Introduction to NLP

By Russell Jurney <rjurney@graphlet.ai>

Statistical Natural Language Processing

An empiricist approach to NLP suggests that we can learn the complicated and extensive structure by specifying an appropriate general language model, and then inducing the values of parameters by applying statistical, pattern recognition and machine learning methods to a large amount of language use.

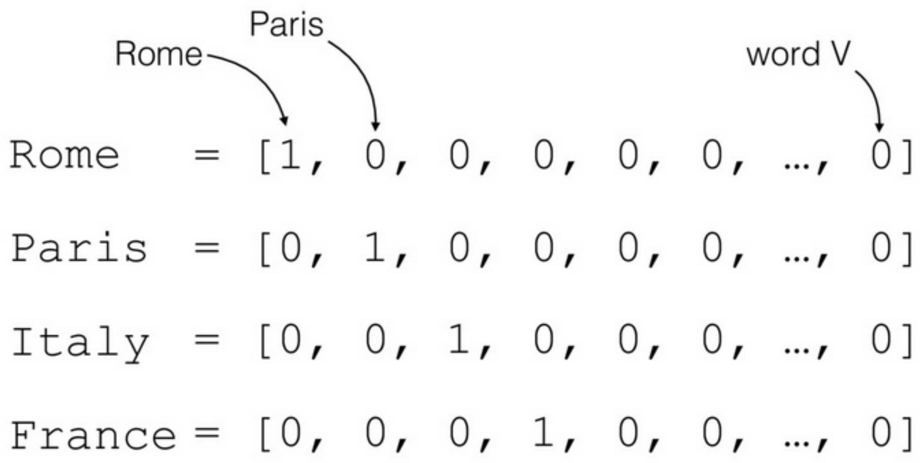
—Foundations of Natural Language Processing (1997)

Statistical Natural Language Processing (NLP) is a field in which the structure of language is analyzed using statistical methods to learn a language model describing the detailed structure of natural language [Manning, Schütze, 1999]. Such statistical language models are used to automate the processing of text data in tasks including parsing sentences to extract their grammatical structure, extracting entities from documents, classifying documents into categories, ranking documents numerically, summarizing documents, answering questions, translating documents from one language to another and others.

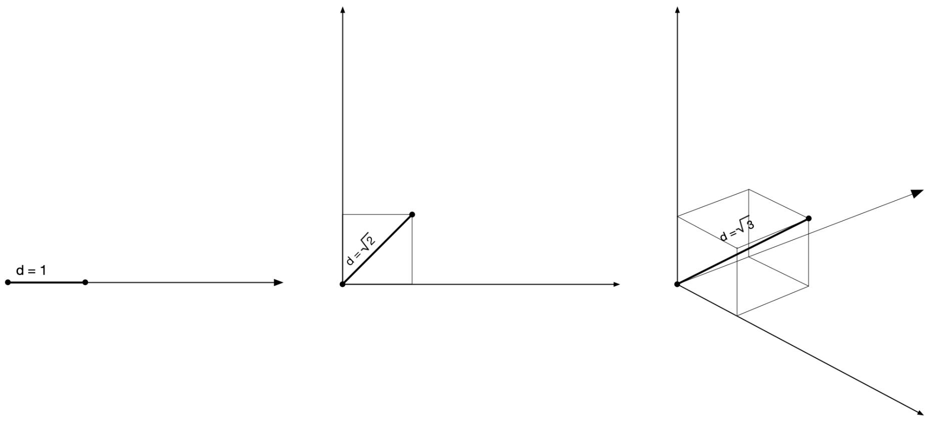
This introduction does not contain an introduction to natural language processing. For that I recommend *Foundations of Natural Language Processing* by Christopher D. Manning and Hinrich Schütze for a theoretical introduction and *Natural Language Processing with Python* for more practical skills. What I will do is to offer a brief explanation of the role of various network models in this book.

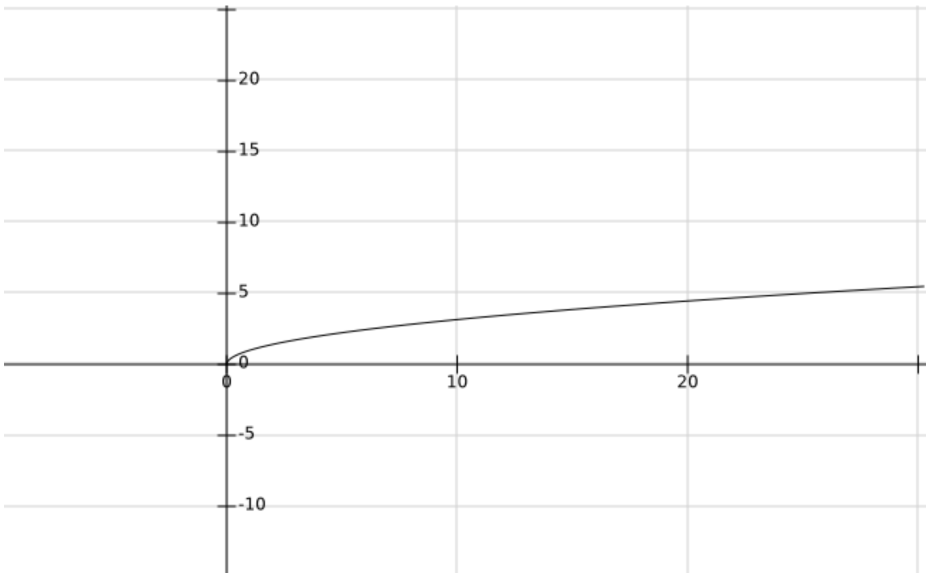
Representing Text

Before the application of neural networks to language modeling, bag of words representations were used to convert from documents made up of words to matrix representations suitable for machine learning algorithms. Le and Mikolov (2013) noted that bag of words feature encoding loses word ordering as well as the semantics of words. They also produced very sparse matrices. Before neural architectures were applied to NLP, "core NLP techniques were dominated by machine-learning approaches that used linear models such as support vector machines or logistic regression, trained over very high dimensional yet very sparse feature vectors." [Goldberg, 2015].


Paris
Rome
word V
Rome = [1, 0, 0, 0, 0, 0, ..., 0]
Paris = [0, 1, 0, 0, 0, 0, ..., 0]
Italy = [0, 0, 1, 0, 0, 0, ..., 0]
France = [0, 0, 0, 1, 0, 0, ..., 0]

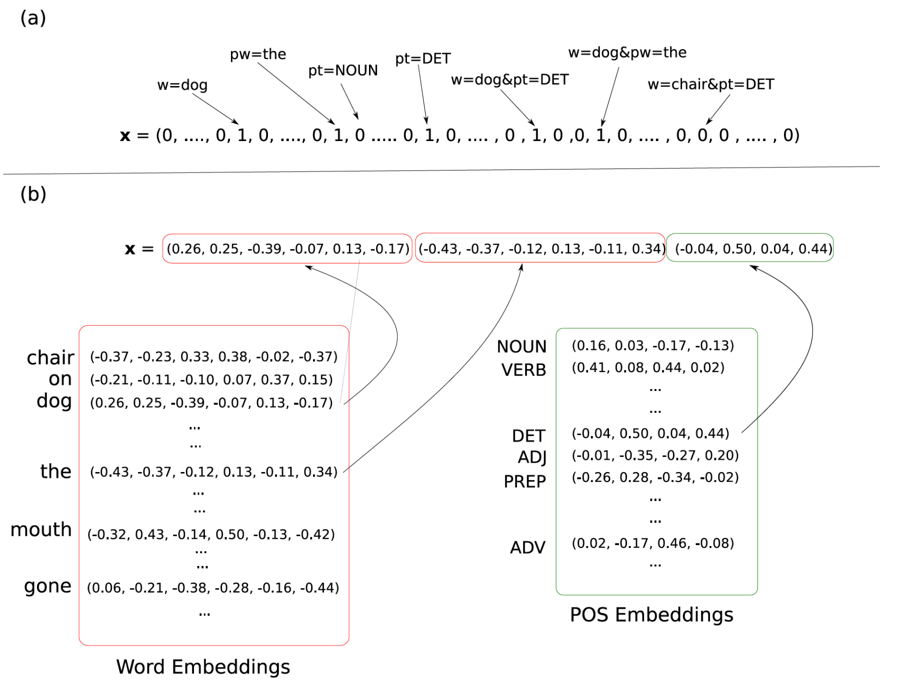
The primary challenge in the performance of these models was the *curse of dimensionality*, under which bag-of-words representations transformed to matrices via one-hot-encoding suffered from statistical insignificance across the deeply nested, high dimensional space in which they were encoded. Pairs of points once divergent become equidistant as more and more dimensions are added, so records appear more similar and there is not enough signal for an algorithm to work well [Theodoridis, Koutroumbas, 2008]. This limited the performance of models across much of the field of natural language processing.





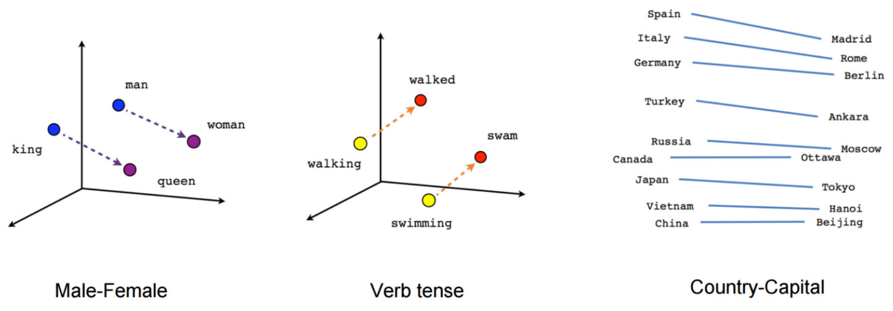
Plot of function sqrt(x), source: fooplot.com

In 2003, the paper A Neural Probabilistic Language Model [Bengio, Ducharme, Vincent, Jauvin, 2003] demonstrated superior performance on several common NLP tasks using a distributed representation for words. A decade later, the rise of dense representations in the form of text embeddings such as Word2Vec [Le, Mikolov, 2013] accelerated the development of DL methods for NLP by offering an encoding method who's output didn't suffer from the curse of dimensionality because it encoded each word in a space with a latent semantic space reduced number of dimensions. Text embeddings changed text encoding from a list of bits identifying the presence of words in a document under the bag-of-words model to a dense representation that describes the semantics of each word in terms of its position in a vector space where each dimension corresponds to a particular meaning Tardy, 2017. Neural networks work better with dense than with sparse representations because they take advantage of the complex signal in the underlying language this encoding exposes. The chart below shows the difference between sparse and dense text feature representations.


(a)
w=dog
pw=the
pt=NOUN
pt=DET
w=dog&pt=DET
w=dog&pw=the
w=chair&pt=DET
x = (0, ..., 0, 1, 0, ..., 0, 1, 0 ..... 0, 1, 0, ... , 0 , 1, 0 ,0, 1, 0, ... , 0, 0, 0 , ... , 0)
(b)
x = (0.26, 0.25, -0.39, -0.07, 0.13, -0.17) (-0.43, -0.37, -0.12, 0.13, -0.11, 0.34) (-0.04, 0.50, 0.04, 0.44)
chair (-0.37, -0.23, 0.33, 0.38, -0.02, -0.37)
on (-0.21, -0.11, -0.10, 0.07, 0.37, 0.15)
dog (0.26, 0.25, -0.39, -0.07, 0.13, -0.17)
the (-0.43, -0.37, -0.12, 0.13, -0.11, 0.34)
mouth (-0.32, 0.43, -0.14, 0.50, -0.13, -0.42)
gone (0.06, -0.21, -0.38, -0.28, -0.16, -0.44)
Word Embeddings
NOUN (0.16, 0.03, -0.17, -0.13)
VERB (0.41, 0.08, 0.44, 0.02)
DET (-0.04, 0.50, 0.04, 0.44)
ADJ (-0.01, -0.35, -0.27, 0.20)
PREP (-0.26, 0.28, -0.34, -0.02)
ADV (0.02, -0.17, 0.46, -0.08)
POS Embeddings

Sparse vs. dense text encoding (Goldberg, 2015)

Word and character embeddings are the most common method of feature encoding for deep networks using text. Since Word2Vec many word embedding methods have emerged such as GloVe and Elmo. They appear as the first step of most networks in the field of NLP. We'll be using them throughout the book. We'll also use text and program source code embeddings during the chapter on *transfer learning*.



Source: Embeddings: Translating to a Lower-Dimensional Space at developers.google.com

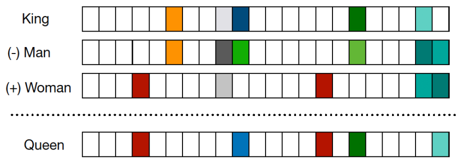


Fig. 2: Distributional vectors represented by a **D**-dimensional vector where $\mathbf{D} << \mathbf{V}$, where **V** is size of Vocabulary. Figure Source: http://veredshwartz.blogspot.sg.

Transfer Learning

Transfer learning uses a model trained for one task to perform another, related task. This enables the rapid creation of high performance models without an abundance of labeled training data. Transfer learning handles the encoding of the data in a meaningful representation at the start of your network, and then additional layers encode the specifics of your problem.

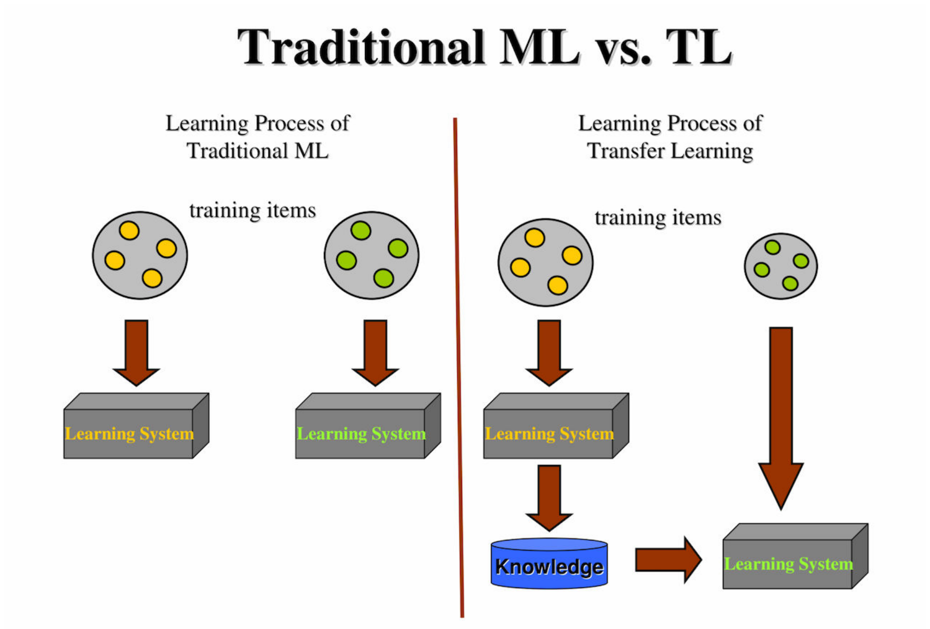


Figure 1.1: Traditional Machine Learning vs. Transfer Learning

Farajidavar, N; Transductive Transfer Learning for Computer Vision, 2015

In addition to acquiring sufficient labeled data for each problem your business wants to solve, there is the need to develop sophisticated representations of text to drive downstream tasks. Language models are commonly developed using unsupervised learning on massive text corpuses to associate words with one another based on the context of nearby words. In deep learning for natural language processing, transfer learning takes two forms: static embeddings or pre-trained networks.

A static embedding is a lookup table that maps each character or word to its meaning in the entire corpus across anywhere from 10 to 1024 semantic dimensions. A pre-trained network contains the structure and weights of an entire neural network that determines a word's semantics based on the context in which it occurs within a given sentence. These models form the top of a neural network with additional layers underneath connected to the labeled data and that is additionally trained or tuned using supervised learning with labeled data specific to each task.

Sophisticated language models are most commonly created by large research organizations with the sophistication to do breakthrough research and the resources to work with truly massive datasets and clusters of many computers, each with multiple GPUs or TPUs: all the text on the web, all news articles for many years or the content of all the pages linked to on Reddit. Examples of these models abound, are constantly evolving and are often named after characters from Sesame Street: GloVe (Stanford), ULMFiT (Fast.ai, University of Galway), ELMo (AllenNLP), BERT (Google), and GPT and it's successor GPT-2 (OpenAI). Thus far there is a strong tradition of open publishing in the field of deep learning and natural language processing, so these models are freely available for anyone to use.

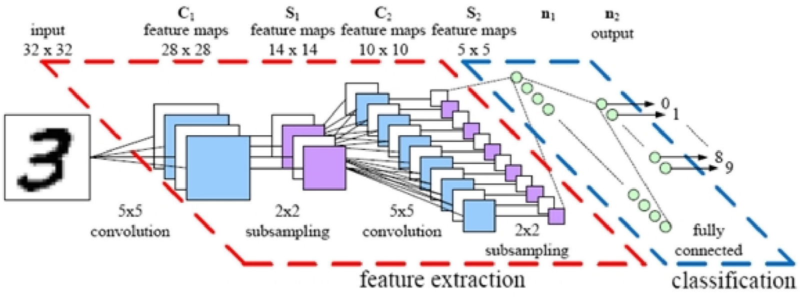


Figure 2.3: Convolutional Neural network structure (©Torch7 documentation [27]): A CNN is a trainable architecture composed of multiple convolution and pooling stages.

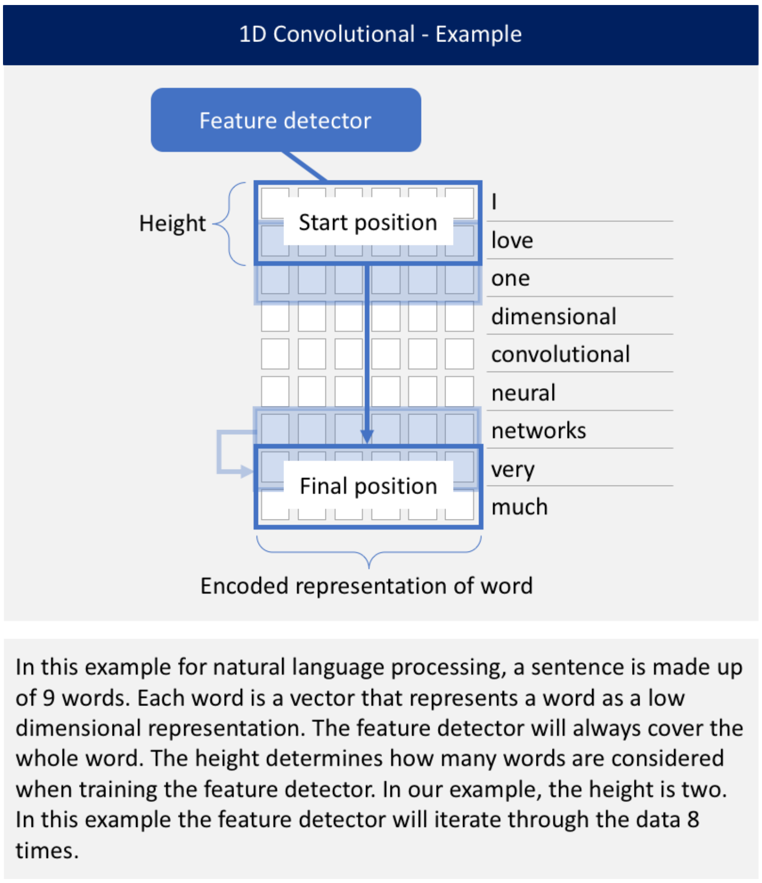
The top of a CNN architecture uses convolution/max pool layers to learn a representation of the key features of the data while one or more fully connected layers learn to perform classification. Soure: Transductive Transfer Learning for Computer Vision, Farajidavar, N, 2015

If another pre-trained model handles learning the data's representation, it takes far less labeled data to train a model for a specific task. If we don't have to train a model to extract features into an effective representation, then labeling data becomes the primary bottleneck for building and deploying AI. In combination with *weak supervision* and *distant supervision*, we'll use transfer learning to rapidly build applications using very little hand labeled training data. This book's coverage of transfer learning is organized around this use case and won't cover every form of transfer learning.

Convolutional Neural Networks

One dimensional Convolutional Neural Networks (CNNs) are used for NLP tasks where local features are sufficient and it doesn't matter where in the document they appear in relation to one another. Depending on the kernel size this may involve n-grams of different size to characterize features of different size. "CNNs have the ability to extract salient n-gram features from the input sentence to create an informative latent semantic representation of the sentence for downstream tasks." [Young, Hazarik, Pori, Cambri, 2018].

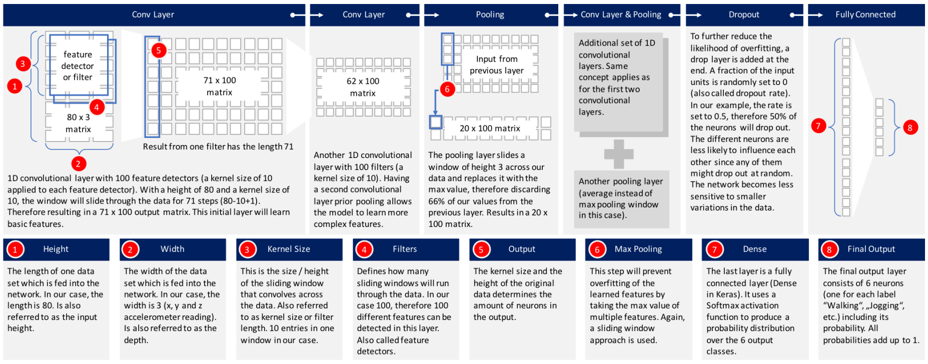
The image below by Nils Ackerman shows how 1 dimensional (1D) convolutions over text work. In the diagram, the width of the matrix stores the semantic dimensions of the embedding, which represent the meaning of each word. The height represents the sequence of individual words in a document. The meaning of each word is then inspected by the feature detector - the kernel (in this case of size 2) - that scans more than one word at a time and combines the meaning of words near one another via a linear convolution into a new semantic summary of the words in that window. The larger the kernel size - the more words are considered for each feature, the higher level semantics that are summarized. Multiple kernel sizes are often used and there are multiple random filter values for each kernel size.



In this example for natural language processing, a sentence is made up of 9 words. Each word is a vector that represents a word as a low dimensional representation. The feature detector will always cover the whole word. The height determines how many words are considered when training the feature detector. In our example, the height is two. In this example the feature detector will iterate through the data 8 times.

"1D CNN Example" by Nils Ackerman licensed under Creative Commons CC BY-ND 4.0

Note that the CNN model has no idea what the specific relationship between adjacent words is. Recurrent Neural Networks model the state of the sequence - for example a sentence - and remember previous words to understand context. Convolutional Neural Networks do not. They simply perform a linear convolution over the size of the kernel using random weights in the filter to identify different aspects of the semantics of the words in each window. With a kernel size of 2, they do not remember that 'hot' preceded 'dog' in 'hot dog.' They rely on the embedding for 'dog' containing both the animal and carnival food meanings. For tasks that rely on stateful parsing of text like language models, translation and text generation, RNNs work best. For tasks that rely on identifying key features in the text regardless of their position in the document - like document classification - CNNs work best. Note that competitive architectures for many tasks exist in either technique but 'works best' indicates what is the most natural, direct approach for these tasks.



Result from one filter has the length 71

1D convolutional layer with 100 feature detectors (a kernel size of 10 applied to each feature detector). With a height of 80 and a kernel size of 10, the window will slide through the data for 71 steps (80-10+1). Therefore resulting in a 71 x 100 output matrix. This initial layer will learn basic features.

Another 1D convolutional layer with 100 filters (a kernel size of 10). Having a second convolutional layer prior pooling allows the model to learn more complex features.

The pooling layer slides a window of height 3 across our data and replaces it with the max value, therefore discarding 66% of our values from the previous layer. Results in a 20 x 100 matrix.

Additional set of 1D convolutional layers. Same concept applies as for the first two convolutional layers.

Another pooling layer (average instead of max pooling window in this case).

To further reduce the likelihood of overfitting, a drop layer is added at the end. A fraction of the input units is randomly set to 0 (also called dropout rate). In our example, the rate is set to 0.5, therefore 50% of the neurons will drop out. The different neurons are less likely to influence each other since any of them might drop out at random. The network becomes less sensitive to smaller variations in the data.

Height: The length of one data set which is fed into the network. In our case, the length is 80. Is also referred to as the input height.

Width: The width of the data set which is fed into the network. In our case, the width is 3 (x, y and z accelerometer reading). Is also referred to as the depth.

Kernel Size: This is the size / height of the sliding window that convolves across the data. Also referred to as kernel size or filter length. 10 entries in one window in our case.

Filters: Defines how many sliding windows will run through the data. In our case 100, therefore 100 different features can be detected in this layer. Also called feature detectors.

Output: The kernel size and the height of the original data determines the amount of neurons in the output.

Max Pooling: This step will prevent overfitting of the learned features by taking the max value of multiple features. Again, a sliding window approach is used.

Dense: The last layer is a fully connected layer (Dense in Keras). It uses a Softmax activation function to produce a probability distribution over the 6 output classes.

Final Output: The final output layer consists of 6 neurons (one for each label "Walking", „Jogging", etc.) including its probability. All probabilities add up to 1.

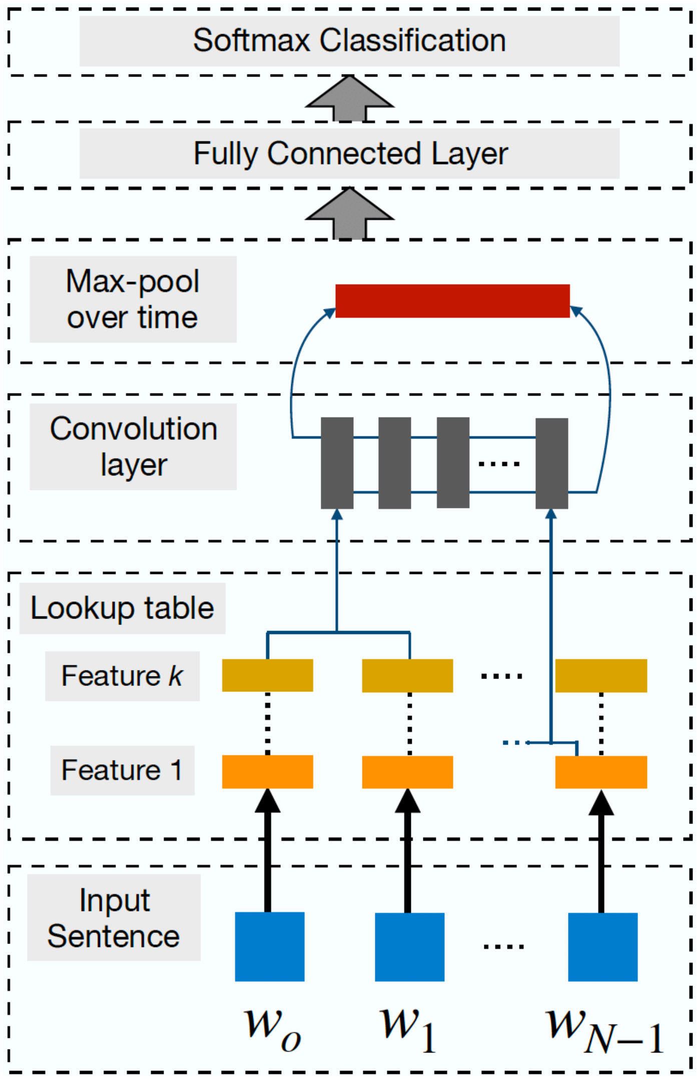
"1D CNN Example" by Nils Ackerman licensed under Creative Commons CC BY-ND 4.0

Convolutional layers use randomly initialized kernels that result in their describing different features among different regions of the text in combination with pooling layers, which identify the most important features detected by the convolution layer that precedes them. Max pooling is a common method of pooling and simply selects the strongest feature - a semantic summary of one aspect of a window of text - by taking the maximum value in a given window of summaries. When combined with multiple random kernel filter values, this has the effect of extracting the strongest features of the different semantic dimensions of each window of text. The combination of convolutional and max pooling layers learn the features of the documents most essential to the task for which we are training. Stacked instances of convolutional/max pooling layers summarize and extract higher level features, enabling effective representations for tasks like document classification.

> Firstly, max pooling provides a fixed-length output which is generally required for classification. Thus, regardless the size of the filters, max pooling always maps the input to a fixed dimension of outputs. Secondly, it reduces the output's dimensionality while keeping the most salient n-gram features across the whole sentence. This is done in a translation invariant manner where each filter is now able to extract a particular feature (e.g., negations) from anywhere in the sentence and add it to the final sentence representation.

— Recent Trends in Deep Learning Based Natural Language Processing by Young, Hazarik, Pori, Cambri (2018)

The most common application of 1D CNNs is document classification. As we'll see when we investigate the workings of our convolutional models using LIME and SHAP, the most important signal for classifying documents are marker words and phrases common to documents of a given type and infrequent in the corpus overall. The diagram below shows how sentences of tokens are mapped to dense representations using a text embedding lookup table and then run through a series of convolution and max pooling layers to extract the most salient features for a fully connected layer that performs the mapping of the features into a softmax output layer corresponding to the classes into which we are classifying.

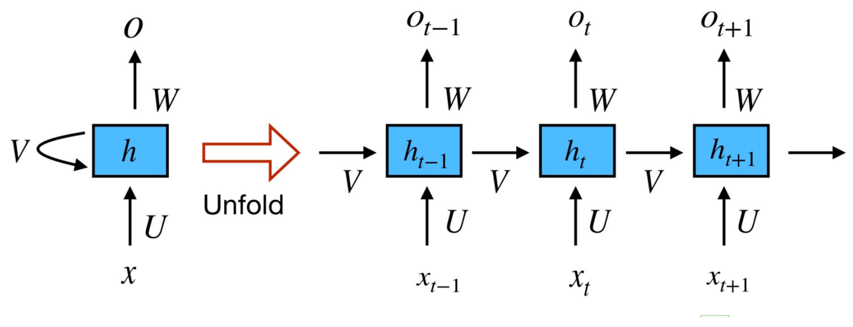

Softmax Classification
Fully Connected Layer
Max-pool over time
Convolution layer
....
Lookup table
Feature *k*
Feature 1
....
Input Sentence
....
w_o
w_1
w_{N-1}

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) and Long Short Term Memory (LSTM) cells are used for NLP tasks where the model needs to be aware of a broader context within a sequence of words which is stored as internal state in each neuron and referenced in each output decision. In other words, the structure of a recurrent neural network matches the structure of a sentence: items in a sequence that impact other items in the sequence. The fact that the word 'hot' precedes 'dog' impacts the meaning of the sequence 'hot dog.' This is something CNNs can't do and enables RNNs to perform on for tasks where temporal behavior is exhibited.

RNNs became popular in 2014/15, as Andrej Karpathy documented in his blog post in May of 2015, The Unreasonable Effectiveness of Recurrent Neural Networks:

> There's something magical about Recurrent Neural Networks (RNNs). I still remember when I trained my first recurrent network for Image Captioning . Within a few dozen minutes of training my first baby model (with rather arbitrarily-chosen hyperparameters) started to generate very nice looking descriptions of images that were on the edge of making sense. Sometimes the ratio of how simple your model is to the quality of the results you get out of it blows past your expectations, and this was one of those times. What made this result so shocking at the time was that the common wisdom was that RNNs were supposed to be difficult to train (with more experience I've in fact reached the opposite conclusion). Fast forward about a year: I'm training RNNs all the time and I've witnessed their power and robustness many times, and yet their magical outputs still find ways of amusing me.
>
> — The Unreasonable Effectiveness of Recurrent Neural Networks, Andrej Karpathy, May, 2015



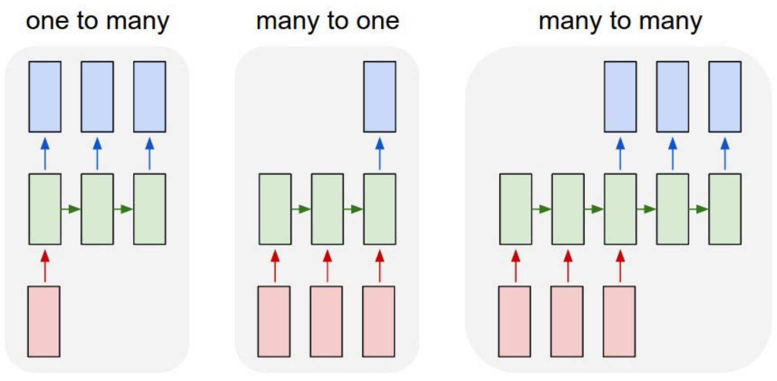
The unraveling of a simple RNN network into the components of its state as it encounters subsequent tokens, Young et al., 2018

How do RNNs compare with CNNs? For starters, CNNs operate on a fixed sequence length while RNNs are able to operate on variable length sequences because they operate on a different basis in which :

> The term "recurrent" applies as they perform the same task over each instance of the sequence such that the output is dependent on the previous computations and results. Generally, a fixed-size vector is produced to represent a sequence by feeding tokens one by one to a recurrent unit. In a way, RNNs have "memory" over previous computations and use this information in current processing.
>
> …
>
> Both CNNs and RNNs have different objectives when modeling a sentence. While RNNs try to create a composition of an arbitrarily long sentence along with unbounded context, CNNs try to extract the most important n-grams. Although they prove an effective way to capture n-gram features, which is approximately sufficient in certain sentence classification tasks, their sensitivity to word order is restricted locally and long-term dependencies are typically ignored.
>
> — Recent Trends in Deep Learning Based Natural Language Processing, Young et al, 2018

RNNs come in three main forms that depend on the data format of the task involved:

- One to many - one input and a sequence of outputs. An example is an RNN that creates a text description for a single image.
- Many to one - a sequence of inputs and a single output. An RNN binary classifier is an example of a many to one RNN in that a sequence of multiple elements is used to predict a single output class.
- Many to many - an encoder-decoder is an RNN that maps from a sequence to a sequence or seq-to-seq. A translation model is an example of a many too many RNN model.



The Unreasonable Effectiveness of RNNs: The Unreasonable Effectiveness of Recurrent Neural Networks, Karpathy, Andrej, May, 2015

The Limitations of Recurrent Neural Networks

RNNs are, like other neural networks, trained through *back propagation*. The features of an example are fed through the network along with its labeled results fitted to the end. Then at each layer, starting with the last one and moving backwards, the differ-ence between the layer's actual output and its expected output is used to calculate an error, the derivative of which is used to indicate the direction in which to update the weights for neurons in that layer. Because they model sequences with internal knowl-edge at each step about the previous elements, RNNs are trained through a technique called *back propagation through time (BPTT).* In order to store knowledge in a general way across elements and to be computationally feasible, RNNs share parameters across layers.

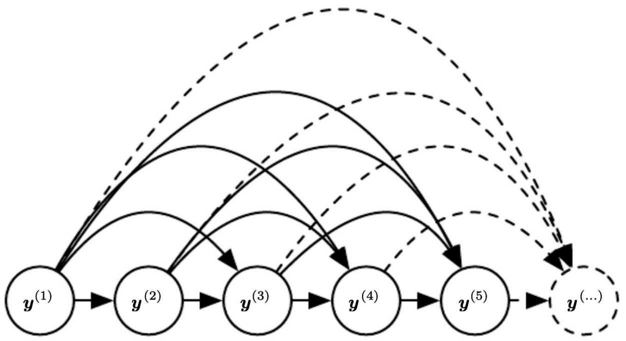


Figure 10.7: Fully connected graphical model for a sequence $y^{(1)}, y^{(2)}, \ldots, y^{(t)}, \ldots$. Every past observation $y^{(i)}$ may influence the conditional distribution of some $y^{(t)}$ (for $t > i$), given the previous values. Parametrizing the graphical model directly according to this graph (as in equation 10.6) might be very inefficient, with an ever growing number of inputs and parameters for each element of the sequence. RNNs obtain the same full connectivity but efficient parametrization, as illustrated in figure 10.8.

The computational graph of a Recurrent Neural Network without shared parameters would have many dependencies. Goodfellow et al. 2017.

The problem with Recurrent Neural Networks is that they are deep and have short term memory. Which is to say that because they span both layers in the network architecture and the time-steps of the sequences they model, they are very deep when unraveled. This depth means there is much more back propagation that must occur via BPTT to model a sequence in an RNN than there are to model a fixed length vector in a CNN via normal back propagation. Is is the depth of RNNs combined with parameter sharing that results in what is called the *vanishing gradient problem.* Typical activation functions have gradients ranging between -1 and 1, which means the update to the weights of each layer may be small. Since each layer moving backwards is dependent upon the previous layer's update of the same parameter, gradients may vanish as feedback becomes smaller and smaller as it propagates backwards through the network. In this case, the network stops learning.

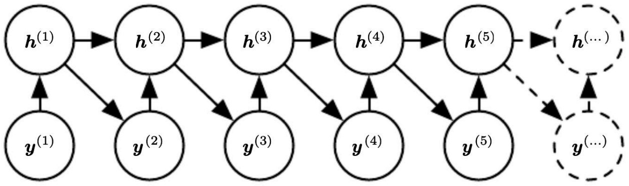


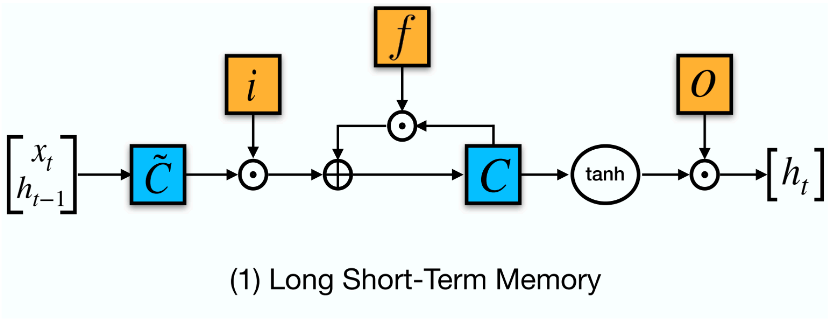
Figure 10.8: Introducing the state variable in the graphical model of the RNN, even though it is a deterministic function of its inputs, helps to see how we can obtain a very efficient parametrization, based on equation 10.5. Every stage in the sequence (for $\boldsymbol{h}^{(t)}$ and $\boldsymbol{y}^{(t)}$) involves the same structure (the same number of inputs for each node) and can share the same parameters with the other stages.

Recurrent Neural Networks simplify the computational graph by sharing parameters between items in a sequence in memory. This simplifies their computation but introduces the vanishing gradient problem. Goodfellow et al, 2017.

Architectures like Long Short Term Memory networks (LSTMs) were created to overcome the problem of unstable gradients - both vanishing and iexploding.

Long Short Term Memory Networks

Long Short Term Memory networks (LSTMs) improve upon the original RNN model by adding additional mechanics to the way memory works to enable longer term memories to be retained across arbitrary gaps in sequence. It does so by splitting the operations governing its retained state into input gates, forget gates and output gates. The input gate works similarly to an RNN, deciding how to update the cell's memory. The forget gate compares the previous state and the previous cell's output to determine what it is safe to forget from memory. If the previous state did not affect the output, it is no longer significant. The output gate uses input and the memory block to decide what to output to the next cell.

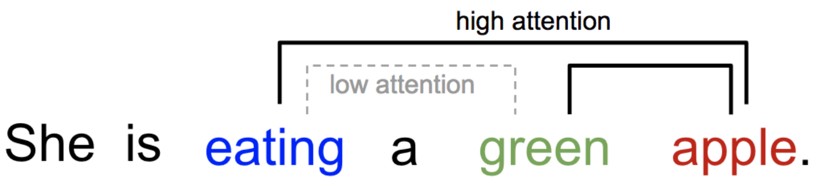


LSTMs improve memory within cells by splitting the retention of state between three gates: an input gate, a forget gate and an output gate. Young et al, 2018.

The result is that LSTMs remember relevant information for longer periods of time because their default behavior is for cell state to move from one cell to the next. This behavior is regulated by the three gates such that the important information is retained. This makes LSTMs and their variants ideal for tasks needing to understand longer term dependencies within documents: language modeling, translation, text summarization and text generation.

Attention Mechanisms in Recurrent Neural Networks

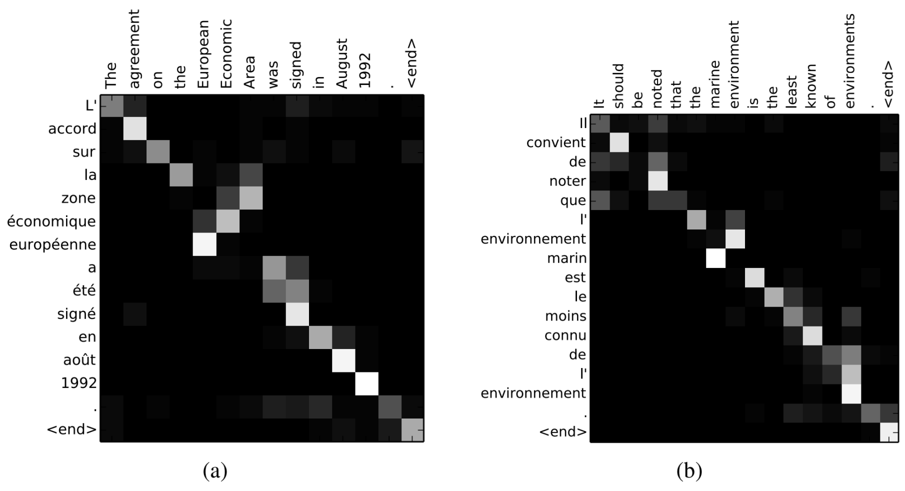
Recurrent Neural Networks rely on a single store of state that is propagated across time in a simple (RNN) or complex (LSTM, GRU) way as they walk across the input sequence. If a part of that state is down-regulated by a particular item or set of nearby items in a sequence, it will disappear moving forward. If the topic of the document is mentioned early on and this has a strong relationship with words that appear late in the document… an RNN doesn't model this relationship well. What is needed is an attention mechanism that can pay attention to important words longer than unimportant words.



Attention between words depends on not just on their distance via a recurrent state but dynamically based on how related the words are. Weng, Lilian, Attention? Attention!, Jun, 2018.

By contrast, attention networks have an awareness of the relationships between the words that occur in a give sequence. They maintain a lookup table of weights defining the relationships between words that are used to dynamically alter the cell's state. This enables dynamic attention as the context of the sequence develops over much longer spans than traditional RNNs can handle - the entire document [Bahdanau et al, 2016].

> The most important distinguishing feature of this approach from the basic encoder–decoder is that it does not attempt to encode a whole input sentence into a single fixed-length vector. Instead, it en-codes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation. This frees a neural translation model from having to squash all the information of a source sentence, regardless of its length, into a fixed-length vector. We show this allows a model to cope better with long sentences.
>
> — Bahdanau, D; Cho, K; Bengio, Y; Neural Machine Translation by Jointly Learning to Align and Translate, 2016.

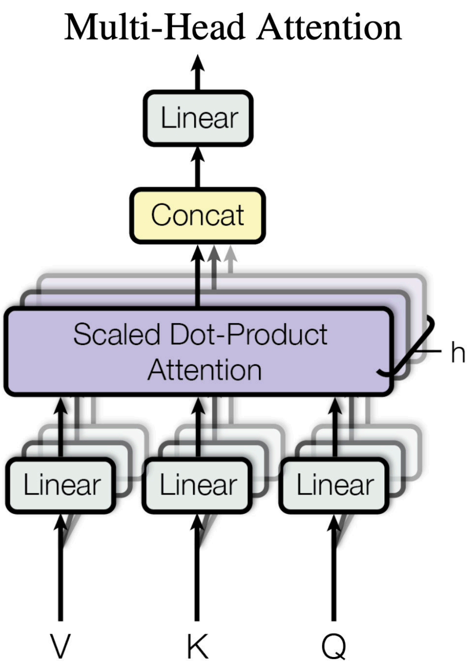


Visualizations of the attention mapping between words from both sides of an English/French sequence-to-sequence translation model, Bahdanau, D; Cho, K; Bengio, Y; Neural Machine Translation by Jointly Learning to Align and Translate, Jacobs University, Bremen, Germany and Université de Montréal, 2016.

Transformers

Transformers are a type of network architecture that do away with recurrence and use a self-attention mechanism directly to model language. This makes them much more efficient to train, which enables more complex language models using more data for less computation cost [Vaswani et al, 2017]. They are now state of the art for tasks like translation and text generation.

> In contrast [to RNNs], the Transformer only performs a small, constant number of steps (chosen empirically). In each step, it applies a self-attention mechanism which directly models relationships between all words in a sentence, regardless of their respective position. In the earlier example "I arrived at the bank after crossing the river", to determine that the word "bank" refers to the shore of a river and not a financial institution, the Transformer can learn to immediately attend to the word "river" and make this decision in a single step. In fact, in our English-French translation model we observe exactly this behavior.
>
> — Google AI Blog: Transformer: A Novel Neural Network Architecture for Language Understanding



"Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions."

—https://arxiv.org/abs/ 1706.03762[Attention Is All You Need], Vaswani et al, 2017.

Transformers for Language Modeling

OpenAI used Transformers to create a language models in the form of GPT via unsupervised pre-training that are then fine-tuned for each specific downstream task [Radford et al, 2018].

> Our system works in two stages; first we train a transformer model on a very large amount of data in an unsupervised manner — using language modeling as a training signal — then we fine-tune this model on much smaller supervised datasets to help it solve specific tasks. […] Here, we wanted to further explore this idea: can we develop one model, train it in an unsupervised way on a large amount of data, and then fine-tune the model to achieve good performance on many different tasks? Our results indicate that this approach works surprisingly well; the same core model can be fine-tuned for very different tasks with minimal adaptation.
>
> — Improving Language Understanding with Unsupervised Learning

OpenAI made a lot of waves when it delayed releasing the dataset behind GPT-2, citing a concern in the OpenAI blog post Better Language Models and Their Implications with the ease of use for which it could be used to generate fake content for fraud and abuse. Transformers worked so well for language modeling the sudden jump in performance was felt to represent a threat to public safety. Transformer based language models are now used in many state of the art systems for a variety of tasks.