Lecture 10
Neural Networks and Neural Language Models

CS 6320

Outline

- A neural unit
- The XOR problem and solution
- Feed-forward network
- Training Neural Nets
- The Loss function
- Computing the Gradient
- Neural Language Models
- Training the neural language model

Introduction

- Neural networks share the same mathematics as logistic regression.
- Neural networks are more powerful; $A\ NN$ with one hidden layer can learn any function.
- NN classifiers do not need the rich set of hand-picked features, instead take raw words as inputs and parameters are trained/learned to classify correctly.
- Deep learning refers to modern NN with many layers.

Basic Unit

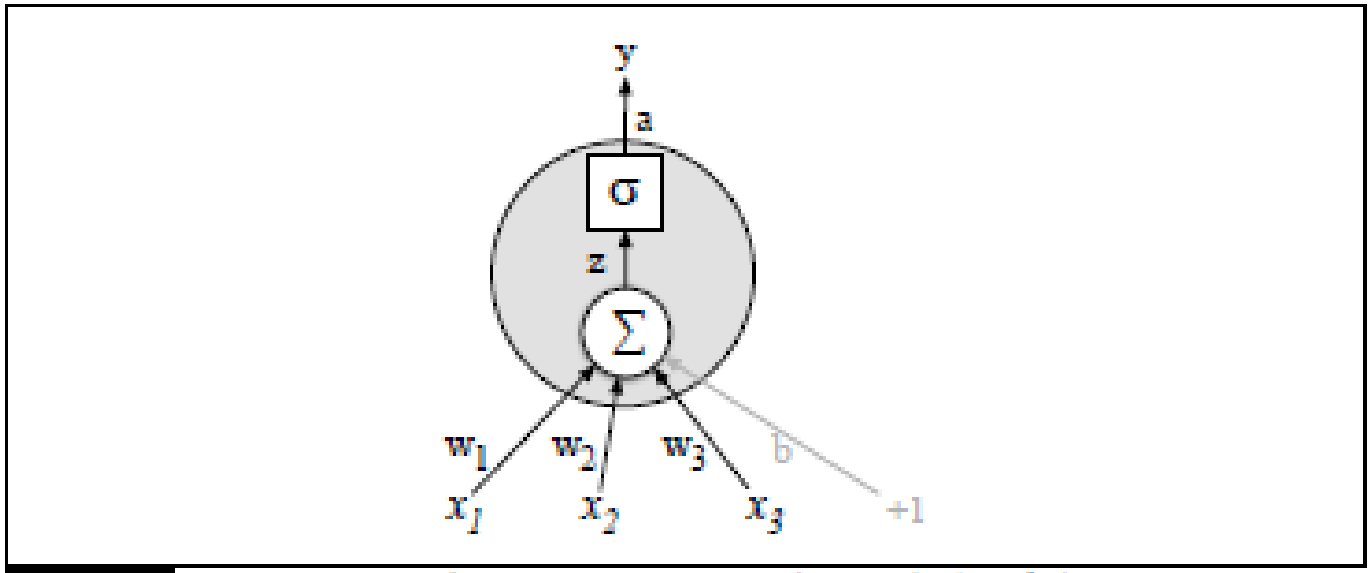


Figure 7.2 A neural unit, taking 3 inputs x_1, x_2, and x_3 (and a bias b that we represent as a weight for an input clamped at +1) and producing an output y. We include some convenient intermediate variables: the output of the summation, z, and the output of the sigmoid, a. In this case the output of 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.

Basic Unit

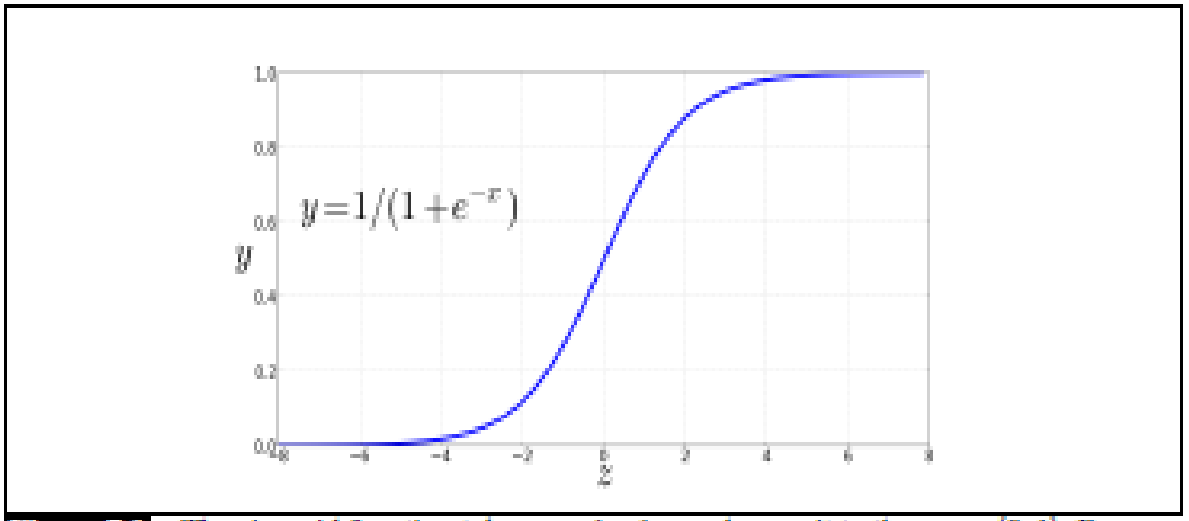


Figure 7.1 The sigmoid function takes a real value and maps it to the range [0, 1]. Because it is nearly linear around 0 but has a sharp slope toward the ends, it tends to squash outlier values toward 0 or 1.

Basic Unit

$$z = b + \sum_i w_i \, x_i$$

$z = w \cdot x + b$

Apply a non-linear function to z.

The output is called activation value

$y = a = f(z)$

Non-linear functions f:

- sigmoid
- tanh
- rectified linear ReLU

Sigmoid

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

Advantages of sigmoid

- Maps input z into output range [0, 1]
- Differentiable
- Acts as a probability

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

Example

$$w = [0.2, 0.3, 0.9]$$
$$b = 0.5$$
$$x = [0.5, 0.6, 0.1]$$

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

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

Tanh and ReLU

- Tanh

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

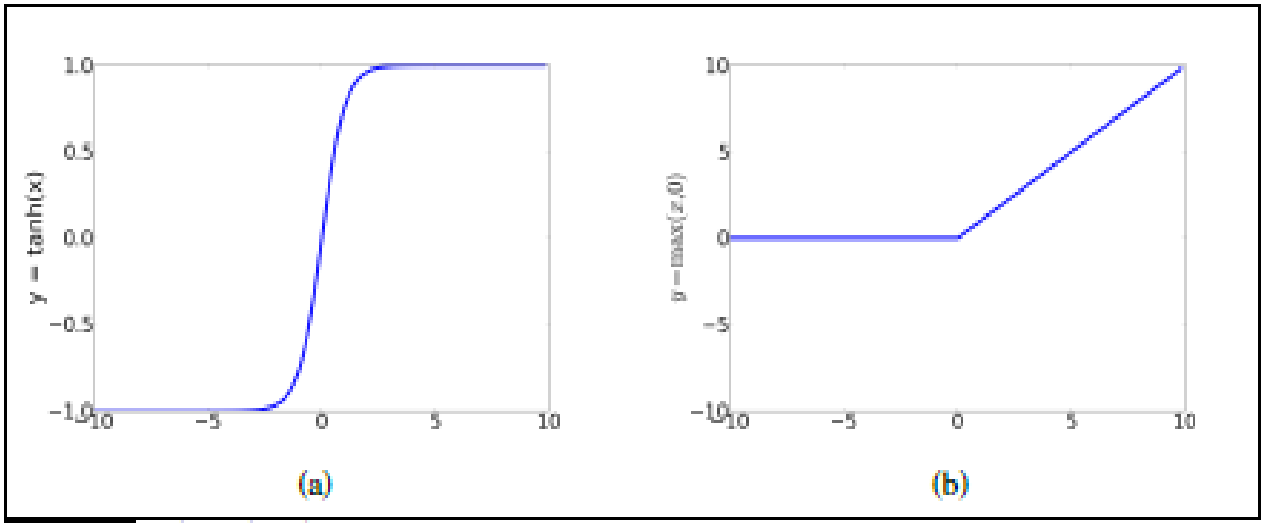


Figure 7.3 The tanh and ReLU activation functions.

- ReLU

$$y = \max(x, 0)$$

The XOR problem

AND			OR			XOR		
x1	x2	y	x1	x2	y	X1	X2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

Perceptron – a basic NN unit without non-linear activation.

$$y = \begin{cases} 0, & if\ w \cdot x + b \leq 0 \\ 1, & if\ w \cdot x + b > 0 \end{cases}$$

The XOR Problem

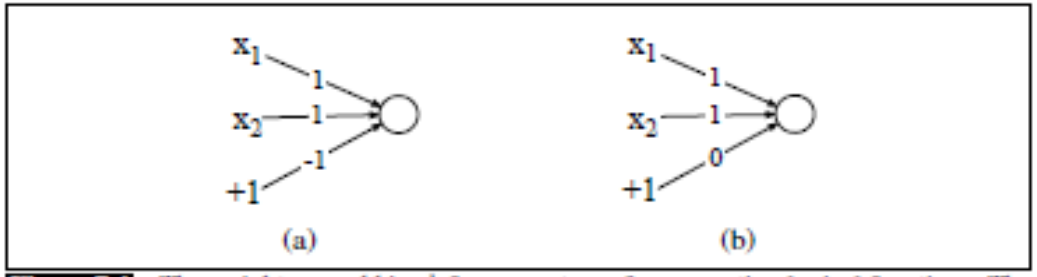


Figure 7.4 The weights w and bias b for perceptrons for computing logical functions. The inputs are shown as x_1 and x_2 and the bias as a special node with value $+1$ which is multiplied with the bias weight b. (a) logical AND, showing weights $w_1 = 1$ and $w_2 = 1$ and bias weight $b = -1$. (b) logical OR, showing weights $w_1 = 1$ and $w_2 = 1$ and bias weight $b = 0$. These weights/biases are just one from an infinite number of possible sets of weights and biases that would implement the functions.

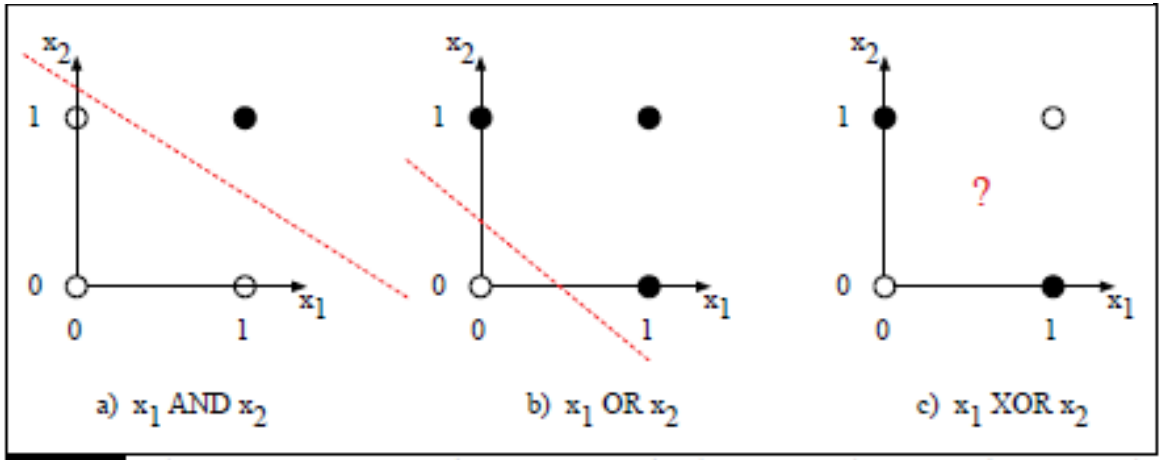


Figure 7.5 The functions AND, OR, and XOR, represented with input x_0 on the x-axis and input x_1 on the y axis, Filled circles represent perceptron outputs of 1, and white circles perceptron outputs of 0. There is no way to draw a line that correctly separates the two categories for XOR. Figure styled after Russell and Norvig (2002).

The XOR Problem

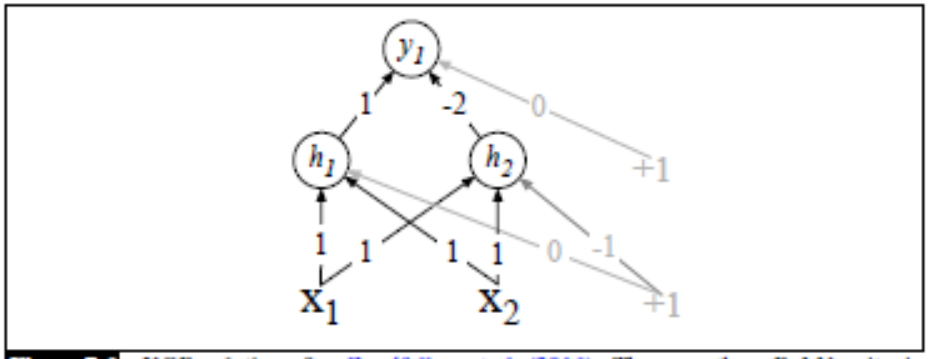


Figure 7.6 XOR solution after Goodfellow et al. (2016). There are three ReLU units, in two layers; we've called them h_1, h_2 (h for "hidden layer") and y_1. As before, the numbers on the arrows represent the weights w for each unit, and we represent the bias b as a weight on a unit clamped to +1, with the bias weights/units in gray.

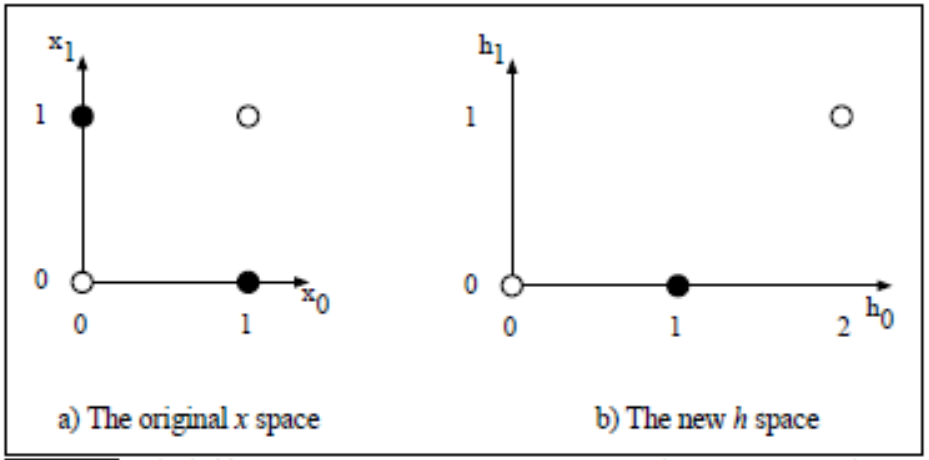


Figure 7.7 The hidden layer forming a new representation of the input. Here is the representation of the hidden layer, h, compared to the original input representation x. Notice that the input point [0 1] has been collapsed with the input point [1 0], making it possible to linearly separate the positive and negative cases of XOR. After Goodfellow et al. (2016).

Feed-Forward Neural Networks

- A feed-forward network is a multilayer network in which the units are connected with no cycles.
- Simple feed-forward networks have:
  - input units-scalar values
  - hidden units
  - output units

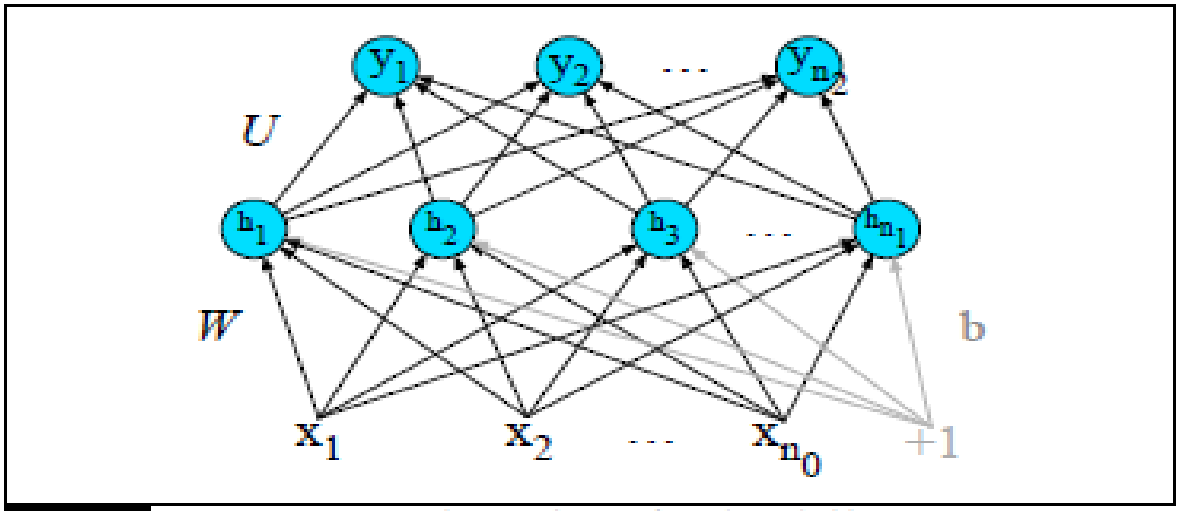


Figure 7.3 A simple 2-layer feed-forward network, with one hidden layer, one output layer, and one input layer (the input layer is usually not counted when enumerating layers).

- Fully-connected network
- Hidden layer is the core of *N.N.*

Feed-Forward Neural Networks

- Unit i has

  Weight vector w_i and bias b_i

- Form a single weight matrix W and a single bias vector b

  w_{ij} - is the weight from input unit x_j to the hidden unit h_i

- The output of hidden layer-vector h

  $h = \sigma(Wx + b)$

- Allow $\sigma(\cdot)$ and activation function $g(\cdot)$ to apply to a vector element-wise.

  $g[z_1, z_2, z_3] = [g(z_1), g(z_2), gz_3)]$

Feed-Forward Neural Networks

- x- is a vector $x \in \mathbb{R}^{n_0}$ with dimensionality n_0
- h- hidden layer 1 has dimensionality n_1

  $h \in \mathbb{R}^{n_1}$ and $b \in \mathbb{R}^{n_1}$
- W- weight matrix has dimensionality n_1 x n_0, W $\in \mathbb{R}^{n_1 \mathrm{x}\, n_0}$

$$h_i = \sum_{j=1}^{n_x} w_{ij} x_j + b_i$$

- Output layer has a weight matrix U of dimensionality $n_2 x n_1$ $U \in \mathbb{R}^{n_2 x n_1}$
- There are n_2 output nodes z $\in \mathbb{R}^{n_2}$

$$\begin{array}{ccccc} \mathrm{z} & = & U & & h \\ n_2\mathrm{x}1 & & n_2\mathrm{x}n_1 & & n_1\mathrm{x}1 \end{array}$$

Feed-Forward Neural Networks

- There is need to normalize the output vector z.
- Softmax function takes a vector z of dimensionality d and creates a probability distribution

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}} \qquad 1 \leq i \leq d$$

Example

z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]

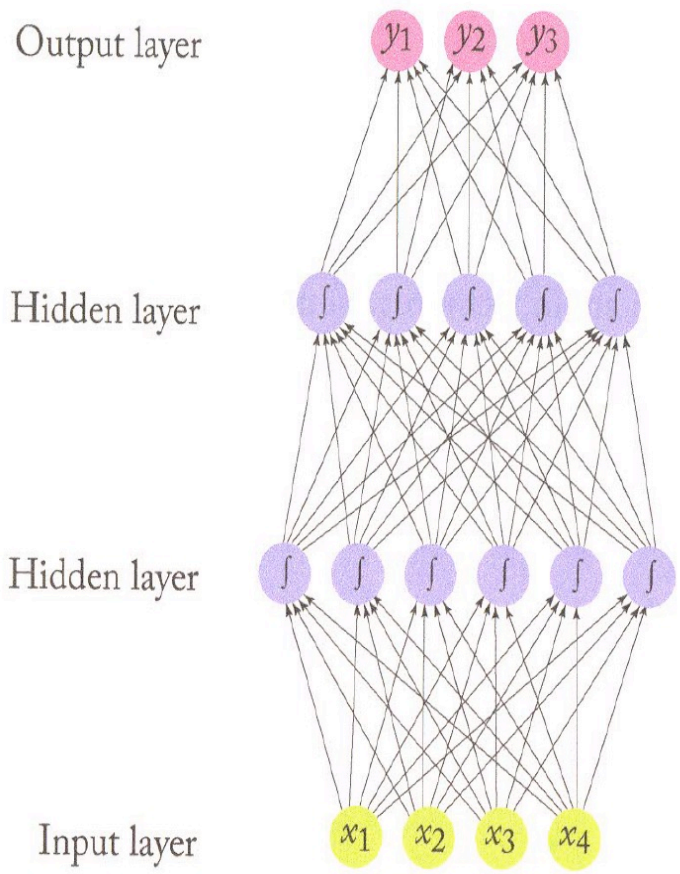
softmax (z)=[0.055, 0.09, 0.0067, 0.10, 0.74, 0.01]

For a 2-layer network

$$h = \sigma(Wx + b)$$
$$z = Uh$$
$$y = softmax(z)$$

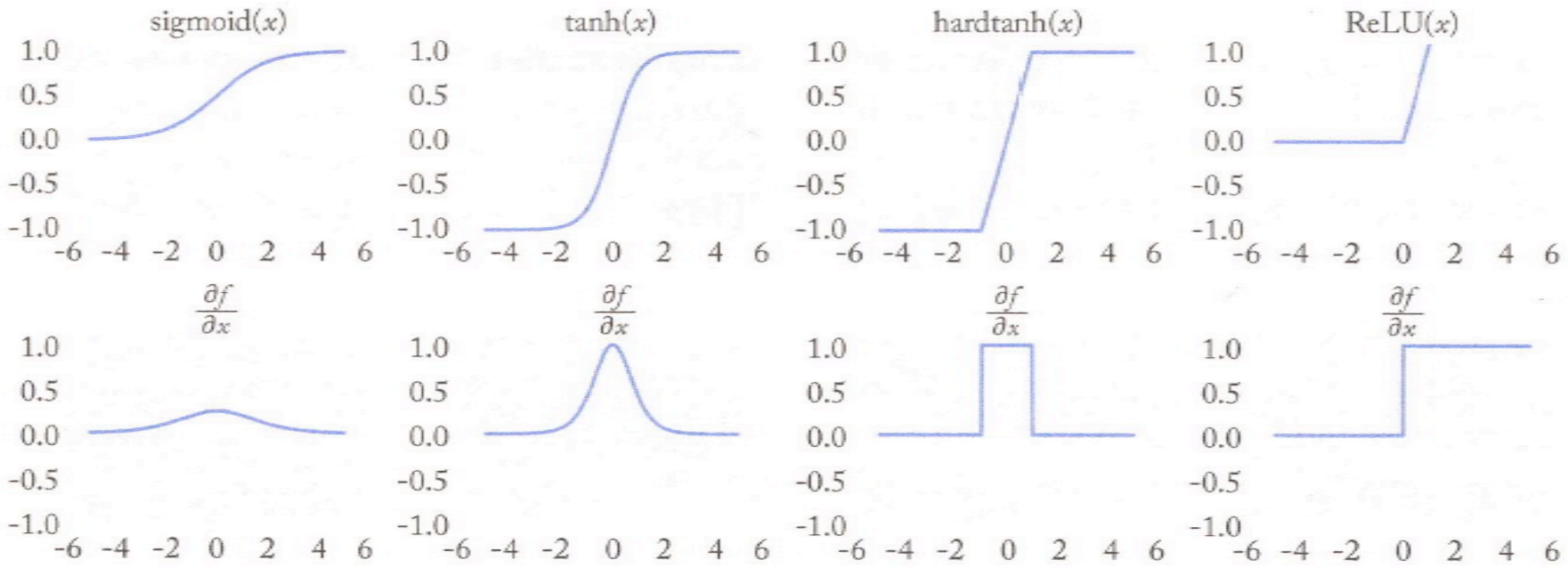
Feed-Forward Neural Networks

2-hidden layer network, and 1 output layer



Feed-Forward Neural Networks

Non-linear functions and their derivatives



Feed-Forward Neural Networks

- Notation: Use super scripts in square brackets to mean layer number. Input layer is layer 0.
  $W^{[1]}$ means the weight matrix for first hidden layer.

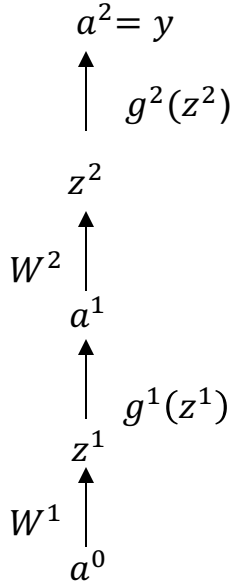
$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$

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

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

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

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



Training Neural Nets

- We know the correct output y for each observation x.
- System produces $\hat{y}$.

- Goal of the training procedure is to learn parameters $W^{[i]}$ and $b^{[i]}$ for each layer i to make $\hat{y}$ for each observation as close as possible to true y.

- Loss function – is the cross entropy loss.

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

- For multinomial classifier with C classes

$$L_{CE}(\hat{y}, y) = -\sum_{i=1}^{C} y_i \log \hat{y}$$

Training

- Assume hard classification task; only one class is the correct one

$$y_i = 1 \text{ and } y_j = 0 \quad \text{for } \forall j \neq i$$

- This is called one-hot vector

  Then

$$L_{CE}(\hat{y}, y) = -\log \hat{y}_i$$

  Only true class has $y_i = 1$, all others $y_j = 0$

- Use the softmax formula with K number of classes.

$$L_{CE} = -\log \frac{e^{z_i}}{\sum_{j-1}^{K} e^{z_j}}$$

Computing the Gradient

- Compute the derivatives of the loss function, as we did for logistic regression.

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

For one hidden layer and softmax output.

$$\frac{\partial L_{CE}}{\partial w_k} = (1\{y = k\} - p(y = k|x)x_k$$

$$= \left(1\{y = k\} - \frac{e^{W_k \cdot x + b_k}}{\sum_{j=1}^{k} e^{W_j \cdot x + b_j}}\right) x_k$$

- This gives correct weights for the last layer only.

- For multi-hidden layer we need an algorithm called error backpropagation or backprop.

Backprop Algorithm

- Computation Graphs
- Consider $L(a,b,c) = c(a+2b)$

$$d = 2 * b$$
$$e = a + d$$
$$L = c * e$$

- Forward pass

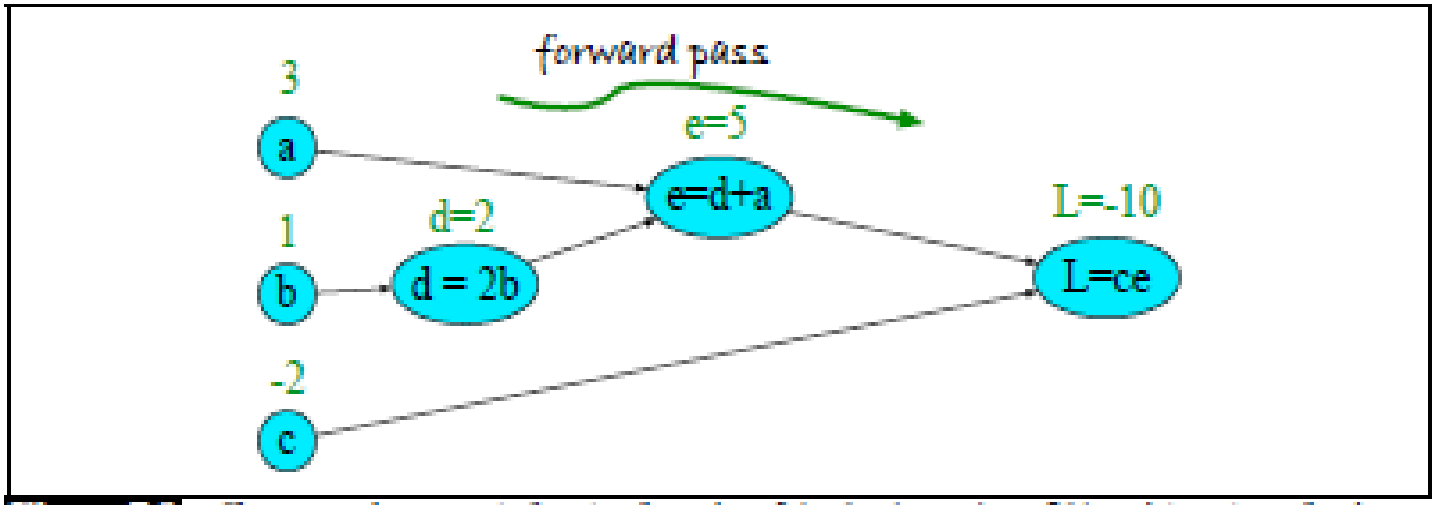


Figure 7.9 Computation graph for the function $L(a,b,c) = c(a+2b)$, with values for input nodes $a = 3, b = 1, c = -1$, showing the forward pass computation of L.

Backprop Algorithm

- Backward differentiation on computation graphs.

  Make use of the chain rule differentiation.

$$f(x) = u(v(w(x)))$$

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

$L = ce$: $\quad \frac{\partial L}{\partial e} = c = -2, \quad \frac{\partial L}{\partial c} = e = 5$

$e = a + d$: $\quad \frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a} = c \text{ x1} = -2, \quad \frac{\partial L}{\partial d} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d} = \text{c x 1} = \text{-2}$

$d = 2b$: $\quad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\ \frac{\partial e}{\partial d}\ \frac{\partial d}{\partial b} = c \text{ x 1 x 2} = -4$

Backprop Algorithm

- Backward pass

  Compute each of the partials along each edge from right to left; multiplying the necessary partials to results.

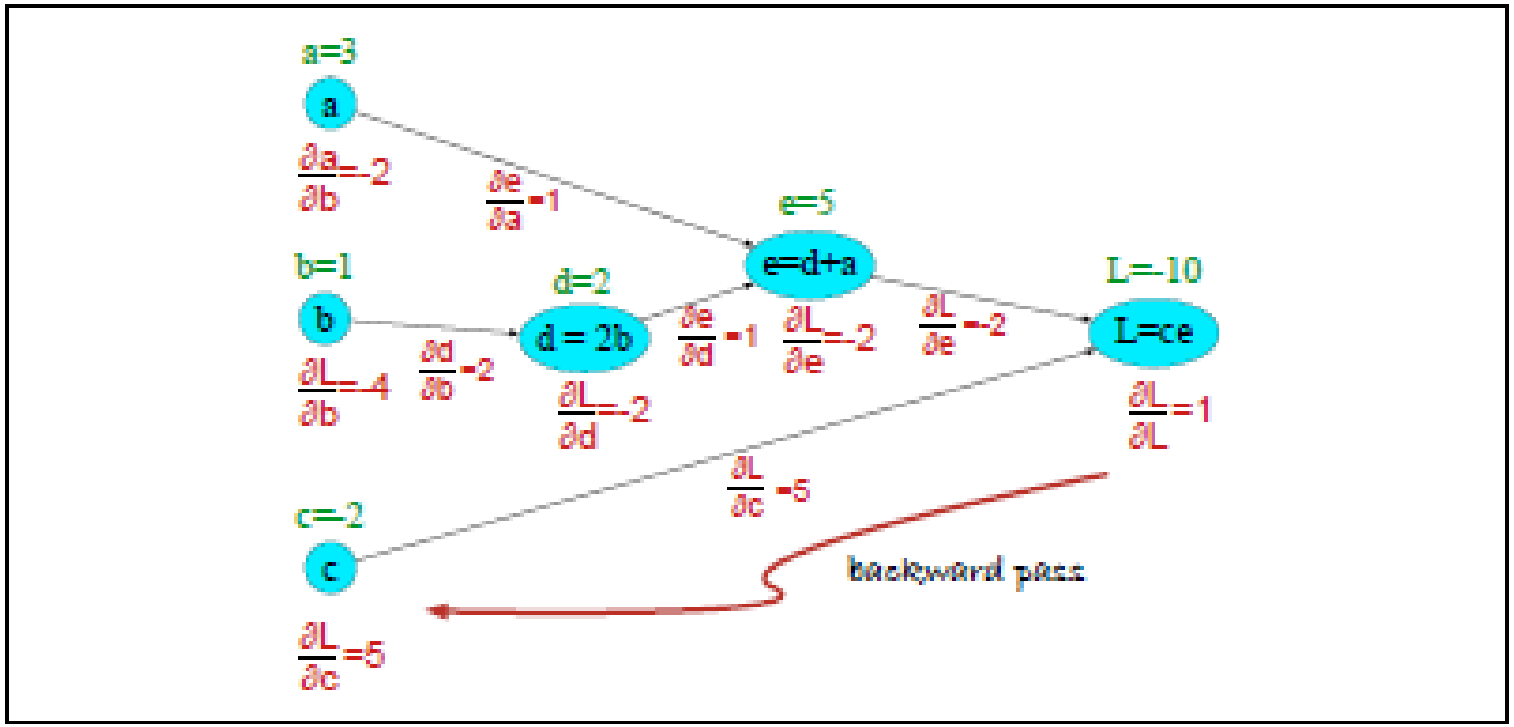


Figure 7.10 Computation graph for the function $L(a,b,c) = c(a+2b)$, showing the backward pass computation of $\frac{\partial L}{\partial a}$, $\frac{\partial L}{\partial b}$, and $\frac{\partial L}{\partial c}$.

Backprop Algorithm

- For each (x, t) in training examples.
  1. Propagate the input forward through the network. Compute the output of every unit u in the network.
  2. Propagate the errors backward through the network.
     - 2.1 For each network output unit k calculate its error term δ_k.
     - 2.2 For each hidden unit h, calculate its error term δ_h.
     - 2.3 Update each network weight w_{ij}.

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

where

$$\Delta w_{ij} = \eta \delta_i x_{ij}$$

Neural Language Models

- Advantages of Neural Language Models
  - Do not need smoothing
  - They can handle long histories
  - Can handle similar words in context

- Disadvantages
  - Slower to train

Feed forward Neural Language Model

- Model takes as input at time t a representation of some number of words and outputs a probability distribution over possible next words.

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

This is N-gram approximation.

For 4-gram

$$P(w_t = i|w_{t-1}, w_{t-2}, w_{t-3})$$

Two cases:

1. Word embeddings are pre-trained.

2. Learn word embedding while training the language model.

Pre-trained embedding

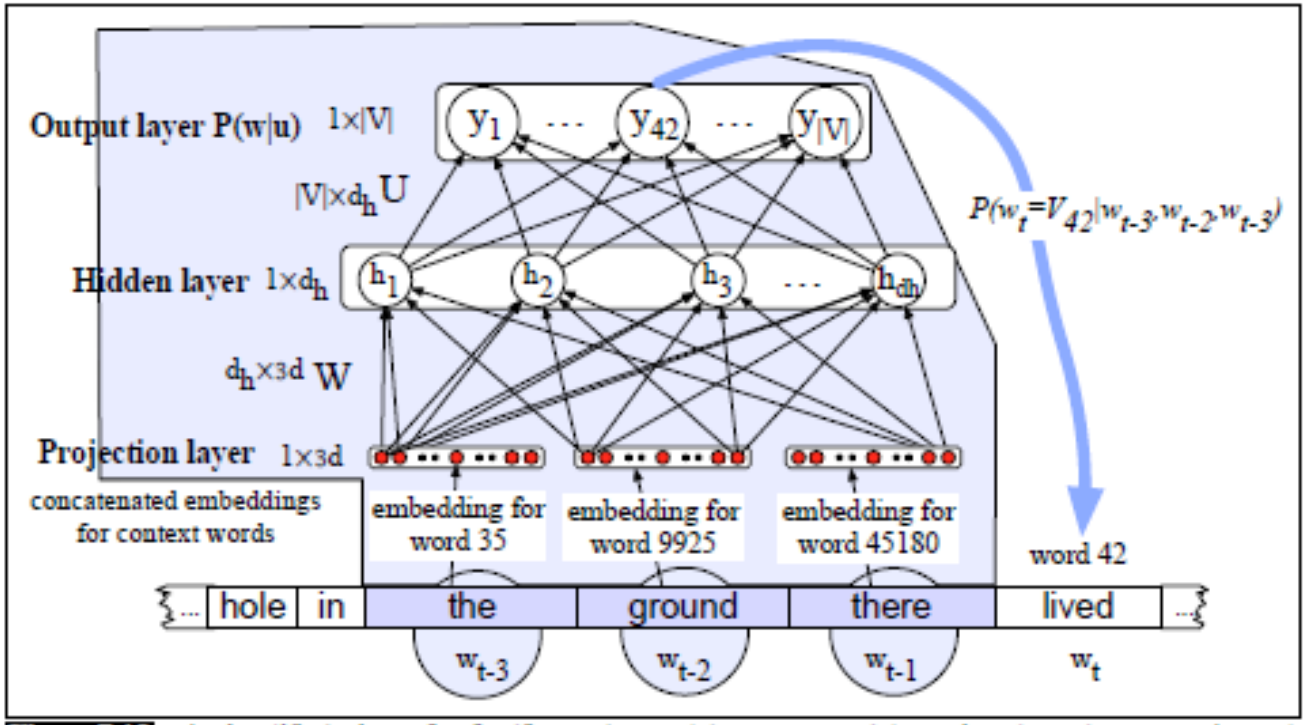


Figure 7.12 A simplified view of a feedforward neural language model moving through a text. At each timestep t the network takes the 3 context words, converts each to a d-dimensional embeddings, and concatenates the 3 embeddings together to get the $1 \times Nd$ unit input layer x for the network. These units are multiplied by a weight matrix W and bias vector b and then an activation function to produce a hidden layer h, which is then multiplied by another weight matrix U. (For graphic simplicity we don't show b in this and future pictures). Finally, a softmax output layer predicts at each node i the probability that the next word w_t will be vocabulary word V_i. (This picture is simplified because it assumes we just look up in an embedding dictionary E the d-dimensional embedding vector for each word, precomputed by an algorithm like word2vec.)

Embeddings to be learned

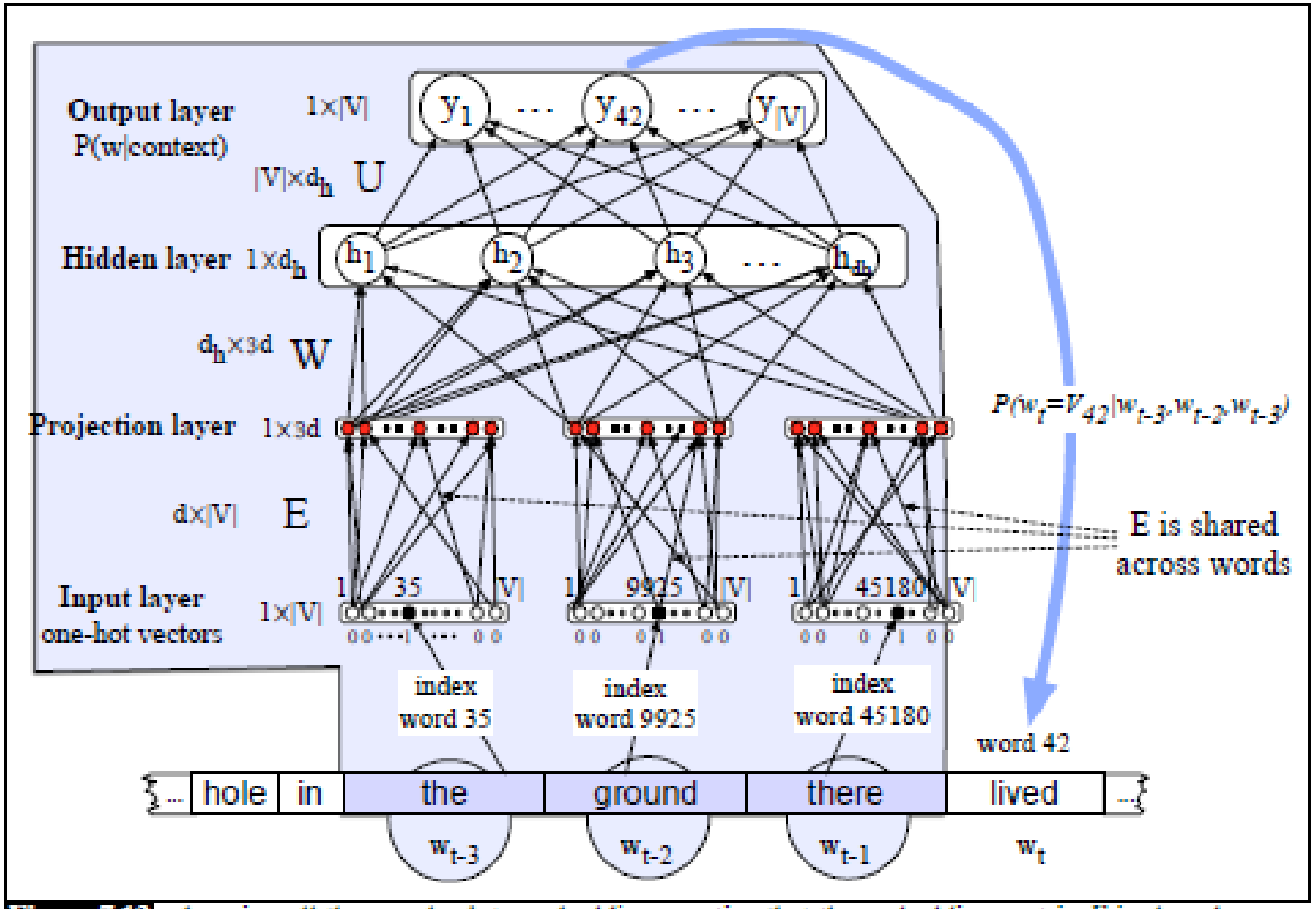


Figure 7.13 learning all the way back to embeddings. notice that the embedding matrix E is shared among the 3 context words.

Embeddings to be learned

- Forward pass
  1. Select three embeddings from E.
  2. Multiply by W.
  3. Multiply by U.
  4. Apply softmax

  Output later estimates

$$P(w_t = i | w_{t-1}, w_{t-2}, w_{t-3})$$

In summary

$$e = (Ex_1, Ex_2, \ldots, Ex)$$
$$h = \sigma(We + b)$$
$$z = Uh$$
$$y = softmax(z)$$

Embeddings to be learned

- Parameters

$$\theta = E, W, U, b$$

- Perform gradient descent using error back propagation on the computation of gradient descent.
- Take as input a very long text, concatenating all the sentences. Start with random weights, and then iterate.
- For each word w_t, the cross-entropy is

$$L = -\log p(w_t | w_{t-1}, \dots, w_{t-n+1})$$

$$\theta^{t+1} = \theta^t - \eta \frac{\partial -\log p(w_t | w_{t-1}, \dots, w_{t-n+1})}{\partial \theta}$$

Training Software Packages

- Several software packages implement the backpropagation algorithm.
  - Theano 2010
  - Tensor Flow 2015
  - Chainer 2015
  - DyNet 2017