OBJ

ETH zürich

Sentiment Analysis with Multi-layer Perceptrons

Ryan Cotterell and Niklas Stoehr

ETH zürich

Administrivia

Project and Course Assignment Update

- We are actively writing the course assignment now
  - There are going to be some good questions for you to answer!
    - The questions will cover Lectures 1–9
  - We hope to get the final version out by November 1st, 2020
    - Why so long? We need to vet the questions with other students (not in the class), so we can assign points to them and make sure everything is doable in the proper amount of time

- We will release a set of guidelines for the course assignment very soon
  - The first draft was written, but the teaching staff need to all agree on a few details
  - Hopefully, it will be up by the end of the week
  - Keep looking for your groups!

Structure of this Lecture

1. Multi-layer Perceptrons
2. Feature Engineering and Word Embeddings
3. Sentiment Analysis

Log-Linear Model Binary Classification
Linear Separability
Non-Linear Feature Functions
MLPs
XOR Problem

Encoding Word
Pooling and *n*-grams
Word Embeddings
Unsupervised Word Repr.
Skip-Gram
BERT

What is it?
Why is it difficult?
How to build it?

bringing together all techniques learned in lecture 2-4

Supplementary Material

Reading: Eisenstein Ch. 3 and Ch. 4; Goodfellow, Bengio and Courville Ch. 6
Supplementary Material: Wikipedia, Cybenko (1989), Hanin and Selke (2018), Pang and Lee (2008), Iyyer et al. (2015)

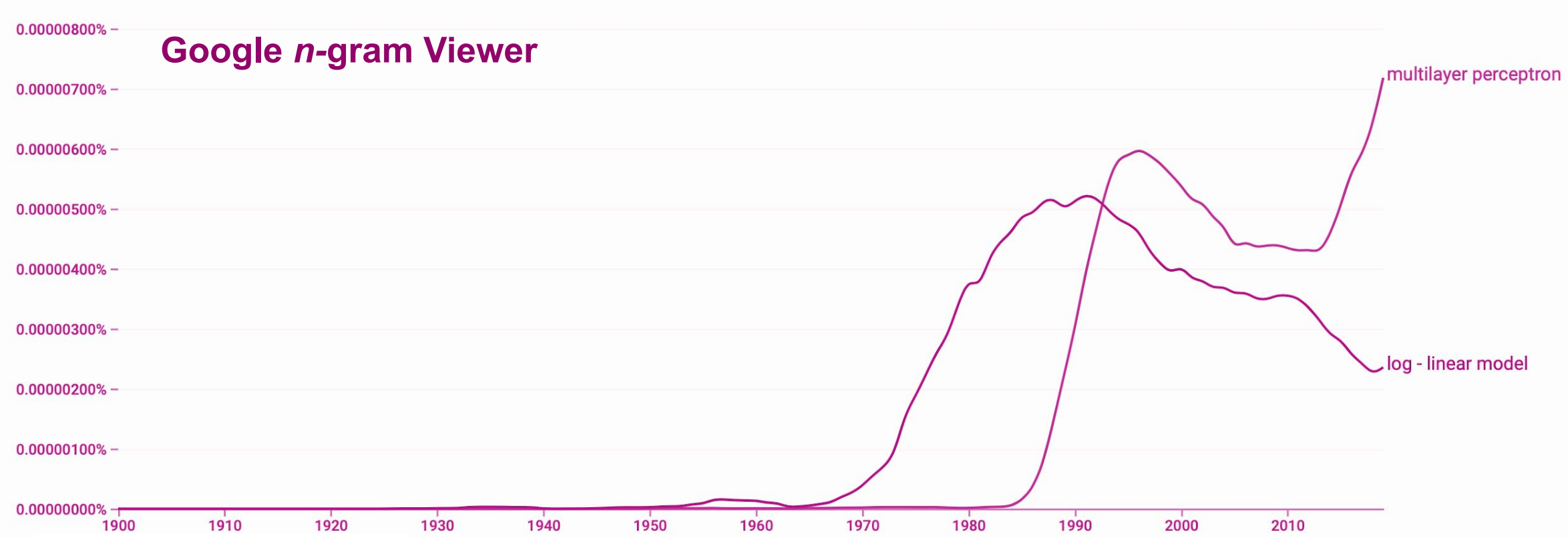
ETH zürich

Multilayer Perceptrons

Motivate why we need MLPs in cases where log-linear models are limited

1 Why Multilayer Perceptrons?

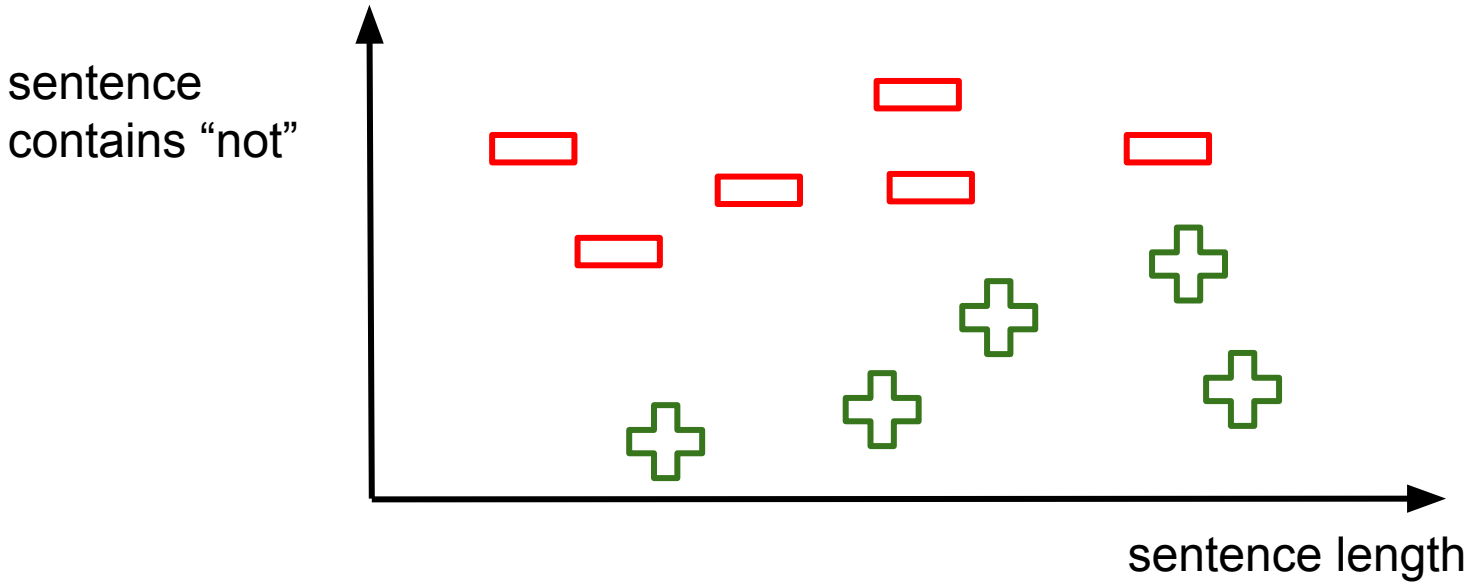
- **Difference between MLP and Neural Network (NN)**:
  MLPs are a subtype of NNs – in particular, they are fully-connected feed-forward NNs



https://books.google.com/ngrams

1 Consider a Two-Dimensional Binary Classification Problem

- Suppose we wish to classify sentence as positive (sentiment) ✚ or negative (sentiment) ▭
- Suppose we have only two features:
  a. 1 if the sentence contains "*not*" and 0 otherwise
  b. the sentence length
- Potential training data are represented below



1 Consider a Two-Feature Log-Linear Model for This Task

- **Our Goal (more formally):** Can we classify sentences $\mathcal{X}$ into a set $\mathcal{Y}$ of cardinality two
  - We have $\mathcal{X}$ = {*I like NLP*, *I dislike apples*, …}
  - We have $\mathcal{Y}$ = {-1, 1}, i.e. a set of binary sentiment where "-1" represents negative and "+1" positive sentiment

- Consider a simple log-linear model with:
  - Feature function $\mathbf{f}(\mathbf{x}, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^2$
  - Parameters $\boldsymbol{\theta} \in \mathbb{R}^2$

- Finally, we define our log-linear model:
  - $h_y = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y)$
  - and our model $p(y \mid \mathbf{x}) = \mathrm{softmax}(\mathbf{h}, y, T) = \frac{\exp(h_y / T)}{\sum_{y' \in \mathcal{Y}} \exp(h_{y'} / T)}$

- This is basically a straightforward application of last week's lecture

1 What is the sigmoid?

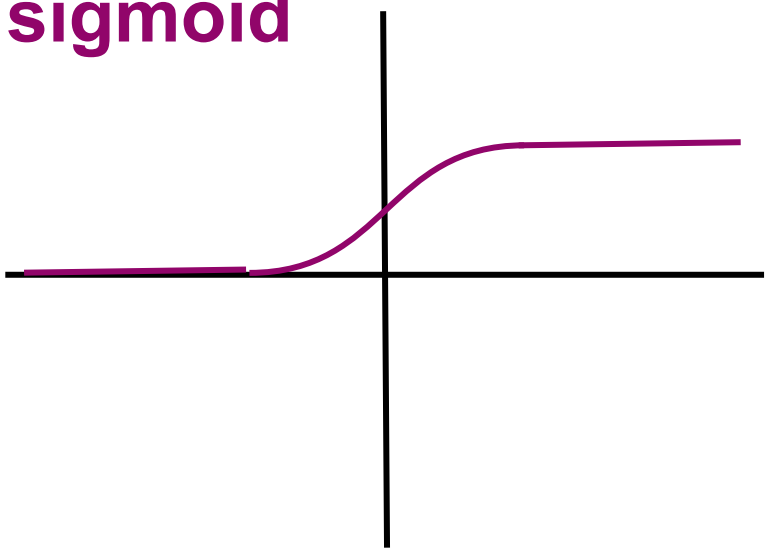
Follows by algebraic manipulation:
Divide the numerator and the denominator by $\exp(x)$

$$\text{sigmoid}(x) = \frac{\exp(x)}{1+\exp(x)} = \frac{1}{1+\exp(-x)}$$

we often use this short notation

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

sigmoid



1 Quick Derivation: Sigmoid as Two-Class Softmax

- **Important overlooked point:** the sigmoid function is a special case of the softmax
- Remember the softmax from last lecture

$$\text{softmax}(\mathbf{h}, y, T) = \frac{\exp(h_y/T)}{\sum_{y' \in \mathcal{Y}} \exp(h_{y'}/T)} \qquad \text{where} \qquad h_y = \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y)$$

- To make it simpler, we abbreviate the softmax as when $T = 1$

$$\text{softmax}(y) = \frac{\exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y'))}$$

- Sigmoid is a **binary** softmax, i.e. a softmax with two classes
- **Derivation:** First, perform the following algebraic manipulation

$$\text{softmax}(y_0) = \frac{\exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y_0))}{\exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y_0)) + \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y_1))} = \frac{\exp(\boldsymbol{\theta} \cdot (\mathbf{f}(\mathbf{x}, y_0) - \mathbf{f}(\mathbf{x}, y_1))}{1 + \exp(\boldsymbol{\theta} \cdot (\mathbf{f}(\mathbf{x}, y_0) - \mathbf{f}(\mathbf{x}, y_1)))}$$

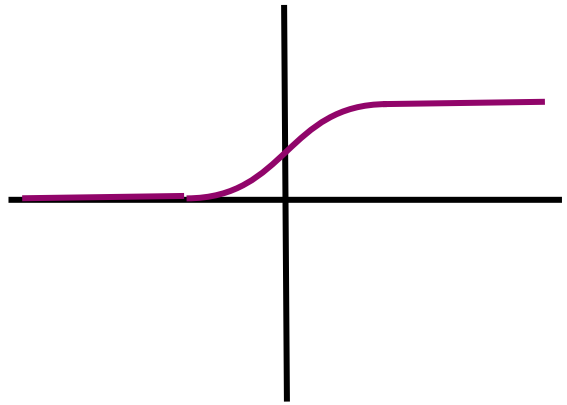
- Now, define $\mathbf{g}(\boldsymbol{x}) = \mathbf{f}(\boldsymbol{x}, y_0) - \mathbf{f}(\boldsymbol{x}, y_1)$, and we have:

$$\text{softmax}(y_0) = \frac{\exp(\boldsymbol{\theta} \cdot \mathbf{g}(\mathbf{x}))}{1 + \exp(\boldsymbol{\theta} \cdot \mathbf{g}(\mathbf{x}))} = \frac{1}{1 + \exp(-\boldsymbol{\theta} \cdot \mathbf{g}(\mathbf{x}))} = \sigma(\boldsymbol{\theta} \cdot \mathbf{g}(\mathbf{x}))$$

1 A Log-Linear Model for Binary Classification

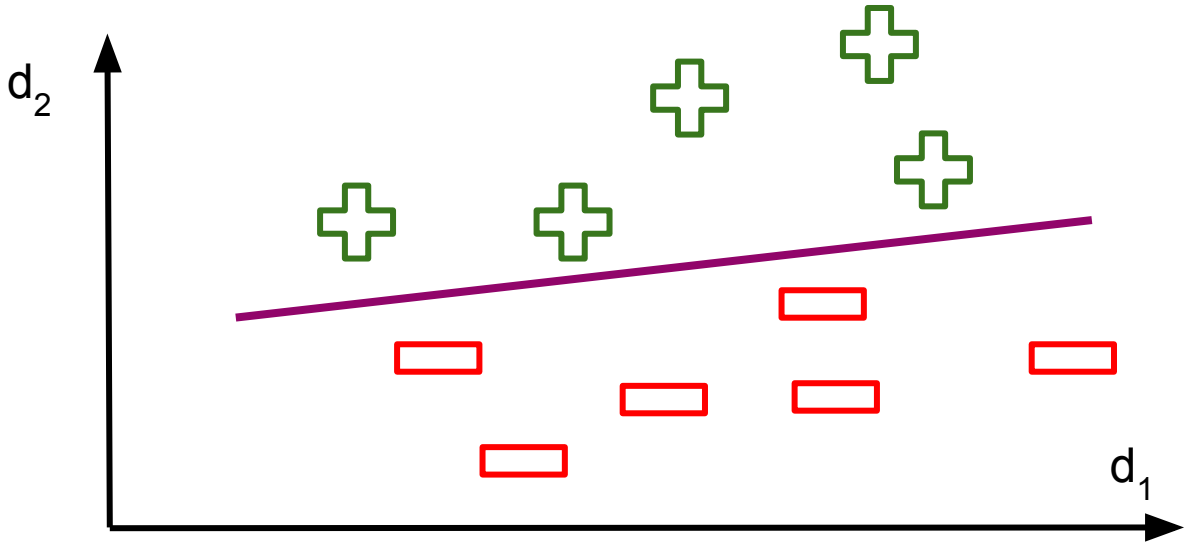
- Now, back to our log-linear model for binary classification; where, recall, we have $\mathbf{g}(\cdot) \in \mathbb{R}^2$

$$p(Y = y_1 \mid \mathbf{x}, \boldsymbol{\theta}) = \sigma\left(\boldsymbol{\theta} \cdot \mathbf{g}(\mathbf{x})\right)$$



- Consider the decision boundary of this two-class log-linear model

$$p(Y = y_1 \mid \mathbf{x}, \boldsymbol{\theta}) > 0.5 \text{ iff } \boldsymbol{\theta} \cdot \mathbf{g}(\mathbf{x}) > 0$$



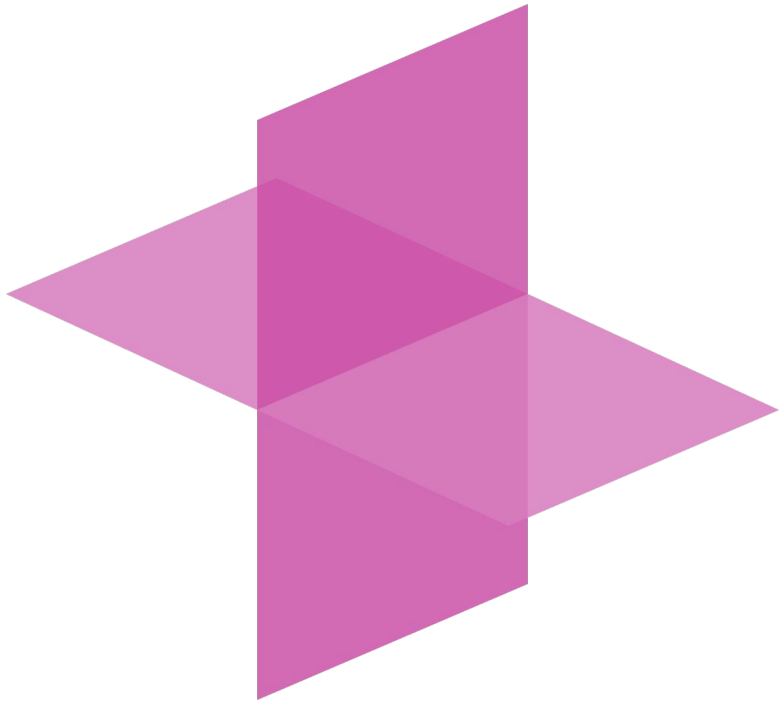
→ **decision boundary**

$$\boldsymbol{\theta} \cdot \mathbf{g}(\mathbf{x}) = 0$$

this is a line!

1 Background: What is a Hyperplane?

- A **hyperplane** is a multidimensional generalization of a line
  - A 1-dimensional hyperplane is called a line
  - A 2-dimensional hyperplane is called a plane
  - A >3-dimensional hyperplane is a just called a hyperplane
- Our feature spaces are typically large so we deal with very high-dimensional hyperplanes
- The general equation for a hyperplane is as below:



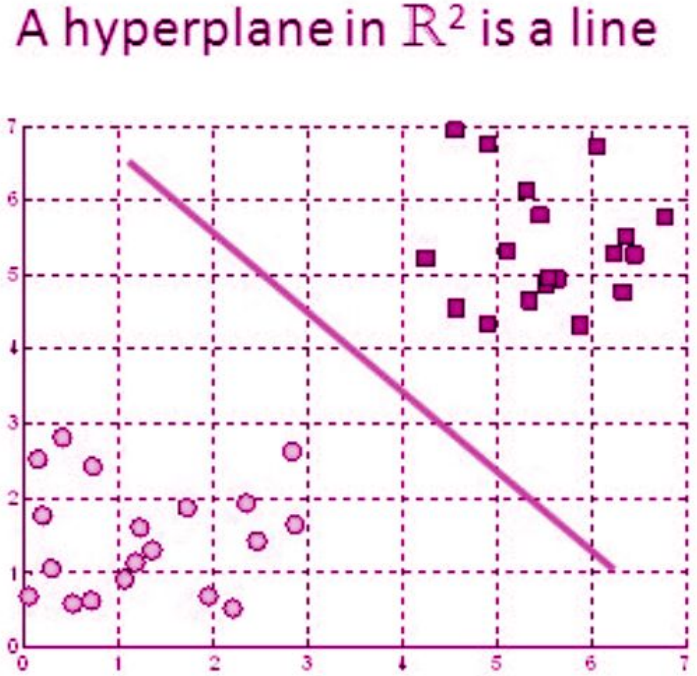
$$u = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} \quad a_1 x_1 + \cdots + a_N x_N = b$$

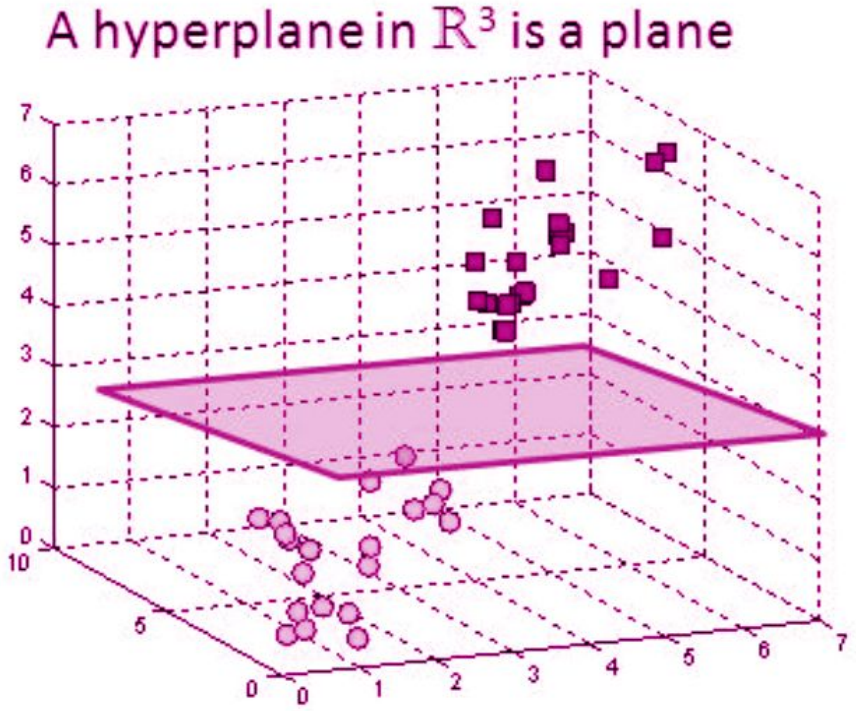
where a_1,..., a_N and b are real numbers, with at least a_1,..., $a_N \neq 0$ and b is referred to as the intercept

https://en.wikipedia.org/wiki/Hyperplane

1 What is a Separating Hyperplane?

- A separating hyperplane is a hyperplane that separates data points
- A log-linear model (and, indeed, all linear models) gives us a separating hyperplane

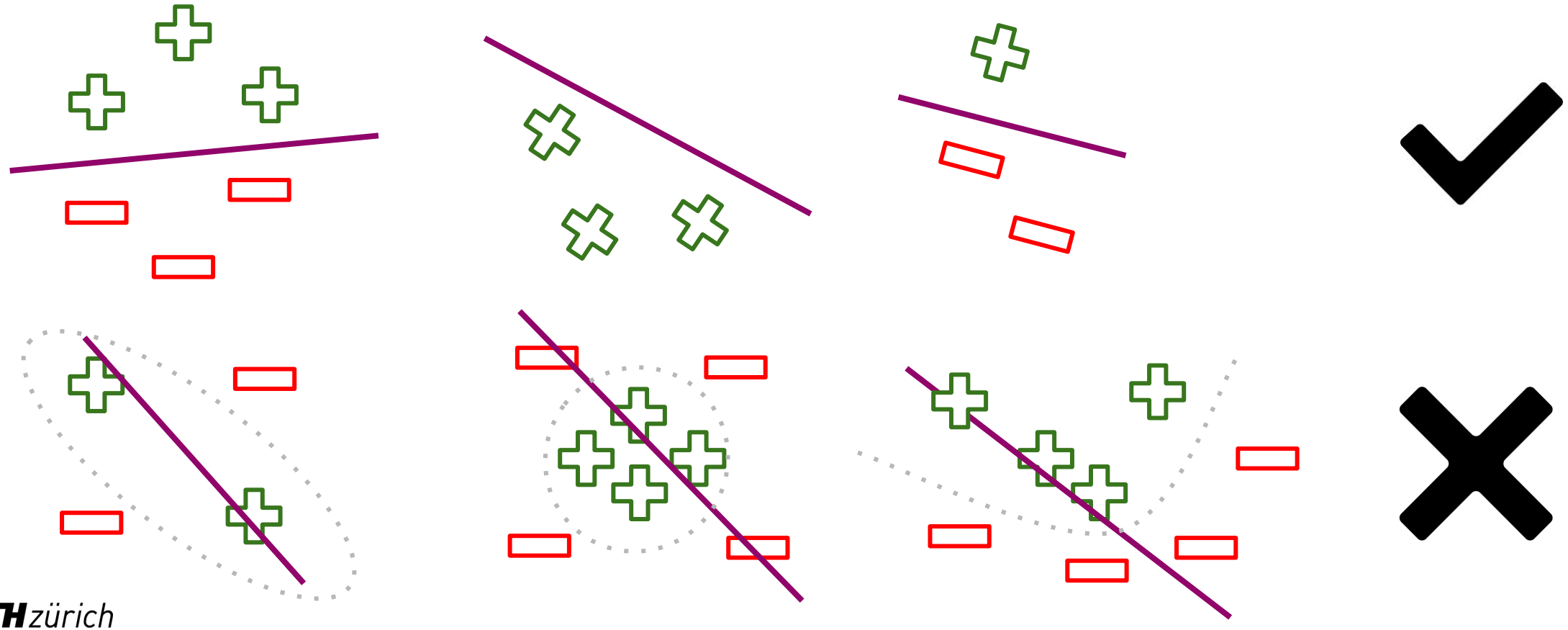




https://deepai.org/machine-learning-glossary-and-terms/hyperplane

1 What is Linear Separability?

- Linear separability is a property of two sets of points (not of a function).
- In two dimensions, two sets of points are linearly separable if there exists at least one line in the plane separating the points in the first set from the second set
- This definition generalizes to higher-dimensional Euclidean spaces if line is replaced by hyperplane.



1 A Log-Linear Model Can Learn Any Separating Hyperplane!

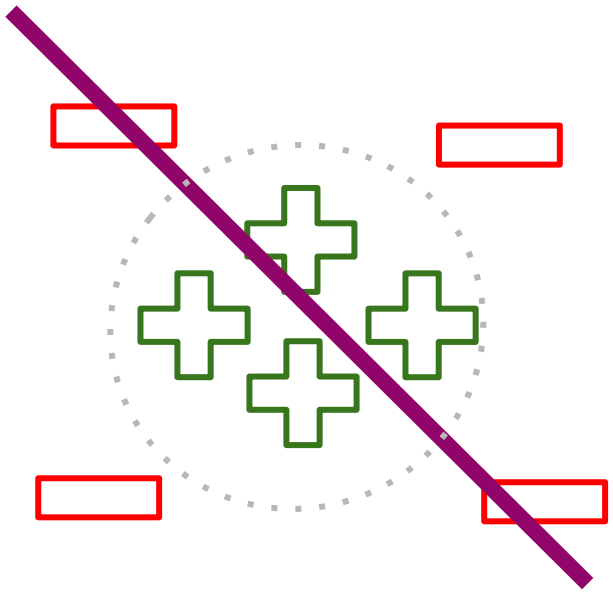
- Consider the case of binary classification
- Suppose the hyperplane $\sum_{n=1}^{N} a_n x_n > b$ separates positive and the negative classes
- Choose $\boldsymbol{\theta} = [a_1, \ldots, a_N, b]$ and $\mathbf{g}(\mathbf{x}) = [x_1, \ldots, x_N, -1]$
- Recall that the decision boundary in a binary-class log-linear model is given by $\boldsymbol{\theta} \cdot \mathbf{g}(\mathbf{x}) = 0$
- Then, we have:

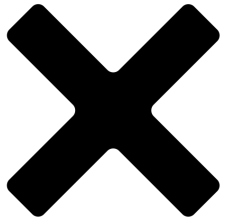
$$\boldsymbol{\theta} \cdot \mathbf{g}(\mathbf{x}) \geq 0 \iff \sum_{n=1}^{N} a_n x_n - b \geq 0$$

$$\iff \sum_{n=1}^{N} a_n x_n \geq b$$

1 Example of a Non-linear Decision Boundary

- Suppose that separating the positive and the negative classes requires an ellipsoidal decision boundary
- The identity function $\mathbf{g}(\mathbf{x}) = [\mathbf{x}]$ is not sufficient to separate the data anymore...

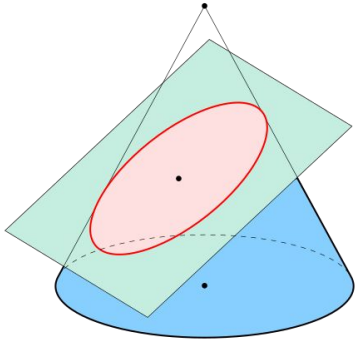




- **Quiz:** What is the function of an ellipse? Remember high school? 😂

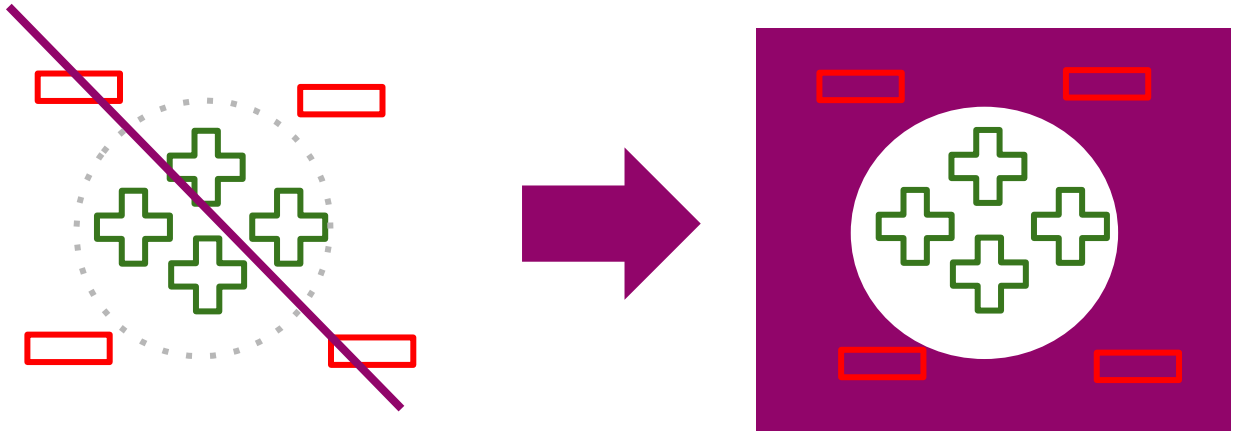
1 Non-linear feature Functions

- What is the function of an ellipse?



$$(a\cos(t), b\sin(t)) \qquad 0 \leq t \leq 2\pi$$

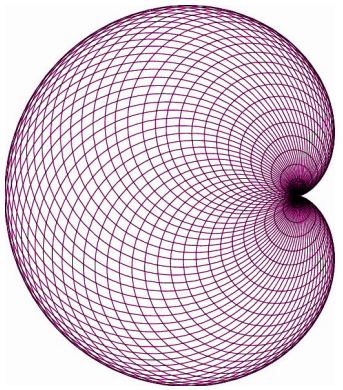
$$\frac{x_1^2}{a^2} + \frac{x_2^2}{b^2} = 1$$



- How do we model it? Use an ellipsoidal feature function: $\mathbf{g}(\mathbf{x}) = [x_1, x_1^2, x_2, x_2^2, -1]$

1 Whence the Non-linearity?

- How did we get a non-linear decision boundary out of a linear model?
  - Ellipsoidal feature function $\mathbf{f}(\mathbf{x}, y) = [x_1, x_1^2, x_2, x_2^2, -1]$
  - The model is still log-linear in the parameters $\boldsymbol{\theta}$
- We hacked the non-linearity into the feature function → Isn't that awesome?



Common bit of fake news: linear models can't model non-linear phenomena

→ Of course they can, It's easy! For any parameterized shape, you find an appropriate decision boundary!

- So, why bother with neural networks? Even with an ellipsoidal feature function, the log-likelihood of a log-linear model is still convex!
- Say, instead of a ellipsoidal decision boundary, we had a cardiodic one?
  - Cardiodic is just fancy-shmancy Greek for heart-shaped
  - Then, we need a different feature function. The ellipsoidal one does not work!
- **The TL;DR**: We need to know the decision boundary's shape *a-priori* to hack the feature function
  - Neural networks allow us to jointly *learn* a non-linear feature function with the model's parameters

1 Multi-Layer Perceptrons

Key Ideas:

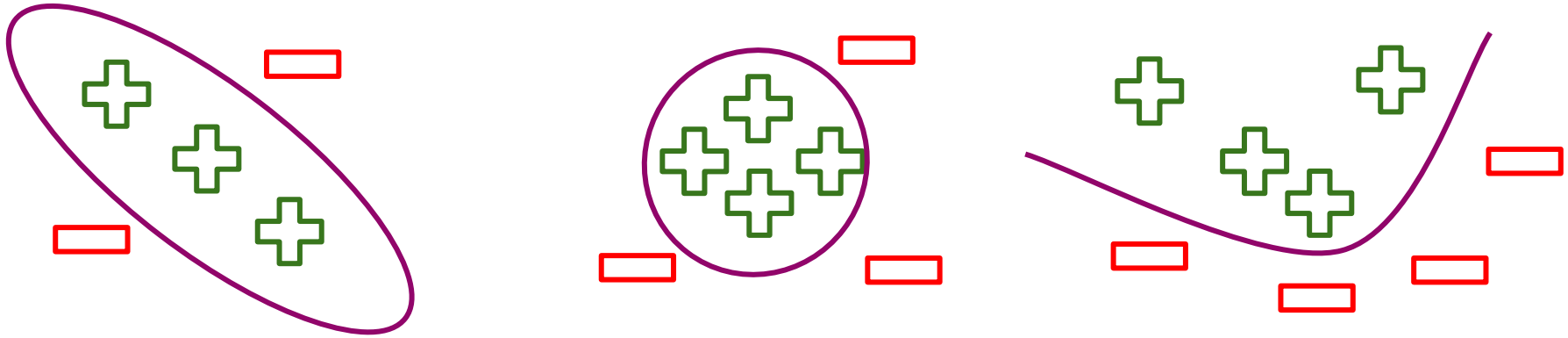
(1) Jointly learn a feature extractor / embedding function with the weights of the log-linear model:

$$\text{softmax}\left(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y)\right)$$

Neural networks joint learn f with θ

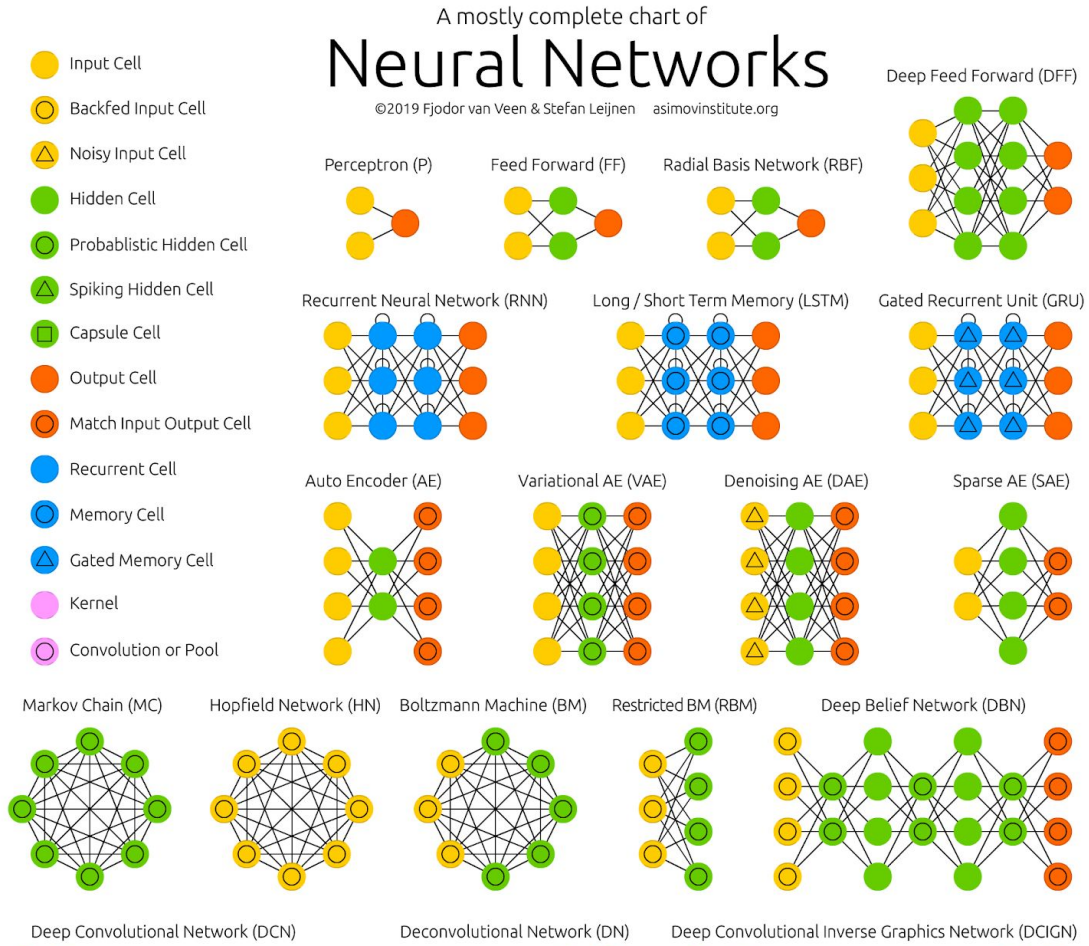
(2) Move from feature engineering to architecture engineering

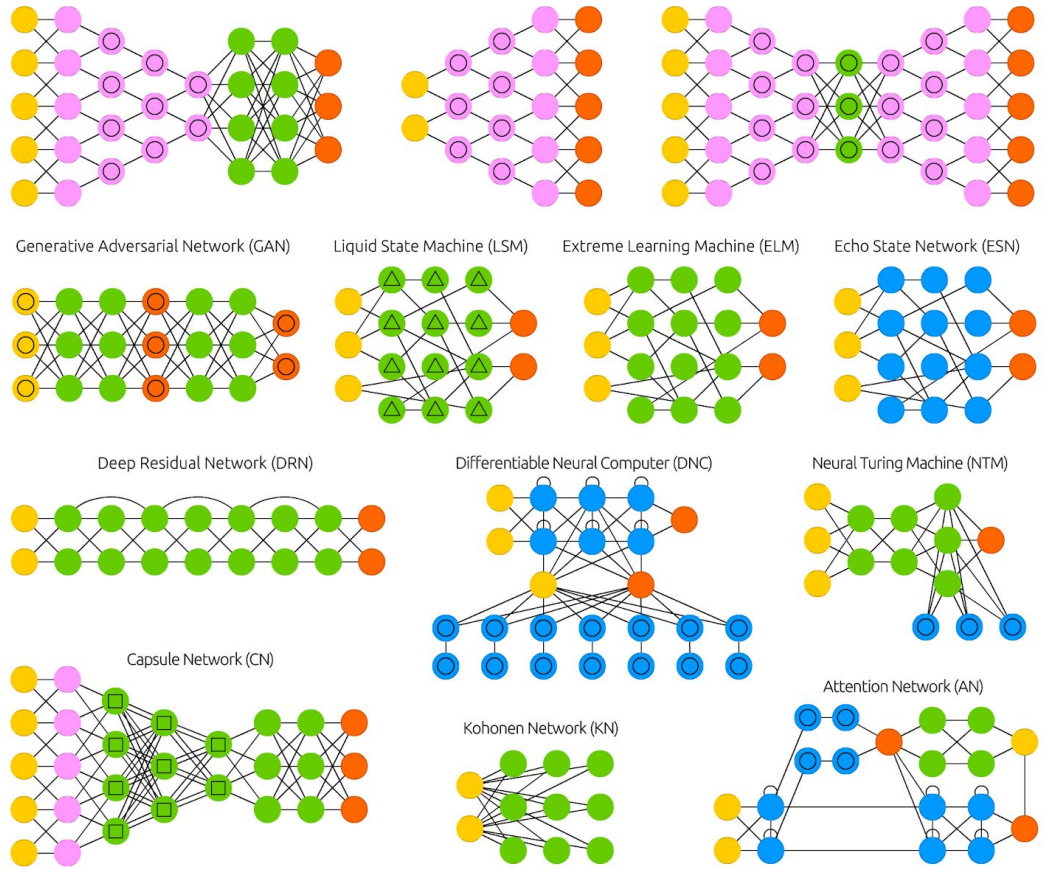
$$\mathbf{f}(\cdot, \cdot)$$



1 Multi-Layer Perceptrons

- NLP stopped engineering features and started engineering **feature extractors** (architectures)





https://www.asimovinstitute.org/neural-network-zoo/

1 What is a Multi-Layer Perceptron?

- One of the simplest neural architectures is the multi-layer perceptron (MLP)
- MLPs consist of alternating linear projections and non-linearities
- Let's break down the math of an MLP:

Each projection layer $W^{(i)}$ lives in $\mathbb{R}^{d_n \times d_{n-1}}$

Example: The final layer lives in $\mathbb{R}^{|\mathcal{Y}| \times d_N}$

Example: The first layer lives in $\mathbb{R}^{d_2 \times d_1}$

$$\mathbf{h}^{(N)} = \sigma(W^{(N)} \cdots \sigma(W^{(2)} \sigma(W^{(1)} \mathbf{e}(\mathbf{x}))))$$

Question: In which space does $\mathbf{h}^{(N)}$ live?

Answer: It lives in $\mathbb{R}^{|\mathcal{Y}|}$ and write h_y for the y^{th} component

The vector $\mathbf{e}(\mathbf{x}) \in \mathbb{R}^{d_1}$ encodes the input x.

1 What is a Multi-Layer Perceptron?

- One of the simplest neural architectures is the multi-layer perceptron
- MLPs consist of alternating linear projections and non-linearities
- Let's break down the math of an MLP:

Each projection layer $W^{(i)}$ lives in $\mathbb{R}^{d_n \times d_{n-1}}$

Example: The final layer lives in $\mathbb{R}^{|\mathcal{Y}| \times d_N}$

Example: The first layer lives in $\mathbb{R}^{d_2 \times d_1}$

$$\mathbf{h}^{(N)} = \sigma(W^{(N)} \cdots \sigma(W^{(2)} \sigma(W^{(1)} \mathbf{e}(\mathbf{x}))))$$

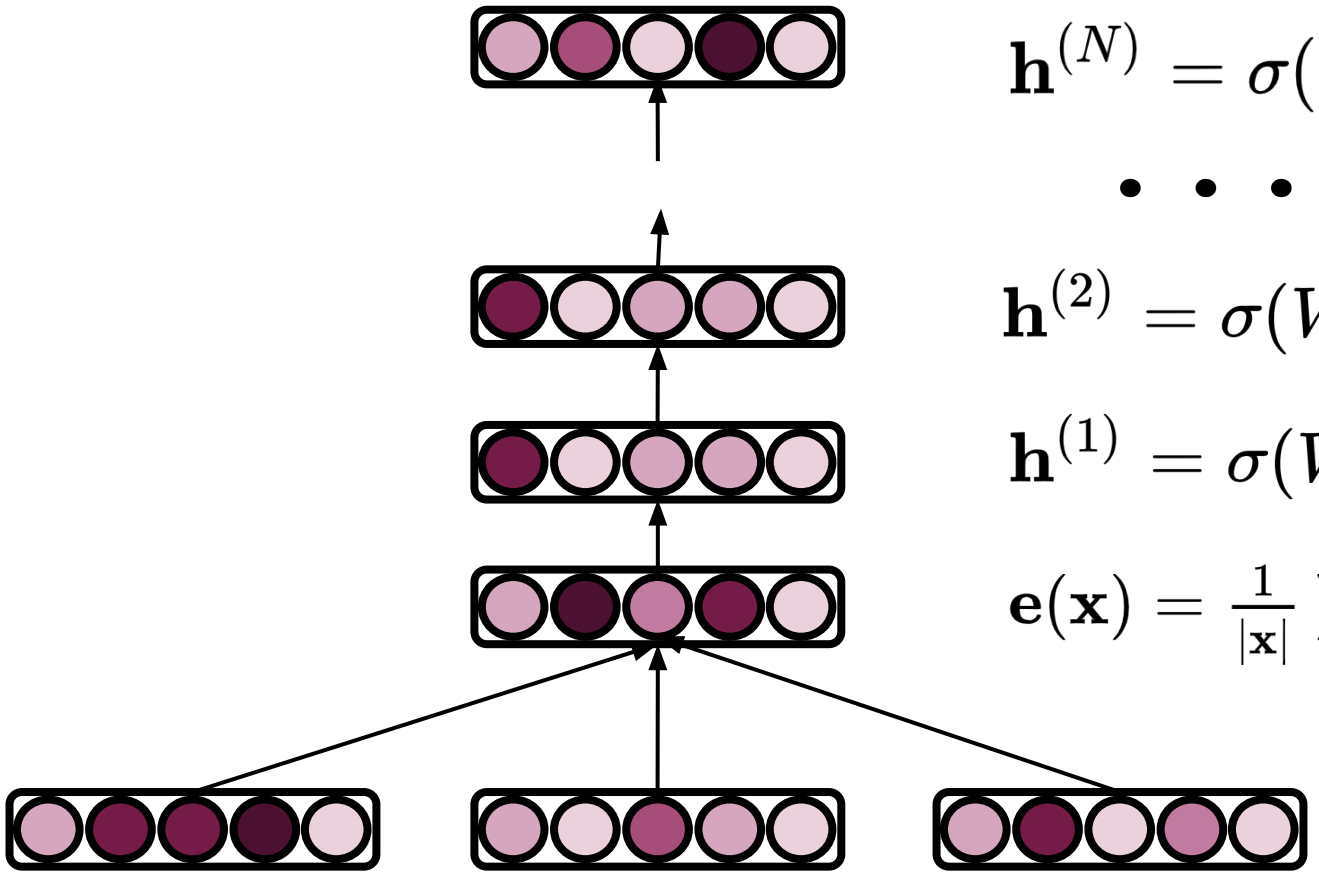
The vector $\mathbf{e}(\mathbf{x}) \in \mathbb{R}^{d_1}$ encodes the input *x*.

Putting It All Together: The Multi-layer Perceptron

$$p(y \mid \mathbf{x}) = \frac{\exp h_y}{\sum_{y' \in \mathcal{Y}} \exp h_y} = \text{softmax}(\mathbf{h}^{(N)}, y)$$

where we define $h_y^{(N)}$, i..e as the y^{th} component

1 Multi-Layer Perceptrons Pictorially



$$p(y \mid \mathbf{x}) = \frac{\exp h_y}{\sum_{y' \in \mathcal{Y}} \exp h_{y'}}$$

$$\mathbf{h}^{(N)} = \sigma(W^{(N)} \mathbf{h}^{(N-1)})$$

$$\bullet \quad \bullet \quad \bullet$$

$$\mathbf{h}^{(2)} = \sigma(W^{(2)} \mathbf{h}^{(1)})$$

$$\mathbf{h}^{(1)} = \sigma(W^{(1)} \mathbf{e}(\mathbf{x}))$$

$$\mathbf{e}(\mathbf{x}) = \frac{1}{|\mathbf{x}|} \sum_{w \in \mathbf{x}} \mathbf{e}(w)$$

$$\mathbf{e}(w) \in \mathbb{R}^d, \quad \forall w \in \mathbf{x}$$

Recall the subscript *y* indexes into the vector **h**

We will discuss how to construct **e**(*w*) next section

1 Multi-Layer Perceptrons Verbally

- A log-linear model (lecture #3) where we learn the feature function **f**
  - Because we learn both **θ** and **f**, the log-likelihood is no longer convex
  - Traditionally, **f** is constructed through alternating linear projections and non-linearities

- The final layer is generally a softmax (lecture #3)
  - If we use a Gaussian final layer, we get something closer to linear regression

1 Training a Multi-Layer Perceptron

- How do we train a multi-layer perceptron?
  - We maximize the log-likelihood of the training data with a gradient-based method!
  - Just like in a log-linear model
- How do compute the gradient?
  - Automatically with backpropagation!
  - This is why we spent all of lecture #2 on backpropagation
  - The previous slides tell you how to forward-propagate:

$$\mathbf{h}^{(N)} = \sigma(W^{(N)} \cdots \sigma(W^{(2)} \sigma(W^{(1)} \mathbf{e}(\mathbf{x}))))$$

  - This is super easy to write down in MATLAB or numpy or pytorch!
  - No pesky more calculus by hand 
- What is the “final layer” of a multi-layer perceptron?
  - A softmax, just like in lecure #3

1 Digression: What is a perceptron anyway?

What is a perceptron?

- The perceptron is like a happy meal:
  - It's a lot of things in one and it comes as ***a package deal***
- Specifically, the perceptron (Rosenblatt 1958) is
  - A log-linear model like we discussed in Lecture #3
    - as the temperature $T \to \infty$ (recall this means the softmax becomes a true max)
    - trained with stochastic gradient descent with a minibatch size of 1 (called the **perceptron update rule**)

- **The TL;DR**: The perceptron is a log-linear model with a specific setting of the hyperparameters
  - Fancy name for a special case of a log-linear model!

Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

F. ROSENBLATT

Cornell Aeronautical Laboratory

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

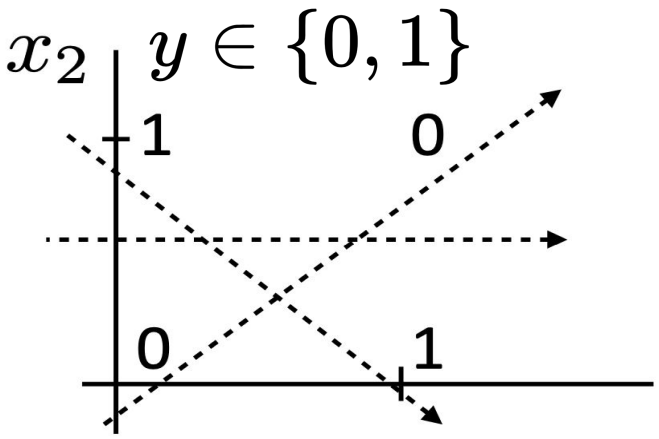
1. How is information about the

and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have

1 Multi-Layer Perceptrons and the XOR Problem

The XOR-problem

x_1	x_2	$y = x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0



Not linearly separable!

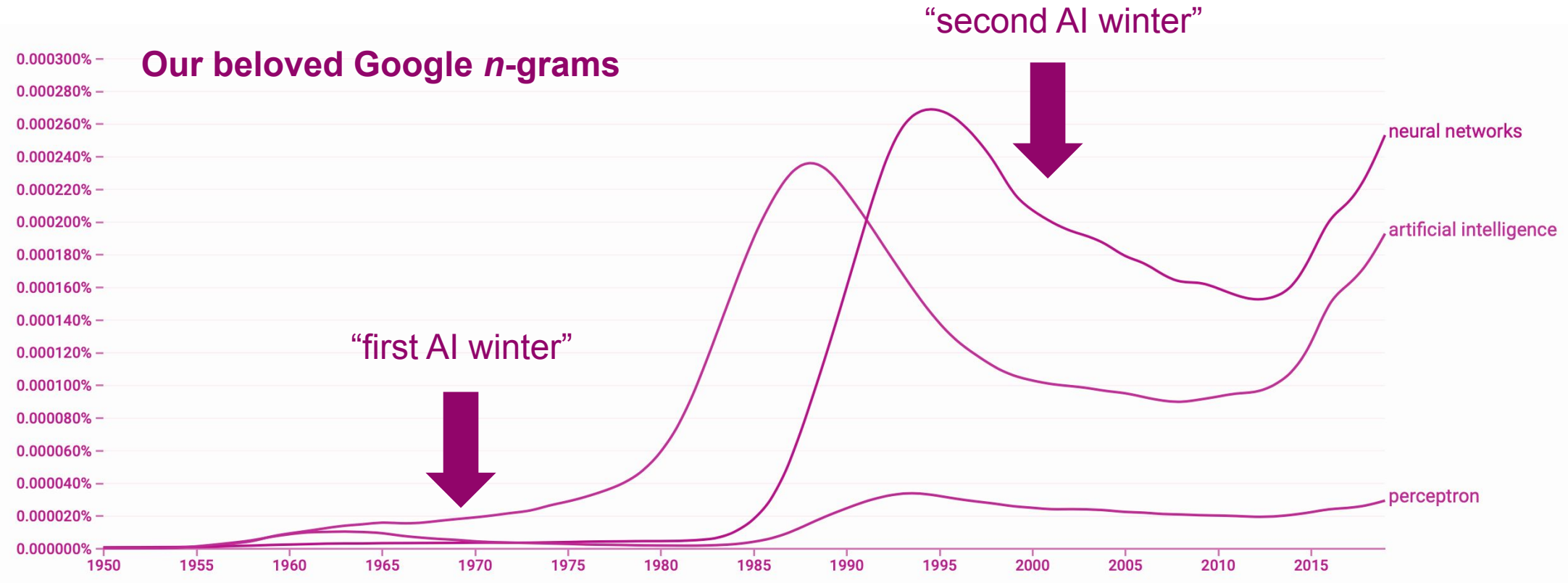
$$y = \alpha_1 \times x_1 + \alpha_2 \times x_2$$

Key Book: Perceptrons: An Introduction to Computational Geometry (Minsky and Papert, 1969)

- A (single-layer) perceptron cannot solve the XOR problem
- This contributed to the first AI winter resulting in funding cuts for neural networks

Multi-Layer Perceptrons

Solving the XOR-problem

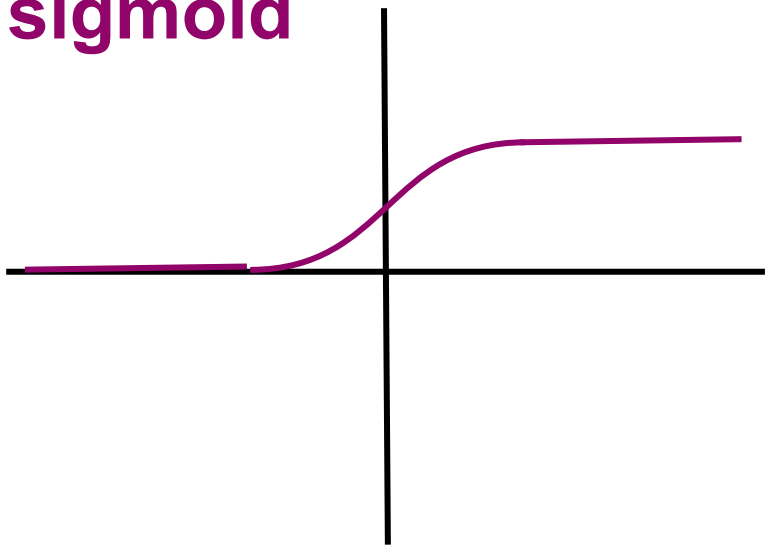


→ How about multi-layer perceptrons, may they solve the problem?

https://books.google.com/ngrams

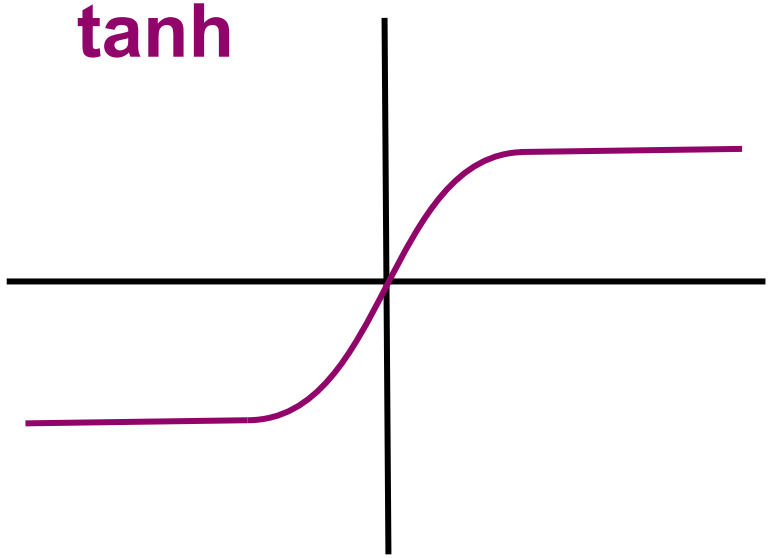
1 Common Non-Linearities Used in Multi-Layer Perceptrons

sigmoid



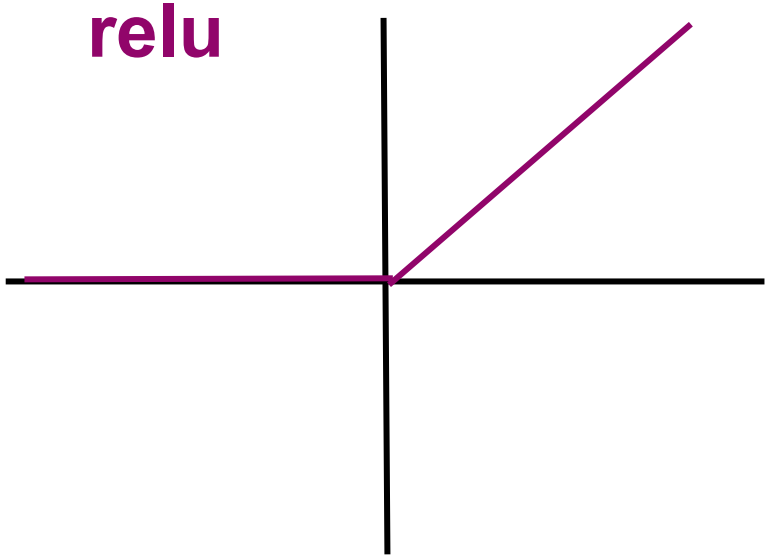
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

tanh



$$\tanh(x) = \sigma(x)(2 \times x) - 1$$

relu



$$\text{relu}(x) = \max(0, x)$$

- A sigmoid function is function whose graph is an s-shaped curve
- The tanh (hyperbolic tangent) function is simply a scaled and shifted version of the sigmoid function
- The relu function computationally cheaper

1 Why does tanh help with the XOR problem?

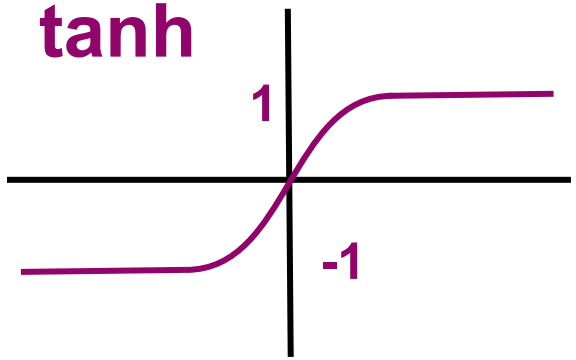
The XOR-problem

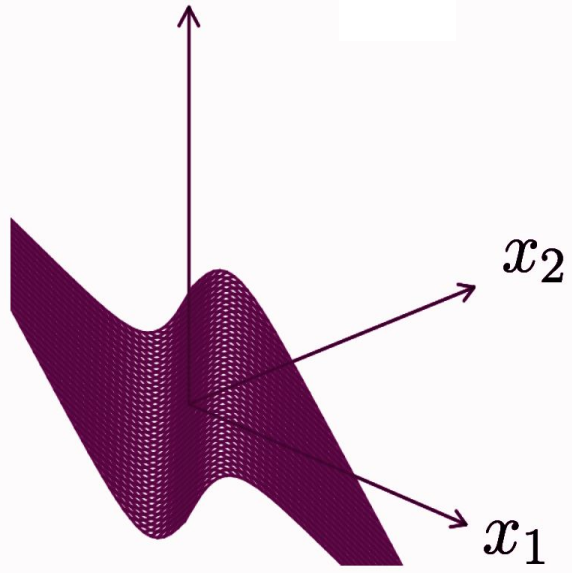
x_1	x_2	$y = x_1$ XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

$$y = \theta_1 \times x_1 + \theta_2 \times x_2 + \theta_3 \times \tanh(x_1 + x_2)$$

$$y = (-1) \times x_1 + (-1) \times x_2 + (+2) \times \tanh(x_1 + x_2)$$

$$\lceil (-1) \times 0 + (-1) \times 0 + (+2) \times \tanh(0 + 0) \rceil \approx 0$$
$$\lceil (-1) \times 0 + (-1) \times 1 + (+2) \times \tanh(0 + 1) \rceil \approx 1$$
$$\lceil (-1) \times 1 + (-1) \times 0 + (+2) \times \tanh(1 + 0) \rceil \approx 1$$
$$\lceil (-1) \times 1 + (-1) \times 1 + (+2) \times \tanh(1 + 1) \rceil \approx 0$$





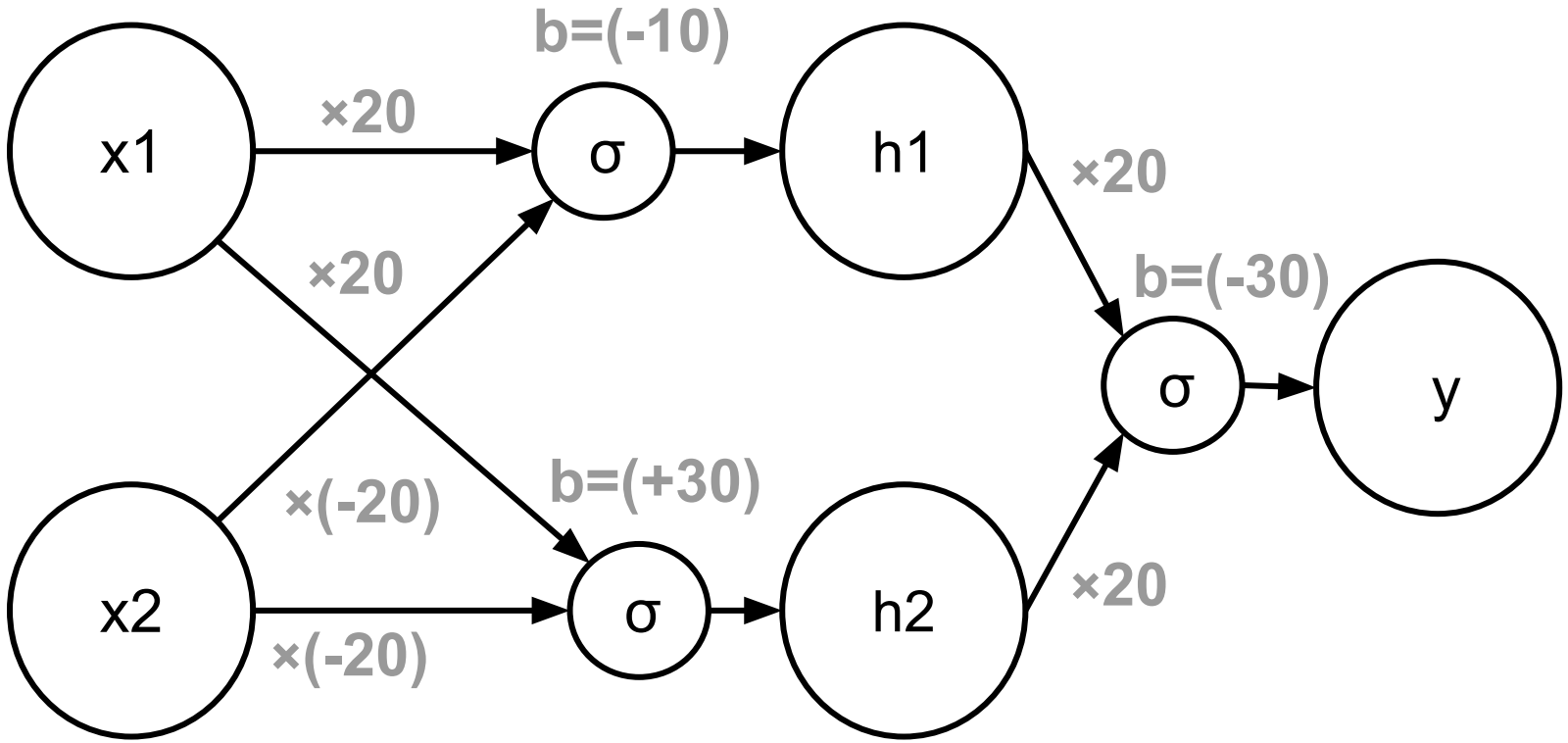
1 Why does sigmoid help with the XOR problem?

The XOR-problem

x_1	x_2	$y = x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

h1 = σ((x$_1$ × 20) + (x$_2$ × 20) - 10)
h2 = σ((x$_1$ × -20) + (x$_2$ × -20) + 30)
y = σ((h$_1$ × 20) + (h$_2$ × 20) - 30)

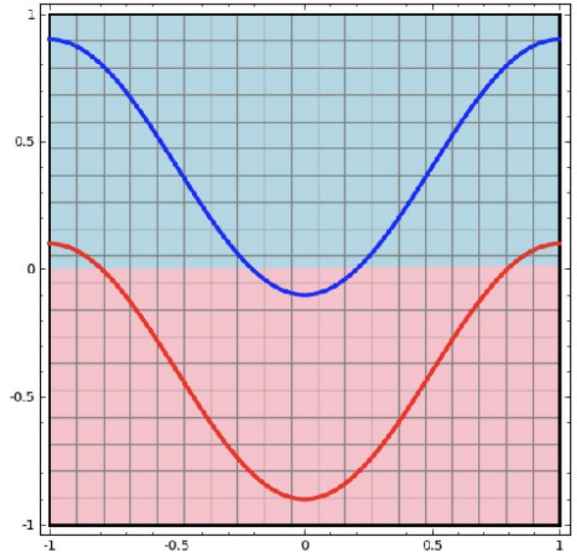
...another way of solving it with sigmoid



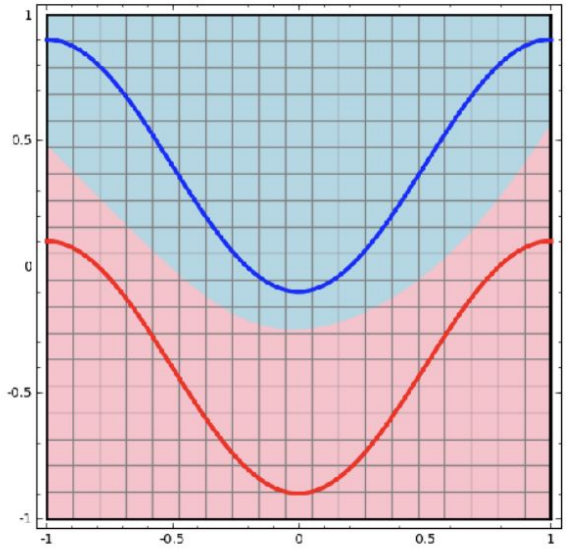
https://www.youtube.com/watch?v=kNPGXgzxoHw

1 Why do Multi-layer Perceptrons Work so well?

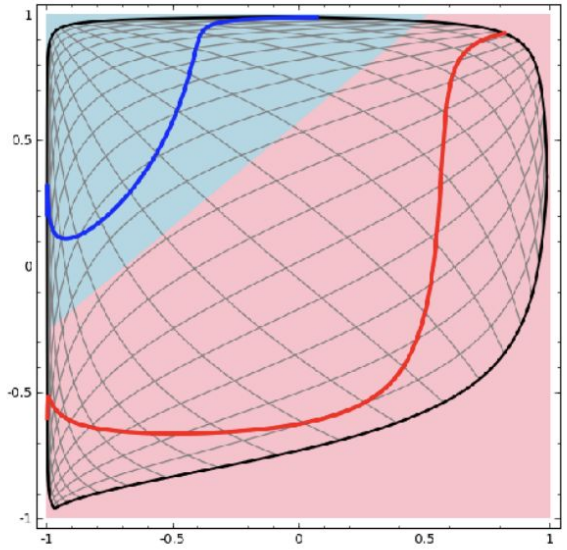
Linear classifier



Neural network



…possible because we transformed the space!



http://colab.github.io/posts/2014-03-NN-Manifolds-Topology/

1 Representational Capacity of Multi-layer Perceptrons

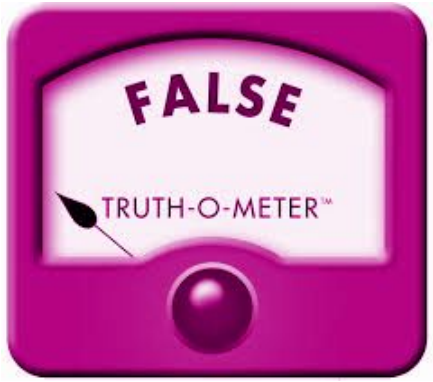
- Why should we consider the multi-layer perceptron family of functions?
- It may seem arbitrary; however, MLPs are universal approximators

- The MLP universal approximation theorem states that an MLP with single hidden layer and a sigmoidal activation function can represent any function in the unit cube
  - **Huge caveat**: you may need an arbitrary large number of hidden states
  - The result is due to George Cybenko (1989)
- The theorem has been further generalized to other non-linearities

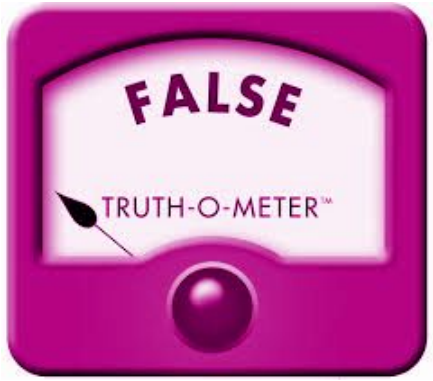
- **TL;DR:** MLPs can represent almost any any function you would want!

1 Fake News Myths Dispensed With

- **Claim:** Log-linear models can't model non-linear decision boundaries
  - It's easy! Just hack the nonlinearity into the feature function
    - Just like the Gaussian example from lecture #3
  - Neural networks are popular because we don't know *a-priori* which non-linearity we need, not because they are non-linear



- **Claim:** Multilayer perceptrons are just stacked perceptrons
  - Multi-layer perceptrons generally use a softmax (not a max)
  - Are not specifically trained with the perceptron update rule
  - If you stack linear models, you get back a linear model!
    - Need non-linearities!



ETH zürich

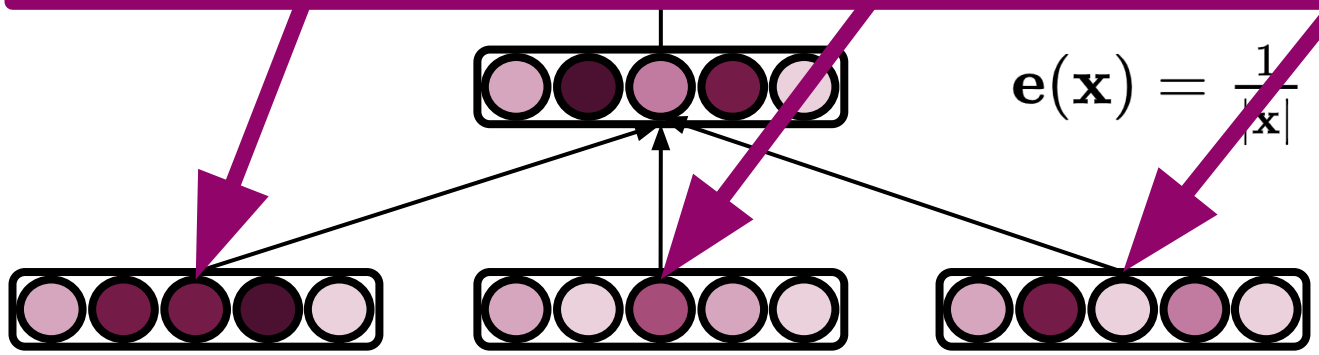
How do we Feed Language into an MLP?

A Hitchhiker’s Guide to Word Embeddings!

2 Multi-Layer Perceptrons Pictorially

softmax $p(y \mid \mathbf{x}) = \frac{\exp \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y)}{\sum_{y' \in \mathcal{Y}} \exp \boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y')}$ where we define $\mathbf{f}(\mathbf{x}, y) := h_y^{(N)}$

How do we encode words to feed into a MLP, i.e. how do we construct e(*w*)?



$\mathbf{e}(\mathbf{x}) = \frac{1}{|\mathbf{x}|} \sum_{w \in \mathbf{x}} \mathbf{e}(w)$

$\mathbf{e}(w) \in \mathbb{R}^d, \quad \forall w \in \mathbf{x}$

nlp *is* *interesting*

2 Encoding Words: One-hot Encoding

- **Preprocessing**
  - Tokenization
  - Stemming
  - Stop Word / Punctuation Removal
  - ...

NLP is interesting and cool!

↓

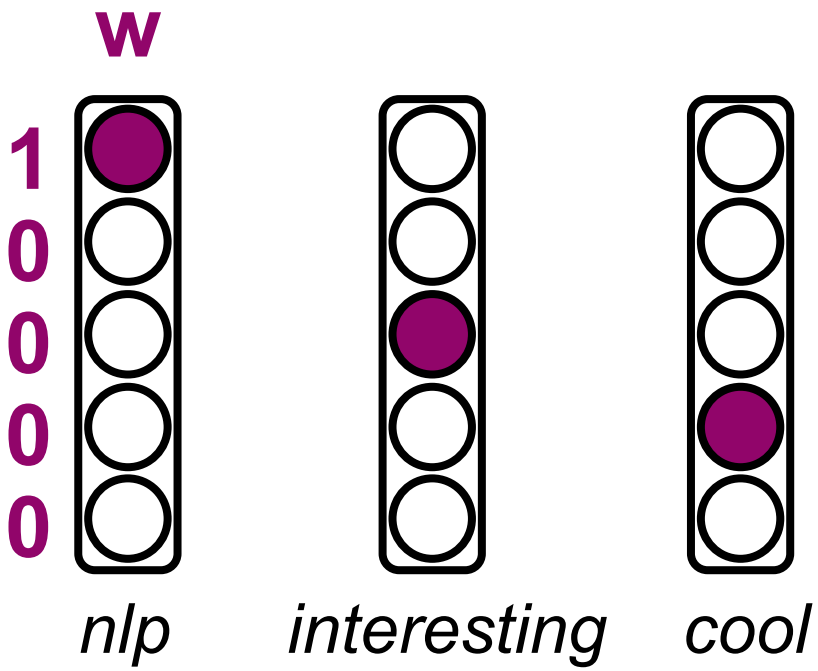
nlp interesting cool

- **One-hot Encoding**

Vocabulary

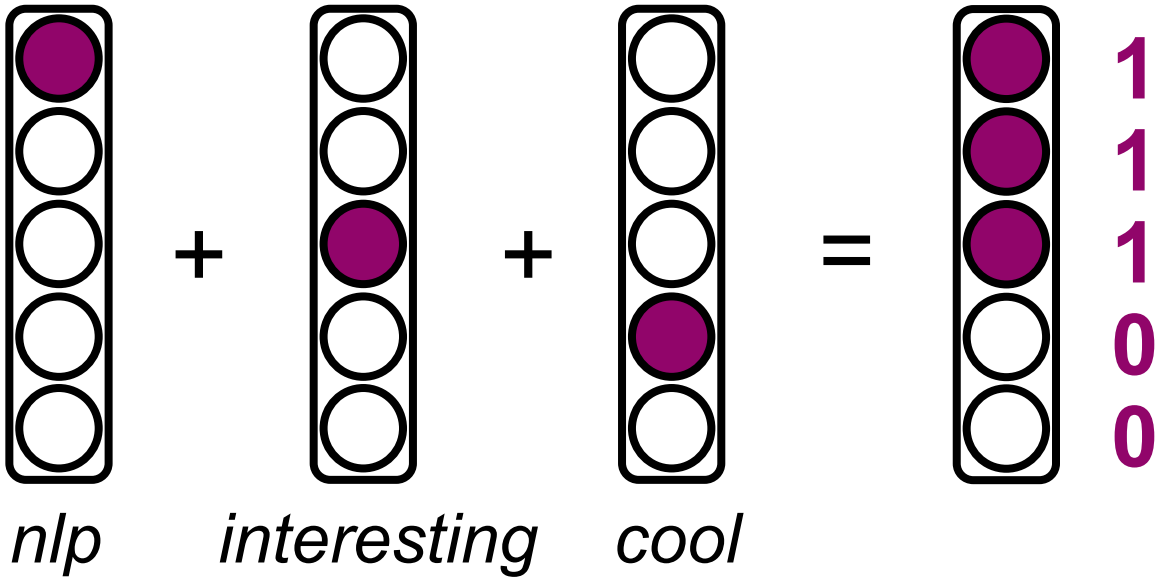
V =	*alphabet*	1
	apple	2
	.	.
	.	.
	.	.
	zebra	m

$$\mathbf{e}(w) \in \mathbb{R}^d$$



2 How do we Combine the One-hot Embeddings?

Bag of Words

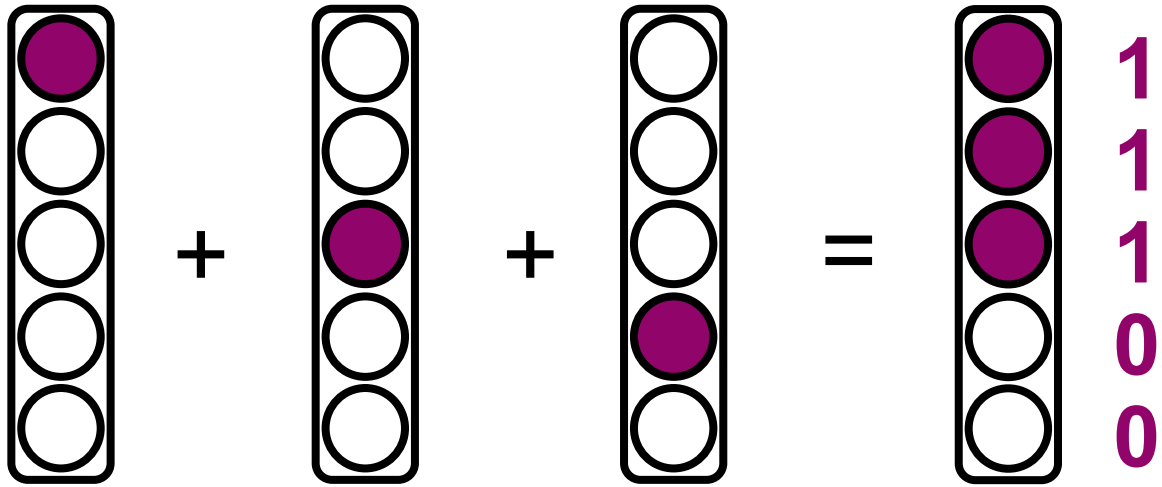


→ we could sum up the one-hot embeddings of word vectors

- This (sentence-level) aggregation of one-hot embeddings is referred to as: **Pooling**
  - **... what other forms of pooling can you think of?**

2 How do we Combine the One-hot Embeddings?

Bag of Words



1	1/3	1
1	1/3	1
1	1/3	1
0	0	0
0	0	0

pooling:

sum pooling: $\mathbf{e}(x) = \sum_{w \in x} \mathbf{e}(w)$

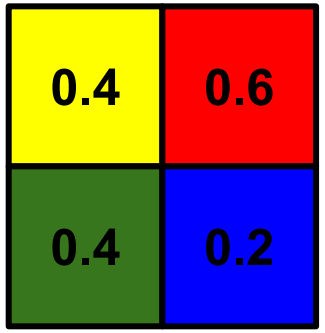
mean pooling: $\mathbf{e}(x) = \frac{1}{|x|} \sum_{w \in x} \mathbf{e}(w)$

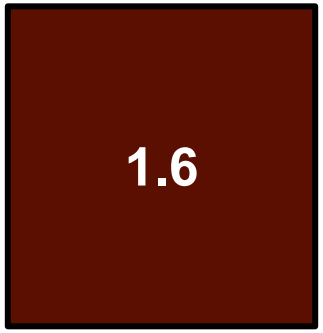
max pooling: $\mathbf{e}(x) = \max_{w \in x} (\mathbf{e}(w))$

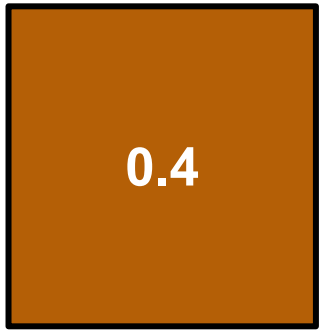
2 How do we Combine the One-hot Embeddings?

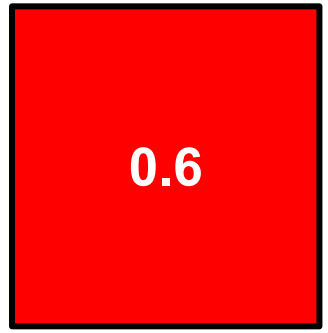
Bag of Words

Comparison with computer vision





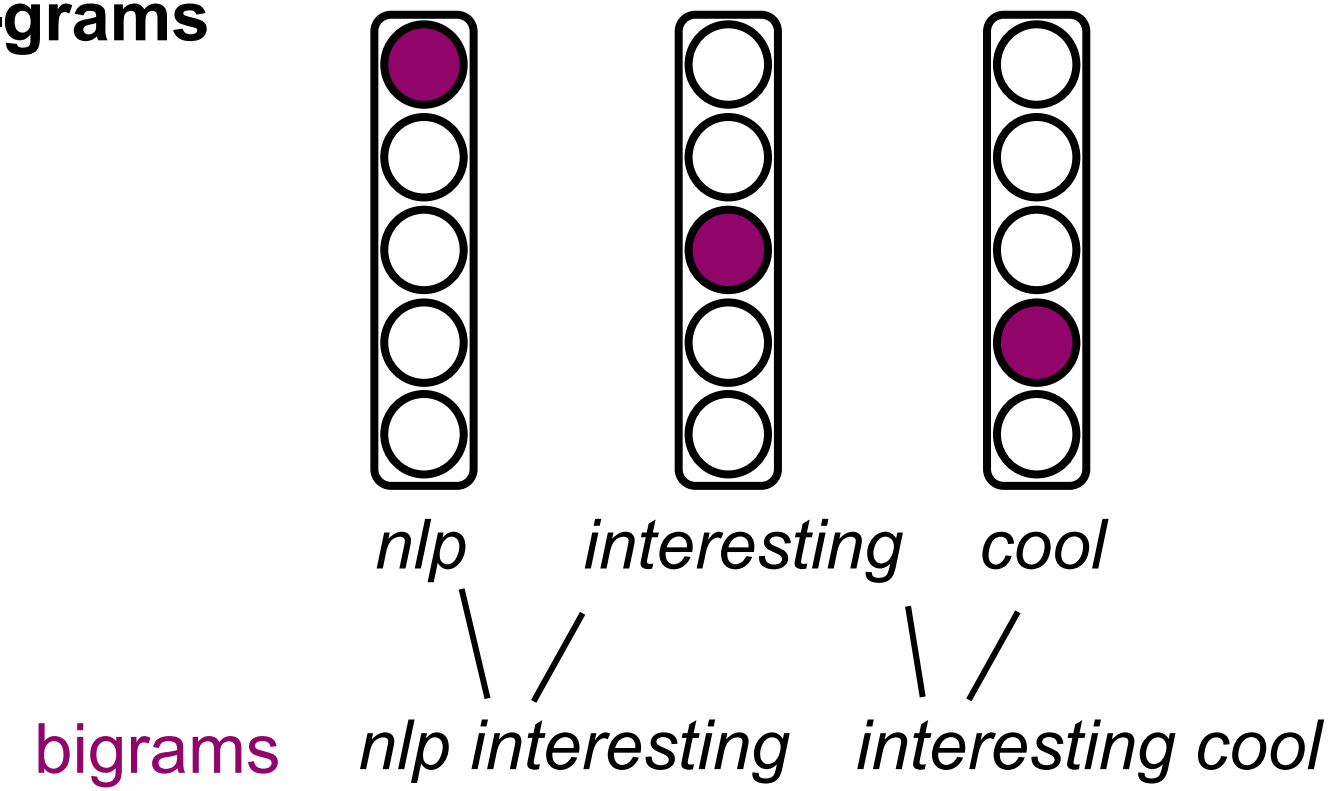




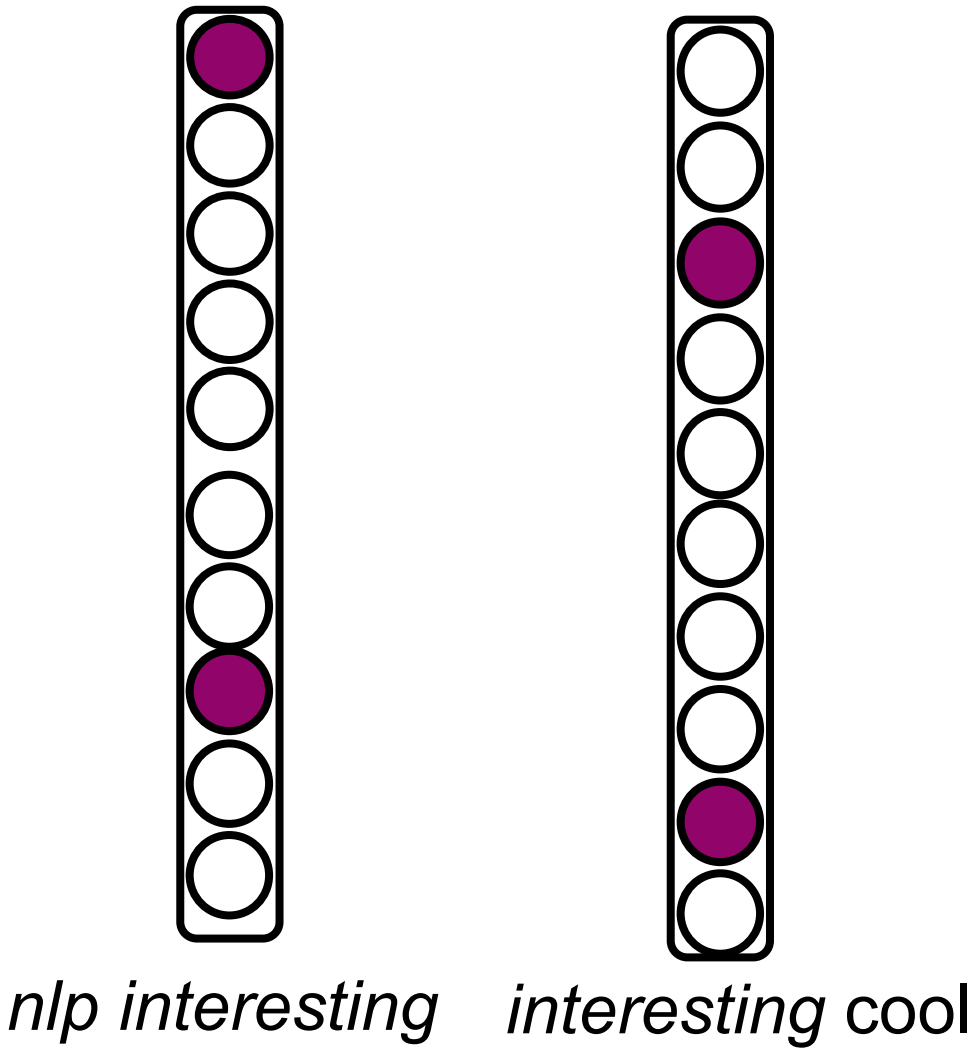
	sum pooling	mean pooling	max pooling
pooling:	$\mathbf{e}(x) = \sum_{w \in x} \mathbf{e}(w)$	$\mathbf{e}(x) = \frac{1}{\lvert x \rvert} \sum_{w \in x} \mathbf{e}(w)$	$\mathbf{e}(x) = \max_{w \in x} (\mathbf{e}(w))$

2 We could also encode bigrams in a one-hot way?

***n*-grams**



- **Explicit conjunction:** these vectors have length of $O(|V|^2)$ instead of $O(|V|)$



2 A Great Idea in NLP: Unsupervised Word Representations

- We just talked about training MLPs
  - Implicit in this discussion was the fact that this was for a *supervised* task

- To train a supervised model, we need lots of annotated training data
  - Annotating data is expensive and may require domain experts

- But, unlabeled text abounds on the internet! Can we exploit it to train better models?

2 A Great Idea in NLP: Unsupervised Word Representations

We use these word representations in supervised NLP tasks where we have fewer data!

Unannotated Text on the Internet

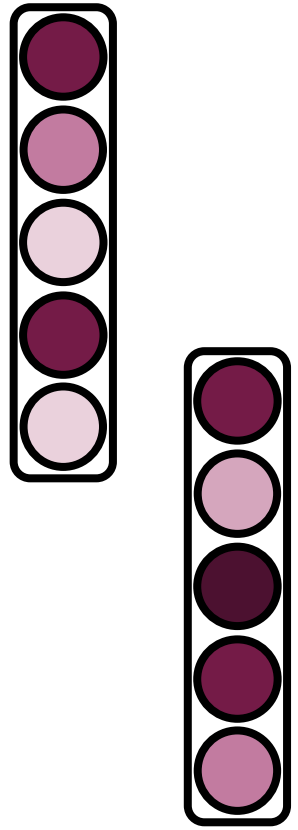
WIKIPEDIA
The Free Encyclopedia

Quora

reddit

Word-Embedder

Real-Valued Word Representations

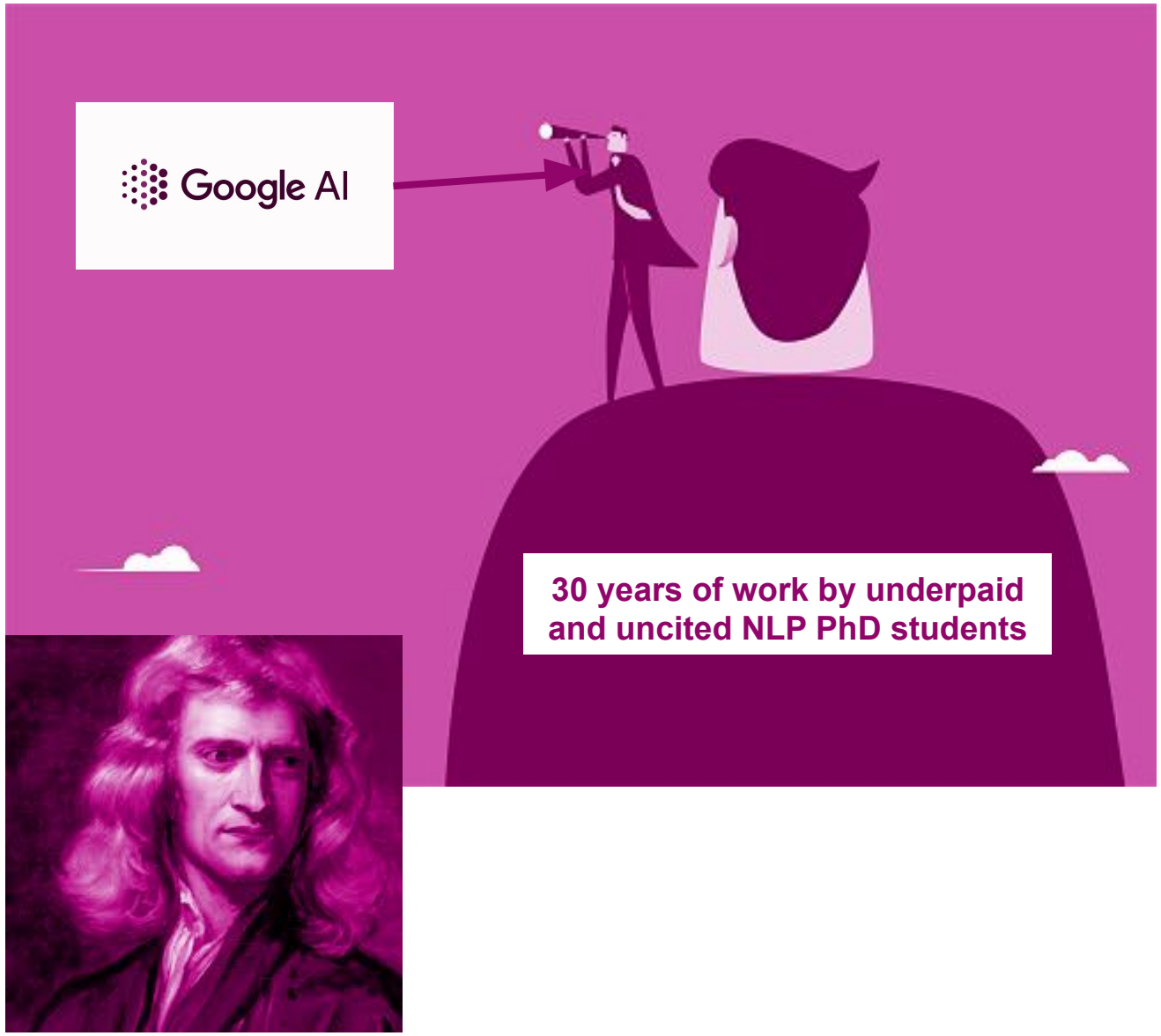


2 A Marxist History of Unsupervised Word Representations

- The general idea of deriving reusable word representations from unlabeled text is old
  - Salton (1975) for information retrieval
  - Deerwester (1990) for word vectors
- The latest incarnations of this idea, e.g. BERT are just variations on this theme.

Contrasting Ideologies:

- Isaac Newton: *If I have seen further it is by standing on the shoulders of Giants*.
- Google: ~~*Don't be evil*~~.



2 The Skip-Gram Model

- This lecture is going to go into a relative deep dive on ***one*** word-embedder
  - The skip-gram model, part of Google's word2vec toolkit (Mikolov et al. 2013)
  - Popular variants include fastText (Bojanowski et al. 2017)
- Skip-gram is a very famous and influential model (≈ 20k individual citations)
  - For the first two years of my PhD, it was all people talked about 🤦
- The embeddings are ***not contextual***, e.g. every word type gets a single representation

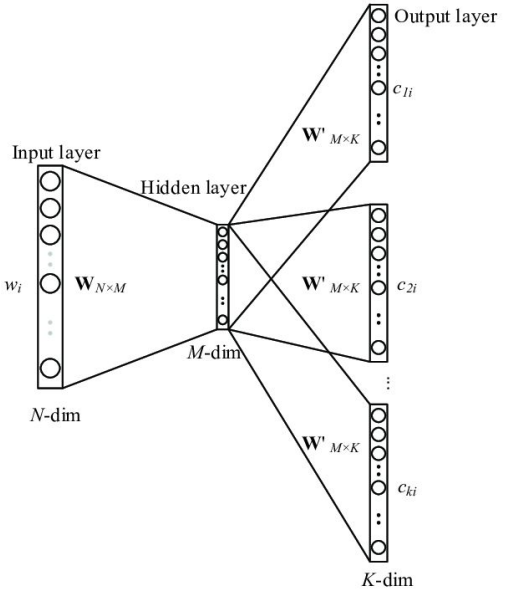
Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com



2 Skip-Gram: Pre-processing

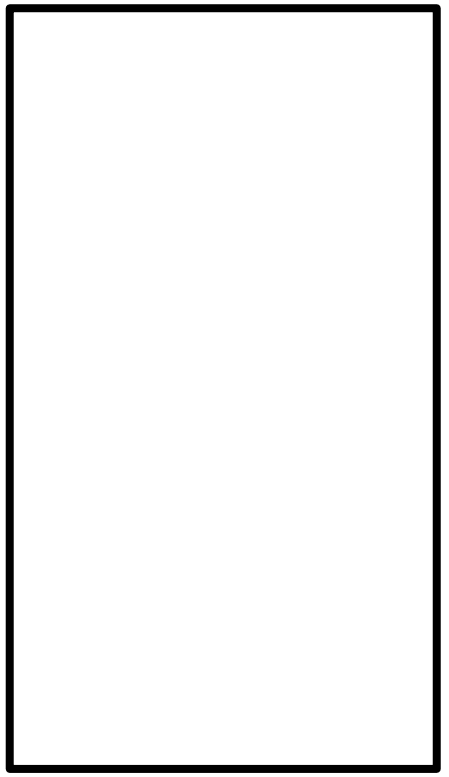
- The most important part about understanding skip-gram is the pre-processing step
- The general idea is that we seek to create word–context pairs for a corpus using a window
  - This is best seen through animation
- The input to the skip-gram model is a large unannotated corpus
  - the corpus is generally sentencized and tokenized
- For every sentence in the corpus, we perform a simple linear-time pre-processing step
- Consider, for example, the sentence below

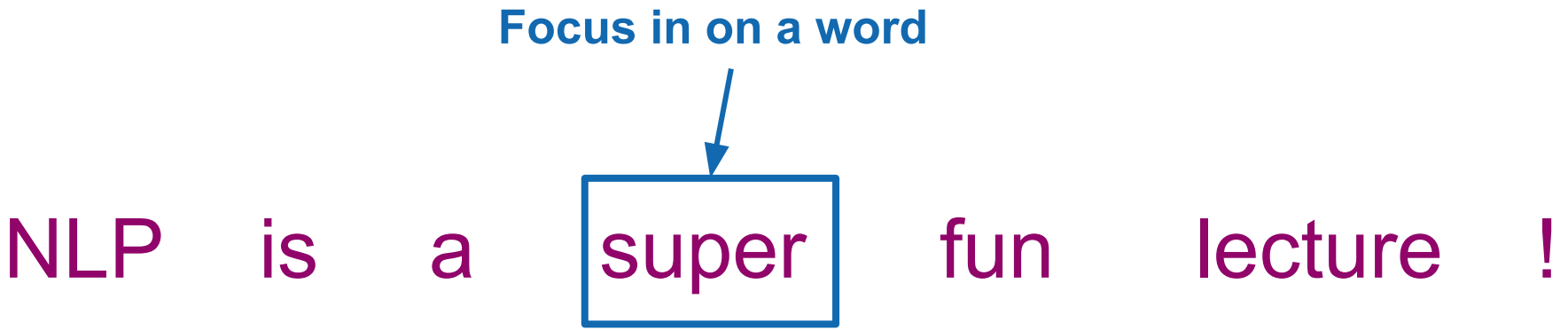
NLP is a super fun lecture !

2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

Accumulated Training Data





2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

Accumulated Training Data

(super, NLP)

And Find its Contexts!

NLP is a super fun lecture !

2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

Accumulated Training Data

(super, NLP)
(super, is)

And Find its Contexts!

NLP **is** a **super** fun lecture !

2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

Accumulated Training Data

(super, NLP)
(super, is)
(super, a)

And Find its Contexts!

NLP is a super fun lecture !

2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

Accumulated Training Data

(super, NLP)
(super, is)
(super, a)
(super, fun)

And Find its Contexts!

NLP is a super fun lecture !

2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

Accumulated Training Data

(super, NLP)
(super, is)
(super, a)
(super, fun)
(super, lecture)

And Find its Contexts!

NLP is a **super** fun **lecture** !

2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

And Find its Contexts!

NLP is a **super** fun lecture **!**

Accumulated Training Data

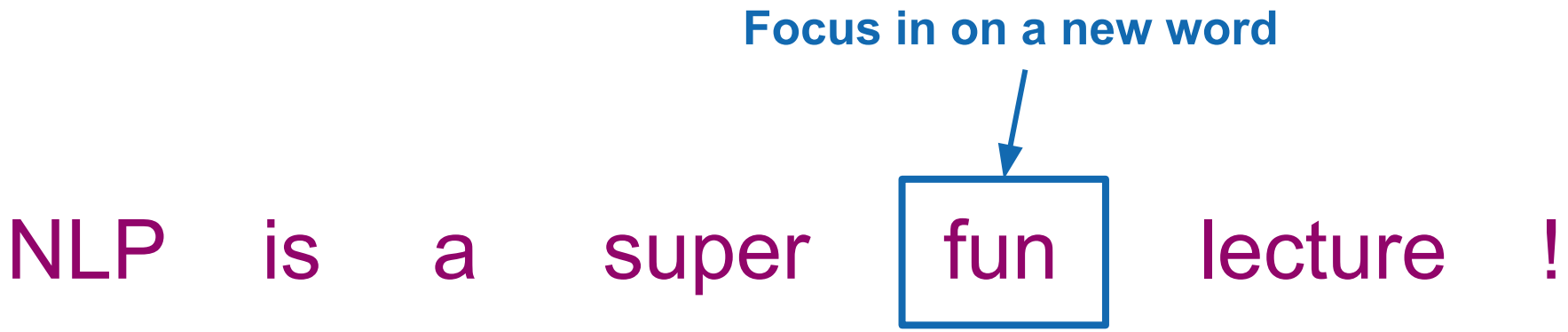
(super, NLP)
(super, is)
(super, a)
(super, fun)
(super, lecture)
(super, !)

2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

Accumulated Training Data

(super, NLP)
(super, is)
(super, a)
(super, fun)
(super, lecture)
(super, !)



2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

And find its contexts!

NLP **is** a super **fun** lecture !

Accumulated Training Data

(super, NLP)
(super, is)
(super, a)
(super, fun)
(super, lecture)
(super, !)
(fun, is)

2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

And find its contexts!

NLP is a super fun lecture !

Accumulated Training Data

(super, NLP)
(super, is)
(super, a)
(super, fun)
(super, lecture)
(super, !)
(fun, is)
(fun, a)

2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

And find its contexts!

NLP is a **super** **fun** lecture !

Accumulated Training Data

(super, NLP)
(super, is)
(super, a)
(super, fun)
(super, lecture)
(super, !)
(fun, is)
(fun, a)
(fun, super)

2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

And find its contexts!

NLP is a super **fun** **lecture** !

Accumulated Training Data

(super, NLP)
(super, is)
(super, a)
(super, fun)
(super, lecture)
(super, !)
(fun, is)
(fun, a)
(fun, super)
(fun, lecture)

2 Skip-Gram: Pre-processing

- Skip-gram pre-processing with a window of size 3
- We consider both the left and the right context of every word

And find its contexts!

NLP is a super **fun** lecture **!**

Accumulated Training Data

(super, NLP)
(super, is)
(super, a)
(super, fun)
(super, lecture)
(super, !)
(fun, is)
(fun, a)
(fun, super)
(fun, lecture)
(fun, !)

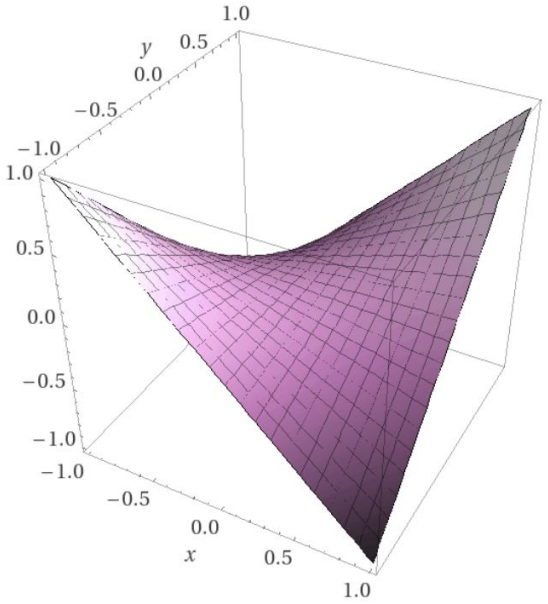
2 Skip-Gram: The Result of the Pre-Processing

- We perform the algorithmic procedure animated in the previous slide until we create a collection of training data

- The result of the pre-processing is a collection of N word-context pairs
  - For a window size of k, we have N is $O(k \cdot C)$ where C is the corpus size
  - Extraction algorithm runs in $O(k \cdot C)$ time

- Besides the window parameter k and whether we consider left or right context
  - We may also may exclude stop words
  - Dampen the frequency of certain words, i.e. drop pairs containing one or two frequent words sometimes
  - Lemmatize or stem the words

2 Skip-Gram: The Model



- We taught you log-linear models; skip-gram is just a log-**bi**linear model
- What's a bilinear function? The simplest example is f(x, y) = xy
  - Clearly, it's non-linear due to curves: See graph →

- **Simple definition**: linear in every variable with the others held constant:
  - For instance, if we hold x constant, then f(x, y) is linear in y
  - Likewise, if we hold y constant, then f(x, y) is linear in x
  - Hence, bi-linear!

- **Quiz question**: Are bilinear functions convex in general?

2 Skip-Gram: The Model

- The general idea of skip-gram is that we model p(w | c) as a log-bilinear model:

$$p(w \mid c) = \frac{1}{Z(c)} \exp(\mathbf{e}_{\text{wrd}}(w) \cdot \mathbf{e}_{\text{ctx}}(c))$$

- We have $\mathbf{e}_{\text{wrd}}(w) \in \mathbb{R}^d$ as a learnable embedding for the word *w* and $\mathbf{e}_{\text{ctx}}(c) \in \mathbb{R}^d$ as a learnable embedding for the context word *c*
  - Note that, in principle, every word can appear as a “word” and a “context”

- So, if we have |*V*| words, then we have 2|*V*|*d* parameters
- Again, *Z*(*c*) is the normalizing constant
  - Exactly the same as the log-linear model

2 Skip-Gram: Estimation

- The training objective is to maximize the log-likelihood of the pre-processed corpus:

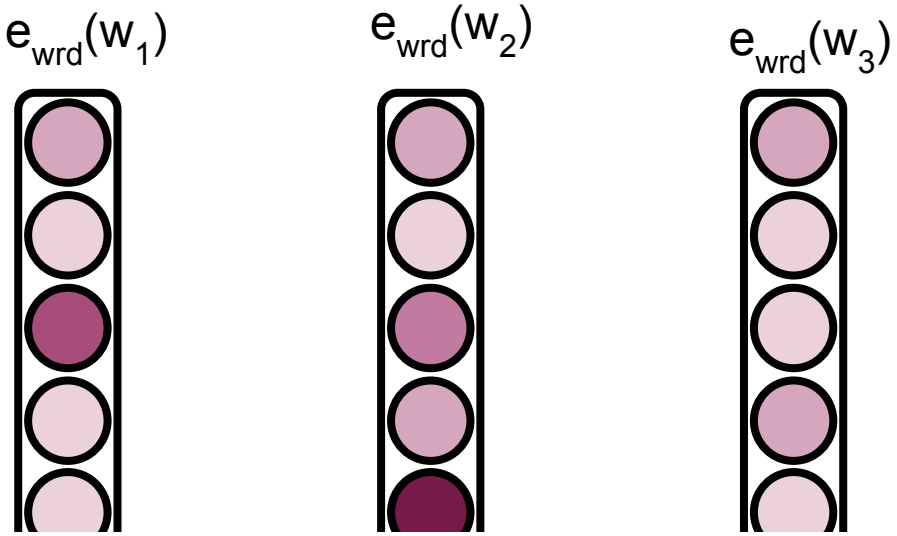
$$\sum_{n=1}^{N} \log p(w^{(n)} \mid c^{(n)}) = \sum_{n=1}^{N} \left(\mathbf{e}_{\text{wrd}}(w^{(n)}) \cdot \mathbf{e}_{\text{ctx}}(c^{(n)}) - \log Z(c^{(n)}) \right)$$

- We compute the gradient with respect to $e_{wrd}(w)$ and $e_{ctx}(w)$ using backpropagation!
  - You know it now, so that's all I have to say 😀
- Run your choice of gradient-based optimizer to learn the embeddings

- Mikolov et al. (2013) found that computing log $Z(c)$ was a bit slow, so they introduced an efficiency hack called **negative-sampling**
  - There are more principled techniques that do the same thing, e.g. importance sampling or noise-contrastive estimation if computation time is an issue for you

2 Skip-Gram: the Output

- The result of training skip-gram is ***two*** collections of word embeddings: $\{\mathbf{e}_{\text{wrd}}(w)\}_{w \in V}$, $\{\mathbf{e}_{\text{ctx}}(w)\}_{w \in V}$
  - Recall that we use the notation *V* for the set of word types in the corpus
- Typically, we either:
  - Throw away the context embeddings $\{\mathbf{e}_{\text{ctx}}(w)\}_{w \in V}$
  - Or, concatenate them to form embeddings of dimensionality 2*d*
- Either way, now we have non-one-hot representations of the words:

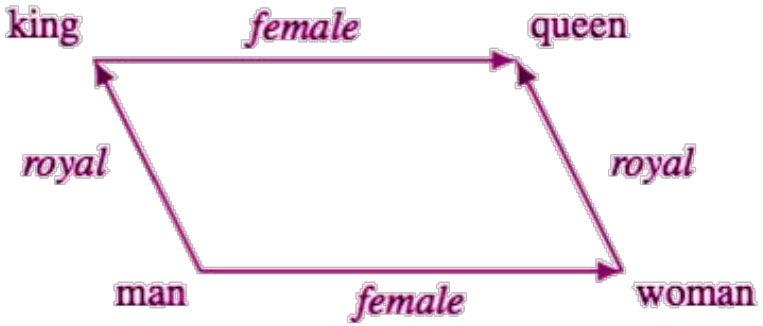


2 How do we Evaluate Word Embeddings?

- **Cosine similarity**
  A basic question for word embeddings is whether the similarity of words *i* and *j* is reflected in the similarity of the vectors u_i and u_j:

$$\cos(u_i, u_j) = \frac{u_i \times u_j}{||u_i||_2 \times ||u_j||_2}$$

- **Word analogies** (e.g., *king* : *queen* :: *man* : *woman*) have also been used to evaluate word embeddings
  - In this evaluation, the system is provided with the first three parts of the analogy ($i_1 : j_1 :: i_2 : ?$), and the final element is predicted by finding the word embedding most similar to $u_{i1} - u_{j1} + u_{i2}$



from Eisenstein, Introduction to Natural Language Processing, Ch. 14
picture from https://towardsdatascience.com/the-magic-behind-embedding-models-c3af62f71fb

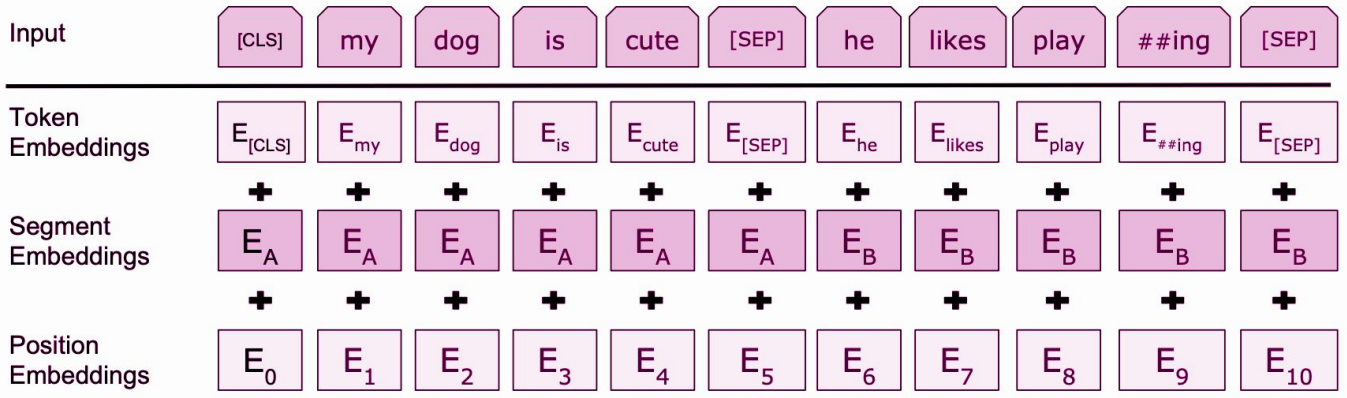
2 What about fastText, BERT and ELMo?

- There are many other word-embedders “on the market”
  - Some of these generalize skip-gram, e.g. fastText
  - Some of them make sure of more complicated neural networks to handle context
- **Important Sociological Note:** These architectures are ephemeral:
  - The “best” architecture at the moment is one of the BERTs
    - The NLP literature created ALBERT, RoBERTa, etc…
    - I can’t say I remember...
  - In a year’s time, it will be something different
- The only constant is architecture engineering is that the architecture will likely be trained with backpropagation (lecture #2) and include a few softmaxes (lecture #3)
  - You should learn building blocks to be a good NLP engineer!
- **Purpose of this Lecture:** I would rather you actually understand one architecture that just know the names of ten!
  - This is why we zoomed in on skip-gram, which is one of the simplest to understand

2 Contextual Word Embeddings

- **Pre-trained contextual representations**
  - BERT (Bidirectional Encoder Representations from Transformers) is the most common

- The key idea is to have a different word embedding for every context



adapted from Devlin et al, BERT https://arxiv.org/pdf/1810.04805.pdf

ETH zürich

Sentiment Analysis

Demonstrate how Sentiment Analysis is a great use case of the previously discussed technical contents

3 Sentiment Analysis

Structure of this section

1. **What is** Sentiment Analysis?
   → Terminology and Example

2. **Why is** Sentiment Analysis **difficult**?
   → Examples
   → Bo Pang and Lillian Lee (2008): Opinion mining and sentiment analysis

3. **How to build** a Sentiment Analyzer?
   → Iyyer et al. (2015): A Multilayer Perceptron for Sentiment Analysis
   → we know it all: **Backprop, Softmax, Log-Linear Model, MLP**



from Eisenstein, Introduction to Natural Language Processing, Ch. 4

3 Sentiment Analysis

- **Sentiment analysis** is the NLP task of classifying utterances according to how they make the interlocutor feel
- In the simplest case, sentiment analysis is a two or three-class problem, with sentiments of **POSITIVE**, **NEGATIVE** (and possibly **NEUTRAL**)

- **Terminology**:
  - **Polarity:** refers to identifying the sentiment orientation
  - **Subjectivity:** refers to expression of personal feelings, views, or beliefs
    - Contrast “This apple is red” versus “This apple tastes good”

- **Fun stuff**: Fool a sentiment analyzer at http://text-processing.com/demo/sentiment/

from Eisenstein, Introduction to Natural Language Processing, Ch. 4

Example

NLP is a really interesting course !

How would you rate the sentiment (positivity vs. negativity) of this sentence?

3 Example

NLP is a really interesting course !

Example

NLP is not an interesting course !

… and how would you rate the sentiment (positivity vs. negativity) of this sentence?

3 Example

NLP is **not** an interesting course **!**

Example

NLP is not not an interesting course !

… and how would you rate the sentiment (positivity vs. negativity) of this sentence?

3 Example

neutral?

NLP is **not not** an interesting course !

3 Example Movie Review

The e, but
the uin
Pho

3 Opinion Mining and Sentiment Analysis

Overview Article: Bo Pang and Lillian Lee (2008)

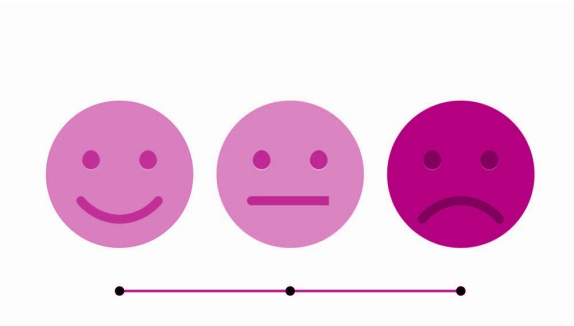
"What other people think" has always been an important piece of information for most of us during the decision-making process.

- **81%** of Internet users have done online research on a product at least once;
- among readers of online reviews of restaurants, hotels, and various services, between **73%** and **87%** report that reviews had a significant influence on their purchase
- consumers report being willing to pay from **20%** to **99%** more for a 5-star-rated item

adapted from Bo Pang and Lillian Lee – Opinion mining and sentiment analysis (2008) https://www.cs.cornell.edu/home/llee/omsa/omsa.pdf

3 Applications of Sentiment Analysis

- Review-related websites (e.g., movie reviews, product reviews…)
- Sub-component technology (e.g., system-level recommender systems (Spotify), spam detection...)
- Business and government intelligence (e.g., shopping cart analysis…)
- Applications across different domains (e.g., political opinion mining, polling, Twitter sentiment tracking for medical purposes…)



3 What Sort of Features are Good for Sentiment Analysis?

Common Features (for a log-linear model)

- term presence vs. term frequency **(tf-idf)**
- *n*-grams
- syntactic features:
  - part-of-speech information
  - dependency tree information
- topic-oriented features
- …

- **Can you think of any others?**

remark: what is tf-idf

tf-idf = tf(*t*, *d*) × idf(*t*, *D*)

t - term
d - document
D - corpus of documents

tf - term frequency
("how frequent is term t in document *d*")

idf - inverse document frequency
(1 / "how frequent is term in corpus *D*")

adapted from Bo Pang and Lillian Lee – Opinion mining and sentiment analysis (2008) https://www.cs.cornell.edu/home/llee/omsa/omsa.pdf

3 A Multilayer Perceptron for Sentiment Analysis (Iyyer et al. 2015)

problem: Sentiment analysis used to be done with a pipeline or with complicated models, can we simplify?

idea: apply a simple (syntactically-unaware) MLP to sentiment analysis

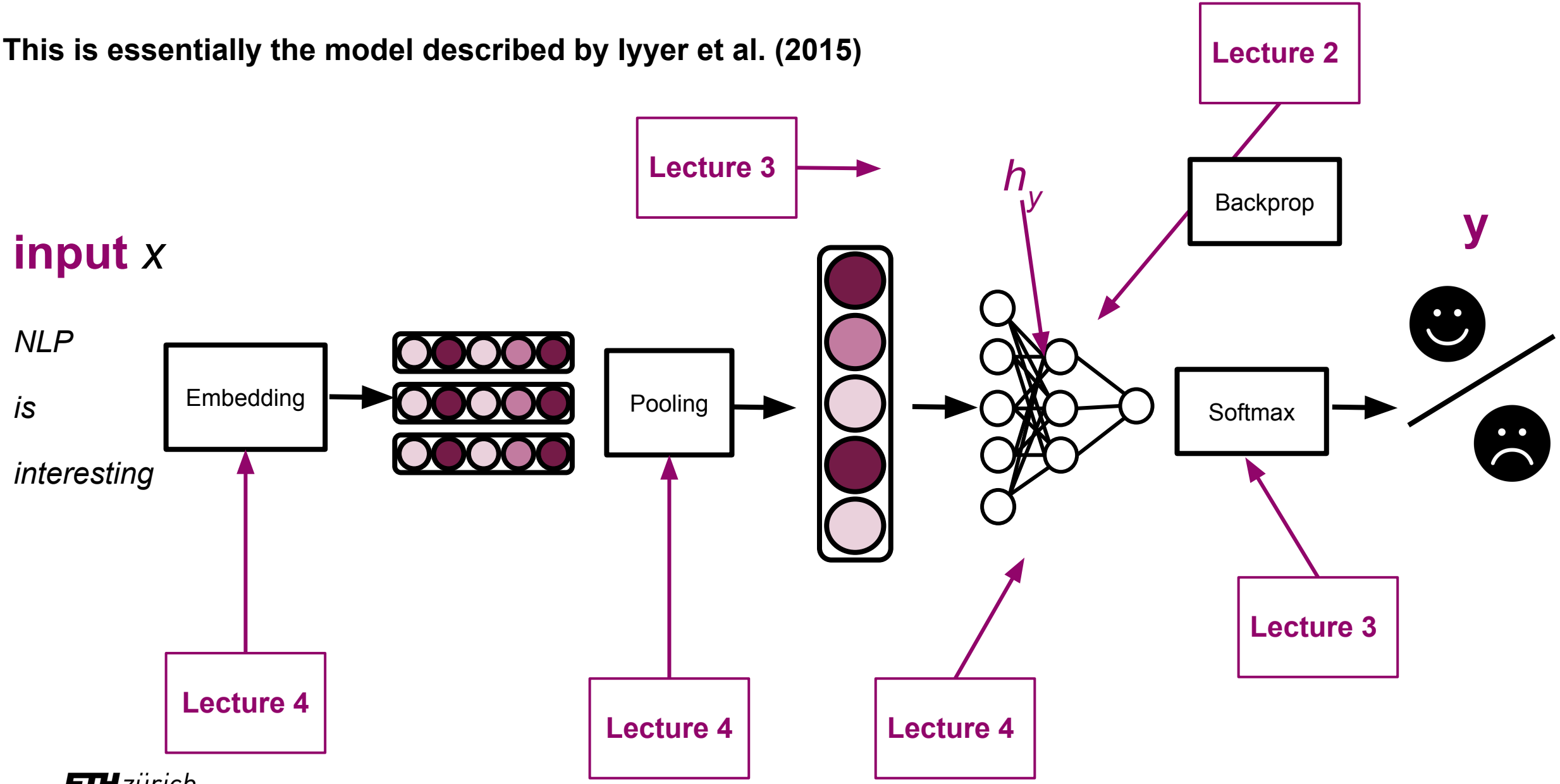
- no computationally expensive syntactic analysis
- no domain-specific lexica

finding: nonlinear transformations of the input are more important than tailoring a network to incorporate word order and syntax

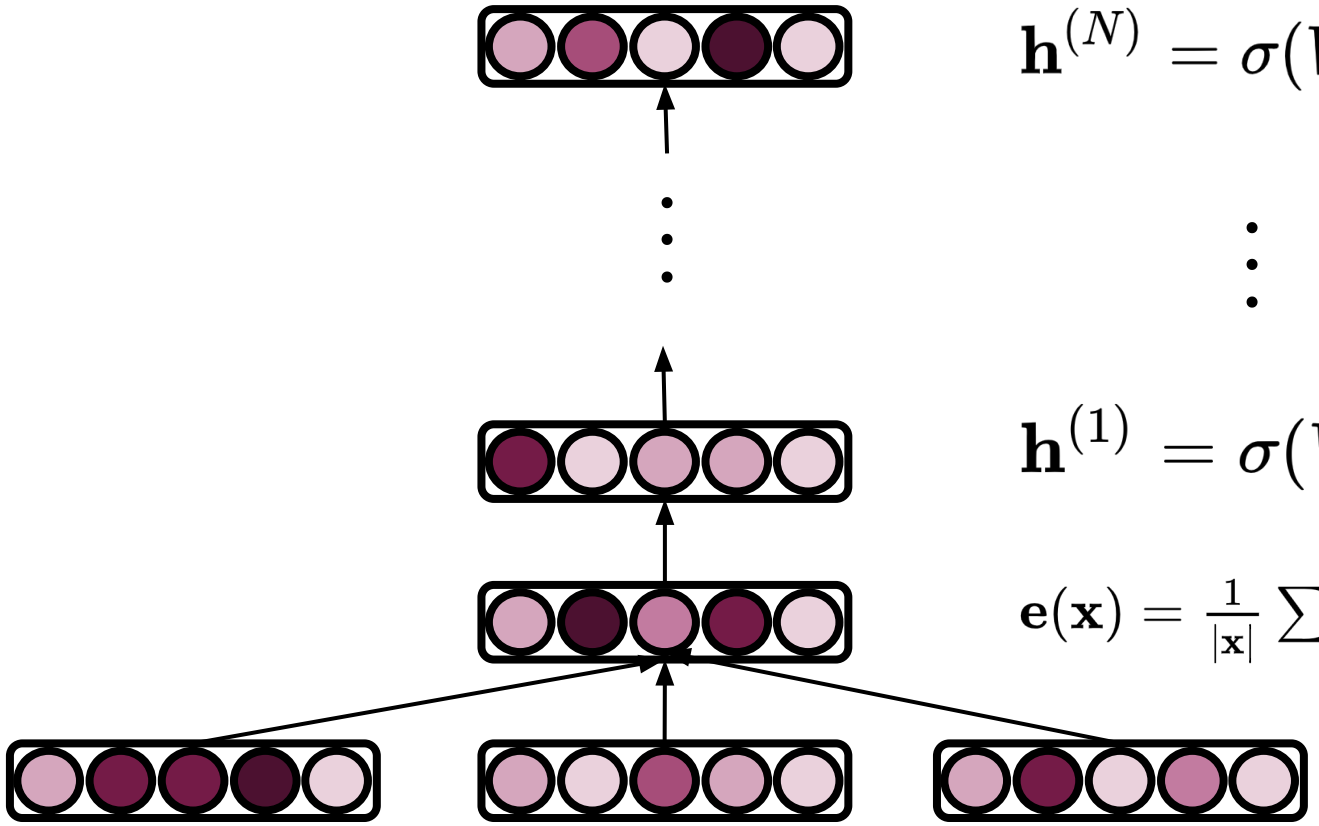
adapted from Iyyer et al. – Deep Unordered Composition Rivals Syntactic Methods for Text Classification (2015)
https://www.aclweb.org/anthology/P15-1162.pdf

3 Putting it all together: The Sentiment Analysis Pipeline

This is essentially the model described by Iyyer et al. (2015)



3 A Multilayer Perceptron for Sentiment Analysis



$$p(y \mid \mathbf{x}) = \frac{\exp h_y}{\sum_{y' \in \mathcal{Y}} \exp h_{y'}} = \text{softmax}(\mathbf{h}, y)$$

$$\mathbf{h}^{(N)} = \sigma(W^{(2)}\mathbf{h}^{(N-1)})$$

$$\mathbf{h}^{(1)} = \sigma(W^{(1)}\mathbf{e}(\mathbf{x}))$$

$$\mathbf{e}(\mathbf{x}) = \frac{1}{|\mathbf{x}|}\sum_{w \in \mathbf{x}} \mathbf{e}(w)$$

$$\mathbf{e}(w) \in \mathbb{R}^d$$

$$w \in \mathbf{x}$$

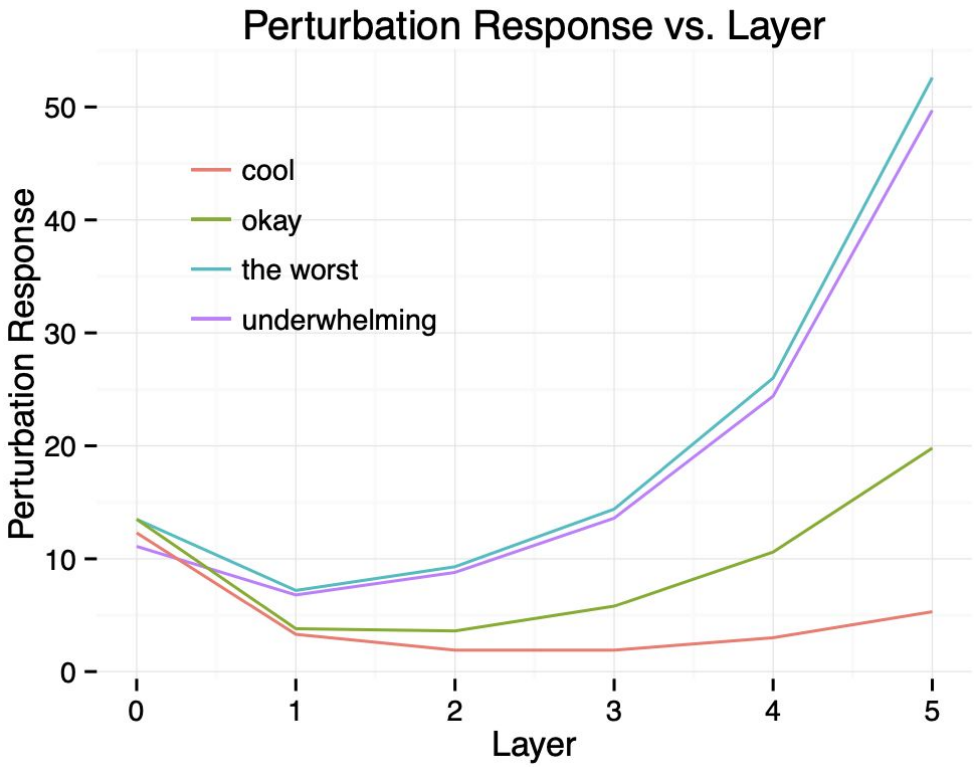
3 A Multilayer Perceptron for Sentiment Analysis

- **Iyyer et al. (2015)'s strategy for applying an MLP to sentiment analysis**
  - Pool the embeddings associated with an input sequence of tokens
  - Pass the pooled vector through one or more feed-forward layers
  - Add a softmax at the end to get a probabilistic classifier

- You now have all the tools to build your first NLP model!
  - Dataset for training the MLP available here: https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews.

3 Empirical Findings (Iyyer et al. 2015)

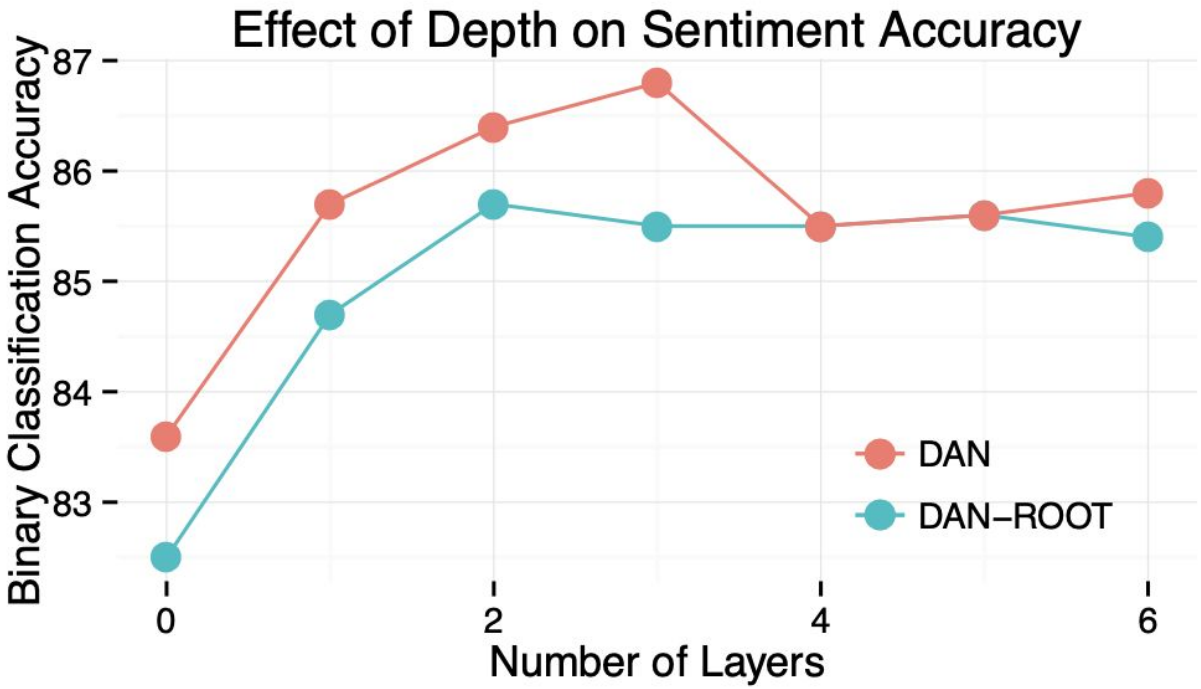
- **Evaluation of an MLP on Sentiment Analysis Tasks**



- Let's look at the the perturbation response (difference in 1-norm) at each layer of a 5-layer MLP after replacing *awesome* in "*the film's performances were **awesome**"* with four words of varying sentiment polarity (**cool, okay, the worst, underwhelming**)
  - Basically a counterfactual analysis
- As the MLP gets deeper, negative sentences are increasingly different from the original positive sentence
  - Why do you think this is?

3 Empirical Findings (Iyyer et al. 2015)

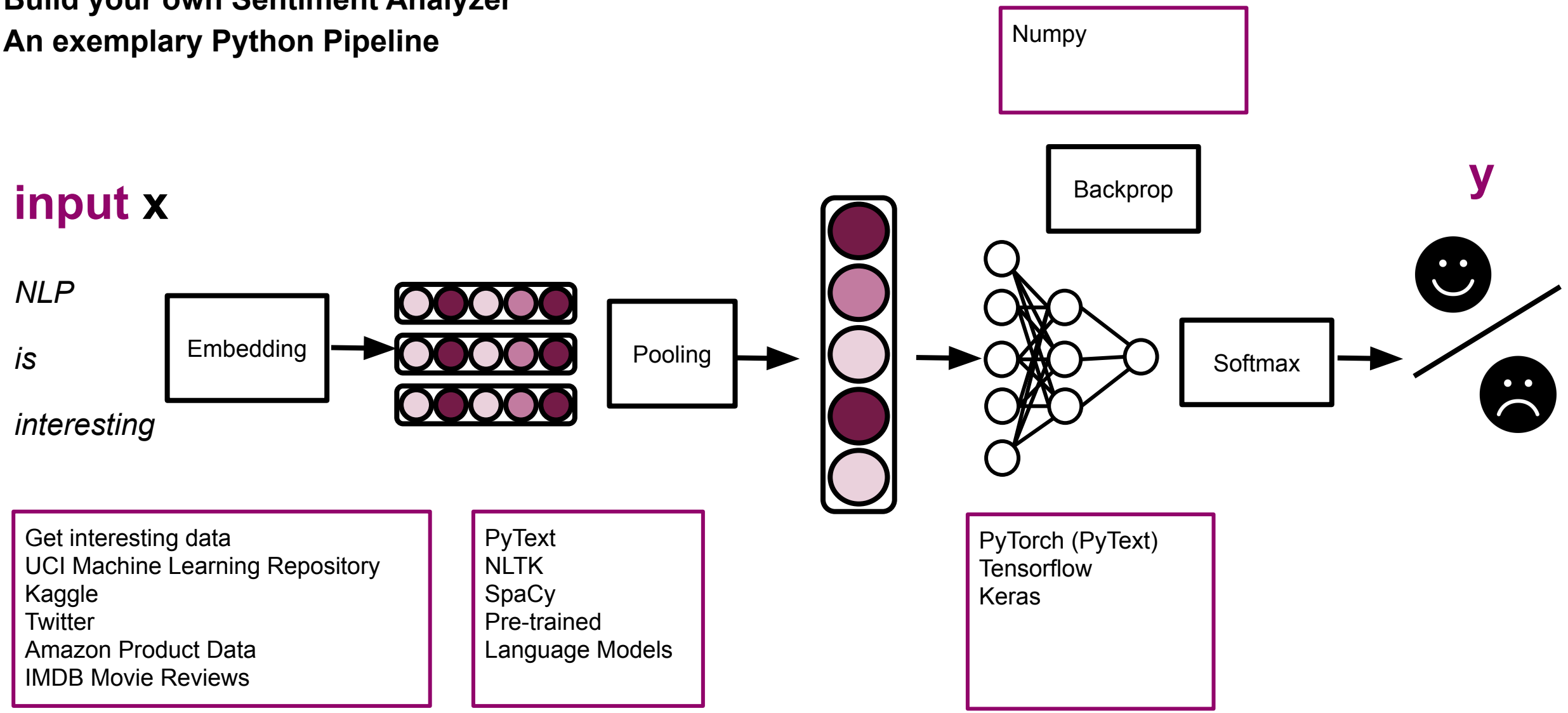
- **Evaluation of DAN on Sentiment Analysis Tasks**



- Two-to-three layers is the optimal choice for the DAN on the SST binary sentiment analysis task
- Adding any depth at all is an improvement over the log-linear model that does not model non-linear interactions

3 Putting it all together: The Sentiment Analysis Pipeline

Build your own Sentiment Analyzer
An exemplary Python Pipeline

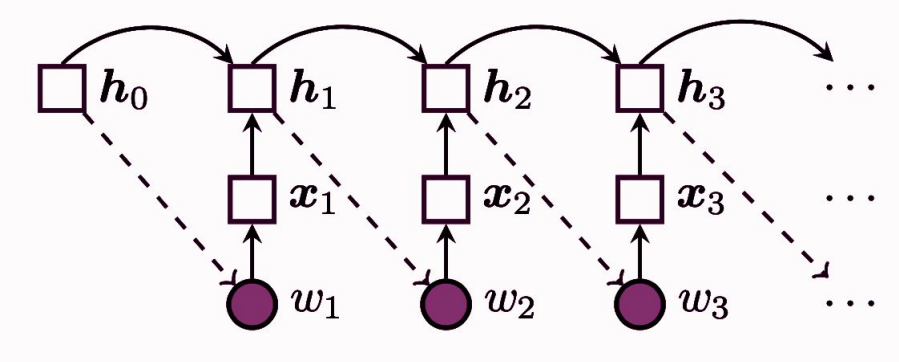


ETH zürich

Sneak Preview of Next Lecture

Language Modeling with *n*-grams and RNNs

- Why is machine learning taught in computer science? Why is it not just statistics?
- **One good answer:** structured prediction
  - What happens when you have to do classification over an exponentially large space?
  - You need algorithms! Therefore, you need computer science!
- Next time, we will discuss models for classification over a countably infinite space
  - Specifically, we will model the space of all sentences
  - In NLP, we call this language modeling



ETH zürich

Fin