Natural Language Processing with Deep Learning

CS224N/Ling284

Matthew Lamm

Lecture 3: Word Window Classification, Neural Networks, and PyTorch

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!

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

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 12:30-3:20pm, Thu 6:30–9:00pm
  - Gates ART 350 (and 320-190) – 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 4-6 pm, Gates 248. 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. Implementing WW Classifier in Pytorch (30 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

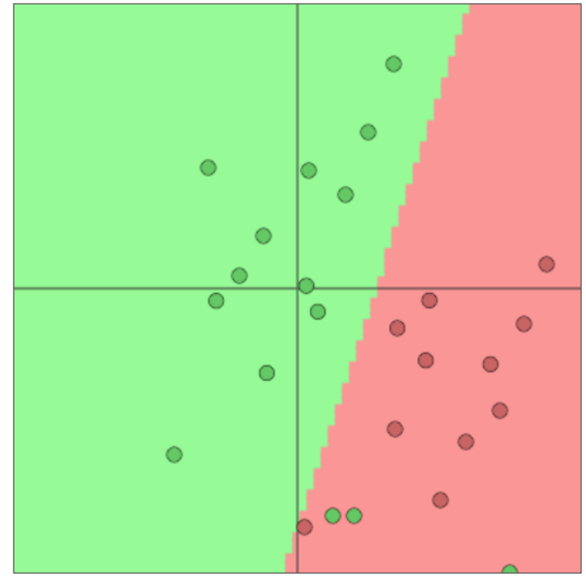
$$\{x_i, y_i\}^N_{i=1}$$

- 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: $\{x_i, y_i\}^N_{i=1}$
- 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

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

We can tease apart the prediction function into two steps:

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

$$W_y.x = \sum_{i=1}^{d} W_{yi} x_i = f_y$$

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*

- This is equivalent to minimizing the negative log probability of that class:

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

- Using log probability converts our objective function to sums, which is easier to work with on paper and in implementation.

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:

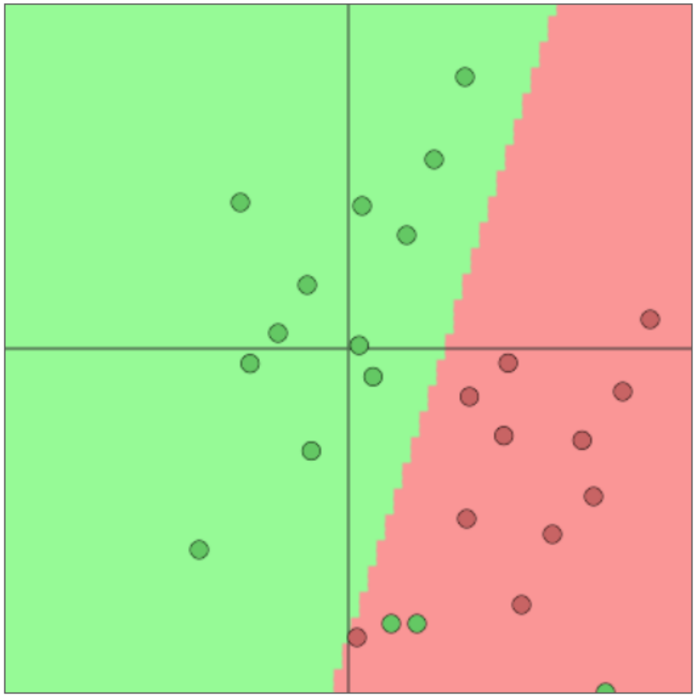
$$H(p, q) = -\sum_{c=1}^{C} p(c) \log q(c)$$

- 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 $\{x_i, y_i\}^N_{i=1}$

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



- Instead of

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

We will write f in matrix notation:

$$f = Wx$$

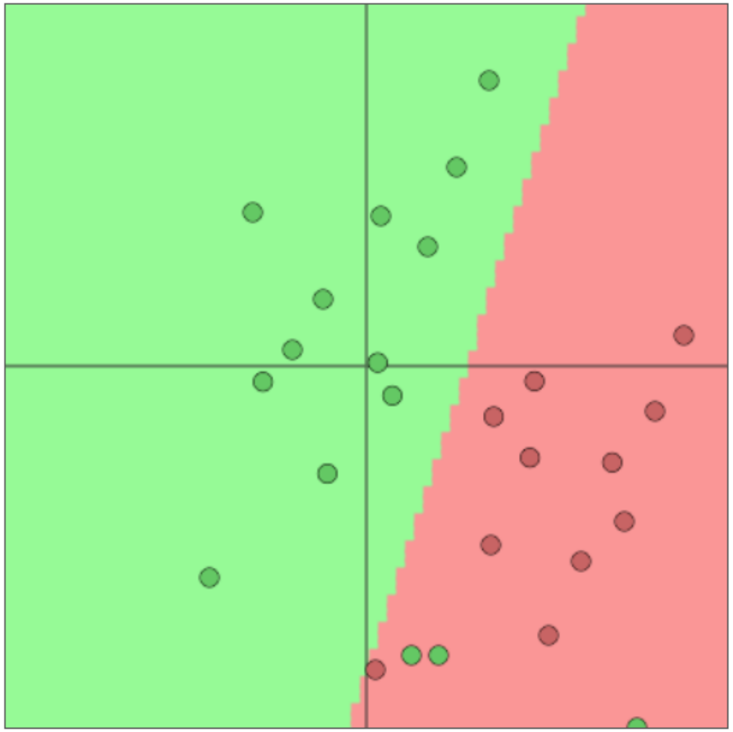
Traditional ML optimization

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

$$\theta = \begin{bmatrix} W_{1\cdot} \\ \vdots \\ W_{C\cdot} \end{bmatrix} = W \in \mathbb{R}^{Cd}$$

- So we only update the decision boundary via

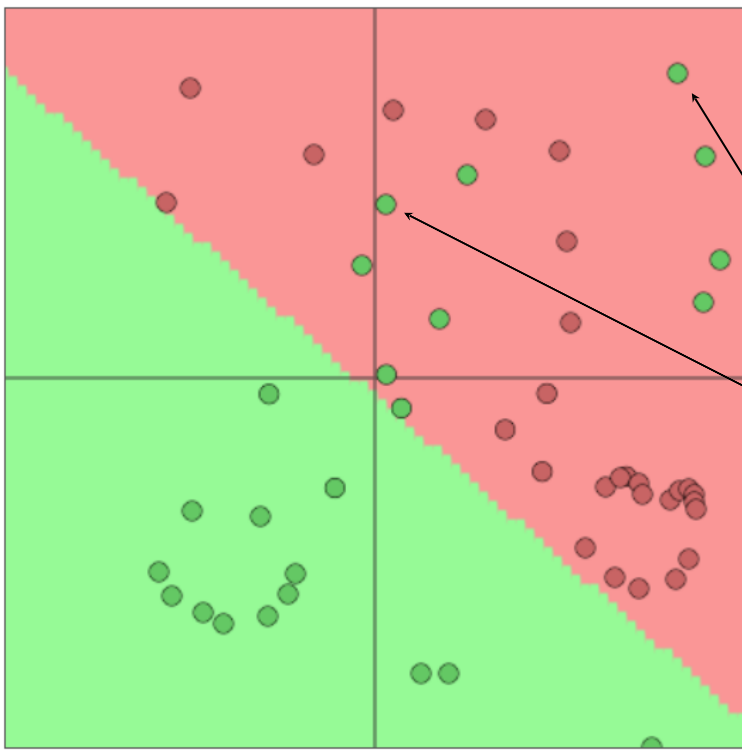
$$\nabla J(\theta) = \begin{bmatrix} \nabla W_{1\cdot} \\ \vdots \\ \nabla W_{C\cdot} \end{bmatrix} \in \mathbb{R}^{Cd}$$



Visualizations with ConvNetJS by Karpathy

3. Neural Network Classifiers

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



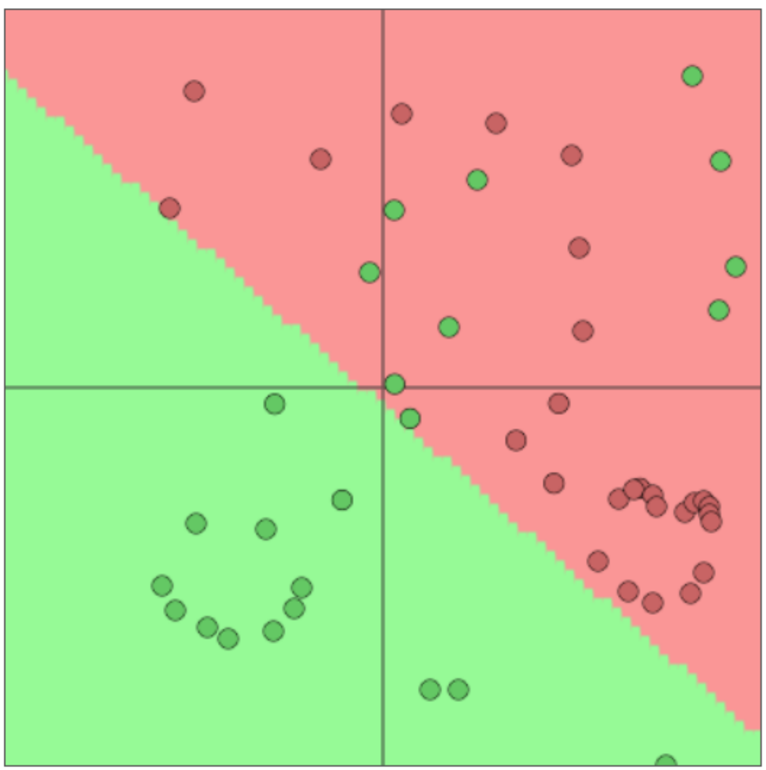
This can be quite limiting

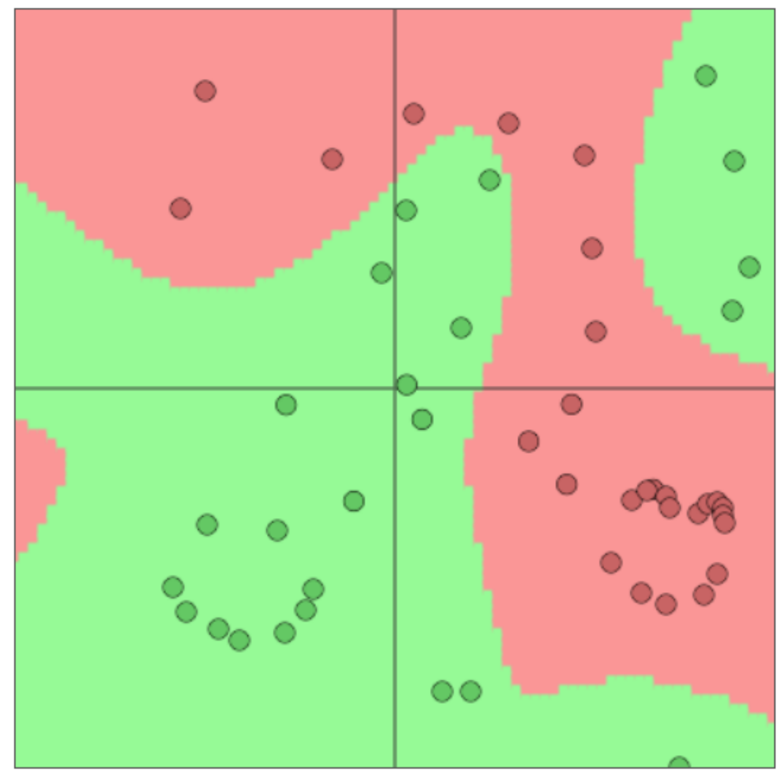
Unhelpful when a problem is complex

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!





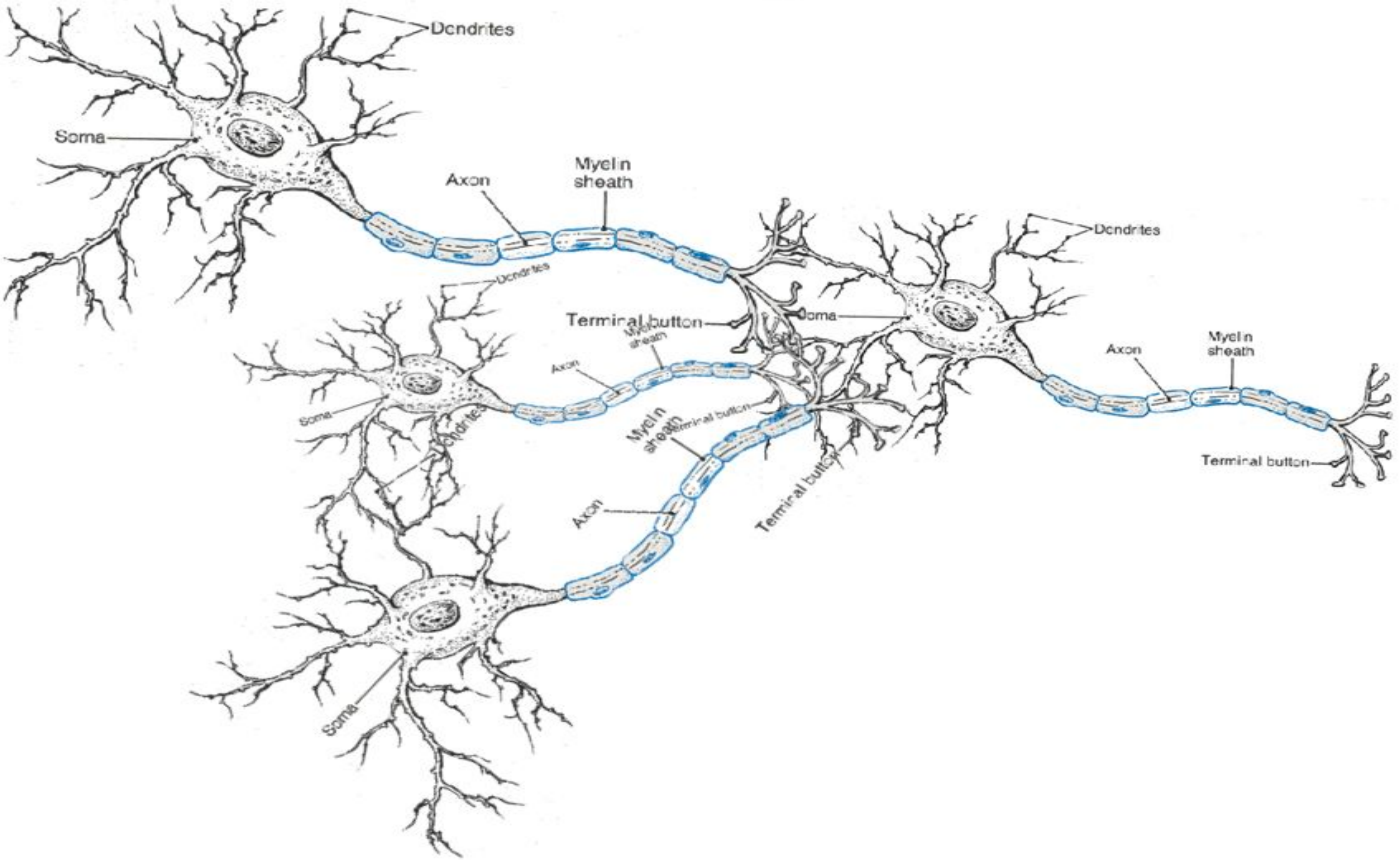
Classification difference with word vectors

- 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

Very large number of parameters!

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

Neural computation



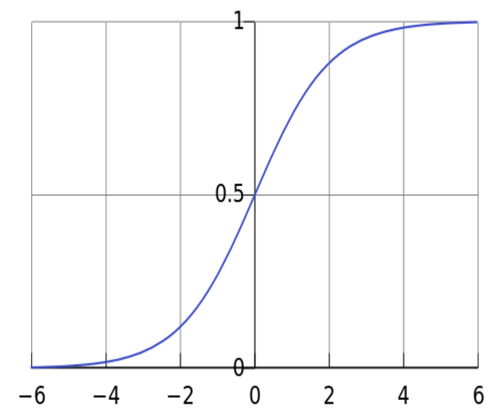
A neuron can be a binary logistic regression unit

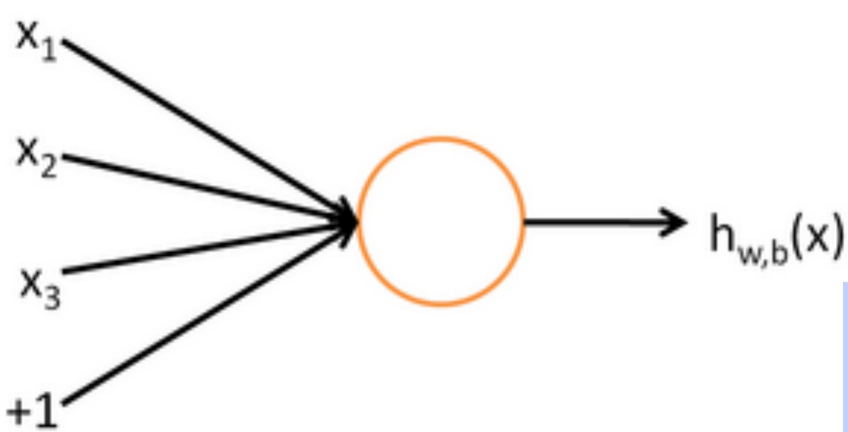
f = nonlinear activation fct. (e.g. sigmoid), *w* = weights, *b* = bias, *h* = hidden, *x* = inputs

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

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

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

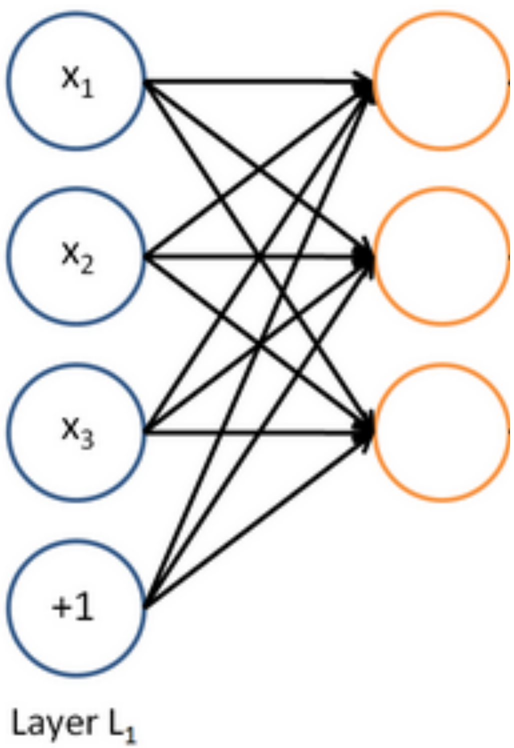




w, *b* are the parameters of this neuron
i.e., this logistic regression model

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

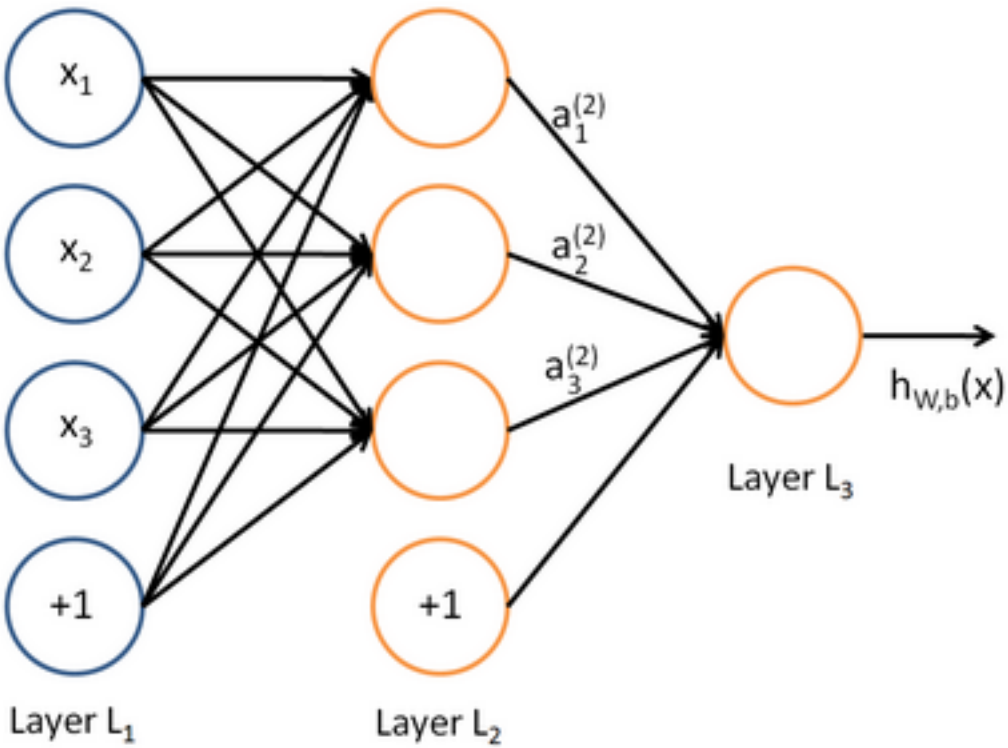
If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs …



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

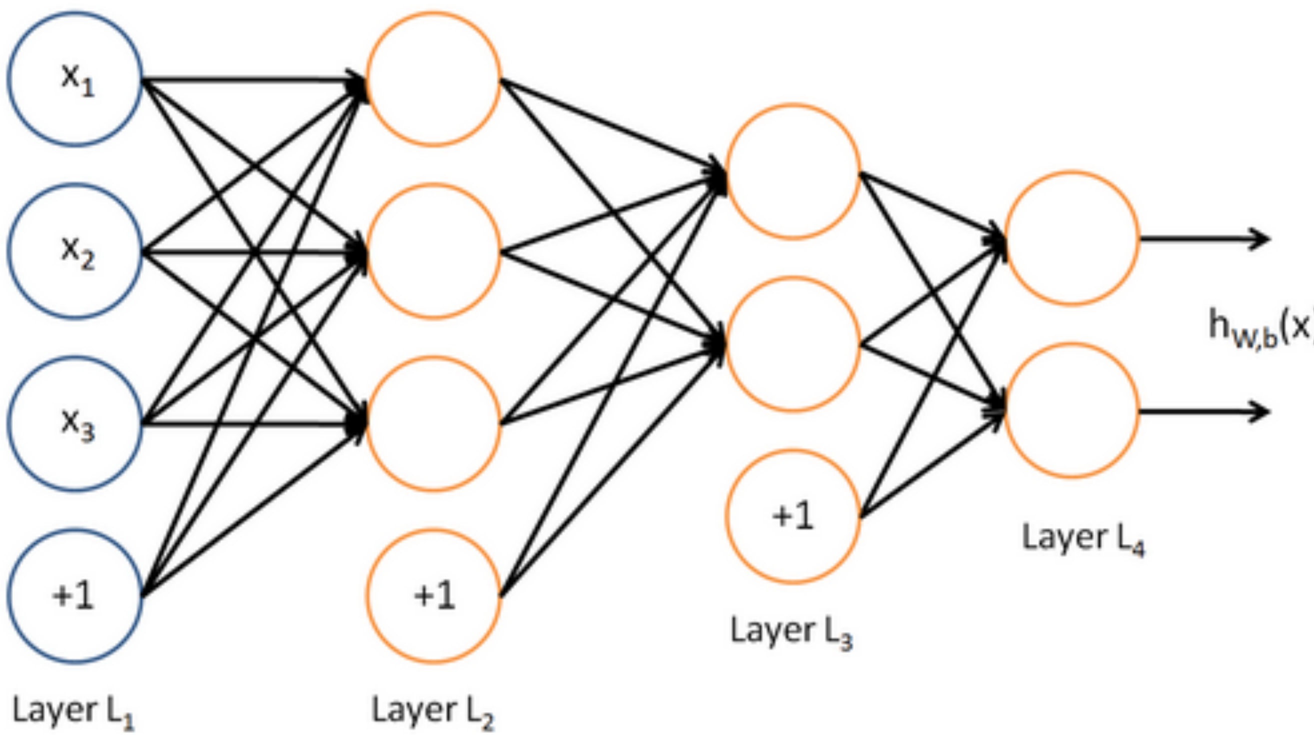
… which we can feed into another logistic regression function



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.

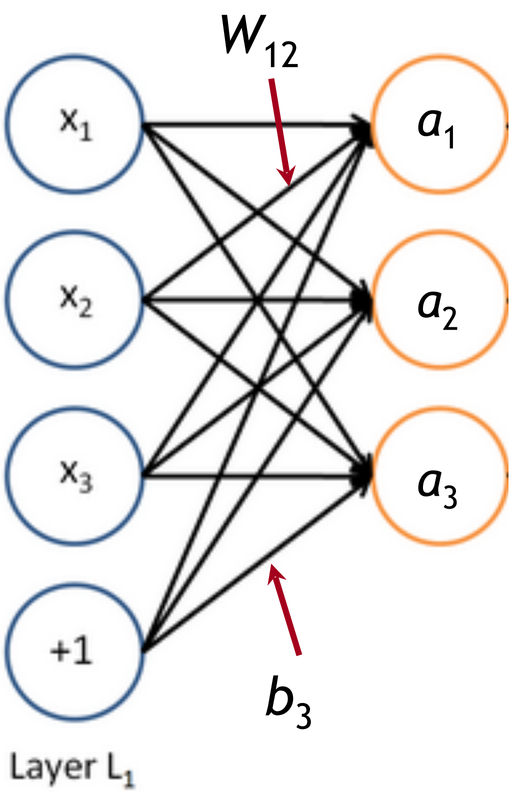
In matrix notation

$$z = Wx + b$$

$$a = f(z)$$

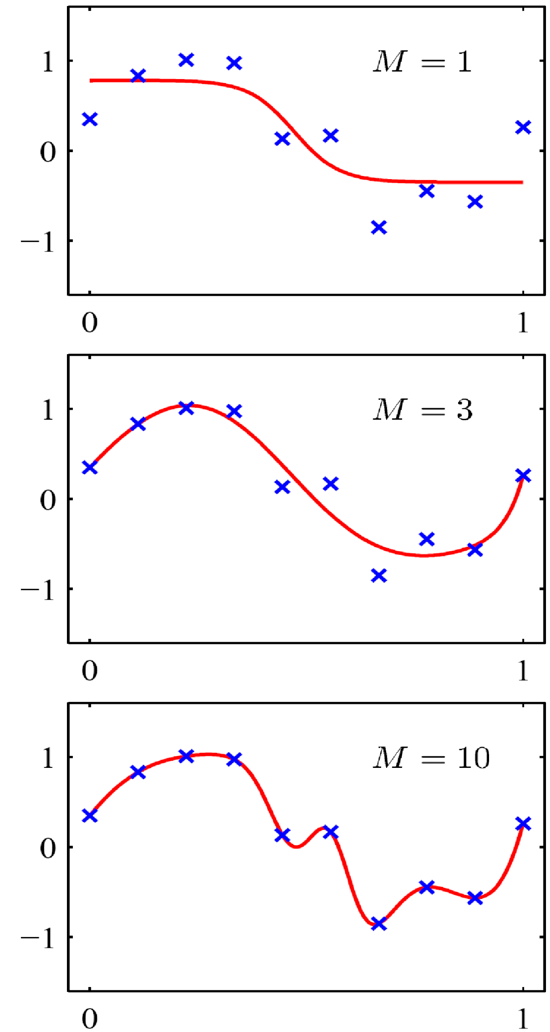
Activation f is applied element-wis

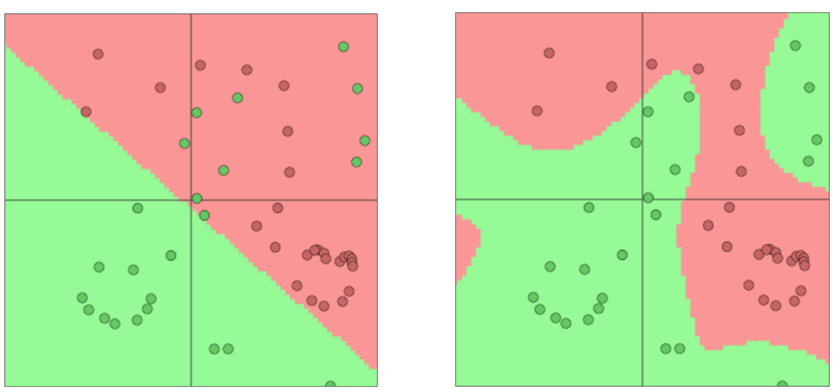
$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$



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

  First National Bank Donates 2 Vans To Future School Of Fort Smith

  POSTED 3:43 PM, JANUARY 11, 2019, BY 5NEWS WEB STAFF

  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:

  To find out more about Zig Ziglar and read features by other Creators Syndicate writers and

  What class is "Zig Ziglar"? (A person.)
- Entity class is ambiguous and depends on context

  "Charles Schwab" is PER not ORG here! 👉

  where Larry Ellison and Charles Schwab can live discreetly amongst wooded estates. And

5. Word-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

- <u>Example</u>: 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

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

- With cross entropy error as before:

same

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

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

Slightly more complex: Multilayer Perceptron

- Introduce an additional layer in our softmax classifier with a non-linearity.
- MLPs are fundamental building blocks of more complex neural systems!
- 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 s.t. positions that are true NER Locations should assign high probability to that class, and others should assign low probability.

Neural Network Feed-forward Computation

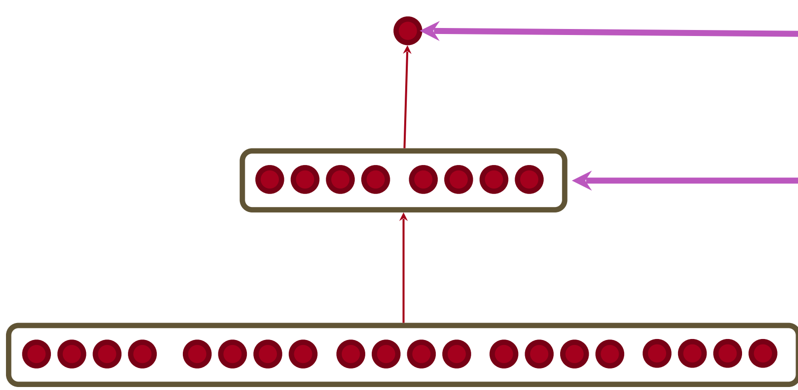
$$score(x) \quad = \quad U^T a \in \mathbb{R}$$

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

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

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

$$x \in \mathbb{R}^{20\times1}, W \in \mathbb{R}^{8x20}, U \in \mathbb{R}^{8x2}$$



$$s \quad = \quad U^T a$$

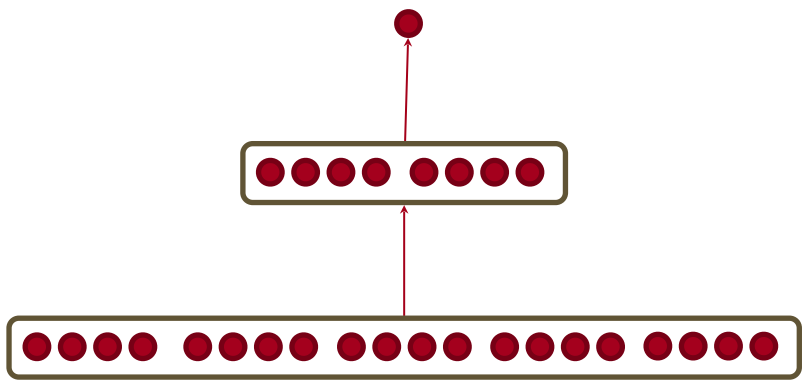
$$a \quad = \quad f(z)$$

$$z \quad = \quad Wx + b$$

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.



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

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

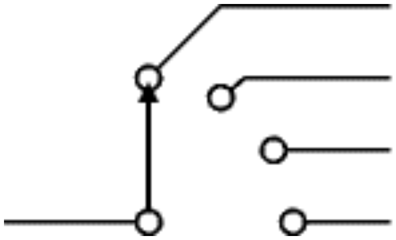
Let’s do some coding!

Alternative: Max-margin loss (no Softmax!)

- 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

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

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



Each option is continuous

Remember: Stochastic Gradient Descent

Compute gradients of the cost function, and iteratively update parameters:

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

α = *step size* or *learning rate*

Next Lecture: How do we compute gradients?
- By hand (using your knowledge of calculus)
- Backpropagation (algorithmic approach)
  - Think: loss.backward()