Natural Language Processing with Deep Learning

CS224N/Ling284

Christopher Manning

Lecture 3: Word Window Classification, Neural Networks, and Matrix Calculus

1. Course plan: coming up

Week 2: We learn neural net fundamentals

- We concentrate on understanding (deep, multi-layer) neural networks and how they can be trained (learned from data) using backpropagation (the judicious application of matrix calculus)
- We'll look at an NLP classifier that adds context by taking in windows around a word and classifies the center word (not just representing it across all windows)!

Week 3: We learn some natural language processing

- We learn about putting syntactic structure (dependency parses) over sentence (this is HW3!)
- We develop the notion of the probability of a sentence (a probabilistic language model) and why it is really useful

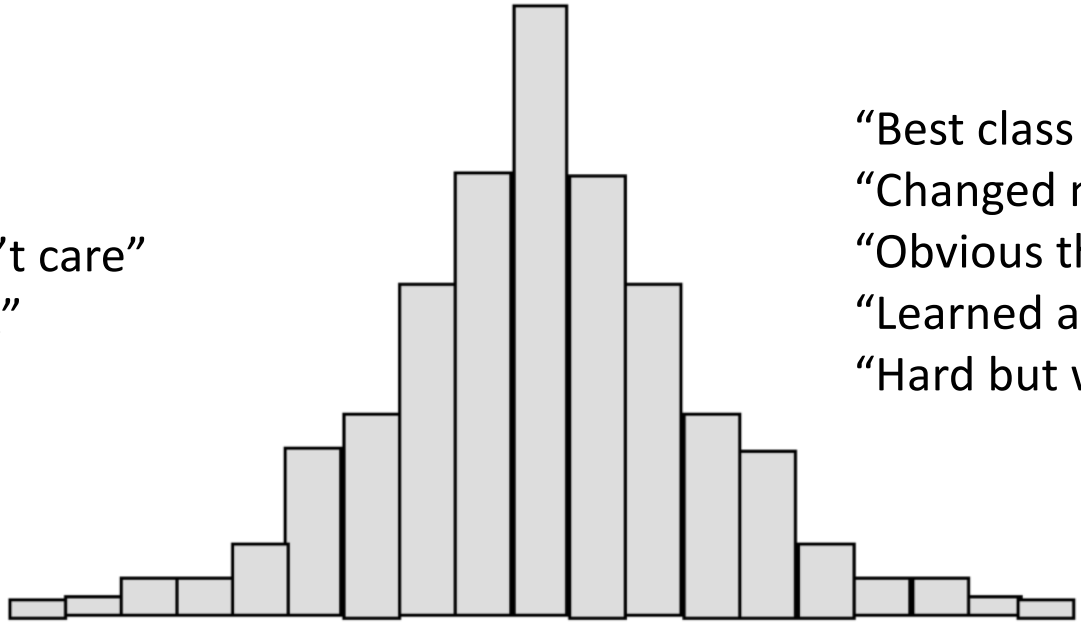
Homeworks

- HW1 was due … a couple of minutes ago!
  - We hope you've submitted it already!
  - Try not to burn your late days on this easy first assignment!

- HW2 is now out
  - Written part: gradient derivations for word2vec (OMG … calculus)
  - Programming part: word2vec implementation in NumPy
  - (Not an IPython notebook)
  - You should start looking at it early! Today's lecture will be helpful and Thursday will contain some more info.
  - Website has lecture notes to give more detail

A note on your experience 😀

"Terrible class"
"Don't take it"
"Instructors don't care"
"Too much work"



"Best class at Stanford"
"Changed my life"
"Obvious that instructors care"
"Learned a ton"
"Hard but worth it"

- This is a hard, advanced, graduate level class
- I and all the TAs really care about your success in this class
- Give Feedback. Work to address holes in your knowledge
- **Come to office hours/help sessions**

Office Hours / Help sessions

- **Come to office hours/help sessions!**
  - Come to discuss final project ideas as well as the homeworks
  - Try to come early, often and off-cycle
- **Help sessions:** daily, at various times, see calendar
  - Coming up: Wed 12-2:30pm, Thu 6:30–9:00pm
  - Gates Basement B21 (and B30) – bring your student ID
    - No ID? Try Piazza or tailgating—hoping to get a phone in room
  - Attending in person: Just show up! Our friendly course staff will be on hand to assist you
  - SCPD/remote access: Use queuestatus
- **Chris's office hours:**
  - Mon 1–3pm. Come along next Monday?

Lecture Plan

Lecture 3: Word Window Classification, Neural Nets, and Calculus

1. Course information update (5 mins)
2. Classification review/introduction (10 mins)
3. Neural networks introduction (15 mins)
4. Named Entity Recognition (5 mins)
5. Binary true vs. corrupted word window classification (15 mins)
6. Matrix calculus introduction (20 mins)

- This will be a tough week for some! →
  - Read tutorial materials given in syllabus
  - Visit office hours

2. Classification setup and notation

- Generally we have a training dataset consisting of samples

$$\{x_i, y_i\}_{i=1}^{N}$$

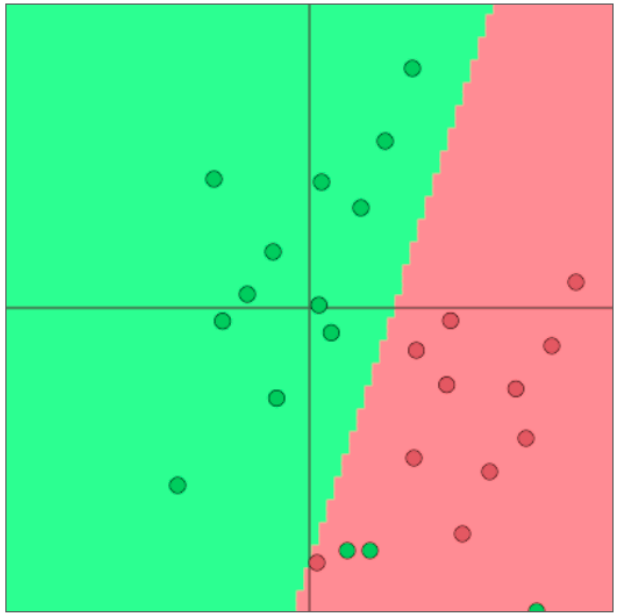
- x_i are inputs, e.g. words (indices or vectors!), sentences, documents, etc.
  - Dimension *d*

- y_i are labels (one of *C* classes) we try to predict, for example:
  - classes: sentiment, named entities, buy/sell decision
  - other words
  - later: multi-word sequences

Classification intuition

- Training data: $\{x_i, y_i\}_{i=1}^{N}$

- Simple illustration case:
  - Fixed 2D word vectors to classify
  - Using softmax/logistic regression
  - Linear decision boundary



Visualizations with ConvNetJS by Karpathy!
http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

- **Traditional ML/Stats approach:** assume x_i are fixed, train (i.e., set) softmax/logistic regression weights $W \in \mathbb{R}^{C \times d}$ to determine a decision boundary (hyperplane) as in the picture

- **Method**: For each *x*, predict:

$$p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C} \exp(W_c.x)}$$

Details of the softmax classifier

$$p(y|x) = \frac{\exp(W_{y\cdot}x)}{\sum_{c=1}^{C}\exp(W_{c\cdot}x)}$$

We can tease apart the prediction function into two steps:

1. Take the y^{th} row of W and multiply that row with x:

$$W_{y\cdot}x = \sum_{i=1}^{d} W_{yi}x_i = f_y$$

   Compute all f_c for $c = 1, \ldots, C$

2. Apply softmax function to get normalized probability:

$$p(y|x) = \frac{\exp(f_y)}{\sum_{c=1}^{C}\exp(f_c)} = \text{softmax}(f_y)$$

Training with softmax and cross-entropy loss

- For each training example (x,y), our objective is to maximize the probability of the correct class *y*

- Or we can minimize the negative log probability of that class:

$$-\log p(y|x) = -\log\left(\frac{\exp(f_y)}{\sum_{c=1}^{C}\exp(f_c)}\right)$$

Background: What is "cross entropy" loss/error?

- Concept of "cross entropy" is from information theory
- Let the true probability distribution be *p*
- Let our computed model probability be *q*
- The cross entropy is:

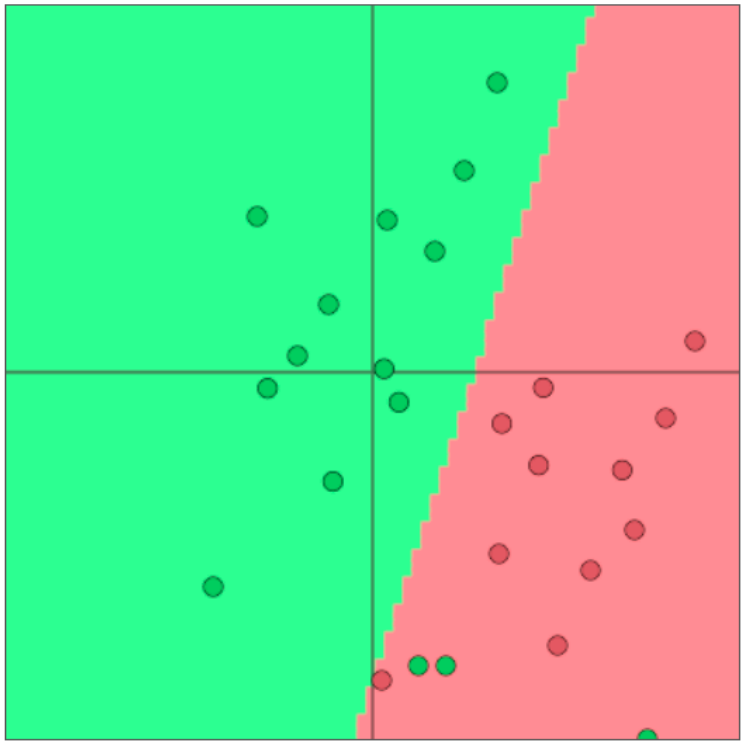
$$H(p,q) = -\sum_{c=1}^{C} p(c)\log q(c)$$

- Assuming a ground truth (or true or gold or target) probability distribution that is 1 at the right class and 0 everywhere else: *p* = [0,…,0,1,0,…0] then:
- **Because of one-hot *p*, the only term left is the negative log probability of the true class**

Classification over a full dataset

- Cross entropy loss function over full dataset $\{x_i, y_i\}_{i=1}^N$

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} -\log \left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}} \right)$$



- Instead of

$$f_y = f_y(x) = W_{y.}x = \sum_{j=1}^{d} W_{yj} x_j$$

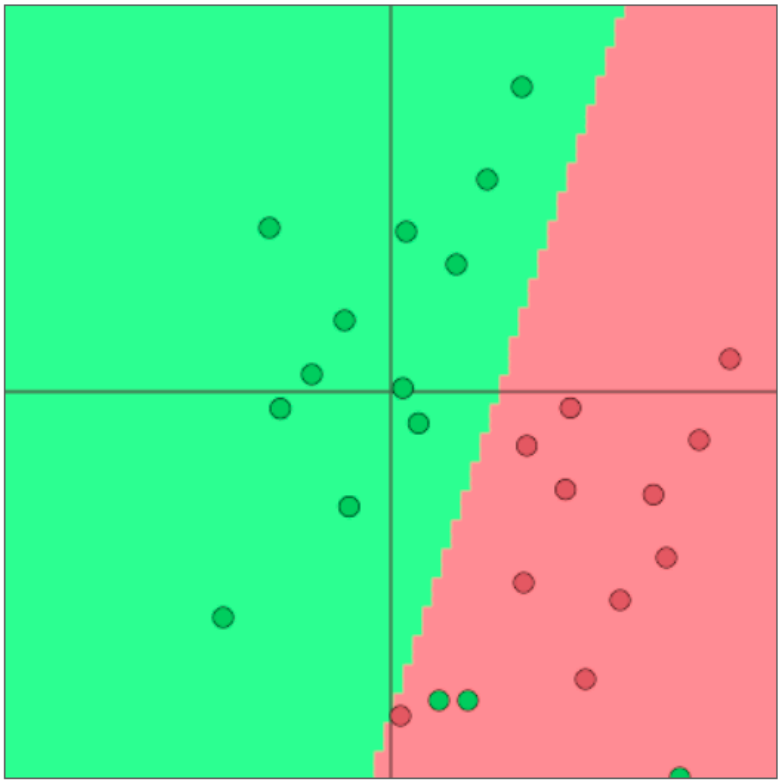
We will write *f* in matrix notation:

$$f = Wx$$

Traditional ML optimization

- For general machine learning θ usually only consists of columns of W:

$$\theta = \begin{bmatrix} W_{\cdot 1} \\ \vdots \\ W_{\cdot d} \end{bmatrix} = W(:) \in \mathbb{R}^{Cd}$$



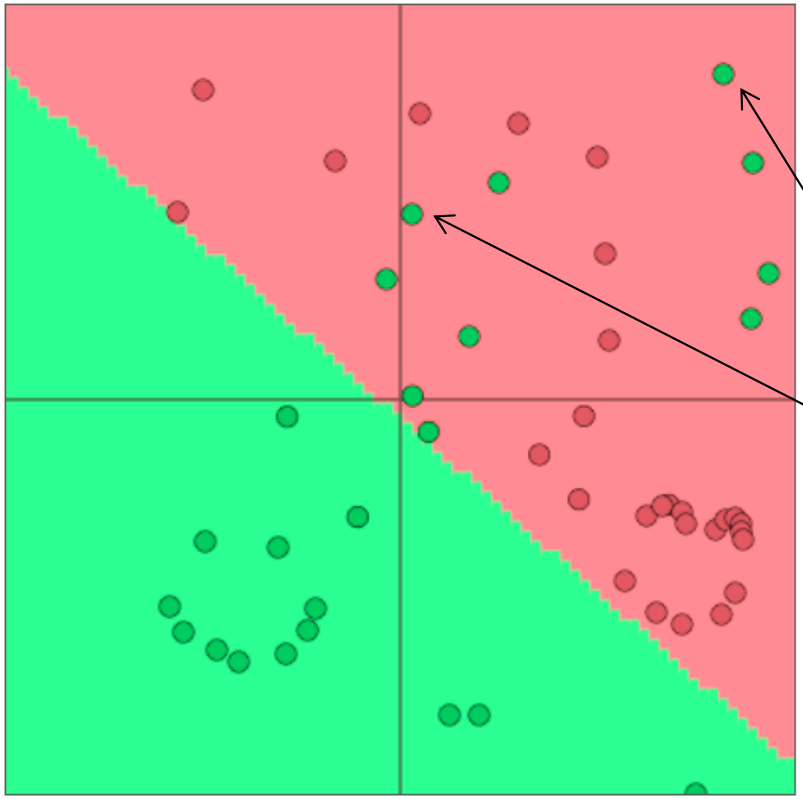
Visualizations with ConvNetJS by Karpathy

- So we only update the decision boundary via

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla_{W_{\cdot 1}} \\ \vdots \\ \nabla_{W_{\cdot d}} \end{bmatrix} \in \mathbb{R}^{Cd}$$

3. Neural Network Classifiers

- Softmax (≈ logistic regression) alone not very powerful
- Softmax gives only linear decision boundaries



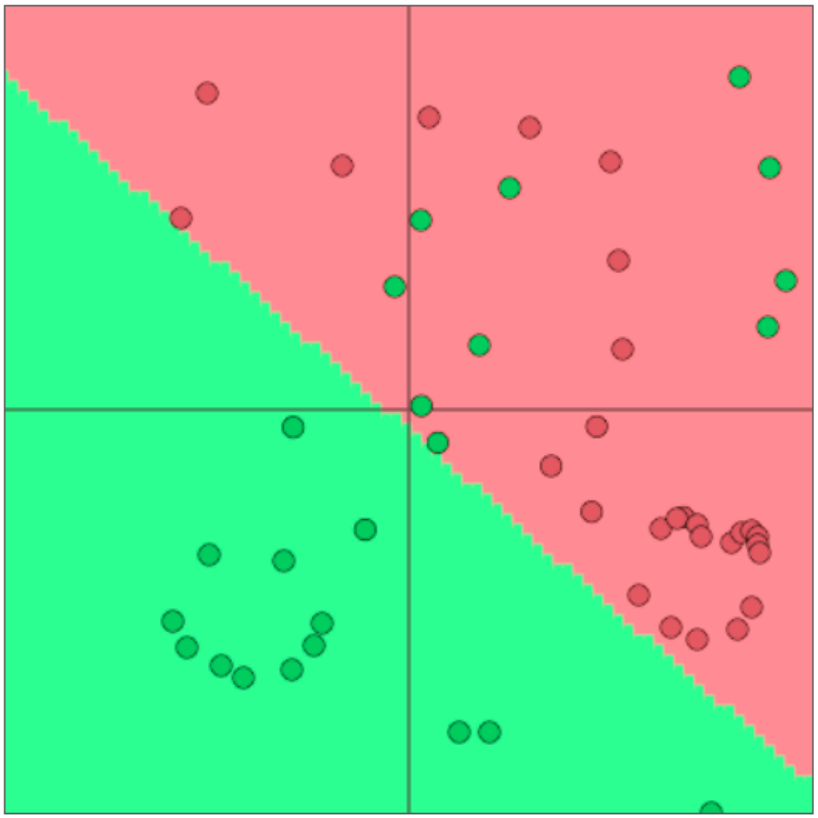
This can be quite limiting

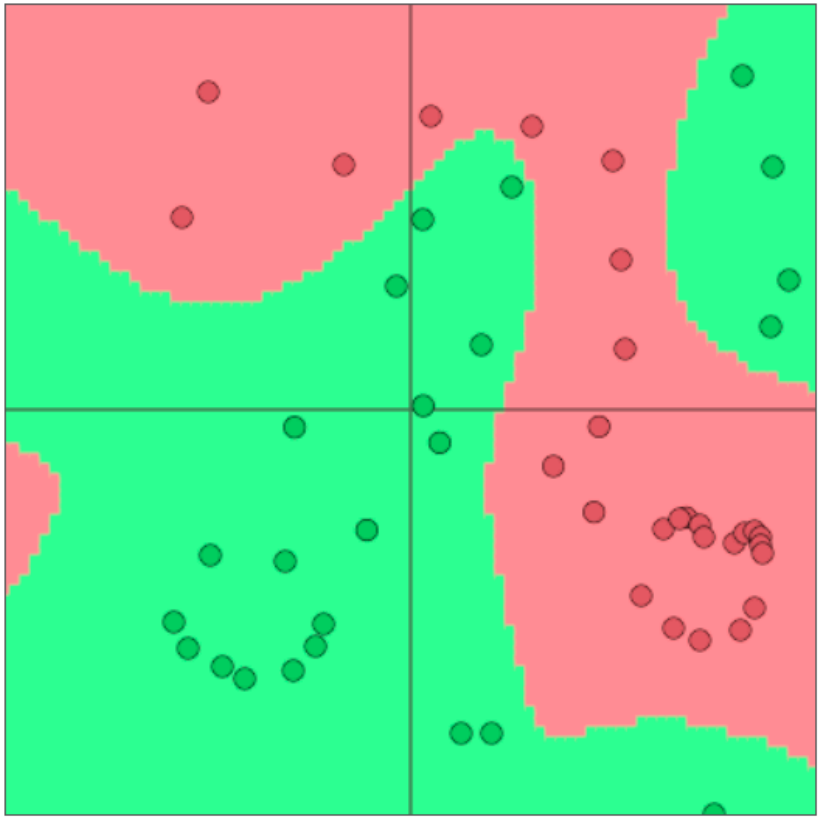
→ Unhelpful when a problem is complex

Wouldn't it be cool to get these correct?

Neural Nets for the Win!

- Neural networks can learn much more complex functions and nonlinear decision boundaries!
  - In original space





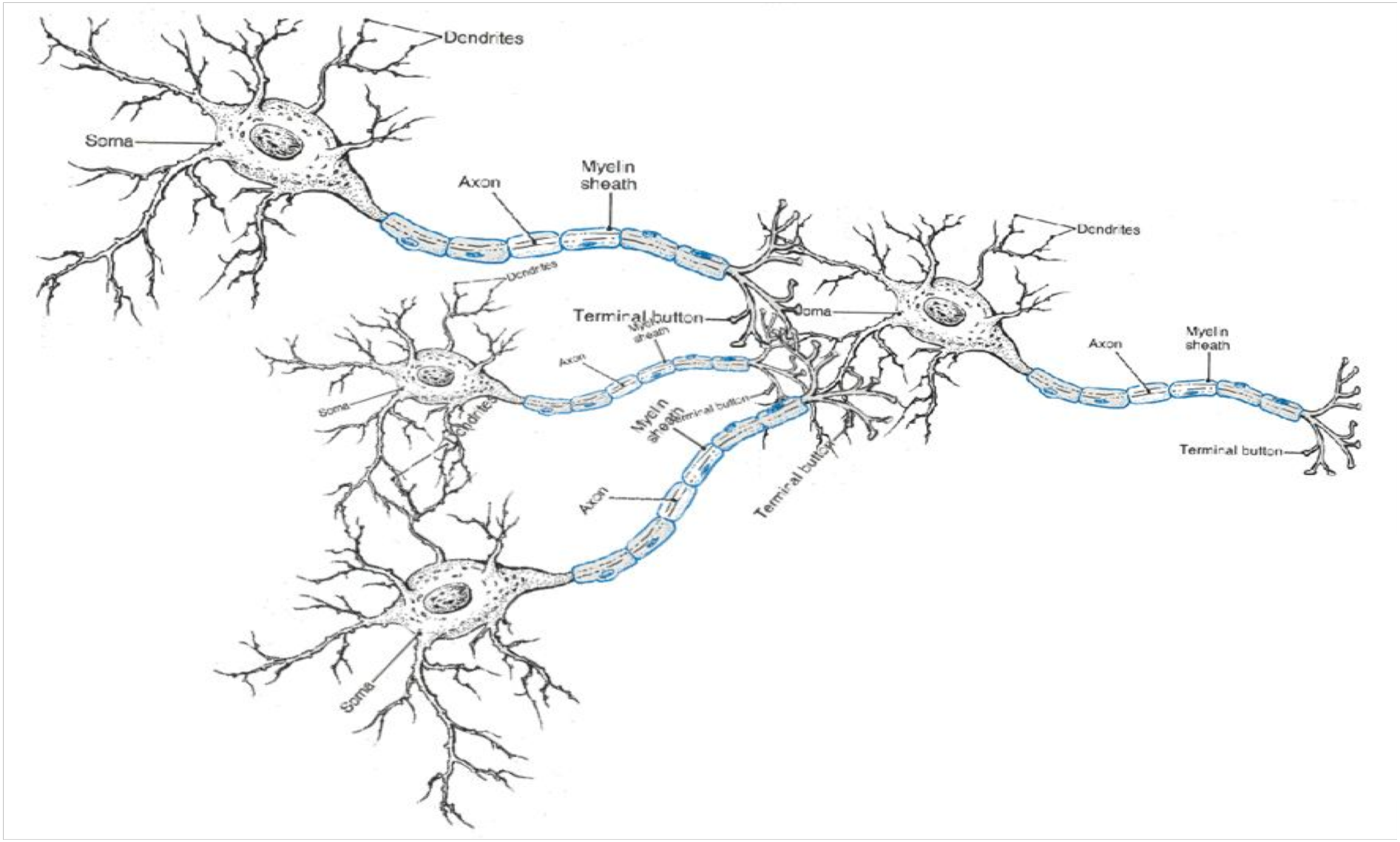
Classification difference with word vectors

- Commonly in NLP deep learning:
  - We learn **both** *W* **and** word vectors *x*
  - We learn **both** conventional parameters **and** representations
  - The word vectors re-represent one-hot vectors—move them around in an intermediate layer vector space—for easy classification with a (linear) softmax classifier via layer *x* = *Le*

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla_{W_{\cdot 1}} \\ \vdots \\ \nabla_{W_{\cdot d}} \\ \nabla_{x_{aardvark}} \\ \vdots \\ \nabla_{x_{zebra}} \end{bmatrix} \in \mathbb{R}^{Cd+Vd}$$

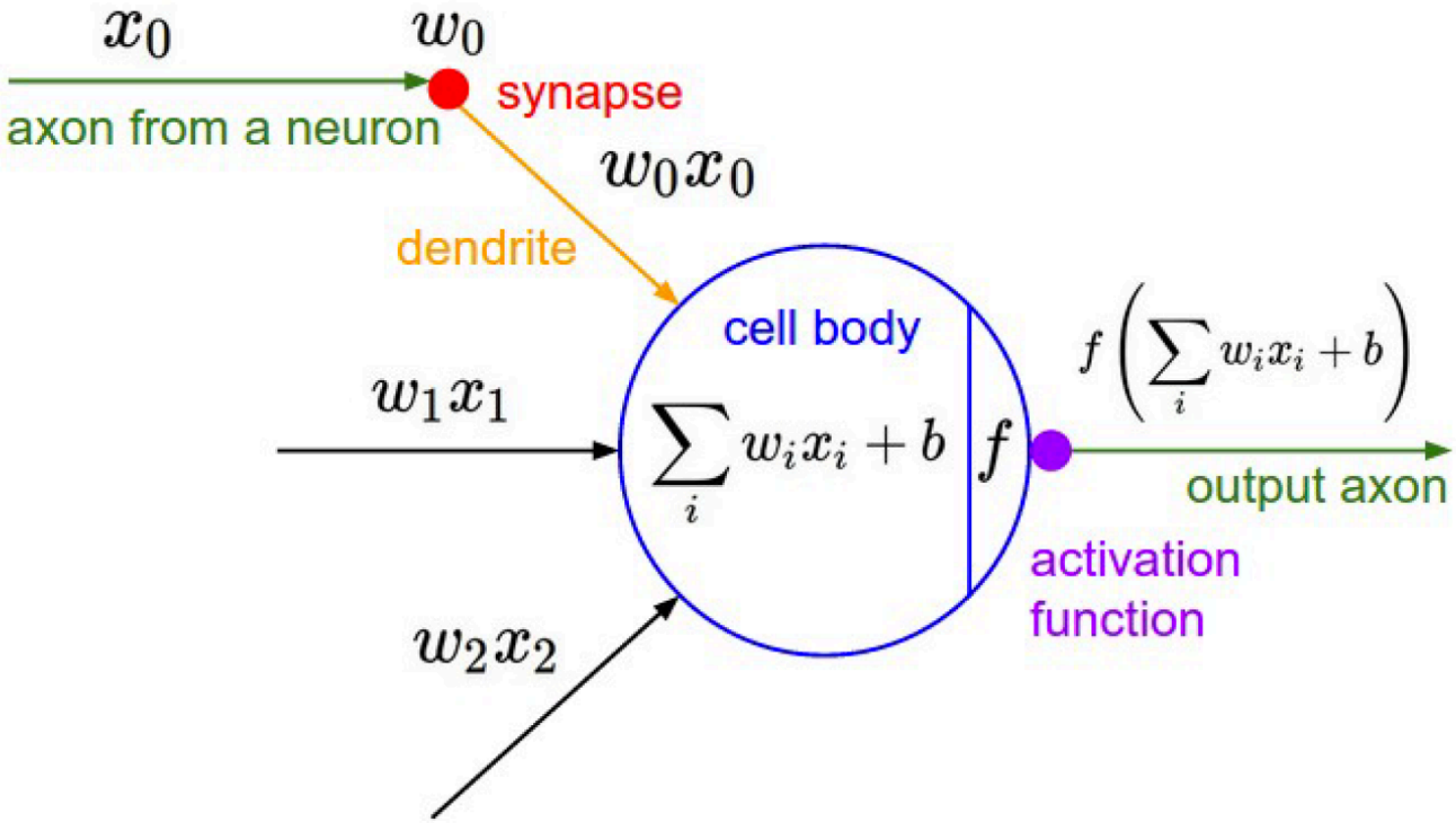
Very large number of parameters!

Neural computation



An artificial neuron

- Neural networks come with their own terminological baggage
- But if you understand how softmax models work, then you can easily understand the operation of a neuron!



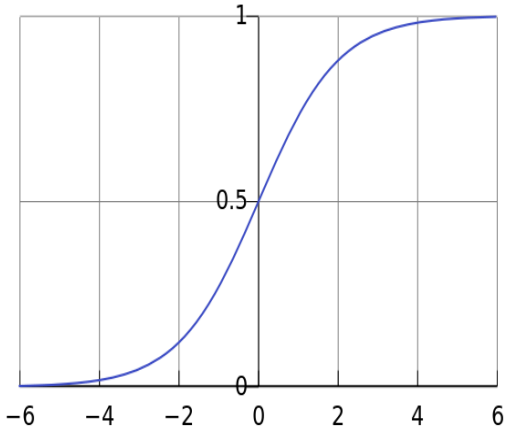
A neuron can be a binary logistic regression unit

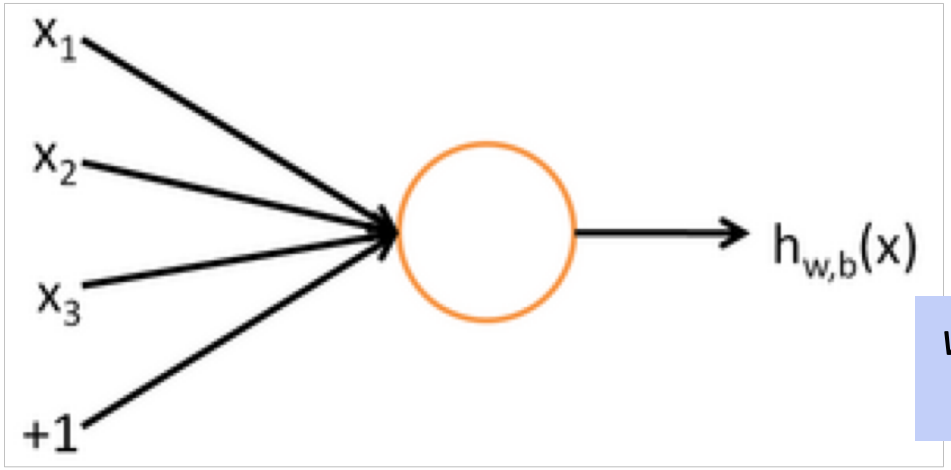
f = nonlinear activation fct. (e.g. sigmoid), *w* = weights, *b* = bias, *h* = hidden, *x* = inputs

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

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

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

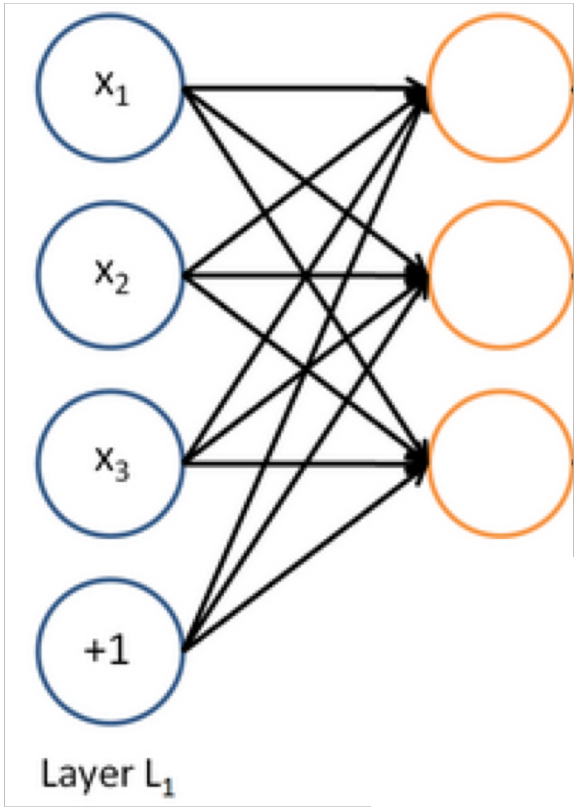




w, *b* are the parameters of this neuron i.e., this logistic regression model

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

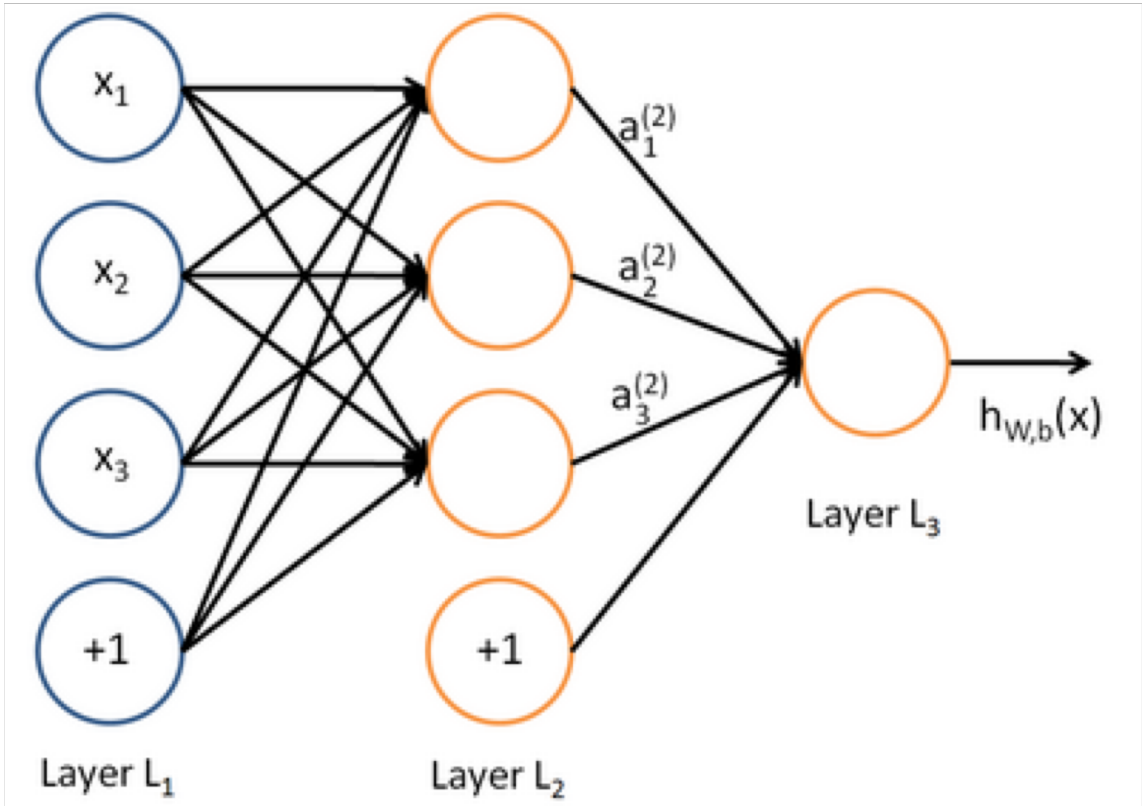
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs …



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

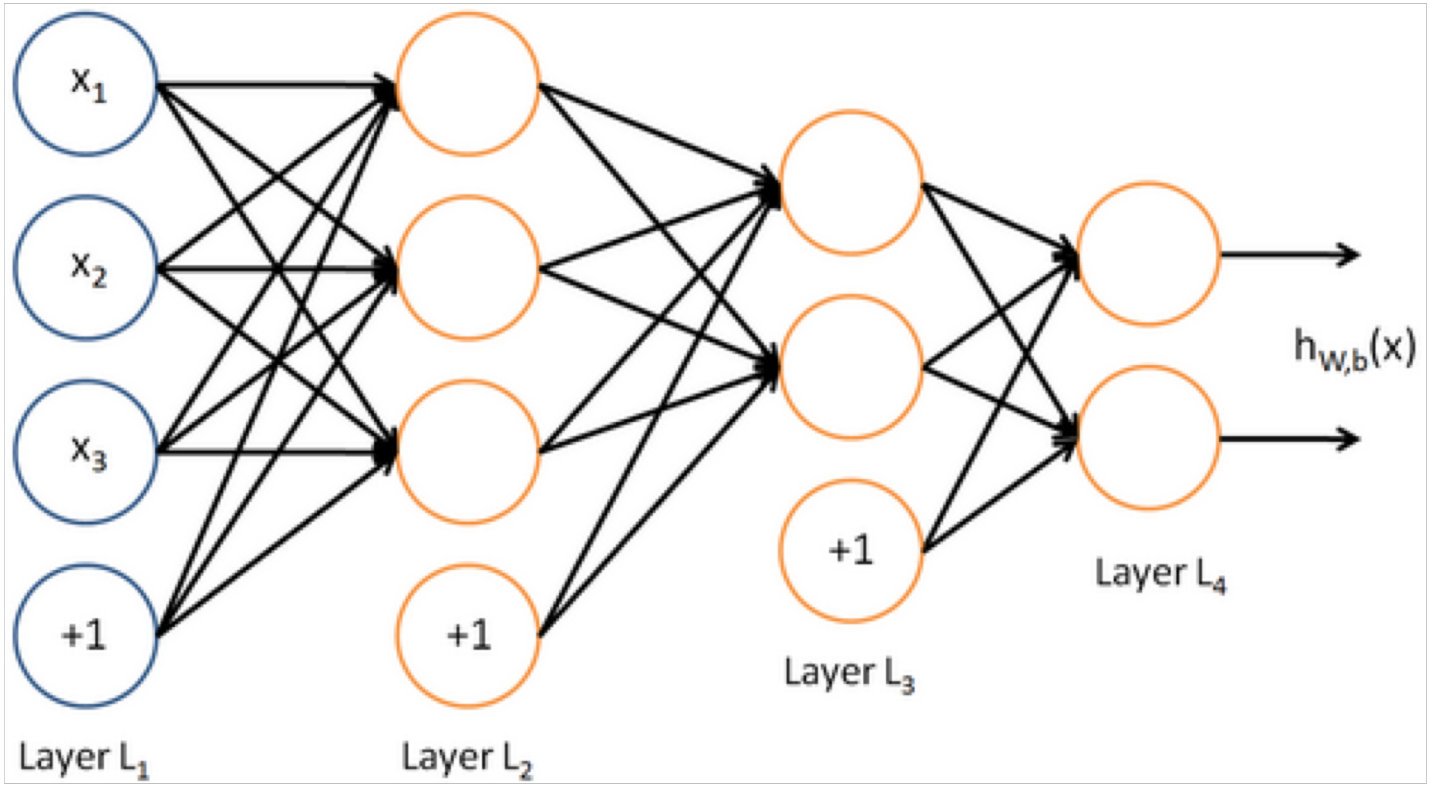
… which we can feed into another logistic regression function



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.

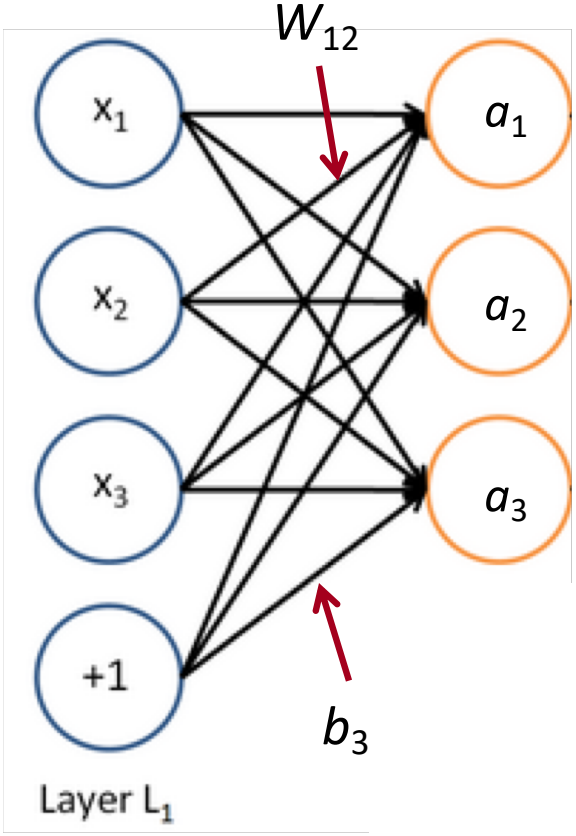
In matrix notation

$$z = Wx + b$$

$$a = f(z)$$

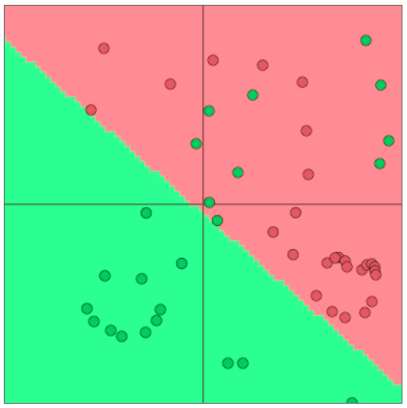
Activation f is applied element-wise:

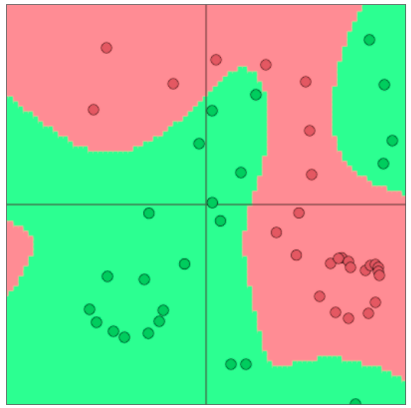
$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

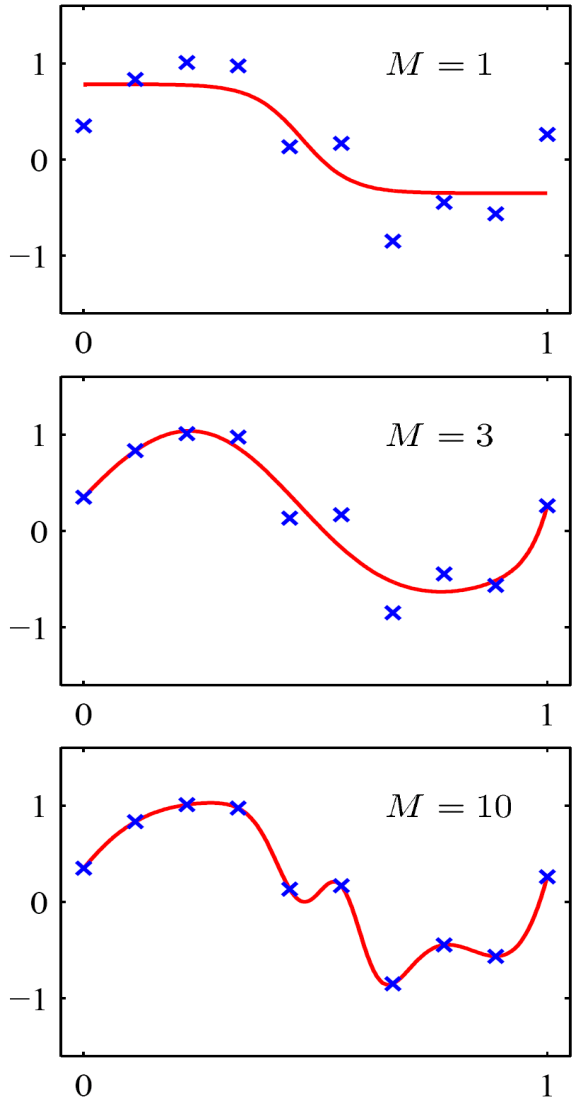


Non-linearities (aka "*f*"): Why they're needed

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







4. Named Entity Recognition (NER)

- The task: find and classify names in text, for example:

The European Commission [ORG] said on Thursday it disagreed with German [MISC] advice.

Only France [LOC] and Britain [LOC] backed Fischler [PER] 's proposal .

“What we have to be extremely careful of is how other countries are going to take Germany 's lead”, Welsh National Farmers ' Union [ORG] (NFU [ORG]) chairman John Lloyd Jones [PER] said on BBC [ORG] radio .

- Possible purposes:
  - Tracking mentions of particular entities in documents
  - For question answering, answers are usually named entities
  - A lot of wanted information is really associations between named entities
  - The same techniques can be extended to other slot-filling classifications
- Often followed by Named Entity Linking/Canonicalization into Knowledge Base

Named Entity Recognition on word sequences

We predict entities by classifying words in context and then extracting entities as word subsequences

Foreign	ORG	B-ORG
Ministry	ORG	I-ORG
spokesman	O	O
Shen	PER	B-PER
Guofang	PER	I-PER
told	O	O
Reuters	ORG	B-ORG
that	O	O
:	:	👆 BIO encoding

Why might NER be hard?

- Hard to work out boundaries of entity

  Is the first entity “First National Bank” or “National Bank”

- Hard to know if something is an entity

  Is there a school called “Future School” or is it a future school?

- Hard to know class of unknown/novel entity:

  What class is “Zig Ziglar”? (A person.)

- Entity class is ambiguous and depends on context

  “Charles Schwab” is PER not ORG here! 👉

5. Binary word window classification

- In general, classifying single words is rarely done
- Interesting problems like ambiguity arise in context!
- Example: auto-antonyms:
  - "To sanction" can mean "to permit" or "to punish”
  - "To seed" can mean "to place seeds" or "to remove seeds"
- Example: resolving linking of ambiguous named entities:
  - Paris → Paris, France vs. Paris Hilton vs. Paris, Texas
  - Hathaway → Berkshire Hathaway vs. Anne Hathaway

Window classification

- **Idea**: classify a word in its context window of neighboring words.

- For example, **Named Entity Classification** of a word in context:
  - Person, Location, Organization, None

- A simple way to classify a word in context might be to average the word vectors in a window and to classify the average vector
  - Problem: that would lose position information

Window classification: Softmax

- Train softmax classifier to classify a center word by taking concatenation of word vectors surrounding it in a window

- Example: Classify "Paris" in the context of this sentence with window length 2:

… museums in Paris are amazing … .

$X_{window} = [\ x_{museums}\quad x_{in}\quad x_{Paris}\quad x_{are}\quad x_{amazing}\ ]^T$

- Resulting vector $x_{window} = \boxed{x \in R^{5d}}$, a column vector!

Simplest window classifier: Softmax

- With $x = x_{window}$ we can use the same softmax classifier as before

predicted model output probability

$$\hat{y}_y = p(y|x) = \frac{\exp(W_y.x)}{\sum_{c=1}^{C} \exp(W_c.x)}$$

- With cross entropy error as before:

same

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} -\log \left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}} \right)$$

- How do you update the word vectors?
- Short answer: Just take derivatives like last week and optimize

Binary classification with unnormalized scores

Method used by Collobert & Weston (2008, 2011)

- Just recently won ICML 2018 Test of time award

- For 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
- 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.
- E.g., the positions that have an actual NER Location in their center are "true" positions and get a high score

Binary classification for NER Location

- 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.

museums in Paris are amazing

- “Corrupt“ windows are easy to find and there are many: Any window whose center word isn’t specifically labeled as NER location in our corpus

Not all museums in Paris

Neural Network Feed-forward Computation

Use neural activation *a* simply to give an unnormalized score

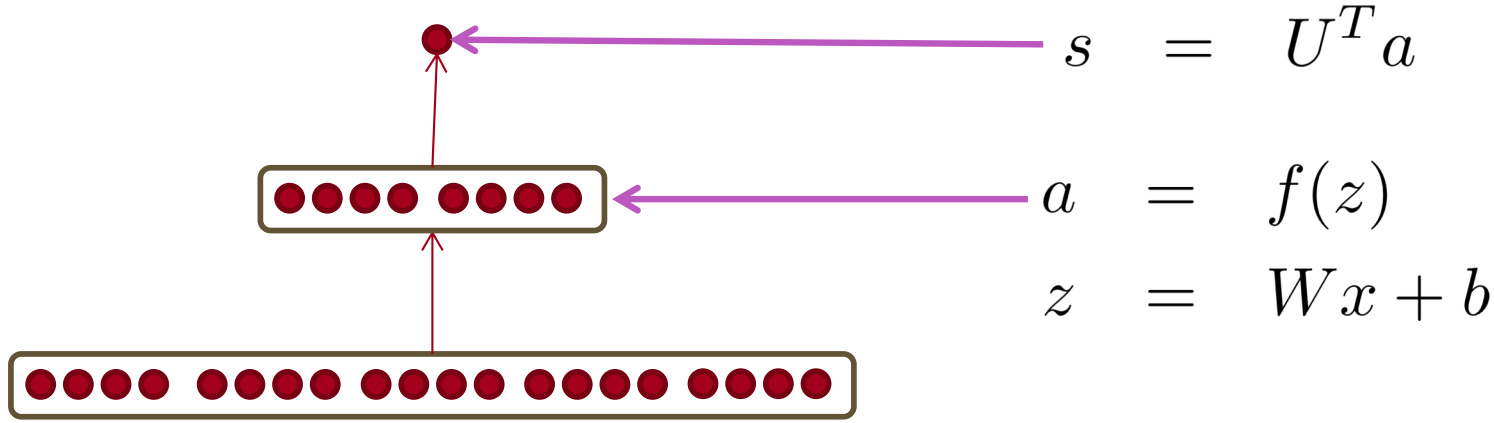
$$score(x) = U^T a \in \mathbb{R}$$

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

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

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

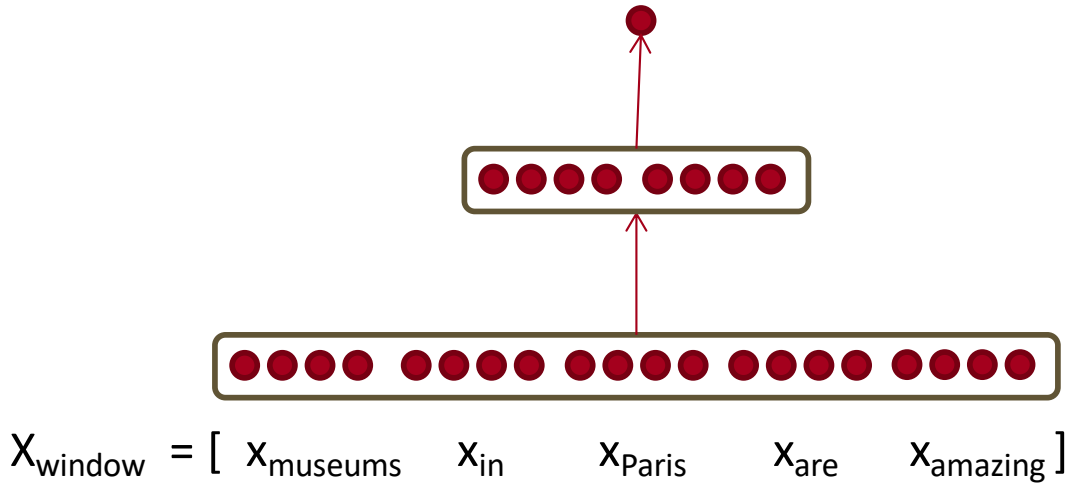
$$x \in \mathbb{R}^{20\times 1}, W \in \mathbb{R}^{8\times 20}, U \in \mathbb{R}^{8\times 1}$$



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

Main intuition for extra layer

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



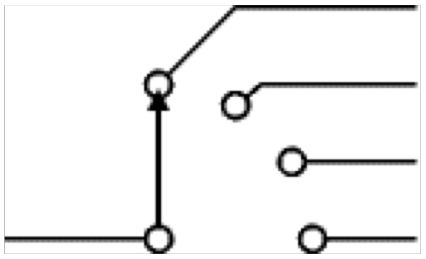
Example: only if "*museums*" is first vector should it matter that "*in*" is in the second position

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)
- s = score(museums in Paris are amazing)
- s_c = score(Not all museums in Paris)
- Minimize

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

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



Each option is continuous

Max-margin loss

- Objective for 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
- xxx |← 1 →| ooo

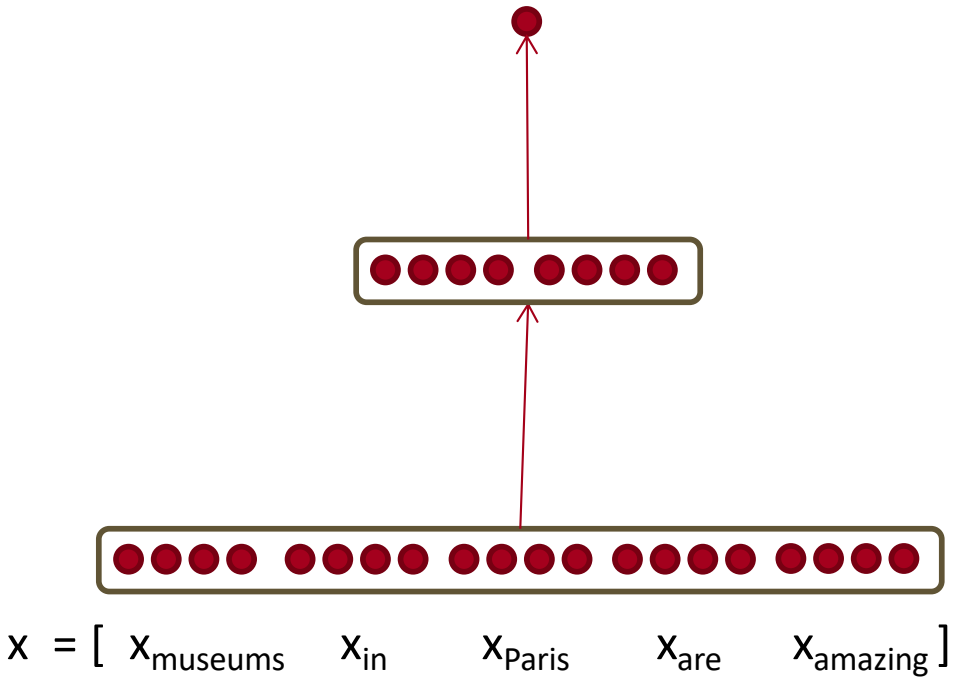
- For full objective function: Sample several corrupt windows per true one. Sum over all training windows.
- Similar to negative sampling in word2vec

Simple net for score

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\boldsymbol{x} \quad \text{(input)}$$



Remember: Stochastic Gradient Descent

- Update equation:

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

α = *step size* or *learning rate*

- How do we compute $\nabla_\theta J(\theta)$?
  - By hand (this lecture)
  - Algorithmically: the backpropagation algorithm (next lecture)

Computing Gradients by Hand

- Review of multivariable derivatives
- Matrix calculus: Fully vectorized gradients
  - Much faster and more useful than non-vectorized gradients
  - But doing a non-vectorized gradient can be good practice; watch last week's lecture for an example
  - **Lecture notes cover this material in more detail**

Gradients

- Given a function with 1 output and 1 input

$$f(x) = x^3$$

- It's gradient (slope) is its derivative

$$\frac{df}{dx} = 3x^2$$

Gradients

- Given a function with 1 output and *n* inputs

$$f(\boldsymbol{x}) = f(x_1, x_2, \ldots, x_n)$$

- It's gradient is a vector of partial derivatives with respect to each input

$$\frac{\partial f}{\partial \boldsymbol{x}} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right]$$

Jacobian Matrix: Generalization of the Gradient

- Given a function with ***m* outputs** and *n* inputs

$$\boldsymbol{f}(\boldsymbol{x}) = [f_1(x_1, x_2, ..., x_n), ..., f_m(x_1, x_2, ..., x_n)]$$

- It's Jacobian is an ***m* x *n* matrix** of partial derivatives

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial f_i}{\partial x_j}$$

Chain Rule

- For one-variable functions: **multiply derivatives**

$$z = 3y$$

$$y = x^2$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = (3)(2x) = 6x$$

- For multiple variables at once: **multiply Jacobians**

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \dots$$

Example Jacobian: Elementwise activation Function

$$\boldsymbol{h} = f(\boldsymbol{z}), \text{what is } \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}? \qquad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$$

$$h_i = f(z_i)$$

Example Jacobian: Elementwise activation Function

$$\boldsymbol{h} = f(\boldsymbol{z}), \text{what is } \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}? \qquad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$$

$$h_i = f(z_i)$$

Function has *n* outputs and *n* inputs → *n* by *n* Jacobian

Example Jacobian: Elementwise activation Function

$$\boldsymbol{h} = f(\boldsymbol{z}), \text{what is } \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}? \qquad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$$

$$h_i = f(z_i)$$

$$\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \qquad \text{definition of Jacobian}$$

Example Jacobian: Elementwise activation Function

$$\boldsymbol{h} = f(\boldsymbol{z}), \text{what is } \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}? \qquad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$$

$$h_i = f(z_i)$$

$$\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \qquad \text{definition of Jacobian}$$

$$= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \qquad \text{regular 1-variable derivative}$$

Example Jacobian: Elementwise activation Function

$$\boldsymbol{h} = f(\boldsymbol{z}), \text{what is } \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}? \qquad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$$

$$h_i = f(z_i)$$

$$\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \qquad \text{definition of Jacobian}$$

$$= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \qquad \text{regular 1-variable derivative}$$

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = \text{diag}(\boldsymbol{f}'(\boldsymbol{z}))$$

Other Jacobians

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{W}$$

Other Jacobians

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{W}$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{I} \quad (\text{Identity matrix})$$

Other Jacobians

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{W}$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{I} \quad (\text{Identity matrix})$$

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T\boldsymbol{h}) = \boldsymbol{h}^{\boldsymbol{T}}$$

Other Jacobians

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{W}$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{I} \quad (\text{Identity matrix})$$

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T\boldsymbol{h}) = \boldsymbol{h}^{\boldsymbol{T}}$$

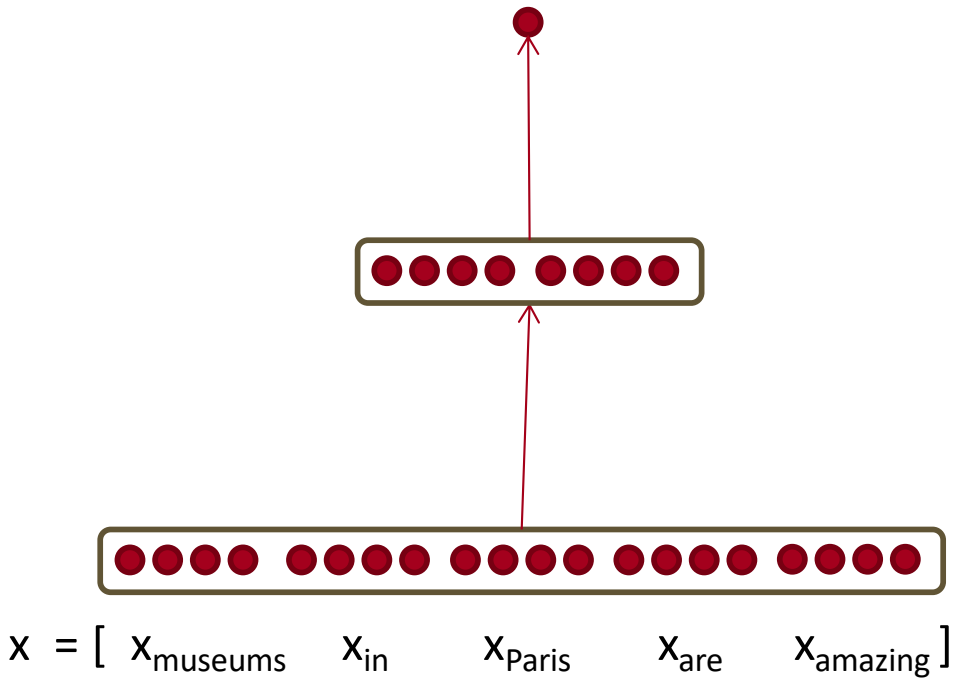
- Compute these at home for practice!
  - Check your answers with the lecture notes

Back to our Neural Net!

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\boldsymbol{x} \quad (\text{input})$$



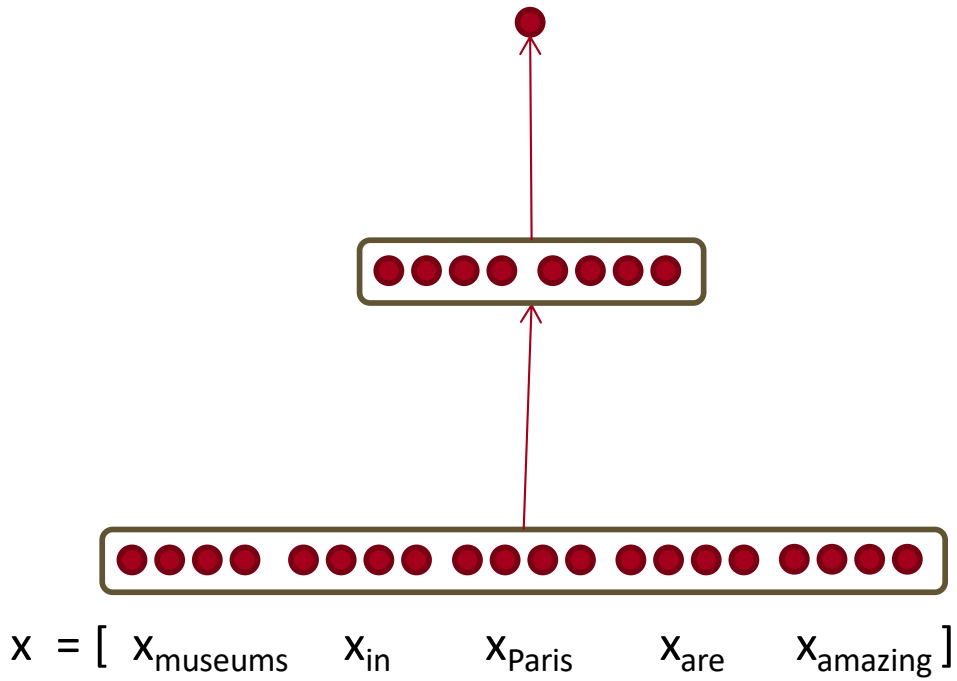
Back to our Neural Net!

- Let's find $\frac{\partial s}{\partial \boldsymbol{b}}$
  - In practice we care about the gradient of the loss, but we will compute the gradient of the score for simplicity

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\boldsymbol{x} \quad \text{(input)}$$



1. Break up equations into simple pieces

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\boldsymbol{x} \quad \text{(input)}$$

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad \text{(input)}$$

2. Apply the chain rule

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad (\text{input})$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

2. Apply the chain rule

$$\boxed{s = \boldsymbol{u}^T \boldsymbol{h}}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad (\text{input})$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \boxed{\frac{\partial s}{\partial \boldsymbol{h}}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

2. Apply the chain rule

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad (\text{input})$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

2. Apply the chain rule

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boxed{\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}}$$

$$\boldsymbol{x} \quad (\text{input})$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \boxed{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}}$$

3. Write out the Jacobians

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad (\text{input})$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \quad \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$
$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \text{diag}(f'(\boldsymbol{z}))$$
$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$

3. Write out the Jacobians

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

$$= \boldsymbol{h}^T$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$

$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \text{diag}(f'(\boldsymbol{z}))$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$

3. Write out the Jacobians

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad (\text{input})$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

$$= \boldsymbol{h}^T \text{diag}(f'(\boldsymbol{z}))$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$

$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \text{diag}(f'(\boldsymbol{z}))$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$

3. Write out the Jacobians

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

$$= \boldsymbol{h}^T \text{diag}(f'(\boldsymbol{z}))\boldsymbol{I}$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$

$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \text{diag}(f'(\boldsymbol{z}))$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$

3. Write out the Jacobians

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$
$$= \boldsymbol{h}^T \text{diag}(f'(\boldsymbol{z}))\boldsymbol{I}$$
$$= \boldsymbol{h}^T \circ f'(\boldsymbol{z})$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$
$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \text{diag}(f'(\boldsymbol{z}))$$
$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$

Re-using Computation

- Suppose we now want to compute $\frac{\partial s}{\partial \boldsymbol{W}}$
  - Using the chain rule again:

$$\frac{\partial s}{\partial \boldsymbol{W}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$

Re-using Computation

- Suppose we now want to compute $\frac{\partial s}{\partial \boldsymbol{W}}$
  - Using the chain rule again:

$$\frac{\partial s}{\partial \boldsymbol{W}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

The same! Let's avoid duplicated computation…

Re-using Computation

- Suppose we now want to compute $\frac{\partial s}{\partial \boldsymbol{W}}$
  - Using the chain rule again:

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \boldsymbol{\delta}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} = \boldsymbol{\delta}$$

$$\boldsymbol{\delta} = \frac{\partial s}{\partial \boldsymbol{h}}\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \boldsymbol{h}^T \circ f'(\boldsymbol{z})$$

δ is local error signal

Derivative with respect to Matrix: Output shape

- What does $\frac{\partial s}{\partial \boldsymbol{W}}$ look like? $\boldsymbol{W} \in \mathbb{R}^{n \times m}$
- 1 output, *nm* inputs: 1 by *nm* Jacobian?
  - Inconvenient to do $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$

Derivative with respect to Matrix

- What does $\frac{\partial s}{\partial \boldsymbol{W}}$ look like? $\boldsymbol{W} \in \mathbb{R}^{n \times m}$
- 1 output, *nm* inputs: 1 by *nm* Jacobian?
  - Inconvenient to do $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$

- Instead follow convention: shape of the gradient is shape of parameters
  - So $\frac{\partial s}{\partial \boldsymbol{W}}$ is *n* by *m*:

$$\begin{bmatrix} \frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}} \end{bmatrix}$$

Derivative with respect to Matrix

- Remember $\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$
  - $\boldsymbol{\delta}$ is going to be in our answer
  - The other term should be $\boldsymbol{x}$ because $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$

- It turns out $\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^T \boldsymbol{x}^T$

δ is local error signal at z

x is local input signal

Why the Transposes?

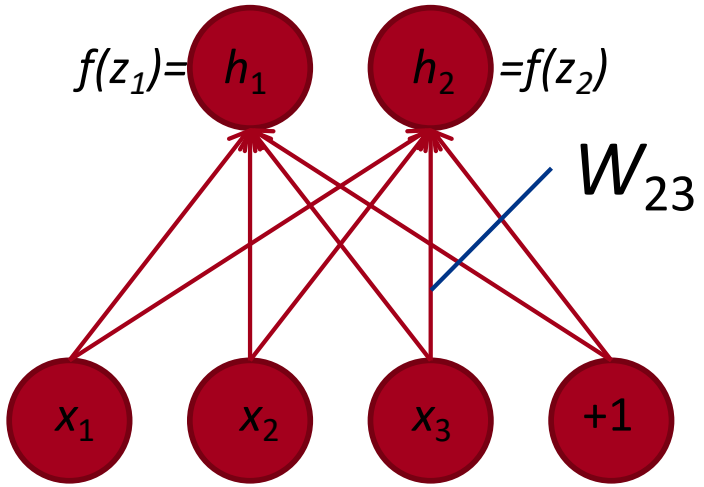
$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^T \quad \boldsymbol{x}^T$$

$$[n \times m] \quad [n \times 1][1 \times m]$$

- Hacky answer: this makes the dimensions work out!
  - Useful trick for checking your work!
- Full explanation in the lecture notes
  - Each input goes to each output – you get outer product

Why the Transposes?

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^T \boldsymbol{x}^T = \begin{bmatrix} \delta_1 \\ \vdots \\ \delta_n \end{bmatrix} [x_1, \ldots, x_m] = \begin{bmatrix} \delta_1 x_1 & \ldots & \delta_1 x_m \\ \vdots & \ddots & \vdots \\ \delta_n x_1 & \ldots & \delta_n x_m \end{bmatrix}$$



What shape should derivatives be?

- $\frac{\partial s}{\partial \boldsymbol{b}} = \boldsymbol{h}^T \circ f'(\boldsymbol{z})$ is a row vector
  - But convention says our gradient should be a column vector because $\boldsymbol{b}$ is a column vector…
- Disagreement between Jacobian form (which makes the chain rule easy) and the shape convention (which makes implementing SGD easy)
  - We expect answers to follow the **shape convention**
  - But Jacobian form is useful for computing the answers

What shape should derivatives be?

- Two options:
- 1. Use Jacobian form as much as possible, reshape to follow the convention at the end:
  - What we just did. But at the end transpose $\frac{\partial s}{\partial \boldsymbol{b}}$ to make the derivative a column vector, resulting in $\boldsymbol{\delta}^T$
- 2. Always follow the convention
  - Look at dimensions to figure out when to transpose and/or reorder terms.

Next time: Backpropagation

Backpropagation

- Computing gradients algorithmically and efficiently
- Converting what we just did by hand into an algorithm
- Used by deep learning software frameworks (TensorFlow, PyTorch, Chainer, etc.)