Convolutional Neural Networks

Predict the sentiment (positive, negative, or neutral) of sentences

- *Part of the charm of Mickey Mouse is that it avoids the obvious with humor and lightness.*
- *Still, this flick is fun and host to some truly excellent sequences.*

Order matters!

- CBOW/SkipGram ignores the ordering information completely and will give same sentiment to both the following sentences:

  it was not good, it was actually quite bad

  It was not bad, it was actually quite good

- The local ordering of the words (that the word "*not*" appears right before the word "*bad*") is very important

Solution 1: n-gram embedding

- Embed word-pairs (bi-grams) or word-triplets (trigrams) rather than words, and building a CBOW over the embedded n-grams

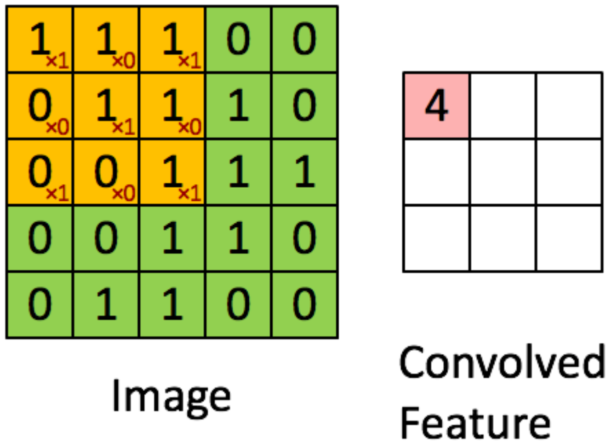
- Problems:
  - it will result huge embedding matrices
  - It will not scale for longer n-grams
  - It will suffer from data sparsity problems as it does not share statistical strength between different n-grams (“*quite good*” vs. “*very good*”)

Solution 2: CNN

BASIC CONVOLUTION + POOLING

- Apply a **nonlinear (learned) function** over each instantiation of a k-word sliding window over the sentence
  - Nonlinear **function/filter/kernel** transforms a window of *k* words into a scalar value

- Several such filters can be applied, resulting in *l* dimensional vector
- Focus on the most important “features” in the sentence, regardless of their location
  - each filter extracts a different indicator from the window, and the pooling operation zooms in on the important indicators



1D Convolutions over Text

- Lets discuss on whiteboard

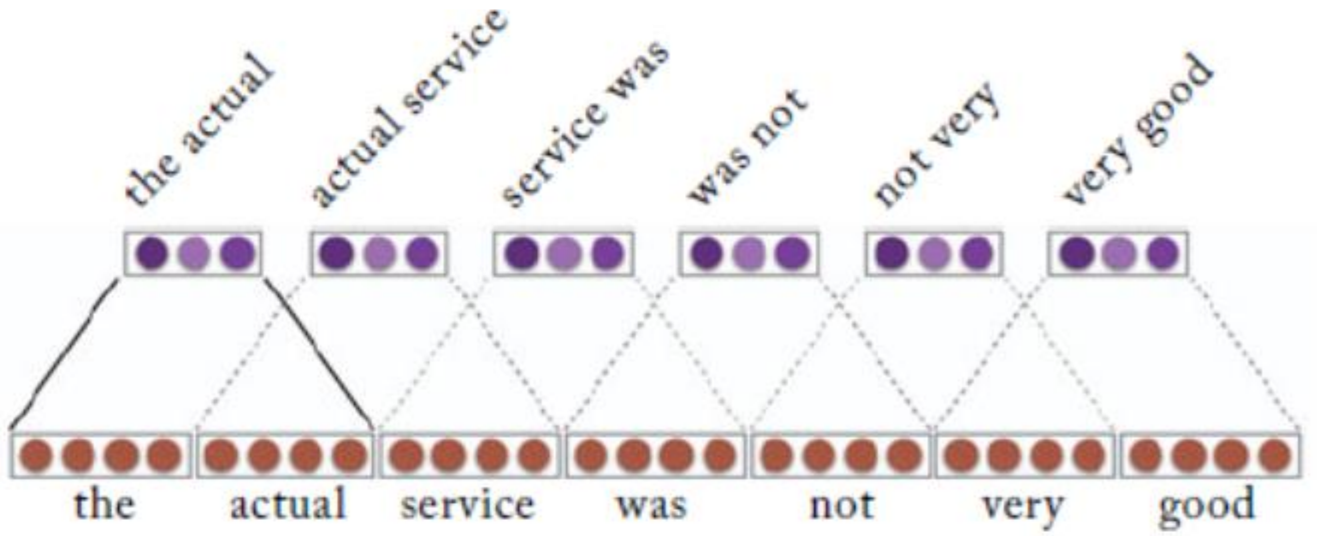
Narrow vs. Wide Convolutions

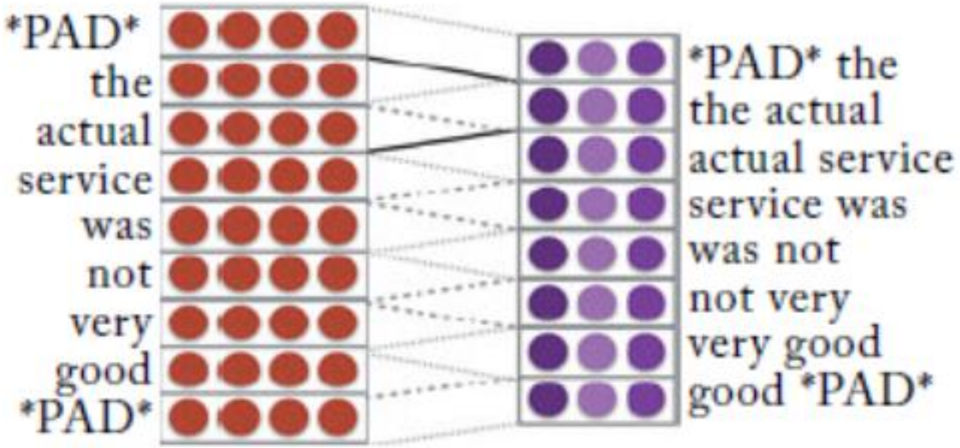
- How many vectors p_i do we have?

- For a sentence of length n with a window of size k, there are n-k+1 positions in which to start the sequence, and we get n-k+1 vectors $p_{1:n-k+1}$. This is called a *narrow convolution*.

- An alternative is to pad the sentence with k-1 padding-words to each side, resulting in n+k-1 vectors $p_{1:n+k-1}$. This is called a *wide convolution*

An Alternative Formulation of Convolutions





Vector Pooling

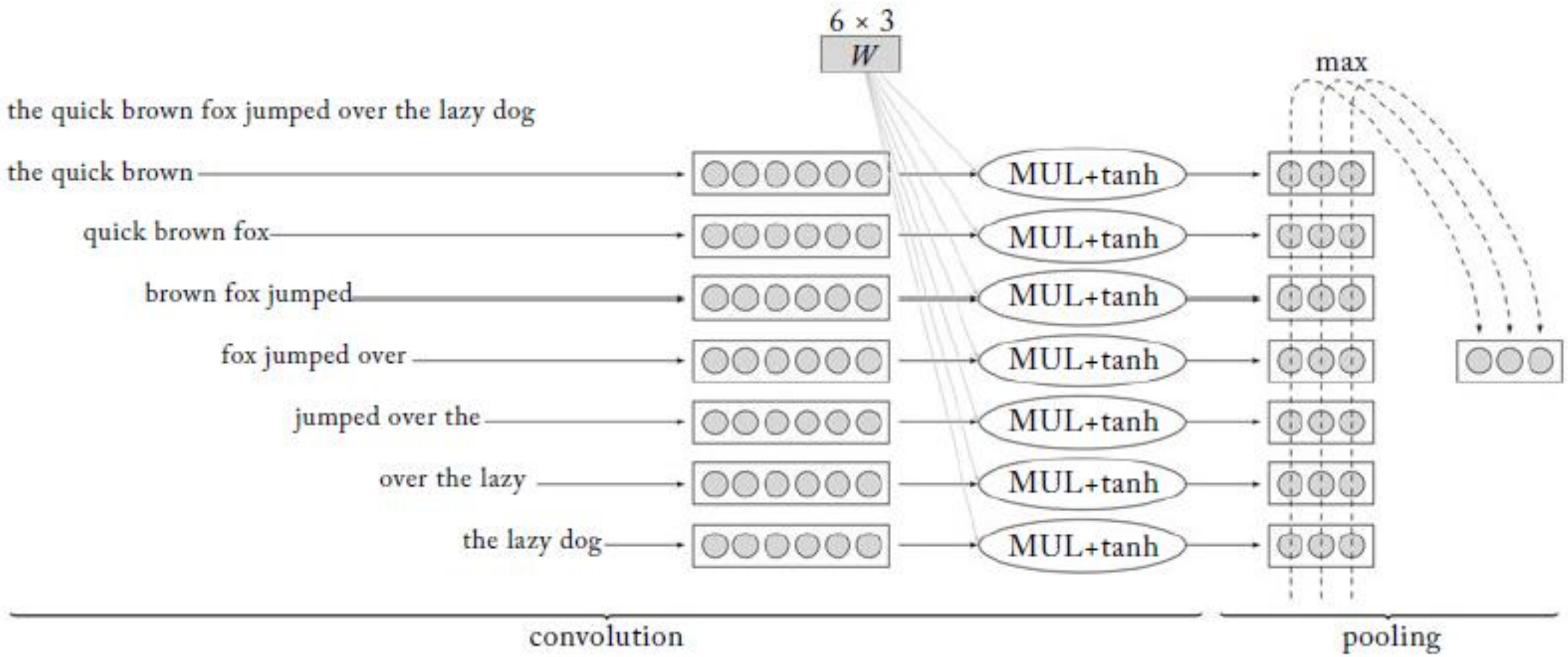
- Applying the convolution over the text results in m vectors $p_{1:m}$. These vectors are then combined (*pooled*) into a single vector *c*, representing the entire sequence.

- During training, the vector *c* is fed into downstream network layers (i.e., an MLP), culminating in an output layer which is used for prediction.

Vector Pooling

The most common pooling operation is *max pooling*, taking the maximum value across each dimension.

$$c_{[j]} = \max_{1<i\leq m} p_{i\,[j]} \qquad \forall j \in [1, \ell],$$



Vector Pooling

- **Average pooling:** $c = \frac{1}{m}\sum_{i=1}^{m} p_i \, .$

- **K-max pooling**: the top k values in each dimension are retained instead of only the best one, while preserving the order in which they appeared in the text

$$\begin{bmatrix} 1 & 2 & 3 \\ 9 & 6 & 5 \\ 2 & 3 & 1 \\ 7 & 8 & 1 \\ 3 & 4 & 1 \end{bmatrix} \xrightarrow{\text{2-max pooling}} \begin{bmatrix} 9 & 6 & 3 \\ 7 & 8 & 5 \end{bmatrix} \xrightarrow{\text{Concatenate}} \begin{bmatrix} 9 & 6 & 3 & 7 & 8 & 5 \end{bmatrix}$$

- **Dynamic Pooling:** Split the vectors into r distinct groups and apply pooling on each group separately

Variations

- We may have four different convolutional layers, each with a different window size in the range 2–5, capturing k-gram sequences of varying lengths

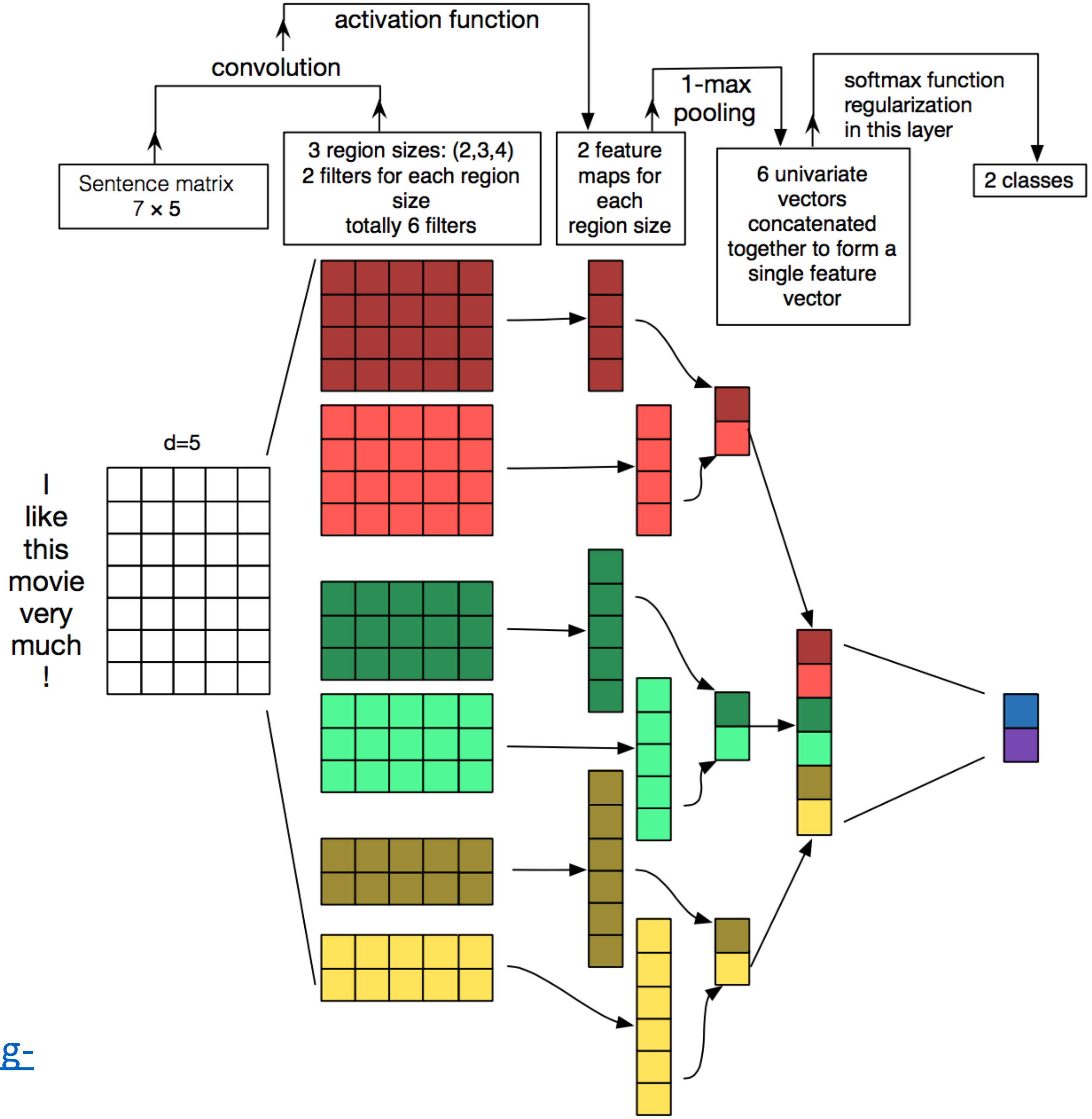
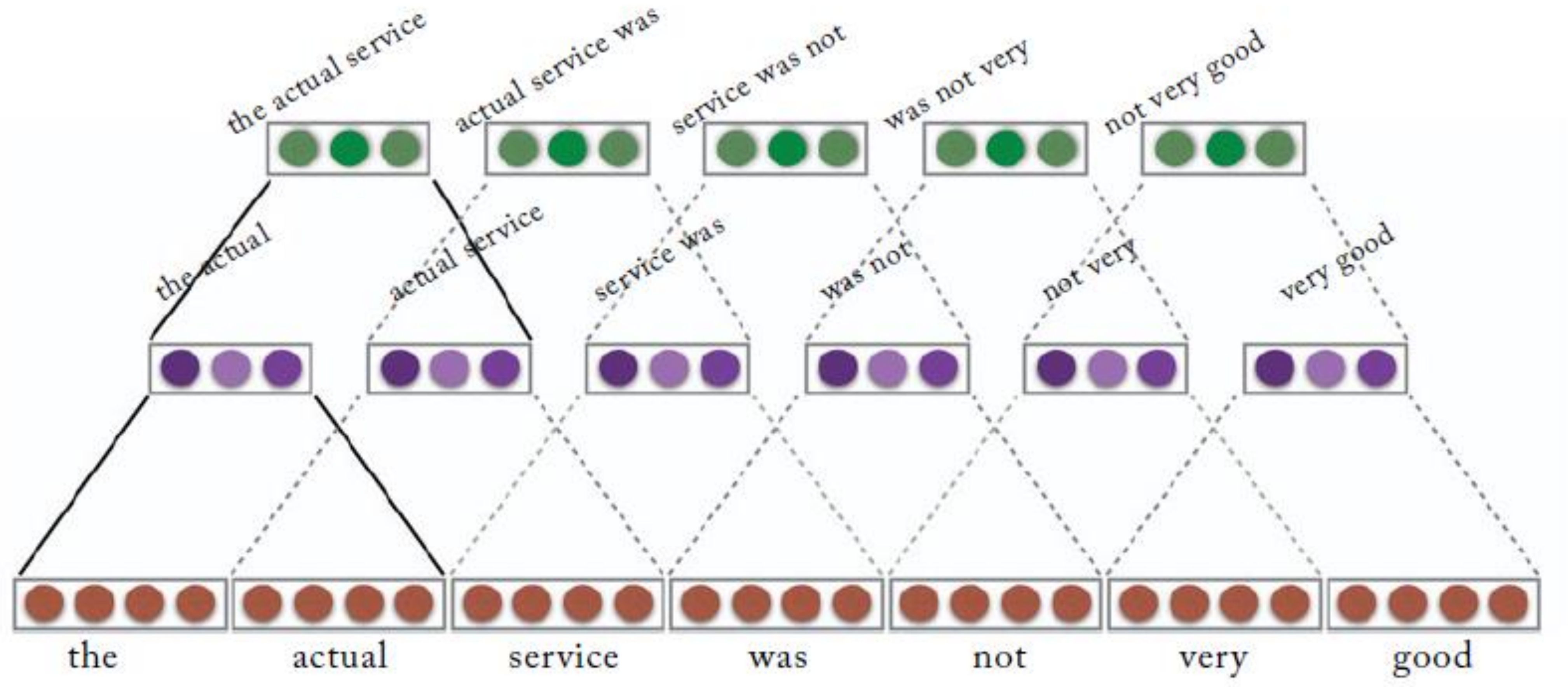


Image Reference
: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/

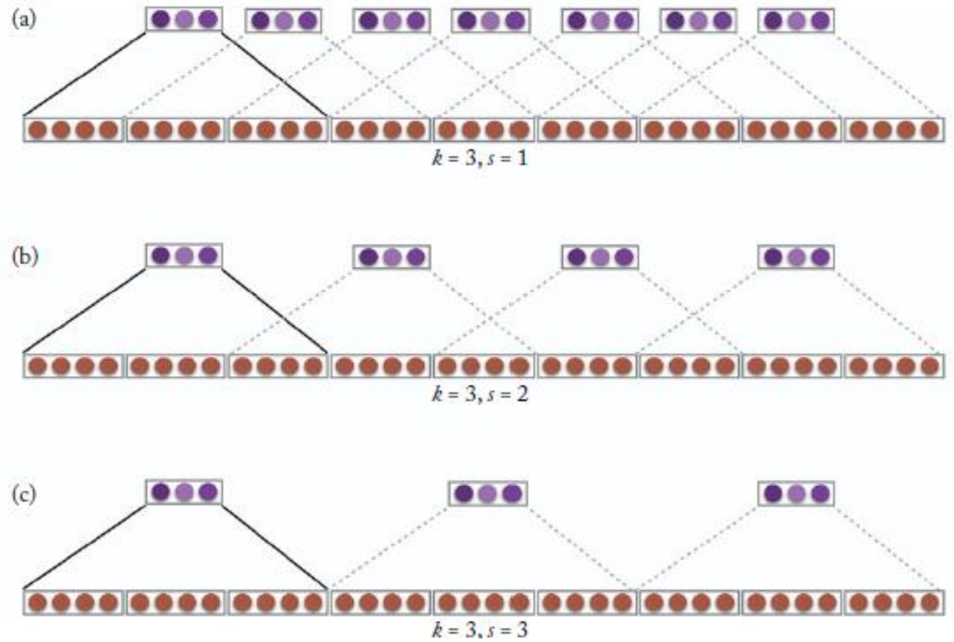
Hierarchical Convolutions



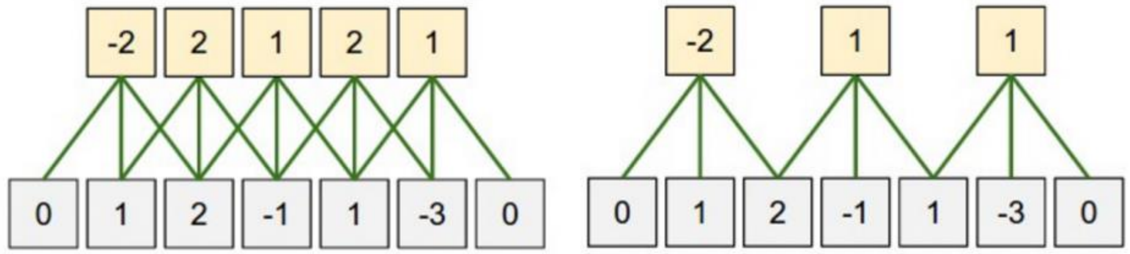
Strides

- how much you want to shift your filter at each step.

- A larger stride size leads to fewer applications of the filter and a smaller output size.

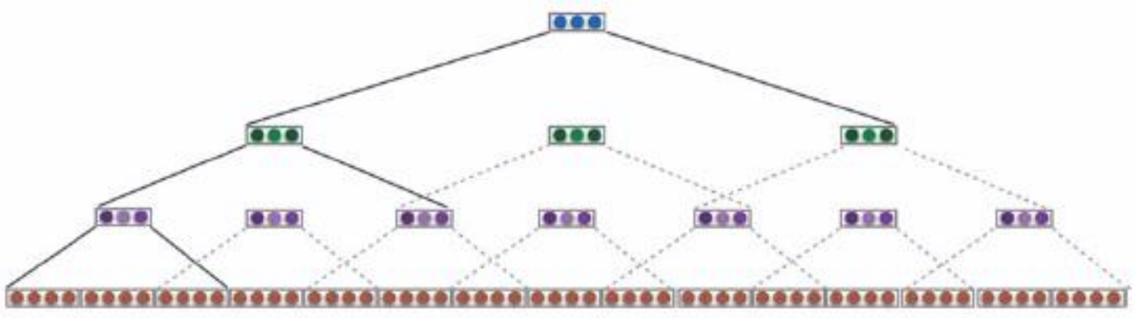


Strides. (a–c) Convolution layer with k=3 and stride sizes 1, 2, 3.



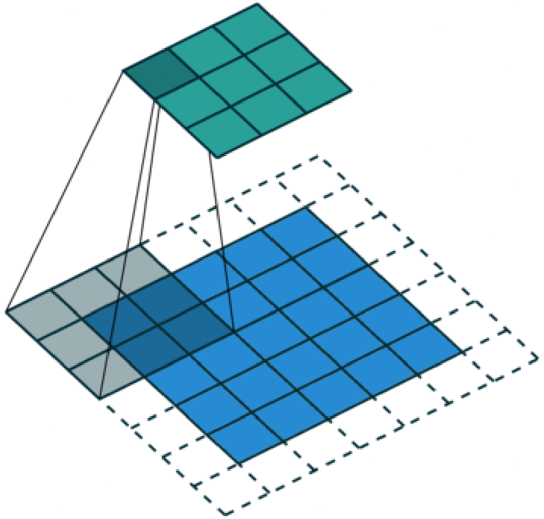
Convolution Stride Size. Left: Stride size 1. Right: Stride size 2.
Source: http://cs231n.github.io/convolutional-networks/

Dilation

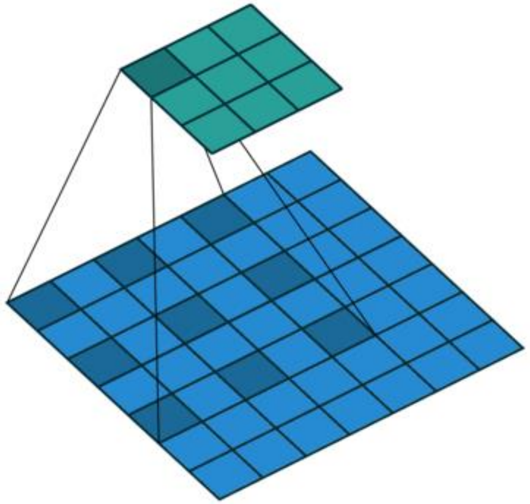


- In a dilated convolution architecture, the hierarchy of convolution layers each has a stride of k-1.

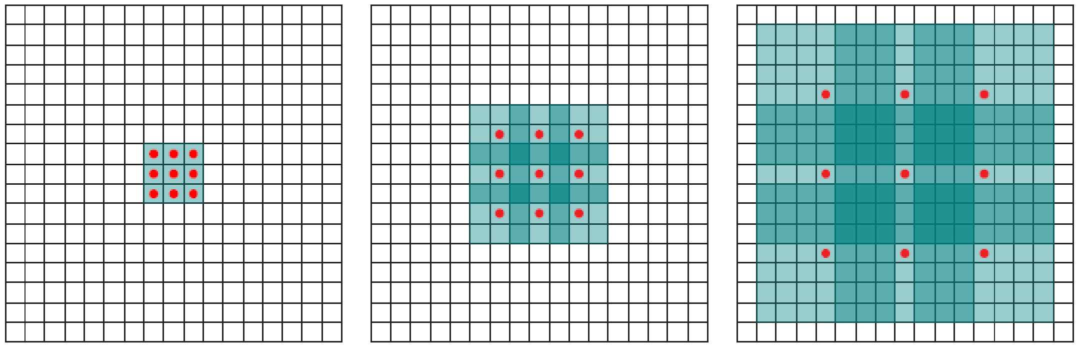
- This allows an exponential growth in the effective window size as a function of the number of layers.



Standard Convolution (l=1)



Dilated Convolution (l=2)



l=1 (left), l=2 (Middle), l=4 (Right)

https://towardsdatascience.com/review-dilated-convolution-semantic-segmentation-9d5a5bd768f5

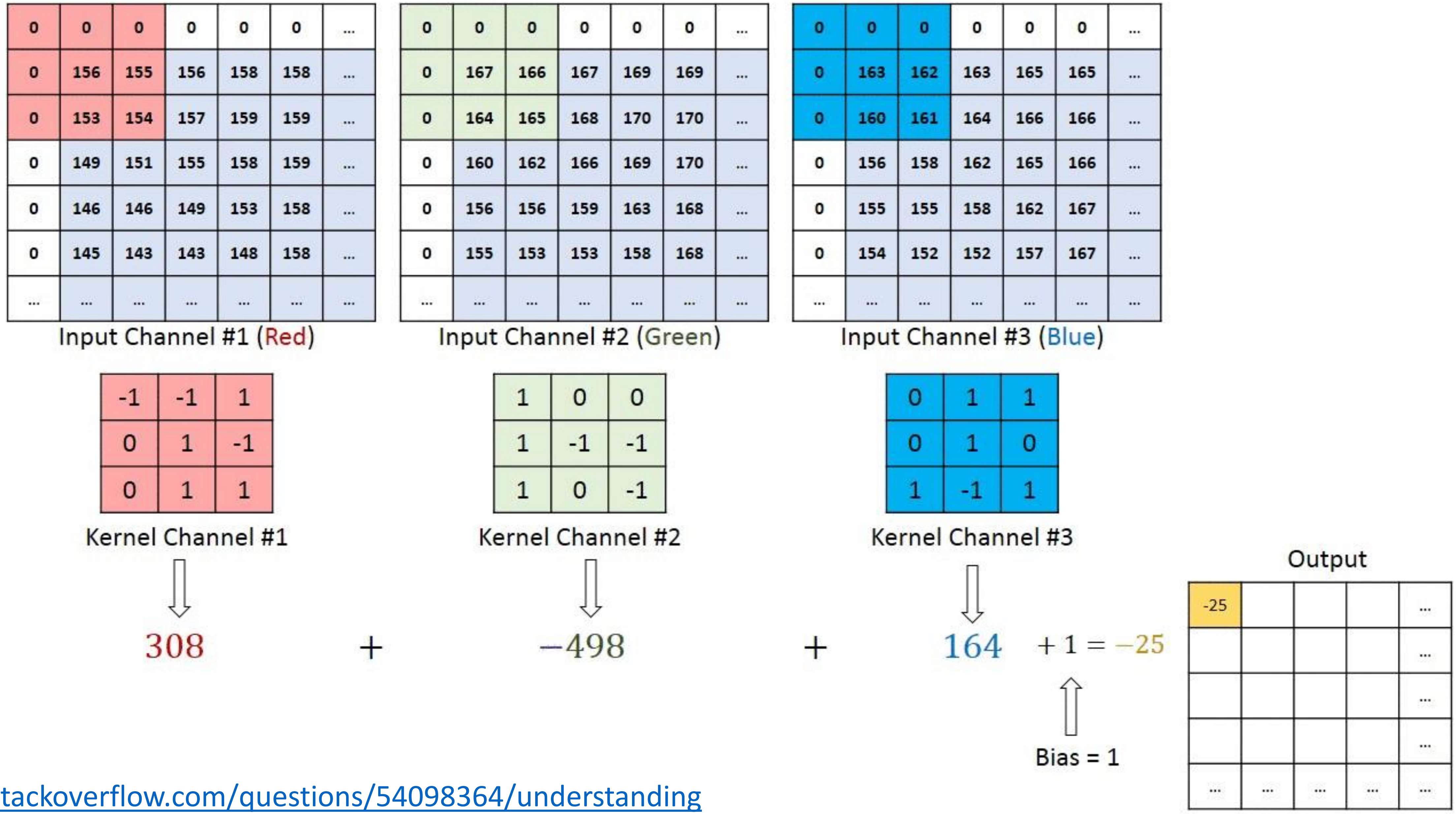
Channel

- Channels are different “views” of your input data.

- For example, in image recognition you typically have RGB (red, green, blue) channels. You can apply convolutions across channels, either with different or equal weights.

- In NLP you could imagine having various channels as well: You could have a separate channels for different word embeddings (e.g., word2vec and GloVe), or you could have a channel for the same sentence represented in different languages, or phrased in different ways.

http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/



https://stackoverflow.com/questions/54098364/understanding-channel-in-convolution-neural-network-cnn-input-shape-and-output

CNN for sentence classification (Kim, 2014)

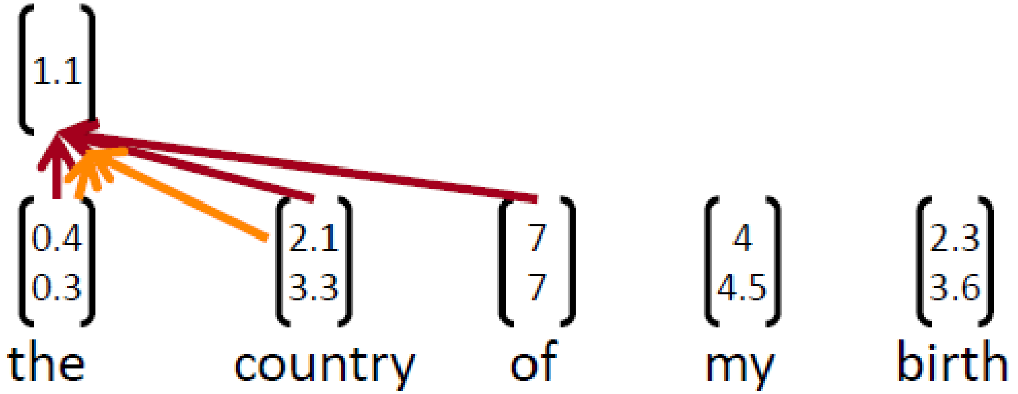
- Goal: Sentence classification:
  - Mainly positive or negative sentiment of a sentence

- Other tasks like:
  - Subjective or objective language sentence
  - Question classification: about person, location, number

EMNLP 2014. https://arxiv.org/pdf/1408.5882.pdf
Code: https://arxiv.org/pdf/1408.5882.pdf
Slide: http://web.stanford.edu/class/cs224n/

Single Layer CNN for Sentence Classification

- A simple use of one convolutional layer and **pooling**
- Word vectors: $\mathbf{x}_i \in \mathbb{R}^k$
- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus x_2 \oplus \cdots \oplus \mathbf{x}_n$ (vectors concatenated)
- Concatenation of words in range: $\mathbf{x}_{i:i+j}$ (symmetric more common)
- Convolutional filter: $\mathbf{w} \in \mathbb{R}^{hk}$ (over window of *h* words)
- Note, filter is a vector!
- Filter could be of size 2, 3, or 4:

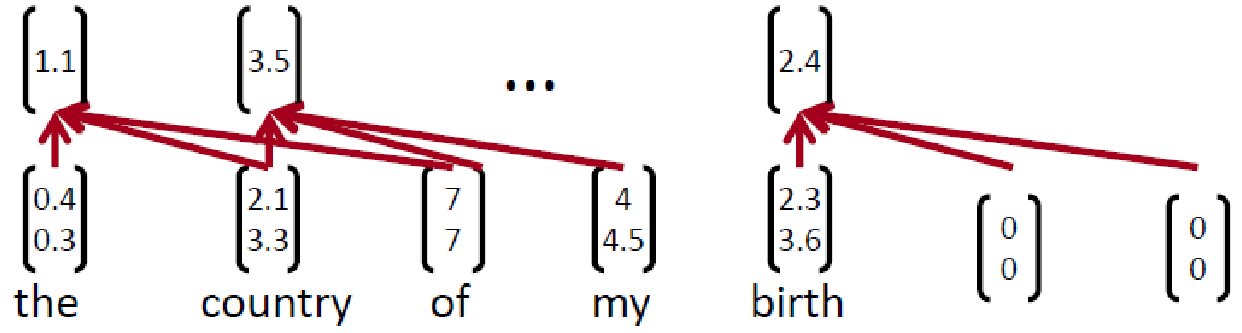


Single-Layer CNN

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

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

- Sentence: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$
- All possible windows of length *h*: $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \ldots, \mathbf{x}_{n-h+1:n}\}$
- Result is a feature map: $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



Pooling and Channels

- Pooling: max-over-time pooling layer
- Idea: capture most important activation (maximum over time)
- From feature map $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
- Pooled single number: $\hat{c} = \max\{\mathbf{c}\}$

- Use multiple filter weights **w**
- Useful to have different window sizes *h*
- Because of max pooling $\hat{c} = \max\{\mathbf{c}\}$, length of **c** irrelevant

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

- So we could have some filters that look at unigrams, bigrams, tri-grams, 4-grams, etc.

Multi-channel Input

- Initialize with pre-trained word vectors (word2vec or Glove)
- Start with two copies
- Backprop into only one set, keep other “static”
- Both channel sets are added to c_i before max-pooling

Classification after one CNN layer

- First one convolution, followed by one max-pooling
- To obtain final feature vector: $\mathbf{z} = [\hat{c}_1, \ldots, \hat{c}_m]$ (assuming *m* filters **w**)
  - Used 100 feature maps each of sizes 3, 4, 5
- Simple final softmax layer $y = softmax\left(W^{(S)}z + b\right)$