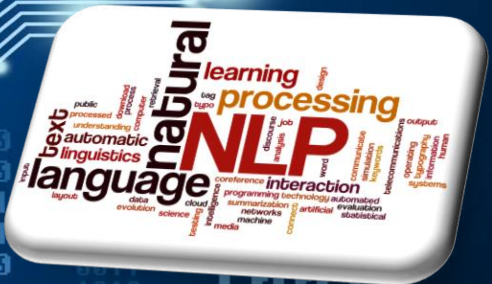
Cairo University
Faculty of Engineering
Computer Engineering Department

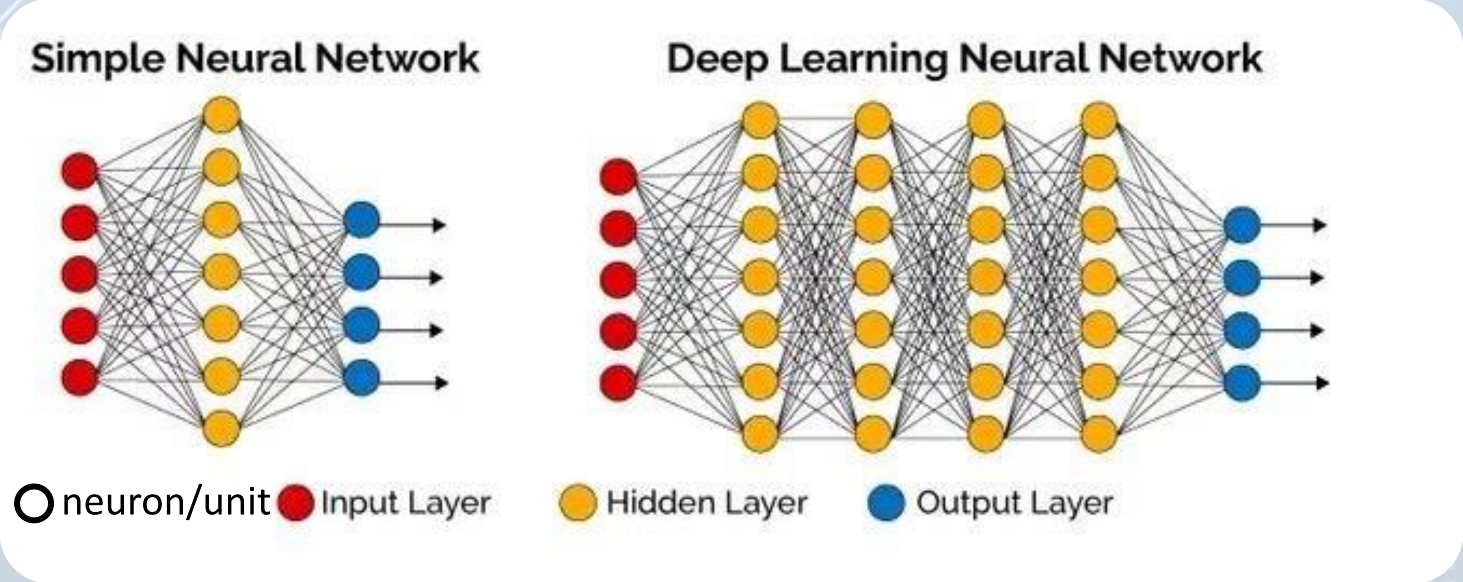
Natural Language Processing

Dr. Sandra Wahid



Overview

- Neural networks are a fundamental computational tool for language processing.
- Their name and structure are inspired by the **human brain**, mimicking the way that biological neurons signal to one another.
- **Feedforward Network:** the computation proceeds iteratively from one layer of *units* to the next.
- **Deep Learning:** involves the use of modern neural nets that are often *deep* → have many layers.



Units/Neurons

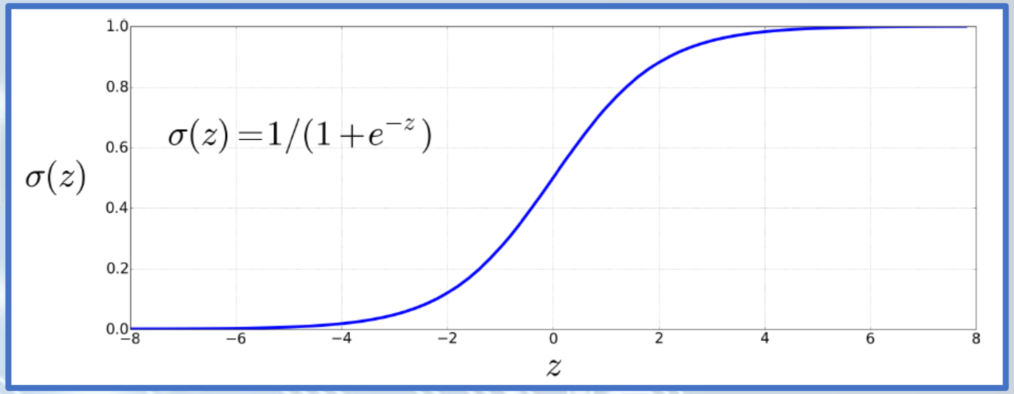
- The building block of a neural network is a single computational unit.
  - A unit takes a set of real valued numbers as **input**, performs some computation on them, and produces an **output**.
- A neural unit takes a **weighted sum of its inputs**, with one additional term in the sum called a **bias term**.
  - Given a set of inputs $x_1, x_2, \ldots x_n$, a unit has a set of corresponding **weights** $w_1, w_2, \ldots w_n$ and a bias b, so the weighted sum z can be represented as: $\boxed{z = b + \sum_i w_i x_i}$
  - Using vectors: $\boxed{z = \mathbf{w} \cdot \mathbf{x} + b}$ where **.** is the dot product
- Finally, instead of using z (a linear function of x) as the output, neural units apply a **non-linear function** f to z.
  - The output of this function is known as the **activation** value for the unit, a.

$$\boxed{a = f(z)}$$

- The final output of the network is referred to as y.

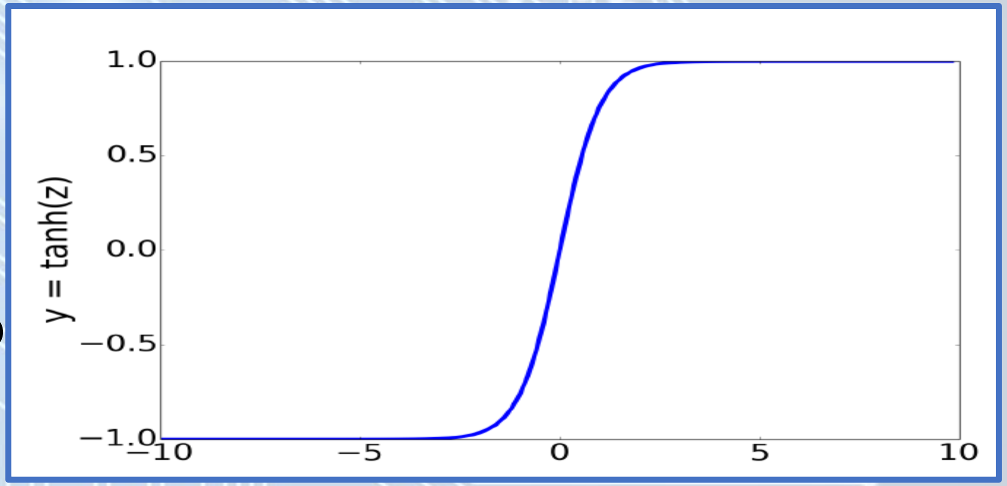
Example Activation Functions

- **Sigmoid function:** $$\sigma(z) = \frac{1}{1+e^{-z}}$$



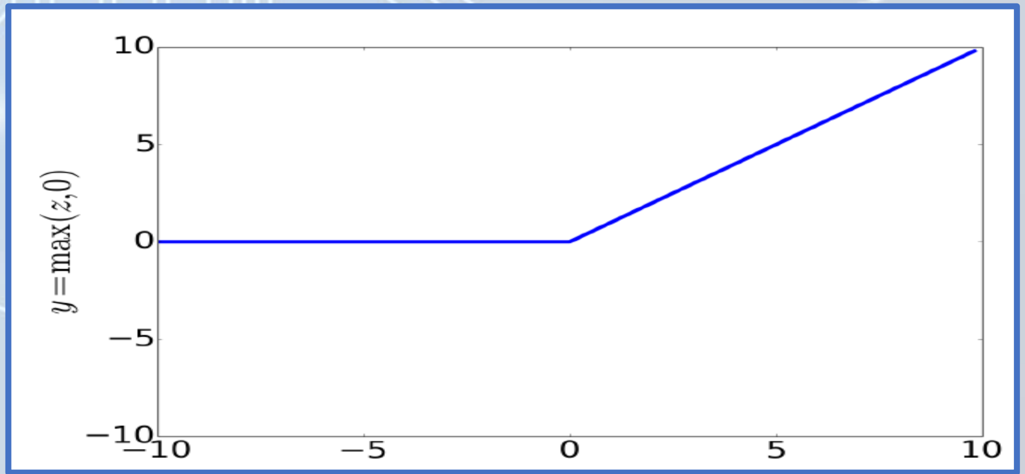
  - It maps the output into the range [0,1].
  - It is useful in squashing outliers toward 0 or 1.
  - It is differentiable.

- **tanh function:** $$\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$



  - It is a variant of the sigmoid that maps the output into the range [-1,1].
  - It is smoothly differentiable.

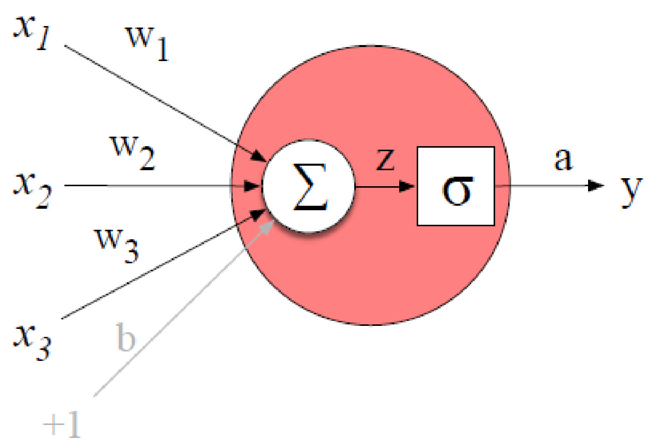
- **ReLU function:** $$\mathrm{ReLU(z)} = max(z,0)$$



  - Rectified Linear Unit.
  - It is just the same as *z* when *z* is positive, and 0 otherwise.
  - It gives a result that is very close to linear.

Basic Neural Unit Example

- The unit takes 3 input values *x1*, *x2*, and *x3*, and computes a weighted sum, multiplying each value by a weight (*w1*, *w2*, and *w3*, respectively), adds them to a bias term *b*, and then passes the resulting sum through a sigmoid function to result in a number between 0 and 1.
  - In this case the output of the unit *y* is the same as *a*, but in deeper networks we'll reserve *y* to mean the final output of the entire network, leaving *a* as the activation of an individual node.



- Using the following weight and bias values, what would this unit do with the following input vector *x*?

$$
\begin{aligned}
\mathbf{w} &= [0.2, 0.3, 0.9] \\
b &= 0.5 \\
\mathbf{x} &= [0.5, 0.6, 0.1]
\end{aligned}
$$

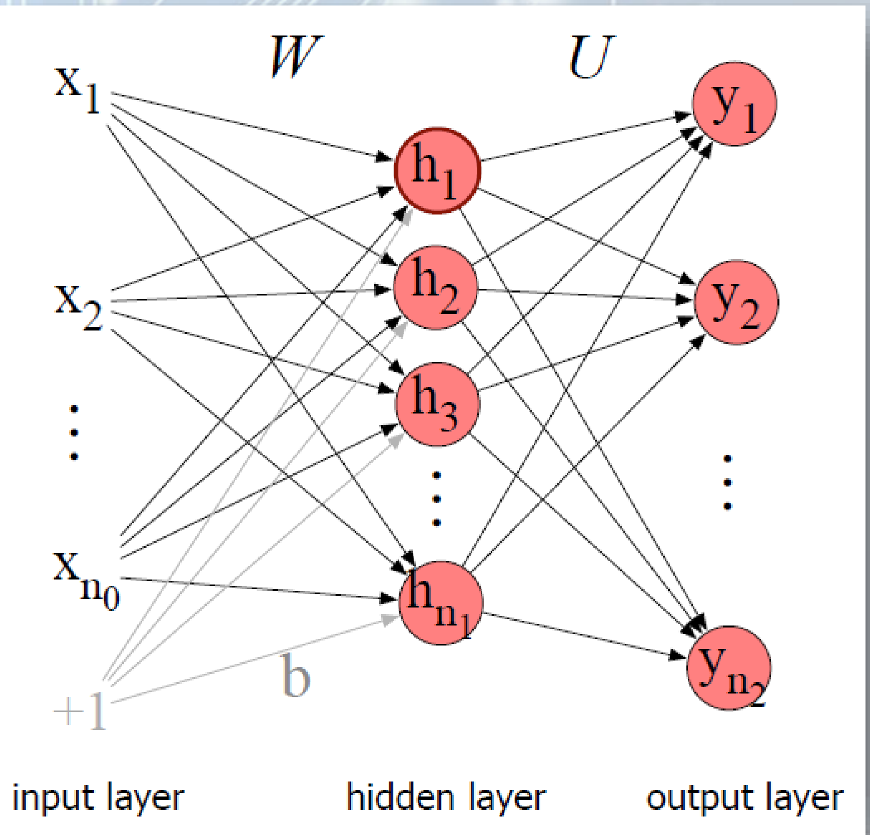
$$y = \sigma(\mathbf{w}\cdot\mathbf{x}+b) = \frac{1}{1+e^{-(\mathbf{w}\cdot\mathbf{x}+b)}} = \frac{1}{1+e^{-(.5*.2+.6*.3+.1*.9+.5)}} = \frac{1}{1+e^{-0.87}} = .70$$

Feedforward Neural Networks

- A feedforward network:
  - is a multilayer network
  - the units are connected with no cycles
    - the outputs from units in each layer are passed to units in the next higher layer
    - no outputs are passed back to lower layers
  - has three kinds of nodes:
    - input units, hidden units, and output units.

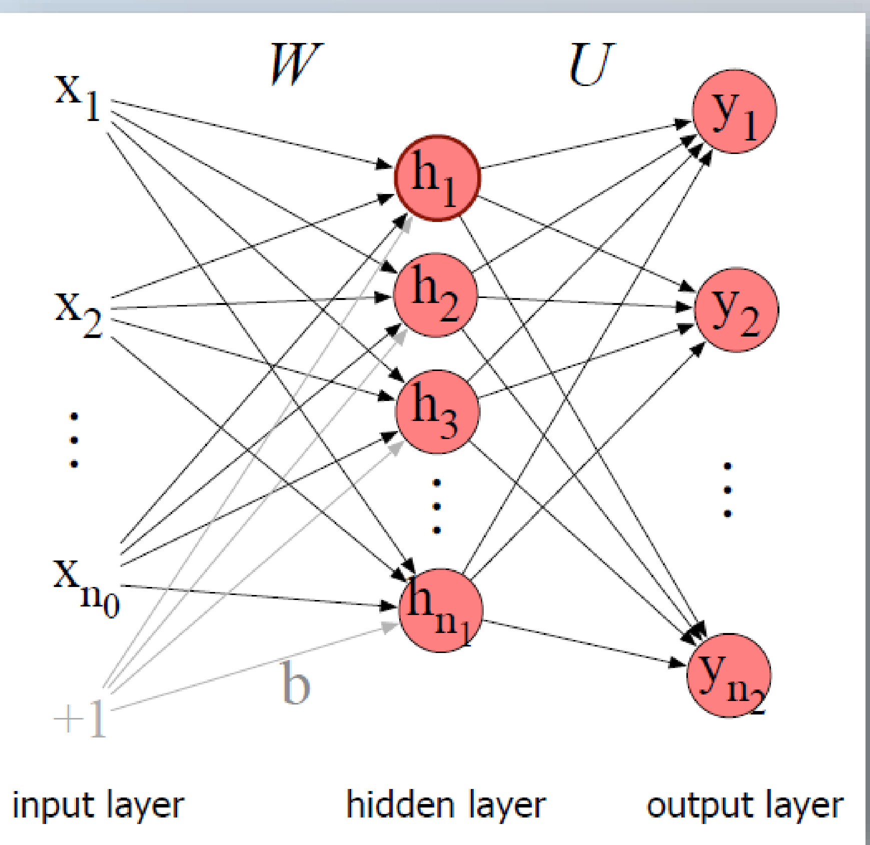
- A simple *2-layer* feedforward network:
  - one hidden layer
  - one output layer
  - one input layer

the input layer is usually not counted when enumerating layers



Feedforward Neural Networks

- In the standard architecture, each layer is **fully-connected:**
  - each unit in each layer takes as input the outputs from all the units in the previous layer.
  - there is a link between every pair of units from two adjacent layers.
- A single hidden unit has as parameters a *weight vector* and a *bias*.
  - We represent the parameters for the entire hidden layer as a **single weight matrix *W*:** with dimensionality *[n1,n0]* and a **single bias vector *b*:** with dimensionality *[n1,1]*
  - Each element ***Wji*** of the weight matrix ***W*** represents the weight of the connection from the *ith* input unit *xi* to the *jth* hidden unit *hj*.

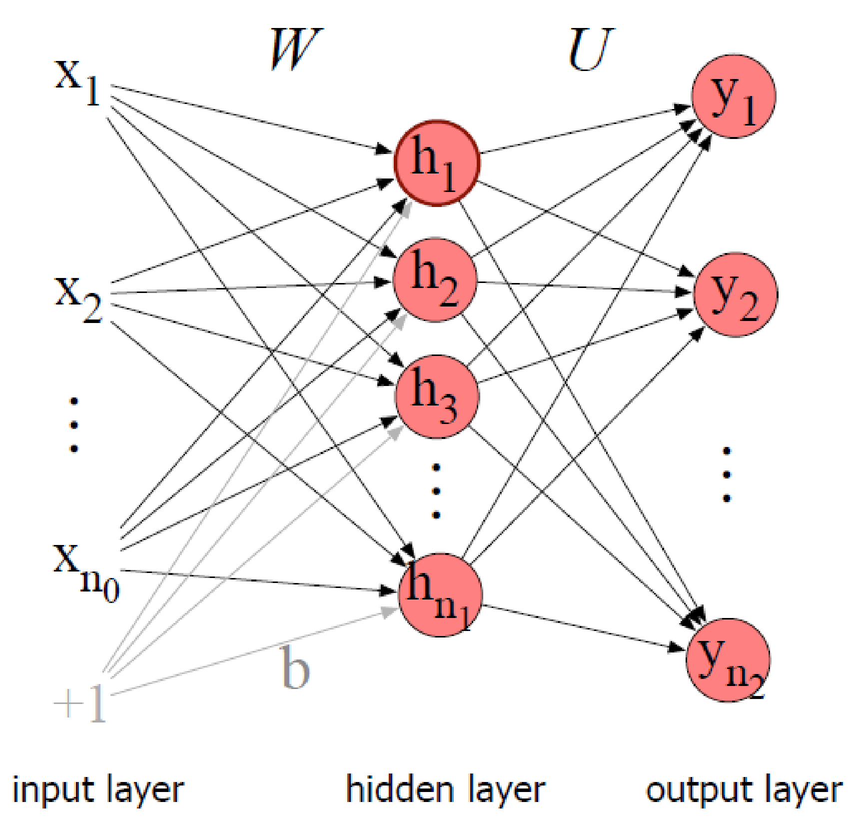


Why??

the hidden layer computation for a feedforward network can be done very efficiently with simple matrix operations.

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Feedforward Neural Networks



- Like the hidden layer, the output layer has ***a weight matrix U:*** with dimensionality *[n2,n1]* but some models don't include a bias vector ***b*** in the output.
- The intermediate output *z* is given by: $\boxed{\mathbf{z} = \mathbf{U}\mathbf{h}}$
- However, *z* can't be the output of a classifier, since it's a vector of real-valued numbers →while what we need for classification is **a vector of probabilities** →we need to perform ***normalization***
  - A popular function used is ***softmax***

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

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

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

- Summing Up:

$$\begin{aligned} \mathbf{h} &= \sigma(\mathbf{W}\mathbf{x}+\mathbf{b}) \\ \mathbf{z} &= \mathbf{U}\mathbf{h} \\ \mathbf{y} &= \text{softmax}(\mathbf{z}) \end{aligned}$$

General Terminology

- Superscripts in square brackets to mean layer numbers, starting at 0 for the input layer.
  - ***$W^{[1]}$:*** weight matrix for the first hidden layer.
  - ***$b^{[1]}$:*** bias vector for the first hidden layer.
- *n_j:* number of units at layer *j*.
- *g(.):* activation function.
  - (ReLU or tanh for intermediate layers and sigmoid or softmax for output layers).
- ***$a^{[i]}$:*** output from layer *i*.
  - ***$a^{[0]}$*** refers to the inputs ***x.***
- ***$z^{[i]}$:*** combination of weights and biases ***$W^{[i]}a^{[i-1]} + b^{[i]}$.***

- For a 2-layer net:

$$
\begin{aligned}
\mathbf{z}^{[1]} &= \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]} \\
\mathbf{a}^{[1]} &= g^{[1]}(\mathbf{z}^{[1]}) \\
\mathbf{z}^{[2]} &= \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]} \\
\mathbf{a}^{[2]} &= g^{[2]}(\mathbf{z}^{[2]}) \\
\hat{\mathbf{y}} &= \mathbf{a}^{[2]}
\end{aligned}
$$

- The algorithm for computing the forward step in an n-layer feedforward network:

for i **in** 1,...,n

$$
\begin{aligned}
\mathbf{z}^{[i]} &= \mathbf{W}^{[i]}\,\mathbf{a}^{[i-1]} + \mathbf{b}^{[i]} \\
\mathbf{a}^{[i]} &= g^{[i]}(\mathbf{z}^{[i]})
\end{aligned}
$$

$$\hat{\mathbf{y}} = \mathbf{a}^{[n]}$$

Why non-linear activation functions?

- If we did not use them, the resulting network is exactly equivalent to a single-layer network.
- How??
  - Consider the first two layers of such a network of purely linear layers:

$$
\begin{aligned}
\mathbf{z}^{[1]} &= \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]} \\
\mathbf{z}^{[2]} &= \mathbf{W}^{[2]}\mathbf{z}^{[1]} + \mathbf{b}^{[2]}
\end{aligned}
$$

  - Rewriting:

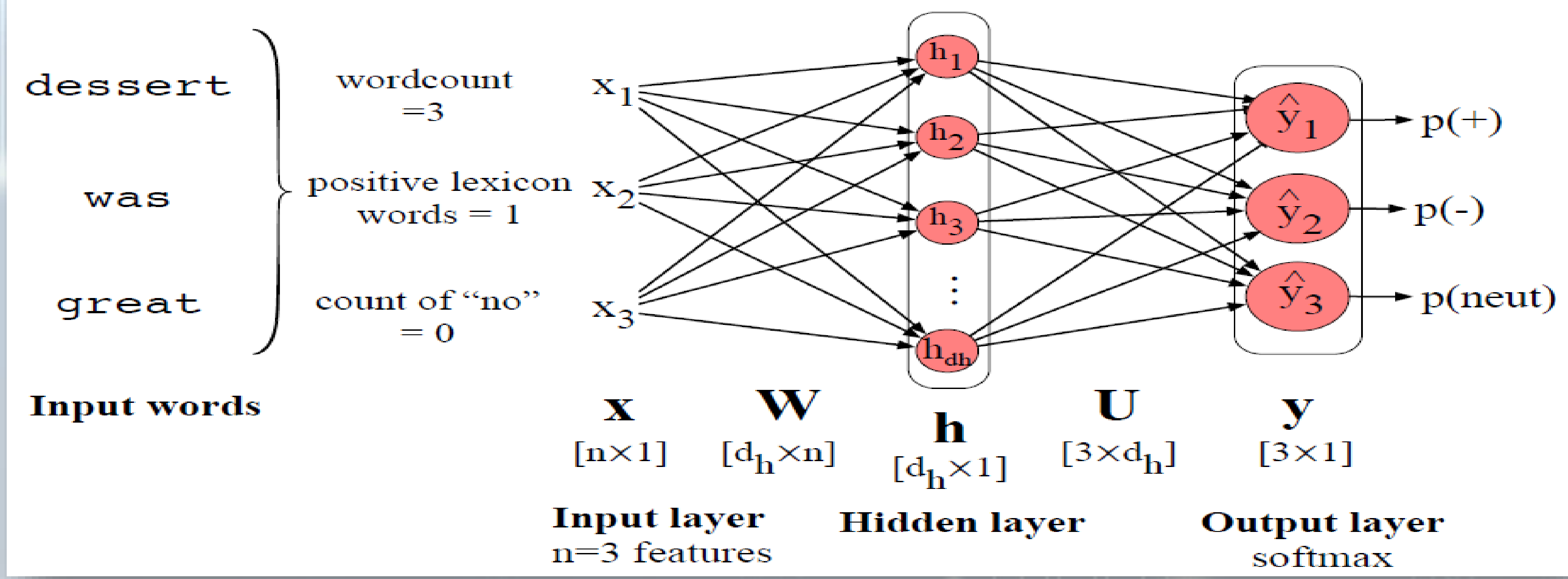
$$
\begin{aligned}
\mathbf{z}^{[2]} &= \mathbf{W}^{[2]}\mathbf{z}^{[1]} + \mathbf{b}^{[2]} \\
&= \mathbf{W}^{[2]}(\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]} \\
&= \mathbf{W}^{[2]}\mathbf{W}^{[1]}\mathbf{x} + \mathbf{W}^{[2]}\mathbf{b}^{[1]} + \mathbf{b}^{[2]} \\
&= \mathbf{W}'\mathbf{x} + \mathbf{b}'
\end{aligned}
$$

- This generalizes to any number of layers

without non-linear activation functions, a multilayer network is just a **single layer network with a different set of weights**, and we lose all the representational power of multilayer networks.

NLP Example

- A 2-layer sentiment classifier.
- The inputs are scalar features
  - e.g.: x_1=count(words in doc), x_2=count(positive lexicon words in doc), x_3 = 1 if “no” in doc, and so on.
- The output layer $\hat{\boldsymbol{y}}$ has 3 nodes (positive, negative, neutral):
  - $\widehat{\boldsymbol{y}_1}$ the estimated probability of positive sentiment
  - $\widehat{\boldsymbol{y}_2}$ the estimated probability of negative sentiment
  - $\widehat{\boldsymbol{y}_3}$ the estimated probability of neutral sentiment.



Training Neural Nets

- A feedforward neural net is an instance of **supervised machine learning** in which we know the correct output *y* for each observation *x*.
- The goal of the training procedure is to learn parameters $\boldsymbol{W}^{[i]}$ and $\boldsymbol{b}^{[i]}$ for each layer *i* that make $\hat{y}$ (network output) for each training observation as close as possible to the true *y*.
- Components:

1. **Loss function:** to model the distance between the network output and the gold output.
2. **Gradient descent optimization:** to find the parameters that minimize this loss function.

$$w_{ji(new)} = w_{ji(old)} - \eta \frac{\partial L}{\partial w_{ji}} \qquad \text{where } \eta \text{ is the learning rate}$$

3. **Error backpropagation:** gradient descent requires knowing the gradient of the loss function, *the vector that contains the partial derivative of the loss function with respect to each of the parameters*.
   - For neural networks, with millions of parameters in many layers, it's much harder to see how to compute the partial derivative of some weight in layer 1 when the loss is attached to some much later layer→ The answer is the algorithm called error backpropagation or backward differentiation.
   - The gradients are calculated starting from the final layer and then through use of the *chain rule*, the gradients can be passed backwards to calculate the gradients in the previous layers.
   - Chain rule: given y=g(u), and u=f(x)

$$\frac{dy}{dx} = \frac{dy}{du} \times \frac{du}{dx}$$


Thank You