CS 760: Machine Learning
Neural Networks II

Kirthi Kandasamy

University of Wisconsin-Madison

February 27, 2023

Logistics

- **Announcements**:
  - HW 3 was due today
  - HW 4 will be out soon

- **Midterm**
  - 90 minutes
  - Cheat sheet: one sheet of paper (no larger than Legal size), both sides, printed or hand-written
  - Will cover material up to next Monday's class.
  - Email me about alternative dates by tonight.

Outline

- **Neural Networks**
  - Introduction, Setup, Components, Activations
- **Training Neural Networks**
  - SGD, Computing Gradients, Backpropagation
- **Regularization**
  - Views, Data Augmentation, Other approaches

Outline

- **Neural Networks**
  - Introduction, Setup, Components, Activations
- **Training Neural Networks**
  - SGD, Computing Gradients, Backpropagation
- **Regularization**
  - Views, Data Augmentation, Other approaches

Multilayer Neural Network

- Input: two features from spectral analysis of a spoken sound
- Output: vowel sound occurring in the context "h__d"

output units

hidden units

input units

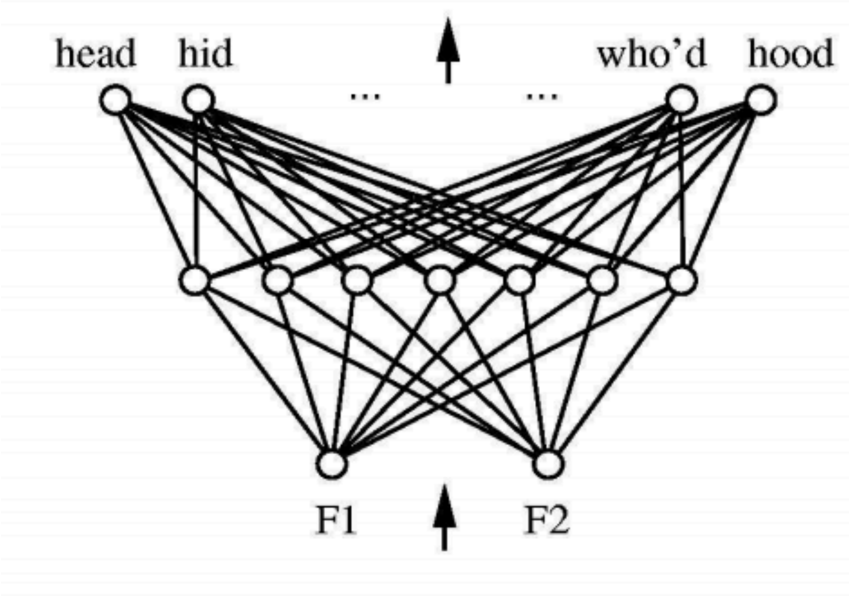


figure from Huang & Lippmann, *NIPS* 1988

Neural Network **Decision Regions**

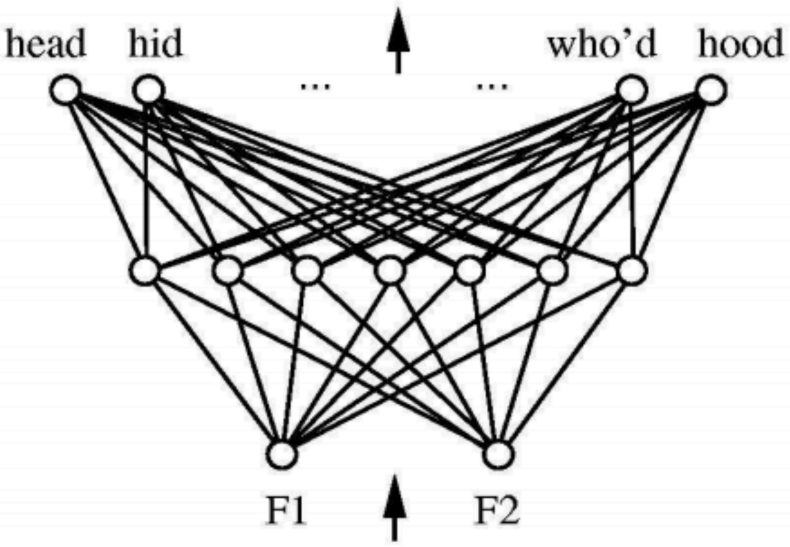
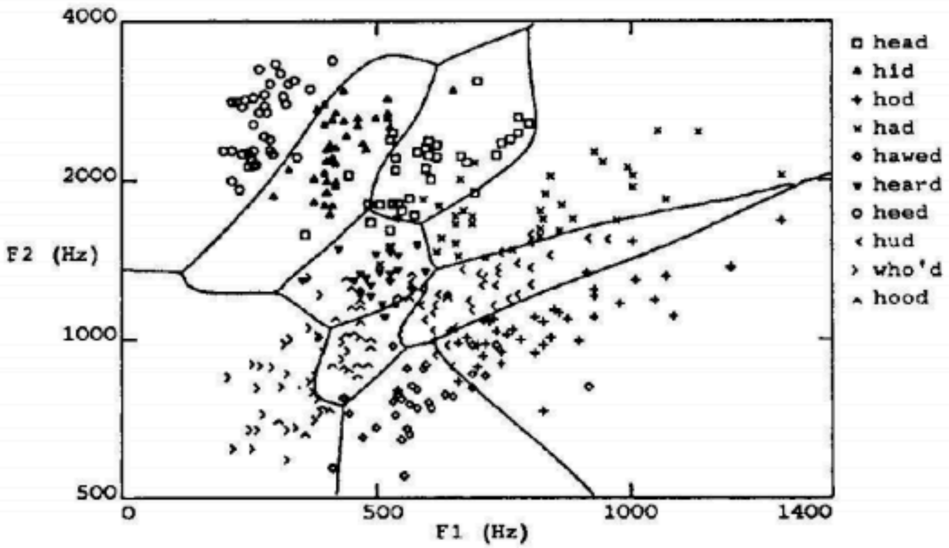
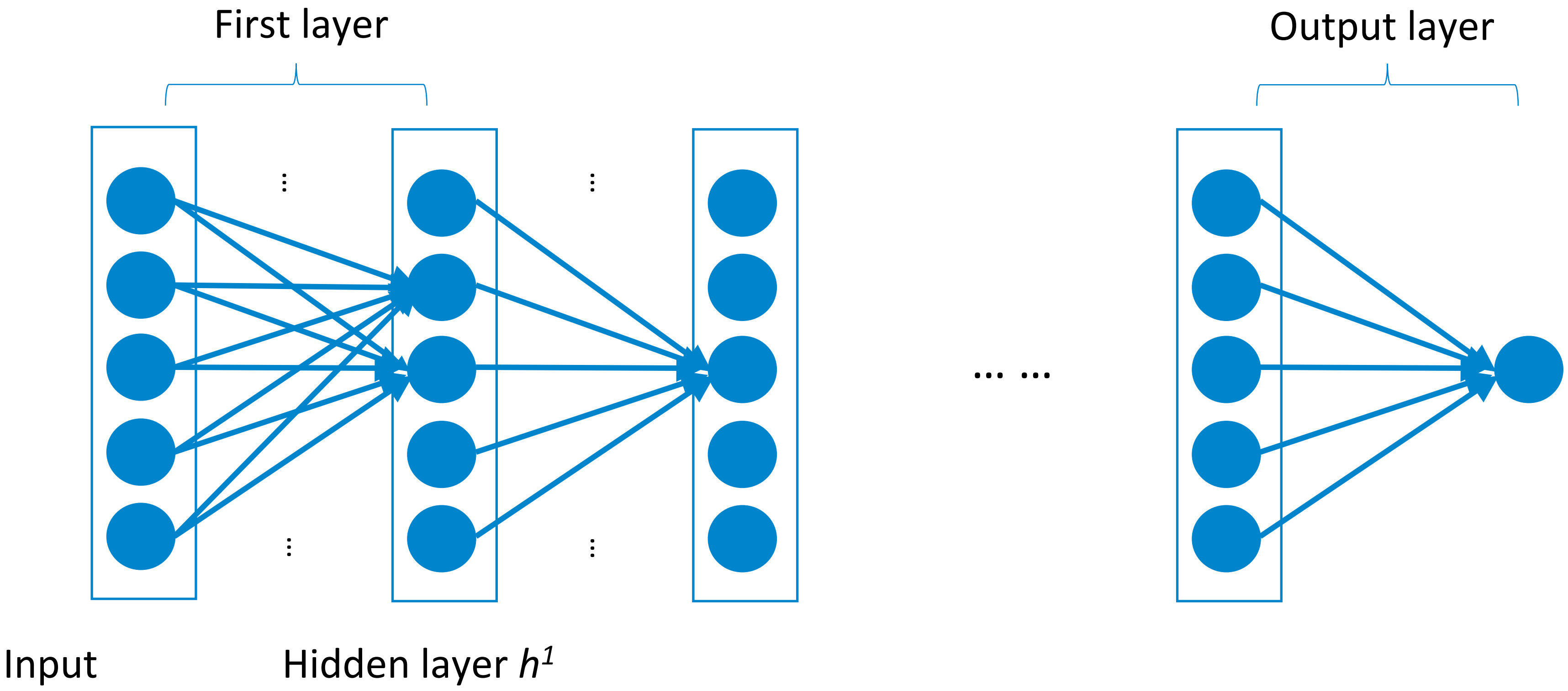


Figure from Huang & Lippmann, *NIPS* 1988



Neural Network Components

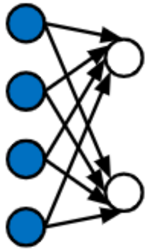
An $(L+1)$-layer network



Feature Encoding for NNs

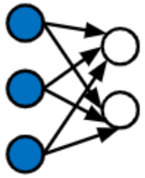
- Nominal features usually a one hot encoding

$$A = \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad C = \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad G = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad T = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$



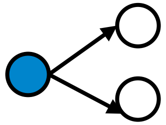
- Ordinal features: use a *thermometer* encoding

$$\text{small}= \begin{bmatrix}1\\0\\0\end{bmatrix} \quad \text{medium}= \begin{bmatrix}1\\1\\0\end{bmatrix} \quad \text{large}= \begin{bmatrix}1\\1\\1\end{bmatrix}$$



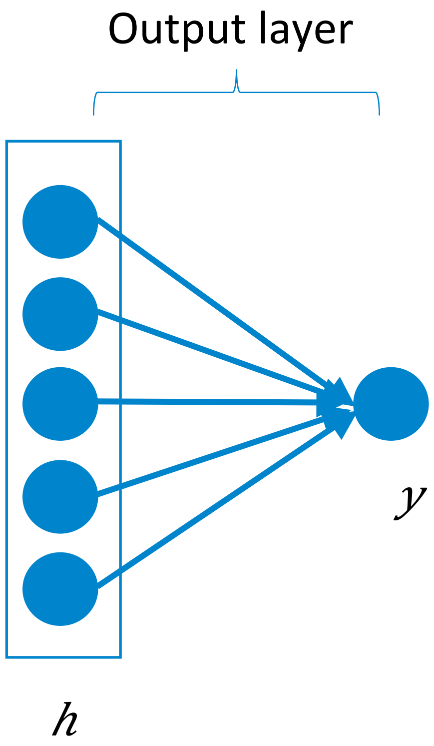
- Real-valued features use individual input units (may want to scale/normalize them first though)

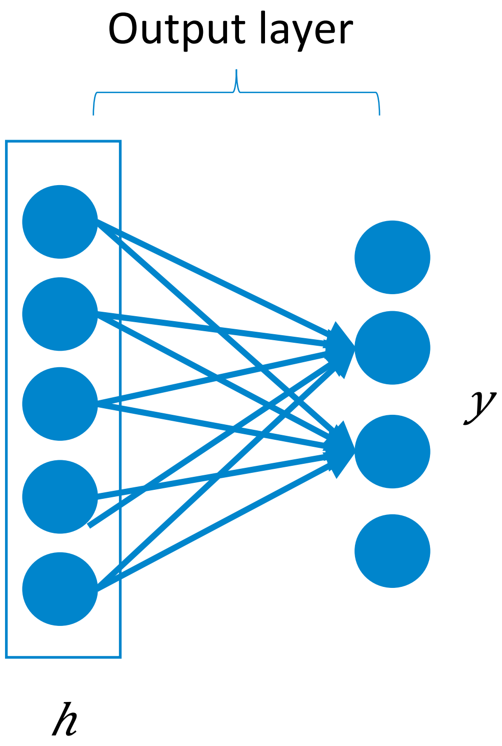
$$\text{precipitation} = \begin{bmatrix}0.68\end{bmatrix}$$



Output Layer: Examples

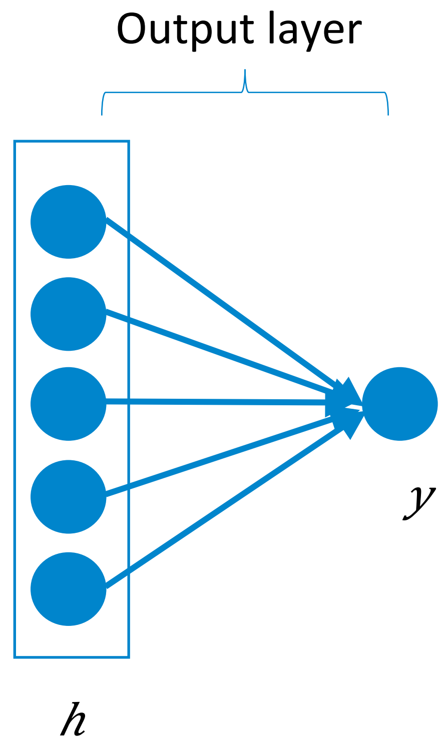
- Regression: $y = w^\top h + b$
  - Linear units: no nonlinearity
- Multi-dimensional regression: $Y = W^\top h + b$
  - Linear units: no nonlinearity

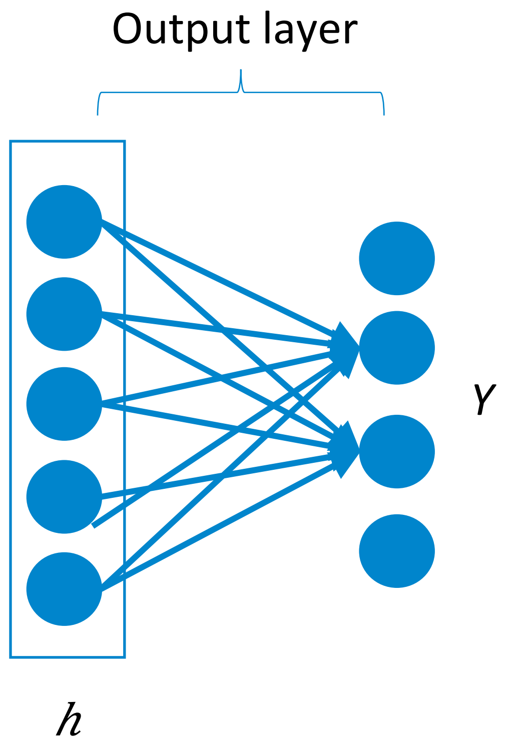




Output Layer: Examples

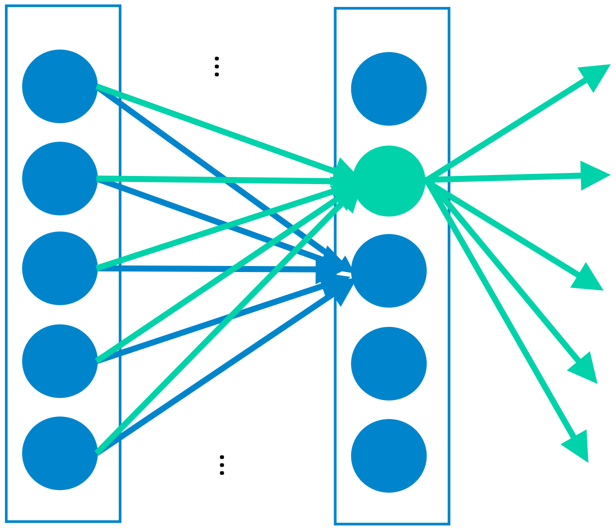
- Binary classification: $y = \text{sigmoid}(w^\top h + b)$
  - Corresponds to using logistic regression on
- Multiclass classification: $Y = \text{sigmoid}(W^\top h + b)$





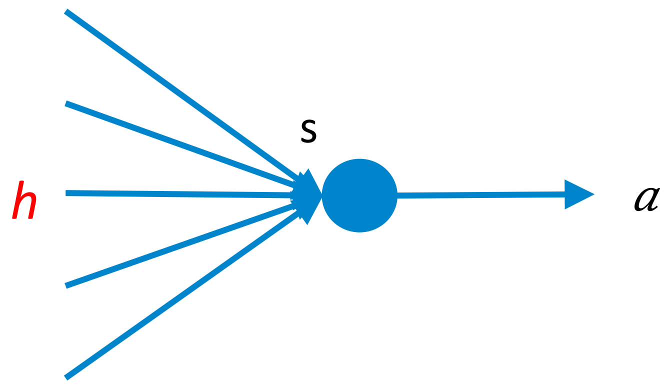
Hidden Layers

- Neuron takes weighted linear combination of the previous representation layer
  - Outputs one value for the next layer



Hidden Layers

- Outputs $a = s(w^\top h + b)$



- Typical activation function
  - Threshold: $s(z) = \mathbb{1}(z \geq 0)$
  - Sigmoid: $s(z) = \sigma(z) = 1/(1 + e^{-z})$
  - Tanh: $s(z) = 2\sigma(2z) - 1$

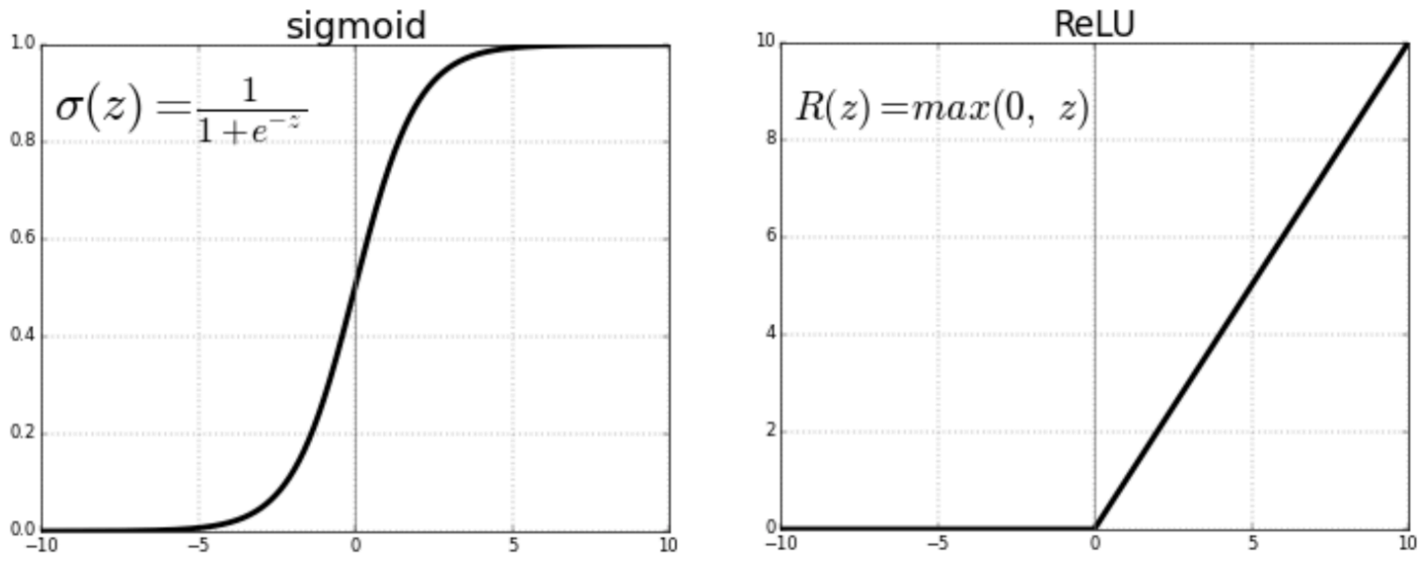
- Why not **linear activation** functions?
  - Model would be linear.

More on Activations

- Outputs $a = s(w^\top h + b)$

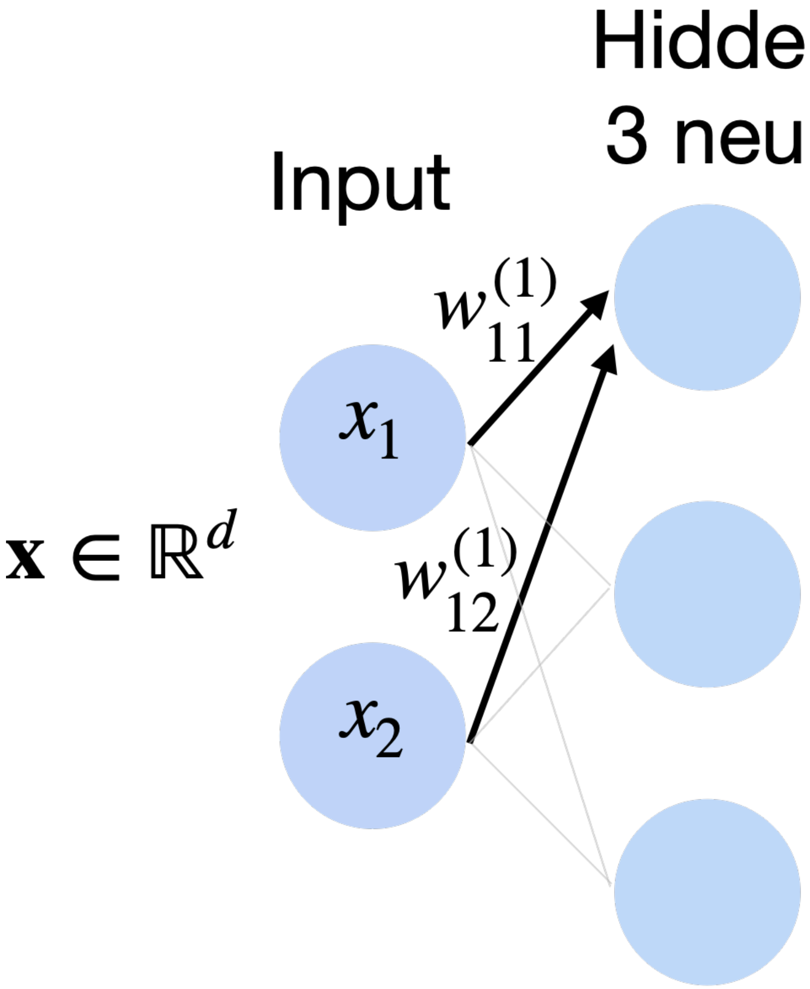
  Activation (a), Weight (w), Bias (b)

- Consider **gradients**… saturating vs. nonsaturating



MLPs: Multilayer Perceptron

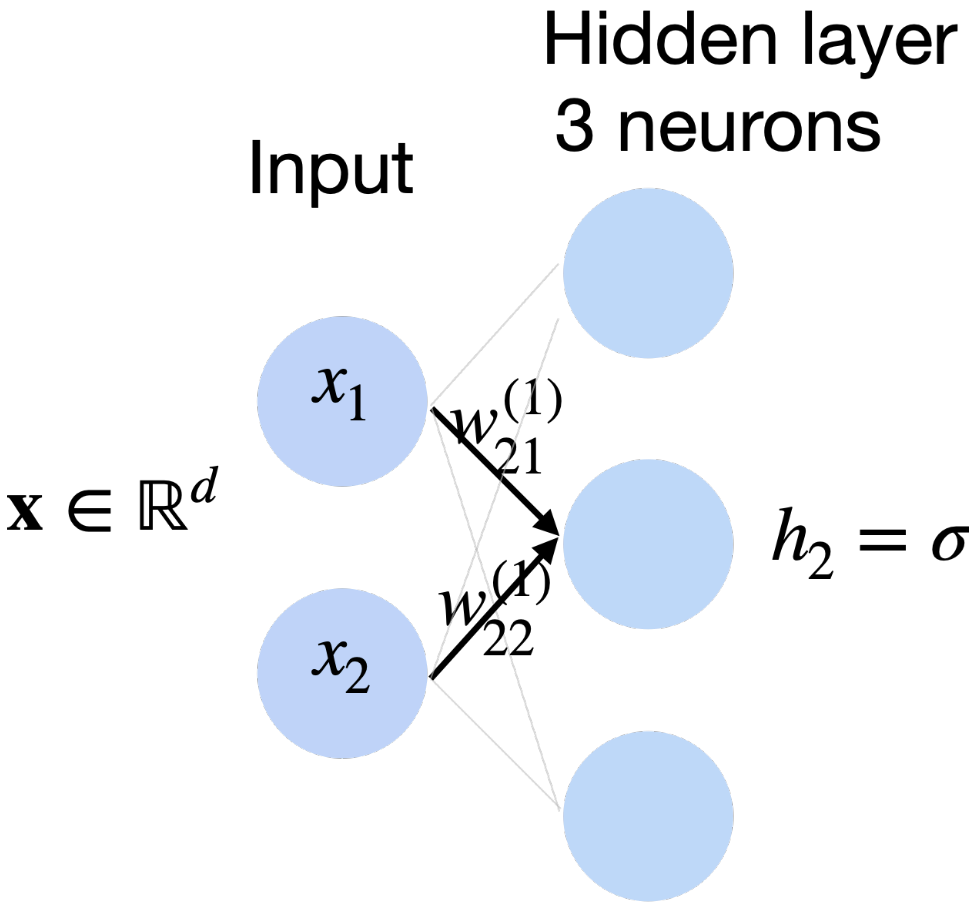
- **Ex**: 1 hidden layer, 1 output layer: depth 2



$$h_1 = \sigma\left(\sum_{i=1}^{d} x_i w_{1i}^{(1)} + b_1\right)$$

MLPs: Multilayer Perceptron

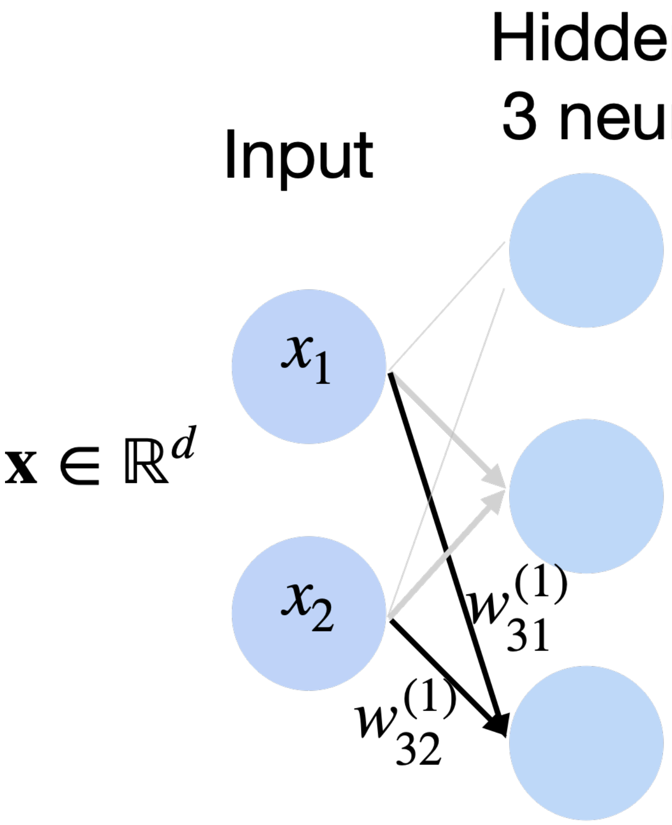
- **Ex**: 1 hidden layer, 1 output layer: depth 2



$$h_2 = \sigma\left(\sum_{i=1}^{d} x_i w_{2i}^{(1)} + b_2\right)$$

MLPs: Multilayer Perceptron

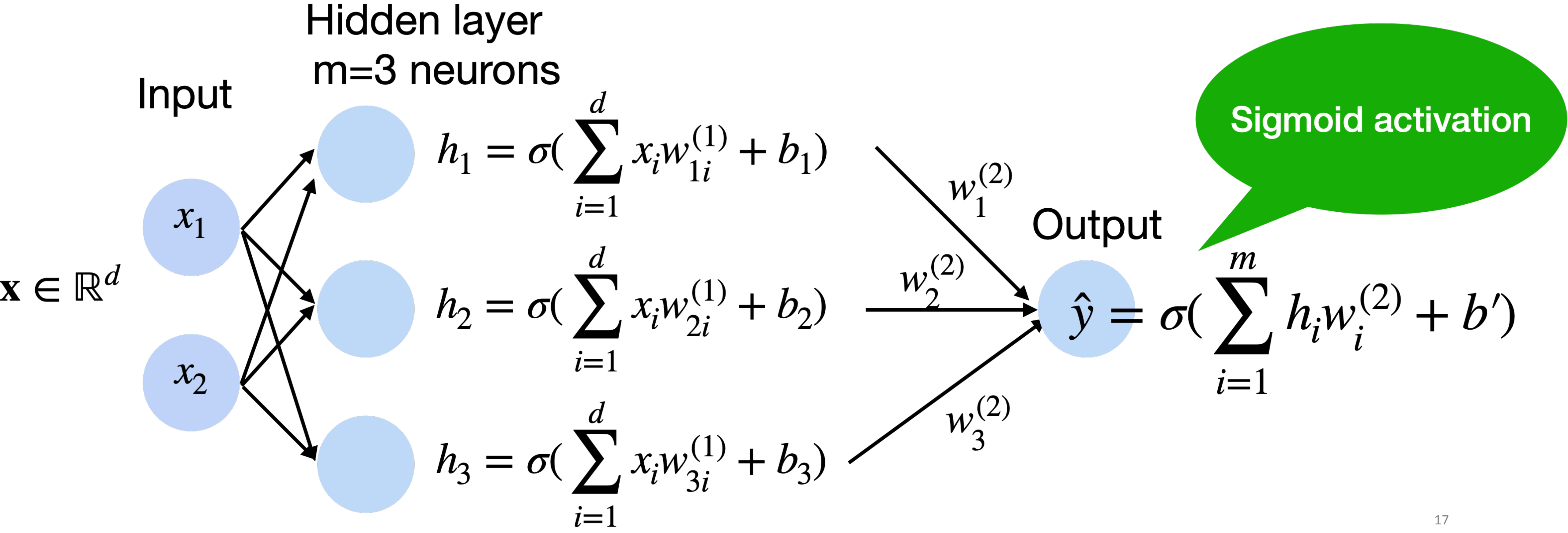
- **Ex**: 1 hidden layer, 1 output layer: depth 2



$$h_3 = \sigma\left(\sum_{i=1}^{d} x_i w_{3i}^{(1)} + b_3\right)$$

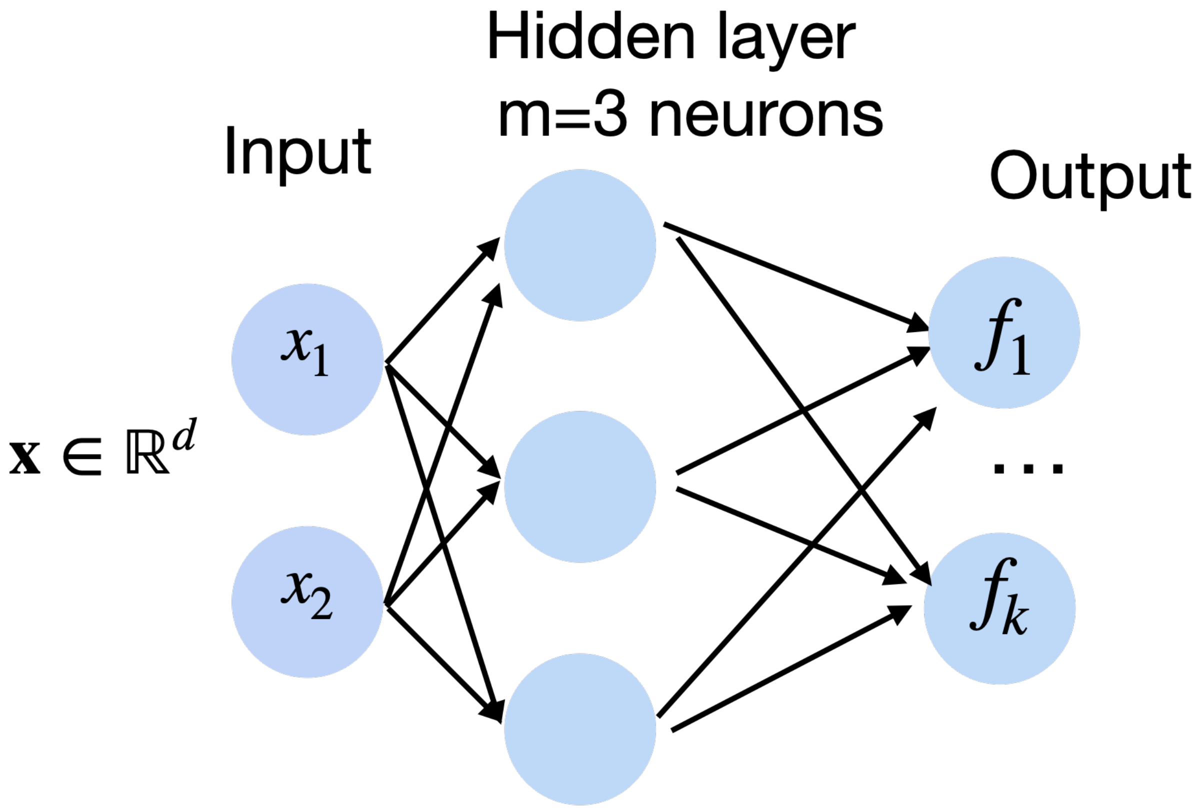
MLPs: Multilayer Perceptron

- **Ex**: 1 hidden layer, 1 output layer: depth 2



Multiclass Classification Output

- Create k output units
- Use softmax (just like logistic regression)

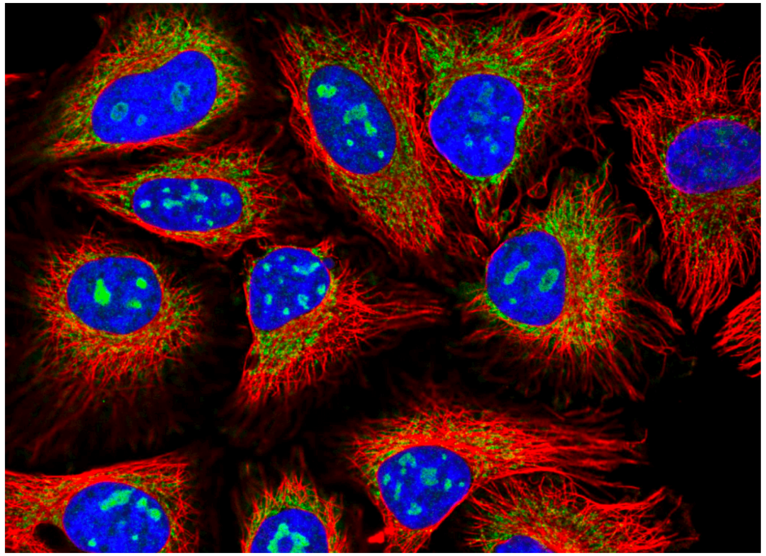


$$p(y \mid \mathbf{x}) = \text{softmax}(f)$$

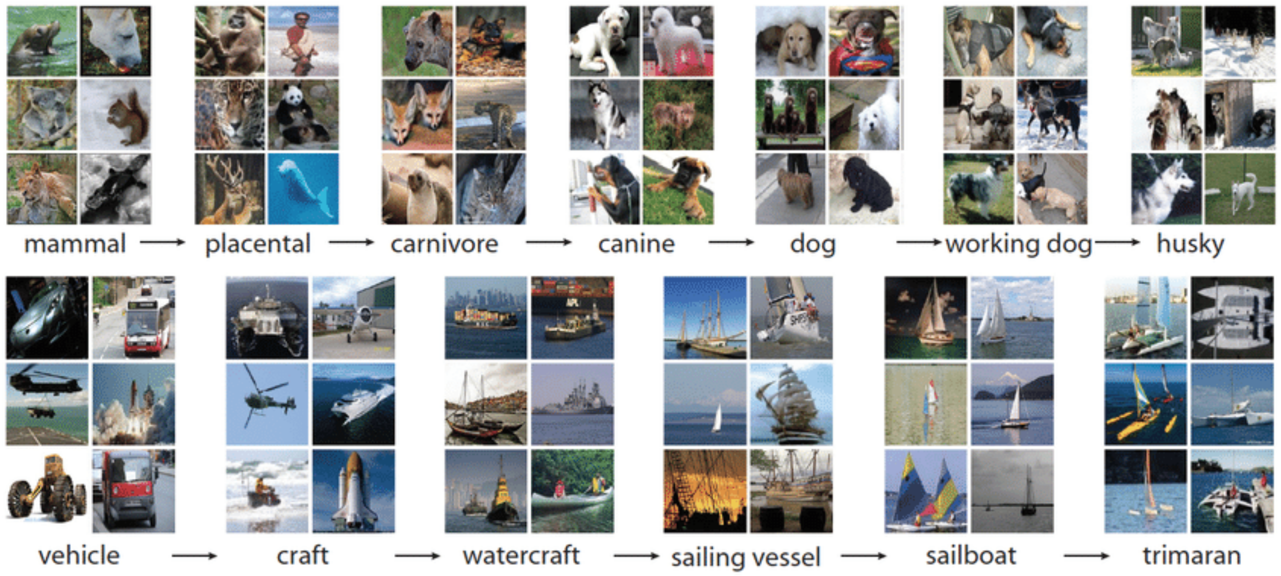
$$= \frac{\exp f_y(x)}{\sum_i^k \exp f_i(x)}$$

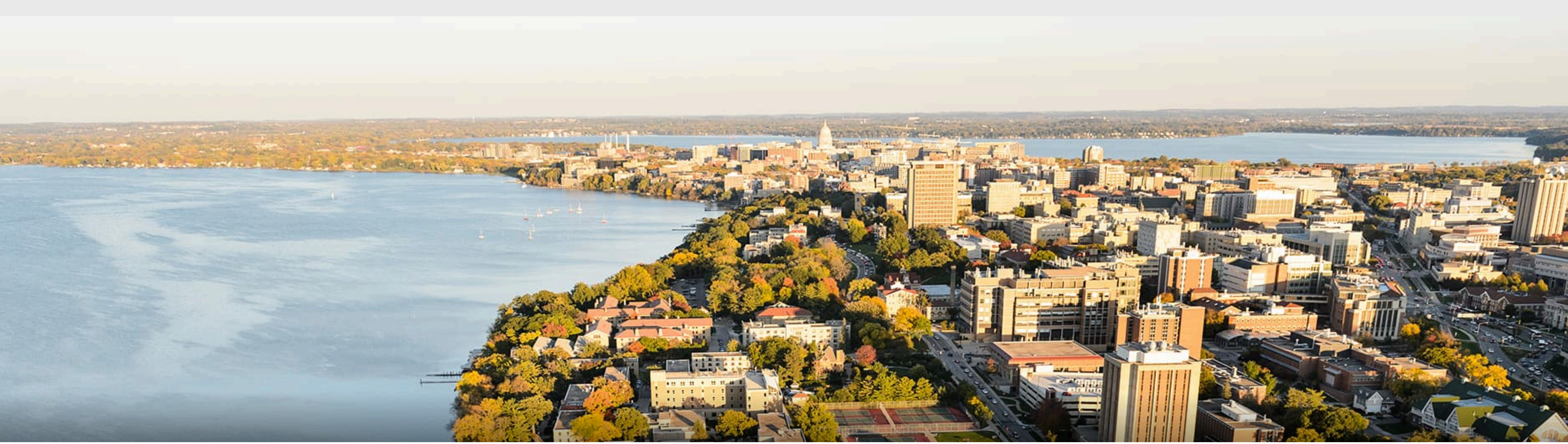
Multiclass Classification Examples

- Protein classification (Kaggle challenge)
- ImageNet



0. Nucleoplasm
1. Nuclear membrane
2. Nucleoli
3. Nucleoli fibrillar
4. Nuclear speckles





Break & Quiz

Q: Select the correct option.

A. *The more hidden-layer units a Neural Network has, the better it can predict desired outputs for new inputs that it was not trained with.*

B. *A 3-layers Neural Network with 5 neurons in the input and hidden representations and 1 neuron in the output has a total of 55 connections.*

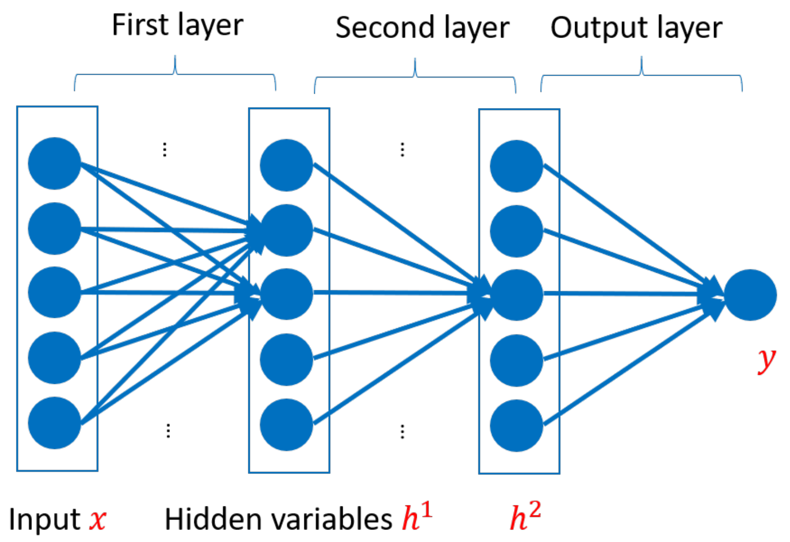
1. Both statements are true.
2. Both statements are false.
3. Statement A is true, Statement B is false.
4. Statement B is true, Statement A is false.

Q: Select the correct option.

A. *The more hidden-layer units a Neural Network has, the better it can predict desired outputs for new inputs that it was not trained with.*

B. *A 3-layers Neural Network with 5 neurons in the input and hidden representations and 1 neuron in the output has a total of 55 connections.*

1. Both statements are true.
2. Both statements are false.
3. Statement A is true, Statement B is false.
4. Statement B is true, Statement A is false.



Outline

- **Neural Networks**
  - Introduction, Setup, Components, Activations
- **Training Neural Networks**
  - SGD, Computing Gradients, Backpropagation
- **Regularization**
  - Views, Data Augmentation, Other approaches

Training Neural Networks

- Training the usual way. Pick a loss and optimize
- **Example**: 2 scalar weights

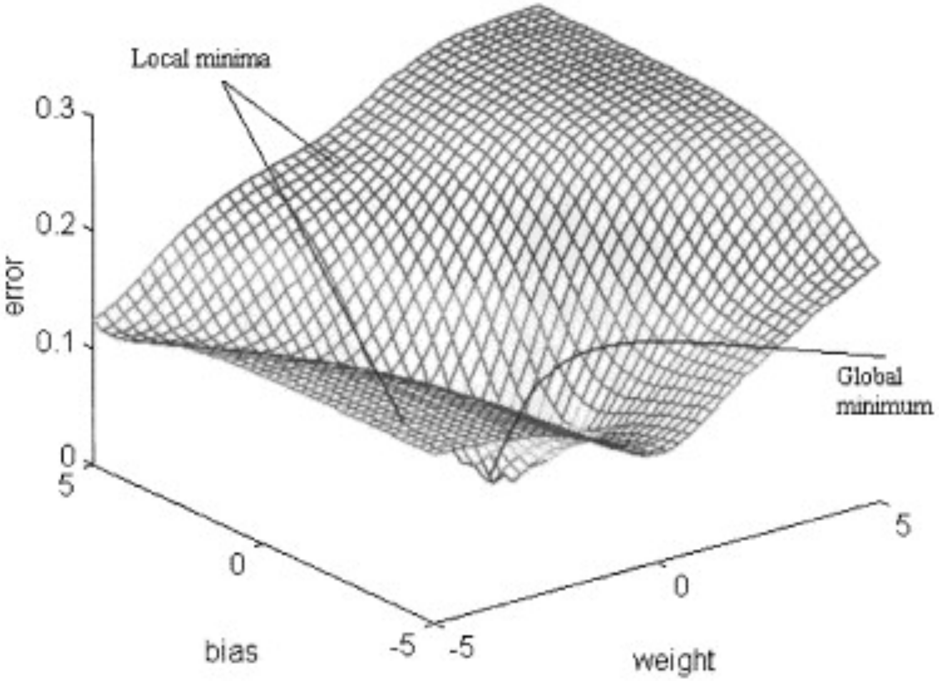


figure from Cho & Chow, *Neurocomputing* 1999

Training Neural Networks: SGD

- Algorithm:
  - Get $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$
  - Initialize weights
  - Until stopping criteria met,
    - Sample training point. $(x^{(i)}, y^{(i)})$ without replacemet
    - Compute: $f_{\text{network}}(x^{(i)})$ ⟵ **Forward Pass**
    - Compute gradient: $\nabla L^{(i)}(w) = \left[\frac{\partial L^{(d)}}{\partial w_0}, \frac{\partial L^{(d)}}{\partial w_1}, \ldots, \frac{\partial L^{(d)}}{\partial w_m}\right]^T$ ⟵ **Backward Pass**
    - Update weights: $w \leftarrow w - \alpha \nabla L^{(i)}(w)$

Training Neural Networks: Minibatch SGD

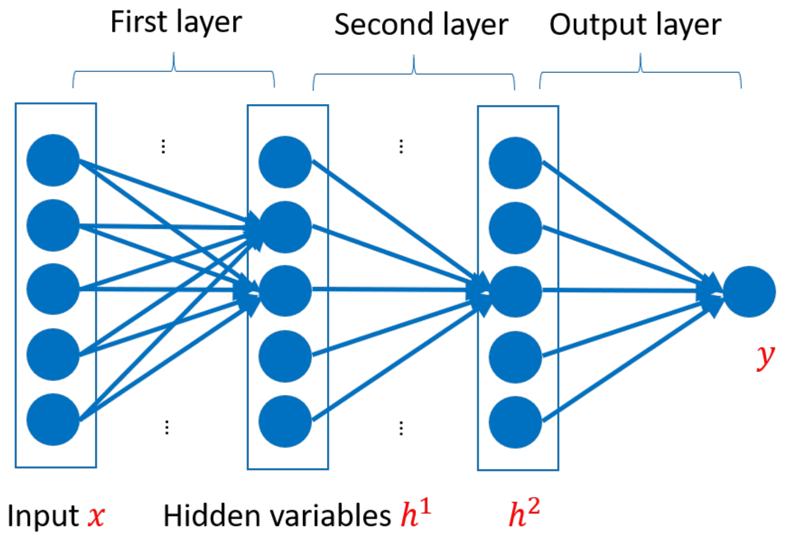
- Algorithm:
  - Get $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$
  - Initialize weights
  - Until stopping criteria met,
    - Sample b points $j_1, j_2, \ldots, j_b$
    - Compute: $f_{\text{network}}(x^{(j_1)}), \ldots, f_{\text{network}}(x^{(j_b)})$ ⟵ **Forward Pass**
    - Compute gradients: $\nabla L^{(j_1)}(w), \ldots, \nabla L^{(j_b)}(w)$ ⟵ **Backward Pass**
    - Update weights:

$$w \leftarrow w - \frac{\alpha}{b} \sum_{k=1}^{b} \nabla L^{(j_k)}(w)$$

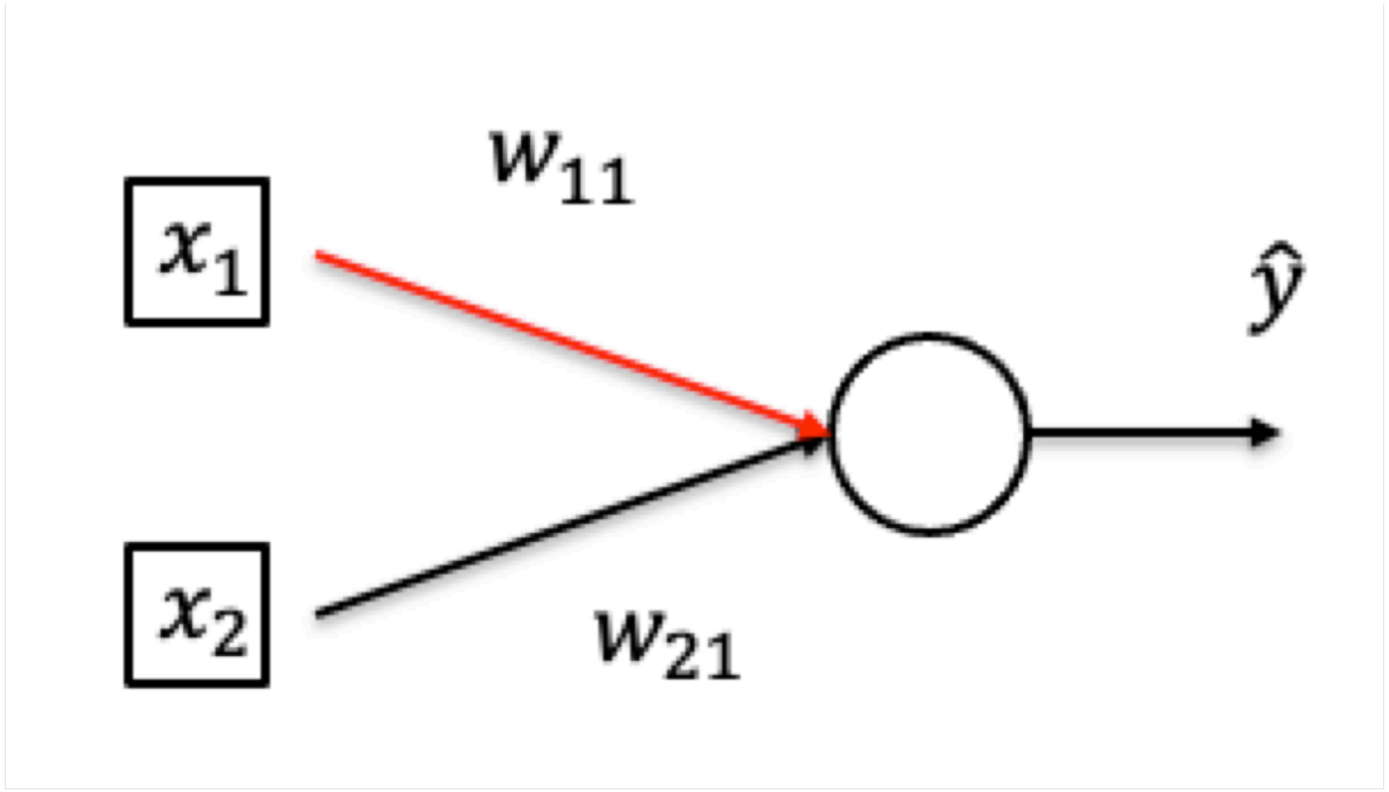
Training Neural Networks: Chain Rule

- Will need to compute terms like: $$\frac{\partial L}{\partial w_1}$$
  - But, L is a composition of:
    - Loss with output y
    - Output itself a composition of softmax with outer layer
    - Outer layer a combination of outputs from previous layer
    - Outputs from prev. layer a composition of activations and linear functions…
- Need the **chain rule**!
  - Suppose $L = L(g_1, \ldots, g_k) \quad g_j = g_j(w_1, \ldots, w_p)$
  - Then,

$$\frac{\partial L}{\partial w_i} = \sum_{j=1}^{k} \frac{\partial L}{\partial g_j} \frac{\partial g_j}{\partial w_i}$$

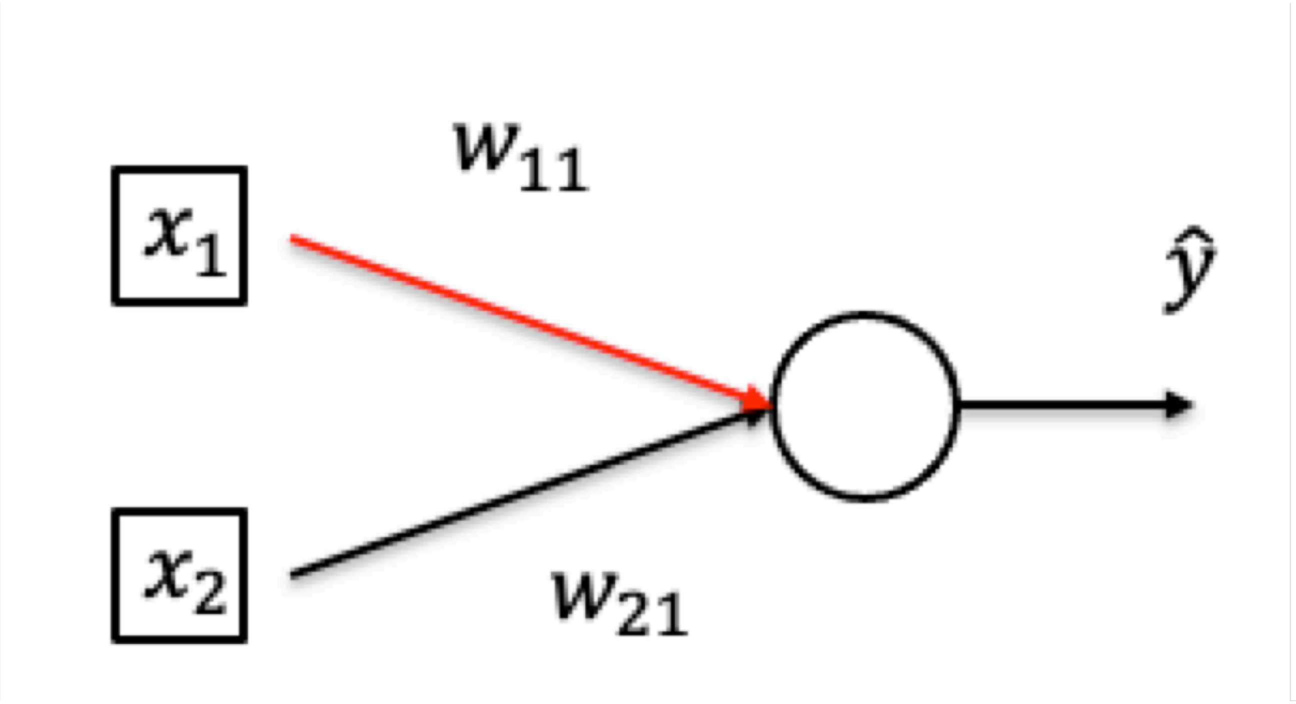


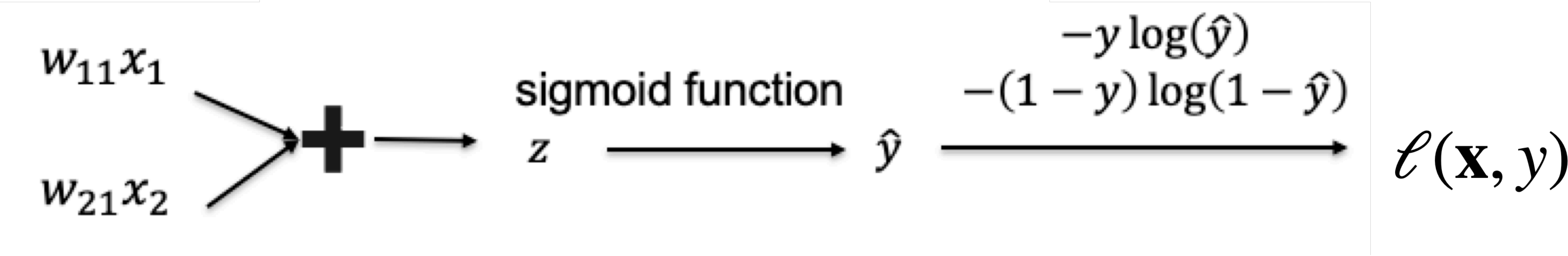
Computing Gradients



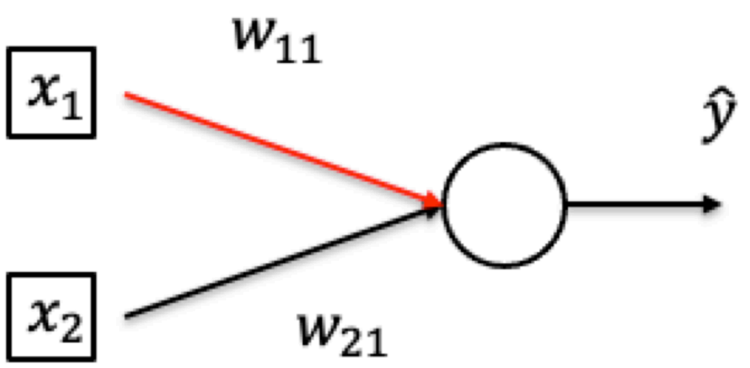
- Want to compute $\dfrac{\partial \ell(\mathbf{x}, y)}{\partial w_{11}}$

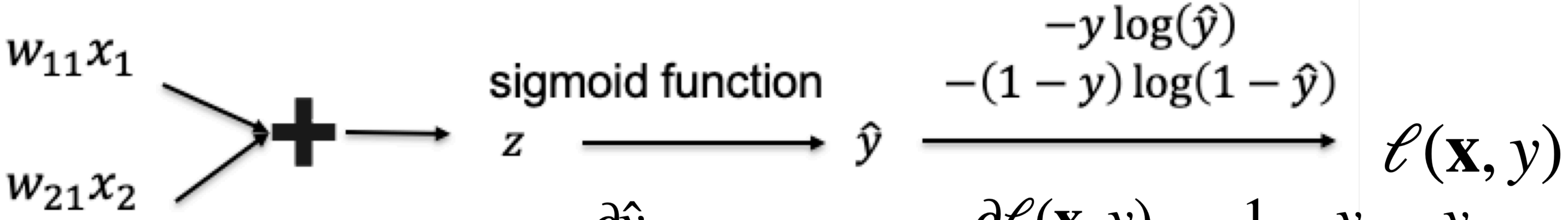
Computing Gradients





Computing Gradients



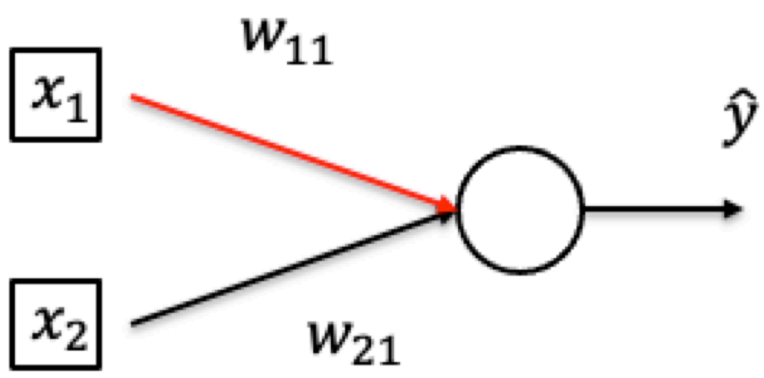


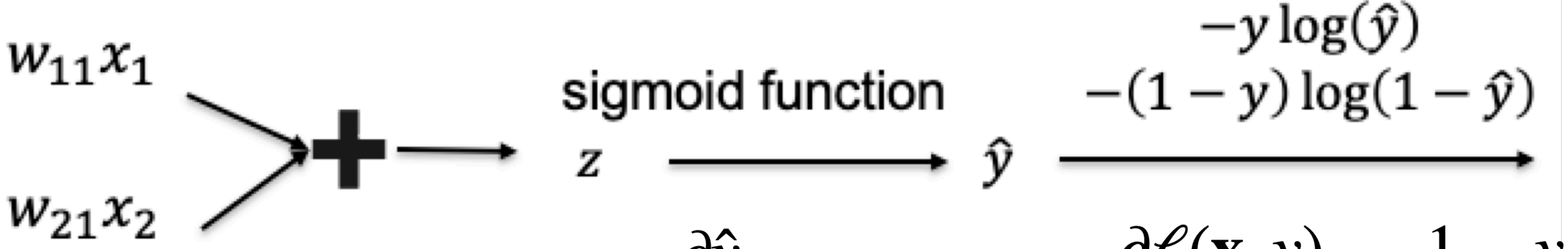
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) \qquad \frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_{11}}$$

Computing Gradients





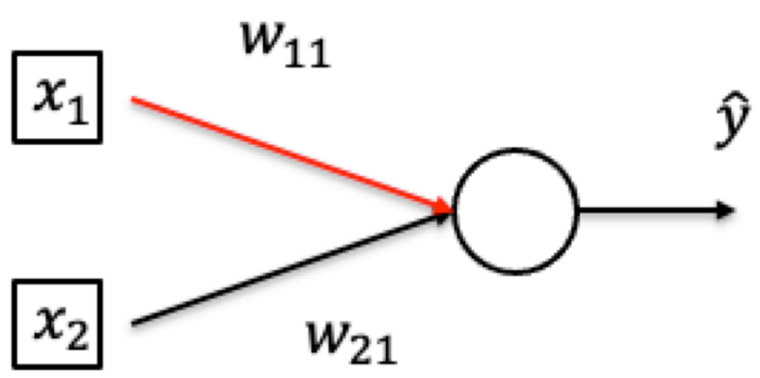
$\ell(\mathbf{x}, y)$

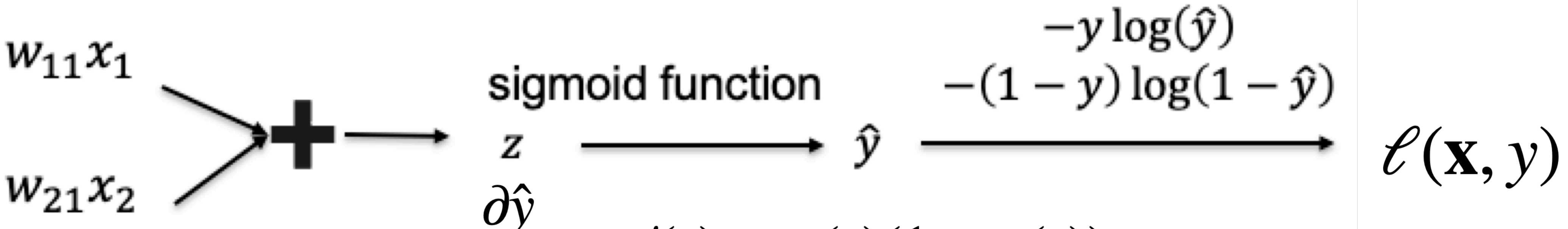
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z)$$

$$\frac{\partial \ell(\mathbf{x}, y)}{\partial \hat{y}} = \frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}$$

- By chain rule: $$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} x_1$$

Computing Gradients



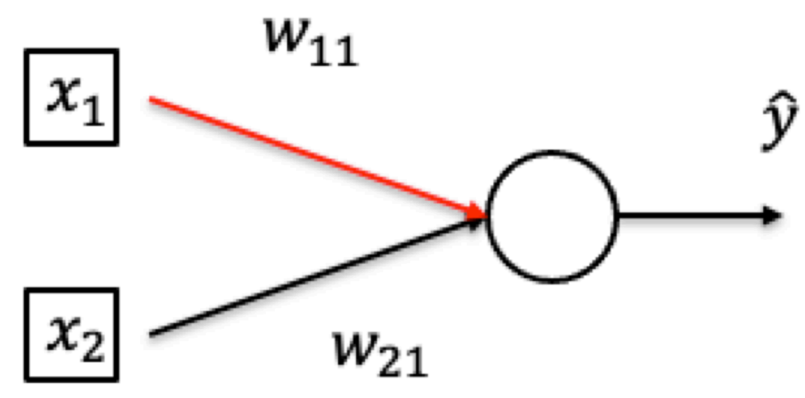


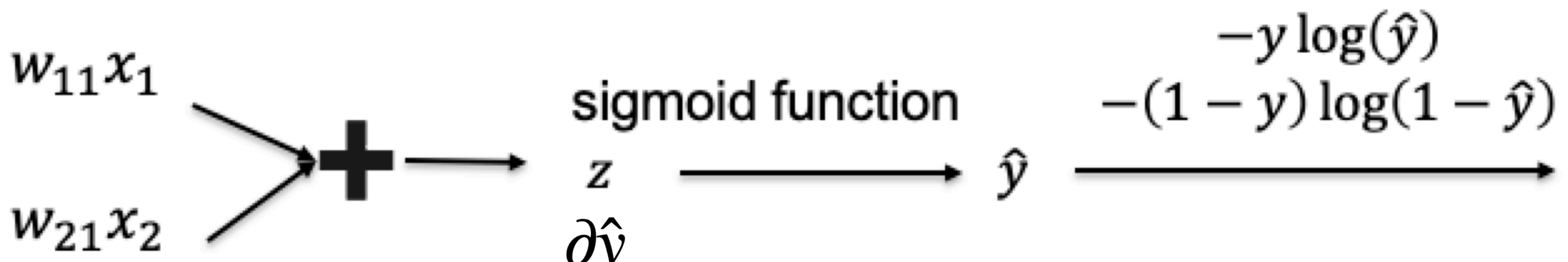
$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial \hat{y}}\ \hat{y}(1-\hat{y})x_1$$

Computing Gradients



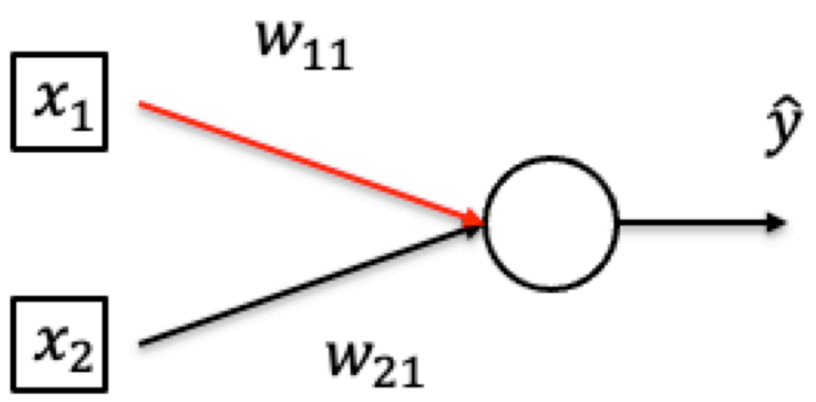


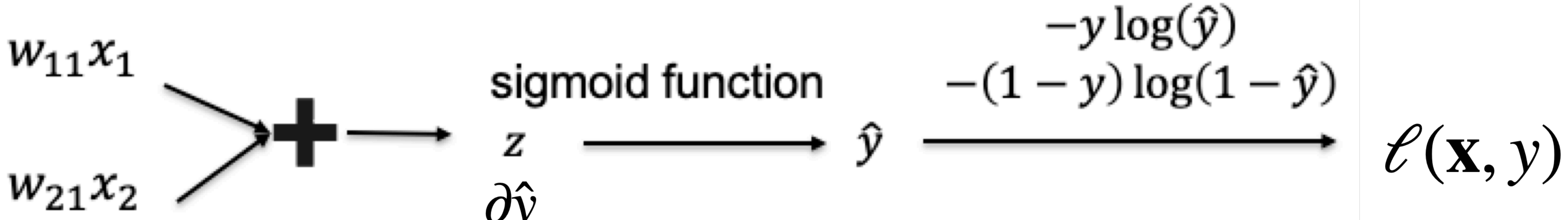
$\ell(\mathbf{x}, y)$

$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

- By chain rule: $$\frac{\partial l}{\partial w_{11}} = \left(\frac{1-y}{1-\hat{y}} - \frac{y}{\hat{y}}\right)\hat{y}(1-\hat{y})x_1$$

Computing Gradients

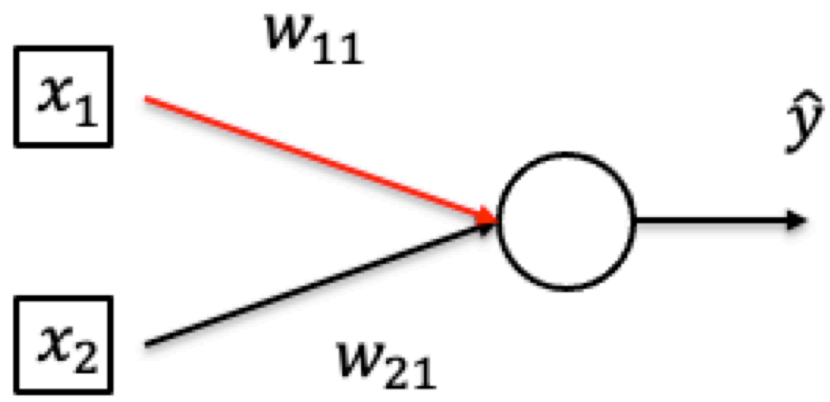


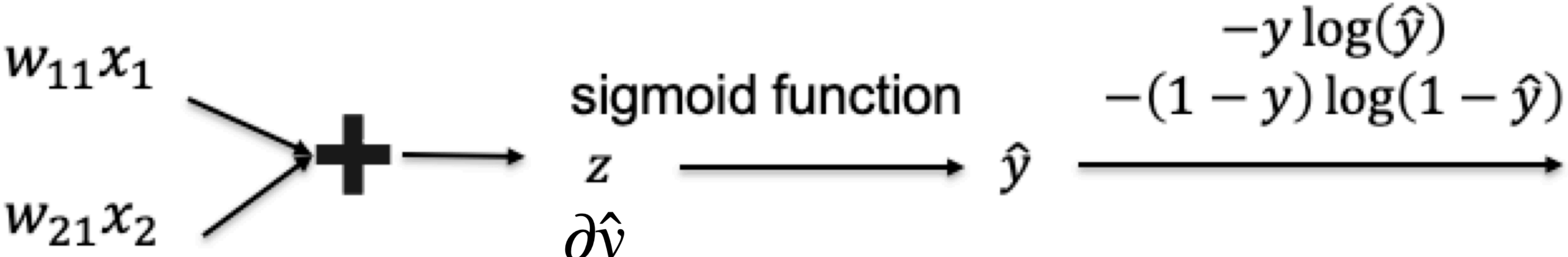


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

- By chain rule: $\dfrac{\partial l}{\partial w_{11}} = (\hat{y} - y)x_1$

Computing Gradients





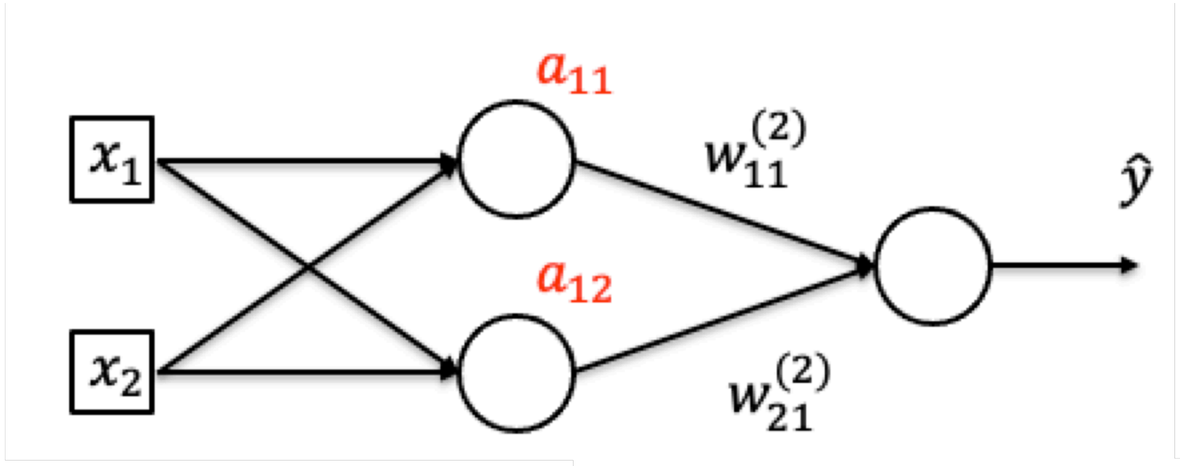
$\ell(\mathbf{x}, y)$

$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

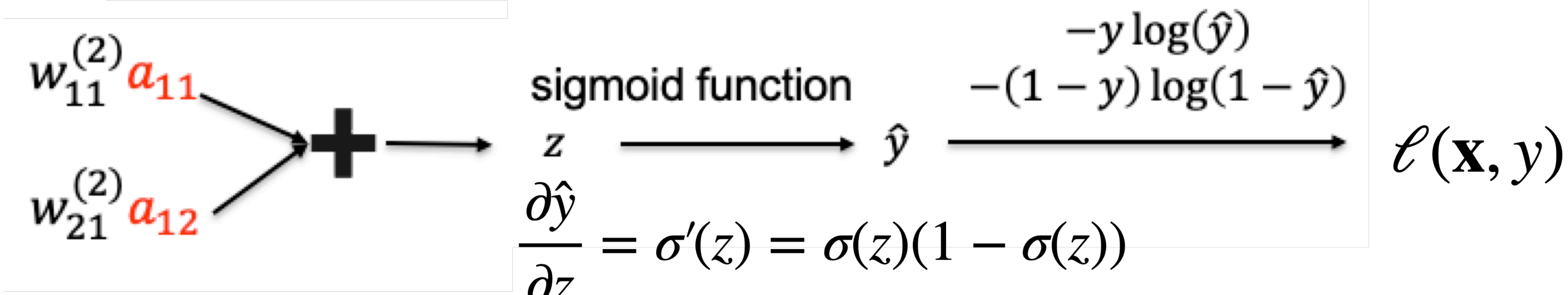
- By chain rule:

$$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} w_{11} = (\hat{y} - y) w_{11}$$

Computing Gradients: More Layers



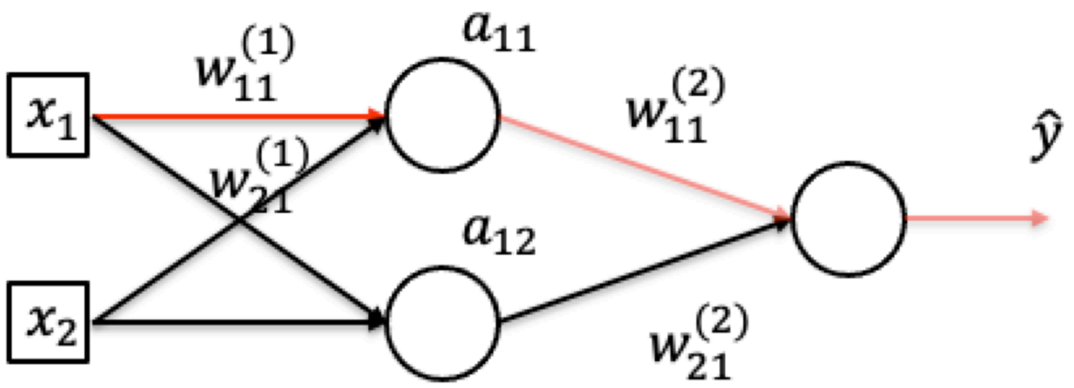
Make it deeper

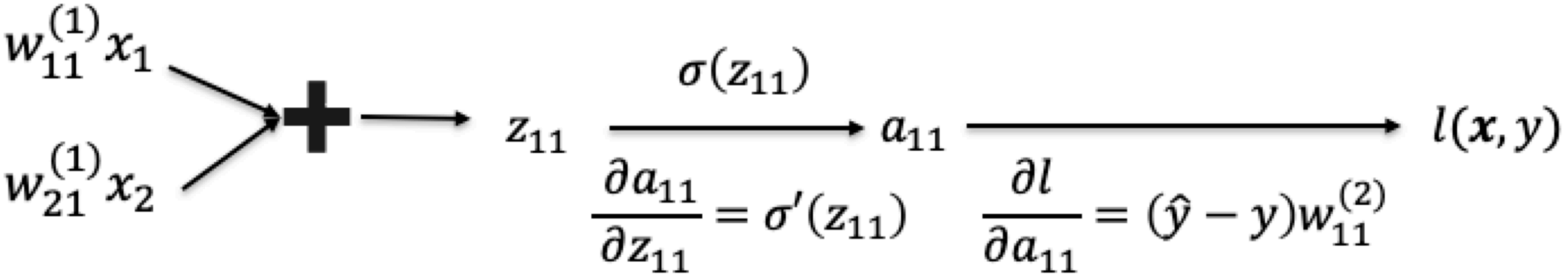


$$\frac{\partial \hat{y}}{\partial z} = \sigma'(z) = \sigma(z)(1-\sigma(z))$$

- By chain rule: $\dfrac{\partial l}{\partial a_{11}} = (\hat{y} - y) w_{11}^{(2)}, \quad \dfrac{\partial l}{\partial a_{12}} = (\hat{y} - y) w_{21}^{(2)}$

Computing Gradients: More Layers

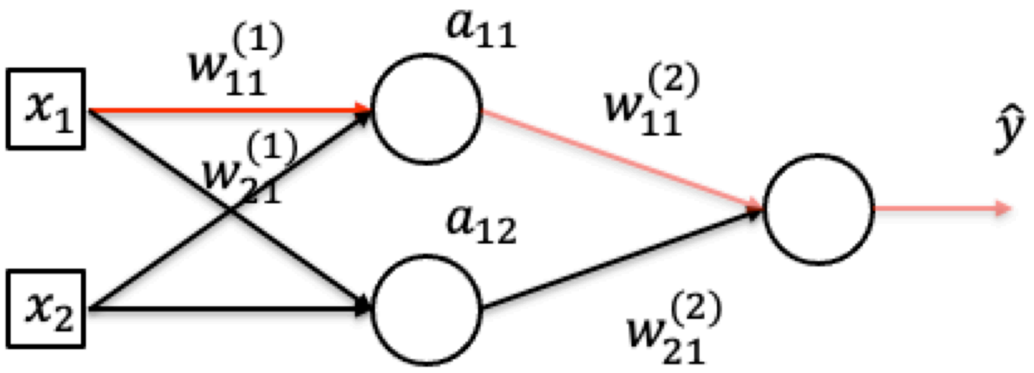


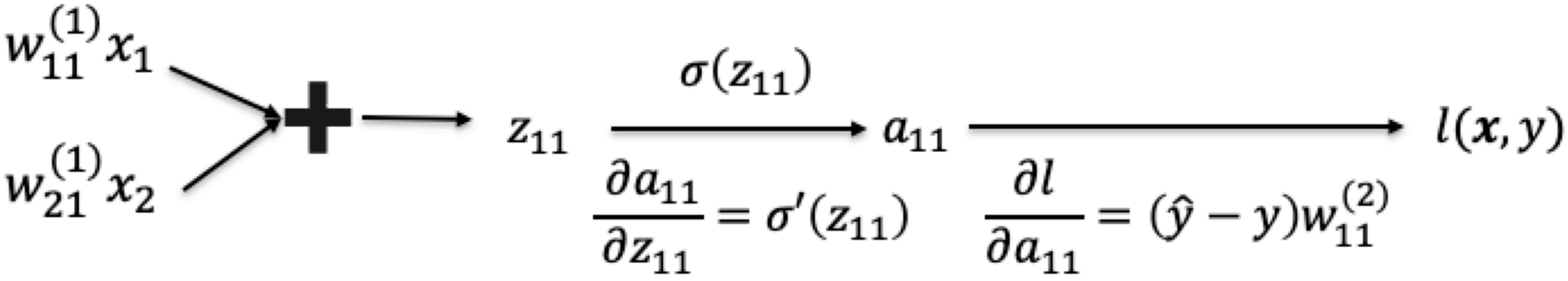


- By chain rule:

$$\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}^{(1)}} = (\hat{y} - y)w_{11}^{(2)} \frac{\partial a_{11}}{\partial w_{11}^{(1)}}$$

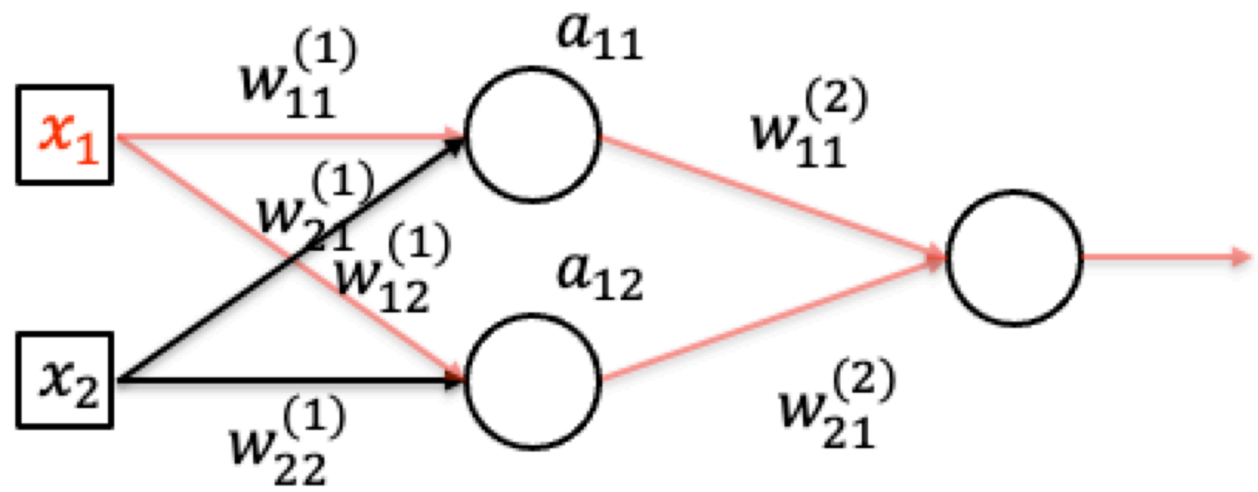
Computing Gradients: More Layers

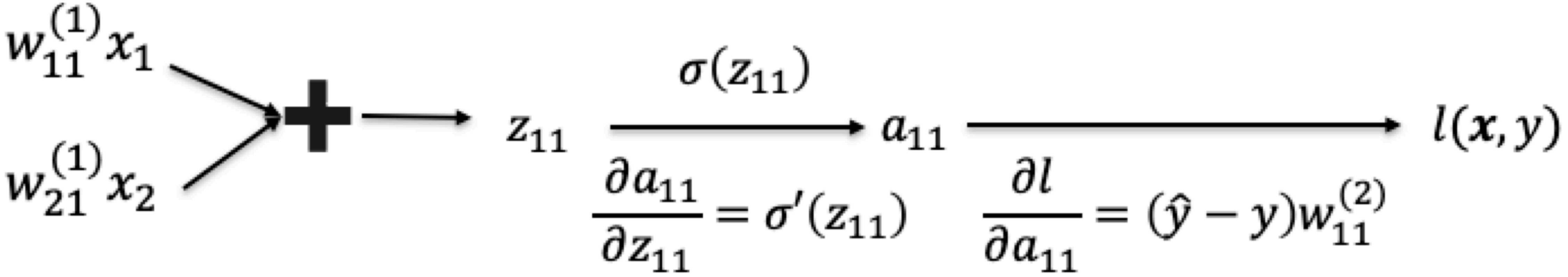




- By chain rule: $\frac{\partial l}{\partial w_{11}} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial w_{11}^{(1)}} = (\hat{y} - y) w_{11}^{(2)} a_{11}(1 - a_{11}) x_1$

Computing Gradients: More Layers



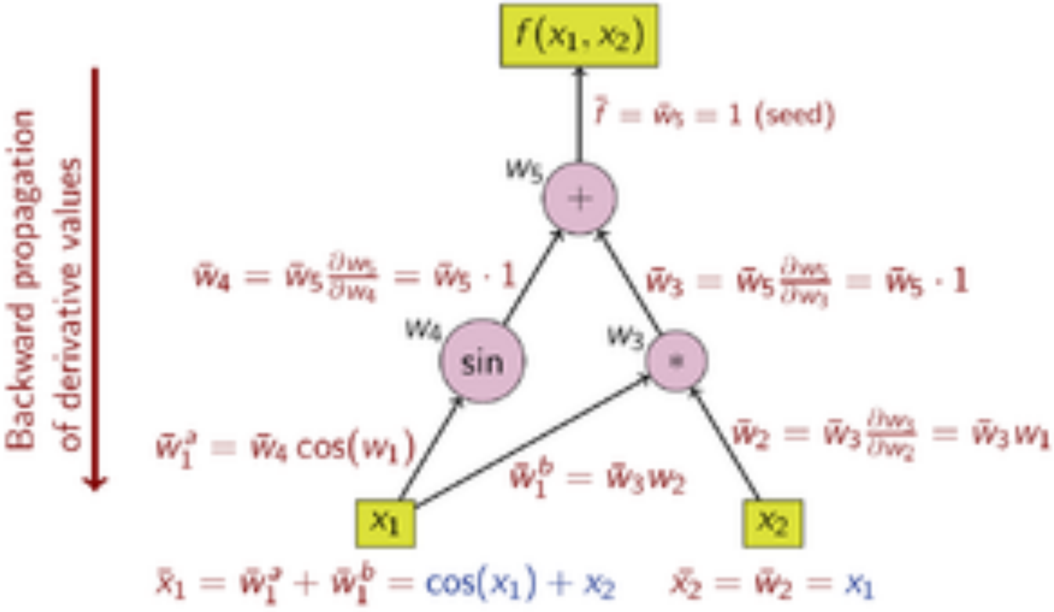


- By chain rule:

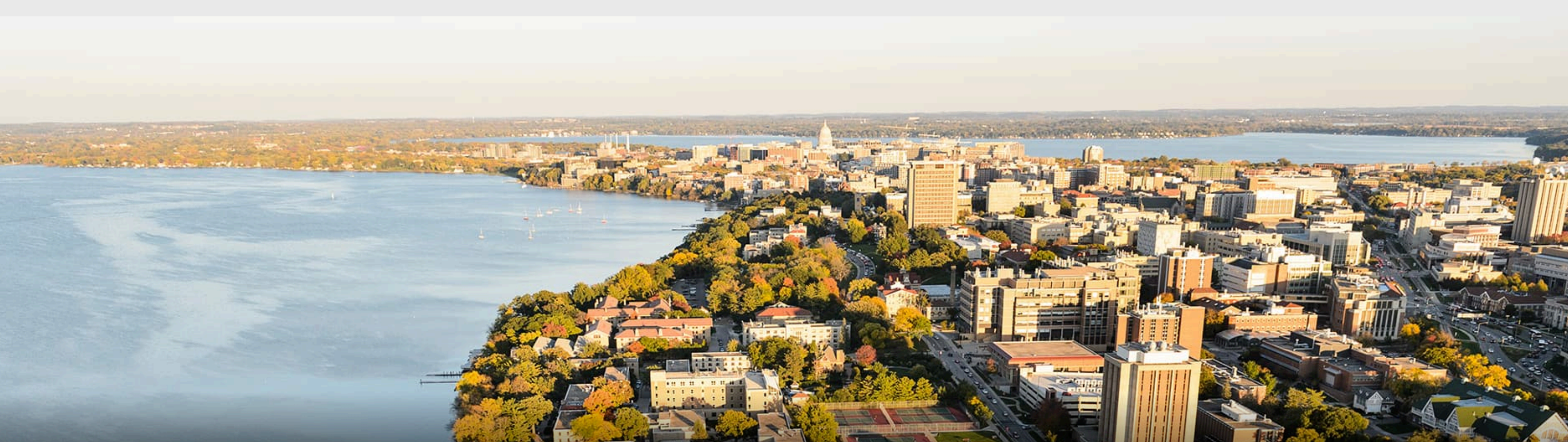
$$\frac{\partial l}{\partial x_1} = \frac{\partial l}{\partial a_{11}} \frac{\partial a_{11}}{\partial x_1} + \frac{\partial l}{\partial a_{12}} \frac{\partial a_{12}}{\partial x_1}$$

Backpropagation

- Now we can compute derivatives for particular neurons, but we want to automate this process
- Set up a computation graph and run on the graph
- Go backwards from top to bottom, recursively computing gradients



Wiki



Break & Quiz

Q2-1: Are these statements true or false?
(A) Backpropagation is based on the chain rule.
(B) Backpropagation contains only forward passes.

1. True, True
2. True, False
3. False, True
4. False, False

Q2-1: Are these statements true or false?
(A) Backpropagation is based on the chain rule.
(B) Backpropagation contains only forward passes.

1. True, True
2. True, False ←
3. False, True
4. False, False

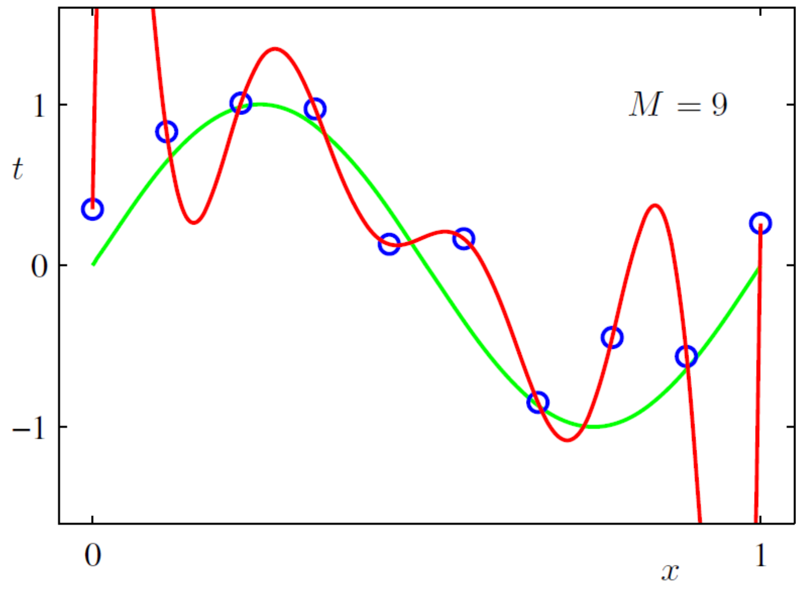
(A) We use chain rule to calculate the partial derivatives of composite functions like neural network.
(B) It contains both forward and backward passes.

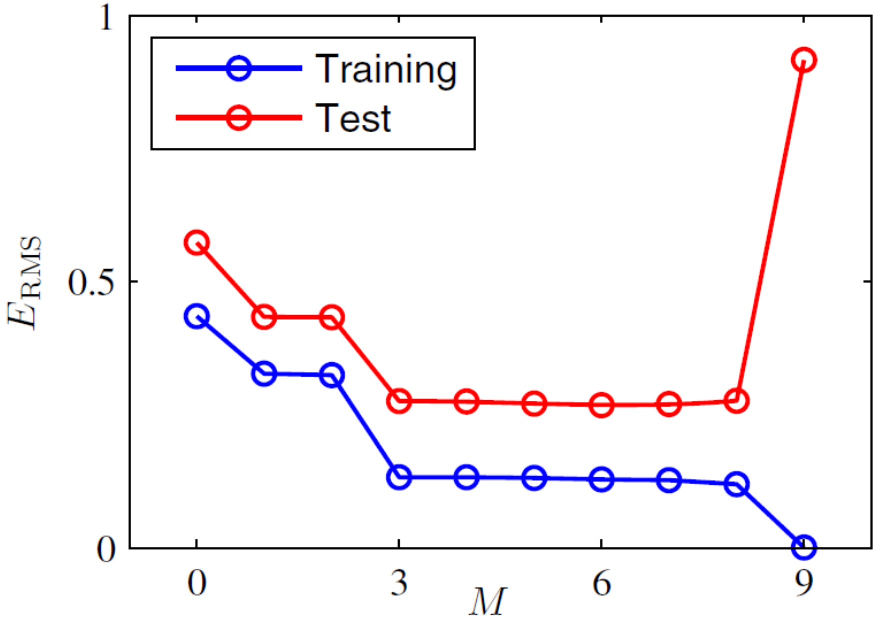
Outline

- **Neural Networks**
  - Introduction, Setup, Components, Activations
- **Training Neural Networks**
  - SGD, Computing Gradients, Backpropagation
- **Regularization**
  - Views, Data Augmentation, Other approaches

Review: Overfitting

- What is it? When empirical loss and expected loss are different
- Possible solutions:
  - Larger data set
  - Throwing away useless hypotheses also helps (**regularization**)





Review: Regularization

- In general: any method to **prevent overfitting**

- One approach: modify the optimization objective

- Different "views"
  - Hard constraint,
  - Soft constraint,
  - Bayesian view

Regularization: Hard Constraint View

- Training objective / parametrized version

$$\min_f \hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} l(f, x_i, y_i) \qquad \min_\theta \hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i)$$

subject to: $f \in \mathcal{H}$ subject to: $\theta \in \Omega$

- Constrain beyond it's natural choice

$$\min_\theta \hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i) \qquad \min_\theta \hat{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i)$$

subject to: $R(\theta) \leq r$ subject to: $||\theta||_2^2 \leq r^2$ ← **L2 Regularization**

Regularization: Soft Constraint View

- Equivalent to, for some parameter $\lambda \uparrow * > 0$

$$\min_{\theta} \hat{L}_R(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i) + \lambda^* R(\theta)$$

- For L2,

$$\min_{\theta} \hat{L}_R(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(\theta, x_i, y_i) + \lambda^* ||\theta||_2^2$$

- Comes from **Lagrangian duality**

Regularization: Bayesian Prior View

- Recall our MAP version of training. Bayes law:

$$p(\theta \mid \{x_i, y_i\}) = \frac{p(\theta)p(\{x_i, y_i\}|\theta)}{p(\{x_i, y_i\})}$$

- MAP:

$$\max_{\theta} \log p(\theta \mid \{x_i, y_i\}) = \min_{\theta} - \underbrace{\log p(\theta)}_{\text{Regularization}} - \log \underbrace{p(\{x_i, y_i\} \mid \theta)}_{\text{MLE loss}}$$

Choice of View?

- Typical choice for optimization: soft-constraint

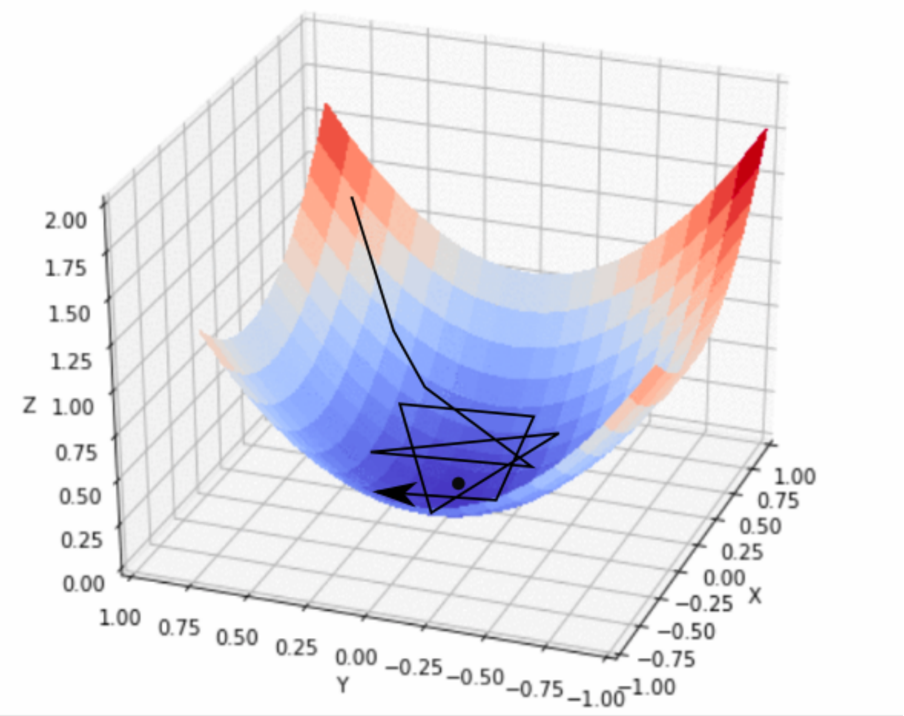
$$\min_{\theta} \hat{L}_R(\theta) = \hat{L}(\theta) + \lambda R(\theta)$$

- Hard constraint / Bayesian view: conceptual / for derivation
- Hard-constraint preferred if
  - Know the explicit bound
- Bayesian view preferred if
  - Domain knowledge easy to represent as a prior

Examples: L2 Regularization

$$\min_{\theta} \hat{L}_R(\theta) = \hat{L}(\theta) + \frac{\lambda}{2} ||\theta||_2^2$$

- Questions: what are the
  - Effects on (stochastic) gradient descent?
  - Effects on the optimal solution?



L2 Regularization: **Effect on GD**

- Gradient of regularized objective

$$\nabla \hat{L}_R(\theta) = \nabla \hat{L}(\theta) + \lambda\theta$$

- Gradient descent update

$$\theta \leftarrow \theta - \eta \nabla \hat{L}_R(\theta) = \theta - \eta \nabla \hat{L}(\theta) - \eta\lambda\theta$$

$$= (1 - \eta\lambda)\theta - \eta \nabla \hat{L}(\theta)$$

- In words, **weight decay**

L2 Regularization: **Effect on Optimal Solution**

- Consider a quadratic approximation around

$$\hat{L}(\theta) \approx \hat{L}(\theta^*) + (\theta - \theta^*)^T \nabla \hat{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T H (\theta - \theta^*)$$

- Since is optimal,

$$\hat{L}(\theta) \approx \hat{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T H (\theta - \theta^*)$$

$$\nabla \hat{L}(\theta) \approx H(\theta - \theta^*)$$

L2 Regularization: **Effect on Optimal Solution**

- Gradient of regularized objective: $\nabla \hat{L}_R(\theta) \approx H(\theta - \theta^*) + \lambda\theta$

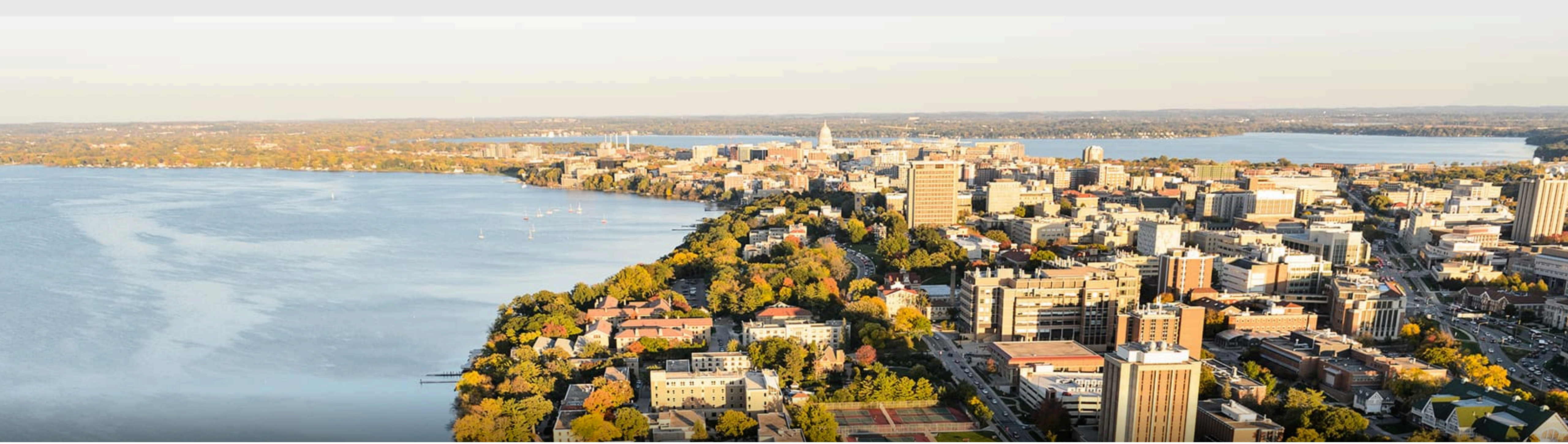
- On the optimal θ_R^*: $0 = \nabla \hat{L}_R(\theta_R^*) \approx H(\theta_R^* - \theta^*) + \lambda\theta_R^*$

$$\theta_R^* \approx (H + \lambda I)^{-1} H\theta^*$$

- H has eigendecomp. $H = Q\Lambda Q^T$, assume $(\Lambda + \lambda I)^{-1}$ exists:

$$\theta_R^* \approx (H + \lambda I)^{-1} H\theta^* = Q(\Lambda + \lambda I)^{-1}\Lambda Q^T\theta^*$$

Effect: **rescale along eigenvectors of** ***H***



Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov, Sharon Li