Lecture 5: Convolutional Neural Networks for NLP

Andrei Arsene Simion

Columbia University

February 15, 2023

Outline

- Convolutional Neural Networks
- CNN for NLP
- Example: Text Classification
- Example: Language Modeling with Gated CNN

Outline

- Convolutional Neural Networks
- CNN for NLP
- Example: Text Classification
- Example: Language Modeling with Gated CNN

Main Idea and goals of a CNN

- Given a sequence of text and a representation of each token in that text (one-hot, word vectors, etc.) how can we aggregate subsequences and get representations for each subsequence?
- Example: "I went to the Apple store to buy the cool new laptop."
  - How can we summarize the above by using batches of length 3?
  - There are 10 such contiguous groups of words for the above sentence
  - They are: {I went to}, {went to the}, {to the Apple} ... {cool new laptop}

Main Idea

- The phrases we make, we do not care if they are grammatical
- Or linguistically valid
- Or cognitively plausible
- What we care is trying to figure out if we can understand some sort of summary context from this information

Main Idea: Convolution

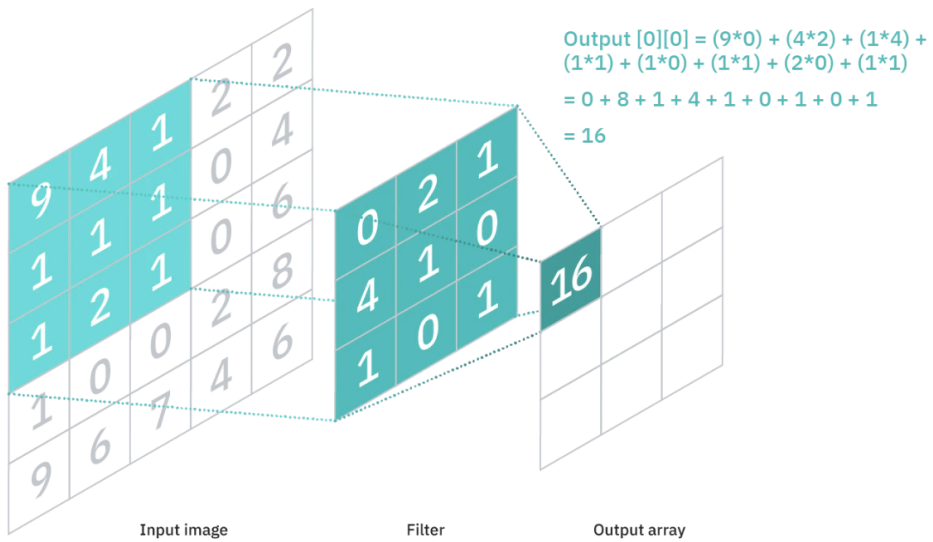
- CNN's were mainly used for Computer Vision, but we'll now see how they can be used for NLP
- In math, convolution is usually written as

$$(f * g)(n) = \sum_{m=-M}^{M} f[n-m]g[m]$$

- The idea above is to run some filter g with a function f and get a new function "mixing" f and g

Example of a Convolution for Images

- Images are usually stored as (data) boxes of a certain width, height, and depth (channel)
- Assume the depth is 1 (Black - White picture). For a rectangle of pixels, run a filter over the data over with a certain stride
- To note:
  - Data has size 5×5
  - Filter has size 3×3
  - We move the filter across each column 1 step at a time and when we come to the end columns we go 1 row down and repeat
  - Result has size 3×3

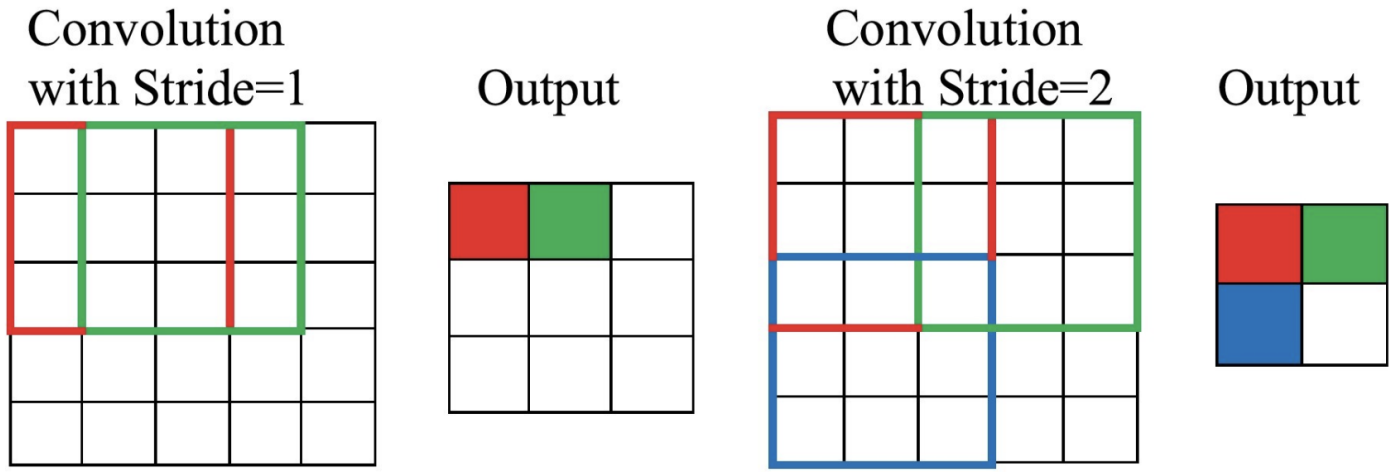


Effects of Convolution

- Note that the original data was shrunk in dimensions
- Note that we moved the filter across 1 column at a time
  - We could have moved this 2 (or rows) columns at a time, but, then we'd have a smaller result
  - The filter is applied to a sub-image of the exact same dimension as the filter (3×3), but we could have added "padding" with this padded image to get a result of the *same size as before*

Strides

- Example:
  - Data has size 5×5
  - Filter has size 3×3
  - Stride is 1 on the left, 2 on the right



Padding

- Example:
  - Data has size 5×5
  - Filter has size 3×3
  - Stride is 1
  - Added padding on the edges
    - Note that now a 3×3 kernel can convolve with a 2×2 sub image because this sub-image has been padded

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114	328	-26	470	158
53	266	-61	-30	344
403	116	-47	295	244
108	-135	256	-128	344
314	346	279	153	421

Outline

- Convolutional Neural Networks
- CNN for NLP
- Example: Text Classification
- Example: Language Modeling with Gated CNN

Example

- How can we represent text and do convolutions?
- Use word vectors!
  - Example: Assume each word has dimension 5
  - We can use GloVe, Word2Vec, etc to load these word vectors
    - Note: I assume here that the vectors are constants for demonstration

i	went	to	the	apple	store	to	buy	the	cool	new	laptop
0.1	0.2	0.31	0.41	0.45	0.5	0.7	0.8	0.9	0.91	0.92	0.93
0.1	0.2	0.31	0.41	0.45	0.5	0.7	0.8	0.9	0.91	0.92	0.93
0.1	0.2	0.31	0.41	0.45	0.5	0.7	0.8	0.9	0.91	0.92	0.93
0.1	0.2	0.31	0.41	0.45	0.5	0.7	0.8	0.9	0.91	0.92	0.93
0.1	0.2	0.31	0.41	0.45	0.5	0.7	0.8	0.9	0.91	0.92	0.93

- Apply a filter of dimension 3 but note that the filter's depth (i.e. channel depth) is 5, which is the dimension of the word vectors
- For NLP, these are know as 1-dimensional convolutions
- The all have the same number of rows - we can have 5×2, 5×1, 5×5 filters, etc

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

Example

- The result of this convolution will be a new vector with 1 channel and length 10
- Each group of 3 words is convolved with the filter so each column knows about 3 words in sequence

{i went to}	{went to the}	...	{cool new laptop}
-2.14		...	-0.2

- If we had another 5×3 filter filter we might have a result that looks like the above but with another row

{i went to}	{went to the}	...	{cool new laptop}
-2.14		...	-0.2
-0.23		...	0.9

- In PyTorch, this example would be done by applying
  - `m=Conv1d(5, 2, 3)`
  - `data=torch.rand(5, 12)`
  - `m(data)`
- Typically, a filter also has a bias - each $D \times k$ filter has $kD + 1$ parameters

Padding

- We can also apply padding to the above, so that the resulting sentence's width is the same as the start
- You can get `Conv1d(5, 2, 3, padding=1)` to get the above behavior

NULL	i	went	to	the	...	laptop	NULL
0	0.1					0.4	0
0	0.1					0.4	0
0	0.1					0.4	0
0	0.1					0.4	0
0	0.1					0.4	0

- If we have 2 kernels, the result now is 2×12 instead of 2×10

{NULL i went }	{i went to}	...	{new laptop NULL}
0.1	0.54	...	-0.21
-0.23	0.7	...	0.9

Dilation

- Question: What does `Conv1d(..., dilation=2)` do?
  - If you have words $\{a, b, c, d, e, f, g\}$ and a filter width two, dilation 1, stride 1
  - A standard filter gives features made up of $\{a, b\}, \{b, c\}, \{c, d\}, \{d, e\}, \{e, f\}, \{f, g\}$
  - With dilation 2 we get features made up of $\{a, c\}, \{b, d\}, \{c, e\}, \{d, f\}, \{e, g\}$
  - At a high level, you can get a wider context with less layers

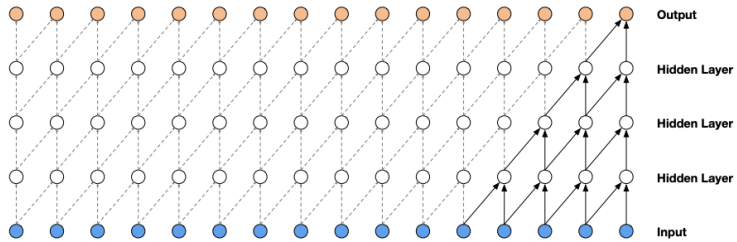


Figure 2: Visualization of a stack of causal convolutional layers.

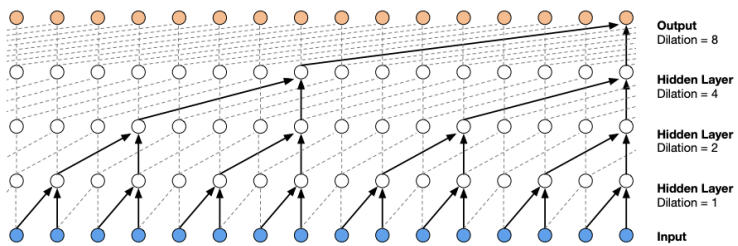


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

Max Pooling

- What if you want to summarize your data across dimensions but don't want to collapse all the channels?
- For each dimension of your word vectors, take the maximum coordinate as you move across the sentence 3 words at a time
- In PyTorch, this can be done with
  - `m=MaxPool1d(3, stride=1)`
  - `data=torch.rand(5, 10)`
  - `m(data)`

Max Pooling

- For the example we had before, this might look like below
- Notice that we have the same number of channels, but less vectors (sentence length)

{i went to}	...	{cool new laptop}
max(0.1, 0.2, 0.31)	...	max(0.91, 0.92, 0.93)
max(0.1, 0.2, 0.31)	...	max(0.91, 0.92, 0.93)
max(0.1, 0.2, 0.31)	...	max(0.91, 0.92, 0.93)
max(0.1, 0.2, 0.31)	...	max(0.91, 0.92, 0.93)
max(0.1, 0.2, 0.31)	...	max(0.91, 0.92, 0.93)

Average Pooling

- We could also do Average Pooling
- `AvgPool1d`

{i went to}	...	{cool new laptop}
avg(0.1, 0.2, 0.31)	...	avg(0.91, 0.92, 0.93)
avg(0.1, 0.2, 0.31)	...	avg(0.91, 0.92, 0.93)
avg(0.1, 0.2, 0.31)	...	avg(0.91, 0.92, 0.93)
avg(0.1, 0.2, 0.31)	...	avg(0.91, 0.92, 0.93)
avg(0.1, 0.2, 0.31)	...	avg(0.91, 0.92, 0.93)

Sum Pooling

- We could also do Sum Pooling
- `SumPool1d`

{i went to}	...	{cool new laptop}
sum(0.1, 0.2, 0.31)	...	sum(0.91, 0.92, 0.93)
sum(0.1, 0.2, 0.31)	...	sum(0.91, 0.92, 0.93)
sum(0.1, 0.2, 0.31)	...	sum(0.91, 0.92, 0.93)
sum(0.1, 0.2, 0.31)	...	sum(0.91, 0.92, 0.93)
sum(0.1, 0.2, 0.31)	...	sum(0.91, 0.92, 0.93)

Big Benefit: We can apply multiple filters in parallel!

- One of the big benefits of CNN is we don't need to wait for one filter to process data through another filter
- On the same layer level, we can apply all the filters in parallel and then put these together to get the result

Outline

- Convolutional Neural Networks
- CNN for NLP
- Example: Text Classification
- Example: Language Modeling with Gated CNN

CNN For Classification

- Goal of the model: for each sentence, extract the "sentiment" of classification of the sentence
- Older notation, but very good insights for its time
- Model has dropout on the final layer (0.5)
- Model also has a "hack": After back prop, normalize all weights so that $||w||_2 = s$ ... Do not let parameters get too large
- This basically is L_2 regularization
  - It turns out the problems below are equivalent, but there is no (analytic) mapping between t and λ
  - $min_\theta(f(\theta) + \lambda||\theta||_2^2)$
  - $min_{\theta,||\theta||_2 \leq t}(f(\theta))$

CNN For Classification

- Note that the Max Pooling is over the entire sequence length

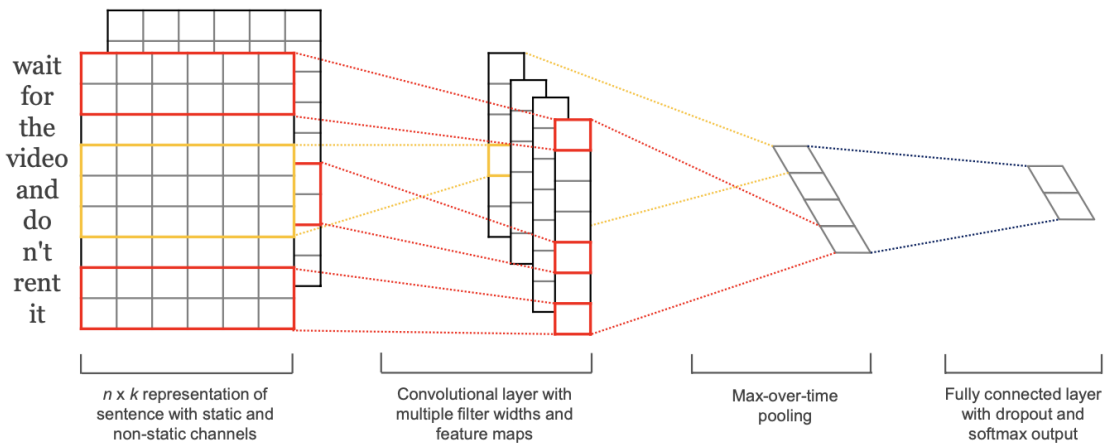


Figure 1: Model architecture with two channels for an example sentence.

CNN For Classification

- The author looked at a variety of datasets with different statistics per data set

Data	c	l	N	$\|V\|$	$\|V_{pre}\|$	*Test*
MR	2	20	10662	18765	16448	CV
SST-1	5	18	11855	17836	16262	2210
SST-2	2	19	9613	16185	14838	1821
Subj	2	23	10000	21323	17913	CV
TREC	6	10	5952	9592	9125	500
CR	2	19	3775	5340	5046	CV
MPQA	2	3	10606	6246	6083	CV

Table 1: Summary statistics for the datasets after tokenization. c: Number of target classes. l: Average sentence length. N: Dataset size. $|V|$: Vocabulary size. $|V_{pre}|$: Number of words present in the set of pre-trained word vectors. *Test*: Test set size (CV means there was no standard train/test split and thus 10-fold CV was used).

CNN For Classification

- Main goal was sentence classification: does the sentence have positive or negative sentiment?
- Question classification: is the sentence about a person, location, number, ...
- Subjective (news) or objective (opinion) language?

CNN For Classification

- Start with word vectors from Word2Vec: $x_i \in \mathbb{R}^k$
- A typical sentence looks like: $x_1 \oplus x_2 \ldots \oplus x_n$
- Concatenate words in a range $x_{i:i+h-1}$
- Multiply the concatenated vector with vector filters $w \in \mathbb{R}^{hk}$
- Note this is a little different than the notation we introduced, which is more standard now; but it's the same thing!
- Filter could be of size 2, 3, 4

CNN For Classification

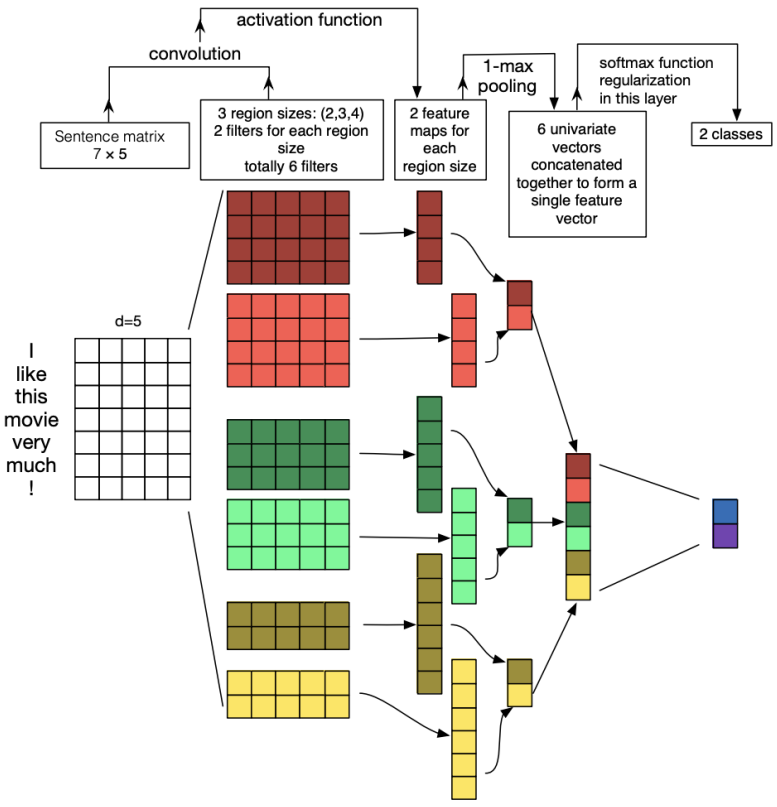
- A filter w and bias b is applied to all windows of size k
- We get a feature

$$c_i = f(w^\intercal x_{i:i+h-1} + b)$$

- Sentence: $x_1 \oplus x_2 \ldots \oplus x_n$
- All possible windows of length h: $\{x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}\}$
- You get features $\{c_1, c_2, \ldots, c_{n-h+1}\} \in \mathbb{R}^{n-h+1}$
- Do this for multiple filters and concatenate the result

CNN For Classification

- A nice picture of the idea



CNN For Classification

- 4 different models were considered
  - CNN-rand: Initialize randomly and let the model learn the word embeddings
  - CNN-static: Initialize with Word2Vec, but do not allow word vectors to be changed (`e.requires_grad=False`)
  - CNN-non-static: Initialize with Word2Vec, Embedding layer is allowed to be changed by back propagation
  - CNN-multichannel: Two Embeddings, both are Word2Vec but one is static another can be fine-tuned by back propagation
- Note: Word2Vec was compared with the embeddings from "NLP Almost From Scratch" (Lecture 1) - they did much better

CNN For Classification

- For a very cheap model, CNNs got SOTA on 4 / 7 of the tasks considered
- Note that drop out was new at this time; most other methods below would benefit from dropout

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Table 2: Results of our CNN models against other methods. **RAE**: Recursive Autoencoders with pre-trained word vectors from Wikipedia (Socher et al., 2011). **MV-RNN**: Matrix-Vector Recursive Neural Network with parse trees (Socher et al., 2012). **RNTN**: Recursive Neural Tensor Network with tensor-based feature function and parse trees (Socher et al., 2013). **DCNN**: Dynamic Convolutional Neural Network with k-max pooling (Kalchbrenner et al., 2014). **Paragraph-Vec**: Logistic regression on top of paragraph vectors (Le and Mikolov, 2014). **CCAE**: Combinatorial Category Autoencoders with combinatorial category grammar operators (Hermann and Blunsom, 2013). **Sent-Parser**: Sentiment analysis-specific parser (Dong et al., 2014). **NBSVM, MNB**: Naive Bayes SVM and Multinomial Naive Bayes with uni-bigrams from Wang and Manning (2012). **G-Dropout, F-Dropout**: Gaussian Dropout and Fast Dropout from Wang and Manning (2013). **Tree-CRF**: Dependency tree with Conditional Random Fields (Nakagawa et al., 2010). **CRF-PR**: Conditional Random Fields with Posterior Regularization (Yang and Cardie, 2014). **SVM_S**: SVM with uni-bi-trigrams, wh word, head word, POS, parser, hypernyms, and 60 hand-coded rules as features from Silva et al. (2011).

CNN For Classification

- Side effect: Fine-tuning the embeddings allows the words to know better what is related and unrelated

	Most Similar Words for	
	Static Channel	Non-static Channel
bad	*good* *terrible* *horrible* *lousy*	*terrible* *horrible* *lousy* *stupid*
good	*great* *bad* *terrific* *decent*	*nice* *decent* *solid* *terrific*
n't	*os* *ca* *ireland* *wo*	*not* *never* *nothing* *neither*
!	*2,500* *entire* *jez* *changer*	*2,500* *lush* *beautiful* *terrific*
,	*decasia* *abysmally* *demise* *valiant*	*but* *dragon* *a* *and*

Table 3: Top 4 neighboring words—based on cosine similarity—for vectors in the static channel (left) and fine-tuned vectors in the non-static channel (right) from the multichannel model on the SST-2 dataset after training.

Outline

- Convolutional Neural Networks
- CNN for NLP
- Example: Text Classification
- Example: Language Modeling with Gated CNN

CNN For Language Modeling

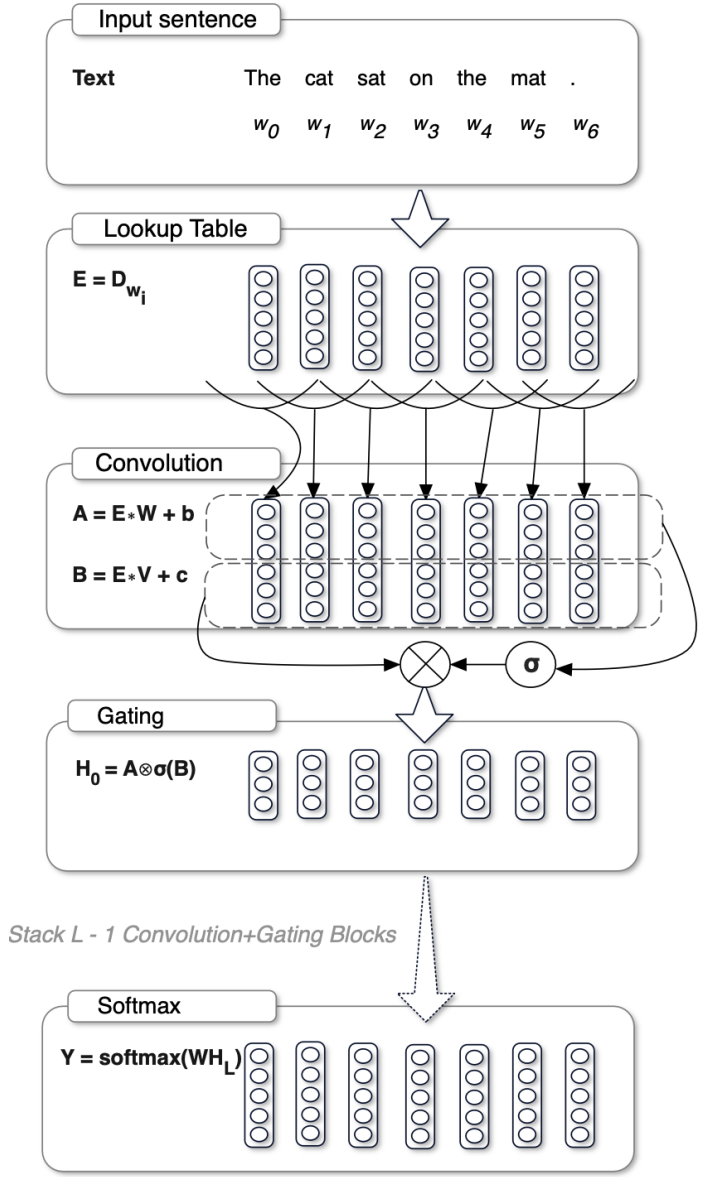
- Can you use CNNs for Language Modeling?
- In this case, we want to predict w_{N+1} from a past context window $(w_0, w_1, \ldots, w_N)$
- Idea: As usual, get the embeddings for all words in a context and convolve the embedding layer with n filters
- Assume the the kernels have a filter size of k
- The data is initially $X \in \mathbb{R}^{m \times N}$ so that the word dimension is m

CNN For Language Modeling

- From one layer to the next, allow the model the chance to pass some information
- Notation is a little dated:
  $h_l(X) = (X * W + b) \otimes \sigma(X * V + c)$
  - Here, we convolve X with two filters such that we get the same dimensional result
  - One result of a filter goes through a nonlinearity, another does not
  - Every so often, add a skip connection (this is done with the aid of padding and 1 dimensional convolutions to transform the input's number of channels to the same as the output - will explain at the end)
- Paper has other tricks - adaptive softmax
- Nice argument on why the architecture enables gradient flow - they compare with LSTMs which we'll see next week or in March

CNN For Language Modeling

- Architecture is fairly simple



CNN Word Vectors

- Another idea is to get embeddings for each *character* in a word and use these as the initial embedding
- Then, put a convolutional layer after this layer so that effectively each *word* gets an embedding
- We are effectively mixing the character embeddings to *learn* the word embeddings

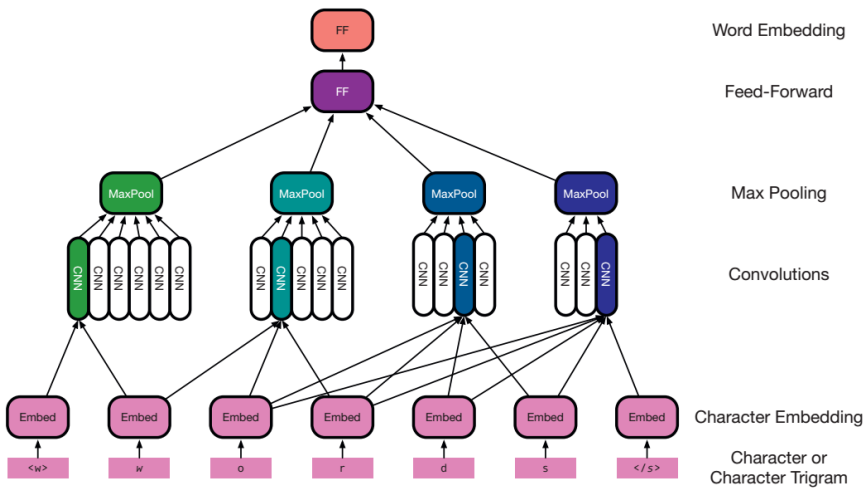
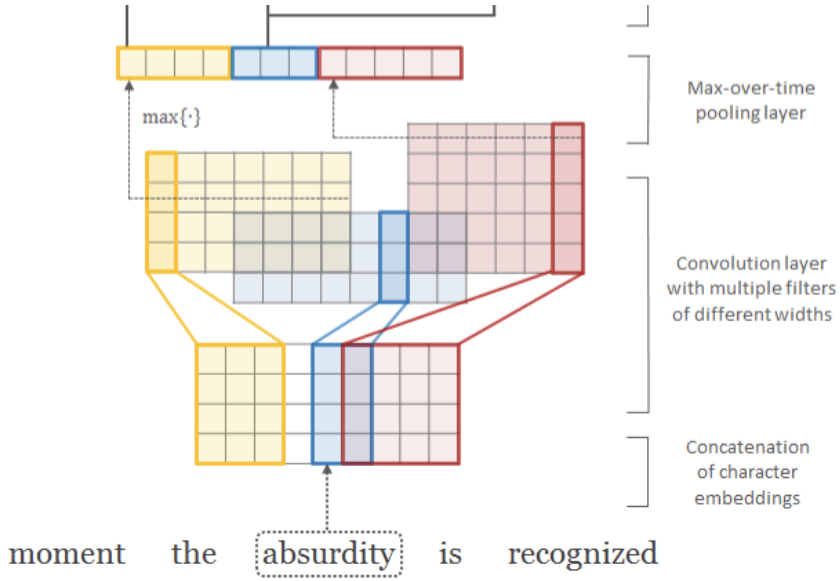


Figure 12.8 Word representation composed from a letter sequence using a convolutional neural network. Convolutions over 2-, 3-, 4-, and 5-letter *n*-grams are shown.

CNN Word Vectors

- These type of embeddings can be a part of larger systems, we'll look at this later



Convolutions with kernels of size 1

- Sometimes we have data that is D dimensions and T sequence length
- One idea is to use d a kernels of width 1 (!) and apply them to the data
- This way, you now have data that is d dimensional and T in sequence length
- This general idea is called down sampling, and it is useful especially when you have residual connections and need to add $h + x$ where x is your original data and h is the output of some block layer
- This is used in the Gated CNN LM discussed above

References

- Nice animations
- CNN Pictures
- CNN for Text Classification
- CNN for Language Modeling
- NLP Almost from Scratch
- CNN for Sentence Classification
- Character Aware Neural Language Models
- Wave Net
- Neural Machine Translation