Amirkabir University of Technology
(Tehran Polytechnic)

Natural Language Processing

Lecture 6: Neural LM

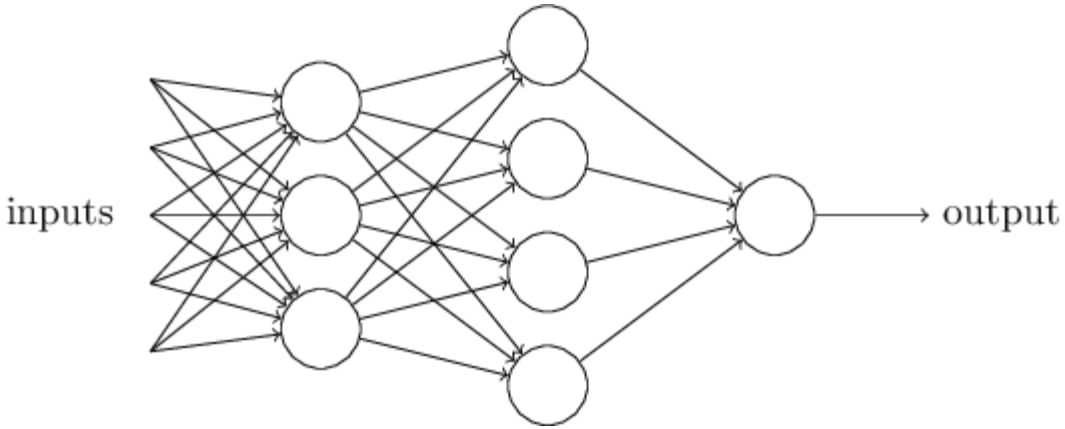
Amirkabir University of Technology
Dr Momtazi

Outline

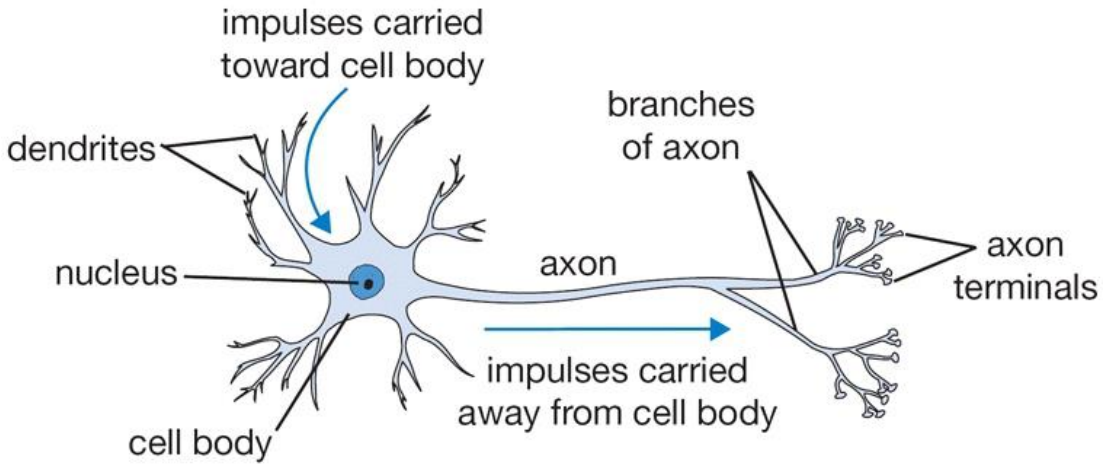
- **Architecture of Neural Networks**
- Units in Neural Networks
- Feedforward Networks
- Neural Language Model

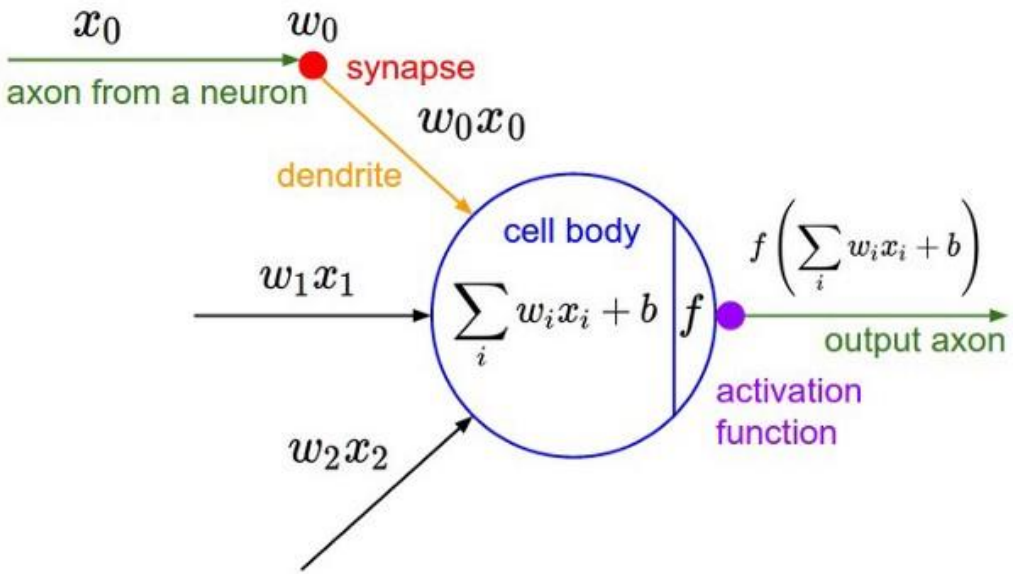
Neural Networks

- Neuron: the basic unit of computation in a neural network (also called a node or unit)
- Architecture
  - Input Nodes (input layer)
  - Hidden nodes (hidden layer)
  - Output Nodes (output layer)
  - Connections and weights
  - Activation function
  - Learning rule



Neural Networks





Neural Networks

- Input Nodes (input layer):
  - No computation is done here within this layer
  - They just pass the information to the next layer.
  - A block of nodes is also called layer.

Neural Networks

- Hidden nodes (hidden layer):
  - Hidden layers is where intermediate processing or computation is done.
  - They perform computations and then transfer the weights (signals or information) from the input layer to the following layer.
  - Next layer is another hidden layer or to the output layer.

Neural Networks

- Output Nodes (output layer):
  - Here we finally use an activation function that maps to the desired output format.

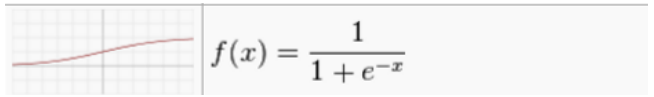
Neural Networks

- Connections and weights:
  - The network consists of connections, each connection transferring the output of a neuron to the input of another neuron.
  - Each connection is assigned a weight assigned on the basis of its relative importance to other inputs.

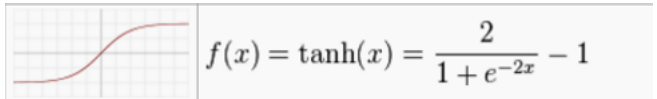
Neural Networks

- Activation function:
  - The activation function of a node defines the output of that node given an input or set of inputs (also called the transfer function).
  - It introduces non-linearity into the output of a neuron.
  - Taking a single number and performing a certain fixed mathematical operation on it.

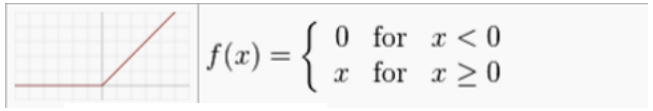
  - Types:
    - Sigmoid
    - Tanh
    - ReLU
    - Softmax



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



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



$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

Neural Networks

- Learning rule:
  - The learning rule is a rule or an algorithm which modifies the parameters of the neural network, in order to produce a favored output given an input to the network.
  - The learning process typically amounts to modifying the weights and thresholds.

Outline

- Architecture of Neural Networks
- **Units in Neural Networks**
- Feedforward Networks
- Neural Language Model

Neural Network Unit

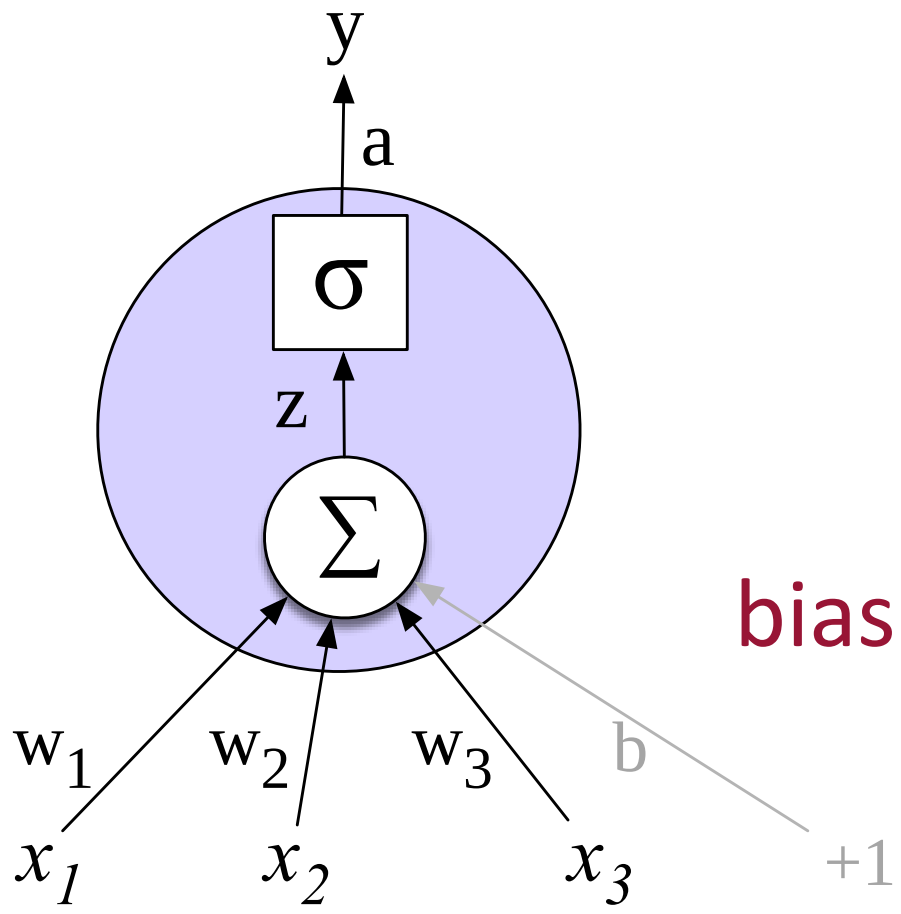
Output value

Non-linear transform

Weighted sum

Weights

Input layer



Neural unit

- Take weighted sum of inputs, plus a bias

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

$$Z = w \cdot x + b$$

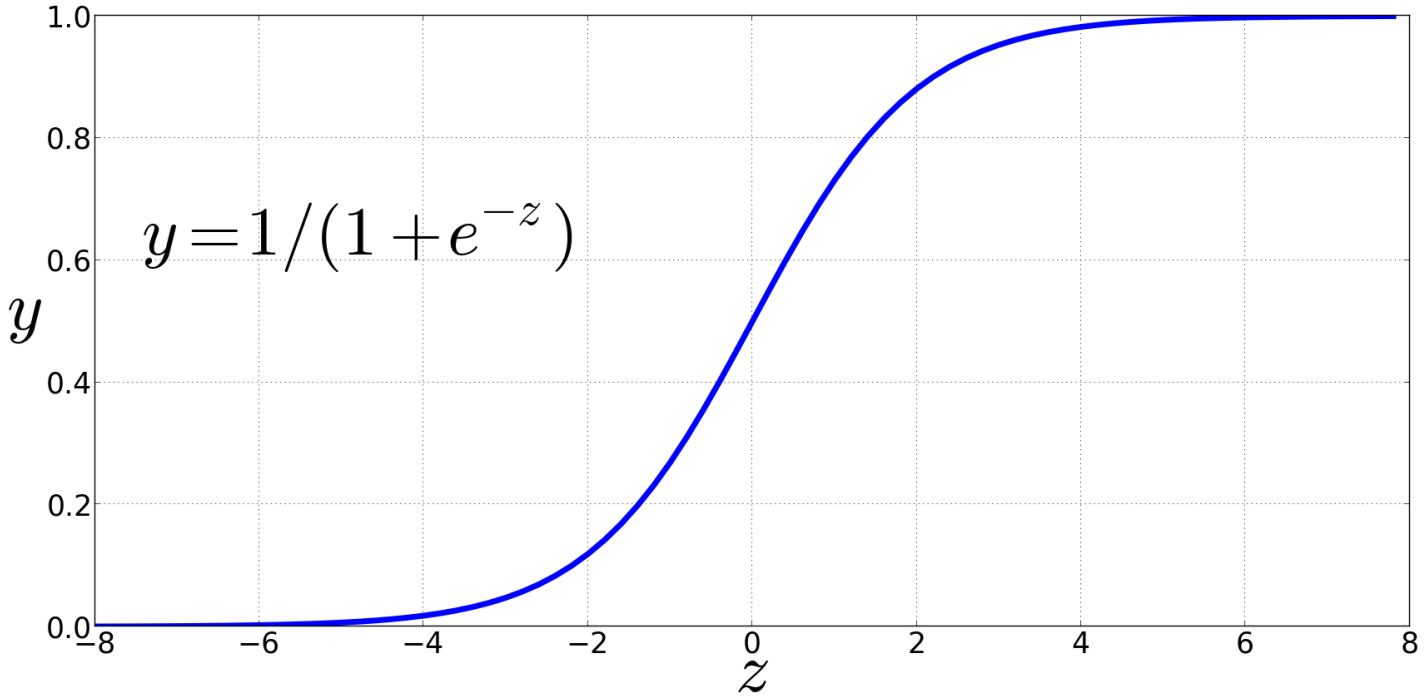
- Instead of just using z, we'll apply a nonlinear activation function f:

$$y = a = f(z)$$

Non-Linear Activation Functions

Sigmoid

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



Final function the unit is computing

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + exp(-(w \cdot x + b))}$$

Neural Network Unit

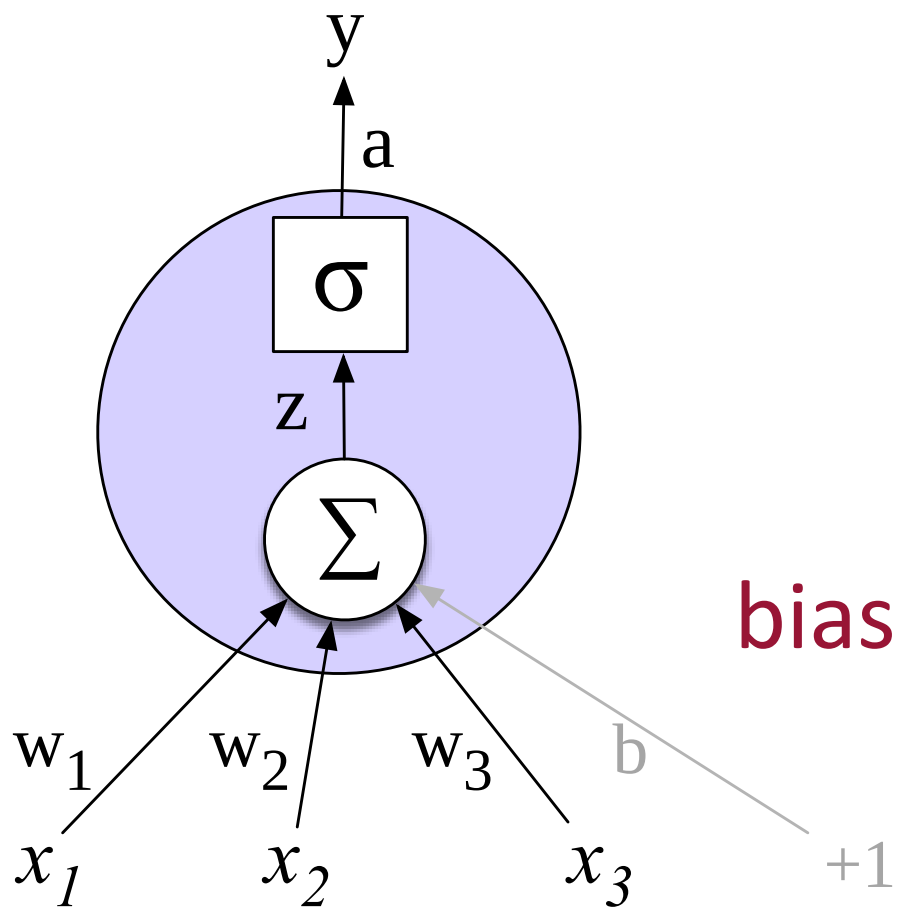
Output value

Non-linear transform

Weighted sum

Weights

Input layer



An example

- *Suppose a unit has:*

  w = [0.2,0.3,0.9]

  b = 0.5

- What happens with input x:

  x = [0.5,0.6,0.1]

$$y = \sigma(w \cdot x + b) =$$

An example

- *Suppose a unit has:*

  $w = [0.2, 0.3, 0.9]$

  $b = 0.5$

- What happens with input x:

  $x = [0.5, 0.6, 0.1]$

$$y = \sigma(w \cdot x + b) = \frac{1}{1+e^{-(w \cdot x+b)}}$$

An example

- *Suppose a unit has:*

  $w = [0.2, 0.3, 0.9]$

  $b = 0.5$

- What happens with input x:

  $x = [0.5, 0.6, 0.1]$

$$y = \sigma(w \cdot x + b) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

$$\frac{1}{1+e^{-(.5*.2+.6*.3+.1*.9+.5)}} =$$

An example

- *Suppose a unit has:*

  $w = [0.2, 0.3, 0.9]$

  $b = 0.5$

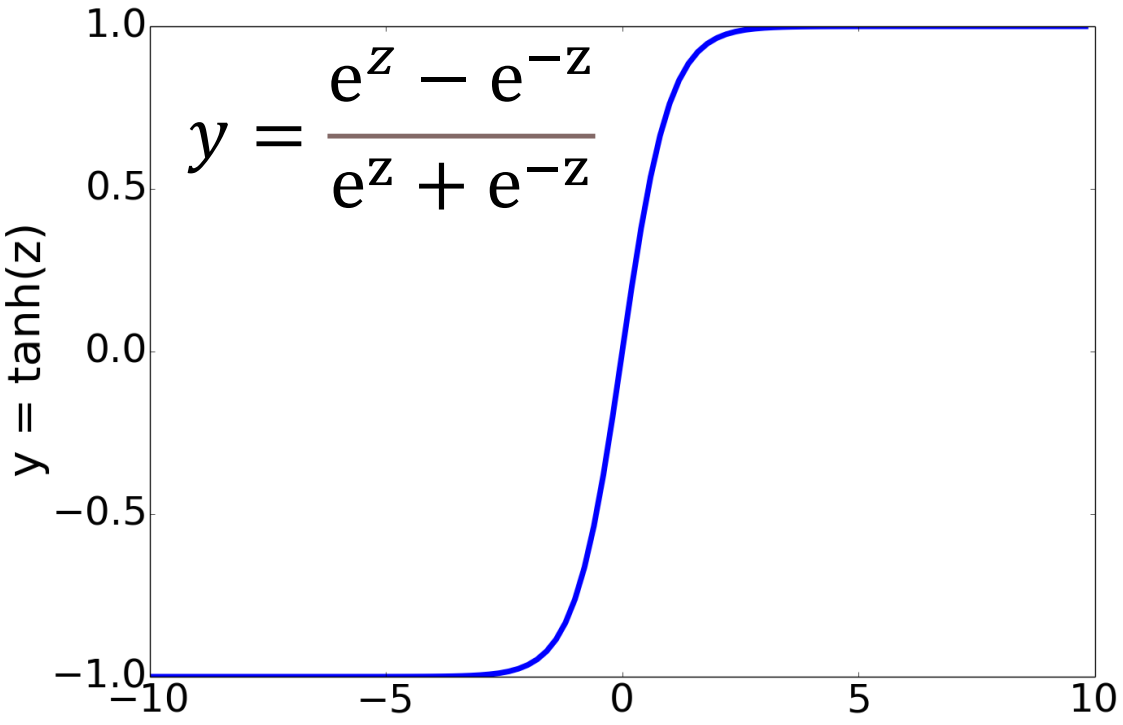
- What happens with input x:

  $x = [0.5, 0.6, 0.1]$

$$y = \sigma(w \cdot x + b) = \frac{1}{1+e^{-(w \cdot x + b)}}$$

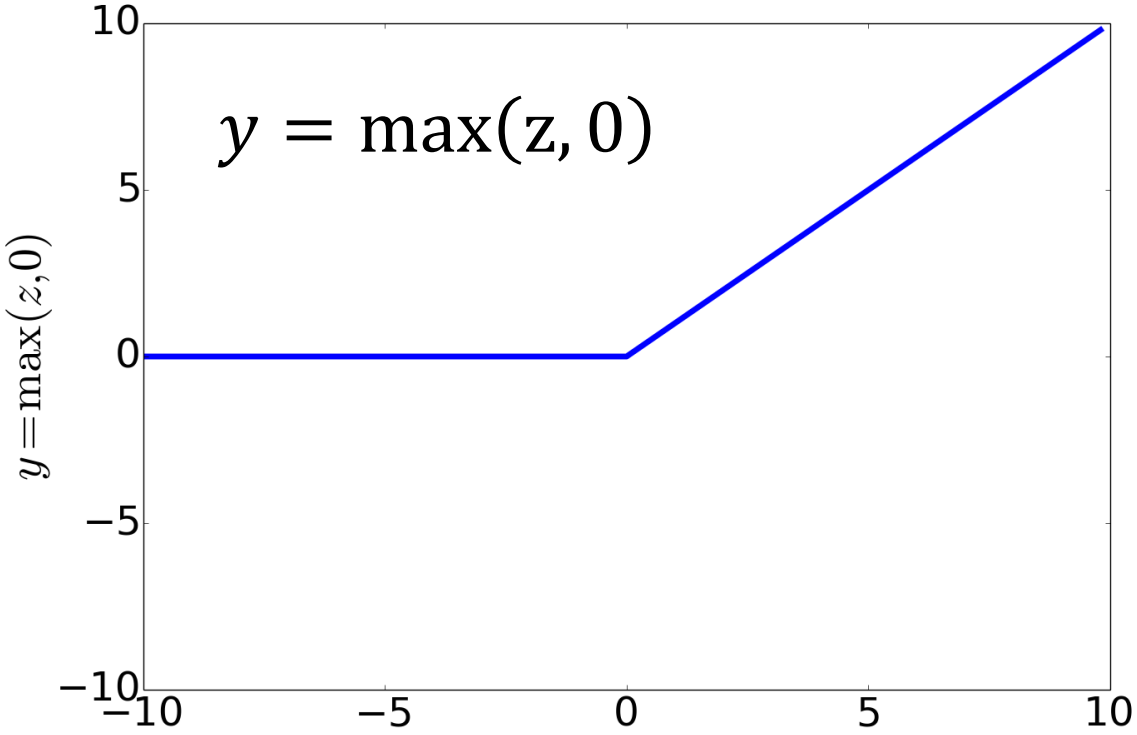
$$\frac{1}{1+e^{-(.5*.2+.6*.3+.1*.9+.5)}} = \frac{1}{1+e^{-0.87}} = .70$$

Non-Linear Activation Functions besides sigmoid



tanh

Most Common:



ReLU (Rectified Linear Unit)

Outline

- Architecture of Neural Networks
- Units in Neural Networks
- **Feedforward Networks**
- Neural Language Model

Feedforward Neural Network

- The first type of artificial neural network
- Connections between units do not form a cycle.
  - Information only travels forward in the network (no loops)
  - First through the input nodes, then through the hidden nodes (if present), and finally through the output nodes.
- Are primarily used for supervised learning in cases where the data to be learned is neither sequential nor time-dependent.

Feedforward Neural Network

- Single-Layer Perceptron
- Multi-Layer Perceptron

Single-Layer Perceptron

- The simplest type of feedforward neural network
- It has no hidden units (only an input layer and an output layer)
- The output units are computed directly from the sum of the product of their weights with the corresponding input units, plus some bias.
- Historically, the perceptron's output has been binary
  - It outputs a value of 0 or 1
  - This is achieved by passing the aforementioned product sum into the step function, which is defined as:

$$H(x) = \begin{cases} 1 \; if \; x \geq 0 \\ 0 \; if \; x < 0 \end{cases}$$

Single-Layer Perceptron

- Error Function
  - The learning process requires the definition of an error function that quantifies the difference between the computed output of the perceptron and the true value for an input over a set of multiple input-output pairs .
  - Historically, this error function is the mean squared error (MSE), defined for a set of input-output pairs
  - Therefore, a natural objective is to attempt to change weights and bias such that error is as close to zero as possible.
  - Thus, minimizing error with respect to weights and bias should yield a good classification boundary.

Single-Layer Perceptron

- Updating weights and bias
  - Error function is typically minimized using gradient descent
  - The perceptron adjusts weights and bias in the direction of the negative gradient of the error function
  - Gradient descent works for any error function, not just the mean squared error.
  - This iterative process reduces the value of the error function until it converges on a value (usually a local minimum)
  - The values for initializing weights and bias are set randomly and then updated using gradient descent

$$\overrightarrow{w_{i+1}} = \overrightarrow{w_i} - \alpha \frac{\partial E(X)}{\partial \overrightarrow{w_i}}$$

$$b_{i+1} = b_i - \alpha \frac{\partial E(X)}{\partial b_i}$$

Single-Layer Perceptron

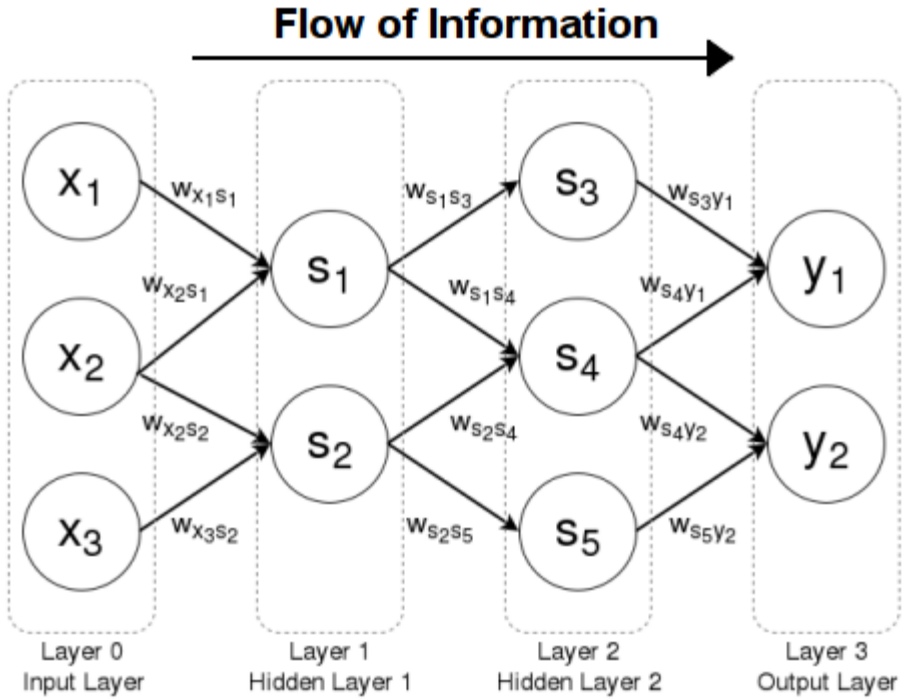
- Limitation
  - Single-layer perceptrons are linear classifiers
  - They can only learn linearly separable patterns
  - It is impossible for a perceptron to learn even simple non-linearly separable functions

Multi-Layer Perceptron

- The mulit-layer perceptron (MLP) is an artificial neural network composed of many perceptrons.
- Unlike single-layer perceptrons, MLPs are capable of learning to compute non-linearly separable functions.
- They are one of the primary machine learning techniques for both regression and classification in supervised learning.

Multi-Layer Perceptron

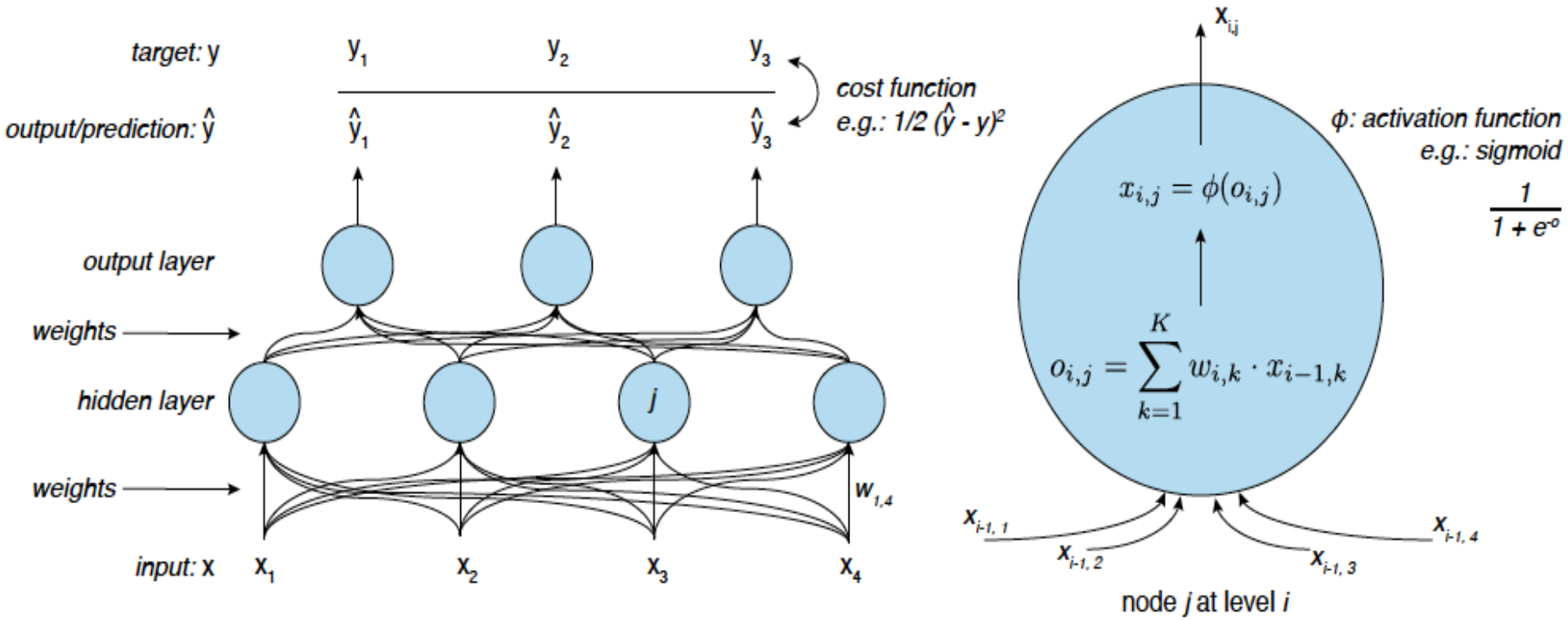
- MLPs are organized into a set of units, called layers.
- In the case of a single-layer perceptron, there are no hidden layers, so the total number of layers is two.
- MLPs have at least one hidden layer, each composed of multiple perceptrons.



Multi-Layer Perceptron

- Feedforward neural networks have structure similar to a directed acyclic graph (DAG)
  - No layer's output depends on itself
  - Each layer (other than the input layer) is only connected to the layer directly to its left.
- Feedforward neural networks, by having DAGs for their computation graphs, have a greatly simplified learning algorithm compared to recurrent neural networks, which have cycles in their dependency graphs.

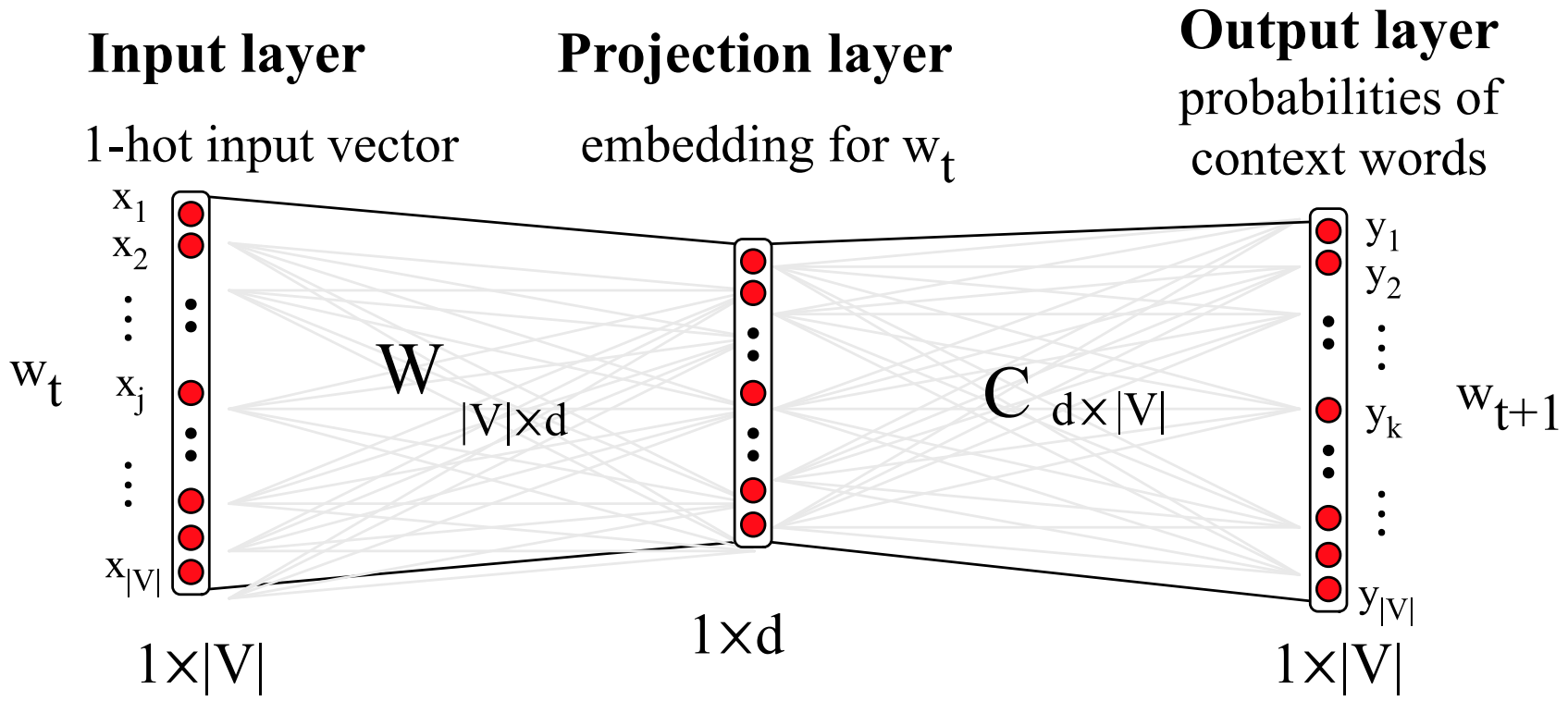
Multi-layer Perceptron (Feedforward Neural Network)



Outline

- Architecture of Neural Networks
- Units in Neural Networks
- Feedforward Networks
- **Neural Language Model**

Visualizing W and C as a network for doing error back propagation



Neural Language Models (LMs)

- **Language Modeling**: Calculating the probability of the next word in a sequence given some history.
  - We've seen N-gram based LMs
  - But neural network LMs far outperform n-gram language models

- State-of-the-art neural LMs are based on more powerful neural network technology like Transformers
- But **simple feedforward LMs** can do almost as well!

Simple feedforward Neural Language Models

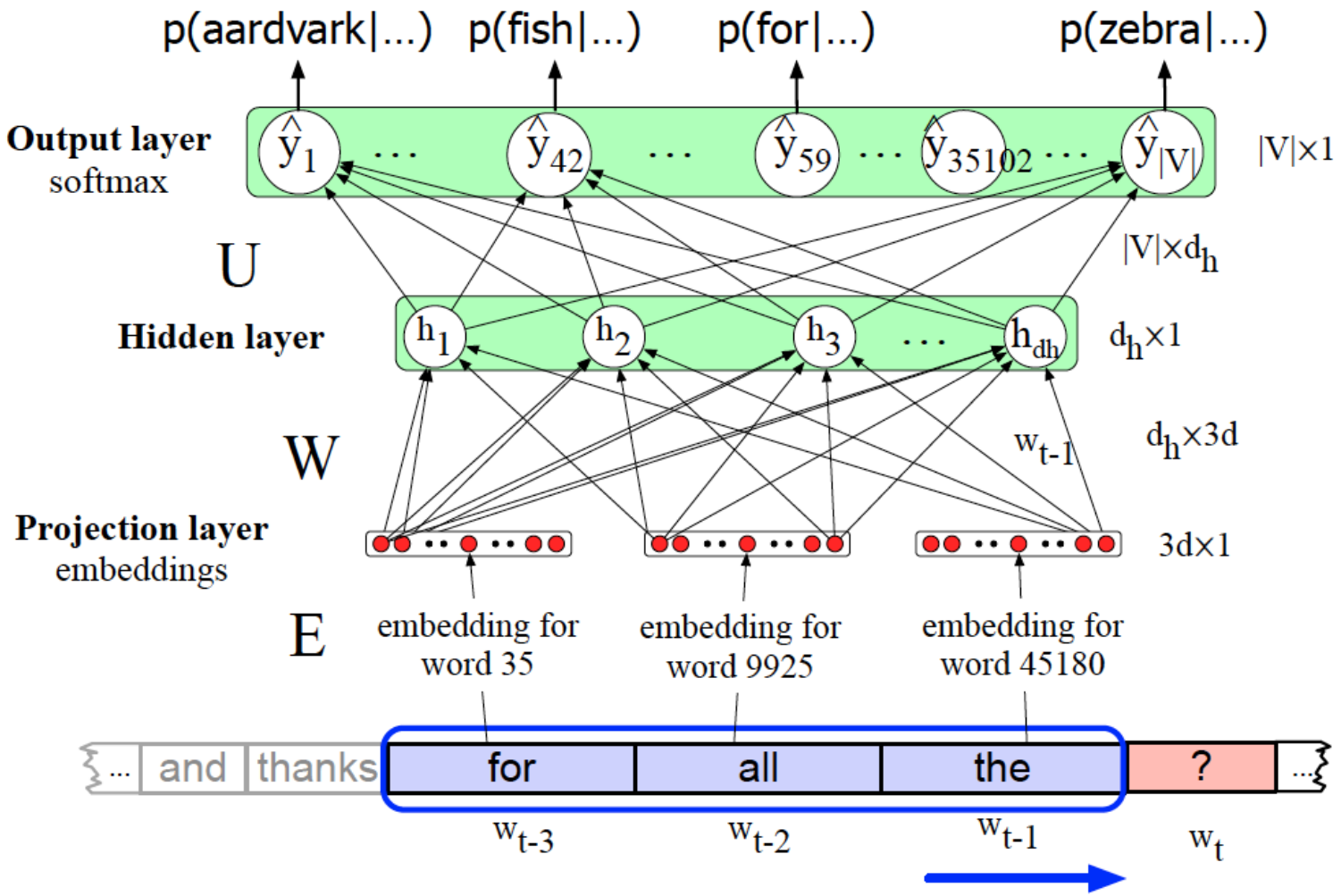
- **Task**: predict next word w_t

  given prior words w_{t-1}, w_{t-2}, w_{t-3}, …

- **Problem**: Now we're dealing with sequences of arbitrary length.
- **Solution**: Sliding windows (of fixed length)

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

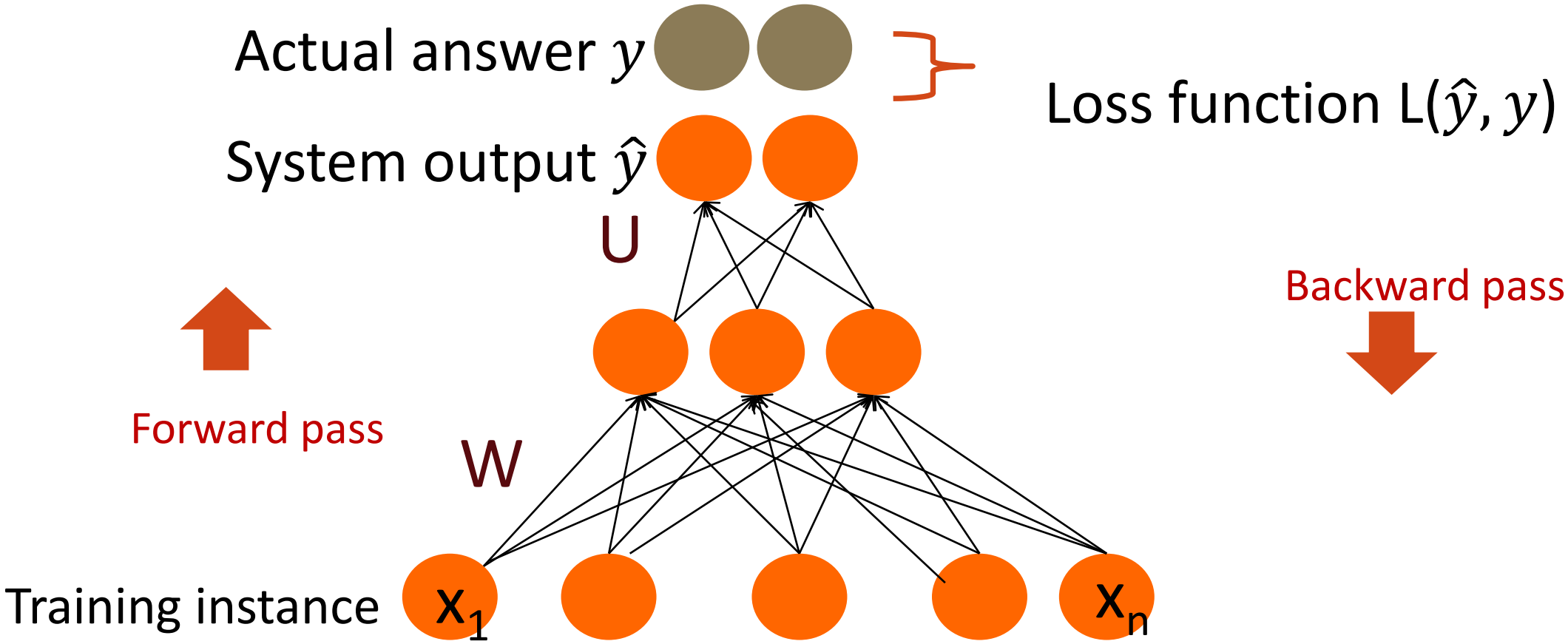
Neural Language Model



Why Neural LMs work better than N-gram LMs

- **Training data:**
  - We've seen: I have to make sure that the cat gets fed.
  - Never seen: dog gets fed
- **Test data:**
  - I forgot to make sure that the dog gets ___
  - N-gram LM can't predict "fed"!
  - Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict “fed” after dog

Intuition: training a 2-layer Network



Intuition: Training a 2-layer network

- For every training tuple (x, y)
  - Run forward computation to find our estimate $\hat{y}$
  - Run backward computation to update weights:
    - For every output node
      - Compute loss L between true y and the estimated $\hat{y}$
      - For every weight w from hidden layer to the output layer
        - Update the weight
    - For every hidden node
      - Assess how much blame it deserves for the current answer
      - For every weight w from input layer to the hidden layer
        - Update the weight

Reminder: Loss Function for binary logistic regression

- A measure for how far off the current answer is to the right answer
- Cross entropy loss for logistic regression:

$$L_{\text{CE}}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

$$= -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$

Reminder: gradient descent for weight updates

- Use the derivative of the loss function with respect to weights $\frac{d}{dw} L(f(x;w),y)$
- To tell us how to adjust weights for each training item
  - Move them in the opposite direction of the gradient
  - For logistic regression

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w),y)$$

$$\frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

Where did that derivative come from?

- Using the chain rule! $f(x) = u(v(x))$
- Intuition (see the text for details)

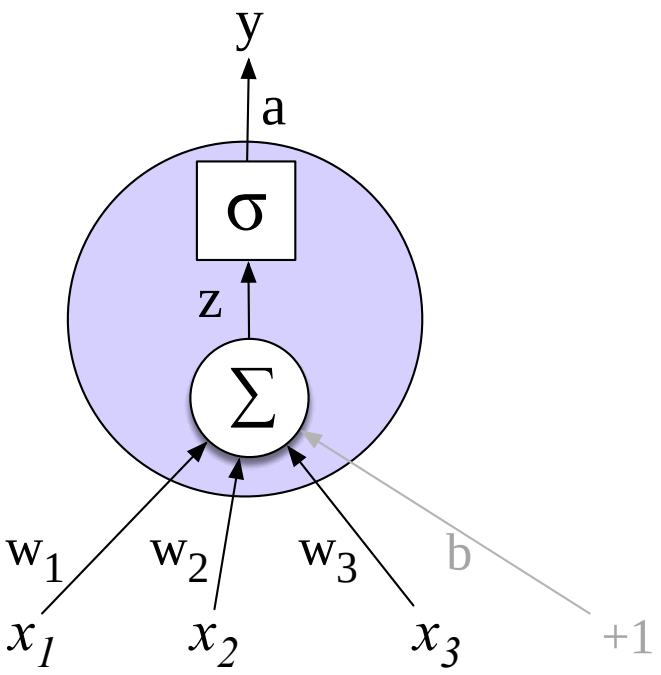
$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

Derivative of the weighted sum

Derivative of the Activation

Derivative of the Loss

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i}$$



Further Reading

- Speech and Language Processing (3rd ed. draft)
  - Chapter 7