WILDML

AI, DEEP LEARNING, NLP

MENU

UNDERSTANDING CONVOLUTIONAL NEURAL NETWORKS FOR NLP

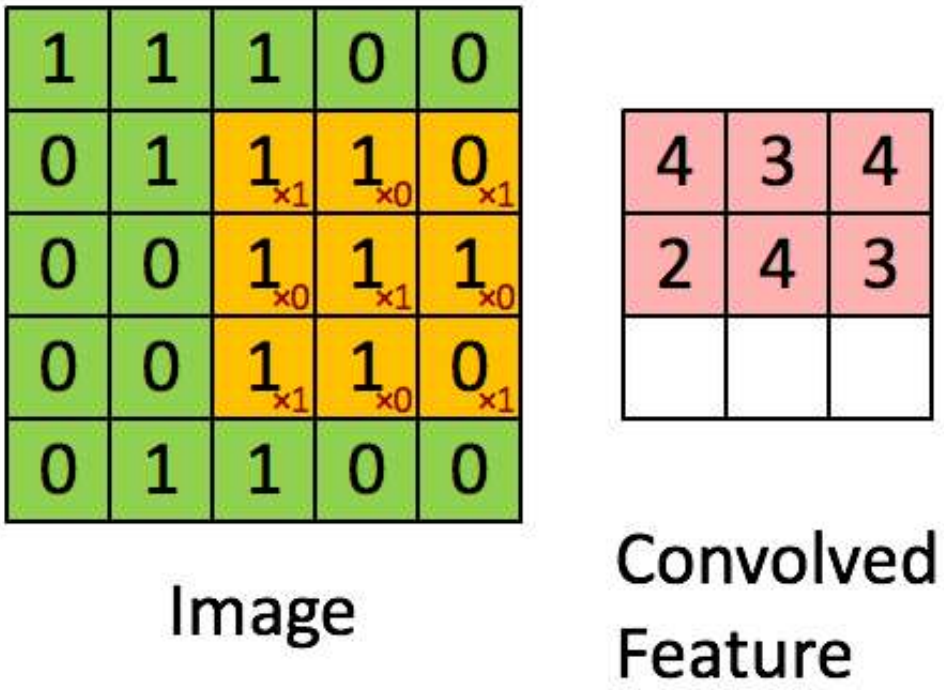
November 7, 2015

When we hear about Convolutional Neural Network (CNNs), we typically think of Computer Vision. CNNs were responsible for major breakthroughs in Image Classification and are the core of most Computer Vision systems today, from Facebook's automated photo tagging to self-driving cars.

More recently we've also started to apply CNNs to problems in Natural Language Processing and gotten some interesting results. In this post I'll try to summarize what CNNs are, and how they're used in NLP. The intuitions behind CNNs are somewhat easier to understand for the Computer Vision use case, so I'll start there, and then slowly move towards NLP.

WHAT IS CONVOLUTION?

The for me easiest way to understand a convolution is by thinking of it as a sliding window function applied to a matrix. That's a mouthful, but it becomes quite clear looking at a visualization:



Convolution with 3×3 Filter. Source: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Imagine that the matrix on the left represents an black and white image. Each entry corresponds to one pixel, 0 for black and 1 for white (typically it's between 0 and 255 for grayscale images). The sliding window is called a kernel, filter, or feature detector. Here we use a 3×3 filter, multiply its values element-wise with the original matrix, then sum them up. To get the full convolution we do this for each element by sliding the filter over the whole matrix.

You may be wondering wonder what you can actually do with this. Here are some intuitive examples.

AVERAGING EACH PIXEL WITH ITS NEIGHBORING VALUES BLURS AN IMAGE:

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

TAKING THE DIFFERENCE BETWEEN A PIXEL AND ITS NEIGHBORS DETECTS EDGES:

(To understand this one intuitively, think about what happens in parts of the image that are smooth, where a pixel color equals that of its neighbors: The additions cancel and the resulting value is 0, or black. If there’s a sharp edge in intensity, a transition from white to black for example, you get a large difference and a resulting white value)

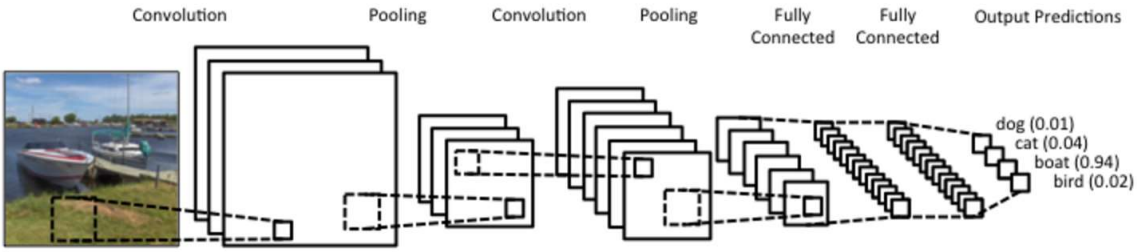
	0	1	0	
	1	-4	1	
	0	1	0	

The GIMP manual has a few other examples. To understand more about how convolutions work I also recommend checking out Chris Olah's post on the topic.

WHAT ARE CONVOLUTIONAL NEURAL NETWORKS?

Now you know what convolutions are. But what about CNNs? CNNs are basically just several layers of convolutions with nonlinear activation functions like ReLU or tanh applied to the results. In a traditional feedforward neural network we connect each input neuron to each output neuron in the next layer. That's also called a fully connected layer, or affine layer. In CNNs we don't do that. Instead, we use convolutions over the input layer to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer applies different filters, typically hundreds or thousands like the ones showed above, and combines their results. There's also something something called pooling (subsampling) layers, but I'll get into that later. During the training phase, a CNN automatically learns the values of its filters based on the task you want to perform.

For example, in Image Classification a CNN may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features, such as facial shapes in higher layers. The last layer is then a classifier that uses these high-level features.



There are two aspects of this computation worth paying attention to: Location Invariance and Compositionality. Let's say you want to classify whether or not there's an elephant in an image. Because you are sliding your filters over the whole image you don't really care where the elephant occurs. In practice, pooling also gives you invariance to translation, rotation and scaling, but more on that later. The second key aspect is (local) compositionality. Each filter composes a local patch of lower-level features into higher-level representation. That's why CNNs are so powerful in Computer Vision. It makes intuitive sense that you build edges from pixels, shapes from edges, and more complex objects from shapes.

SO, HOW DOES ANY OF THIS APPLY TO NLP?

Instead of image pixels, the input to most NLP tasks are sentences or documents represented as a matrix. Each row of the matrix corresponds to one token, typically a word, but it could be a character. That is, each row is vector that represents a word. Typically, these vectors are word embeddings (low-dimensional representations) like word2vec or GloVe, but they could also be one-hot vectors that index the word into a

vocabulary. For a 10 word sentence using a 100-dimensional embedding we would have a 10×100 matrix as our input. That's our "image".

In vision, our filters slide over local patches of an image, but in NLP we typically use filters that slide over full rows of the matrix (words). Thus, the "width" of our filters is usually the same as the width of the input matrix. The height, or region size, may vary, but sliding windows over 2-5 words at a time is typical. Putting all the above together, a Convolutional Neural Network for NLP may look like this (take a few minutes and try understand this picture and how the dimensions are computed. You can ignore the pooling for now, we'll explain that later):

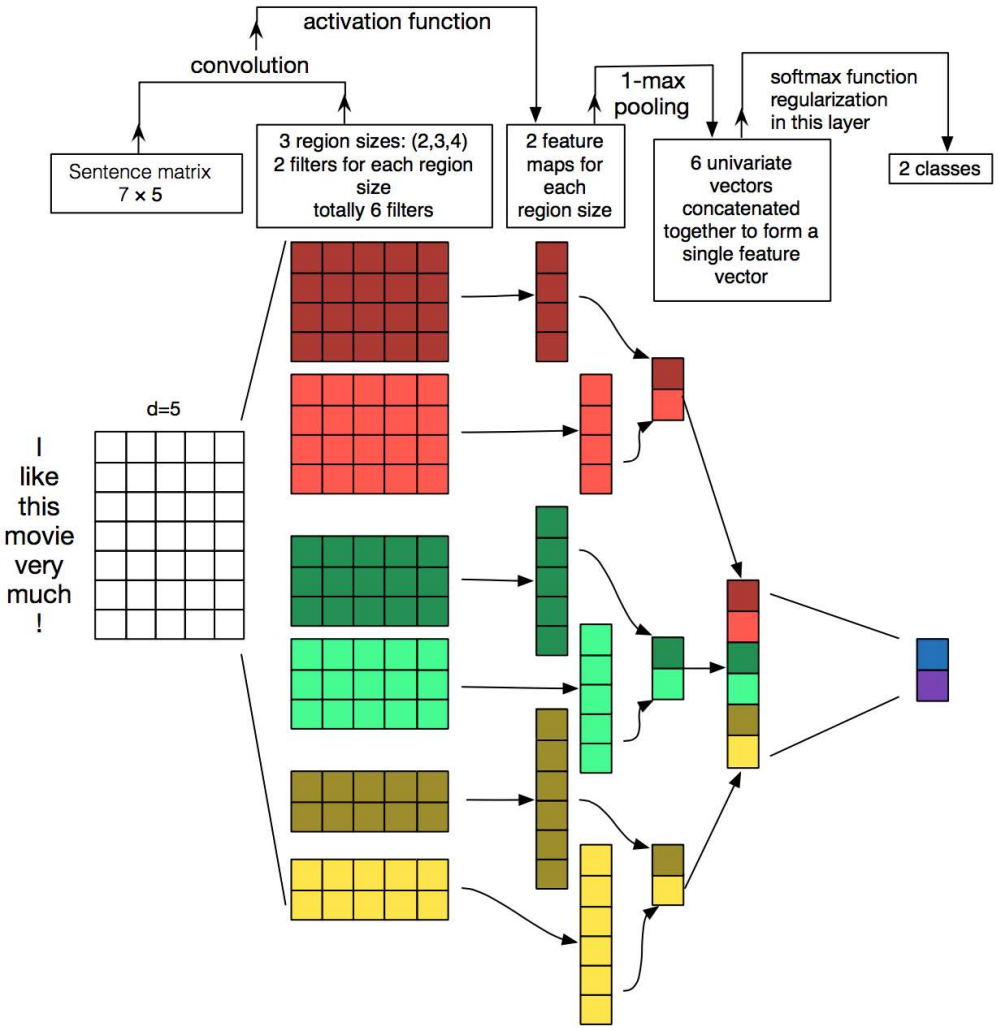


Illustration of a Convolutional Neural Network (CNN) architecture for sentence classification. Here we

depict three filter region sizes: 2, 3 and 4, each of which has 2 filters. Every filter performs convolution on the sentence matrix and generates (variable-length) feature maps. Then 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states. Source: Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.

What about the nice intuitions we had for Computer Vision? Location Invariance and local Compositionality made intuitive sense for images, but not so much for NLP. You probably do care a lot where in the sentence a word appears. Pixels close to each other are likely to be semantically related (part of the same object), but the same isn't always true for words. In many languages, parts of phrases could be separated by several other words. The compositional aspect isn't obvious either. Clearly, words compose in some ways, like an adjective modifying a noun, but how exactly this works what higher level representations actually "mean" isn't as obvious as in the Computer Vision case.

Given all this, it seems like CNNs wouldn't be a good fit for NLP tasks. Recurrent Neural Networks make more intuitive sense. They resemble how we process language (or at least how we think we process language): Reading sequentially from left to right. Fortunately, this doesn't mean that CNNs don't work. All models are wrong, but some are useful. It turns out that CNNs applied to NLP problems perform quite well. The simple Bag of Words model is an obvious oversimplification with incorrect assumptions, but has nonetheless been the standard approach for years and lead to pretty good results.

A big argument for CNNs is that they are fast. Very fast. Convolutions are a central part of computer graphics and implemented on a hardware level on GPUs. Compared to something like n-grams, CNNs are also efficient in terms of

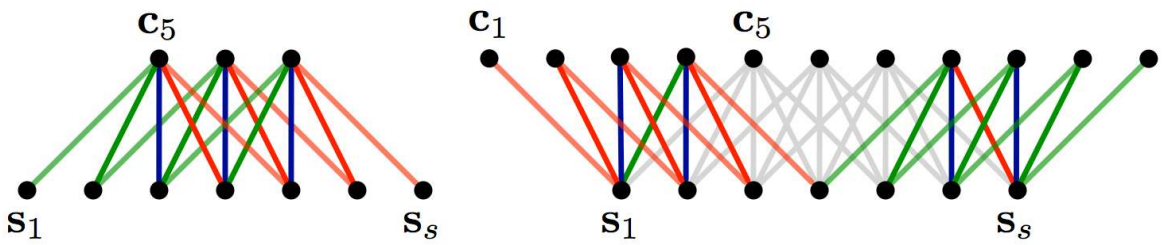
representation. With a large vocabulary, computing anything more than 3-grams can quickly become expensive. Even Google doesn't provide anything beyond 5-grams. Convolutional Filters learn good representations automatically, without needing to represent the whole vocabulary. It's completely reasonable to have filters of size larger than 5. I like to think that many of the learned filters in the first layer are capturing features quite similar (but not limited) to n-grams, but represent them in a more compact way.

CNN HYPERPARAMETERS

Before explaining at how CNNs are applied to NLP tasks, let's look at some of the choices you need to make when building a CNN. Hopefully this will help you better understand the literature in the field.

NARROW VS. WIDE CONVOLUTION

When I explained convolutions above I neglected a little detail of how we apply the filter. Applying a 3×3 filter at the center of the matrix works fine, but what about the edges? How would you apply the filter to the first element of a matrix that doesn't have any neighboring elements to the top and left? You can use zero-padding. All elements that would fall outside of the matrix are taken to be zero. By doing this you can apply the filter to every element of your input matrix, and get a larger or equally sized output. Adding zero-padding is also called wide convolution, and not using zero-padding would be a narrow convolution. An example in 1D looks like this:

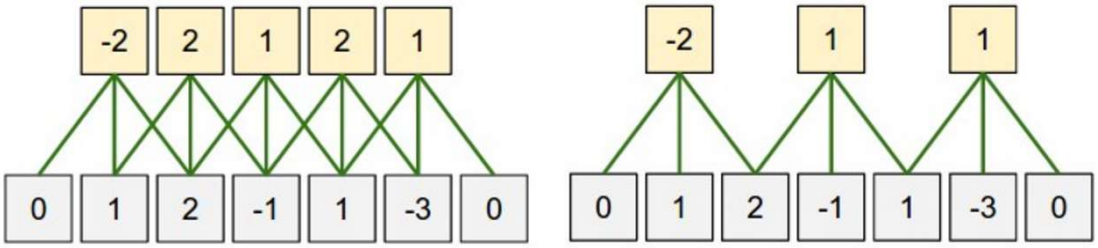


Narrow vs. Wide Convolution. Filter size 5, input size 7. Source: A Convolutional Neural Network for Modelling Sentences (2014)

You can see how wide convolution is useful, or even necessary, when you have a large filter relative to the input size. In the above, the narrow convolution yields an output of size $(7-5)+1=3$, and a wide convolution an output of size $(7+2*4-5)+1=11$. More generally, the formula for the output size is $n_{out}=(n_{in}+2*n_{padding}-n_{filter})+1$.

STRIDE SIZE

Another hyperparameter for your convolutions is the stride size, defining by how much you want to shift your filter at each step. In all the examples above the stride size was 1, and consecutive applications of the filter overlapped. A larger stride size leads to fewer applications of the filter and a smaller output size. The following from the Stanford cs231 website shows stride sizes of 1 and 2 applied to a one-dimensional input:



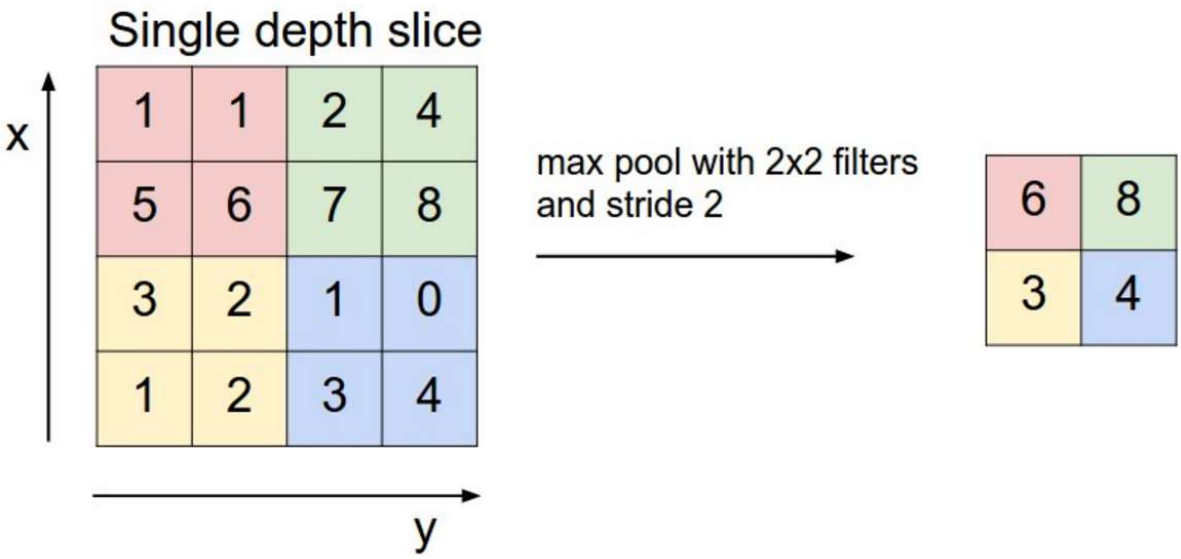
Convolution Stride Size. Left: Stride size 1. Right: Stride size 2. Source: http://cs231n.github.io/convolutional-networks/

In the literature we typically see stride sizes of 1, but a larger stride size may allow you to build a model that behaves somewhat similarly to a Recursive Neural Network, i.e. looks like a tree.

POOLING LAYERS

A key aspect of Convolutional Neural Networks are pooling

layers, typically applied after the convolutional layers. Pooling layers subsample their input. The most common way to do pooling it to apply a max operation to the result of each filter. You don't necessarily need to pool over the complete matrix, you could also pool over a window. For example, the following shows max pooling for a 2×2 window (in NLP we typically are apply pooling over the complete output, yielding just a single number for each filter):



Max pooling in CNN. Source: http://cs231n.github.io/convolutional-networks/#pool

Why pooling? There are a couple of reasons. One property of pooling is that it provides a fixed size output matrix, which typically is required for classification. For example, if you have 1,000 filters and you apply max pooling to each, you will get a 1000-dimensional output, regardless of the size of your filters, or the size of your input. This allows you to use variable size sentences, and variable size filters, but always get the same output dimensions to feed into a classifier.

Pooling also reduces the output dimensionality but (hopefully) keeps the most salient information. You can think of each filter as detecting a specific feature, such as detecting if the sentence contains a negation like "not amazing" for example. If this phrase occurs somewhere in the sentence, the result of

applying the filter to that region will yield a large value, but a small value in other regions. By performing the max operation you are keeping information about whether or not the feature appeared in the sentence, but you are losing information about where exactly it appeared. But isn't this information about locality really useful? Yes, it is and it's a bit similar to what a bag of n-grams model is doing. You are losing global information about locality (where in a sentence something happens), but you are keeping local information captured by your filters, like "not amazing" being very different from "amazing not".

In imagine recognition, pooling also provides basic invariance to translating (shifting) and rotation. When you are pooling over a region, the output will stay approximately the same even if you shift/rotate the image by a few pixels, because the max operations will pick out the same value regardless.

CHANNELS

The last concept we need to understand are channels. 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 (word2vec and GloVe for example), or you could have a channel for the same sentence represented in different languages, or phrased in different ways.

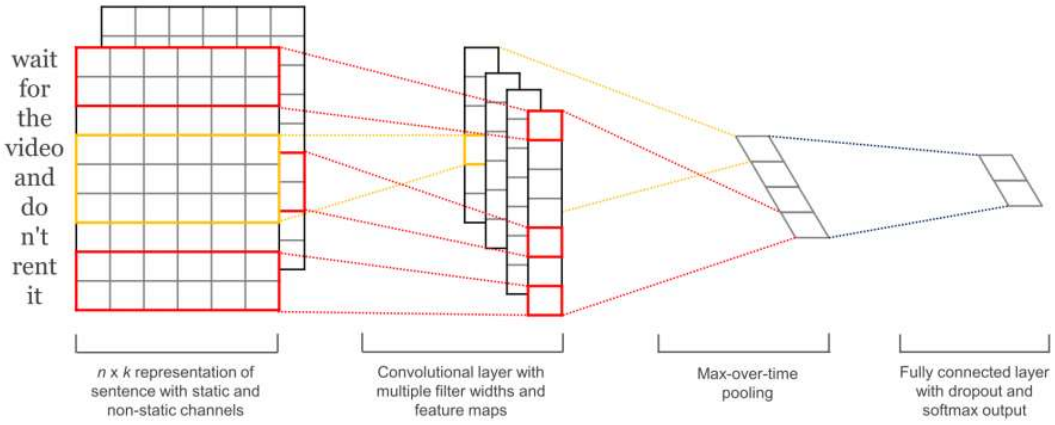
CONVOLUTIONAL NEURAL NETWORKS APPLIED TO NLP

Let's now look at some of the applications of CNNs to Natural Language Processing. I'll try it summarize some of the research results. Invariably I'll miss many interesting applications (do

let me know in the comments), but I hope to cover at least some of the more popular results.

The most natural fit for CNNs seem to be classifications tasks, such as Sentiment Analysis, Spam Detection or Topic Categorization. Convolutions and pooling operations lose information about the local order of words, so that sequence tagging as in PoS Tagging or Entity Extraction is a bit harder to fit into a pure CNN architecture (though not impossible, you can add positional features to the input).

[1] Evaluates a CNN architecture on various classification datasets, mostly comprised of Sentiment Analysis and Topic Categorization tasks. The CNN architecture achieves very good performance across datasets, and new state-of-the-art on a few. Surprisingly, the network used in this paper is quite simple, and that's what makes it powerful.The input layer is a sentence comprised of concatenated word2vec word embeddings. That's followed by a convolutional layer with multiple filters, then a max-pooling layer, and finally a softmax classifier. The paper also experiments with two different channels in the form of static and dynamic word embeddings, where one channel is adjusted during training and the other isn't. A similar, but somewhat more complex, architecture was previously proposed in [2]. [6] Adds an additional layer that performs "semantic clustering" to this network architecture.



Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification

[4] Trains a CNN from scratch, without the need for for pre-trained word vectors like word2vec or GloVe. It applies convolutions directly to one-hot vectors. The author also proposes a space-efficient bag-of-words-like representation for the input data, reducing the number of parameters the network needs to learn. In [5] the author extends the model with an additional unsupervised "region embedding" that is learned using a CNN predicting the context of text regions. The approach in these papers seems to work well for long-form texts (like movie reviews), but their performance on short texts (like tweets) isn't clear. Intuitively, it makes sense that using pre-trained word embeddings for short texts would yield larger gains than using them for long texts.

Building a CNN architecture means that there are many hyperparameters to choose from, some of which I presented above: Input represenations (word2vec, GloVe, one-hot), number and sizes of convolution filters, pooling strategies (max, average), and activation functions (ReLU, tanh). [7] performs an empirical evaluation on the effect of varying hyperparameters in CNN architectures, investigating their impact on performance and variance over multiple runs. If you are looking to implement your own CNN for text classification, using the results of this paper as a starting point would be an excellent idea. A few results that stand out are that max-pooling always beat average pooling, that the ideal filter sizes are important but task-

dependent, and that regularization doesn't seem to make a big different in the NLP tasks that were considered. A caveat of this research is that all the datasets were quite similar in terms of their document length, so the same guidelines may not apply to data that looks considerably different.

[8] explores CNNs for Relation Extraction and Relation Classification tasks. In addition to the word vectors, the authors use the relative positions of words to the entities of interest as an input to the convolutional layer. This models assumes that the positions of the entities are given, and that each example input contains one relation. [9] and [10] have explored similar models.

Another interesting use case of CNNs in NLP can be found in [11] and [12], coming out of Microsoft Research. These papers describe how to learn semantically meaningful representations of sentences that can be used for Information Retrieval. The example given in the papers includes recommending potentially interesting documents to users based on what they are currently reading. The sentence representations are trained based on search engine log data.

Most CNN architectures learn embeddings (low-dimensional representations) for words and sentences in one way or another as part of their training procedure. Not all papers though focus on this aspect of training or investigate how meaningful the learned embeddings are. [13] presents a CNN architecture to predict hashtags for Facebook posts, while at the same time generating meaningful embeddings for words and sentences. These learned embeddings are then successfully applied to another task - recommending potentially interesting documents to users, trained based on clickstream data.

CHARACTER-LEVEL CNNS

So far, all of the models presented were based on words. But there has also been research in applying CNNs directly to characters. [14] learns character-level embeddings, joins them with pre-trained word embeddings, and uses a CNN for Part of Speech tagging. [15][16] explores the use of CNNs to learn directly from characters, without the need for any pre-trained embeddings. Notably, the authors use a relatively deep network with a total of 9 layers, and apply it to Sentiment Analysis and Text Categorization tasks. Results show that learning directly from character-level input works very well on large datasets (millions of examples), but underperforms simpler models on smaller datasets (hundreds of thousands of examples). [17] explores to application of character-level convolutions to Language Modeling, using the output of the character-level CNN as the input to an LSTM at each time step. The same model is applied to various languages.

What's amazing is that essentially all of the papers above were published in the past 1-2 years. Obviously there has been excellent work with CNNs on NLP before, as in Natural Language Processing (almost) from Scratch, but the pace of new results and state of the art systems being published is clearly accelerating.

QUESTIONS OR FEEDBACK? LET ME KNOW IN THE COMMENTS. THANKS FOR READING!

PAPER REFERENCES

- [1] Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014),

1746 – 1751.

- [2] Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. Acl, 655 – 665.
- [3] Santos, C. N. dos, & Gatti, M. (2014). Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts. In COLING-2014 (pp. 69 – 78).
- [4] Johnson, R., & Zhang, T. (2015). Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. To Appear: NAACL-2015, (2011).
- [5] Johnson, R., & Zhang, T. (2015). Semi-supervised Convolutional Neural Networks for Text Categorization via Region Embedding.
- [6] Wang, P., Xu, J., Xu, B., Liu, C., Zhang, H., Wang, F., & Hao, H. (2015). Semantic Clustering and Convolutional Neural Network for Short Text Categorization. Proceedings ACL 2015, 352 – 357.
- [7] Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification,
- [8] Nguyen, T. H., & Grishman, R. (2015). Relation Extraction: Perspective from Convolutional Neural Networks. Workshop on Vector Modeling for NLP, 39 – 48.
- [9] Sun, Y., Lin, L., Tang, D., Yang, N., Ji, Z., & Wang, X. (2015). Modeling Mention , Context and Entity with Neural Networks for Entity Disambiguation, (Ijcai), 1333 – 1339.
- [10] Zeng, D., Liu, K., Lai, S., Zhou, G., & Zhao, J. (2014). Relation Classification via Convolutional Deep Neural Network. Coling, (2011), 2335 – 2344.
- [11] Gao, J., Pantel, P., Gamon, M., He, X., & Deng, L. (2014). Modeling Interestingness with Deep Neural Networks.
- [12] Shen, Y., He, X., Gao, J., Deng, L., & Mesnil, G. (2014). A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval. Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management - CIKM '14, 101 – 110.

- [13] Weston, J., & Adams, K. (2014). # T AG S PACE : Semantic Embeddings from Hashtags, 1822–1827.
- [14] Santos, C., & Zadrozny, B. (2014). Learning Character-level Representations for Part-of-Speech Tagging. Proceedings of the 31st International Conference on Machine Learning, ICML-14(2011), 1818–1826.
- [15] Zhang, X., Zhao, J., & LeCun, Y. (2015). Character-level Convolutional Networks for Text Classification, 1–9.
- [16] Zhang, X., & LeCun, Y. (2015). Text Understanding from Scratch. arXiv E-Prints, 3, 011102.
- [17] Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2015). Character-Aware Neural Language Models.

Posted in: Convolutional Neural Networks, Deep Learning, Neural Networks

← *Recurrent Neural Network Tutorial, Part 4 – Implementing a GRU/LSTM RNN with Python and Theano*

Implementing a CNN for Text Classification in TensorFlow →

61 Comments WildML 1 Login

Recommend 16 Share Sort by Best

Join the discussion…

Nando de Freitas • a year ago

Please see also

Dimitrios Kotzias, Misha Denil, Nando de Freitas and Padhraic Smyth. From Group to Individual Labels using Deep Features. In ACM SIGKDD. 2015.

Misha Denil, Alban Demiraj and Nando de Freitas. Extraction of Salient Sentences from Labelled Documents. University of Oxford. 2014.

Instead of going down to characters or morphemes, we chose to go up to sentences and entire documents. This enabled us to do multi-instance transfer. The convnet used by us is closer to what is used in vision.

7 • Reply • Share ›

Oren Etzioni • a year ago

Fantastic, intuitive explanation!

3 • Reply • Share ›

ffmpbgrnn • a year ago

Awesome work! Thank you!

2 • Reply • Share ›

Thomas Phifer • 2 months ago

Here's another application that's interesting:
Using Neural Networks to classify atom states
https://www.wired.com/2017/...

1 • Reply • Share ›

BitcoinKing • 7 months ago

Super article.

Could you point me on how to use CNN (or other network) to make IR complex tasks: for example sentiment analysis on large text (movie review) is useless because not capture aspects (good scenario but bad effects). So the task is make for text a list of pos\neg pairs, each is sentiment that belong to object it describes.

Next level is to build tree of terms, that describes the topic of text (ontology): for banks texts i need categories of terms i.e. problems of people using some bank categorized by type of service. Note here this is not sentiments, because

categorized by type of service. Note here this is not sentiments, because problems may be described in domain-specific language

1 • Reply • Share ›

Shaohua Li • a year ago

I really couldn't come up with an intuition how the convolution works for NLP. So I'm so surprised that it could perform well. You know in CV, convolution operators detect edges formed by aligned pixels. So it's easy to understand it works.

While in the space of word embeddings, things are totally different. Different dimensions capture different semantics/syntax. So we can suppose that the convolution layer detects some common (but unknown) semantics within n-grams. But the following 1-max pooling makes things mysterious. You know after max-pooling, only the max value matters, but we don't care which dimensionality this value is at! Then the salient semantics are fed into a softmax classifier (or LR -- logistic regression). That means the classifier cannot tell the difference between different semantics, and only know how salient they are. The learning of LR weights assigns different weights to different features, however since we've lost the information which dimension this feature comes from, such differentiation of different dimensions seems pointless. For example, we could slide the text window, and the same salient semantics will end up in a different dimension of the final feature vector. But this time it's assigned with a different weight. I'm confused how learning the weights could help the classification? Sorry if my comments are difficult to understand...

1 • Reply • Share ›

Shaohua Li ➦ Shaohua Li • a year ago

Sorry I misunderstood. The pooling happens along the word axis (rows), not the embedding axis (columns)... Now it makes sense to me.

1 • Reply • Share ›

Price Elder • a year ago

Great stuff, would love to see some recent work on character-level CNNs being added here!

Zhang et al. 2015: Character-level CNN for document classification
http://arxiv.org/abs/1509.0...
Kim et al. 2015: Character-level CNN and LSTM for language modeling
http://arxiv.org/abs/1508.0...

1 • Reply • Share ›

Denny Britz Mod ➦ Price Elder • a year ago

Thanks! I'll review these and add them, I remember having seen them before ;) Character-level models are super promising.

• Reply • Share ›

Ricardo Teixeira • a year ago

Ricardo Teixeira • a year ago

Thank you, awesome stuff!

1 • Reply • Share ›

Mohamed Galal • a month ago

Fantastic, you save months of reading papers
Thank you

• Reply • Share ›

Nams • a month ago

One of the best posts I have read till now. nicely presented, very intuitive. Thank you.

• Reply • Share ›

Benjamin Feifke • 2 months ago

can someone please explain to me the matrix representation of a sentence?

• Reply • Share ›

scharfy → Benjamin Feifke • a month ago

Each word in your entire corpus is first embedded into a vector space of a fixed dimension that you choose (let's say 5) using an embedding algorithm like word2vec. So all the words are now 5d vectors.

Take a sentence with say 10 words. The first word is row 1. The second word is row 2. ..etc ... the last word is row 10. Your sentence is now a 10 row * 5 column matrix in order. Each row is a word, each column is a dimension in the word embedding (no real meaning).

1 • Reply • Share ›

Farhad Bin Siddique • 2 months ago

For my CNN model, I would like to check which n-grams are max-pooled the most after the max pooling, during training, in order to see which n-grams are the most significant for the classification, and if it makes sense. Would you please let me know how I can do that? My model is basically the same as the paper by Yoon Kim, with one layer CNN on word2vec initialized vectors, and a max pooling layer followed by a fully connected layer. Thank you!

• Reply • Share ›

Tom • 4 months ago

lstm output is vector, thoungh softmax why tag ???

• Reply • Share ›

Tom • 4 months ago

lstm output is vector, thoungh softmax why tag ???

• Reply • Share ›

Hritik Jain • 4 months ago

Hi. Thanks for a great analogy with images.
Can you please elaborate on this statement you wrote above, "PoS Tagging or Entity Extraction is a bit harder to fit into a pure CNN architecture (though not impossible, you can add positional features to the input)"
Can we in advance add a bias to our CNN to focus on specific areas of an image?
I require that for an image classification task that I am doing. It would be really helpful if you could shed some light.
Thanks.

• Reply • Share ›

Oneil Hsiao • 4 months ago

It was an awesome article, thank you for sharing!!

There is a thing about the filter values I didn't get for a while: as many articles like you mentioned "... to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer ... ", and said "the filters are initialized randomly and automatically learned from the data during training."

My question is if the filter values are not arranged in a CNN (i.e., values from randomly learned), how could we know a CNN learns edges at the first layer, and detect shapes at the second layer, etc.? Thank you!

• Reply • Share ›

Denny Britz Mod → Oneil Hsiao • 4 months ago

You can visualize where in an image (and for which images) certain filters activate (i.e. how high their values are) and therefore find out what they "represent". There are various techniques of visualizing the representations of such intermediate layers.

1 • Reply • Share ›

Oneil Hsiao → Denny Britz • 4 months ago

Hi Denny, thank you very much for the reply :-)

Yes, by visualizing the layers we can know how they represent (there are some references indicated e.g., MD Zeiler, R Fergus, Visualizing and understanding convolutional networks, 2014).

My original question may not be clear. I'd like to ask: if the filter vectors are learned from the arbitrary values, which I know they are, how a CNN seems always to learn an image from edges, shapes, and so on? It looks like a CNN could find its own way (or say pattern?) to put the filter vectors down in an order. My guess is that the 'filtering-pooling' process resizes the original image, so the CNN would learn the image features in the hierarchical nature of it. Is it correct or ? Thank you

or it. Is it correct or not? Thank you.

Reply • Share ›

Miguel • 6 months ago

Hello,
Thanks for the article. It was very helpful for me.
But I didn't understand very well "POOLING LAYERS" section. Can you clarify this part?

" For example, if you have 1,000 filters and you apply max pooling to each, you will get a 1000-dimensional output, regardless of the size of your filters, or the size of your input. This allows you to use variable size sentences, and variable size filters, but always get the same output dimensions to feed into a classifier."

I didn't understand, because for the example that is given, if I had a bigger sentence wouldn't it result in a bigger matrix after the max pool? E.g., imagine you have an input with two more words, i.e., a input matrix of size 6 x 4 (added to more rows). Why in such case the max poo will result with the same size?

Reply • Share ›

Stefan • 6 months ago

How could filters for this application look like? Are they the same as the filters for image classification?

Reply • Share ›

Desy • 7 months ago

I am still confused how mutiple filter with 3 region size (two filters for each region) can do convolution.

eg I have the data.
Maximum length of this sentence is 10, the number 10,3,18 and 20 are indices of the word, and since Max sentence length is 10 remaining in padding to 0. The indices each have a dimension of 300.

[10 3 18 20 0 0 0 0 0 0]

when we want to do convolution why we must do the padding on the left and on the right of the sentence with 4 zeros, whether it is in wide convolution mean? so that it becomes:

[0 0 0 0 10 3 18 20 0 0 0 0 0 0 0 0 0 0]

how a filter with size (3,4,5) performs convolution on the data above.

Reply • Share ›

Zhiyi Luo • 8 months ago

Thanks for this great article.
But I'm a little confused about the variables creation in the github code.
Why do you use `tf.Variable` to create W and b in the conv_relu_layer? Why not

Why do you use "tf.variable" to create W and b in the convolution layer? Why not use tf.get_variable? If we use `tf.Variable`, when we process each training batch, a different set of variables will be created, isn't it?

Reply • Share ›

Justine Parer • 9 months ago

What do you mean by "textual analysis"?

Reply • Share ›

Diego Vidaurre • 9 months ago

really nice overview - thanks a lot

Reply • Share ›

Artur Aiguzhinov • a year ago

Hi, Thank you for the great tutorial! I would like to as a question about a filter (or kernel). In the graphical example above, the 3x3 matrix of filter has values of [[1,0,1],[0,1,0],[1,0,1]]. I understand that these are arbitrary values, but in the real CNN design who sets these values?

Reply • Share ›

Denny Britz Mod → Artur Aiguzhinov • a year ago

The filters are initialized randomly and automatically learned from the data during training.

Reply • Share ›

李小婉 • a year ago

Thanks~~ Can I share this article in weibo?

Reply • Share ›

Denny Britz Mod → 李小婉 • a year ago

Sure :)

Reply • Share ›

niderhoff • a year ago

Excellent article, thank you!

Reply • Share ›

tkengo • a year ago

Thank you for your awesome work!

I'm a programmer in Japan. I have read this article and I think this is a great work, so I would like to deliver the article to many Japanese.
May I translate it into Japanese and publish it on my blog http://tkengo.github.io/?

Reply • Share ›

Denny Britz Mod → tkengo • a year ago

Hi Kengo,

Definitely, that would be great! よろしくお願いします！ :)

Reply • Share ›

tkengo → Denny Britz • a year ago

I appreciate it. ありがとうございます！

Reply • Share ›

fideldidum • a year ago

I'm not sure if i get the channel thing right. I imagine the matrix of a sentence with dimension sentence_length x word_embedding_size. If I want to add another channel of the same size, it would be sentence_length x number_of_channels x embeddings_size. Is this right? I have problems with that, because in Tensorflow the number_of_channels is actually the last dimension of the input tensor. So the dimensions have to be sentence_length x embedding_size x number_of_channels. The first dimension, which I left out, is the batch_size. In addition most papers use embeddings of different sizes. For example dos Santos, Xiang and Zhou use a Word Embedding of size 400 and a Word Position Embedding of size 70 in their paper 'Classifying Relations by Ranking with Convolutional Neural Networks' . I don't see any possibility how to combine this into one matrix except 'padding' or simply concatenating the vectors and not use channels. But in fact those are different channels of the same input, so I want my network to see it as different channels, and I am not quite sure if simply concatenating them yields the same results.... Did I get something wrong here?

Reply • Share ›

Denny Britz Mod → fideldidum • a year ago

The order of dimensions doesn't really matter, you can always shuffle them if you want.

But you're right in that different channels must be of the same size (embedding_dim x sentence_length) if you want to use Tensorflow's built-in channel support. That's because channels are mostly used in image recognition where RGB channels must have the same size by definition.

But you could just represent 2 different channels as 2 tensors with different dimensions and then perform the convolutions on each of them and combine the results afterwards. This will give the same results, depending on whether or not you decide to share the convolutional filter weights or not.

Reply • Share ›

fideldidum → Denny Britz • a year ago

Thank you very much :)

Reply • Share ›

Volkan Cirik • a year ago

I would rather use Figure I of this paper http://arxiv.org/pdf/1508.0... instead of Zhang and Wallace's figure.

Assuming stride I, the number of feature maps is not correct in that figure. For region size 2, there should be filter size 6x5. For region size 3, filter size should be 5x5, and for region size 4, filter size should be 4x5.

More importantly, there is no alignment between above and below of the diagram : it seems as if activation function reduces the size of 6 filters to nx1. IMHO the suggested figure explains better.

• Reply • Share ›

Denny Britz Mod → Volkan Cirik • a year ago

Hm, can you be more specific? Looking at it I don't see the error in the picture, but maybe I'm missing something. A 2x5 filter with stride 1 sliding over a 7x5 input produces a 6-dimensional feature map. That seems right to me?

• Reply • Share ›

Volkan Cirik → Denny Britz • a year ago

but none of them is 6dimensional it shows (4,3,2) for (2,3,4) respectively. It should be (6,5,4) as you say.

• Reply • Share ›

Denny Britz Mod → Volkan Cirik • a year ago

In the second column, the last filter at the very bottom is region size 2 (a 2x5 filter), and it produces a 6-dimensional feature map, which is the 3rd column. Or are we talking about different things?

• Reply • Share ›

Volkan Cirik → Denny Britz • a year ago

3rd column is okay but not the 2nd one. that's why it is confusing.

• Reply • Share ›

lufo816 • a year ago

Fantastic!

• Reply • Share ›

heynow88 • a year ago

I just stumbled into this. This is very intriguing and interesting. I wonder if you've come across a larger scale conceptual understanding of the entire scope of

textual recognition, rather than character to character, or word to word, etc. I think a key factor is isolating differences of between the objects, persons, places, things, groups, etc. and grouping them, while including time, including relationships (different types) and somehow taking from all this a way of establishing an understanding of the words between the objects. There are innumerable categories of things, with infinitesimal relationships (differences and similarities) between each category and items within that category. Then each objects has it own library (or history) of relationships and that has to be reviewable, etc. I consider more of how do I found a concept rather than search a key word. Further, I really hate operating systems' current (in)ability to analyze unstructured data and there is no easy way (without adding everything into a database) to simply tag or categorize everything. It's just too impossible when dealing with millions of documents. There must be a way to fix this and it should show you trends as they are happening. Which sort of depends on a relationship/graph type model as well. Have you thought of these issues much?

Reply • Share ›

Denny Britz Mod → heynow88 • a year ago

There's a field called Knowledge Base Construction (KBC) that tries to automatically extract concepts (entities) and relationships (relations) from text, sometimes with time, and puts them into a graph structure (knowledge graph). NELL [1] or Google's Knowledge Graph/Vault [2] are examples of systems like that. Google has been moving into that direction more and more. If you type a question into it, something like "what is the official language of Angola?" it will directly give you the answer, which is part of their knowledge graph connected to the "Angola" concept. And it'll give you structured information about Angola on the right side of the search results. There's a lot of research in KBC going on, some approaches are supervised, some unsupervised. We can do pretty well in closed domains, like extracting specific relations/entities from medical journals for example, but we're very far away from being able to extract very generic types of knowledge from any text.

[1] http://rtw.ml.cmu.edu/rtw/
[2] http://www.cs.ubc.ca/~murph...

Reply • Share ›

haha • a year ago

Your posts are really really great!

- CONNECT -

- RECENT POSTS -

Learning Reinforcement Learning (with Code, Exercises and Solutions)

RNNs in Tensorflow, a Practical Guide and Undocumented Features

Deep Learning for Chatbots, Part 2 – Implementing a Retrieval-Based Model in Tensorflow

Deep Learning for Chatbots, Part 1 – Introduction

Attention and Memory in Deep Learning and NLP

Implementing a CNN for Text Classification in TensorFlow

Understanding Convolutional Neural Networks for NLP

Recurrent Neural Network Tutorial, Part 4 – Implementing a GRU/LSTM RNN with Python and Theano

- LINKS -

Home

About

NLP Consulting

- ARCHIVES -

October 2016

August 2016

July 2016

April 2016

January 2016

December 2015

November 2015

October 2015

September 2015

- CATEGORIES -

Conversational Agents

Convolutional Neural Networks

Deep Learning

GPU

Language Modeling

Memory

Neural Networks

NLP

Recurrent Neural Networks

Reinforcement Learning

RNNs

Tensorflow

- SUBSCRIBE TO BLOG VIA EMAIL -

Enter your email address to subscribe to this blog and receive notifications of new posts by email.

Email Address

SUBSCRIBE

- META -

Log in

Entries RSS

Comments RSS

WordPress.org

PROUDLY POWERED BY WORDPRESS | THEME: SELA BY WORDPRESS.COM.