Unit 3
Deep Feedforward Networks

TFIP-AI Artificial Neural Networks and Deep Learning

Roadmap

- Example: Learning XOR
- Gradient-Based Learning
- Hidden Units
- Architecture Design
- Back-Propagation

XOR is not linearly separable

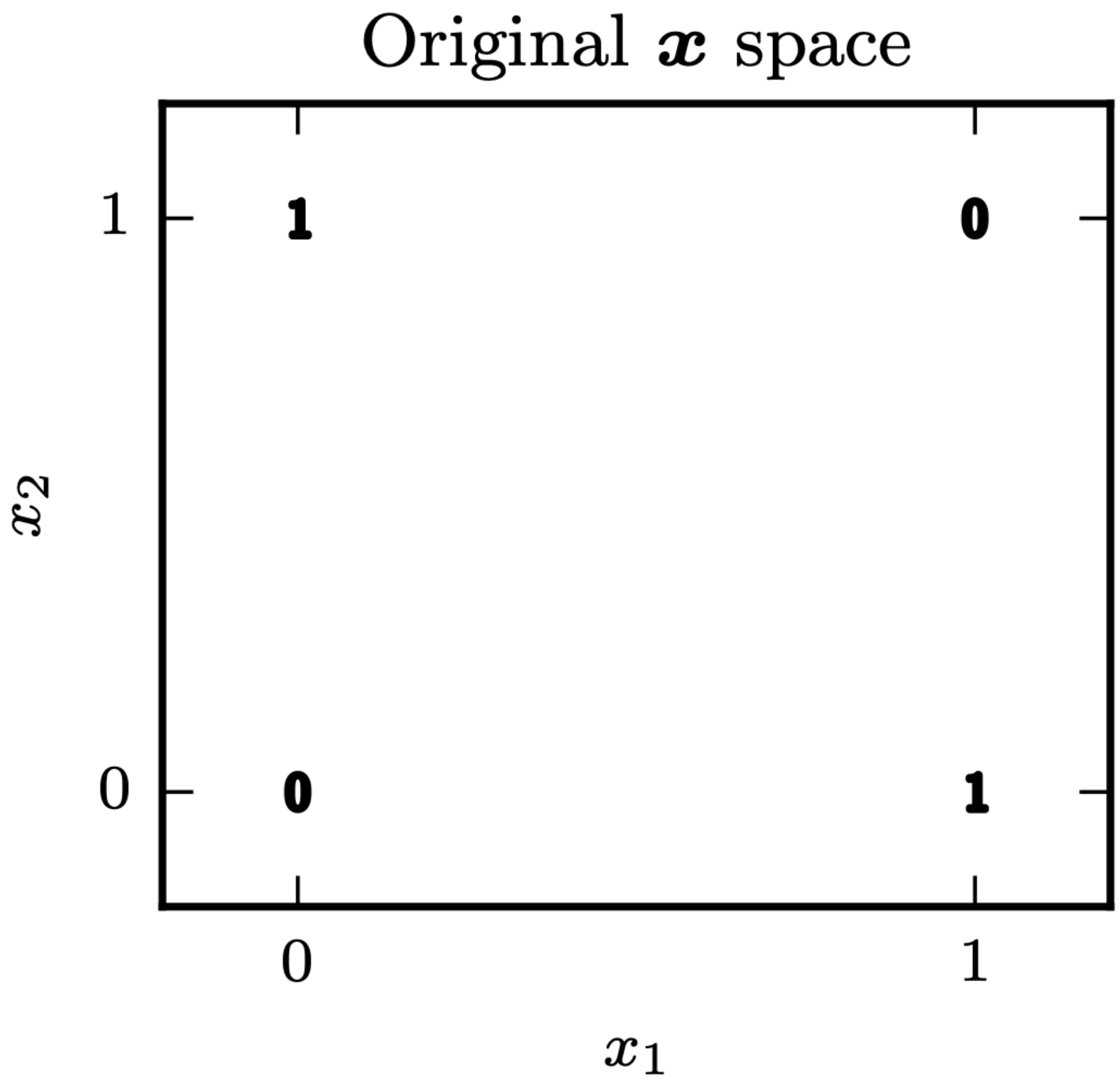


Figure 6.1, left

XOR is not linearly separable cont…

The MSE loss function is:

$$J(\boldsymbol{\theta}) = \frac{1}{4} \sum_{\boldsymbol{x} \in \mathbb{X}} (f^*(\boldsymbol{x}) - f(\boldsymbol{x}; \boldsymbol{\theta}))^2 .$$

Suppose we choose a linear model as follows:

$$f(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{x}^\top \boldsymbol{w} + b.$$

After solving the normal equations, we obtain w=0 and b=1/2. The linear model simply outputs 0.5 everywhere.

Rectified Linear Activation

In modern neural networks, the default recommendation is to use the rectified linear unit (ReLU) defined by the following activation function depicted in Figure 6.3 in the next slide:

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

Rectified Linear Activation cont…

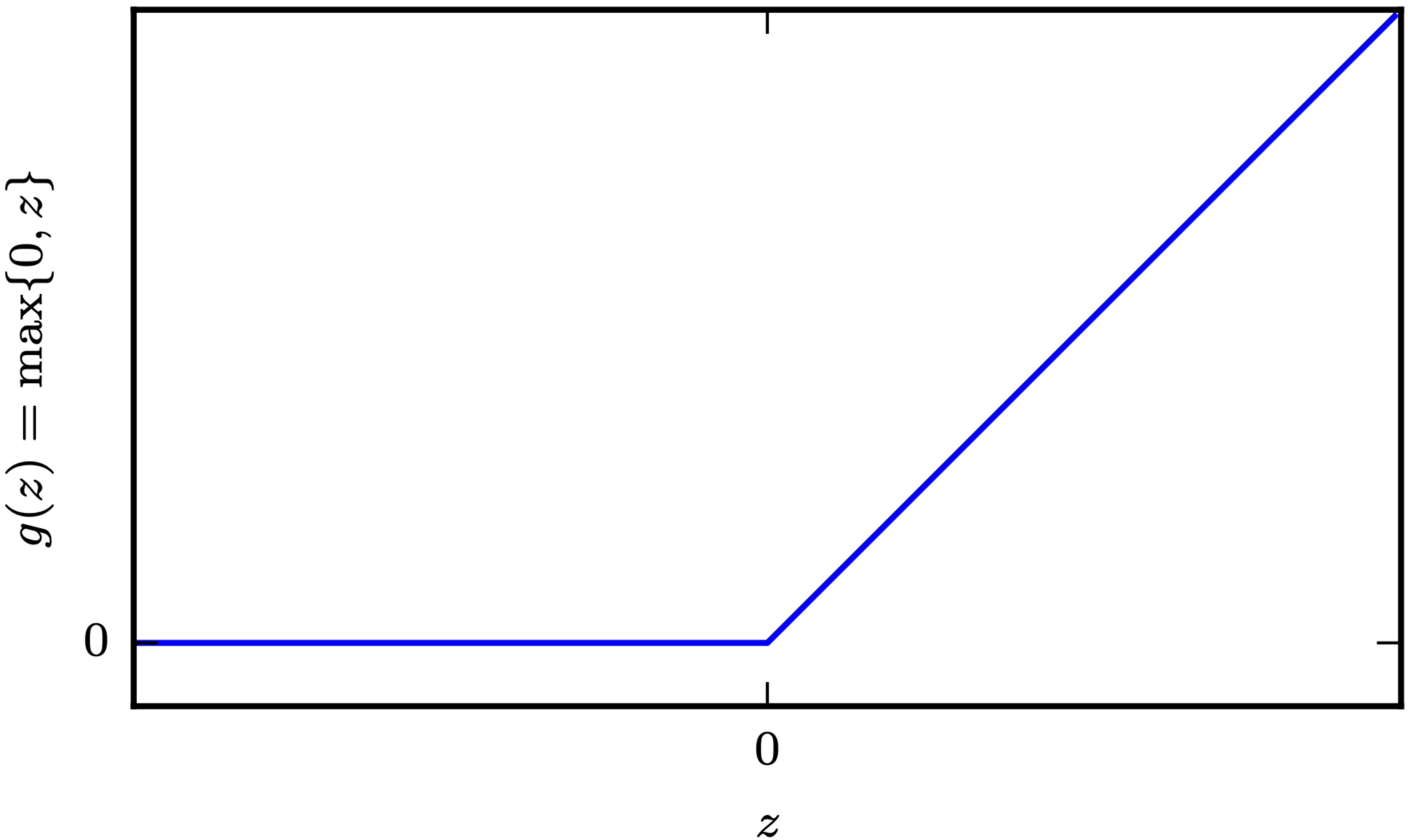


Figure 6.3

Network Diagrams

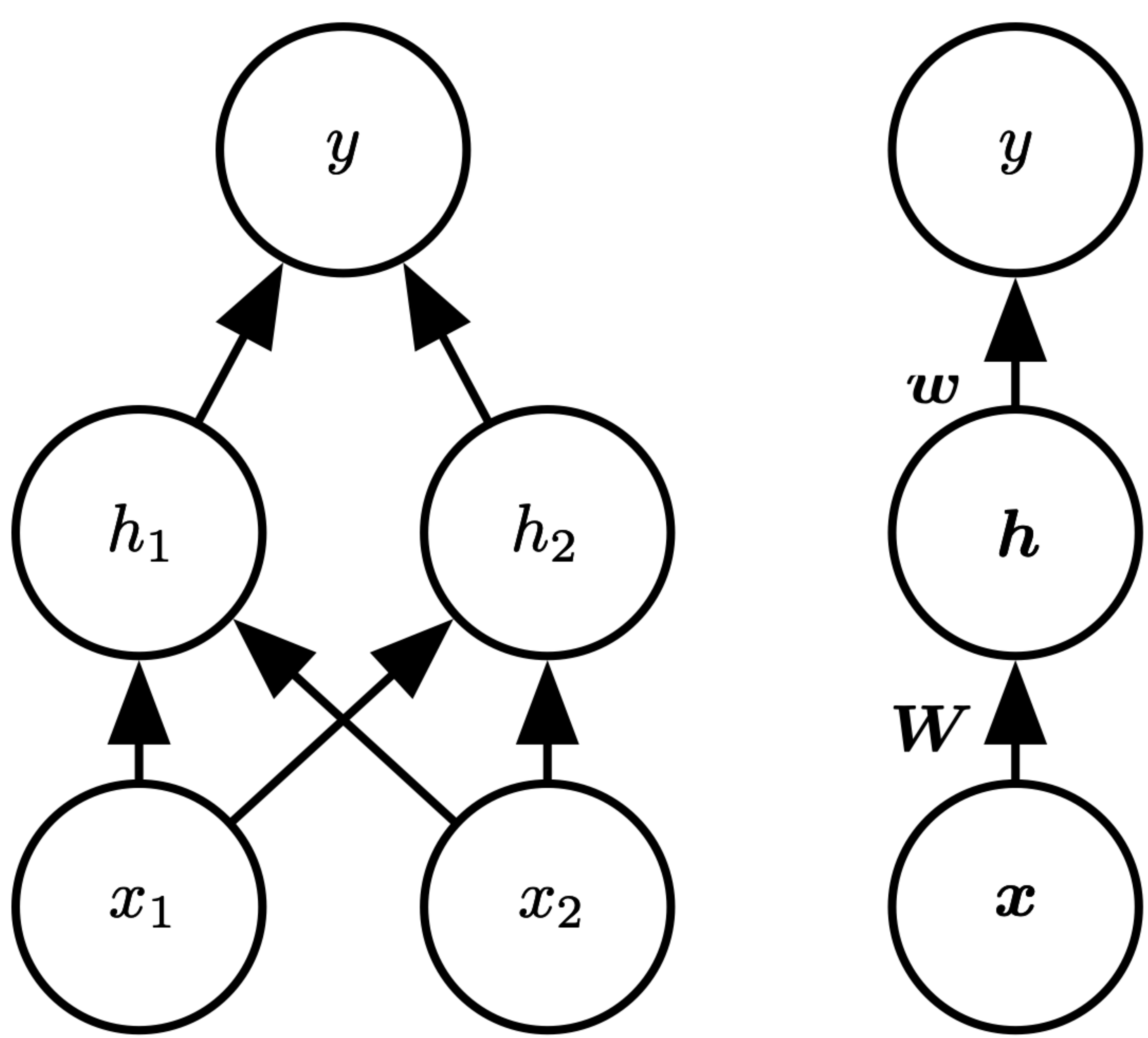


Figure 6.2

Solving XOR

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b. \tag{6.3}$$

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \tag{6.4}$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \tag{6.5}$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \tag{6.6}$$

Solving XOR

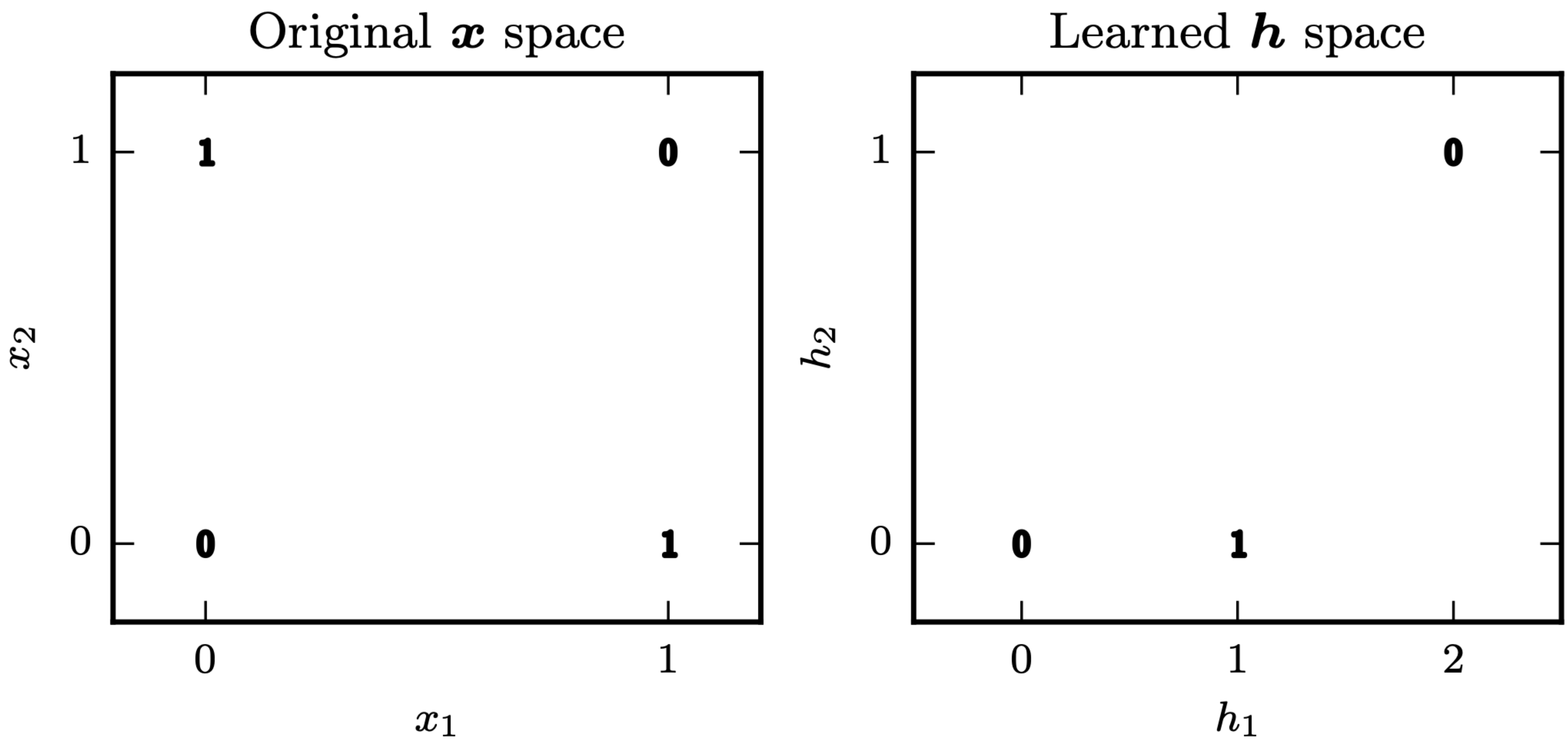


Figure 6.1

Roadmap

- Example: Learning XOR
- Gradient-Based Learning
- Hidden Units
- Architecture Design
- Back-Propagation

Gradient-Based Learning

- Specify
  - Model
  - Cost
- Design model and cost so cost is smooth
- Minimize cost using gradient descent or related techniques

Conditional Distributions and Cross-Entropy

Unfortunately, mean squared error and mean absolute error often lead to poor results when used with gradient-based optimization. Some output units that saturate produce very small gradients when combined with these cost functions. This is one reason that the cross-entropy cost function is more popular than mean squared error or mean absolute error, even when it is not necessary to estimate an entire distribution $p(y \mid x)$.

The choice of cost function is tightly coupled with the choice of output unit. Most of the time, we simply use the cross-entropy between the data distribution and the model distribution. The choice of how to represent the output then determines the form of the cross-entropy function.

Conditional Distributions and Cross-Entropy cont…

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x},\mathbf{y}\sim\hat{p}_{\text{data}}} \log p_{\text{model}}(\boldsymbol{y} \mid \boldsymbol{x}). \tag{6.12}$$

Output Types

Output Type	Output Distribution	Output Layer	Cost Function
Binary	Bernoulli	Sigmoid	Binary cross-entropy
Discrete	Multinoulli	Softmax	Discrete cross-entropy
Continuous	Gaussian	Linear	Gaussian cross-entropy (MSE)
Continuous	Mixture of Gaussian	Mixture Density	Cross-entropy
Continuous	Arbitrary	See part III: GAN, VAE, FVBN	Various

Mixture Density Outputs

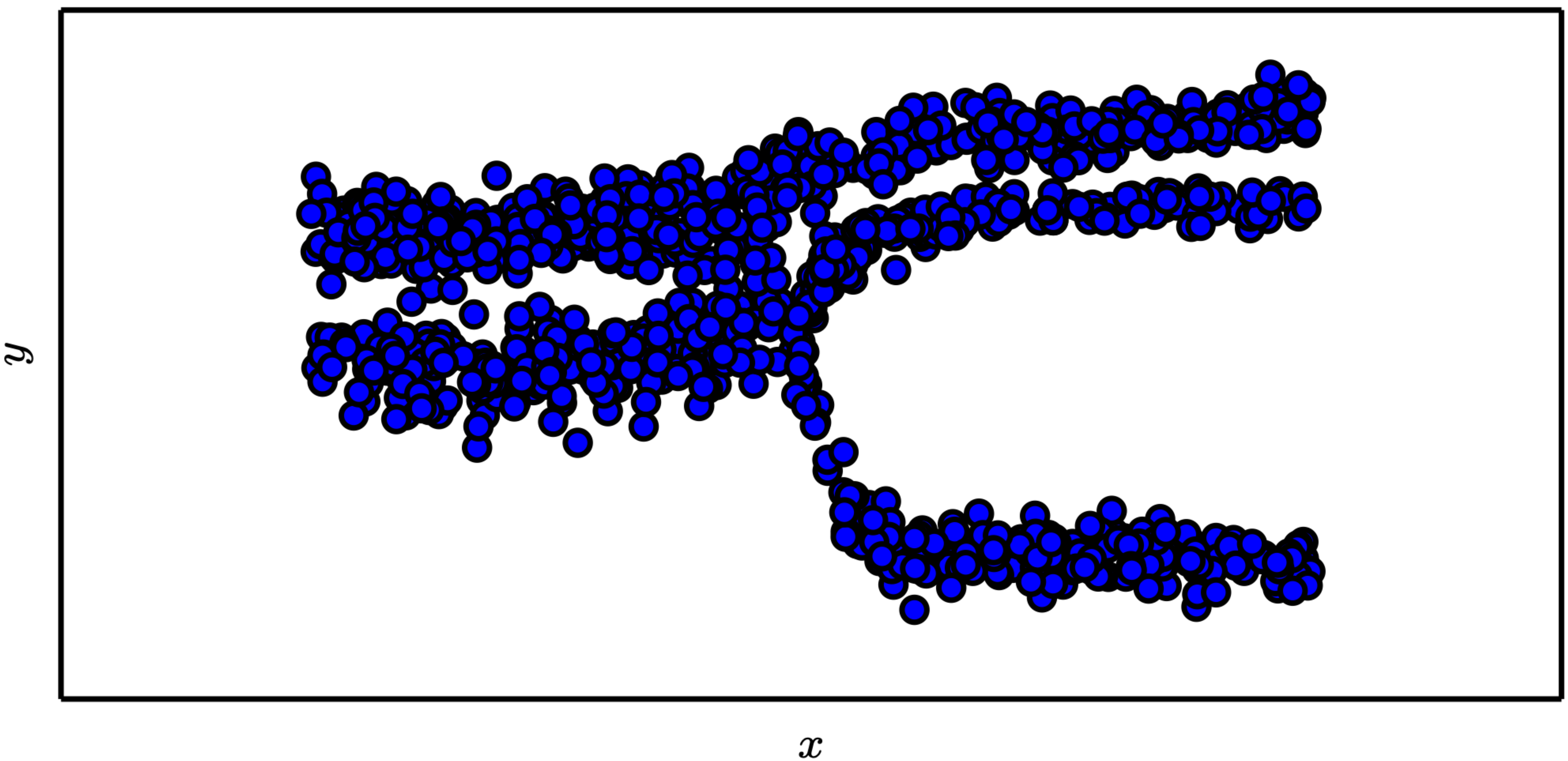
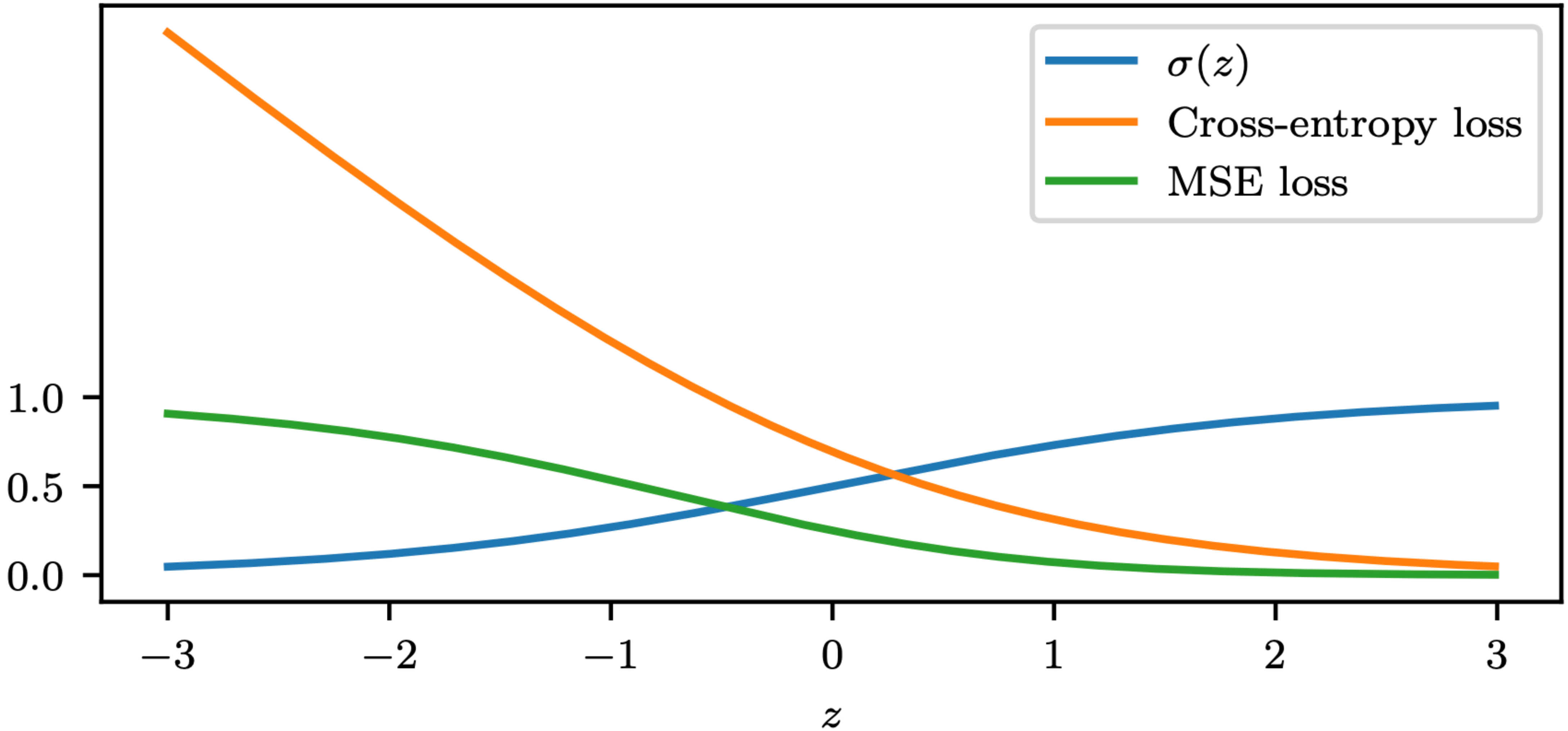


Figure 6.4

Don't mix and match

Sigmoid output with target of 1



Roadmap

- Example: Learning XOR
- Gradient-Based Learning
- Hidden Units
- Architecture Design
- Back-Propagation

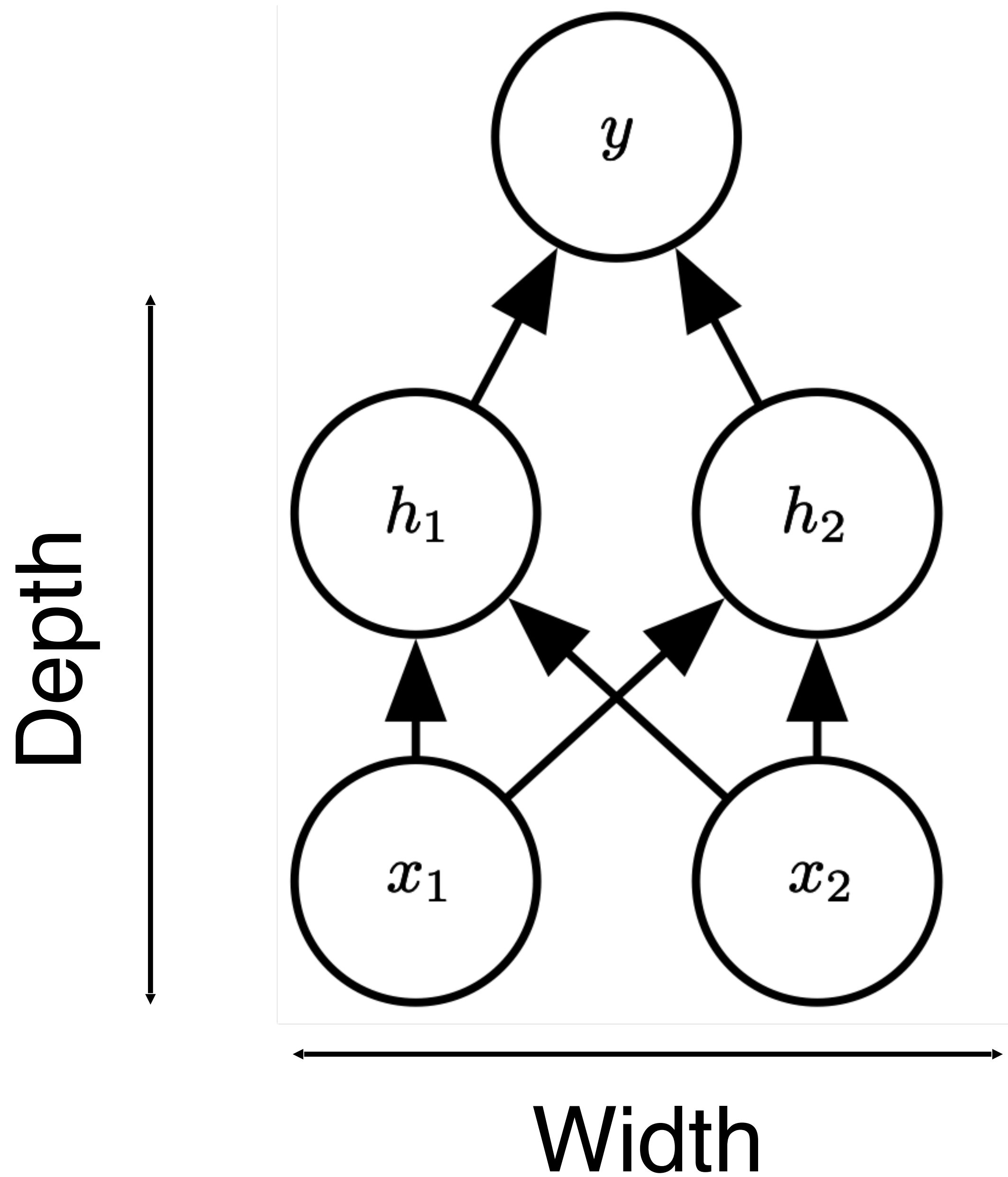
Hidden units

- Use ReLUs, 90% of the time
- For RNNs, see Chapter 10
- For some research projects, get creative
- Many hidden units perform comparably to ReLUs. New hidden units that perform comparably are rarely interesting.

Roadmap

- Example: Learning XOR
- Gradient-Based Learning
- Hidden Units
- Architecture Design
- Back-Propagation

Architecture Basics



Universal Approximator Theorem

- One hidden layer is enough to *represent* (not *learn*) an approximation of any function to an arbitrary degree of accuracy
- So why deeper?
  - Shallow net may need (exponentially) more width
  - Shallow net may overfit more

Exponential Representation Advantage of Depth

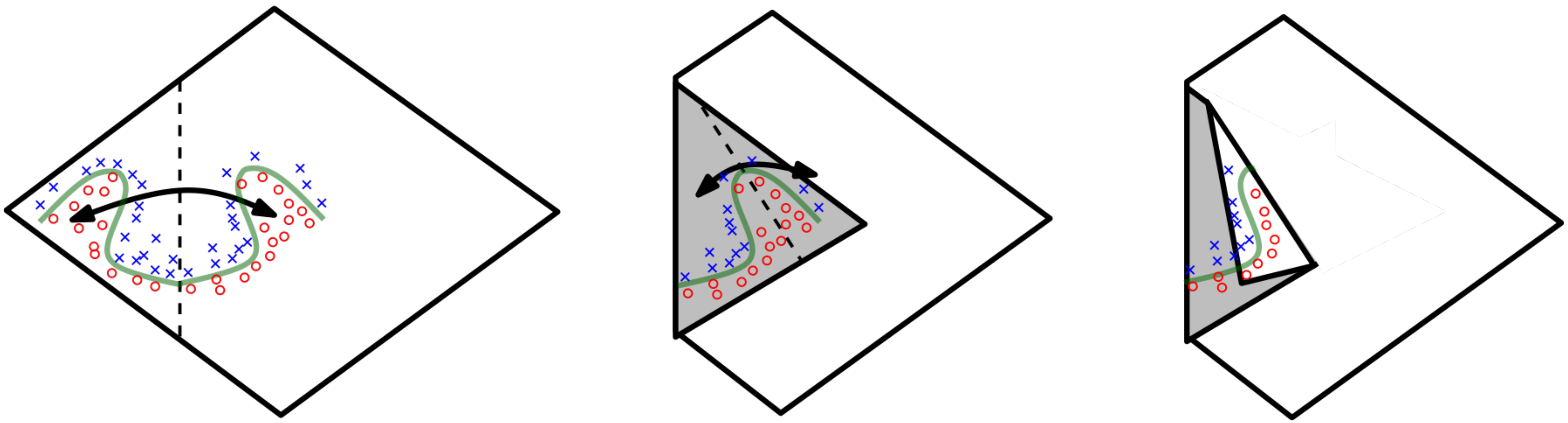


Figure 6.5

Better Generalization with Greater Depth

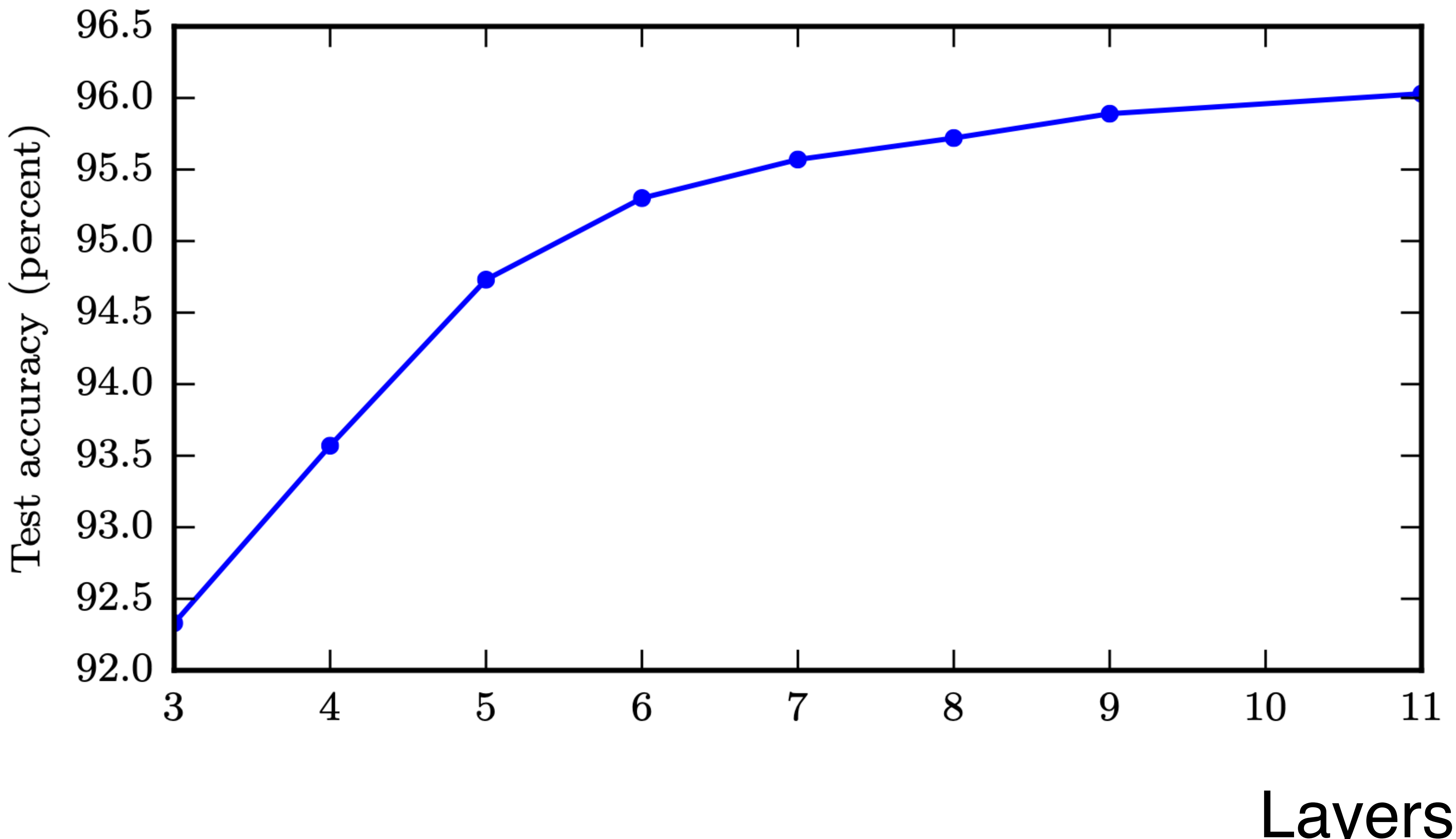


Figure 6.6

Large, Shallow Models Overfit More

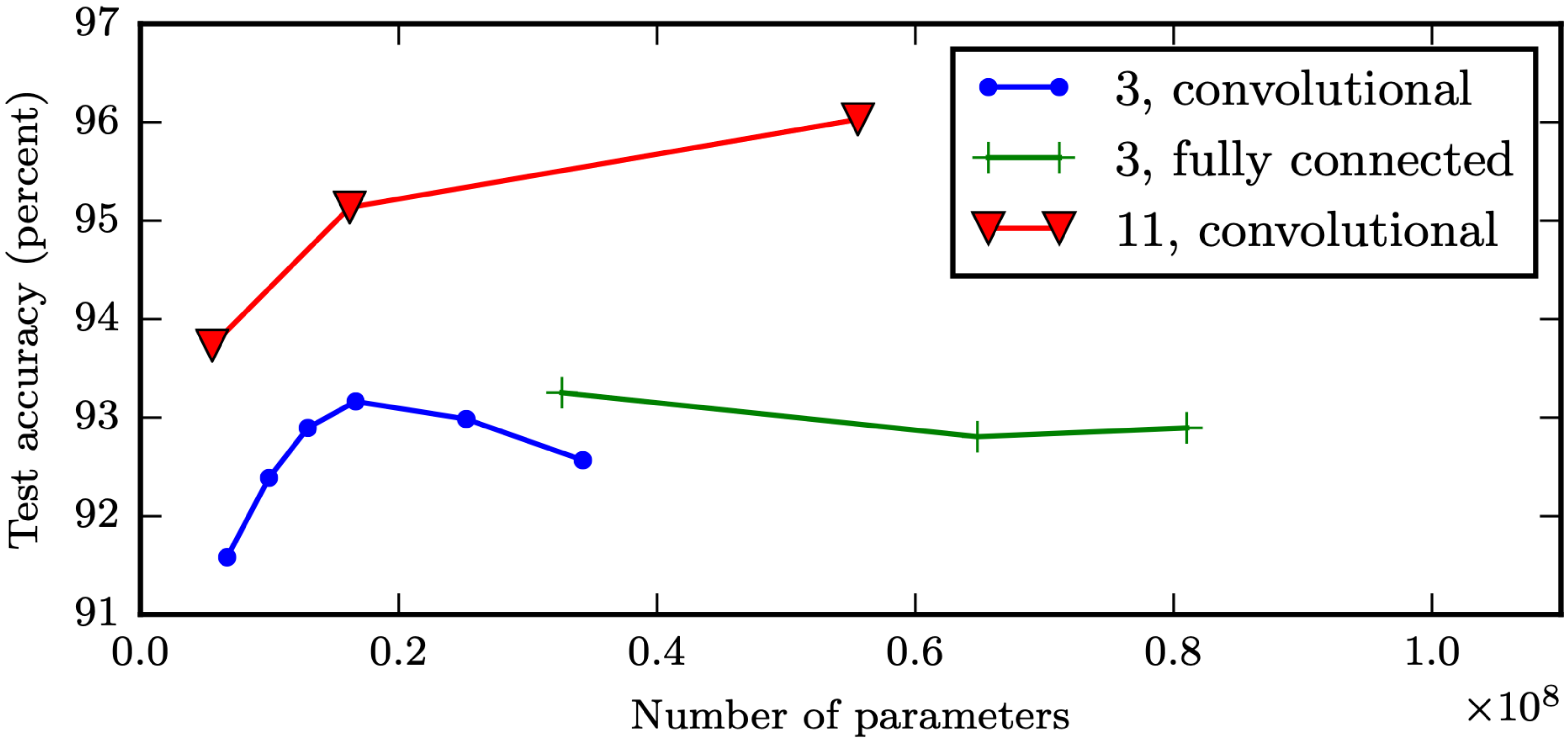


Figure 6.7

Roadmap

- Example: Learning XOR
- Gradient-Based Learning
- Hidden Units
- Architecture Design
- Back-Propagation

Back-Propagation

- Back-propagation is “just the chain rule” of calculus

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}. \tag{6.44}$$

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z, \tag{6.46}$$

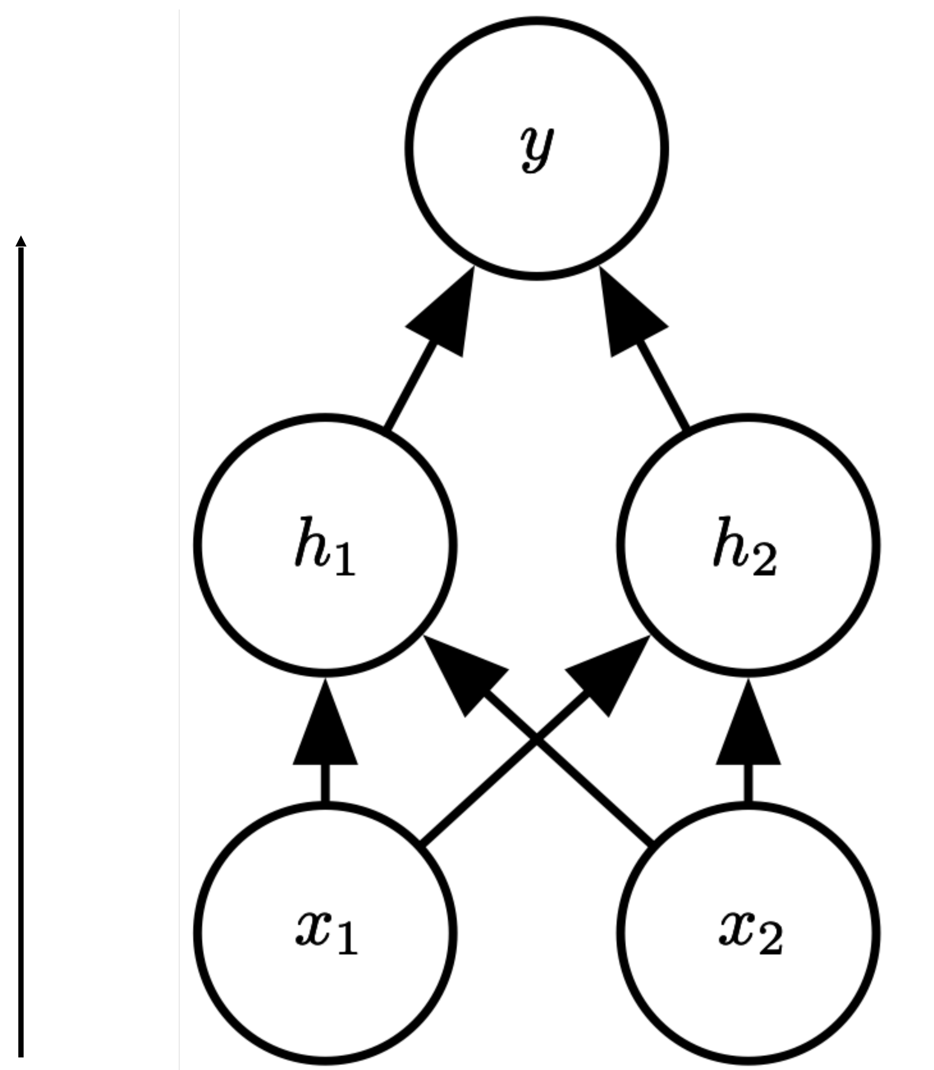
- But it’s a particular implementation of the chain rule
  - Uses dynamic programming (table filling)
  - Avoids recomputing repeated subexpressions
  - Speed vs memory tradeoff

Simple Back-Prop Example

Compute loss

Compute activations

Forward prop



Back-prop

Compute derivatives

Computation Graphs

Multiplication

Logistic regression

ReLU layer

Linear regression and weight decay

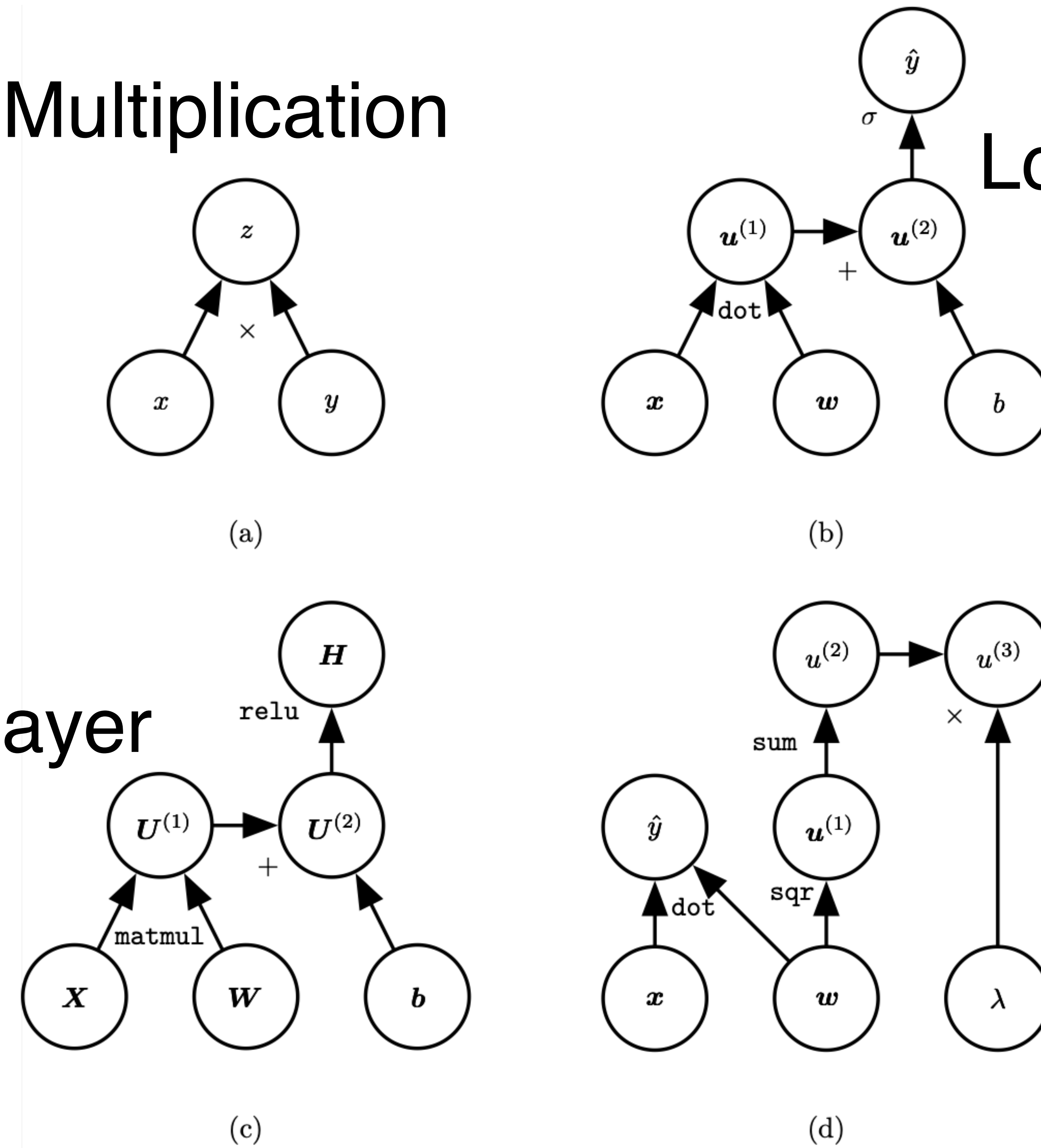
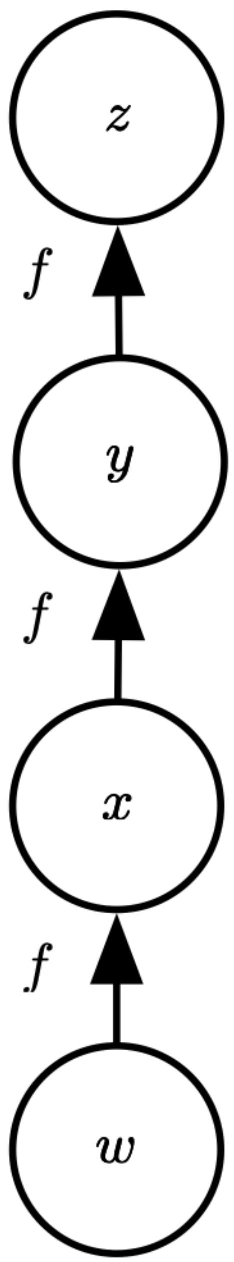


Figure 6.8

Repeated Subexpressions



$$\frac{\partial z}{\partial w} \tag{6.50}$$

$$= \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} \tag{6.51}$$

$$= f'(y) f'(x) f'(w) \tag{6.52}$$

$$= f'(f(f(w))) f'(f(w)) f'(w) \tag{6.53}$$

Back-prop avoids computing this twice

Figure 6.9

Symbol-to-Symbol Differentiation

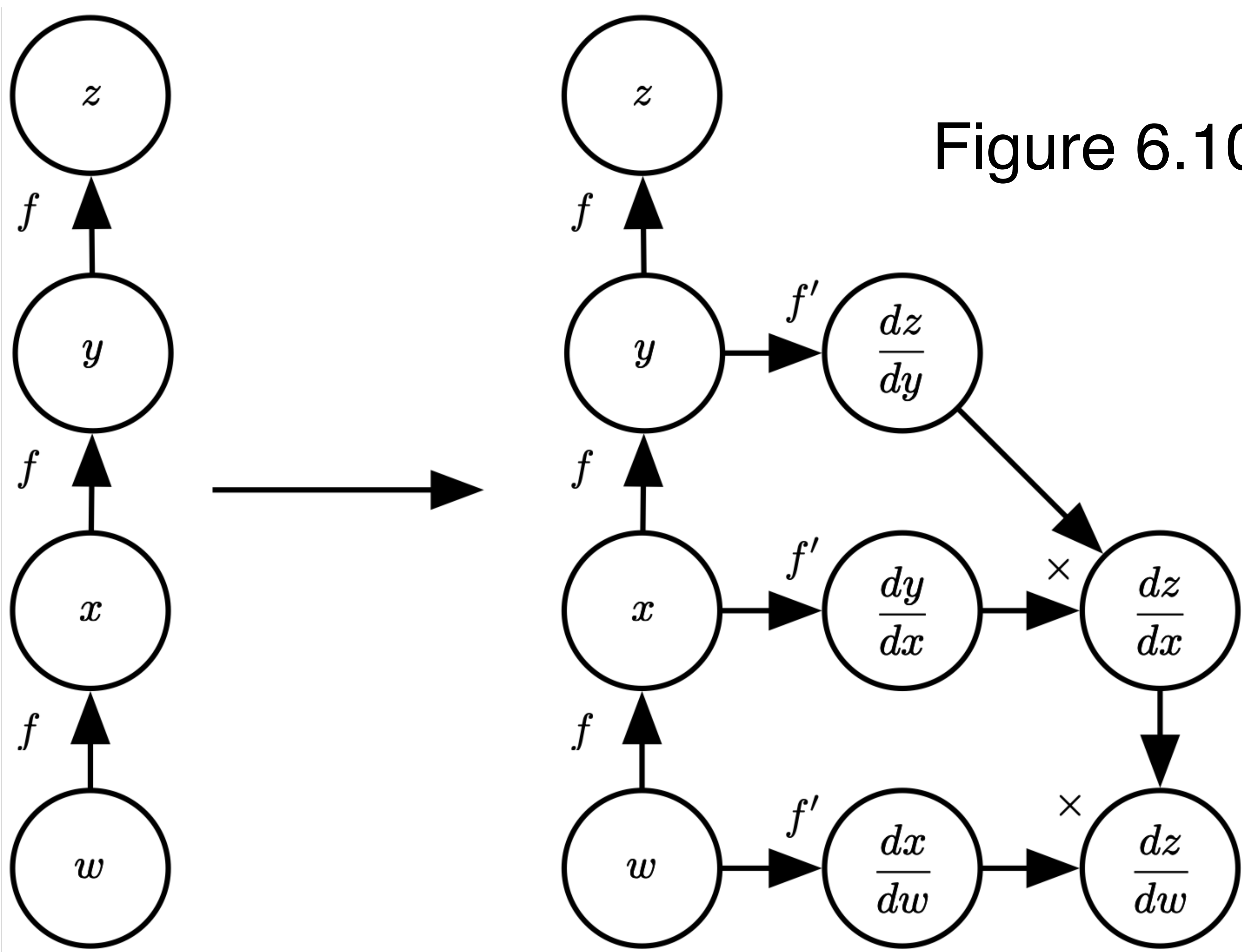


Figure 6.10

Neural Network Loss Function

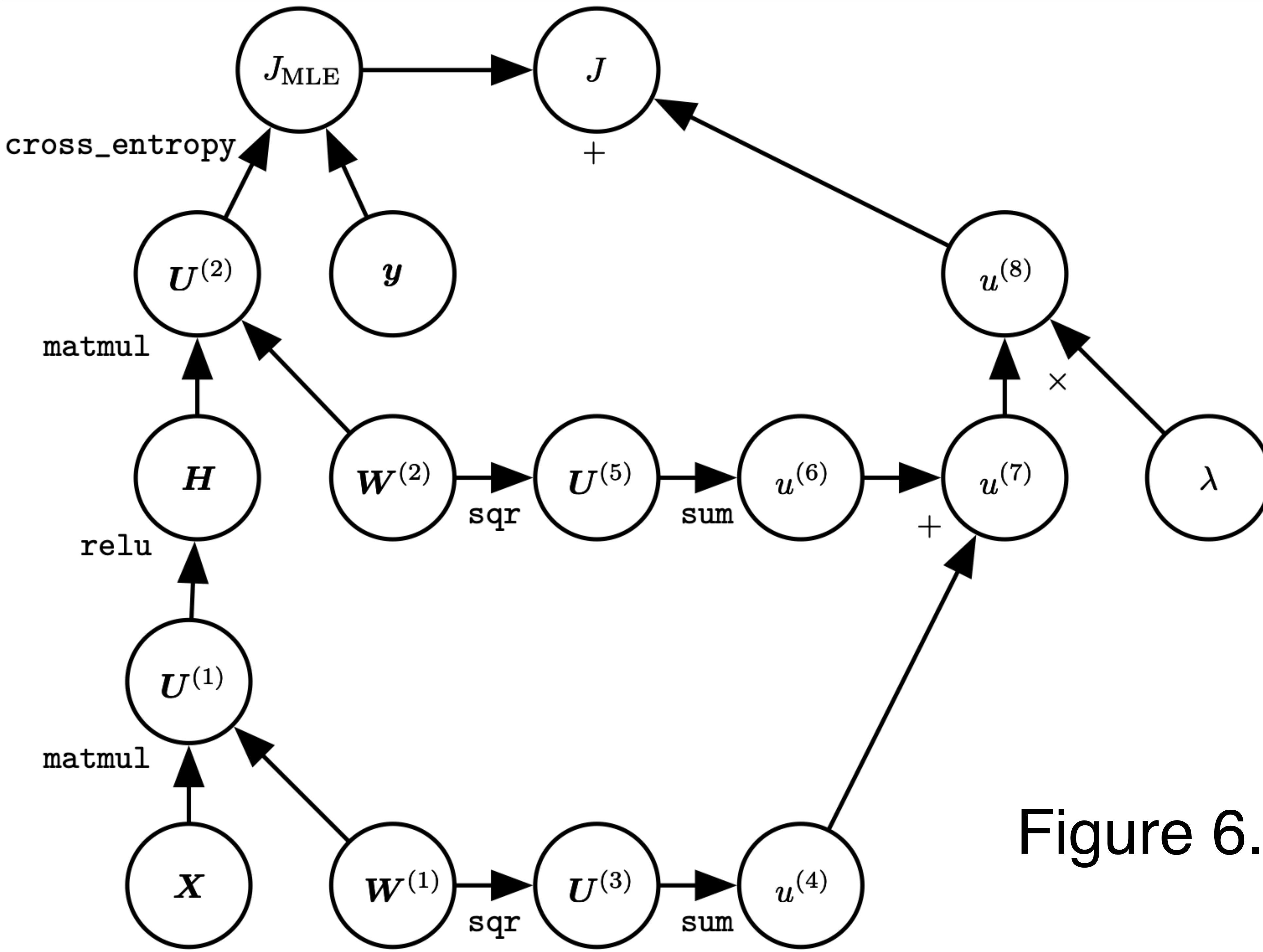


Figure 6.11

Hessian-vector Products

$$\boldsymbol{H}\boldsymbol{v} = \nabla_{\boldsymbol{x}} \left[\left(\nabla_{\boldsymbol{x}} f(x) \right)^{\top} \boldsymbol{v} \right]. \tag{6.59}$$

Hessian-vector Products cont…

Both of the gradient computations in this expression may be computed automatically by the appropriate software library. Note that the outer gradient expression takes the gradient of a function of the inner gradient expression.

If v is itself a vector produced by a computational graph, it is important to specify that the automatic differentiation software should not differentiate through the graph that produced v.

While computing the Hessian is usually not advisable, it is possible to do with Hessian vector products. One simply computes $He^{(i)}$ for all $i = 1, \ldots, n$, where $e^{(i)}$ is the one-hot vector with $e_i^{(i)} = 1$ and all other entries equal to 0.