Foundations for Natural Language Processing

Neural Classifiers

Ivan Titov

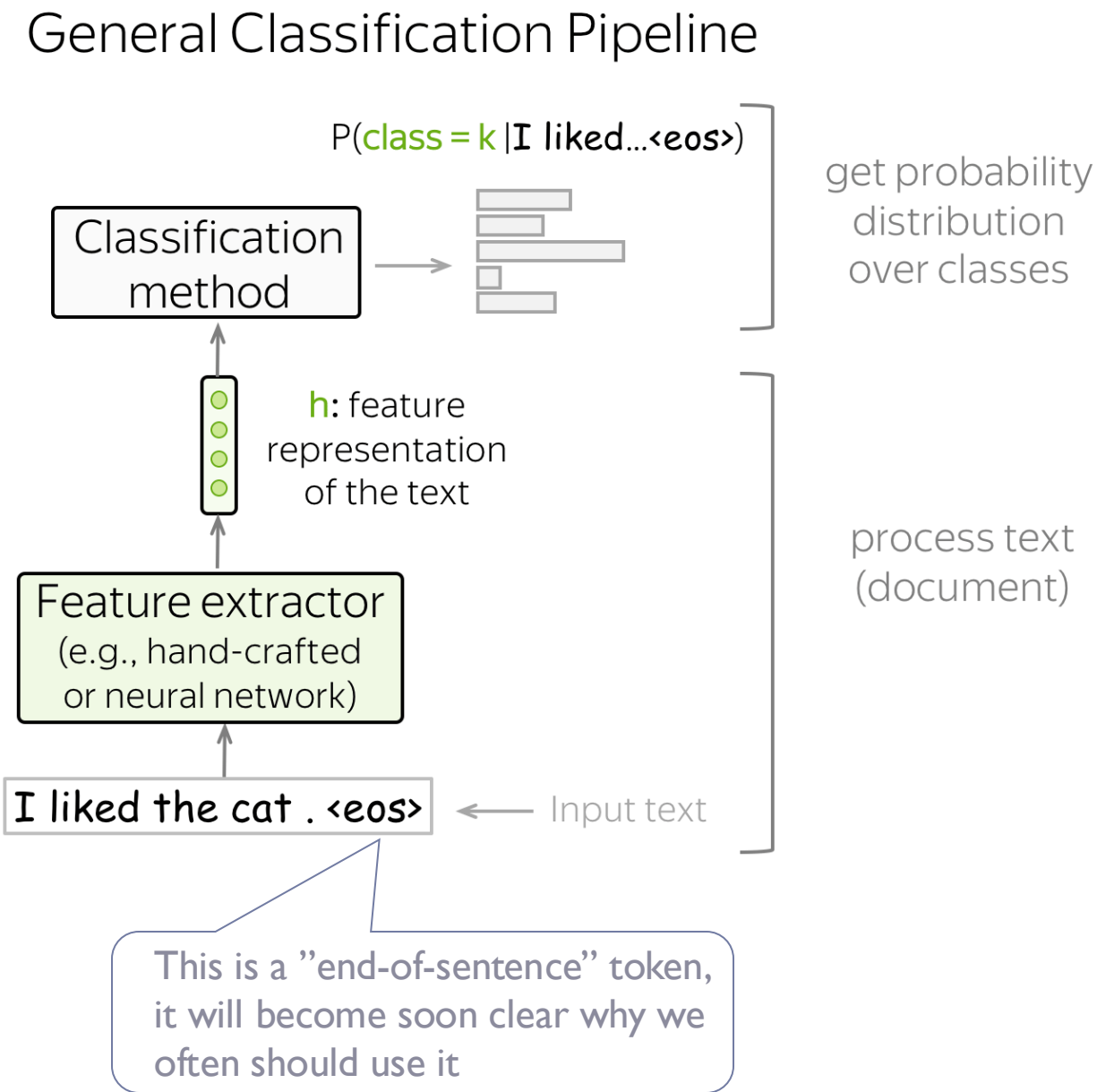
with some graphics / materials
from Lena Voita and Edoardo Ponti

School of informatics

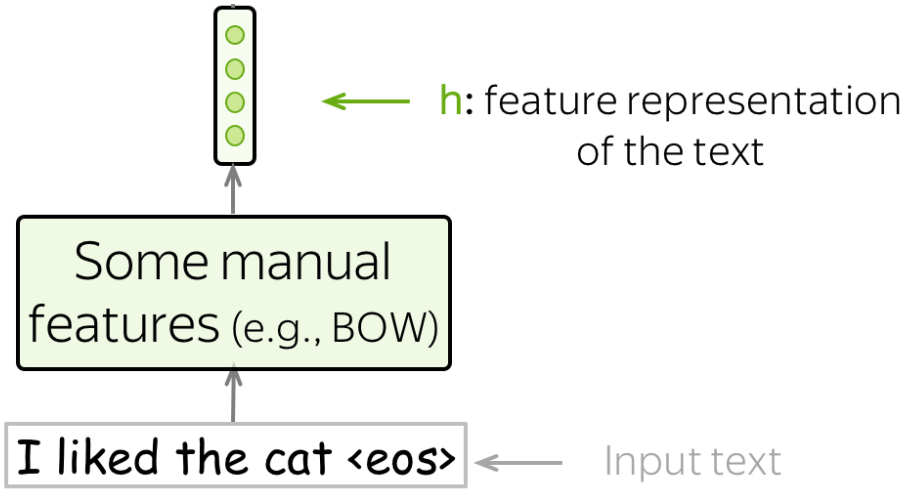
Plan for today

- Discuss relation between logistics regression and Naïve Bayes
- Basic neural text classifiers

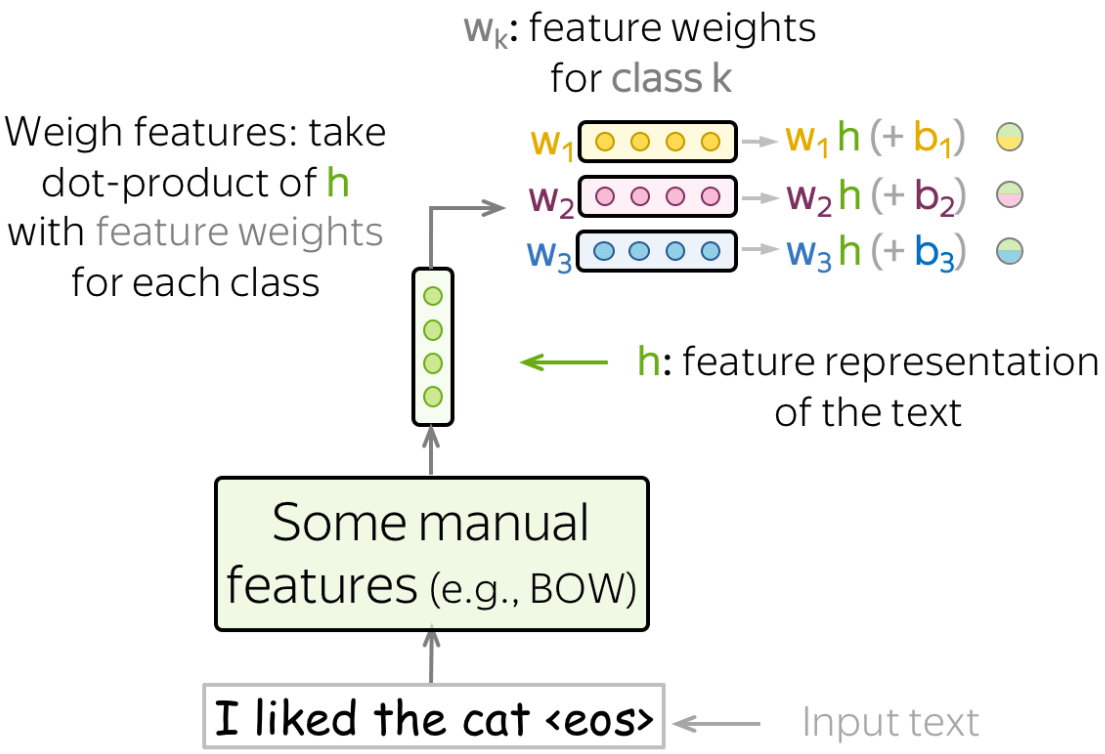
Classification



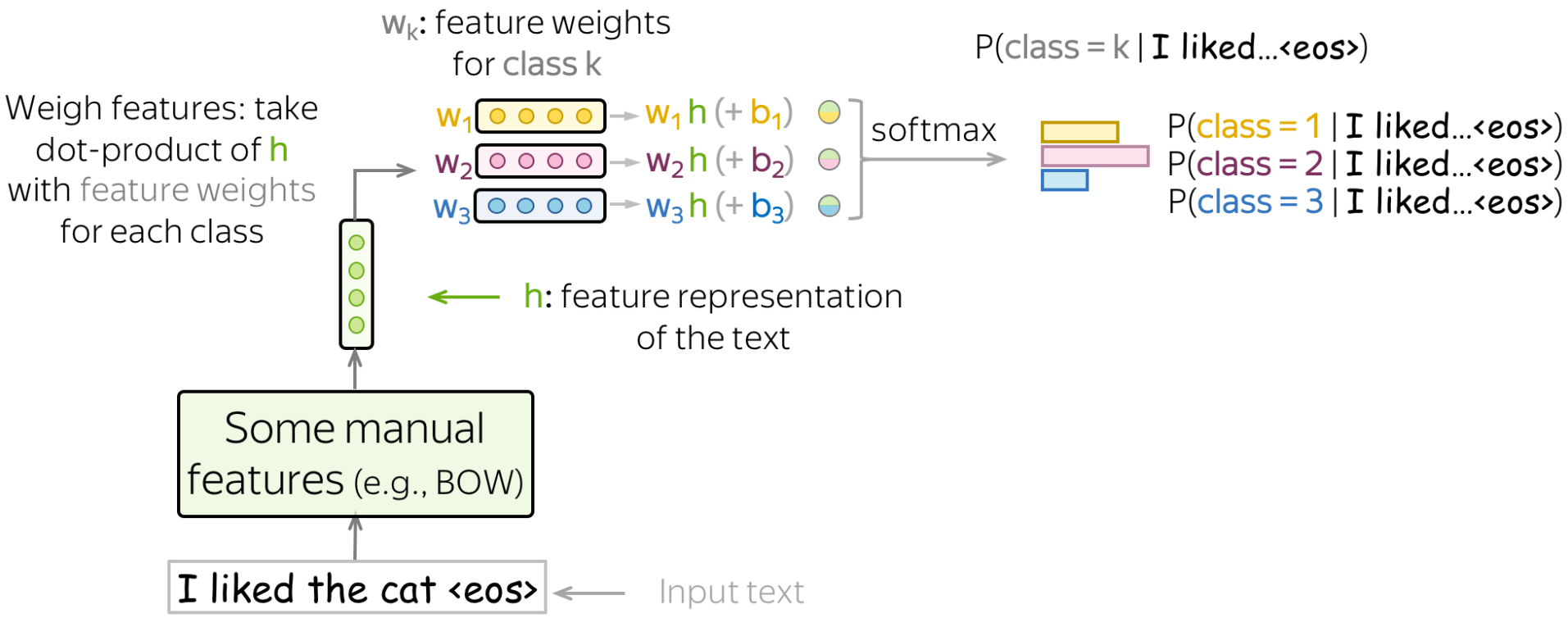
Recap: Logistic regression



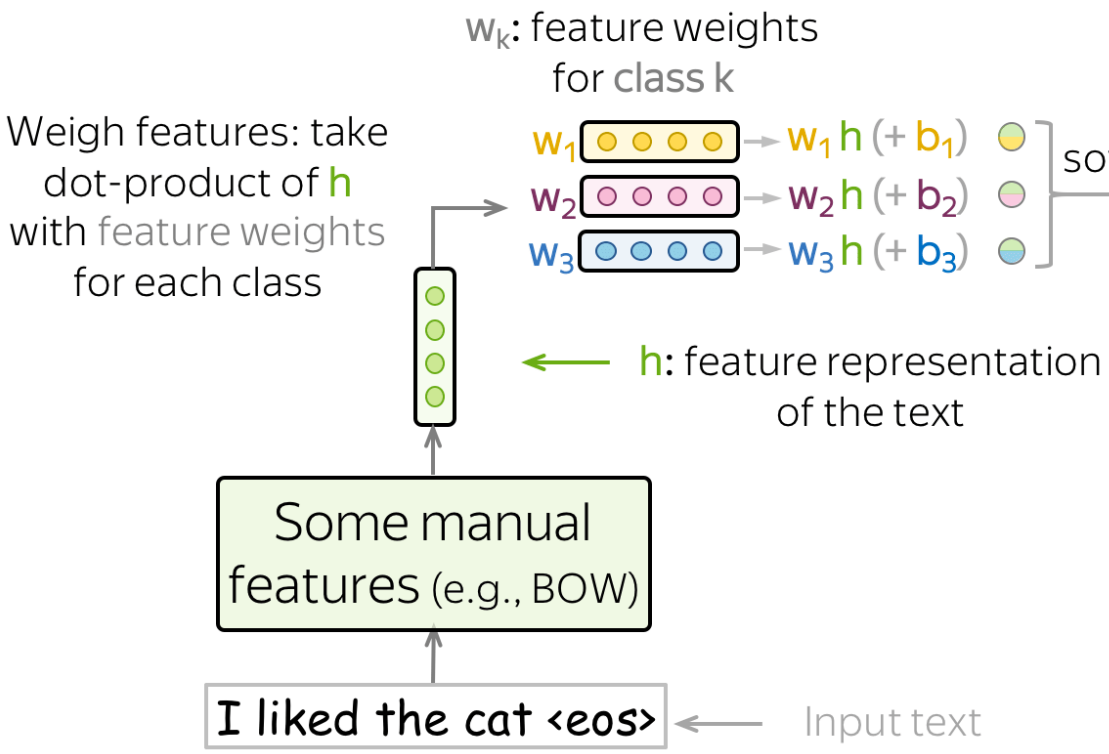
Recap: Logistic regression



Recap: Logistic regression



Recap: Logistic regression



Softmax squashes a vector of scores z into the range [0, 1] and normalizes them to sum to 1 (thus, yielding a *Categorical distribution*)

$$\text{softmax}(\mathbf{z}) = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}$$

for the i-th output dimension

Recap: mini-batch gradient descent

$w \leftarrow \text{Random}()$

repeat

$B \leftarrow \text{RandomSubset}([1, \ldots, N])$

$w \leftarrow w + \eta \cdot \nabla_w \sum_{j \in B} \log P(y^{(j)}|x^{(j)})$

until Converged()

Choosing a "**batch**": Indexes of a random subset of examples (e.g., choose 10 random examples)

Sum only over examples in the current batch

Gradient for LR: recap

$$\frac{d}{dw_l^{(k)}} \log P(y^{(j)}|x^{(j)}) = \text{(I)} - \text{(II)}$$

$$= [y^{(j)} = k] \cdot f_l(x^{(j)}) - P(y = k|x^{(j)}) \cdot f_l(x^{(j)})$$

$$= \underbrace{([y^{(j)} = k] - P(y = k|x^{(j)}))}_{\alpha} f_l(x^{(j)})$$

Close to zero if the classifier confidently predicts the correct class

$$P(y = k|x^{(j)}) \approx \begin{cases} 1 & \text{if } y^{(j)} = k \\ 0 & \text{otherwise} \end{cases}$$

If the classifier is already confident, gradient is close to 0 and no learning is happening

Relation to Naïve Bayes

Naïve Bayes can also be interpreted as a linear classifier, as decision boundaries are linear when expressed in terms of log-probabilities of the features.

Relation to Naïve Bayes

f_1 : `contains('ski')`

$$w_1^{(1)} = \log \hat{P}(\text{`ski'} | c = 1)$$
$$w_1^{(2)} = \log \hat{P}(\text{`ski'} | c = 2)$$
$$w_1^{(3)} = \log \hat{P}(\text{`ski'} | c = 3)$$

f_2 : `contains('beach')`

$$w_2^{(1)} = \log \hat{P}(\text{`beach'} | c = 1)$$
$$w_2^{(2)} = \log \hat{P}(\text{`beach'} | c = 2)$$
$$w_2^{(3)} = \log \hat{P}(\text{`beach'} | c = 3)$$

f_3 : **1**

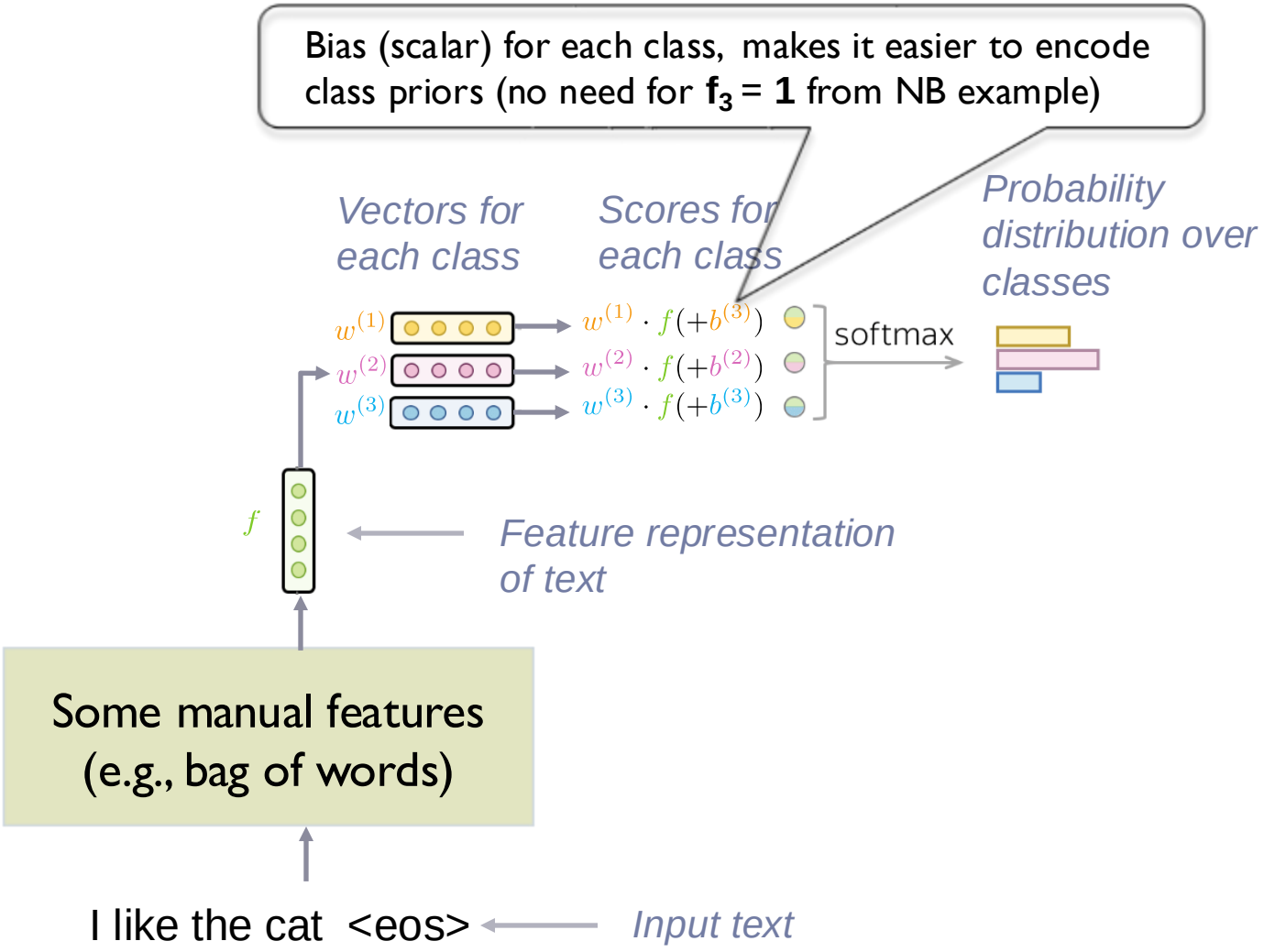
$$w_3^{(1)} = \log \hat{P}(c = 1)$$
$$w_3^{(2)} = \log \hat{P}(c = 2)$$
$$w_3^{(3)} = \log \hat{P}(c = 3)$$

Relation to Naïve Baves

If the features were truly conditionally independent (a condition that rarely holds in practice), NB would converge to the same decision boundary as LR, given enough training data.*

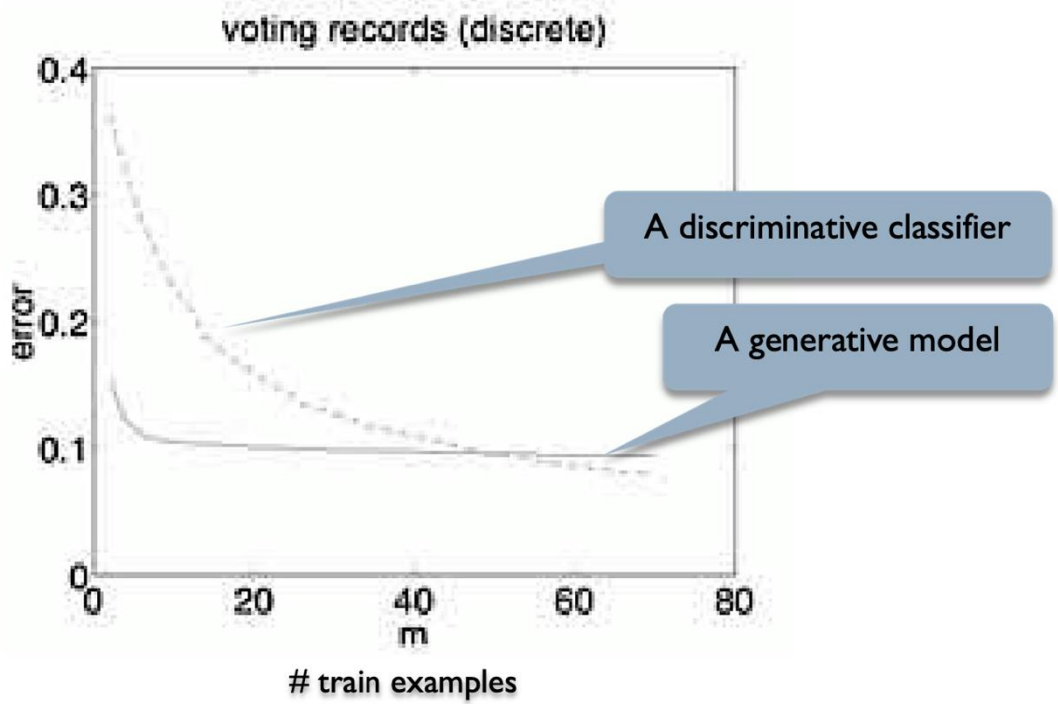
* i.e. models' predictions will be the same but not necessarily the weights corresponding to individual features are the same

Schematic view of logistic regression



Data efficiency of LR vs NB

- Theoretical results: generative classifiers converge faster with training set size to their optimal error [Ng & Jordan, NeurIPS 2001]
- Empirical:



Predicting Democrat vs Republican, based on voting records

Cons of LR

- Supervised MLE in generative models is easy: compute counts and normalize.
- Supervised CMLE in LR model not so easy
  - requires multiple iterations over the data to gradually improve weights (using gradient ascent).
  - each iteration computes $P(y^{(j)}|x^{(j)})$ for all j.
  - this can be time-consuming, especially if there are a large number of classes and/or thousands of features to extract from each training example.

Robustness of LR

- Imagine that in training there is one very frequent predictive feature
  - E.g., in training sentiment data contained emoticons but not at test time
- The model can quickly learn to rely on this feature
  - model is confident on examples with emoticons
  - the gradient on these examples gets close to zero
  - the model does not learn other features

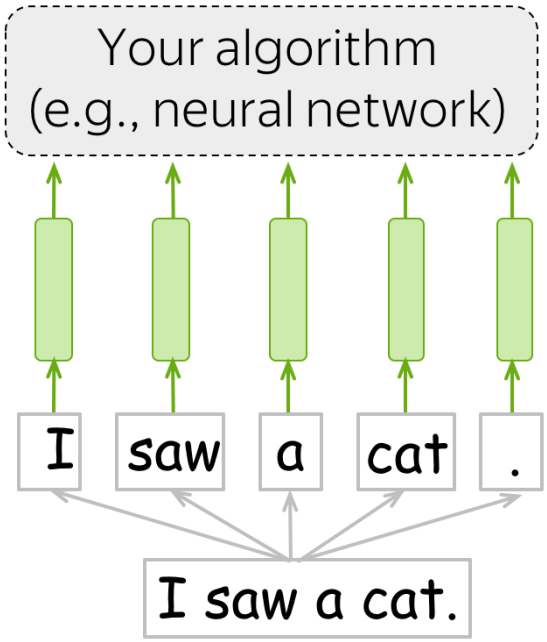
Robustness of LR (2)

- In LR, **a feature weight will depend on the presence of other predictive features**
- Naive Bayes will rely on all features
  - The weight of a feature is not affected by how predictive other features are
- This makes NB more robust that (**basic**) Logistic Regression when test data is (distributionally) different from train data

Plan for today

- Discuss relation between logistics regression and Naïve Bayes
- Basic neural network classifiers

Neural models and word embeddings



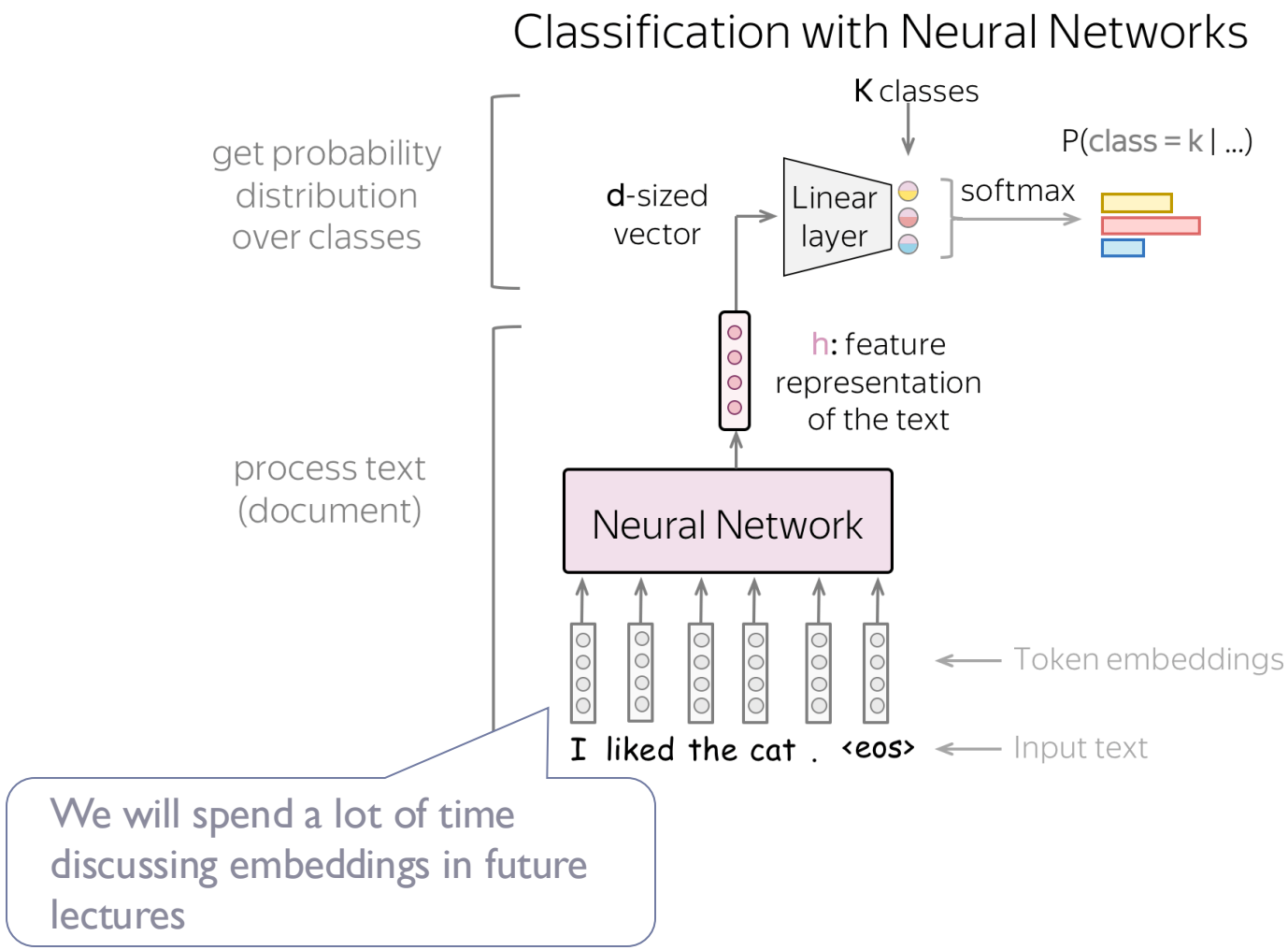
Any algorithm for solving a task

Word representation - vector
(input for your model/algorithm)

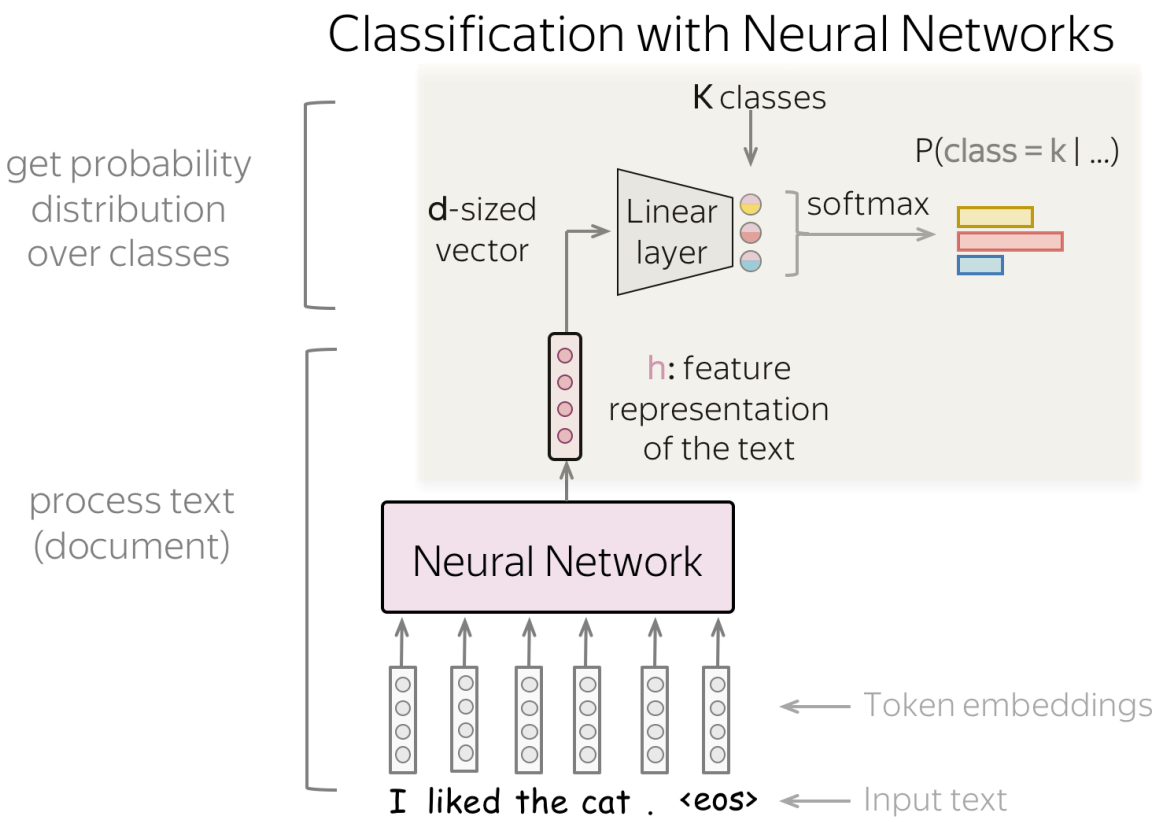
Sequence of tokens

Text (your input)

NN Classifier

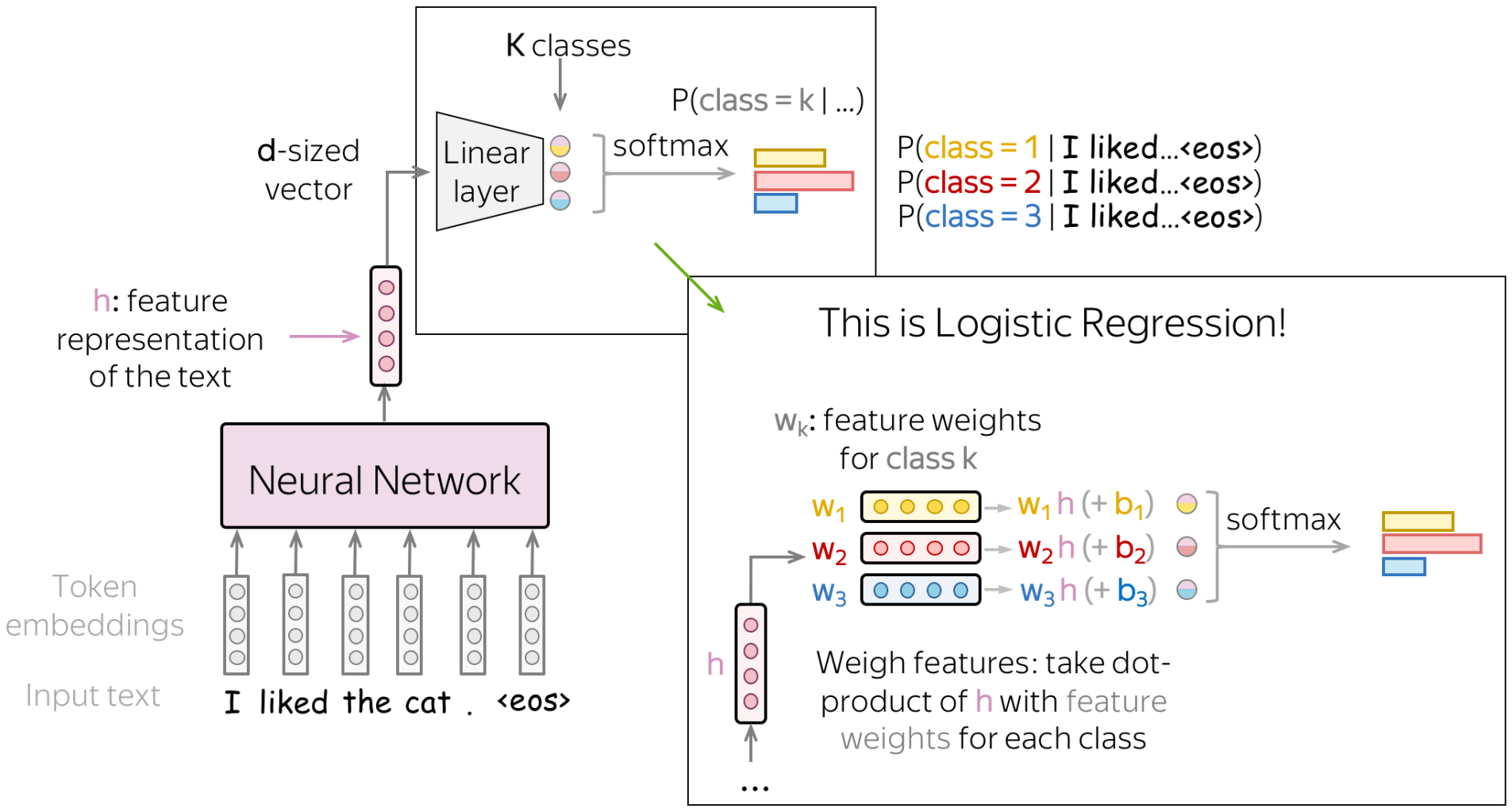


NN Classifier

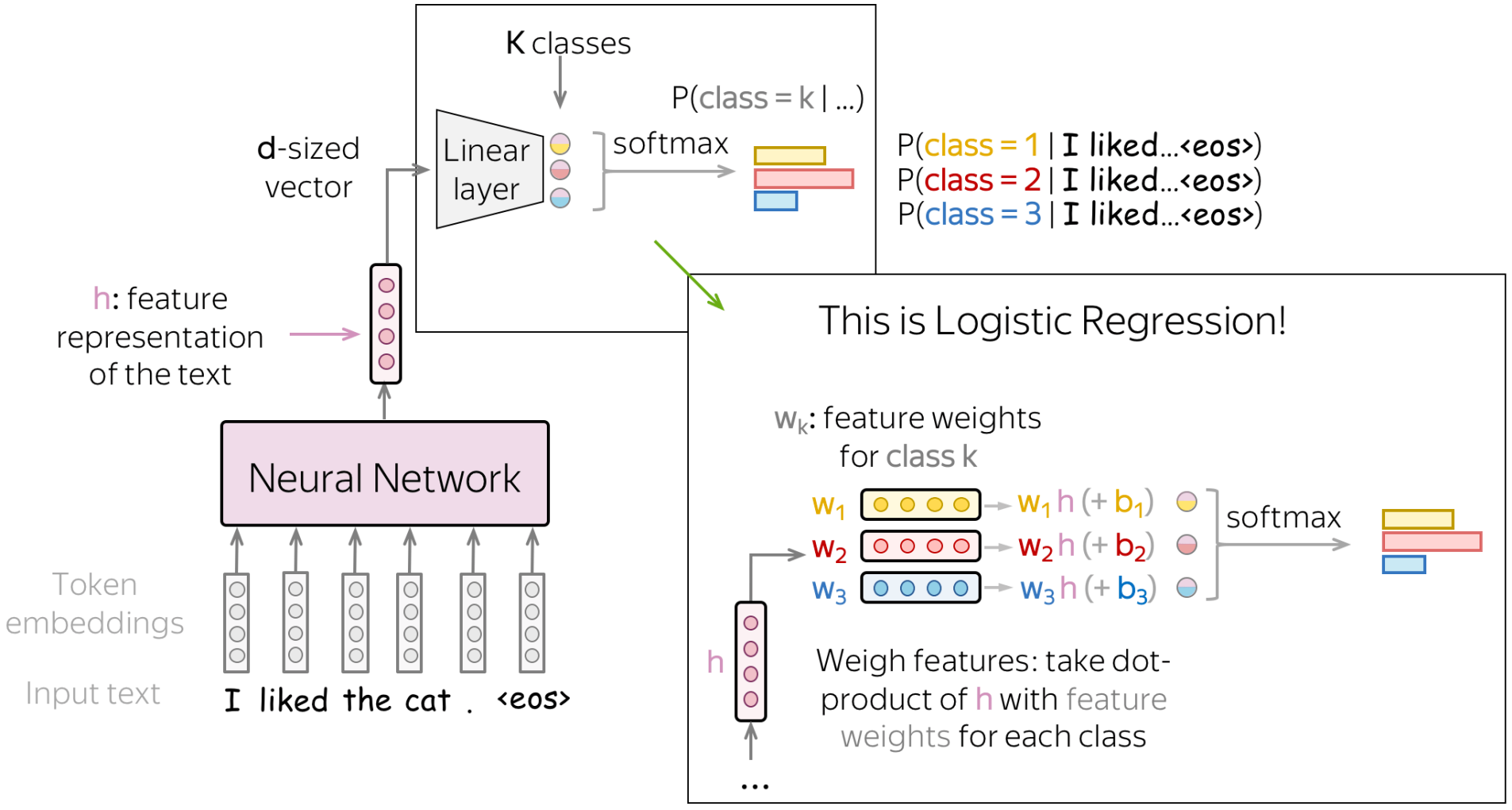


The highlighted part is the logistic regression!

NN Classifier

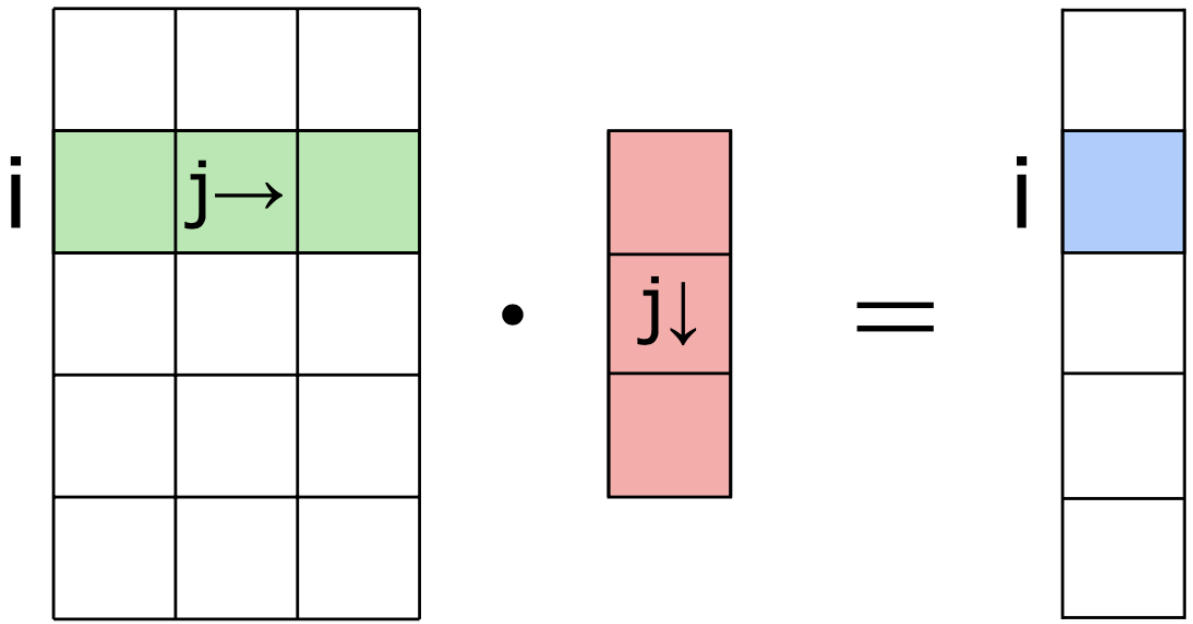


NN Classifier

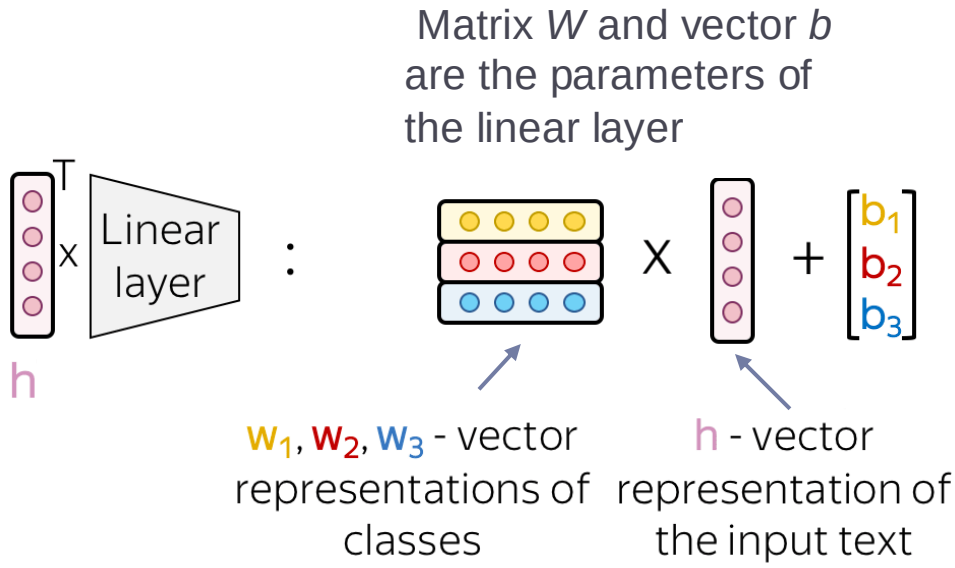


Intuition: the representation of the document points in the direction of the class representation

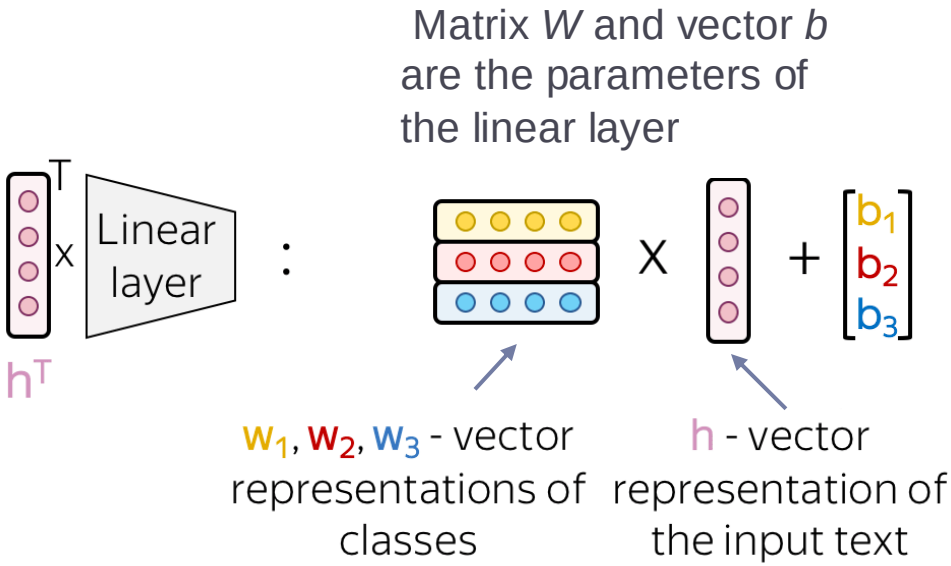
Reminder: Matrix-vector multiplication



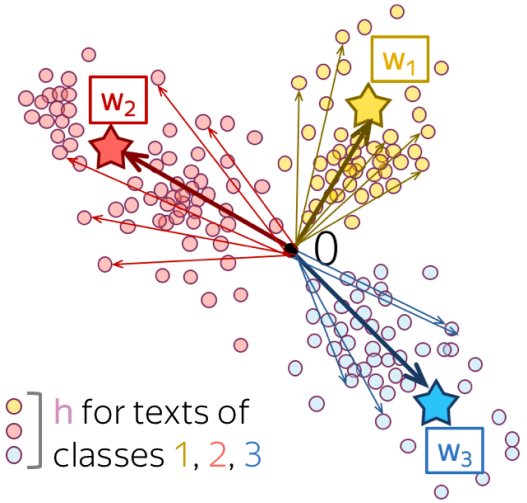
Representation of the document



Representation of the document



Intuition: the representation of the document points in the direction of the class representation



What do we optimize? (recap)

Optimize conditional log-likelihood, as with logistic regression, which is equivalent to using cross-entropy loss

Training example: **I liked the cat on the mat <eos>**

Label: k

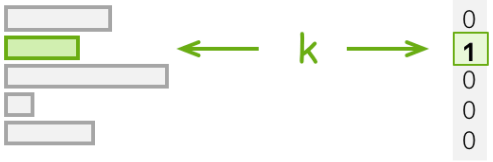
target

Model prediction:

P(class = i |I liked...<eos>)

Target:

p^*



Cross-entropy loss:

$$-\sum_{i=1}^{K} p_i^* \cdot \log P(y=i|x) \rightarrow min \qquad (p_k^* = 1, p_i^* = 0, i \neq k)$$

For one-hot targets, this is equivalent to

$$-\log P(y=k|x) \rightarrow min$$

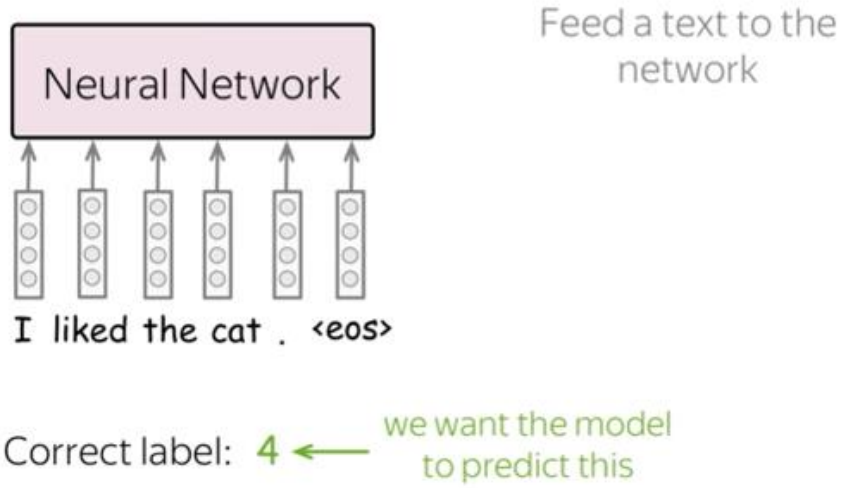
The target distribution is one-hot:

$$p^* = (0, \ldots, 0, 1, 0, \ldots)$$

Recall: we derived the gradient in the previous lecture

What do we optimize?

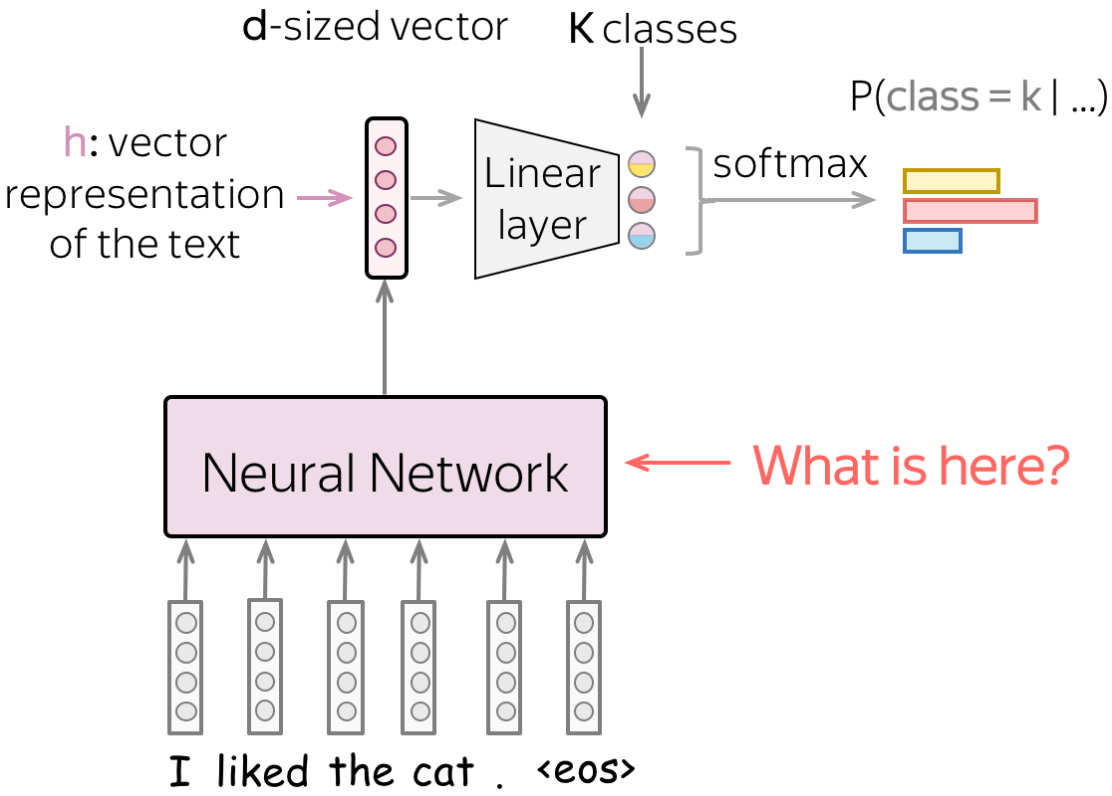
Optimize conditional log-likelihood, as with logistic regression



(video, not visible in pdf)

Recall, we derived last time that gradient updates to results in these decreases / increases

Neural models for text classification



The neuron

Most basic computational unit.

The input $\mathbf{x} \in \mathbb{R}^d$ is a vector with d dimensions.

The output $z \in \mathbb{R}$. This means that we have d inputs and 1 output.

The output is obtained as:

$$z = \sum_{i=1}^{d} w_i \, x_i + b = \mathbf{w}^\top \mathbf{x} + b$$

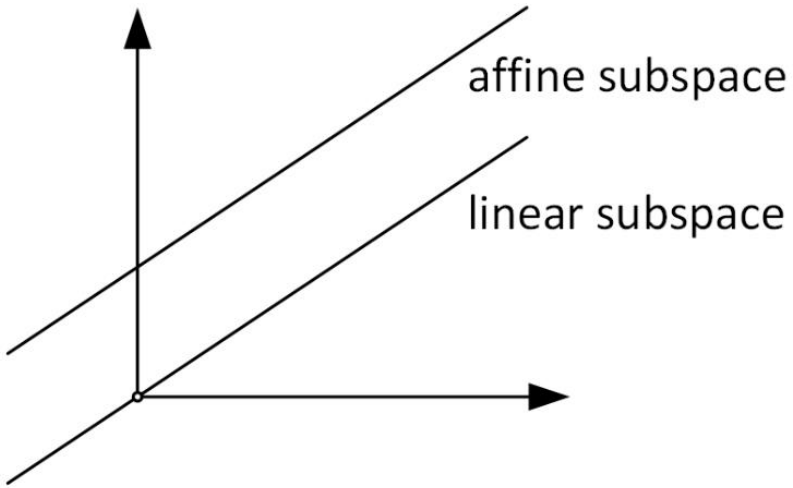
$\mathbf{w} \in \mathbb{R}^d$ are called the **weights**: they multiply each dimension of the input by its 'importance'.

$b \in \mathbb{R}$ is called the **bias** and provides an additive shift.

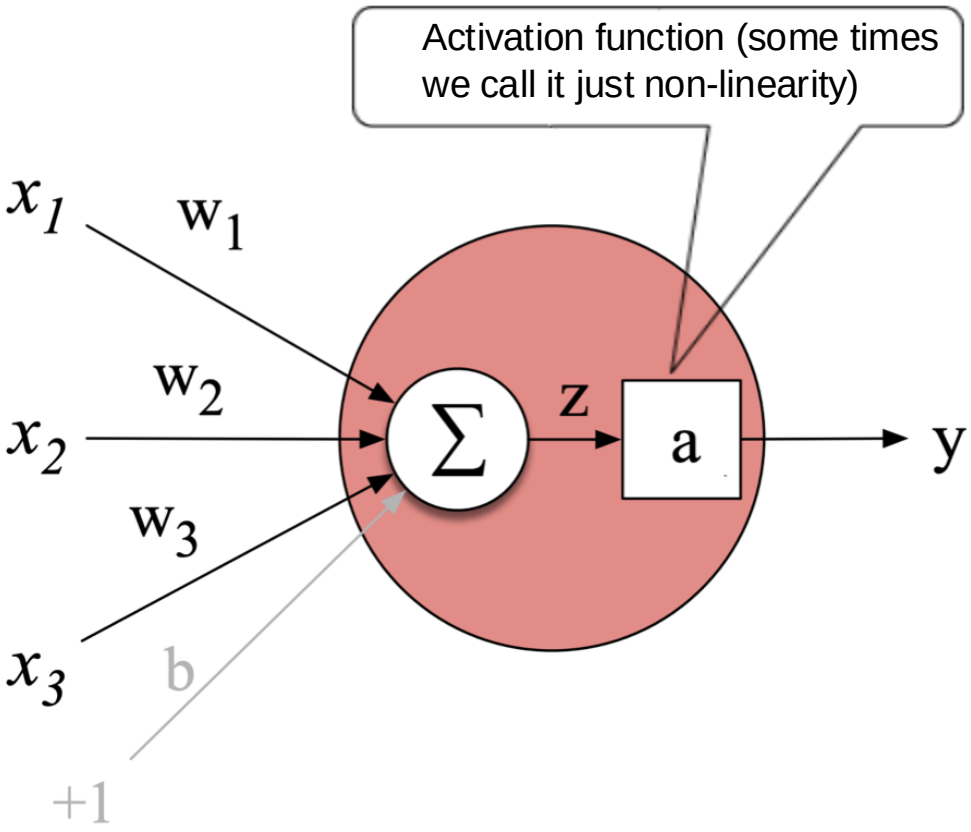
Why the bias?

Neurons with weights only index functions passing via the origin.

The bias allows for modelling the set of **affine** functions, which is a superset of **linear** functions.



Visualization of a neuron



From J&M3, §7.1

Activation functions $a(z)$

Identity (results in a **linear** model, can perform **regression**):

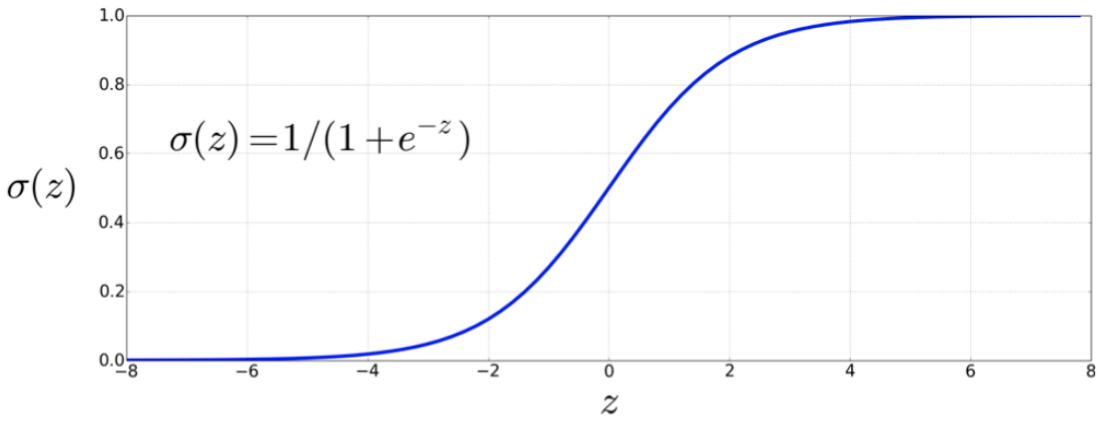
$$y = f(\mathbf{x}) = a(z) = z$$

Sigmoid (results in a **log-linear** model, can perform **logistic regression** / **binary classification**):

$$y = f(\mathbf{x}) = a(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Softmax with 2 classes is equivalent to sigmoid (a small exercise: check that this is true; first it may seem to you that softmax has more parameters)

Sigmoid function



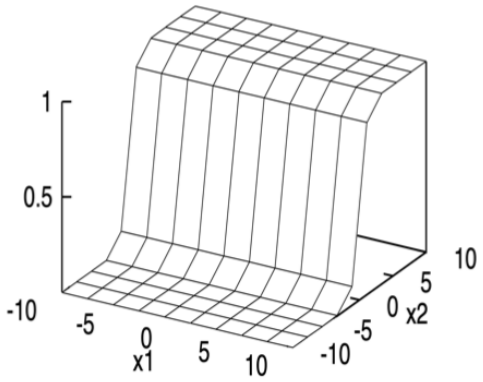
$y \in [0, 1]$. To see why:

$\lim_{z \to \infty} \sigma(z) = 1$

$\lim_{z \to -\infty} \sigma(z) = 0$

Example: 2-dimensional input

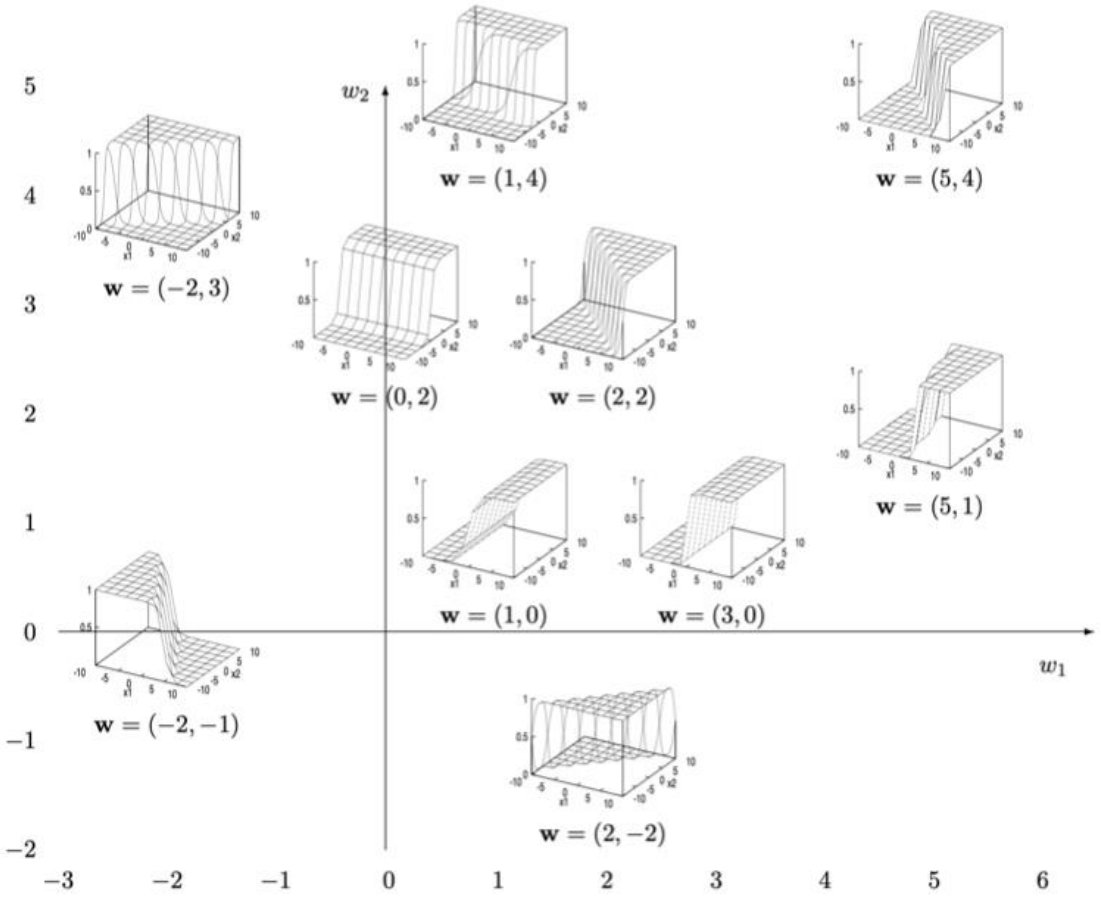
The full neural network is $y = \frac{1}{1+e^{-(w_1 x_1 + w_2 x_2 + b)}}$ If we set $\mathbf{w} = (0, 2)$:



From MacKay, §39.2

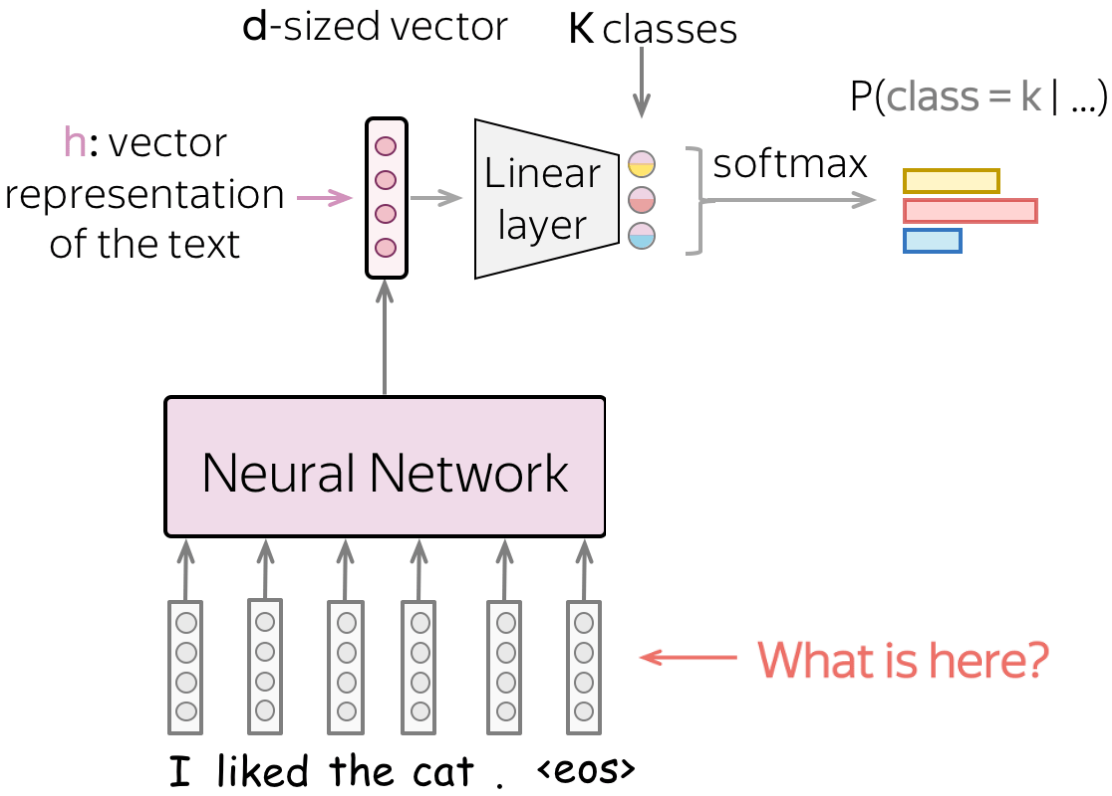
Each choice of $\mathbf{w}$ (a point in $\mathbf{w} \in \Theta$) indexes a function from the space $f(\cdot) : \mathcal{X} \to \mathcal{Y}$.

Parameter space



From MacKay, §39.2

Back to text models



Input: word embeddings

Word embeddings are a parameter matrix $E \in \mathbb{R}^{d \times |\mathcal{V}|}$ (with as many columns as words in the vocabulary)

For any word, we can fetch the corresponding column in E to obtain its representation.

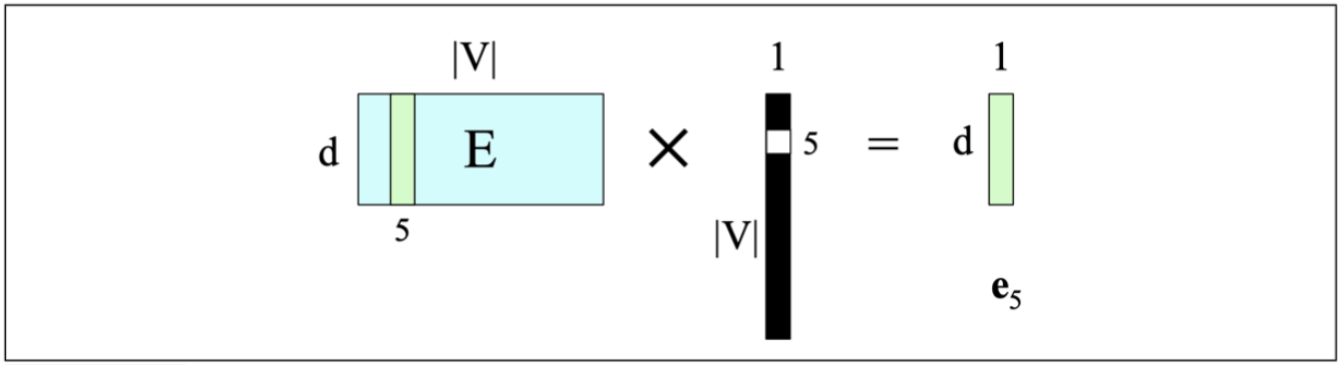
As a pre-processing step, we encode each $x \in \mathcal{V}$ as a distinct **one-hot vector** $\text{one-hot}(x)$. E.g., for $\mathcal{V} = \{all, happy, families\}$:

- $\text{one-hot}(all) = [1, 0, 0]$
- $\text{one-hot}(happy) = [0, 1, 0]$
- $\text{one-hot}(families) = [0, 0, 1]$

Input: word embeddings

During training and inference, any word in a context can be embedded via matrix-vector multiplication.

E.g., for the word with 1 in its 5th dimension of the one-hot vector:



So $\text{enc}(x) = E\,\text{one-hot}(x)$

Composing word embedding in a doc representation

To construct the representation of a document length n−1, we could just concatenate the encodings (aka embeddings) of the corresponding words:

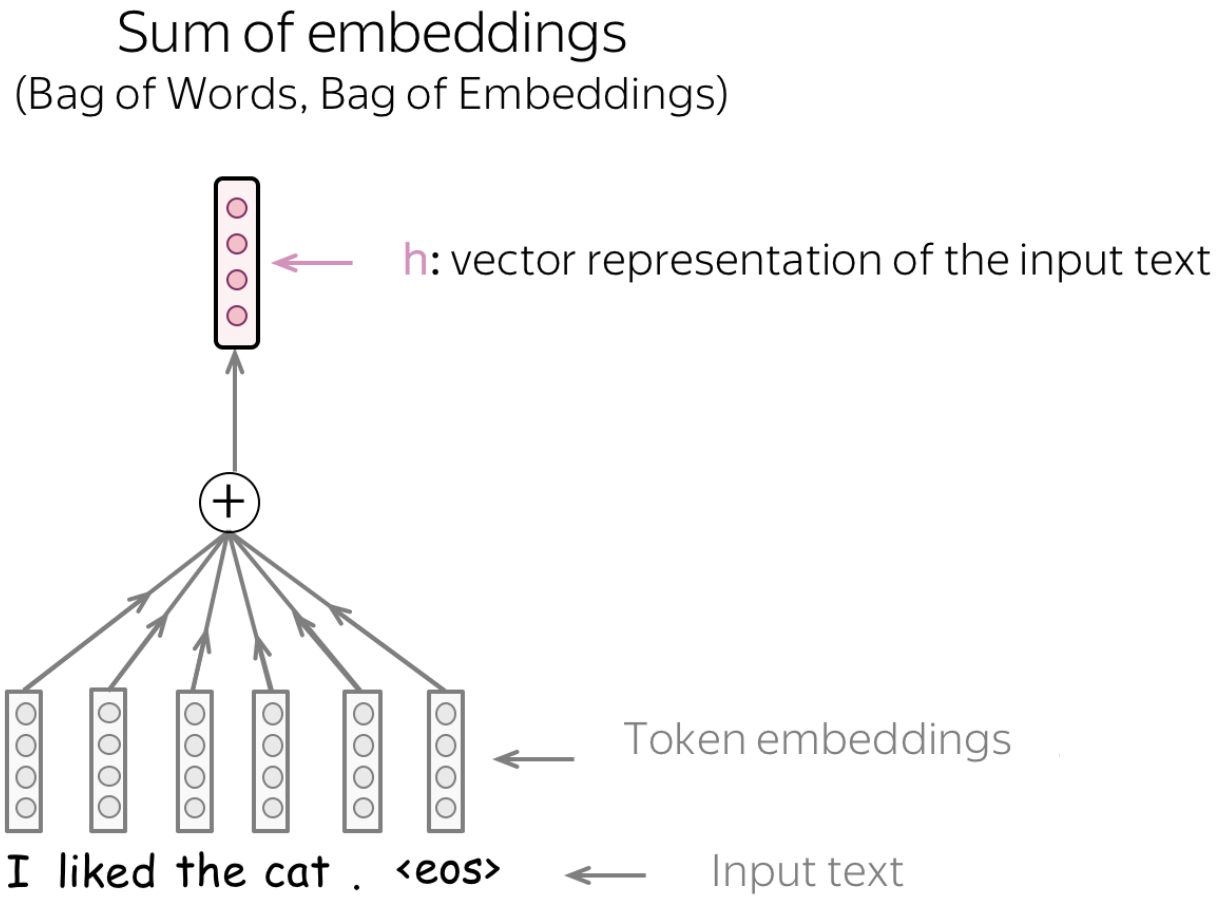
$$\mathrm{enc}(x_1, \ldots, x_n) = \mathrm{enc}(x_1) \circ \cdots \circ \mathrm{enc}(x_n).$$

So $\mathrm{enc}(x_1, \ldots, x_n) \in \mathbb{R}^{dn}$

This may not be a great idea as the document length can be very long and the number of words varies across documents.

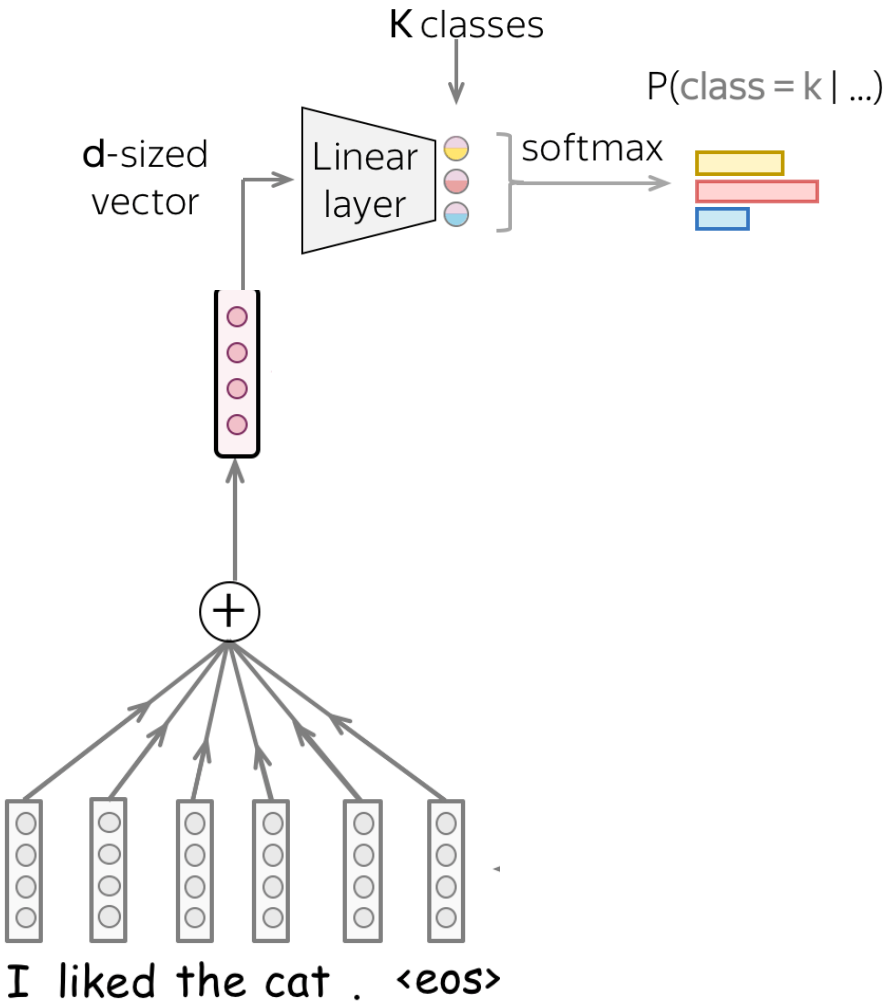
So not really what we do for classification (but this architecture will make much more sense when we get to 'language modelling', i.e., predicting next word).

Basic models: bags of words (= Embeddings)



We will see much more powerful approaches soon

Schematic representation



Perceptron

Neural networks consisting of a linear layer (or multiple linear layers without non-linear activations in between) is called perceptron.

The networks on the previous slide is a perceptron (though not a general one, due to the summation in the first layer).

Logistic regression is (effectively) a perceptron.[1]

[1] when we talk about LR, we imply the specific training objective (cross entropy), whereas "perceptron" may also refer to the same linear mode trained with a different training algorithm / using different loss (perceptron algorithm)

Non-linear problems

Yet, there exists a class of problems that perceptrons cannot solve.[1]

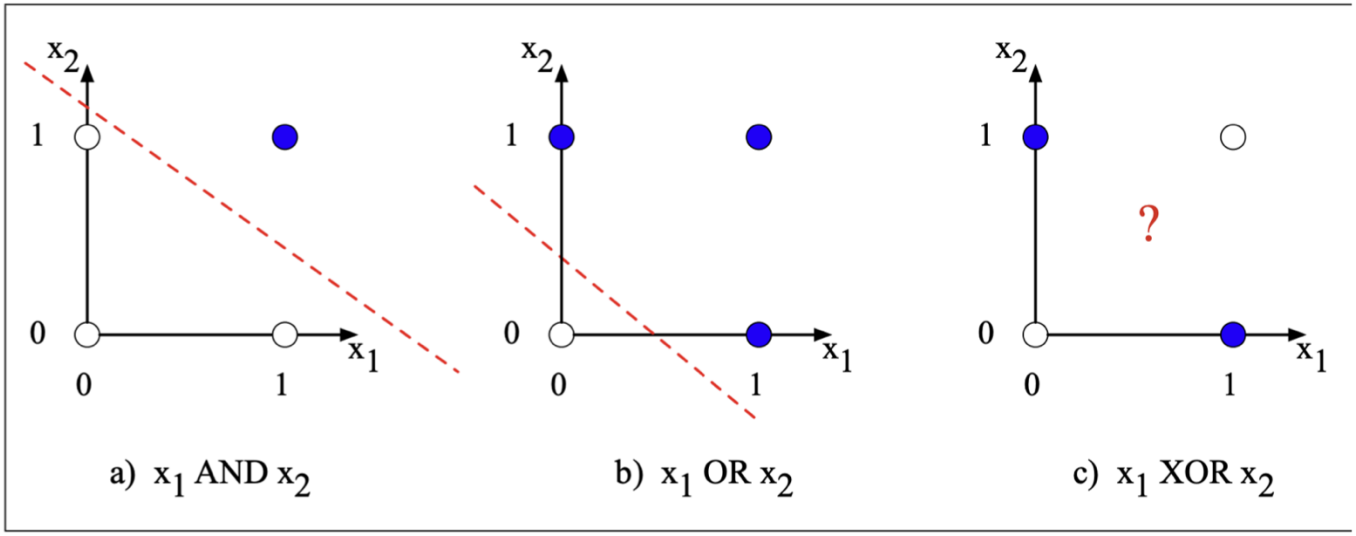
Historically, the first such problem connected to limitations of perceptrons is the XOR operator (Minsky and Papert 1969).

Consider the truth tables for various logical operators :

AND			OR			XOR		
x_1	x_2	z	x_1	x_2	z	x_1	x_2	z
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

[1] In the input space, but they can if extra features like $x_1 \cdot x_2$ are added.

Linear separability



For a threshold τ, we can draw a **decision boundary**, i.e., $y = 1$ if $w_1 x_1 + w_2 x_2 + b > \tau$, else $y = 0$.

This boundary is a line: $x_2 = (-w_1/w_2)x_1 + (-b/w_2) + \tau/w_2$

Multilayer perceptron

How to make neural networks more expressive, i.e., capable of indexing non-linear functions? Recipe:

1. stack perceptrons (layers)

2. use non-linear activations at the end of each layer

The resulting non-linear model is a **multi-layer perceptron** (MLP)!

Stacking layers

Perceptrons can be stacked. We refer to their ordered sequence as **layers**.

This creates **feedforward networks**, so that the output of layer l is passed as input to the next layer $l+1$,[2] but not to the previous ones $1, \ldots, l-1$.

In addition, this family of neural networks is **fully connected**, meaning that for each layer, each output unit is the weighted sum of **all** the input units.

The number of layers is the **depth** of the network (hence, the term deep learning!)

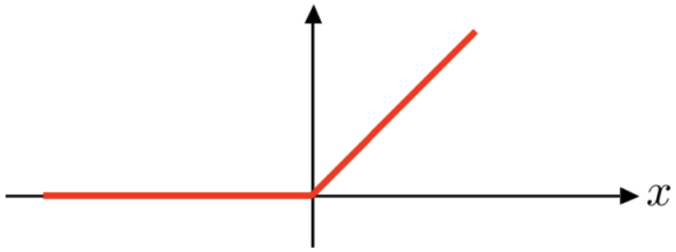
[2] Optionally, it can be passed also to any subsequent layer $l+1, \ldots, L$ (**skip connection**).

Non-linear activation functions

The output $\mathbf{z}$ is passed through a **non-linear function** $a(\cdot)$, which gives us the **hidden representation** $\mathbf{h} \in \mathbb{R}^h$ of intermediate layers.

Any **differentiable**[3] non-linear function can be chosen. A common choice is ReLU (others are sigmoid, tanh, . . .):

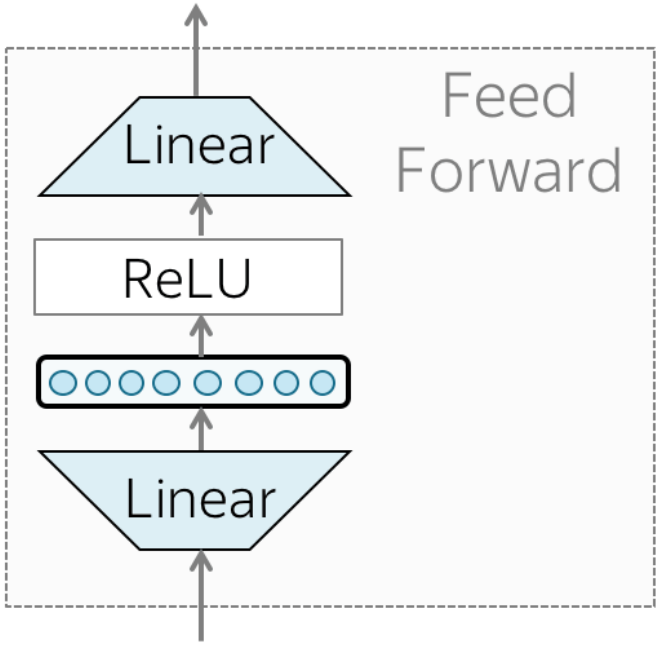
$$\mathrm{ReLU}(x) \triangleq \max(0, x)$$



Note: without non-linearities, a multi-layer network remains linear!

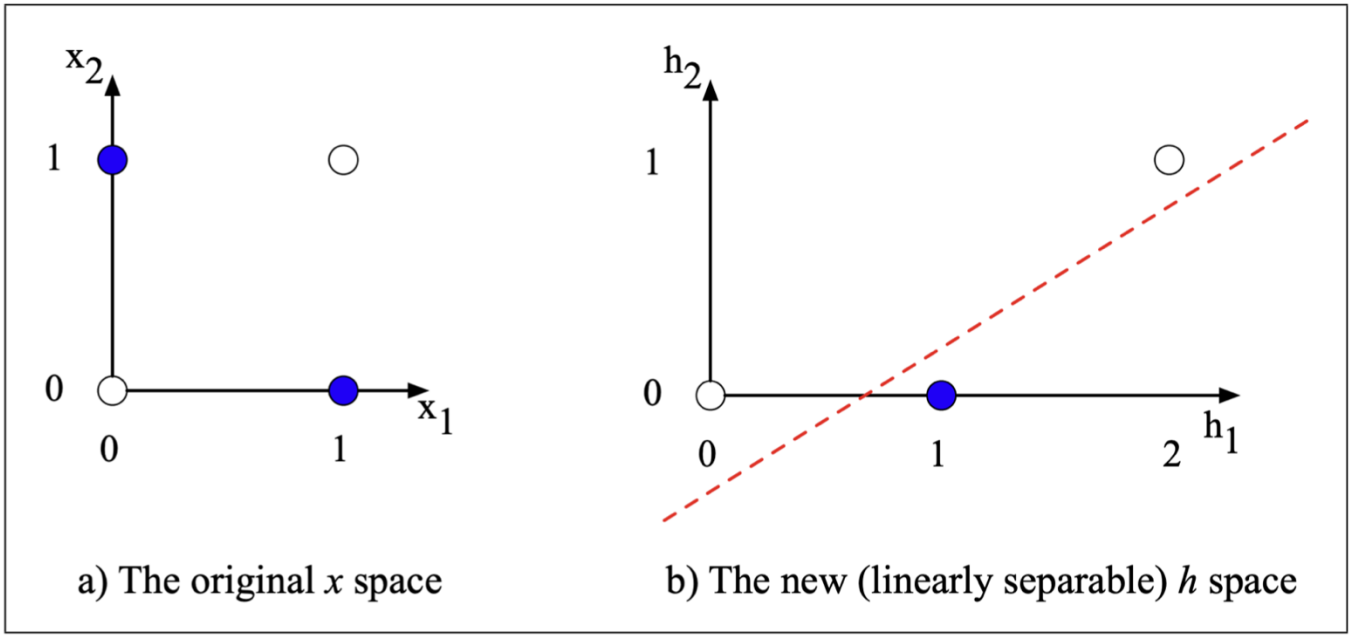
[3] We will see later why this is requirement for training the model

Example: 2-layer MLP (aka 2 layer Feed Forward)



$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

MLP solution to XOR

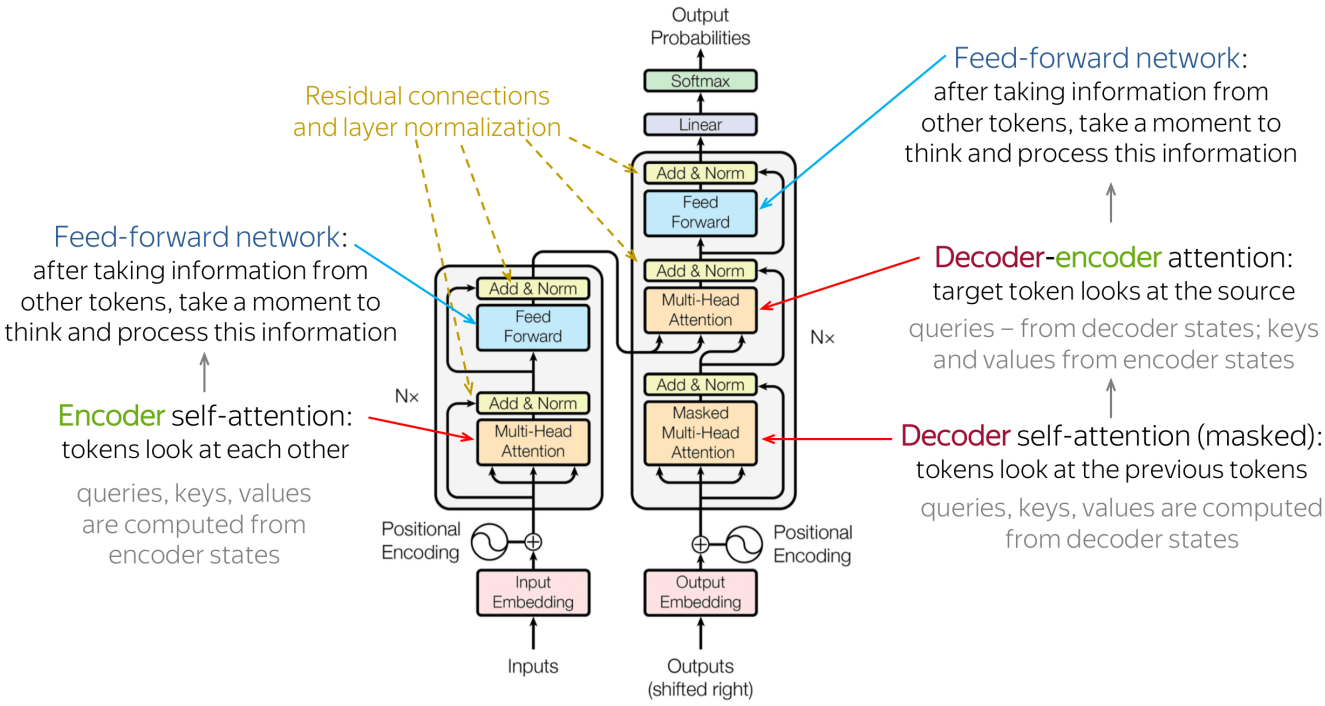


From J&M3, §7.3

Key idea: space folding! $\mathbf{h}$ is linearly separable in the next layer.

Soon more power NLP models!

But we already recognize perhaps half of their components



Conclusions

Text classification with neural networks

- Generalization of logistic regression
- Concept of word embeddings (will see and understand them much more later)
- Bag-of-word models for classification

Neural networks:

- NNs are built out of neurons, a function from many input dimensions to one output dimension.
- A multi-layer perceptron stacks multiple layers, each consisting of multiple units and with a non-linear activation.
- Each layer captures useful features for the next layers.