

CSCI 544

Applied Natural Language Processing

Mohammad Rostami
USC Computer Science Department



Logistical Notes

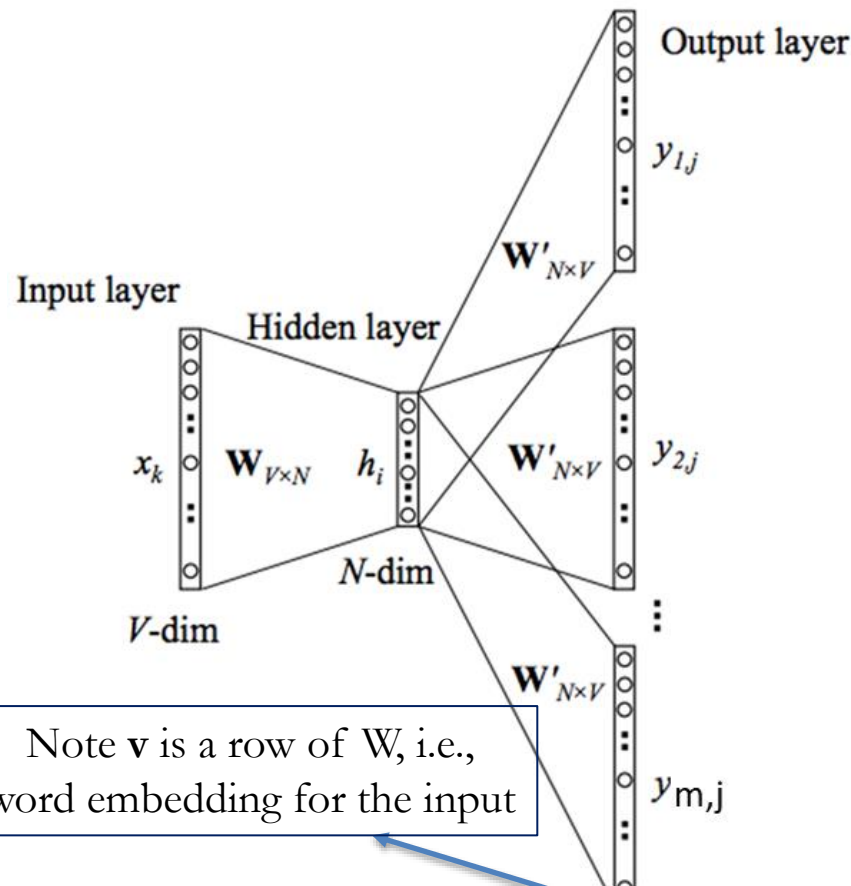
- Project Teams: ~50 groups are formed
 - Topics & written assignment
- Quiz 3
- HW1 deadline:
 - Zipped folder: name-family
- HW2: 3 weeks

Word2Vec

- Core idea: find embeddings using a prediction task involving **neighboring words** in a huge real-world corpus.
 - Input data can be sets of **successive word-patterns** from meaningful sentences in the corpus, e.g., “one of the most important”.
 - Try to build **synthetic** prediction tasks using these patterns, e.g., “(one of __ most important, the)”
 - Train a model to solve the prediction task
 - Embeddings are found as a **byproduct** of this process
- More specifically:
 - **We consider a window with the center word w_t and “context words” $w_{t'}$ with a window fixed size, e.g., $(t'=t-5, \dots, t-1, t+1, \dots, t+5)$.**
 - The model is assumed to be a two-layer neural network
 - We train the network to predict all w_t given $w_{t'}$ such that $p(w_t | w_{t'})$ is maximized
 - We learn embeddings such that the prediction loss is minimized, i.e., if two words occur in close proximity, their representations become similar.

Skip-Gram

- Given a center word, we predict the context words:
- Vocabulary size: V
- Input layer: center word in 1-hot form.
- The row k in $W_{V \times N}$ is the vector embedding of k -th center word.
- The column k of $W'_{N \times V}$ is context vector of the k -th word.
- At output layer y_{ij} , $i=1..M$ is computed:
 - We use the context word 1-hot vector to choose its column in $W'_{N \times V}$
 - dot product with h_i for the center word
 - compute the softmax
 - Match** the output one-hot vector
- After optimization, we will have two vectors for each word. We can set the eventual embedding to be the average of these two vectors



$$h_{N \times 1} = W_{V \times N}^T x_{V \times 1} = v$$

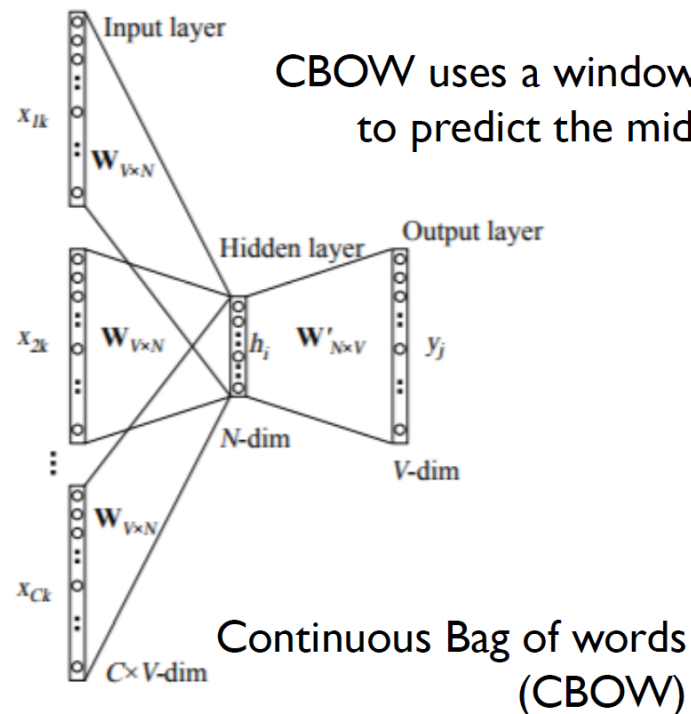
$$\hat{z}_j = h_{N \times 1}^T W'[:, j]_{N \times 1} = v^T u$$

$$\hat{y} = \sigma(W_{N \times V}'^T h_{N \times 1})$$

Note u is a row of W' , i.e., word embedding for the output

Continuous Bag of Words

- Given a context word, we predict the center word:
 - Vocabulary size: V
 - Input layer: context words in 1-hot form.
 - The row k in $W_{V \times N}$ is the vector embedding of k -th context word.
 - The column k of $W'_{N \times V}$ is vector of the k -th center word.
 - The output column y_{ij} , $i=1..M$ is computed:
 - We use the center word 1-hot vector to choose its column in $W'_{N \times V}$
 - dot product with h_i for the context word
 - compute the softmax
 - We can set the eventual embedding to the average of these two vectors
- Skip-gram incorporates non-frequent words better than CBOW



- Ex: Today is really __ day
- Skip-gram: delightful+context
and nice+context

Word2Vec Optimization Problem

- Consider an arbitrary order on vocabulary and let v_t be the word vector for center word t and u_t be the word vector for the context word t :

Ex: [...like to **eat** lunch and...]

- We generate binary tasks: (eat, like), (eat, to),...
- We solve for the word vector by maximizing the following likelihood function

$$v_t, u_t = \arg \max \log(\prod_{t=1}^T P(c|w_t)) =$$

$$\arg \max \log(\prod_{t=1}^T \prod_{j=-M}^M P(w_{t-j}|w_t)) =$$



Important
Step

$$\arg \max \frac{1}{T} \sum_{t=1}^T \sum_{j=-M, j \neq 0}^M \log(P(w_{t-j}|w_t)) =$$

Word2Vec Optimization Problem

- Conditional Probability modeling

Note \mathbf{v} denotes word embeddings for center words

$$P(w_{t-j}|w_t) = \frac{e^{u_{t-j}^T v_t}}{\sum_{t=1}^T e^{u_{t-j}^T v_t}}$$

Is it an extension of the logistic function?

Computationally expensive!

Note \mathbf{u} denotes word embeddings for the context words

- The log-likelihood optimization problem: one sum!

$$u_o, v_c = \arg \min \frac{1}{V} \sum_{w=1}^V -u_o^T v_c + \log\left(\sum_{w'=1}^V e^{u_o^T v_c}\right)$$

- Can be solved similar to logistic regression objective using numerical optimization techniques, e.g., gradient descent

Word2Vec Optimization Problem

- gradient descent step

$$u_o, v_c = \arg \min \frac{1}{V} \sum_{w=1}^V -u_o^T v_c + \log\left(\sum_{w'=1}^V e^{u_o^T v_c}\right) =$$

$$v_c^{i+1} = v_c^i - \eta \nabla f(v_c^i) =$$

$$\nabla f(v_c^i) = -u_o + \sum_v \frac{e^{u_o^T v_c}}{\sum_{w'=1}^V e^{u_o^T v_c}} u_o =$$

$$\nabla f(v_c^i) = -u_o + \sum_{w=1}^V p(u_o | v_c) u_o = -u_o + E(u_o | v_c)$$

- Tutorial:

<https://www.kaggle.com/pierremegret/gensim-word2vec-tutorial>

Negative Sampling

- Word2Vec optimization is a highly computationally intensive problem: the **shallow** network has a large number of weights and we will have billions of pairs
- Because we use one-hot vectors, each training pair (c,o) contributes minimally to updating the weights
- Negative sampling: for each positive pair, we randomly generate negative pairs, for which the network output should be 0.

$$u_o, v_c = \arg \min \sigma(u_o^T v_c) + \sum_{k=1}^K E_{v_{w_i} \sim P(w_c)} \log(-\sigma(v_{w_i}^T v_c))$$

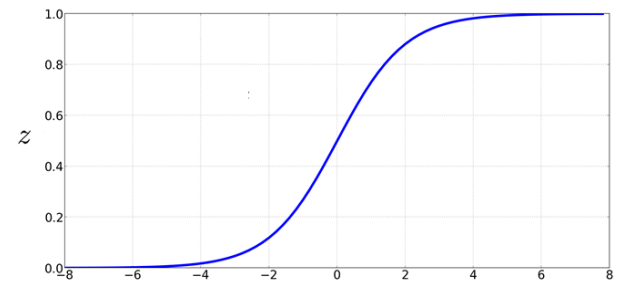
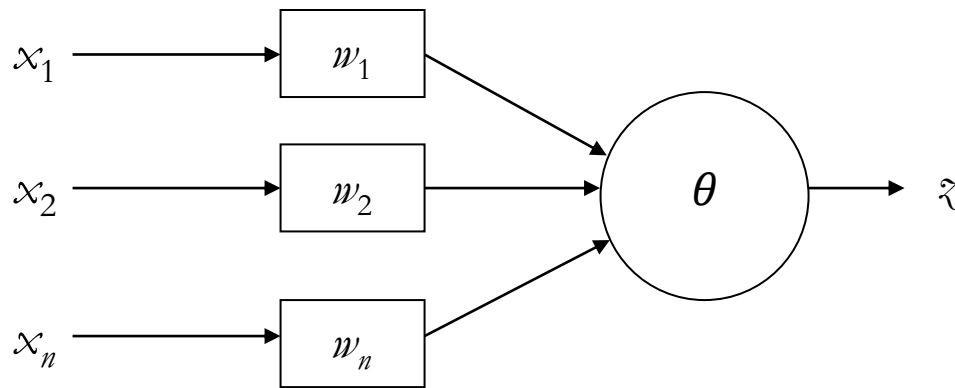
- This is an instance of naïve data augmentation

Neural vs Factorization-Based Embeddings

- **Comparison is challenging** (Levy and Goldberg, NeurIPS 2014):
 - **Hyperparameters**
 - Factorization algorithm
 - Amount of data
- A particular word embedding approach is unlikely to be state-of-the-art for all applications
- Hyperparameters appear to have the largest impact in performance.
- Neural models are less sensitive with respect to hyperparameters, and training data preparation is more straightforward and systematic

Neural Networks

- Perceptron: the neural network learning model in the 1960's
- Simple and limited (single layer models)
- Basic concepts are similar for multi-layer models



$$z = 1/(1 + \exp(-x))$$

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^n x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^n x_i w_i < \theta \end{cases}$$

XOR Problem

- Can perceptron be used to learn XOR?

$$ax_1 + bx_2 + c = 0$$

$$\begin{aligned} a \cdot 0 + b \cdot 0 + c &< 0 \\ a \cdot 0 + b \cdot 1 + c &> 0 \\ a \cdot 1 + b \cdot 0 + c &> 0 \\ a \cdot 1 + b \cdot 1 + c &< 0 \end{aligned}$$



$$\begin{aligned} c &< 0 \\ b + c &> 0 \\ a + c &> 0 \\ a + b + c &< 0 \end{aligned}$$

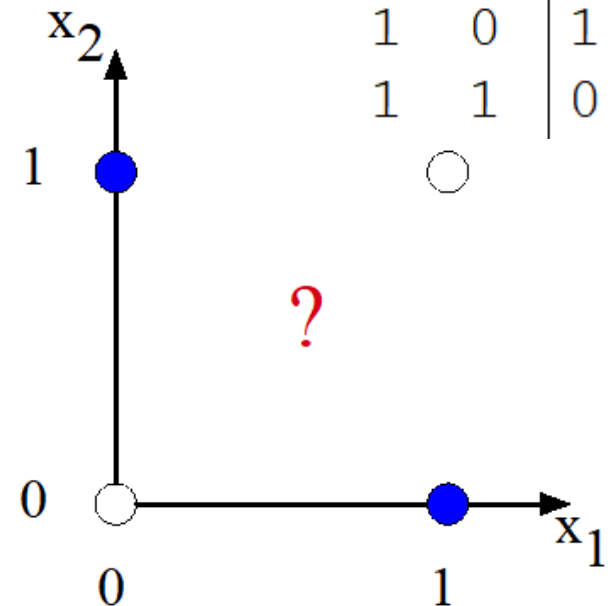


$$\begin{aligned} c &< 0 \\ a + b + 2c &> 0 \\ -a - b - c &> 0 \end{aligned}$$



$$\begin{aligned} c &< 0 \\ c &> 0 \end{aligned} \quad !$$

-



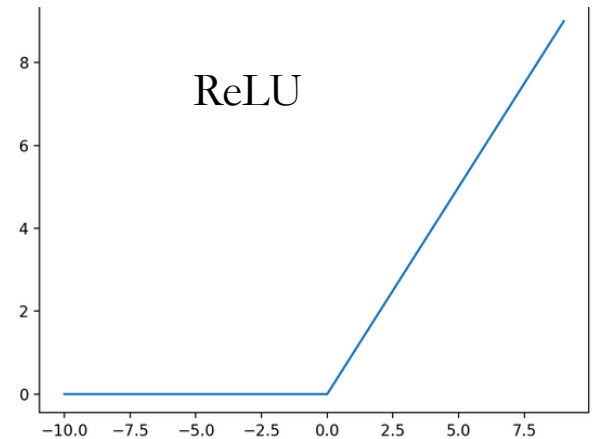
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

c) $x_1 \text{ XOR } x_2$

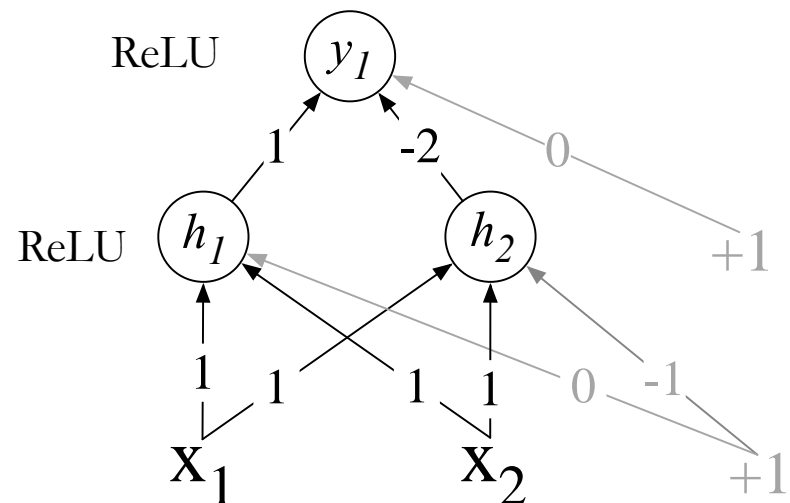
XOR Problem

- Idea: XOR function can be computed using two layers perceptron units.

$$y_1 = \text{Relu}(\text{Relu}(x_2 + x_1) - 2\text{Relu}(x_2 + x_1 - 1))$$

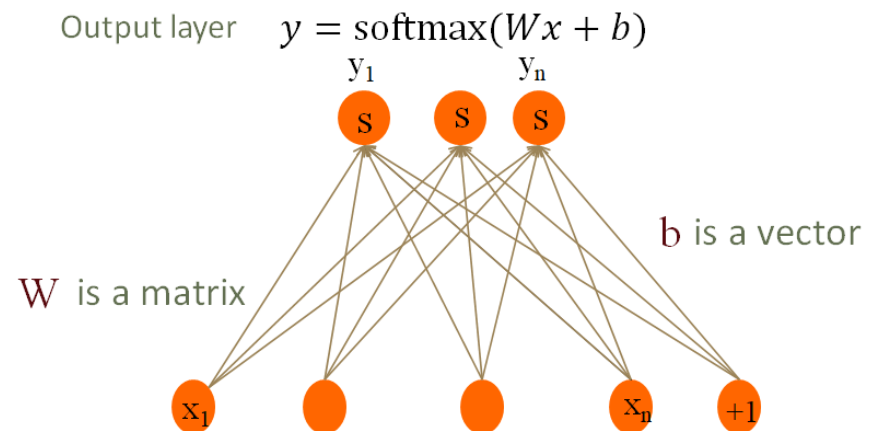
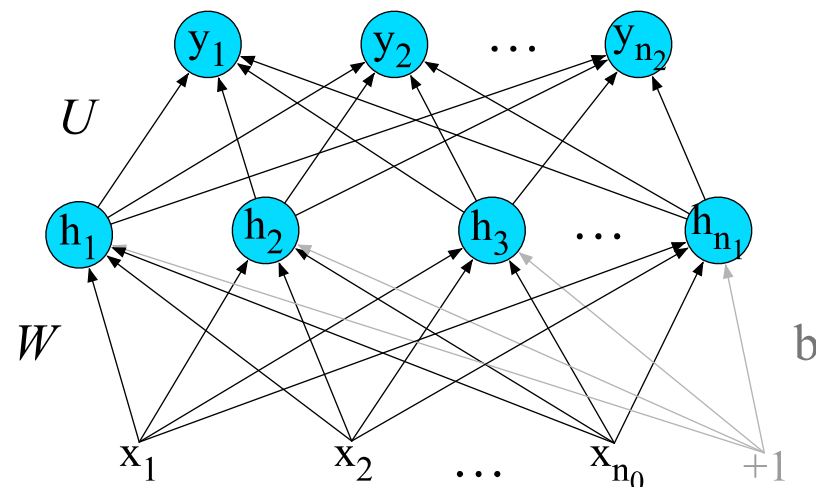


XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



Feedforward Neural Networks

- An extension to perceptron by forming series connection between layers of parallel perceptrons: input, output, and hidden layers
- The number of layers, the nodes at each layer, and non-linear functions are design parameters



Multilayer Perceptron

- In the default setting, a node in hidden layers receives inputs from all nodes in the previous layer and its output is fed to all nodes in the next layer

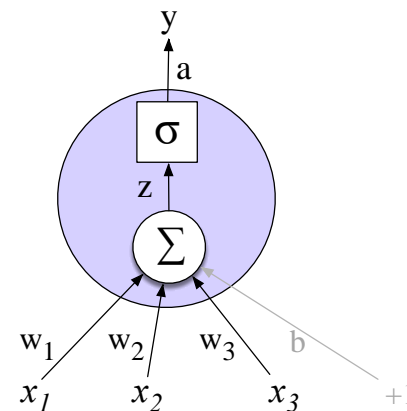
$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

$$a^{[1]} = g^{[1]}(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]})$$

$$\hat{y} = a^{[2]}$$



for i in 1..n

$$z^{[i]} = W^{[i]}a^{[i-1]} + b^{[i]}$$

$$a^{[i]} = g^{[i]}(z^{[i]})$$

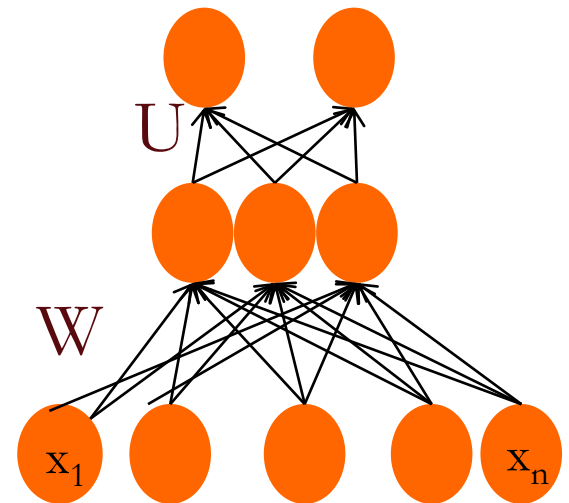
$$\hat{y} = a^{[n]}$$

Multiclass Outputs

- For a multiclass classification problem:
 - We add one output for each class
 - We use a “softmax layer” at the output to generate a probability distribution
 - We use a proper loss function at the output

$$\text{Loss} = - \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

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

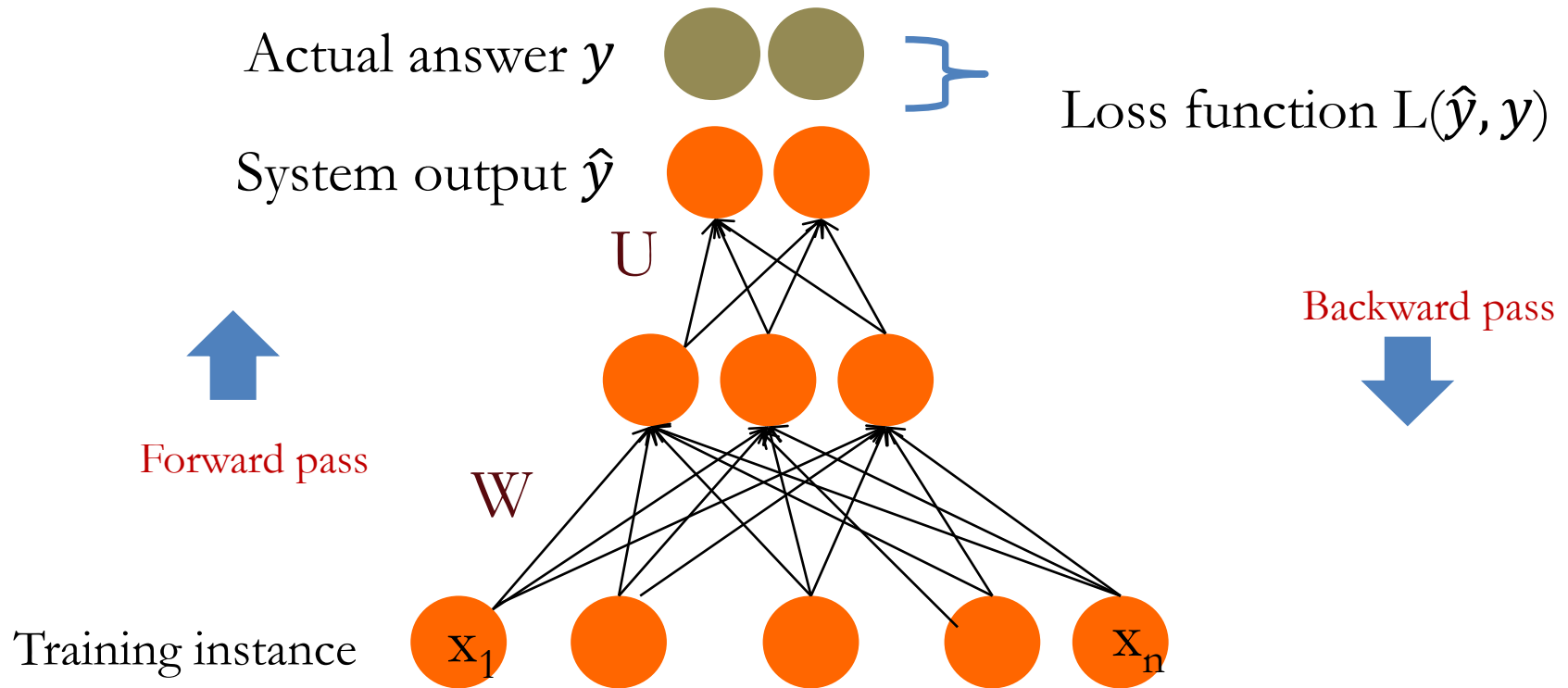


Universal Approximation Theorem

- MLPs can represent a wide range of functions given appropriate values for the weights.
(George Cybenko in 1989)
 - Given sufficient layers, i.e., deep nets
 - Given sufficient nodes, i.e., wide nets
- Only existential result: it merely states that approximating most given functions is possible but does not provide the solution.

Training Feedforward Networks

- Backpropagation algorithm: an iterative algorithm to learn network weights using annotated data

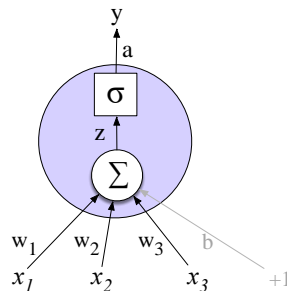


Training Feedforward Networks

- For every training data point (x, y)
 - Run *forward* computation to find model 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 using gradient descent $\frac{d}{dw} L(f(x; w), y)$

 - For all other nodes
 - Assess how much blame it deserves for the current answer



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

Backpropagation for Two Layer Network Output

$$z^{[1]} = W^{[1]}\mathbf{x} + b^{[1]}$$

$$a^{[1]} = \text{ReLU}(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

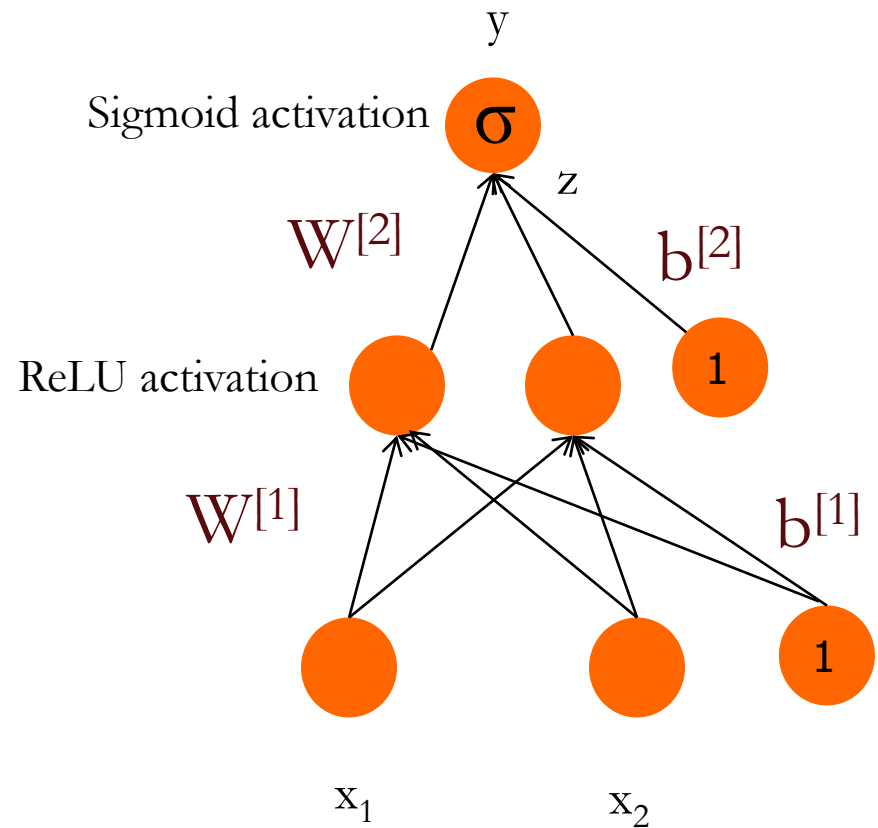
$$a^{[2]} = \sigma(z^{[2]})$$

$$\hat{y} = a^{[2]}$$

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

$$\frac{d \text{ReLU}(z)}{dz} = \begin{cases} 0 & \text{for } z < 0 \\ 1 & \text{for } z \geq 0 \end{cases}$$

$$\frac{\partial y}{\partial z} = \frac{d\sigma(z)}{dz} = \sigma(z)(1 - \sigma(z))$$



Backpropagation for Two Layer Network Output

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

$$L(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

$$y = \sigma(z)$$

$$z = \sum w_i h_i + b$$

$$\begin{aligned} \frac{\partial L}{\partial y} &= - \left(\left(y \frac{\partial \log(a)}{\partial a} \right) + (1 - y) \frac{\partial \log(1 - a)}{\partial a} \right) \\ &= - \left(\left(y \frac{1}{a} \right) + (1 - y) \frac{1}{1 - a} (-1) \right) = - \left(\frac{y}{a} + \frac{y - 1}{1 - a} \right) \end{aligned} \quad \text{Notational Simplicity}$$

$$\frac{\partial a}{\partial z} = a(1 - a)$$

$$\frac{\partial z}{\partial w_i} = h_i$$

$$\frac{\partial L}{\partial w_i} = - \left(\frac{y}{a} + \frac{y - 1}{1 - a} \right) a(1 - a) h_i = (a - y) h_i$$

$$z^{[1]} = W^{[1]} \mathbf{x} + b^{[1]}$$

$$a^{[1]} = \text{ReLU}(z^{[1]})$$

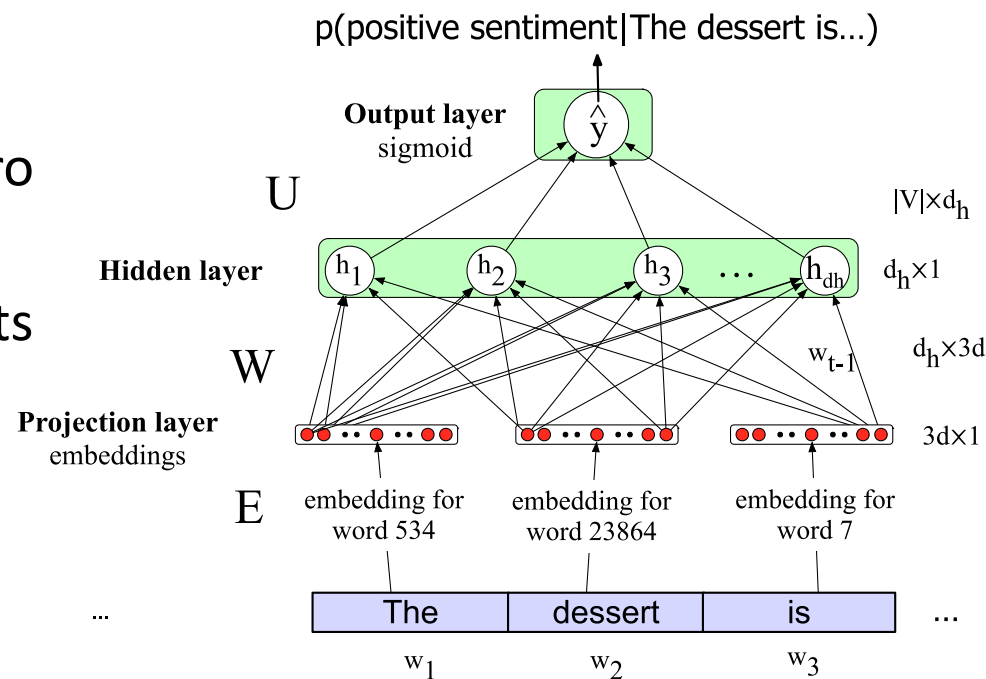
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

$$\hat{y} = a^{[2]}$$

MLP for NLP Tasks

- Assume a fixed size length
1. Make the input the length of the longest input
 - If shorter then pad with zero embeddings
 - Truncate if you longer inputs are observed at test time
 2. Create a single “sentence embedding”
 - Take the mean of all the word embeddings
 - Take the element-wise max of all the word embeddings



Neural Networks vs SVM Rivalry

- Back Propagation: 1986
- SVM: 1992
- Deep Learning: 2012
- Three key reasons for reemergence of deep learning:
 - Computational power: NVIDIA CUDA (2007)
 - Annotated datasets: ImageNet (2009)
 - ReLU !!!