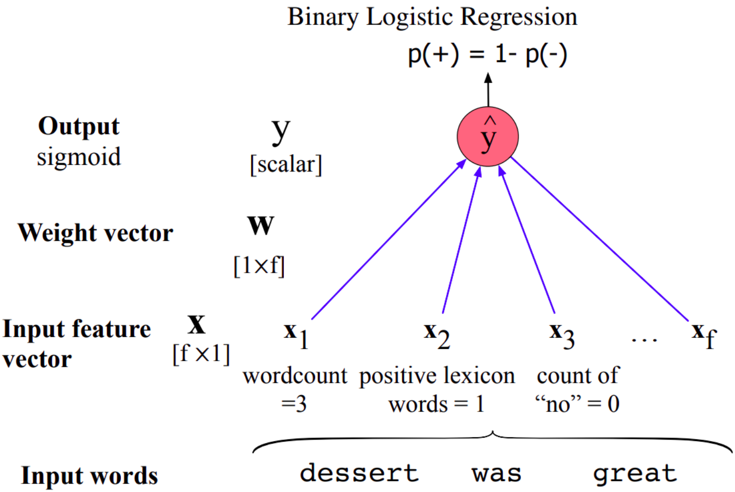
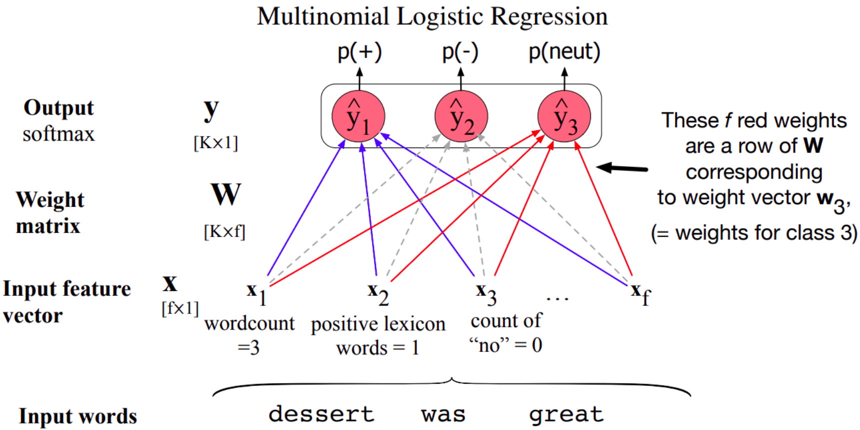
Feedforward neural networks

Gustave Cortal

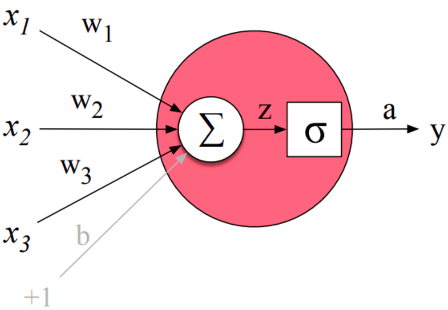
Binary logistic regression



Multinomial logistic regression



Neural unit



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

Activation functions (1)

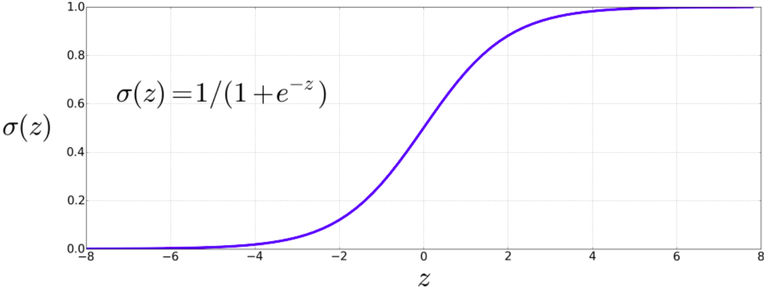


Figure: Sigmoid function.

Activation functions (2)

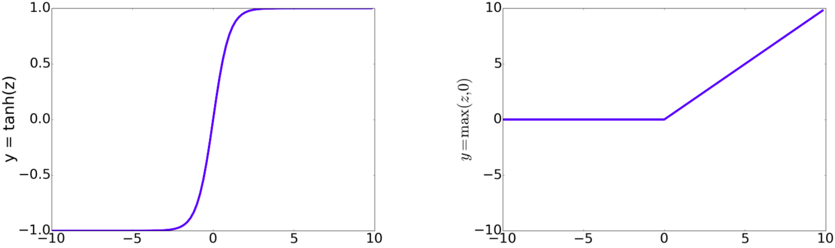
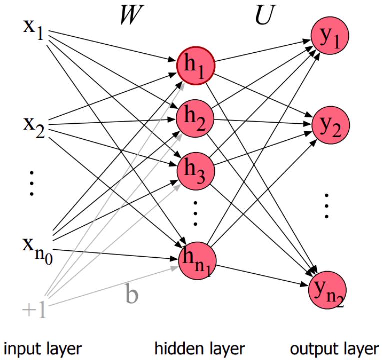


Figure: Tanh and ReLU functions.

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

$$y = \text{ReLU}(z) = \max(z, 0)$$

Feedforward network (1)



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

Feedforward network (2)

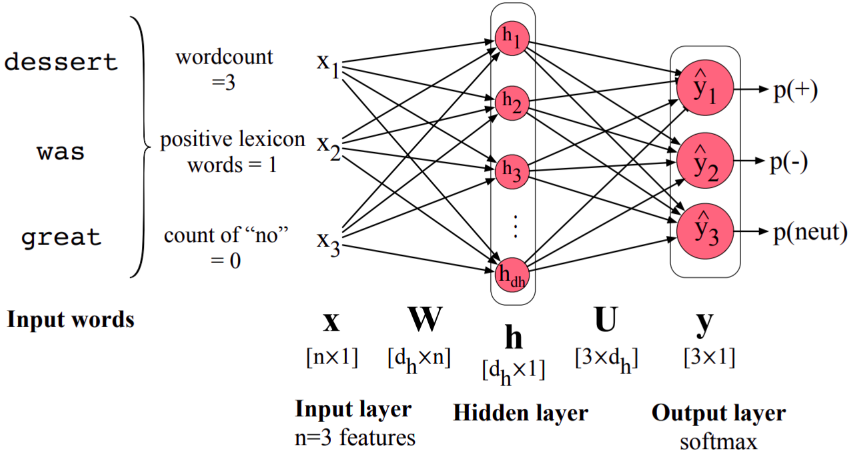


Figure: Feedforward network sentiment analysis using traditional hand-built features.

Feedforward network (3)

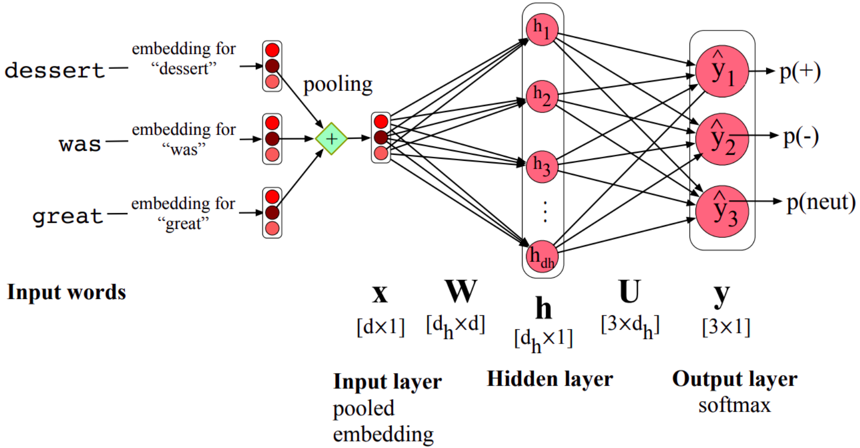


Figure: Feedforward network sentiment analysis using a pooled embedding.

Without activation functions, a multi-layer NN is equivalent to a single-layer NN

Consider the first two layers of a neural network with purely linear transformations:

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

The operations performed by the network can be combined and simplified as follows:

$$\begin{aligned} z^{[2]} &= W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]} \\ &= W^{[2]}W^{[1]}x + W^{[2]}b^{[1]} + b^{[2]} \\ &= W_0x + b_0 \end{aligned}$$

Loss function

The loss function for a single example x in the context of a multi-class classification problem, with K output classes, is defined as the cross-entropy loss L_{CE}:

$$L_{CE}(\hat{y}, y) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$$

The loss L_{CE} for a prediction $\hat{y}$ and true label y, focusing on the correct class c, is represented as:

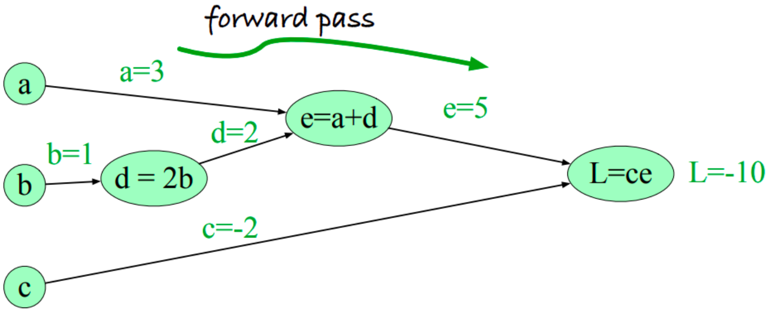
$$\begin{aligned} L_{CE}(\hat{y}, y) &= -\log(\hat{y}_c) \\ &= -\log\left(\frac{\exp(z_c)}{\sum_{j=1}^{K} \exp(z_j)}\right) \end{aligned}$$

Computing the gradients

For deep networks, computing the gradients for each weight is difficult, since we are computing the derivative with respect to weight parameters that appear all the way back in the very early layers of the network

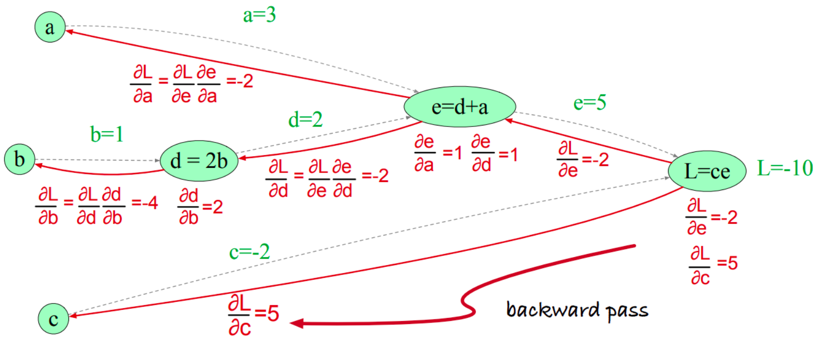
The solution to computing this gradient is an algorithm called **error backpropagation**.

Forward pass



$$L(a, b, c) = c(a + 2b)$$

Backward pass



$$L(a, b, c) = c(a + 2b)$$

Backpropagation calculus, 3Blue1Brown's video

https://www.youtube.com/watch?v=Ilg3gGewQ5U

Feedforward NN for language modeling

A feedforward neural language model takes as input at time t a representation of some number of previous words (w_{t-1}, w_{t-2}, etc.) and outputs a probability distribution over possible next words

Like the n-gram, it approximates the probability of a word given the entire prior context by approximating based on the $n-1$ previous words:

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

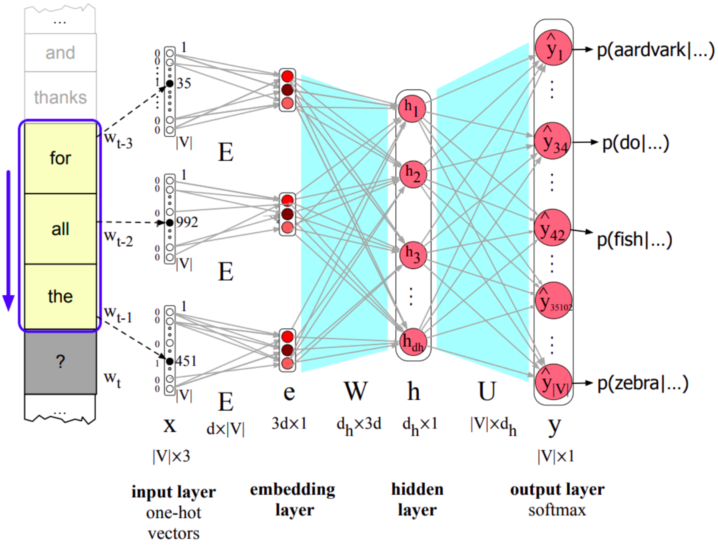
Unlike n-gram models, neural language models can handle much longer histories, generalize better over contexts of similar words, and are more accurate at word prediction.

Feedforward NN for language modeling

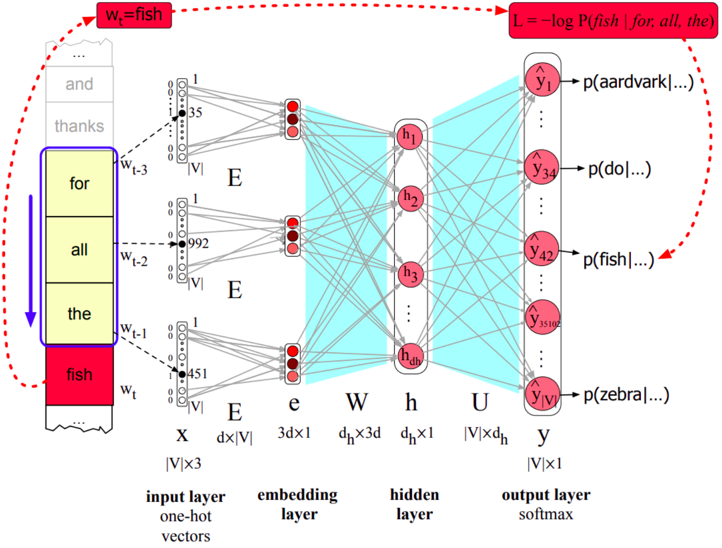
The equations for a neural language model with a window size of 3, given one-hot input vectors for each input context word, are:

$$
\begin{aligned}
e &= [Ex_{t-3}; Ex_{t-2}; Ex_{t-1}] \\
h &= \sigma(We + b) \\
z &= Uh \\
\hat{y} &= \text{softmax}(z)
\end{aligned}
$$

Feedforward NN for language modeling



Feedforward NN for language modeling



How to improve the training?

Find good **hyperparameters**: batch size, learning rate, activation functions, number of hidden layers, number of neural units

Apply **regularization** methods: normalize input values, add dropout, add weight decay

Other techniques include label smoothing, cutting gradients norm, augmenting data, using good weights initialization, gradient descent with momentum (Adam optimizer), etc.

Exercices

Using PyTorch, you need to:

- Implement the logistic regression
- Implement a multi-layer feedforward network for text classification based on word2vec features
- Play with hyperparameters (see slide "How to improve the training?")
- Implement a multi-layer feedforward network for language modeling (optional)
- Study the skip-gram model (optional, see notebook from the last course)

The goal is to have a workable PyTorch training loop for your project!