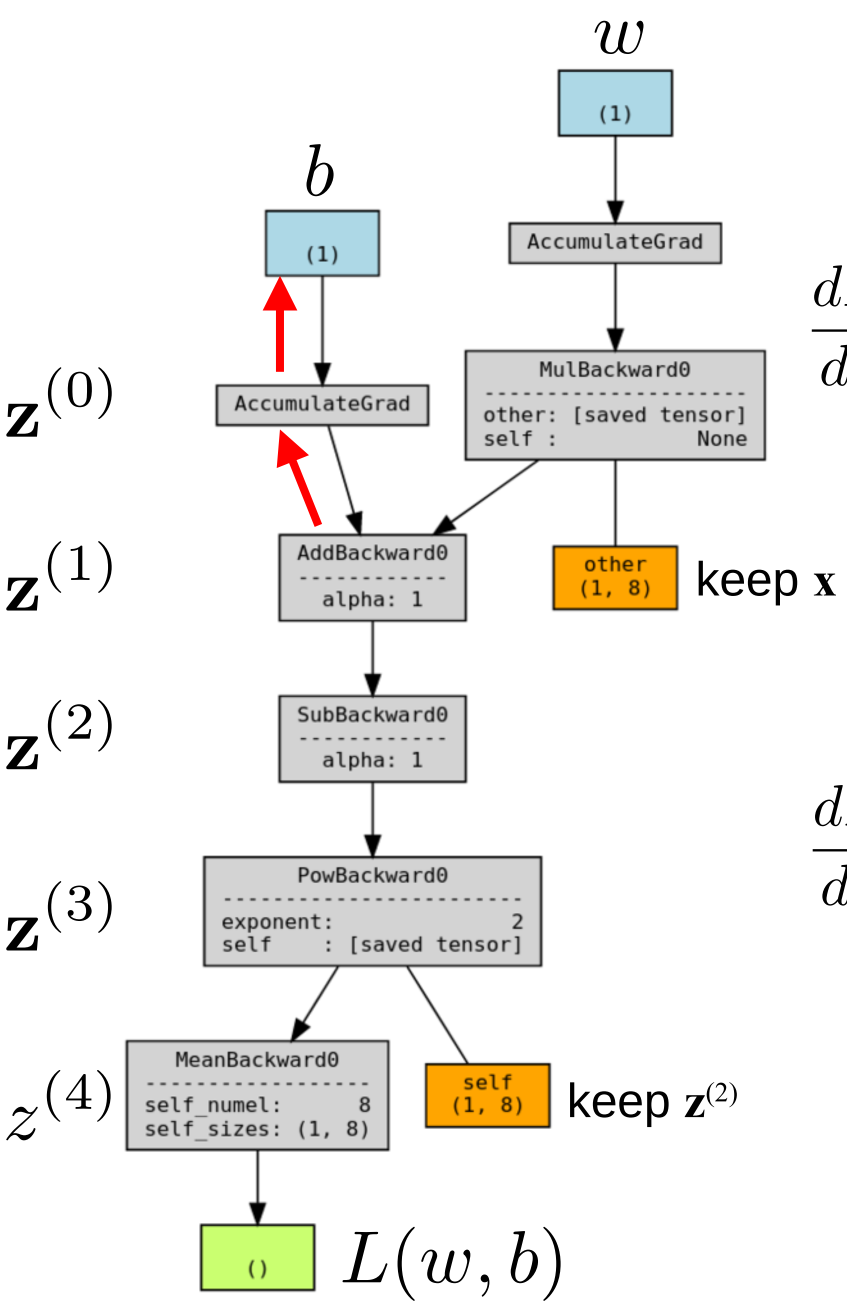
$$\nabla_{\mathbf{z}^{(1)}} \mathbf{z}^{(2)} = \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix} = \mathbf{I}$$

$$\nabla_{\mathbf{z}^{(2)}} \mathbf{z}^{(3)} = \begin{bmatrix} 2z_1^{(2)} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 2z_n^{(2)} \end{bmatrix} = \mathrm{diag}(2\mathbf{z}^{(2)})$$

$$\nabla_{\mathbf{z}^{(3)}} z^{(4)} = [1, 1, \dots, 1] = \mathbf{1}^T$$

# Back-Propagation

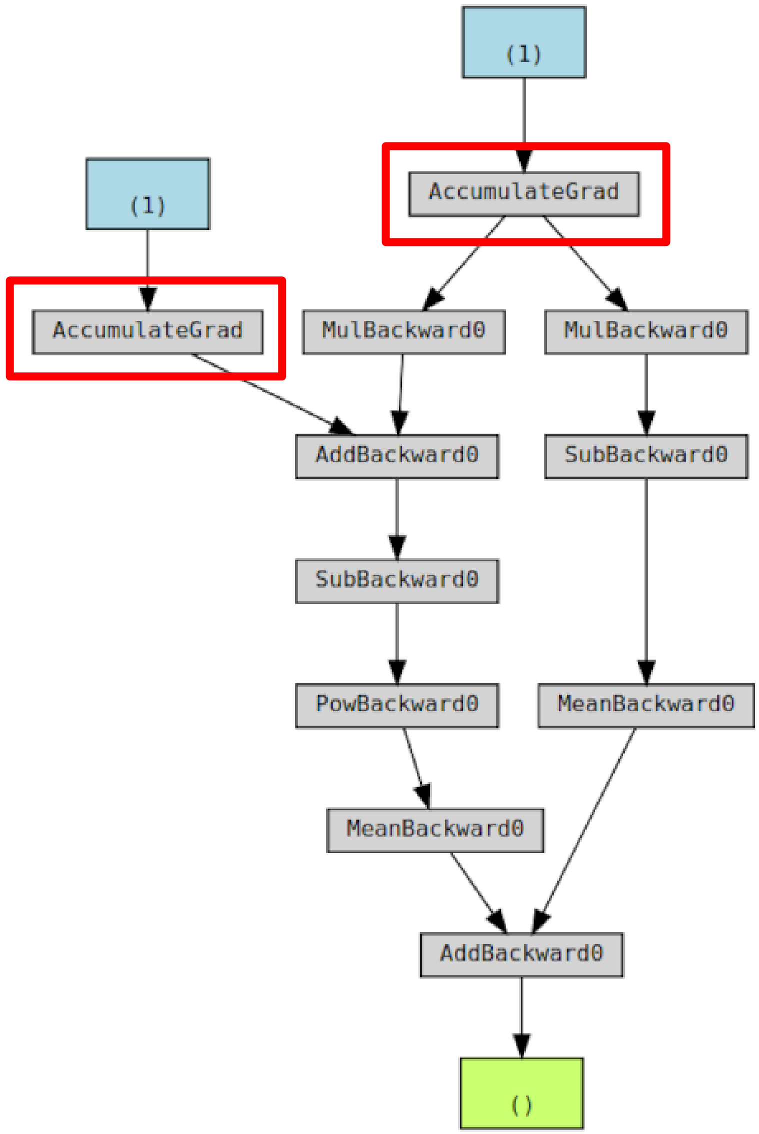
$$\mathbf{z}^{(0)} = w\mathbf{x}$$
$$\mathbf{z}^{(1)} = \mathbf{z}^{(0)} + b$$
$$\mathbf{z}^{(2)} = \mathbf{z}^{(1)} - \mathbf{y}$$
$$z_i^{(3)} = [z_i^{(2)}]^2$$
$$z^{(4)} = \sum_i z_i^{(3)}$$




$$\frac{dL}{db} = \nabla_{\mathbf{z}^{(3)}} z^{(4)} . \nabla_{\mathbf{z}^{(2)}} \mathbf{z}^{(3)} . \nabla_{\mathbf{z}^{(1)}} \mathbf{z}^{(2)} . \nabla_b \mathbf{z}^{(1)}$$

$$\frac{dL}{db} = \sum_i 2 z_i^{(2)}$$

# Complex Computational Graphs




PytorchViz

# Learning vs Optimization

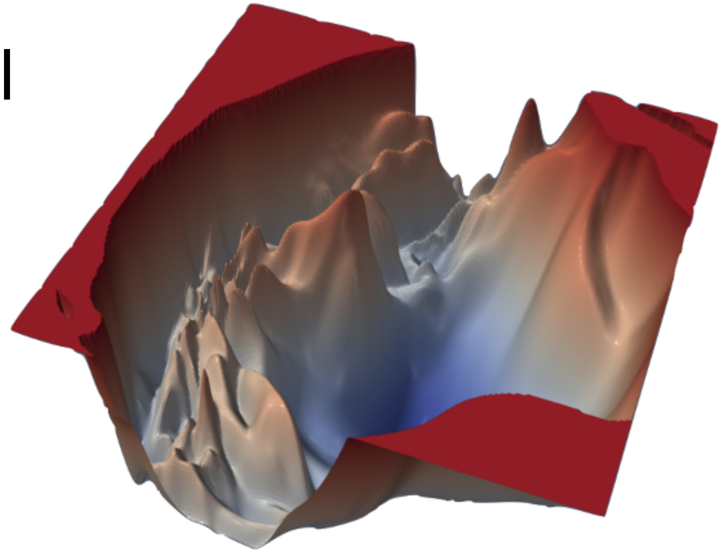
- Maximizing Performance $\neq$ Minimizing cost function
- Cost function

$$J(\theta) = \mathbb{E}_{(x,y)\sim p_{\text{data}}} L(f(x,\theta), y)$$

- Instead, Empirical risk

$$J^*(\theta) = \sum_i L(f(x^{(i)},\theta), y^{(i)})$$

- Many minima, which one is better?
- Reaching a minimum is often not a goal



# Batch vs Minibatch Gradient Methods

- Infeasible to calculate the derivative over the whole dataset

$$\{(x^{(i)}, y^{(i)})\}_{i \in \mathbb{D}} \qquad \nabla_\theta J^*(\theta) = \sum_{i \in \mathbb{D}} \nabla_\theta L(f(x^{(i)}, \theta), y^{(i)})$$

- Instead, derivatives over random data subsets (minibatches)

$$\mathbb{B} = \{\text{random set from } \mathbb{D}\}$$

$$\nabla_\theta J^*(\theta) \approx \sum_{i \in \mathbb{B}} \nabla_\theta L(f(x^{(i)}, \theta), y^{(i)})$$

  - Parallel implementation on GPUs
  - Small minibatches → good generalization properties

# Stochastic Gradient Descent

1. Initialization: parameters $\theta^{(0)}$, learning rate $\epsilon_t$

2. Sample a minibatch $\mathbb{B} \subset \mathbb{D}$

3. Compute gradient $g = \frac{1}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} \nabla_\theta L(f(x^{(i)}, \theta^{(t)}), y^{(i)})$

4. Apply update $\theta^{(t+1)} = \theta^{(t)} - \epsilon_t g$

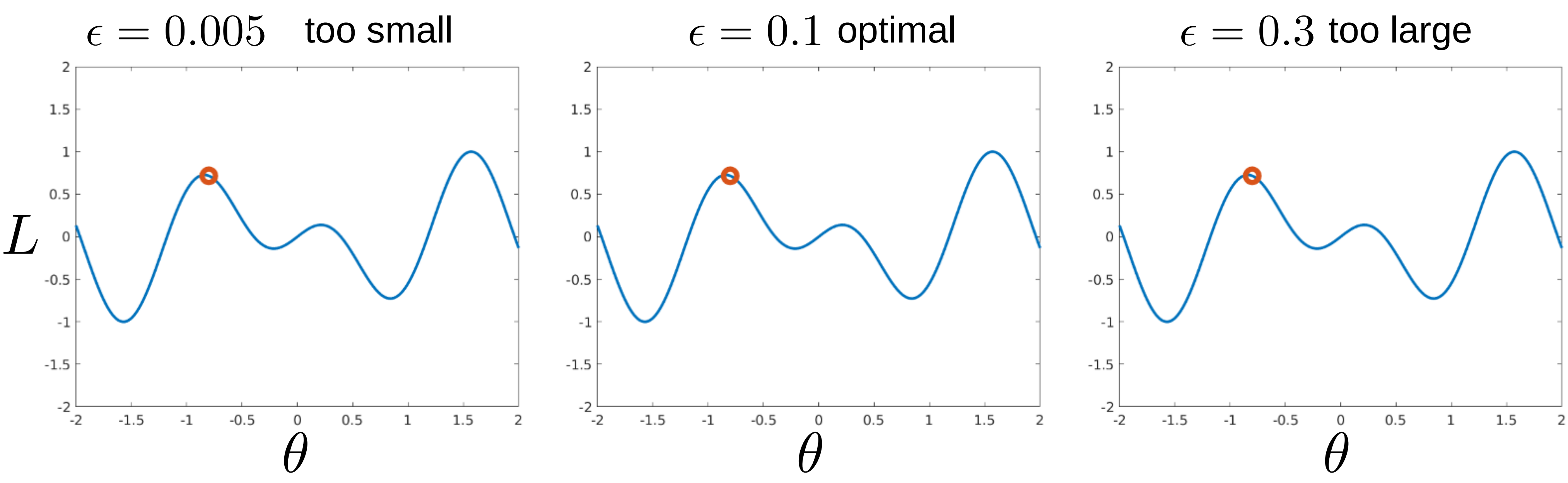
5. Repeat 2-4 until stopping criterion is met

# SGD

- Sufficient condition to guarantee convergence of SGD

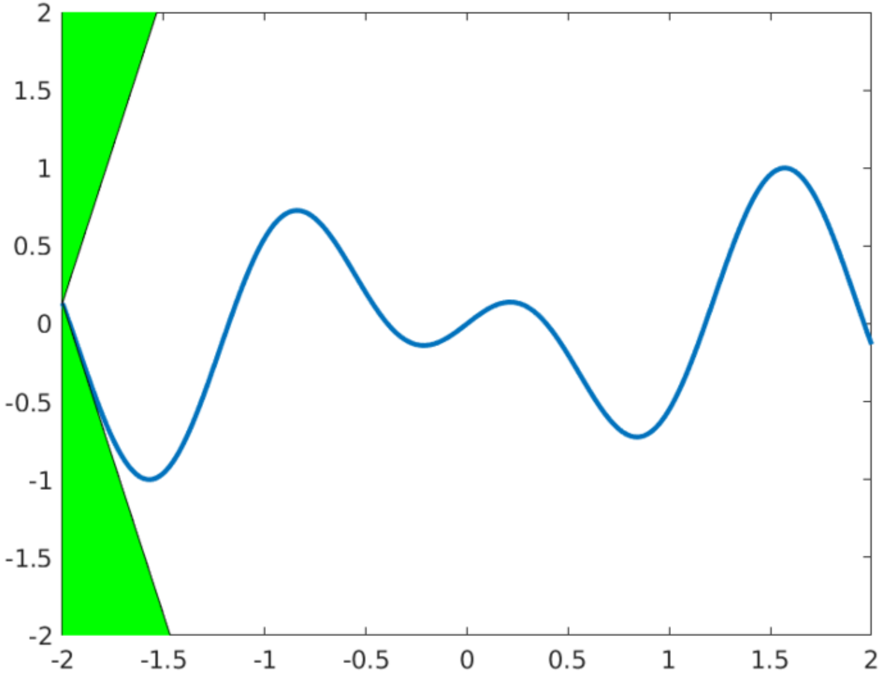
$$\sum_{t=1}^{\infty} \epsilon_t = \infty \qquad \sum_{t=1}^{\infty} \epsilon_t^2 < \infty$$

- Choice of the LR is critical! $\theta^{(t+1)} = \theta^{(t)} - \epsilon_t g$



# Lipschitz Constant

- $$|f(x_1) - f(x_2)| \leq K\|x_1 - x_2\| \quad \forall x_1, x_2$$




$$f(x) = \sin(x)\cos(4x)$$

$$\sup_{\mathbb{R}} f' = 4$$

$$K = 4$$

- Learning rate: $0 < \epsilon < 2/K$

# SGD with Momentum

- Adds momentum to the gradient

$$g = \frac{1}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} \nabla_\theta L(f(x^{(i)}, \theta^{(t)}), y^{(i)})$$

Momentum factor

$$v^{(t+1)} = \alpha v^{(t)} + g$$

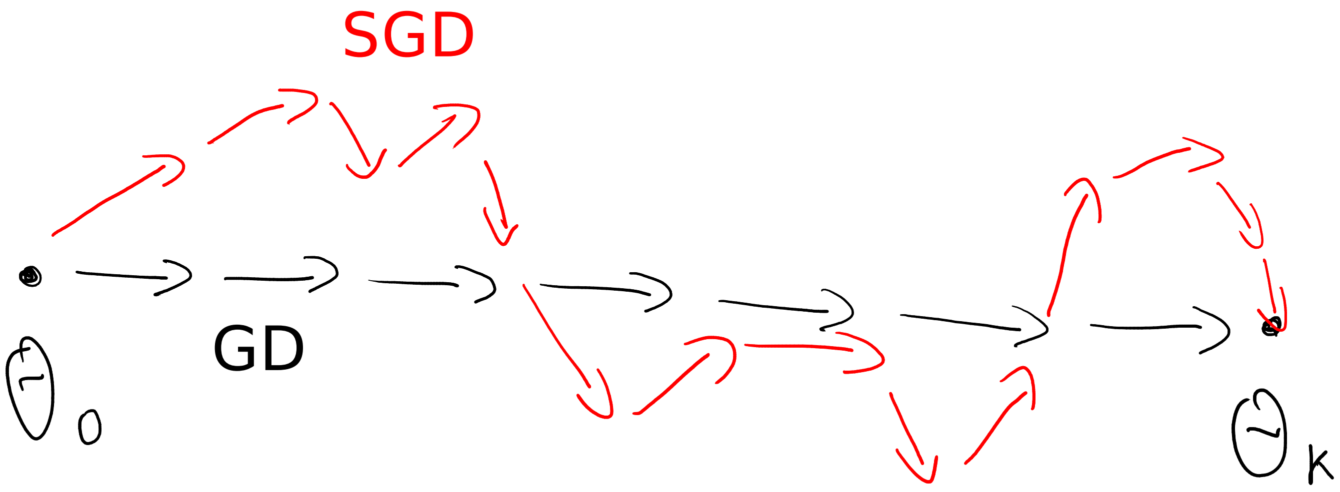
Velocity (momentum of unit mass)

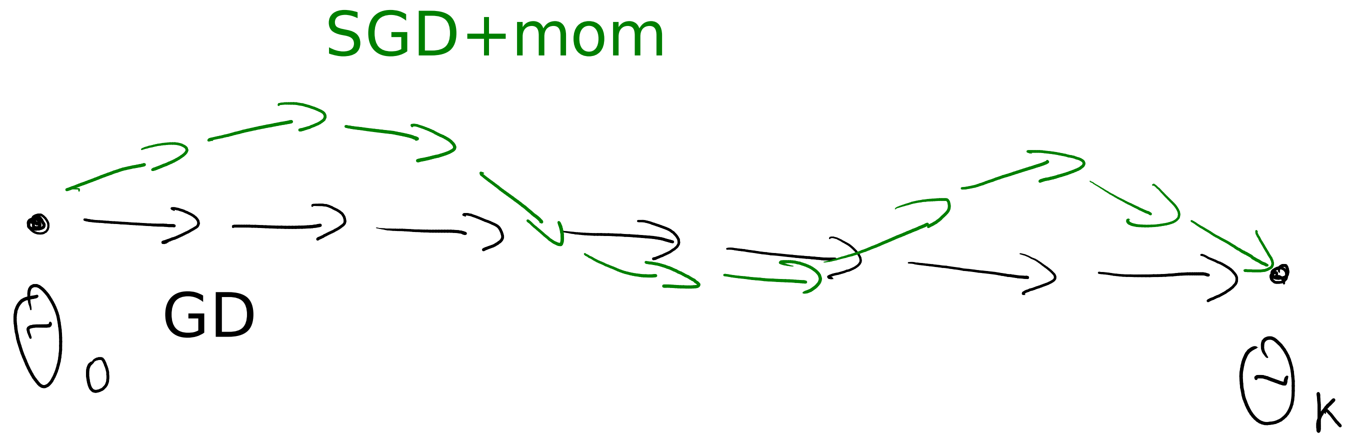
$$\theta^{(t+1)} = \theta^{(t)} - \epsilon v^{(t+1)}$$

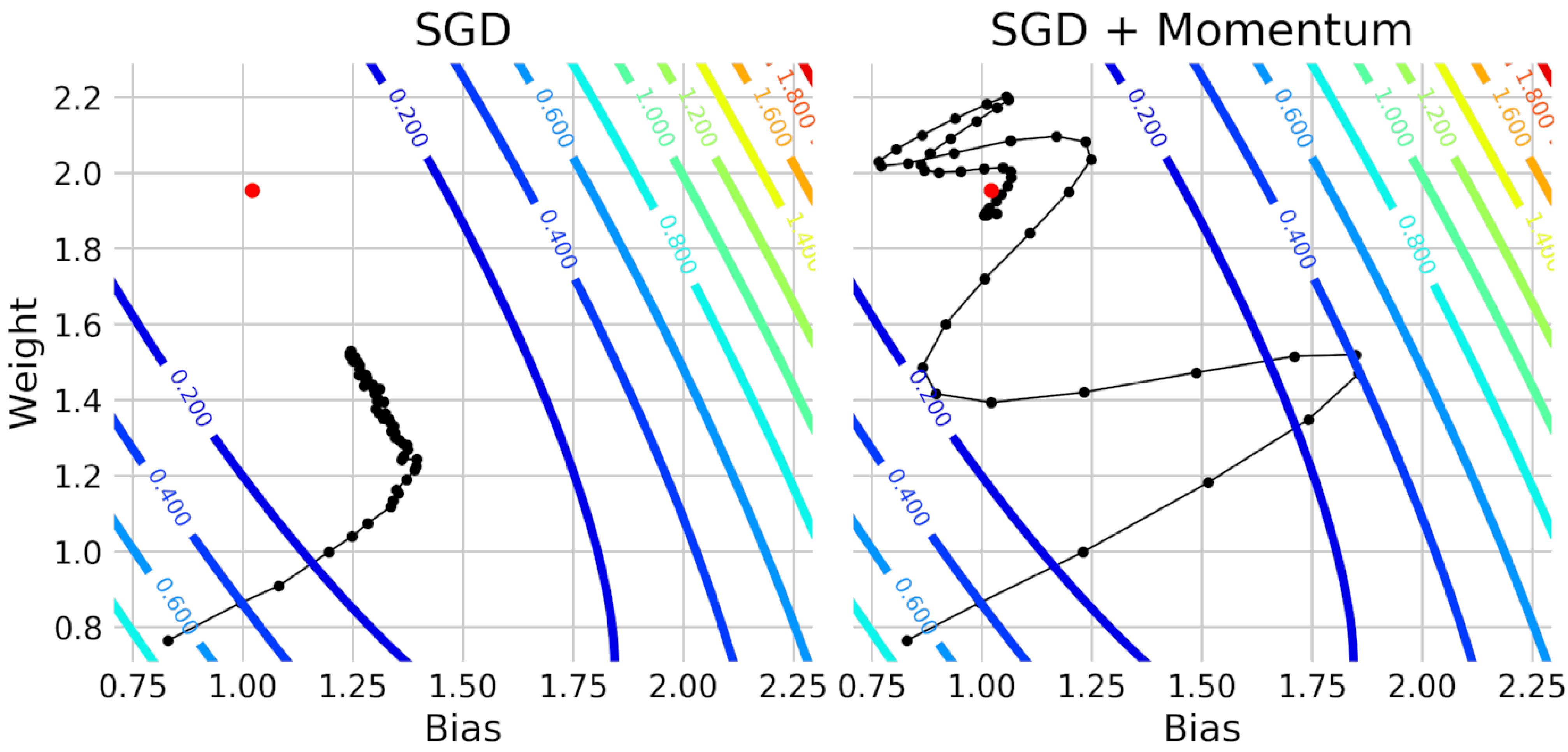
Learning rate

- For a constant gradient

$$\lim_{t \to \infty} v^{(t)} = \frac{g}{1 - \alpha}$$


SGD
GD


SGD+mom
GD


SGD
SGD + Momentum
Weight
Bias
Bias
0.200
0.400
0.600
0.800
1.000
1.200
1.400
1.600
1.800
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.75
1.00
1.25
1.50
1.75
2.00
2.25

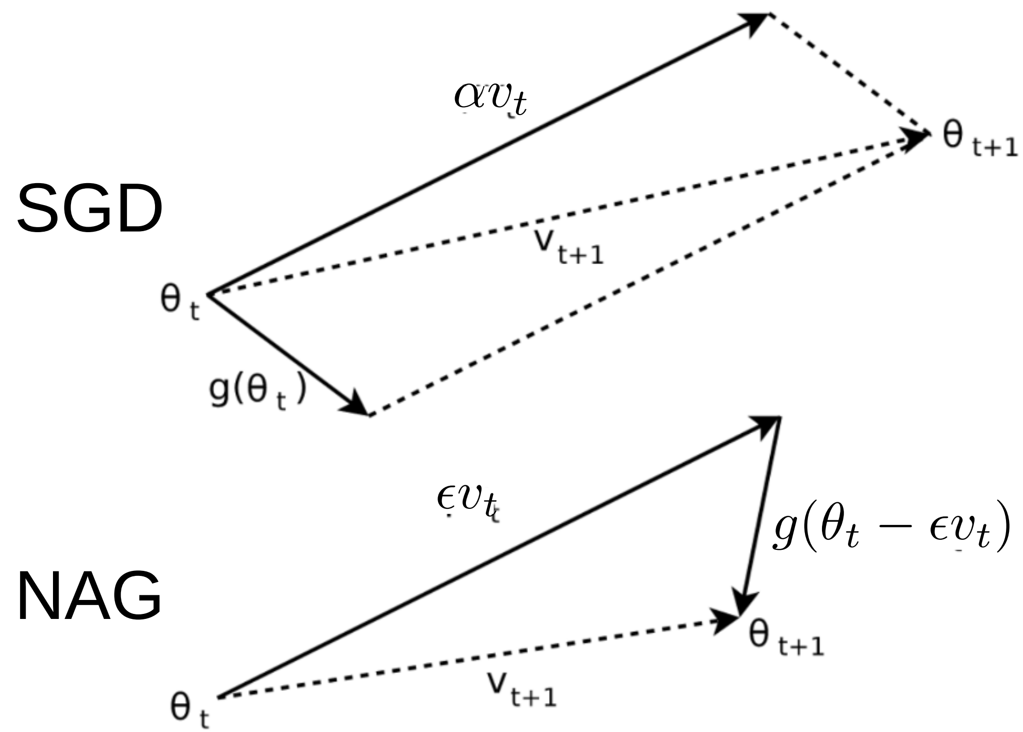
# SGD with Nesterov Momentum (NAG)

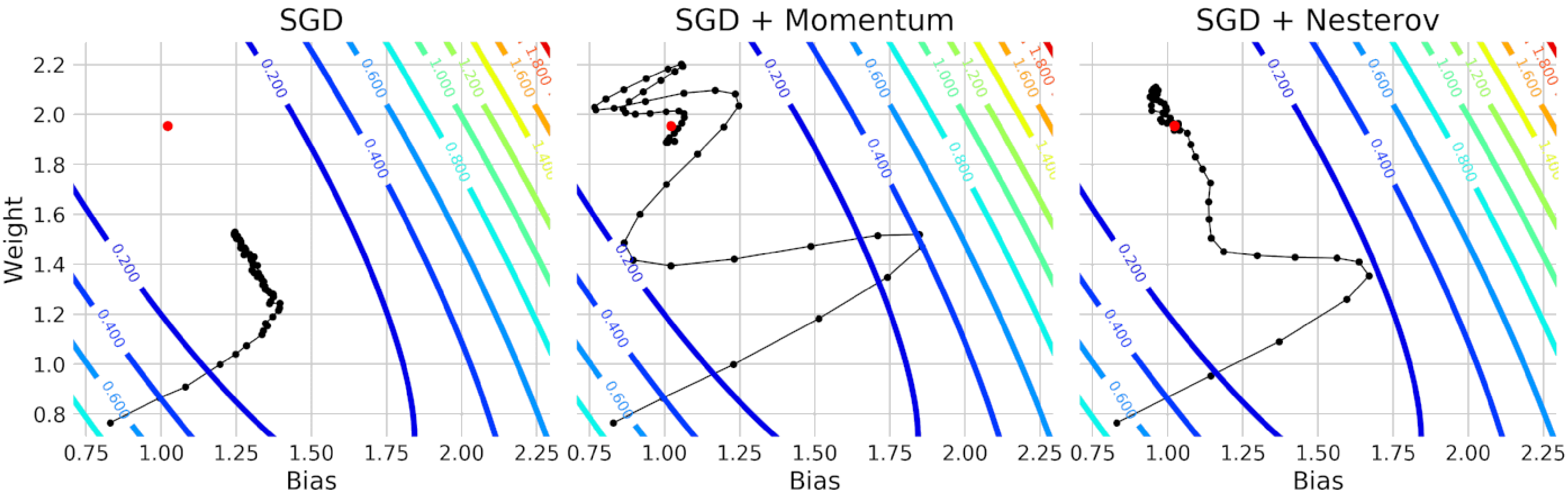
- Calculate gradient in the predicted future position

$$g = \frac{1}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} \nabla_\theta L(f(x^{(i)}, \theta^{(t)} - \epsilon v^{(t)}), y^{(i)})$$

$$v^{(t+1)} = \alpha v^{(t)} + g$$

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon v^{(t+1)}$$





SGD
SGD + Momentum
SGD + Nesterov
Weight
Bias
Bias
Bias
0.200
0.400
0.600
0.800
1.000
1.200
1.400
1.600
1.800

# Adaptive Learning Rate

- Slow progress in gently sloped directions, because

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon_t g$$

- How to adapt LR, so the progress in all directions is approximately the same?

- AdaGrad
- AdaDelta
- RMSProp
- Adam

# RMSProp

$$g = \frac{1}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} \nabla_\theta L(f(x^{(i)}, \theta^{(t)}), y^{(i)})$$

$$r^{(t+1)} = \beta r^{(t)} + (1 - \beta) g^2$$

Exponentially weighted moving average of $g^2$

$$v^{(t+1)} = \alpha v^{(t)} + \frac{1}{\delta + \sqrt{r^{(t+1)}}} g$$

Element-wise operations

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon v^{(t+1)}$$

# Adam

$$g = \frac{1}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} \nabla_\theta L(f(x^{(i)}, \theta^{(t)}), y^{(i)})$$

$$s^{(t+1)} = \beta_1 s^{(t)} + (1 - \beta_1) g$$

$$r^{(t+1)} = \beta_2 r^{(t)} + (1 - \beta_2) g^2$$

Exponentially weighted moving average of g and $g^2$

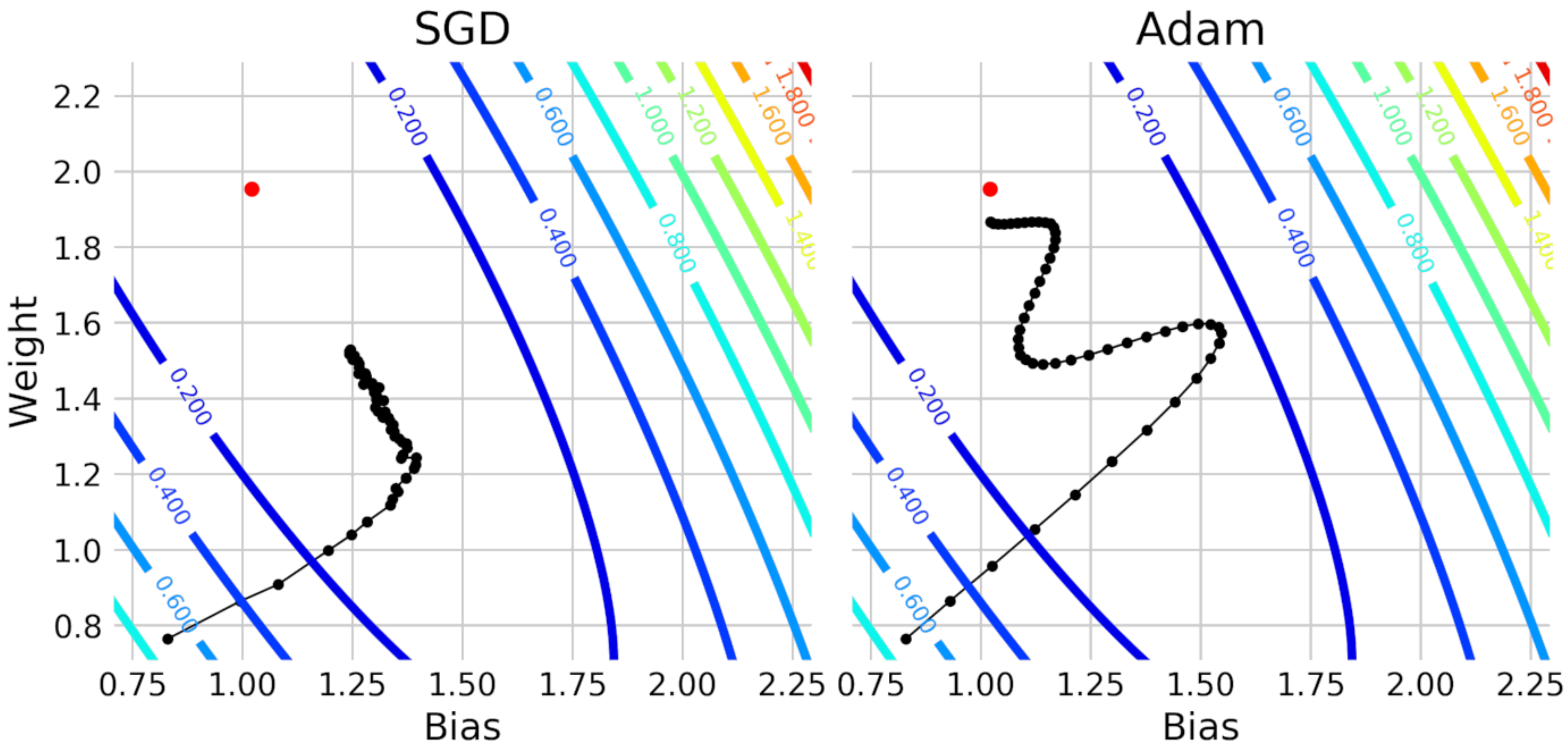
$$\hat{s} = \frac{s}{1 - \beta_1^{t+1}} \qquad \hat{r} = \frac{r}{1 - \beta_2^{t+1}}$$

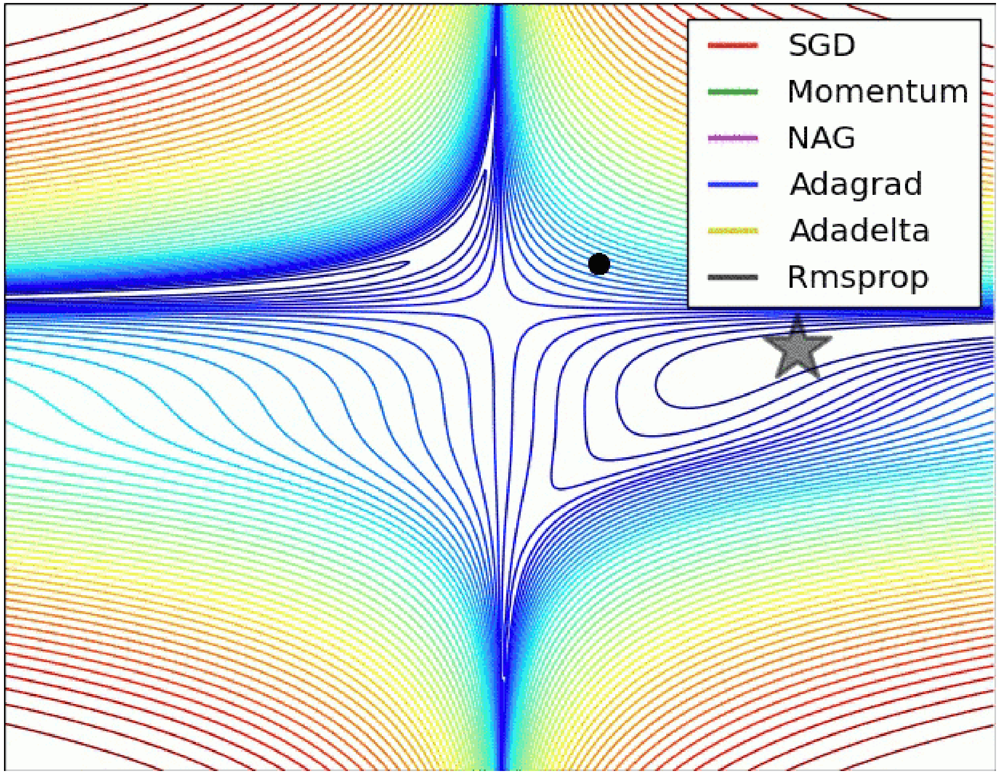
Bias correction for correct initialization

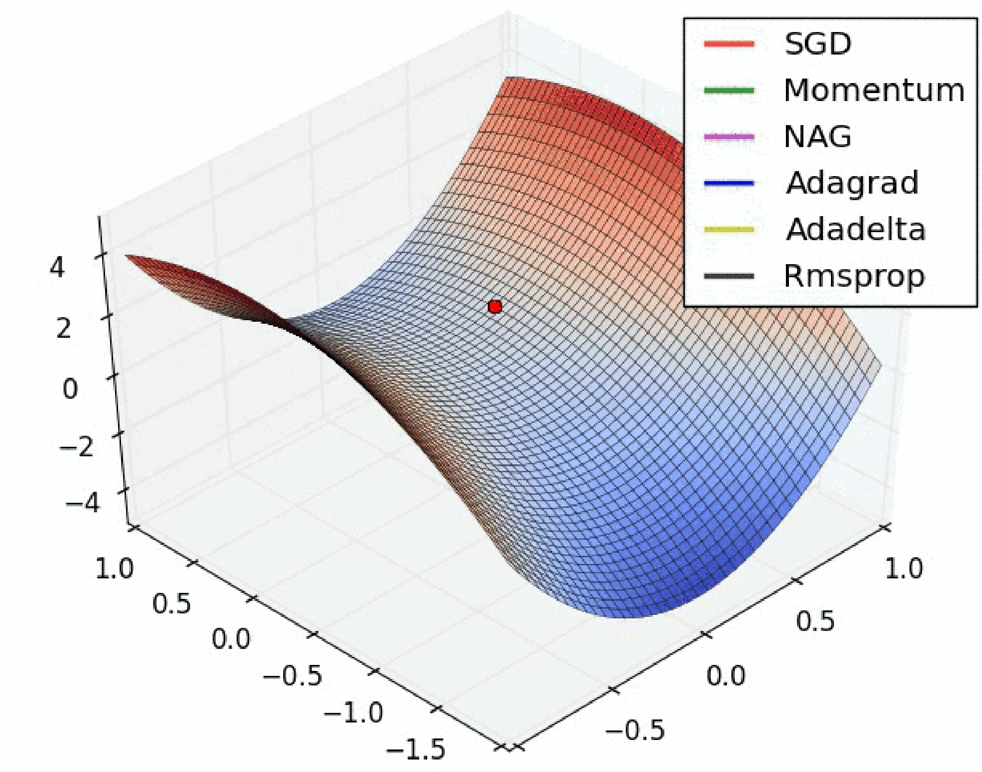
$\hat{s}^{(1)} = g \quad \hat{r}^{(1)} = g^2$

$$\theta^{(t+1)} = \theta^{(t)} - \epsilon \frac{\hat{s}}{\delta + \sqrt{\hat{r}}}$$

Default values: $\beta_1 = 0.9,\ \beta_2 = 0.999,\ \delta = 10^{-8}$


SGD
Adam
Weight
Bias
Bias
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.75 1.00 1.25 1.50 1.75 2.00 2.25
0.75 1.00 1.25 1.50 1.75 2.00 2.25
0.200
0.400
0.600
0.800
1.000
1.200
1.400
1.600
1.800

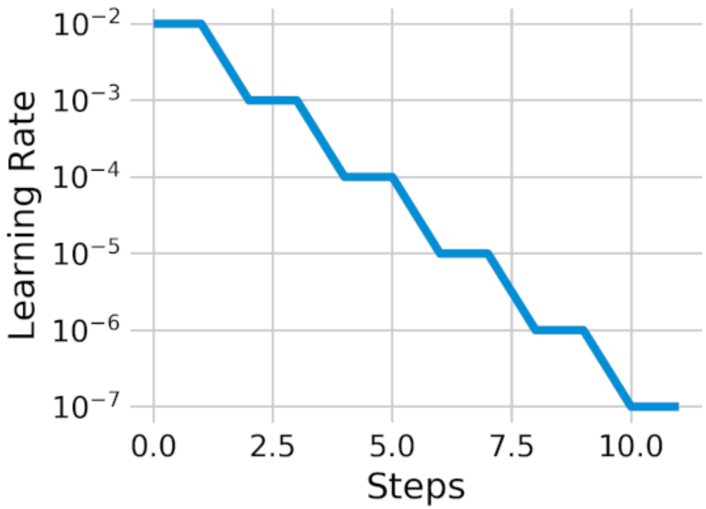


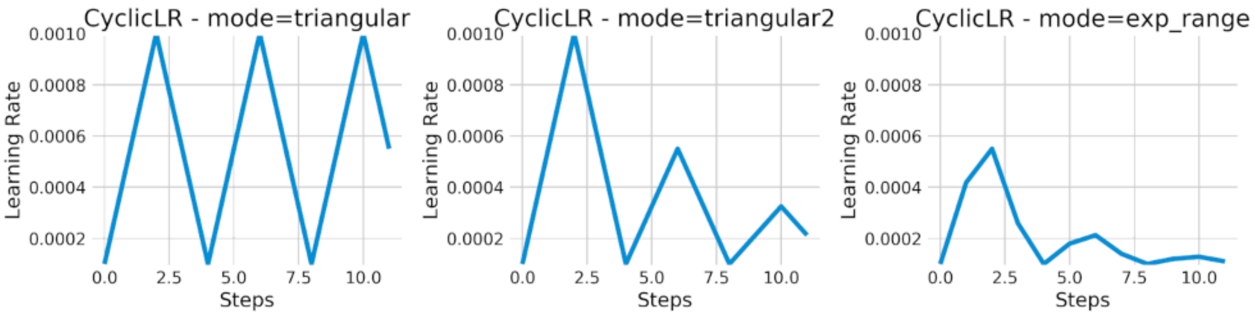




https://imgur.com/a/Hqolp

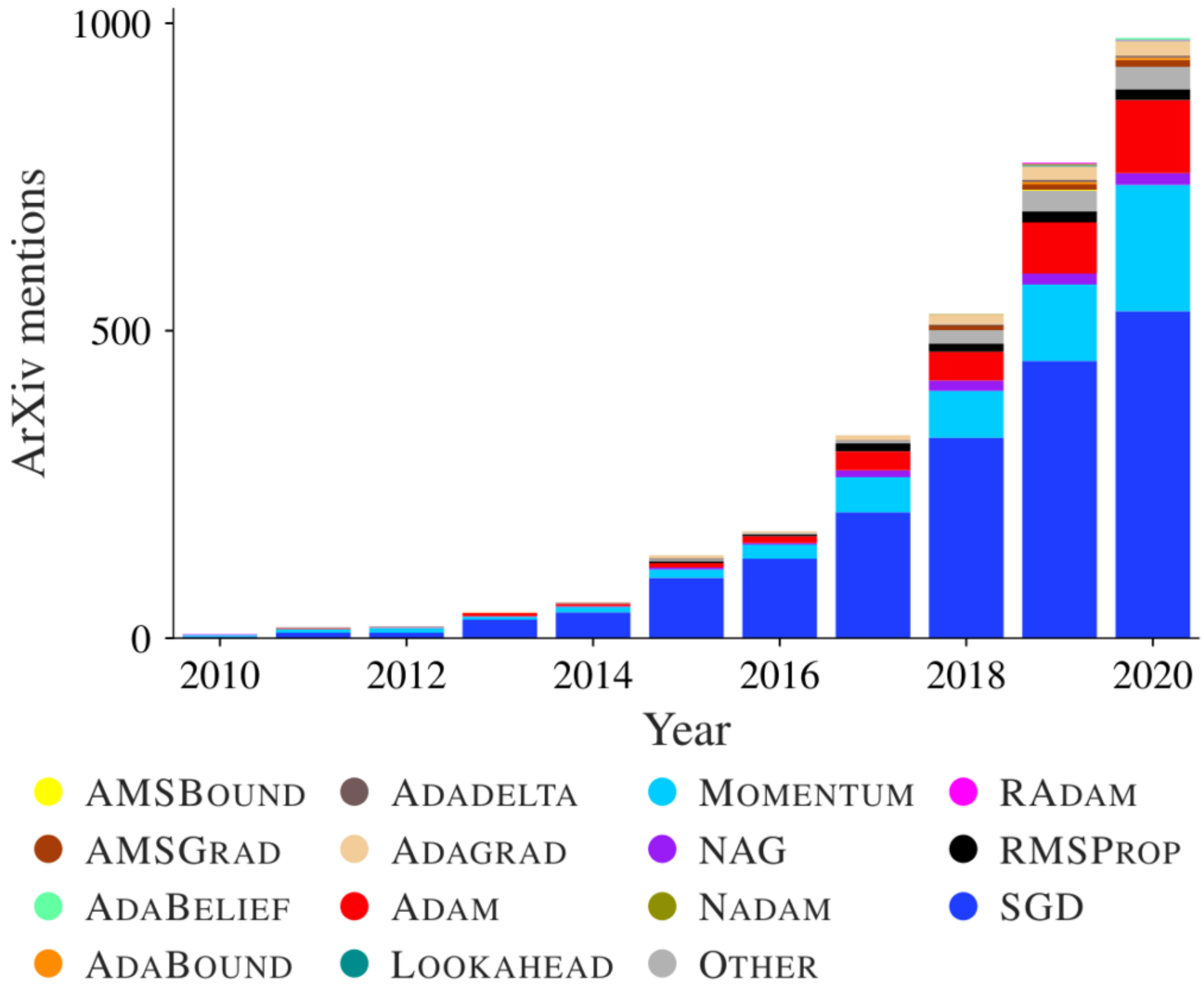
# Learning Rate Scheduler

- Linear decay
- Reduce on Plateau
- Cyclic LR
- ...



# Solver Popularity




Schmidt et al.,"Benchmarking Deep Learning Optimizers", ICML 2021.

# Solver takeaway

- No clear winner
- Adam (RMSprop, NAG) remains a viable choice for many problems

- Instead of trying a new solver, re-tuning and re-running your favorite one seems to be the best choice.

Schmidt et al.,"Benchmarking Deep Learning Optimizers", ICML 2021.

# Vanishing&Exploding Gradients

- Common in very deep nets

$$J(\theta) = z^{(P)}(\ldots z^{(p)}(z^{(2)}(z^{(1)}(x)))\ldots)$$

$$z^{(p)}(x) = Wx \quad p = 1, \ldots, P$$

$$J(\theta) = WW\ldots Wx = W^P x = V\Lambda^P V^{-1} x$$

$$\lim_{p\to\infty} \lambda_i^p$$

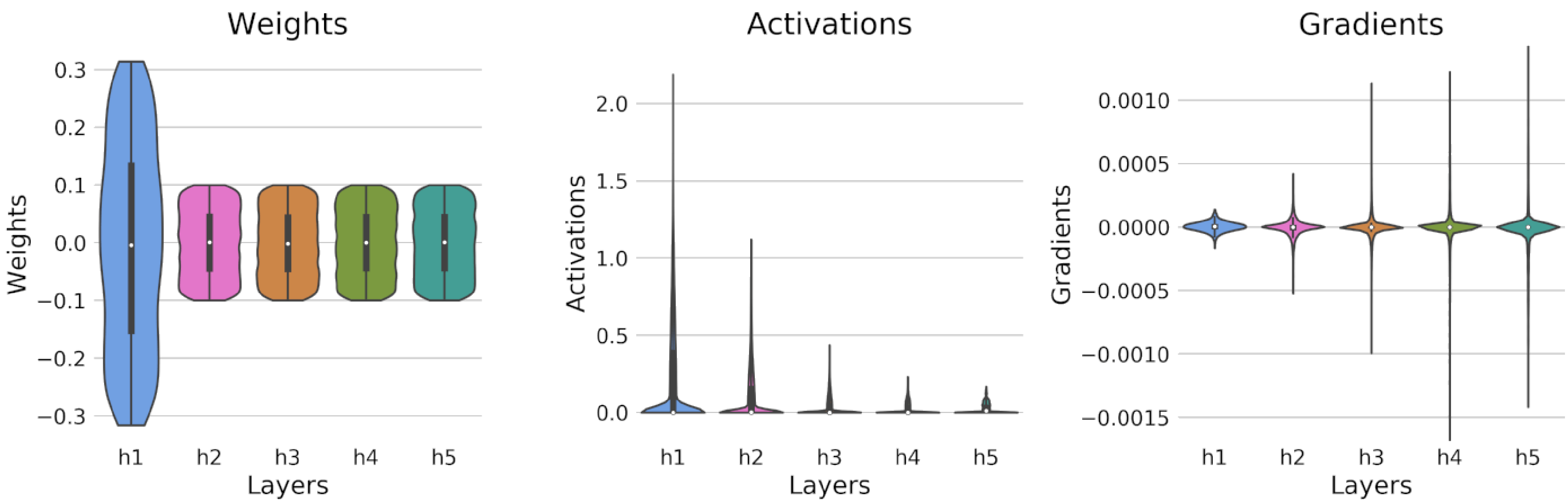
Eigenvalues<1 go to zero

Eigenvalues>1 go to infinity

- Proper initialization of weights
- Gradient clipping
- Batch normalization

# Initialization

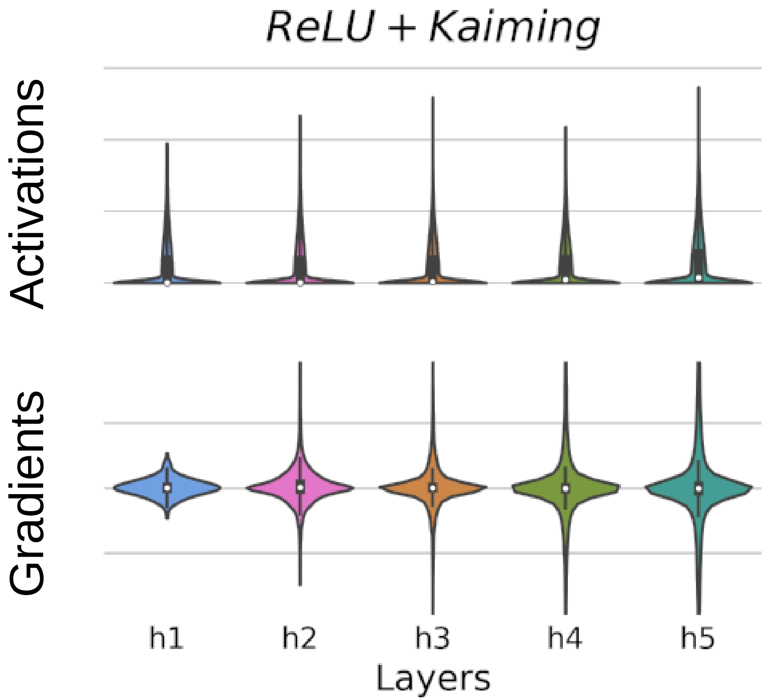
- Main principle: “break symmetry” - - > random initialization




- But we should also adapt the variance of random initialization

# Initialization

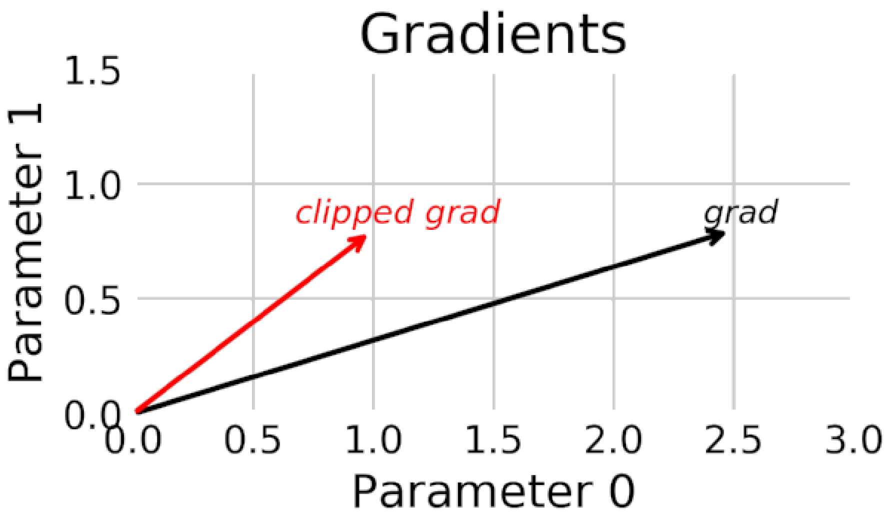
- *Kaiming* initilization for layers with ReLU activation



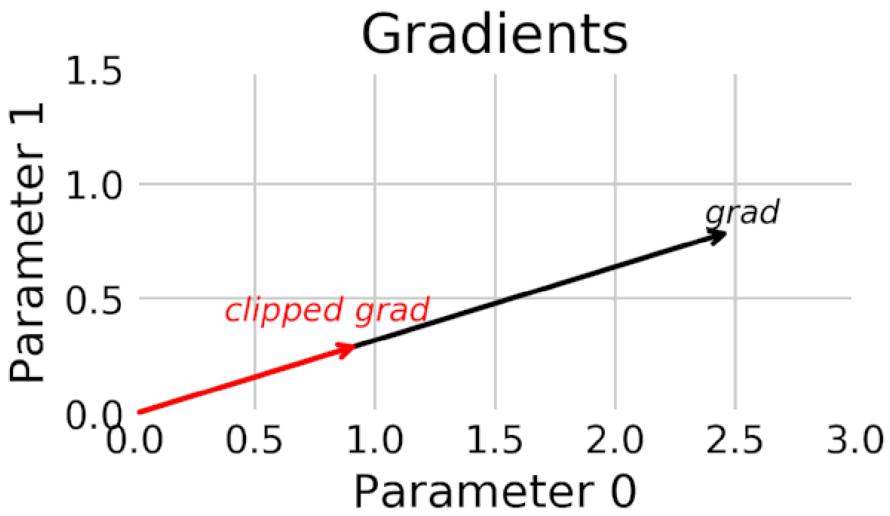

- *Glorot initialization* for layers with tanh activation

# Gradient Clipping

- Good for exploding gradients

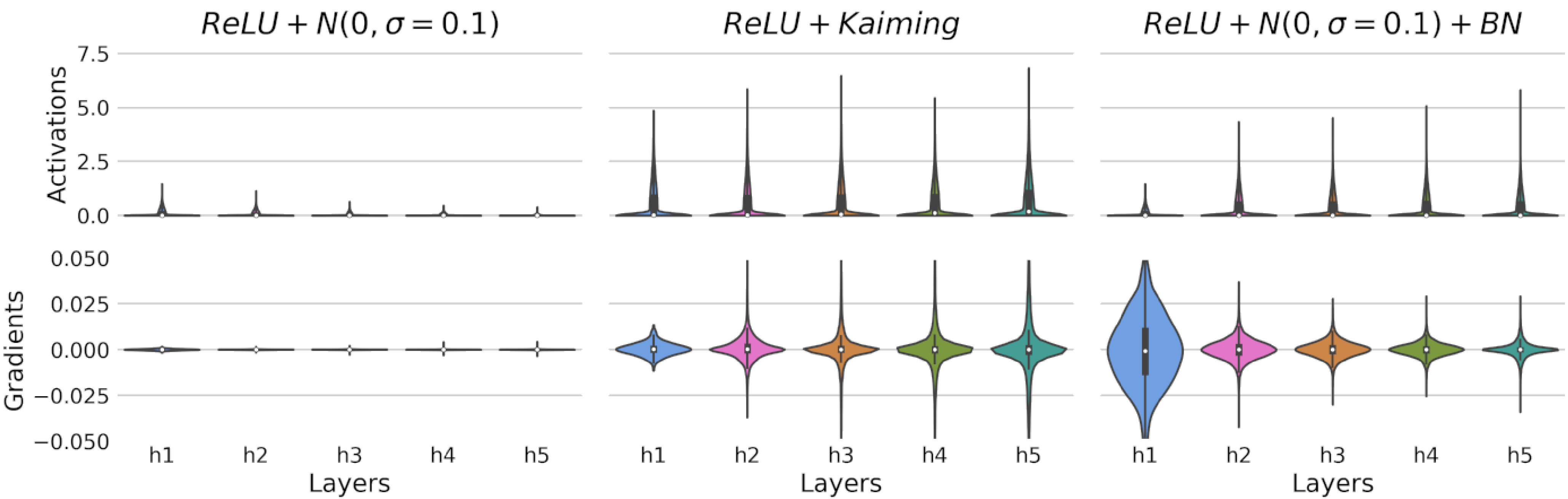



Value clipping




Norm clipping

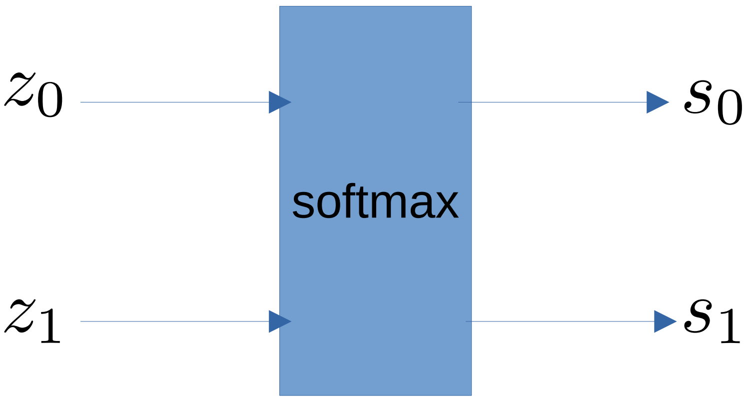
# Batch Normalization



- Extra Slides….

# Derivation of softmax

- Consider two-class problem
- z_0, z_1: network output (not probabilities)




$$s_0 + s_1 = 1$$

- S_0 ... probability of class 0
- S_1 ... probability of class 1

- GT labels: $\{t^i\}_{i=1}^{N} \quad \text{e.g.} \quad = \{0, 1, 1, \ldots\}$

$$(z_0^i, z_1^i) \quad i = 1, \ldots, N$$

# Derivation of softmax

- Maximize entropy $$s_0^i \log s_0^i + s_1^i \log s_1^i$$
- Subject to:

$$s_0^i, s_1^i \geq 0$$

$$s_0^i + s_1^i = 1$$

$$\sum_i s_0^i z_0^i = \sum_{\{i|t^i=0\}} z_0^i \equiv N\mu_0$$

$$\sum_i s_1^i z_1^i = \sum_{\{i|t^i=1\}} z_1^i \equiv N\mu_1$$

$$L = \sum_i s_0^i \log s_0^i + s_1^i \log s_1^i +$$
$$\lambda_0(\mu_0 - s_0^i z_0^i) + \lambda_1(\mu_1 - s_1^i z_1^i) +$$
$$\lambda(s_0^i + s_1^i - 1)$$

# Derivation of softmax

$$L = \sum_i s_0^i \log s_0^i + s_1^i \log s_1^i +$$
$$\lambda_0(\mu_0 - s_0^i z_0^i) + \lambda_1(\mu_1 - s_1^i z_1^i) +$$
$$\lambda(s_0^i + s_1^i - 1)$$

$$\frac{\partial L}{\partial s_0} = \log s_0 + 1 - \lambda_0 z_0 + \lambda = 0$$

$$\frac{\partial L}{\partial s_1} = \log s_1 + 1 - \lambda_1 z_1 + \lambda = 0$$

$$s_0 = e^{\lambda_0 z_0 - \lambda - 1}$$

$$s_1 = e^{\lambda_1 z_1 - \lambda - 1}$$

# Derivation of softmax

$$s_0 = e^{\lambda_0 z_0 - \lambda - 1}$$

$$s_1 = e^{\lambda_1 z_1 - \lambda - 1}$$

$$s_0 + s_1 = 1$$

$$e^{-\lambda - 1} = \frac{1}{e^{\lambda_0 z_0} + e^{\lambda_1 z_1}}$$

$$s_0 = \frac{e^{\lambda_0 z_0}}{e^{\lambda_0 z_0} + e^{\lambda_1 z_1}}$$

$$s_1 = \frac{e^{\lambda_1 z_1}}{e^{\lambda_0 z_0} + e^{\lambda_1 z_1}}$$

$$s_i = \mathrm{softmax}(\mathbf{z})_i$$

# Derivation of sigmoid

- One-class problem: Bernoulli distribution

$$z_1 = 0$$

$$s_0 = \frac{e^{\lambda_0 z_0}}{e^{\lambda_0 z_0} + 1} = \frac{1}{1 + e^{-\lambda_0 z_0}} = \sigma(z_0)$$

# Example – Gaussian Model

- Training data distribution: $p(x,y) = p(y|x)p(x)$
- Estimator of y: $f(x|\theta)$
- Model distribution: $q(y|x) \equiv N(y|f(x;\theta), \sigma^2)$

$$\hat{f}(x) = \arg\min_{f(x)} \mathbb{E}_{p(x,y)}[-\log q(y|f(x))]$$

$$L = \int \left[ \int (y - f(x))^2 p(y|x) dy \right] p(x) dx + c$$

$$\frac{\partial L}{\partial f} \propto \int (y - f(x)) p(y|x) dy = 0$$

$$\hat{f}(x) = \int y p(y|x) dy = \mathbb{E}_{p(y|x)}[y]$$

# Linear Regression

- Training data: $p(x,y) \to \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$
- Linear estimator of y: $f(x) = wx + b \qquad \theta = \{w, b\}$
- Model distribution: $q(y|x) \equiv N(y|f(x), \sigma^2)$

$$\hat{f}(x) = \arg\min_{f(x)} \mathbb{E}_{p(x,y)}[-\log q(y|f(x))]$$

$$\hat{f}(x) = \arg\min_{f} \frac{1}{2\sigma^2} \sum_i (y^{(i)} - f(x^{(i)}))^2 + \mathrm{c}$$

$$\arg\min_{w,b} \frac{1}{2\sigma^2} \sum_i (y^{(i)} - wx^{(i)} - b)^2 + \mathrm{c}$$

- Closed-form solution

# Example – Laplace Model

- Training data distribution: $p(x,y) = p(y|x)p(x)$
- Estimator of y: f(x)
- Model distribution: $q(y|x) \equiv \mathrm{Laplace}(y|f(x), \gamma)$

$$\hat{f}(x) = \arg\min_{f(x)} \mathbb{E}_{p(x,y)}[-\log q(y|f(x))]$$

$$L = \int \left[ \int |y - f(x)|\, p(y|x) dy \right] p(x) dx + c$$

$$\frac{\partial L}{\partial f} \propto \int \mathrm{sign}(y - f(x)) p(y|x) dy = 0$$

$\hat{f}(x) = m$ … median: $P(y \leq m|x) = P(y \geq m|x)$

# Example – Bernoulli Model

- Training data distribution: $p(x,y) = p(y|x)p(x)\,, \quad y \in \{0,1\}$
- Estimator of y: f(x)
- Model distribution: $q(y|x) \equiv \mathrm{Bernoulli}(y|f(x))$

$$\hat{f}(x) = \arg\min_{f(x)} \mathbb{E}_{p(x,y)} \boxed{[-\log q(y|f(x))]}$$

$$L = \int \sum_{y=0}^{1} \left( \boxed{[y \log f(x) + (1-y)\log(1-f(x))]}\, p(y|x) \right) p(x)\,dx$$

$$\frac{\partial L}{\partial f} = p(x) \sum_{y=0}^{1} \left( y \frac{1}{f(x)} - (1-y) \frac{1}{1-f(x)} \right) p(y|x)$$

$$= p(x) \left( p(y=1|x) \frac{1}{f(x)} - (1 - p(y=1|x)) \frac{1}{1-f(x)} \right)$$

$$= 0$$

# Example – Bernoulli Model

$$p(x)\left(p(y=1|x)\frac{1}{f(x)}-(1-p(y=1|x))\frac{1}{1-f(x)}\right)=0$$

$$\hat{f}(x)=p(y=1|x)=\mathbb{E}_{p(y|x)}[y]$$

$$\mathbb{E}_{p(x)}[\hat{f}(x)]=\int\hat{f}(x)p(x)dx$$

$$=\int p(y=1|x)p(x)dx=p(y=1)$$

… Bernoulli mean

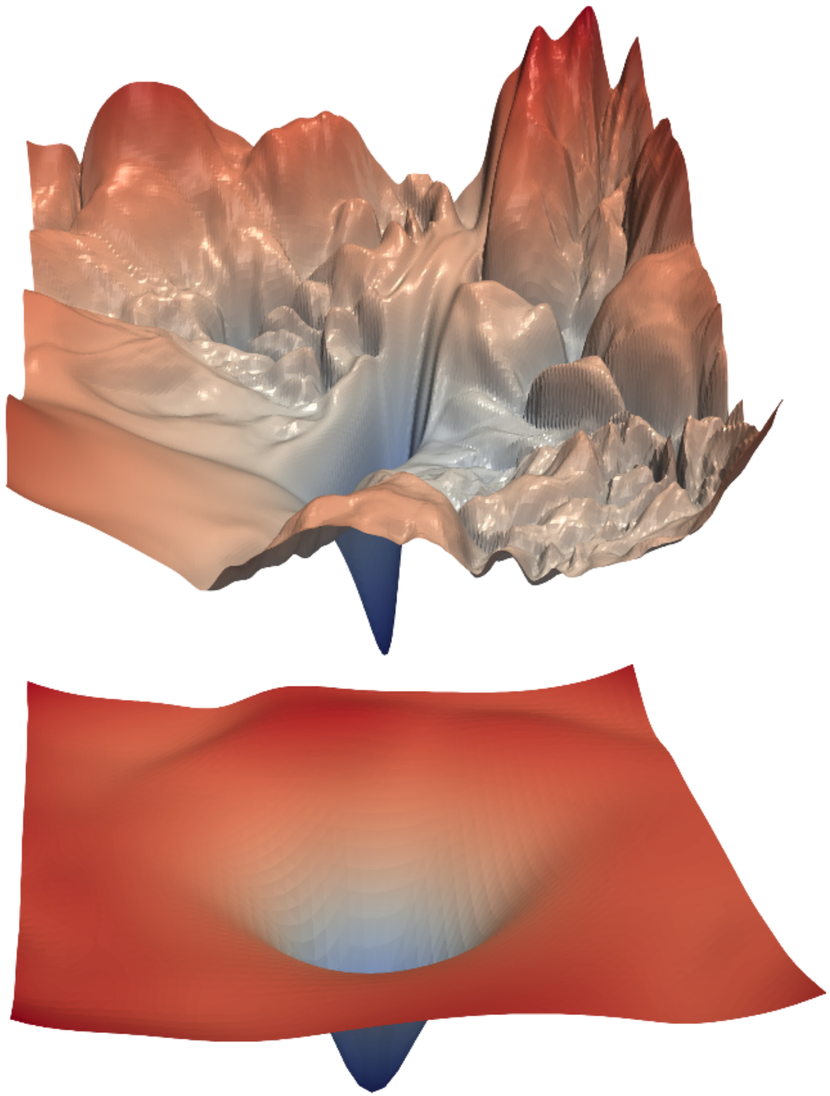
# Strojové učení II

## Regularization

Institute of Information Theory
and Automation of the AS CR

# Regularization

- Reduce overfitting, improve generalization
- Loss terms
  - Parameter Regularization
- Dataset
  - Augmentation
  - Label Smoothing
- Architecture
  - Parameter Tying
  - Bagging
  - Dropout
  - Batch Normalization
  - Residual Connections



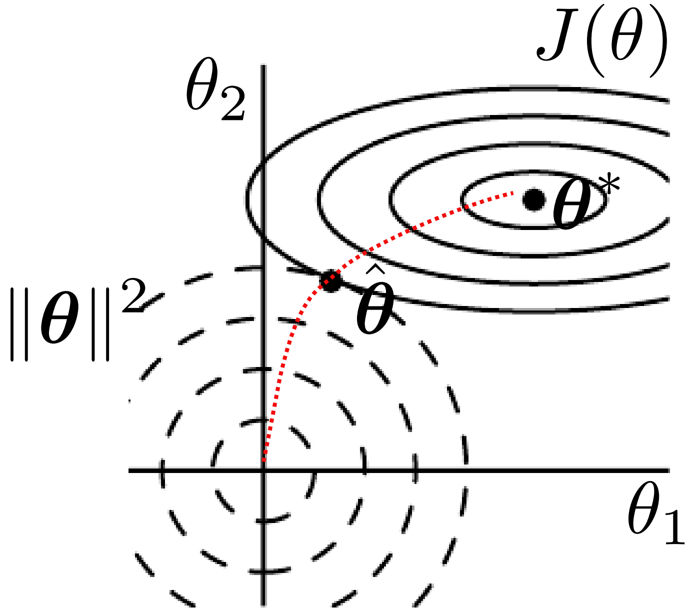
Li et al., NIPS 2018

# Parameter Regularization

- Original loss function $J(\theta)$

  with regularization $\tilde{J}(\theta) = J(\theta) + \lambda \|\theta\|^2$

- L2 norm of parameters is called "Weight Decay"



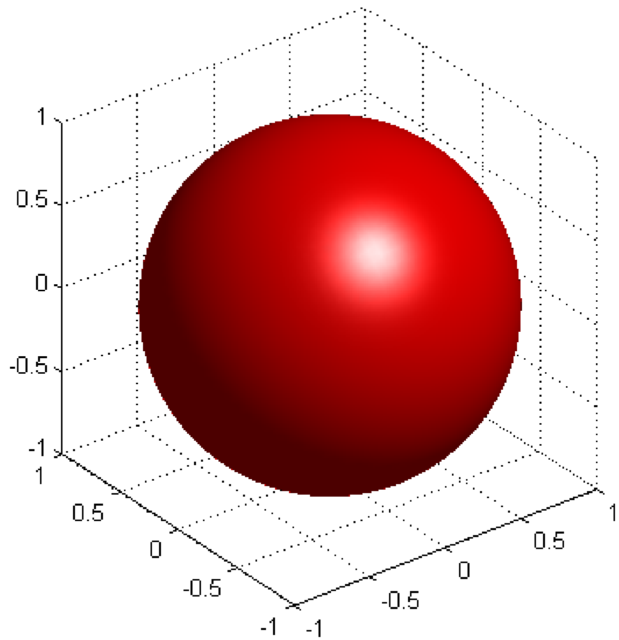

Position on the curve depends on λ

# Unit Ball

$$\tilde{J}(\theta) = J(\theta) + \lambda\|\theta\|_2^2$$

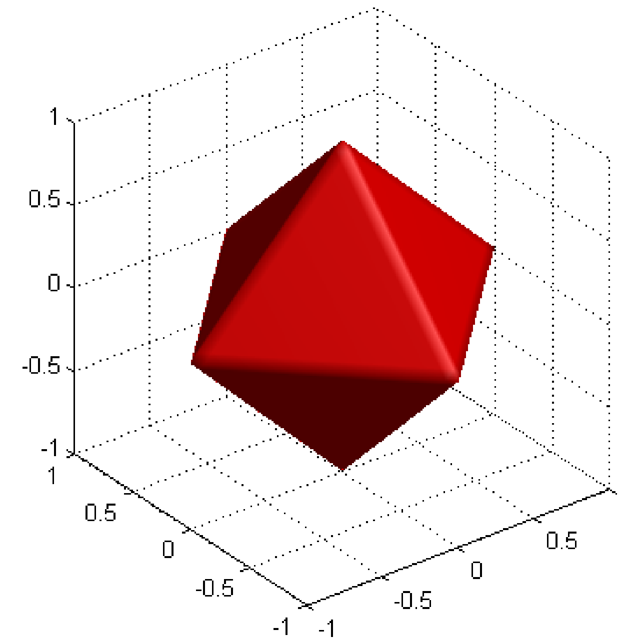
Sometimes is better to use explicit constraints

$$\min J(\theta) \quad \text{s.t.} \quad \|\theta\|_2^2 \leq \epsilon$$




$$\|\theta\|_2^2 = \sum_i |\theta_i|^2$$

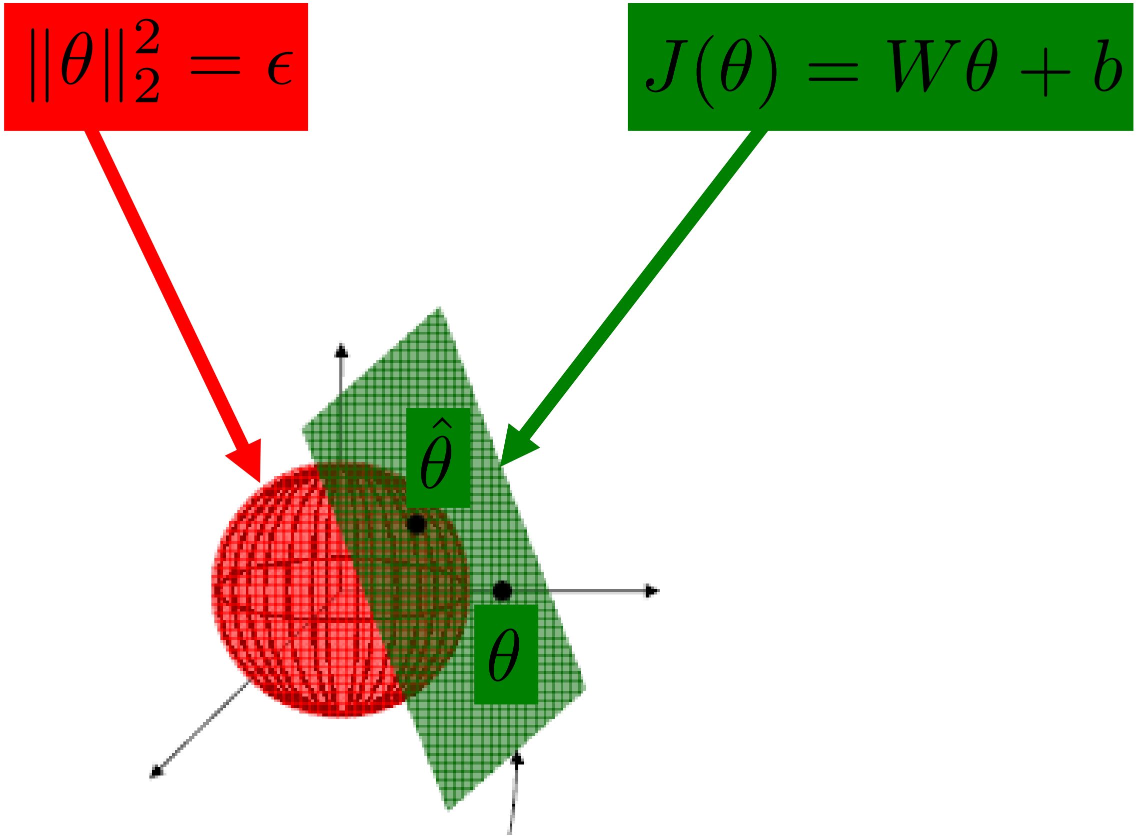
L$_2$ norm



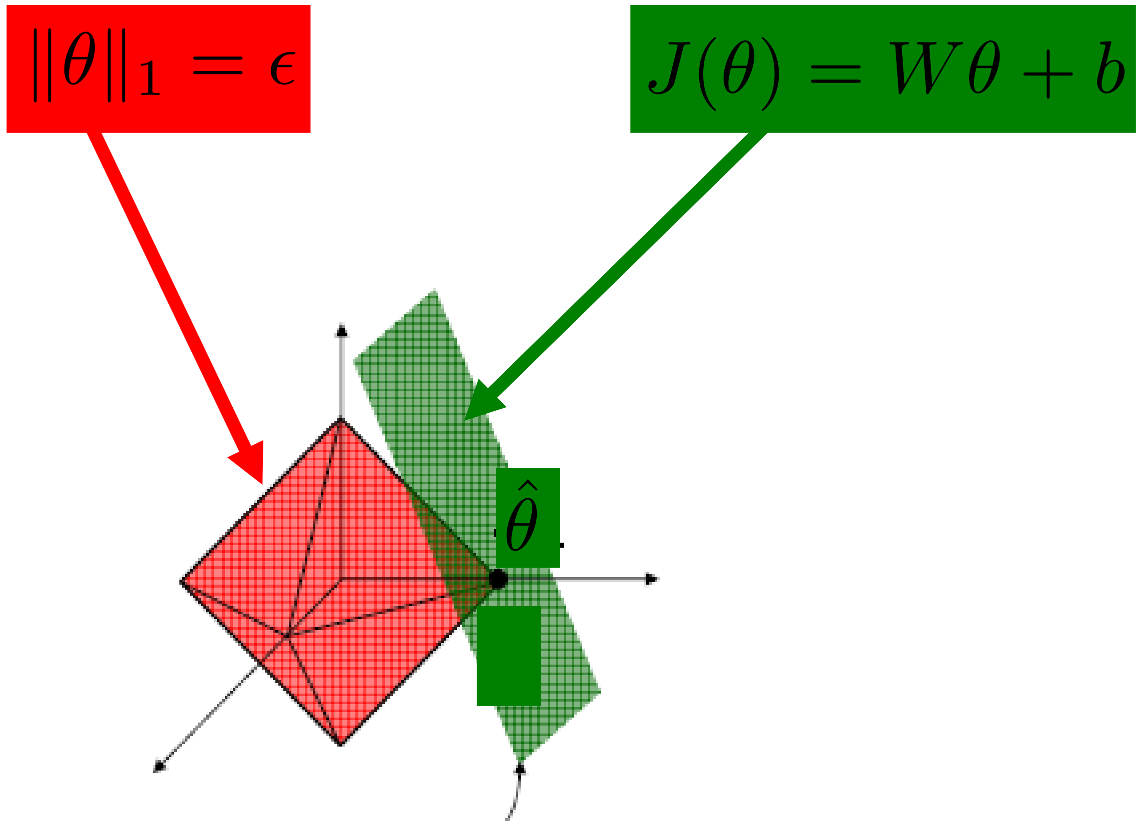

$$\|\theta\|_1 = \sum_i |\theta_i|$$

L$_1$ norm

# L$_2$-norm

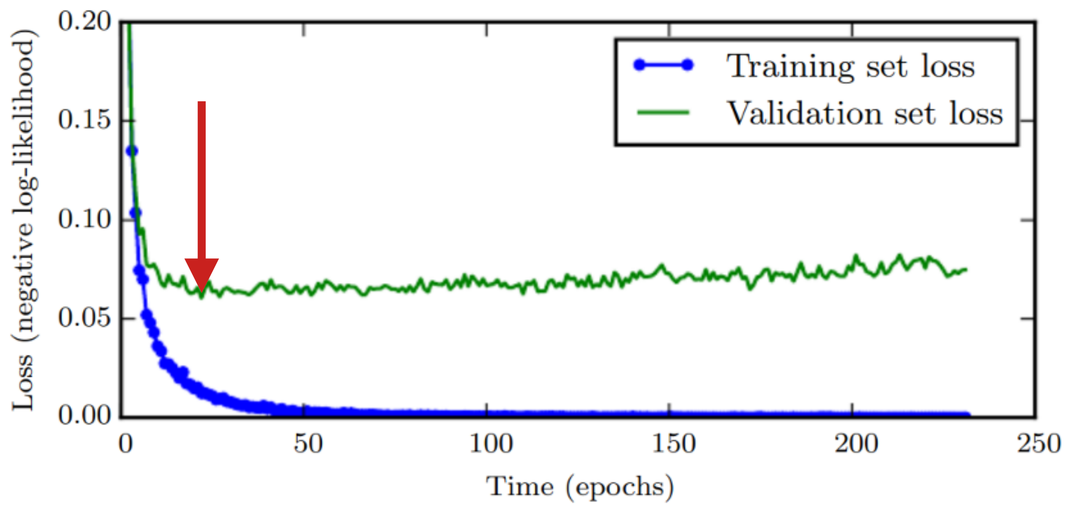


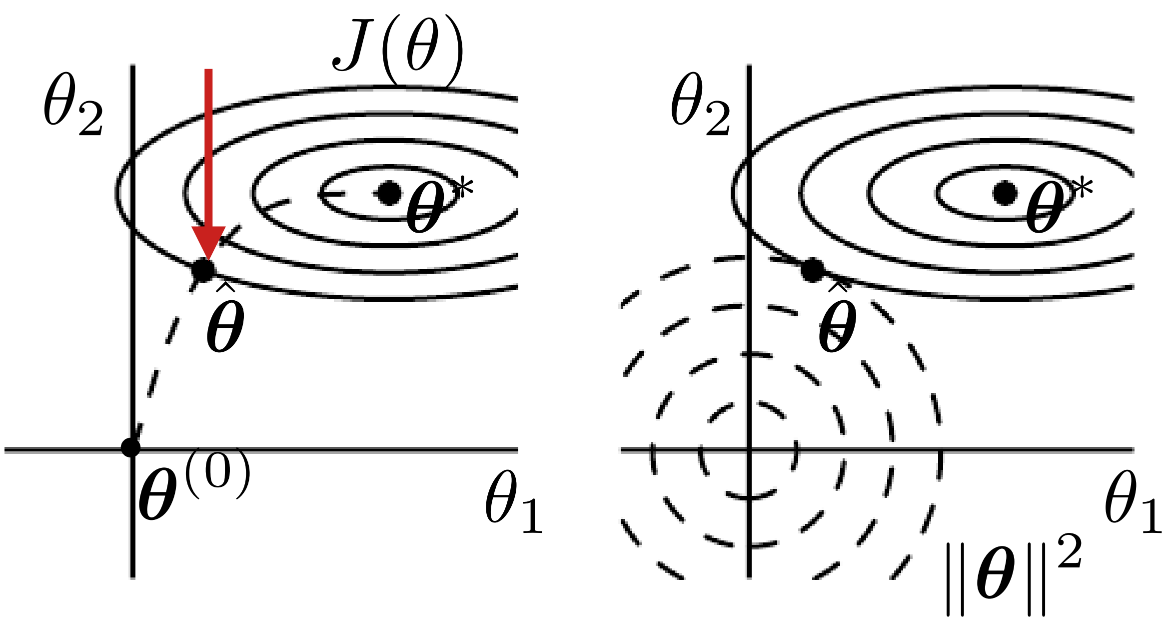
# $L_1$-norm




- Sparse solution

# Early Stopping




Early stopping

L2-norm regularization

# Dataset Augmentation

- Apply random transformation T to inputs
- T = {cropping, rotation, scaling,…}
- This way we learn a representation invariant to T
- Particularly effective for object recognition
- Other transformations: blur, noise, …

# Noise Robustness

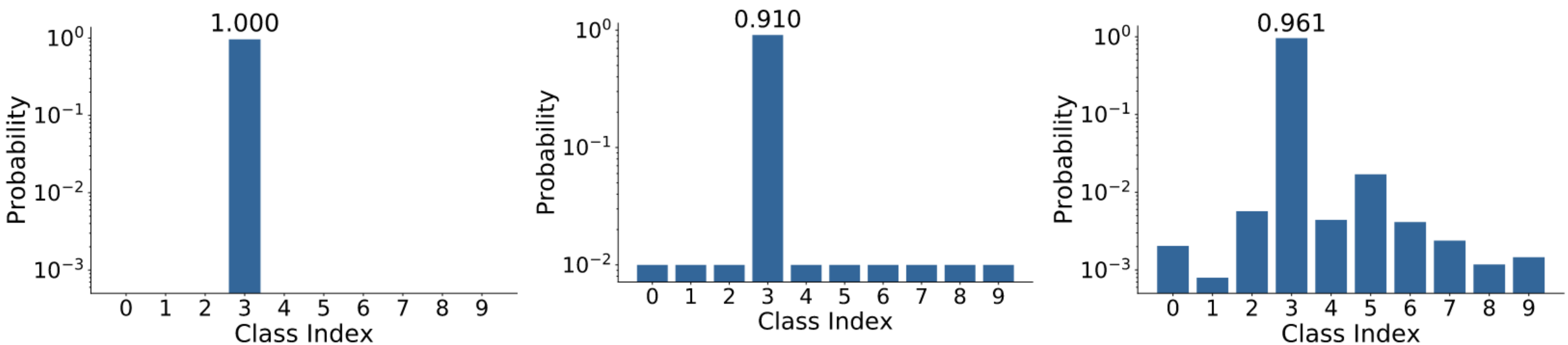
- Injecting noise in
  - Inputs
  - Outputs (labels)
  - Parameters

# Noise in Inputs

- Same principle as data augmentation
- Increases invariance (robustness) to noise

# Noise in Outputs

- Label smoothing




(a) Hard Label (b) LS (c) Ours

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 – *airplane* | 1 – *automobile* | 2 – *bird* | 3 – *cat* | 4 – *deer* | 5 – *dog* | 6 – *frog* | 7 – *horse* | 8 – *ship* | 9 – *truck* |

Zhang et al., Delving deep into label smoothing, TIP 2021

# Noise in Parameters

- Adding noise to weights during learning

  training data: $(x, y) \sim \hat{p}(x, y)$

  loss: $J(\theta) = \mathbb{E}_{p(x,y)}[f(x, \theta) - y]$

  with noise: $\tilde{J}(\theta) = \mathbb{E}_{p(x,y)}[f(x, \theta + n) - y] \quad n \sim N(0, \eta\mathbf{I})$

- Similar to a regularized problem:

$$\tilde{J}(\theta) \sim \mathbb{E}_{p(x,y)}[(f(x, \theta) - y) + \eta\|\nabla_\theta f(x, \theta)\|^2]$$

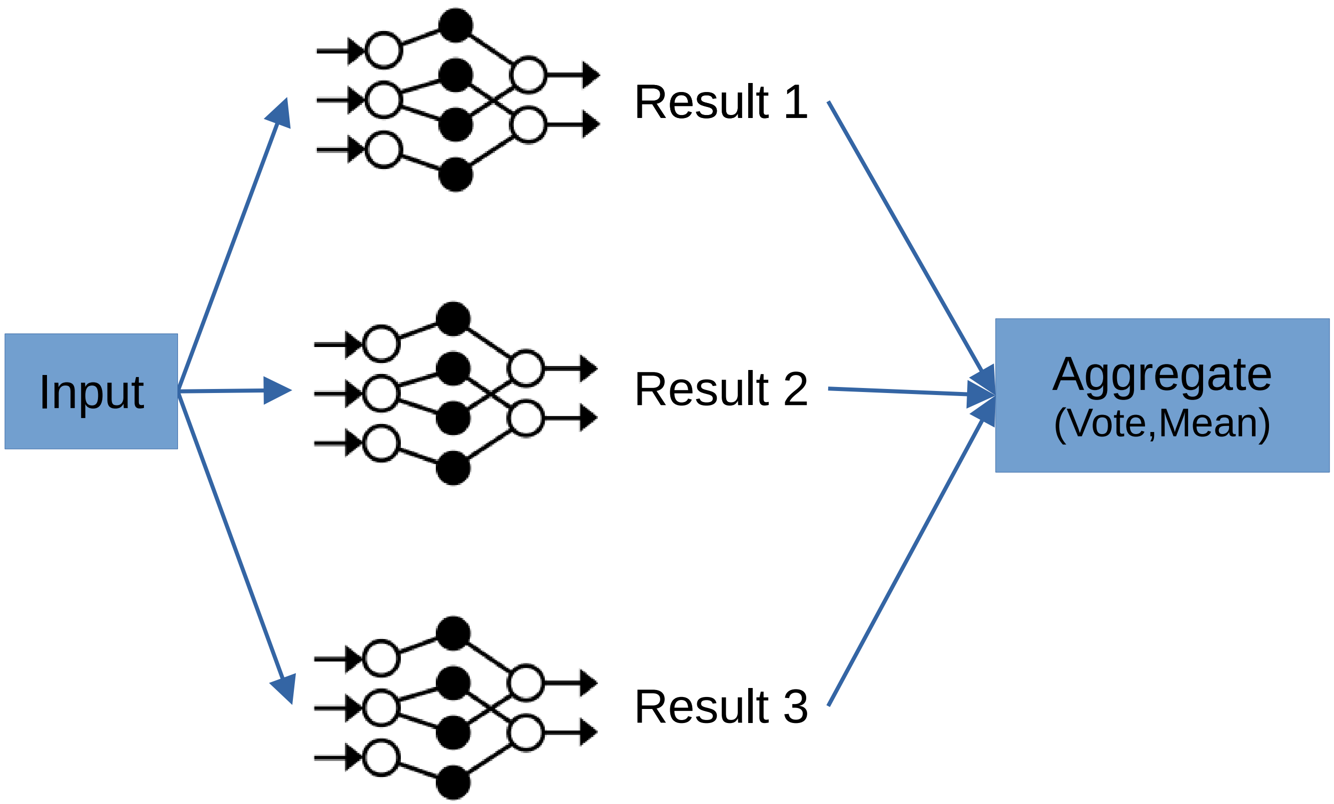
# Parameter Tying

- Decrease the number of parameters
  - Share parameters across layers
  - Share parameters across tasks
  - By assuming structured W in linear units (e.g. convolution)

$$y = Wx + b$$

# Bagging

- Bootstrap aggregation

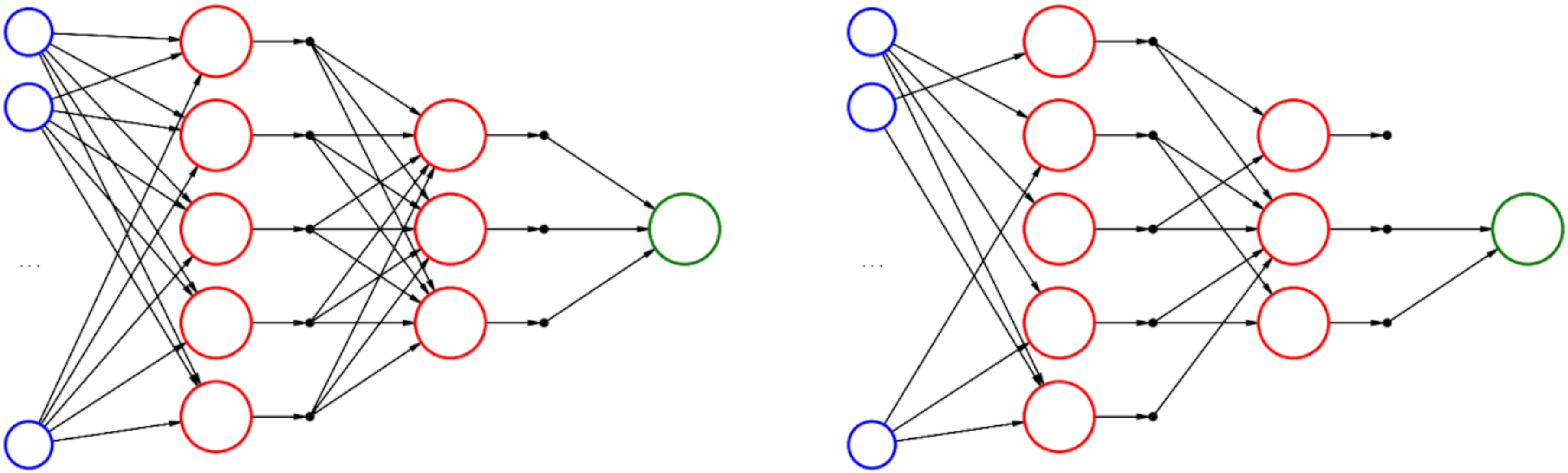



- Uncorrelated results: squared error decreases linearly

Can you prove it?

# Dropout

- Training: randomly remove connections with the probability p and multiply layer outputs by 1/p
- Evaluation: keep all connections



Regular Connections                Applying Dropout

p = 0.5

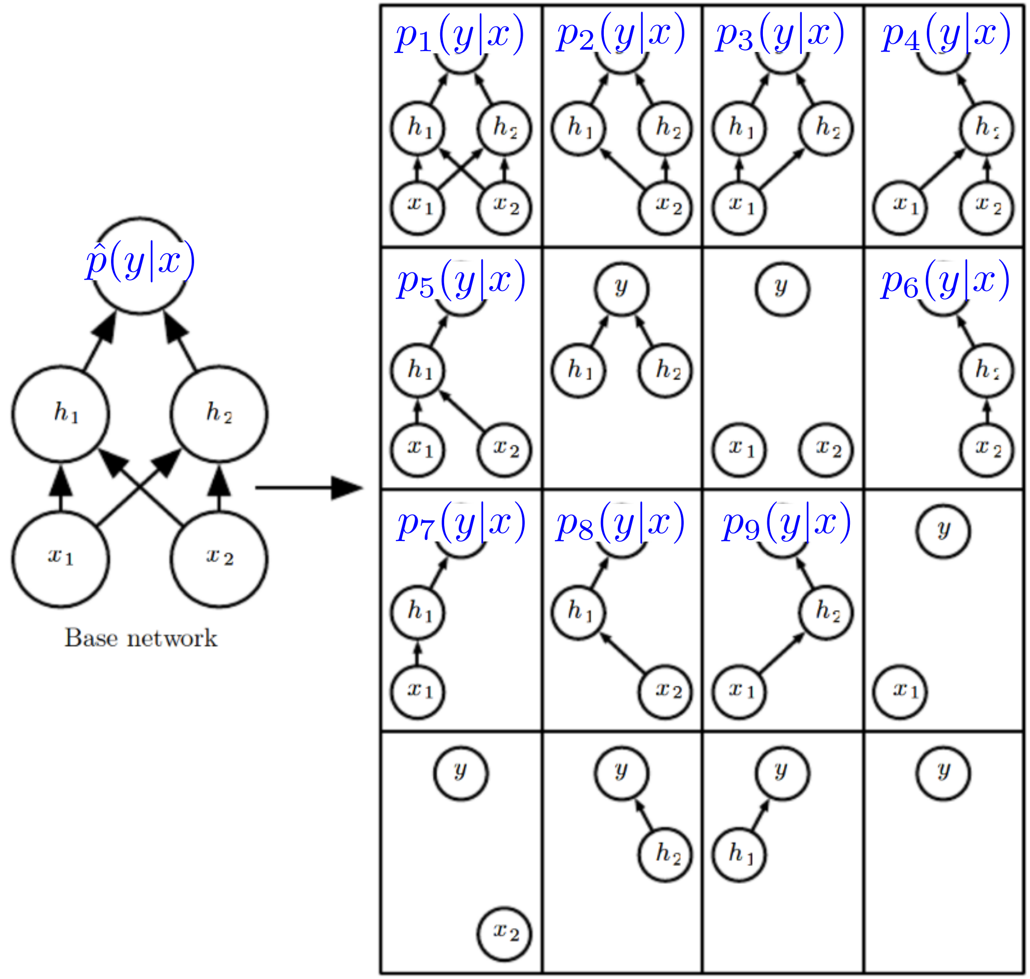
- In 2D (images) dropout removes channels and not pixels!

Why channels and not pixels?

# Dropout

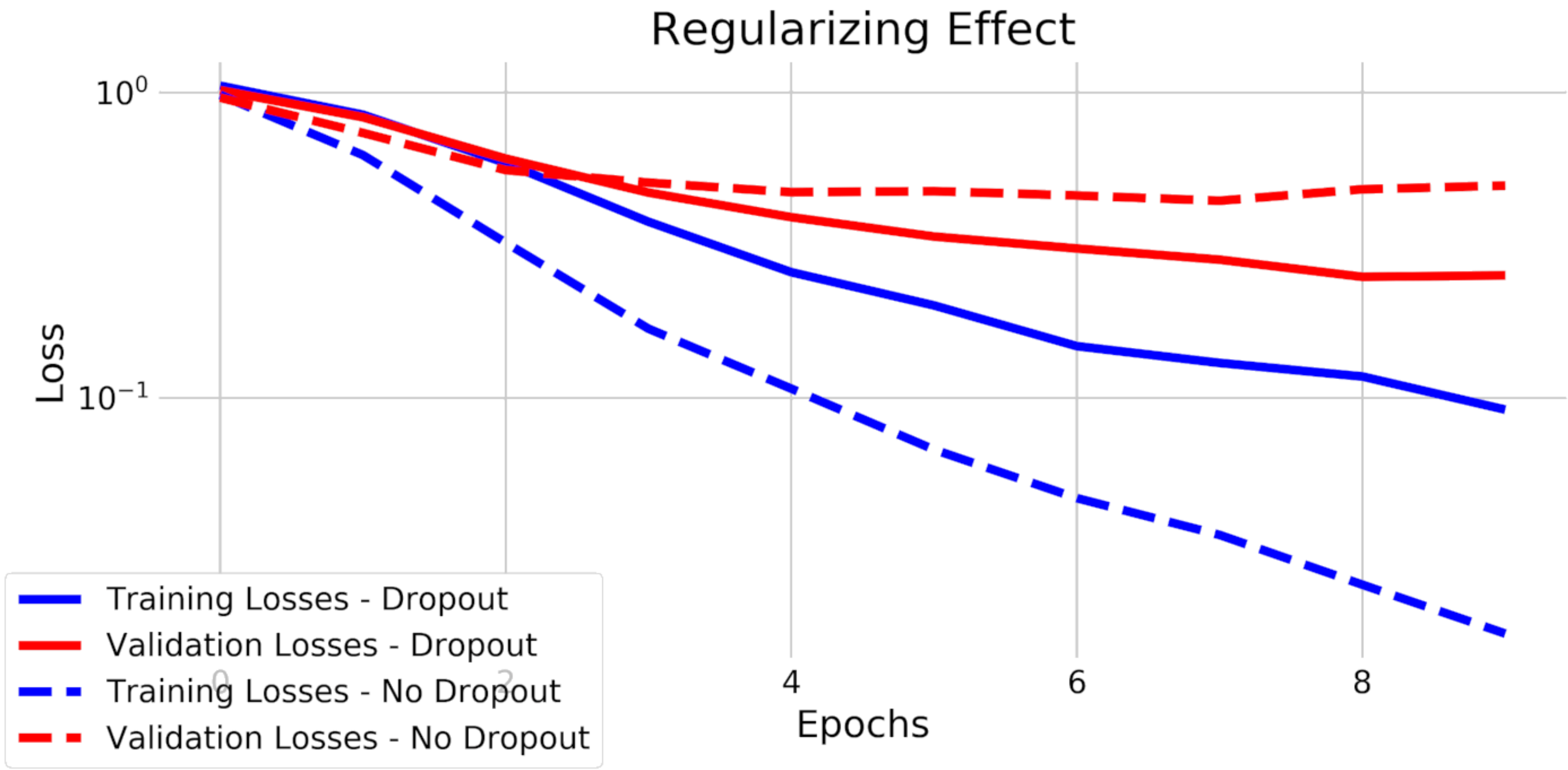
- Equivalent to bagging with an ensemble of subnets
- Inference on the full network is geometric mean

$$\hat{p}(y|x) \approx \sqrt[d]{\prod_{i=1}^{d} p_i(y|x)}$$



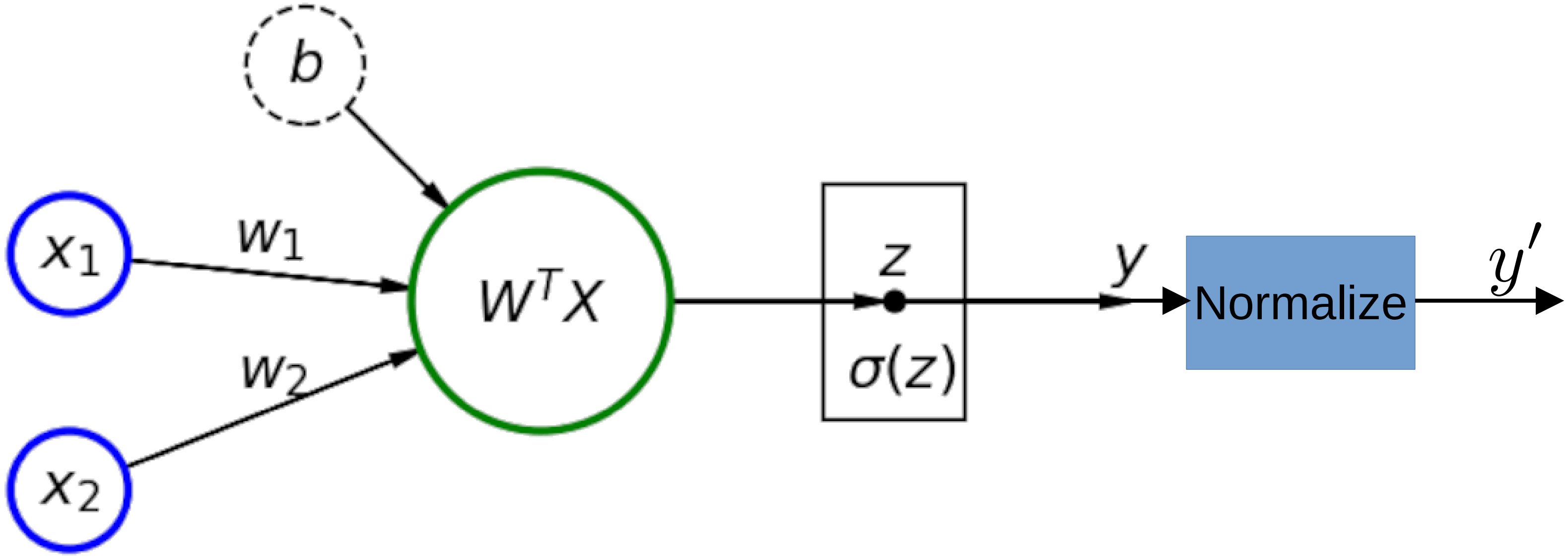

Ensemble of subnetworks

# Dropout Performance



# Batch Normalization

- Standard hidden unit: $y = \sigma(Wx + b)$




- Normalize output across the minibatch $y' = \dfrac{y - \bar{y}}{\sqrt{\mathrm{var}(y) + \epsilon}}$

$$\bar{y} = \frac{1}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} y^{(i)} \qquad \mathrm{var}(y) = \frac{1}{|\mathbb{B}|} \sum_{i \in \mathbb{B}} (y^{(i)} - \bar{y})^2$$

# Batch Normalization

- Training: update $\bar{y}, \mathrm{var}(y)$ with every batch
- Evaluation: keep learned statistics $\bar{y}, \mathrm{var}(y)$ (EWMA) from the training phase
- Output of hidden layers have mean=0 and var=1

Exponential Weighted Moving Average

- Helps to solve Vanishing/Exploding gradients
- In reality BN makes the loss surface smoother!

- Implementation:
  - Extra affine transf. param. are learnable: $\gamma * y' + \beta$
  - Why? - > gradients w.r.t. param. are simpler
  - BN before or after the activation function?

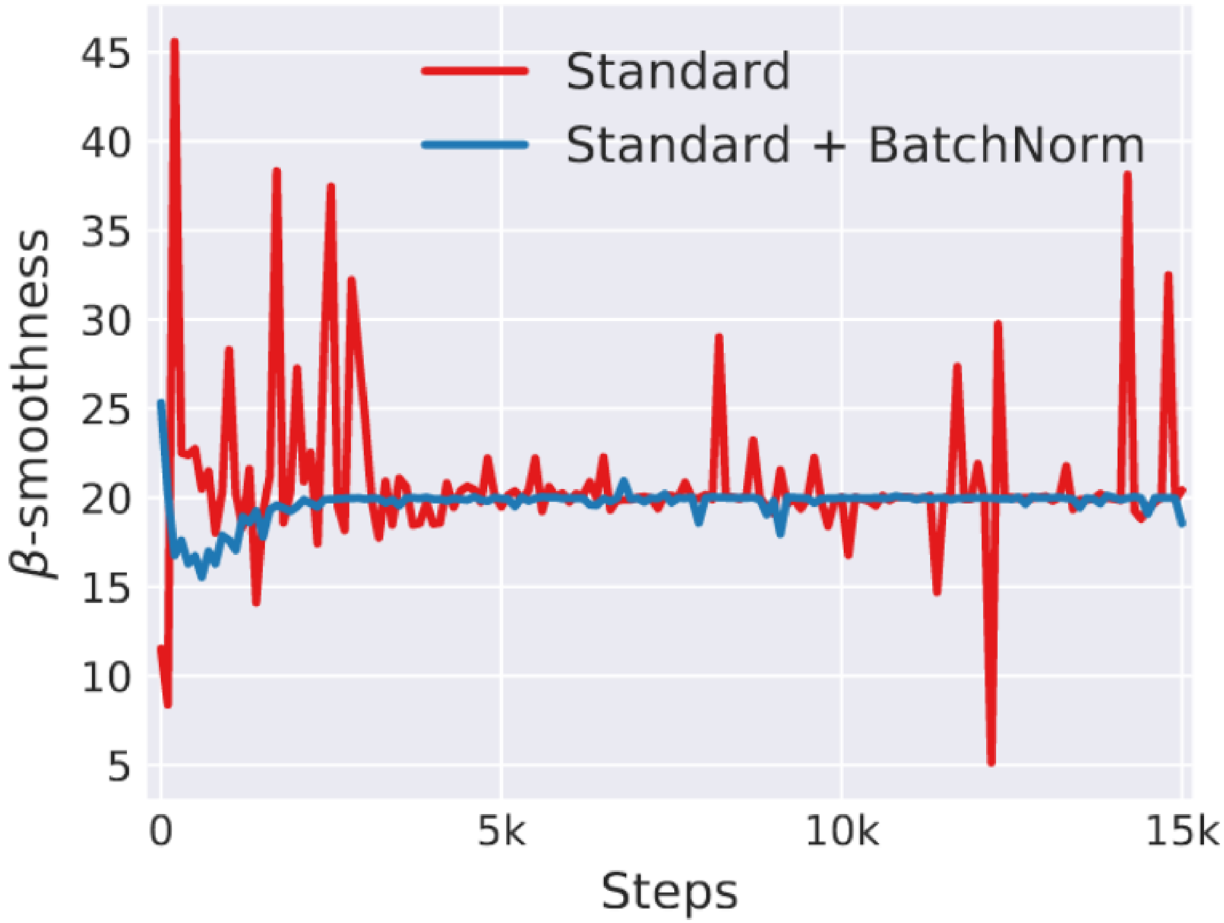
That is the question.

# BN and loss smoothness

- BN decreases Lipschitz constant of the loss

  making it more smooth

$$|f(x_1) - f(x_2)| \leq \beta \|x_1 - x_2\| \quad \forall x_1, x_2$$




Santurkar et al., CoRR 2018

# BN takeaway

- Use BN
  - Architectures with LARGE batches
  - Mean & variance stable across batches
  - Image classification
- Avoid BN
  - Architectures with SMALL batches
  - Unstable mean & variance
  - Object detection / Segmentation / Synthesis

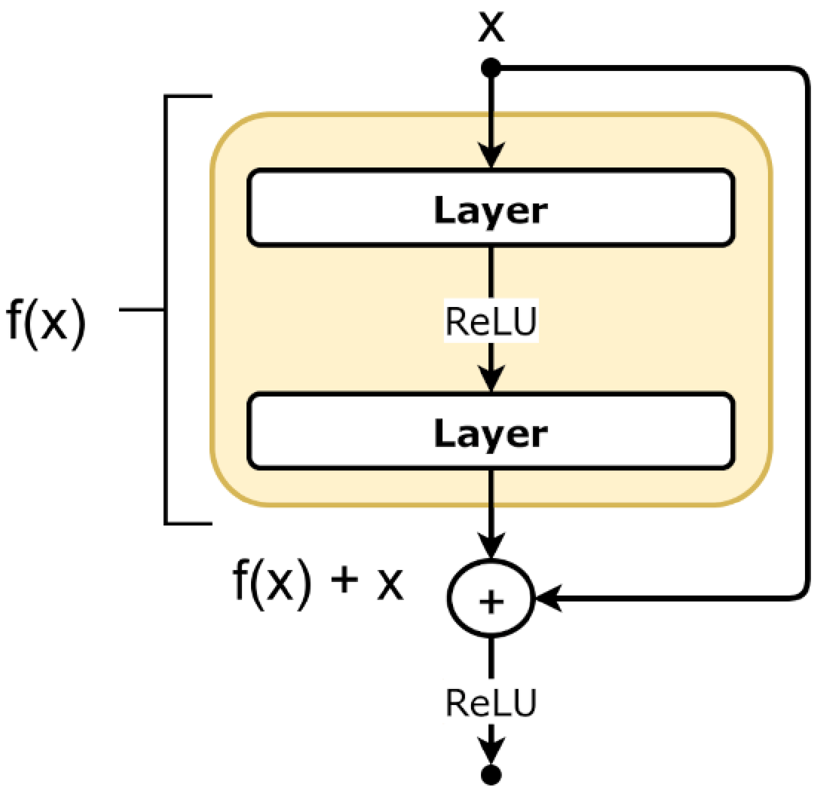
# Residual Connections

- Not trivial to learn "trivial" identity with nonlinearities.

  $y = f(x;\theta)$ learn $\theta$ such that $y = x$

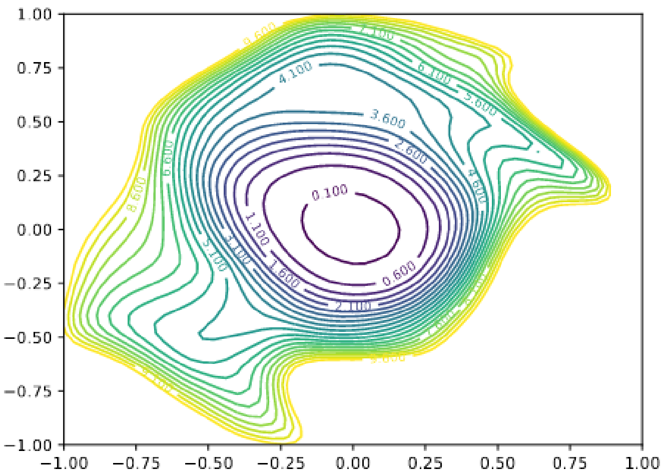
- Easy if

  $$y = f(x;\theta) + x$$

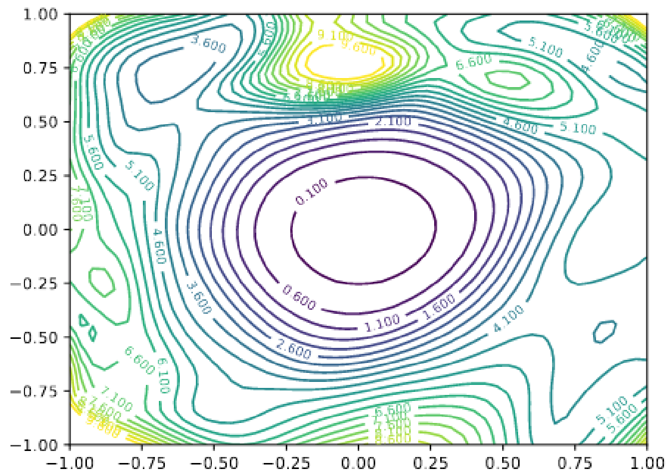


Residual Block

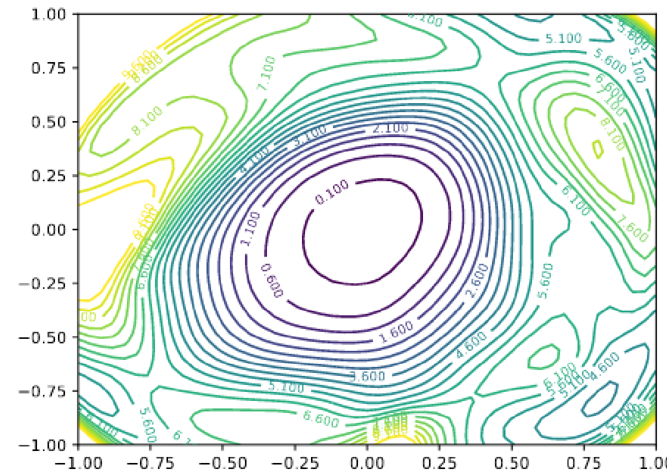
- RC makes the loss surface smoother



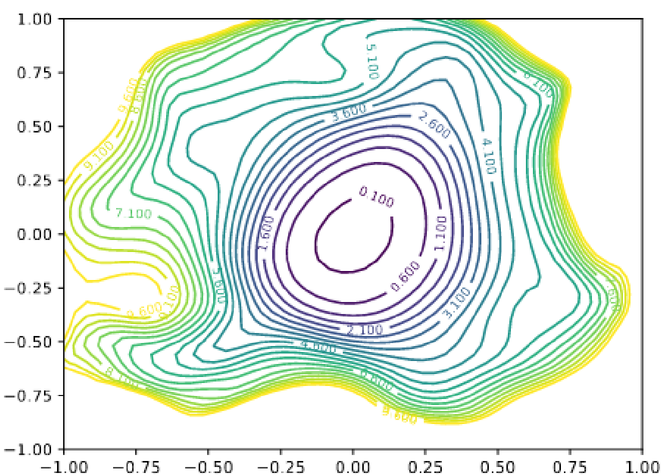
(a) ResNet-20, 7.37%



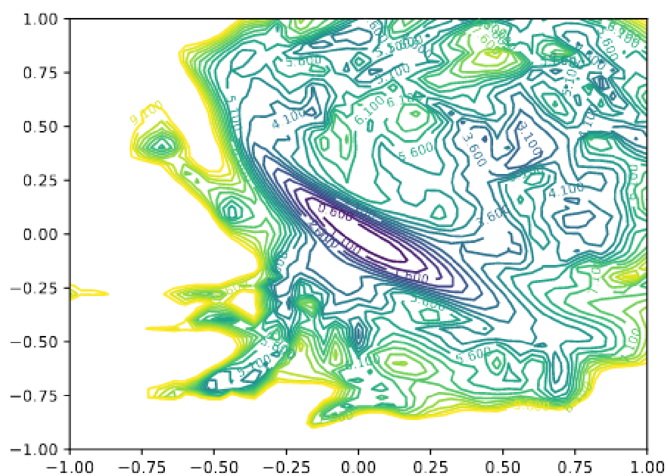
(b) ResNet-56, 5.89%



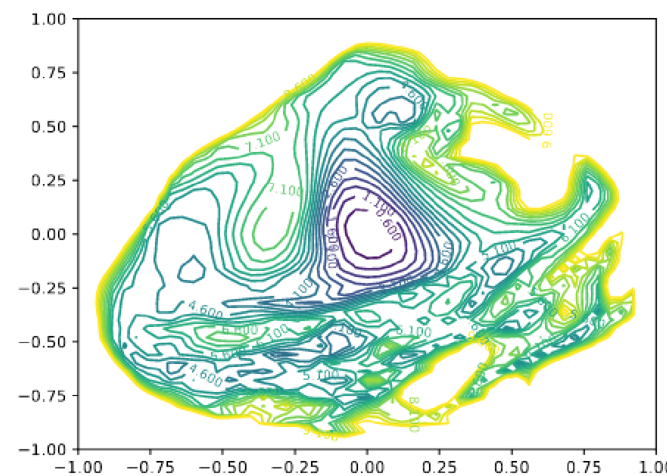
(c) ResNet-110, 5.79%



(d) ResNet-20-NS, 8.18%



(e) ResNet-56-NS, 13.31%



(f) ResNet-110-NS, 16.44%

Li et al., Visualizing the loss landscape, 2018