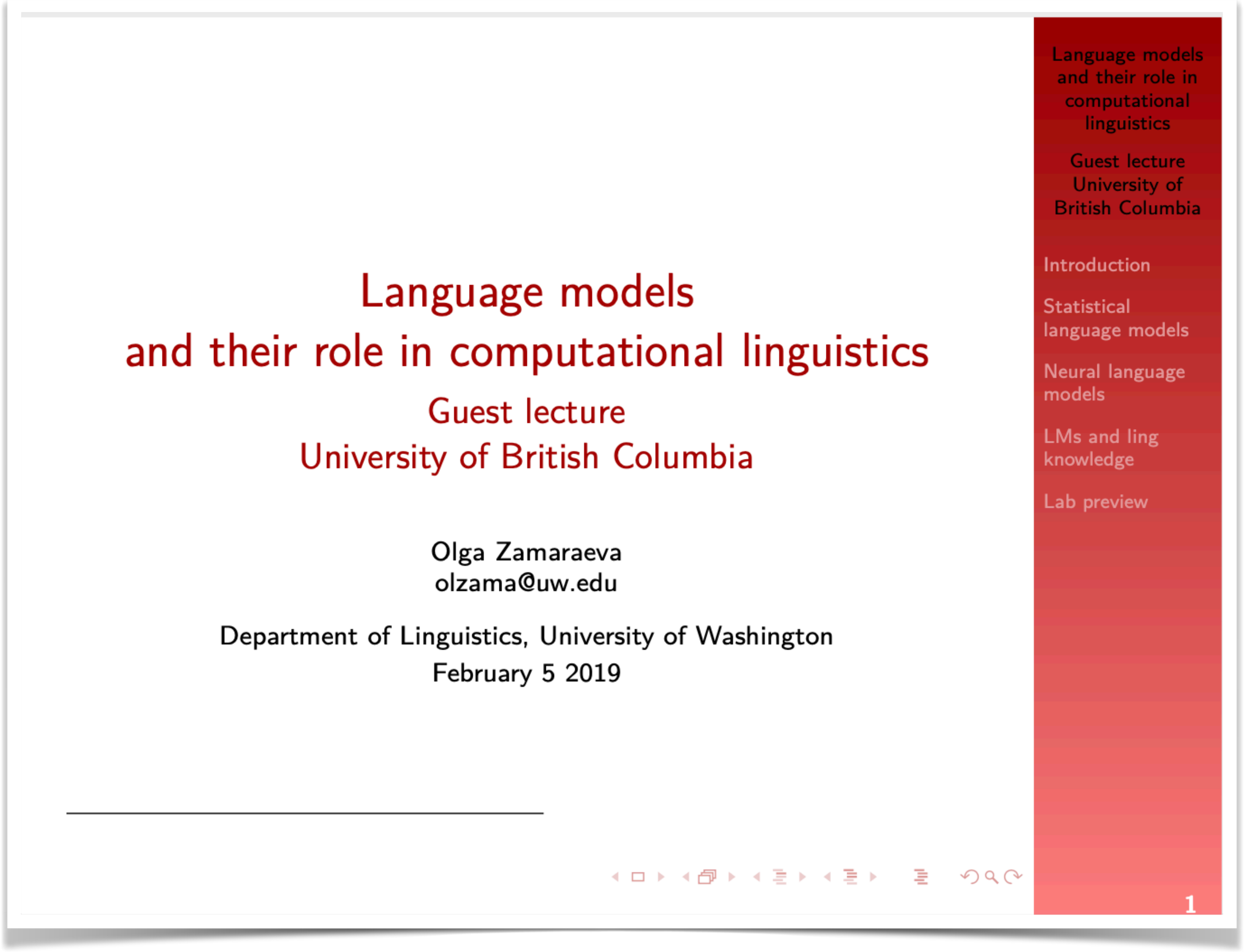
Computational Methods for Linguists

Ling 471

Olga Zamaraeva (Instructor)
Yuanhe Tian (TA)
05/18/21

Reminders
and announcements

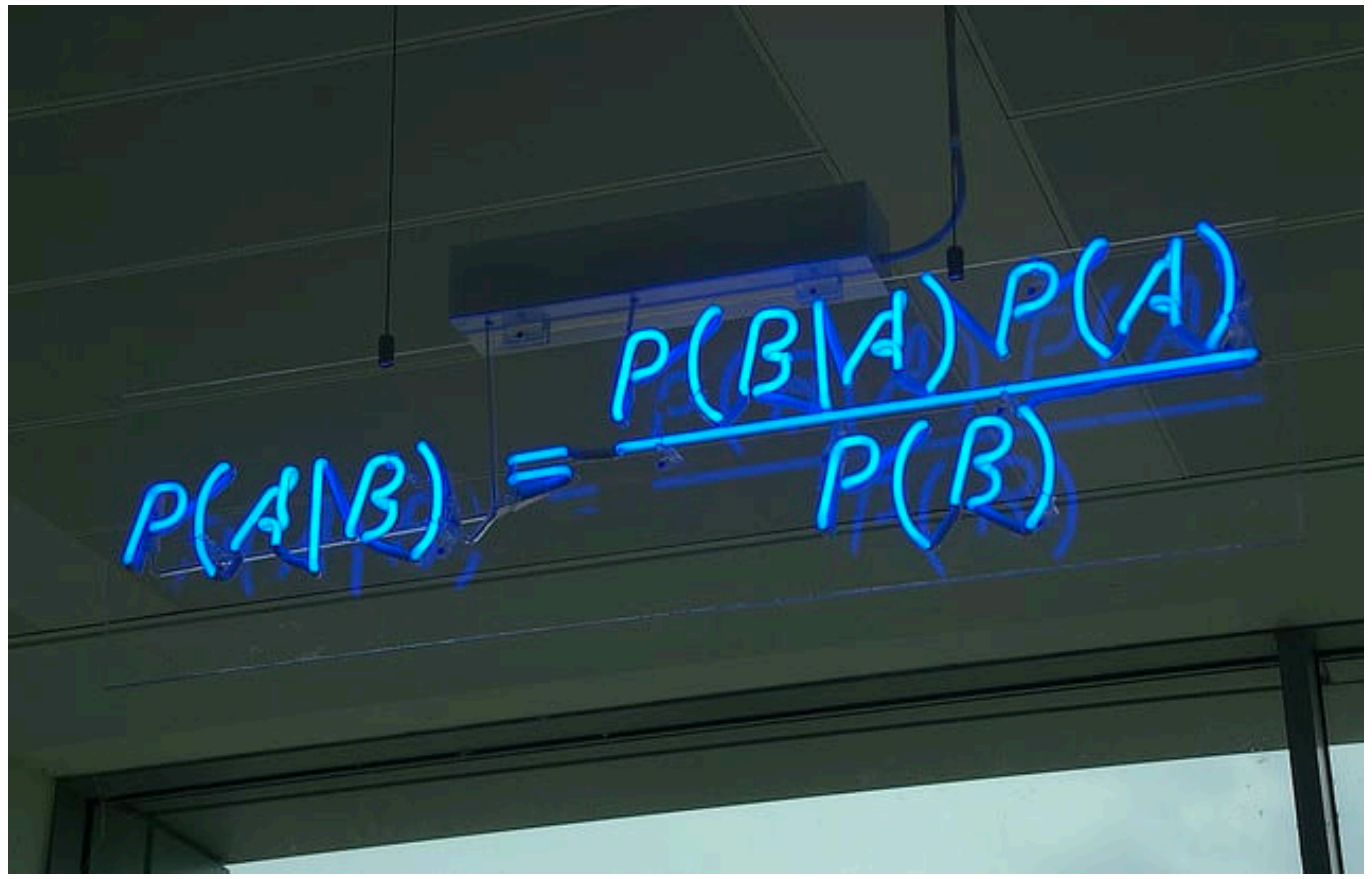
- Presentation topic suggestions
- Some of today's slides will look different
  - (I cheated and used a guest lecture I once did, as well as a Ling472 lecture I did)
  - (material probably overlaps with 472)
  - aside: LaTeX
    - maybe a demo next week



Plan for today

- Smoothing
- Language models
  - N-gram
  - Neural
    - maybe spill over to Thu

Naive Bayes
a classification algorithm



https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/

- Naive Bayes relies on word counts to estimate probailities of word sequences
  - …and trains on labeled data
  - …to predict labels for unseen/unlabeled data

- What's "nontrivial" about it?
  - What if you have never seen a word before?
  - It's count will be 0
  - It's probability will be 0
  - You multiply your terms by 0…
    - …and P(entire text) = 0!
      - Not good!

Smoothing

for out-of-vocabulary" items

- Crucial technique for **all** probabilistic modeling
  - **Don't** want **zeros** in your counts, **ever**!
- Add a **fake "unknown" word** to your **training** vocabulary
- For every real word, **subtract** some small probability mass and **add** it to the unknown's!
- Now in testing, every UNK gets a **non-zero** probability!
- Why **subtract** from real words though?
- And **how** is this related to **smoothing curves** in linear regression?

Laplacian Smoothing

$$P(\mathrm{w_i}|\mathrm{class}) = \frac{\mathrm{freq}(\mathrm{w_i}, \mathrm{class})}{\mathrm{N_{class}}}$$

class ∈ {Positive, Negative}

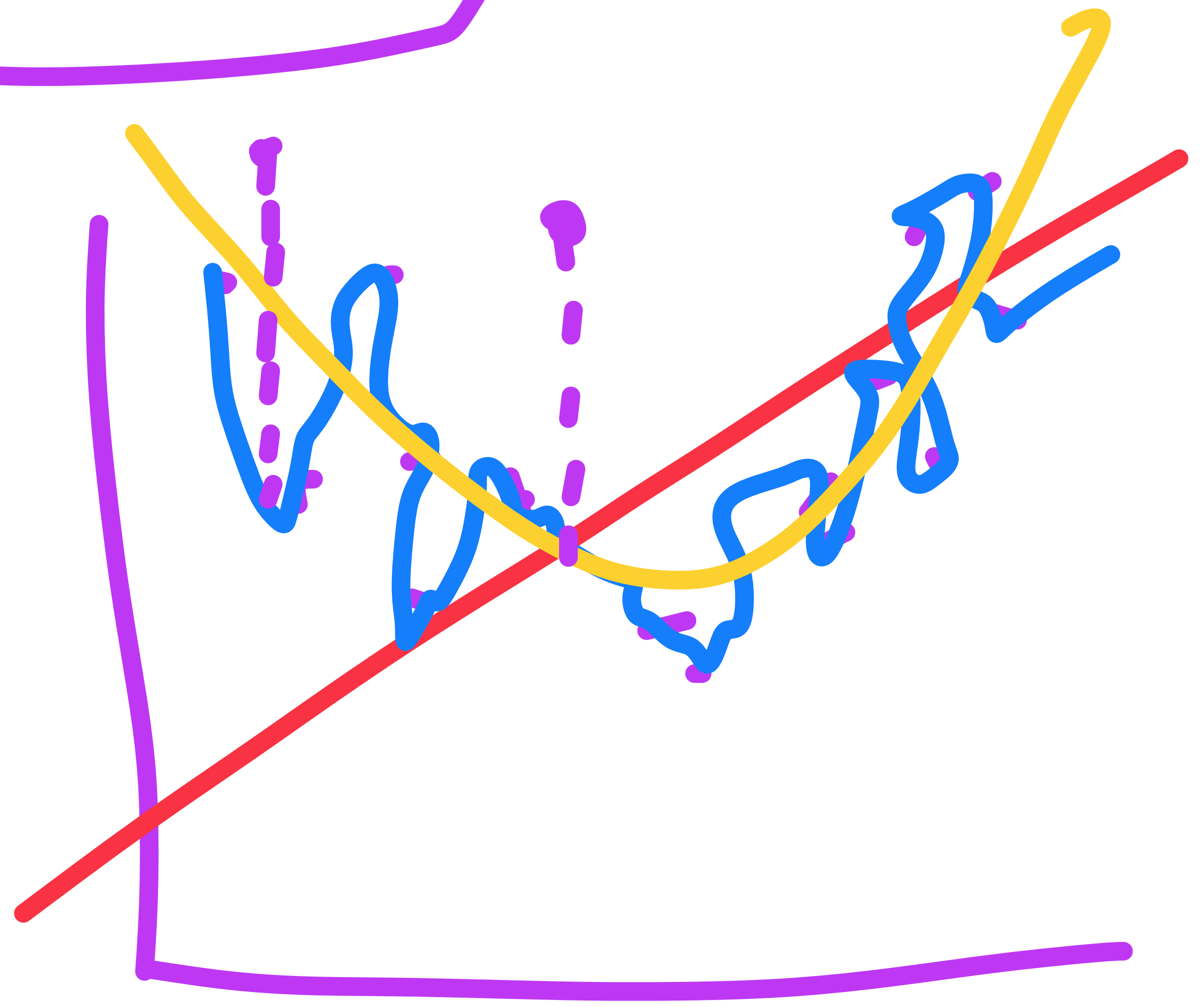
$$P(\mathrm{w_i}|\mathrm{class}) = \frac{\mathrm{freq}(\mathrm{w_i}, \mathrm{class}) + 1}{\mathrm{N_{class}} + \mathrm{V_{class}}}$$

N_{class} = frequency of all words in class

V_{class} = number of unique words in class

https://laptrinhx.com/tweet-sentiment-analysis-using-naive-bayes-classifier-3354548227/

'UNK"



Language Models

"A grammar is better, but in practice people use language models."

D. Jurafsky

"You are uniformly charming!" cried he, with a smile of associating and now and then I bowed and they perceived a chaise and four to wish for.

Generated by a trigram LM trained on Austen's books

"What comes out of a 4-gram model of Shakespeare looks like Shakespeare because it is Shakespeare."

D. Jurafsky

Language models and their role in computational linguistics
Guest lecture University of British Columbia
Introduction
Statistical language models
Neural language models
LMs and ling knowledge
Lab preview

Language Models

bigram

France

London is the capital of ... England

unigram

- Language models are programs which output the most probable word given some context
  - That's it!

Language models and their role in computational linguistics
Guest lecture University of British Columbia
Introduction
Statistical language models
Neural language models
LMs and ling knowledge
Lab preview

N-grams: The (simplified) math behind the simplest LM

- The LM is *trained* on a corpus and can then assign probabilities to new, *test* sentences
- Train by estimating actual probabilities of word sequences from actual corpora
- E.g. what probability will a LM trained on corpus TC assign to the sentence:

"London is the capital of England"

- In corpus TC, how many times did we see *England* after *London is the capital of*?

Language models and their role in computational linguistics

Guest lecture University of British Columbia

Introduction

Statistical language models

Neural language models

LMs and ling knowledge

Lab preview

- Language is very creative!

N-grams: The simplest LM

London is the capital of England

- What we'd like to calculate:

$$\frac{C(London, is, the, capital, of, England)}{C(London, is, the, capital, of)}$$

5 gram

- In some cases, it is possible (using e.g. the web)
- But in most cases, we'd never find a corpus big enough

Markov assumption

Andrey Markov (1856-1922)
(Not-so-fun-fact: In 1908, Markov was fired from the University for refusing to spy on his students)

- Markov assumption: The probability of a given word only depends on **a few** previous words, not the entire sequence
- *Approximate* the history given the last (few) word(s)

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

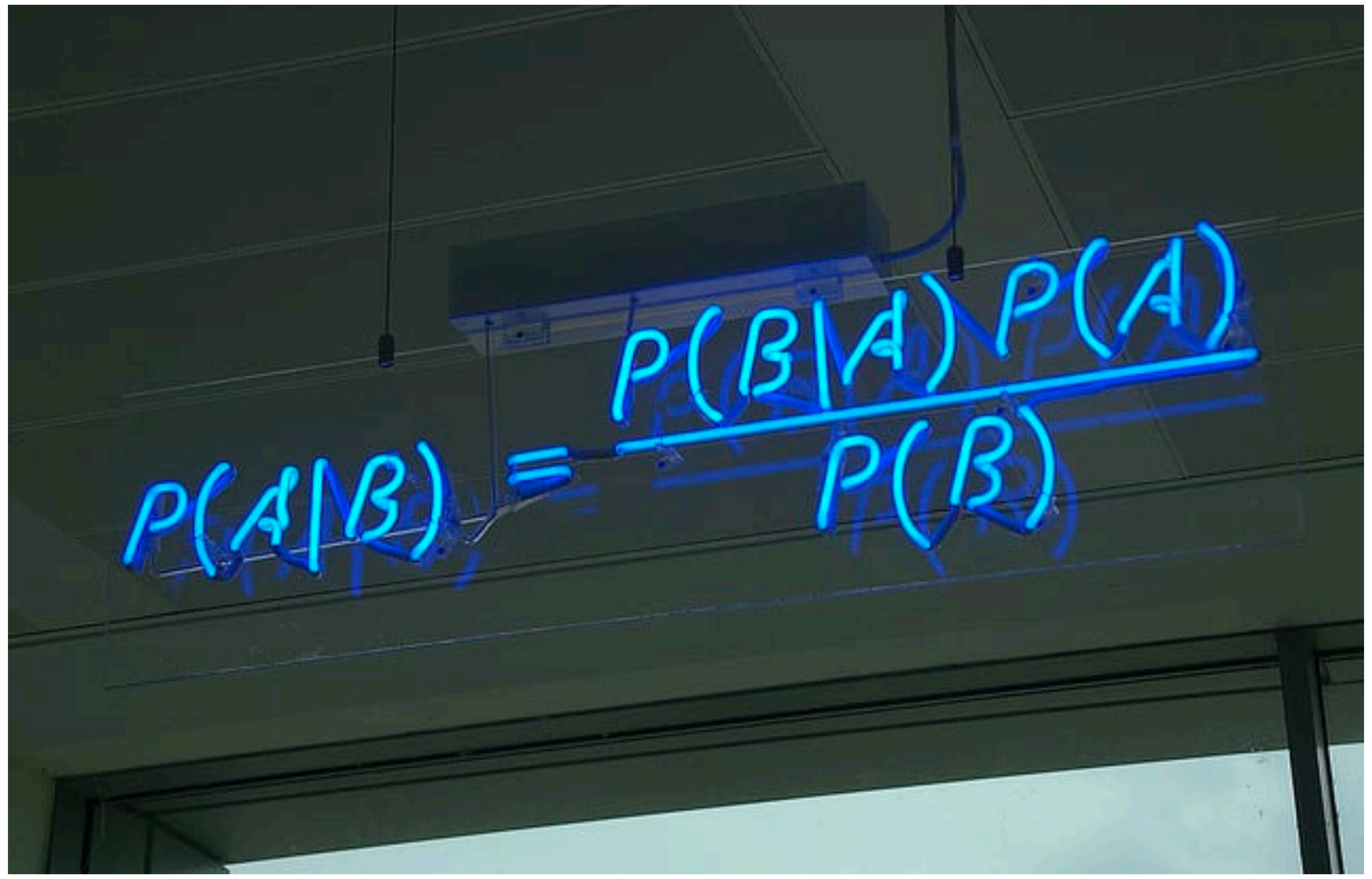
w_1^{n-1}

List c of E

$\underbrace{w_1 \; w_2 \qquad\qquad}$ w_n

N-grams
and Nance Bayes

- What's the relationship?
  - N-grams are not a classifier
    - they are good for text generation
    - and for estimating word probabilities
      - ...which in turn is what Naive Bayes needs!
  - Naive Bayes is a classifier which uses word frequencies
    - it can use unigram, bigram, n-gram



https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/

N-gram: bigger N means closer approximation

- P(England |London is the capital of)
  - P(England |of) – **bigram**
  - P(England |capital of) – **trigram**
  - P(England |the capital of)
  - P(England |is the capital of)

N-gram: bigger N means closer approximation

Consider *generating* from such models:

- P(Horatio |Alas, poor Yorick! I knew him,)
  - P(Horatio |him,) – **bigram**
  - P(Horatio |knew him,) – **trigram**
  - P(Horatio |I knew him,)
  - P(Horatio |Yorick! I knew him,)
  - P(Horatio |poor Yorick! I knew him,)

Small N = “silly” model, big N = rigid model (how interesting is it to generate exact strings from Shakespeare’s *Hamlet*?)

Desireable: Generalizing over contexts

- *London* is the capital of...
- *Causton* is the capital of...

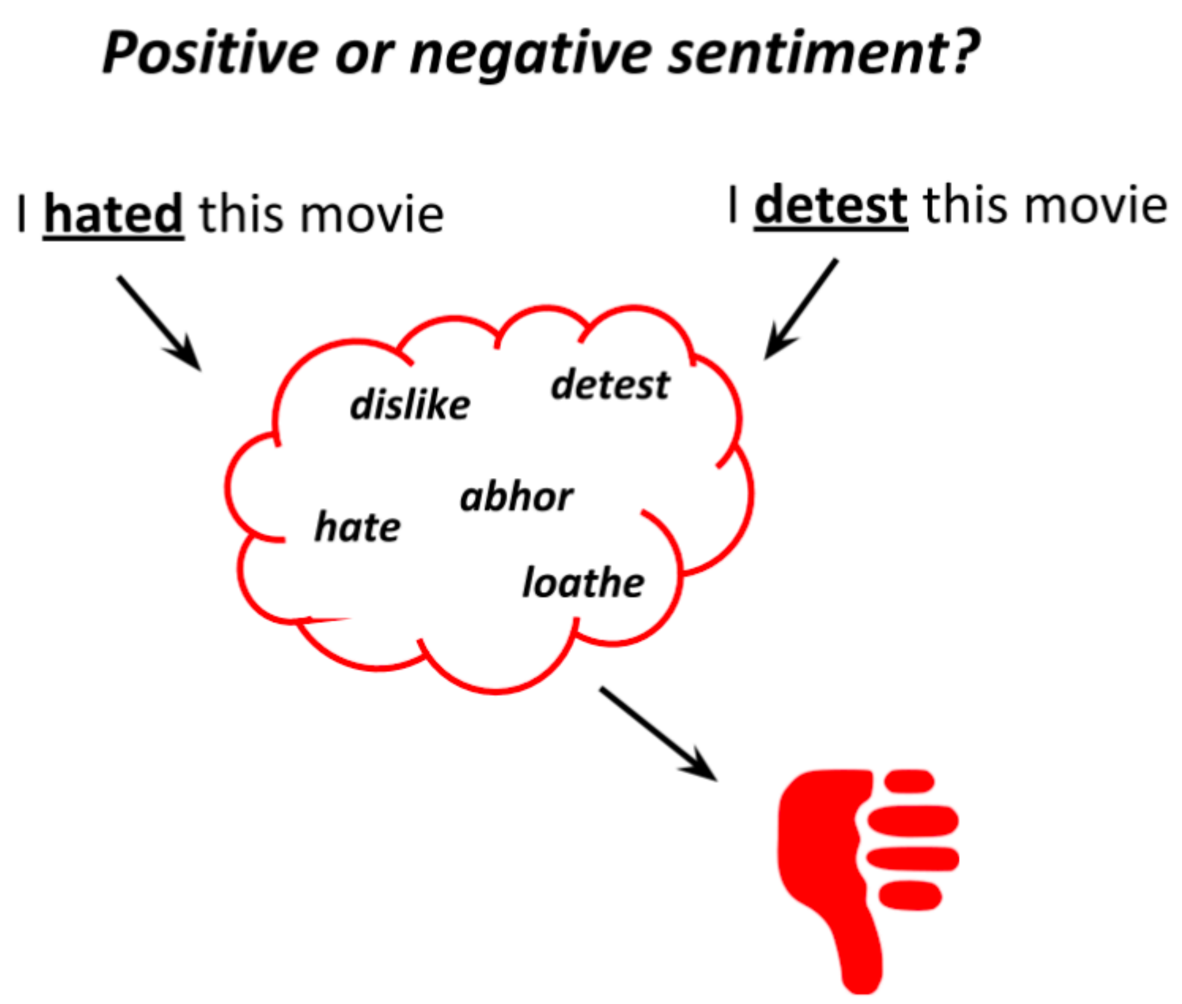


Figure from Allyson Ettinger's tutorial at SCiL 2019

Interim summary

- N-grams are simple, easily implementable, trainable on small amounts of data
  - but, are either silly (approximate the corpus poorly) or start generating Shakespeare (approximate too much)
- Today, NLP mostly uses more flexible *neural* LMs

Neural language models

Neural* language models

- Predict the word given context (or vice versa)
- Generalize over contexts, are more "creative" than n-grams:
  - Learn which words occur in similar contexts
  - It is possible to build a neural model that creates representations for unknown words "on the fly"**
- But:
  - Are more complex to train
  - Require lots of training data to start working well
  - Learn the training data biases

*These are *simplified* neural architectures

**Not the same architecture as in the lecture

$y \quad t^{\circ} \leftarrow tod$

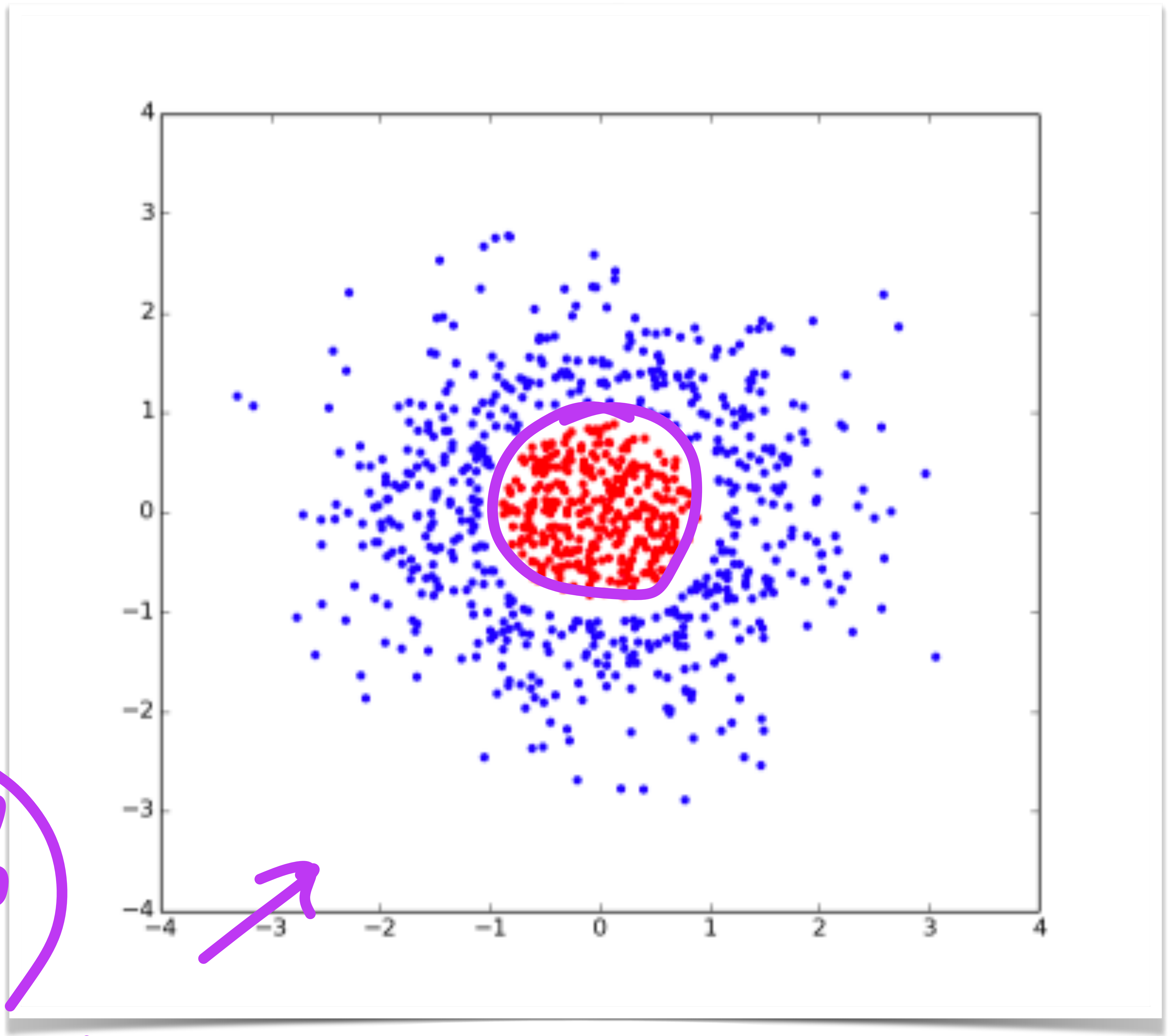
XOR

x

A case for neural nets

$d_1: [x_1 \quad x_2 \quad x_3]$

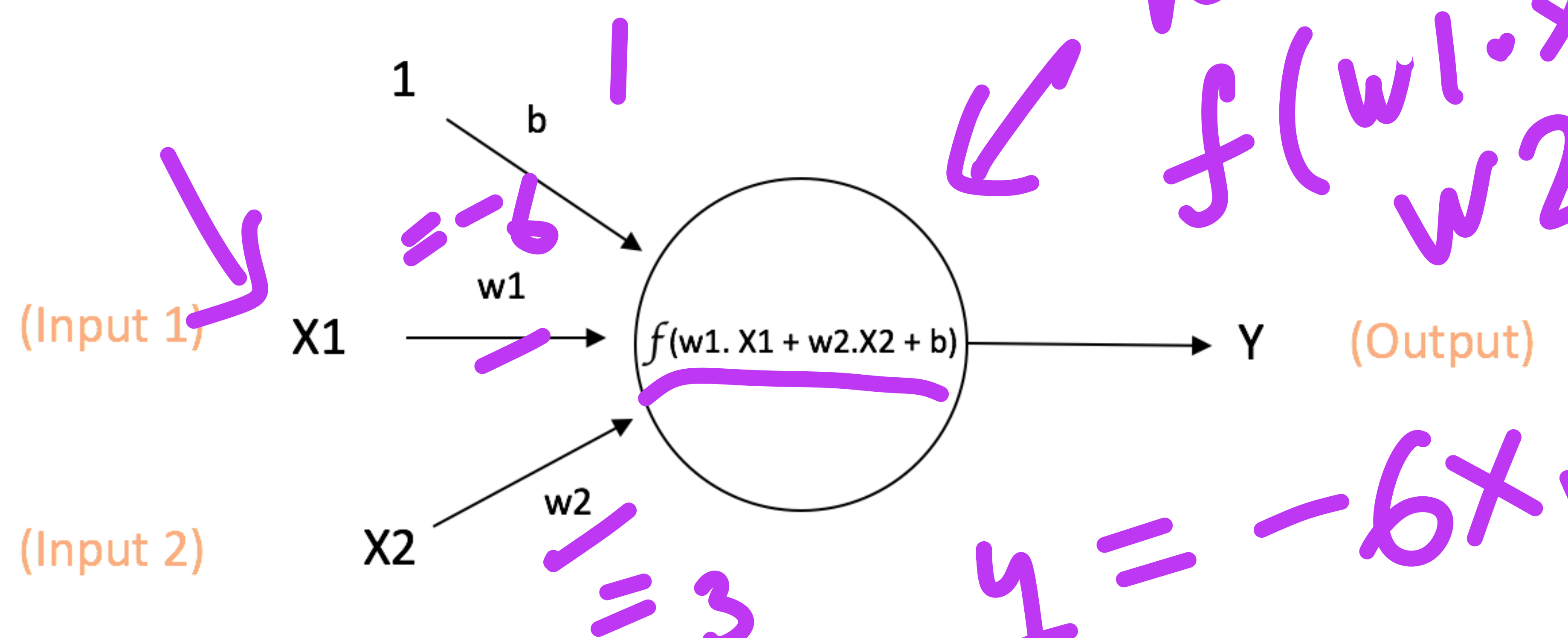
- Huge power of neural nets: tod month geo
  - they deal well with **non-linearly-separable** data



https://stackoverflow.com/questions/1148513/difference-between-a-linear-problem-and-a-non-linear-problem-essence-of-dot-pro

neuron

$f(w1 \cdot x1 + w2 \cdot x2 + b)$



$y = -6x_1 + 3x_2 + 1$

Output of neuron = Y= f(w1. X1 + w2.X2 + b)

https://www.kdnuggets.com/2016/11/quick-introduction-neural-networks.html

$y = mx + b$

$y = m x_1^{1} + n x_2^{2} + b$

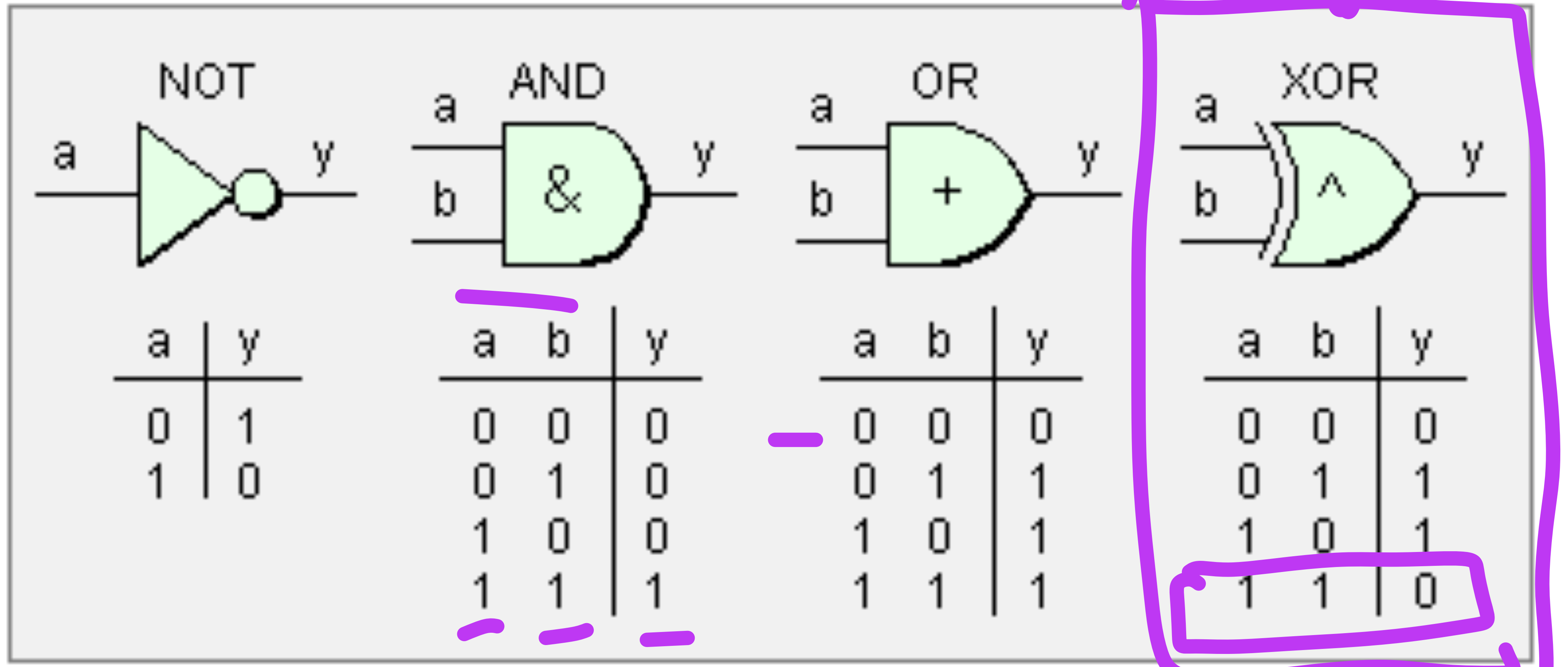
exclusive OR

0 = False
1 = True

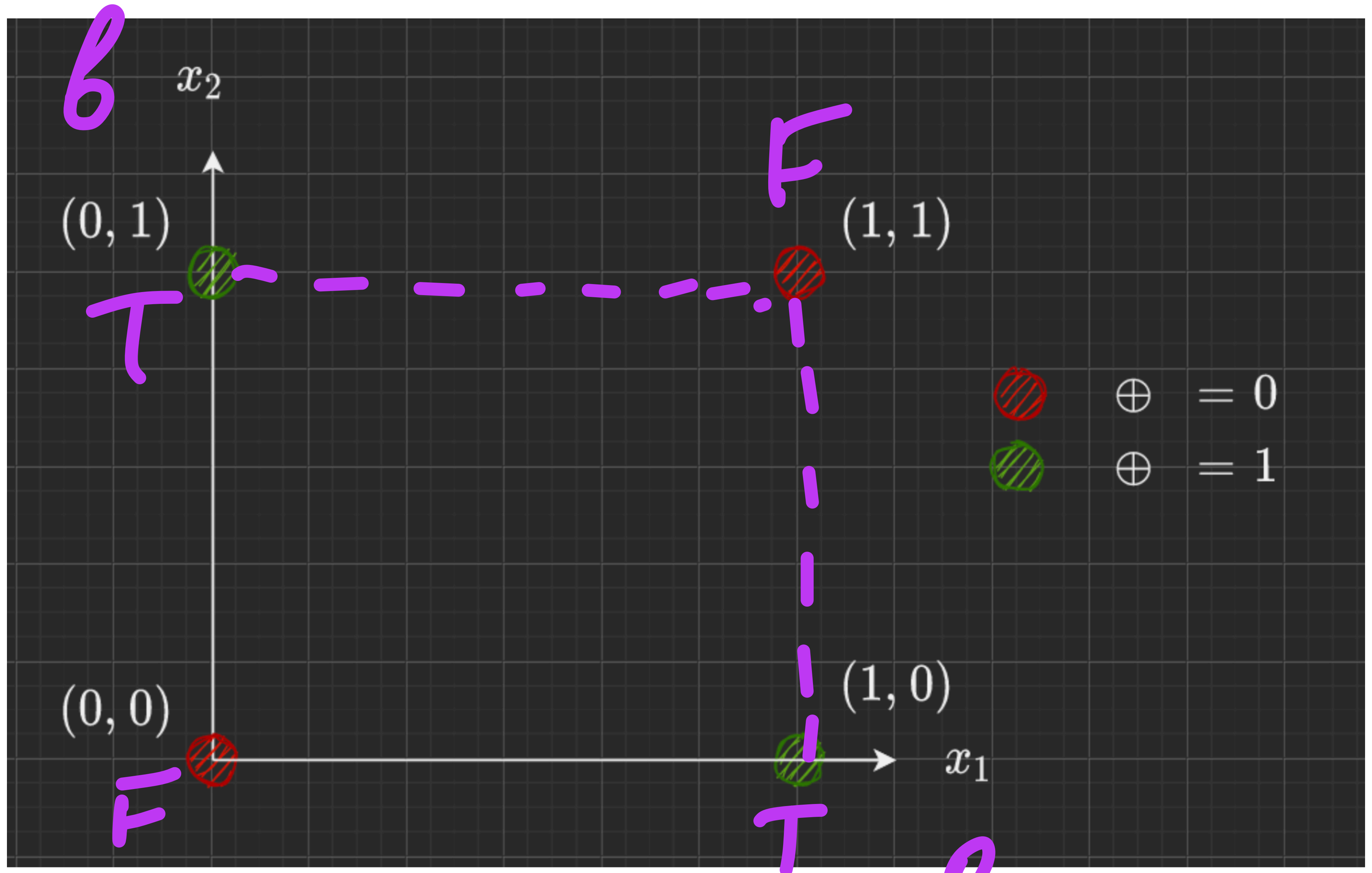
XOR

A case for neural nets

- The XOR function
  - Similar to our familiar **OR** in python and other programming languages
  - ...but **XOR** is True **only** when **one** of the expressions is True



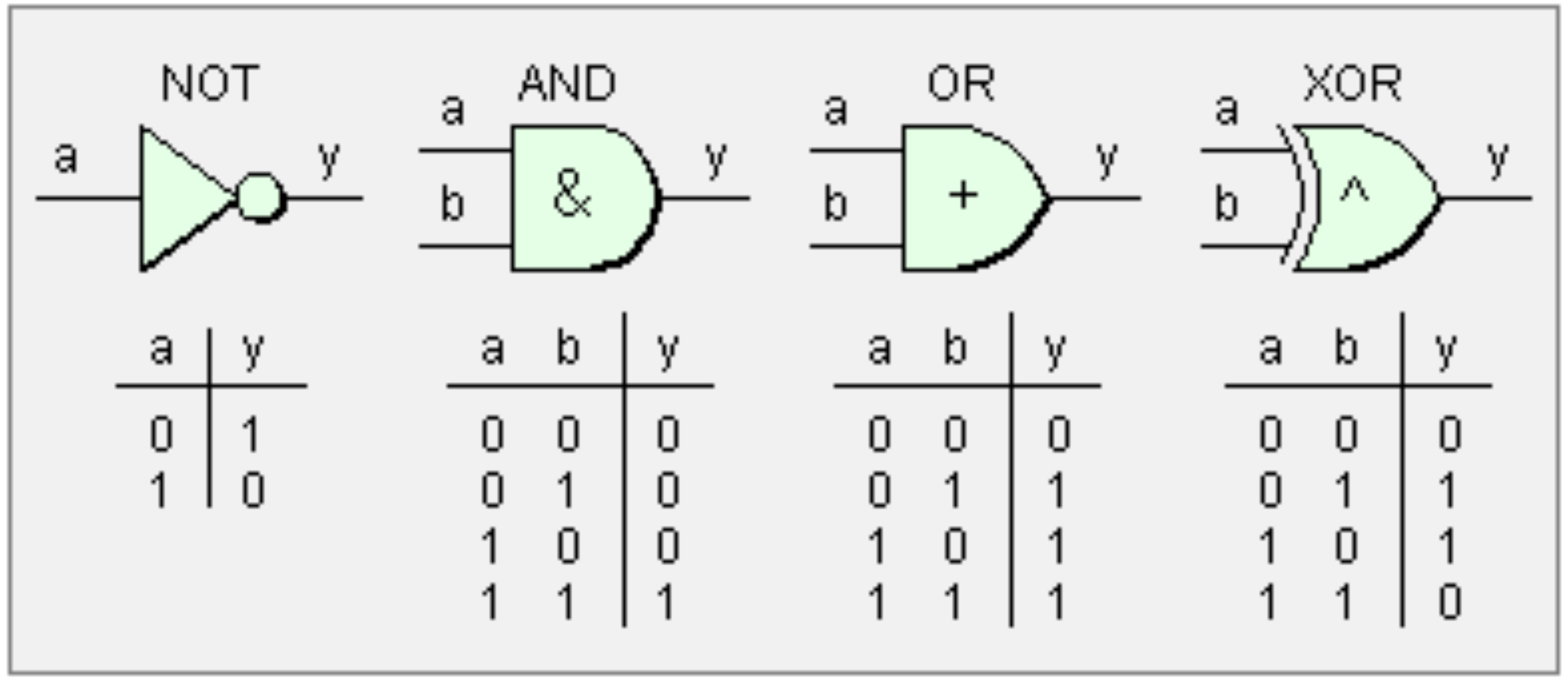
https://www.eetimes.com/how-to-invert-three-signals-with-only-two-not-gates-and-no-xor-gates-part-1/



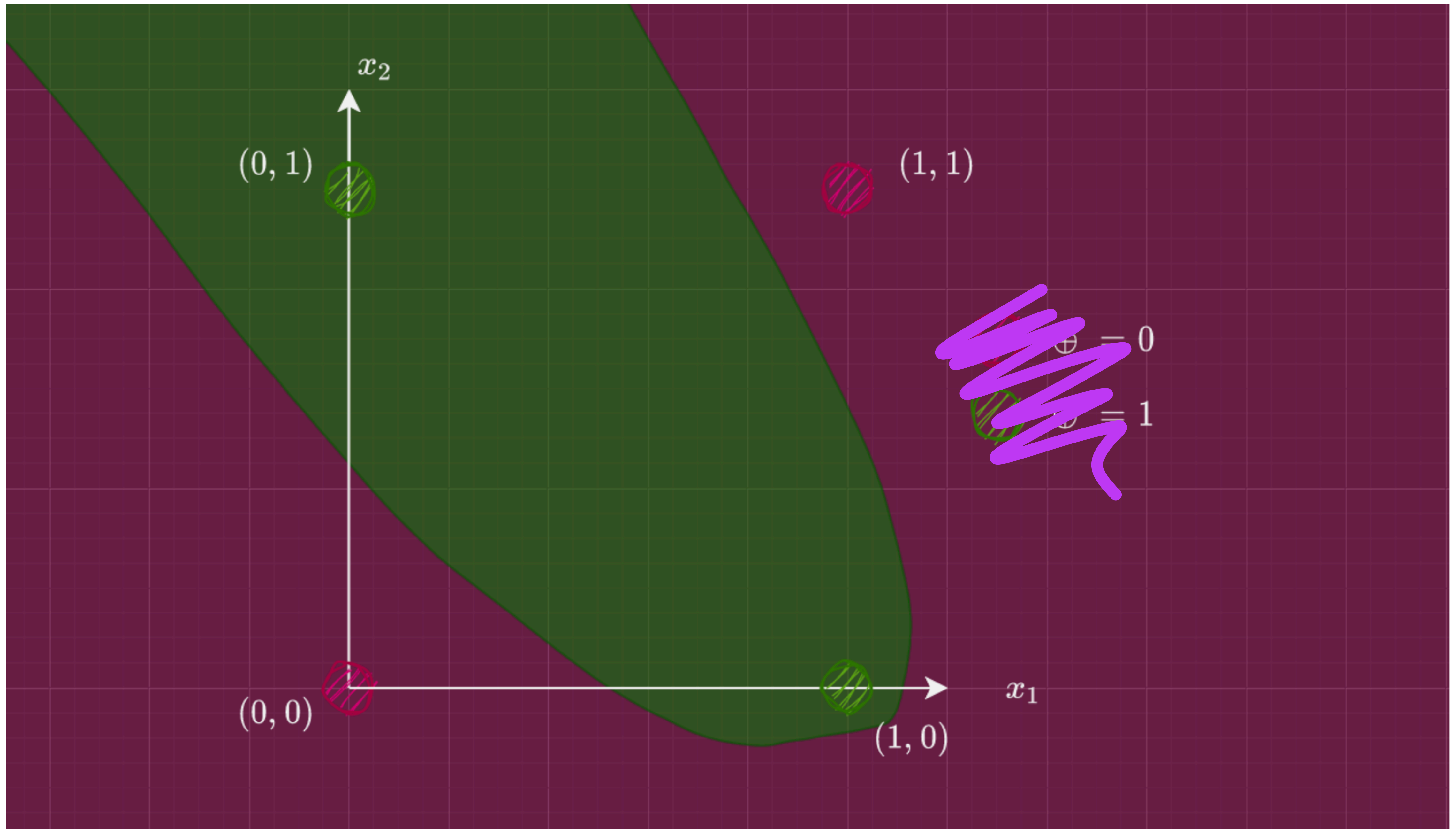
The XOR output plot — Image by Author using draw.io

XOR
A case for neural nets

- XOR is not linearly separable
  - need a more complex decision boundary
  - The **data points** are:
    - (0,0),(0,1),(1,0), and (1,1)
    - **(x1,x2)**
  - The output: **y** is either 1 or 0
    - True or False
  - Can we **map** x1 and x2 to a different space **such that** we can separate the data points linearly and correctly output the y?



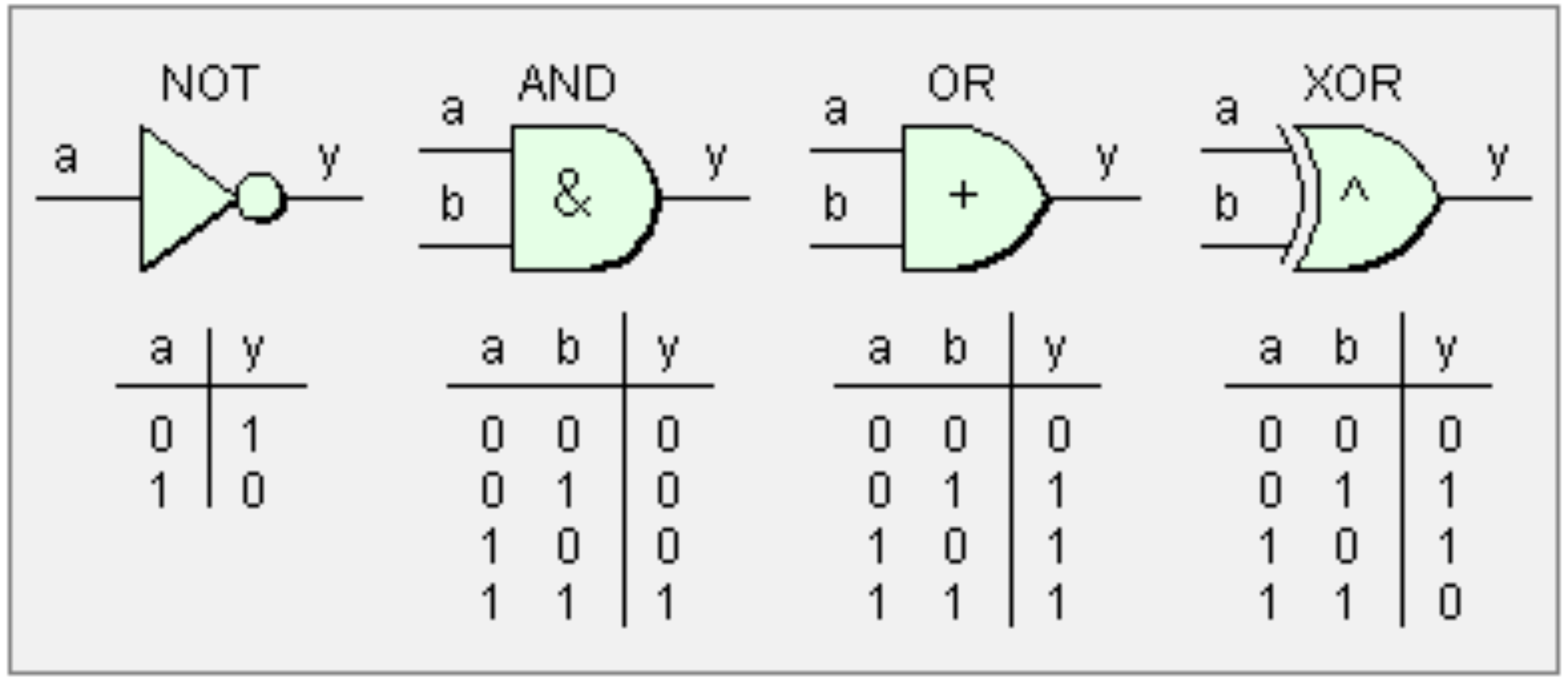
https://www.eetimes.com/how-to-invert-three-signals-with-only-two-not-gates-and-no-xor-gates-part-1/



https://towardsdatascience.com/how-neural-networks-solve-the-xor-problem-59763136bdd7

XOR
A case for neural nets

- XOR is not linearly separable
  - need a more complex decision boundary
  - The **data points** are:
    - (0,0),(0,1),(1,0), and (1,1)
    - **(x1,x2)**
  - The output: **y** is either 1 or 0
    - True or False
  - Can we **map** x1 and x2 to a different space **such that** we can separate the data points linearly and correctly output the y?



a	y
0	1
1	0

a	b	y (AND)	y (OR)	y (XOR)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

https://www.eetimes.com/how-to-invert-three-signals-with-only-two-not-gates-and-no-xor-gates-part-1/

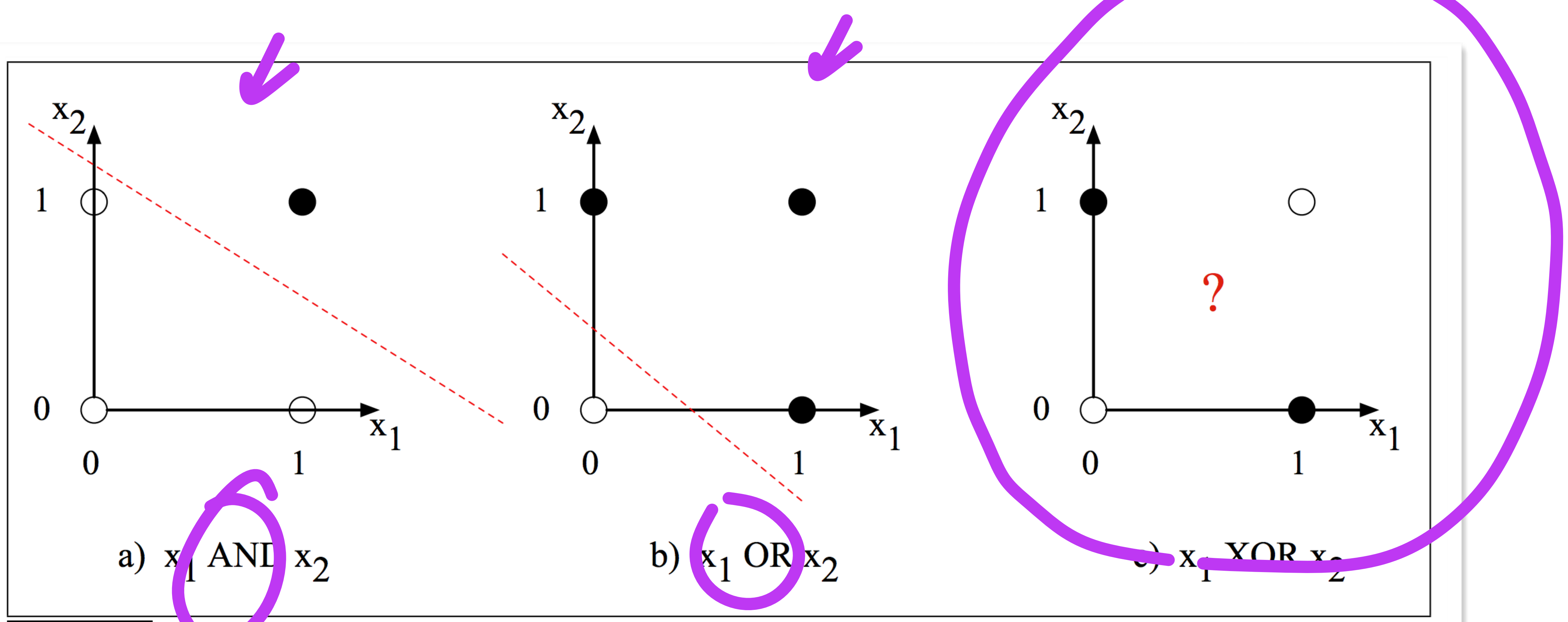
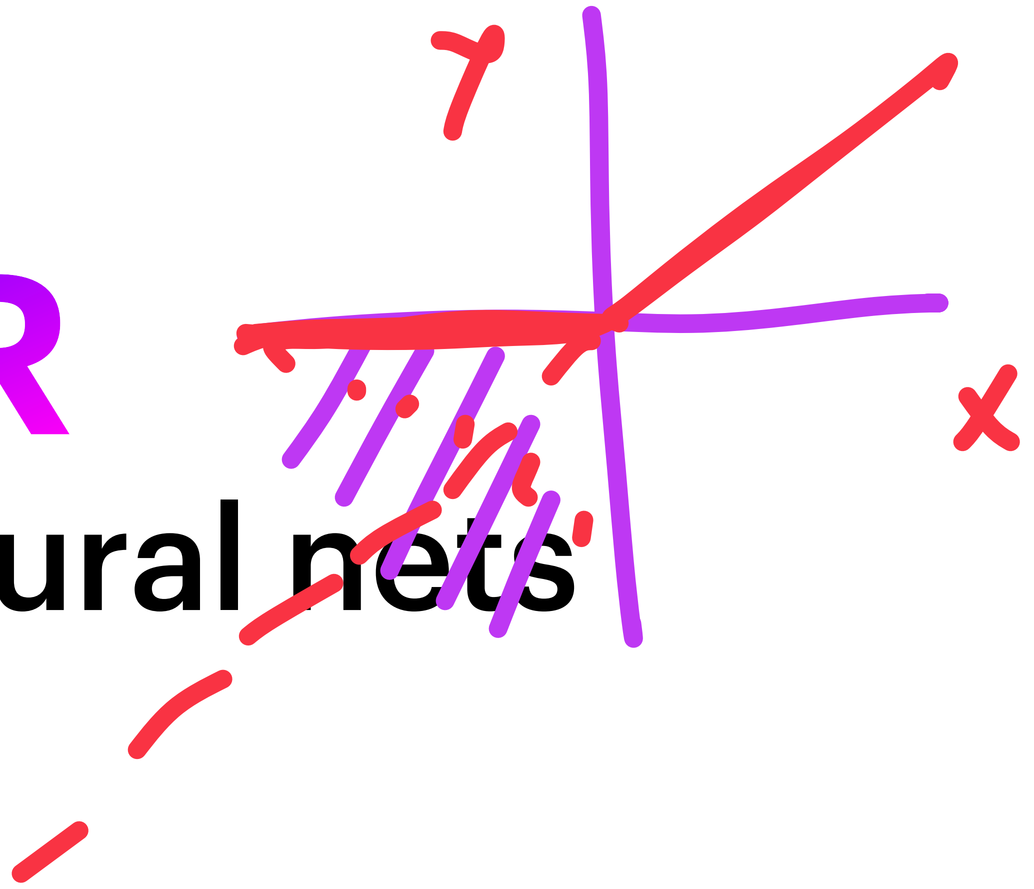


Figure 7.5 The functions AND, OR, and XOR, represented with input x_0 on the x-axis and input x_1 on the y axis, Filled circles represent perceptron outputs of 1, and white circles perceptron outputs of 0. There is no way to draw a line that correctly separates the two categories for XOR. Figure styled after Russell and Norvig (2002).

Speech and Language Processing (Jurafsky and Martin 2004)

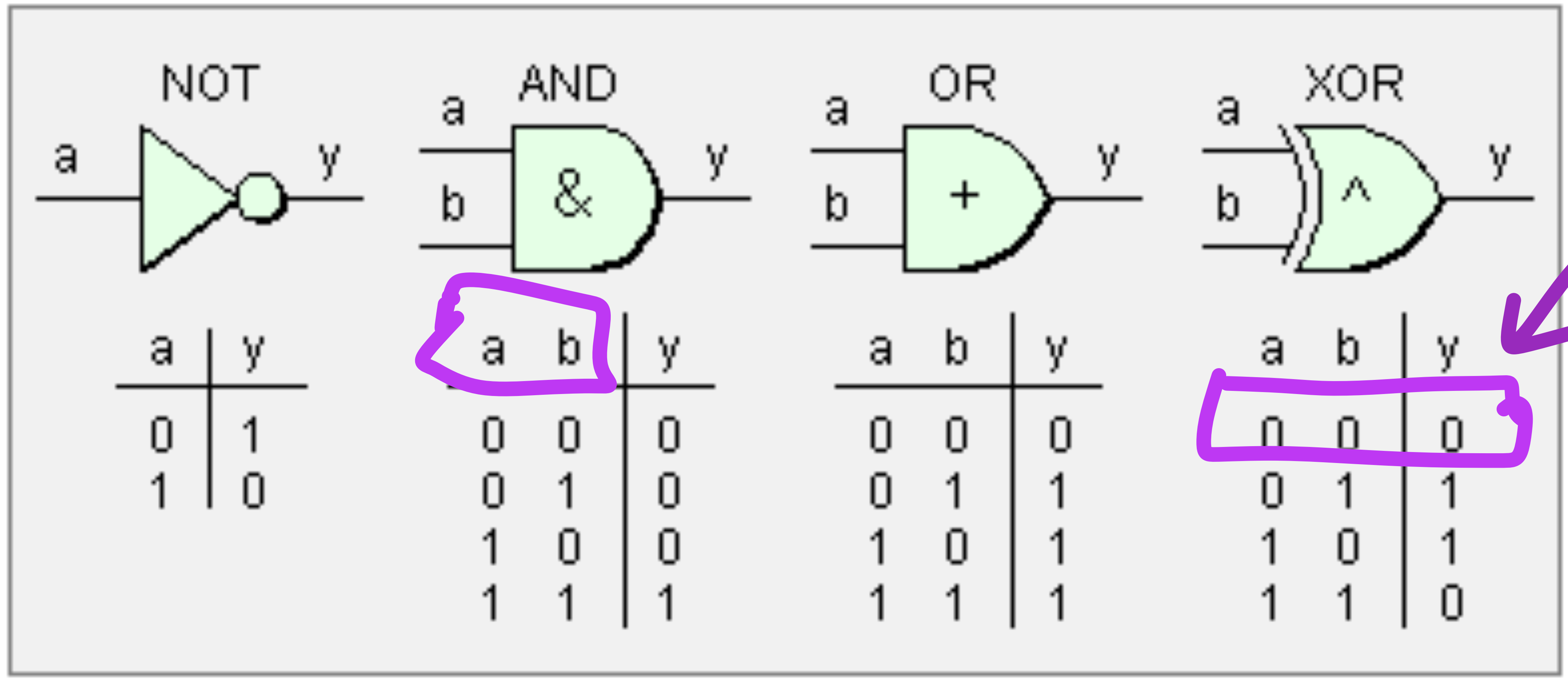
XOR

A case for neural nets



- Construct a simple **neural network**
  - Each “neuron” is a **function**
    - computes the sum of w1x1+w2x2+cb
    - if result < 0: returns 0
  - Each x is **weighted** upon entering each neuron
  - So, like a linear equation but a network, and nonlinear :)

$a = x_1 \quad b = x_2$



https://www.eetimes.com/how-to-invert-three-signals-with-only-two-not-gates-and-no-xor-gates-part-1/

$y_1 : h_1 \cdot W_{21} + h_2 \cdot W_{22} + b$

$y_1 : 0 \cdot 1 - 2 \cdot 0 + 0 = 0$

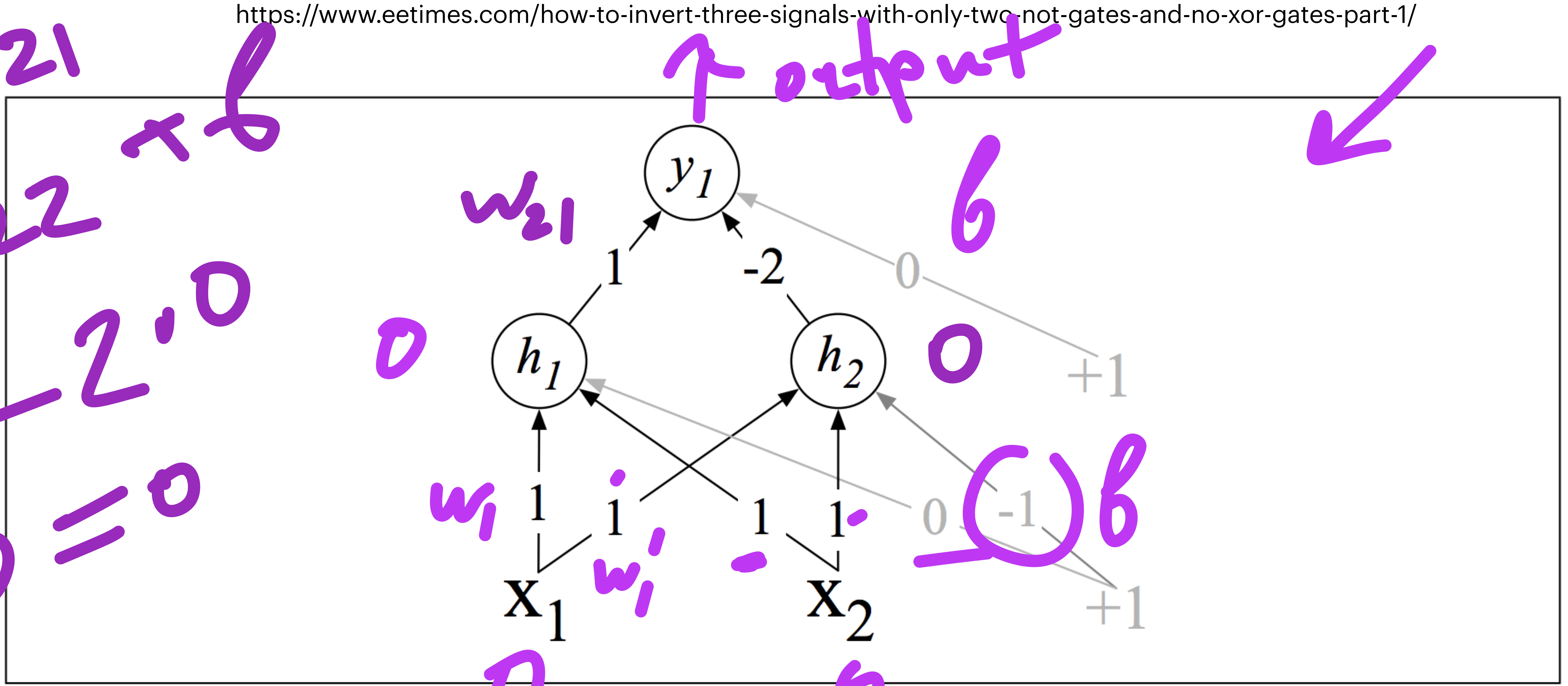


Figure 7.6 XOR solution after Goodfellow et al. (2016). There are three ReLU units, in two layers; we’ve called them h_1, h_2 (h for “hidden layer”) and y_1. As before, the numbers on the arrows represent the weights w for each unit, and we represent the bias b as a weight on a unit clamped to +1, with the bias weights/units in gray.

Speech and Language Processing (Juratsky and Martin 2004)

$h_2 : 0 \cdot 1 + 0 \cdot 1 - 1 = -1 \rightarrow 0$

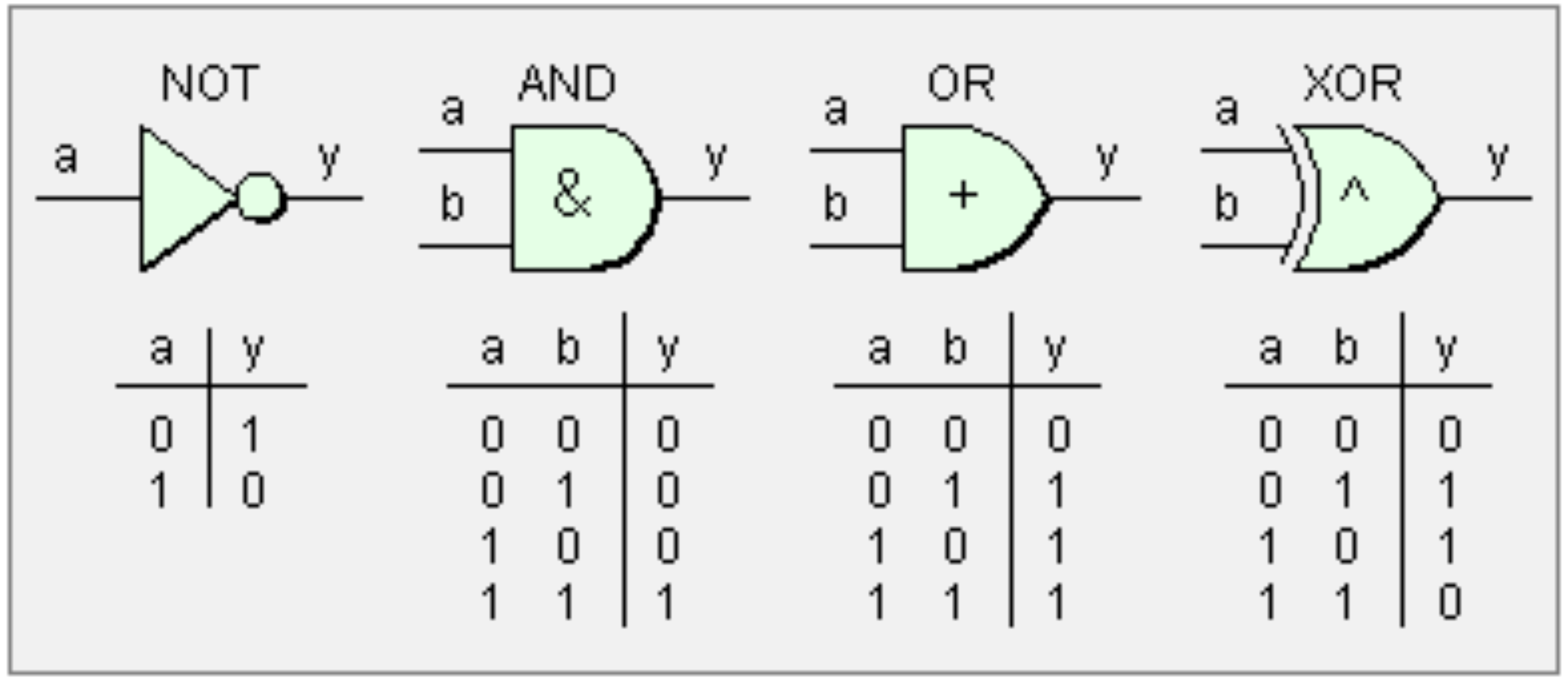
Activity (which I know you always wanted to do):
(Manually) compute the neural XOR for:
[x1=1, x2=1] and [x1=1, x2=0]

https://olzama.github.io/Ling471/assignments/activity-May18.html

XOR

A case for neural nets

- Our x1 and x2:
  - now turned into h1 and h2
  - ...which exist in a different space
  - ...and are linearly separable



https://www.eetimes.com/how-to-invert-three-signals-with-only-two-not-gates-and-no-xor-gates-part-1/

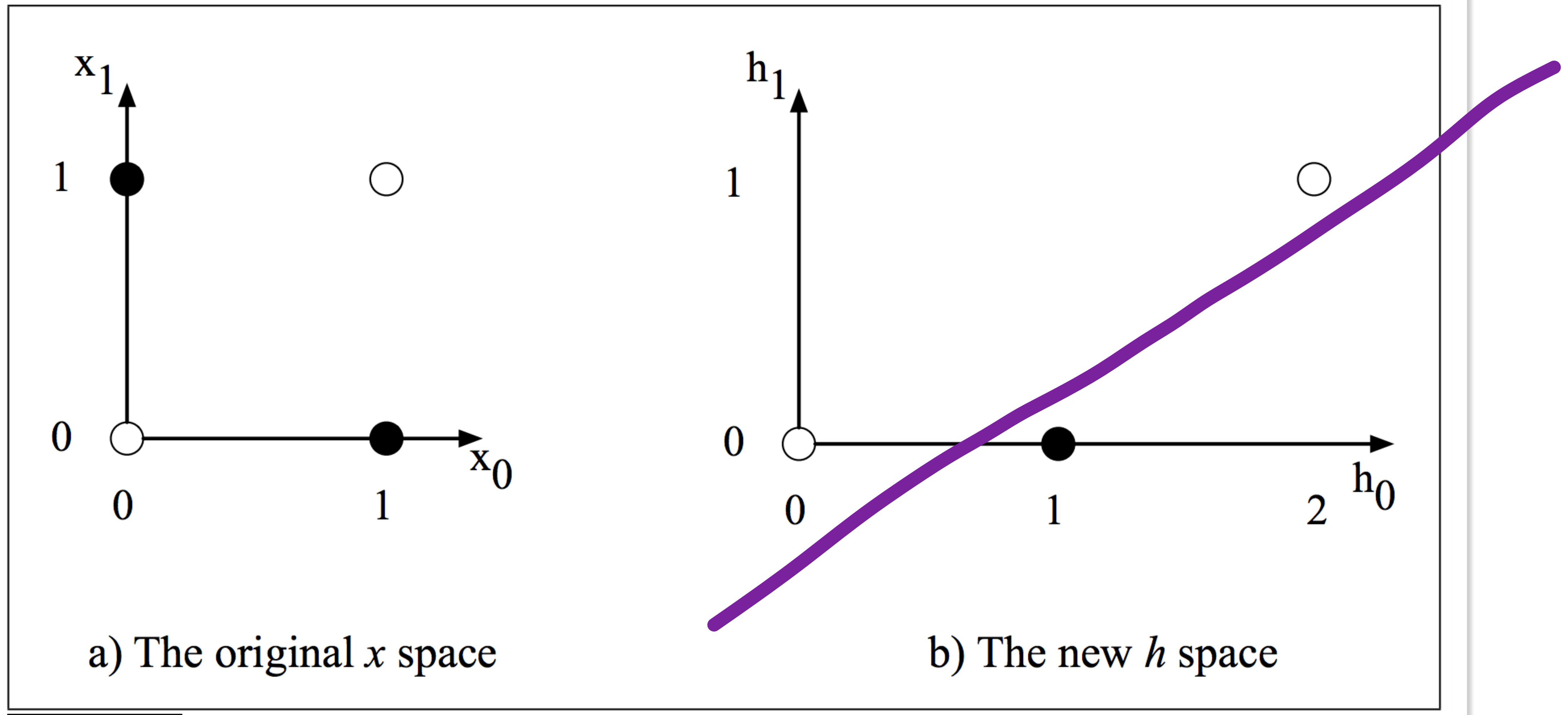
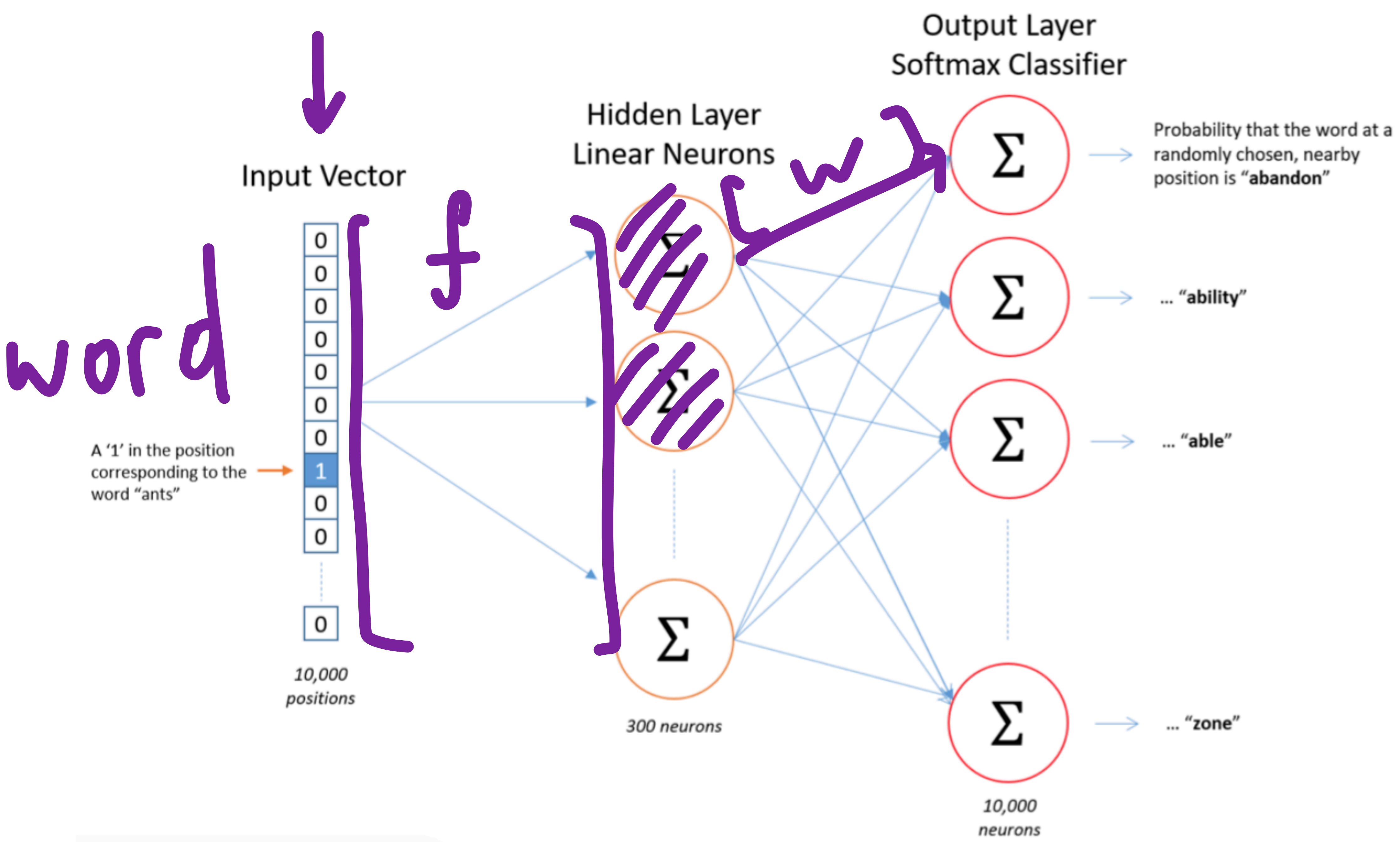


Figure 7.7 The hidden layer forming a new representation of the input. Here is the representation of the hidden layer, h, compared to the original input representation x. Notice that the input point [0 1] has been collapsed with the input point [1 0], making it possible to linearly separate the positive and negative cases of XOR. After Goodfellow et al. (2016).

Speech and Language Processing (Jurafsky and Martin 2004)

(Simplified) neural models architecture

- The *feed-forward* SkipGram model (Mikolov et al)
- Input: a word from the vocabulary
- Middle: two matrices and some matrix multiplication
- Output: a probability for each word in the vocabulary occurring *somewhere nearby* the input word



Pic from: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

- What are the "two matrices"?!

Lecture survey in the chat!