Lecture 5: Sequential data

Peter Bloem, David Romero

Deep Learning 2020

dlvu.github.io

VRIJE UNIVERSITEIT AMSTERDAM

OUTLINE

part one: Learning from sequences

part two: RNNs

part three: LSTMs

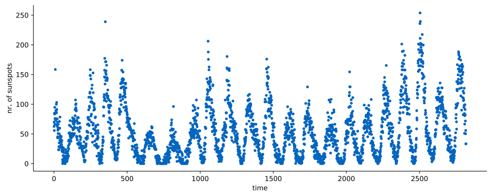
part four: CNNs for sequential data

part five: ELMo, a case study

PART ONE: LEARNING FROM SEQUENCES

In the first part we'll look at the technical details of setting up a sequence learning problem. How should we prepare our data, represent it as a tensor, and what do sequence-based models look like in deep learning systems?

NUMERIC 1-DIMENSIONAL

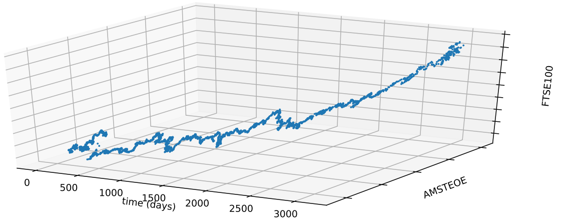


Before we start looking at different models to learn from sequential data, it pays to think a little bit about the different types of sequential datasets we might encounter.

As with the traditional, tabular setting, we can divide our features into numeric and discrete.

A single 1D sequence might look like this. We could think of this as a stock price over time, traffic to a webserver, or atmospheric pressure over Amsterdam. In this case, the data shows the number of sunspots observed over time.

NUMERIC N-DIMENSIONAL



Sequential numeric data can also be multidimensional. In this case, we see the closing index of the AEX and the FTSE100 over time. This data is a sequence of 2D vectors.

SYMBOLIC (CATEGORICAL)

the, cat, sat, on, the, mat

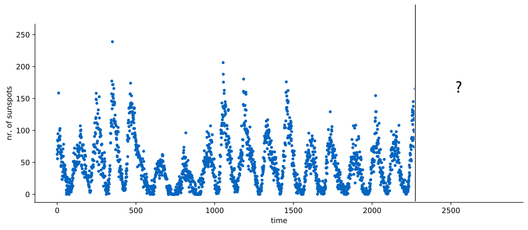
t, h, e, _, c, a, t, _, s, a, t, _, o, n, _, t, h, e, _, m, a, t

If the elements of our data are discrete (analogous to a categorical feature), it becomes a sequence of symbols. Language is a prime example. In fact, we can model language as a sequence in two different ways: as a sequence of words, or as a sequence of characters.

SINGLE SEQUENCE VS SET OF SEQUENCES

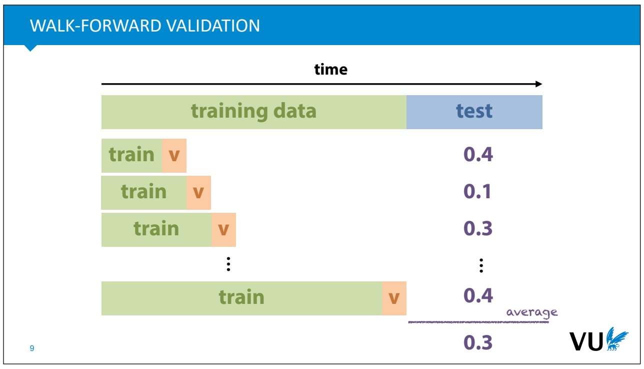
Dear recipient, Avangar Technologies announces the beginning of a new unprecendented global employment campaign ... reviser yeller winers butchery twenties Due to company's exploding growth Avangar is expanding business to the European region. During last employment campaign over 1500 people worldwide took part in Avangar's business and more than half of them are currently employed by the company. And now we are offering you one more opportunity to earn extra money working with Avangar Technologies. druggists blame classy gentry Aladdin	spam
BENCHMARK SUPPLY 7540 BRIDGEGATE COURT ATLANTA GA 30350 ***LASER PRINTER TONER CARTRIDGES*** ***FAX AND COPIER TONER*** CHECK OUT OUR NEW CARTRIDGE PRICES :	spam
On 08/08/02 15:26 -0700, Chip Paswater wrote:) > Well, a little more than one bit -- you have to transmit the signature, plus now the entire ... ID.) > Plus the rights to all intellectual property contained in any) > email message you submit (I guess technically there are no bits) > in the rights assignment))) Perhaps a feature can be added to razor-report, so that it checks whether a	ham
If you own a travel related website, why not submit your site to our directory. Just select the appropriate category and subcategory and enter your title ... description. Click here to start: http://www.holprop-travel-directory.com This footnote confirms that this email message has been swept by Anti-Virus	spam
| hmmm, I assume you're going to report this to the nmh folks? Yes, I will, sometime, after I look at the nmh sources and see what they have managed to break, ...why. But we really want exmh to operate with all the versions of nmh that exist, don't we? The patch to have exmh do the right thing, whether this bug exists, or not, is trivial, so I'd suggest including it.	ham
Hello I am emailing you as we found your holiday property at [Source] and would love you to list it on our website for free: http://www.holidayrentals.org This web site now has over 1000 holiday rentals listed in less than 2 months, so please register and then add as many properties as you wish. All for free forever	spam
Hello again self-catering.co.uk was recently launched and if you have not yet added your property for free then please do as we have had many properties added in the past few days.	spam

SINGLE SEQUENCE



An entirely different setting is one where the dataset as a whole is a sequence, and the instances are ordered.

In that case, we often want to predict the future values of the sequence based on what we've seen in the past. To keep things simple, let's stick with 1D numeric data, like our sunspots example.



WALK-FORWARD VALIDATION

time

training data | test

train v 0.4

train v 0.1

train v 0.3

⋮ ⋮

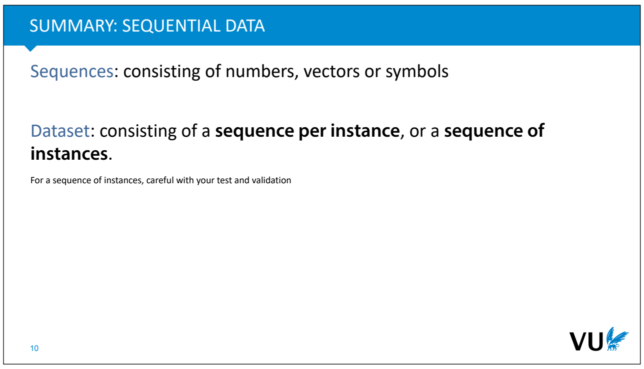
train v 0.4

average

0.3

Here's one approach: keep your data aligned in time and test your model at various points, training on the past only, and using a small stretch afterwards as validation. This simulates how well your model does if you had trained it at a particular point in time.

You can average the different measurements for a general impression of how well your model does, but you can also look at the individual measurements to see, for instance, if there are seasonal effects, how well the model does with little data, what the overall variance is in the performance, etc.



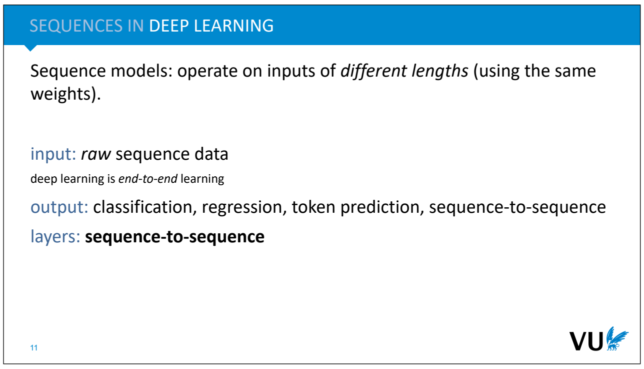
SUMMARY: SEQUENTIAL DATA

Sequences: consisting of numbers, vectors or symbols

Dataset: consisting of a **sequence per instance**, or a **sequence of instances**.

For a sequence of instances, careful with your test and validation

But, often it's better to take a model that can consume sequences natively.



SEQUENCES IN DEEP LEARNING

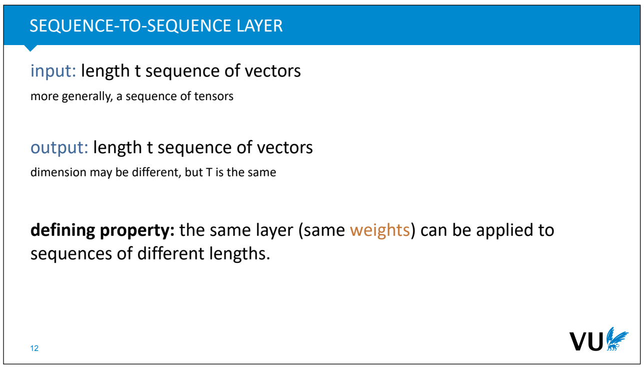
Sequence models: operate on inputs of *different lengths* (using the same weights).

input: *raw* sequence data

deep learning is *end-to-end* learning

output: classification, regression, token prediction, sequence-to-sequence

layers: **sequence-to-sequence**



SEQUENCE-TO-SEQUENCE LAYER

input: length t sequence of vectors

more generally, a sequence of tensors

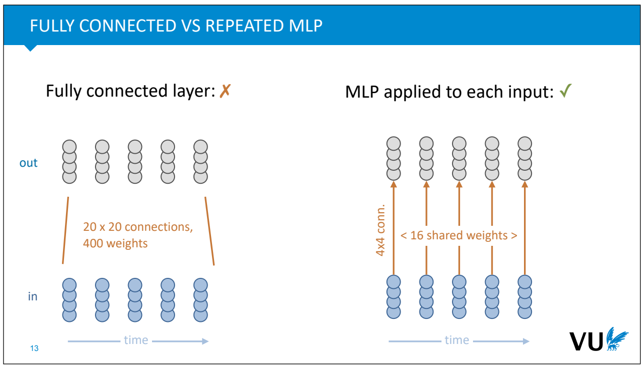
output: length t sequence of vectors

dimension may be different, but T is the same

defining property: the same layer (same weights) can be applied to sequences of different lengths.

We will define all our learning in terms of a single (abstract) type of layer: a sequence to sequence layer.

This layer take a sequence of vectors, and outputs a sequence of vectors. Both sequences have the same number of vectors, but the dimension of the vectors may change. We can generalize this to sequences of tensors (for instance, to analyze film frames), but we'll stick to vectors for this lecture.



FULLY CONNECTED VS REPEATED MLP

Fully connected layer: ✗

MLP applied to each input: ✓

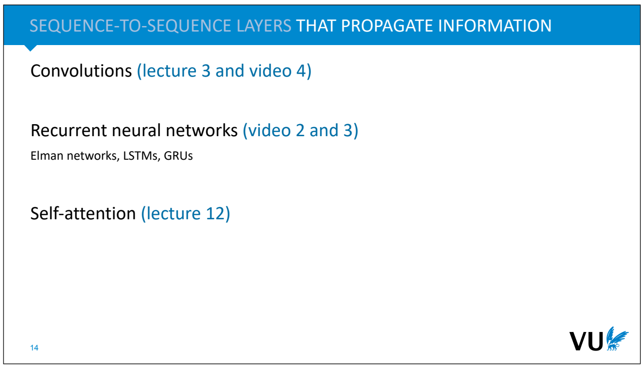
Here is an example: we need a layer that consumes a sequence of five vectors with four elements each and produces another sequence of five vectors with four elements each.

A fully connected layer would simply connect every input with every output, giving us 400 connections with a weight each. This is *not* a sequence-to-sequence layer. Why not? Imagine that the next instance has 6 vectors: we wouldn't be able to feed it to this layer without adding extra weights.

The version on the right also uses an MLP, but only applies it to each vector in isolation: this gives us 4x4=16 connections per vector and 80 in total. These eighty connection share only 16 unique weights, which are repeated at each step.

This is a sequence-to-sequence layer. If the next instance has 6 vectors, we can simple repeat the same MLP again, we don't need any extra weights.

We call the sequence dimension "time", but it doesn't necessarily always represent time.



SEQUENCE-TO-SEQUENCE LAYERS THAT PROPAGATE INFORMATION

Convolutions (lecture 3 and video 4)

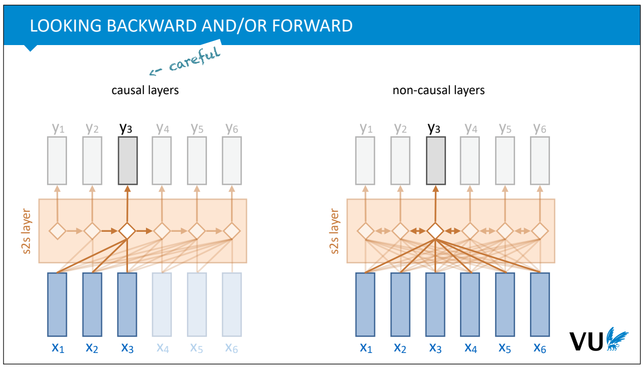
Recurrent neural networks (video 2 and 3)

Elman networks, LSTMs, GRUs

Self-attention (lecture 12)

Of course, the per-element MLP doesn't propagate information across the time dimension, which is the while idea of sequence learning.

These are the most important models that do propagate information.



LOOKING BACKWARD AND/OR FORWARD

<- careful

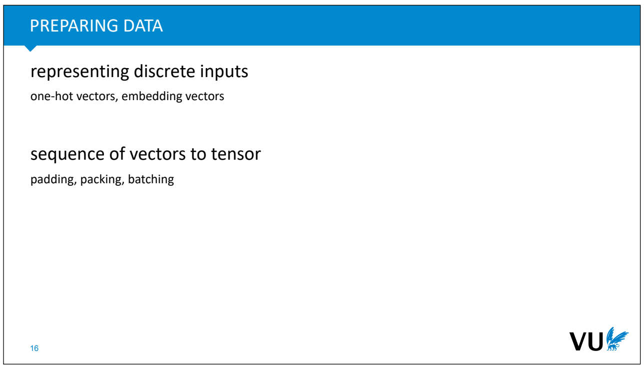
causal layers

non-causal layers

Some sequence-to-sequence layers can only look backward in the input sequence. That means that to provide an output y_{n+1}, the model can only use x_1 to x_n as inputs. This is a very natural assumption for *online* learning tasks, since we usually want to make a prediction about the next token in the sequence before it has come in. They can either reference these inputs directly, or take the output of the previous computation as an additional input. But in either case, information only ever flows from left to right, and the future tokens in the sequence cannot be used to compute the current token in the output.

In many other tasks (say, spam classification) we have access to the whole of the sequence before we need to make our predition. In this case non-causal sequence-to-sequence models are prefereable: these can look at the whole sequence to produce their output.

Note that the name causal is **just a name**: there is no guarantee that these models will actually help you prove causal relations (unless you use them in a particular way).



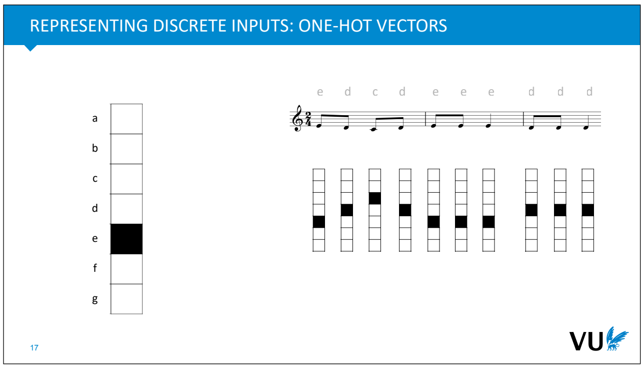
PREPARING DATA

representing discrete inputs

one-hot vectors, embedding vectors

sequence of vectors to tensor

padding, packing, batching

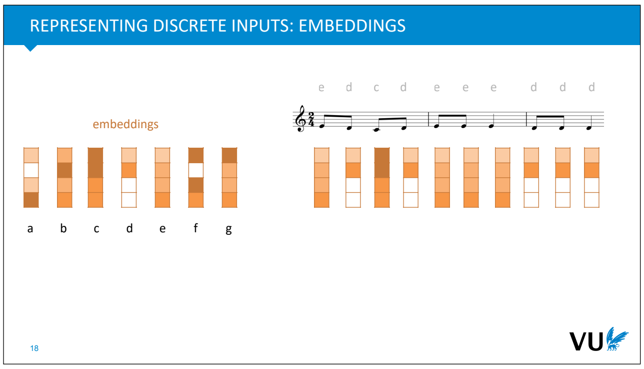


REPRESENTING DISCRETE INPUTS: ONE-HOT VECTORS

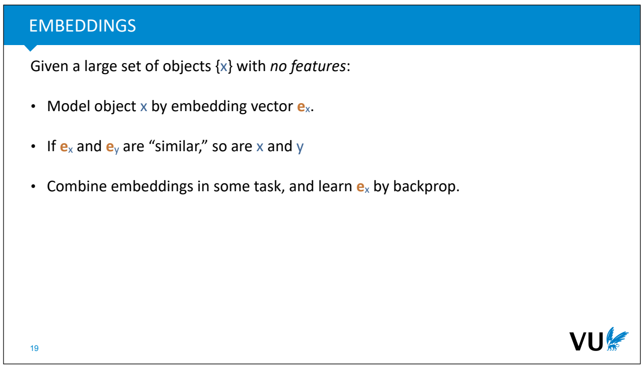
As we've seen, when we want to do deep learning, our input should be represented as a tensor. Preferably in a way that retains all information (i.e. we want to be learning from the raw data, or something as close to it as possible).

Here is an example: to encode a simple monophonic musical sequence, we just one-hot encode the notes, and encode note sequence as a matrix: one dimension for time, one dimension for the notes. We can do the same thing for characters.

source: https://violinsheetmusic.org



REPRESENTING DISCRETE INPUTS: EMBEDDINGS

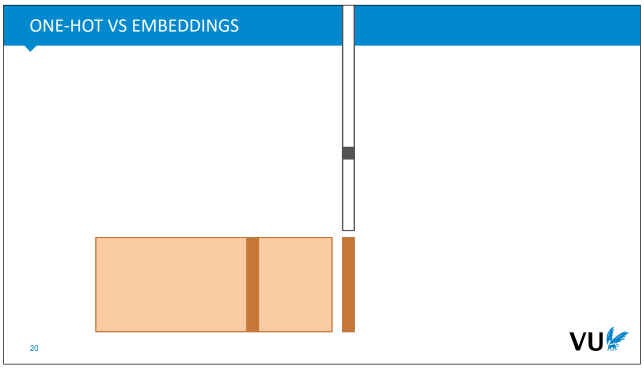


EMBEDDINGS

Given a large set of objects {x} with *no features*:

- Model object x by embedding vector e_x.
- If e_x and e_y are “similar,” so are x and y
- Combine embeddings in some task, and learn e_x by backprop.

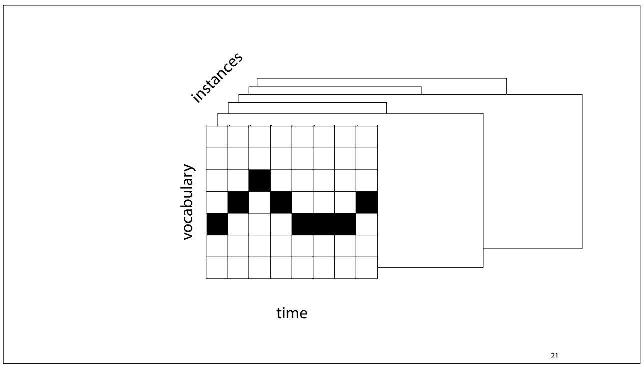
The idea of embedding discrete objects is not specific to sequences. We find it also in matrix decomposition and graph neural networks. Here is the basic principle defined in the most generic terms.



ONE-HOT VS EMBEDDINGS

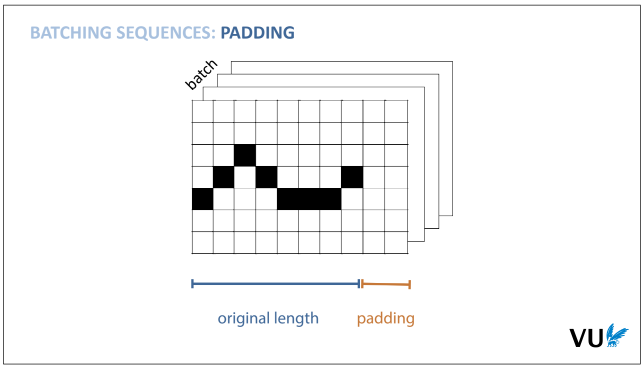
Note that there is not much difference between the two approaches. As soon as we multiply a one-hot vector by a weight matrix, we are selecting a column from that matrix, so that we can see the columns of the weight matrix as a collection of embeddings.

Practically, we rarely implement the one-hot vectors explicitly, because we’d just be storing a large amount of zeros, so the two approaches are likely to lead to the same implementation, once things are properly optimized.



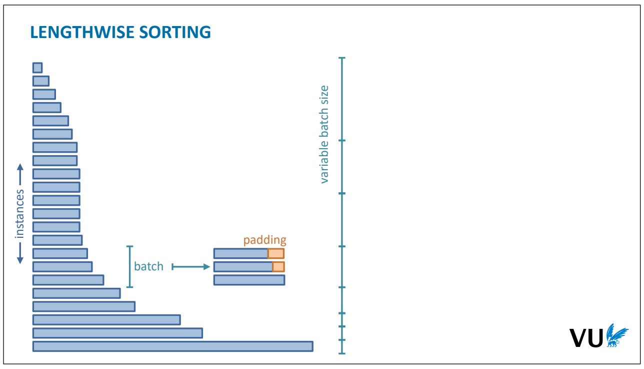
If we have multiple sequences of different lengths, this leads to a data set of matrices of different sizes. This means that our dataset as a whole can’t be stored in a single tensor.

That’s not a problem, we can simply maintain a list of tensors. However, the single batch we feed to our network does need to be a tensor, otherwise we don’t get any parallelism across the batch dimension from our tensor library.



BATCHING SEQUENCES: PADDING

To create batches of a uniform length we can pad our sequences with zeros, or with a special “<pad>” token that we introduce to our vocabulary



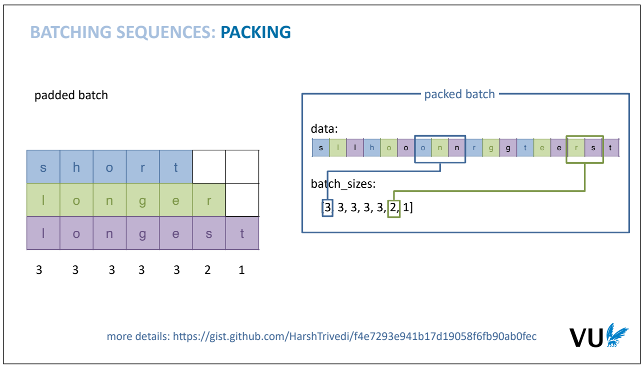
LENGTHWISE SORTING

The lengths of sequences are often broadly powerlaw-distributed with a few very long outliers. If we shuffled the data, we would end up padding batches to the length of the longest member of the sequence, which costs a lot of memory.

A common approach is to sort the data by sequence length and then cut into batches. The first advantage is that most elements in the batch have similar lengths, which minimizes the amount of padding.

The second advantage is that we can lower the batch size as we get further into the dataset: it may be that we only have enough memory to feed the long sequences to the model one at a time, but for the short sequences, we can still train on a large batch in one pass.

Note that this does mean that our instances are no longer i.i.d. This may confuse certain layers (like batch norm) that assume i.i.d. batches.



BATCHING SEQUENCES: PACKING

padded batch

s	h	o	r	t		
l	o	n	g	e	r	
l	o	n	g	e	s	t
3	3	3	3	3	2	1

packed batch

data:

s	l	l	h	o	o	o	n	n	r	g	g	t	e	e	r	s	t

batch_sizes:

[3, 3, 3, 3, 3, 2, 1]

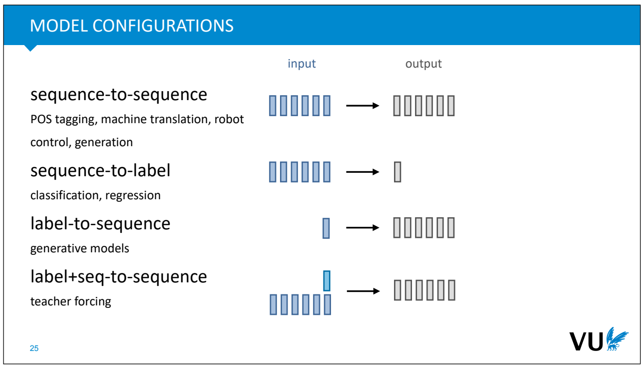
more details: https://gist.github.com/HarshTrivedi/f4e7293e941b17d19058f6fb90ab0fec

In addition to padding your sequences, you can also *pack* them. This is a neat trick that means that you won't use any memory for the zero-padding of your sequences.

The data will be stored in a single sequence that interleaves the different instances in the batch. This is stored together with a count of, reading from left to right how many instances are still in the batch at that point.

Using this information, a sequence layer can process the batch by a sliding window, representing the current "timestep". The window contains all the tokens that may be processed in parallel. As we move from left to right, the window occasionally shrinks, when we reach the end of one of the sequences.

Packing is primarily used for recurrent neural networks, as these actually process sequences serially (i.e. with a sliding window). For self-attention and CNNs, as we shall see, we get a big boost from processing all time steps in parallel, which requires us to operate on padded batches.



MODEL CONFIGURATIONS

input

output

sequence-to-sequence

POS tagging, machine translation, robot control, generation

sequence-to-label

classification, regression

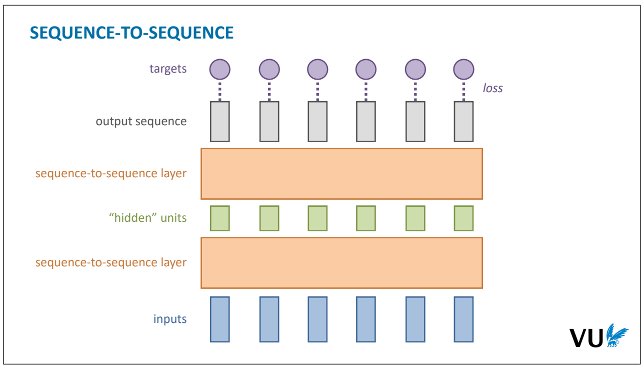
label-to-sequence

generative models

label+seq-to-sequence

teacher forcing

25



SEQUENCE-TO-SEQUENCE

targets

output sequence

sequence-to-sequence layer

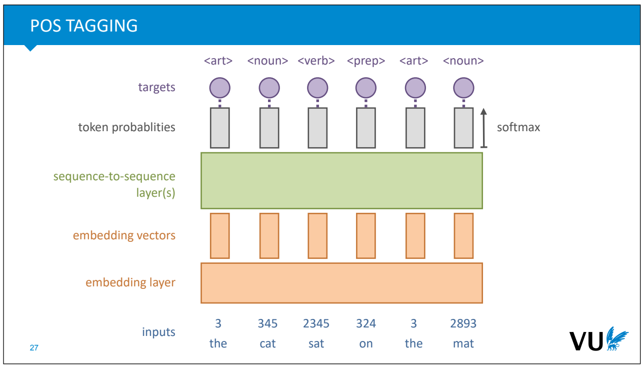
“hidden” units

sequence-to-sequence layer

inputs

loss

A sequence-to-sequence task is probably the simplest set-up. Our dataset consists of a set of input and output sequences. We simply create a model by stacking a bunch of sequence to sequence layers, and our loss is the difference between the target sequence and the output sequence.



POS TAGGING

	<art>	<noun>	<verb>	<prep>	<art>	<noun>
inputs	3	345	2345	324	3	2893
	the	cat	sat	on	the	mat

targets

token probablities

softmax

sequence-to-sequence layer(s)

embedding vectors

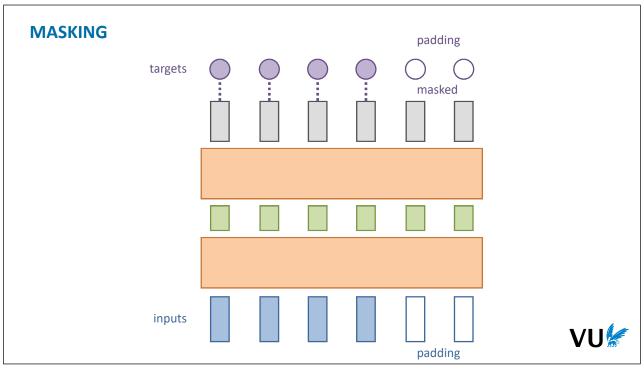
embedding layer

27

Here’s a simple example of a sequence to sequence task: tag each word in a sentence with its grammatical category. This is known as part-of-speech tagging. All we need is a large collection of sentences that have been tagged.

For the embedding layer, we convert our input sequence to positive integers. We have to decide beforehand what the size of our vocabulary is. If we keep a vocabulary of 10 000 tokens, the embedding layer will create 10 000 embedding vectors for us.

It then takes a sequence of positive integers and translates this to a sequence of the corresponding embedding vectors. These are fed to a stack of s2s layers, which produce a sequence of vectors with al many elements as output tokens. After applying a softmax activation to each vector in this sequence, we get a sequence of probabilities over the target tokens.



MASKING

padding

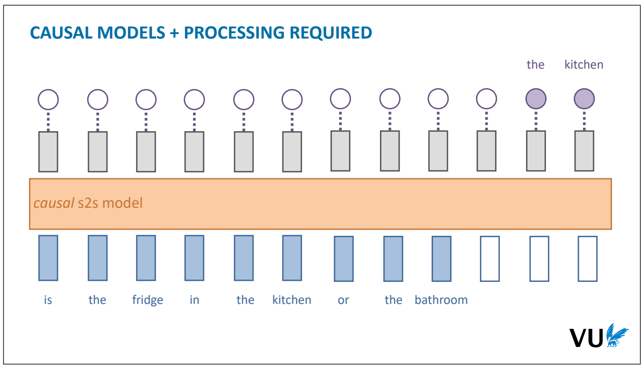
targets

masked

inputs

padding

If we have a padded batch, it's a good idea to mask the computation of the loss for the padded part of the sequence. That is, we compute the loss only for the non-masked tokens, since we don't really care what the model predicts for the pad tokens.



CAUSAL MODELS + PROCESSING REQUIRED

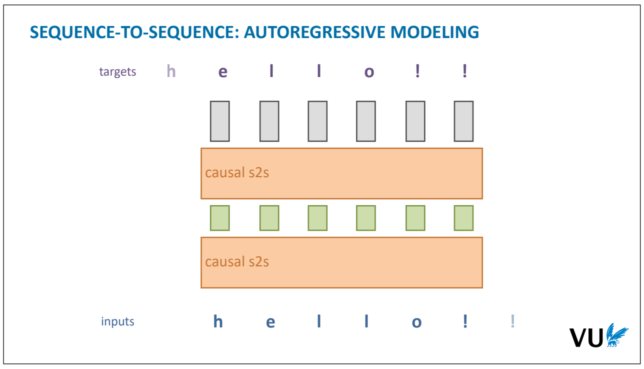
the kitchen

causal s2s model

is the fridge in the kitchen or the bathroom

If we have a causal model, and there is likely some processing required between the input and the output, it's common to let the network read the whole input before it starts processing the output.

Note that we've even given the model one empty step between the end of the question and the beginning of the answer, for extra processing.



SEQUENCE-TO-SEQUENCE: AUTOREGRESSIVE MODELING

targets h e l l o ! !

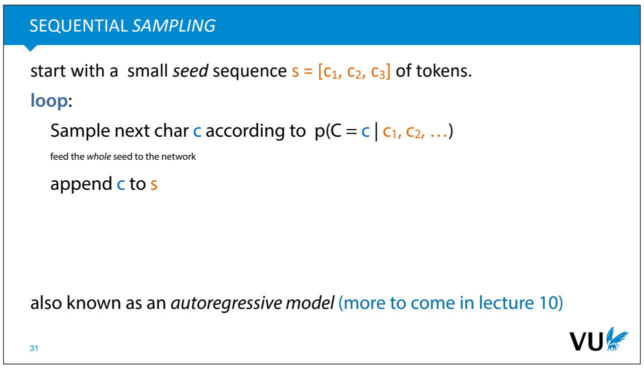
causal s2s

causal s2s

inputs h e l l o ! !

On interesting trick we can use on a causal model, is to feed it some sequence, and to set the target as the same sequence, *shifted one token to the left*.

This effectively trains the model to predict the next character in the sequence. Note that this only works with causal models, because non-causal models can just look ahead in the sequence to see the next character.



SEQUENTIAL *SAMPLING*

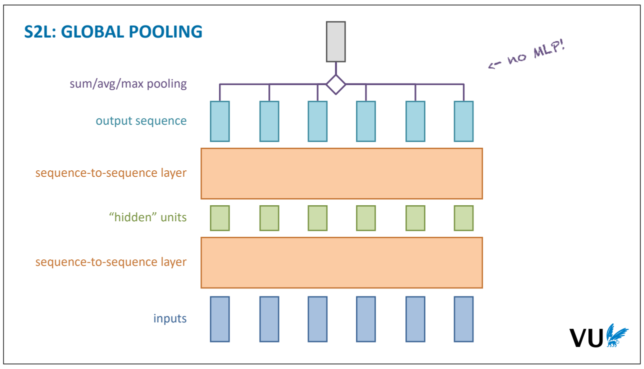
start with a small *seed* sequence $s = [c_1, c_2, c_3]$ of tokens.

```
loop:
    Sample next char $c$ according to $p(C = c \mid c_1, c_2, \ldots)$
    feed the whole seed to the network
    append $c$ to $s$
```

also known as an *autoregressive model* (more to come in lecture 10)

After the network is trained, we can start with a small seed of tokens, and sequentially sample a likely sequence. We'll see some examples of this after we've explained LSTM networks.

In lecture 10, Jakub will go into autoregressive modeling is much greater detail.



S2L: GLOBAL POOLING

sum/avg/max pooling

output sequence

sequence-to-sequence layer

“hidden” units

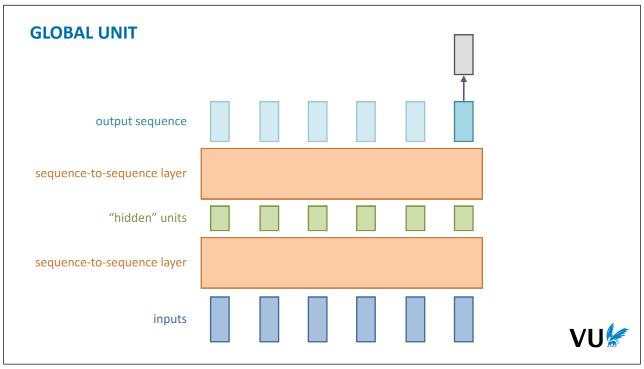
sequence-to-sequence layer

inputs

In a sequence-to-label setting, we get a sequence as input, and we need to produce a single output. This can be a softmaxed vector over classes, or simply an output vector if we need multiple outputs (this vector may also be passed through some further feedforward layers, which we haven’t drawn here).

Here, we first stack one or more sequence to sequence layers. At some point in the network, we need to reduce in the time dimension. A global pooling layer sums, averages or maxes across the time dimension, and results in a single vector.

Note that we can’t use a fully connected layer here: we need an operation that can be applied to



GLOBAL UNIT

output sequence

sequence-to-sequence layer

“hidden” units

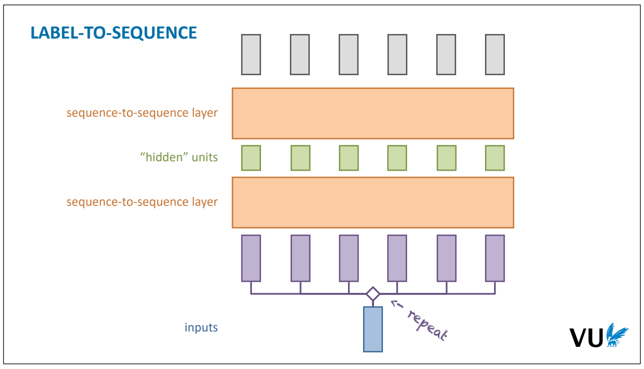
sequence-to-sequence layer

inputs

Another approach is to simply take one of the vectors in the output sequence, use that as the output vector and ignore the rest.

If you have *causal* s2s layers, it’s important that you use the last vector, since that’s the only one that gets to see the whole sequence.

For some layers (like RNNs), this kind of approach puts more weight on the end of the sequence, since the early nodes have to propagate through more intermediate steps in the s2s layer. For others (like self-attention), all inputs in the sequence are treated equally, and there is little difference between a global unit and a pooling layer.



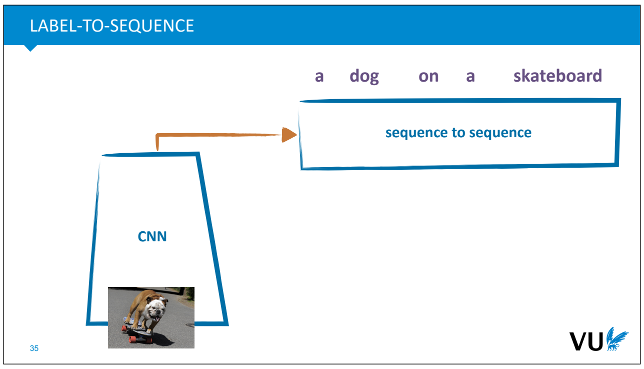
LABEL-TO-SEQUENCE

sequence-to-sequence layer

“hidden” units

sequence-to-sequence layer

inputs



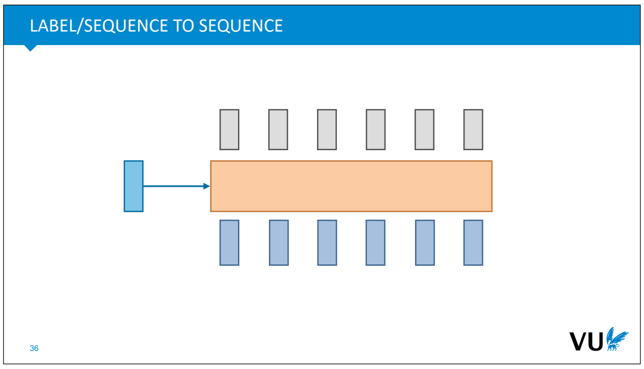
LABEL-TO-SEQUENCE

a dog on a skateboard

sequence to sequence

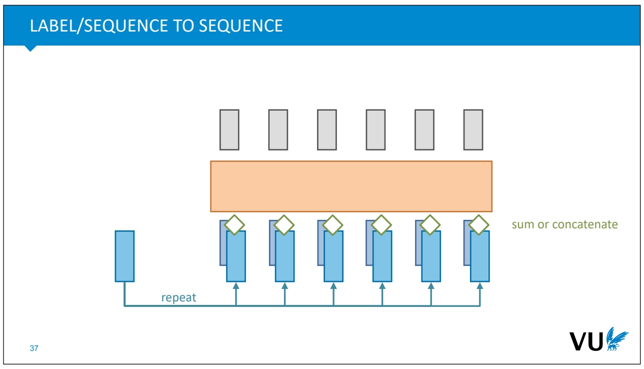
CNN

Here’s one example of a label to sequence task. Taking a simple image, and generating a caption for it. The “label”, here is the input image, which is transformed to a single feature vector by a CNN



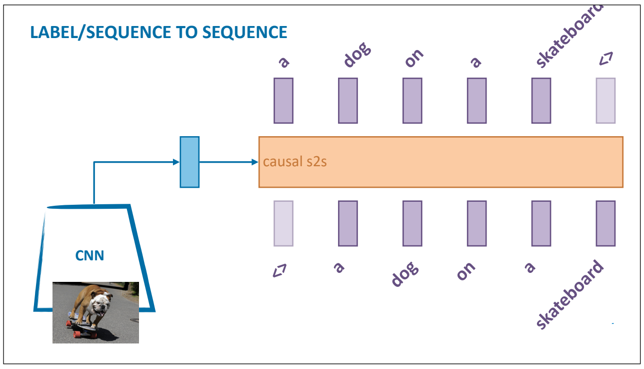
LABEL/SEQUENCE TO SEQUENCE

Our final configuration is the case where we have both a label and a sequence as input. Some sequence-to-sequence layers support a second vector input natively (as we shall see later).



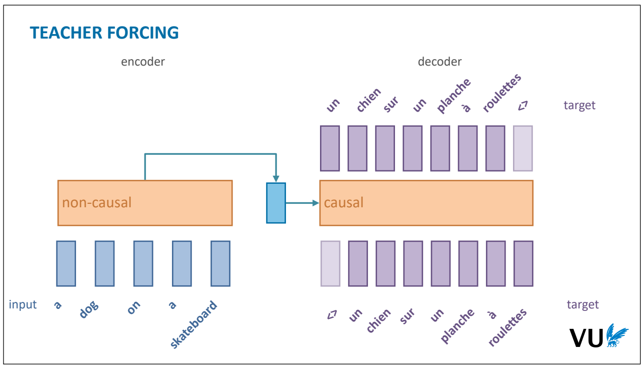
LABEL/SEQUENCE TO SEQUENCE

If there is no native way for the sequence model to accept a second input, we can just repeat the label into a sequence, and concatenate or sum it to the input sequence.



LABEL/SEQUENCE TO SEQUENCE

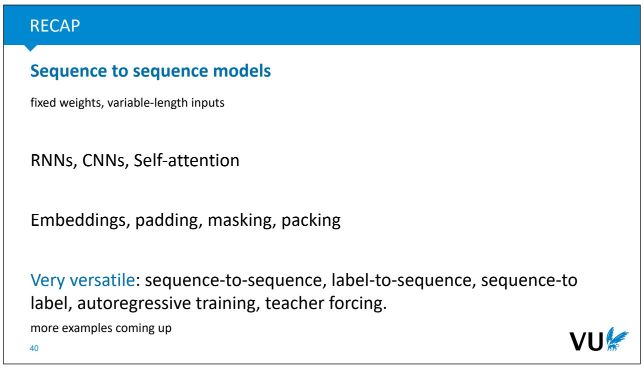
What does this allow us to do? In the image captioning task, we can now train our language model *autoregressively*. This can help a lot to make the output sentences look more natural.



TEACHER FORCING

We can also apply this principle for complex sequence-to-sequence tasks. Here we first use a non-causal model to learn a global representation of the input sequence in a single vector. This is then used to condition a causal model of the output sequence, which is trained like the autoregressive model we saw earlier. It looks complicated, but given a set of inputs and targets, this model can be trained end-to-end by backpropagation.

Once it's trained, we first feed an input to the encoder to get a global representation, and then perform sequential sampling



RECAP

Sequence to sequence models

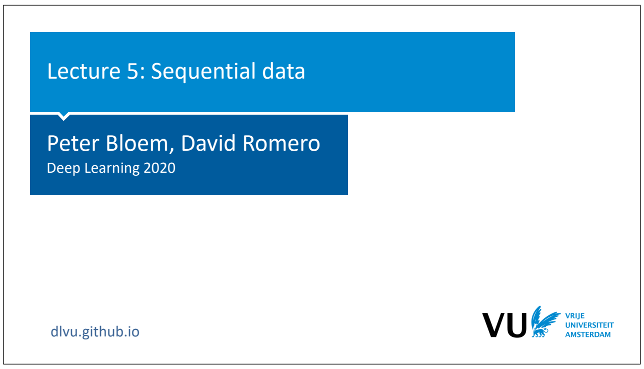
fixed weights, variable-length inputs

RNNs, CNNs, Self-attention

Embeddings, padding, masking, packing

Very versatile: sequence-to-sequence, label-to-sequence, sequence-to label, autoregressive training, teacher forcing.

more examples coming up

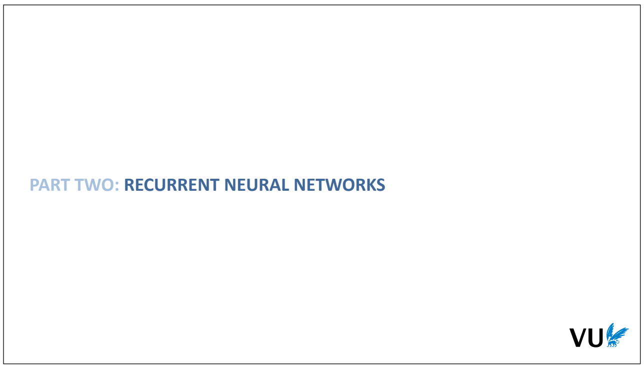


Lecture 5: Sequential data

Peter Bloem, David Romero

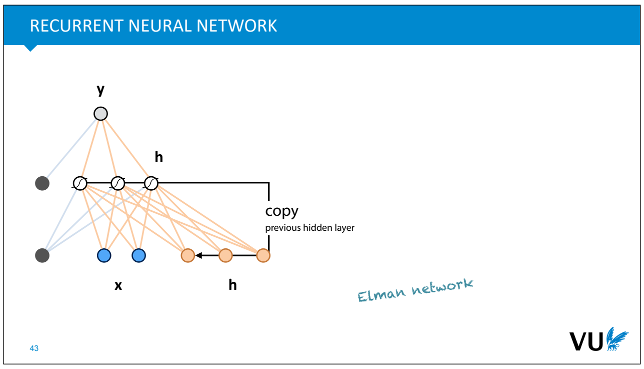
Deep Learning 2020

dlvu.github.io



PART TWO: RECURRENT NEURAL NETWORKS

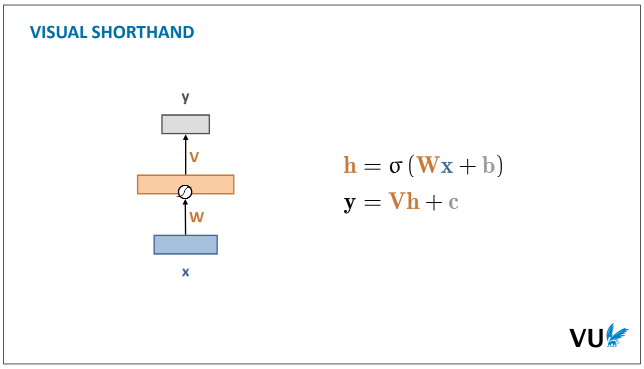
A recurrent neural network is any neural network that has a cycle in it



RECURRENT NEURAL NETWORK

This figure shows a popular configuration. It's a basic fully connected network, except that its input x is extended by three nodes to which the hidden layer is copied.

This particular configuration is sometimes called an **Elman network**. These were popular in the 80s and early 90s, so you'll usually see them with a sigmoid activation.



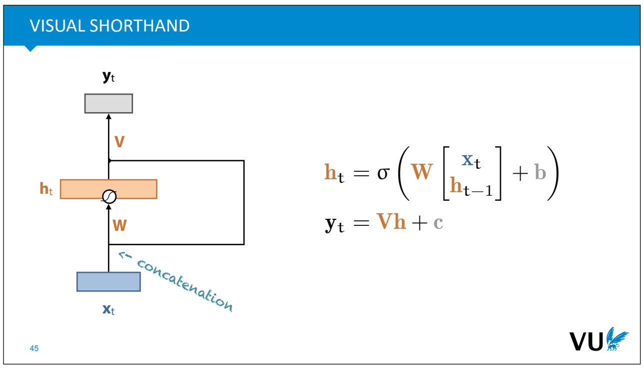
VISUAL SHORTHAND

$$h = \sigma(Wx + b)$$
$$y = Vh + c$$

To keep things clear we will adopt this visual shorthand: a rectangle represents a vector of nodes, and an arrow feeding into such a rectangle annotated with a weight matrix represents a fully connected transformation.

We will assume bias nodes are included without drawing them.

This image shows a simple (nonrecurrent) feedforward net in our new shorthand.

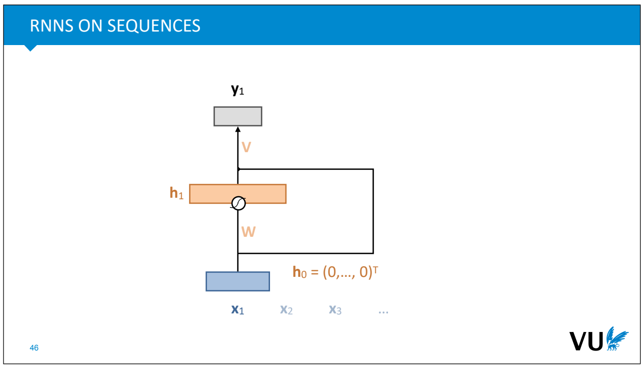


VISUAL SHORTHAND

$$h_t = \sigma\left(W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b\right)$$
$$y_t = Vh + c$$

A line with no weight matrix represents a copy of the input vector. When two lines flow into each other, we concatenate their vectors.

Here, the added line copies h, concatenates it to x, and applies weight matrix W.

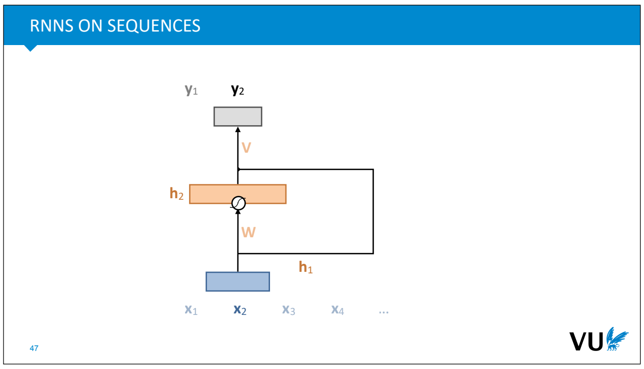


RNNS ON SEQUENCES

We can now apply this neural network to a sequence. We feed it the first input, x1, result in a first value for the hidden layer, h1, and retrieve the first output y1.

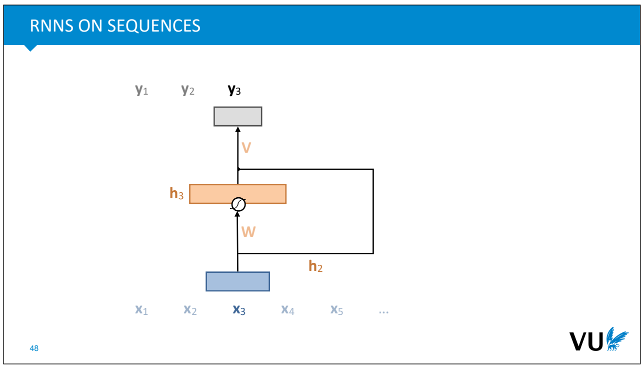
In the first iteration the recurrent inputs are set equal to zero, so the network just behaves like an MLP.

The network provides an output y1, which is the first element of our output sequence.



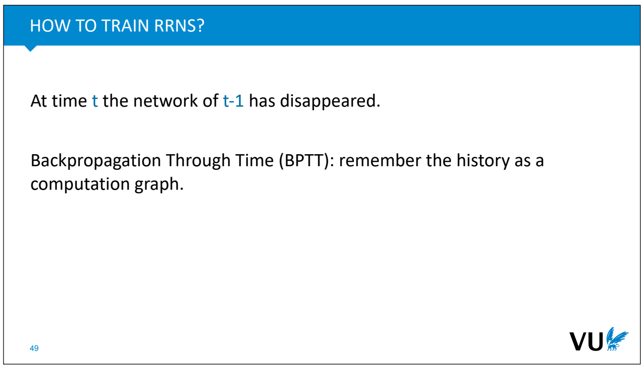
RNNS ON SEQUENCES

In the second step, we feed it the second element in our sequence, concatenated with the hidden layer from the previous sequence.



RNNS ON SEQUENCES

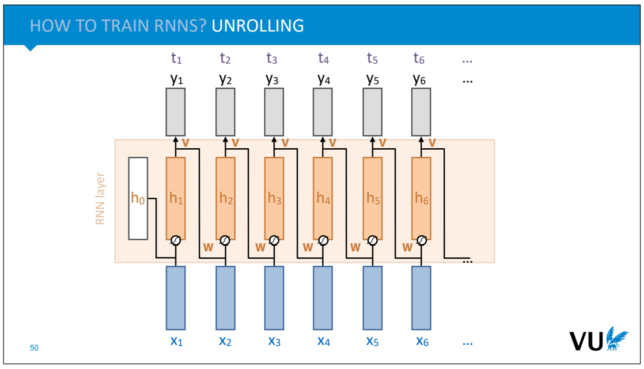
And so on.



HOW TO TRAIN RRNS?

At time t the network of t-1 has disappeared.

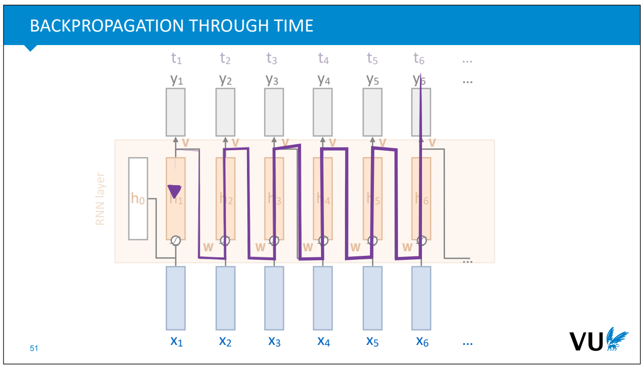
Backpropagation Through Time (BPTT): remember the history as a computation graph.



HOW TO TRAIN RNNS? UNROLLING

Instead of visualising a single small network, applied at every time step, we can unroll the network. Every step in the sequence is applied in parallel to a copy of the network, and the recurrent connection flows from the previous copy to the next.

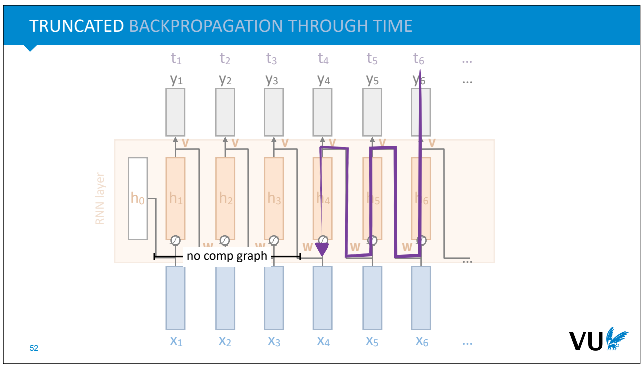
Now the whole network is just one big, complicated feedforward net, that is, a network without cycles. Note that we have a lot of shared weights, but we know how to deal with those.



BACKPROPAGATION THROUGH TIME

Now the whole network is just one big, complicated feedforward net. Note that we have a lot of shared weights, but we know how to deal with those.

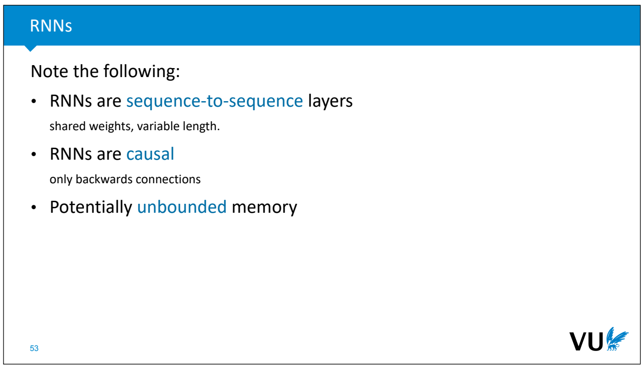
Here, we've only drawn the loss for one output vector, but in a sequence-to-sequence task, we'd get a loss for every vector in the output sequence, which we would then sum.



TRUNCATED BACKPROPAGATION THROUGH TIME

In truncated backpropagation through time, we limit how far back the backpropagation goes, to save memory. The output is still entirely dependent on the whole sequence, but the weights are only trained based on the last few steps. Note that the weights are still affected everywhere, because they are shared between timesteps.

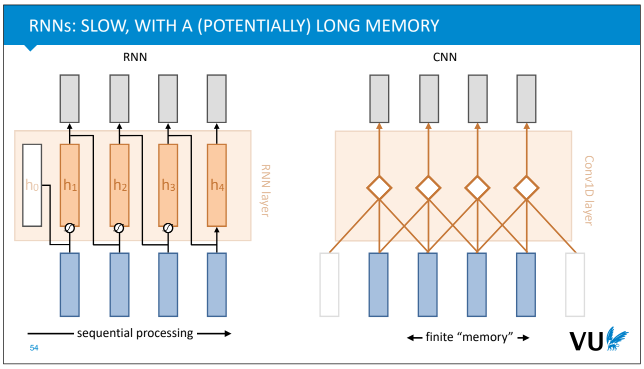
Before the truncation point, we do not need to maintain a computation graph, so up to the computation of **h**$_3$, we do not need to store any intermediate values.



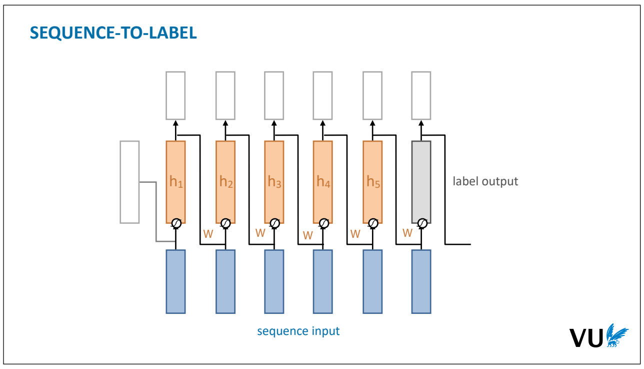
RNNs

Note the following:

- RNNs are sequence-to-sequence layers
  shared weights, variable length.
- RNNs are causal
  only backwards connections
- Potentially unbounded memory



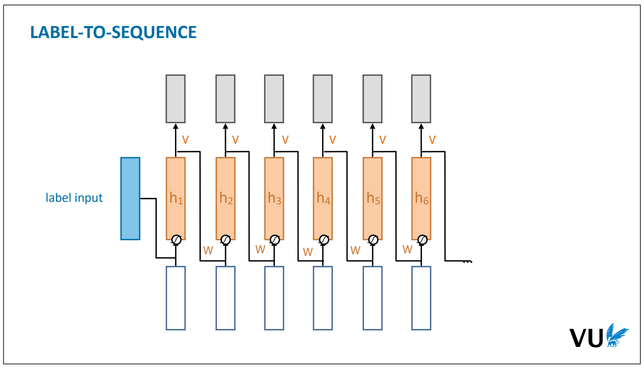
RNNs: SLOW, WITH A (POTENTIALLY) LONG MEMORY



SEQUENCE-TO-LABEL

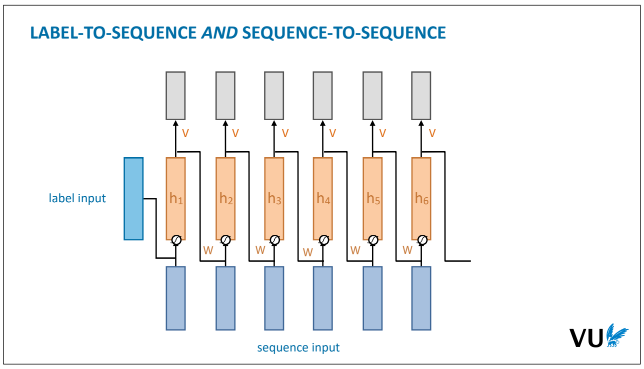
When training sequence-to-label, it’s quite common to take the last hidden state as the label output of the network (possible passed through an MLP to reshape it).

Thos is broadly equivalent to the global unit shown in the first video, so this does mean that the last part of the sequence likely has more influence on the outptu than the first part. Nevertheless, it is a common configuration.



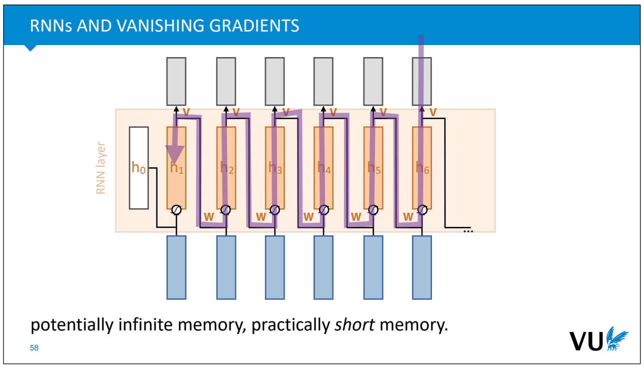
LABEL-TO-SEQUENCE

Similarly, in a label-to-sequence task, you can pass the label vector as the first hidden state. This is a compact way to do it, but do note that the last tokens in the sequence are further way than the first (there are more computations in between). For this reason a repeat strategy as shown on layer 36, may be more powerful (at the cost of a little more memory).



LABEL-TO-SEQUENCE *AND* SEQUENCE-TO-SEQUENCE

If you want to do teacher forcing, or something similar, the hidden state is a neat way to combine the label input and the sequence input.

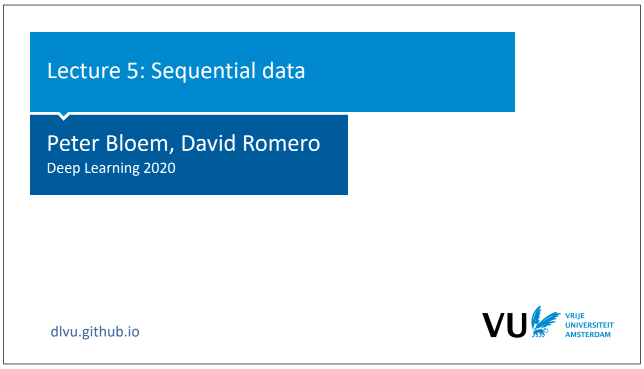


RNNs AND VANISHING GRADIENTS

potentially infinite memory, practically *short* memory.

In theory, RNNs are a wonderful model for sequences, because they can remember things forever. In practice, it turns out that these kinds of RNNs don't. Why not? consider the path taken by the backpropagation algorithm: it passes many activation layers (and these are sigmoids in the most common RNNs). At each step the gradient is multiplied by at most 0.25. The problem of vanishing gradients is very strong in RNNs like this.

We could of course initialize the weight matrices **W** very carefully, use ReLU activations, and perhaps even add an occasional batch-norm-style centering of the activations. Unfortunately, in the 90s, none of these tricks we known yet. Instead researchers am up with something entirely different: the LSTM network.



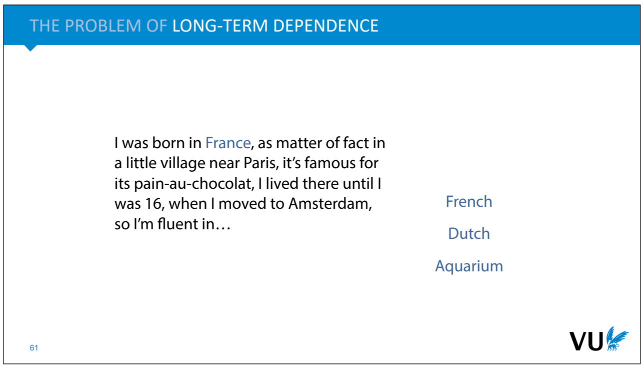
Lecture 5: Sequential data

Peter Bloem, David Romero
Deep Learning 2020

dlvu.github.io

PART THREE: LSTMs and friends

adapted from **http://colah.github.io/posts/2015-08-Understanding-LSTMs/**



THE PROBLEM OF LONG-TERM DEPENDENCE

I was born in France, as matter of fact in a little village near Paris, it's famous for its pain-au-chocolat, I lived there until I was 16, when I moved to Amsterdam, so I'm fluent in…

French

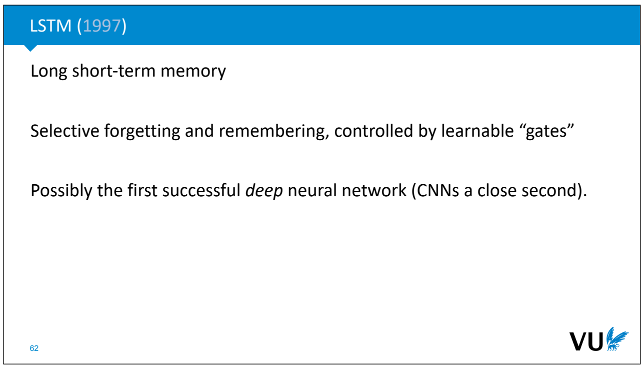
Dutch

Aquarium

Basic RNNs work pretty well, but they do not learn to remember information for very long. Technically they can, but the gradient vanished too quickly over the timesteps.

You can't have a long term memory for everything. You need to be selective, and you need to learn to select words to be stored for the long term when you first see them.

In order to remember things long term you need to forget many other things.



LSTM (1997)

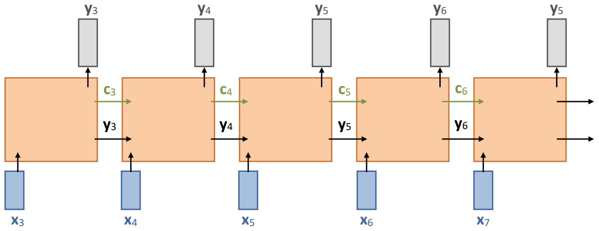
Long short-term memory

Selective forgetting and remembering, controlled by learnable "gates"

Possibly the first successful *deep* neural network (CNNs a close second).

An enduring solution to the problem are LSTMs. LSTMs have a complex mechanism, which we'll go through step by step, but the main component is a gating mechanism.

CELLS



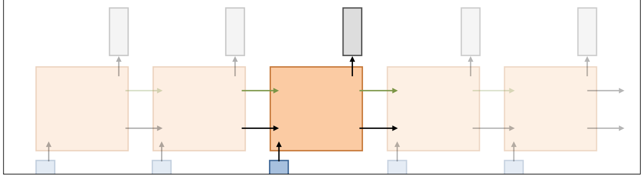
The basic operation of the LSTM is called a cell (the orange square, which we'll detail later). Between cells, there are two recurrent connections, y, the current output, and C the **cell state**.

LSTM CELL

$$x'_t = \begin{bmatrix} y_{t-1} \\ x_t \end{bmatrix}$$

$$f_t = \sigma_s\left(W_f x'_t + b_f\right)$$
$$i_t = \sigma_s\left(W_i x'_t + b_i\right)$$
$$\bar{c}_t = \sigma_t\left(W_c x'_t + b_c\right)$$
$$o_t = \sigma_s\left(W_o x'_t + b_o\right)$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \bar{c}_t$$
$$y_t = o_t \otimes \sigma_t(c_t)$$



Inside the LSTM cell, these formulas are applied. They're a complicated bunch, so we'll first represent what happens visually.

NOTATION

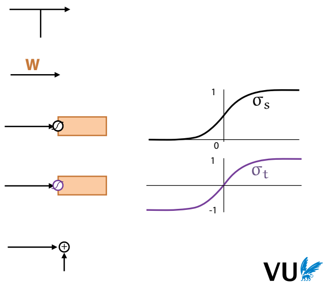
concatenate

apply weights
and corresponding bias

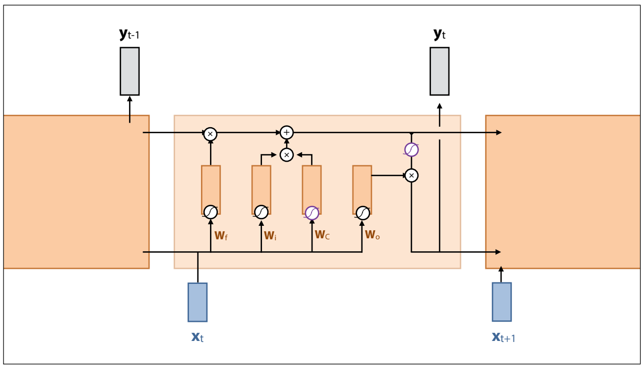
sigmoid activation

tanh activation

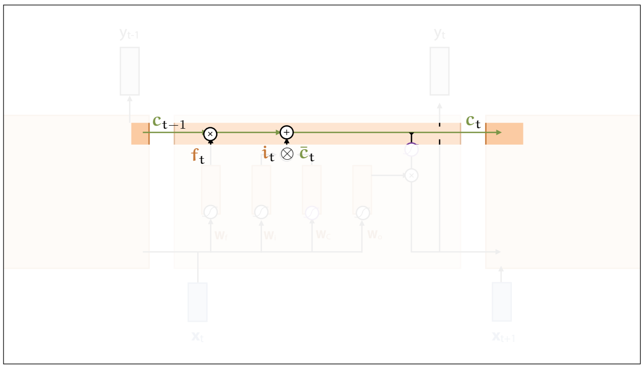
element-wise operation



Here is our visual notation.

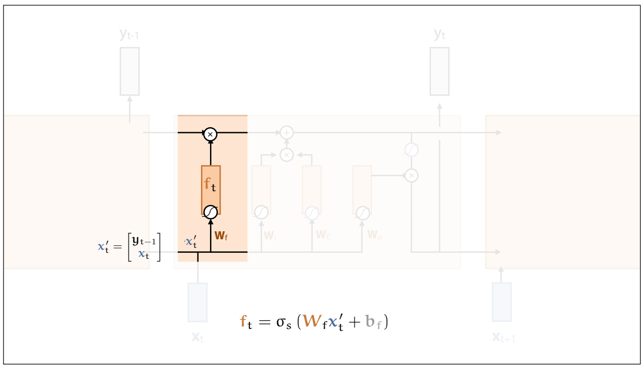


Here is what happens inside the cell. It looks complicated, but we'll go through all the elements step by step.



The first is the “conveyor belt”. It passes the previous cell state to the next cell. Along the way, the current input can be used to manipulate it.

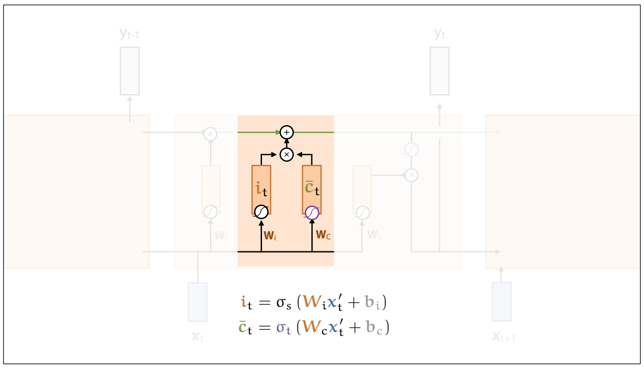
Note that the connection from the previous cell to the next has *no activations*. This means that along this path, gradients do not decay: everything is purely linear. It’s also very easy for an LSTM cell to ignore the current information and just pass the information along the conveyor belt.



$f_t = \sigma_s\left(W_f x'_t + b_f\right)$

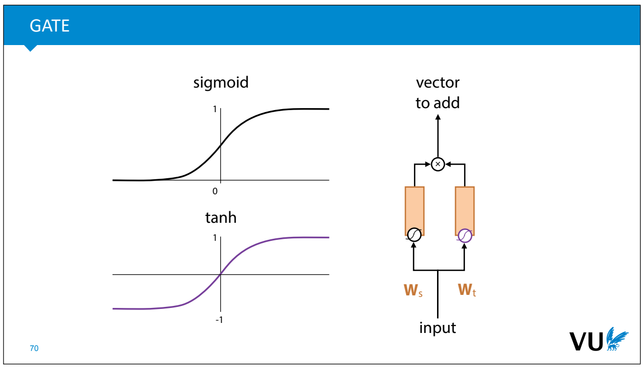
Here is the first manipulation of the conveyor belt. This is called the **forget gate**.

It looks at the current input, concatenated with the previous output, and applies an element-wise scaling to the current value in the conveyor belt. Outputting all 1s will keep the current value on the belt what it is, and outputting all values near 0, will decay the values (forgetting what we’ve seen so far, and allowing it to be replaces by our new values in the next step).



$i_t = \sigma_s\left(W_i x'_t + b_i\right)$
$\bar{c}_t = \sigma_t\left(W_c x'_t + b_c\right)$

in the next step, we pass the input through a generic gate, as described earlier, and add the resulting vector to the value on the conveyor belt.

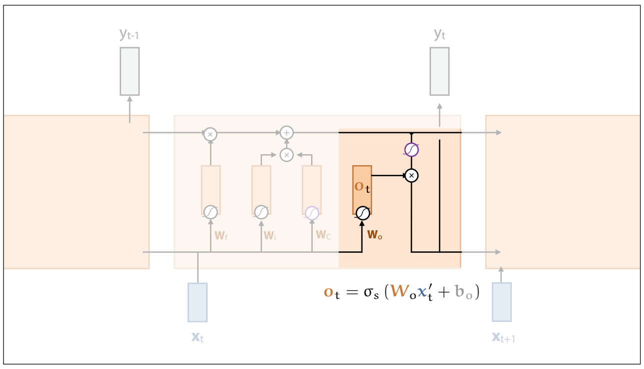

GATE

The gate combines the sigmoid and tanh activations. The sigmoid we’ve seen already. The tanh is just a the sigmoid rescaled so that its outputs are between -1 and 1.

The gating mechanism takes two input vectors, and combines them using a sigmoid and a tanh activation. The gate is best understand as producing an additive value: we want to figure out how much of the input to add to some other vector (if it’s import, we want to add most of if, otherwise, we want to forget it, and keep the original value).

The input is first transformed by two weight metrics and then passed though a sigmoid and a tanh. The tanh should be though of as a mapping of the input to the range [-1, 1]. This ensures that the effect of the addition vector can’t be too much. The sigmoid acts as a selection vector. For elements of the input that are important, it outputs 1, retaining all the input in the addition vector. For elements of the input that are not important, it outputs 0, so that they are zeroed out. The sigmoid and tanh vectors are element-wise multiplied.

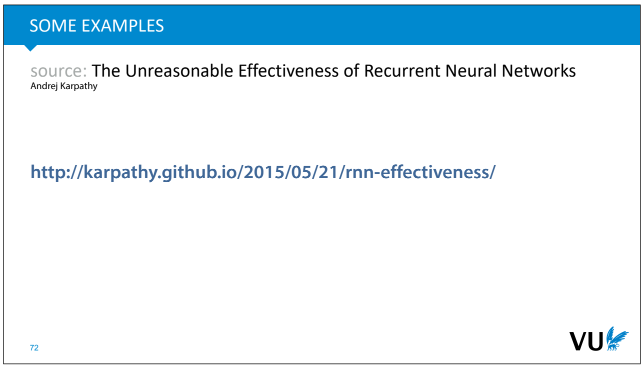
Note that if we initialise Wt and Ws to zero, the input is entirely ignored.



$o_t = \sigma_s(W_o x'_t + b_o)$

Finally, we need to decide what to output now. We take the current value of the conveyor belt, tanh it to rescale, and element-wise multiply it by another sigmoid activated layer. This layer is sent out as the current output, and sent to the next cell along the second recurrent connection.

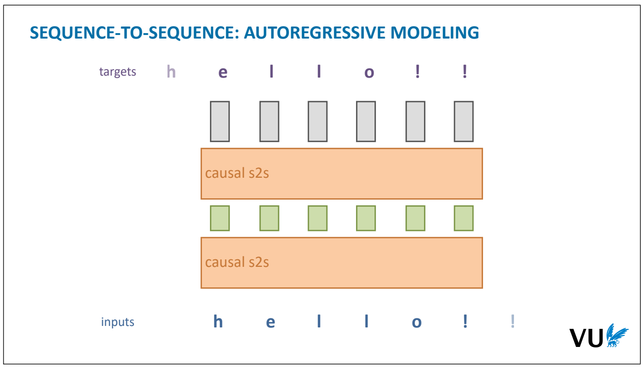
Note that this is another gate construction: the current **c** value is passed though a tanh and multiplied by a filter **o**.



SOME EXAMPLES

source: The Unreasonable Effectiveness of Recurrent Neural Networks
Andrej Karpathy

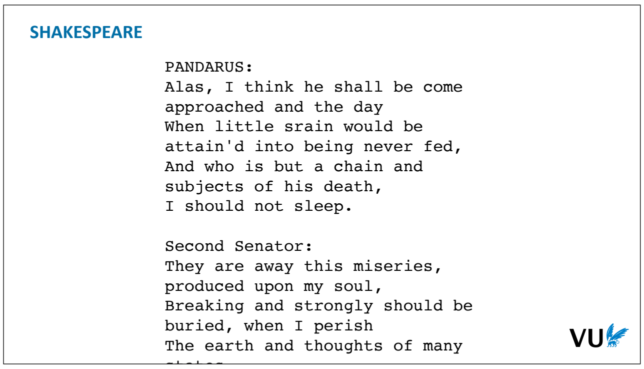
http://karpathy.github.io/2015/05/21/rnn-effectiveness/



SEQUENCE-TO-SEQUENCE: AUTOREGRESSIVE MODELING

On interesting trick we can use on a causal model, is to feed it some sequence, and to set the target as the same sequence, *shifted one token to the left*.

This effectively trains the model to predict the next character in the sequence. Note that this only works with causal models, because non-causal models can just look ahead in the sequence to see the next character.

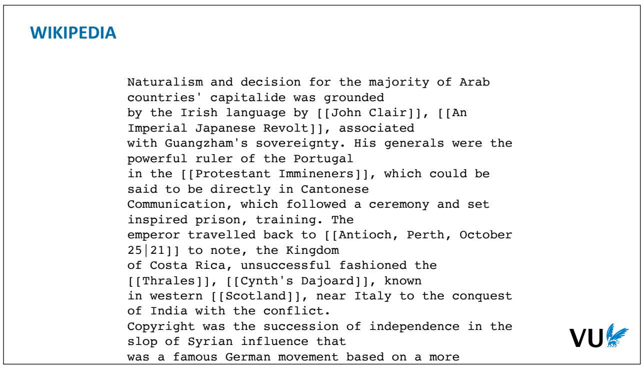


SHAKESPEARE

```
PANDARUS:
Alas, I think he shall be come
approached and the day
When little srain would be
attain'd into being never fed,
And who is but a chain and
subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries,
produced upon my soul,
Breaking and strongly should be
buried, when I perish
The earth and thoughts of many
states
```

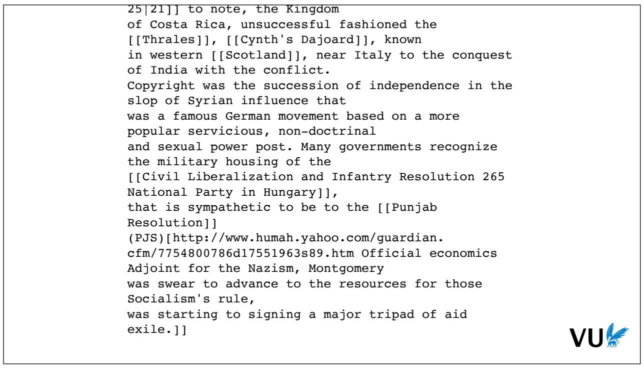
Remember, this is a **character level** language model.



WIKIPEDIA

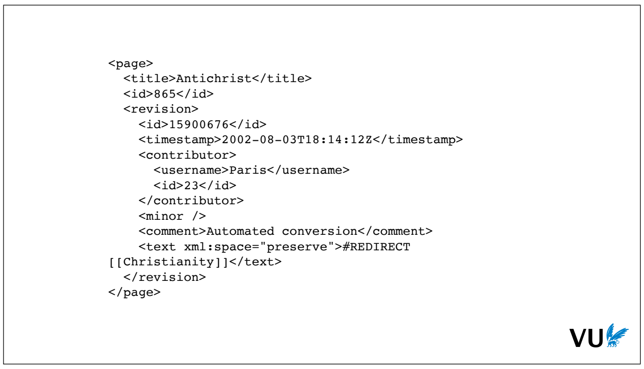
```
Naturalism and decision for the majority of Arab
countries' capitalide was grounded
by the Irish language by [[John Clair]], [[An
Imperial Japanese Revolt]], associated
with Guangzham's sovereignty. His generals were the
powerful ruler of the Portugal
in the [[Protestant Immineners]], which could be
said to be directly in Cantonese
Communication, which followed a ceremony and set
inspired prison, training. The
emperor travelled back to [[Antioch, Perth, October
25|21]] to note, the Kingdom
of Costa Rica, unsuccessful fashioned the
[[Thrales]], [[Cynth's Dajoard]], known
in western [[Scotland]], near Italy to the conquest
of India with the conflict.
Copyright was the succession of independence in the
slop of Syrian influence that
was a famous German movement based on a more
```

Note that not only is the language natural, the wikipedia markup is also correct (link brackets are closed properly, and contain key concepts).



```
25|21]] to note, the Kingdom
of Costa Rica, unsuccessful fashioned the
[[Thrales]], [[Cynth's Dajoard]], known
in western [[Scotland]], near Italy to the conquest
of India with the conflict.
Copyright was the succession of independence in the
slop of Syrian influence that
was a famous German movement based on a more
popular servicious, non-doctrinal
and sexual power post. Many governments recognize
the military housing of the
[[Civil Liberalization and Infantry Resolution 265
National Party in Hungary]],
that is sympathetic to be to the [[Punjab
Resolution]]
(PJS)[http://www.humah.yahoo.com/guardian.
cfm/7754800786d17551963s89.htm Official economics
Adjoint for the Nazism, Montgomery
was swear to advance to the resources for those
Socialism's rule,
was starting to signing a major tripad of aid
exile.]]
```

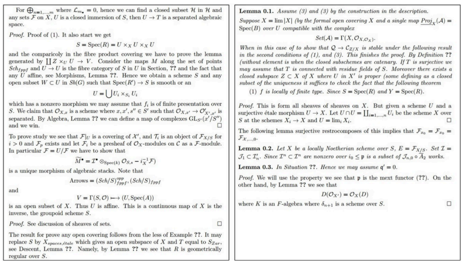
The network can even learn to generate valid (looking) URLs for external links.



```
<page>
  <title>Antichrist</title>
  <id>865</id>
  <revision>
    <id>15900676</id>
    <timestamp>2002-08-03T18:14:12Z</timestamp>
    <contributor>
      <username>Paris</username>
      <id>23</id>
    </contributor>
    <minor />
    <comment>Automated conversion</comment>
    <text xml:space="preserve">#REDIRECT
[[Christianity]]</text>
  </revision>
</page>
```

Sometimes wikipedia text contains bits of XML for structured information. The model can generate these flawlessly.

LATEX



For $\bigoplus_{n=1,\dots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \operatorname{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\operatorname{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\operatorname{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \operatorname{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\underline{Proj}_X(\mathcal{A}) =$ $\operatorname{Spec}(B)$ over U compatible with the complex

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S. Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1) *f is locally of finite type. Since $S = \operatorname{Spec}(R)$ and $Y = \operatorname{Spec}(R)$.*

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\dots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\dots,0}$.

Lemma 0.2. *Let X be a locally Noetherian scheme over S, $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.*

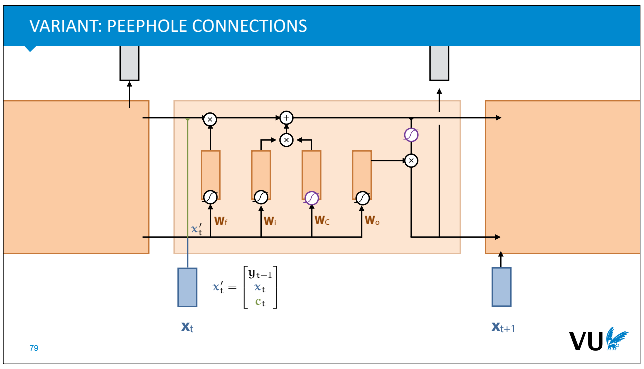
Lemma 0.3. *In Situation ??. Hence we may assume $\mathfrak{q}' = 0$.*

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

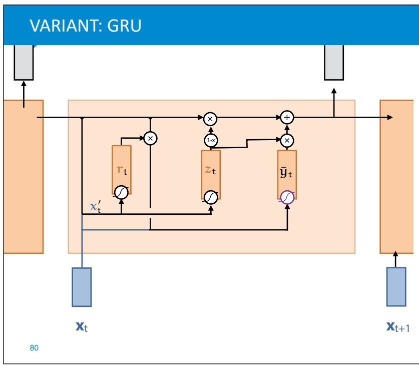
where K is an F-algebra where δ_{n+1} is a scheme over S. □

VARIANT: PEEPHOLE CONNECTIONS



$$x'_t = \begin{bmatrix} y_{t-1} \\ x_t \\ c_t \end{bmatrix}$$

VARIANT: GRU



$$x'_t = \begin{bmatrix} y_{t-1} \\ x_t \end{bmatrix}$$

$$z_t = \sigma_s\left(W_z x'_t + b_z\right)$$
$$r_t = \sigma_s\left(W_r x'_t + b_r\right)$$
$$\bar{y}_t = \sigma_t\left(W_y \begin{bmatrix} y_{t-1} \otimes r_t \\ x_t \end{bmatrix} + b_c\right)$$

$$y_t = (1 - z_t) \otimes y_{t-1} + z_t \otimes \bar{y}_t$$

VARIANT: ConvLSTM

For high-dimensional data (like a sequence of images) the weight matrices **W** get very big.

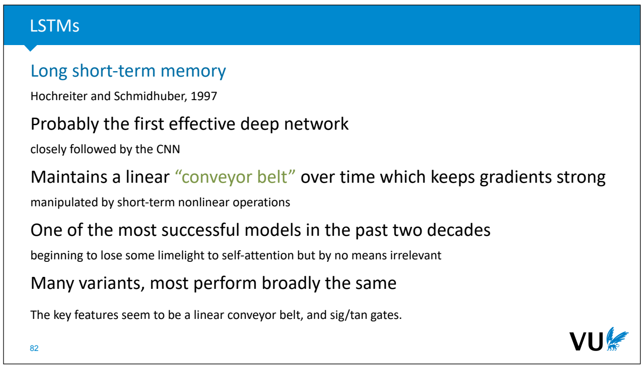
Solution: replace the linear operations with convolutions.

all the vectors becomes 3-tensors.

$$f_t = \sigma_s\left(W_f * x'_t + b_f\right)$$
$$i_t = \sigma_s\left(W_i * x'_t + b_i\right)$$
$$\bar{c}_t = \sigma_t\left(W_c * x'_t + b_c\right)$$
$$o_t = \sigma_s\left(W_o * x'_t + b_o\right)$$

conv kernel ->

image 3-tensor ->



LSTMs

Long short-term memory

Hochreiter and Schmidhuber, 1997

Probably the first effective deep network

closely followed by the CNN

Maintains a linear “conveyor belt” over time which keeps gradients strong

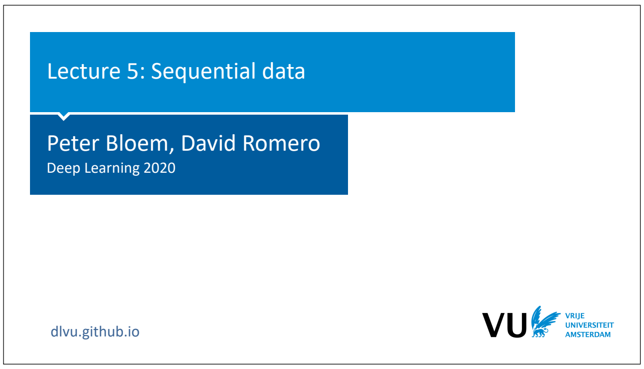
manipulated by short-term nonlinear operations

One of the most successful models in the past two decades

beginning to lose some limelight to self-attention but by no means irrelevant

Many variants, most perform broadly the same

The key features seem to be a linear conveyor belt, and sig/tan gates.



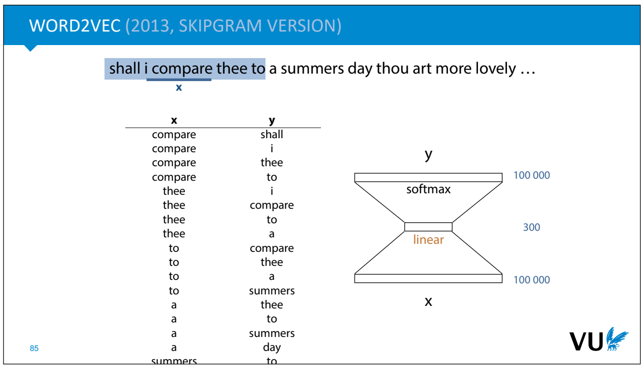
Lecture 5: Sequential data

Peter Bloem, David Romero

Deep Learning 2020

dlvu.github.io

PART FIVE: ELMo, A CASE STUDY



WORD2VEC (2013, SKIPGRAM VERSION)

shall i compare thee to a summers day thou art more lovely …

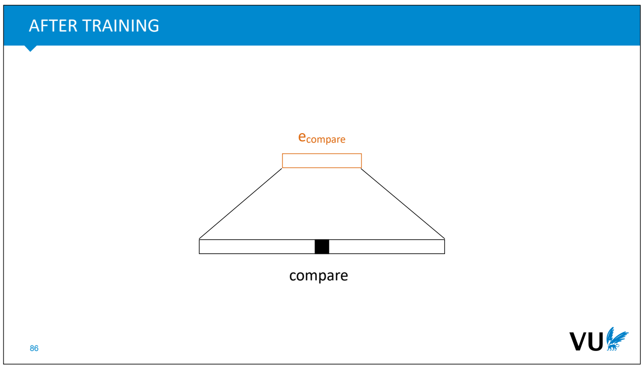
x	y
compare	shall
compare	i
compare	thee
compare	to
thee	i
thee	compare
thee	to
thee	a
to	compare
to	thee
to	a
to	summers
a	thee
a	to
a	summers
a	day
summers	to

To place ELMo into context, let’s first look at one of it’s predecessors: Word2Vec.

We slide a context window over the sequence. The task is to predict the distribution p(y|x): that predict which words are likely to occur in the context window given the middle word.

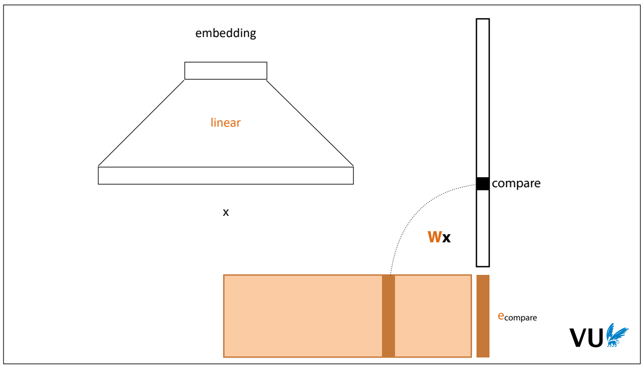
We create a dataset of word pairs from the entire text and feed this to a very simple two-layer network. This is a bit like an autoencoder, except we’re not reconstructing the output, but predicting the *context*.

The softmax activation over 10k outputs is very expensive to compute, and you need some clever tricks to make this feasible (called *hierarchical* softmax or negative sampling). We won't go into them here.



AFTER TRAINING

After training, we discard the second layer, and use only the embeddings produced by the first layer.

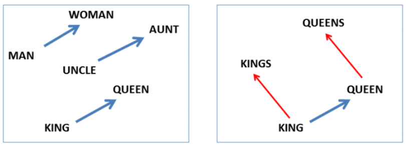


Here, we can see a very direct example of the principle noted at the start of the lecture: that multiplying by

Because the input layer is just a matrix multiplication, and the input is just a one-hot vector, what we end up doing when we compute the embedding for word i, is just extracting the i-th column from **W**.

In other words, we're not really training a function that *computes* an embedding for each word, we are actually learning the embeddings directly: every element of every embedding vector is a separate parameter.

$e_{\text{king}} + \underbrace{e_{\text{woman}} - e_{\text{man}}}_{\text{“feminine” direction}} \approx e_{\text{queen}}$

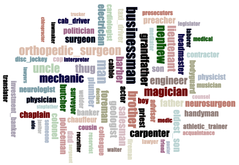


(Mikolov et al., NAACL HLT, 2013)

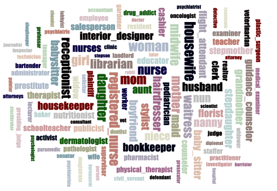
Famously, Word2Vec produces not just an informative embedding, where similar words are close together, but for many concepts, there seems to be a kind of algebraic structure, similar to the smile vector example from the autoencoder.

DATA BIAS

he (84)



she (116)



source: wordbias.org

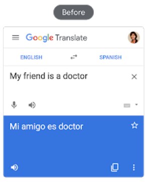
Word2Vec was also one of the first systems that clea

As useful as word embeddings are, it’s very important to be aware that they will reflect the bias in your data. Any large collection of text, for instance, will reflect gender biases that exist in society.

In itself, this is not a bad thing: it may even help to map out those biases and study them better.

DATA BIAS

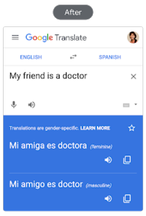
Before



My friend is a doctor

Mi amigo es doctor

After



My friend is a doctor

Translations are gender-specific. LEARN MORE

Mi amiga es doctora (feminine)

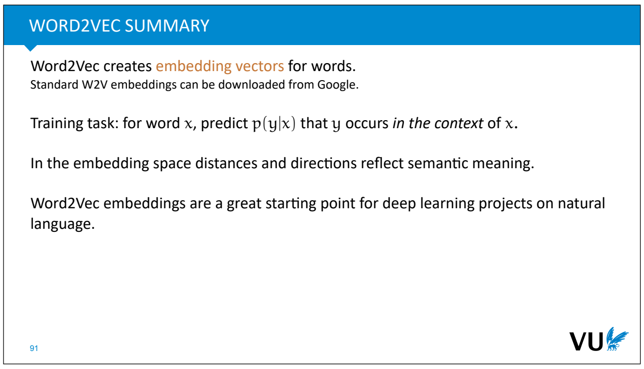
Mi amigo es doctor (masculine)

source: https://ai.googleblog.com/2020/04/a-scalable-approach-to-reducing-gender.html

However when you use these statistics in an application, you need to turn your predictions into actions, which will almost certainly end up reinforcing the existing biases.

Shown here is google’s machine translation system. A sentence which is gender-neutral in English, like “My friend is a doctor” cannot be translated in a gender-neutral way into Spanish. In the earlier versions of Google Translate, a gender was chosen (implicitly), mostly dictated by the statistics of the dataset. You may argue that these statistics are in a sense reflective of biases that exist in society, so that it is indeed more likely that this sentence should be translated for a male. However, that doesn’t mean that we’re *certain* that the user wants the sentence translated in this way. And by reducing uncertain predictions to discrete, certain actions, we run the risk of not just reproducing the bias in our data, but also amplifying it.

The solution (in this case) was not to reduce the uncertainty by guessing more accurately, but to detect it, and communicate it to the user. In this case, by showing the two possible translations to the user.



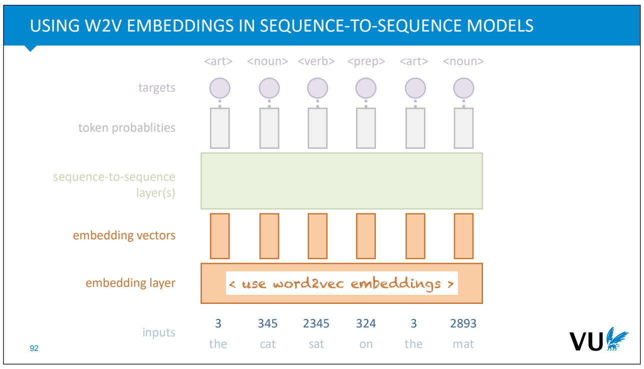
WORD2VEC SUMMARY

Word2Vec creates embedding vectors for words.
Standard W2V embeddings can be downloaded from Google.

Training task: for word x, predict $p(y|x)$ that y occurs *in the context* of x.

In the embedding space distances and directions reflect semantic meaning.

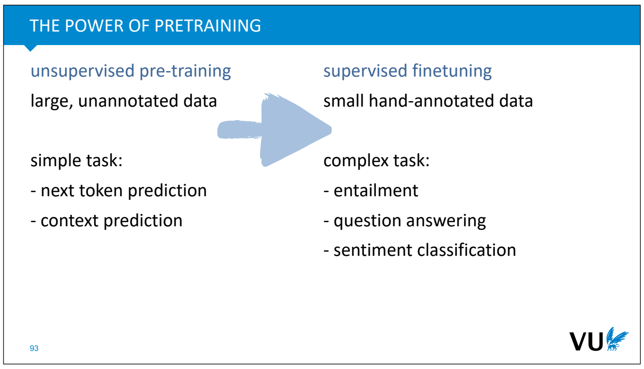
Word2Vec embeddings are a great starting point for deep learning projects on natural language.



USING W2V EMBEDDINGS IN SEQUENCE-TO-SEQUENCE MODELS

W2V embeddings have many uses. For our current purposes, the most interesting application is that if we have a sequence-based model, with an embedding layer, we can use word2vec embeddings instead of embeddings learned from scratch. We can then fine tune these by backpropagation, or just leave them as is.

We find that adding W2V embeddings often improves performance. This is because the s2s model is likely trained on a relatively small datasets, since it needs to be hand-annotated. W2V, in contrast, can easily be trained on great volumes of data, since all we need is a large corpus of high-quality un-annotated text. By combining the two, we are adding some of the power of that large volume of data, to our low-volume setting.



THE POWER OF PRETRAINING

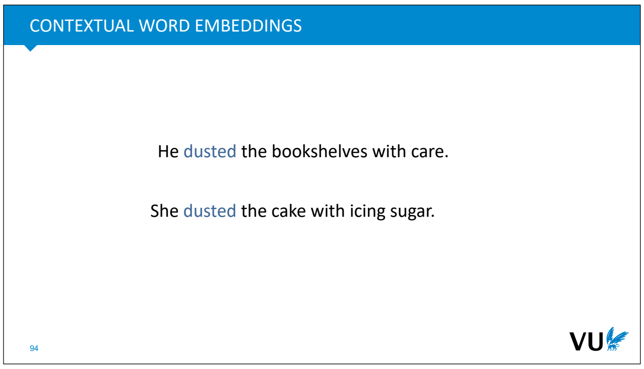
unsupervised pre-training
large, unannotated data

simple task:
- next token prediction
- context prediction

supervised finetuning
small hand-annotated data

complex task:
- entailment
- question answering
- sentiment classification

This would prove to be a great breakthrough in natural language processing: pre-training on an unsupervised task, and finetuning on supervised data, could lead to much greater performance than had been seen before.



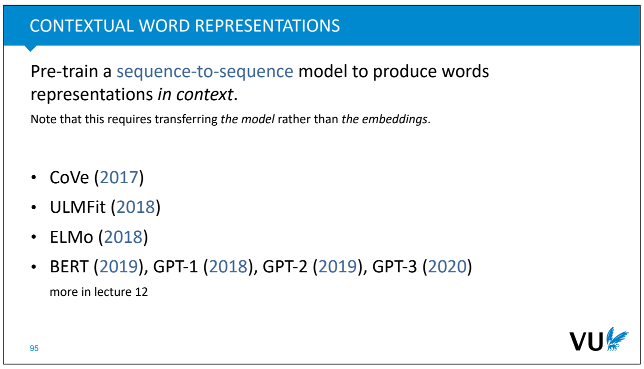
CONTEXTUAL WORD EMBEDDINGS

He dusted the bookshelves with care.

She dusted the cake with icing sugar.

To take this principle, and build on it, the first thing we must do is to learn contextual representations of words. The same word can mean different things in different contexts.

While Word2Vec uses the context of a word as a training signal, it ultimately provides only a single embedding vector for any given word. To get contextual representations for our words we need to pre-train a sequence to sequence model on a large amount of unsupervised data.



CONTEXTUAL WORD REPRESENTATIONS

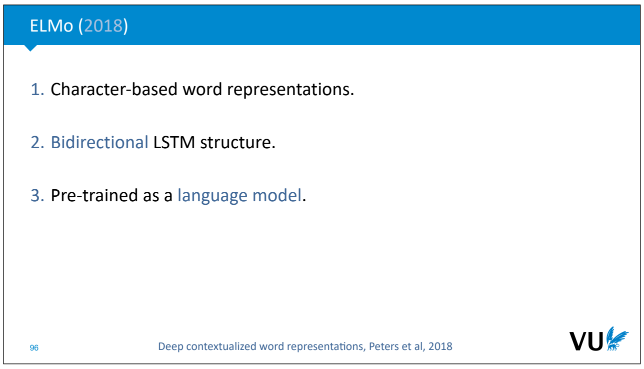
Pre-train a sequence-to-sequence model to produce words representations *in context*.

Note that this requires transferring *the model* rather than *the embeddings*.

- CoVe (2017)
- ULMFit (2018)
- ELMo (2018)
- BERT (2019), GPT-1 (2018), GPT-2 (2019), GPT-3 (2020)

  more in lecture 12

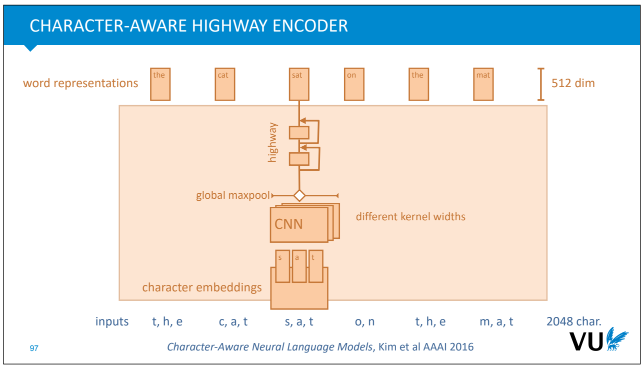
ELMo wasn't the first model to do this, nor is it currently the best option available, but it was the first model that achieved state-of-the-art performance on a wide range of fine tuning tasks, and it was the last model that used RNNS (the later models all use only self-attention), so it seems suitable to highlight it at this point.



ELMo (2018)

1. Character-based word representations.
2. Bidirectional LSTM structure.
3. Pre-trained as a language model.

Deep contextualized word representations, Peters et al, 2018

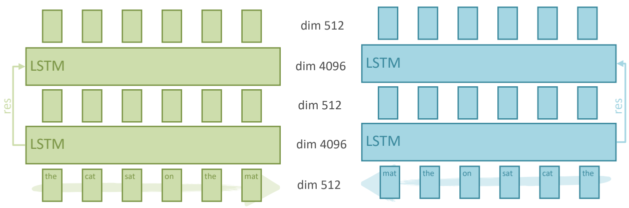


CHARACTER-AWARE HIGHWAY ENCODER

Character-Aware Neural Language Models, Kim et al AAAI 2016

BIDIRECTIONAL LSTM

two multilayer LSTMs: forward and backward.



After each LSTM the output is porjected down to 512 dimensions by a hidden layer applied token-wise(and then projected back up again to 512 for the next LSTM).

LANGUAGE MODELS

$$\text{p}(\text{“congratulations you have won a prize”})$$

$$= \text{p}(W_1{=}\text{congratulations}, W_2{=}\text{you}, W_3{=}\text{have}, W_4{=}\text{won}, W_5{=}\text{a}, W_6{=}\text{prize})$$

$$\text{p}(W_1, W_2, W_3, W_4, W_5, W_6)$$

When modelling probability, we usually break the sequence up into its tokens (in this case the words of the sentence) and model each as a random variable. Note that these random variables are decidedly not independent.

This leaves us with a joint distribution over 6 variables, which we would somehow like to model and fit to a dataset.

$$\text{p}(x, y) = \text{p}(x \mid y)\text{p}(y)$$

If we have a joint distribution over more than two variables on the left, we can apply this rule multiple times.

CHAIN RULE *OF PROBABILITY*

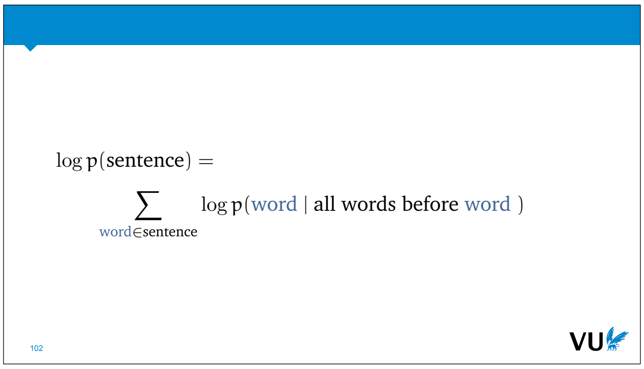
$$\begin{aligned}
&\text{p}(W_4, W_3, W_2, W_1) \\
&\quad = \text{p}(W_4, W_3, W_2 \mid W_1)\text{p}(W_1) \\
&\quad = \text{p}(W_4, W_3 \mid W_2, W_1)\text{p}(W_2 \mid W_1)\text{p}(W_1) \\
&\quad = \text{p}(W_4 \mid W_3, W_2, W_1)\text{p}(W_3 \mid W_2, W_1)\text{p}(W_2 \mid W_1)\text{p}(W_1)
\end{aligned}$$

$$\text{p}(\text{prize} \mid \text{a}, \text{won}, \text{have}, \text{you}, \text{congratulations})$$

This gives us the **chain rule of probability** (not to be confused with the chain rule of calculus, which is entirely different), which is often used in modelling sequences.

The chain rule allows us to break a joint distribution on many variables into a product of conditional distributions. In sequences, we often apply it so that each word becomes conditioned on the words before it.

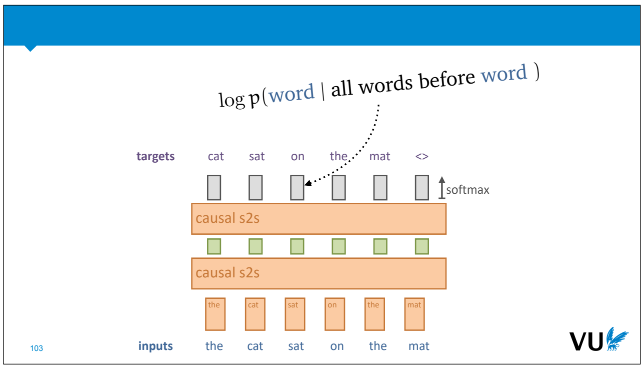
This tells us that if we build a model that can estimate the probability p(x|y, z) of a word x based on the words y, z that precede it, we can then *chain* this estimator to give us the joint probability of the whole sentence x, y, z.



$$\log p(\text{sentence}) = \sum_{\text{word} \in \text{sentence}} \log p(\text{word} \mid \text{all words before word})$$

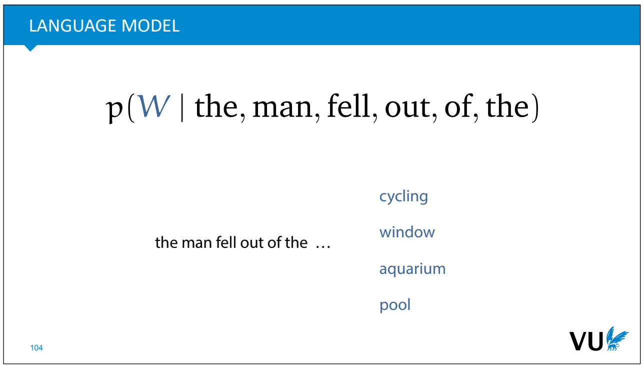
In other words, we can rewrite the probability of a sentences as the product of the probability of each word, conditioned on its history. If we use the log probability, this becomes a sum.

Note that applying the chain rule in a different order would allow us to condition any word on any other word, but conditioning on the history fits well with the sequential nature of the data, and will allow us to make some useful simplifications later.



$\log p(\text{word} \mid \text{all words before word})$

If we train our LSTM autoregressively, we are essentially maximizing this language model loss, by optimizing for a conditional probability at each token in our sequence.



LANGUAGE MODEL

$$p(W \mid \text{the, man, fell, out, of, the})$$

the man fell out of the …

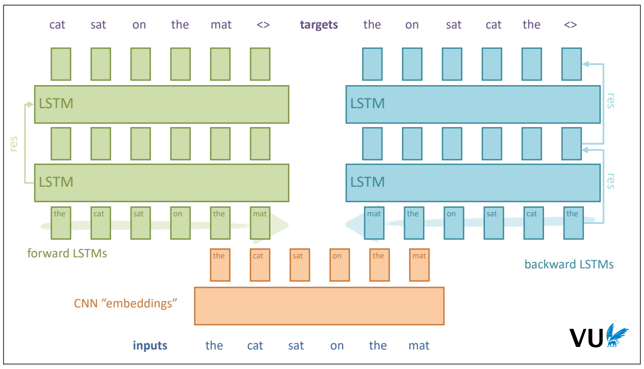
cycling

window

aquarium

pool

A perfect language model would encompass everything we know about language: the grammar, the idiom and the physical reality it describes. For instance, it would give window a very high probability, since that is a very reasonable way to complete the sentence. Aquarium is less likely, but still physically possible and grammatically correct. A very clever language model might know that falling out of a pool is not physically possible (except under unusual circumstances), so that should get a lower probability, and finally cycling is ungrammatical, so that should get very low probability (perhaps even zero).



The residual connections are drawn only for one token but they are applied to every token in the sequence.

Ultimately, this gives us 5 different representations for every input word. Which one should we use in our downstream task?

FINETUNING

Take a weighted mixture of all word embeddings **h**.

L is the LSTM depth, all purple values are trainable in finetuning.

Learn the weights, together with a downstream network.

$$e_k = \gamma i \mathbf{h}_k^{\text{init}} + \gamma \sum_{j=0}^{L} f_j \mathbf{h}_{k,j}^{\text{forward}} + \gamma \sum_{j=0}^{L} b_j \mathbf{h}_{k,j}^{\text{backward}}$$

RESULTS

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE
SQuAD	Liu et al. (2017)	84.4	81.1	85.8
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17
SRL	He et al. (2017)	81.7	81.4	84.6
Coref	Lee et al. (2017)	67.2	67.2	70.4
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5

RECAP

ELMo (2018):

Large unsupervised pretraining, small-scale supervised finetuning

BiLSTM structure

Elaborate finetuning architectures still required.

more on this when we get to self-attention

Lecture 5: Sequential data

Peter Bloem, David Romero

Deep Learning 2020

dlvu.github.io

VRIJE UNIVERSITEIT AMSTERDAM

PART FOUR: CNNS FOR SEQUENTIAL DATA

SUMMARY OF PREVIOUS TECHNIQUES

So far we have seen recurrent architectures, e.g., RNNs , LSTMs , …

Properties:

- Able to handle arbitrarily long sequences (via recurrence).
- **BUT** suffer from vanishing / exploding gradients problem. **(Difficult to train and to learn from the far past)**.

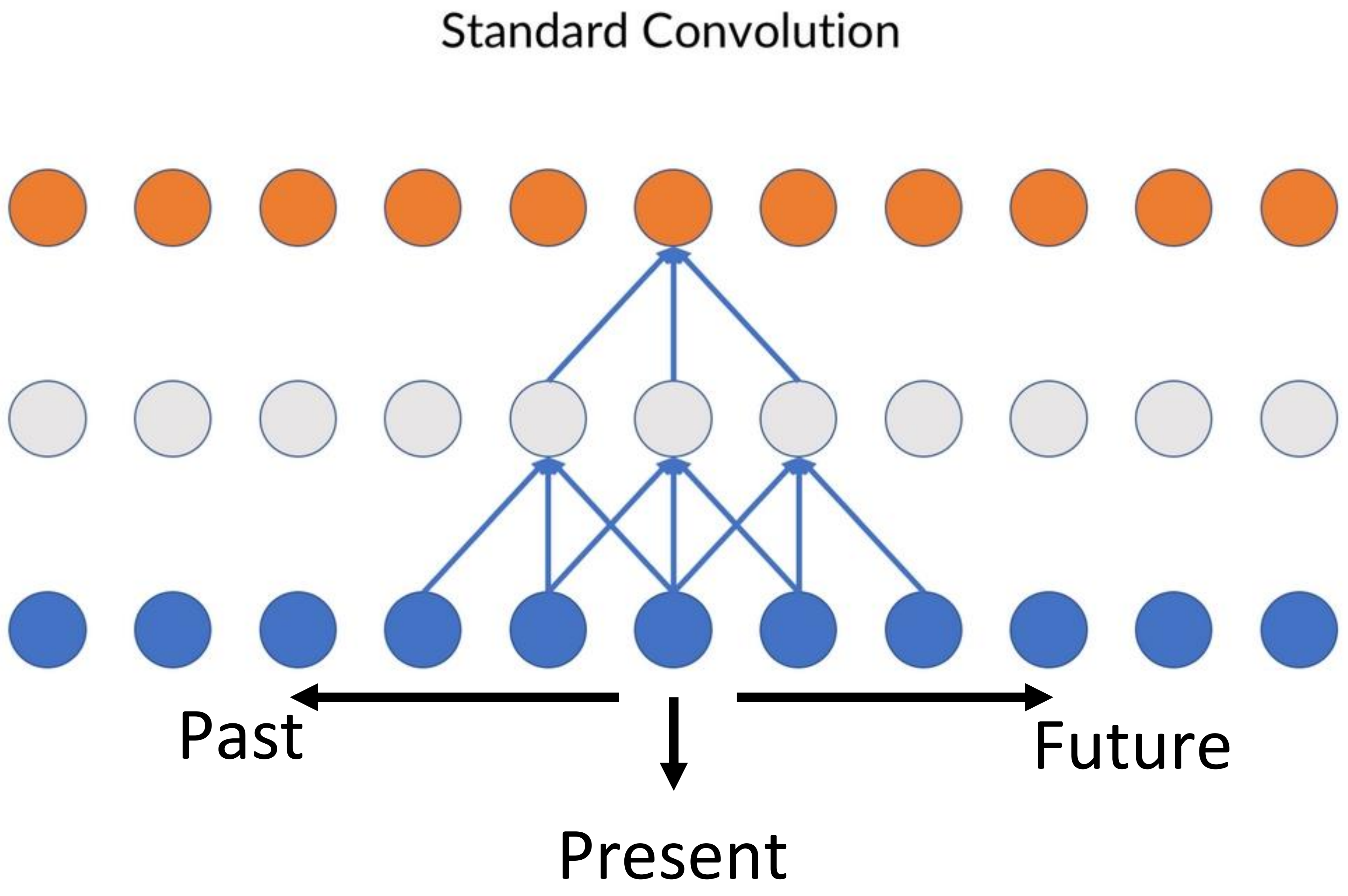
CNNs offer an interesting alternative for sequence modelling.

CONVOLUTION FOR TIME SERIES

Recall from Lecture 3 (CNNs) that **Conv1D** can be used for time-series.

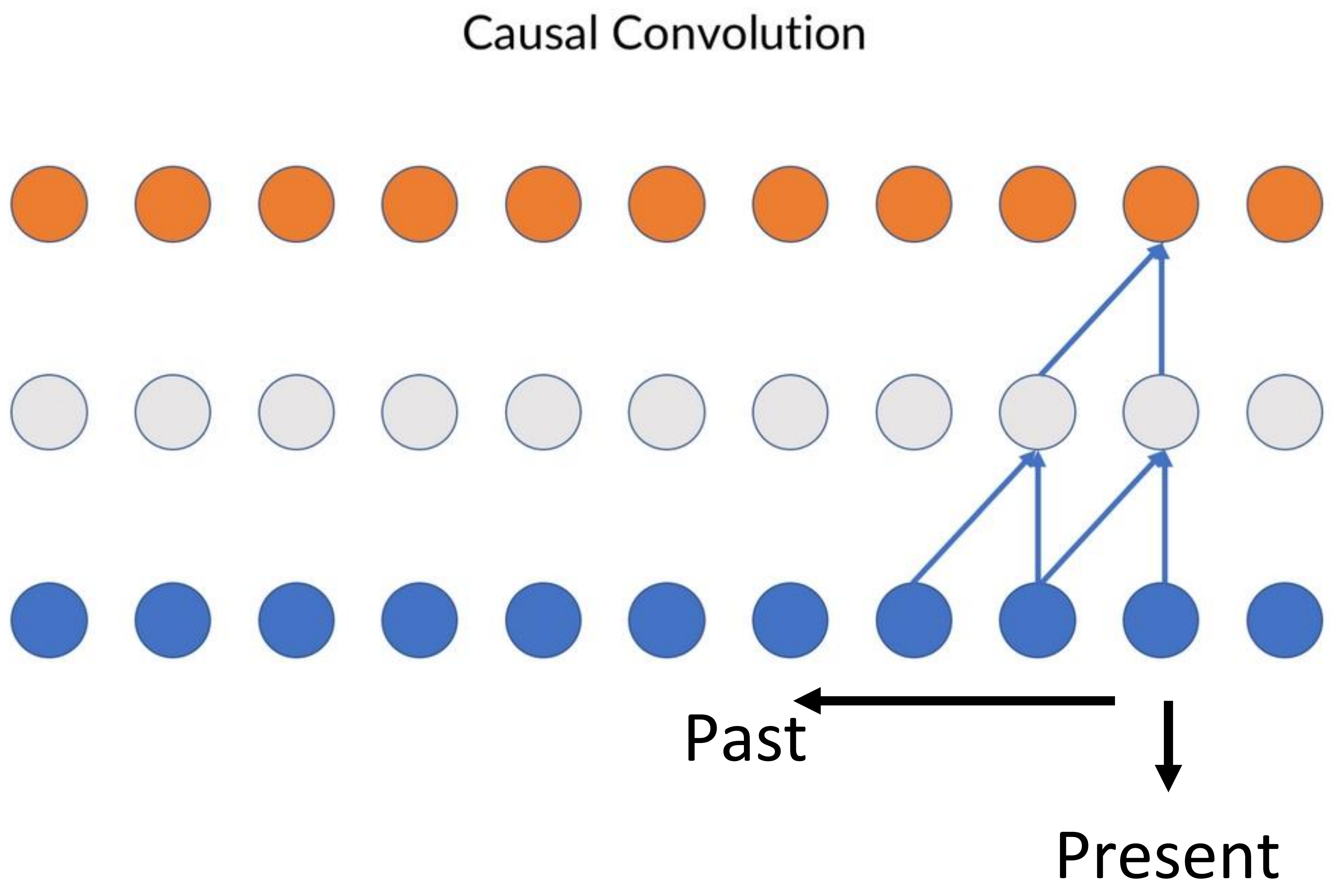
However, the standard convolution considers “future” values in the computation.

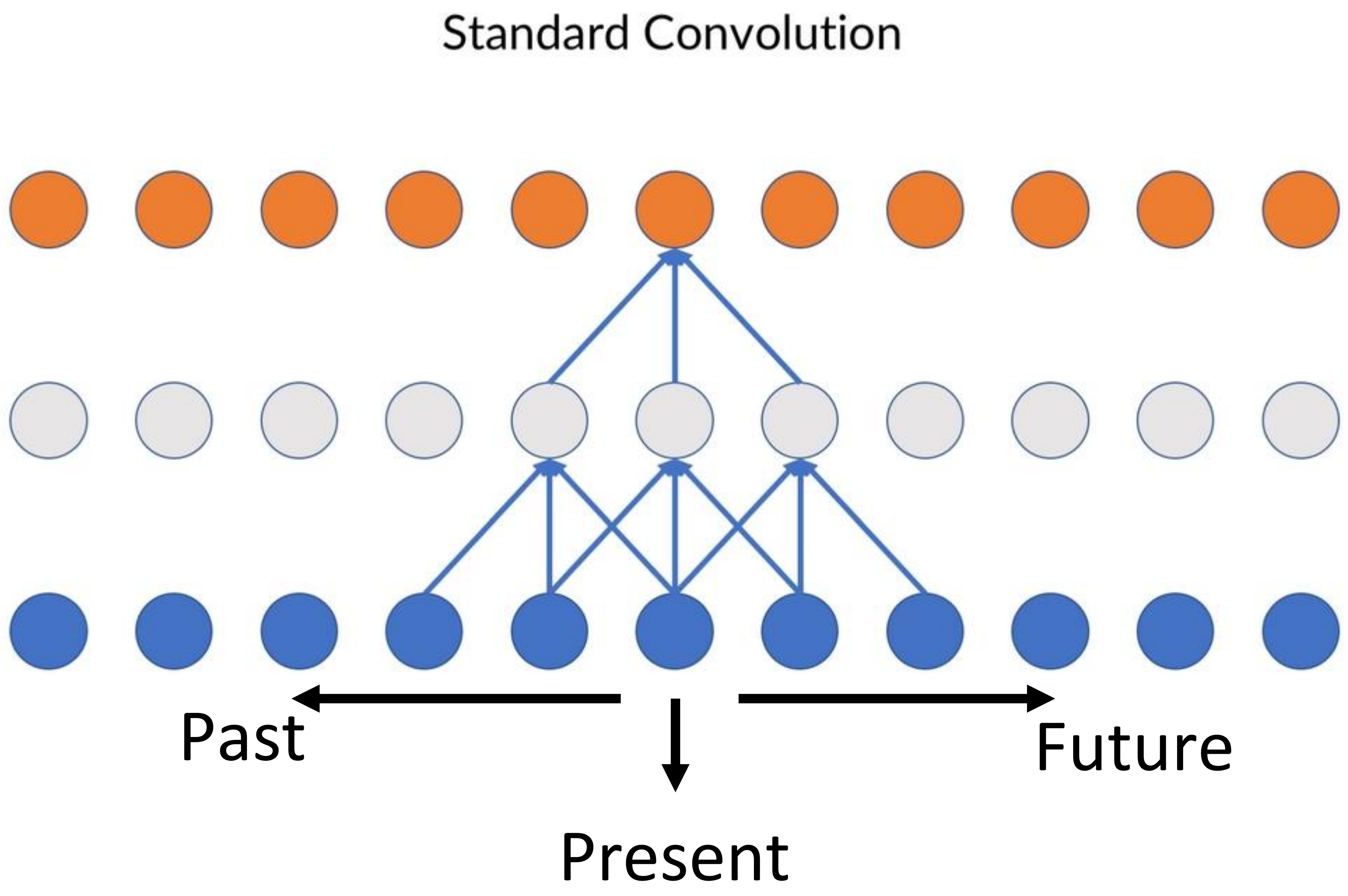
Unconvenient for several applications, e.g., sequential sampling, regression, ...



CAUSAL CONVOLUTION

Solved by providing a **causal formulation** to convolutions. That is, a formulation in which the present value only depends on past and present input values.

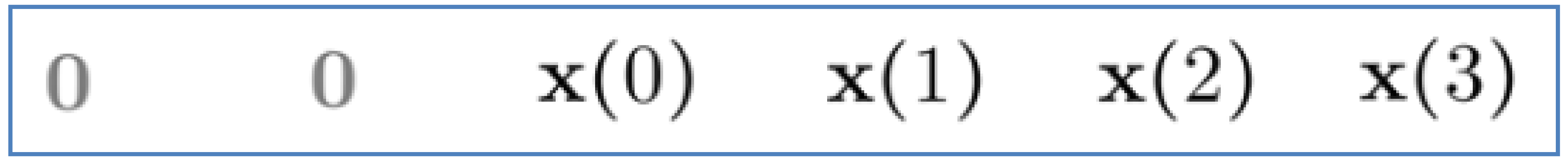




CAUSAL CONVOLUTION

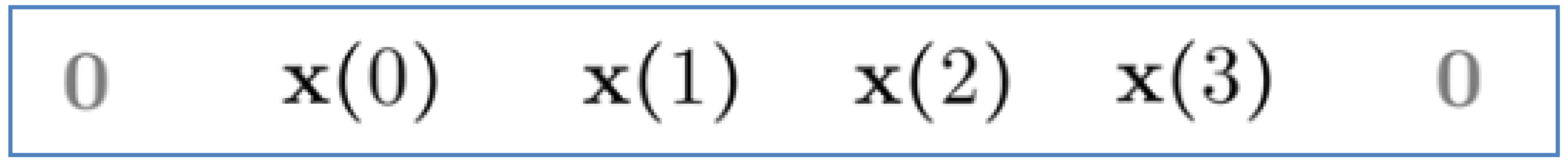
Causality is easily obtained by padding asymetrically.
For a convolutional kernel of size K add padding of $K - 1$ in the “past direction”.

For $K = 3$, pad as



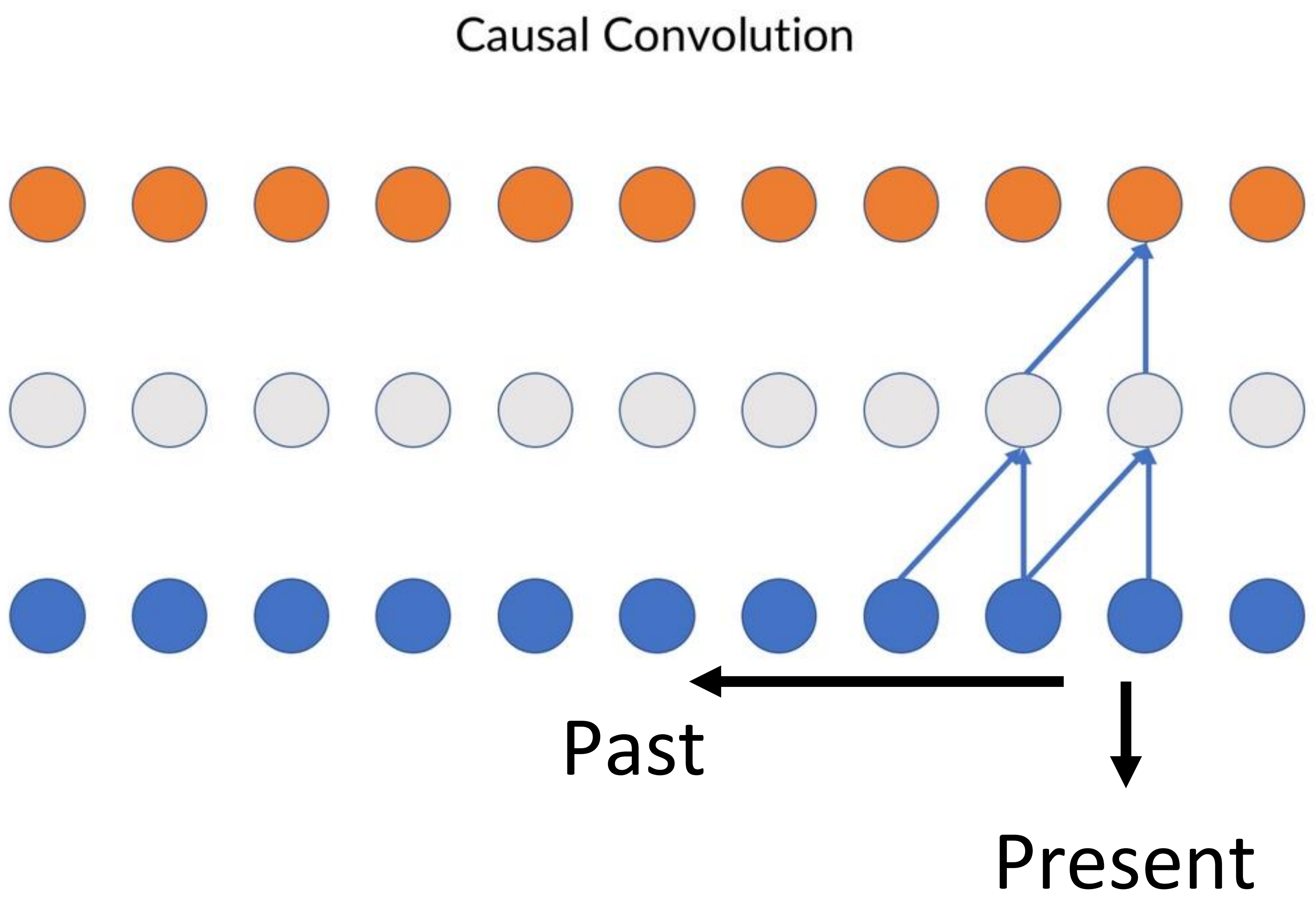
0	0	$\mathbf{x}(0)$	$\mathbf{x}(1)$	$\mathbf{x}(2)$	$\mathbf{x}(3)$

instead of



0	$\mathbf{x}(0)$	$\mathbf{x}(1)$	$\mathbf{x}(2)$	$\mathbf{x}(3)$	0

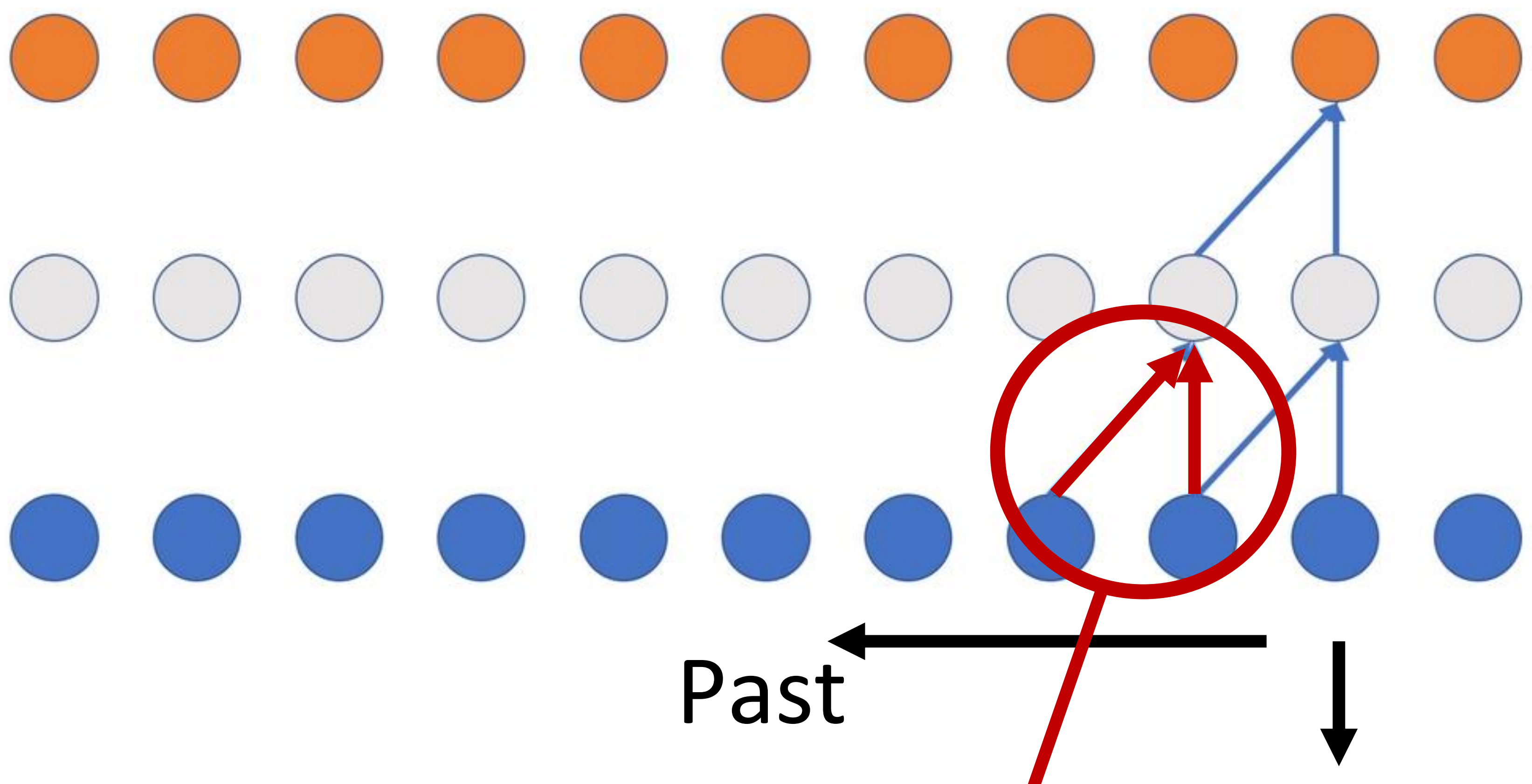
As a result, the convolutional kernel will only see present and past input values only.



THE RECEPTIVE FIELD

For a convolutional kernel of size K, the output at position t can be dependent on input values up to $K - 1$ steps in the past. The space it 'sees' is called **receptive field.**

Causal Convolution



For filter of size 2, up to 1 step back in the past !

How to deal with long range dependencies?

Option 1. **Large filters -> A lot of weights!**
(No parameter efficient).

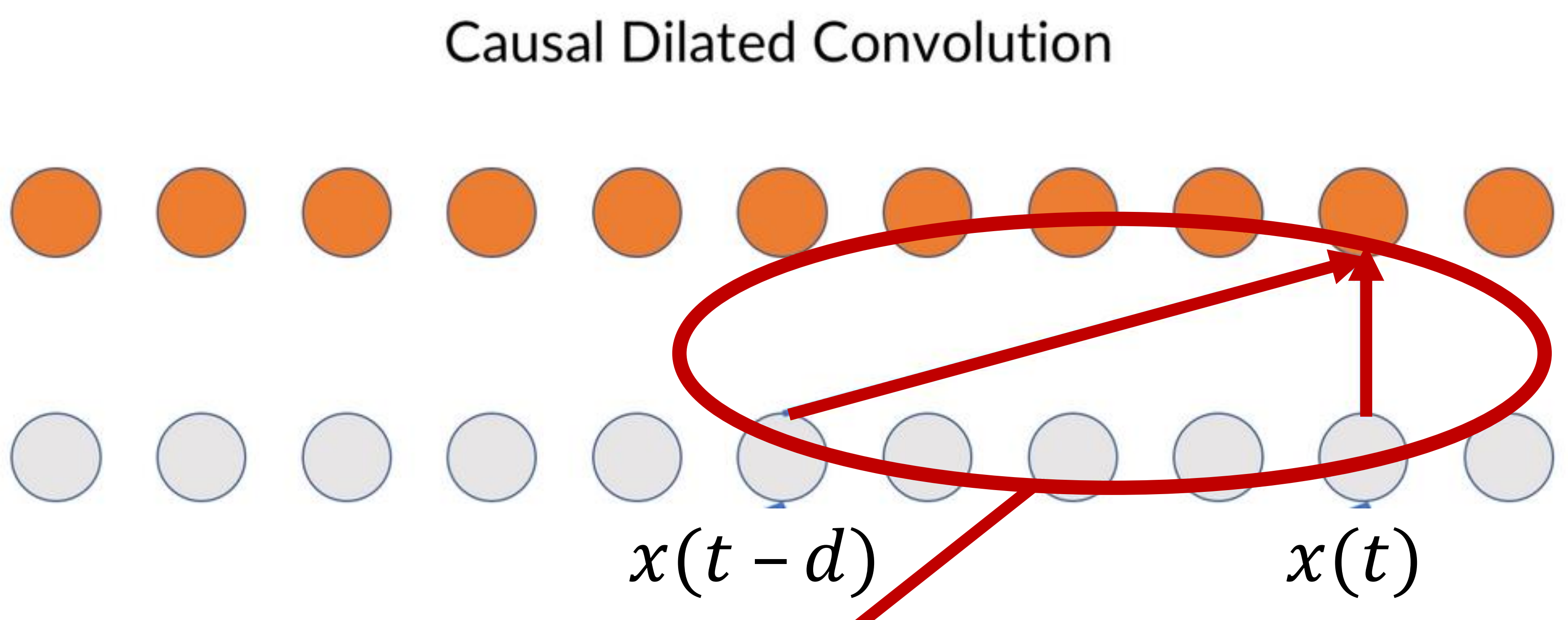
-- *sidenote* -- This is important because time-series are very long. A second of audio is often sampled at 22.05Khz. That is 22050 points per second of audio.

Is there any other option? **YES!**

Option 2. **Dilate the convolutional filter.**

DILATED CAUSAL CONVOLUTIONS

Dilate the convolutional filter of size K by a **dilation factor** d. The output at position t can be dependent on input values up to $d(K - 1)$ steps in the past.



For filter of size 2 and dilation factor of 4, up to 4 steps back in the past !

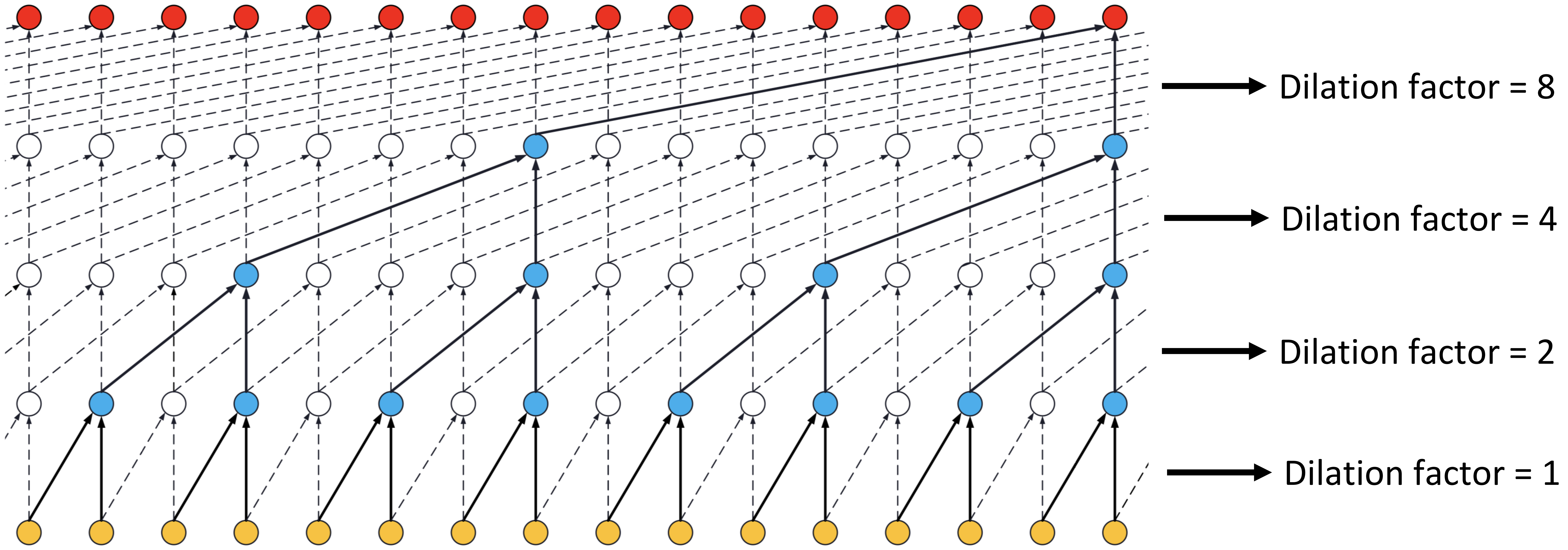
With dilated convolutions, we can look back far in time without increasing the number of weights.

Issues? **YES! -> Extreme sparsity. We cannot see input values between $x(t)$ and $x(t-d)$.**

Solution: Stack convolutions with different dilation factors.

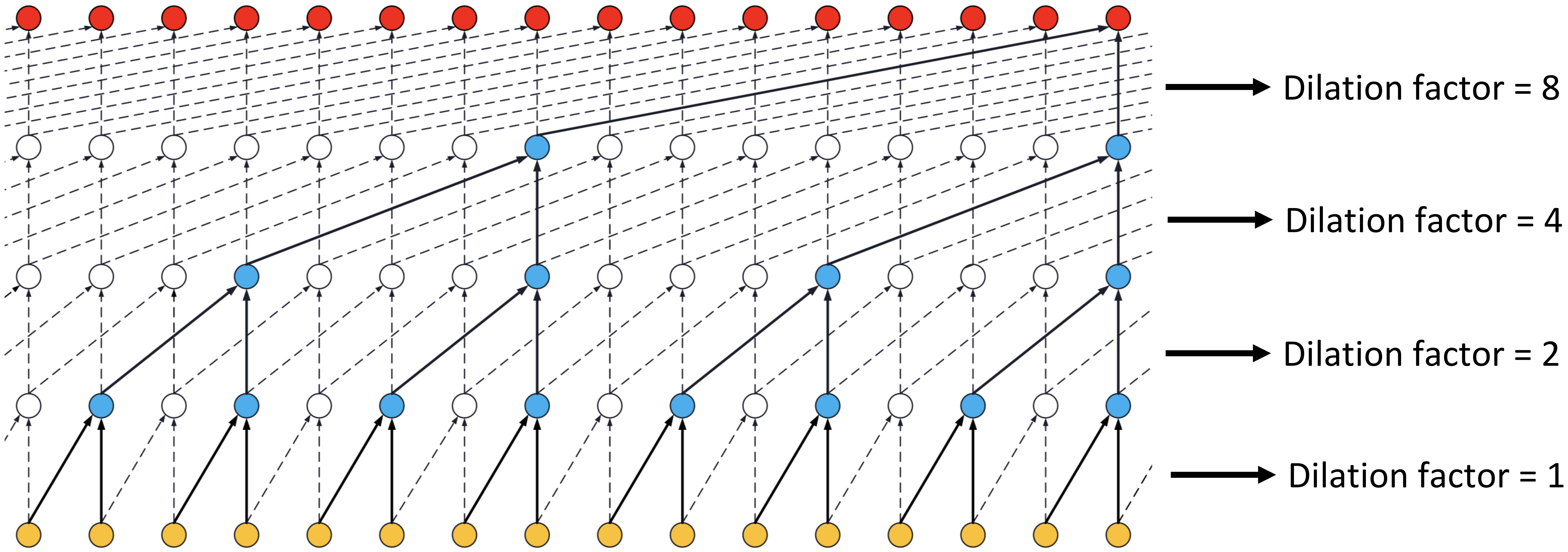
TEMPORAL CONVOLUTIONAL NETWORKS

We can stack several convolutional layers to form **Dilated Causal Convolutional Networks**, a.k.a., **Temporal Convolutional Networks (TCNs)**.



With exponentially growing receptive fields, we can observe all the input values within the receptive field of the entire network.

TEMPORAL CONVOLUTIONAL NETWORKS

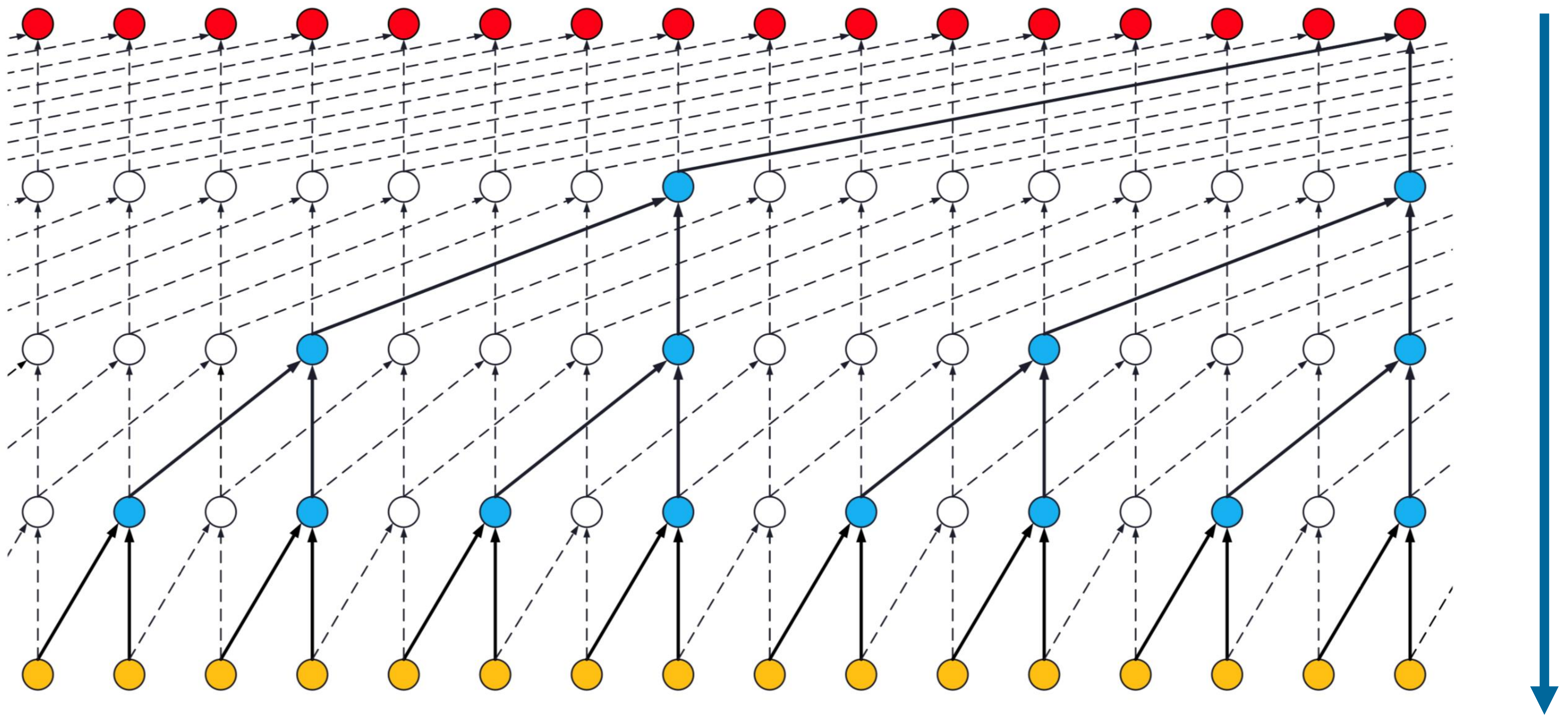


With the shown dilation scheme, the receptive field R of a TCN with l layers and convolutional kernels of size k is calculated as:

$$R = 2^l(k-1)$$

That is, the network can see all values up to $R-1$ steps in the past.

TEMPORAL CONVOLUTIONAL NETWORKS - LEARNING



TCNs have a different gradient flow direction than recurrent nets. Since they do not have recurrent connections, they **do not use Back-Propagation Trough Time!**. Hence:

1. They can be **trained in parallel** -> **Much faster training + optimal GPU usage**.
2. They **do not exhibit exploding / vanishing gradient problems along the time axis** -> They can **learn from the far past without problems** (for input values within their receptive fields).

TEMPORAL CONVOLUTIONAL NETWORKS

In comparison with recurrent architectures, TCNs bring the following **advantages**:

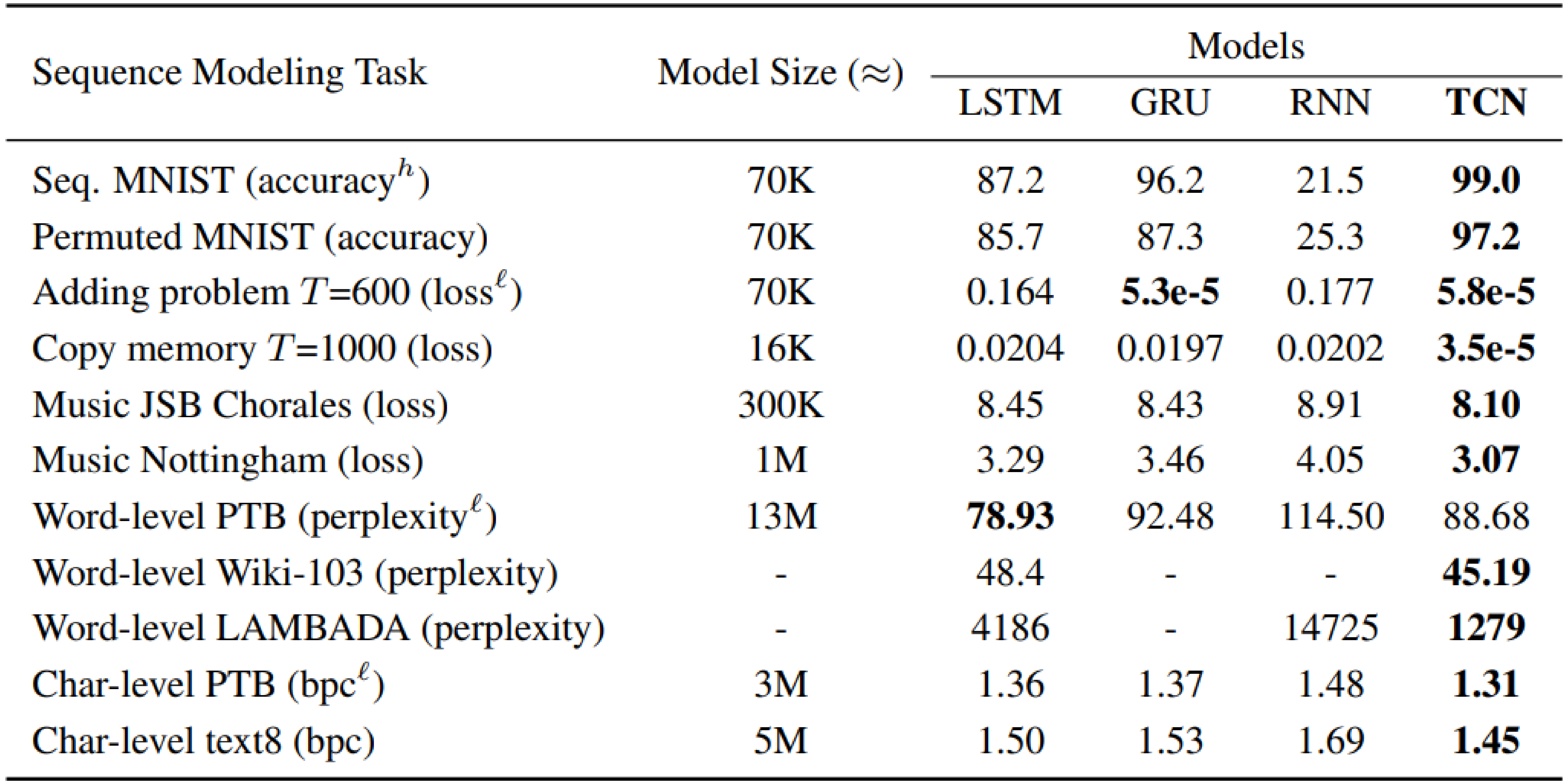
1. They can be **trained in parallel** -> **Much faster training + optimal GPU usage**.
2. They **do not exhibit exploding / vanishing gradient problems along the time axis** -> They can **learn from the far past without problems** (for input values within their receptive fields).

However, they present the following **disadvantages**:

1. The receptive field of TCNs is fixed a priori. Input values outside cannot be considered for the calculation of the output at a particular position.
2. TCNs cannot be unrolled for arbitrarily long inputs. Hence, they always see the input part within their receptive field as input.

TCNS IN PRACTICE

TCNs are broadly use in practice. Applications can be found for text, audio, time-series, recognition, classification, generative modelling, etc.



Sequence Modeling Task	Model Size (≈)	Models			
		LSTM	GRU	RNN	**TCN**
Seq. MNIST (accuracy^h)	70K	87.2	96.2	21.5	**99.0**
Permuted MNIST (accuracy)	70K	85.7	87.3	25.3	**97.2**
Adding problem T=600 (loss$^\ell$)	70K	0.164	**5.3e-5**	0.177	**5.8e-5**
Copy memory T=1000 (loss)	16K	0.0204	0.0197	0.0202	**3.5e-5**
Music JSB Chorales (loss)	300K	8.45	8.43	8.91	**8.10**
Music Nottingham (loss)	1M	3.29	3.46	4.05	**3.07**
Word-level PTB (perplexity$^\ell$)	13M	**78.93**	92.48	114.50	88.68
Word-level Wiki-103 (perplexity)	-	48.4	-	-	**45.19**
Word-level LAMBADA (perplexity)	-	4186	-	14725	**1279**
Char-level PTB (bpc$^\ell$)	3M	1.36	1.37	1.48	**1.31**
Char-level text8 (bpc)	5M	1.50	1.53	1.69	**1.45**

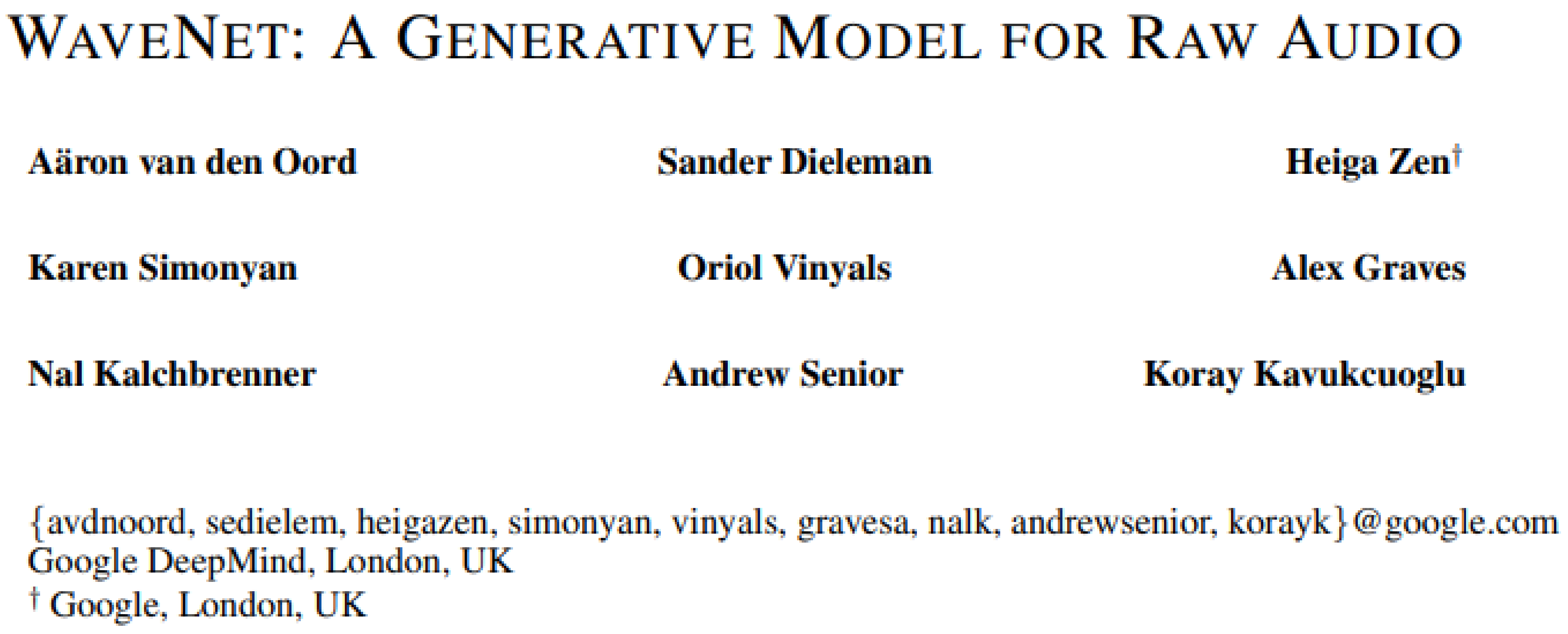
Bai et. al. ’18

TCNs outperform recurrent nets in their “home-turf”.

TCNS IN PRACTICE

TCNs are broadly use in practice. Applications can be found for text, audio, time-series, recognition, classification, generative modelling, etc.

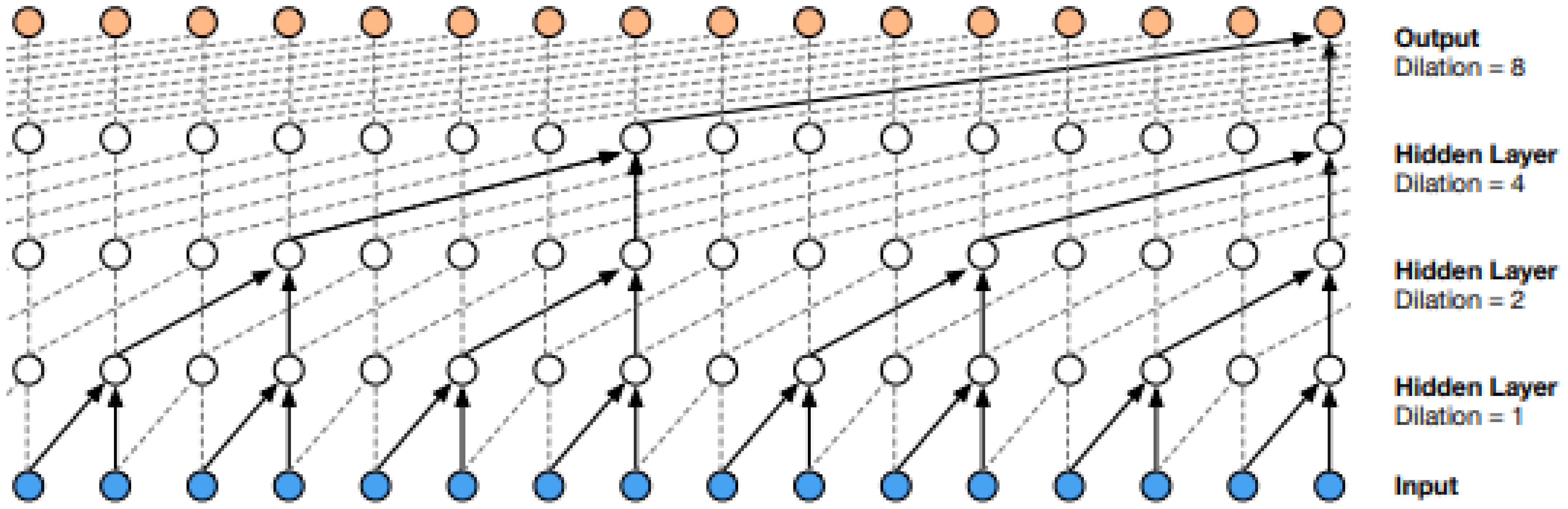
In fact you know and probably some of you use one such networks ;) Each time you say *“Ok Google …”* who answers you is:



WAVENET: A GENERATIVE MODEL FOR RAW AUDIO

Aäron van den Oord, Sander Dieleman, Heiga Zen[†]
Karen Simonyan, Oriol Vinyals, Alex Graves
Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu

{avdnoord, sedielem, heigazen, simonyan, vinyals, gravesa, nalk, andrewsenior, korayk}@google.com
Google DeepMind, London, UK
[†] Google, London, UK



You can find several cool examples at:
https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

CONSTRUCTING TEMPORAL CONVOLUTIONAL NETWORKS

Time-series have long dependencies. For example, a second of audio at 22.05KHz corresponds to 22050 samples in a single second.

With the dilation scheme presented before and convolutional filters of size $k = 7$, we require $\hat{l}$ layers in order to represent a single second of audio, where:

$$R = 2^{l}(k-1) \;\rightarrow 22050 = 2^{\hat{l}}(7-1) \;\leftrightarrow \hat{l} = \log_2(3675)$$
$$\hat{l} = 11.843$$

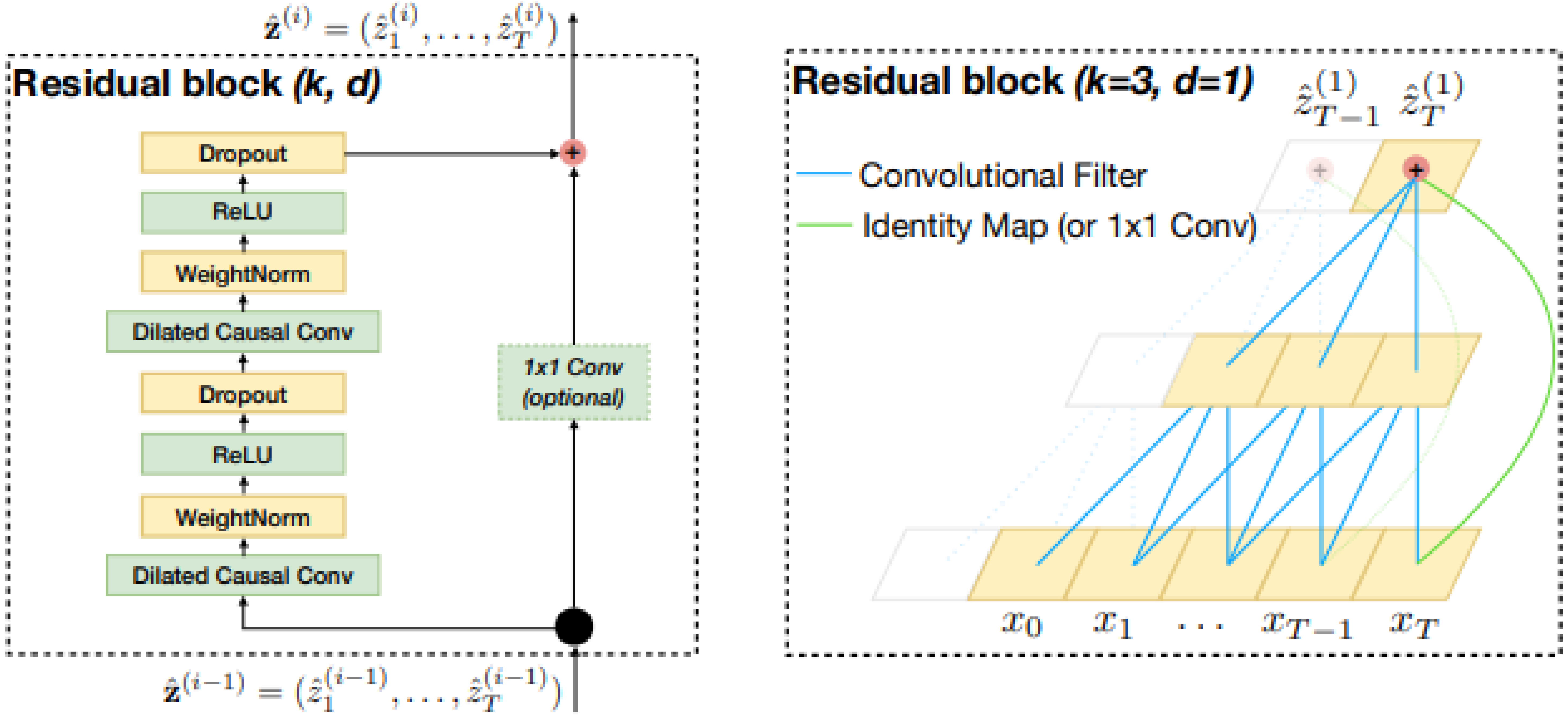
In other words, we need 12 layers to have a receptive field of 1 second. If receptive fields of multiple seconds are required, we need even deeper networks.

However, neural networks can present vanishing gradients if they are too deep (see video 4 of lecture 4). **How can we train TCNs? -> We need some tricks!**

CONSTRUCTING TEMPORAL CONVOLUTIONAL NETWORKS

In order to avoid vanishing gradients and improve learning, TCNs use **batch normalization, residual connections** and (optionally) **dropout**. (see video 4 of lecture 4).

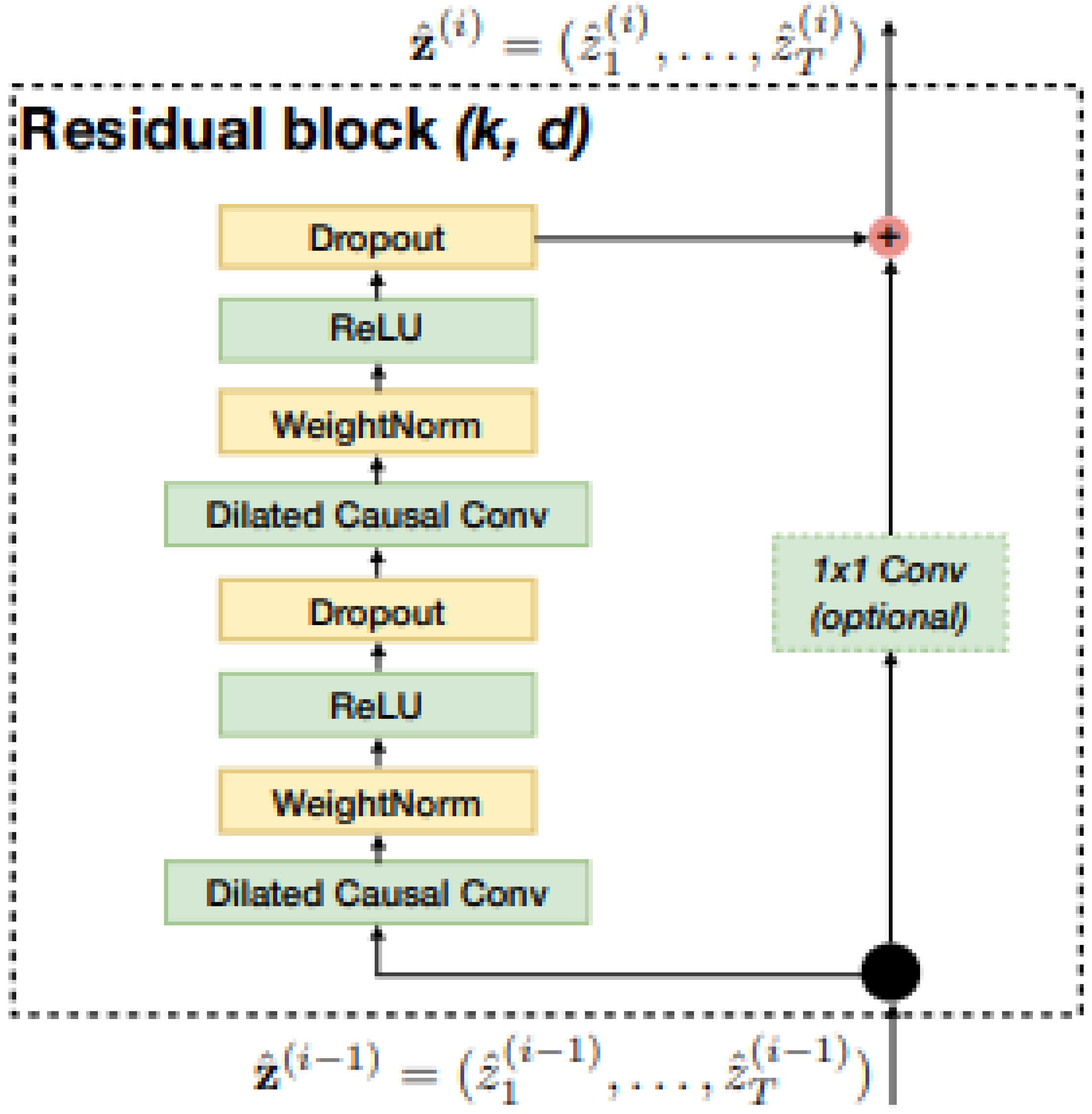
A single residual block looks as:



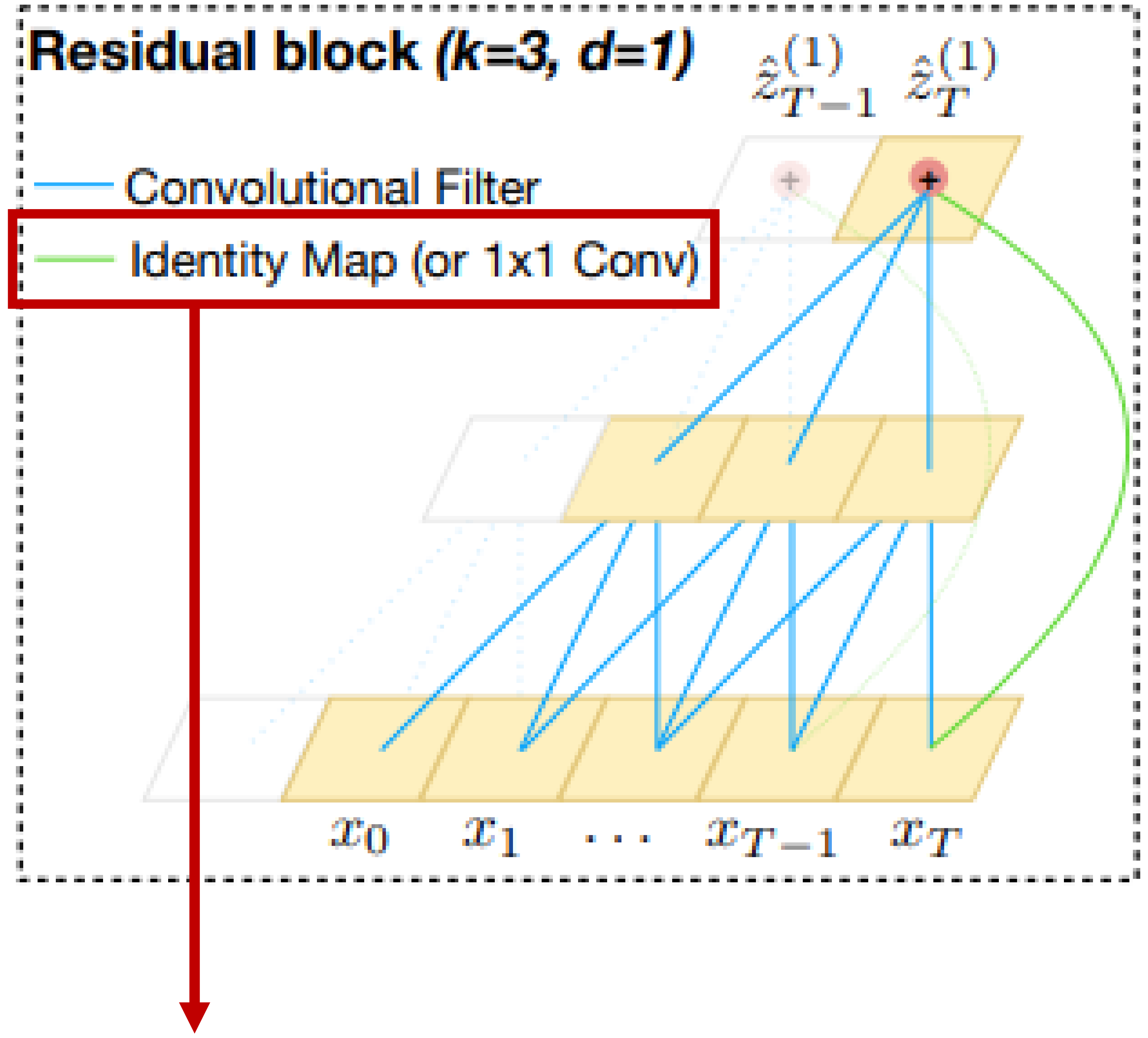
CONSTRUCTING TEMPORAL CONVOLUTIONAL NETWORKS

In order to avoid vanishing gradients and improve learning, TCNs use **batch normalization, residual connections** and (optionally) **dropout**. (see video 4 of lecture 4).

A single residual block looks as:



$\hat{\mathbf{z}}^{(i)} = (\hat{z}_1^{(i)}, \ldots, \hat{z}_T^{(i)})$
Residual block (k, d)
Dropout
ReLU
WeightNorm
Dilated Causal Conv
Dropout
ReLU
WeightNorm
Dilated Causal Conv
1x1 Conv (optional)
$\hat{\mathbf{z}}^{(i-1)} = (\hat{z}_1^{(i-1)}, \ldots, \hat{z}_T^{(i-1)})$

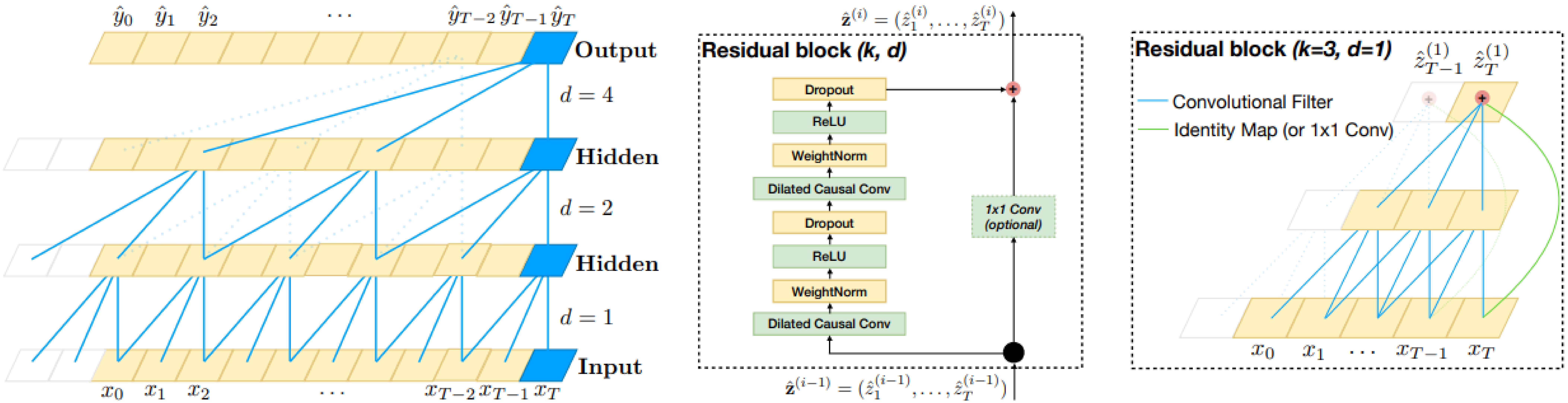


Residual connection

CONSTRUCTING TEMPORAL CONVOLUTIONAL NETWORKS

In order to avoid vanishing gradients and improve learning, TCNs use **batch normalization, residual connections** and (optionally) **dropout** . (see video 4 of lecture 4).

And the residual blocks can be stacked as before:


$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ $\cdots$ $\hat{y}_{T-2}$ $\hat{y}_{T-1}$ $\hat{y}_T$
Output
$d = 4$
Hidden
$d = 2$
Hidden
$d = 1$
Input
x_0 x_1 x_2 $\cdots$ x_{T-2} x_{T-1} x_T
$\hat{\mathbf{z}}^{(i)} = (\hat{z}_1^{(i)}, \ldots, \hat{z}_T^{(i)})$
Residual block (k, d)
Dropout
ReLU
WeightNorm
Dilated Causal Conv
Dropout
ReLU
WeightNorm
Dilated Causal Conv
1x1 Conv (optional)
$\hat{\mathbf{z}}^{(i-1)} = (\hat{z}_1^{(i-1)}, \ldots, \hat{z}_T^{(i-1)})$
Residual block (k=3, d=1)
$\hat{z}_{T-1}^{(1)}$ $\hat{z}_T^{(1)}$
Convolutional Filter
Identity Map (or 1x1 Conv)
x_0 x_1 $\cdots$ x_{T-1} x_T

SUMMARY

TCNs are an strong alternative to recurrent networks.

They present some important improvements and some important limitations.

They are often used in practice and have found a lot of important applications.

Selecting the current method is dependent on the task at hand. **But** TCNs have a lot of potential in making recurrent nets “obsolete”.*

TCNs seem to be a lightweight contender of Transformer networks (Lecture 12).

* We are currently doing reserach in this direction. New paper at the QDA group to come out soon. If you are interested and would like to write your master thesis in this topic, let us know ;)