DeepHack.CISS

Neural Networks basics

Valentin Malykh

Based on Stanford CS224d

Demystifying neural networks

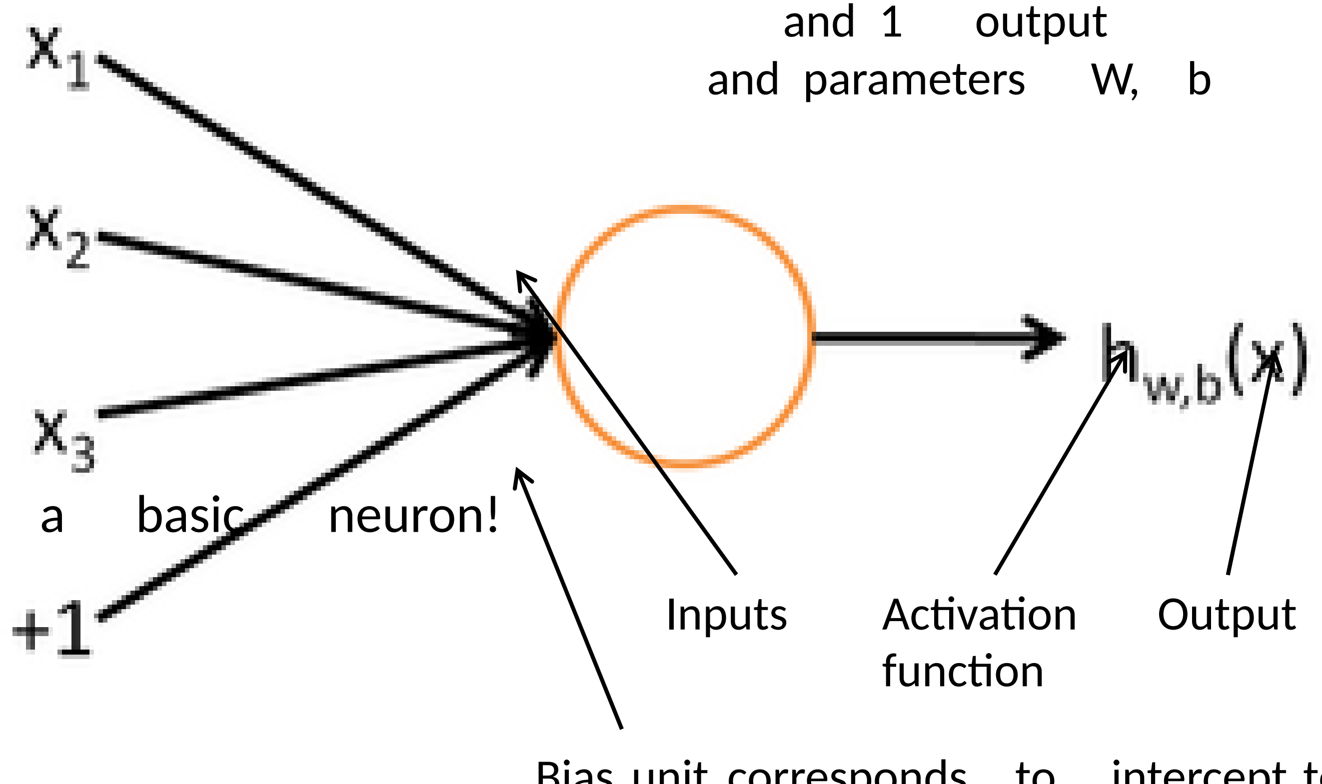
Neural networks come with their own terminological baggage

But if youunde softmax models

Then **you al** operation of a basic neuron!

A single neuron

A computational unit with n (3) inputs and 1 output and parameters W, b



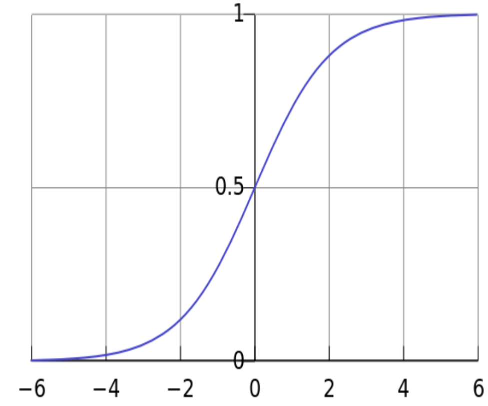
A neuron is essentially a binary logistic regression

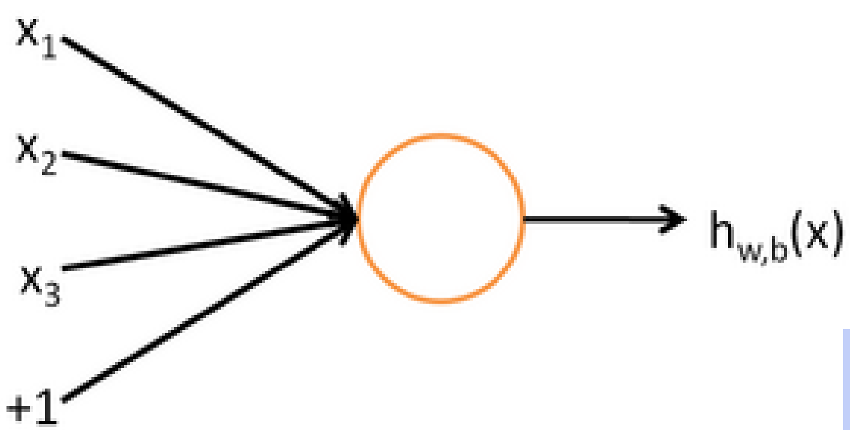
F = nonlinear activation fct. (e.g. sigmoid), w = weights, b = bias, h = hidden, x = inputs

$$h_{w,b}(x) = f(w^{\mathsf{T}}x + b)$$

b: We can have an “always on” Feature, which gives a class prior, or separate it out, as a bias term

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



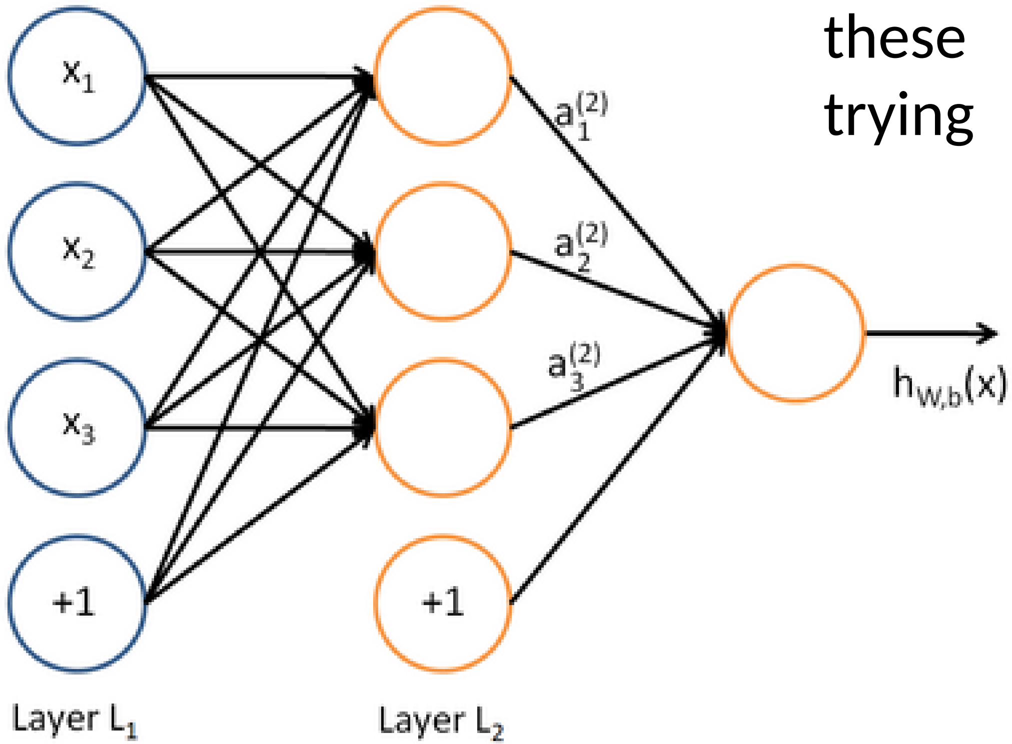


W, b are the parameters of this neuron i.e., this logistic regression model

A neural network = running several logistic regressions at the same time

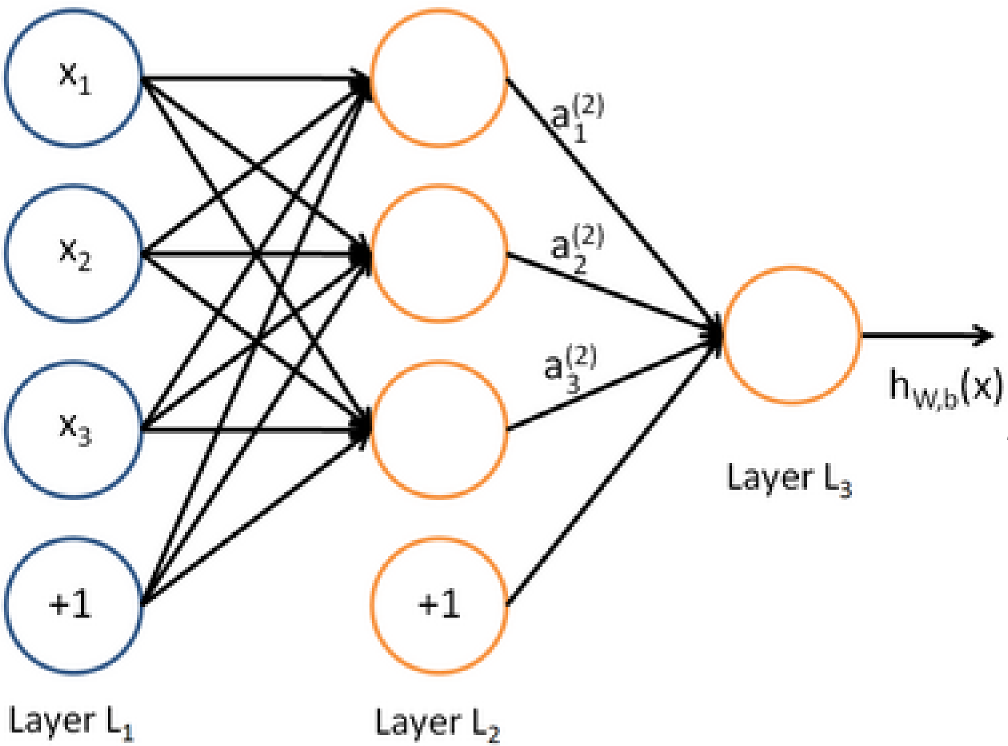
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs …

But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!



A neural network = running several logistic regressions at the same time

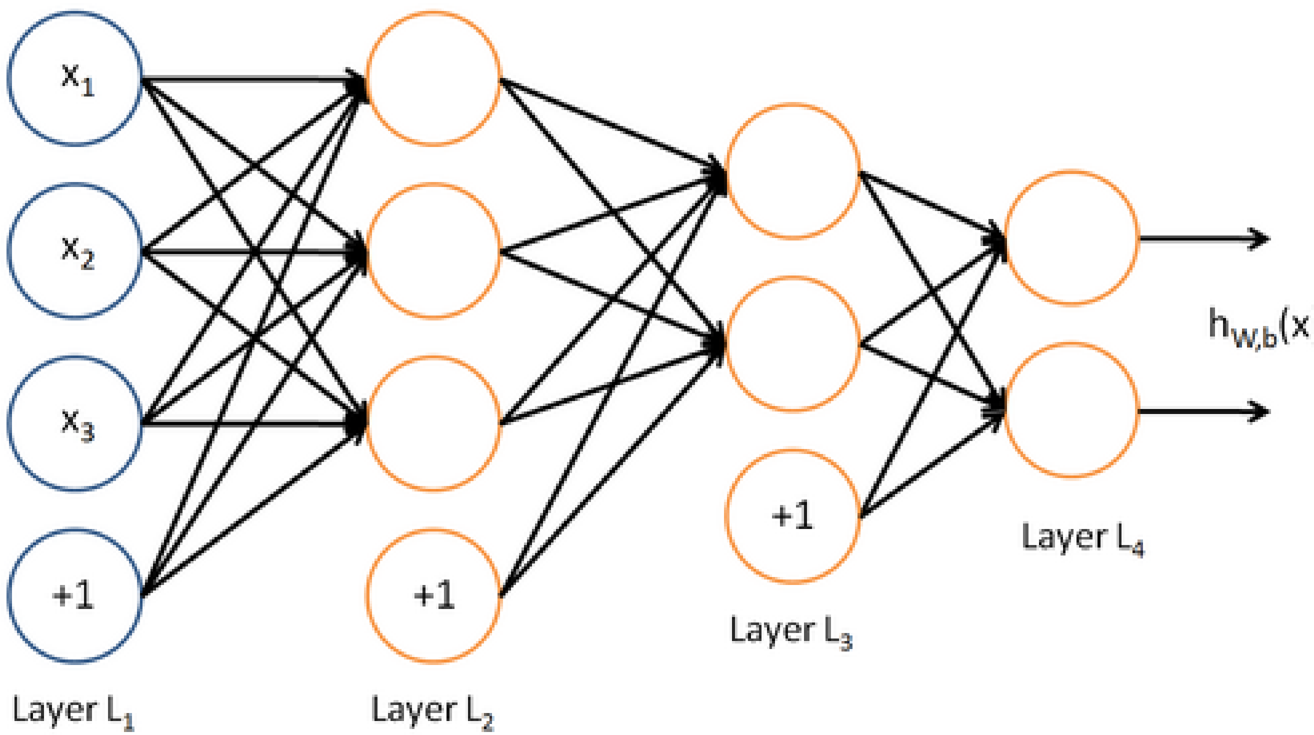
… which we can feed into another logistic regression function



It is the loss function that will direct what the intermediate hidden variables should be, so as to do a good job at predicting the targets for the next layer, etc.

A neural network = running several logistic regressions at the same time

Before we know it, we have a multilayer neural network….



Matrix notation for a layer

We have

$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$

$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

etc.

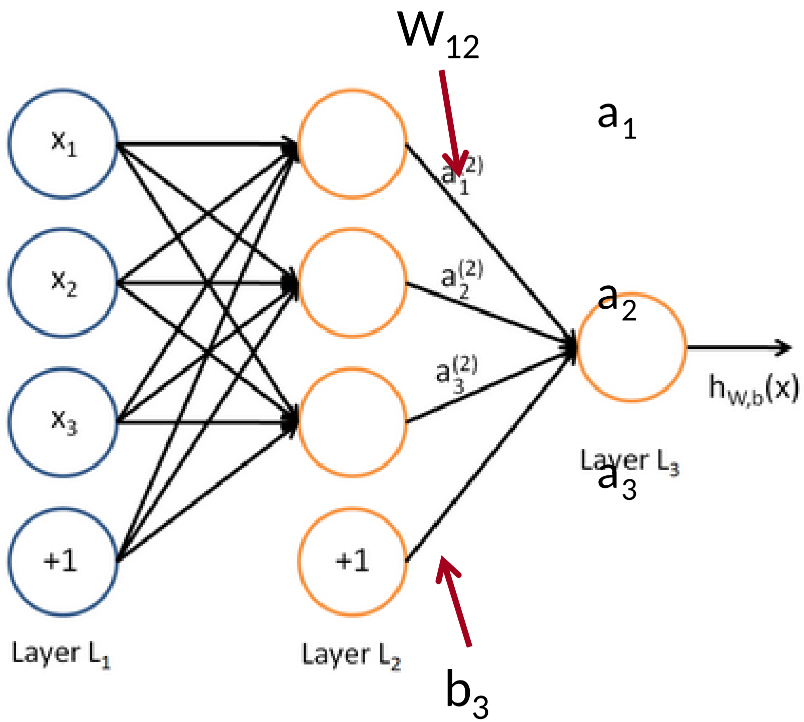
In matrix notation

$$z = Wx + b$$

$$a = f(z)$$

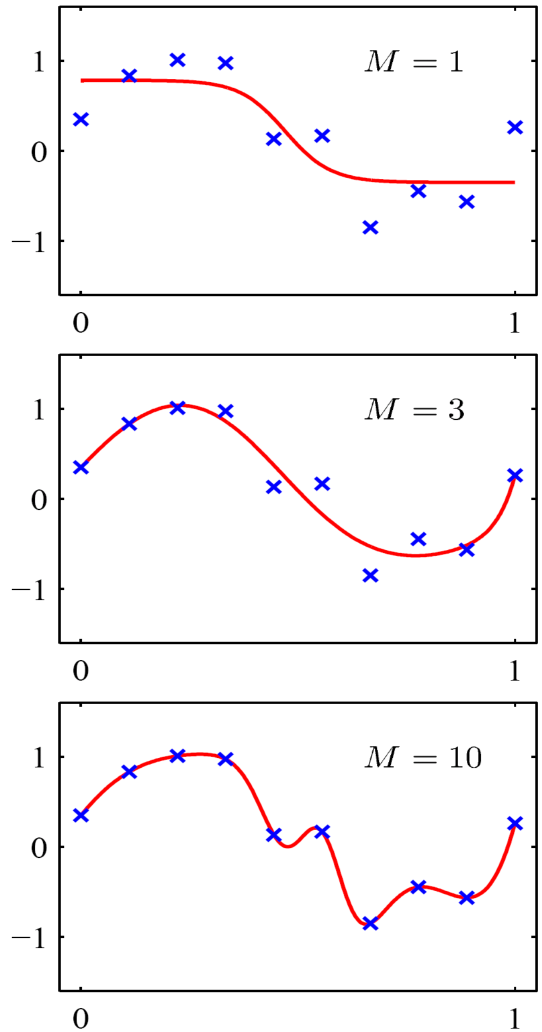
where f is applied element-wise:

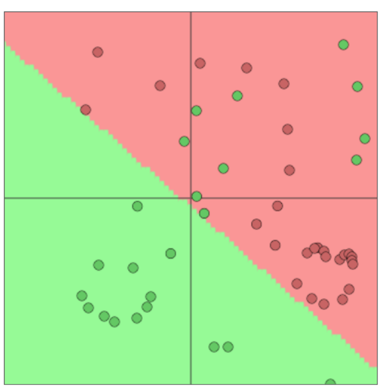
$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

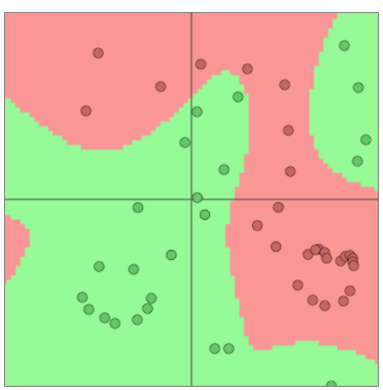


Non-linearities (aka "f"): Why they're needed

- Example: function approximation, e.g., regression or classification
  - Without non-linearities, deep neural networ can't do anything more than a linear transform
  - Extra layers could just be compiled do a single linear transform: $W_1 W_2 x = Wx$
  - With more layers, they can approximat complex functions!







Binary classification with unnormalized scores

- Revisiting our previous example:
  $X_{window} = [\ x_{museums}\quad x_{in}\quad x_{Paris}\quad x_{are}\quad x_{amazing}\ ]$
- Assume we want to classify whether the center word is a Location (Named Entity Recognition)

- Similar to word2vec, we will go over all positions in a Corpus. But this time, it will be supervised and only some positions should get a high score.
- The positions that have an actual NER location in their center are called "true" positions.

Binary classification for NER

- Example: Not all museums in Paris are amazing.
- Here: one true window, the one with Paris in its center and all other windows are “corrupt”in terms of not having a named entity location in their center.
- “Corrupt“ windows are easy to find and there are Many: Any window that isn’t specifically labeled as NER location in our corpus

A Single Layer Neural Network

- A single layer is a combination of a linear layer and a nonlinearity:

$$\begin{aligned} z &= Wx + b \\ a &= f(z) \end{aligned}$$

- The neural activations a can then be used to c
some output.

  - For instance, a probability via softmax:

$$p(y|x) = \text{softmax}(Wa)$$

  - Or an unnormalized score (even simpler):

$$score(x) = U^T a \in \mathbb{R}$$

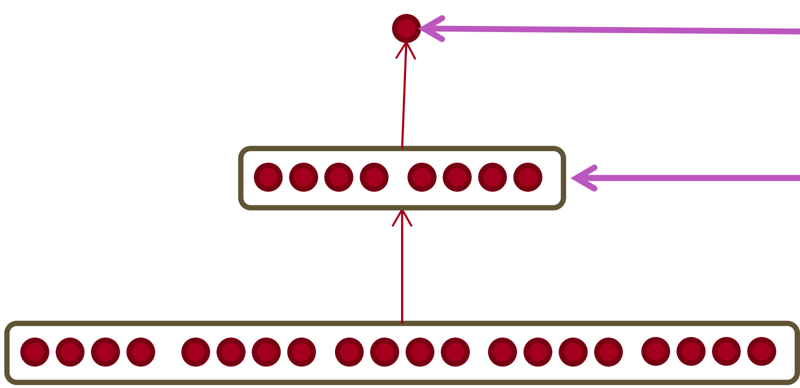
Summary: Feed-forwardComputation

We compute a window's score with a 3-layer neural

- s = score("museums in Paris are amazing")

$$s = U^T f(Wx + b)$$

$$x \in \mathbb{R}^{20\times 1}, W \in \mathbb{R}^{8\times 20}, U \in \mathbb{R}^{8\times 1}$$



$$s = U^T a$$

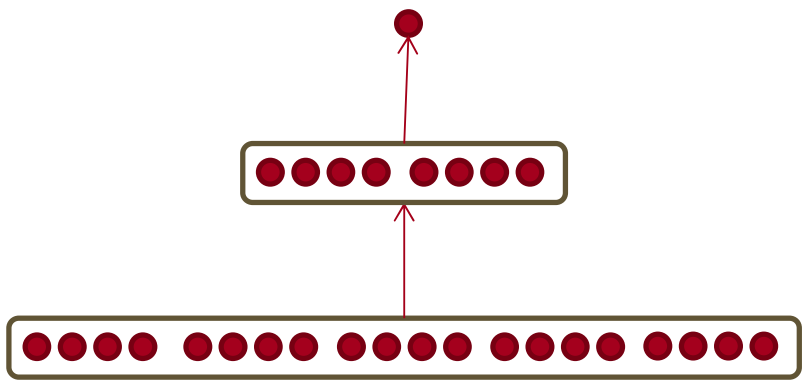
$$a = f(z)$$

$$z = Wx + b$$

x_{window} = [$x_{museums}$ x_{in} x_{Paris} x_{are} $x_{amazing}$]

Main intuition for extra layer

The layer learns non-linear interactions between the input word vectors.



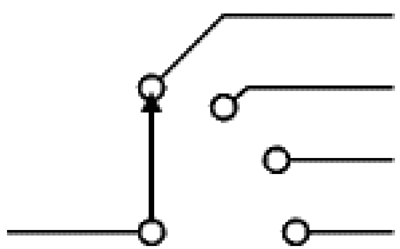
$x_{window} = [\, x_{museums} \quad x_{in} \quad x_{Paris} \quad x_{are} \quad x_{amazing} \,]$

Example: only if "museums" is first vector should it matter that "in" is in the second position

The max-margin loss

- Idea for training objective: Make true window's score larger and corrupt window's score lower (until they're good enough): minimize
- s = score(museums in Paris are amazing)
- s_c = score(Not all museums in Paris)

$$J = \max(0, 1 - s + s_c)$$



- This is not differentiable but it is continuous --> we can use SGD.

Each option is continuous

Max-margin loss

- Objectivefor a single window:

$$J = \max(0, 1 - s + s_c)$$

- Each window with an NER location at its center should have a score +1 higher than any window without a location at its center

- For full objective function: Sample several corrupt windows per true one. Sum over all training windows.
- Similar to negative sampling in word2vec

Deriving gradients for backprop

$$\boxed{J = \max(0, 1 - s + s_c)}$$

$$s = U^T f(Wx + b)$$
$$s_c = U^T f(Wx_c + b)$$

Assuming cost J is > 0,

compute the derivatives of s and s involved variables: U, W, b, x

$$\frac{\partial s}{\partial U} = \frac{\partial}{\partial U} U^T a \quad \text{à} \quad \frac{\partial s}{\partial U} = a$$

Deriving gradients for backprop

- For this function:

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W} U^T a = \frac{\partial}{\partial W} U^T f(z) = \frac{\partial}{\partial W} U^T f(Wx + b)$$

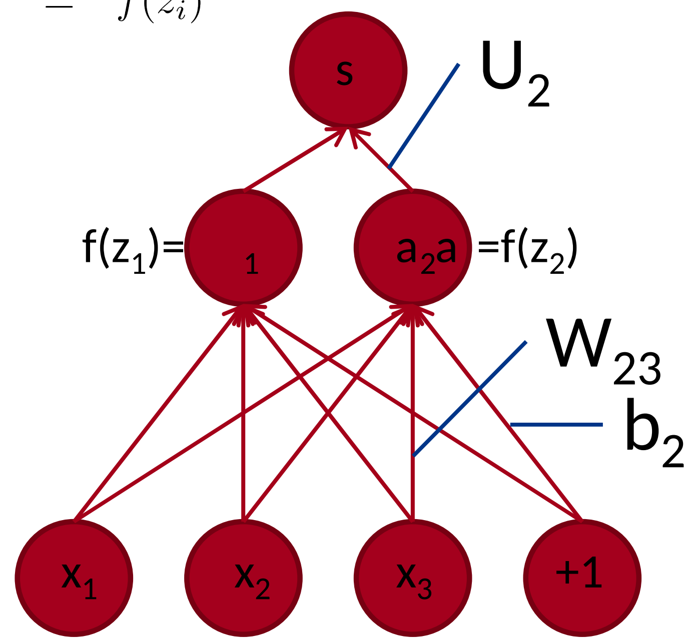
- Let's consider the derivative of a single weight W_{ij}

- W_{ij} only appears inside a_i

- For example: W_{23} is only used to compute a_2 not a_1

$$z_i = W_{i.}x + b_i = \sum_{j=1}^{3} W_{ij} x_j + b_i$$

$$a_i = f(z_i)$$



Deriving gradients for backprop

Derivative of single weight W_{ij}:

$$\frac{\partial}{\partial W_{ij}} U^T a \quad \rightarrow \quad \frac{\partial}{\partial W_{ij}} U_i a_i$$

Ignore constants in which W_{ij} doesn't appear

$$U_i \frac{\partial}{\partial W_{ij}} a_i = U_i \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}}$$

Pull out U_i since it's constant, apply chain rule

$$= U_i \frac{\partial f(z_i)}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}}$$

Apply definition of a

$$= U_i f'(z_i) \frac{\partial z_i}{\partial W_{ij}}$$

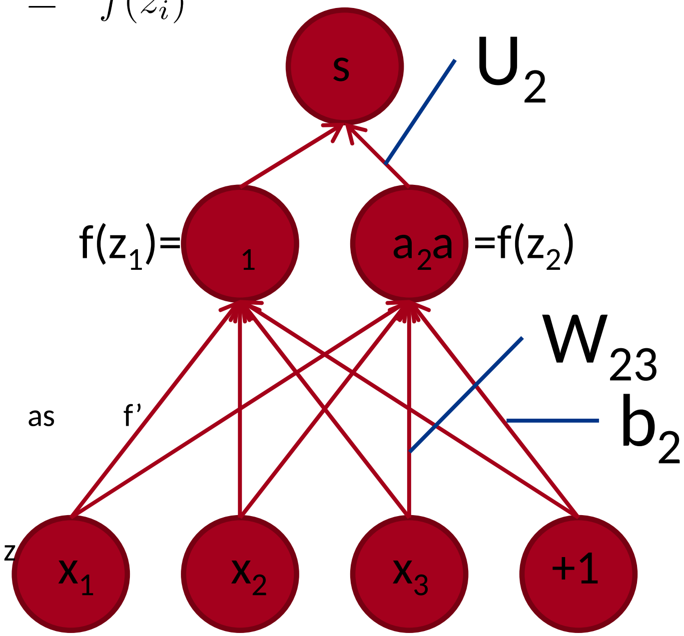
Just define derivative of f as f'

$$= U_i f'(z_i) \frac{\partial W_{i\cdot} x + b_i}{\partial W_{ij}}$$

Plug in definition of z

$$z_i = W_{i\cdot}x + b_i = \sum_{j=1}^{3} W_{ij} x_j + b_i$$

$$a_i = f(z_i)$$



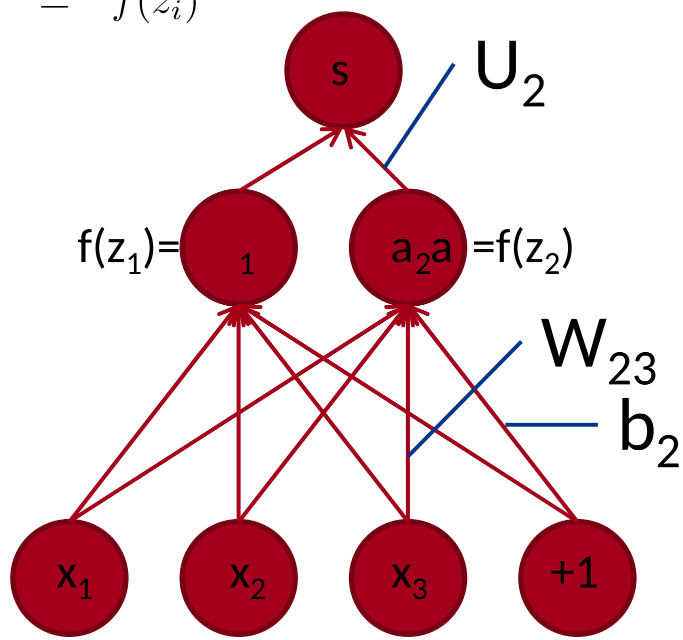
Deriving gradients for backprop

Derivative of single weight W_{ij} continued:

$$
\begin{aligned}
U_i \frac{\partial}{\partial W_{ij}} a_i &= U_i f'(z_i) \frac{\partial W_{i\cdot} x + b_i}{\partial W_{ij}} \\
&= U_i f'(z_i) \frac{\partial}{\partial W_{ij}} \sum_k W_{ik} x_k \\
&= \underbrace{U_i f'(z_i)}\, x_j \\
&= \underbrace{\delta_i}_{\text{Local error signal}} \quad \underbrace{x_j}_{\text{Local input signal}}
\end{aligned}
$$

where $f'(z) = f(z)(1 - f(z))$

$$
\begin{aligned}
z_i &= W_{i\cdot} x + b_i = \sum_{j=1}^{3} W_{ij} x_j + b_i \\
a_i &= f(z_i)
\end{aligned}
$$



Deriving gradients for backprop

- So far, derivative of single W_{ij} only, but we want gradient for full W.

$$\frac{\partial s}{\partial W_{ij}} = \underbrace{U_i f'(z_i)}\, x_j = \delta_i \quad x_j$$

- We want all combinations of i = 1, 2 and j = 1, 2,
- Solution: Outer product: $\frac{\partial s}{\partial W} = \delta x^T$

where $\delta \in \mathbb{R}^{2\times 1}$ is the "responsibility" or error signal coming from each activation a

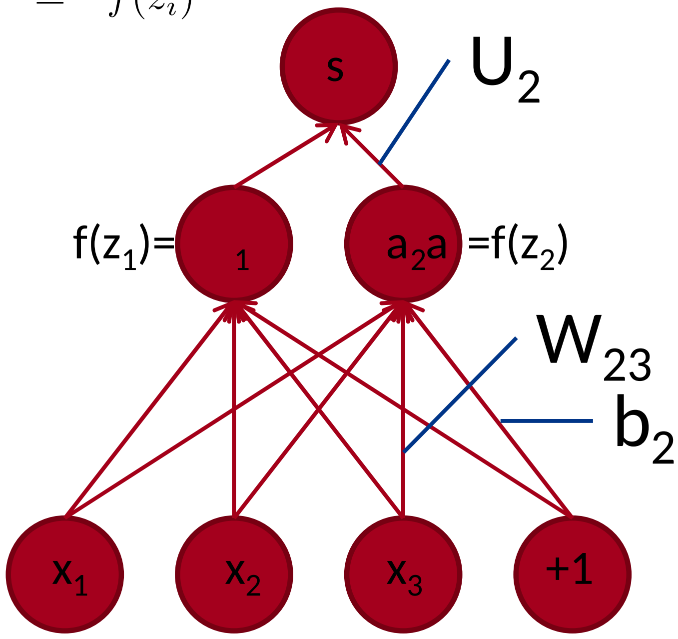
Deriving gradients for backprop

- How to derive gradients for biases b?

$$\frac{\partial s}{\partial\, b} = a$$

$$\begin{aligned} & U_i \frac{\partial}{\partial b_i} a_i \\ = \quad & U_i f'(z_i) \frac{\partial W_{i\cdot} x + b_i}{\partial b_i} \\ = \quad & \delta_i \end{aligned}$$

$$\begin{aligned} z_i &= W_{i\cdot} x + b_i &= \sum_{j=1}^{3} W_{ij} x_j + b_i \\ a_i &= f(z_i) \end{aligned}$$



Training with Backpropagation

That's almost backpropagation

It's taking derivatives and using the chain rule

Remaining trick: we can **re-use** derivatives computed for Higher layers in computing derivatives for lower layers!

Example: last derivatives of model, the word vectors in x

Trainingwith Backpropagation

- Take derivative of score with Respect to single element of word vector
- Now, we cannot justtake into consideration one a because each is x connected to all the neurons above and hence x influences the overall score through all of these, hence:

$$
\begin{aligned}
\frac{\partial s}{\partial x_j} &= \sum_{i=1}^{2} \frac{\partial s}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{2} \frac{\partial U^T a}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{2} U_i \frac{\partial f(W_{i\cdot}x + b)}{\partial x_j} \\
&= \sum_{i=1}^{2} \underbrace{U_i f'(W_{i\cdot}x + b)} \frac{\partial W_{i\cdot}x}{\partial x_j} \\
&= \sum_{i=1}^{2} \delta_i W_{ij} \\
&= W_{\cdot j}^T \underbrace{\delta'_{\cdot j}}_{\text{Re-used part of previous derivative}}
\end{aligned}
$$

Training with Backpropagation

- With $\frac{\partial s}{\partial x_j} = W_{\cdot j}^T \delta$

$$\frac{\partial s}{\partial x} = W^T \delta$$

- Observations: The error message that arrives at a hidden layer has the same dimensionality as that hidden layer

Putting all gradients together:

- Remember: Full objective function for each window was:

$$\boxed{J = \max(0, 1 - s + s_c)}$$

$$s = U^T f(Wx + b)$$
$$s_c = U^T f(Wx_c + b)$$

- For example: gradient for just U:

$$\frac{\partial J}{\partial U} = 1\{1 - s + s_c > 0\} \left(-f(Wx + b) + f(Wx_c + b)\right)$$
$$\frac{\partial J}{\partial U} = 1\{1 - s + s_c > 0\} \left(-a + a_c\right)$$

Summary

Congrats! Super useful basic components and real model

- Word vectortraining
- Windows
- Softmax and cross entropy error
- Scores and max-margin loss
- Neural network