

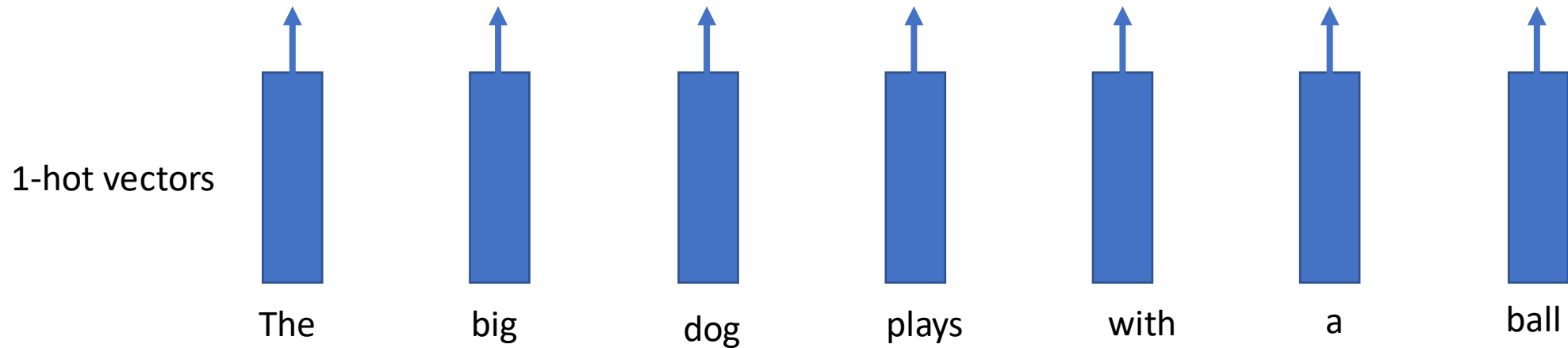


# Deep Learning for Vision & Language

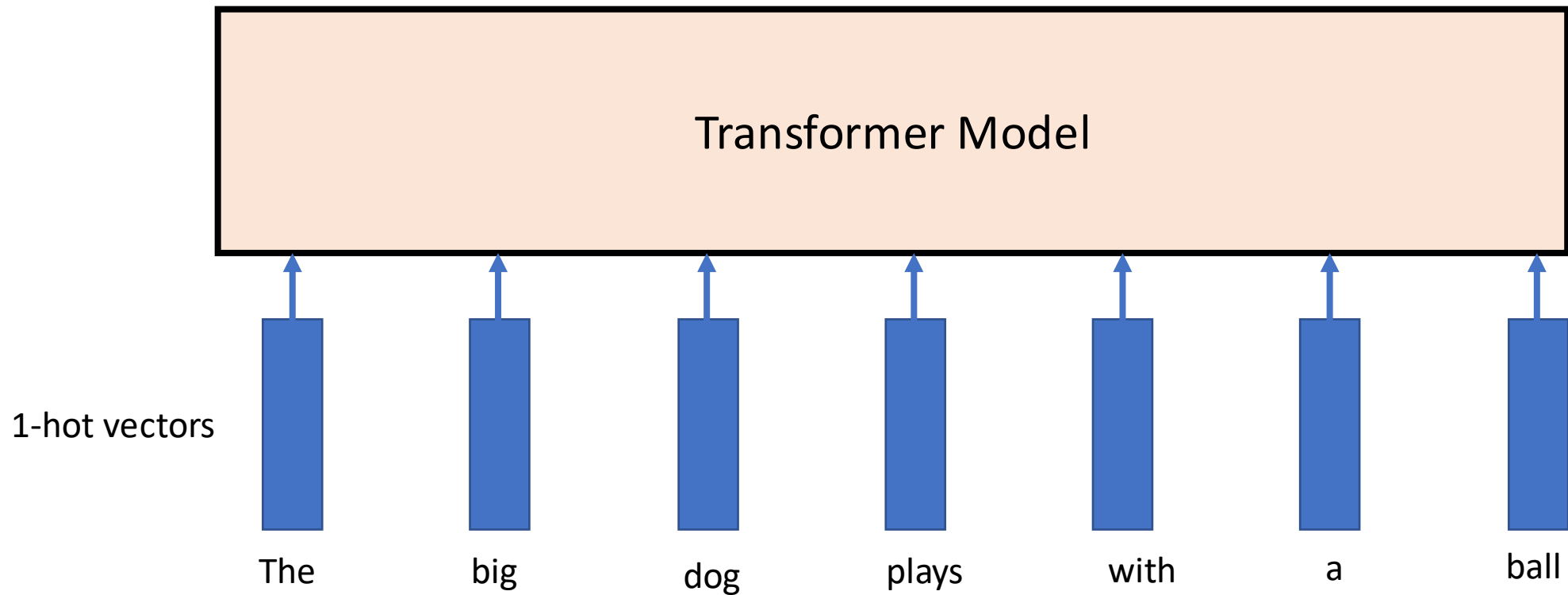
Natural Language Processing: Tokenizers, Sequence Modeling



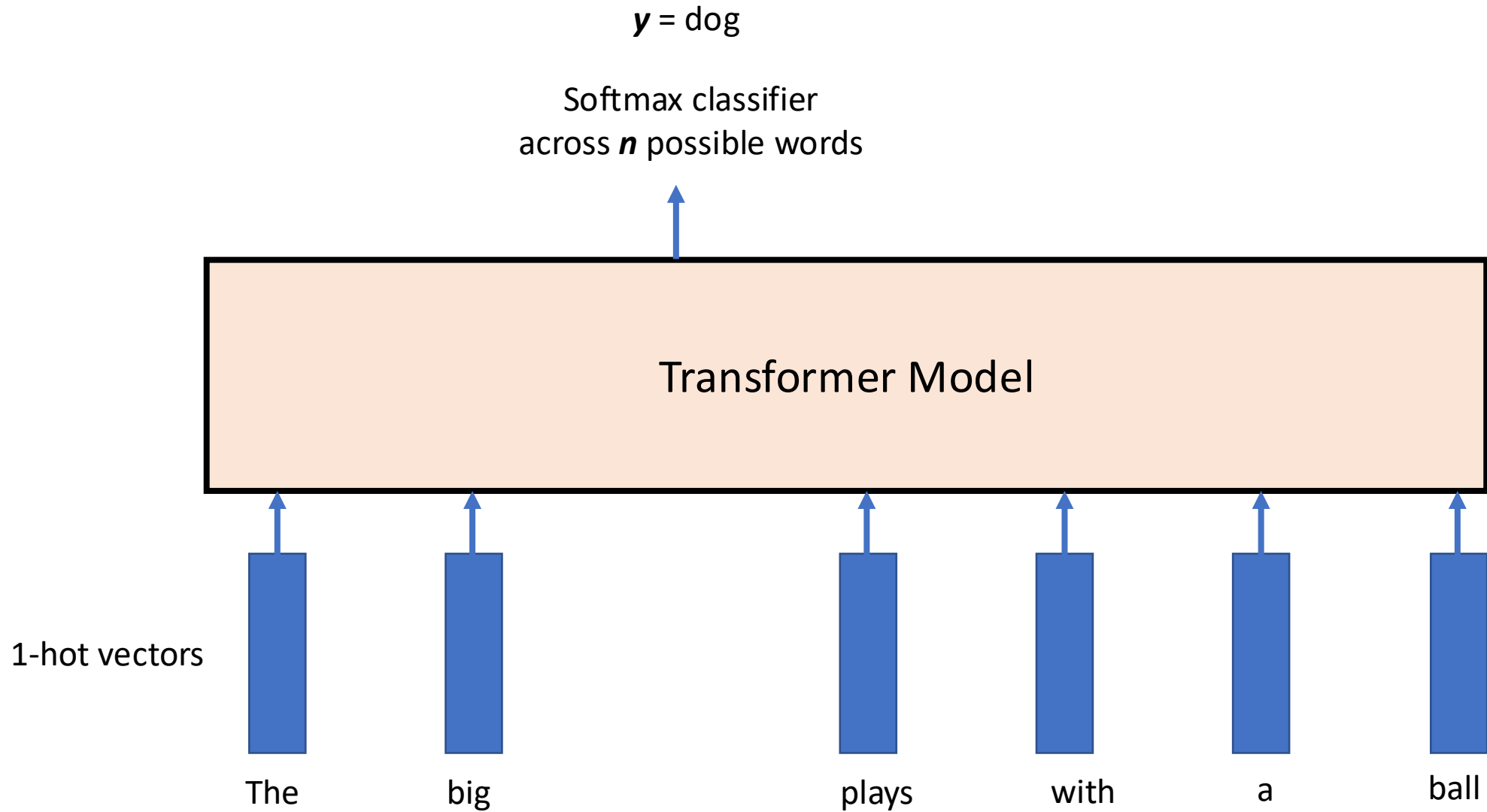
# Pre-trained Language Models



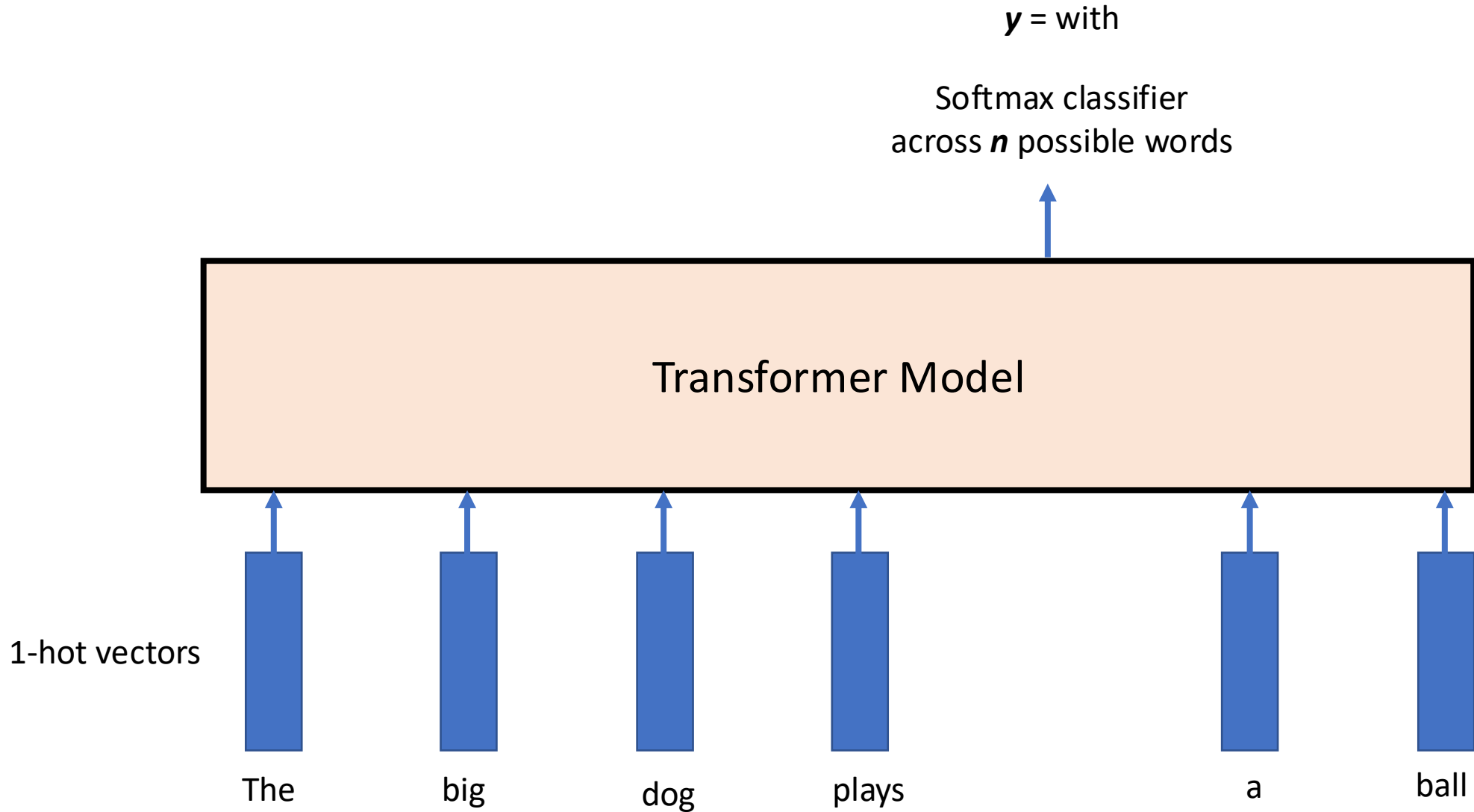
# Pre-trained Language Models



# Pre-trained Language Models



# Pre-trained Language Models



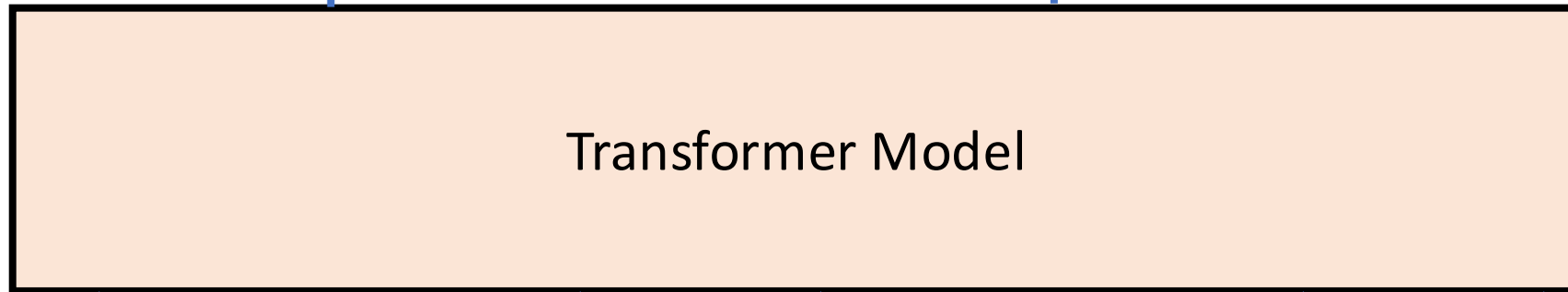
# Pre-trained Language Models

$y_1 = \text{big}$

$y_2 = \text{with}$

Softmax classifier  
across  $n$  possible words

Softmax classifier  
across  $n$  possible words



Transformer Model

1-hot vectors



The



dog



plays

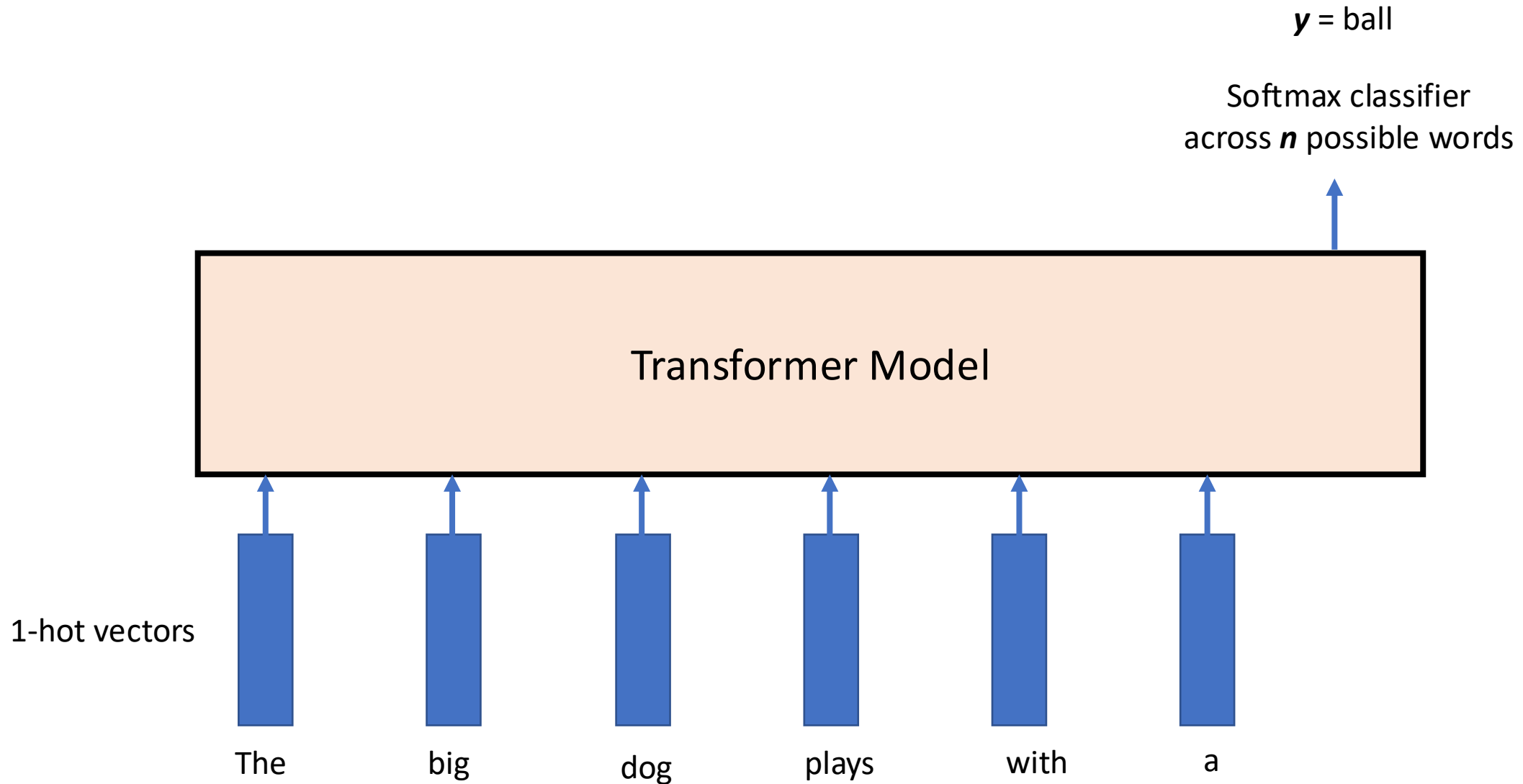


a



ball

# Generative Language Models



# Practical Issues - Tokenization

- For each text representation we usually need to separate a sentence into tokens – we have assumed words in this lecture (or pairs of words) – but tokens could also be characters and anything in-between.
- Word segmentation can be used as tokenization.
  - In the assignment I was lazy I just did “my sentence”.split(“ ”) and called it a day.
  - However, even English is more difficult than that because of punctuation, double spaces, quotes, etc. For English I would recommend you too look up the great word tokenization tools in libraries such as Python’s NLTK and Spacy before you try to come up with your own word tokenizer.

# Issues with Word based Tokenization

- We already mentioned that tokenization can be hard even when word-based for other languages that don't use spaces in-between words.
- Word tokenization can also be bad for languages where the words can be “glued” together like German or Turkish.
  - Remember fünfhundertfünfundfünfzig? It wouldn't be feasible to have a word embedding for every number in the German language.
- It is problematic to handle words that are not in the vocabulary e.g. a common practice is to use a special <OOV> (out of vocabulary) token for those words that don't show up in the vocabulary.

# Tokenization can be complex

- Think of Japanese

- Three vocabularies/sets of symbols:

Katakana and Hiragana symbols represent syllables / sounds

く = ku, き = gi, ナ = na, ア = a

Kanji represent ideas / words (Chinese characters).

日 = day, sun, 大 = big, 凸 = convex 凹 = concave

- They can be combined – e.g. tomorrow = 明日
- Each symbol also has some structure within the symbols. They are not independently created. e.g. bright = 明るい, rising sun = 旭
- And of course there are no spaces in between the characters.

# Solution: Sub-word Tokenization

- Byte-pair Encoding Tokenization (BPE)
  - Start from small strings and based on substring counts iteratively use larger sequences until you define a vocabulary that maximizes informative subtokens. That way most will correspond to words at the end.
- Byte-level BPE Tokenizer
  - Do the same but at the byte representation level not at the substring representation level.

## Tokenizers

 Rust  passing  license  Apache-2.0  downloads/week  169k

Provides an implementation of today's most used tokenizers, with a focus on performance and versatility.

### Main features:

- Train new vocabularies and tokenize, using today's most used tokenizers.
- Extremely fast (both training and tokenization), thanks to the Rust implementation. Takes less than 20 seconds to tokenize a GB of text on a server's CPU.
- Easy to use, but also extremely versatile.
- Designed for research and production.
- Normalization comes with alignments tracking. It's always possible to get the part of the original sentence that corresponds to a given token.
- Does all the pre-processing: Truncate, Pad, add the special tokens your model needs.

[huggingface/tokenizers](https://github.com/huggingface/tokenizers)

# BPE Tokenization Overview

## Neural Machine Translation of Rare Words with Subword Units

**Rico Sennrich and Barry Haddow and Alexandra Birch**

School of Informatics, University of Edinburgh

{rico.sennrich,a.birch}@ed.ac.uk,bhaddow@inf.ed.ac.uk

- Learn BPE operations (python code on the right) – from the paper.
- Use said operations to construct your sub-word vocabulary.
- Treat each sub-word token as a “word” in any models we will discuss.

---

### Algorithm 1 Learn BPE operations

---

```
import re, collections

def get_stats(vocab):
    pairs = collections.defaultdict(int)
    for word, freq in vocab.items():
        symbols = word.split()
        for i in range(len(symbols)-1):
            pairs[symbols[i],symbols[i+1]] += freq
    return pairs

def merge_vocab(pair, v_in):
    v_out = {}
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?!\S)' + bigram + r'(!\S)')
    for word in v_in:
        w_out = p.sub(' '.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
        'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
    print(best)
```

---

[https://colab.research.google.com/drive/1gUjL\\_h2tXdTtPSfxbB-P-6MkE\\_BMck6gm?usp=sharing](https://colab.research.google.com/drive/1gUjL_h2tXdTtPSfxbB-P-6MkE_BMck6gm?usp=sharing)

# Tokenization used in GPT-3

<https://platform.openai.com/tokenizer>

The cat is in the house

| Tokens | Characters |
|--------|------------|
| 6      | 23         |

The cat is in the house

[464, 3797, 318, 287, 262, 2156]

The geologist made an effort to rationalize the explanation

| Tokens | Characters |
|--------|------------|
| 11     | 59         |

The geologist made an effort to rationalize the explanation

[464, 4903, 7451, 925, 281, 3626, 284, 9377, 1096, 262, 7468]

fünfhundertfünfundfünfzig

| Tokens | Characters |
|--------|------------|
| 21     | 29         |

fünfhundertfünfundfünfzig

[69, 9116, 77, 69, 3907, 71, 4625, 83, 3907, 69, 9116, 77, 69, 3907, 917, 3907, 69, 9116, 77, 69, 38262]

La ardilla va a la universidad

| Tokens | Characters |
|--------|------------|
| 8      | 30         |

La ardilla va a la universidad

[14772, 33848, 5049, 46935, 257, 8591, 5820, 32482]

# Tokenization used in GPT-3

<https://platform.openai.com/tokenizer>

深層学

| Tokens | Characters |
|--------|------------|
| 8      | 3          |



[162, 115, 109, 161, 109, 97, 27764, 99]

কেমন আছেন?

| Tokens | Characters |
|--------|------------|
| 20     | 10         |



[48071, 243, 156, 100, 229, 48071, 106, 48071, 101, 220, 48071, 228, 48071, 249, 156, 100, 229, 48071, 101, 30]

வணக்கம்

| Tokens | Characters |
|--------|------------|
| 21     | 7          |



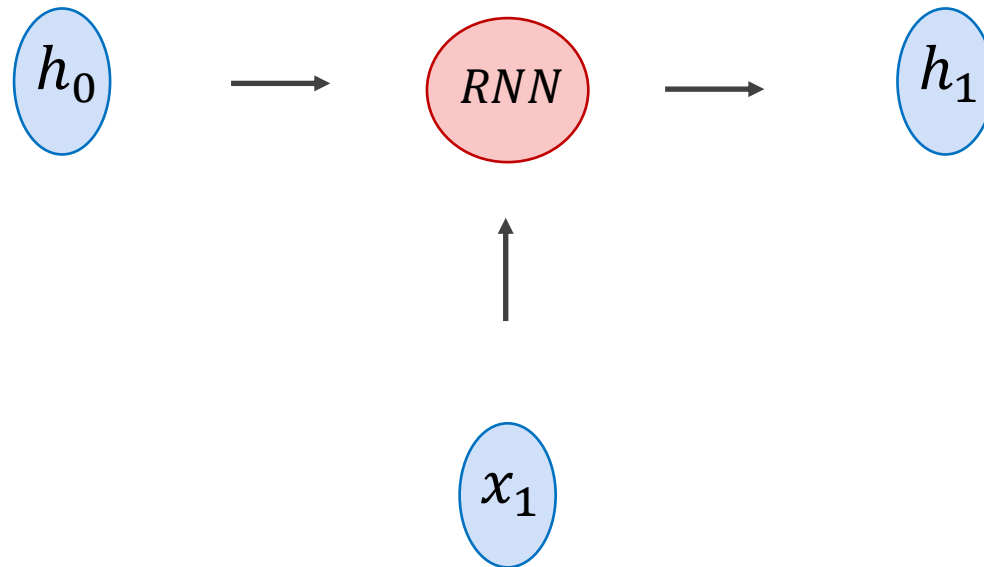
[156, 106, 113, 156, 106, 96, 156, 106, 243, 156, 107, 235, 156, 106, 243, 156, 106, 106, 156, 107, 235]

# Recurrent Neural Networks

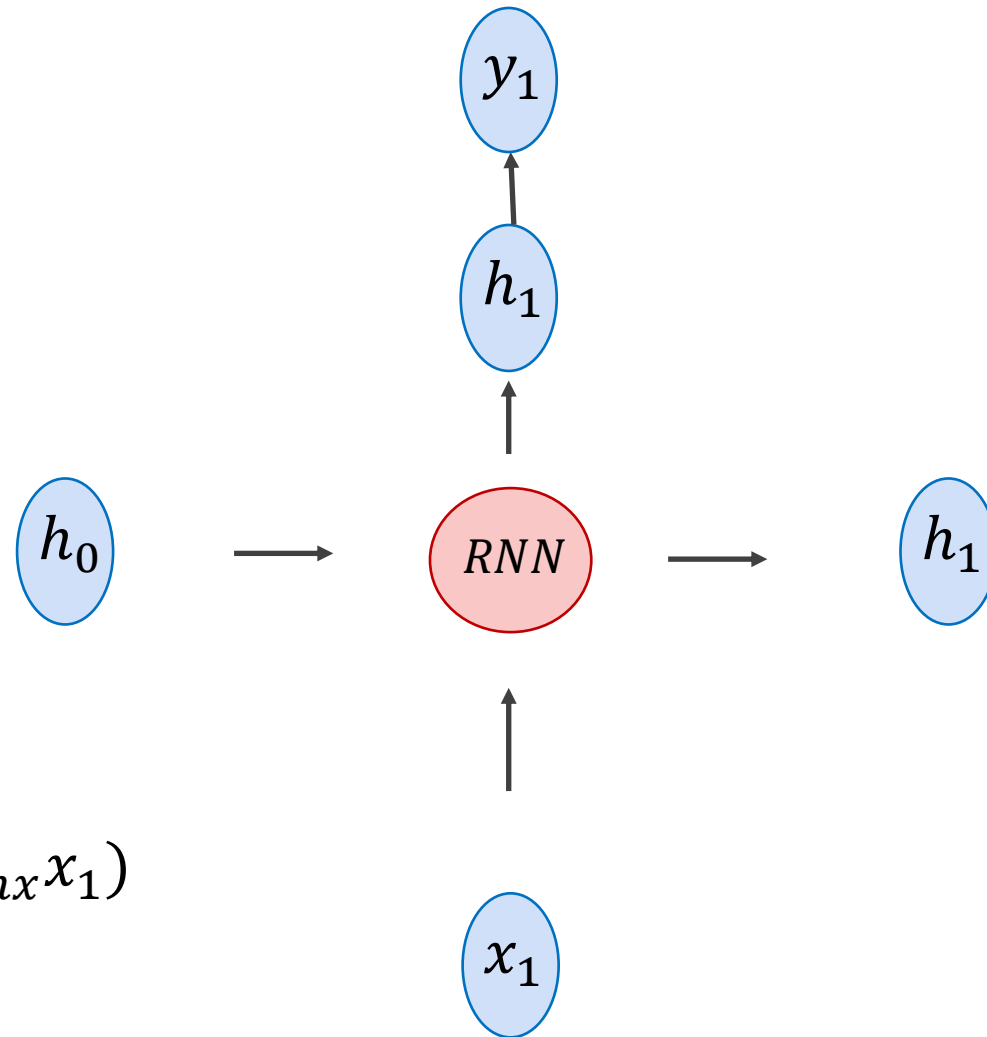
- These are models for handling sequences of things.
- Each input is not a vector but a sequence of input vectors.
- e.g. Each input can be a “word embedding” or any “word” representation – we will use in our first examples one-hot encoded tokens but in practice continuous dense word embeddings are used.

# Recurrent Neural Network Cell

$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$



# Recurrent Neural Network Cell



$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

$$y_1 = \text{softmax}(W_{hy}h_1)$$

# Recurrent Neural Network Cell

$$y_1 = [0.1, 0.05, 0.05, 0.1, 0.7]$$



$$h_1 = [0.1 \ 0.2 \ 0 \ -0.3 \ -0.1]$$

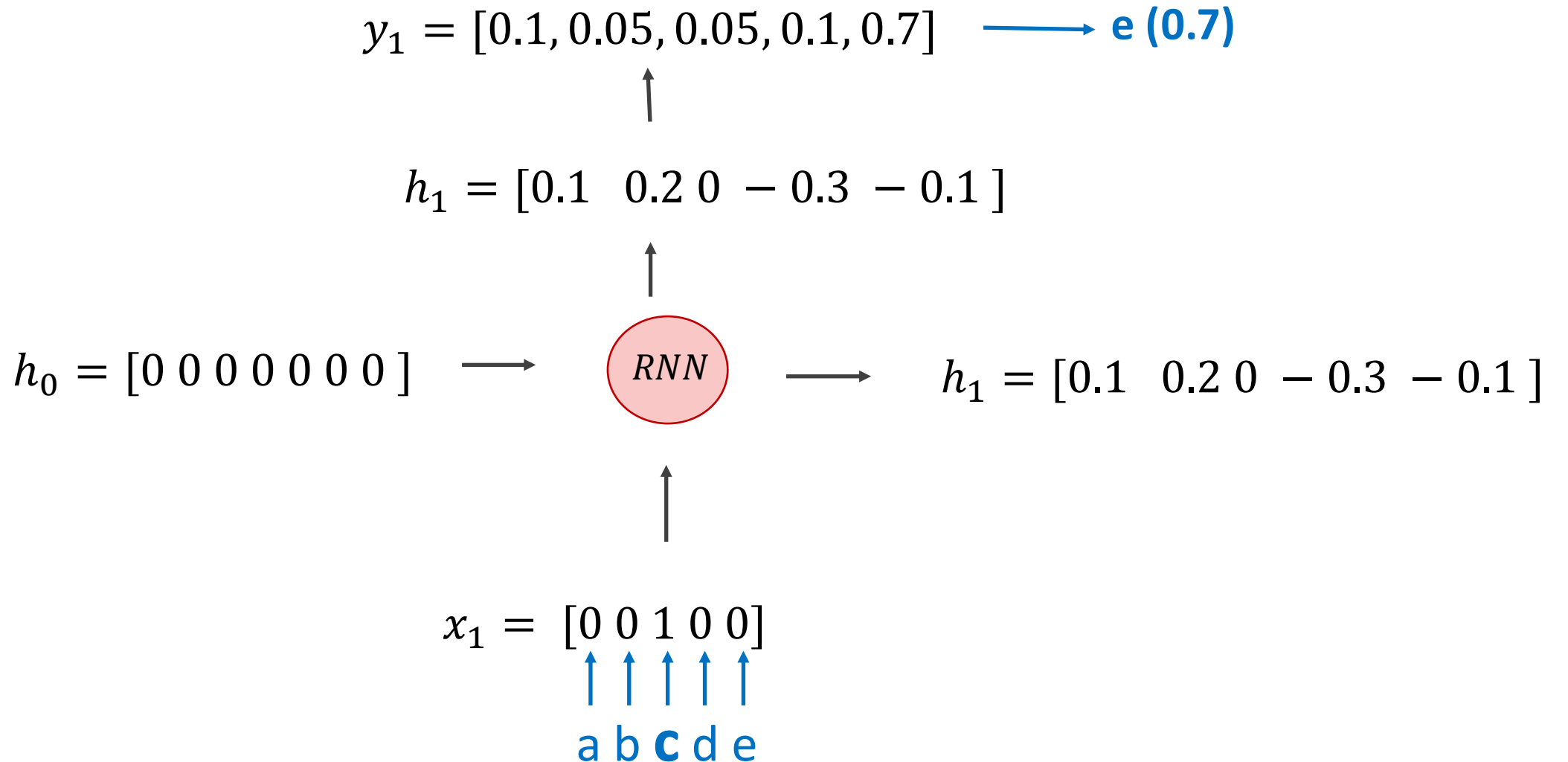


$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

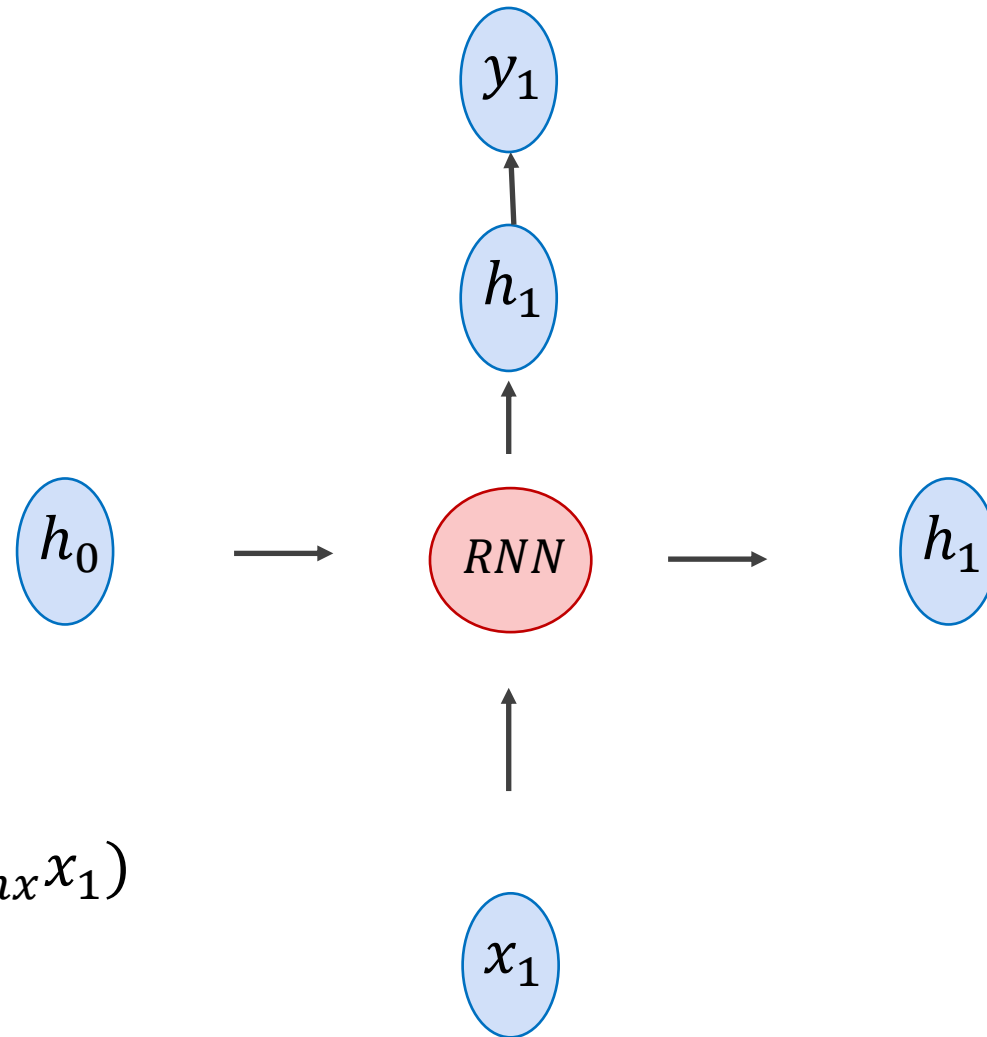
$$x_1 = [0 \ 0 \ 1 \ 0 \ 0]$$

$$y_1 = \text{softmax}(W_{hy}h_1)$$

# Recurrent Neural Network Cell



# Recurrent Neural Network Cell



$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

$$y_1 = \text{softmax}(W_{hy}h_1)$$

# Questions?