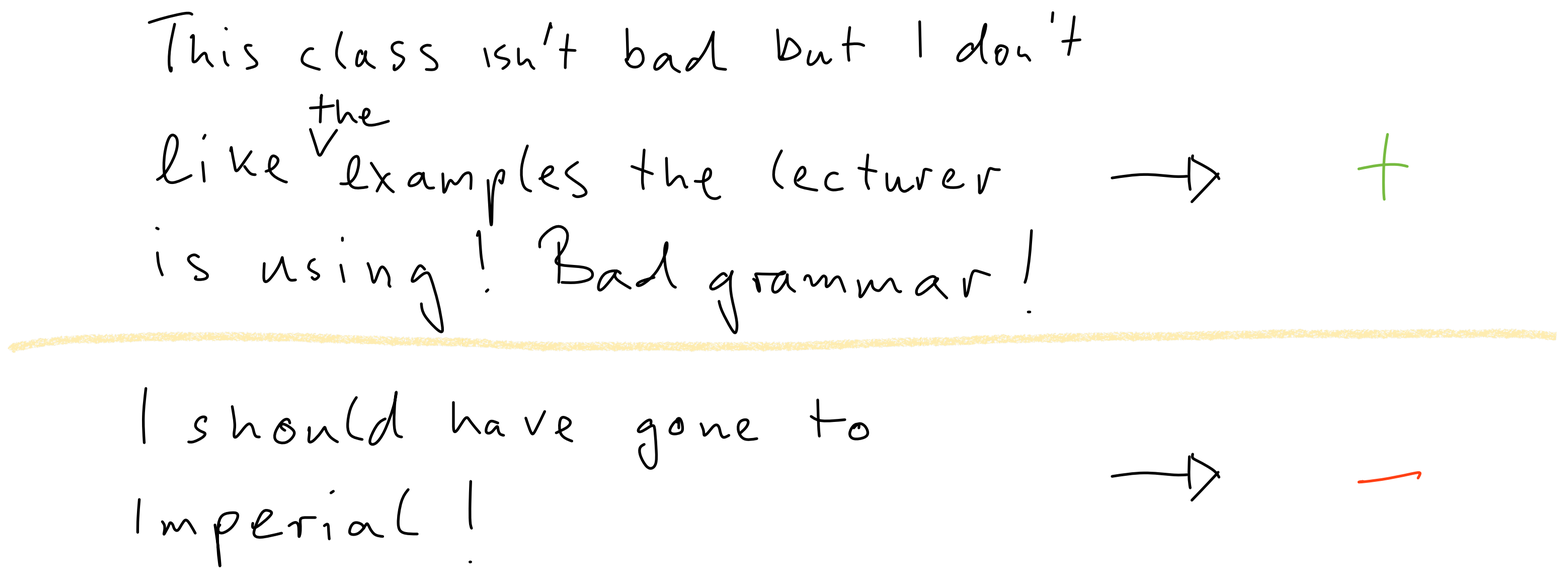
# Statistical NLP
# Manual Feature Engineering
# Linear Models and Classification

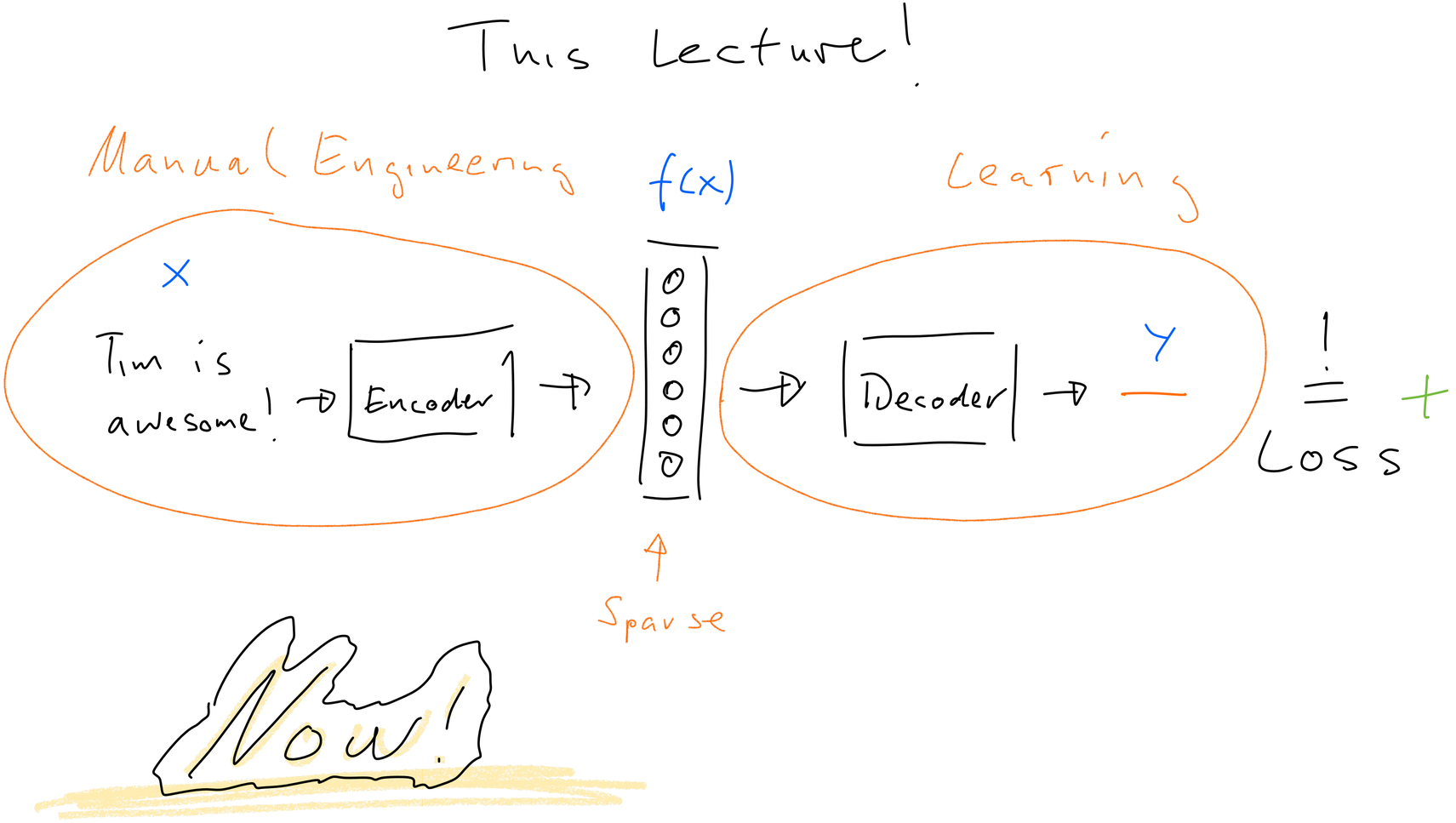
Tim Rocktäschel, Sebastian Riedel

# Sentiment Analysis




- Important task in research and academia (i.e. I see many papers, and get many such requests...)

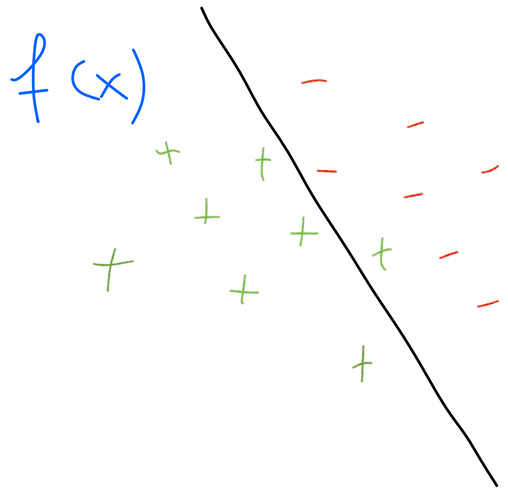
# The NLP Pipeline



# It’s all about the Representation

- Machine Learning is simple when you know how to represent the data

- Eg. the representation separates positive from negative text



# Tokenisation

Machine sees string as sequence of characters, no sense of "words"

Mr. Jones and me.

First step: break up string into tokens

"Easy" for English!? Use whitespace plus a few rules...

今日もしないといけない。 for Japanese?

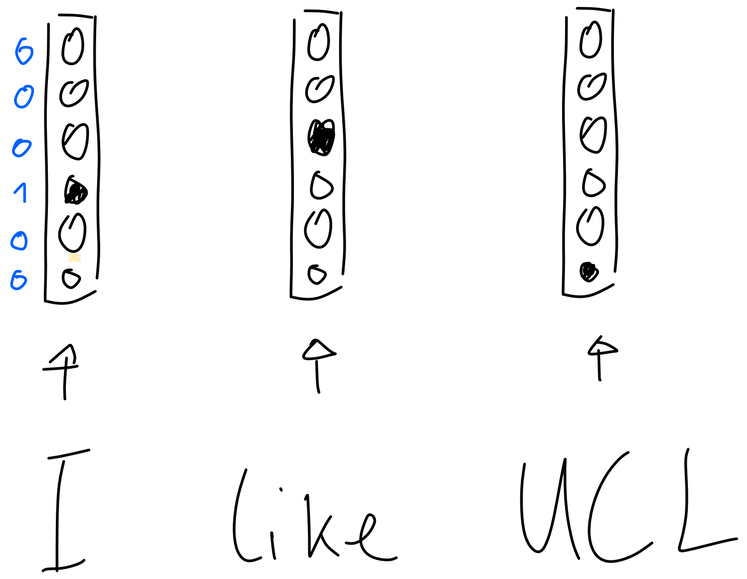
today too don't do it if doesn't work

# In other Domains

We developed a nanocarrier system of herceptin-conjugated nanoparticles of d-alpha-tocopheryl-co-poly(ethylene glycol) 1000 succinate (TPGS)-cisplatin prodrug ...

Need rules and/or machine learning

# Representing Words with One-Hot Vectors




word

$f(w) \in \mathbb{R}^m$

$$f_i(w) = \begin{cases} 1 & \text{if } w = V_i \\ 0 & \text{otherwise} \end{cases}$$

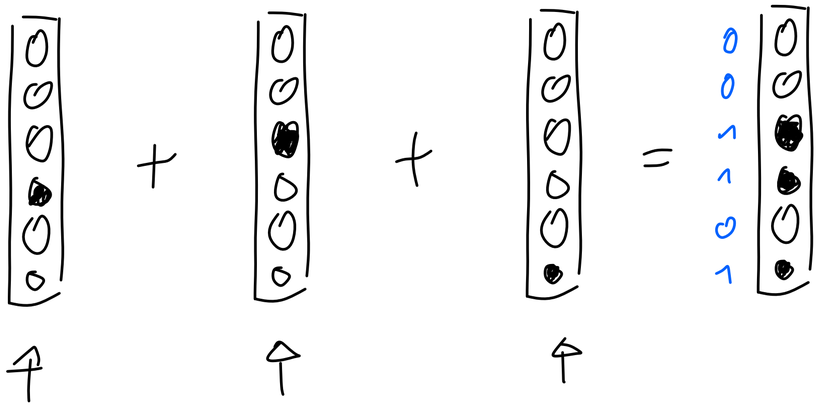
$V =$

| | |
|---|---|
| 0 | a |
| 1 | the |
| 2 | at |
| ⋮ | ⋮ |
| m | York |

Vocabulary

# Representing Sentences

Sum Pooling



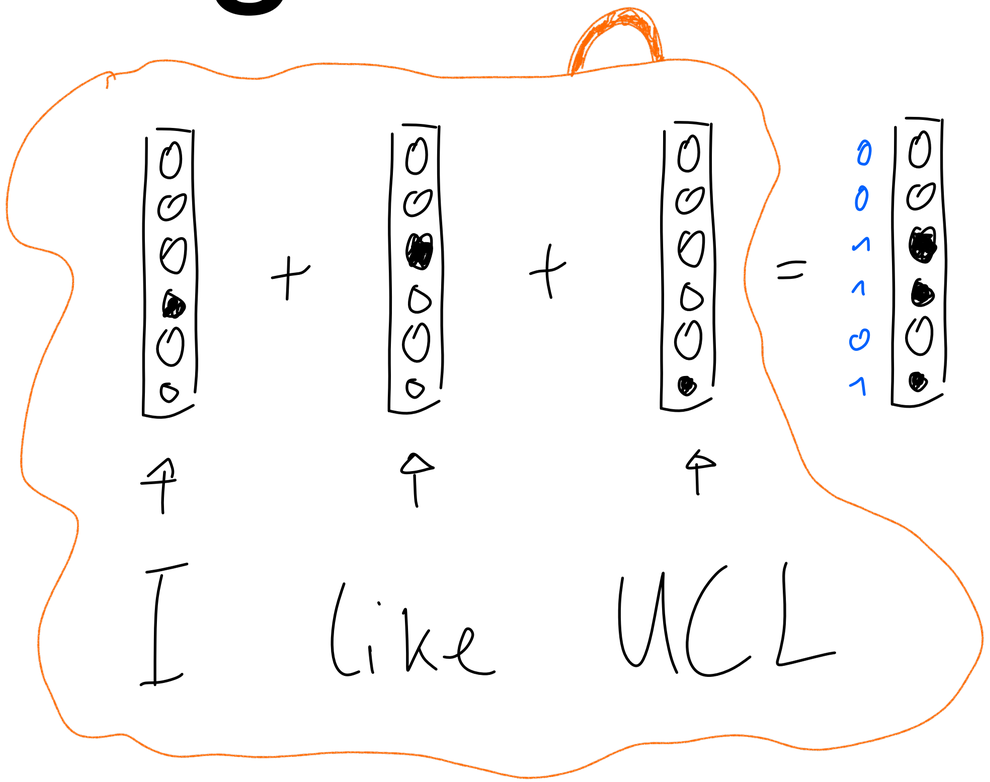

sentence

$f(x) \in \mathbb{R}^m$

token representation

$$f(x) = \sum_{w \in x} f(w)$$

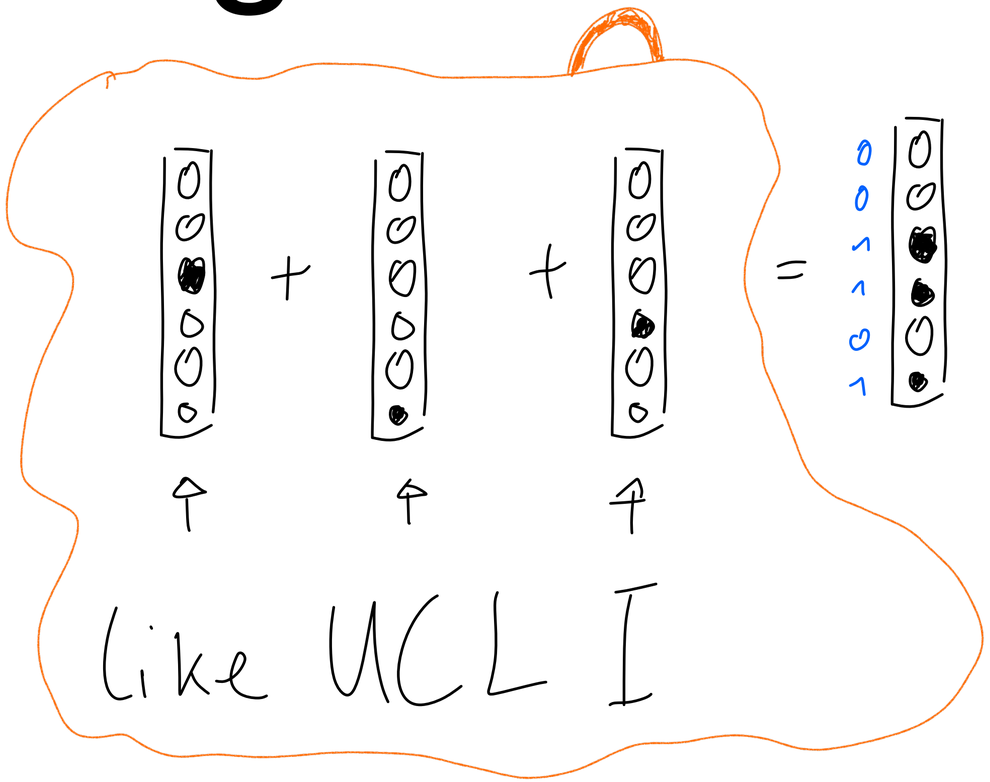
# “Bag of Words”




$f(x) \in \mathbb{R}^m$

$$f(x) = \sum_{w \in x} f(w)$$

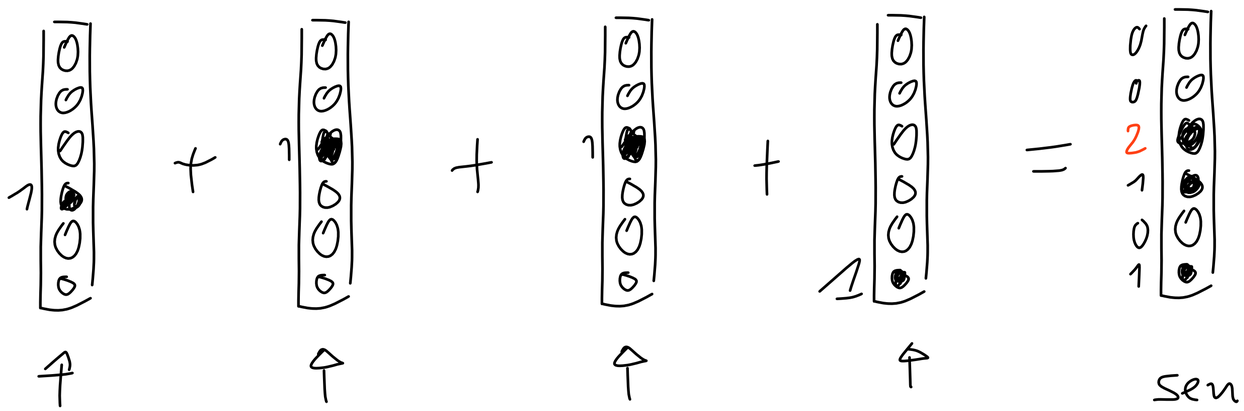
# “Bag of Words”




$f(x) \in \mathbb{R}^m$

$$f(x) = \sum_{w \in x} f(w)$$

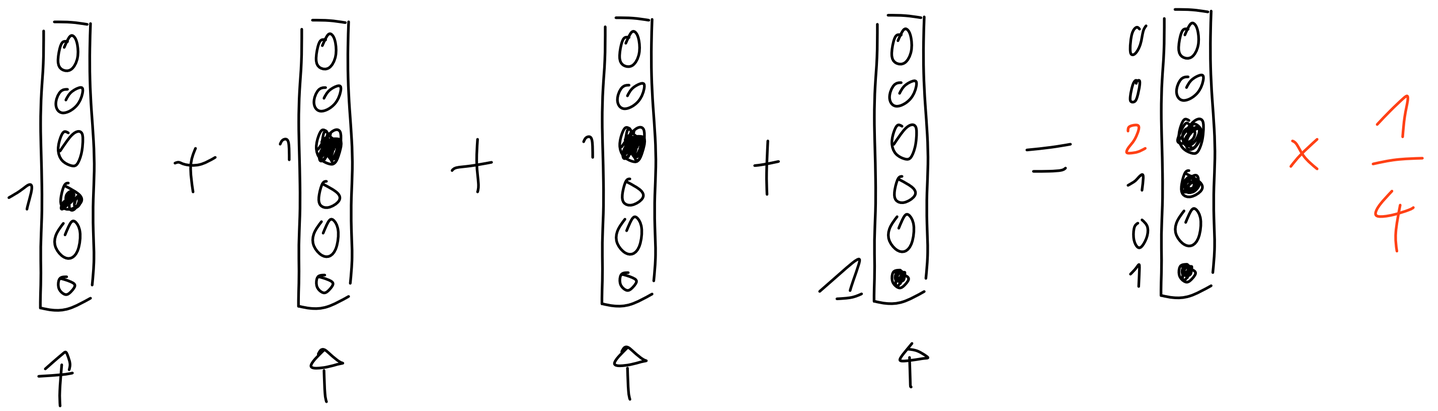
# Aggregation




sensitive to sentence length

Sum Pooling

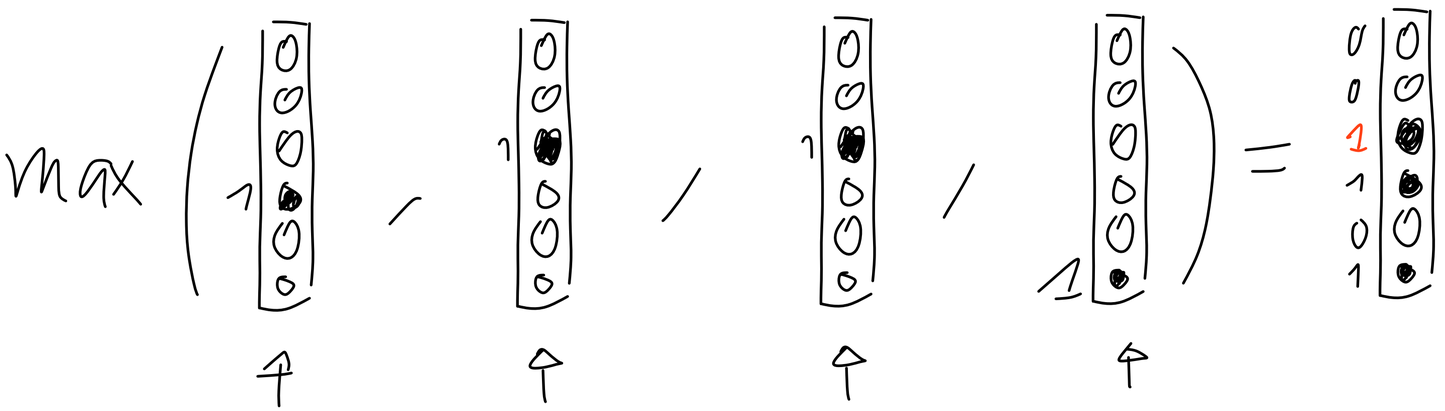
# Aggregation




I, like, like UCL

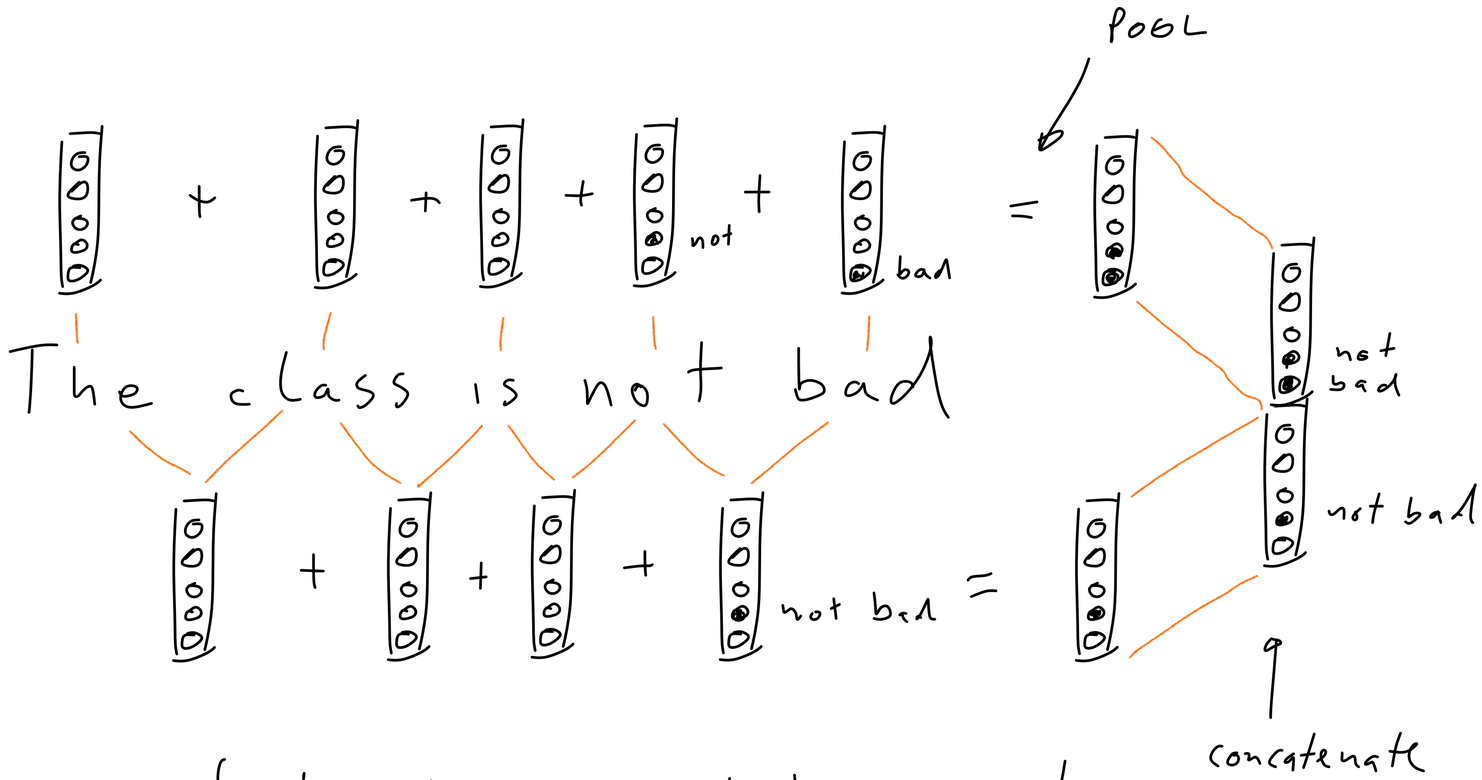
Mean Pooling

# Aggregation




Max Pooling

# More Engineering




unigram features may not be enough

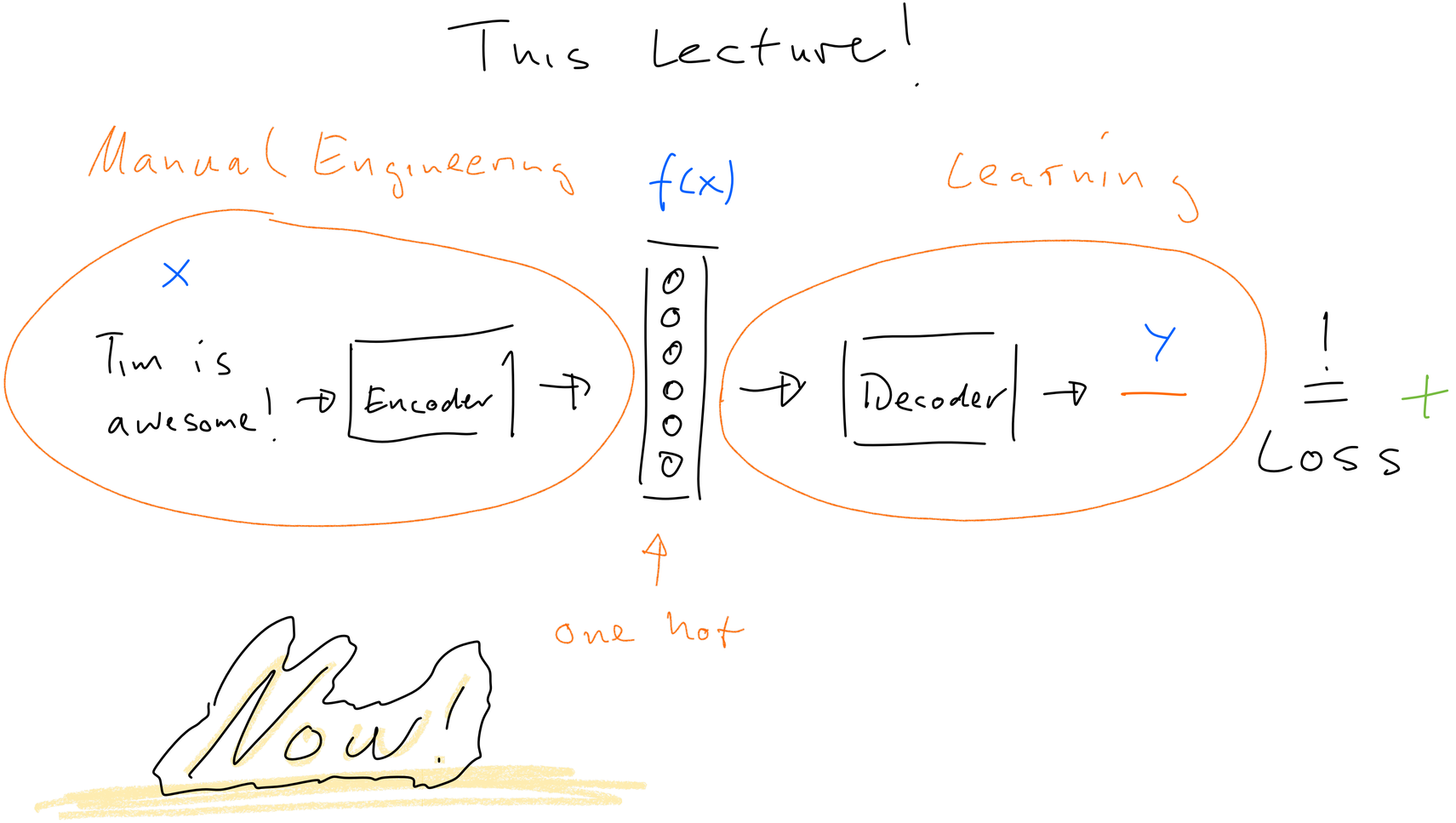
engineer more features! E.g. bigramm!

# More Ideas?

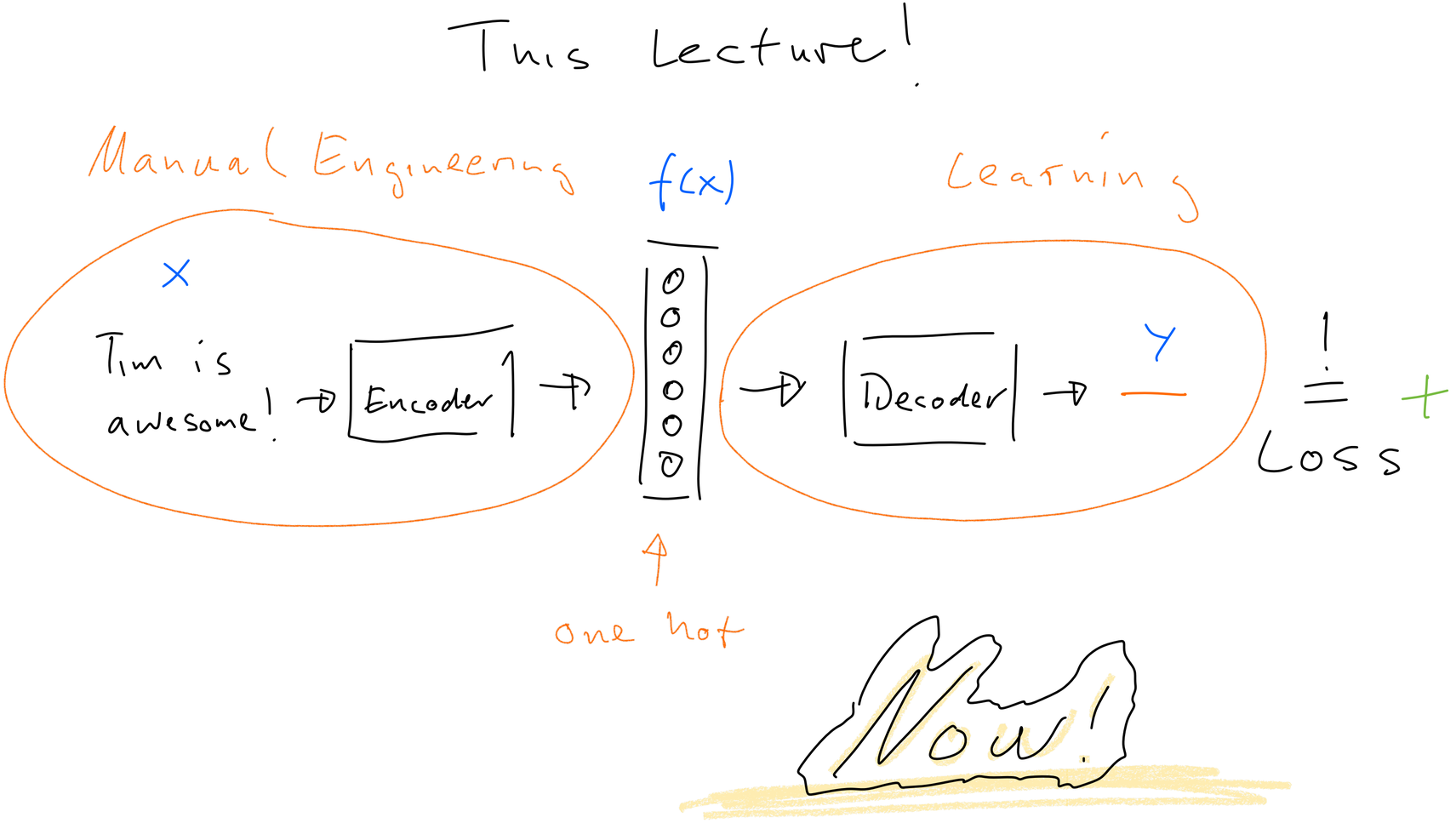
1. Use dictionaries ?
2. Use syntax ?
3. Preprocessing ?

⋮

# The NLP Pipeline



# The NLP Pipeline

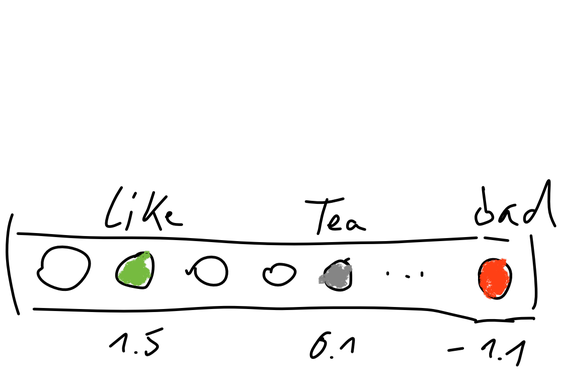


# Linear Classification

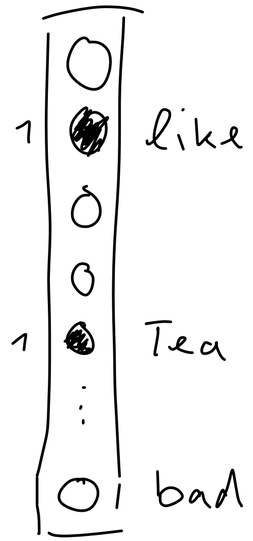
① Assign score to +

$\theta^T$ × $f(x)$ = $s_\theta(x)$

class weights | representation | score/logits

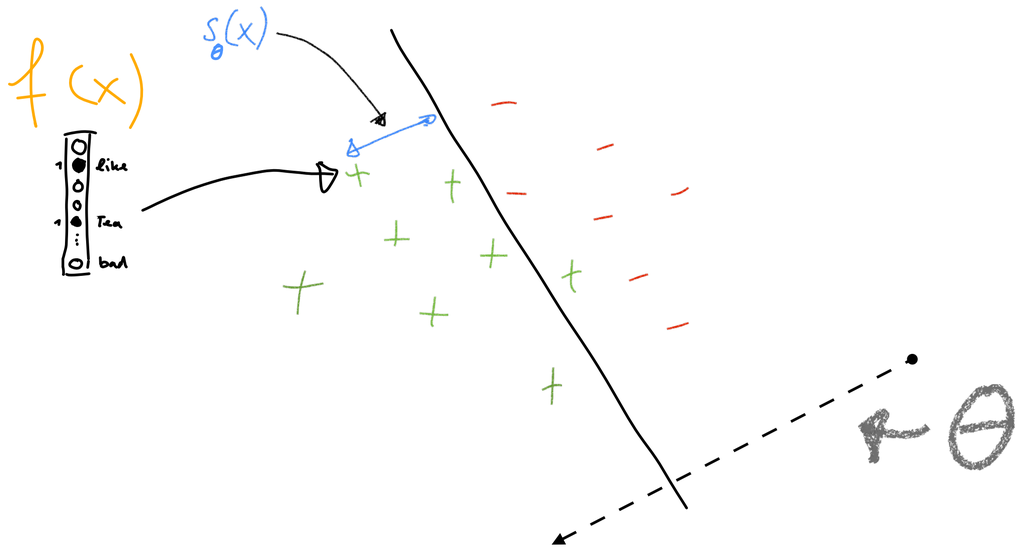



×



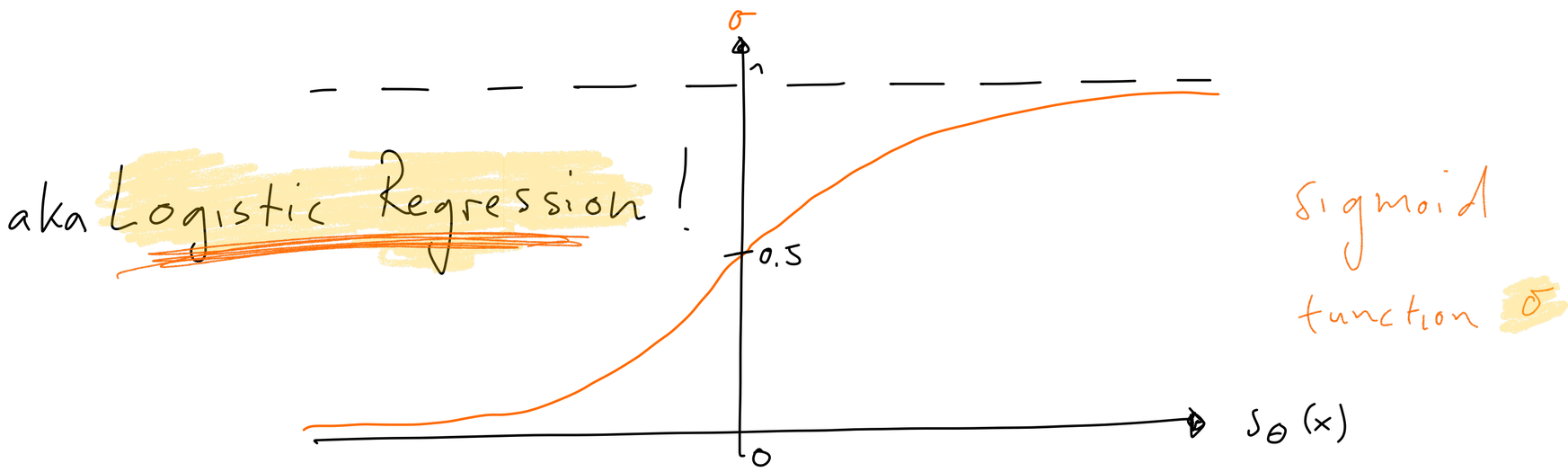

= 1.5 × 1 + 0.1 × 1 = 1.6

# Decision Boundary



# Probabilistic Output

② Assign probabilities to +, −




aka Logistic Regression!

sigmoid function σ

$$p_\theta(y = + \mid x) = \sigma\left(s_\theta(x)\right) = \frac{e^{s_\theta(x)}}{1 + e^{s_\theta(x)}}$$

$$p_\theta(y = - \mid x) = 1 - p_\theta(y = + \mid x)$$

# Search

③ Choose label with highest probability

Easy for two classes:

① calculate $p_\theta(+ \mid x)$

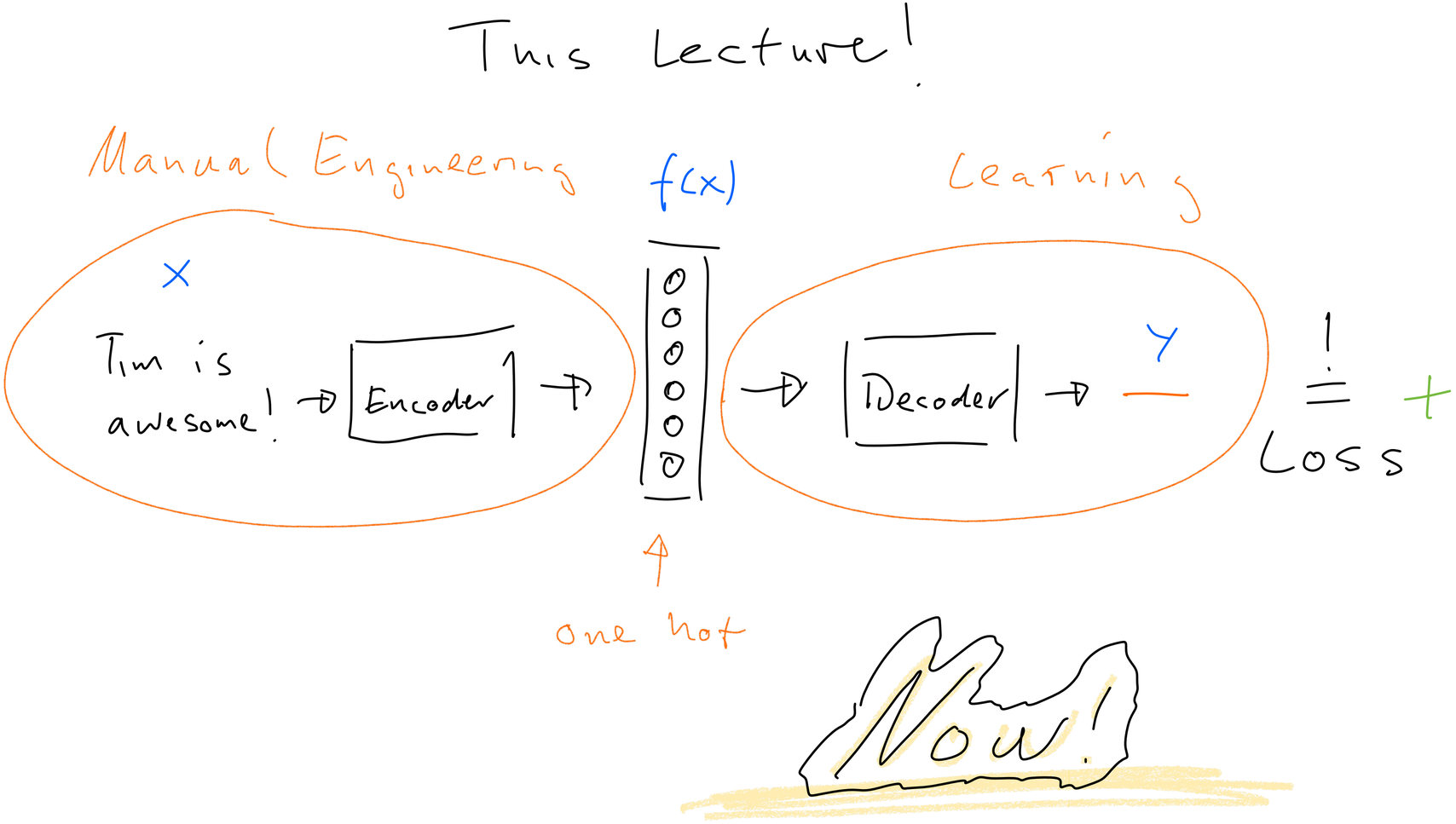
② " $p_\theta(- \mid x)$

③ choose higher one

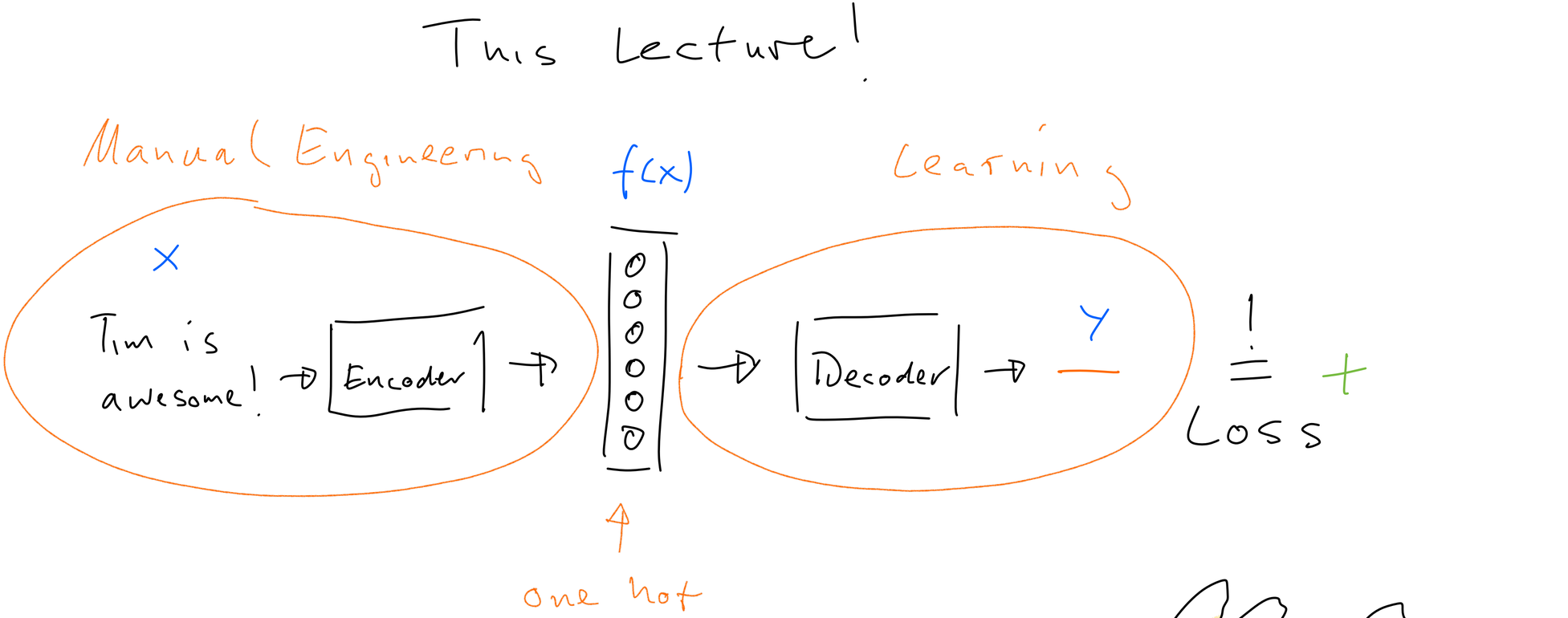
$$y^* = \underset{y \in \{+,-\}}{\operatorname{argmax}} \; p_\theta(y \mid x)$$

But consider Machine Translation!

# The NLP Pipeline



# The NLP Pipeline




# Training Loss

Dataset

$$D = (x_1, y_1) \dots (x_n, y_n)$$

Loss

$$L(D, \theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, y_i, \theta)$$

per instance loss

$$\ell(x, y, \theta) = -\log p_\theta(y \mid x)$$

Conditional Log Likelihood

# Cross Entropy Loss

let's think of label as distribution $\tilde{p}_i(y)$

$$\tilde{p}_i(y) = \begin{cases} 1 & \text{if } y_i = + \\ 0 & \text{otherwise} \end{cases}$$

then

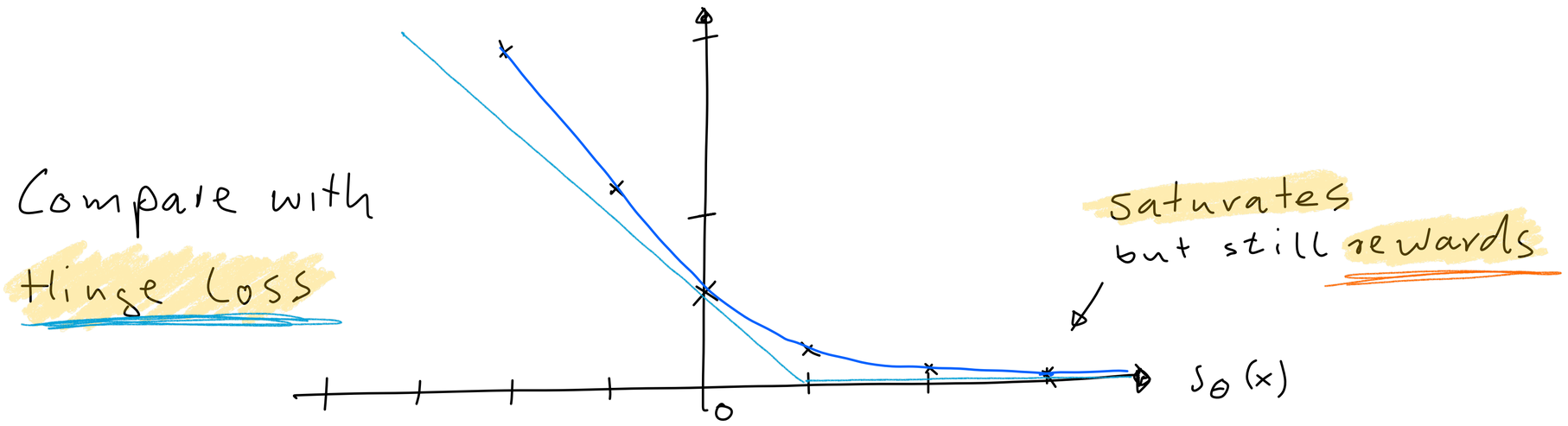
$$\ell(x_i, \tilde{p}_i, \theta) = -\sum_{y \in \{+,-\}} \tilde{p}_i(y) \cdot \log p_\theta(y|x)$$

Cross Entropy $H(\tilde{p}, p)$

can be generalized to soft labels!

# Intuition

$$\ell(s_\theta(x), +) = -\log\left(\frac{e^{s_\theta(x)}}{1+e^{s_\theta(x)}}\right)$$

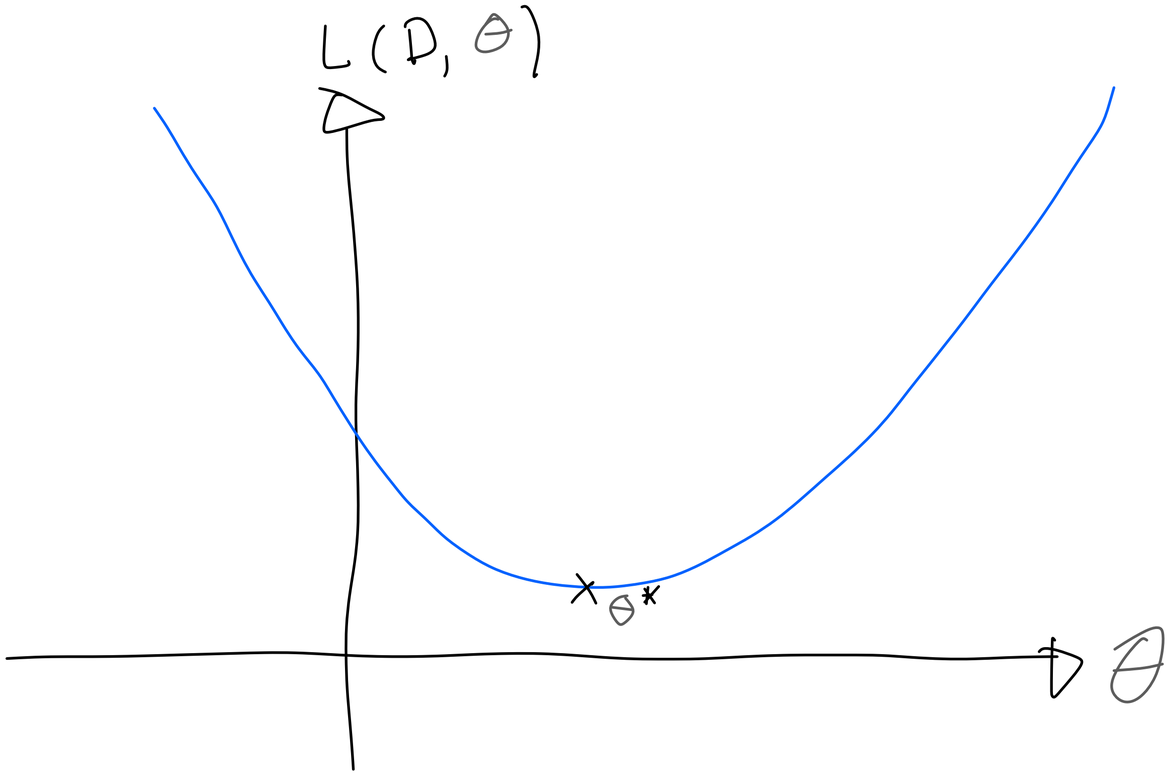


# Training

best parameter

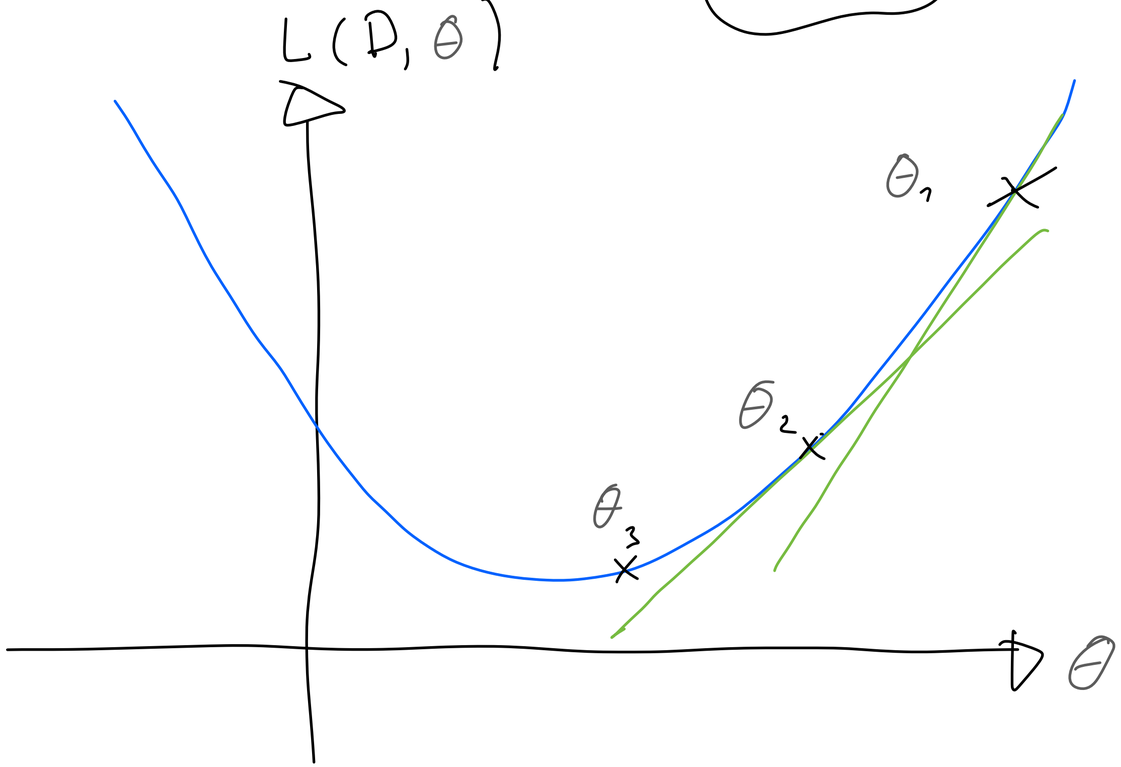
$$\theta^* = \underset{\theta \in \mathbb{R}^m}{\operatorname{argmin}} \; L(D, \theta)$$

Loss



# Gradient Descent

$\theta_0$ = random ; i = 0
repeat until convergence
$\theta_{i+1} = \theta_i - \alpha \nabla_\theta L(D, \theta_i)$
i = i + 1



# Stochastic Gradient Descent

$$\nabla_w L(P, \theta) = \nabla_w \frac{1}{n}\left[\ell(x_1, y_1, \theta) + \dots + \ell(x_n, y_n, \theta)\right]$$

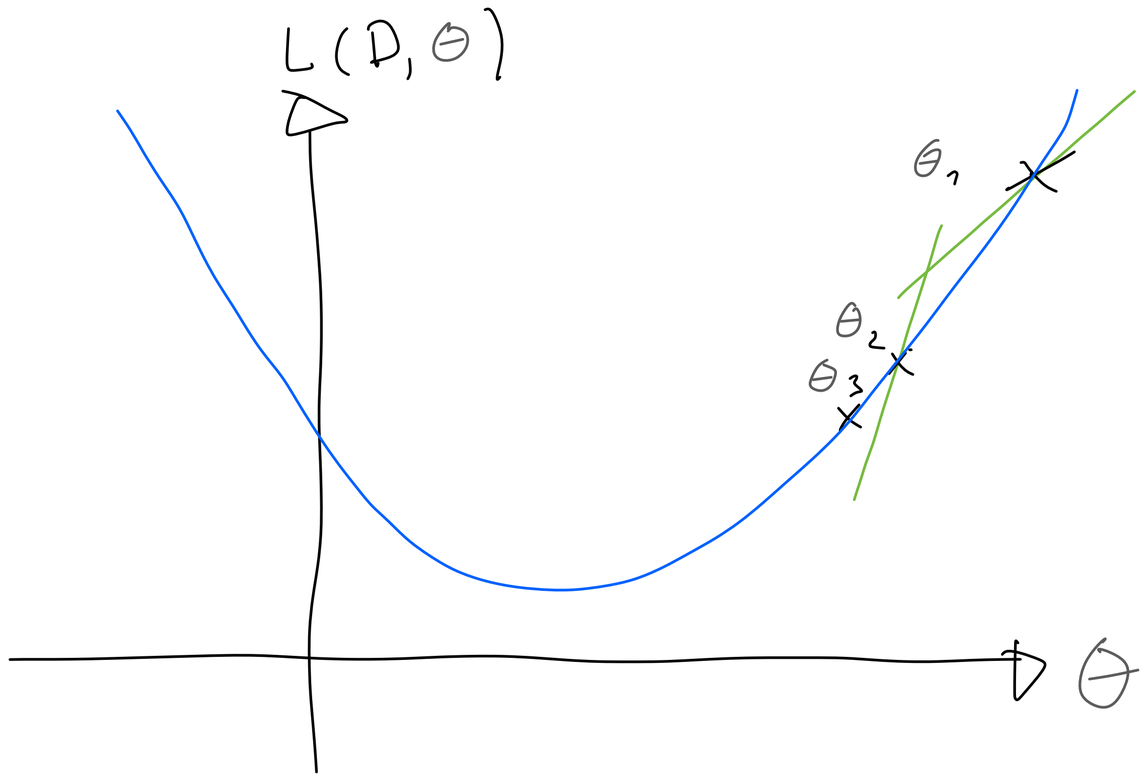
long time spent w/o update using stale weights

Instead: approximate gradient via sampling single instance $j$

$$\nabla_w L(P, \theta) \approx \nabla_w \ell(x_j, y_j, \theta)$$

# Stochastic Gradient Descent

$$\theta_{i+1} = \theta_i - \alpha \nabla_\theta \ell(x_j, y_j, \theta_i)$$



# Regularisation

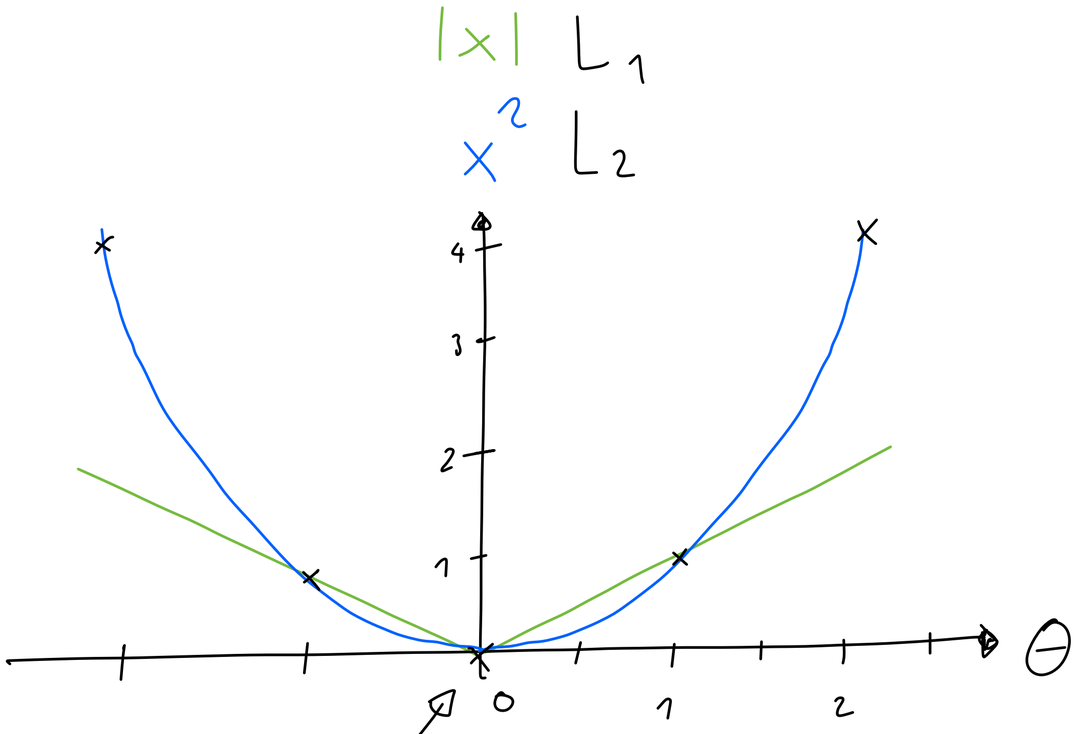

which is better?

| | | |
|---|---|---|
| $L(\theta)$ | 0.02 | 0.02 |
| $\|\theta\|_2^2$ | 2.25 | 30.55 |

$$\tilde{L}_\alpha(\theta) = L(\theta) + \alpha \|\theta\|_2^2$$

$L_2$ regularization

# L2 vs L1




gradient of $x^2$ ?

weights are never zero

Bad for memory usage

# Material

- Stat-NLP-Book
  - Tokenization
  - Text Classification
- Goldberg, chapter 3
- JM, chapter 2, 5