Lecture 4: FFNNs, Error Backdrop, Midterm Review.

Cornell Bowers C·IS
Computer Science

Tanya Goyal

CS 4740 (and crosslists): Introduction to Natural Language Processing

Midterm

- **Closed-book**. No calculators (will not be needed)
- The test will be 2 hours long and will cover all material **up to and including error backpropagation.**
- The written questions in the HWs + practice are good representative questions for the midterm.
- I will include 2 question **very** similar to the HW questions. Make sure to understand them well!

Today

- Recap: Feed Forward Neural Networks
- Deep Averaging Networks
- Error Backpropagation

Feed-forward neural networks (FFNNs) for classification

Given a input representation

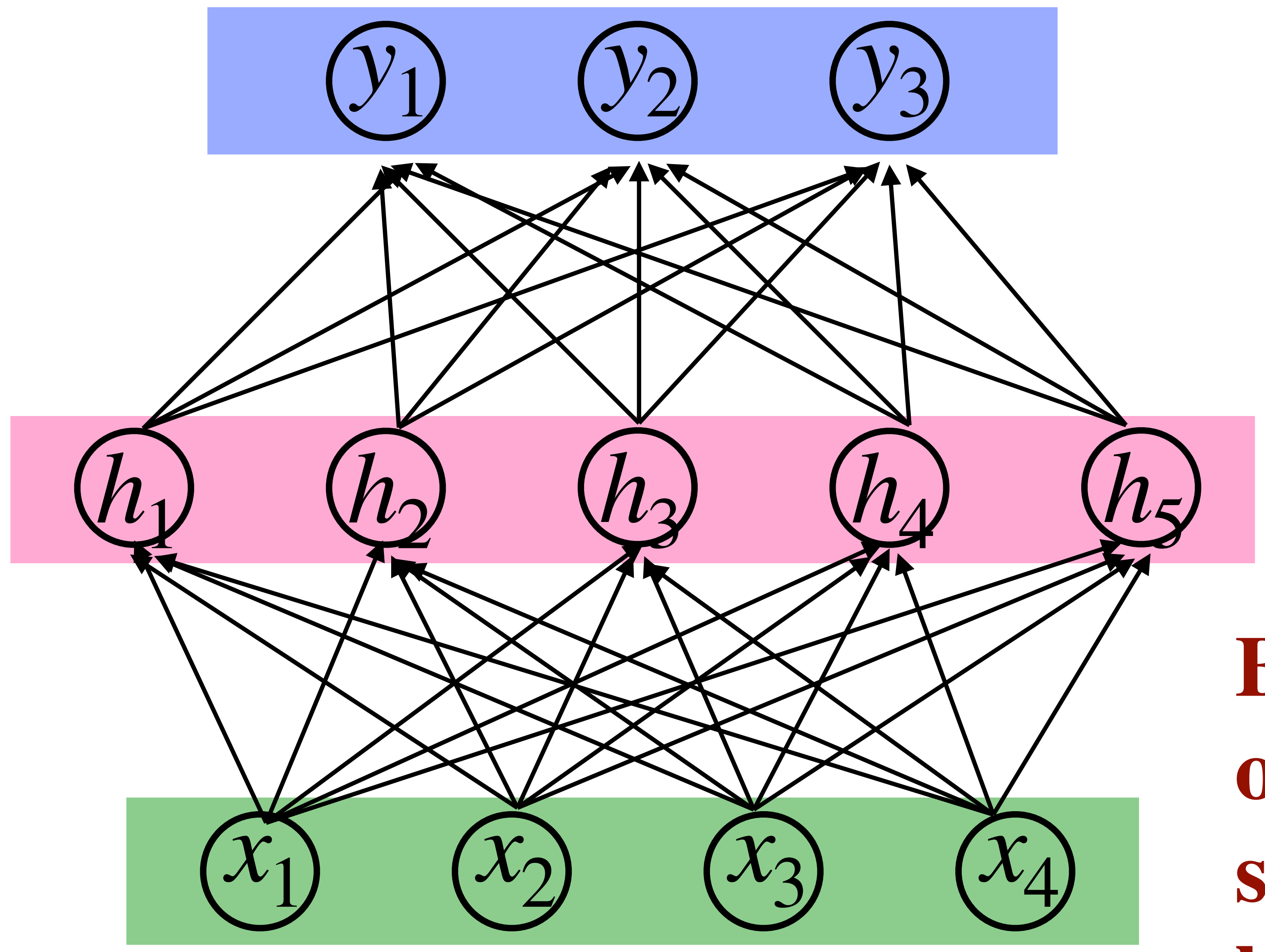
$$\mathbf{x} = [x_1, \cdots x_i, \cdots x_d]$$

New vector representation

$$\mathbf{h} = [h_1, \cdots h_i, \cdots h_{d_1}]$$

Predict: probability distribution over the labels

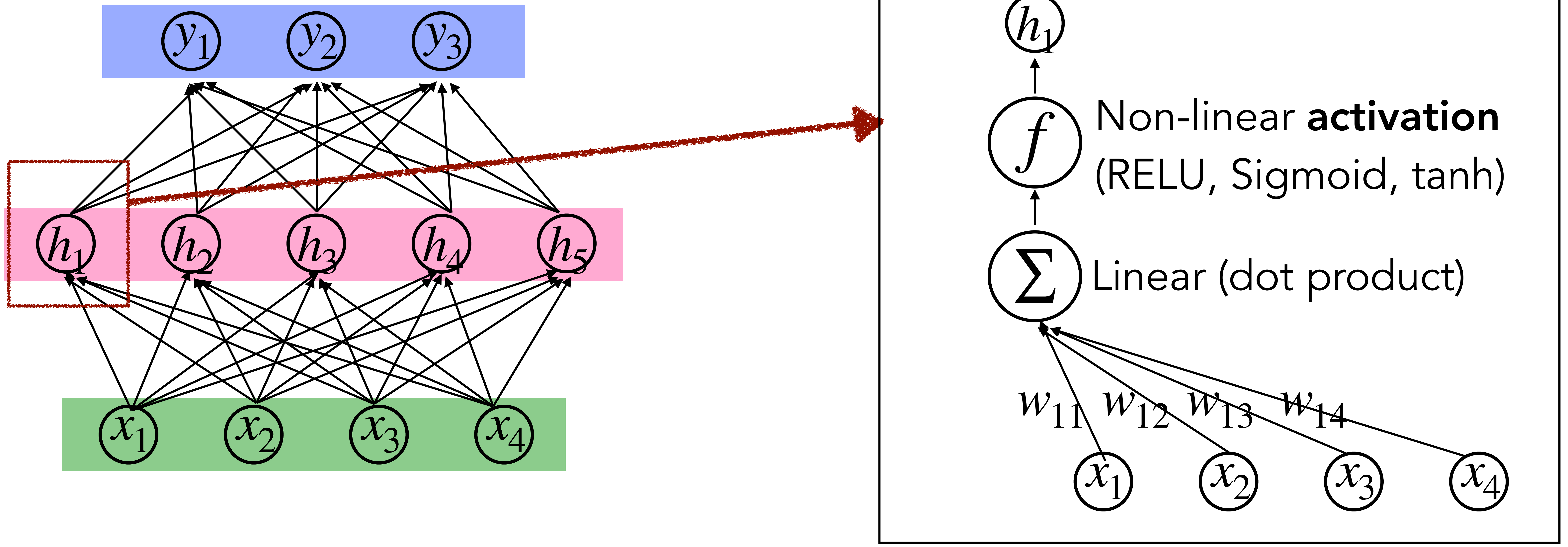
$$\mathbf{y} = [y_1 = P(\text{LOC} \mid \mathbf{x}), P(\text{PER} \mid \mathbf{x}), P(\text{NULL} \mid \mathbf{x})]$$



Bias term omitted for simplicity, let's come back to this.

Intuition: FFNNs learn how to better represent **x** (i.e, as **h**) so as to be able to accurately predict **y**.

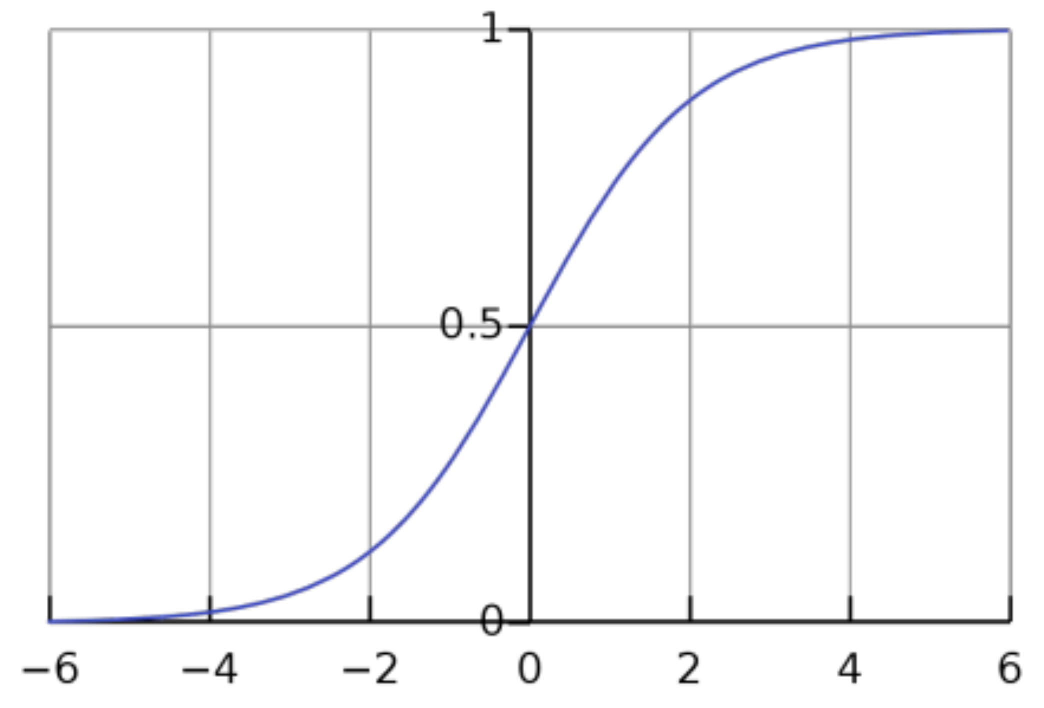
Feed-forward neural networks (FFNNs)



Activation functions

sigmoid

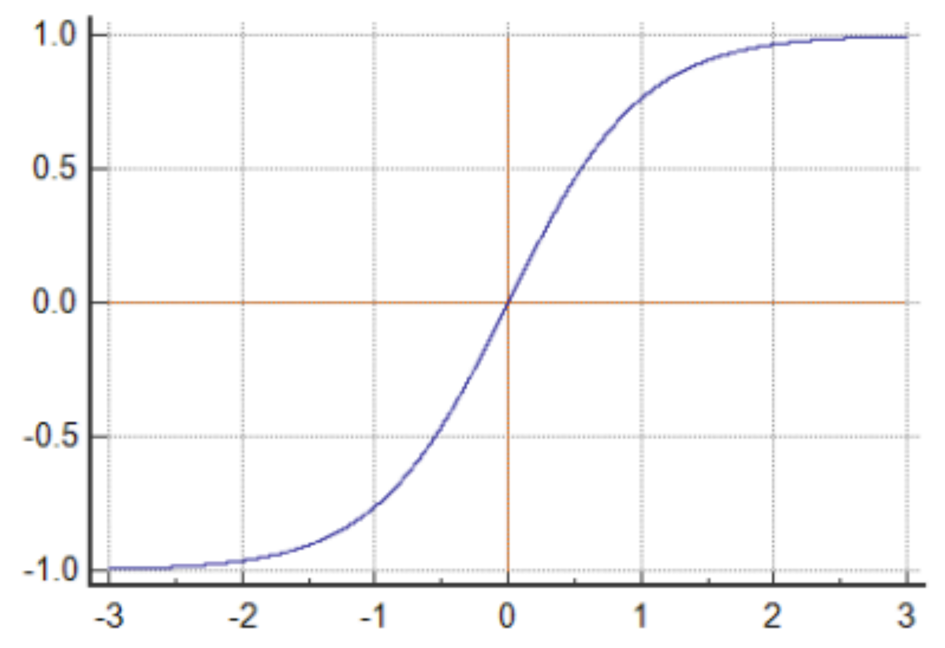
$$f(z) = \frac{1}{1 + e^{-z}}$$



$$f'(z) = f(z) \times (1 - f(z))$$

tanh

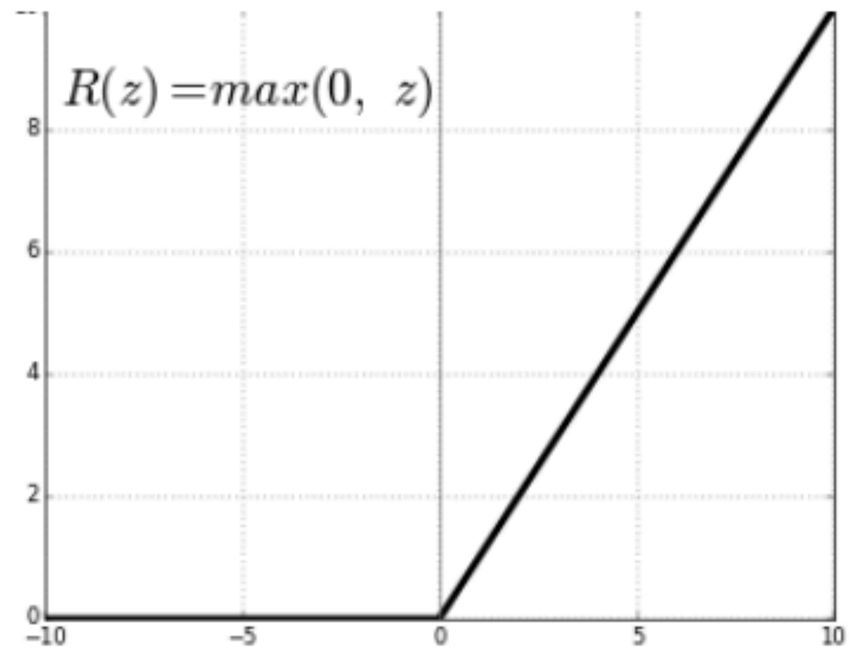
$$f(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$



$$f'(z) = 1 - f(z)^2$$

ReLU
(rectified linear unit)

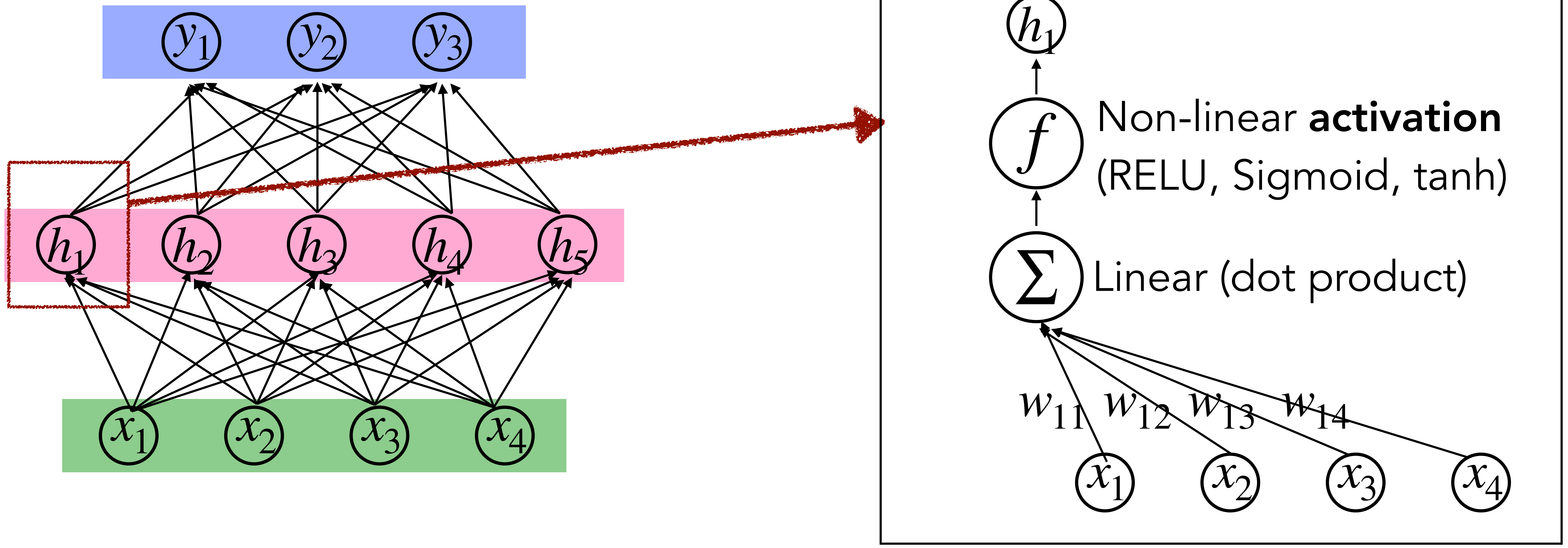
$$f(z) = \max(0, z)$$



$$f'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$$

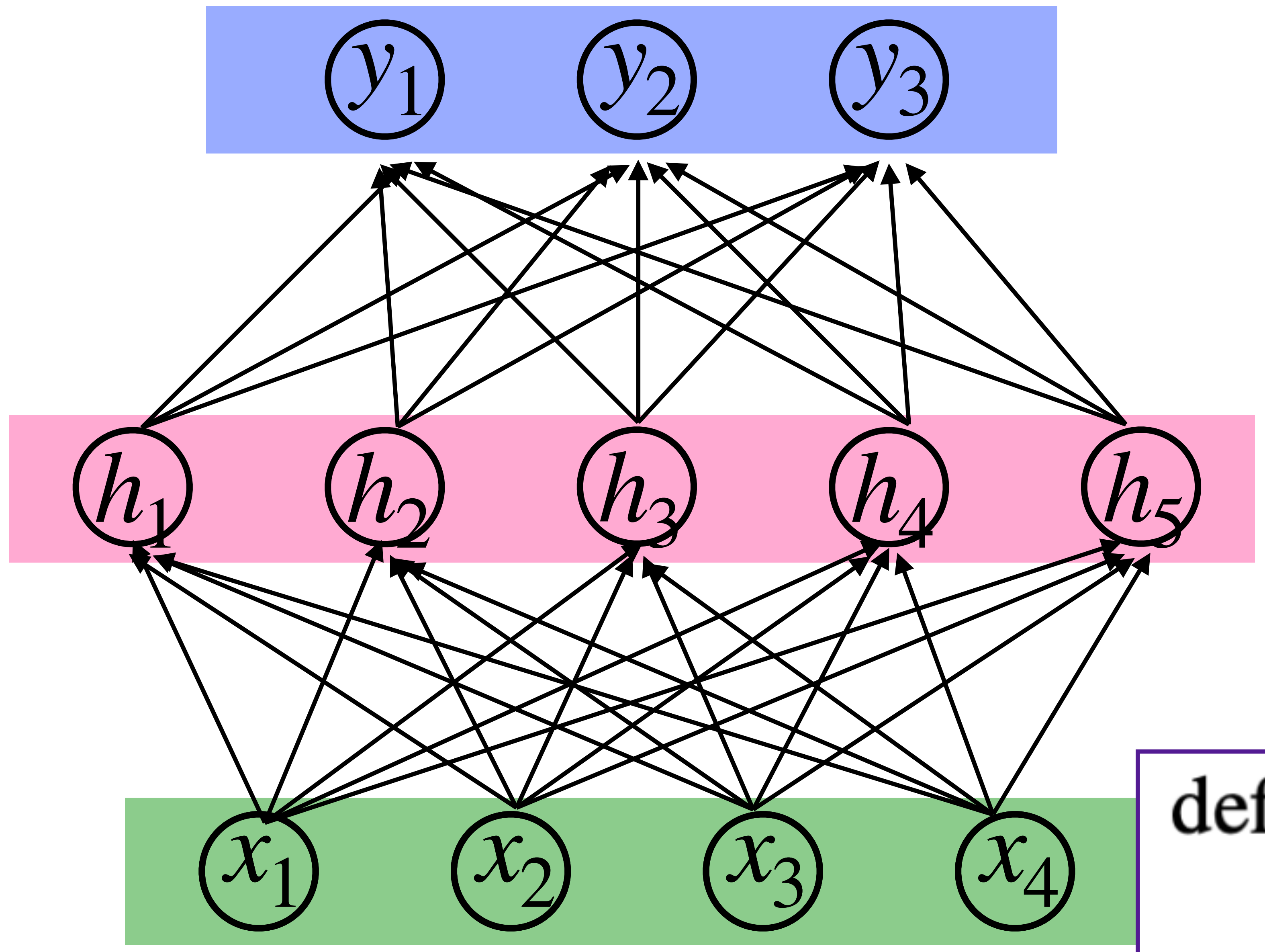
- Applied element-wise: $f([z_1, z_2, .. z_n]) = [f(z_1), f(z_2) ... f(z_n)]$

Feed-forward neural networks (FFNNs)

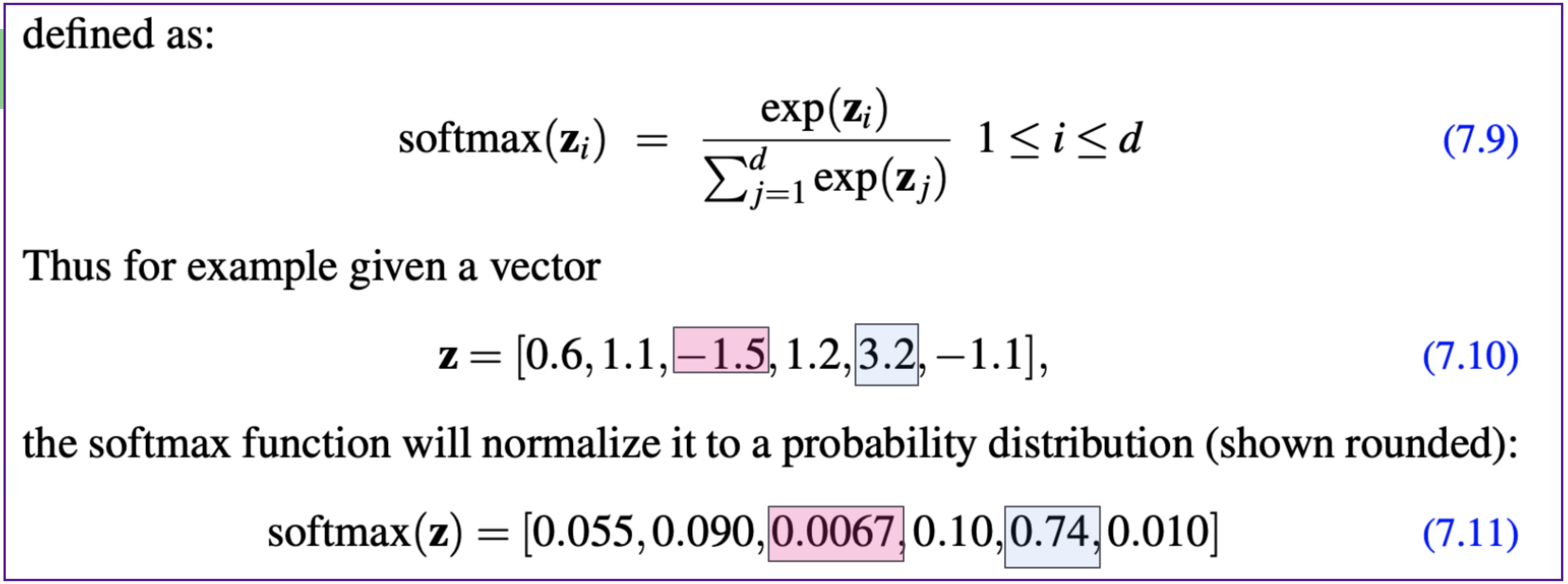


- $h_i = f(\sum_j w_{ij} x_j)$
- Or, in matrix form: $x = [x_1 x_2 \ldots x_d]^T$, $w_i = [w_{i1}, w_{i2}, \ldots w_{id}]$, then $h_i = f(w_i x)$

Feed-forward neural networks (FFNNs), Matrix version



- Input layer $x = [x_1 x_2 \ldots x_4]^T$
- Hidden Layer:
  - $h = f(Wx) \quad \in R^5$
  - $W \in \mathbb{R}^?$



defined as:

$$\text{softmax}(\mathbf{z}_i) = \frac{\exp(\mathbf{z}_i)}{\sum_{j=1}^{d} \exp(\mathbf{z}_j)} \quad 1 \leq i \leq d \tag{7.9}$$

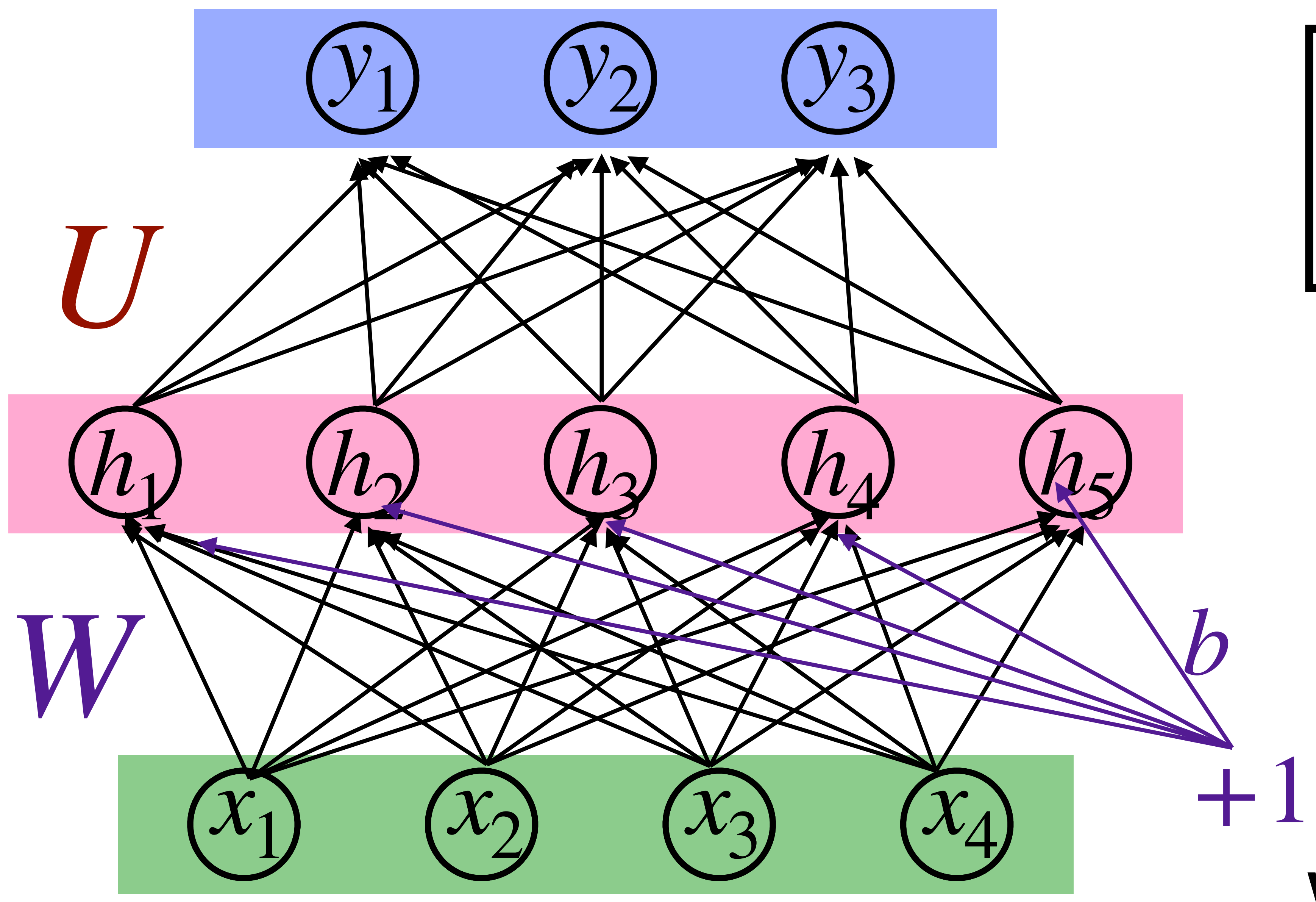
Thus for example given a vector

$$\mathbf{z} = [0.6, 1.1, \boxed{-1.5}, 1.2, \boxed{3.2}, -1.1], \tag{7.10}$$

the softmax function will normalize it to a probability distribution (shown rounded):

$$\text{softmax}(\mathbf{z}) = [0.055, 0.090, \boxed{0.0067}, 0.10, \boxed{0.74}, 0.010] \tag{7.11}$$

Feed-forward neural networks (FFNNs), matrix version



U and W are weights that are learnt during training.

What is this ***b*** term and the extra input value that is always a 1?

Number of layers = Number of learnable layers (2 in this figure)

Parameters, Hyperparameters

- What are parameters? Weights that are learned during training.
- What are hyperparameters? Set by the model developer, e.g. dimensions, learning rates, number of epochs, etc.

Quiz 1: Dimensionality

My input x has dimension 100. I want to classify it as one of 4 labels. I designed a 2 layer FFNN.

Q1. Is this dimension information complete to construct the FFNN?

Q2: What is the dimensions of the weight matrices and the bias matrices?

Q3: Which weights are “learnable”?

Big question: how do we learn the weights?

- In the previous slides, we saw how to derive the final output $\hat{y}$ given an input x and the weights (W and b) for all layers.
- How do we learn these weights?
- Similar strategy to how we learned weights for logistic regression:

```
Initialize $W$ and $b$ randomly.
Repeat
For each training datapoint $(x, y)$
        1. Feedforward pass to compute output $\hat{y}$
        2. Compute error, i.e. loss$(y, \hat{y})$
        3. Update **weights in the network to reduce error.**
Until a stopping criterion is reached.
Return the learned $W$ and $b$
```

Quiz detour.

Constructing Input for different Tasks

In previous slides, we assumed we know how to represent the input text as a vector x.

Consider the following sentiment classification task:

- Input: $w_1 w_2 w_3 \ldots w_n$ “The move was great”
- Output: $y =$ “positive”. Y = {“postive”, “negative”, “neutral”}

How should we convert the text input “The move was great” to a vector representation x?

Constructing Input for different Tasks

In previous slides, we assumed we know how to represent the input text as a vector x.

Consider the following sentiment classification task:

- Input: $w_1 w_2 w_3 \ldots w_n$ "The move was great"
- Output: $y =$ "positive". Y = {"postive", "negative", "neutral"}

How should we convert the text input "The move was great" to a vector representation x?

Simpler question? What can we do if the input is a single word. Word Embeddings!

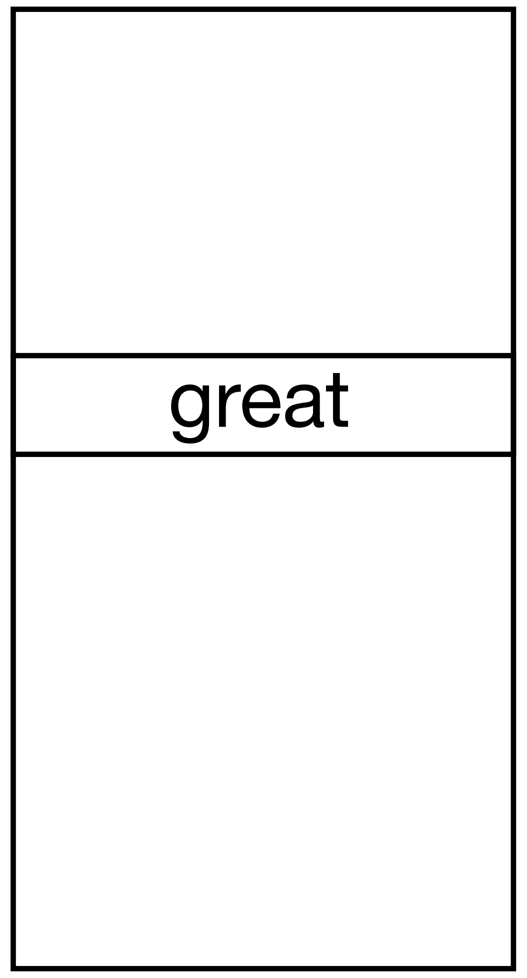
Constructing Input for different Tasks

Simpler question? What can we do if the input is a single word. Word Embeddings!

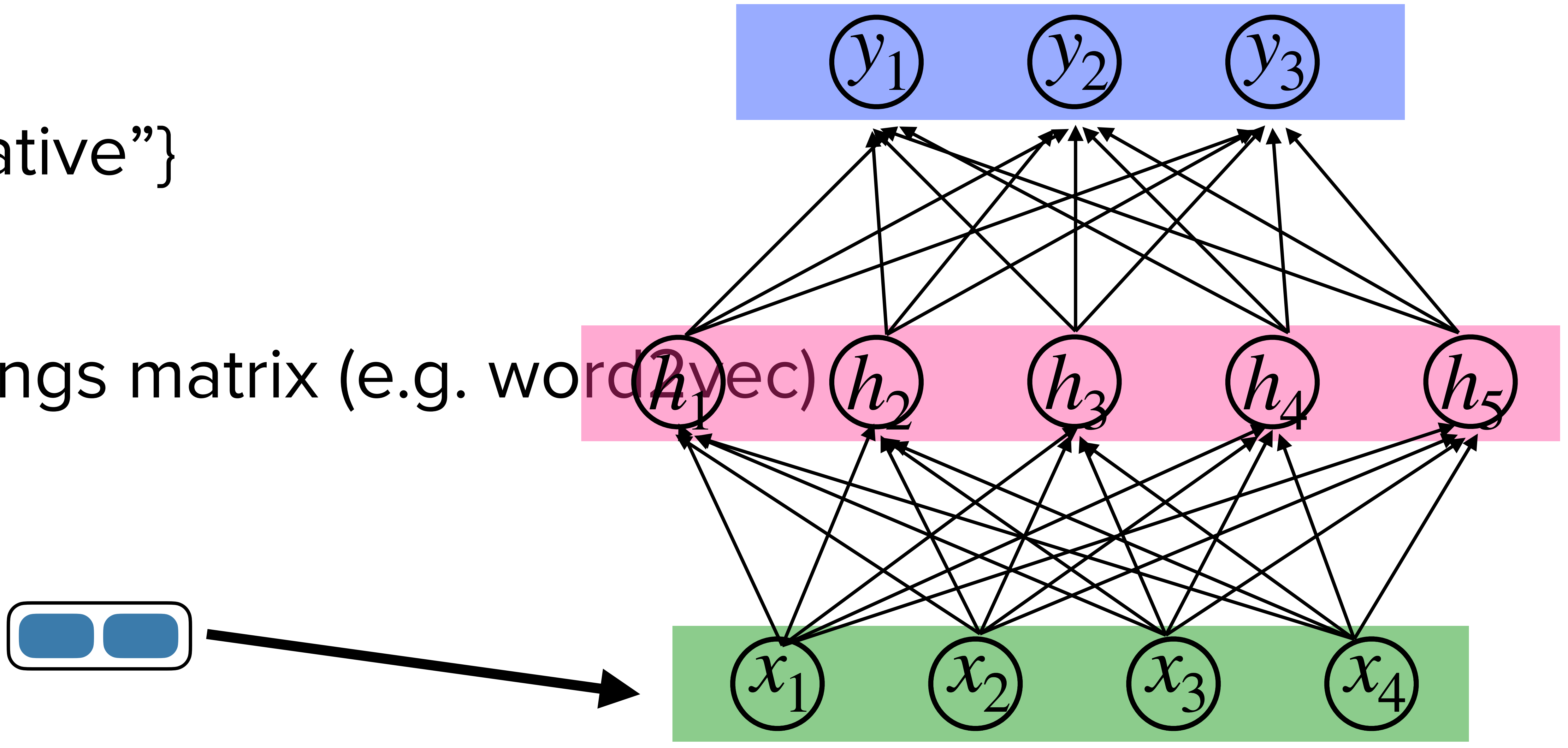
- Input: w_1 "great"
- Output: $y =$ "positive". Y = {"postive", "negative"}

- Let E be an embeddings function / Embeddings matrix (e.g. word2vec)
- Represent w_1 as a one-hot encoding

$[0,0,...1...0] \times$



$= e_{\text{great}}$

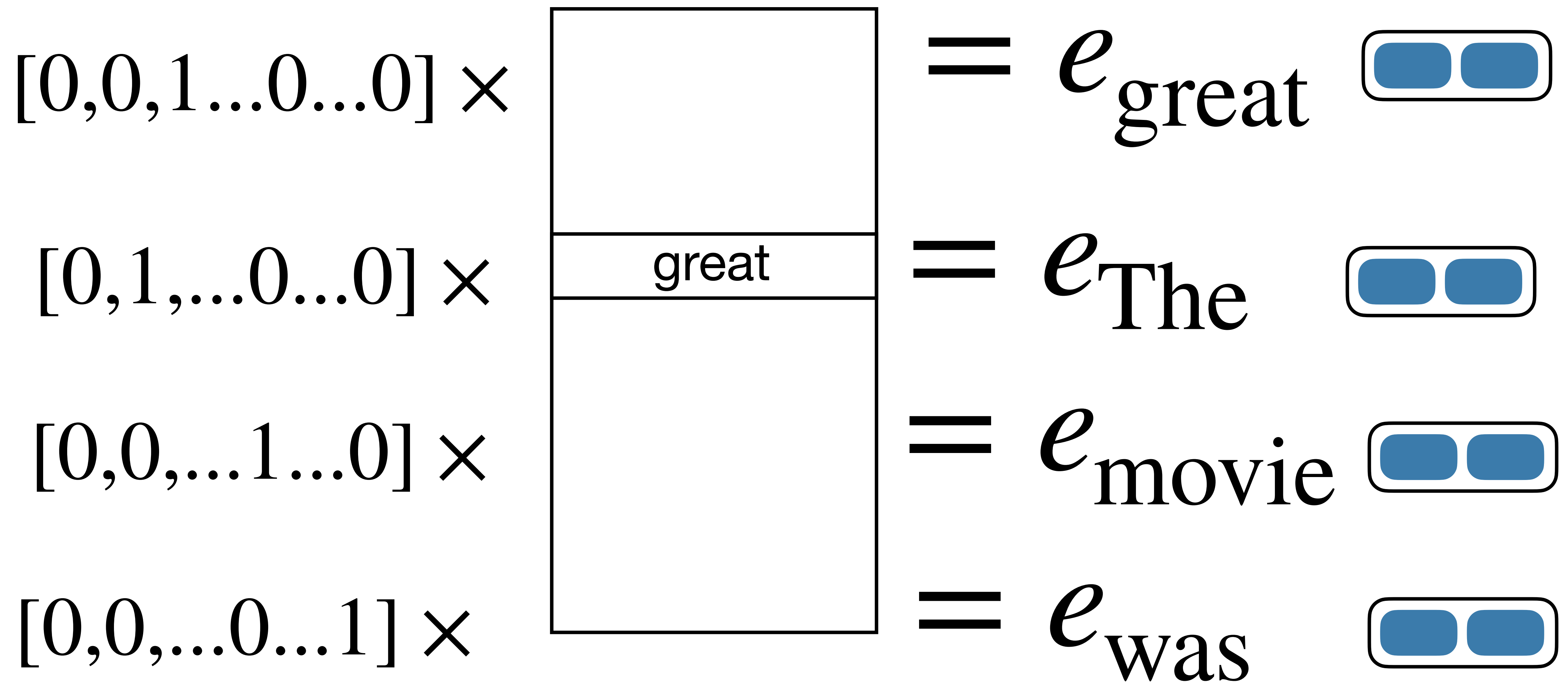


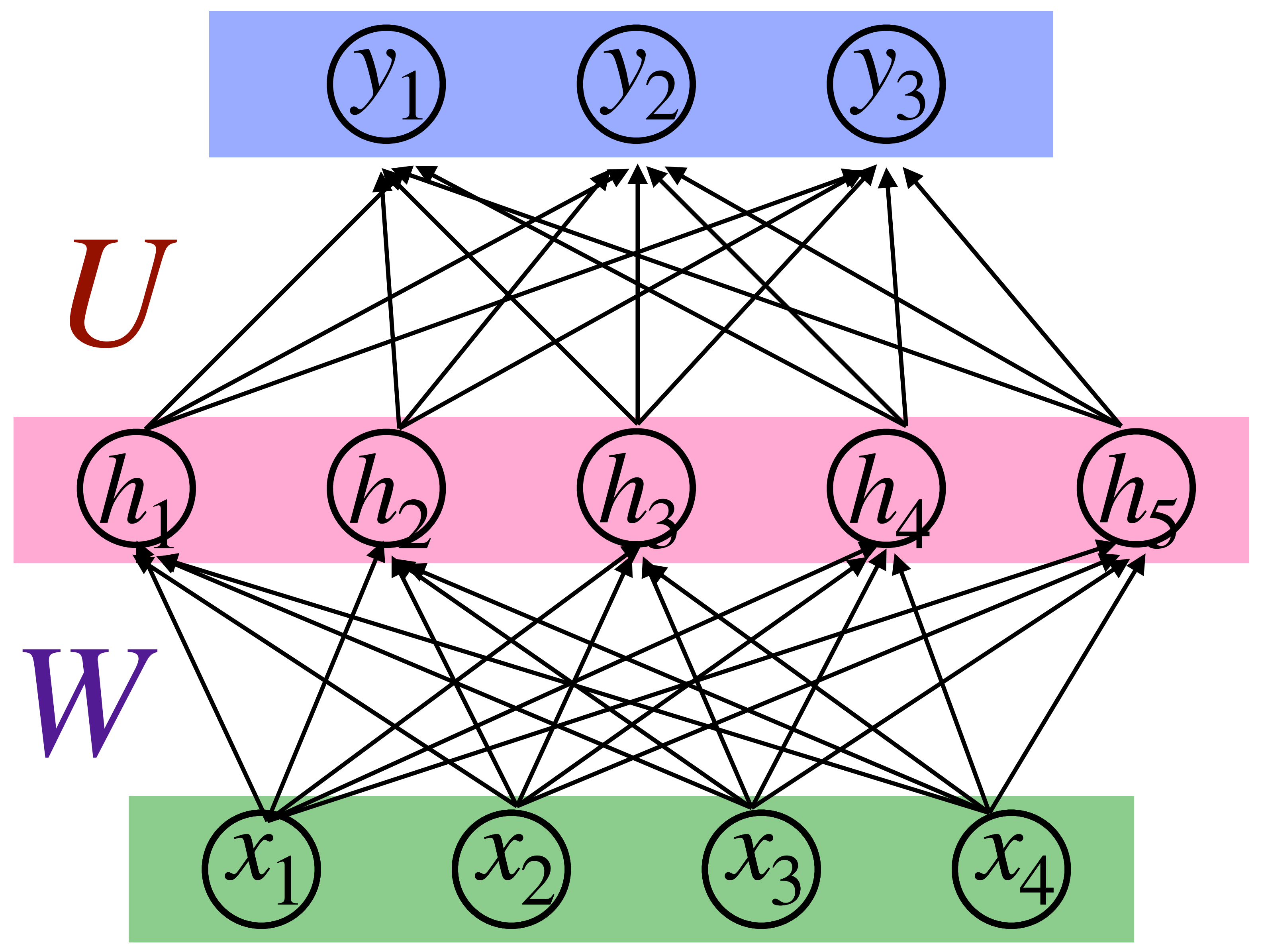
Deep Averaging Network

How should we convert the text input “The move was great” to a vector representation x?

Consider the following sentiment classification task:

- Input: $w_1 w_2 w_3 \ldots w_n$ “The move was great”
- Output: $y =$ “positive”. Y = {“postive”, “negative”, “neutral”}



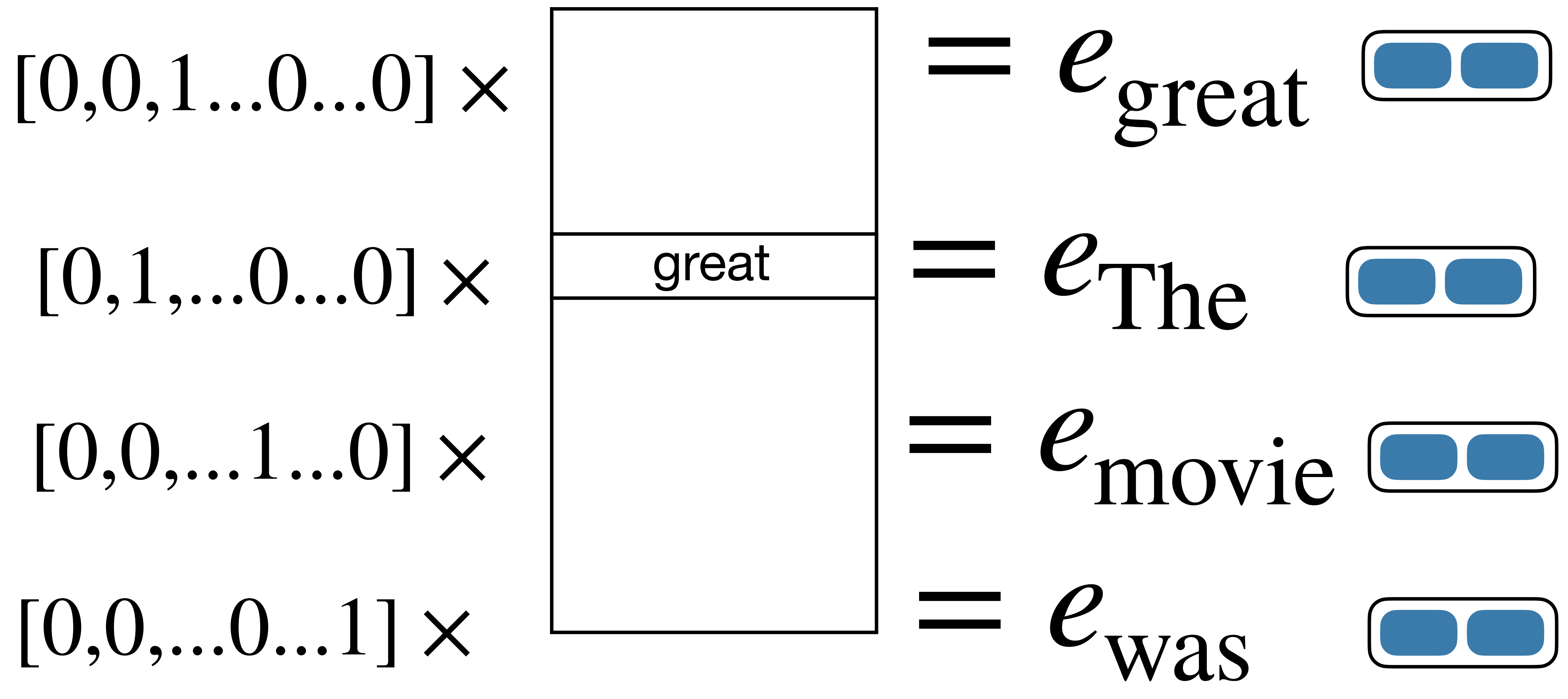


Deep Averaging Network

How should we convert the text input "The move was great" to a vector representation x?

Consider the following sentiment classification task:

- Input: $w_1 w_2 w_3 \dots w_n$ "The move was great"
- Output: $y =$ "positive". Y = {"postive", "negative", "neutral"}

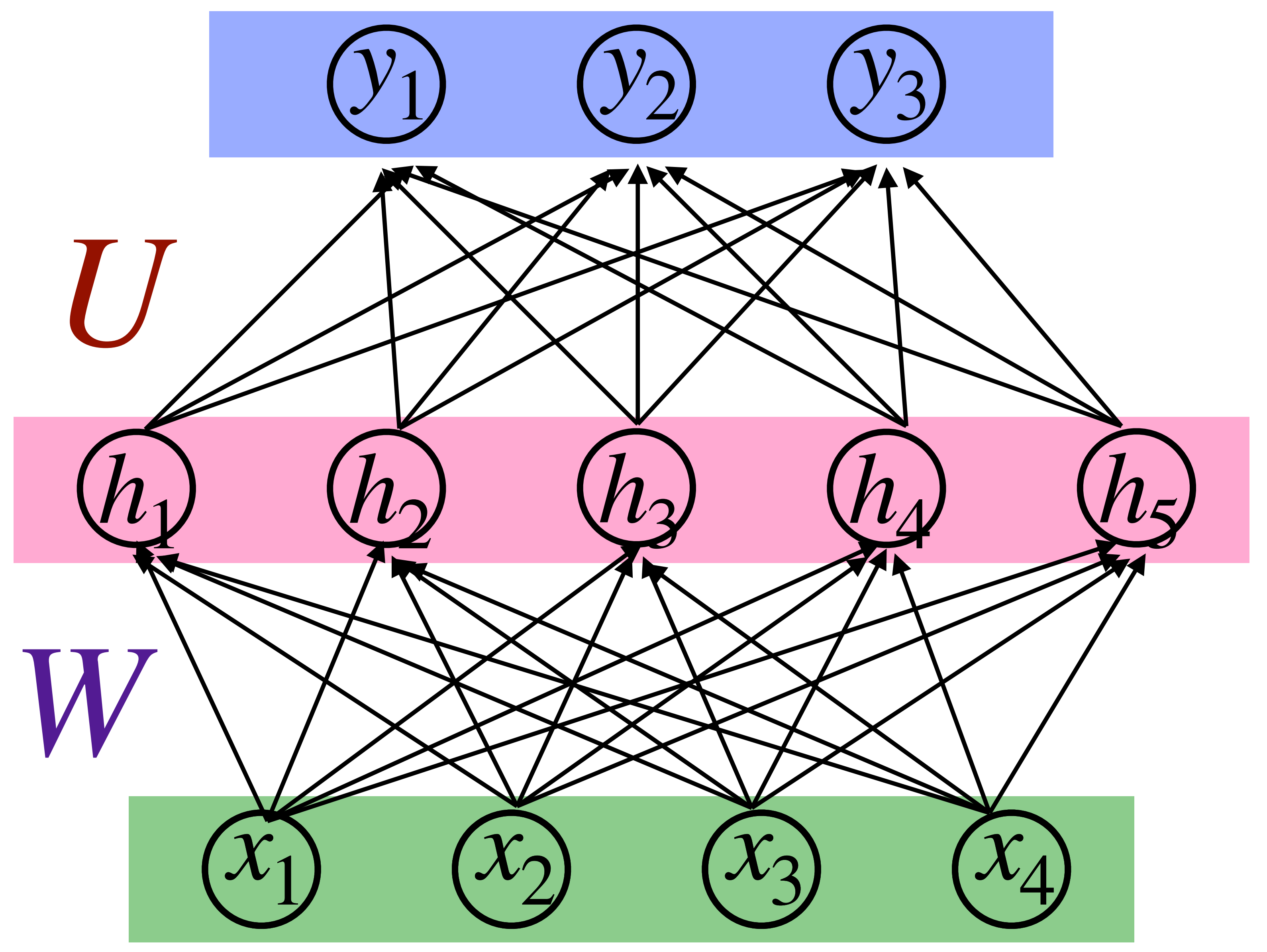


$$[0,0,1...0...0] \times = e_{\text{great}}$$

$$[0,1,...0...0] \times = e_{\text{The}}$$

$$[0,0,...1...0] \times = e_{\text{movie}}$$

$$[0,0,...0...1] \times = e_{\text{was}}$$



$$x = \frac{1}{n} \sum_{i=1:n} e_{w_i}$$

Quiz: Neural N-gram models

Consider constructing a feedforward neural network (FFNN) for the task of trigram language modeling.

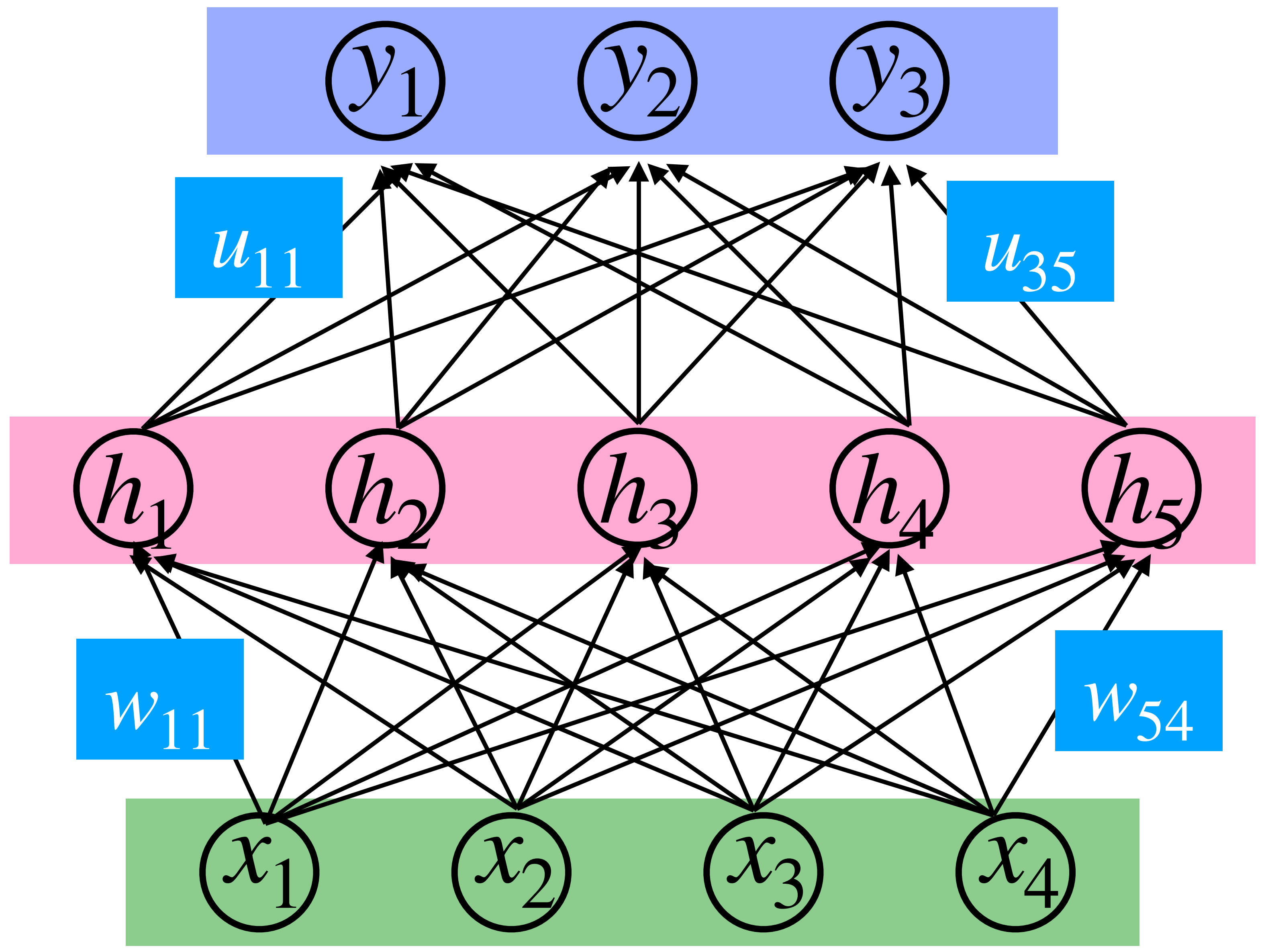
(a) Provide a diagram of your network.

(b) What is purpose of the input layer of your network? What is its dimension (i.e., size)?

(c) What is purpose of the output layer of your network? What is its dimension (i.e., size)?

Big question: how do we learn the weights?

```
Initialize $W$ and $b$ randomly.
Repeat
    For each training datapoint $(x, y)$
        1. Feedforward pass to compute output $\hat{y}$
        2. Compute error, i.e. loss$(y, \hat{y})$
        3. Update **weights in the network to reduce error.**
Until a stopping criterion is reached.
Return the learned $W$ and $b$
```



- We want to learn weight updates to all "parameters" like $w_{11}, \ldots w_{54} \ldots u_{11} \ldots u_{35}, b_1 \ldots b_n$
- We need to know $\dfrac{\partial L}{\partial w_{11}}$, etc. to update the weights.

Challenge: We need to compute this derivative wrt weight parameters that appear in very early layers of the network, even though loss is computed at the very end.

Solution: Backpropagation

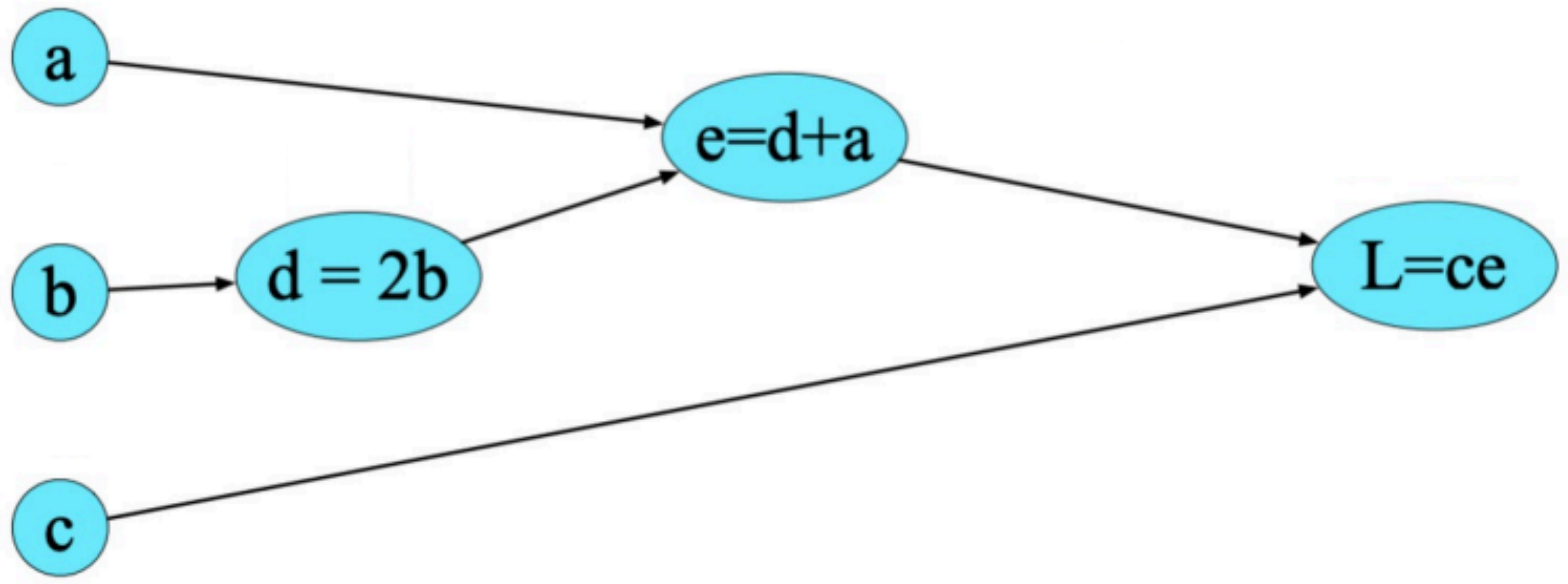
- Computationally efficient algorithm (automatic differentiation)
- Implemented using **computation graphs.**
    - A representation of the process of computing a mathematical expression.
    - Each operation is modeled as a node in a graph.
    - Useful to think of computation graphs as modeling data dependencies

Computational Graphs

Computational Graph Construction

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

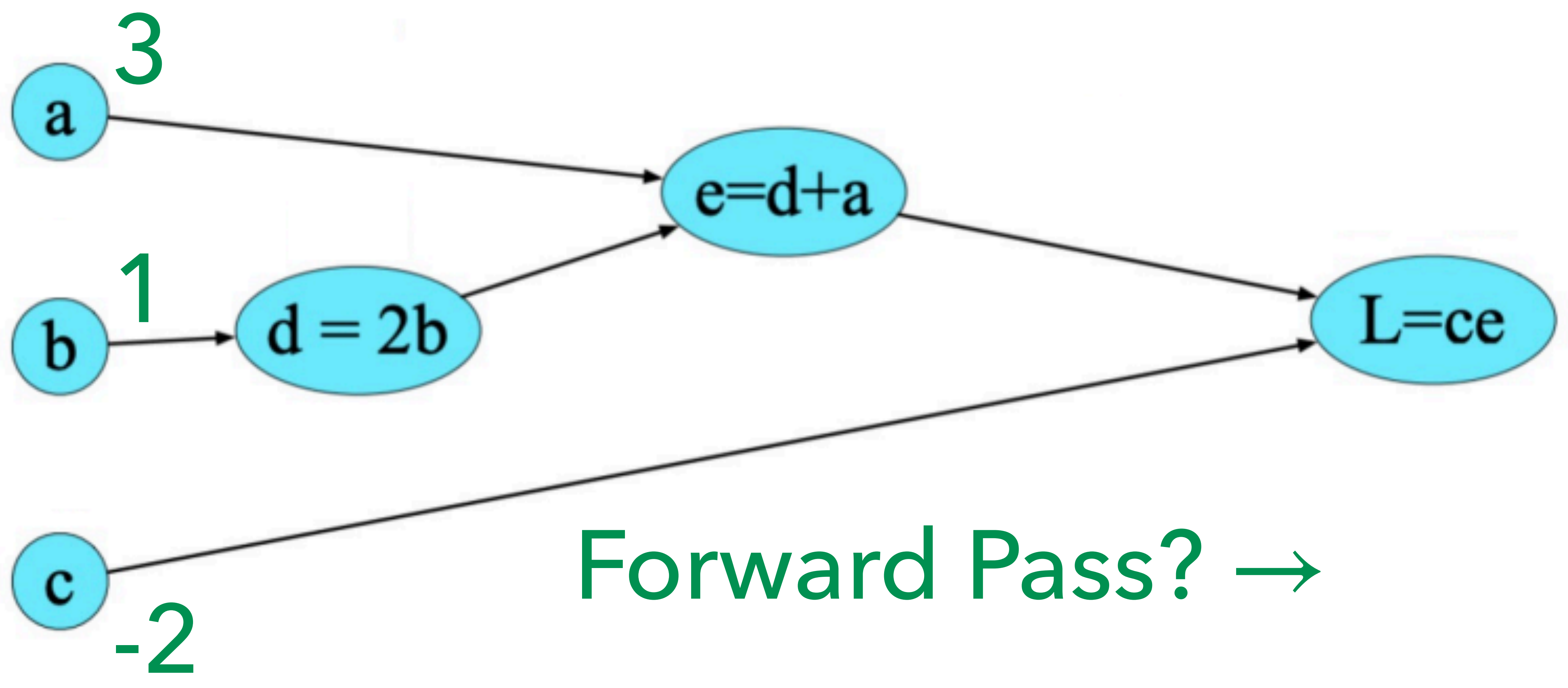


Computational Graphs

Computational Graph Construction

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$



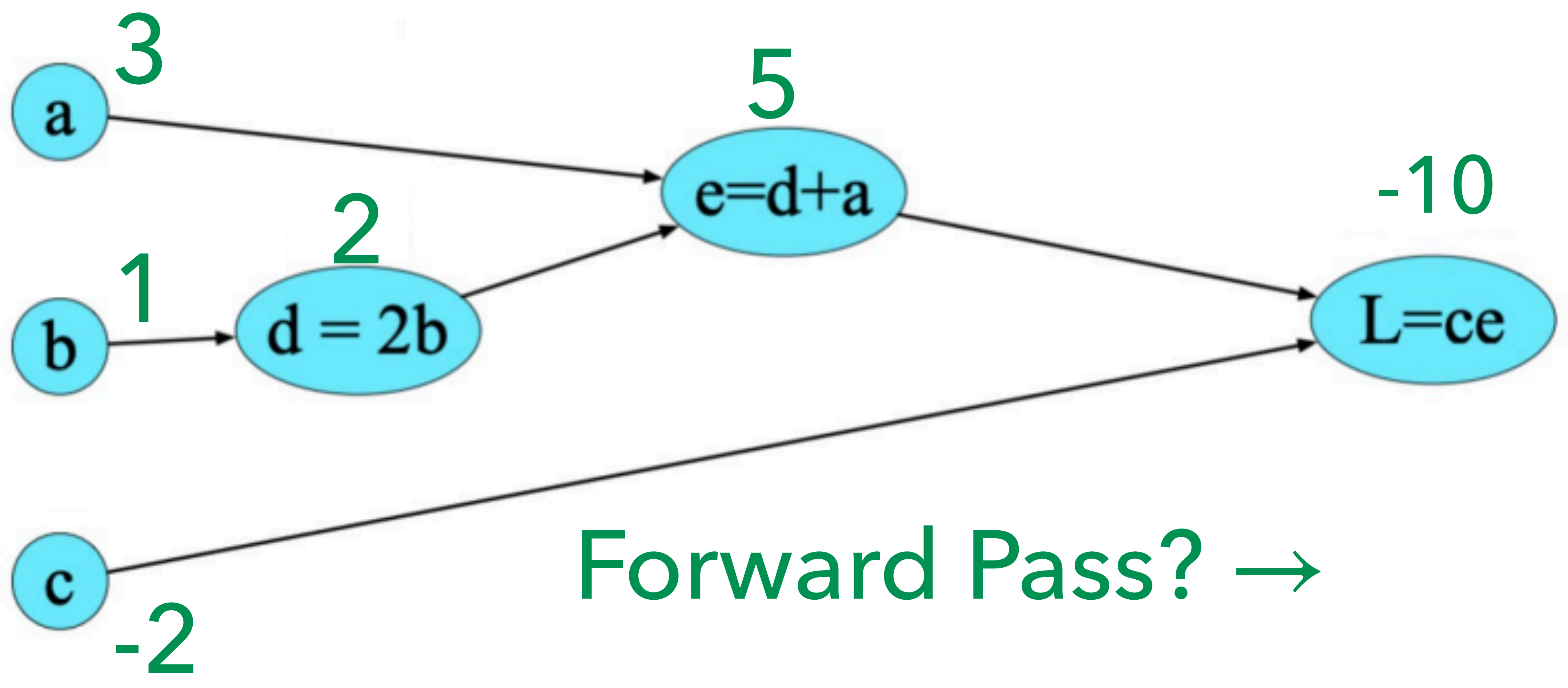
Forward Pass? →

Computational Graphs

Computational Graph Construction

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$



Forward Pass? →

How do we compute derivative of L wrt a, b, c?

Main Idea: use chain rule. If $f(x) = u(v(x))$, then $\frac{\partial f}{\partial x} = \frac{\partial u}{\partial v}\frac{\partial v}{\partial x}$

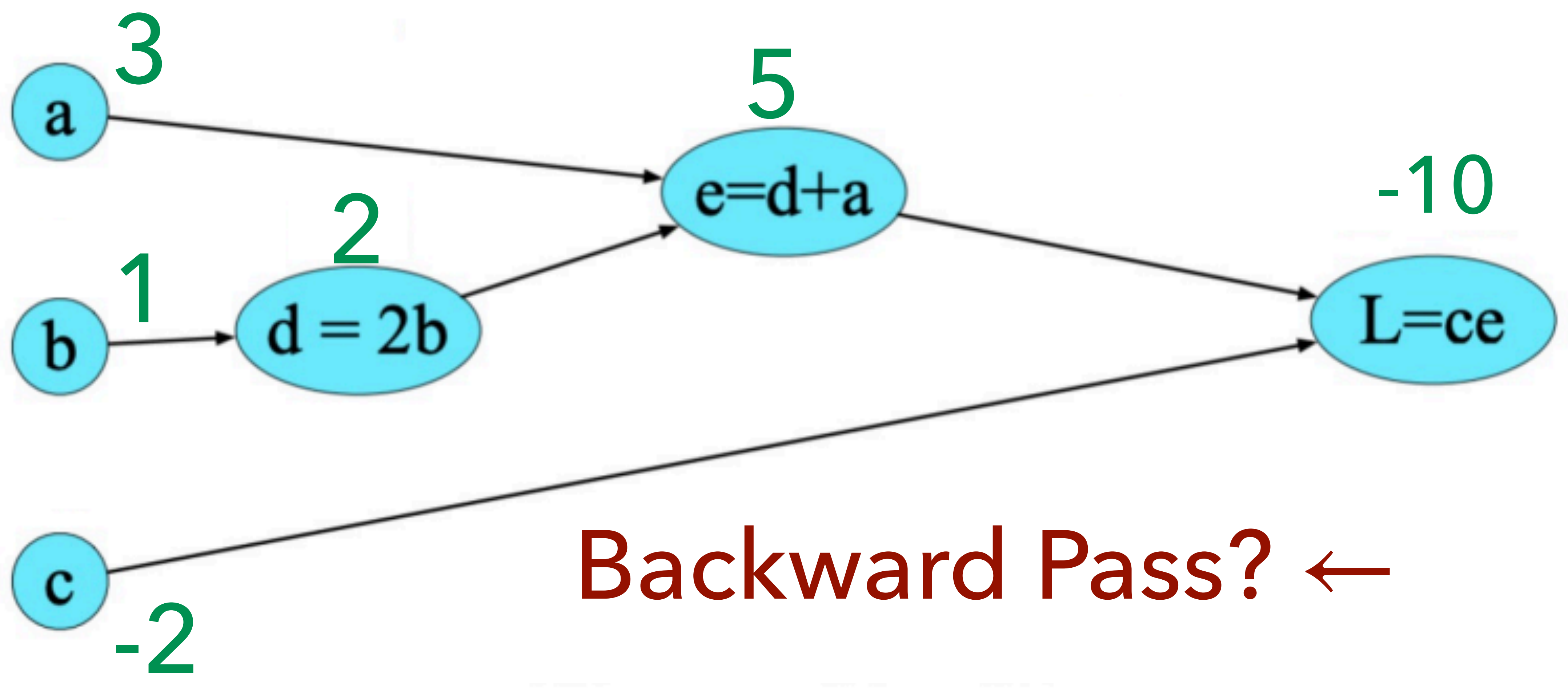
$$\boxed{\frac{\partial L}{\partial c} = e} \quad \boxed{\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}} \quad \boxed{\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}}$$

Computational Graphs

Computational Graph Construction

$$L(a,b,c) = c(a+2b)$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$



Backward Pass? ←

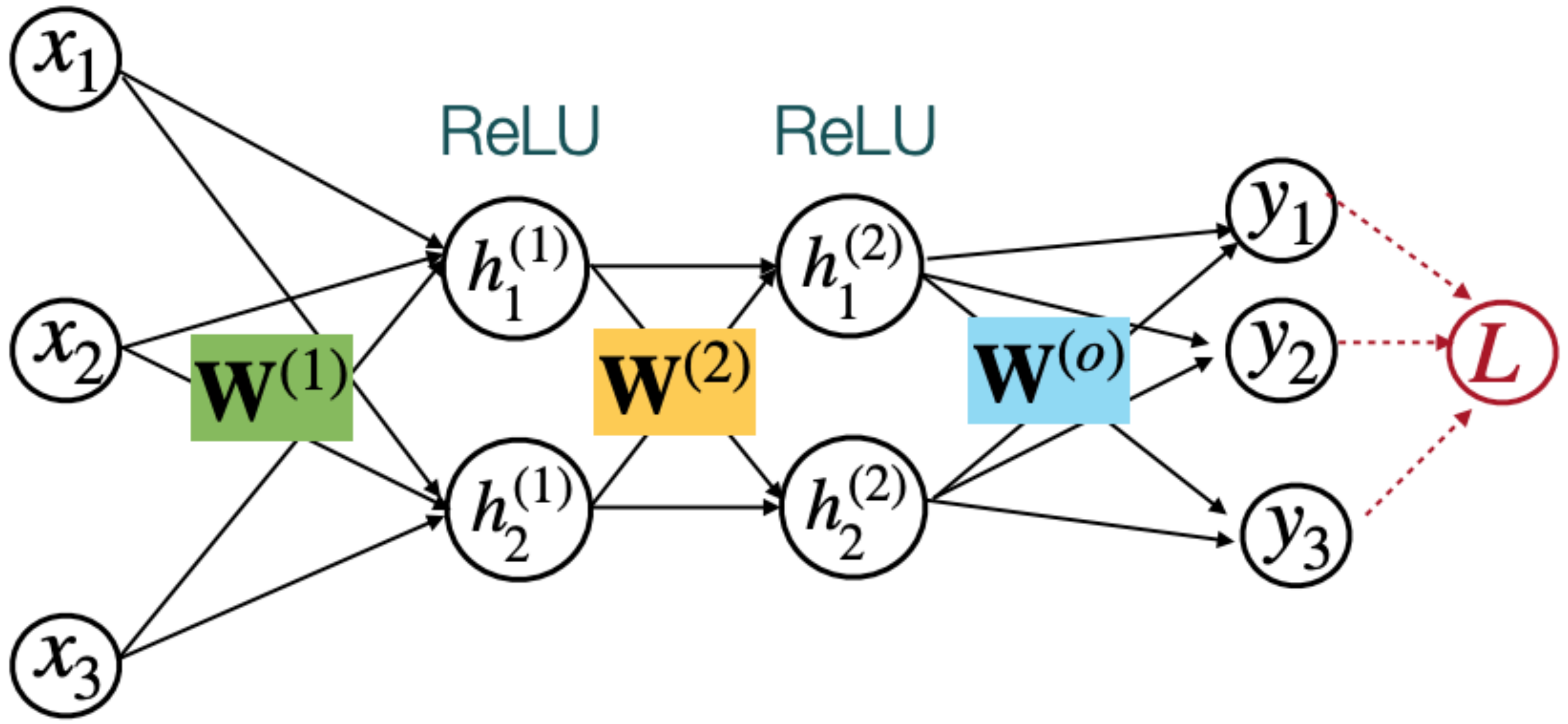
$$\boxed{\frac{\partial L}{\partial c} = e}$$

$$\boxed{\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}}$$

$$\boxed{\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}}$$

1 layer network for binary classification ~ LR

Backpropagation for a 3 layer NN

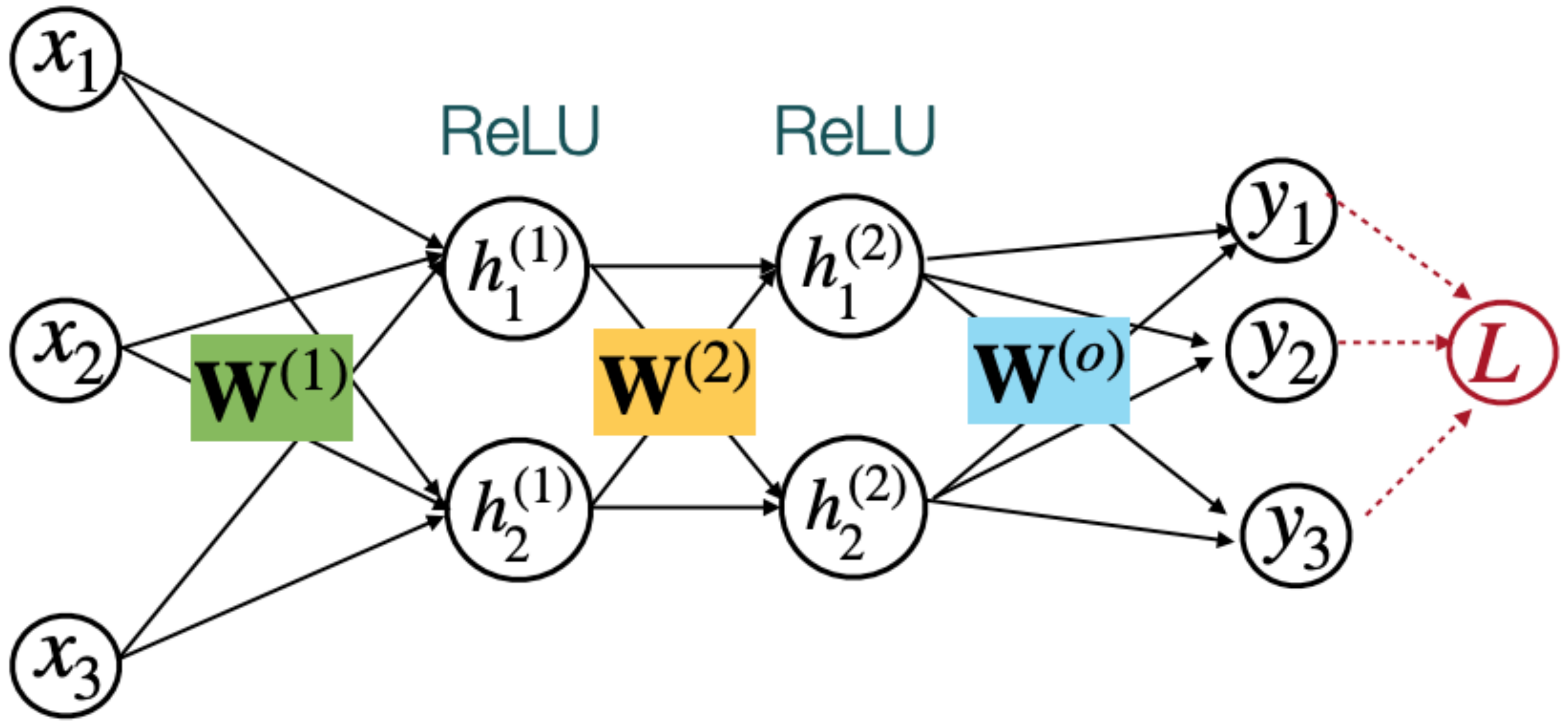


Slide Credit: Alane Suhr

Backpropagation for a 3 layer NN

Given: x_1, x_2, x_3 and the class label y

(a single training example)



Goal:

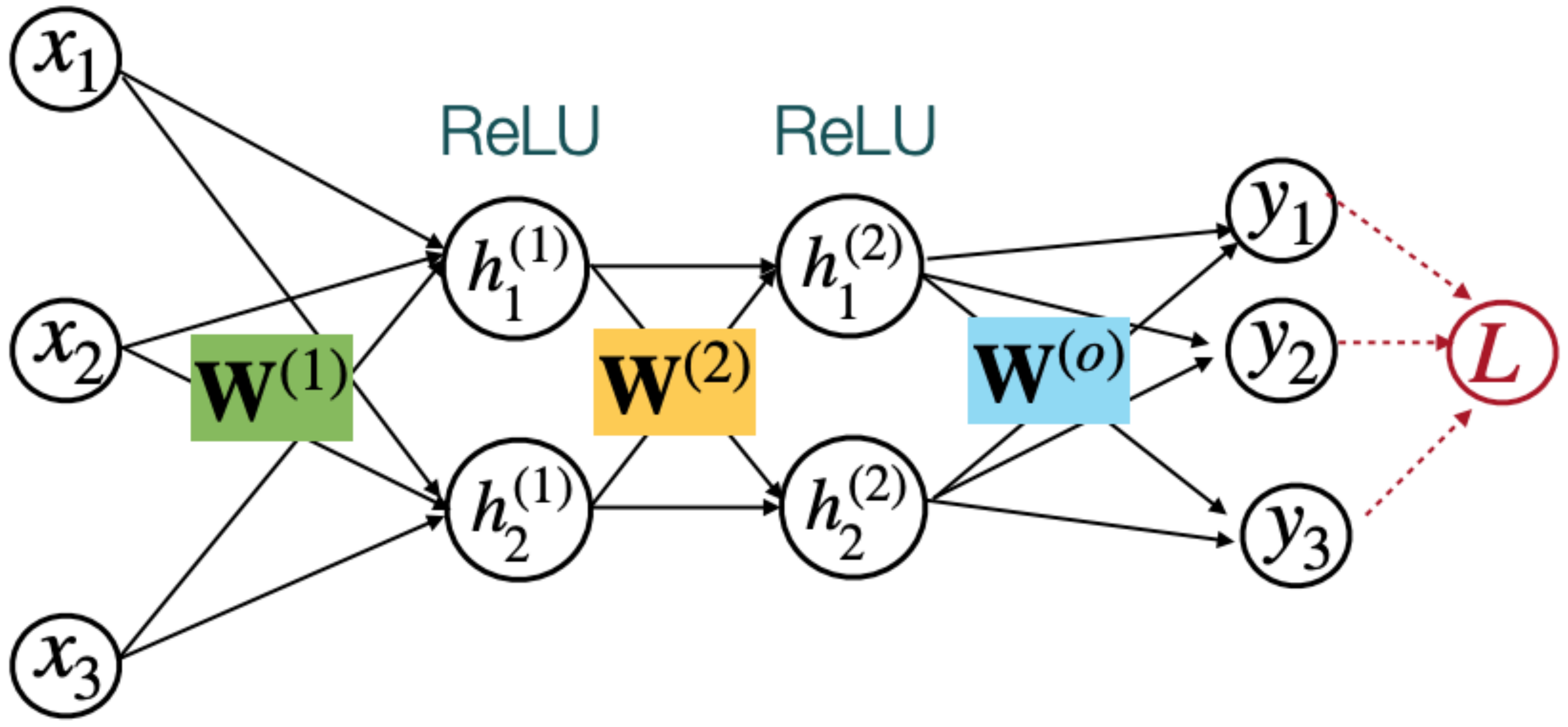
$$\frac{\partial L}{\partial W^{(1)}}, \frac{\partial L}{\partial W^{(2)}}, \frac{\partial L}{\partial W^{(o)}}$$

Slide Credit: Alane Suhr

Backpropagation for a 3 layer NN

Given: x_1, x_2, x_3 and the class label y (a single training example)

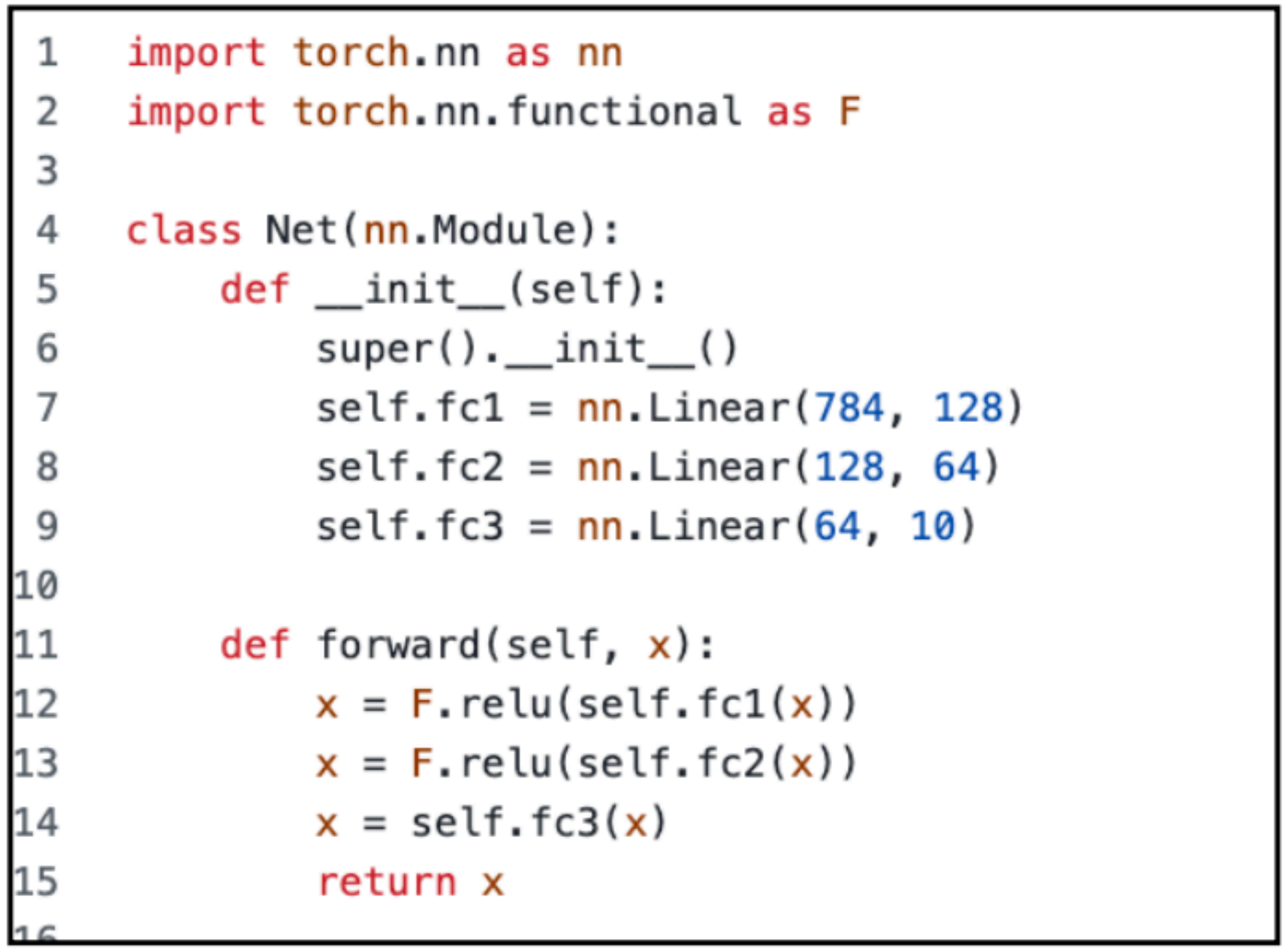
Goal: $\frac{\partial L}{\partial W^{(1)}}, \frac{\partial L}{\partial W^{(2)}}, \frac{\partial L}{\partial W^{(o)}}$



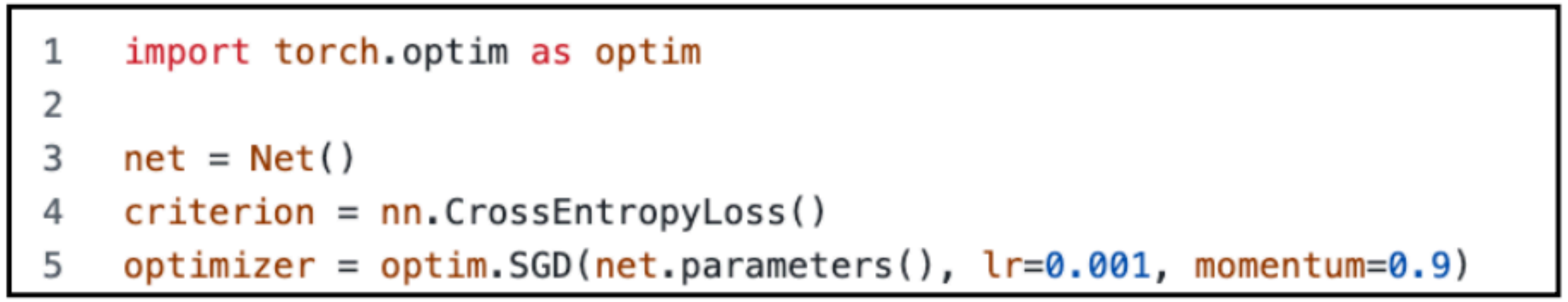
Slide Credit: Alane Suhr

Backpropagation for a 3 layer NN

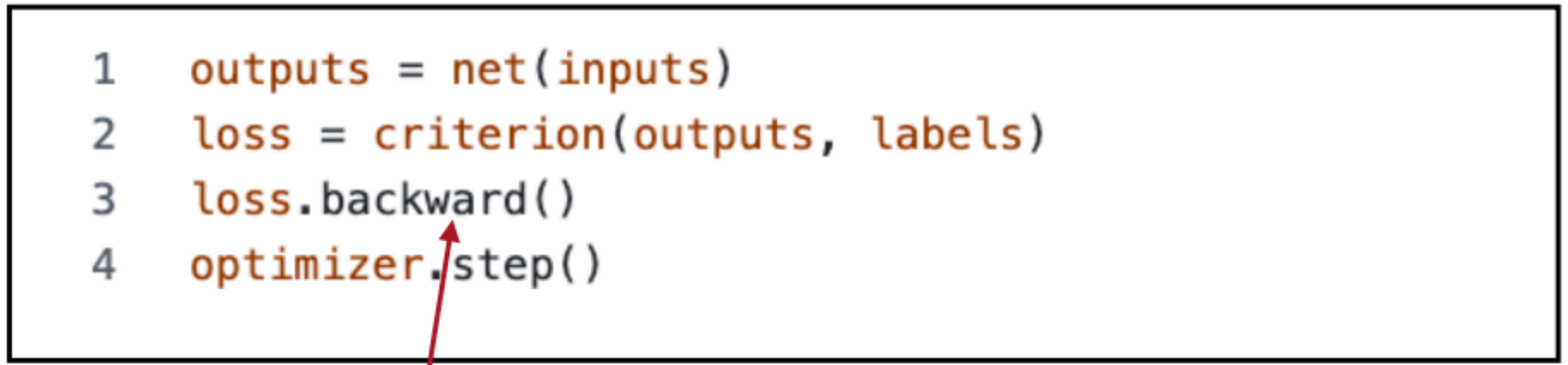
Backpropagation in PyTorch



```python
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```



```python
import torch.optim as optim

net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```



```python
outputs = net(inputs)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
```

PyTorch did back-propagation for you in this one line of code!

Slide Acknowledgements

- Earlier versions of this course offerings including materials from Claire Cardie, Marten van Schijndel, Lillian Lee
- CS 288 course by Alane Suhr (UC Berkeley).