STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

CS 584 Natural Language Processing

CNN, Tokenization

Ping Wang
Department of Computer Science
Stevens Institute of Technology

Submissions

- **HW 2**: Due on Oct 30 at 11:59 AM (noon)

Submission and Late Policy

- **10%** penalty for late submission within **24 hours**.
- **40%** penalty for late submissions within **24-48 hours**.
- After 48 hours, you get **NO** points on the assignment.
- You are encouraged to work and discuss in a group, but you have to write down and **submit your OWN answers and codes. Any similar or the same answers/codes will not be graded.**

Today's lecture

- Convolutional Neural Network (CNN)
- From Text to Tokens (Tokenization)

Convolutional Neural Network

Convolutional Neural Networks

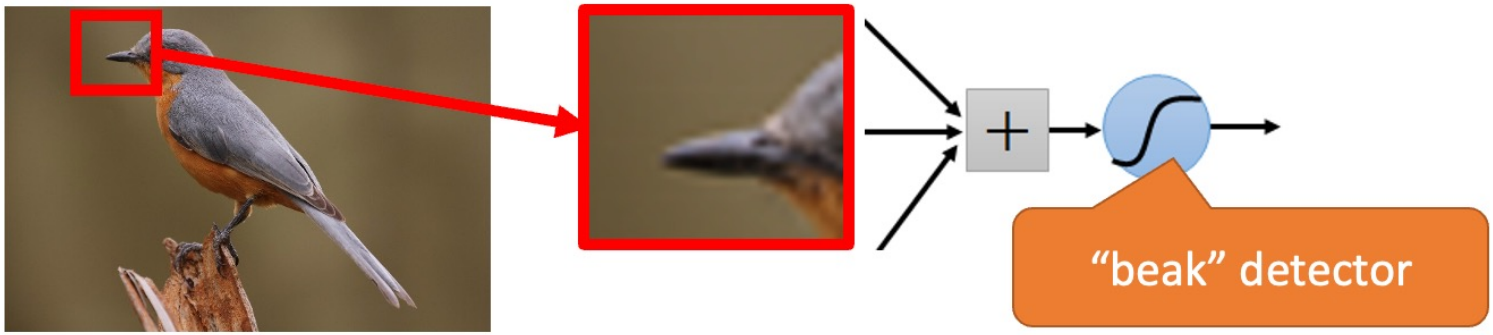
- Originally, convolutional neural networks are specialized networks for application in **computer vision**
  - Accept images as raw input (preserving spatial information)
  - Build up (learn) a hierarchy of features (no hand-crafted features necessary).

Why CNN for Image

- Some patterns are much smaller than the whole image.

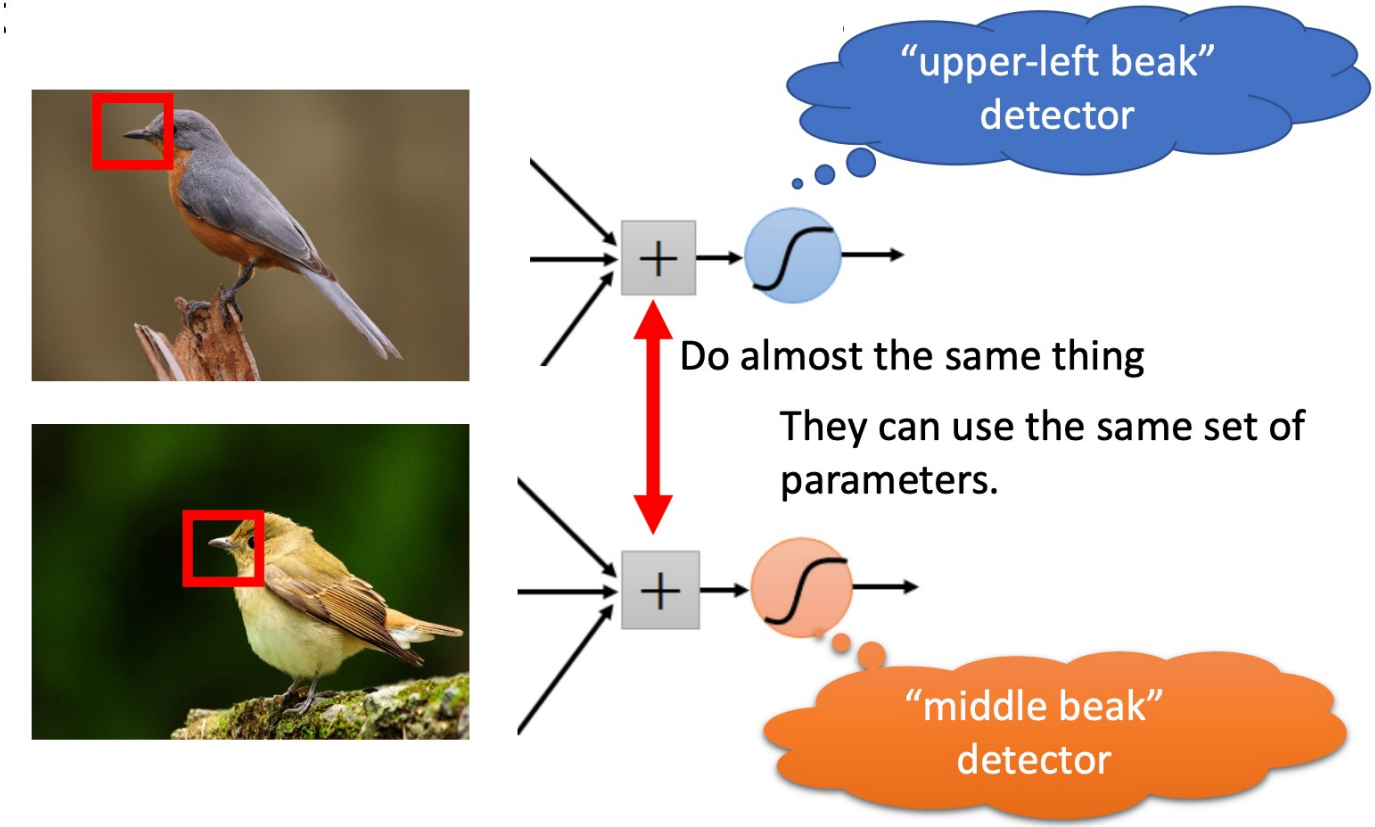
A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters



Why CNN for Image

- The same patterns appear in different regions.



Why CNN for Image

- Subsampling the pixels will not change the object

bird

subsampling

bird

We can subsample the pixels to make image smaller

→ Less parameters for the network to process the image

The Whole CNN

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten

Fully Connected Feedforward network

cat dog ……

The Whole CNN

Property 1

- Some patterns are much smaller than the whole image

Property 2

- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten

CNN – Convolution

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Property 1: Some patterns are much smaller than the whole image

Those are the network parameters to be learned.

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Matrix

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Matrix

Each filter detects a small pattern (3 x 3).

CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

3

6 x 6 image

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

3 -1

6 x 6 image

CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

3

6 x 6 image

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

3 -3

6 x 6 image

We set stride=1 below

CNN – Convolution

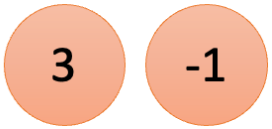
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -1 -3

CNN – Convolution

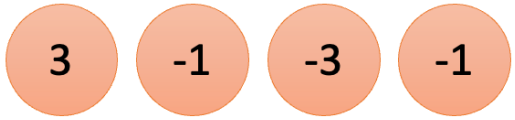
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



CNN – Convolution

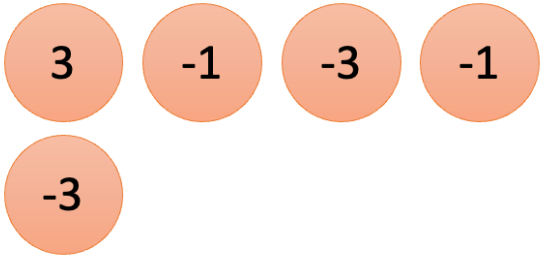
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



CNN – Convolution

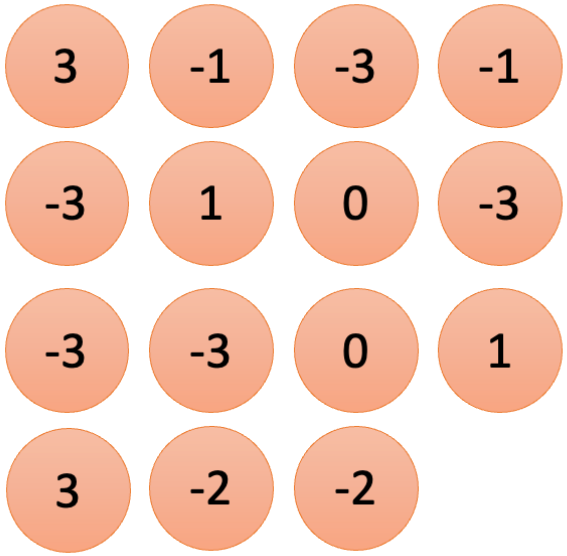
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Property 2: The same patterns appear in different regions.

CNN – Convolution

-1	1	-1
-1	1	-1
-1	1	-1

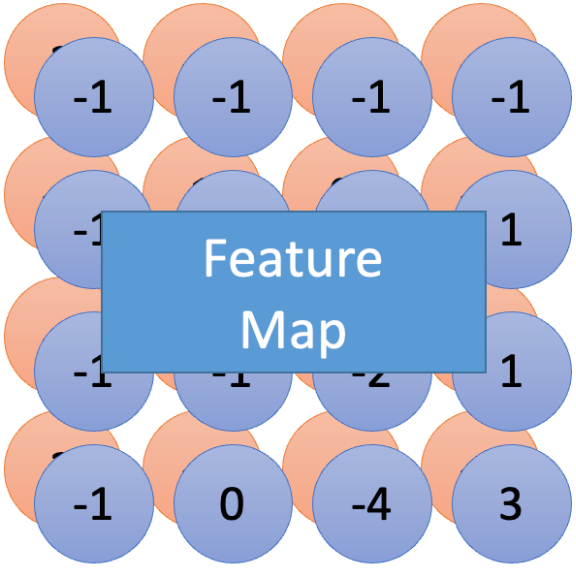
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

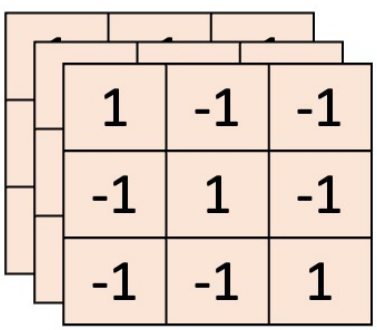
6 x 6 image

Do the same process for every filter

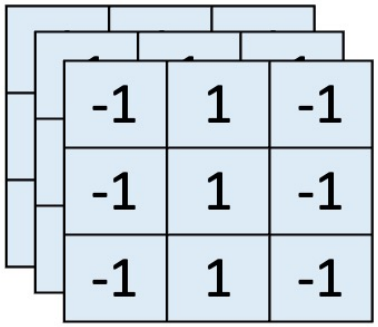


4 x 4 image

CNN – Colorful Image

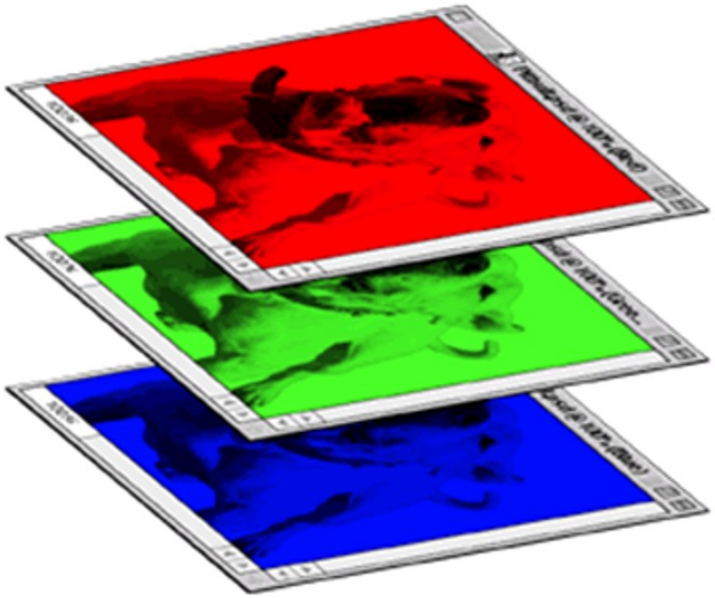


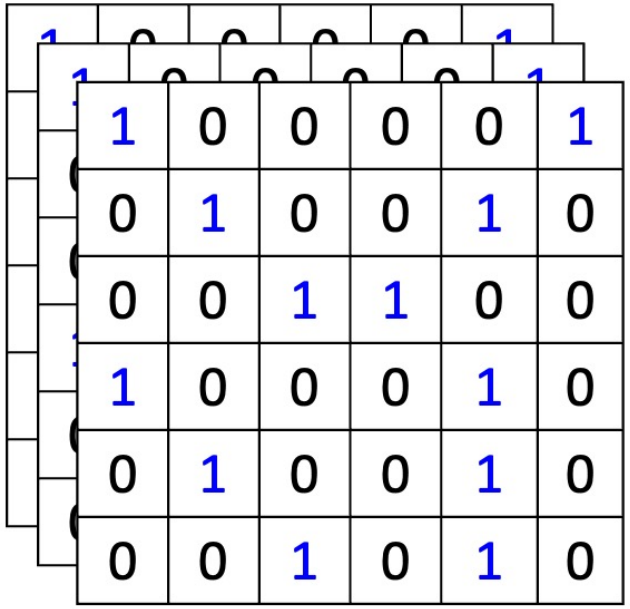
Filter 1



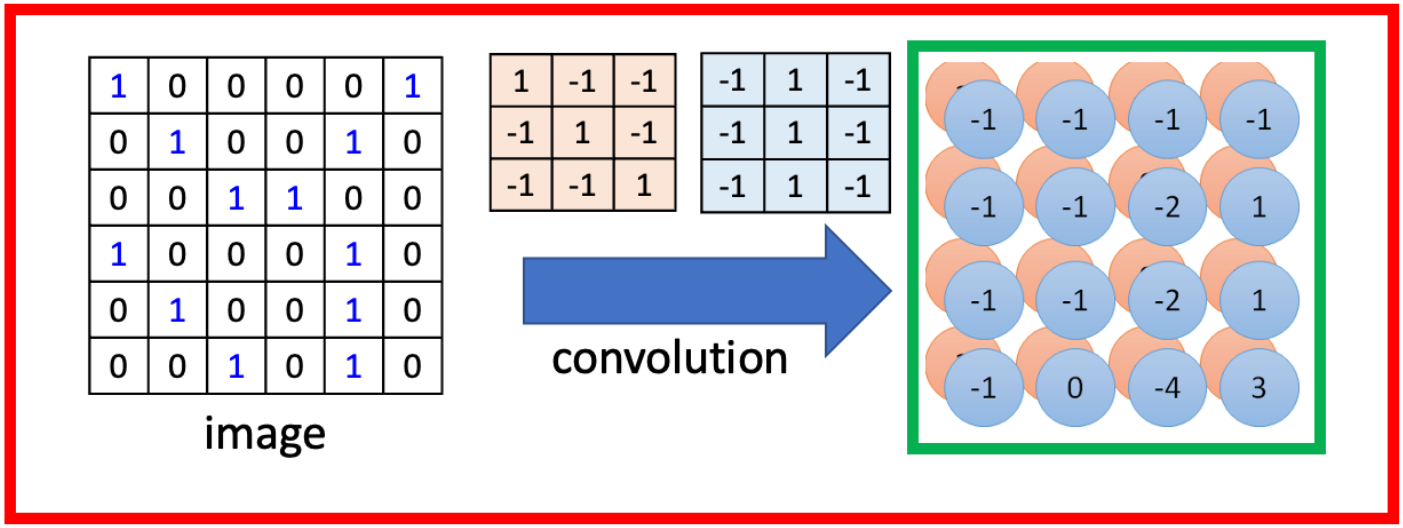
Filter 2

Colorful image

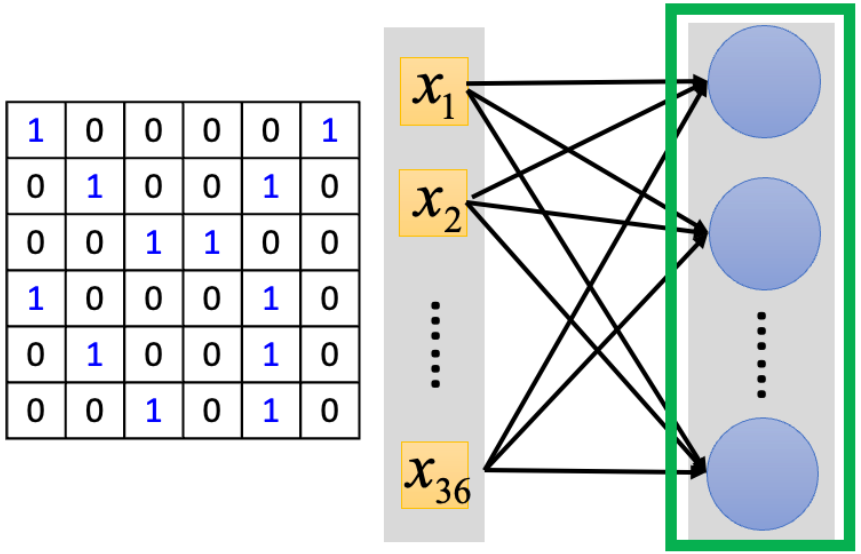


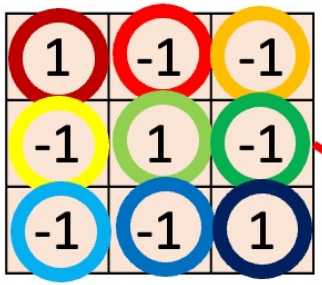


Convolution v.s. Fully Connected

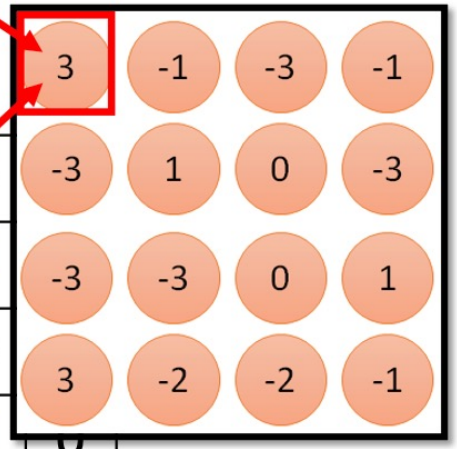


Fully-connected





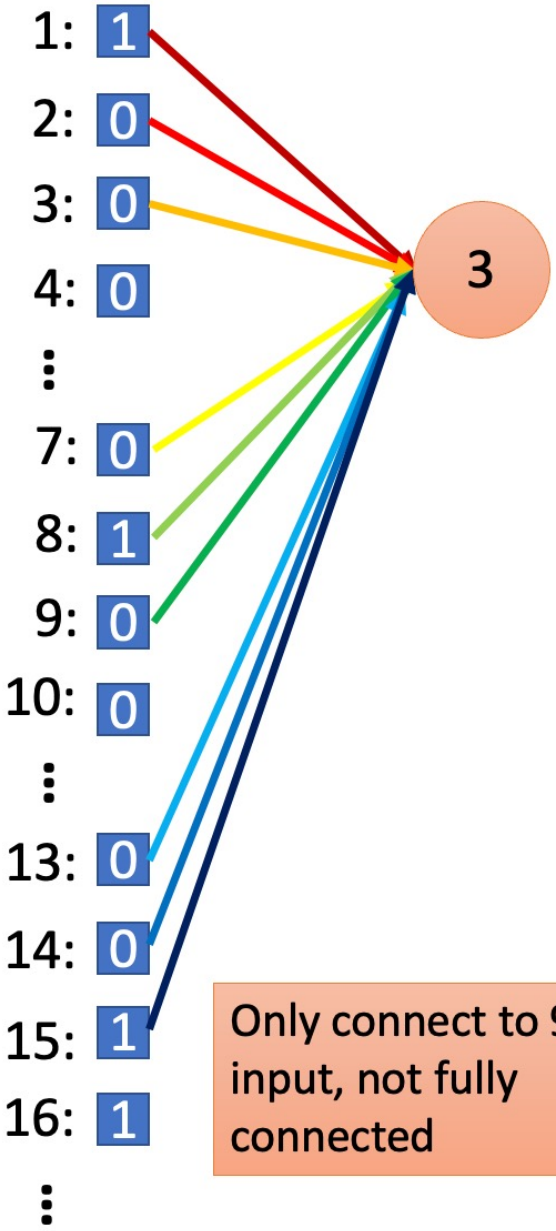
Filter 1



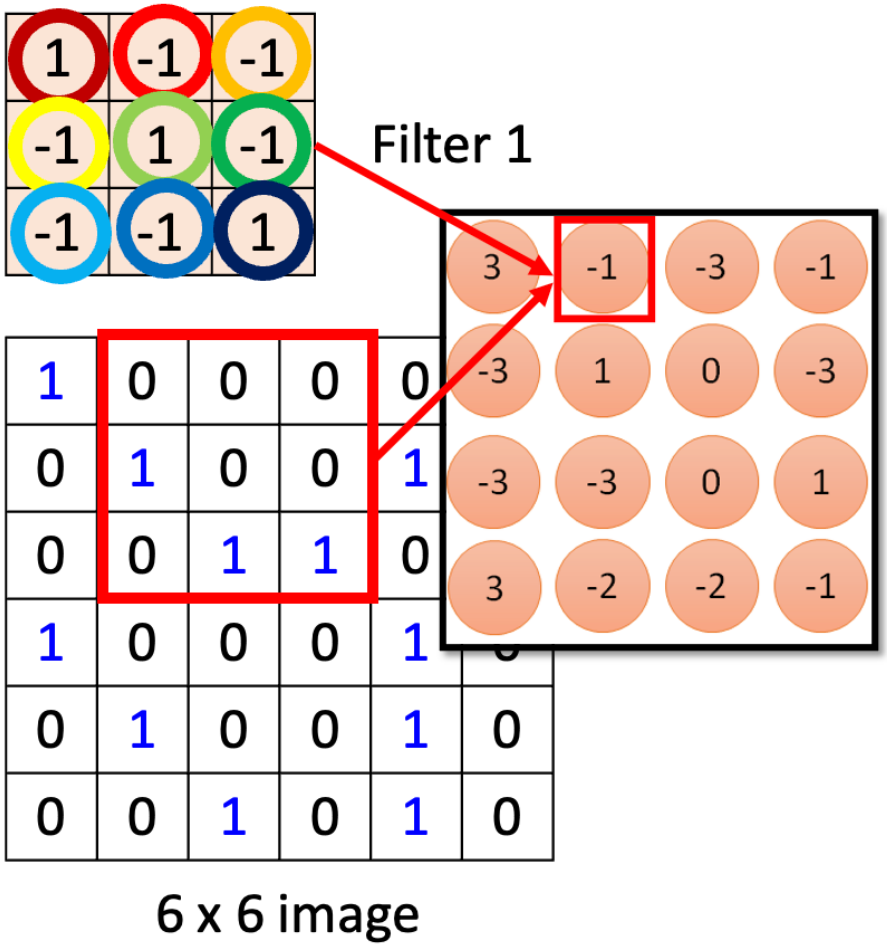
1	0	0	0	0	
0	1	0	0	1	
0	0	1	1	0	
1	0	0	0	1	
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Less parameters!

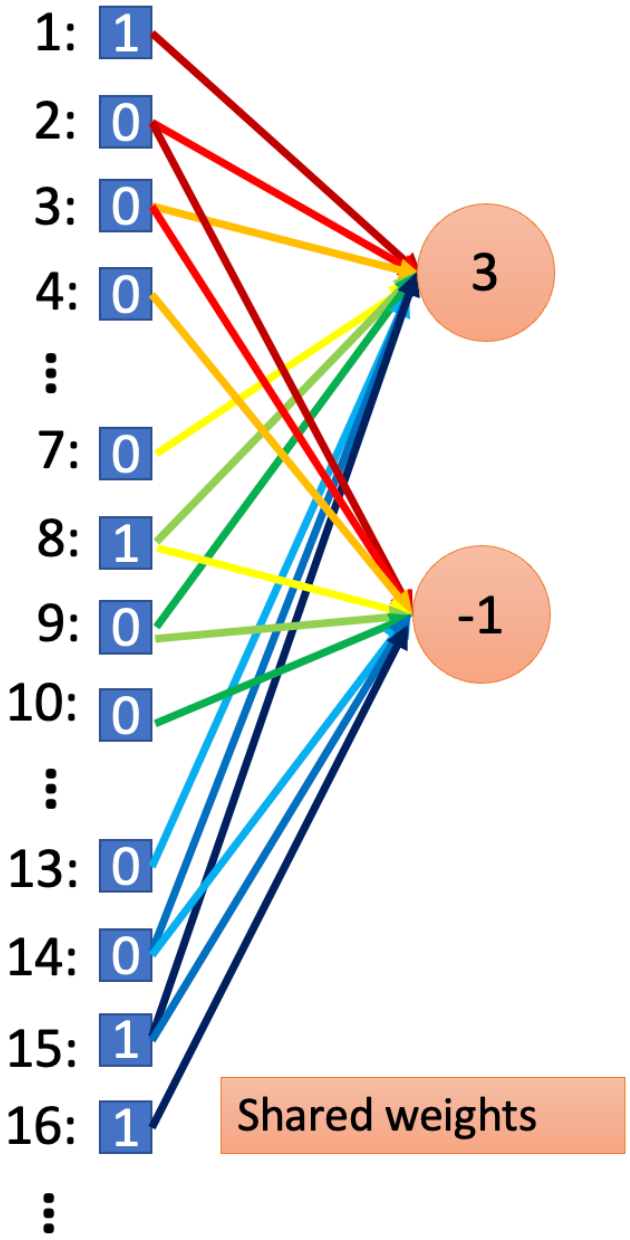


Only connect to 9 input, not fully connected



Less parameters!

Even less parameters!



The Whole CNN

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten

Fully Connected Feedforward network

cat dog ……

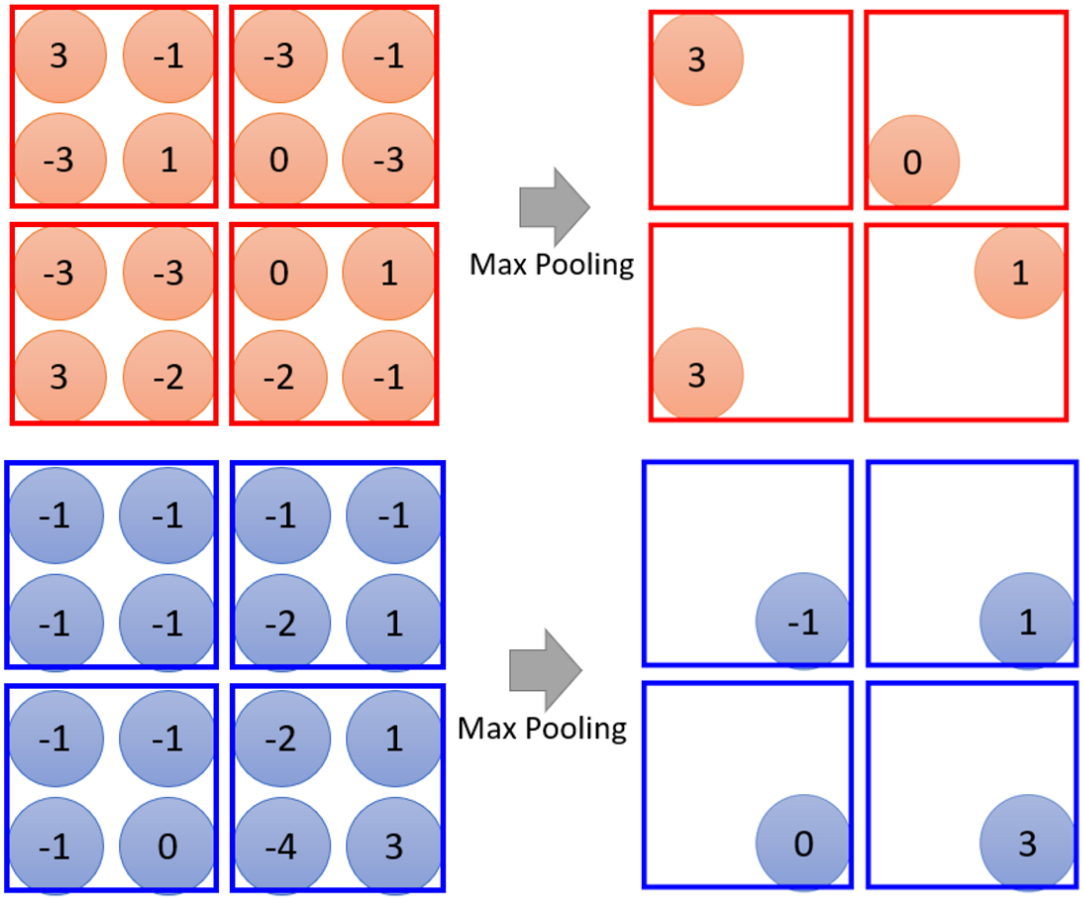
CNN – Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

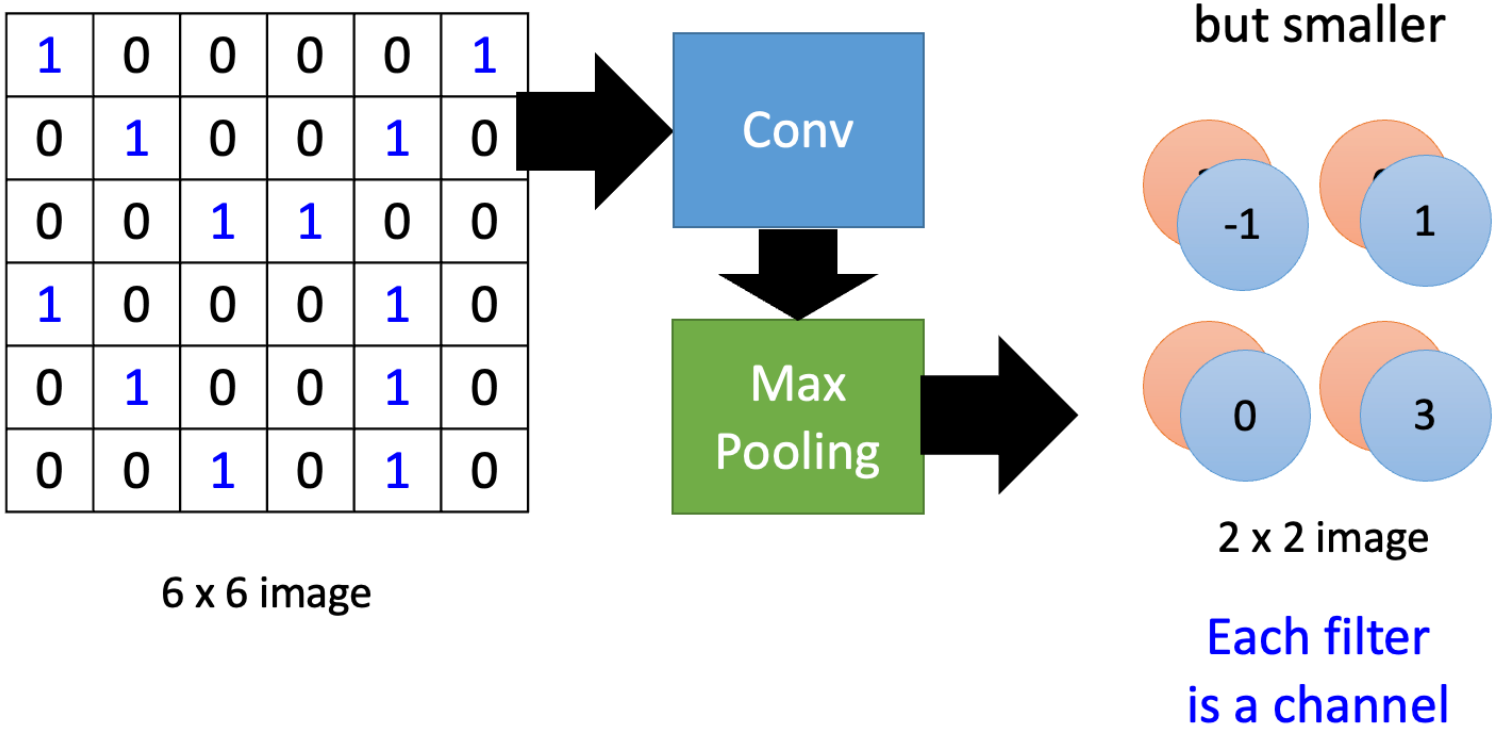
Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2



CNN – Max Pooling



The Whole CNN

Convolution

Max Pooling

Convolution

Max Pooling

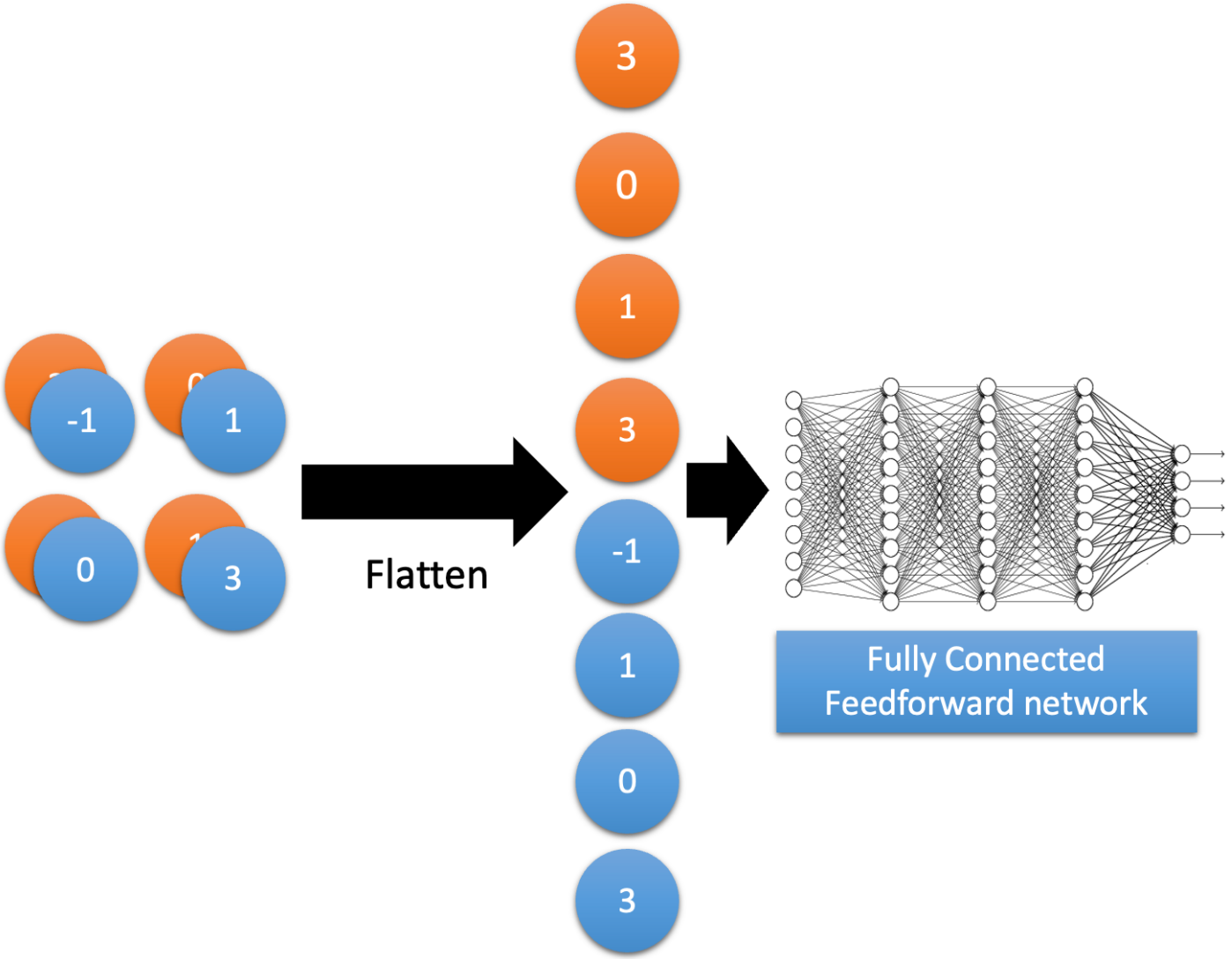
Can repeat many times

Flatten

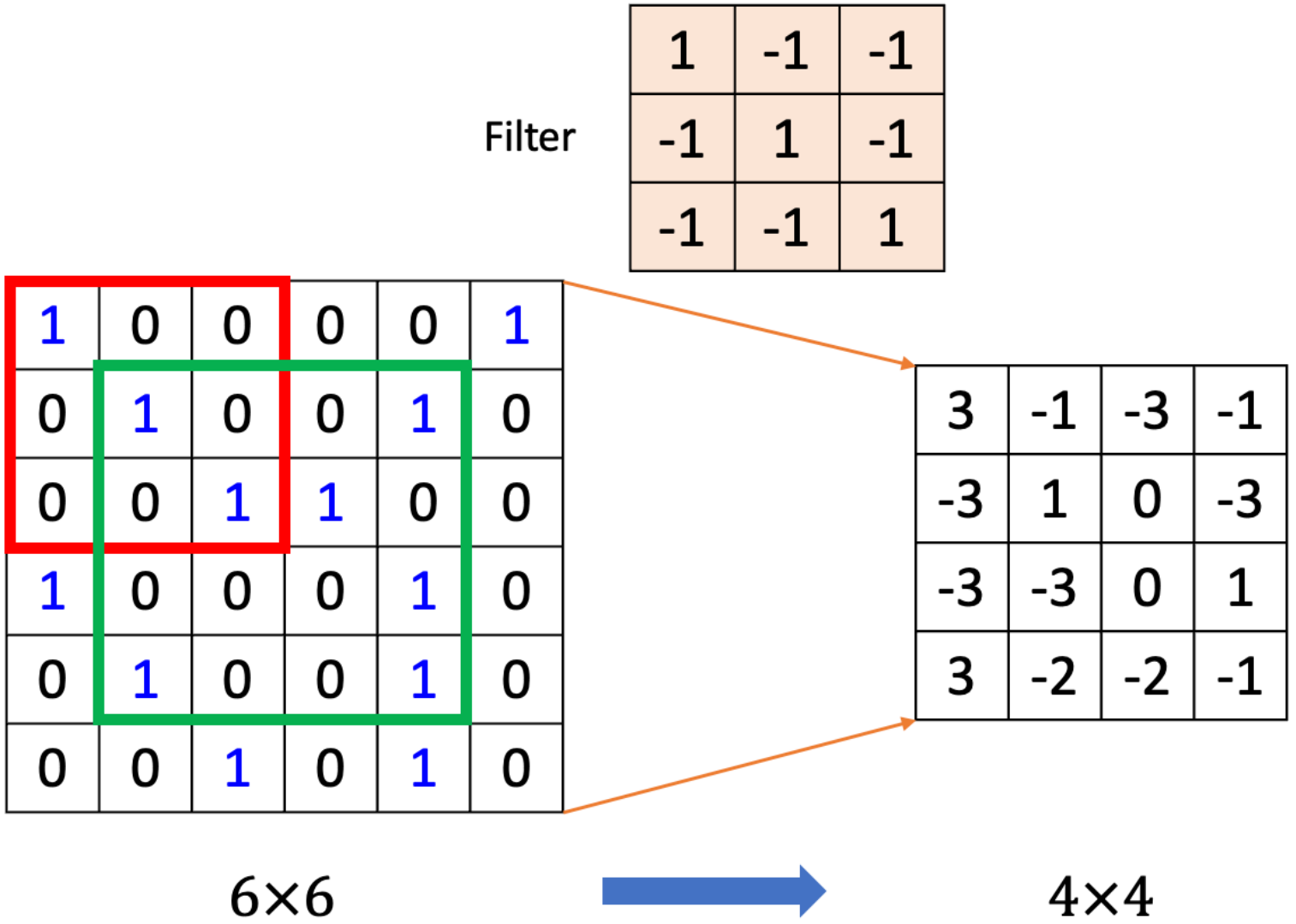
Fully Connected Feedforward network

cat dog ……

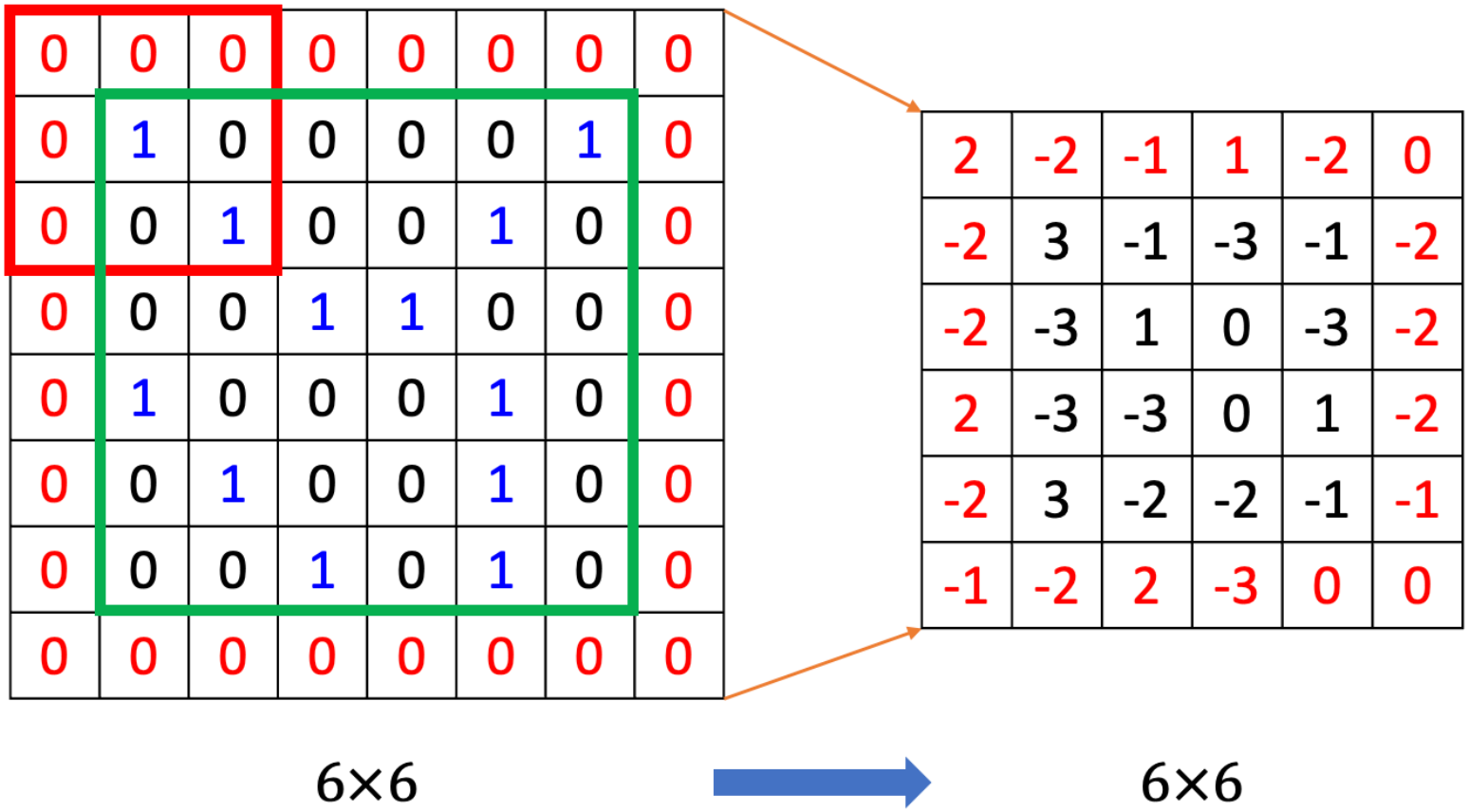
Flatten


-1
1
0
3
Flatten
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

Padding

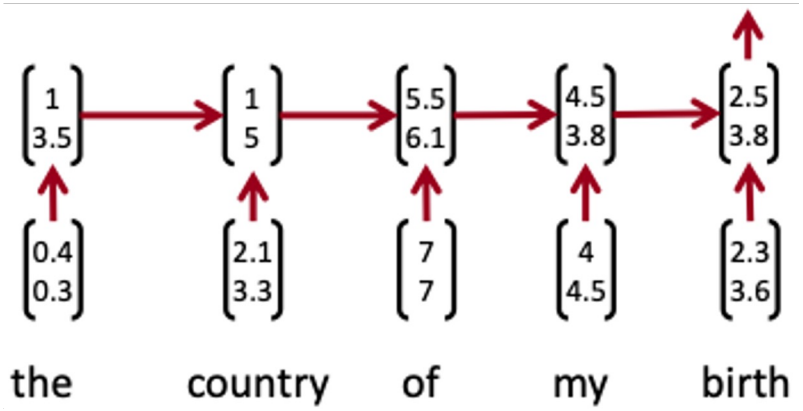

Filter
1 -1 -1
-1 1 -1
-1 -1 1
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
6×6
4×4

Padding


0 0 0 0 0 0 0 0
0 1 0 0 0 0 1 0
0 0 1 0 0 1 0 0
0 0 0 1 1 0 0 0
0 1 0 0 0 1 0 0
0 0 1 0 0 1 0 0
0 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0
2 -2 -1 1 -2 0
-2 3 -1 -3 -1 -2
-2 -3 1 0 -3 -2
2 -3 -3 0 1 -2
-2 3 -2 -2 -1 -1
-1 -2 2 -3 0 0
6×6
6×6

From RNNs to CNNs

- Recurrent neural nets cannot capture phrases without prefix context
- Often capture too much of last words in final vector



- E.g. softmax is often only calculated at the last step

From RNNs to CNNs

- Main CNN idea:
  - What if we compute vectors for every possible word subsequences 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 phrase is grammatical
- Not very linguistically or cognitively plausible
- Then group them afterwards

A 1D convolution for text

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
government	0.1	0.2	−0.1	−0.1
open	−0.4	−0.4	0.2	0.3

t,d,r	−1.0
d,r,t	−0.5
r,t,k	−3.6
t,k,g	−0.2
k,g,o	0.3

Apply a filter (kernel) of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

An animation example: https://cezannec.github.io/CNN_Text_Classification/

A 1D convolution for text with padding

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
government	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
t,d,r	−1.0
d,r,t	−0.5
r,t,k	−3.6
t,k,g	−0.2
k,g,o	0.3
g,o,Ø	−0.5

Apply a filter (kernel) of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

3 channel 1D convolution with padding

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
government	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

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 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
government	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 ave 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
government	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

Convolution with stride = 2

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
government	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
d,r,t	−0.5	−0.1	0.8
t,k,g	−0.2	0.1	1.2
g,o,Ø	−0.5	−0.9	0.1

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

Single layer CNN for sentence classification

- Yoon Kim (2014): Convolutional Neural Networks for Sentence Classification. EMNLP 2014.
  - Paper: https://arxiv.org/pdf/1408.5882.pdf
  - Code: https://github.com/yoonkim/CNN_sentence [Theano!]

- Goal: Sentence classification:
  - Mainly positive or negative sentiment of a sentence
  - Other tasks like:
    - Subjective or objective language sentence
    - Question classification: about person, location, number,...

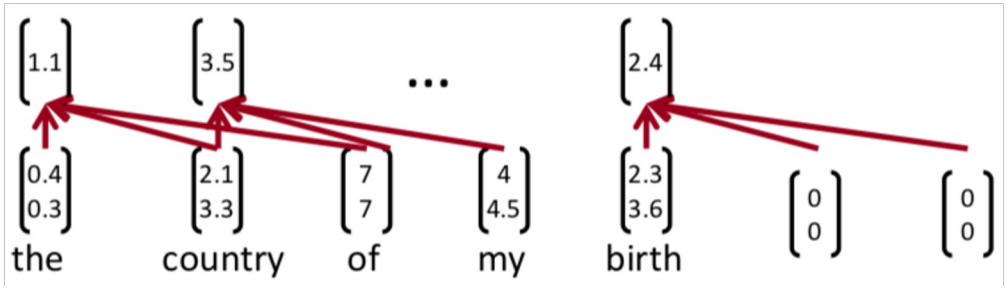
Single layer CNN

- Filter w is applied to all possible windows (concatenated vectors)
- To compute feature (one channel) for CNN layer:

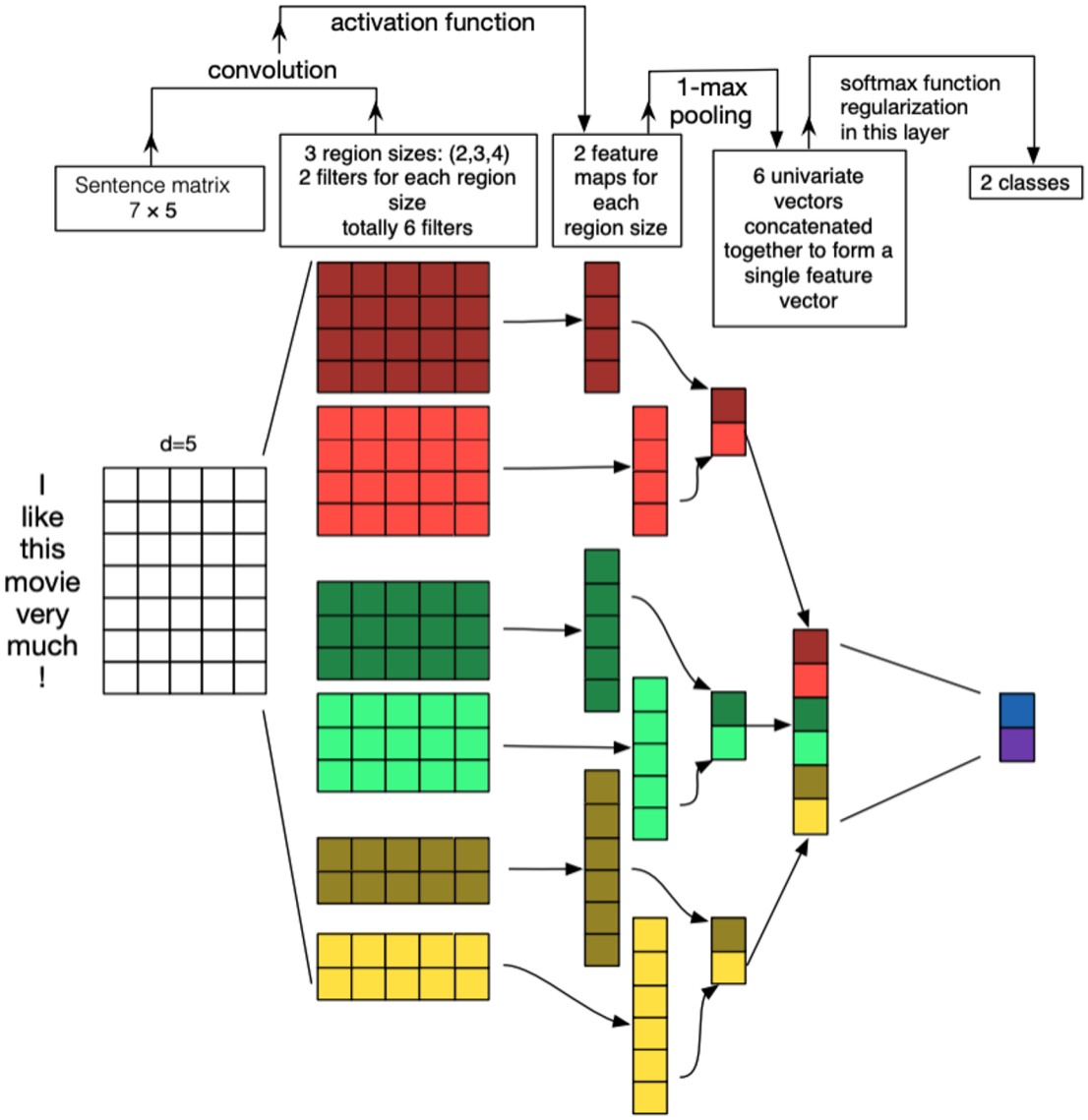
$$c_i = f(\mathbf{w}^\top \mathbf{x}_{i:i+h-1} + b)$$

- Sentence $x_{1:n} = x_1 \oplus x_2 \oplus ... \oplus x_n$
- All possible windows of length h: $\{x_{1:h}, x_{2:h+1}, ..., x_{n-h+1:n}\}$

- Result is a feature map: $c=[c_1, c_2, ..., c_{n-h+1}] \in \mathbb{R}^{n-h+1}$

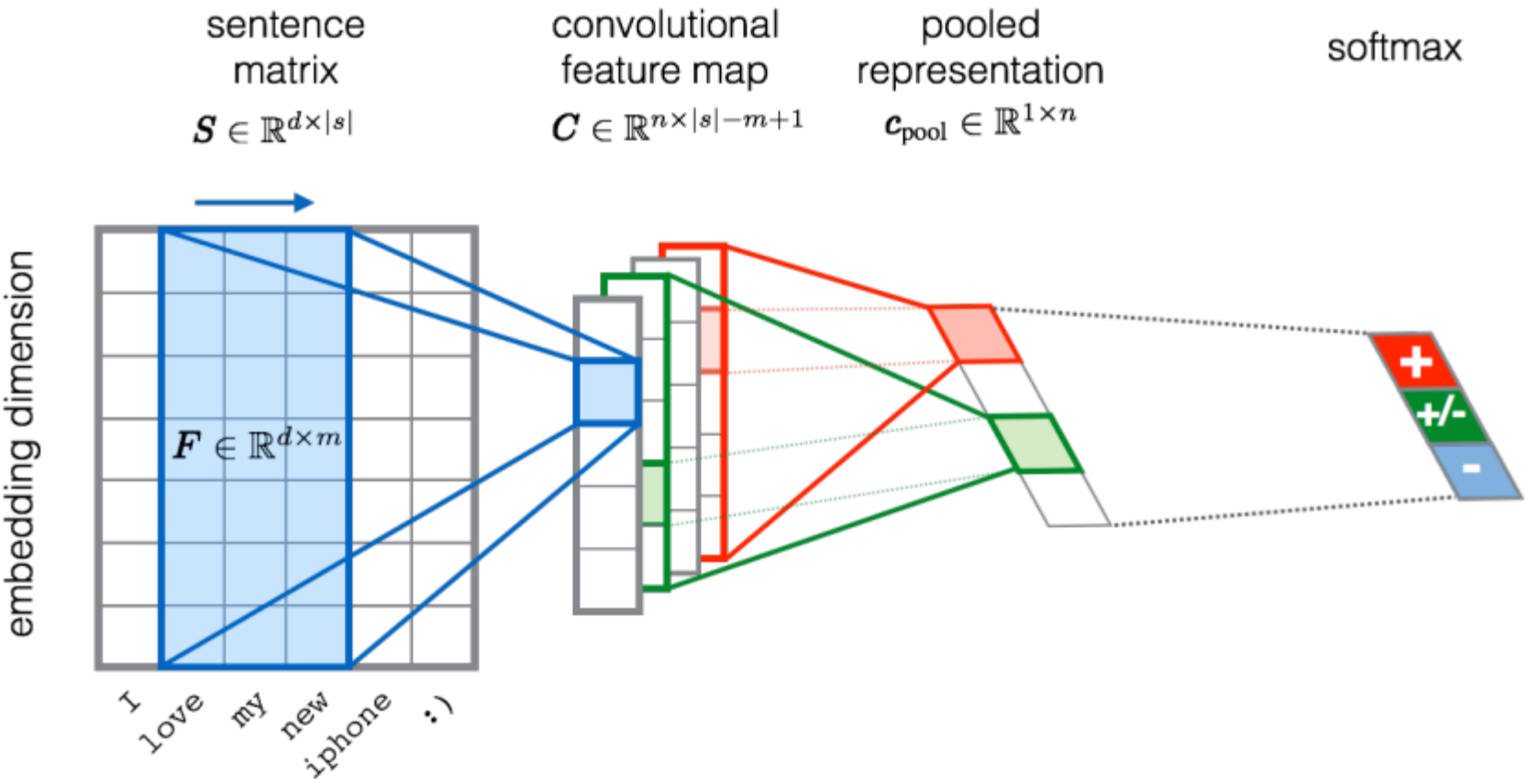


CNN for text classification



Zhang and Wallace (2015) A Sensitivity Analysis of (and Practitioners' Guide to) Convolutio Neural Networks for Sentence Classification
https://arxiv.org/pdf/1510.03820.pdf
(follow on paper, not famous, but a nice picture)

CNN for text classification



Severyn, Aliaksei, and Alessandro Moschitti. "Twitter sentiment analysis with deep convolutional neural networks." *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2015.

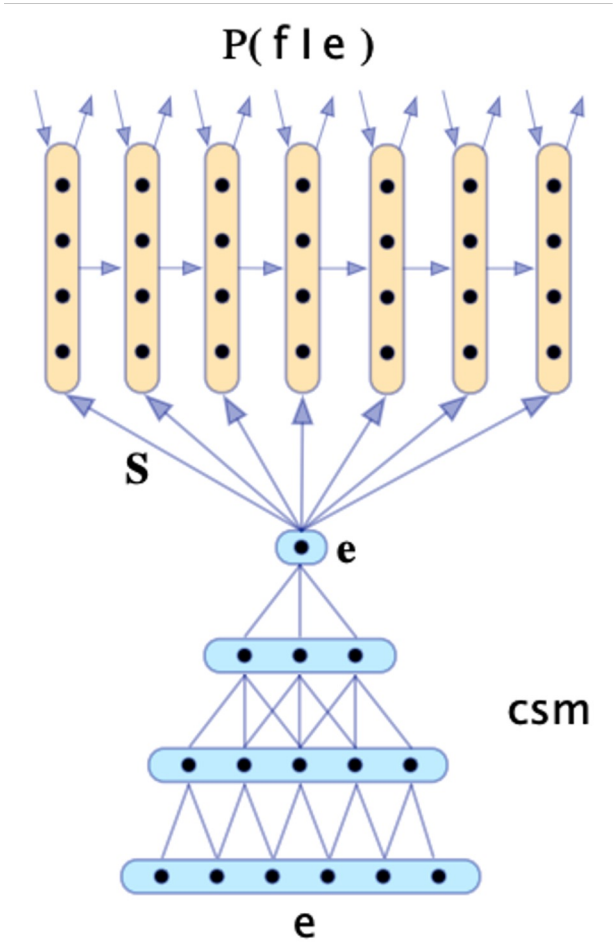
Batch Normalization (BatchNorm)

- Often used in CNNs
- Transform the convolution output of a batch by scaling the activations to have zero mean and unit variance
  - This is the familiar Z-transform of statistics
  - But updated per batch so fluctuation don't affect things much
- Use of BatchNorm makes models much less sensitive to parameter initialization, since outputs are automatically rescaled
  - It also tends to make tuning of learning rates simpler
- PyTorch: nn.BatchNorm1d

Ioffe and Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. http://proceedings.mlr.press/v37/ioffe15.pdf

CNN application: Translation

- One of the first successful neural machine translation efforts
- Uses CNN for encoding and RNN for decoding



Kalchbrenner and Blunsom (2013). "Recurrent Continuous Translation Models". https://www.aclweb.org/anthology/D13-1176

From Text to Tokens

Why do we need tokenization?

- Tokenization is the first step in any NLP pipeline. It has an important effect on the rest of your pipeline.
- A tokenizer breaks unstructured data and natural language text into chunks of information that can be considered as discrete elements.
  - Can be used directly as a vector representing that text.
  - Can also be used directly by a computer to trigger useful actions and responses.
  - Can be used in a machine learning pipeline as features.

https://neptune.ai/blog/tokenization-in-nlp

From Text to Tokens

- Many NLP models assume the input text has been tokenized and encoded as numerical vectors.
- There are several tokenization strategies we can adopt:
  - **Character Tokenization**: The simplest tokenization scheme is to feed each character individually to the model.
  - **Word Tokenization**: split into words and map each word to an integer.
  - **Subword Tokenization**: combine the best aspects of character and word tokenization.

Character tokenization

- Character-level tokenization:

```
text = "Tokenizing text is a core task of NLP."
tokenized_text = list(text)
print(tokenized_text)
```

```
['T', 'o', 'k', 'e', 'n', 'i', 'z', 'i', 'n', 'g', ' ', 't', 'e', 'x', 't', ' ',
'i', 's', ' ', 'a', ' ', 'c', 'o', 'r', 'e', ' ', 't', 'a', 's', 'k', ' ', 'o',
'f', ' ', 'N', 'L', 'P', '.']
```

- Ignores any structure in the text and treats the whole string as a stream of characters.
- Helps deal with misspellings and rare words,
- But the main drawback is that linguistic structures such as words need to be learned from the data. This requires significant compute, memory, and data.

- For this reason, character tokenization is rarely used in practice.
  - Need to preserve some structure of the text during tokenization.

Word Tokenization

- Split the input text into words and map each word to an integer.
- One simple class of word tokenizers uses whitespace to tokenize the text.

```
tokenized_text = text.split()
print(tokenized_text)
```

```
['Tokenizing', 'text', 'is', 'a', 'core', 'task', 'of', 'NLP.']
```

- Reduces the complexity of the training process.
- Punctuation is not accounted for, so 'NLP.' is treated as a single token.
- Given that words can include declinations, conjugations, or misspellings, the size of the vocabulary can easily grow into the millions!

Potential problem of a large vocabulary

- Requires neural networks to have an enormous number of parameters; expensive to train, and larger models are more difficult to maintain.
- A common approach is to limit the vocabulary and discard rare words by considering, say, the 100,000 most common words in the corpus.
  - Words that are not part of the vocabulary are classified as "unknown" and mapped to a shared UNK token.
  - We lose some potentially important, since the model has no information about words associated with UNK.
- Is there a compromise between character and word tokenization that preserved all the input information and some of the input structure?

Subword tokenization

- Basic idea: combine the best aspects of character and word tokenization
  - We want to split rare words into smaller units to allow the model to deal with complex words and misspellings.
  - We want to keep frequent words as unique entities so that we can keep the length of our inputs to a manageable size.
- Learned from the pre-training corpus using a mix of statistical rules and algorithms.
- The word pieces may be full words or parts of words.

Subword tokenization

- One of the main tools they utilize is breaking off affixes.
  - Because prefixes, suffixes, and infixes change the inherent meaning of words, they can also help programs understand a word's function.
  - This can be especially valuable for out of vocabulary words, as identifying an affix can give a program additional insight into how unknown words function.
- The sub word model will search for these sub words and break down words that include them into distinct parts.

"What is the tallest building"

→'what' 'is' 'the' 'tall' 'est' 'build' 'ing'

A blog about NLP tokenization

How does subword tokenization help the issue of OOV words

- A more complicated word, like '**machinating**' (the present tense of verb 'machinate' which means to scheme or engage in plots).
- It's unlikely that machinating is a word included in many basic vocabs.
  - Word tokenization: an unknown token.
  - Subword tokenization: an 'unknown' token and an 'ing' token.
- From there it can make valuable inferences about how the word functions in the sentence.
  - Either functioning as a verb turned into a noun, or as a present verb.
  - This dramatically narrows down how the unknown word, 'machinating,' may be used in a sentence.

A blog about NLP tokenization

Subword modeling

- Subword modeling in NLP encompasses a wide range of methods for reasoning about structure below the word level. (Parts of words, characters, bytes.)

- The dominant modern paradigm is to learn a vocabulary of parts of words (subword tokens).

- At training and testing time, each word is split into a sequence of known subwords.

- Byte pair encoding (BPE) is a simple, effective strategy for defining a subword vocabulary.

Sennrich et al. (2016)

Byte Pair Encoding (BPE)

1. Start with a vocabulary containing only characters and an “end-of-word” symbol.

2. Using a corpus of text, find the most common adjacent characters “a, b”; add “ab” as a subword.

3. Replace instances of the character pair with the new subword; repeat until desired vocab size.

```
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

Sennrich et al. (2016)

Byte Pair Encoding (BPE)

- Doing 8k merges => vocabulary of around 8000 word pieces.
- Includes many whole words
- Most SOTA neural machine translation (NMT) systems use this on both source + target.
- Now a similar method (WordPiece) is used in pretrained models.

Sennrich et al. (2016)

Tokenization Today

- All pretrained models use some kind of subword tokenization with a tuned vocabulary; usually between 50k and 250k pieces (larger number of pieces for multilingual models).
- As a result, classical word embeddings like GloVe are not used. All subword representations are randomly initialized and learned in the Transformer models.
- When using pretrained models, it is really important to make sure that you use the same tokenizer that the model was trained with.
- From the model's perspective, switching the tokenizer is like shuffling the vocabulary.
- If everyone around you started swapping random words like "house" for "cat," you'd have a hard time understanding what was going on too!

Tokenization Today

- In traditional methods for building a co-occurrence matrix and applying techniques like Singular Value Decomposition (SVD) for dimensionality reduction, it's common to remove both punctuation and stopwords.

- However, the models that rely heavily on the context of each word in a sentence. This means that even seemingly insignificant words like stop words and punctuation can be important for understanding the full context of a sentence. Therefore, they are retained during tokenization.

- It's important to note that the specific preprocessing steps can vary depending on the task and the dataset. For instance, in certain cases, you might want to retain certain stopwords if they are relevant to the context of your analysis.

DistilBERT tokenizer in Hugging Face

```
from transformers import AutoTokenizer

model_ckpt = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
```

Automatically retrieve model configuration, pretrained weights, or vocab from the checkpoint

```
from transformers import DistilBertTokenizer

distilbert_tokenizer = DistilBertTokenizer.from_pretrained(model_ckpt)
```

Load the specific tokenizer manually

Text = ”Tokenizing text is a core task of NLP.”

```
encoded_text = tokenizer(text)
print(encoded_text)
```

How this tokenizer works

```
{'input_ids': [101, 19204, 6026, 3793, 2003, 1037, 4563, 4708, 1997, 17953,
2361, 1012, 102], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

```
tokens = tokenizer.convert_ids_to_tokens(encoded_text.input_ids)
print(tokens)
```

Convert them back into tokens

```
['[CLS]', 'token', '##izing', 'text', 'is', 'a', 'core', 'task', 'of', 'nl',
'##p', '.', '[SEP]']
```

Tokenizing the Whole Dataset

Special Token	[PAD]	[UNK]	[CLS]	[SEP]	[MASK]
Special Token ID	0	100	101	102	103

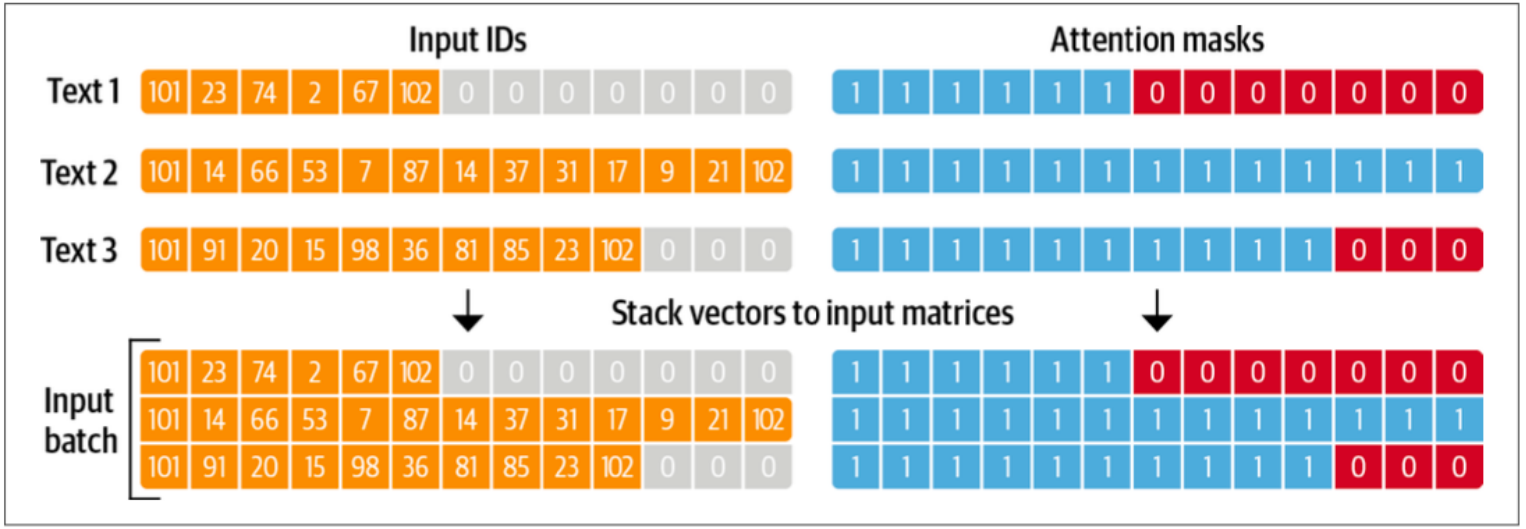


Figure 2-3. For each batch, the input sequences are padded to the maximum sequence length in the batch; the attention mask is used in the model to ignore the padded areas of the input tensors

All sequences have the same length, and you can efficiently process them in batches. The padding tokens are typically ignored by the network during computation.

Natural Language Processing with Transformers (NLPT), Chapter 2

Popular tokenization tools and libraries

- **NLTK (Natural Language Toolkit):**
  - A comprehensive library for natural language processing in python.
  - Includes various tokenization methods, including word tokenizers and sentence tokenizers.

- **Spacy**:
  - A popular library for natural language processing that provides high-performance tokenization along with many other NLP functionalities.

- **Tokenizer (from Hugging Face's Transformers):**
  - Part of the Hugging Face Transformers library, this tokenizer is specifically designed for use with modern transformer-based models like BERT, GPT, etc.

Readings

- **CH9 DL; CH 13 NNLP**
- **CH1-2 NLPT**
- LeCun et al. Gradient-based learning applied to document recognition. 1998
- Sennrich et al. Neural Machine Translation of Rare Words with Subword Units, 2016
- More details and implementation of BPE: https://huggingface.co/learn/nlp-course/chapter6/5?fw=pt
- More about WordPiece tokenization: https://huggingface.co/learn/nlp-course/chapter6/6?fw=pt

STEVENS

INSTITUTE *of* TECHNOLOGY

THE INNOVATION UNIVERSITY®

stevens.edu

Thank You