Natural Language Processing

WBAI059-05

university of groningen

faculty of science and engineering

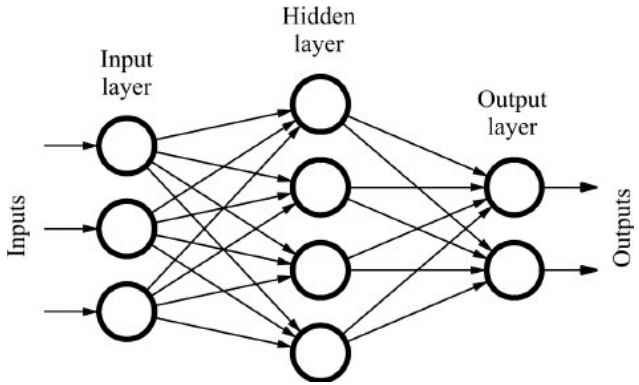
Tsegaye Misikir Tashu

Lecture 4: Neural Networks for NLP

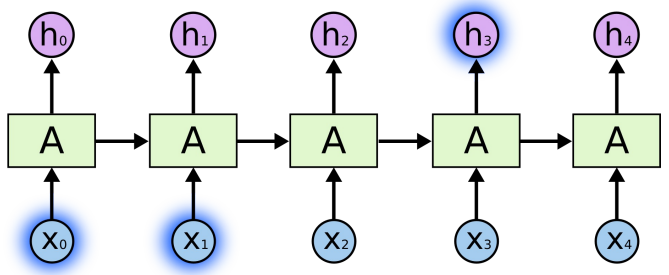
(Some slides are adapted from Peyrin Kao , Danqi Chen and Len voita)

Neural networks in NLP

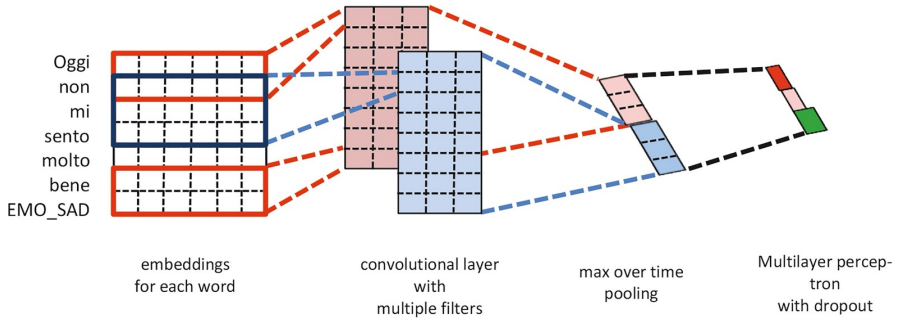
Feed-forward NNs



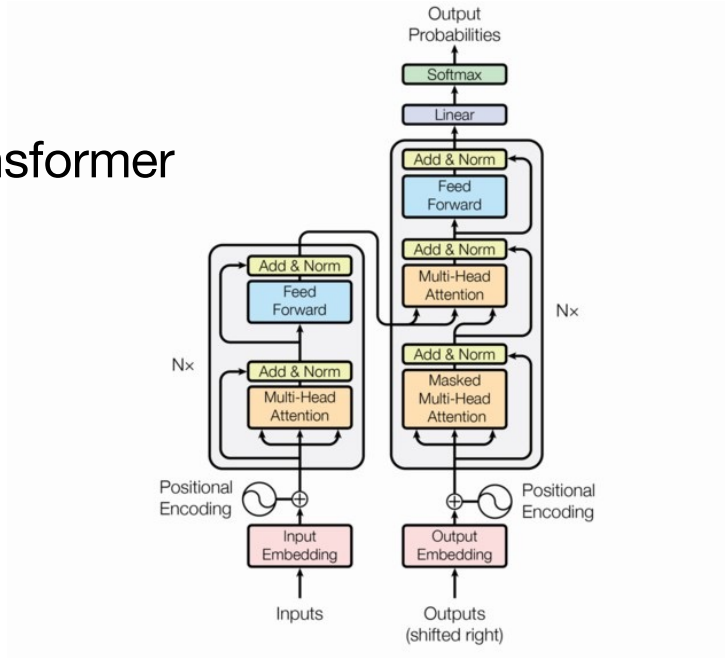
Recurrent NNs



Convolutional NNs



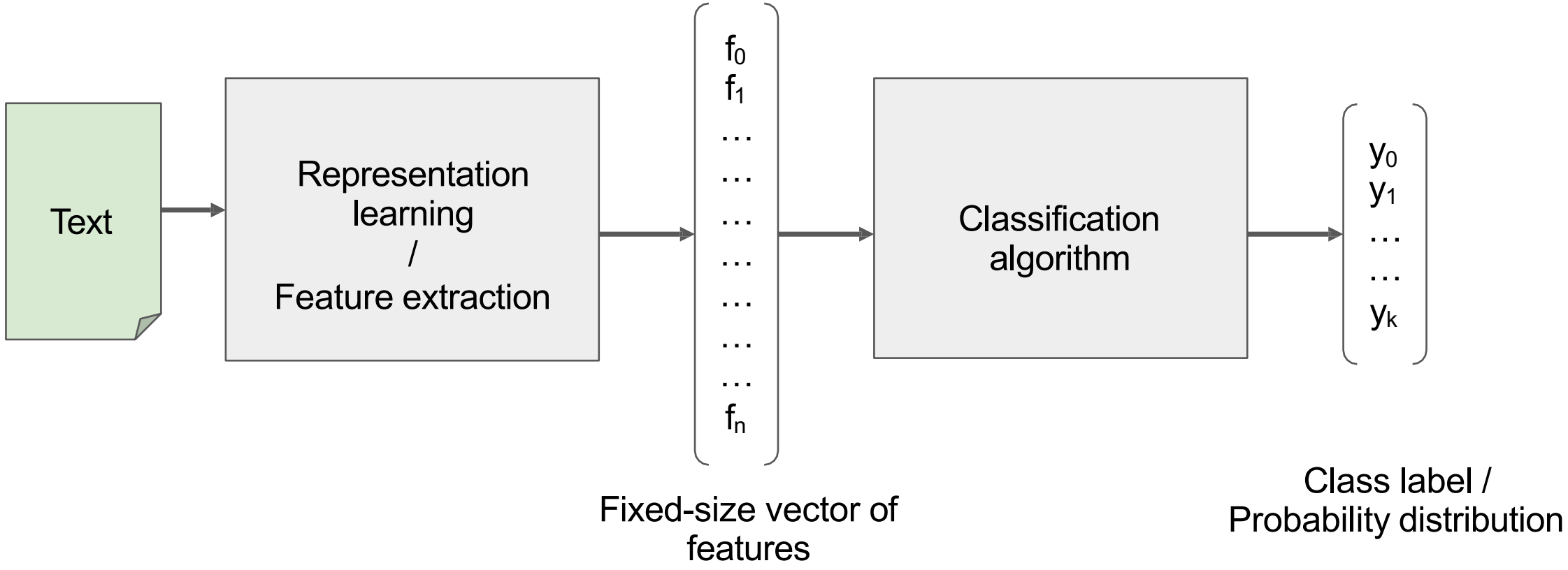
Transformer



Text Classification

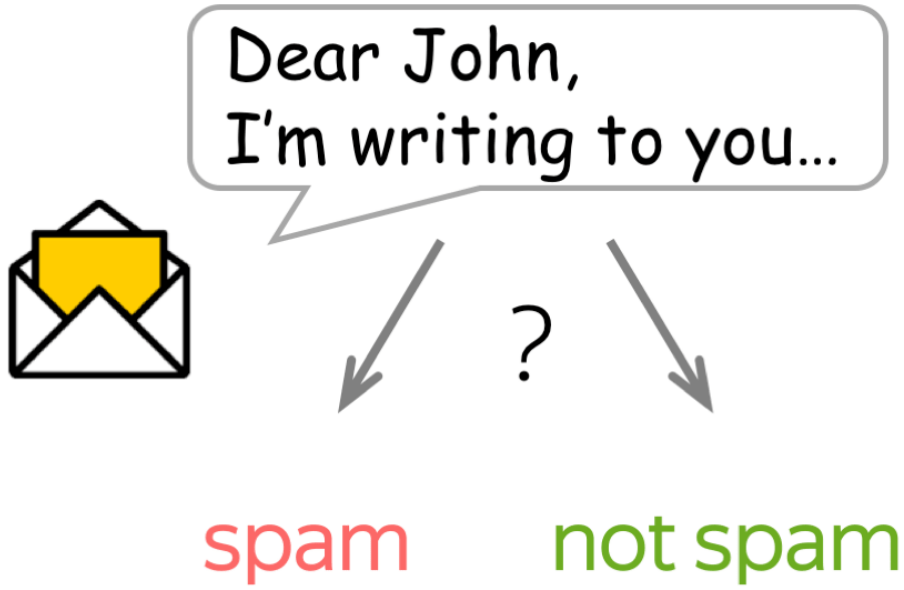
A Neural Network Approach

Text Classification in general



Text Classification

- Binary: two labels
- Single label: only one label is correct

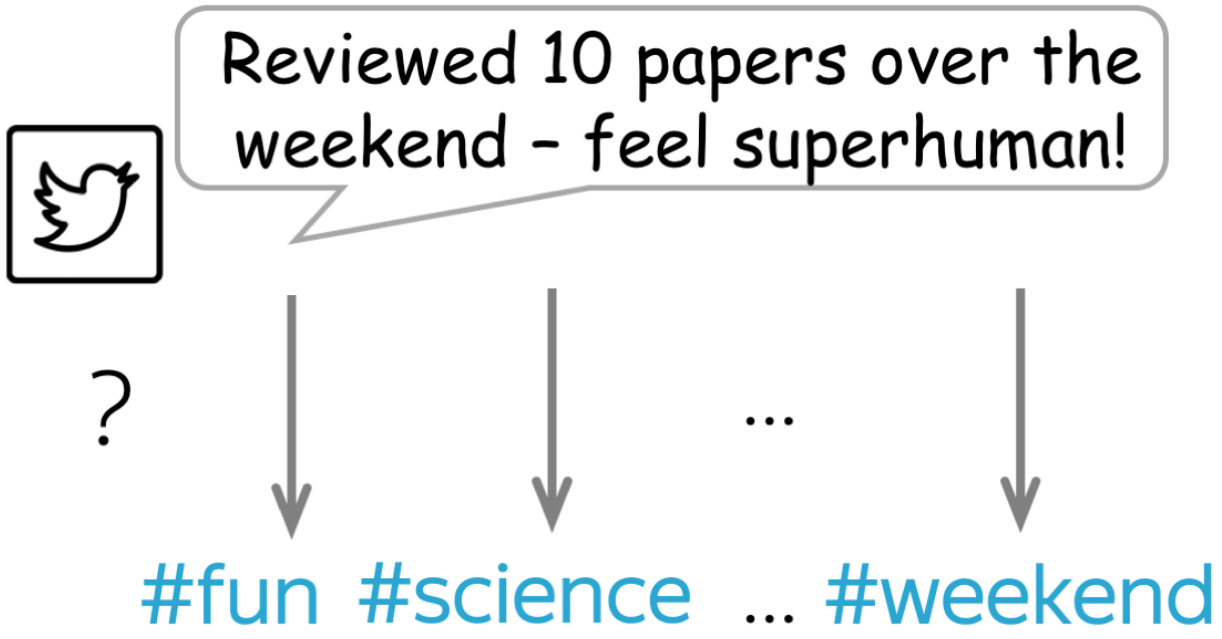


Text Classification

- Multi-class: many labels
- Single label: only one label is correct

Text Classification

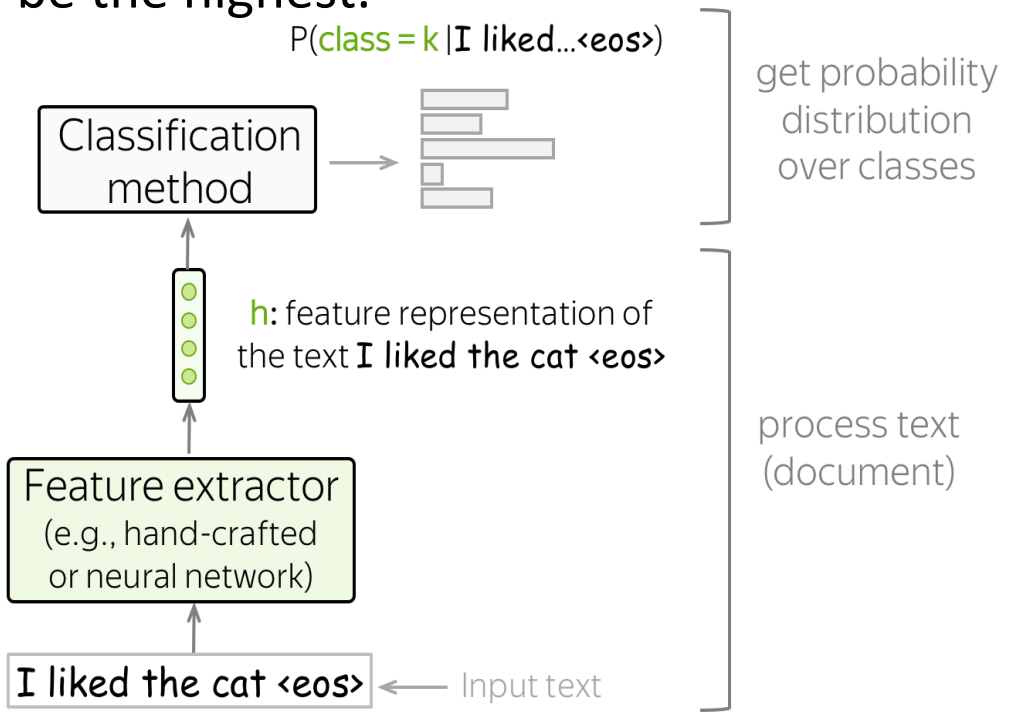
- Multi-class: many labels
- Multi-label: several labels can be correct



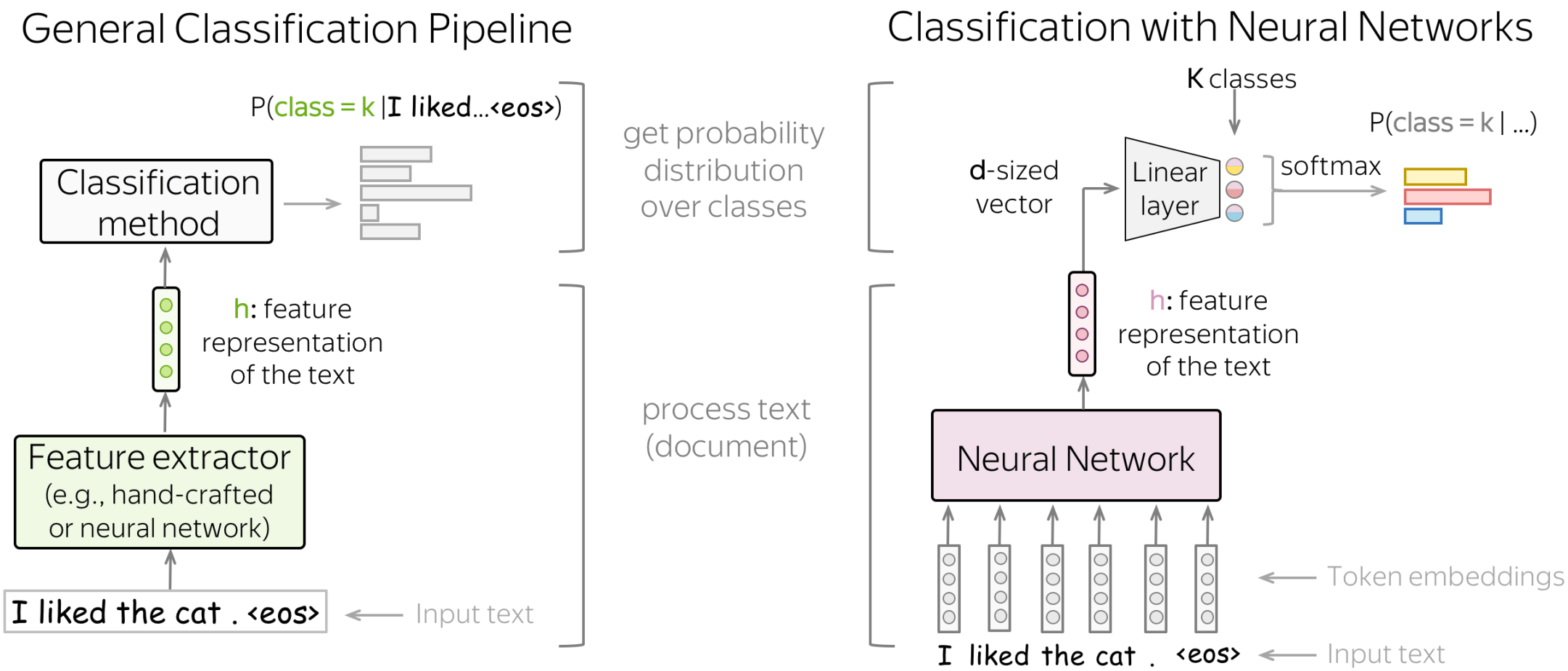
Classification: Get Feature Representation and Classify

- Assume that we have a collection of documents with ground-truth labels
- The input of a classifier is a document $x = \{w_1, w_2, \ldots, w_n\}$, the output is a label $y \in 1, .., k$.
  - a classifier estimates probability distribution over classes, and
  - we want the probability of the correct class to be the highest.

- Feature Extractor
  - A feature extractor can be either manually defined (as in classical approaches) or learned (e.g., with neural networks).
- Classifier
  - Assigns class probabilities given the feature representation of a text.
  - The most common way to do this is using logistic regression, Naive Bayes , SVM or Neural networks



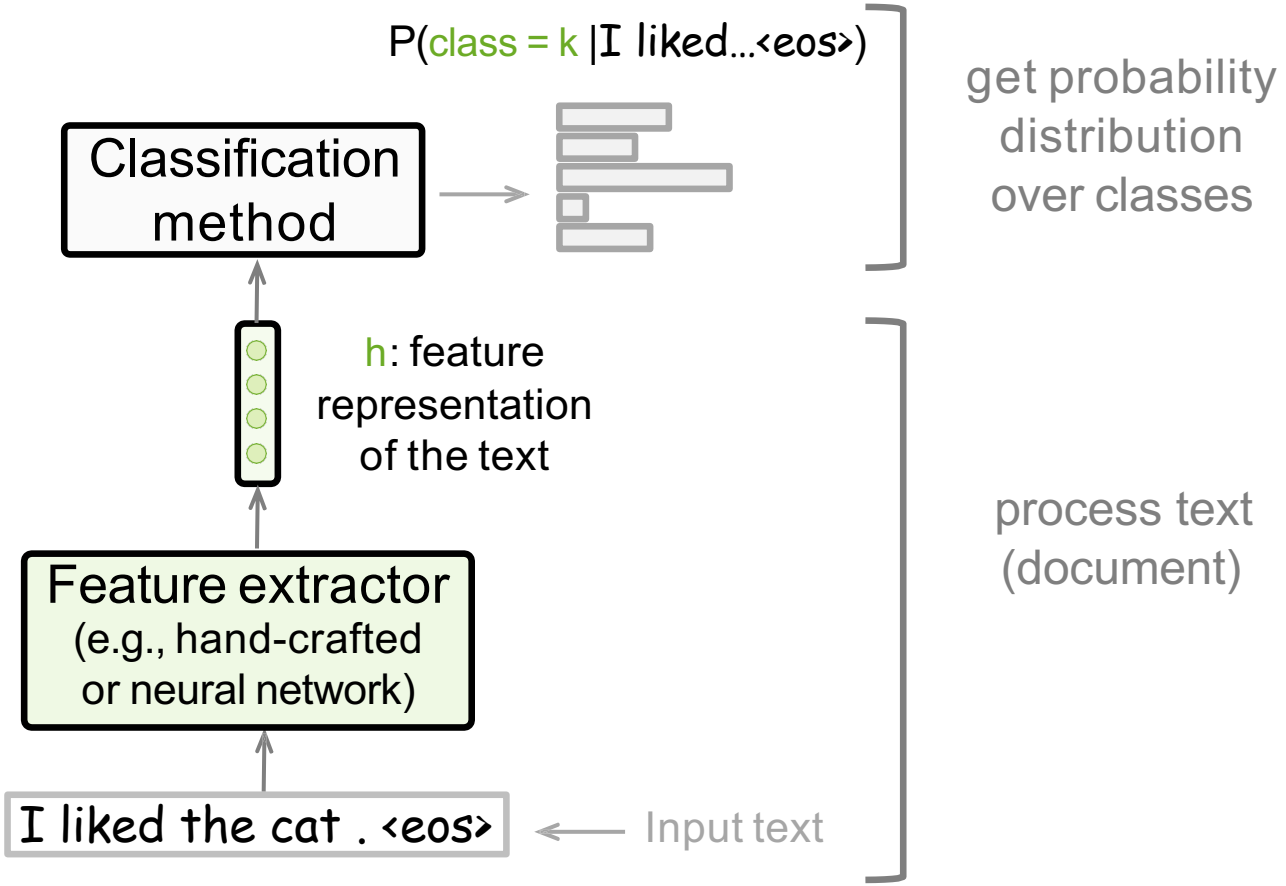
Dealing with neural networks: The main idea of neural-network-based classification is that feature representation of the input text can be obtained using a neural network.



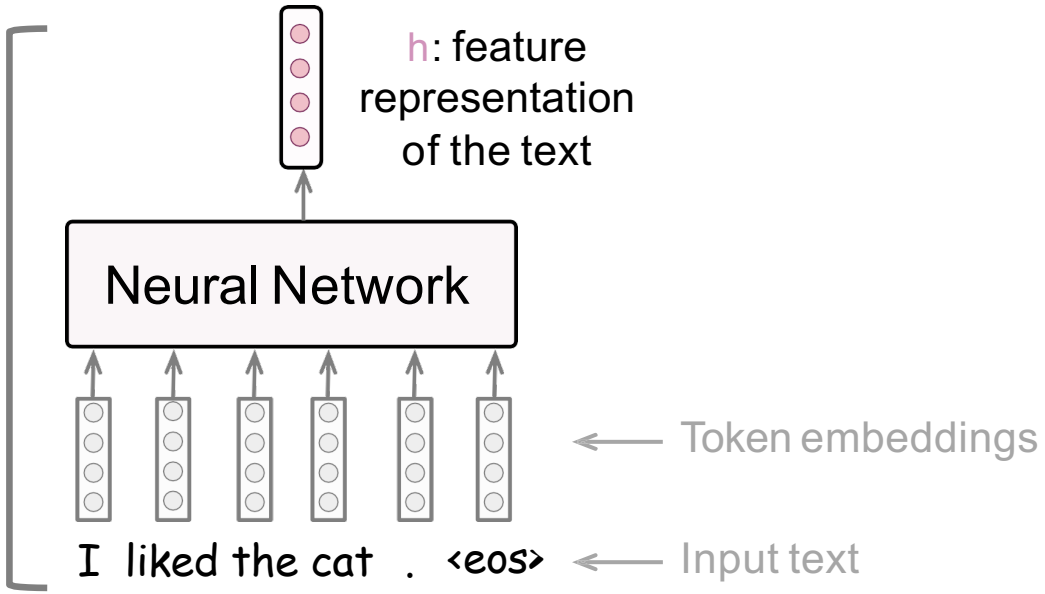
Classification with Neural Networks

> Instead of manually defined features, let a neural network to learn useful features.

- General Classification Pipeline



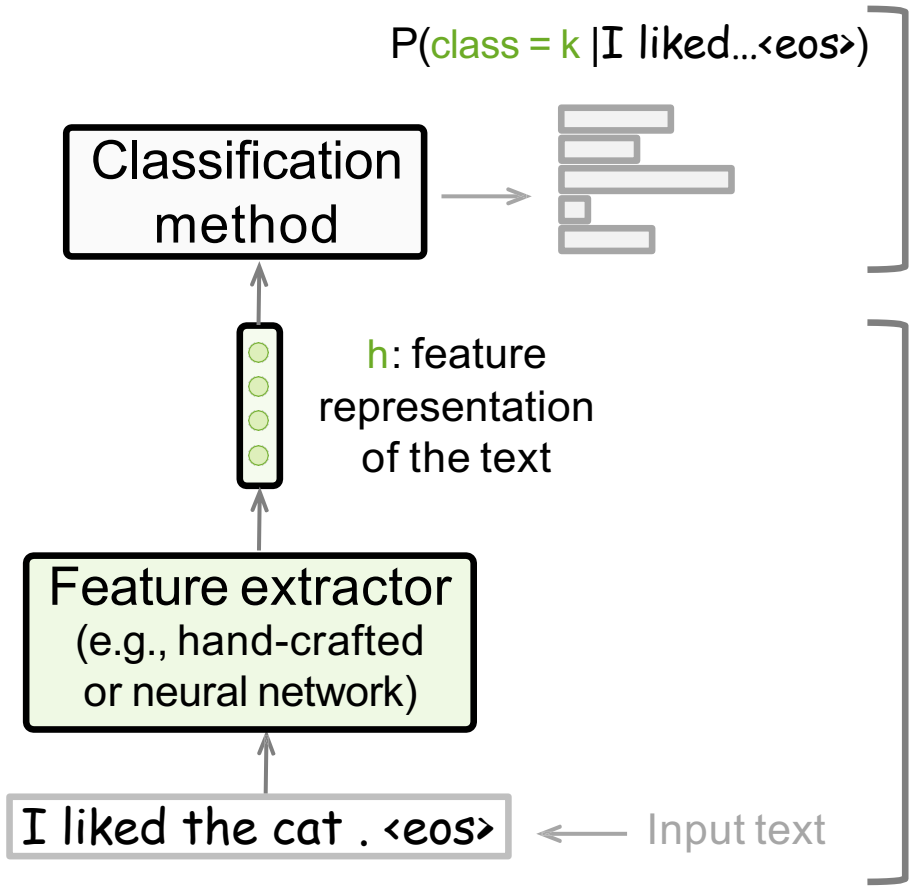
- Classification with Neural Networks



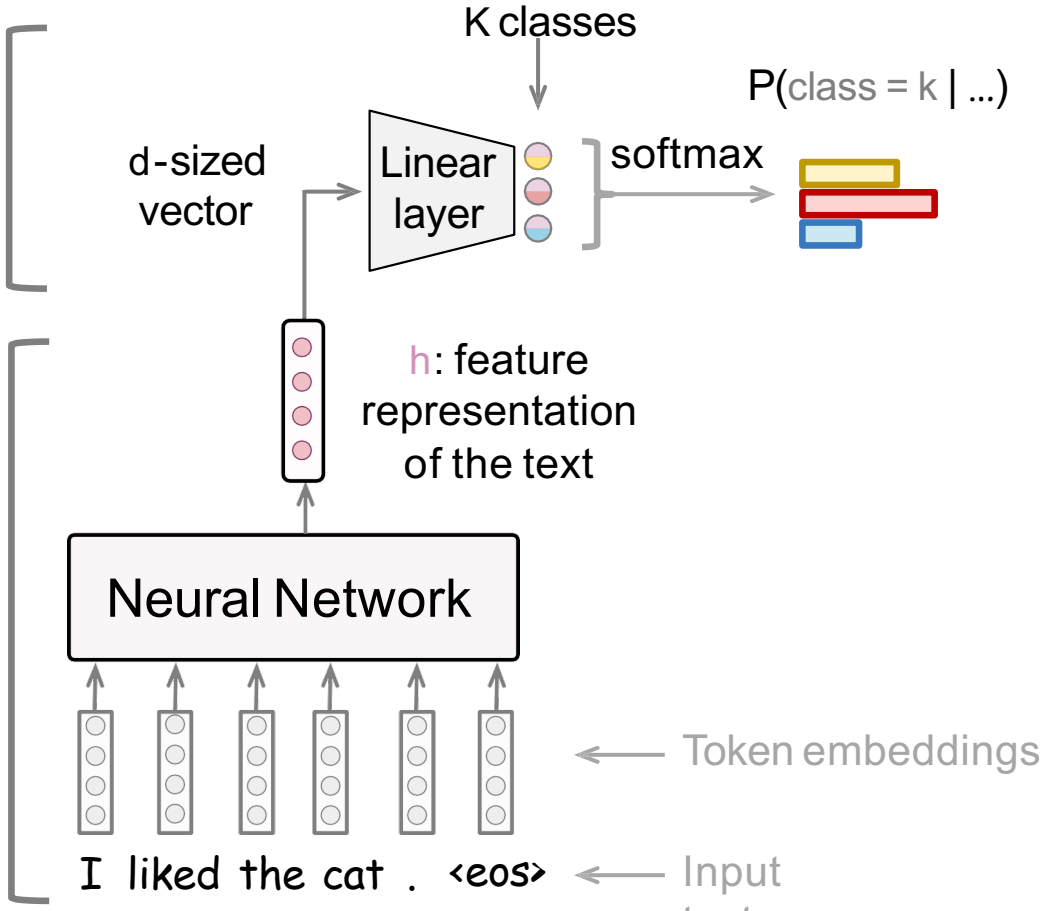
Classification with Neural Networks

> Instead of manually defined features, let a neural network to learn useful features.

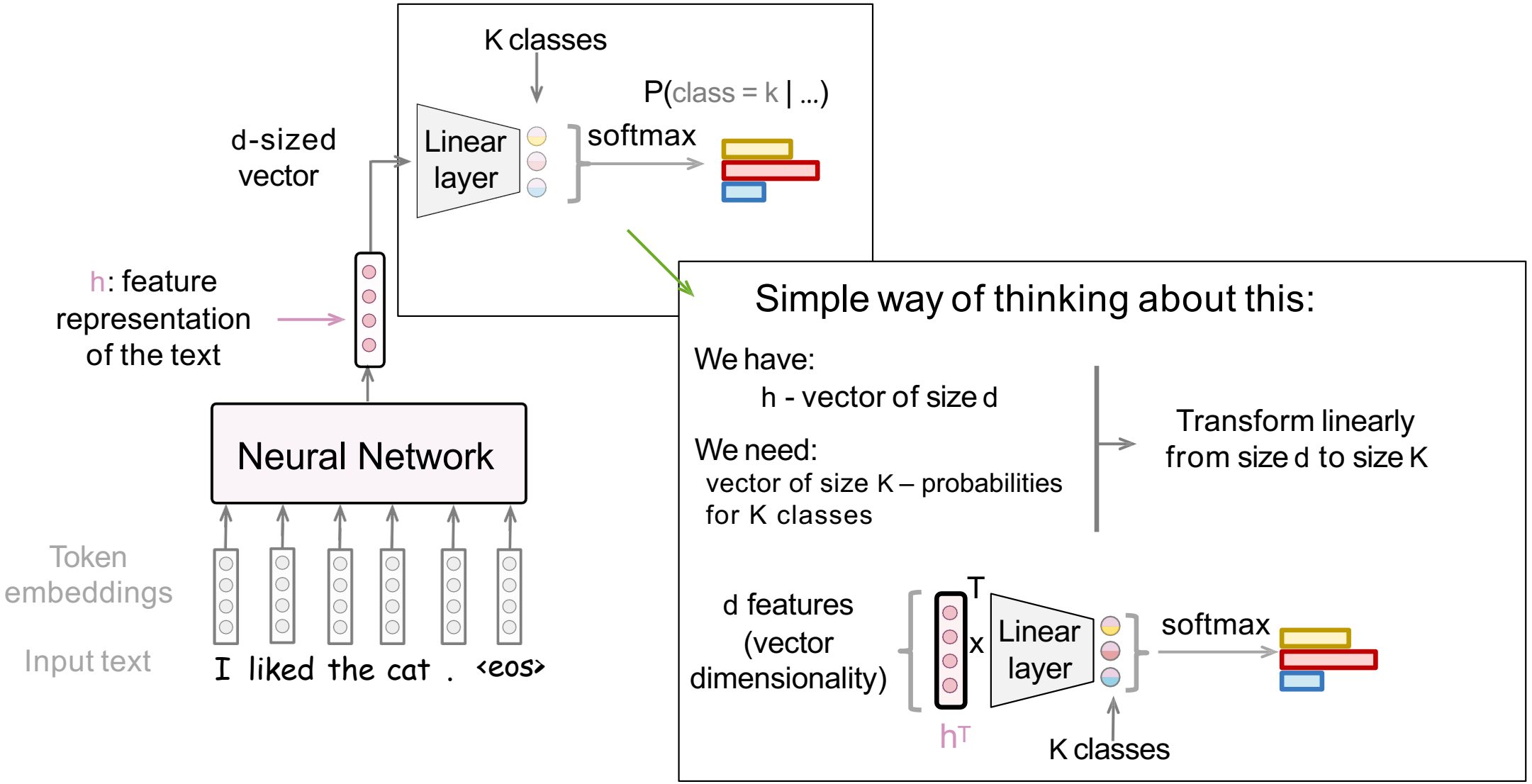
- General Classification Pipeline



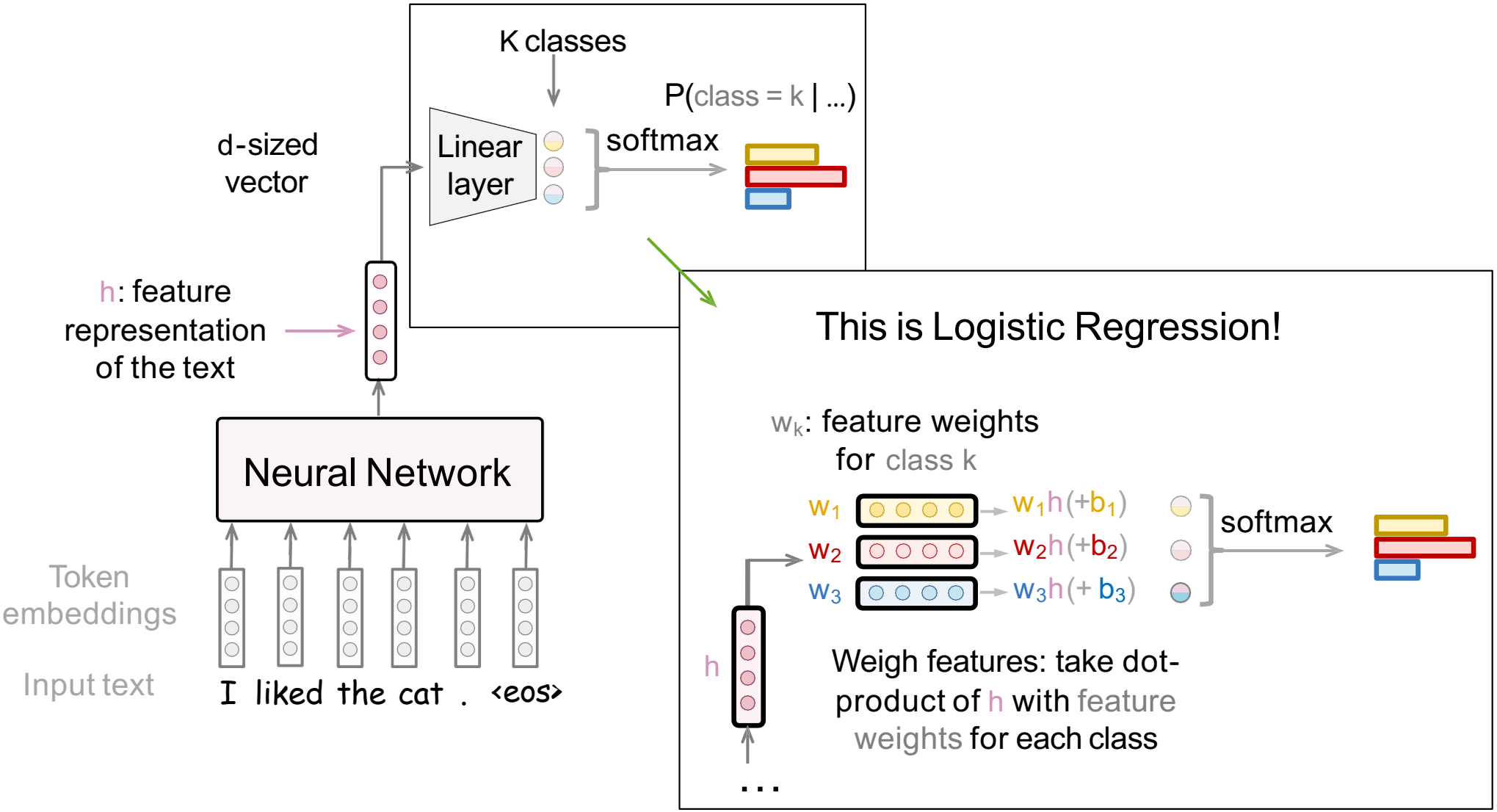
- Classification with Neural Networks



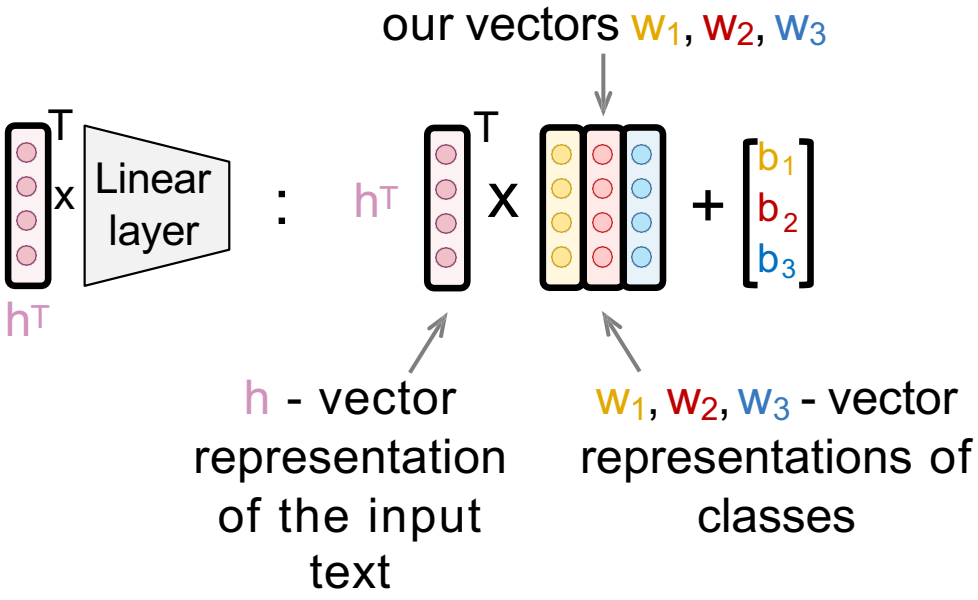
Classification with Neural Networks



Classification with Neural Networks

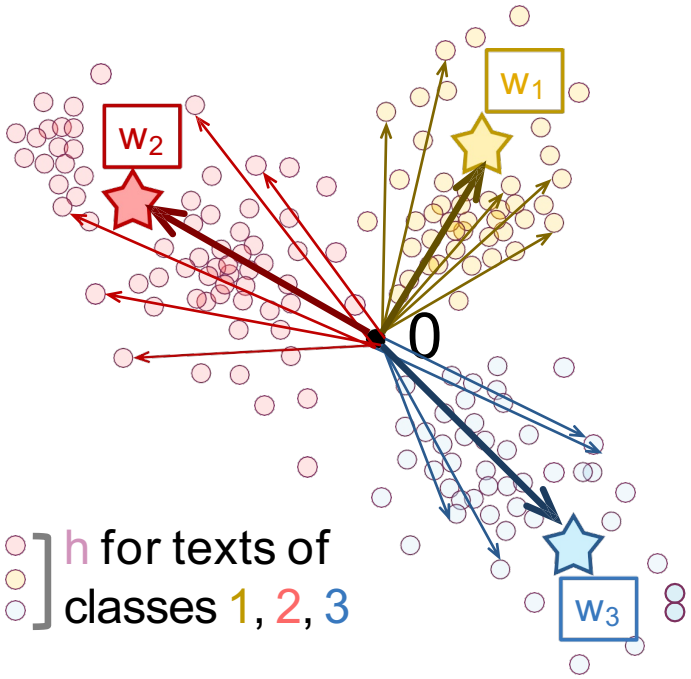


Text Representation and Class Representation



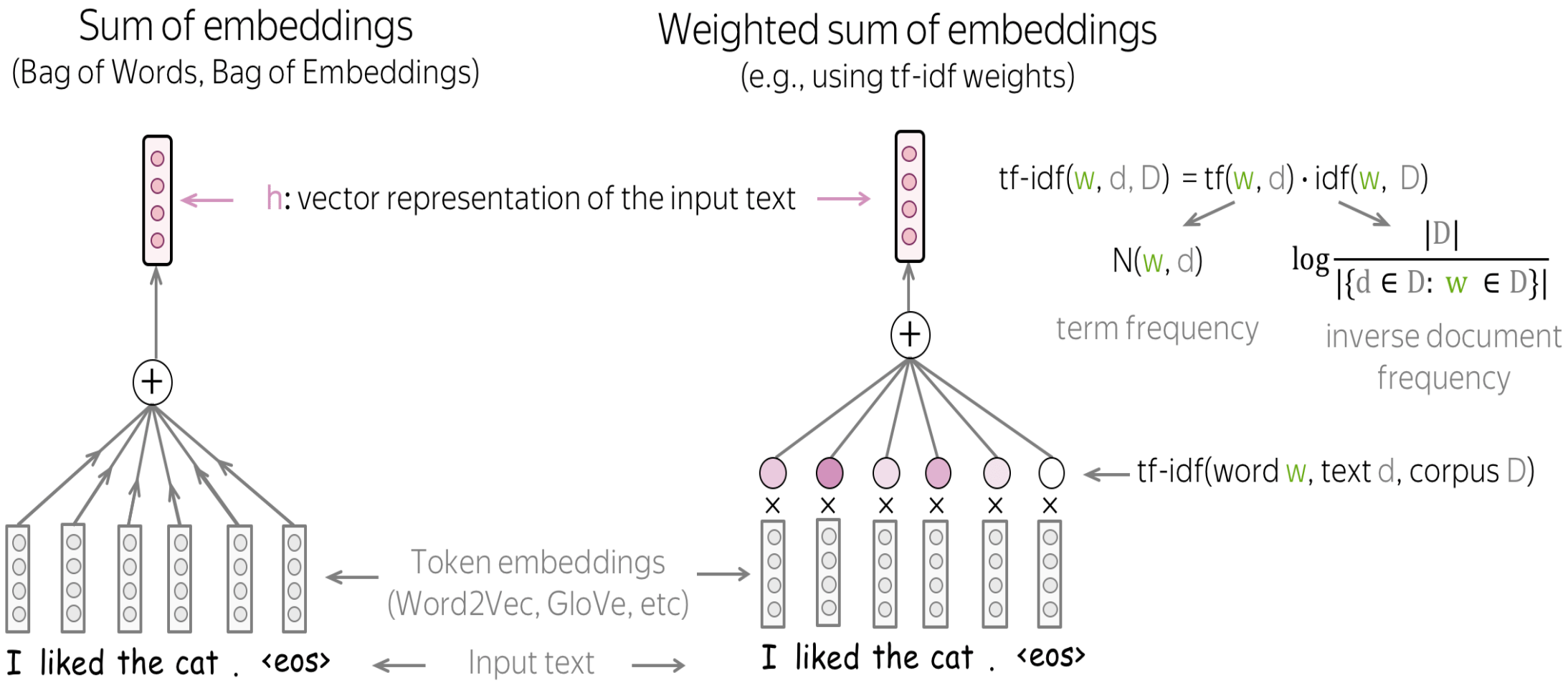
What NN learns (hopefully):

Text vectors point in the direction of the corresponding class vectors



Models Text Classification

Bag of Embeddings (BOE) and Weighted BOE



Feedforward nets for simple classification

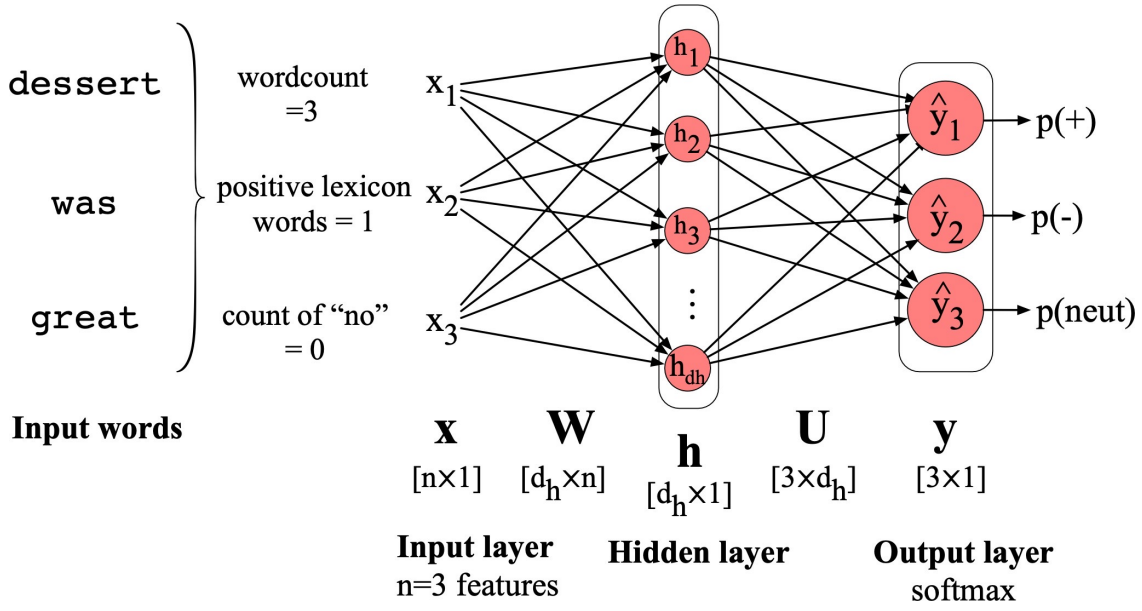
- Allows the network to use non-linear interactions between features which may (or may not) improve performance.
- The real power of neural (deep) learning comes from the ability to **learn** features from the data.
- Instead of using hand-built human-engineered features for classification: use learned representations like embeddings!

Feedforward Neural Networks for text classification

Input: $x_1, x_2, \ldots x_n \in V$, Output: $y \in C$ = {positive, negative, neutral}

- How can we feed the input to a neural network classifier? $x_1, x_2, \ldots, x_n \in V$

Solution #1: Construct a feature vector from the input and feed the vector to a **neural network**



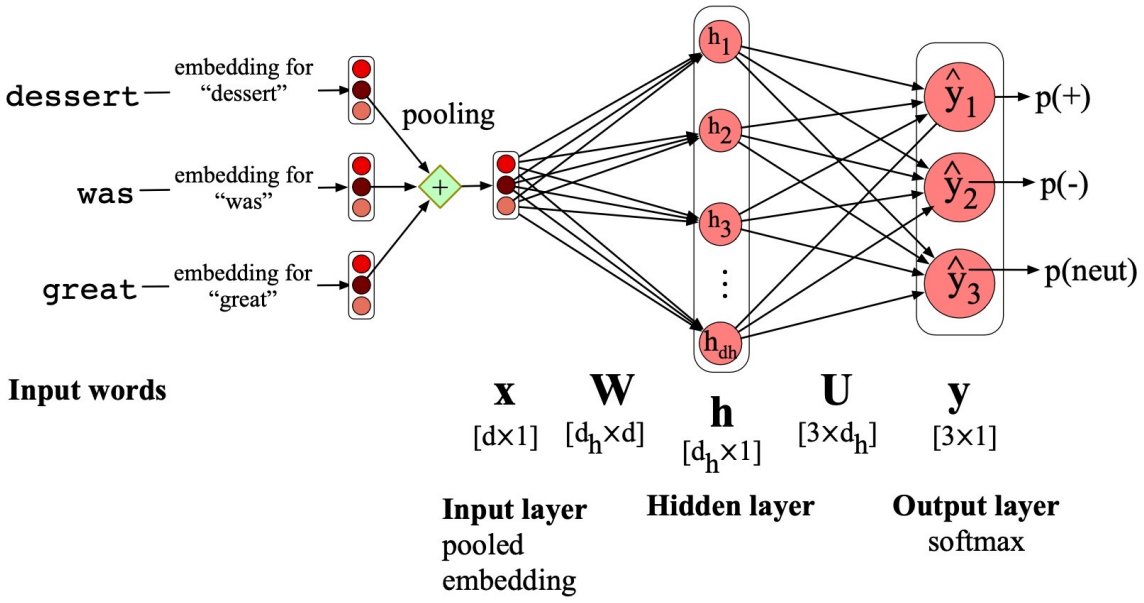
(each $\mathbf{x}_i$ is a hand-designed feature)

- $x = [x_1, x_2, \ldots, x_n]$
- $h = \tanh(Wx + b)$
- $y = Uh$
- $\hat{\mathbf{y}} = \mathit{softmax}(\mathbf{y})$

The neural network has the promise to learn good features automatically.

Feedforward Neural Networks for text classification

- How can we feed the input to a neural network classifier? $x_1, x_2, \ldots, x_n \in V$
  - Solution #2: Take all the word embeddings of these words and aggregate them into a vector through some **pooling** function!
    - **pooling:** sum, mean or max



$$\mathbf{x} = \frac{1}{K}\sum_{i=1}^{K} e(w_i)$$

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{y} = \mathbf{U}\mathbf{h}$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{y})$$

How to train this model?

- Training data: $\{(d^{(1)}, y^{(1)}), \ldots, (d^{(m)}, y^{(m)})\}$
- Parameters: $\{\mathbf{W}, \mathbf{b}, \mathbf{U}\}$
- Optimize these parameters using gradient descent!

- Word embeddings can be treated as parameters too!

$$\mathbf{E} \in \mathbb{R}^{|V| \times d}$$

$$\mathbf{x} = \frac{1}{K} \sum_{i=1}^{K} e(w_i)$$

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{y} = \mathbf{U}\mathbf{h}$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{y})$$

How to train this model?

- Common practice: initialize $\mathbf{E}$ using word2vec or other embedding methods, and optimize them using Stochastic Gradient Descent!
- When the training data is small, don't treat $\mathbf{E}$ as parameters!
- When the training data is very large (e.g., language modeling), initialization doesn't matter much either (can use random initialization).

Why? $\mathbf{v}(\text{good}) \approx \mathbf{v}(\text{bad})$

	Most Similar Words for	
	Static	Non-static
bad	*good* *terrible* *horrible* *lousy*	*terrible* *horrible* *lousy* *stupid*
good	*great* *bad* *terrific* *decent*	*nice* *decent* *solid* *terrific*

(Kim 2014)

Training: Cross-Entropy

- Neural classifiers are trained to predict probability distributions over classes.
  - At each step, we maximize the probability a model assigns to the correct class.
  - The standard loss function is the 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^*

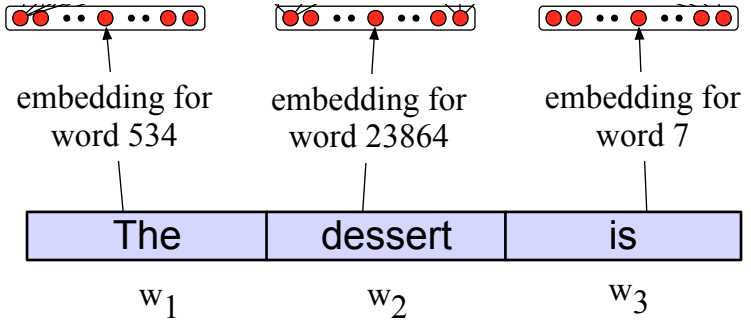
Cross-entropy loss:

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

For one-hot targets, this is equivalent to

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

Issue: texts come in different sizes



- This assumes a fixed size length (3)!
- Kind of unrealistic.
- Some simple solutions (more sophisticated solutions later)

1. Make the input the length of the longest text in the training data
   - If shorter, then pad with zero embeddings
   - Truncate if you get longer reviews at test time

2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
   - Take the mean of all the word embeddings
   - Take the element-wise max of all the word embeddings

Convolutional Neural Network

From FFNN to Convolutional Neural Nets

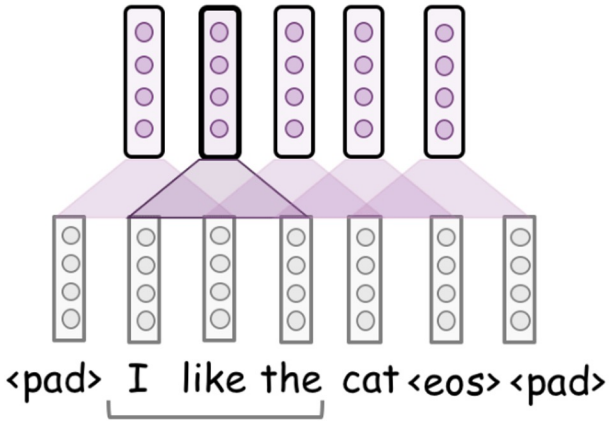
- The main idea of using Convolutional Neural Net (CNN/ConvNet) for NLP:
  - What if we compute vectors for every possible word subsequence of a certain length?

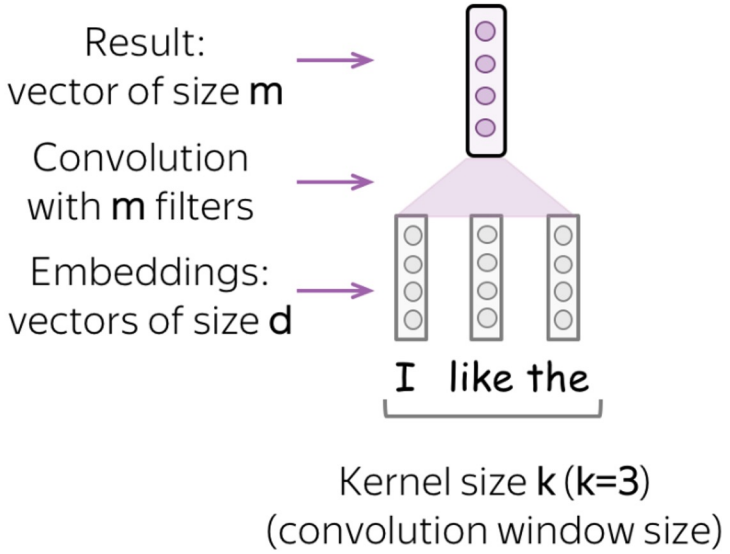
- Example: "tentative deal reached to keep government open" computes vectors for:
  - tentative deal reached, deal reached to, reached to keep, to keep government, keep government open

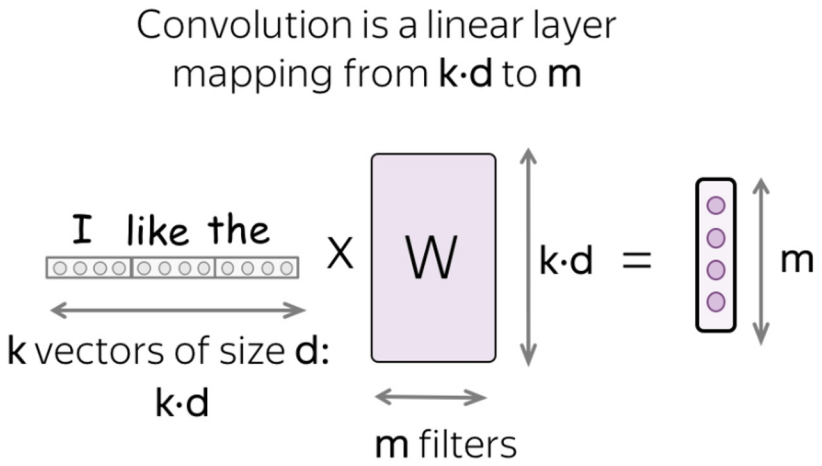
- Regardless of whether the phrase is grammatical or not

Building Blocks: Convolution

- Convolutions in computer vision go over an image with a sliding window and apply the same operation, convolution filter, to each window.
- A convolution layer usually has several filters, and each filter detects a different pattern.
- Differently from images, texts have only one dimension. Therefore, a convolution here is one-dimensional.

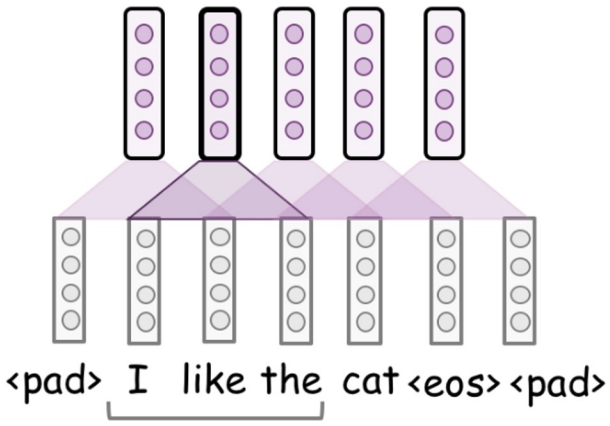


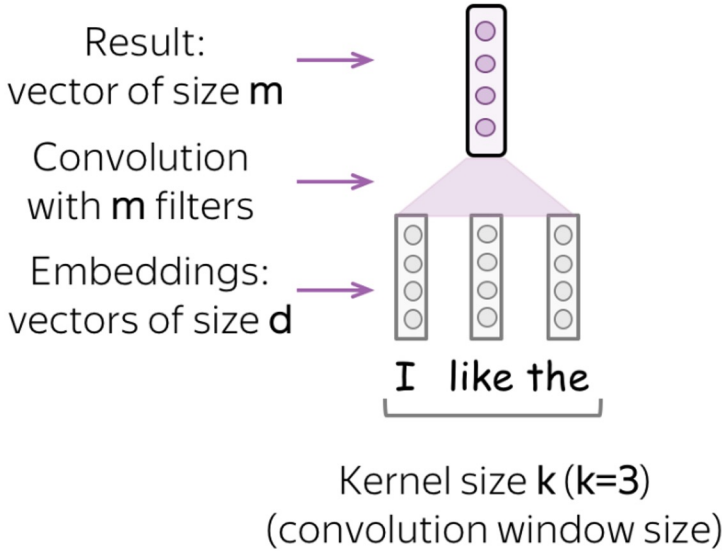


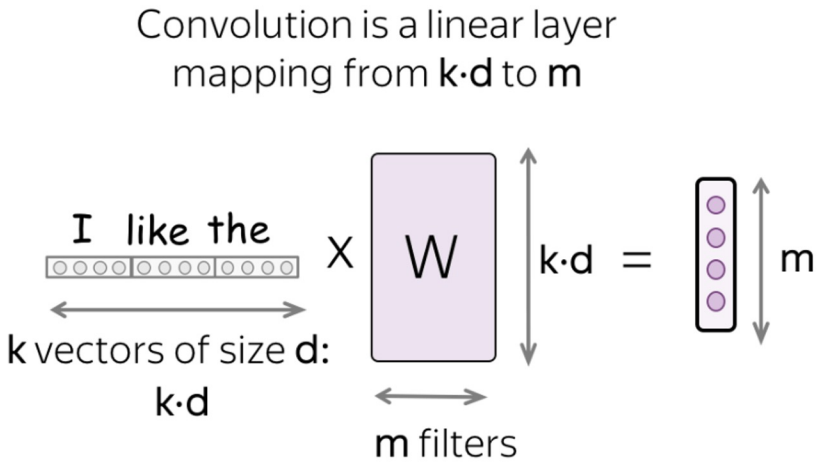


A convolution is a linear layer (followed by a non-linearity) which is applied to each input window. Formally, let us assume that

- $x_1,\ldots,x_n$ is representations of the input word
- K(kernel size): the length of a convolution window
- M(output channels): number of convolution filters
- Then a convolution is a linear layer that maps k.d to m







Building Blocks: Convolution

- For a K-sized window , the convolution takes the concatenation of these vectors $x_i, \dots x_{i+k-1}$

$$u_i = [x_i, \dots x_{i+k-1}] \in R^{k \cdot d}$$

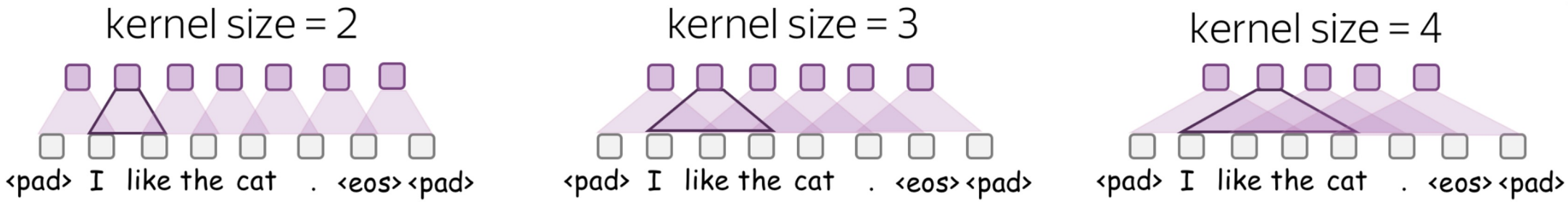
- Multiplies by the convolution matrix:

$$F_i = u_i \times W$$

- A convolution goes over an input with a sliding window and applies the same linear transformation to each window.

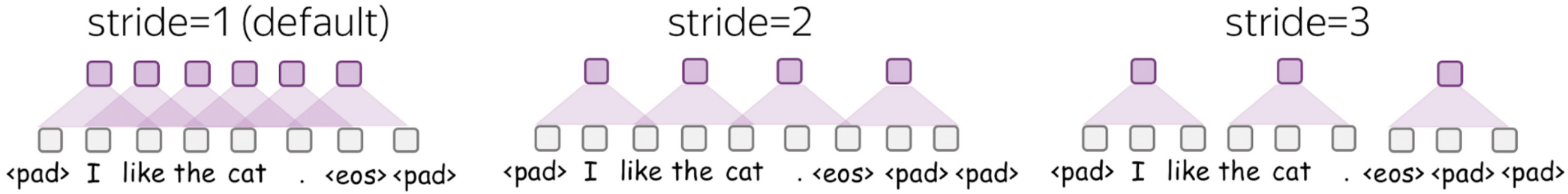
Parameters: Kernel size, Stride, Padding, Bias

- **Kernel size** is the number of input elements (tokens) a convolution looks at each step. For text, typical values are 2-5.



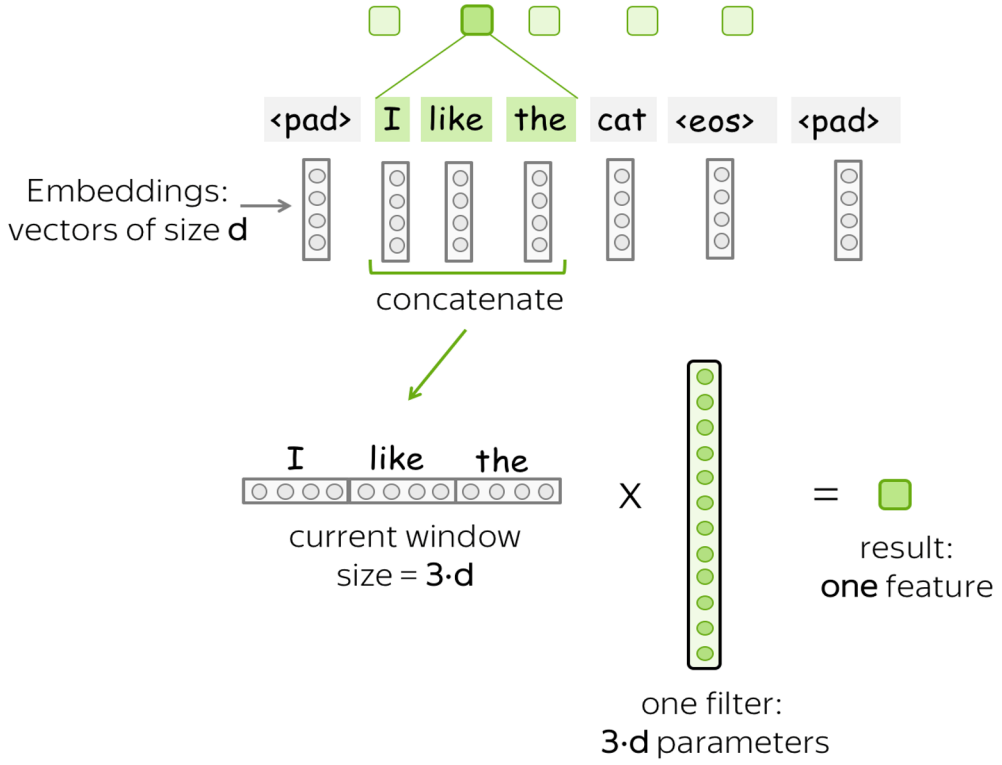
- **Stride** tells how much to move the filter at each step. For example, a stride equal to 1 means that we move the filter by 1 input element (pixel for images, token for texts) at each step.

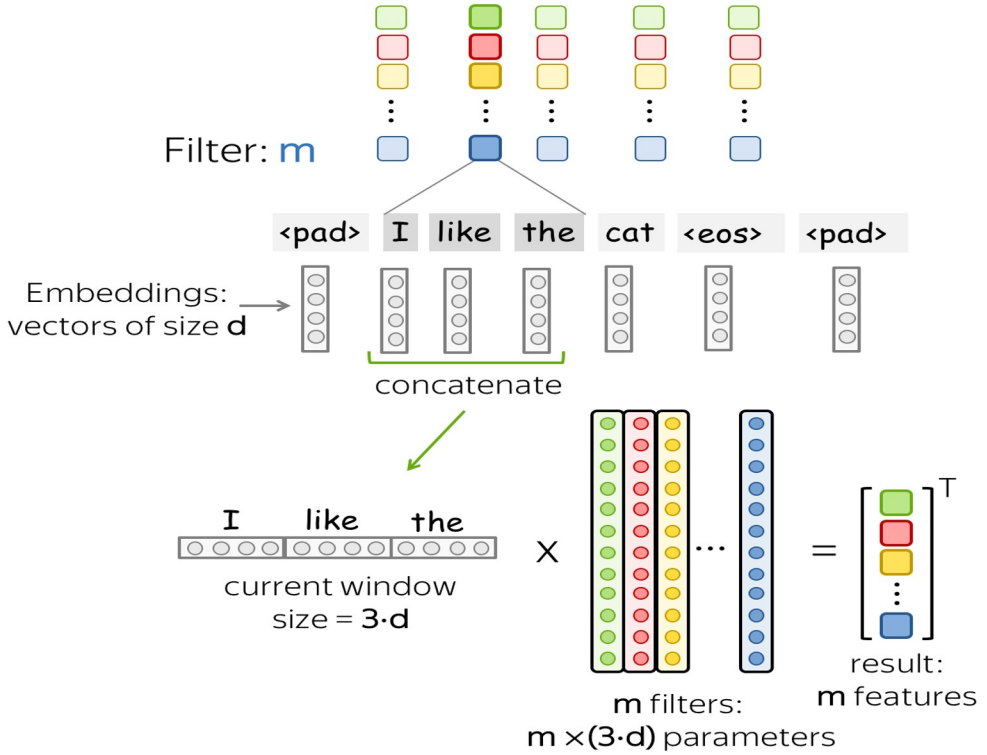
- **Padding**: Add zero vectors to both sides.



- A filter takes vector representations in a current window and transforms them linearly into a single feature.

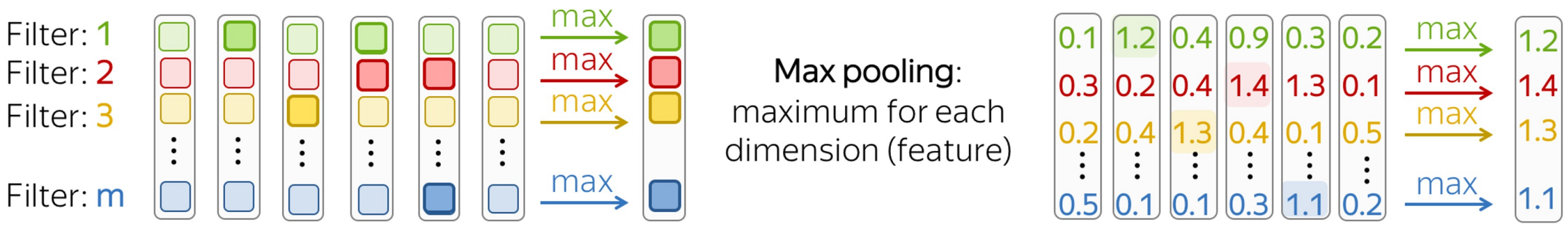
- Each Filter Extracts a Feature
- One filter - one feature extractor
- m filters: m feature extractors





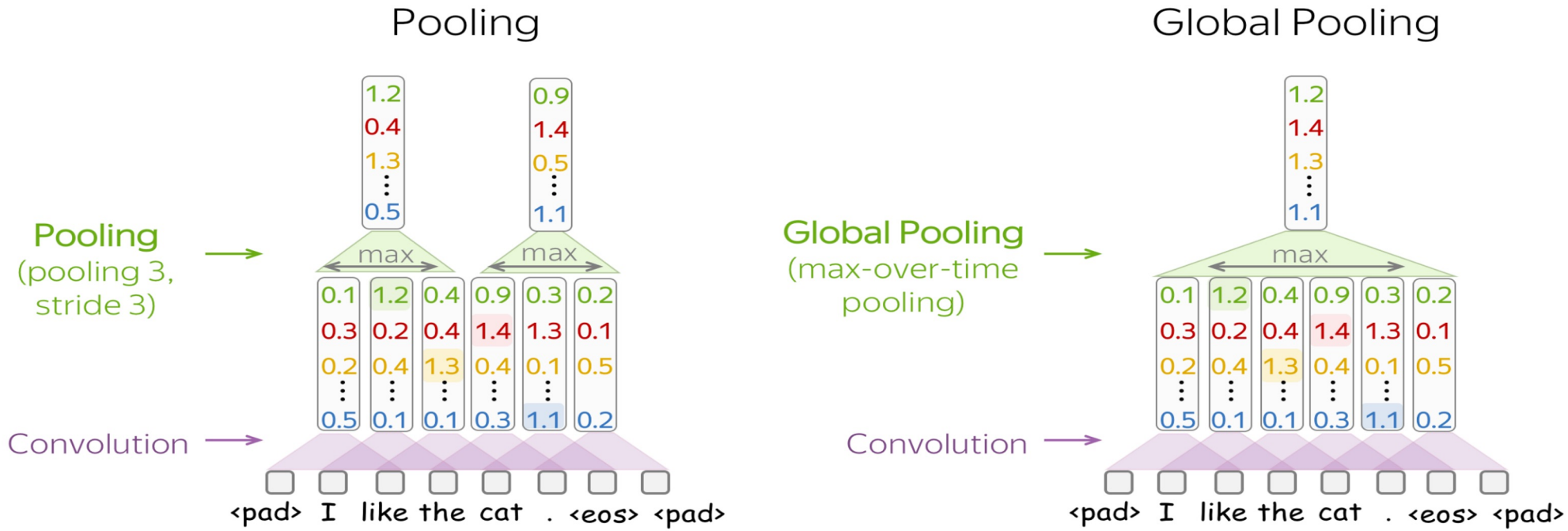
Building Blocks: Pooling

- After a convolution extracted ***m*** features from each window, a pooling layer summarizes the features in some region.
  - Pooling layers are used to reduce the input dimension
- Max-pooling: it takes maximum over each dimension.
- Mean-pooling works similarly but computes mean over each feature instead of maximum.



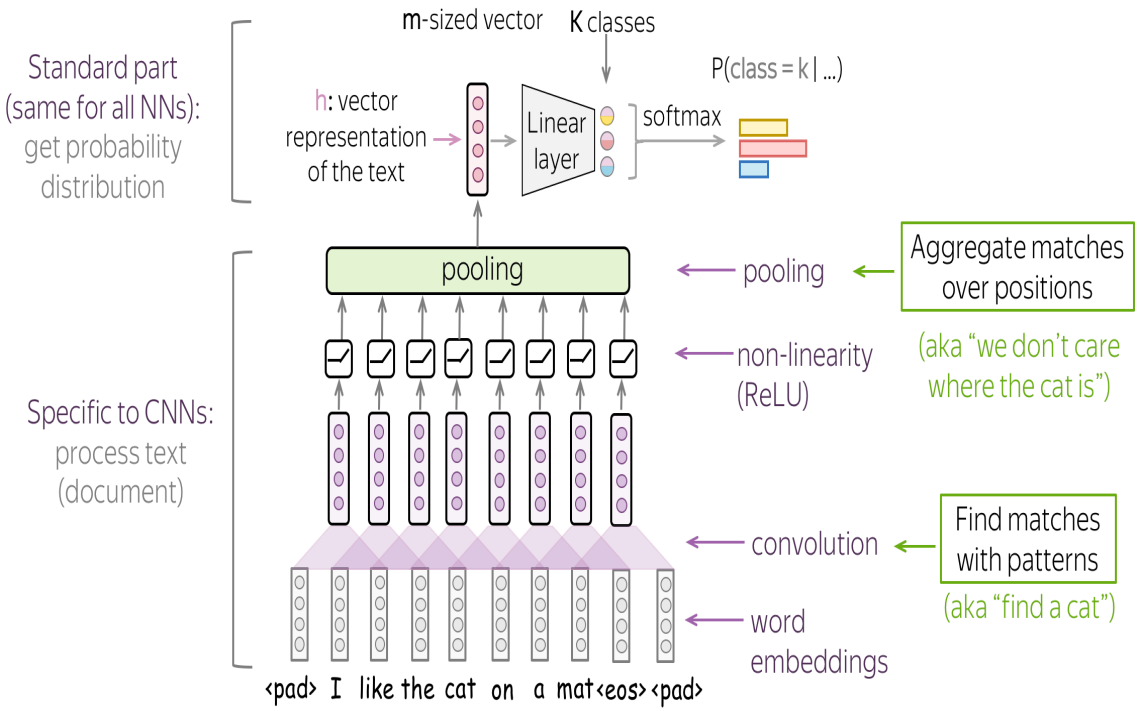
Pooling and Global Pooling

- Pooling is applied over features in each window independently, while global pooling performs over the whole input.
- For texts, global pooling is often used to get a single vector representing the whole text; such global pooling is called max-over-time pooling, where the "time" axis goes from the first input token to the last.

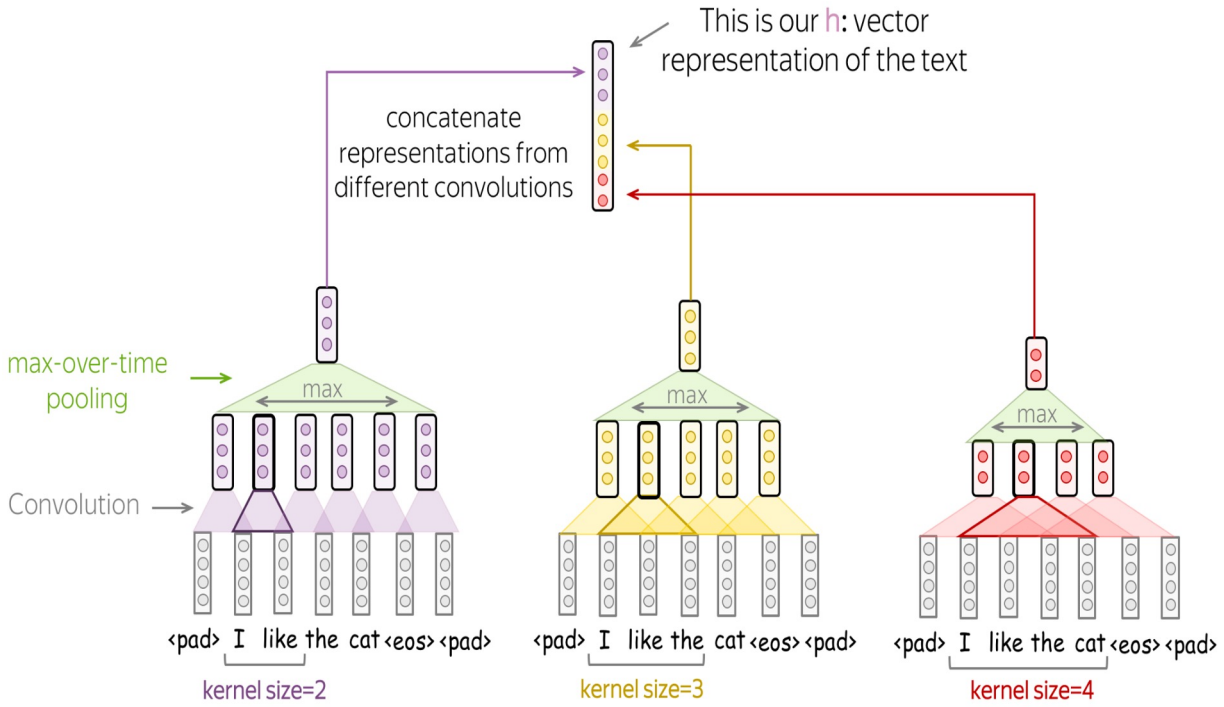


Specific Tasks: Text Classification

One Convolutions with one Kernel



Several Convolutions with Different Kernel Sizes



3 channel, 1D convolution with padding = 1 and max pooling over time

Ø	**0.0**	**0.0**	**0.0**	**0.0**
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
governmen t	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
Ø	**0.0**	**0.0**	**0.0**	**0.0**

Ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,Ø	−0.5	−0.9	0.1

max p	0.3	1.6	1.4

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

conv1d, padded with average pooling over time

ø	**0.0**	**0.0**	**0.0**	**0.0**
tentative	0.2	0.1	−0.3	0.4
deal	0.5	0.2	−0.3	−0.1
reached	−0.1	−0.3	−0.2	0.4
to	0.3	−0.3	0.1	0.1
keep	0.2	−0.3	0.4	0.2
governmen t	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3
ø	**0.0**	**0.0**	**0.0**	**0.0**

ø,t,d	−0.6	0.2	1.4
t,d,r	−1.0	1.6	−1.0
d,r,t	−0.5	−0.1	0.8
r,t,k	−3.6	0.3	0.3
t,k,g	−0.2	0.1	1.2
k,g,o	0.3	0.6	0.9
g,o,ø	−0.5	−0.9	0.1

ave p	−0.87	0.26	0.53

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

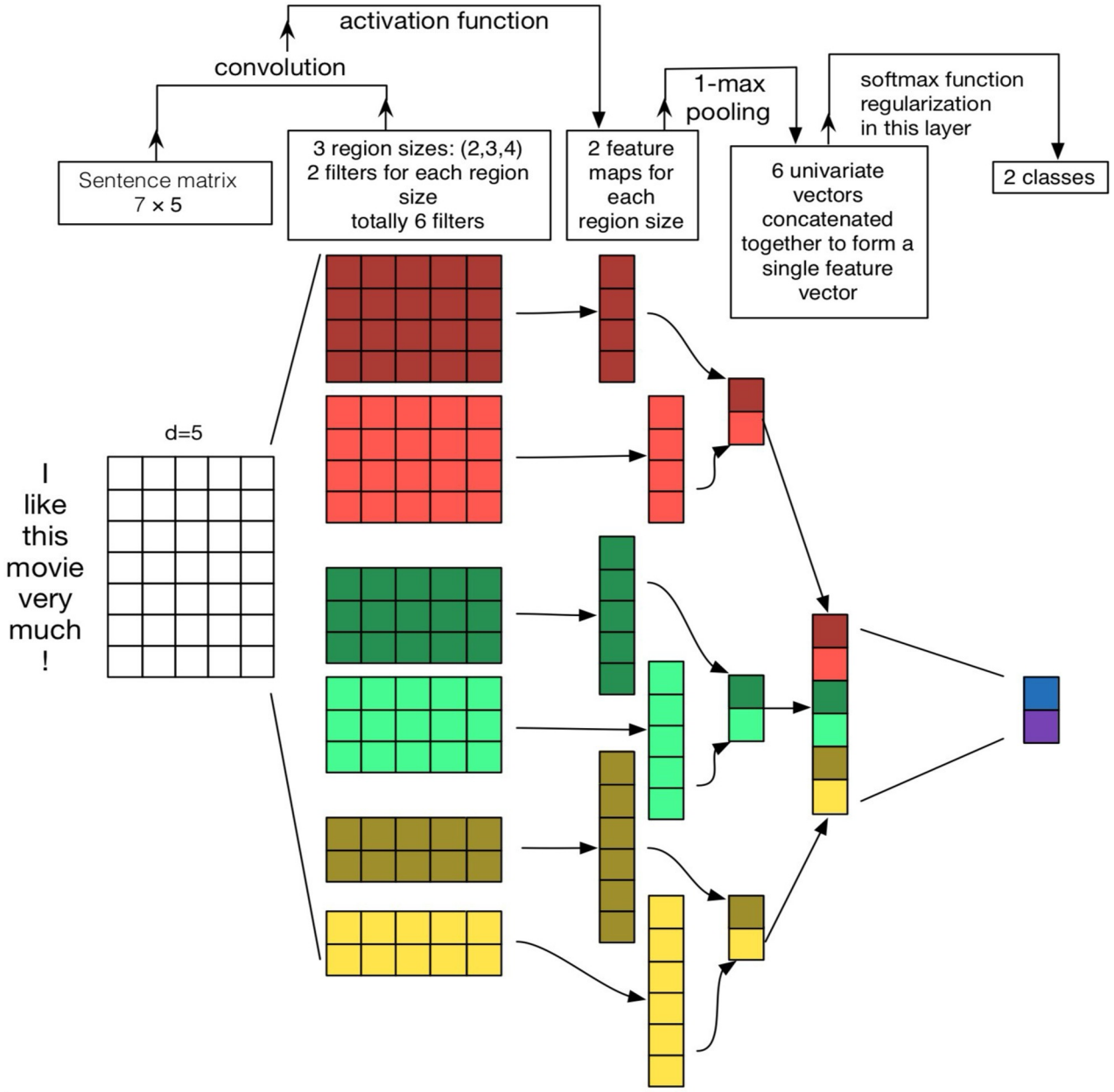
1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

From:

Zhang and Wallace (2015) A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification

https://arxiv.org/pdf/ 1510.03820.pdf



Neural Language Models (LMs)

Feedforward Neural Language Models

- **Language Modeling**: Calculating the probability of the next word in a sequence given some history.
- Different from n-gram models that define formulas based on global corpus statistics, neural models teach a network to predict these probabilities.
  - **Task**: predict the next word w_t given prior words w_{t-1}, w_{t-2}, w_{t-3}, …
  - **Problem**: Now we're dealing with sequences of arbitrary length.
  - **Solution**: Sliding windows (of fixed length)

$$P(w_1, \ldots, y_n) = P(w_1) \cdot \underset{n}{P}(w_2|w_1) \cdot \cdots \cdot$$

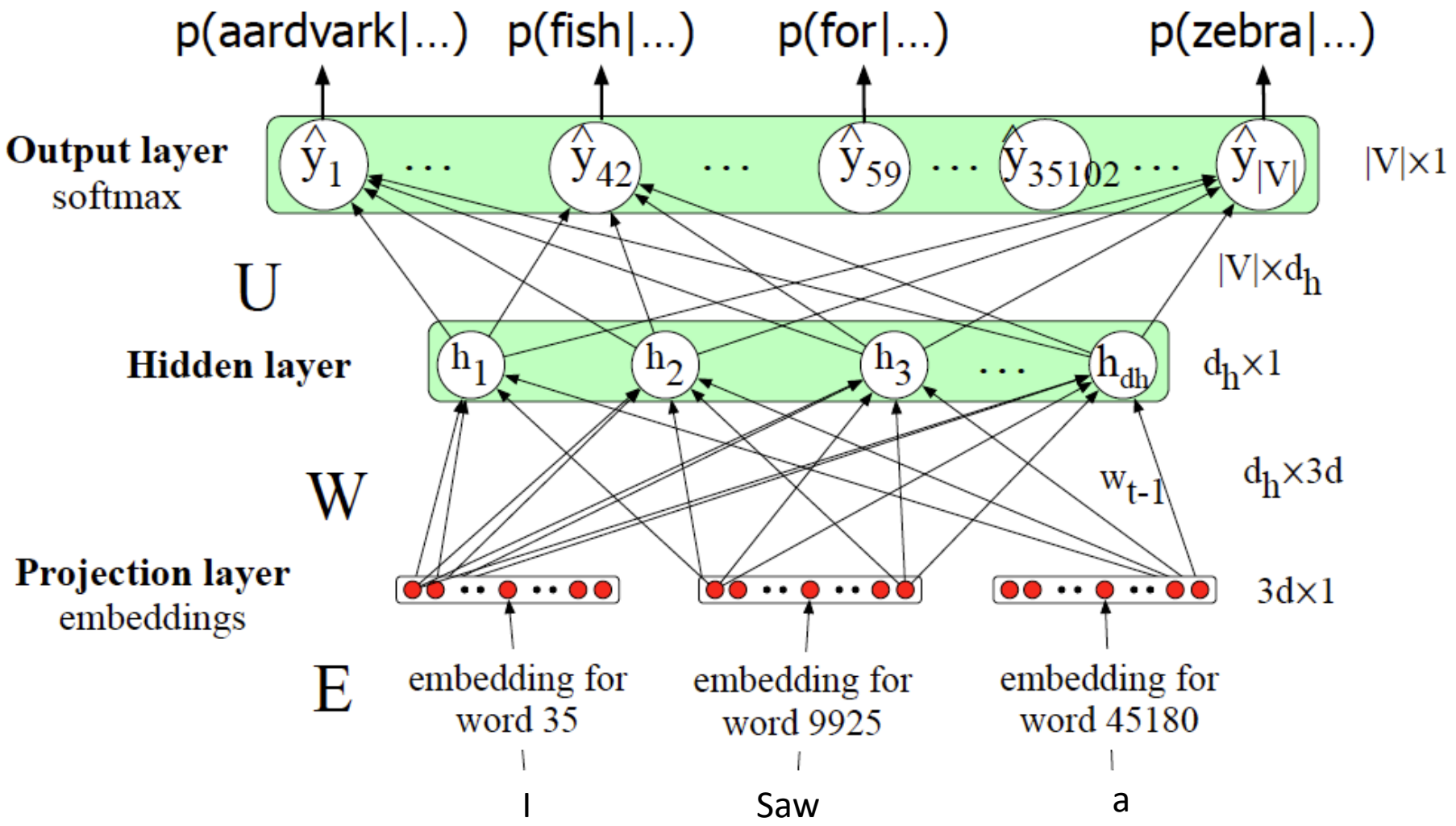
$$P(w_n|w_1, \ldots, w_{n-1}) = \prod_{t=1}^{n} P(w_t|w_{t-k+1}).$$

Feedforward neural language models

- Key idea: Instead of estimating raw probabilities, use a neural network to fit the probabilistic distribution of language!
  - $P(w \mid \text{I am a good})$
  - $P(w \mid \text{I am a great})$

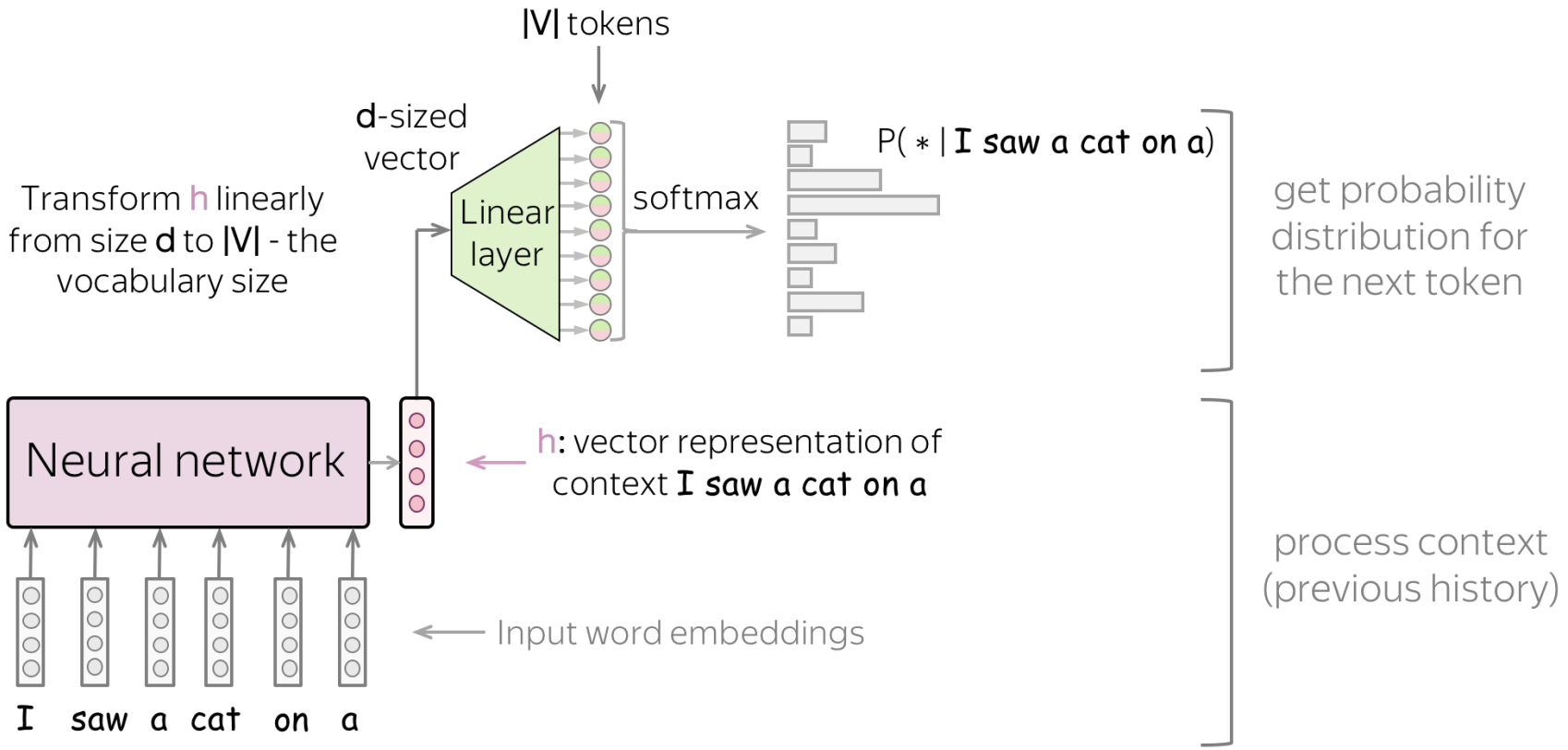
- Key ingredient: word embeddings e(good) e(great)

Neural Language Model



Neural network for LM

- Feed a word embedding for previous (context) words into a network;
- Get vector representation of context from the network;
- From this vector representation, predict a probability distribution for the next token



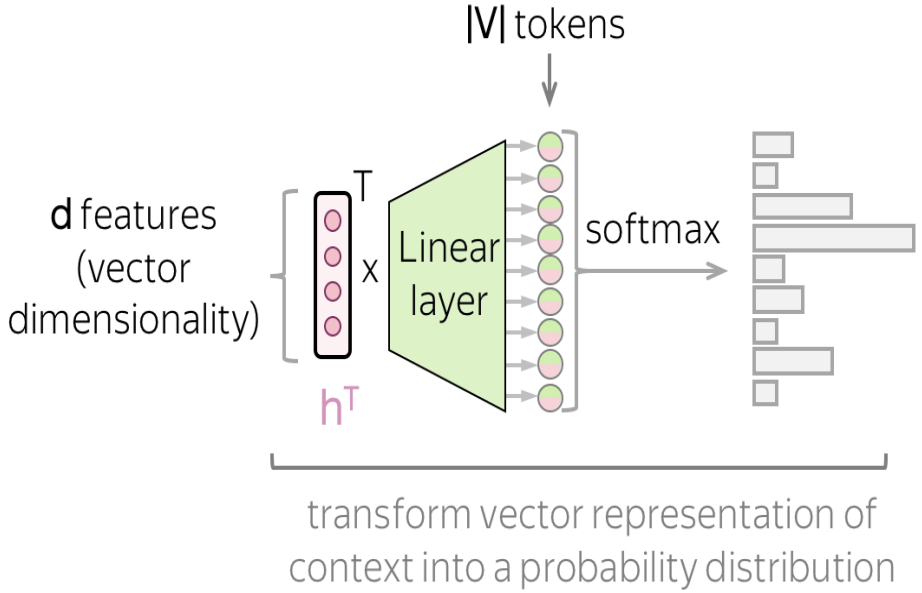
We have:

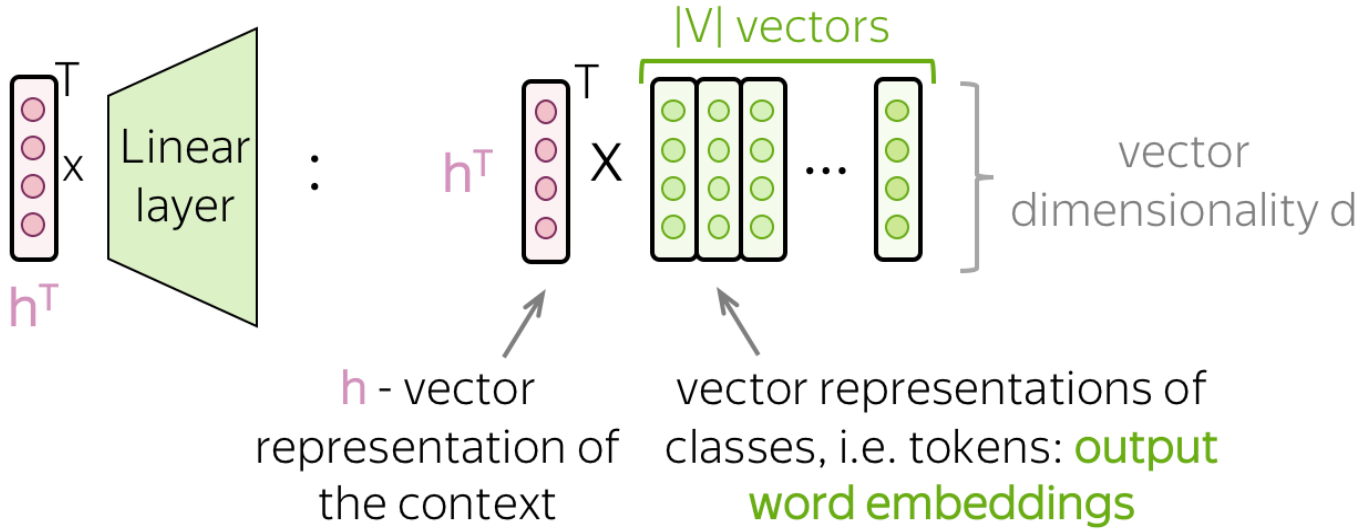
- h - vector of size **d**

We need:

- vector of size |V| – probabilities for all tokens in the vocabulary

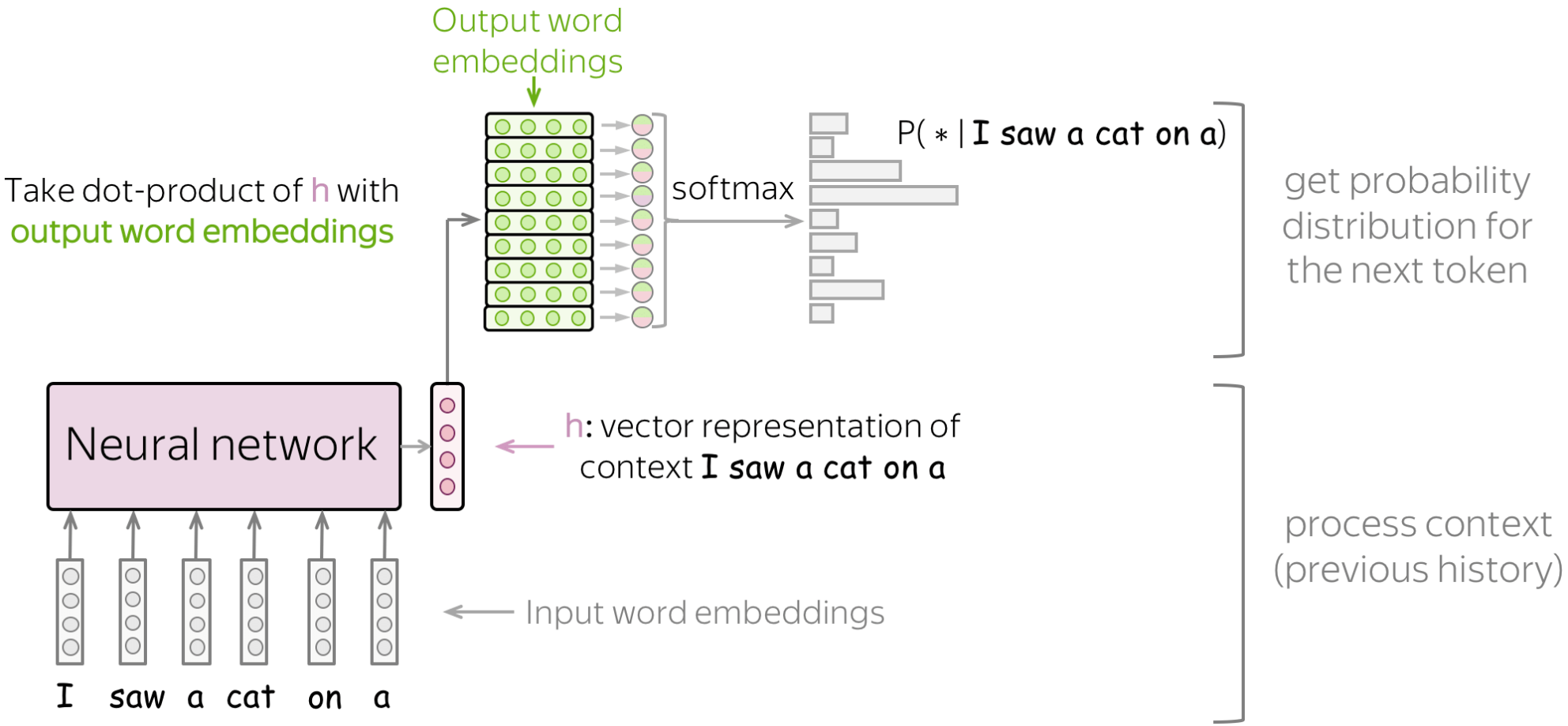
→ Transform linearly from size **d** to size |V|





$$p(w_t|w_{t-k+1}) = \frac{exp(h_t^T e_{w_t})}{\sum_{w \in V} exp(h_t^T e_w)}$$

Applying the final linear layer is equivalent to evaluating the dot product between text representation *h* and each of the output word embeddings.

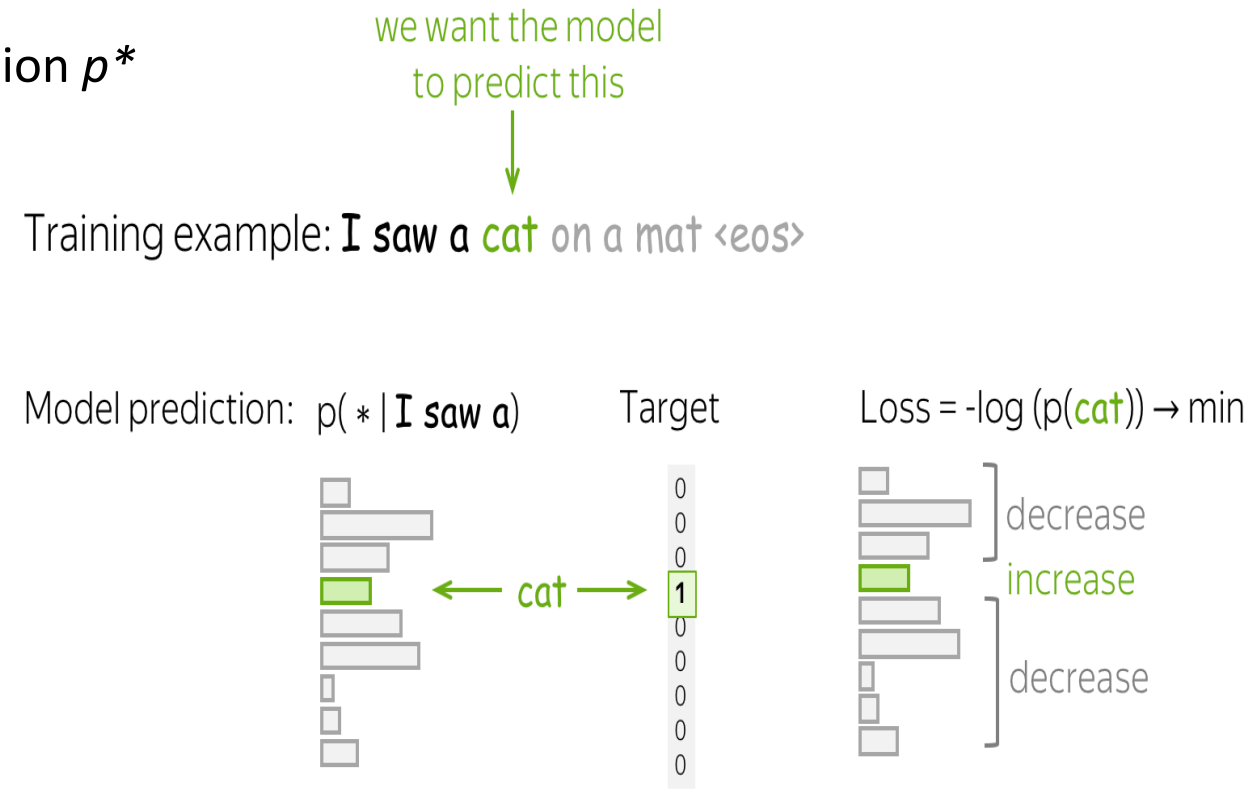


Training and the Cross-Entropy Loss

- Neural LMs are trained to predict probability distributions of the next token given the previous context.
- At each step, we maximize the probability a model assigns to the correct token.

- Cross-entropy loss for the target distribution p^* and the predicted distribution p is

$$Loss(p^*, p) = -p^* \log(p) = -\sum_{i=1}^{|V|} p_i^* \log(p_i).$$



Feedforward neural language models

- Feedforward neural language models approximate the probability based on the previous *m* (e.g., 5) words - *m* is a hyper-parameter!

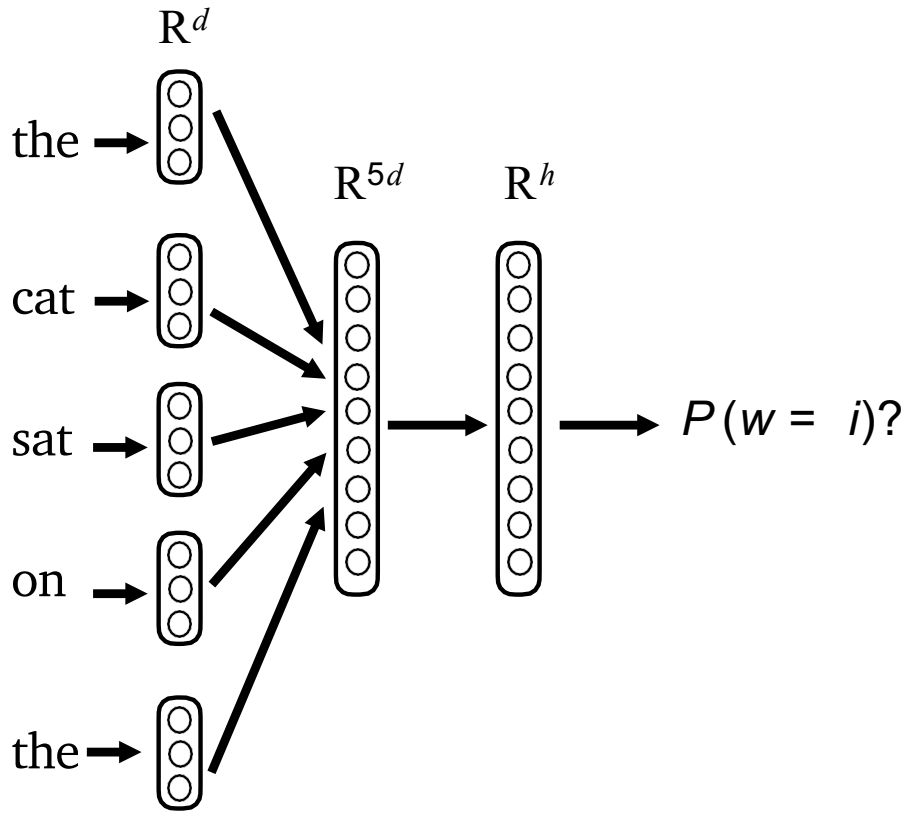
$$P(x_1, x_2, \ldots, x_n) \approx \prod_{i=1}^{n} P(x_i | x_{i-m+1}, \ldots, x_{i-1})$$

P(mat | the cat sat on the) = ?

d: word embedding size

h: hidden size

It is a |V|-way classification problem!



Feedforward neural language models

P(mat | the cat sat on the) = ? d: word embedding size h: hidden size

- Input layer (m= 5):

  x = [*e*(the); *e*(cat); *e*(sat); *e*(on); *e*(the)] ∈ $\mathbb{R}^{md}$

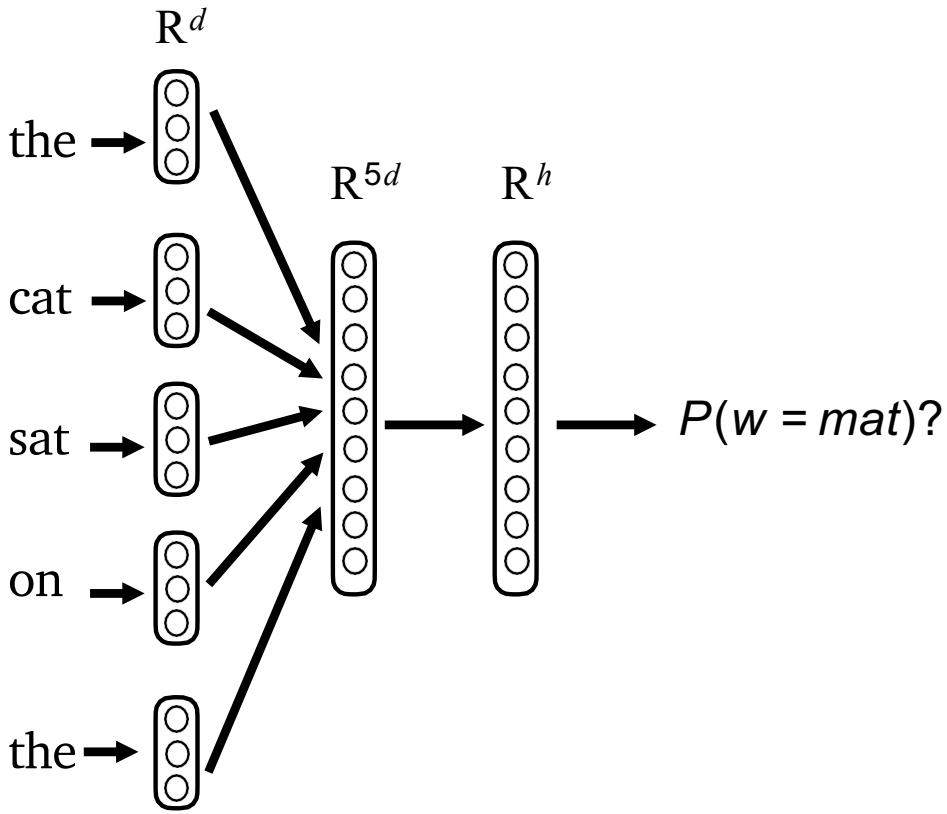
- Hidden layer:

  $h = \tanh(Wx + b) \in \mathbb{R}^h$

- Output layer

  $z = Uh \in R|V|$

  $P(w = mat \mid \text{the cat sat on the}) = \text{softmax}_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_k e^{z_k}}$



What are the dimensions of W and U?

d: word embedding size, h: hidden size

(a) $\mathbf{W} \in \mathbb{R}^{h \times d}, \mathbf{U} \in \mathbb{R}^{|V| \times h}$

(b) $\mathbf{W} \in \mathbb{R}^{h \times 5d}, \mathbf{U} \in \mathbb{R}^{|V| \times h}$

(c) $\mathbf{W} \in \mathbb{R}^{h \times 5d}, \mathbf{U} \in \mathbb{R}^{|V| \times d}$

(d) $\mathbf{W} \in \mathbb{R}^{h \times d}, \mathbf{U} \in \mathbb{R}^{d \times h}$

Correct: (b)

- Input layer (m= 5):

  $\mathbf{x} = [e(\text{the}); e(\text{cat}); e(\text{sat}); e(\text{on}); e(\text{the})] \in \mathbb{R}^{md}$

- Hidden layer:

  $\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + b) \in \mathbb{R}^{h}$

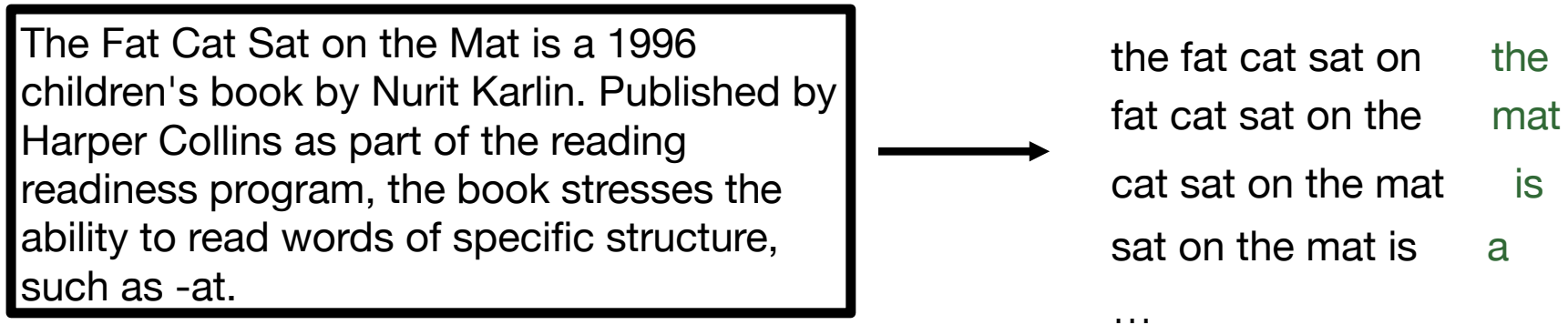
- Output layer

  $\mathbf{z} = \mathbf{U}\mathbf{h} \in \mathbb{R}^{|V|}$

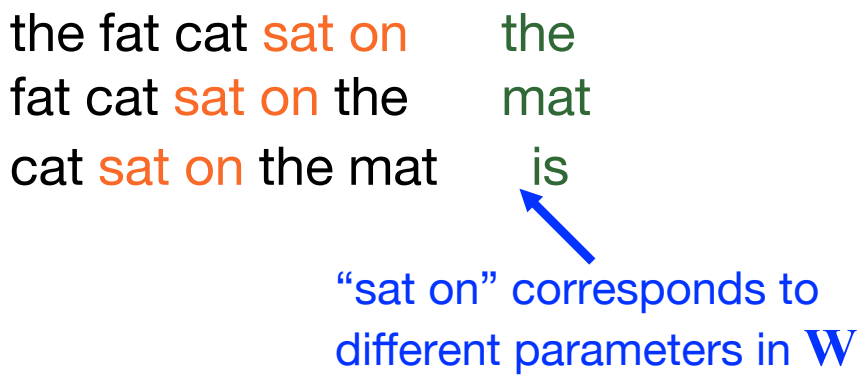
$$P(w = i \mid \text{the cat sat on the}) = \text{softmax}_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

Feed forward neural language models

- How to train this model? A: Use a lot of raw text to create training examples and run gradient-descent optimization!



- Limitations?
  - **W linearly** scales with the context size *m*
  - The models learns separate patterns for different positions!
- Better solutions: recurrent NNs, Transformers..



Next lecture: RNNs and sequence-to-sequence modelling