CS 333: NLP

Fall 2025

Prof. Carolyn Anderson
Wellesley College

Reminders

- HW 6 will be released today:
  - Neural network classifier for song lyrics using contextualized word embeddings
- No class on Tuesday (Tanner)
- CS Colloquium on 10/31
- Quiz 7 next Friday (J&M Chapter 7)
- My next help hours: Monday 4-5:30

CS FALL COLLOQUIUM SERIES

DESIGNING POSITIVE FUTURES FOR AI IN SOCIALLY COMPLEX WORK WITH AND FOR WORKERS

ANNA KAWAKAMI '21
Carnegie Mellon University

Date: Oct 31st,
12:45 - 2:00pm
Location: Sci H-105

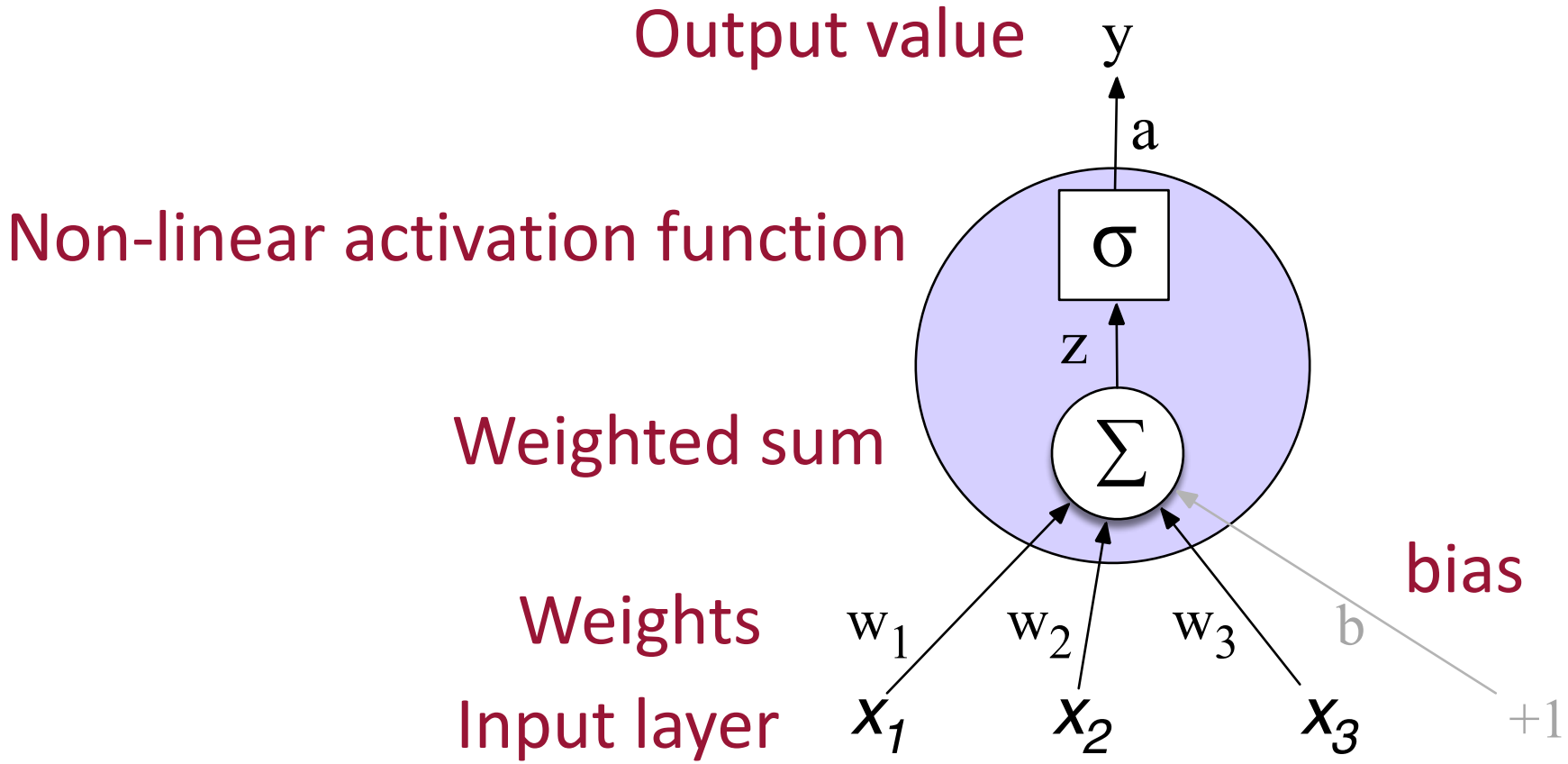
RSVP For Lunch

https://bit.ly/CSKawakami

?sb129@wellesley.edu

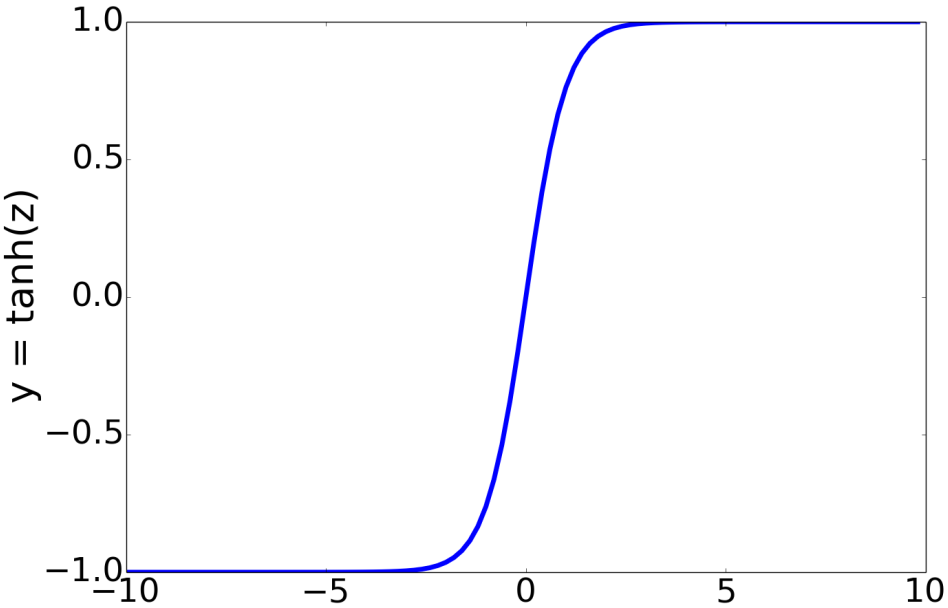
Accessibility and Disability:
accessibility@wellesley.edu

Neural Network Unit



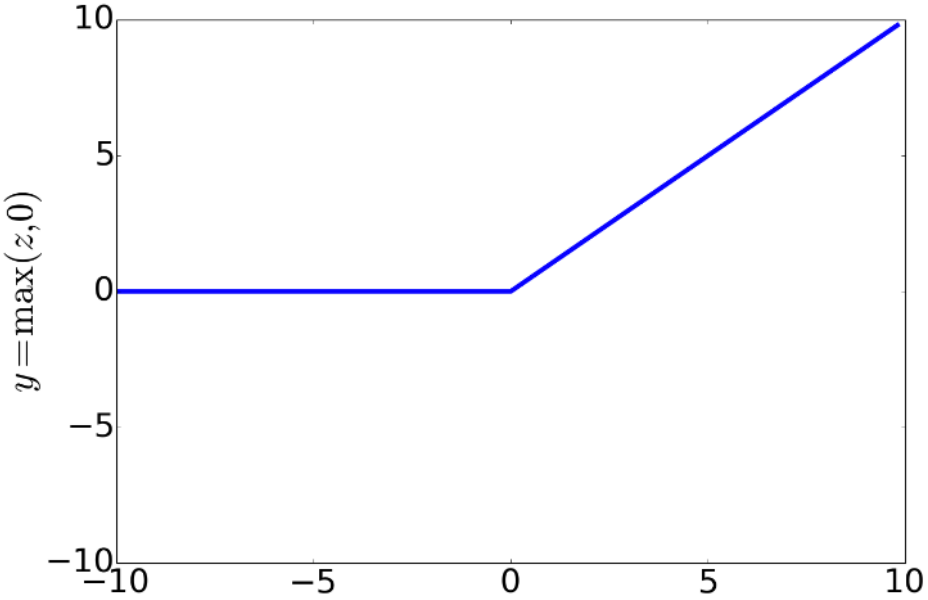
Non-Linear Activation Functions besides sigmoid

Most Common:



tanh

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

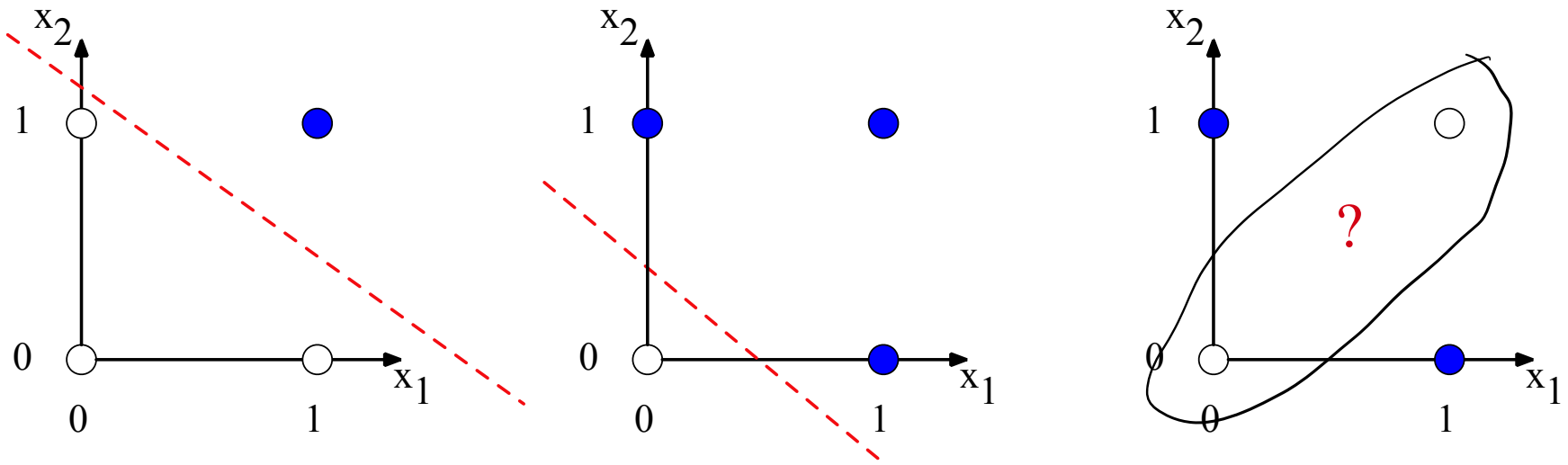


ReLU

Rectified Linear Unit

$$y = max(z, 0)$$

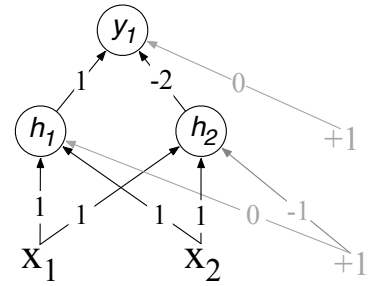
Decision boundaries

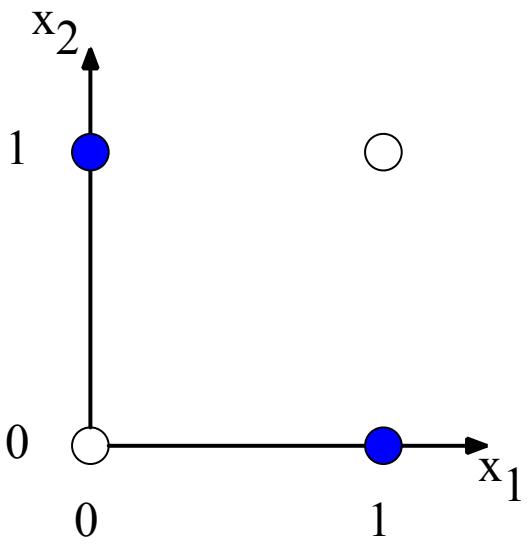


a) x_1 AND x_2 b) x_1 OR x_2 c) x_1 XOR x_2

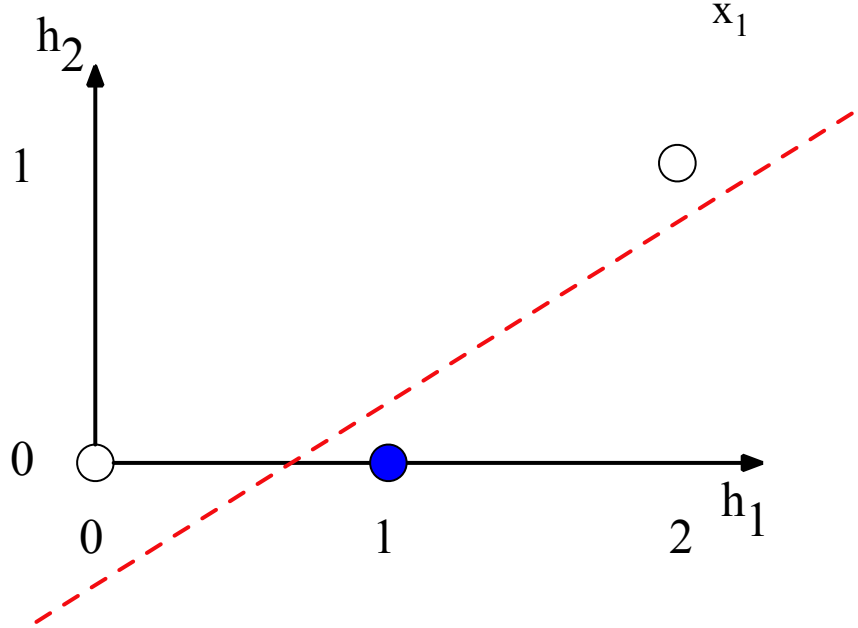
XOR is not a **linearly separable** function!

The hidden representation h





a) The original *x* space



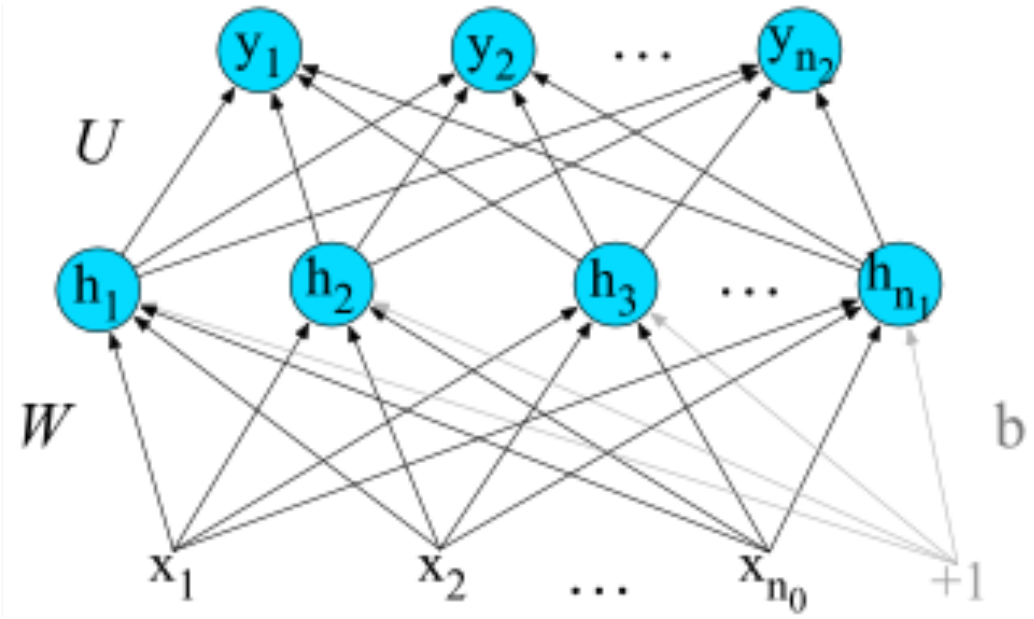
b) The new (linearly separable) *h* space

(With learning: hidden layers will learn to form useful representations)

Feedforward Networks

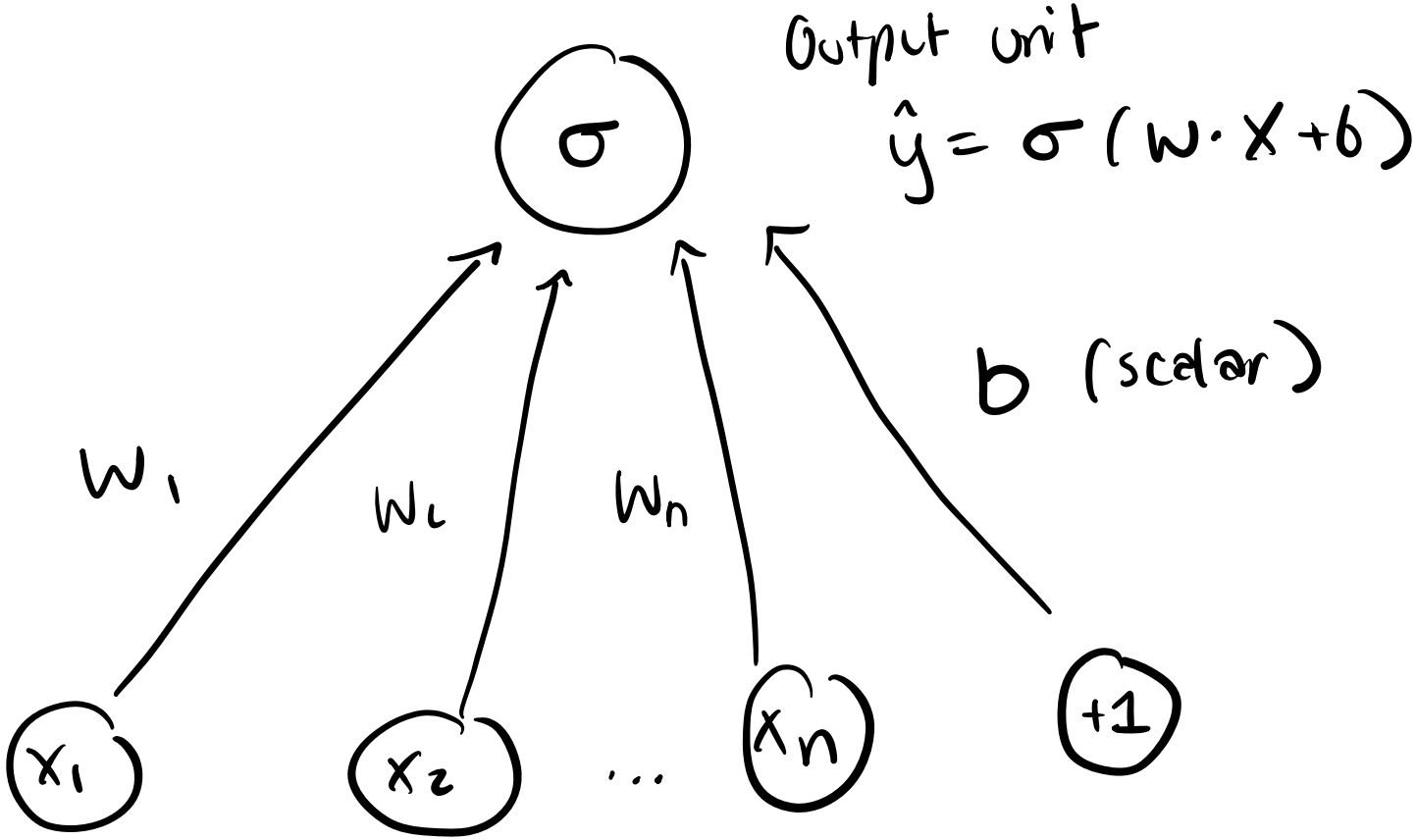
Feedforward Neural Networks

Can also be called **multi-layer perceptrons** for historical reasons



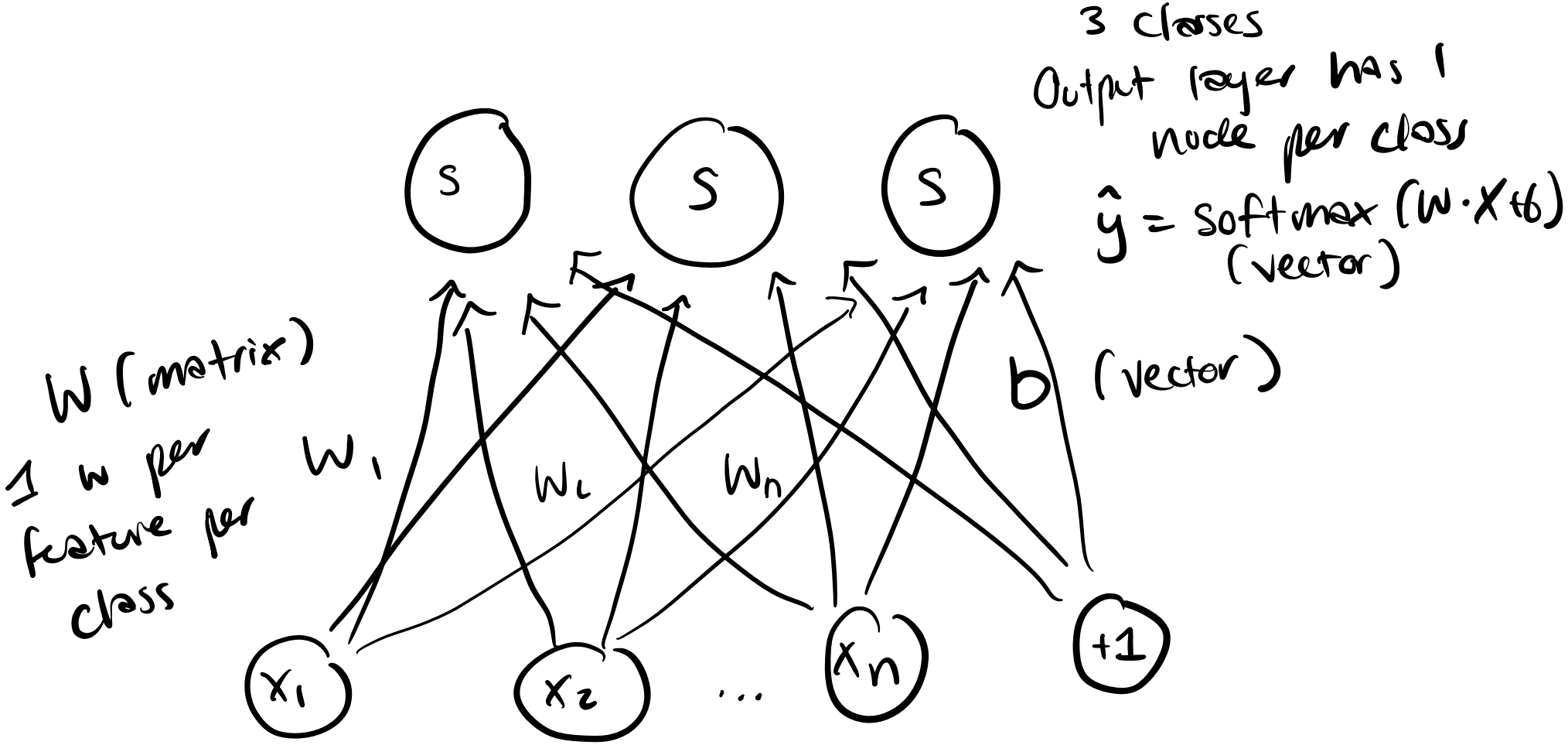
Binary Logistic Regression as a 1-layer Network

(we don't count the input layer in counting layers!)



Multinomial Logistic Regression as a 1-layer Network

Fully connected single layer network



Reminder: softmax: a generalization of sigmoid

For a vector *z* of dimensionality *k*, the softmax is:

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

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

Example:

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

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

Replacing the bias unit

Let's switch to a notation without the bias unit

Just a notational change

1. Add a dummy node $a_0=1$ to each layer
2. Its weight w_0 will be the bias
3. So input layer $a^{[0]}_0=1$,
   - And $a^{[1]}_0=1$, $a^{[2]}_0=1$,…

Replacing the bias unit

Instead of:

$x = x_1, x_2, \ldots, x_{n0}$

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

$$y_j = \sigma\left(\sum_{i=1}^{n_0} \mathbf{W}_{ji}\,\mathbf{x}_i + \mathbf{b}_j\right)$$

We'll do this:

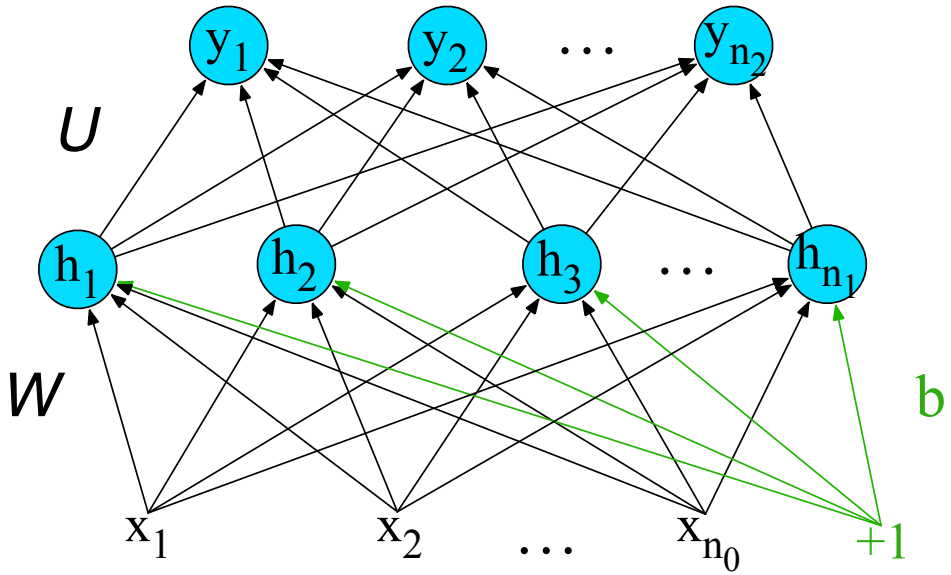
$x = x_0, x_1, x_2, \ldots, x_{n0}$

$$y = \sigma(\mathbf{W}\mathbf{x})$$

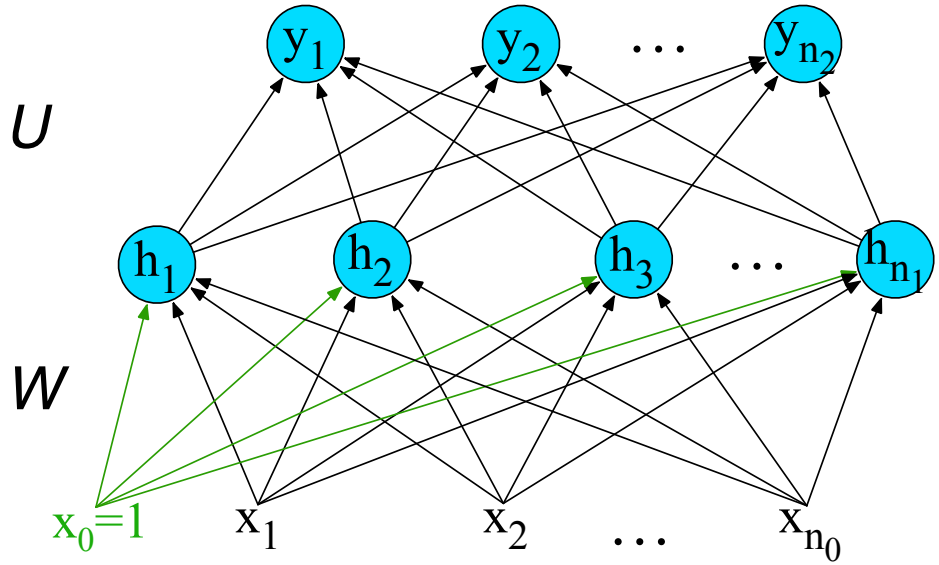
$$\sigma\left(\sum_{i=0}^{n_0} \mathbf{W}_{ji}\,\mathbf{x}_i\right)$$

Replacing the bias unit

Instead of:



We'll do this:



Using feedforward networks

Use cases for feedforward networks

Let's reconsider text classification

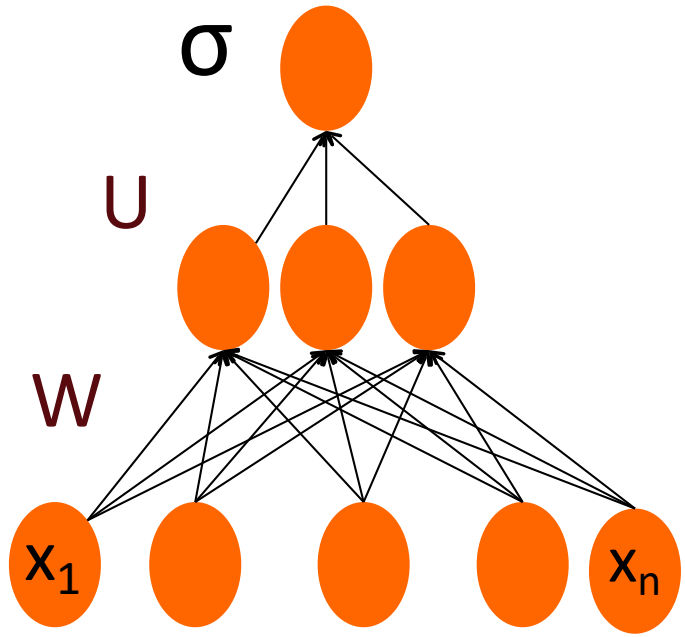
(State-of-the-art systems use more powerful architectures)

Classification: Sentiment Analysis

We could do exactly what we did with logistic regression

Input layer are binary features as before

Output layer is 0 or 1

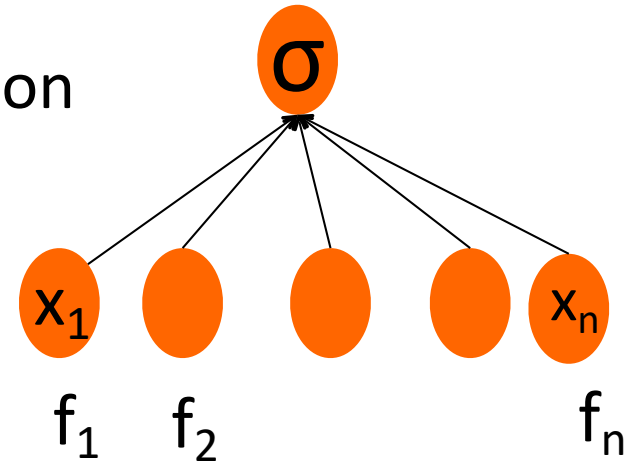


Sentiment Features

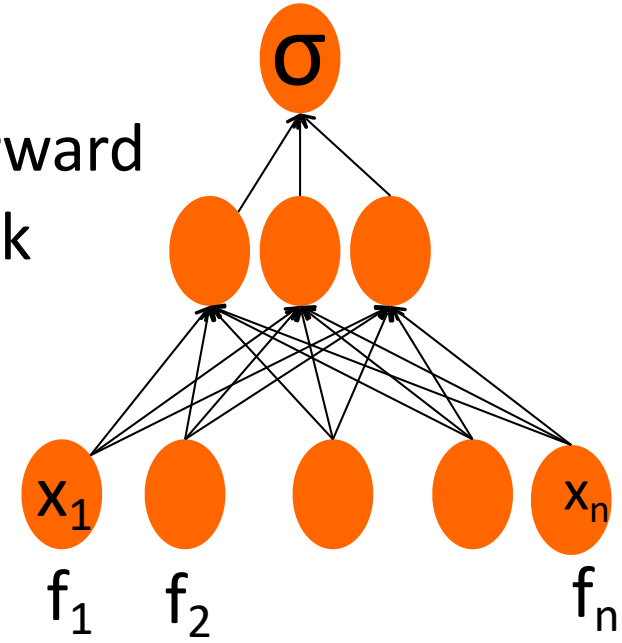
Var	Definition
x_1	count(positive lexicon words $\in$ doc)
x_2	count(negative lexicon words $\in$ doc)
x_3	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Feedforward nets for simple classification

Logistic Regression



2-layer feedforward network



Just add a hidden layer to logistic regression

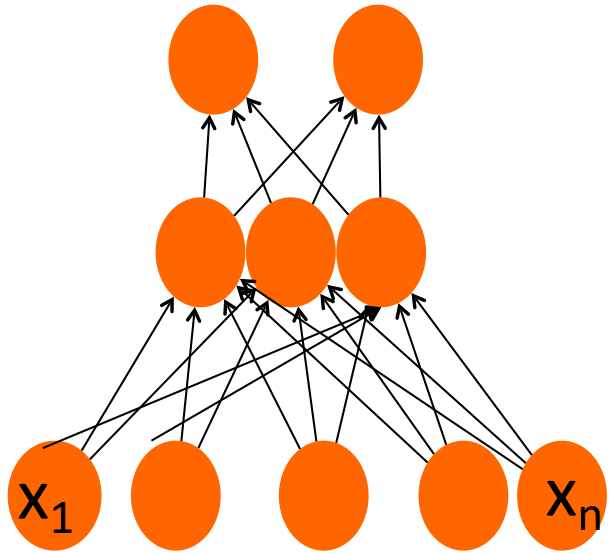
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance.

Reminder: Multiclass Outputs

What if you have more than two output classes?

- Add more output units (one for each class)
- And use a "softmax layer"

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \leq i \leq D$$



Training a Feedforward Network

Learning in Supervised Classification

Supervised classification:

- We know the correct label y (either 0 or 1) for each x.
- But what the system produces is an estimate, $\hat{y}$

We want to set W to minimize the **distance** between our estimate $\hat{y}^{(i)}$ and the true $y^{(i)}$.

- We need a distance estimator: a **loss function** or a **cost function**
- We need an optimization algorithm to update W to minimize the loss.

Step 1: Compute Loss

1) Define a loss function $L(\theta)$ to minimize.

↑ weights

We want:

- measure performance
- smooth / convexish
- differentiable

We could use cross-entropy.

Today: we'll use Square loss:

$$L = \frac{1}{2}(y - \hat{y})^2$$

Hyperparameters

A **hyperparameter** is a variable in our model that is set beforehand rather than learned.

Learning Rate: how much do we update weights at each step?

Batch Size: how often we update weights?

Common compromise: mini batching

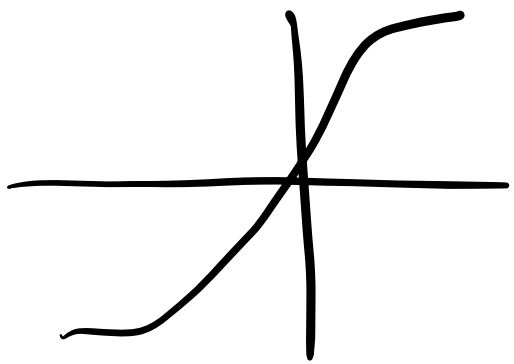
Single Neuron Feedforward Example

Inputs: (x, y)

$h = \tanh(w_1 X)$

$O = \tanh(w_2 h)$

tanh is a non-linear function:



squashes values to between $-1, 1$

Step 1: Compute Loss

Step 1: Compute Loss

$$L = \frac{1}{2}(y - \hat{y})^2$$ Square Loss

Step 2: Compute the Gradient of the Loss

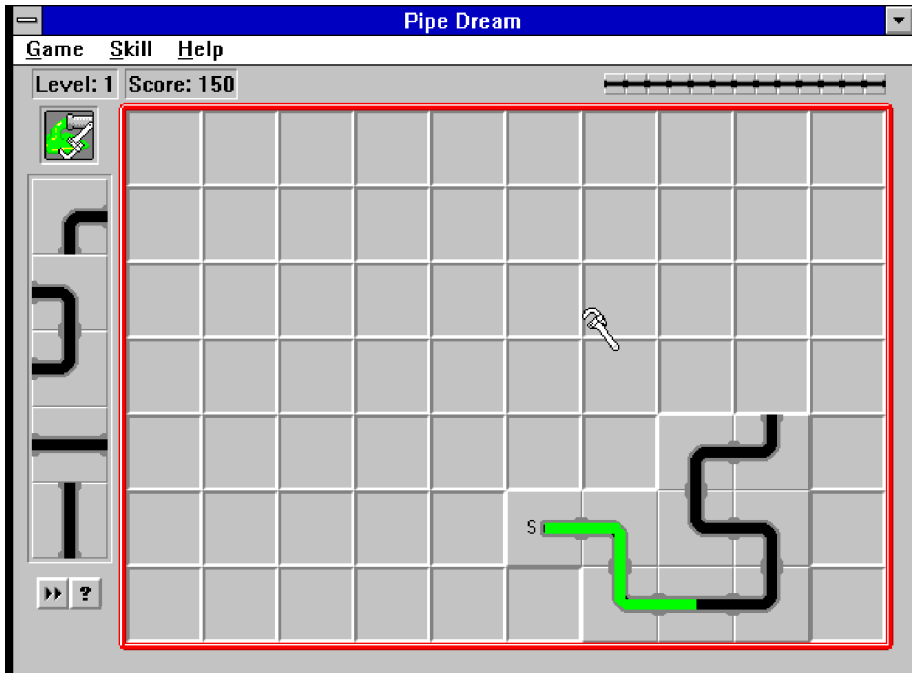
$$\frac{\partial L}{\partial \theta} : \frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}$$

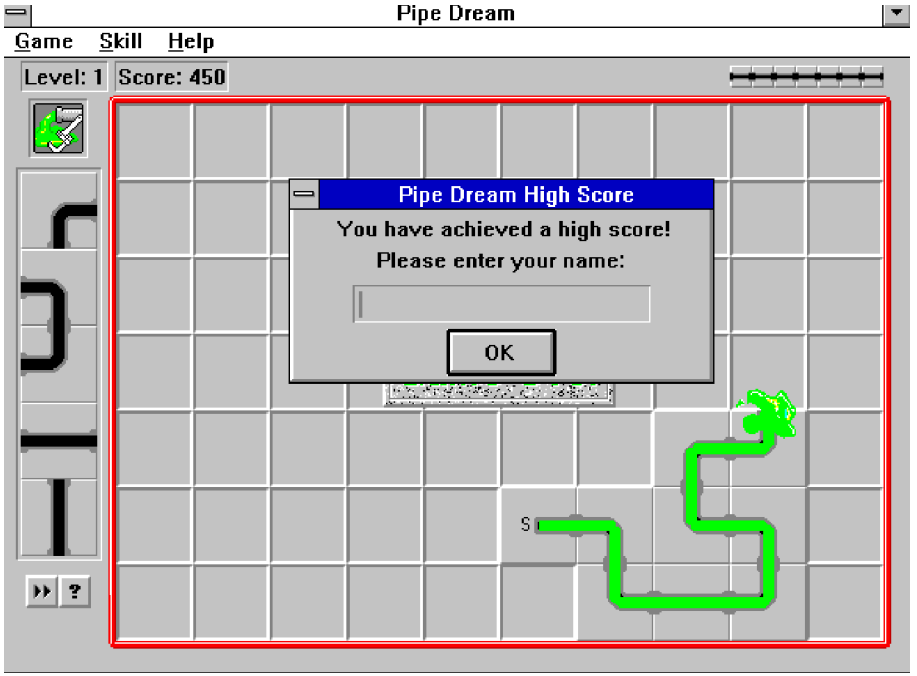
Chain Rule of Calculus: $\frac{\partial g(f(x))}{\partial x} = \frac{\partial g}{\partial f} \cdot \frac{\partial f}{\partial x}$

Derivative of tanh:

$$\frac{\partial \tanh(x)}{\partial x} = 1 - \tanh^2(x)$$

Data Flows Through the Network





https://archive.org/details/win3_PipeDr3x

Step 3: Update the Weights

$$W_{2_{new}} = W_{2_{old}} - \eta \frac{\partial L}{\partial w_2}$$

$$W_{1_{new}} = W_{1_{old}} - \eta \frac{\partial L}{\partial w_1}$$

Incorporating Embeddings

Even better: representation learning

The real power of deep learning comes from the ability to **learn features** from the data, instead of using hand-built human-engineered features for classification.

Neural Net Classification with embeddings as input features!