CMPT 318
Data Analytics

Learning Algorithms II

Adapted in part from
Chapter 8 of Data Science from Scratch (Grus)
Chapter 4, 5 of Deep Learning (Goodfellow)
Chapter 1 of Convex Optimization (Boyd, ISBN: 9780521833783)

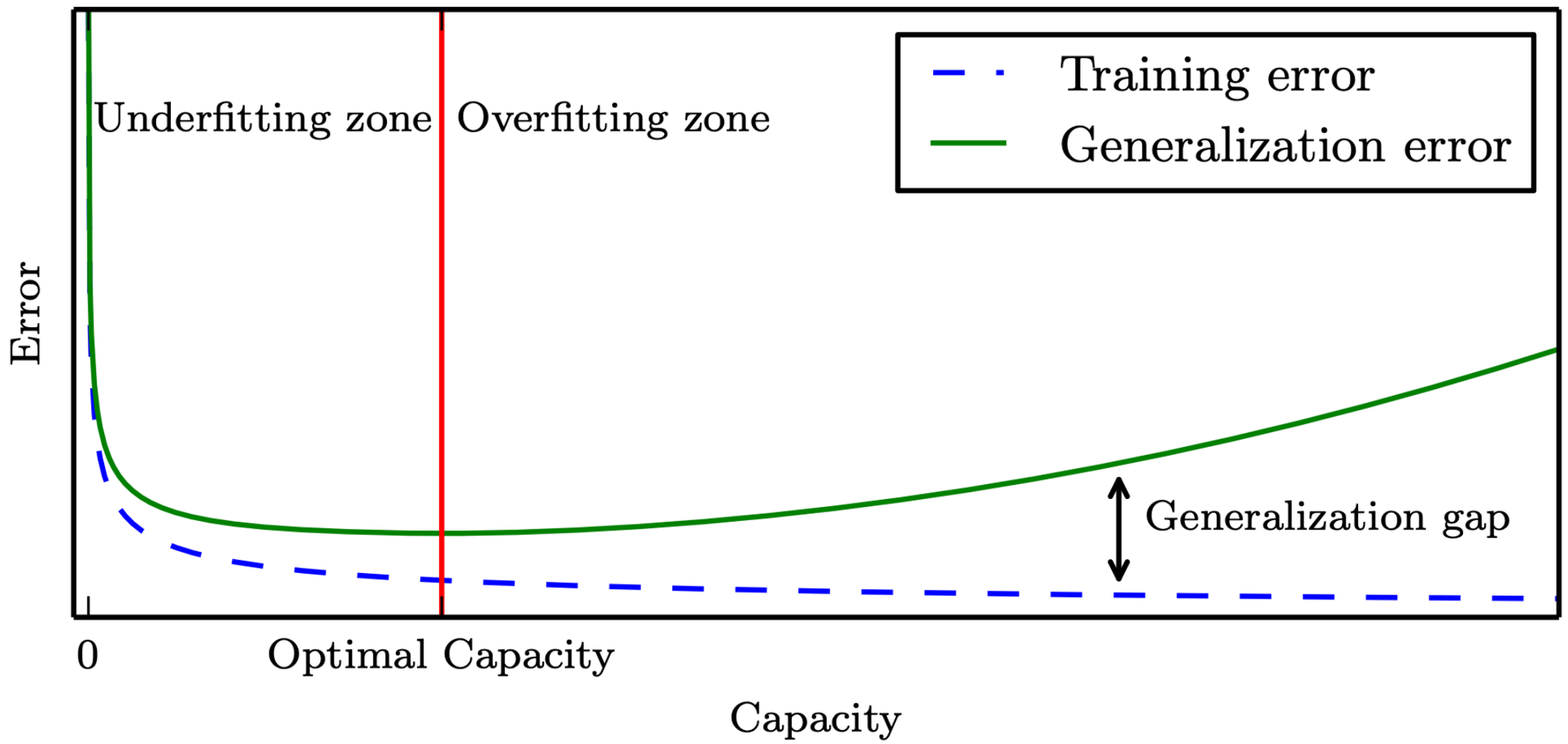
Upcoming lecture schedule

- Proposals will be graded/returned Thursday

- Friday (Nov. 3) - Project Q&A, individual meetings to discuss any project roadblocks. Attendance optional.

- Monday (Nov. 13) - starting on Neural Networks

Generalization and Capacity



Regularization

*A*ny modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

Regularization with Weight Decay

- We can reduce model capacity by:
  1. Allowing lots of functions, but providing a preference for one solution over another, e.g. prefer solutions with smaller norm

$$J(\boldsymbol{w}) = \mathrm{MSE}_{\mathrm{train}} + \lambda \boldsymbol{w}^\top \boldsymbol{w}, \tag{5.18}$$

Weight Decay

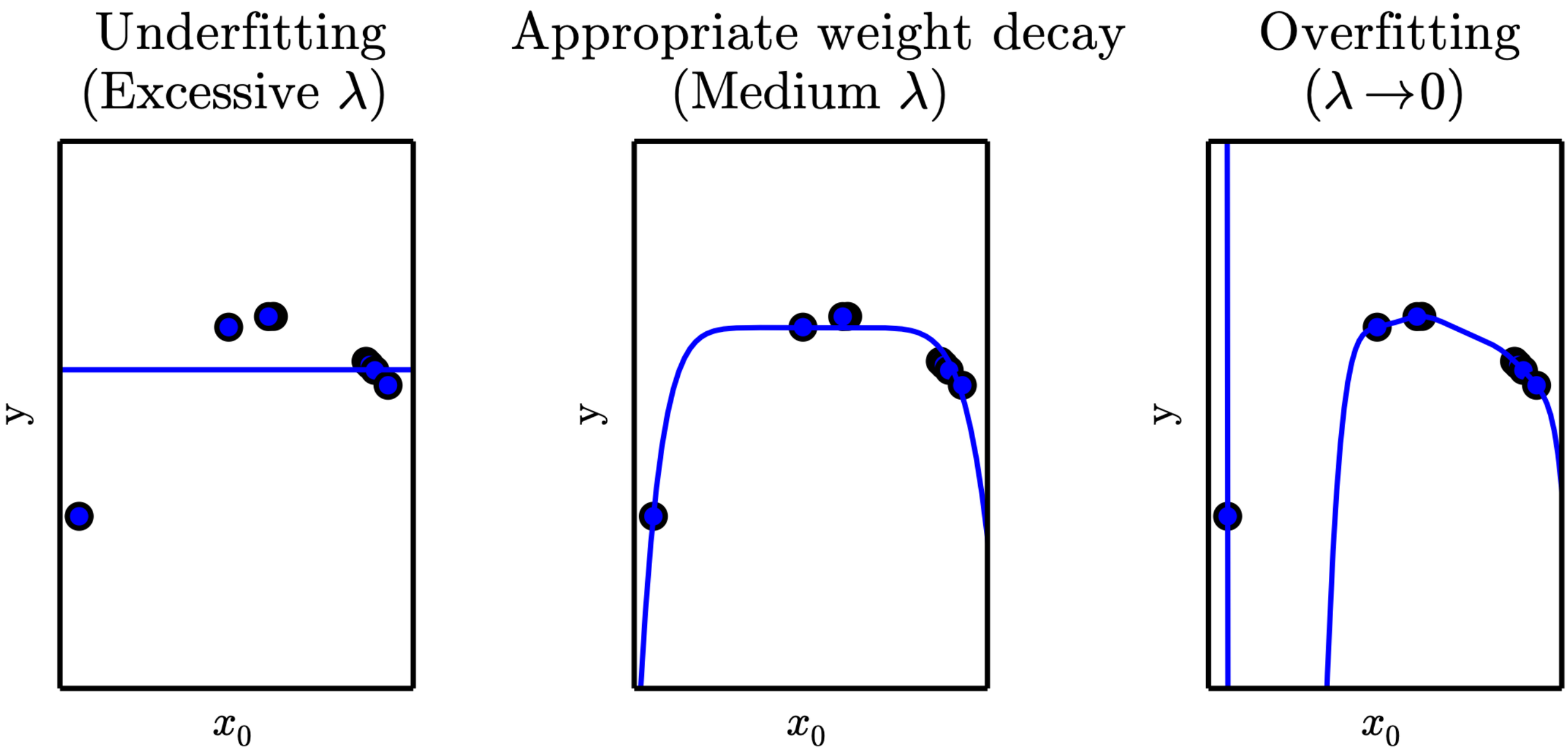


Figure 5.5

(Goodfellow 2017)

L2-norm regularization with Linear Regression

↳ Ridge

(Review ℓ-1 norm → Lasso)

$$\hat{y}^{(i)} = w^T \cdot x^{(i)}$$

$$\hat{y} = Xw \quad (\beta \equiv w)$$

→ Design Matrix / Data Matrix $X \in \mathbb{R}^{n \times m}$

n – #samples

m – # features

$$\min_w \; \frac{1}{2} \| \hat{y} - y^* \|_2^2 + \lambda \|w\|_2^2$$

$$\frac{1}{2} \| Xw - y^* \|_2^2 + \lambda w^T w$$

$$\frac{1}{2} (Xw - y^*)^T (Xw - y^*) + \lambda w^T w$$

$$\frac{1}{2} \left(w^T X^T X w - 2 w^T X^T y^* + y^{*T} y^* \right) + \lambda w^T w$$

(ignore)

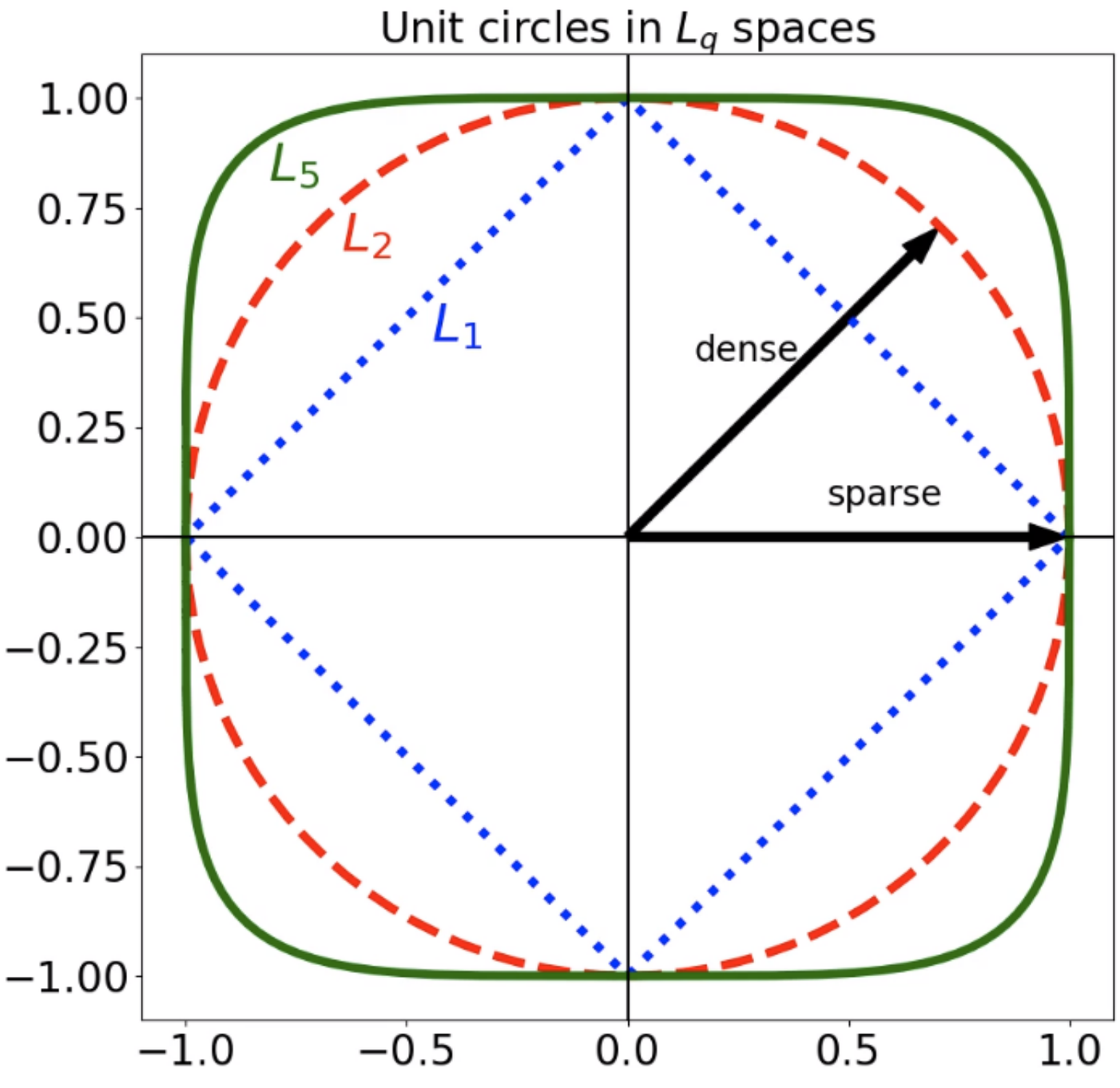
$$\nabla = 0 : \quad \nabla_w = X^T X w - X^T y^* + 2\lambda w$$

$$= (X^T X + 2\lambda I) w - X^T y^* = 0$$

solve for w ⟶ $w = (X^T X + I 2\lambda)^{-1} X^T y^*$

Parameter Norm Penalties

Norm



Training

- Define loss (objective) function
- Compute gradient of loss w.r.t. parameters
- Update parameters by moving “downhill” in the direction of the negative gradient
- If loss stops decreasing (or other convergence criteria satisfied), you are done!

Computing Gradients

1. ***Analytically*** (with equations for partial derivatives)

2. ***Numerically*** (with repeated, perturbed forward solves)

3. ***With Automatic Differentiation*** (built-in to tensorflow, pytorch, etc.)

Gradient Descent

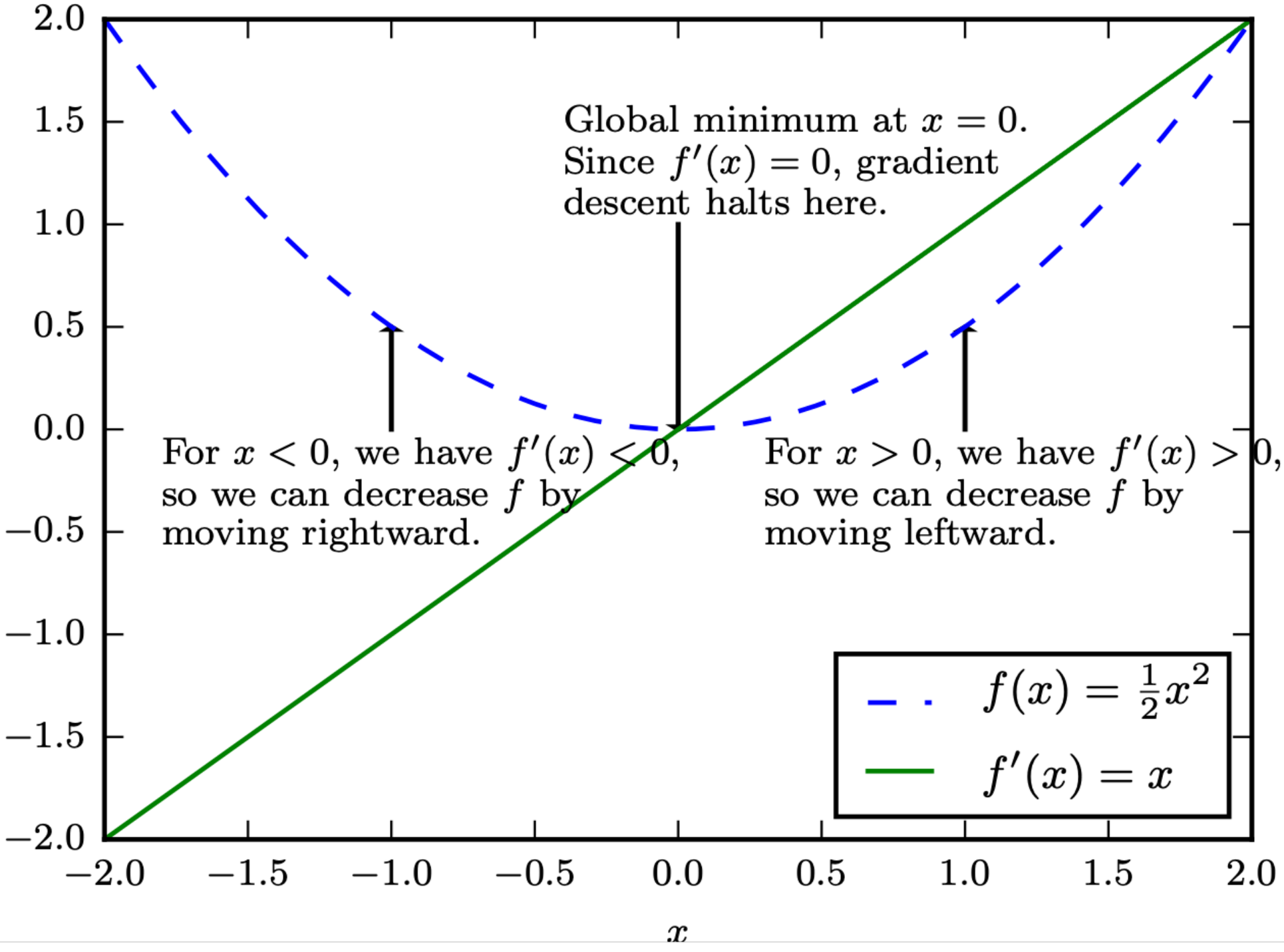


Figure 4.1

Gradient Descent with Poor Conditioning

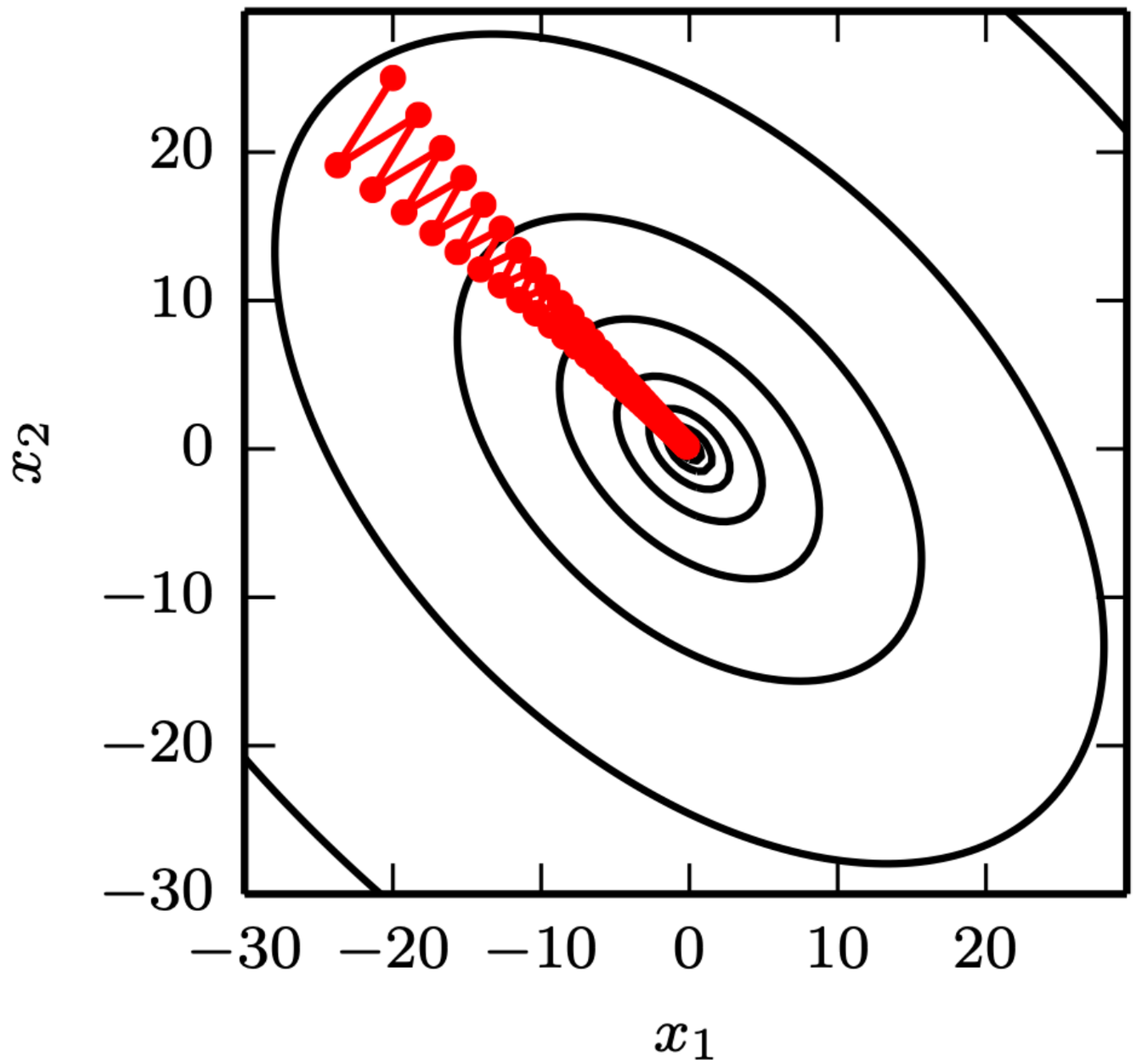


Figure 4.6

(Goodfellow 2017)

Curvature

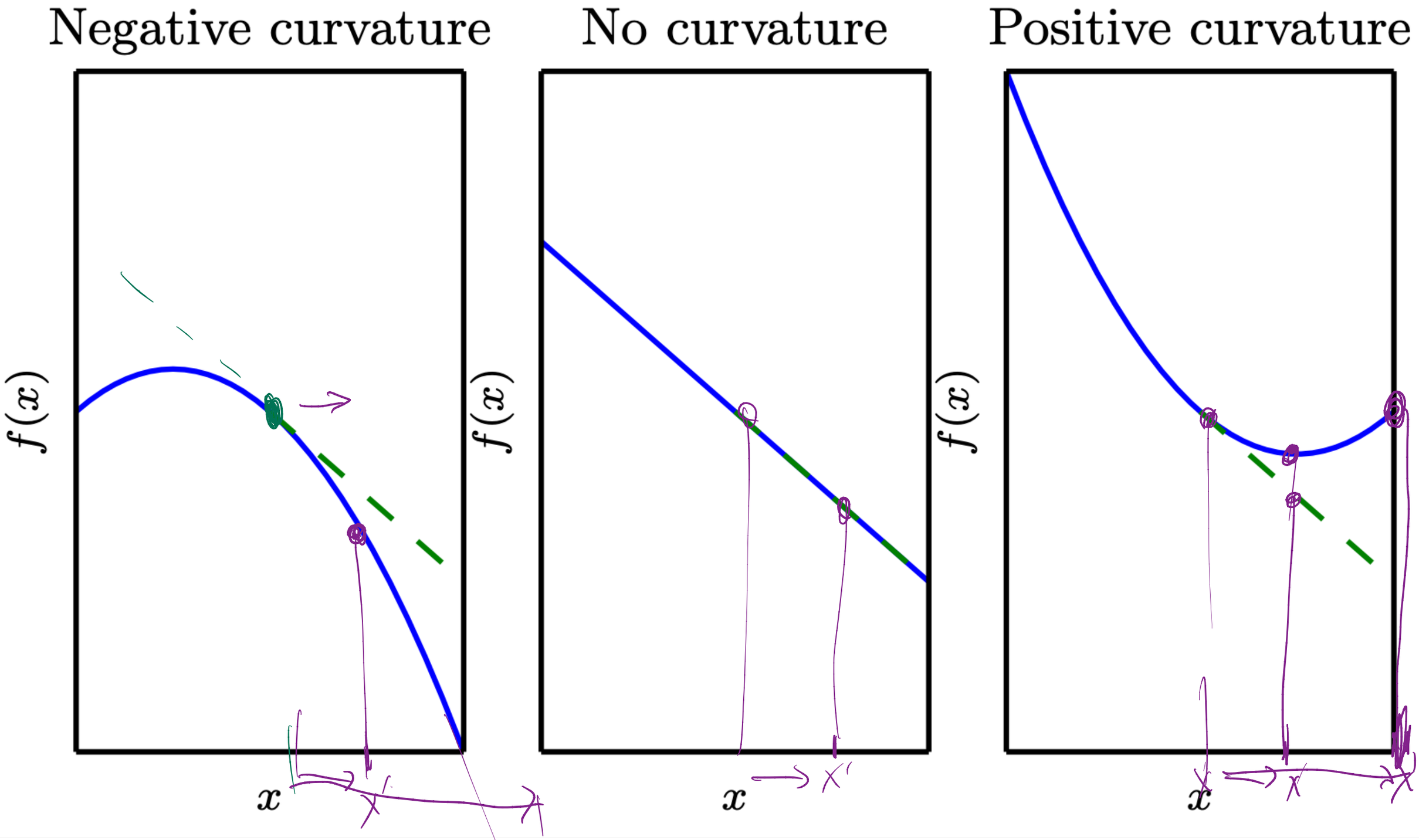


Figure 4.4

(Goodfellow 2017)

Newton's Method + Raphson (trade off b/w 1ST / 2nd order)

Taylor series of loss

1st order

2nd order

$$f(x)$$

$$f(x+\Delta x) = f(x) + f' \Delta x + f'' \Delta x^2 + \cdots$$

1st derivative info "Gradient"

2nd derivative information "Hessian"

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})(\boldsymbol{x} - \boldsymbol{x}^{(0)}).$$

$$\boldsymbol{x}^* = \boldsymbol{x}^{(0)} - \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}). \tag{4.11}$$

Predicting optimal step size using Taylor series

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}. \tag{4.9}$$

$$\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}}. \tag{4.10}$$

Big gradients speed you up

Big eigenvalues slow you down if you align with their eigenvectors

Challenges in Neural Network Optimization

- Ill-Conditioning
- Local Minima

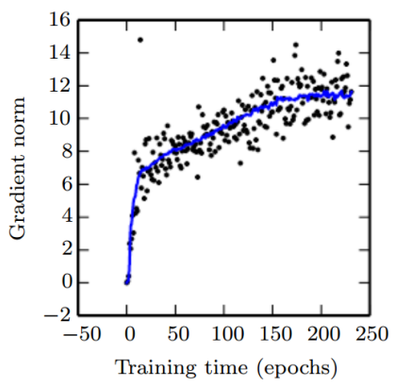


Fig 8.1 Goodfellow et al. 2016

- Plateaus, Saddle Points and Flat regions
- Clifs and Exploding gradients

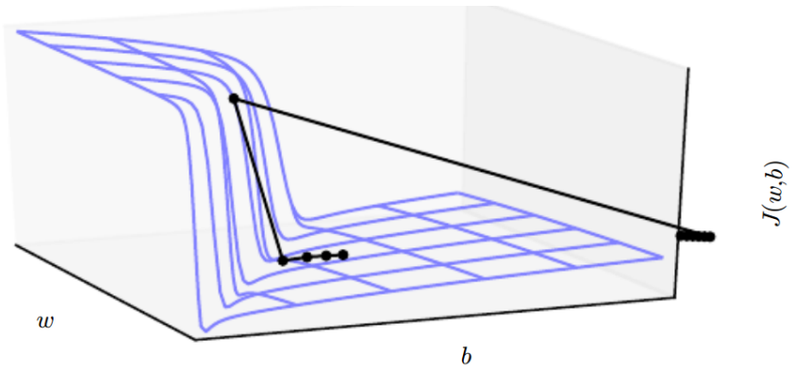


Fig 8.3 Goodfellow et al. 2016

- Inexact Gradients
- Long Term Dependencies

Gradient Descent with Mini-batches

Larger batch size $\rightarrow$ more accurate estimate of gradient

Memory required scales with batch size

Batch sizes of power of 2 (32, 64, 128, 256)

For stability use a lower learning rate if batch size is smaller

Select minibatches randomly so that the samples are independent

Stochastic Gradient Descent

Algorithm 7: Stochastic gradient descent (SGD) at training iteration k

Require: Learning rate ϵ_k

Require: Initial parameter $\boldsymbol{\theta}$

while *stopping criterion not met* **do**

m is subset of training samples

Sample a minibatch of m examples from the training set $\{x(1), \ldots, x(m)\}$ with corresponding targets $y(i)$;

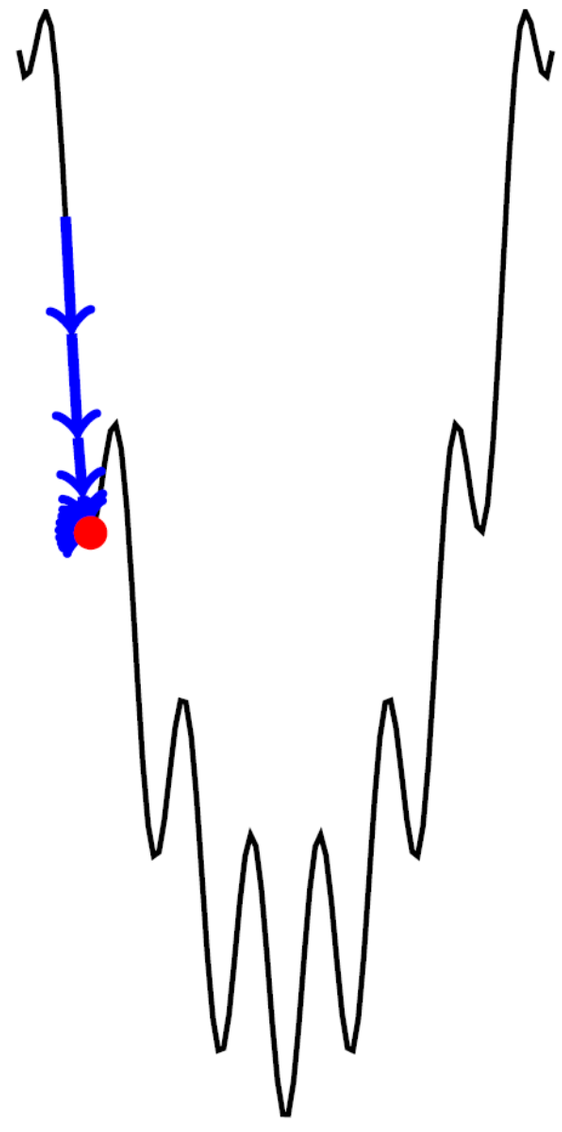
Compute gradient estimate: $\hat{g} \leftarrow \frac{1}{m} \nabla \sum_i L\left(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}), y^{(i)}\right)$;

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\mathbf{g}}$;

end

Stochastic Gradient Descent

Learning rate: too low



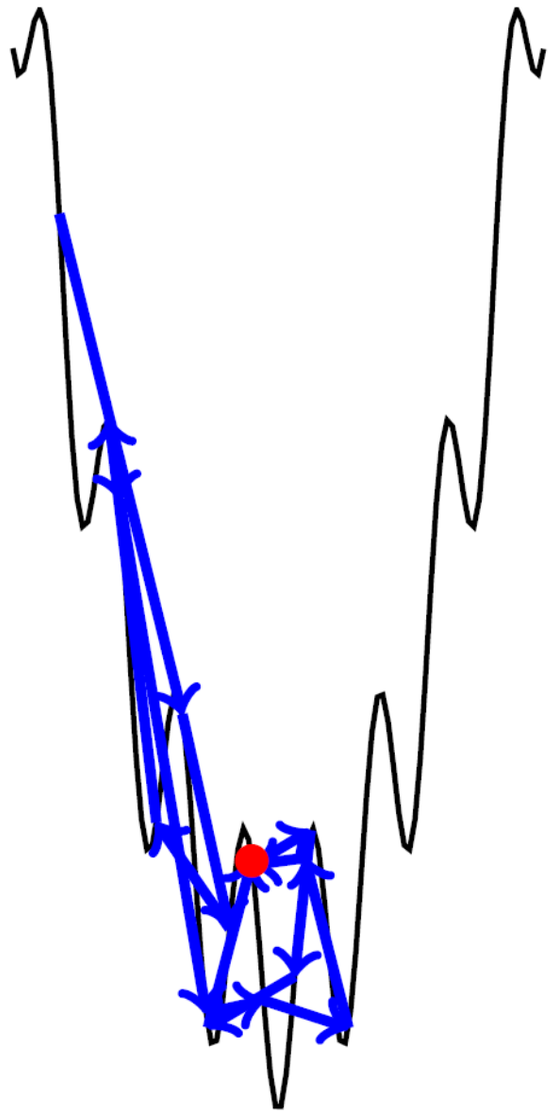
$\nabla J = -0.0538 \qquad \mathbf{w} = -2.92369$

$n = 14 \ \eta = 0.005$

Stochastic Gradient Descent

Learning rate: too high

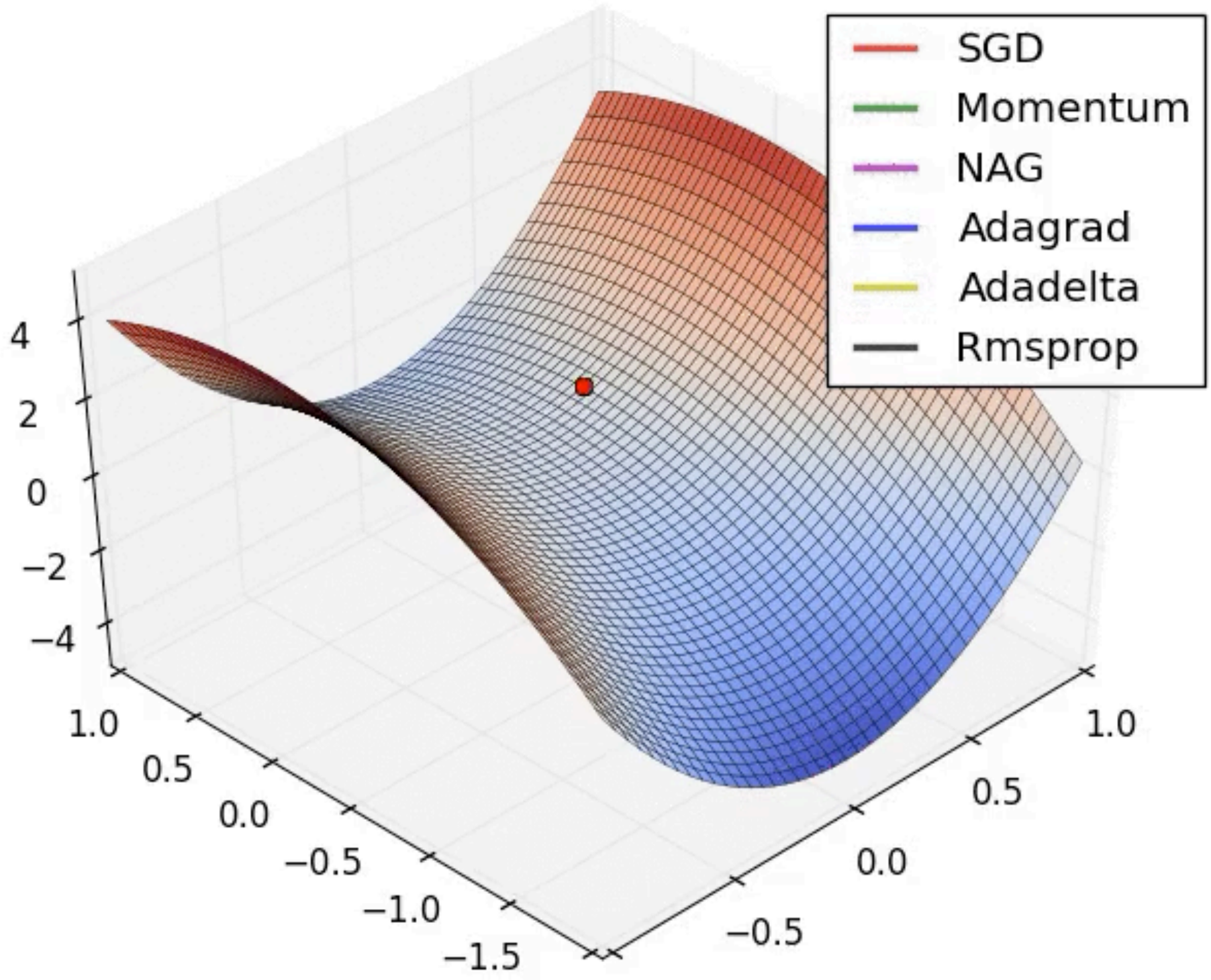
Rate scheduling built-in, eg. Adam



$\nabla J = -8.28964$ $\mathbf{w} = -0.39825$

$n = 14$ $\eta = 0.1$

Visualize Adaptive Learning Rates



https://i.imgur.com/2dKCQHh.gif

Dataset Augmentation

A classifier should be invariant to a wide variety of transformations
Generate new (x, y) pairs by transforming the x inputs in our training set:

- translation
- rotation
- scaling
- cropping

Not always possible, e.g., out-of-plane rotation
General idea: noise injection (at the inputs x)

Injecting Noise at the Output Targets

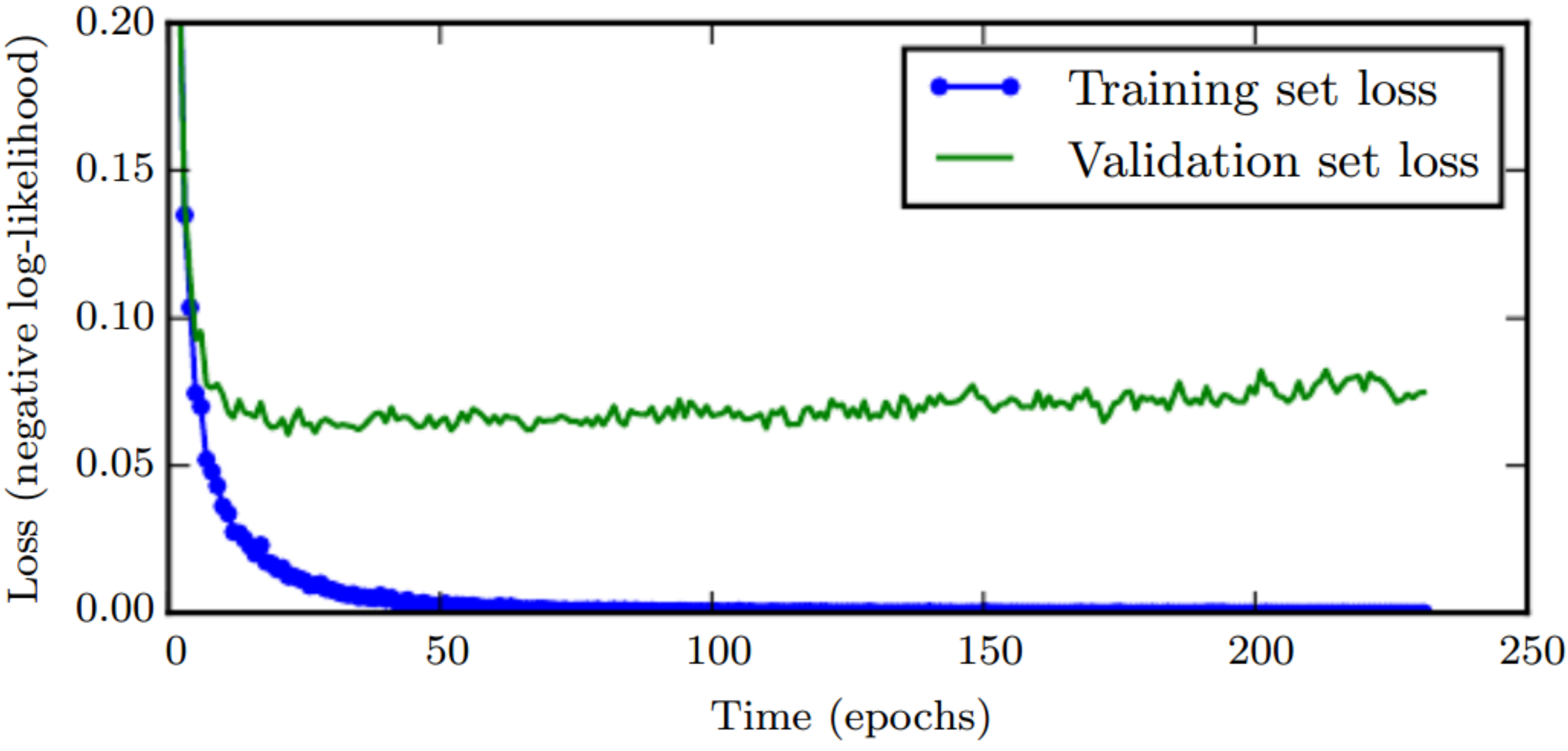
Label Smoothing:
Generally the target is

$$y = [0, 0, 0, 1, 0]$$

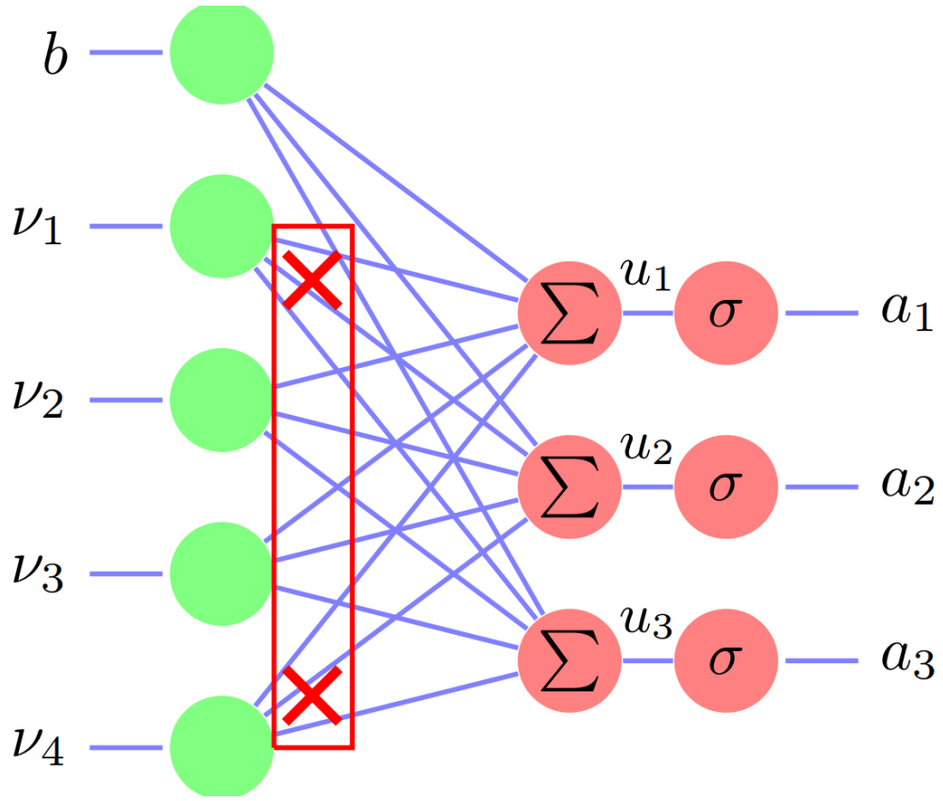
Use instead:

$$y = [\frac{\epsilon}{k-1}, \frac{\epsilon}{k-1}, \frac{\epsilon}{k-1}, 1-\epsilon, \frac{\epsilon}{k-1}]$$

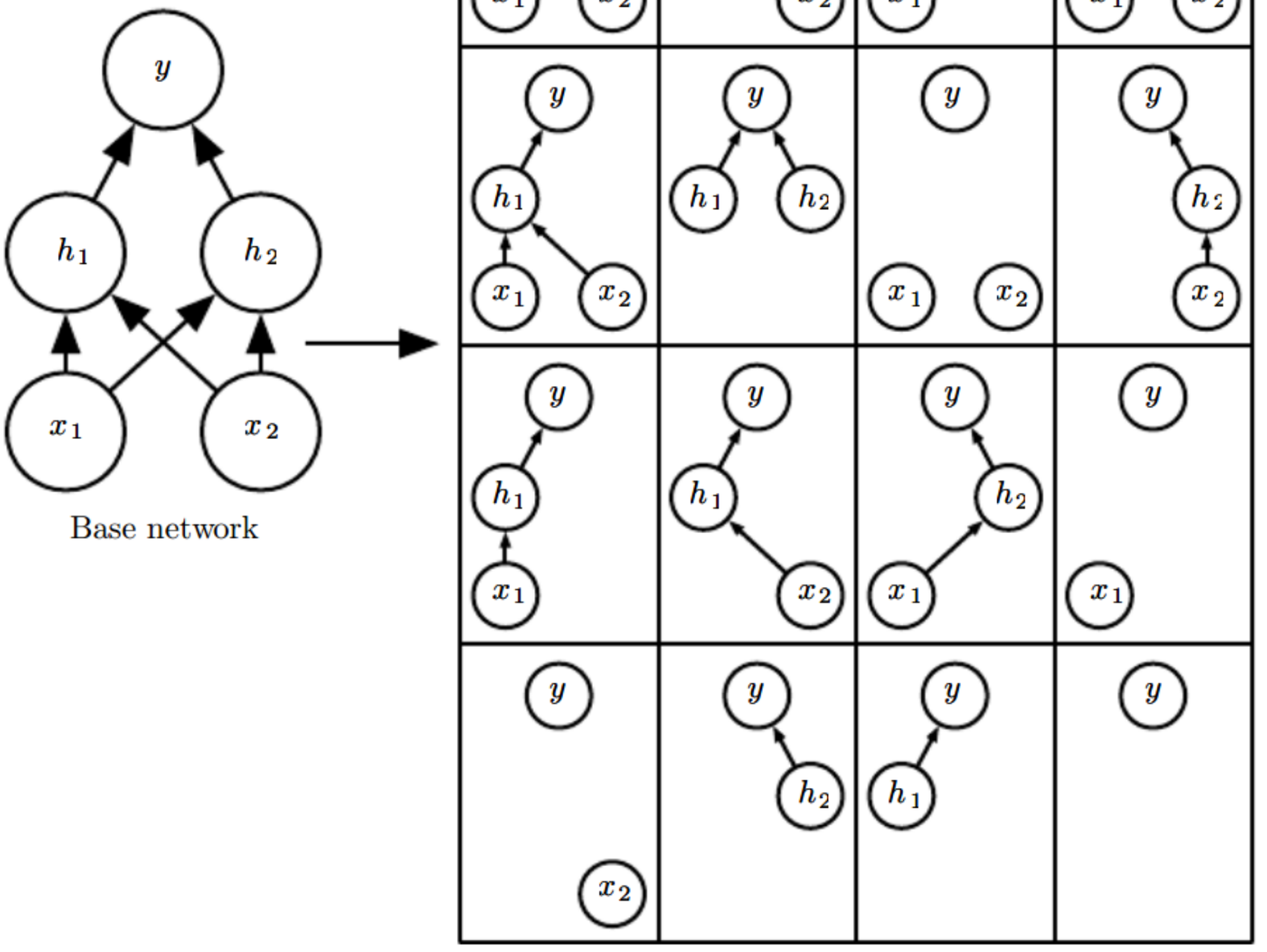
Early Stopping



Dropout



Dropout



Base network

Ensemble of subnetworks