

# EECS 545: Machine Learning

## Lecture 11. Neural Networks and Deep Learning

Honglak Lee

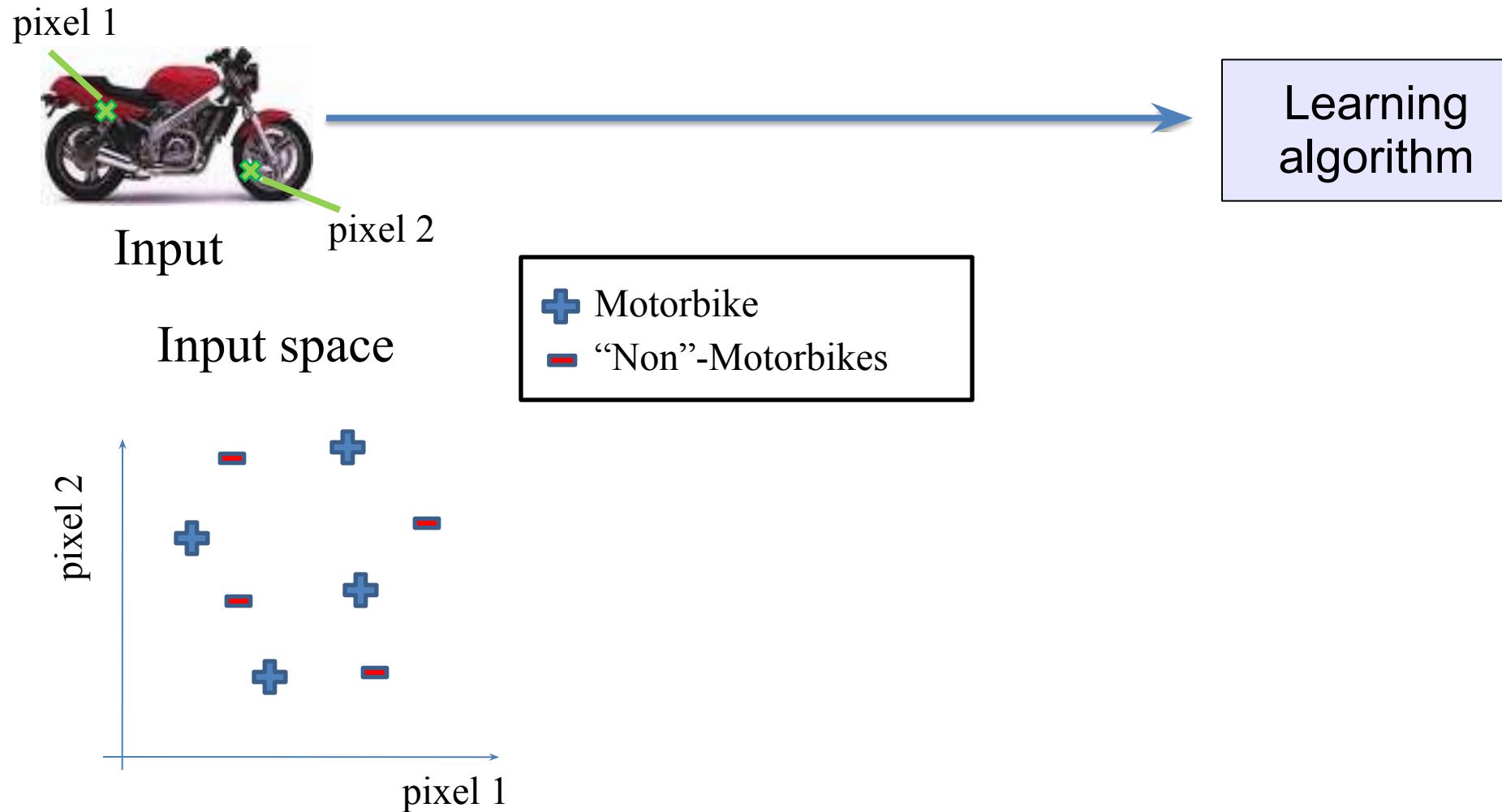
02/17/2020



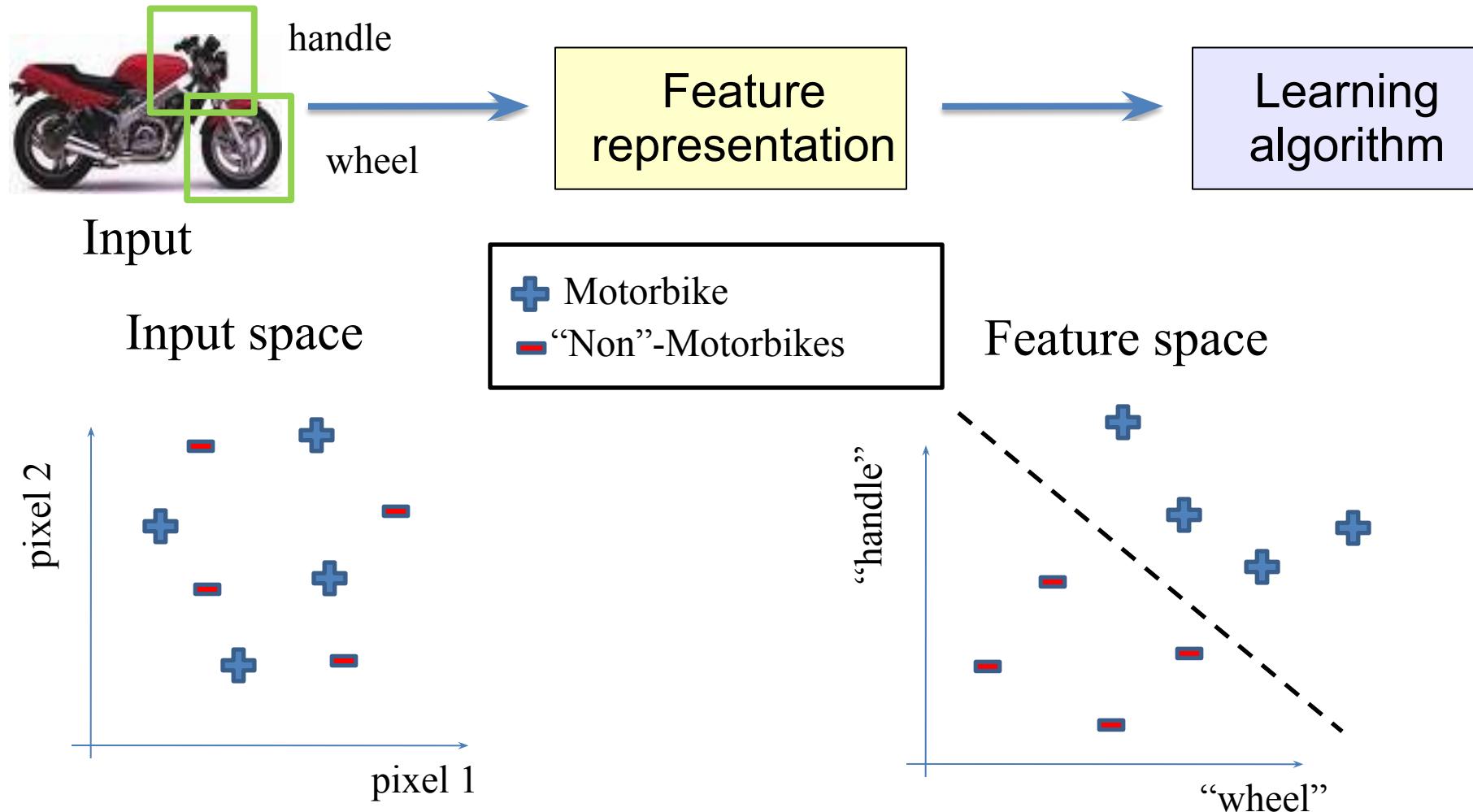
# Representing Data

- The success of machine learning applications relies on having a good representation of the data.
- Machine learning practitioners put lots of efforts in “feature engineering”.
- How can we develop good representations automatically?  
(Focus of this talk: “feature learning”)

# Feature representations

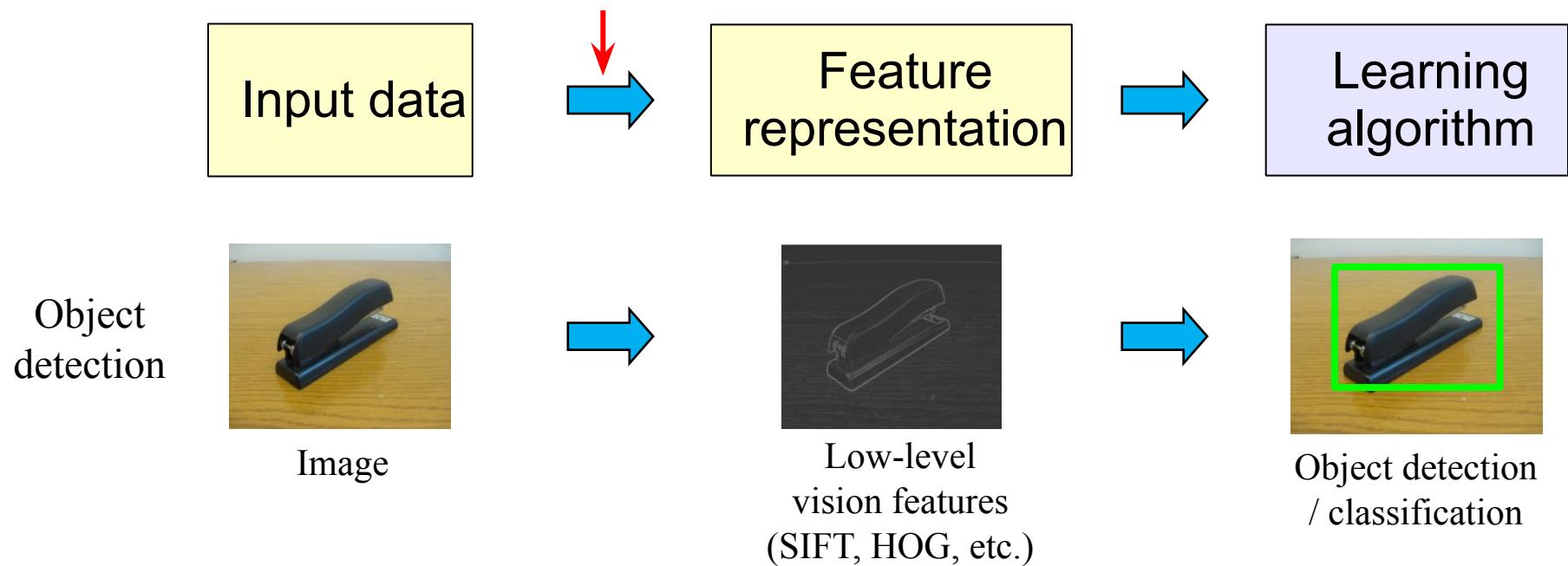


# Feature representations

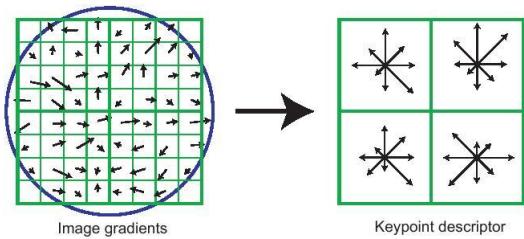


# How is computer perception done?

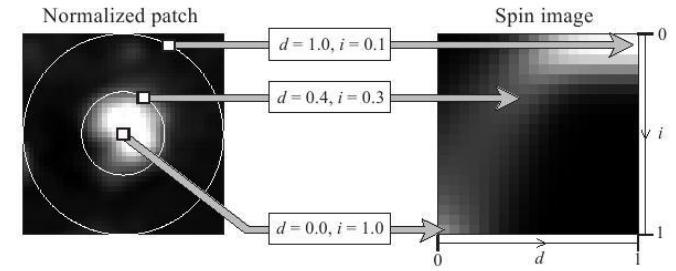
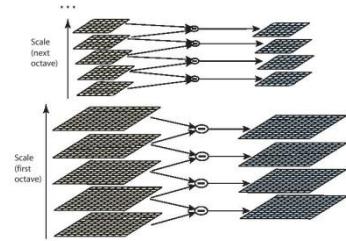
State-of-the-art:  
“hand-crafting”



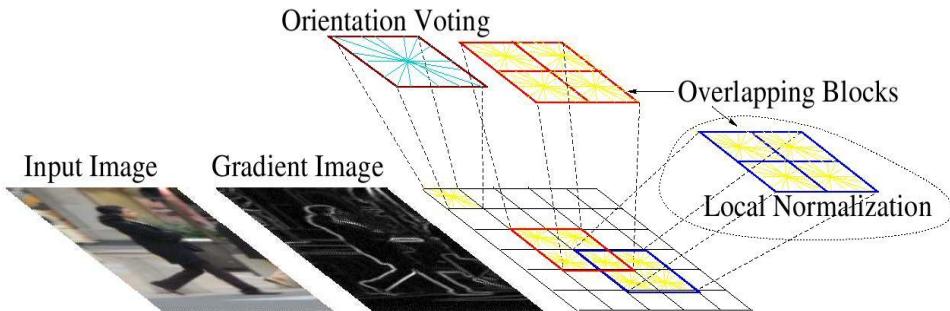
# Computer vision features



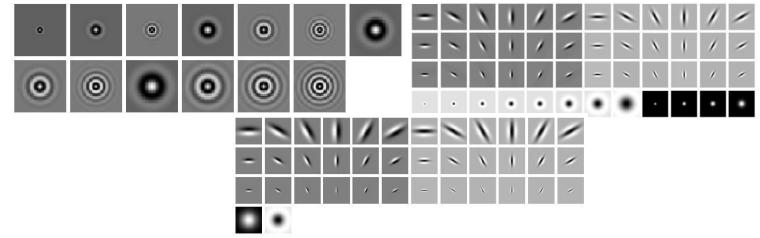
SIFT



Spin image

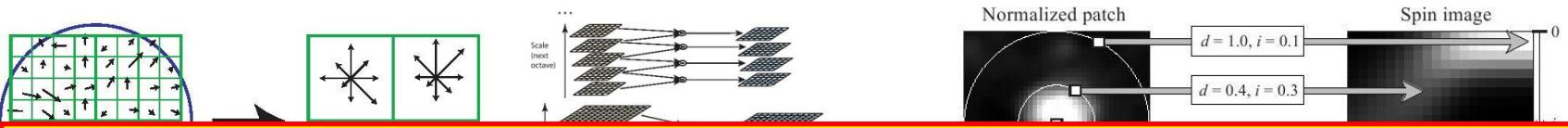


HoG



Textons

# Computer vision features



Hand-crafted features:

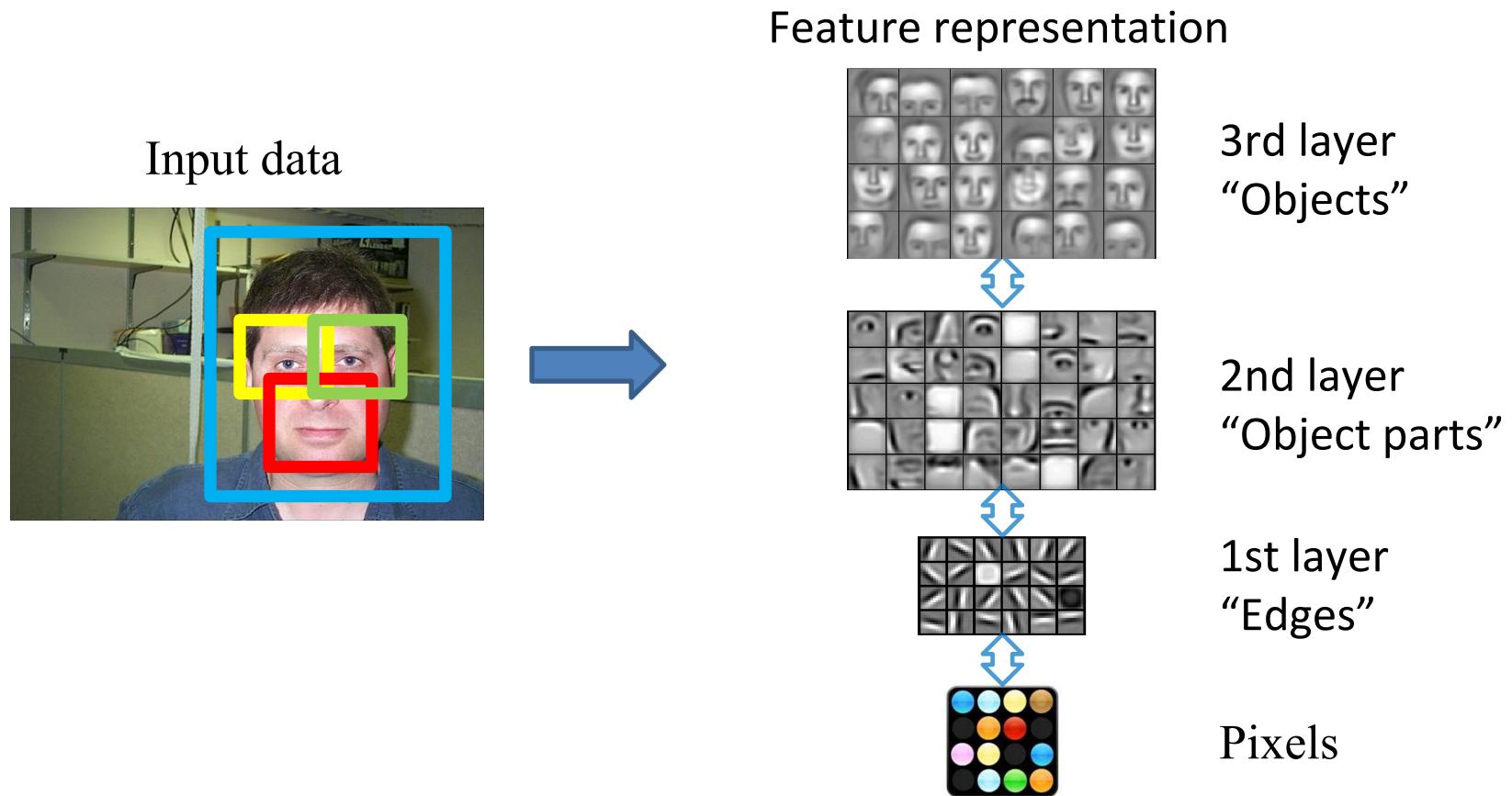
1. Needs expert knowledge
2. Requires time-consuming hand-tuning
3. (Arguably) a key limiting factor in advancing the state-of-the-arts

# Learning Feature Representations

- Key idea of Deep Learning:
  - Learn multiple levels of representation of increasing complexity/abstraction.
  - The representations can be learned in both **supervised** and/or **unsupervised** settings.
  - These features can be used for downstream tasks.

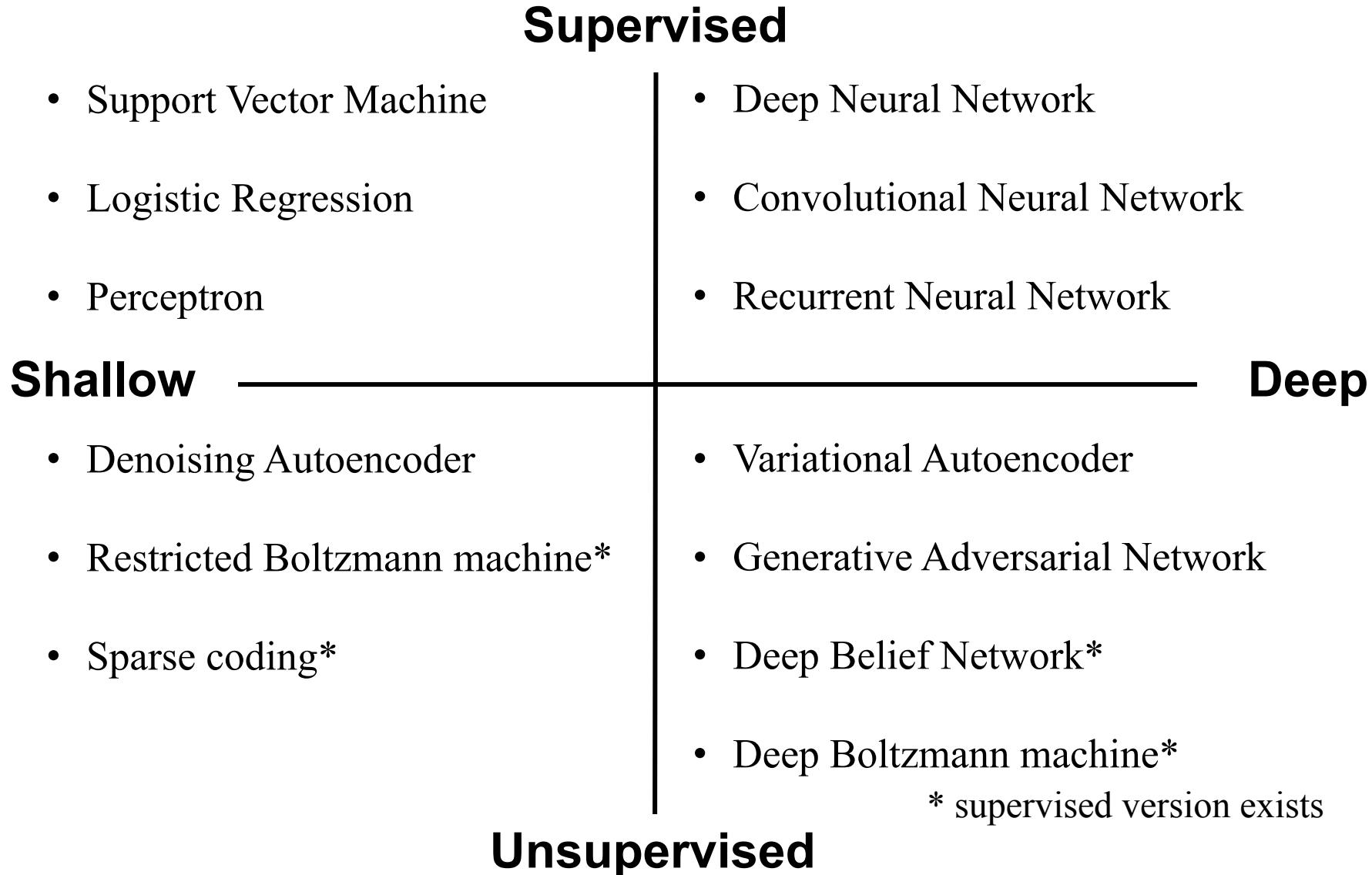
# Example: Learning Feature Hierarchy

1. Efficiently learn **useful attributes (features)** from unlabeled and labeled data



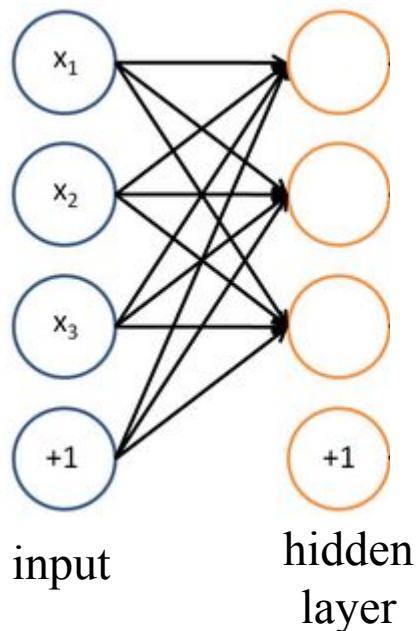
2. **Fill in representation gap** in machine learning

# Taxonomy of feature learning methods



# Neural network

- Neural network: similar to running several logistic regressions at the same time
- If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs

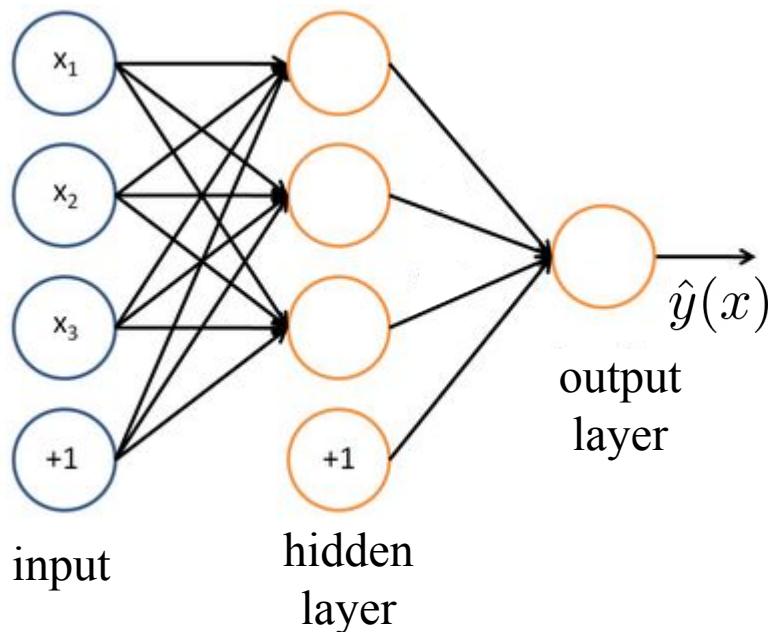


But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!

Slide Credit: Yoshua Bengio

# Neural network

- ... which we can feed into another logistic regression function



and it is the training criterion that will decide what those intermediate binary target variables should be, so as to make a good job of predicting the targets for the next layer, etc.

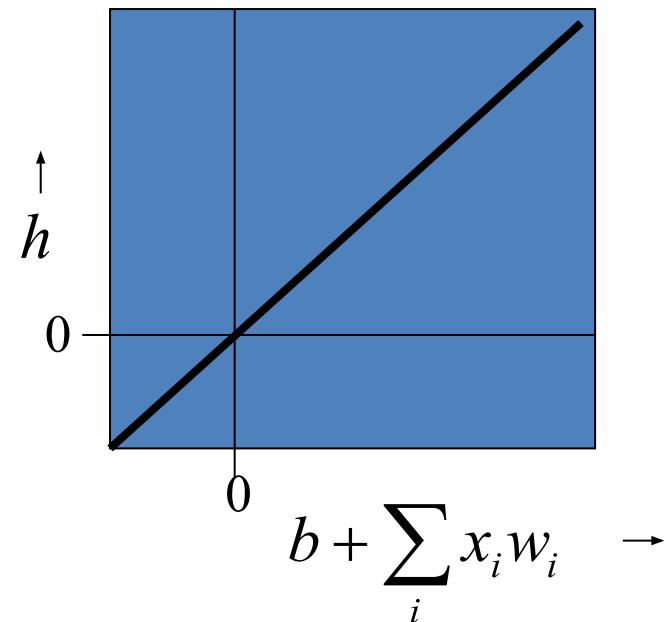
Slide Credit: Yoshua Bengio

# Types of Neurons: Linear Neurons

- These are simple but computationally limited
  - If we can make them learn we **may** get insight into more complicated neurons.

$$h = b + \sum_i x_i w_i$$

bias  
 $i^{\text{th}}$  input  
output  
index over  
input connections  
weight on  
 $i^{\text{th}}$  input



Slide Credit: Geoff Hinton

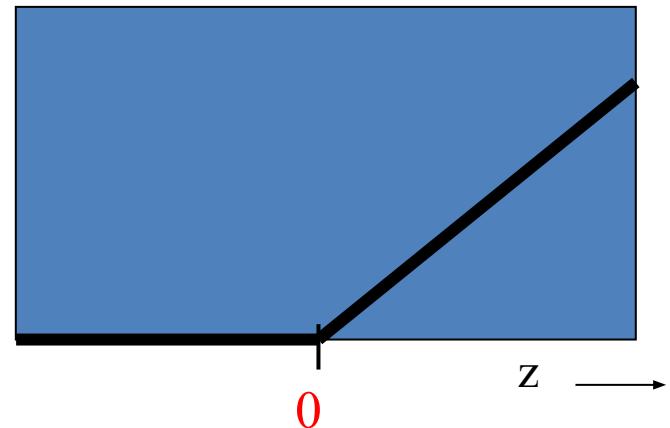
# Rectified Linear (linear threshold) Neurons

- They compute a **linear** weighted sum of their inputs.
- The output is a **non-linear** function of the total input.

$$z = b + \sum_i x_i w_i$$

$$h = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

↑  
h



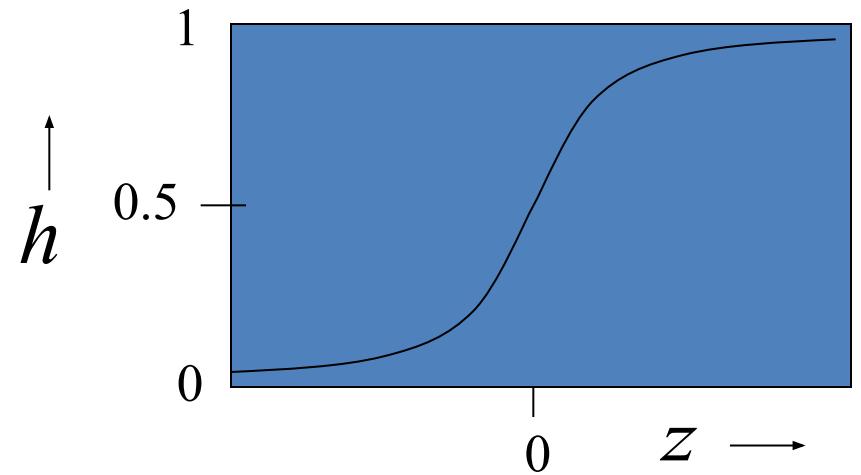
Slide Credit: Geoff Hinton

# Sigmoid (logistic) neurons

- These give a real-valued output that is a smooth and bounded function of their total input.
  - They have nice derivatives which make learning easy

$$z = b + \sum_i x_i w_i$$

$$h = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$



Slide Credit: Geoff Hinton

# Softmax neurons

- These give a real-valued output that is a smooth and bounded function of their total input.
  - The outputs sum up to 1 (useful for classification problems)
  - They have nice derivatives which make learning easy

$$z_k = b_k + \sum_i x_i w_i^k$$

$\mathbf{w}^k$  is the weight vector for the k-th output  
 $b^k$  is the bias for the k-th output

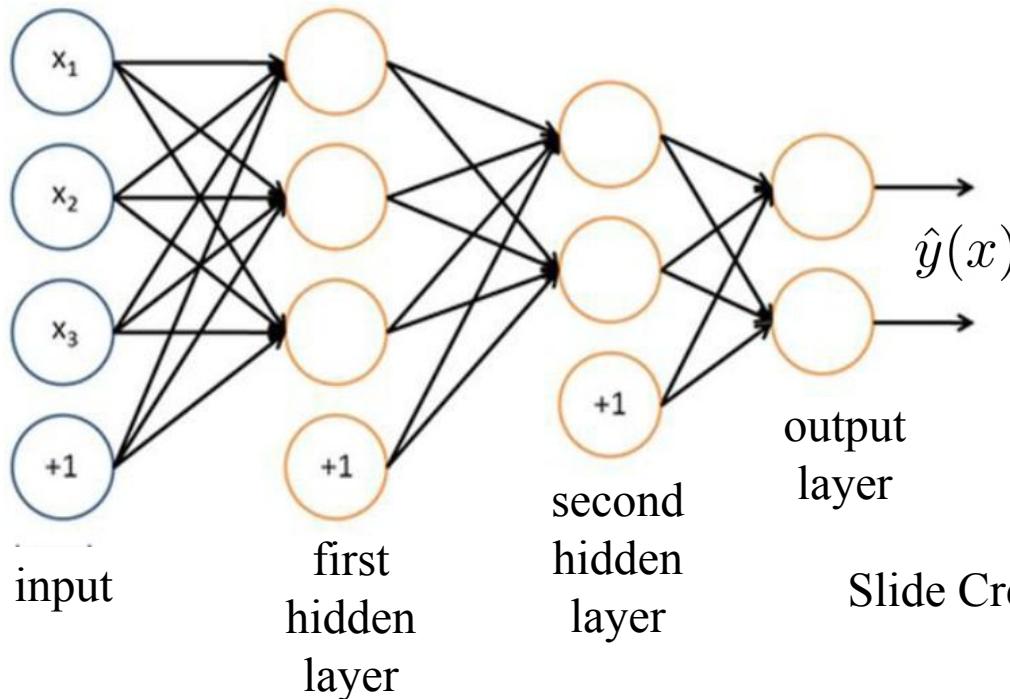
$$h_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Recall:

This softmax function is a generalization of logistic (sigmoid) function.

# Multilayer neural networks

- We can construct a multilayer neural network by defining the network connectivity and (nonlinear or linear) activation functions.
- Example:
  - Sigmoid nonlinearity for hidden layers
  - Softmax for the output layer



Slide Credit: Yoshua Bengio

# Training Neural Network

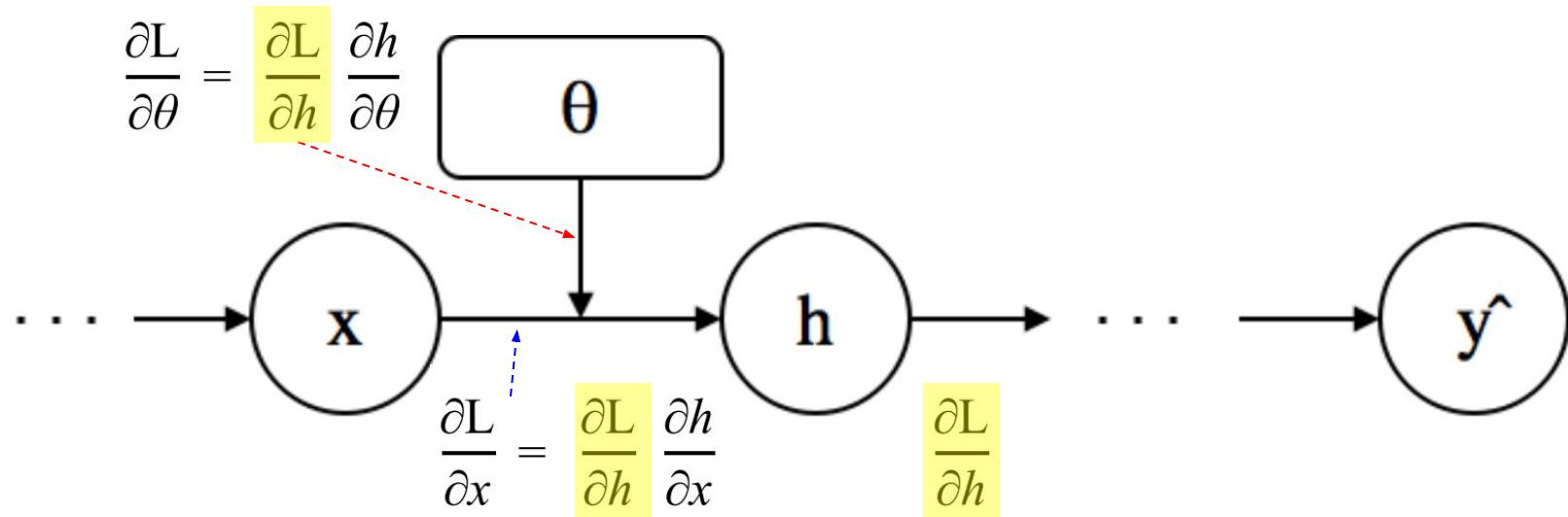
- Repeat until convergence
  - $(\mathbf{x}, \mathbf{y}) \leftarrow$  Sample an example (or a mini-batch) from data
  - $\hat{\mathbf{y}} \leftarrow f(\mathbf{x}; \theta)$  Forward propagation
  - Compute  $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$
  - $\nabla_{\theta} \mathcal{L} \leftarrow$  Backward propagation
  - Update weights using (stochastic) gradient descent
    - $\theta \leftarrow \theta - \alpha \nabla_{\theta} \mathcal{L}$

# Overview of backpropagation

- Recall: Deep Neural Network represent a complex function with nested, composite functions (represented by layer-wise operation and nonlinearity, etc.)
- Q. How can we compute the gradient of the complex function?
- A. Backpropagation:
  - Computing gradient via **chain rule** for compositional function
  - The chain rule can be expressed as a **local computation**
  - Think about it as a **computational graph**

# Backpropagation

- Denote  $x, h, \theta$  is the input, output, and parameter of a layer.
- It is non-trivial to derive the gradient of loss w.r.t. parameters in intermediate layers

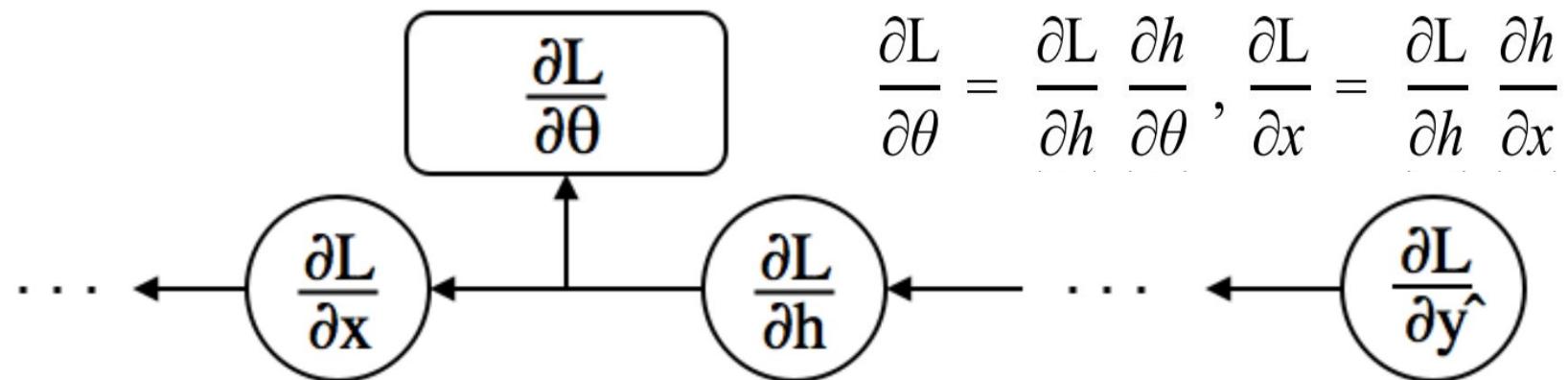


- Assuming that  $\frac{\partial L}{\partial h}$  is given, use the **chain rule** to compute the gradients

$$\frac{\partial L}{\partial \theta} = \underbrace{\frac{\partial L}{\partial h}}_{\text{given}} \underbrace{\frac{\partial h}{\partial \theta}}_{\text{easy}}, \quad \frac{\partial L}{\partial x} = \underbrace{\frac{\partial L}{\partial h}}_{\text{given}} \underbrace{\frac{\partial h}{\partial x}}_{\text{easy}}$$

# Idea behind backpropagation

- We need only  $\frac{\partial L}{\partial \theta}$  for gradient descent. Why compute  $\frac{\partial L}{\partial x}$ ?
  - The previous layer needs it because  $x$  is the output of the previous layer.



# Backpropagation: examples (NN with 1-hidden layer for regression)

# Forward Propagation

- **Example: a network with 1 hidden layer**

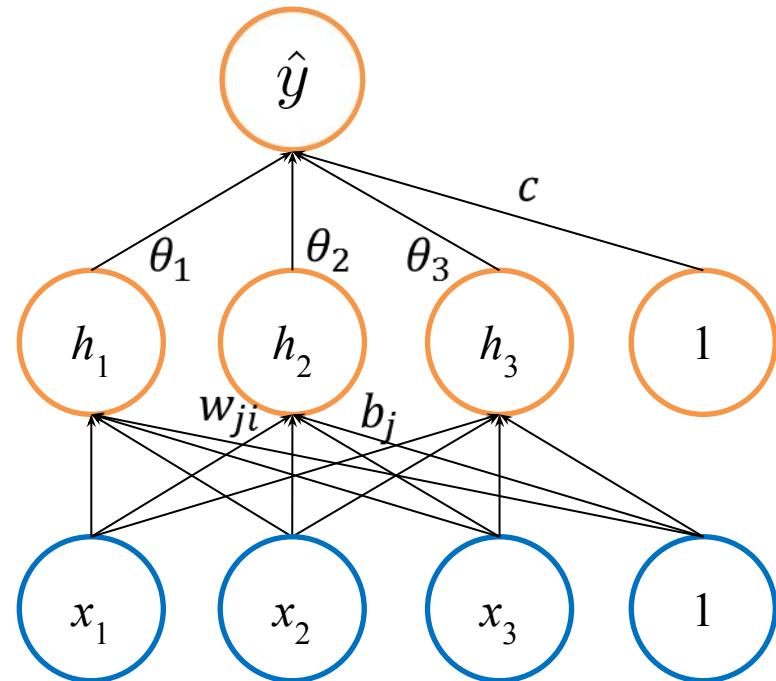
- Input:  $\mathbf{x}$
- Output:  $\hat{y}$
- Target:  $y$
- Loss function: square error  $(\hat{y} - y)^2$

Hidden  
layer

$$h_j = f\left(\sum_i w_{ji}x_i + b_j\right)$$

Output

$$\hat{y} = \sum_j \theta_j h_j + c$$



# Forward Propagation

- Example: a network with 1 hidden layer

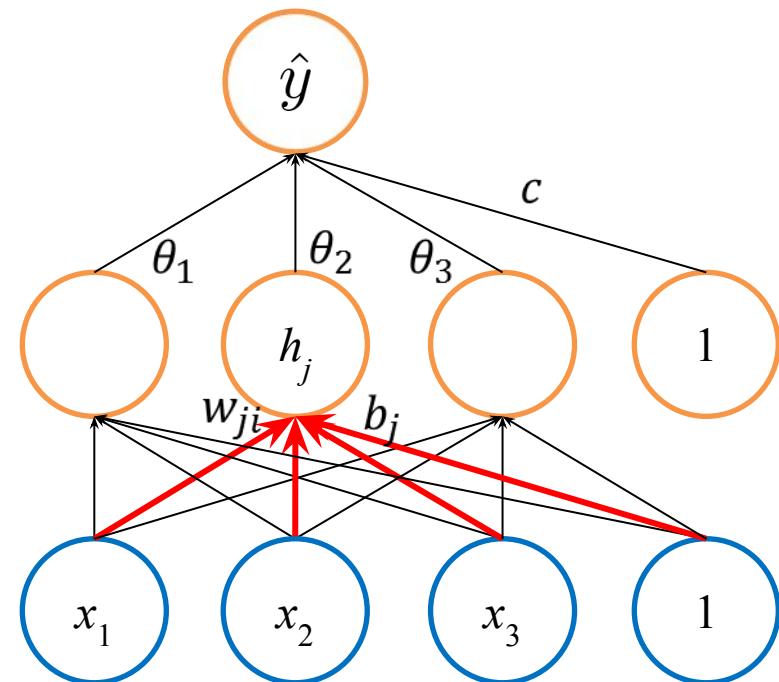
- Input:  $\mathbf{x}$
- Output:  $\hat{y}$
- Target:  $y$
- Loss function: square error  $(\hat{y} - y)^2$

Hidden  
layer

$$h_j = f\left(\sum_i w_{ji}x_i + b_j\right)$$

Output

$$\hat{y} = \sum_j \theta_j h_j + c$$



# Forward Propagation

- Example: a network with 1 hidden layer

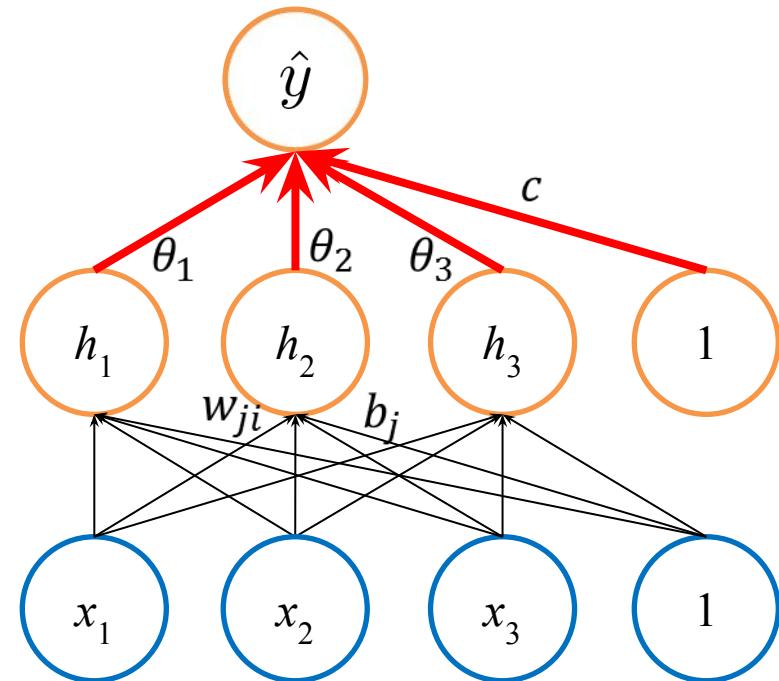
- Input:  $\mathbf{x}$
- Output:  $\hat{y}$
- Target:  $y$
- Loss function: square error  $(\hat{y} - y)^2$

Hidden  
layer

$$h_j = f\left(\sum_i w_{ji}x_i + b_j\right)$$

Output

$$\hat{y} = \sum_j \theta_j h_j + c$$



# Backpropagation

$$h_j = f\left(\sum_i w_{ji}x_i + b_j\right), \forall j$$

$$\hat{y} = \sum_j \theta_j h_j + c$$

$$L = (\hat{y} - y)^2$$

- Goal: Compute gradient w.r.t. parameters  $\{W, b, \theta, c\}$
- Main idea: Apply a chain rule recursively!

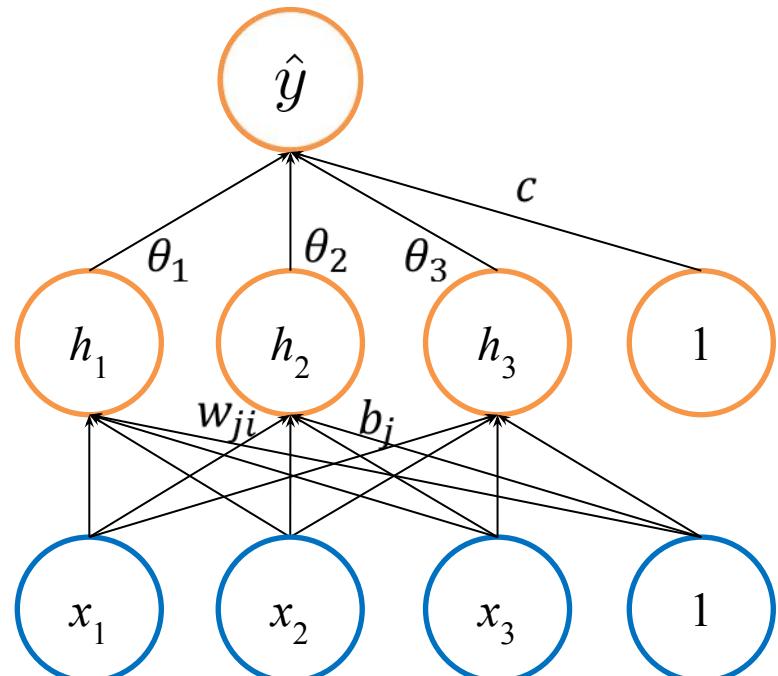
$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial \hat{y}}{\partial \theta_j} \frac{\partial L}{\partial \hat{y}} = h_j \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial h_j} = \frac{\partial \hat{y}}{\partial h_j} \frac{\partial L}{\partial \hat{y}} = \theta_j \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial h_j}{\partial w_{ji}} \frac{\partial L}{\partial h_j} = f' x_i \frac{\partial L}{\partial h_j}$$

where  $f' = f'\left(\sum_i w_{ji}x_i + b_j\right)$

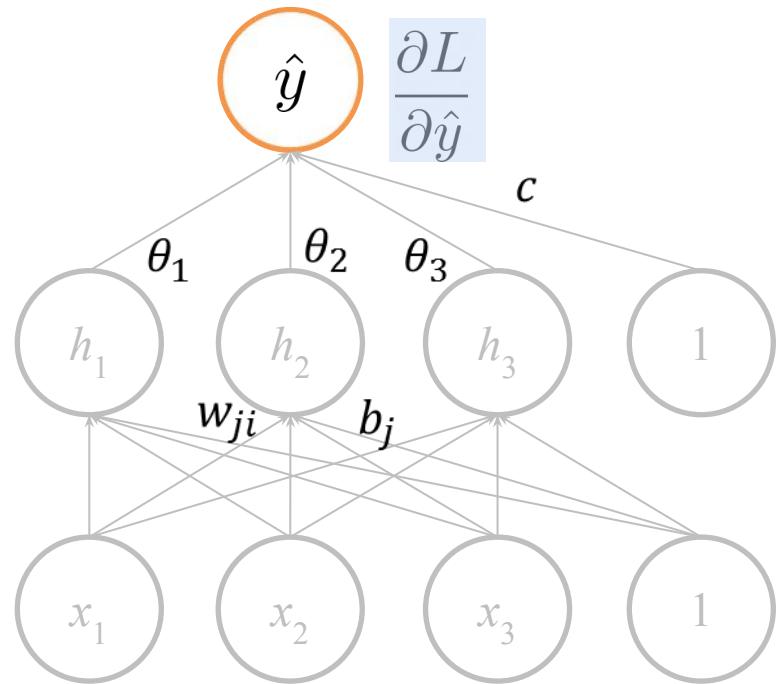


# Backpropagation

$$h_j = f\left(\sum_i w_{ji}x_i + b_j\right), \forall j$$
$$\hat{y} = \sum_j \theta_j h_j + c$$
$$L = (\hat{y} - y)^2$$

- Goal: Compute gradient w.r.t. parameters  $\{W, b, \theta, c\}$
- Main idea: Apply a chain rule recursively!

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$



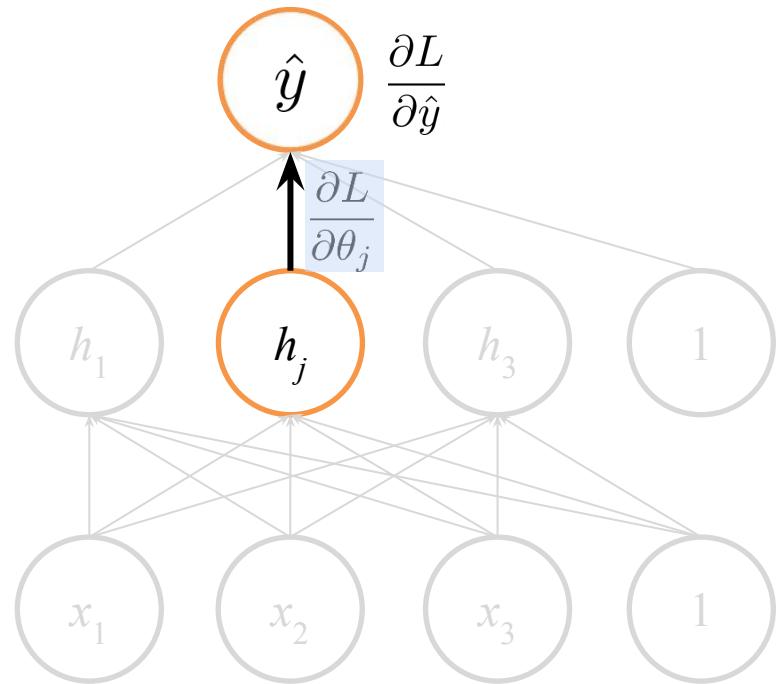
# Backpropagation

$$h_j = f\left(\sum_i w_{ji}x_i + b_j\right), \forall j$$
$$\hat{y} = \sum_j \theta_j h_j + c$$
$$L = (\hat{y} - y)^2$$

- Goal: Compute gradient w.r.t. parameters  $\{W, b, \theta, c\}$
- Main idea: Apply a chain rule recursively!

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial \hat{y}}{\partial \theta_j} \frac{\partial L}{\partial \hat{y}} = h_j \frac{\partial L}{\partial \hat{y}}$$



# Backpropagation

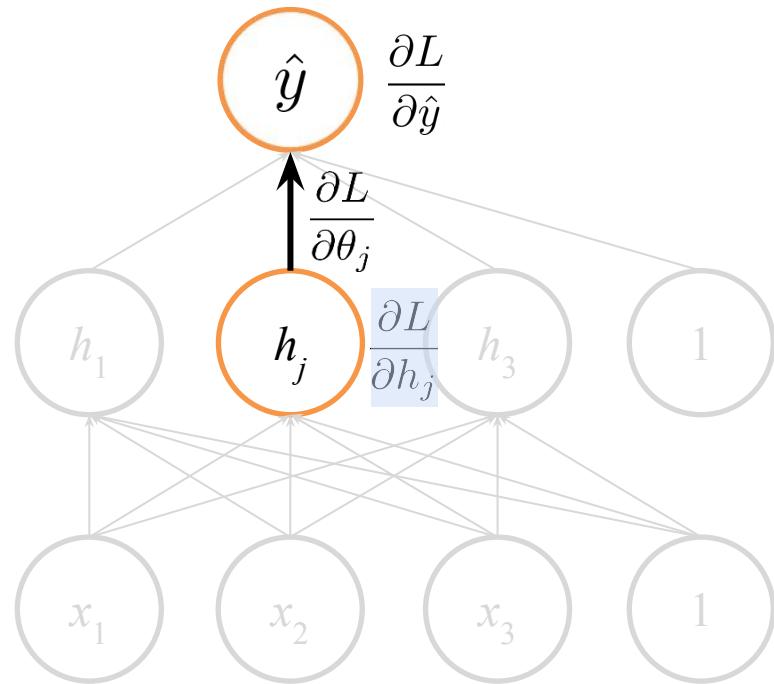
$$h_j = f\left(\sum_i w_{ji}x_i + b_j\right), \forall j$$
$$\hat{y} = \sum_j \theta_j h_j + c$$
$$L = (\hat{y} - y)^2$$

- Goal: Compute gradient w.r.t. parameters  $\{W, b, \theta, c\}$
- Main idea: Apply a chain rule recursively!

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial \hat{y}}{\partial \theta_j} \frac{\partial L}{\partial \hat{y}} = h_j \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial h_j} = \frac{\partial \hat{y}}{\partial h_j} \frac{\partial L}{\partial \hat{y}} = \theta_j \frac{\partial L}{\partial \hat{y}}$$



# Backpropagation

$$h_j = f\left(\sum_i w_{ji}x_i + b_j\right), \forall j$$

$$\hat{y} = \sum_j \theta_j h_j + c$$

$$L = (\hat{y} - y)^2$$

- Goal: Compute gradient w.r.t. parameters  $\{W, b, \theta, c\}$
- Main idea: Apply a chain rule recursively!

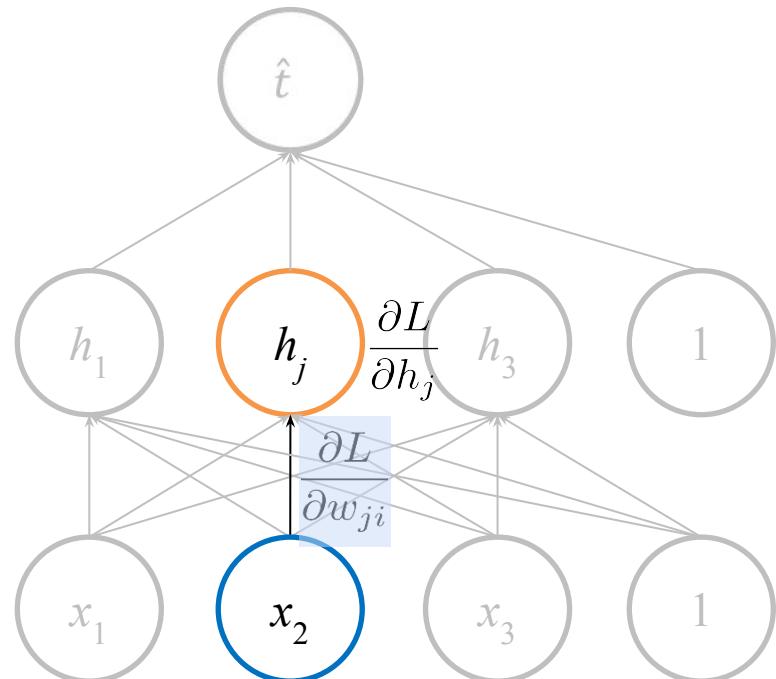
$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial \hat{y}}{\partial \theta_j} \frac{\partial L}{\partial \hat{y}} = h_j \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial h_j} = \frac{\partial \hat{y}}{\partial h_j} \frac{\partial L}{\partial \hat{y}} = \theta_j \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial h_j}{\partial w_{ji}} \frac{\partial L}{\partial h_j} = f' x_i \frac{\partial L}{\partial h_j}$$

where  $f' = f'\left(\sum_i w_{ji}x_i + b_j\right)$



# Backpropagation: examples (NN with 2-hidden layer for regression)

# Multilayer neural network

- Example: a network with 2 hidden layers

- Input:  $\mathbf{x}$
- Output:  $\hat{y}$
- Target:  $y$
- Loss function: square error  $(\hat{y} - y)^2$

First hidden layer

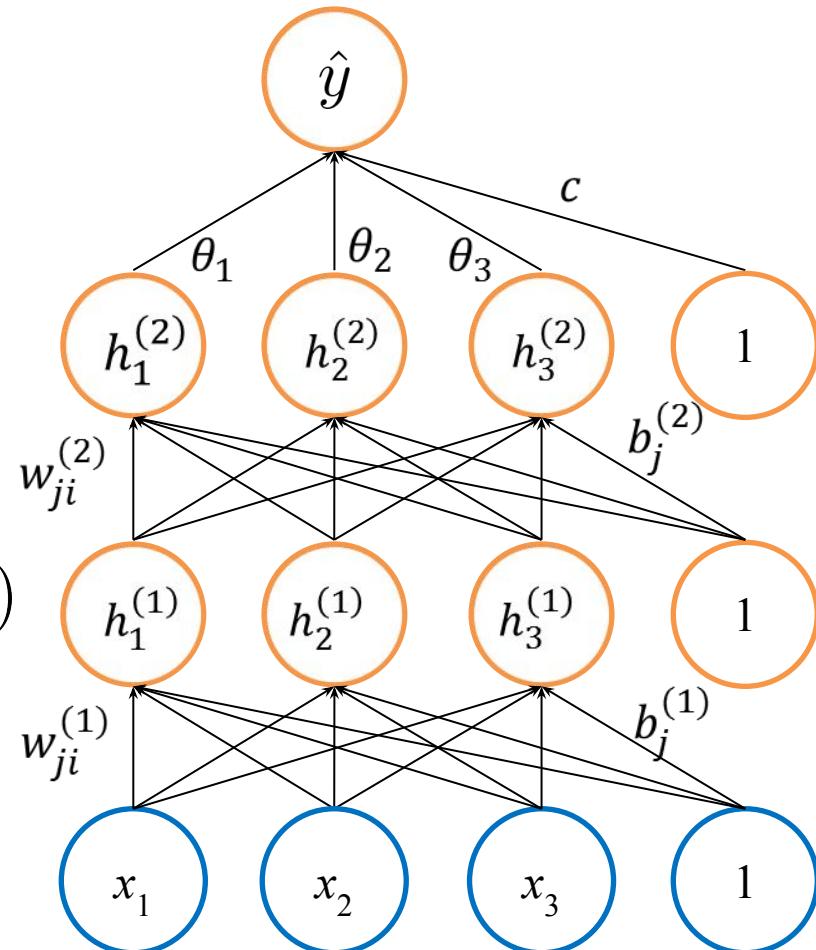
$$h_j^{(1)} = f\left(\sum_i w_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

Second hidden layer

$$h_j^{(2)} = f\left(\sum_i w_{ji}^{(2)} h_i^{(1)} + b_j^{(2)}\right)$$

Output

$$\hat{y} = \sum_j \theta_j h_j^{(2)} + c$$



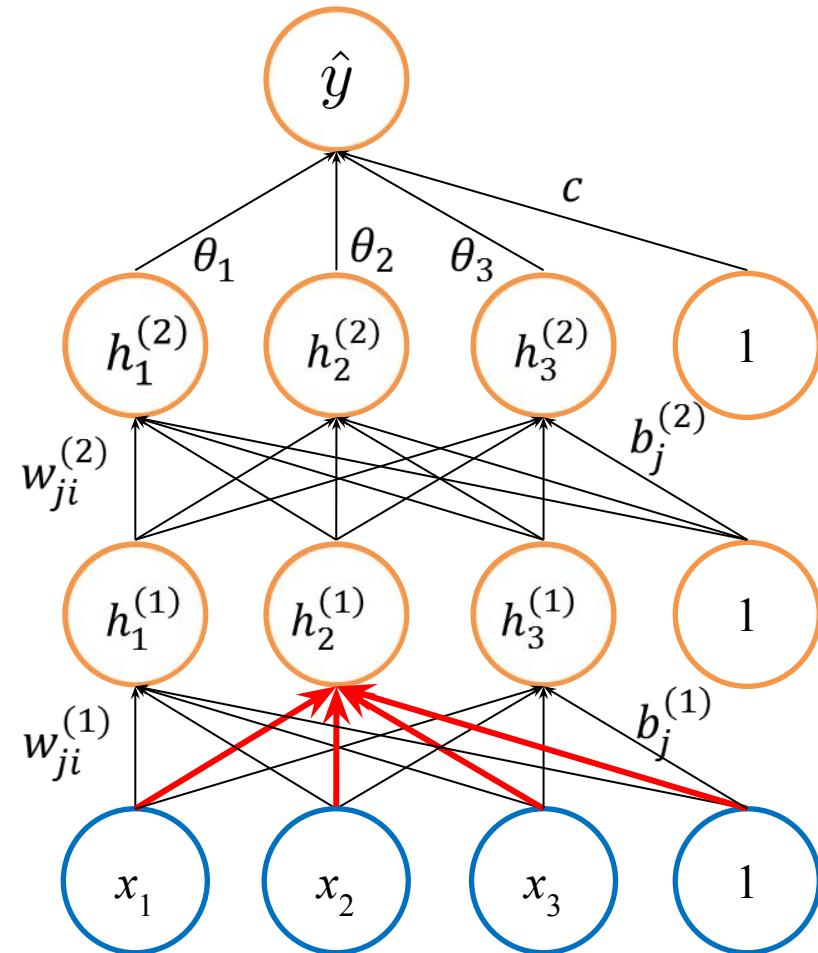
# Multilayer neural network

- Example: a network with 2 hidden layers

- Input:  $\mathbf{x}$
- Output:  $\hat{y}$
- Target:  $y$
- Loss function: square error  $(\hat{y} - y)^2$

First hidden layer

$$h_j^{(1)} = f\left(\sum_i w_{ji}^{(1)} x_i + b_j^{(1)}\right)$$



# Multilayer neural network

- Example: a network with 2 hidden layers

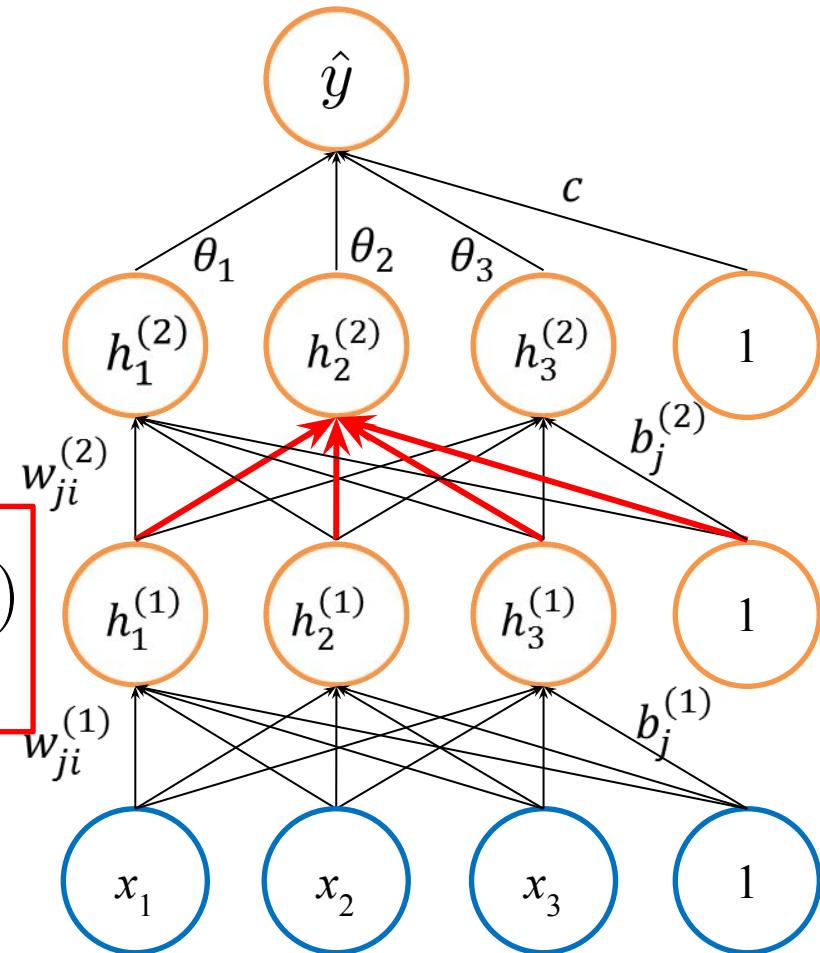
- Input:  $\mathbf{x}$
- Output:  $\hat{y}$
- Target:  $y$
- Loss function: square error  $(\hat{y} - y)^2$

First hidden layer

$$h_j^{(1)} = f\left(\sum_i w_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

Second hidden layer

$$h_j^{(2)} = f\left(\sum_i w_{ji}^{(2)} h_i^{(1)} + b_j^{(2)}\right)$$



# Multilayer neural network

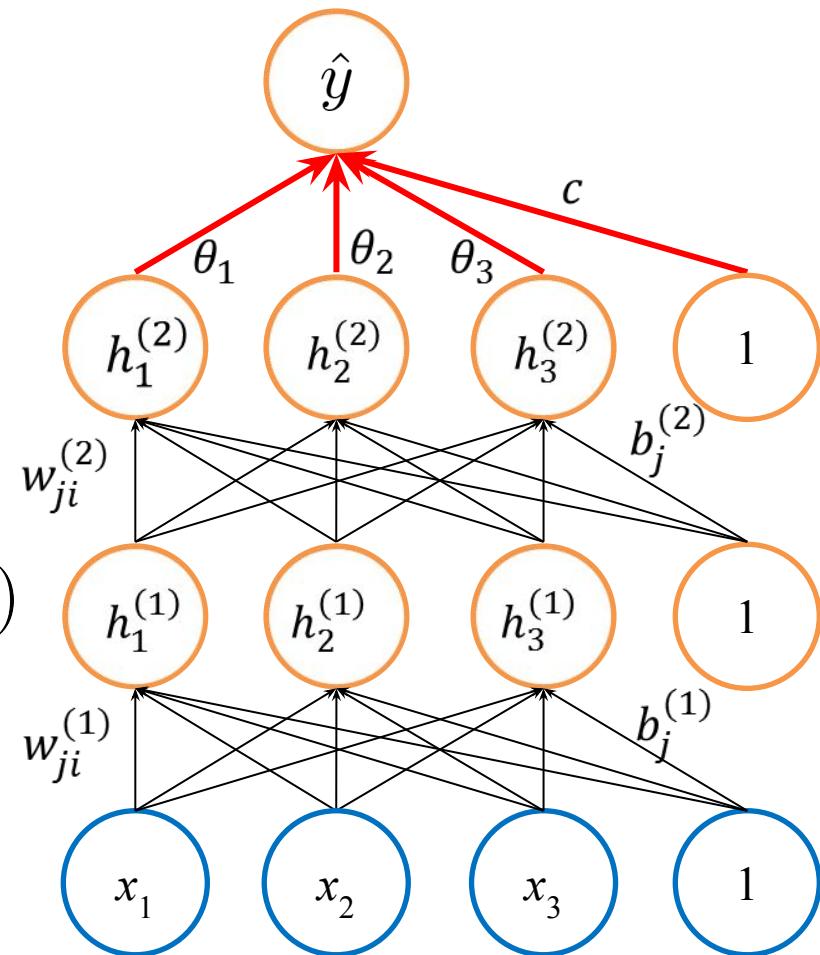
- Example: a network with 2 hidden layers

- Input:  $\mathbf{x}$
- Output:  $\hat{y}$
- Target:  $y$
- Loss function: square error  $(\hat{y} - y)^2$

First hidden layer 
$$h_j^{(1)} = f\left(\sum_i w_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

Second hidden layer 
$$h_j^{(2)} = f\left(\sum_i w_{ji}^{(2)} h_i^{(1)} + b_j^{(2)}\right)$$

Output 
$$\hat{y} = \sum_j \theta_j h_j^{(2)} + c$$



# Backpropagation: Compute gradient w.r.t. $\{W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}, \theta, c\}$

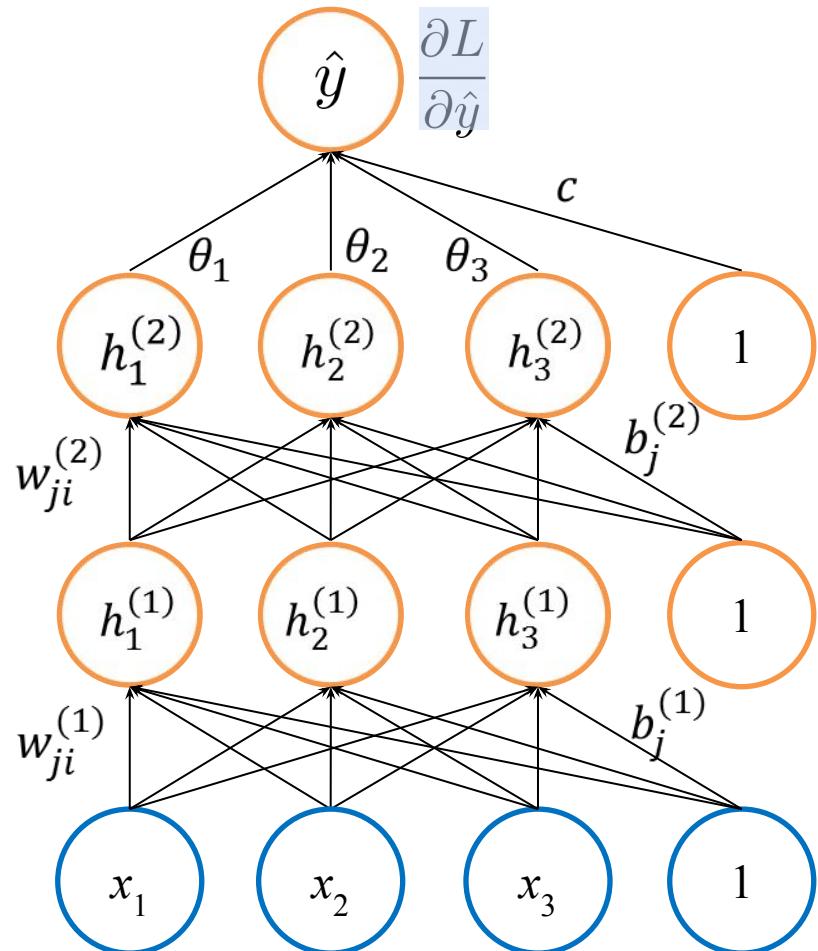
$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$h_j^{(1)} = f\left(\sum_i w_{ji}^{(1)} x_i + b_j^{(1)}\right), \forall j$$

$$h_j^{(2)} = f\left(\sum_i w_{ji}^{(2)} h_i^{(1)} + b_j^{(2)}\right), \forall j$$

$$\hat{y} = \sum_j \theta_j h_j^{(2)} + c$$

$$L = (\hat{y} - y)^2$$



# Backpropagation: Compute gradient w.r.t. $\{W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}, \theta, c\}$

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

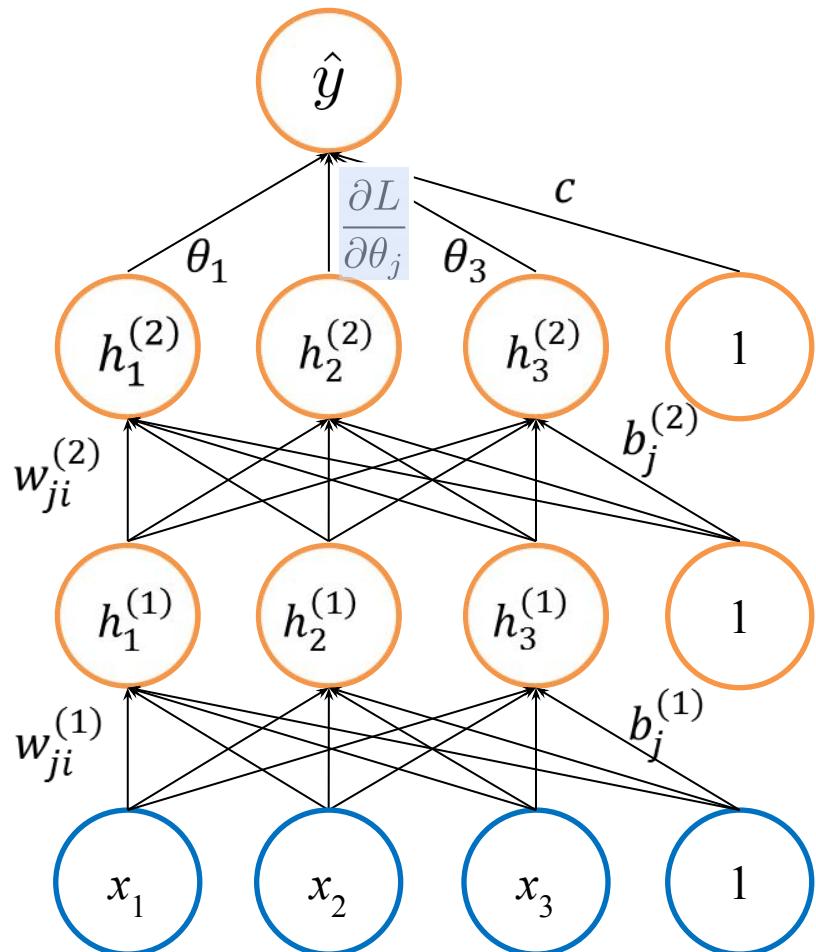
$$\frac{\partial L}{\partial \theta_j} = \frac{\partial \hat{y}}{\partial \theta_j} \frac{\partial L}{\partial \hat{y}} = h_j^{(2)} \frac{\partial L}{\partial \hat{y}}$$

$$h_j^{(1)} = f(\sum_i w_{ji}^{(1)} x_i + b_j^{(1)}), \forall j$$

$$h_j^{(2)} = f(\sum_i w_{ji}^{(2)} h_i^{(1)} + b_j^{(2)}), \forall j$$

$$\hat{y} = \sum_j \theta_j h_j^{(2)} + c$$

$$L = (\hat{y} - y)^2$$



# Backpropagation: Compute gradient w.r.t. $\{W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}, \theta, c\}$

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial \hat{y}}{\partial \theta_j} \frac{\partial L}{\partial \hat{y}} = h_j^{(2)} \frac{\partial L}{\partial \hat{y}}$$

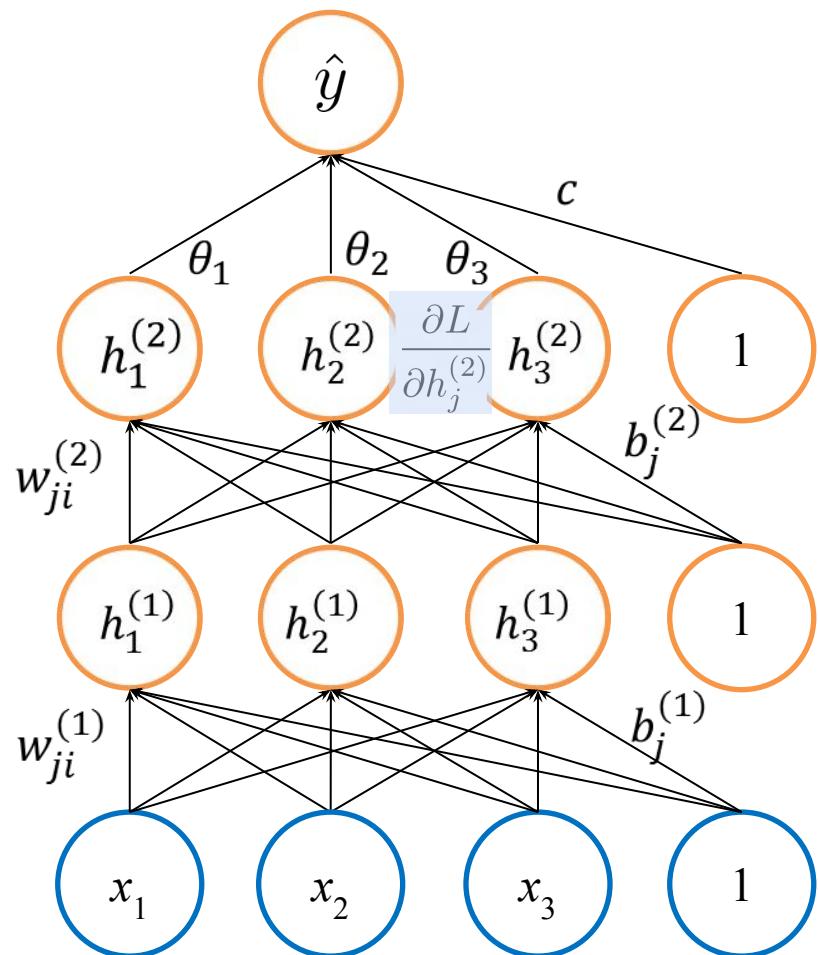
$$\frac{\partial L}{\partial h_j^{(2)}} = \frac{\partial \hat{y}}{\partial h_j^{(2)}} \frac{\partial L}{\partial \hat{y}} = \theta_j \frac{\partial L}{\partial \hat{y}}$$

$$h_j^{(1)} = f(\sum_i w_{ji}^{(1)} x_i + b_j^{(1)}), \forall j$$

$$h_j^{(2)} = f(\sum_i w_{ji}^{(2)} h_i^{(1)} + b_j^{(2)}), \forall j$$

$$\hat{y} = \sum_j \theta_j h_j^{(2)} + c$$

$$L = (\hat{y} - y)^2$$



# Backpropagation: Compute gradient w.r.t. $\{W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}, \theta, c\}$

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial \hat{y}}{\partial \theta_j} \frac{\partial L}{\partial \hat{y}} = h_j^{(2)} \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial h_j^{(2)}} = \frac{\partial \hat{y}}{\partial h_j^{(2)}} \frac{\partial L}{\partial \hat{y}} = \theta_j \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial w_{ji}^{(2)}} = \frac{\partial h_j^{(2)}}{\partial w_{ji}^{(2)}} \frac{\partial L}{\partial h_j^{(2)}} = f'(z_j^{(2)}) h_i^{(1)} \frac{\partial L}{\partial h_j^{(2)}}$$

where

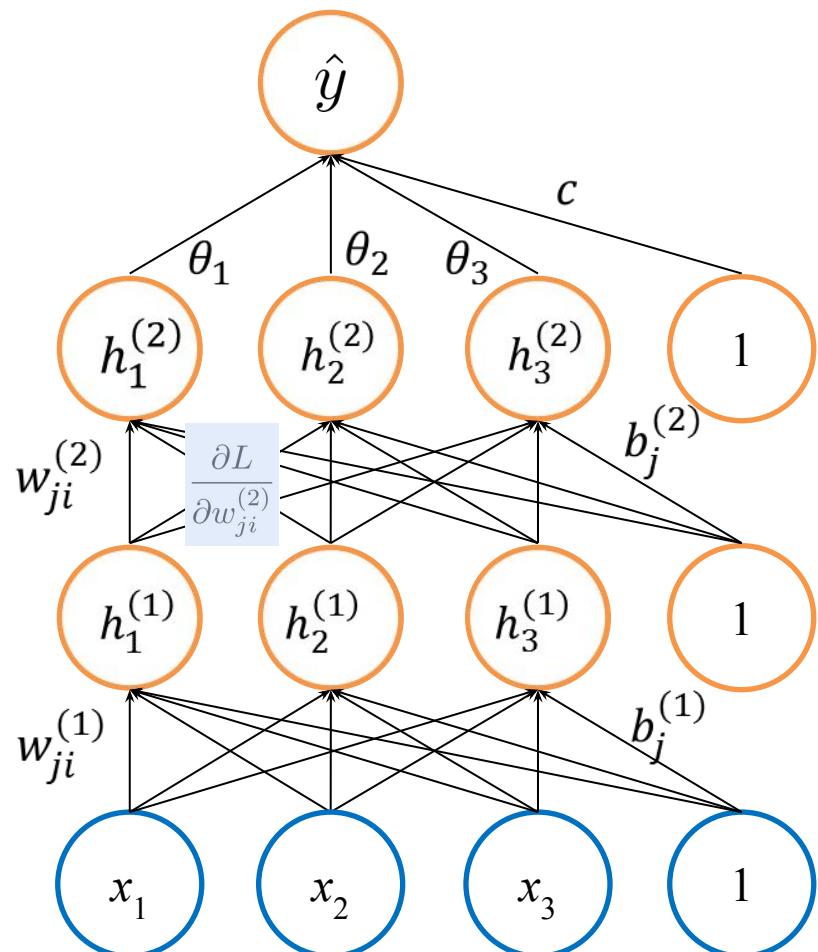
$$z_j^{(2)} = \sum_i h_i^{(1)} w_{ji}^{(2)} + b_j^{(2)}$$

$$h_j^{(1)} = f(\sum_i w_{ji}^{(1)} x_i + b_j^{(1)}), \forall j$$

$$h_j^{(2)} = f(\sum_i w_{ji}^{(2)} h_i^{(1)} + b_j^{(2)}), \forall j$$

$$\hat{y} = \sum_j \theta_j h_j^{(2)} + c$$

$$L = (\hat{y} - y)^2$$



# Backpropagation: Compute gradient w.r.t. $\{W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}, \theta, c\}$

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial \hat{y}}{\partial \theta_j} \frac{\partial L}{\partial \hat{y}} = h_j^{(2)} \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial h_j^{(2)}} = \frac{\partial \hat{y}}{\partial h_j^{(2)}} \frac{\partial L}{\partial \hat{y}} = \theta_j \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial w_{ji}^{(2)}} = \frac{\partial h_j^{(2)}}{\partial w_{ji}^{(2)}} \frac{\partial L}{\partial h_j^{(2)}} = f'(z_j^{(2)}) h_i^{(1)} \frac{\partial L}{\partial h_j^{(2)}}$$

$$\frac{\partial L}{\partial h_i^{(1)}} = \sum_j \frac{\partial h_j^{(2)}}{\partial h_i^{(1)}} \frac{\partial L}{\partial h_j^{(2)}} = \sum_j f'(z_j^{(2)}) w_{ji}^{(2)} \frac{\partial L}{\partial h_j^{(2)}}$$

where

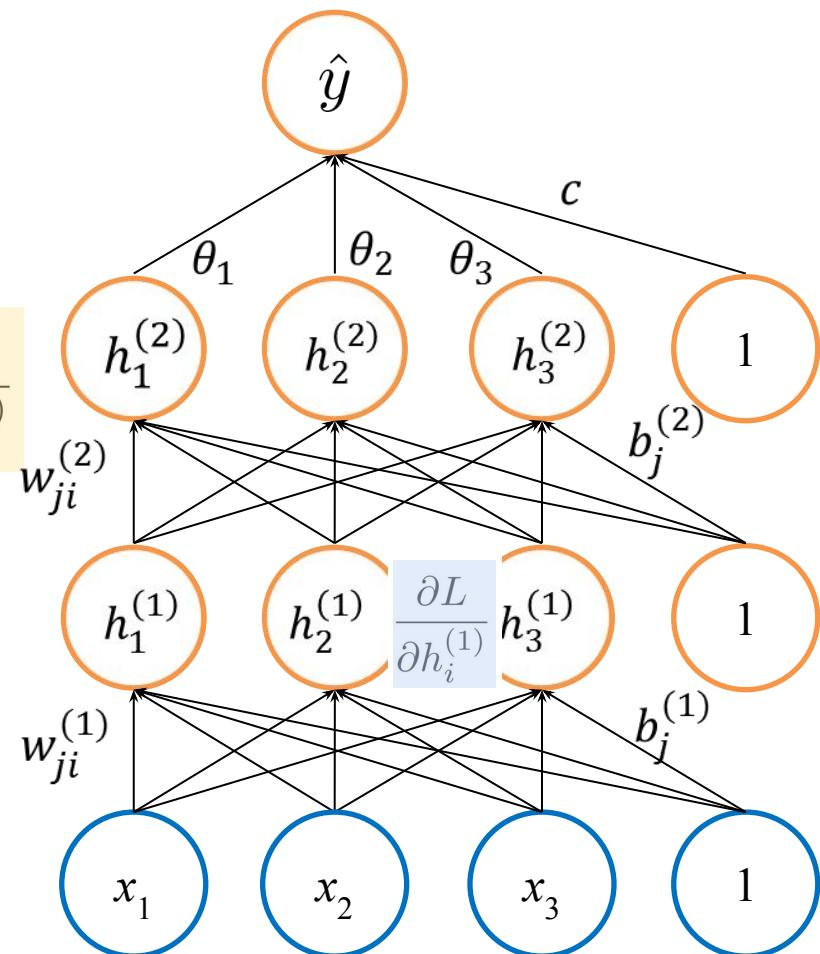
$$z_j^{(2)} = \sum_i h_i^{(1)} w_{ji}^{(2)} + b_j^{(2)}$$

$$h_j^{(1)} = f(\sum_i w_{ji}^{(1)} x_i + b_j^{(1)}), \forall j$$

$$h_j^{(2)} = f(\sum_i w_{ji}^{(2)} h_i^{(1)} + b_j^{(2)}), \forall j$$

$$\hat{y} = \sum_j \theta_j h_j^{(2)} + c$$

$$L = (\hat{y} - y)^2$$



# Backpropagation: Compute gradient w.r.t. $\{W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}, \theta, c\}$

$$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$$

$$\frac{\partial L}{\partial \theta_j} = \frac{\partial \hat{y}}{\partial \theta_j} \frac{\partial L}{\partial \hat{y}} = h_j^{(2)} \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial h_j^{(2)}} = \frac{\partial \hat{y}}{\partial h_j^{(2)}} \frac{\partial L}{\partial \hat{y}} = \theta_j \frac{\partial L}{\partial \hat{y}}$$

$$\frac{\partial L}{\partial w_{ji}^{(2)}} = \frac{\partial h_j^{(2)}}{\partial w_{ji}^{(2)}} \frac{\partial L}{\partial h_j^{(2)}} = f'(z_j^{(2)}) h_i^{(1)} \frac{\partial L}{\partial h_j^{(2)}}$$

$$\frac{\partial L}{\partial h_i^{(1)}} = \sum_j \frac{\partial h_j^{(2)}}{\partial h_i^{(1)}} \frac{\partial L}{\partial h_j^{(2)}} = \sum_j f'(z_j^{(2)}) w_{ji}^{(2)} \frac{\partial L}{\partial h_j^{(2)}}$$

$$\frac{\partial L}{\partial w_{ik}^{(1)}} = \frac{\partial h_i^{(1)}}{\partial w_{ik}^{(1)}} \frac{\partial L}{\partial h_i^{(1)}} = f'(z_i^{(1)}) x_k \frac{\partial L}{\partial h_i^{(1)}}$$

where

$$z_j^{(2)} = \sum_i h_i^{(1)} w_{ji}^{(2)} + b_j^{(2)}$$

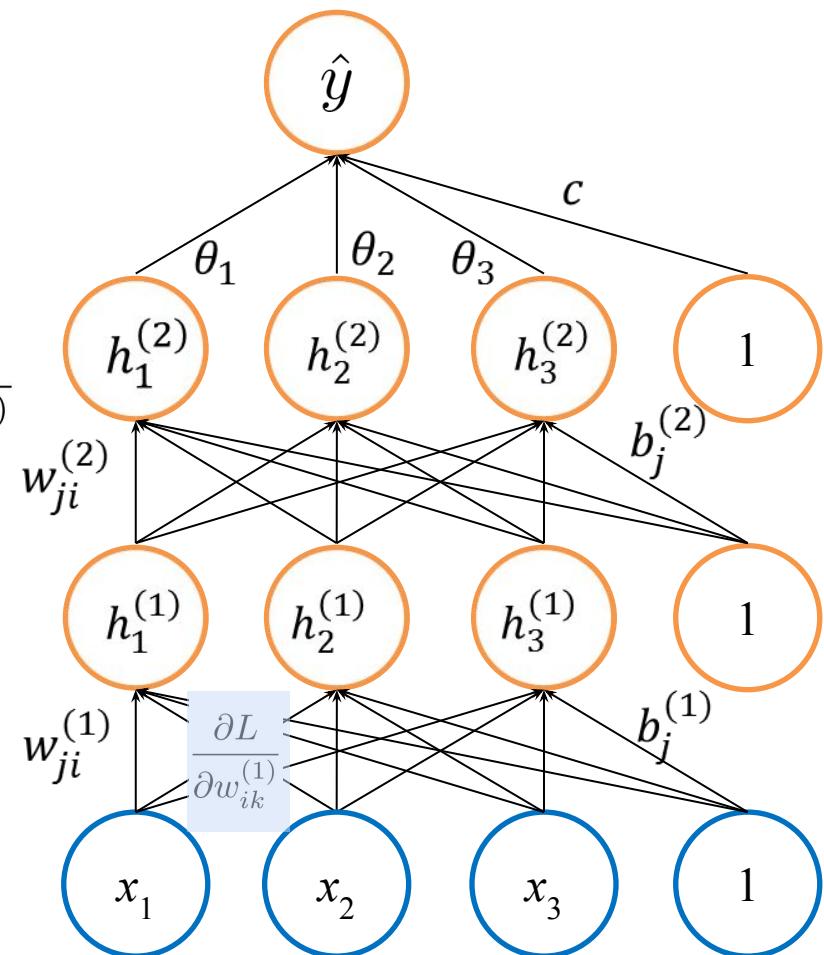
$$z_i^{(1)} = \sum_k x_k w_{ik}^{(1)} + b_i^{(1)}$$

$$h_j^{(1)} = f(\sum_i w_{ji}^{(1)} x_i + b_j^{(1)}), \forall j$$

$$h_j^{(2)} = f(\sum_i w_{ji}^{(2)} h_i^{(1)} + b_j^{(2)}), \forall j$$

$$\hat{y} = \sum_j \theta_j h_j^{(2)} + c$$

$$L = (\hat{y} - y)^2$$



# Backpropagation: examples (NN with 1-hidden layer for classification)

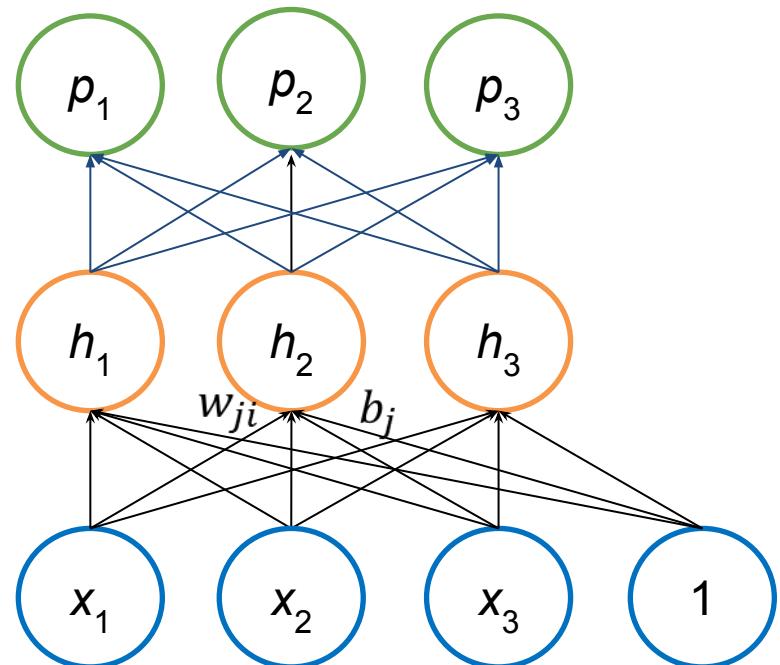
# Forward Propagation

- **Example: a network with 1 hidden layer**

- Input:  $\mathbf{x}$
- Output:  $\mathbf{p}$
- Target:  $\mathbf{y}$
- Loss function: cross entropy  $-\sum_i y_i \log p_i$

Hidden layer     $h_j = f(\sum_i w_{ji}x_i + b_j)$

Output     $p_i = \frac{e^{h_i}}{\sum_j e^{h_j}}$



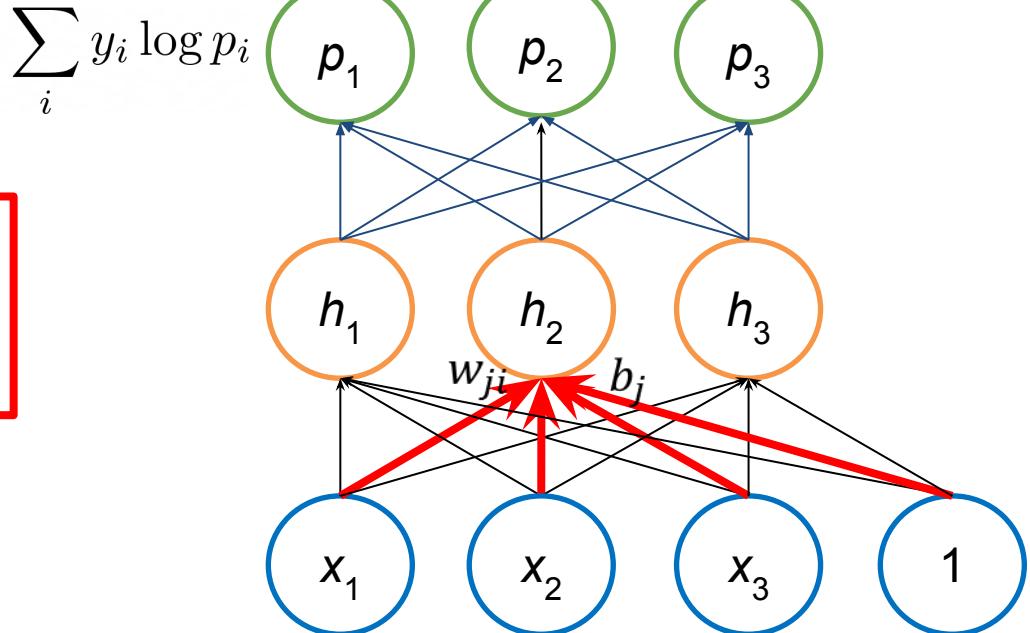
# Forward Propagation

- Example: a network with 1 hidden layer

- Input:  $\mathbf{x}$
- Output:  $\mathbf{p}$
- Target:  $\mathbf{y}$
- Loss function: cross entropy

$$\text{Hidden layer} \quad h_j = f\left(\sum_i w_{ji}x_i + b_j\right)$$

$$\text{Output} \quad p_i = \frac{e^{h_i}}{\sum_j e^{h_j}}$$



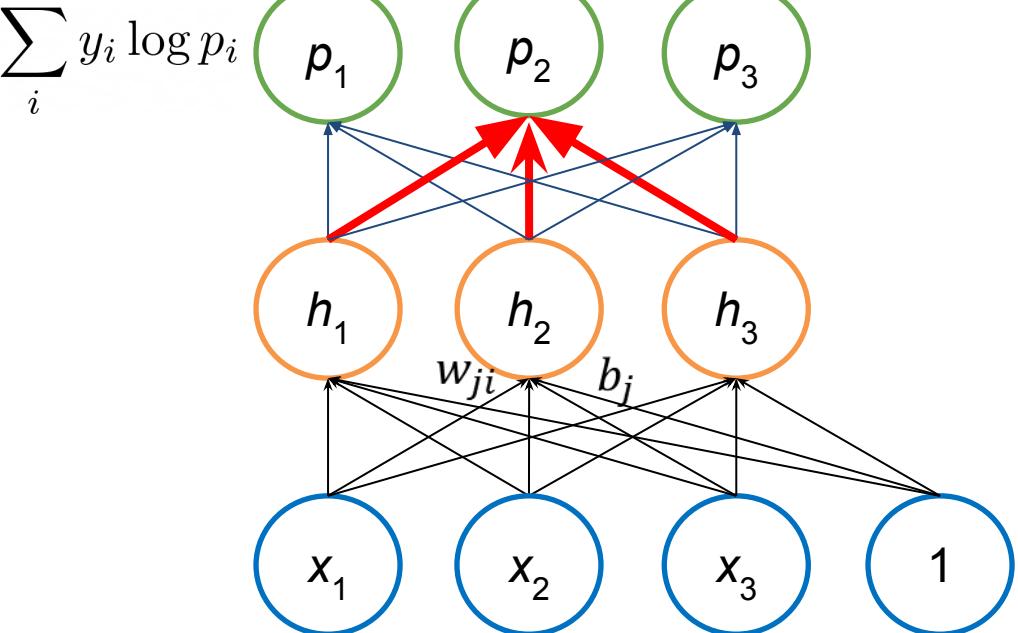
# Forward Propagation

- Example: a network with 1 hidden layer

- Input:  $\mathbf{x}$
- Output:  $\mathbf{p}$
- Target:  $\mathbf{y}$
- Loss function: cross entropy

$$\text{Hidden layer} \quad h_j = f\left(\sum_i w_{ji}x_i + b_j\right)$$

$$\text{Output} \quad p_i = \frac{e^{h_i}}{\sum_j e^{h_j}}$$



# Backpropagation

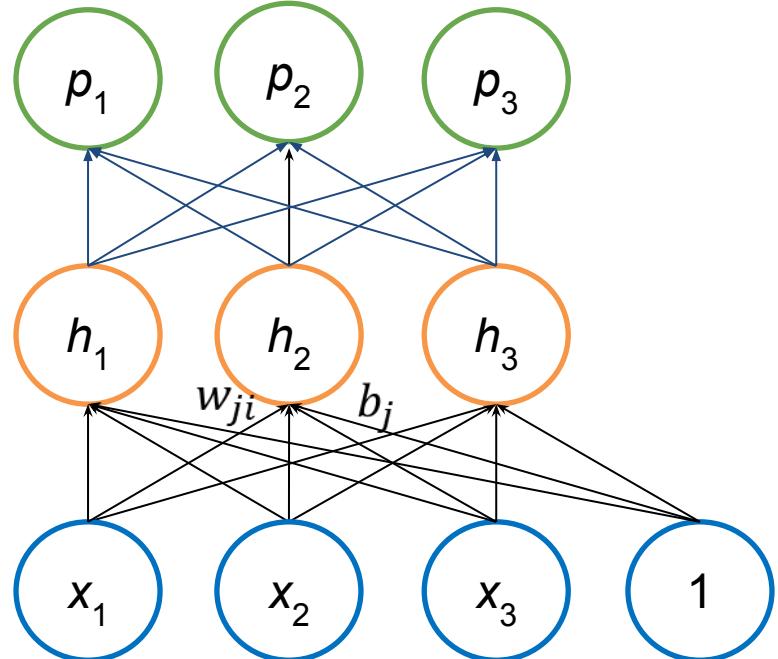
- Goal: compute gradient w.r.t parameters  $\{W, b\}$
- Main idea: apply a chain rule recursively

$$\begin{aligned} L &= - \sum_i y_i \log p_i = - \sum_i y_i \log \frac{e^{h_i}}{\sum_k e^{h_k}} \\ &= - \sum_i y_i h_i + (\sum_i y_i) \log(\sum_k e^{h_k}) \\ &= \log(\sum_k e^{h_k}) - \sum_k y_k h_k \end{aligned}$$

$$\frac{\partial L}{\partial h_j} = \frac{e^{h_j}}{\sum_k e^{h_k}} - y_j = p_j - y_j$$

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial h_j}{\partial w_{ji}} \frac{\partial L}{\partial h_j} = f' x_i \frac{\partial L}{\partial h_j}$$

where  $f' = f'(\sum_i w_{ji} x_i + b_j)$

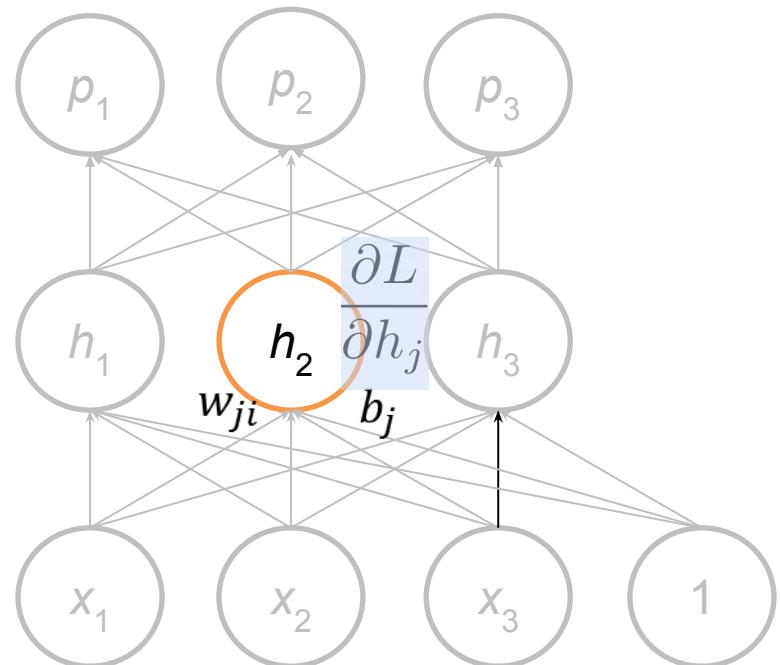


# Backpropagation

- Goal: compute gradient w.r.t parameters  $\{W, b\}$
- Main idea: apply a chain rule recursively

$$\begin{aligned} L &= - \sum_i y_i \log p_i = - \sum_i y_i \log \frac{e^{h_i}}{\sum_k e^{h_k}} \\ &= - \sum_i y_i h_i + (\sum_i y_i) \log(\sum_k e^{h_k}) \\ &= \log(\sum_k e^{h_k}) - \sum_k y_k h_k \end{aligned}$$

$$\frac{\partial L}{\partial h_j} = \frac{e^{h_j}}{\sum_k e^{h_k}} - y_j = p_j - y_j$$



# Backpropagation

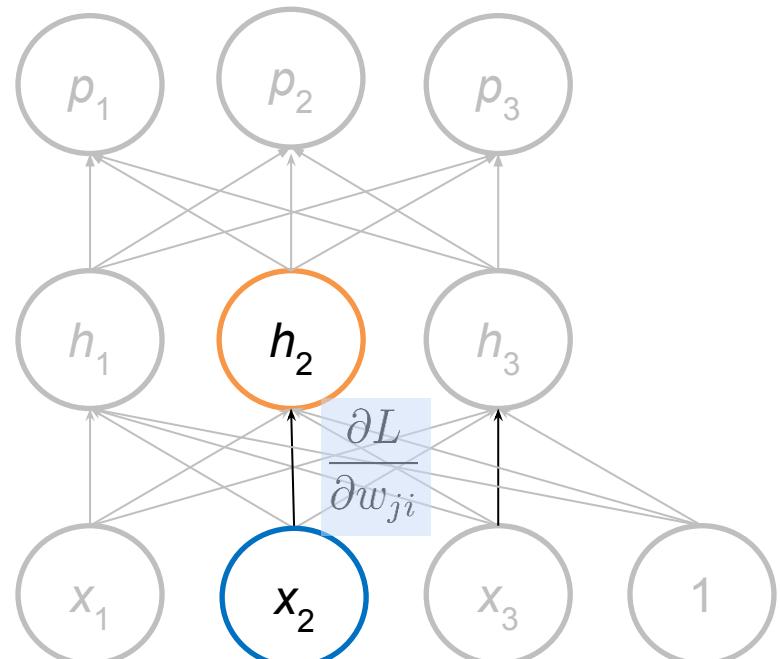
- Goal: compute gradient w.r.t parameters  $\{W, b\}$
- Main idea: apply a chain rule recursively

$$\begin{aligned}
 L &= -\sum_i y_i \log p_i = -\sum_i y_i \log \frac{e^{h_i}}{\sum_k e^{h_k}} \\
 &= -\sum_i y_i h_i + (\sum_i y_i) \log(\sum_k e^{h_k}) \\
 &= \log(\sum_k e^{h_k}) - \sum_k y_k h_k
 \end{aligned}$$

$$\frac{\partial L}{\partial h_j} = \frac{e^{h_j}}{\sum_k e^{h_k}} - y_j = p_j - y_j$$

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial h_j}{\partial w_{ji}} \frac{\partial L}{\partial h_j} = f' x_i \frac{\partial L}{\partial h_j}$$

where  $f' = f'(\sum_i w_{ji}x_i + b_j)$

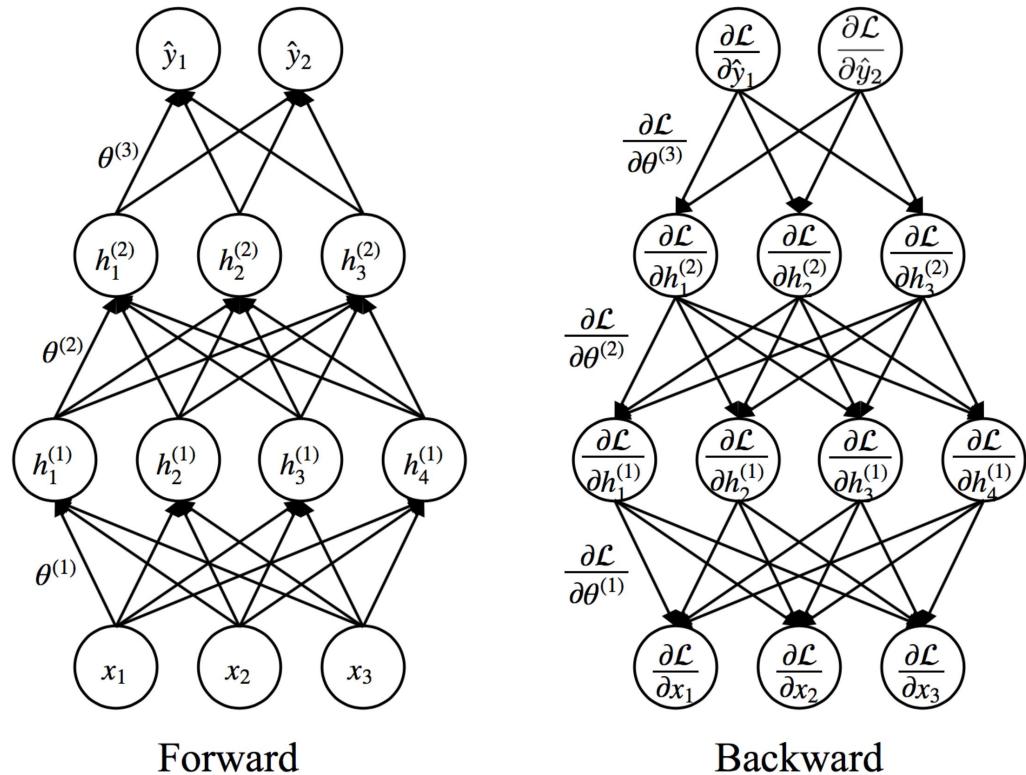


# Deep Neural Networks

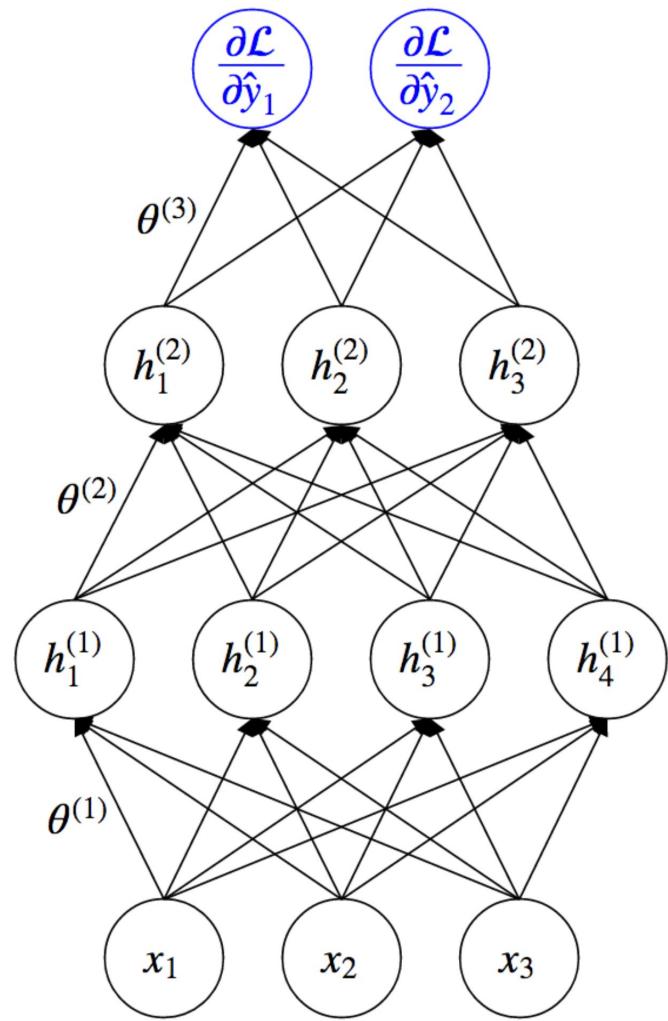
- Construction is straightforward: recursively stacking the blocks of layers
- Computing gradient is straightforward (via back propagation)
  - For general formula, see Bishop's book.

# Backpropagation for deep neural nets

- Computing gradient via **chain rule** for compositional function
- The chain rule can be expressed as a **local computation**
- Think about it as a **computational graph**

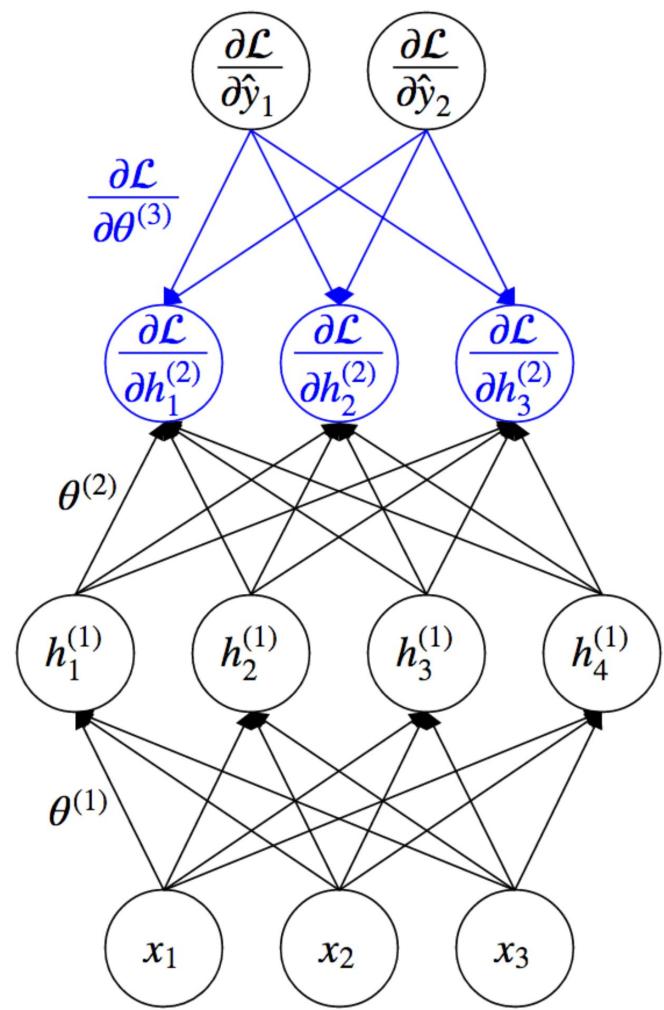


# Backpropagation (for deep neural nets)

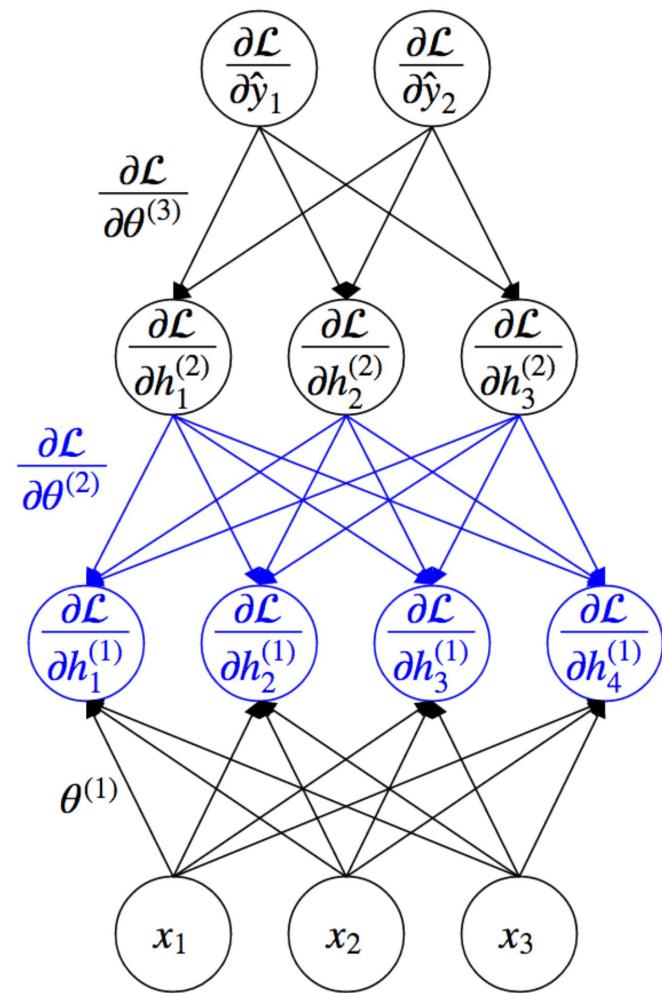


Slide credit: Junhyuk Oh, Jacob Abernethy

# Backpropagation (for deep neural nets)

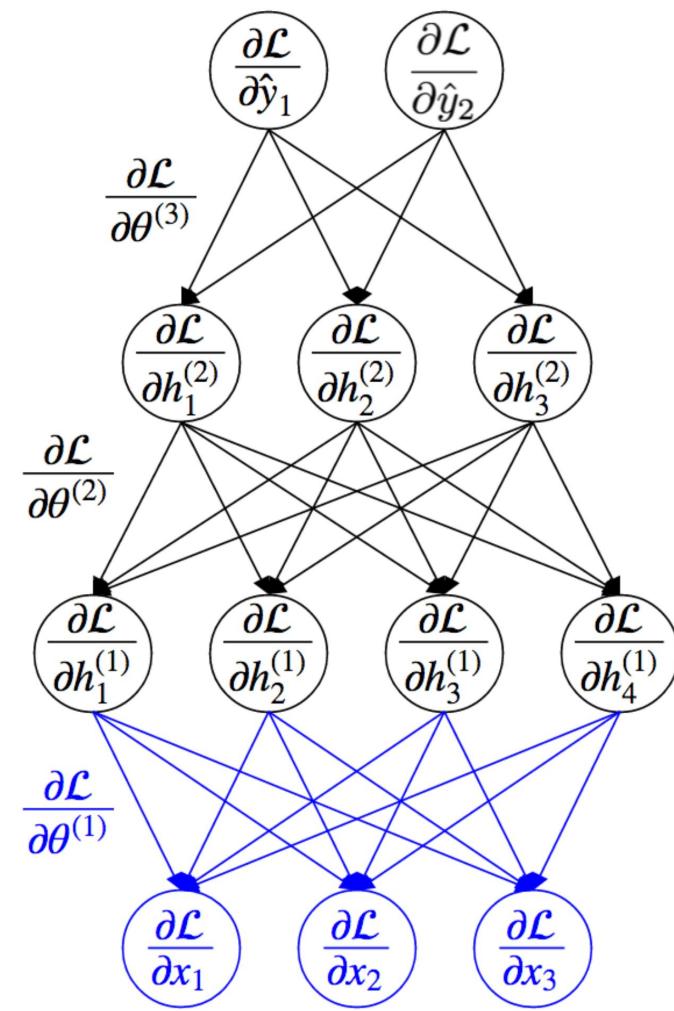


# Backpropagation (for deep neural nets)



Slide credit: Junhyuk Oh, Jacob Abernethy

# Backpropagation (for deep neural nets)



Slide credit: Junhyuk Oh, Jacob Abernethy

# Back-Propagation Algorithm

- Compute  $\nabla_{\mathbf{y}} \mathcal{L} = \left[ \frac{\partial \mathcal{L}}{\partial y_1}, \dots, \frac{\partial \mathcal{L}}{\partial y_n} \right]$  directly from the loss function.
- For each layer (from top to bottom) with output  $\mathbf{h}$ , input  $\mathbf{x}$ , and weights  $\mathbf{W}$ ,
  - Assuming that  $\nabla_{\mathbf{h}} \mathcal{L}$  is given, compute gradients using the **chain rule** as follows:

$$\begin{aligned}\nabla_{\mathbf{W}} \mathcal{L} &= \nabla_{\mathbf{h}} \mathcal{L} \nabla_{\mathbf{W}} \mathbf{h} \\ \nabla_{\mathbf{x}} \mathcal{L} &= \nabla_{\mathbf{h}} \mathcal{L} \nabla_{\mathbf{x}} \mathbf{h}\end{aligned}$$

# Practice: Linear

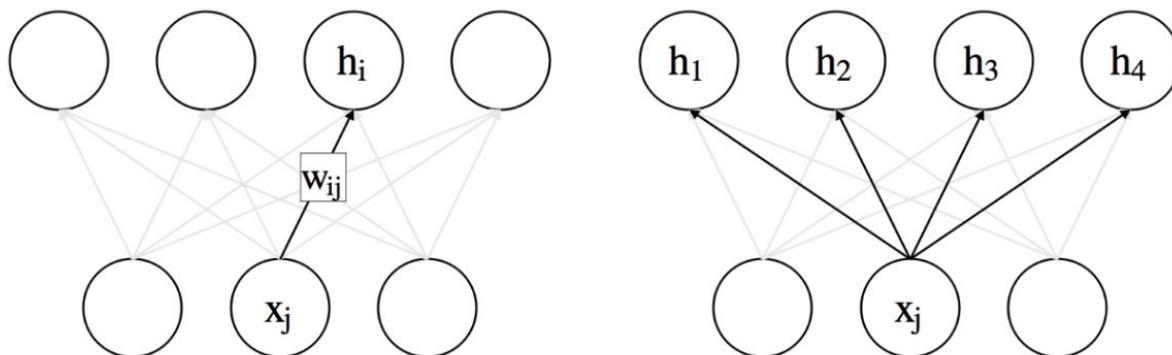
- Forward:  $h_i = \sum_j w_{ij}x_j + b_i \iff \mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{b}$

- Gradient w.r.t. parameters

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial h_i} \frac{\partial h_i}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial h_i} x_j \iff \nabla_{\mathbf{W}} \mathcal{L} = \nabla_{\mathbf{h}} \mathcal{L} \mathbf{x}^T$$
$$\nabla_{\mathbf{b}} \mathcal{L} = \nabla_{\mathbf{h}} \mathcal{L}$$

- Gradient w.r.t. inputs

$$\frac{\partial \mathcal{L}}{\partial x_j} = \sum_i \frac{\partial \mathcal{L}}{\partial h_i} \frac{\partial h_i}{\partial x_j} = \sum_i \frac{\partial \mathcal{L}}{\partial h_i} w_{ij} \iff \nabla_{\mathbf{x}} \mathcal{L} = \mathbf{W}^T \nabla_{\mathbf{h}} \mathcal{L}$$

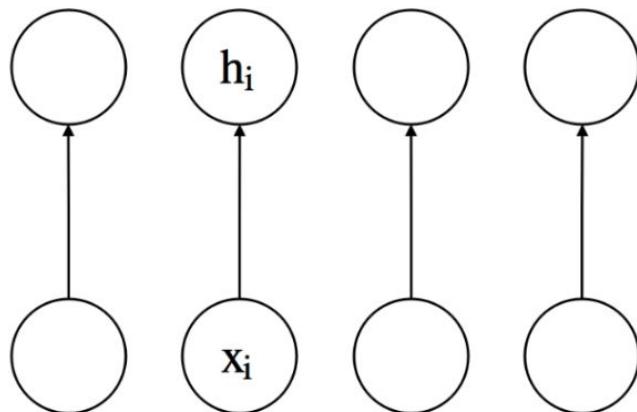


# Practice: Non-Linear Activation (Sigmoid)

- Forward:  $h_i = \sigma(x_i) = \frac{1}{1+\exp(-x_i)} \iff \mathbf{h} = \sigma(\mathbf{x})$

- Gradient w.r.t. inputs

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial h_i} \frac{\partial h_i}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial h_i} \sigma(x_i)(1 - \sigma(x_i)) = \frac{\partial \mathcal{L}}{\partial h_i} h_i(1 - h_i)$$
$$\nabla_{\mathbf{x}} \mathcal{L} = \nabla_{\mathbf{h}} \mathcal{L} \odot \mathbf{h} \odot (\mathbf{1} - \mathbf{h})$$

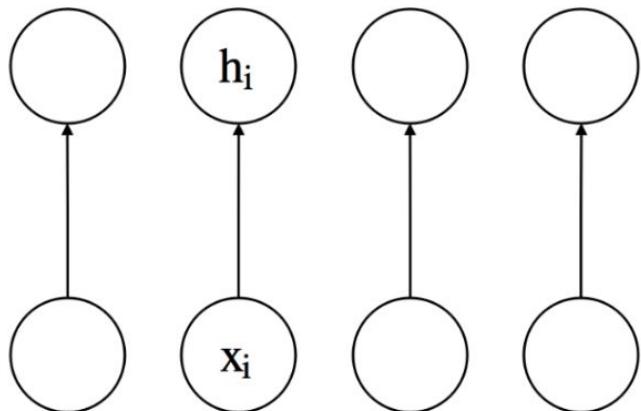


# Practice: Non-Linear Activation (tanh)

- Forward:  $h_i = \tanh(x_i) = \frac{\exp(x_i) - \exp(-x_i)}{\exp(x_i) + \exp(-x_i)} \iff \mathbf{h} = \tanh(\mathbf{x})$
- Gradient w.r.t inputs

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial h_i} \frac{\partial h_i}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial h_i} (1 - \tanh^2(x_i)) = \frac{\partial \mathcal{L}}{\partial h_i} (1 - h_i^2)$$

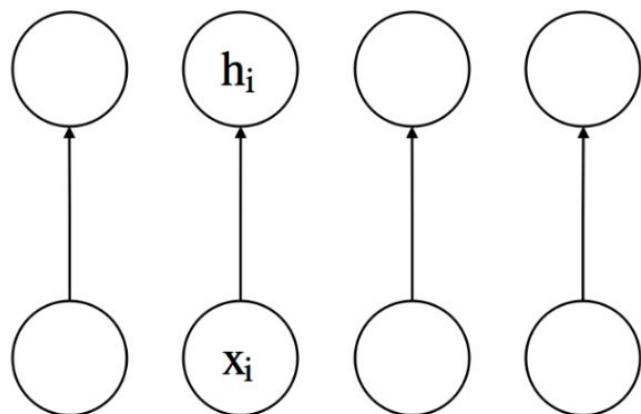
$$\nabla_{\mathbf{x}} \mathcal{L} = \nabla_{\mathbf{h}} \mathcal{L} \odot (\mathbf{1} - \mathbf{h} \odot \mathbf{h})$$



# Practice: Non-Linear Activation (relu)

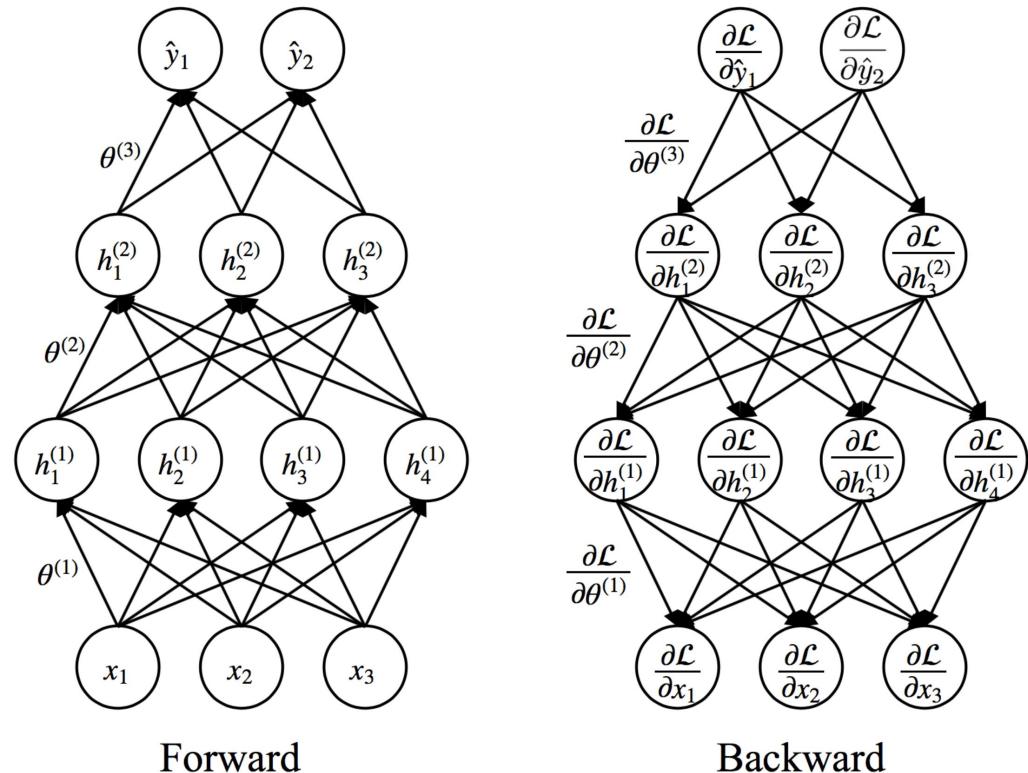
- Forward:  $h_i = \text{relu}(x_i) = \max(x_i, 0) \iff \mathbf{h} = \text{relu}(\mathbf{x})$
- Gradient w.r.t inputs

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial h_i} \frac{\partial h_i}{\partial x_i} = \begin{cases} \frac{\partial \mathcal{L}}{\partial h_i} & x_i > 0 \\ 0 & x_i < 0 \end{cases}$$



# Backpropagation for deep neural nets

- Computing gradient via **chain rule** for compositional function
- The chain rule can be expressed as a **local computation**
- Think about it as a **computational graph**



# Problems with Back Propagation

- Gradient is progressively getting more dilute
  - Below top few layers, correction signals can be significantly reduced
- Easy to get stuck in local minima
  - Especially since they start out far from ‘good’ regions (i.e., random initialization)

# Back Propagation & Amounts of data

- Typically requires lots of labeled data
- Given limited amounts of labeled data, training via backpropagation does not work well
  - Deep networks trained with backpropagation (without any sort of unsupervised pretraining) sometimes perform worse than shallow networks – Overfitting
- However, when there is a large amount of labeled data, backpropagation works surprisingly well.
  - Example of success: Convolutional Neural Networks trained from ImageNet classification (~1M labeled images from 1000 classes)