

CNNs part II leftovers & Recurrent Neural Networks

Beate Sick, Oliver Dürr, Elvis Murina

Institut für Datenanalyse und Prozessdesign

Zürcher Hochschule für Angewandte Wissenschaften

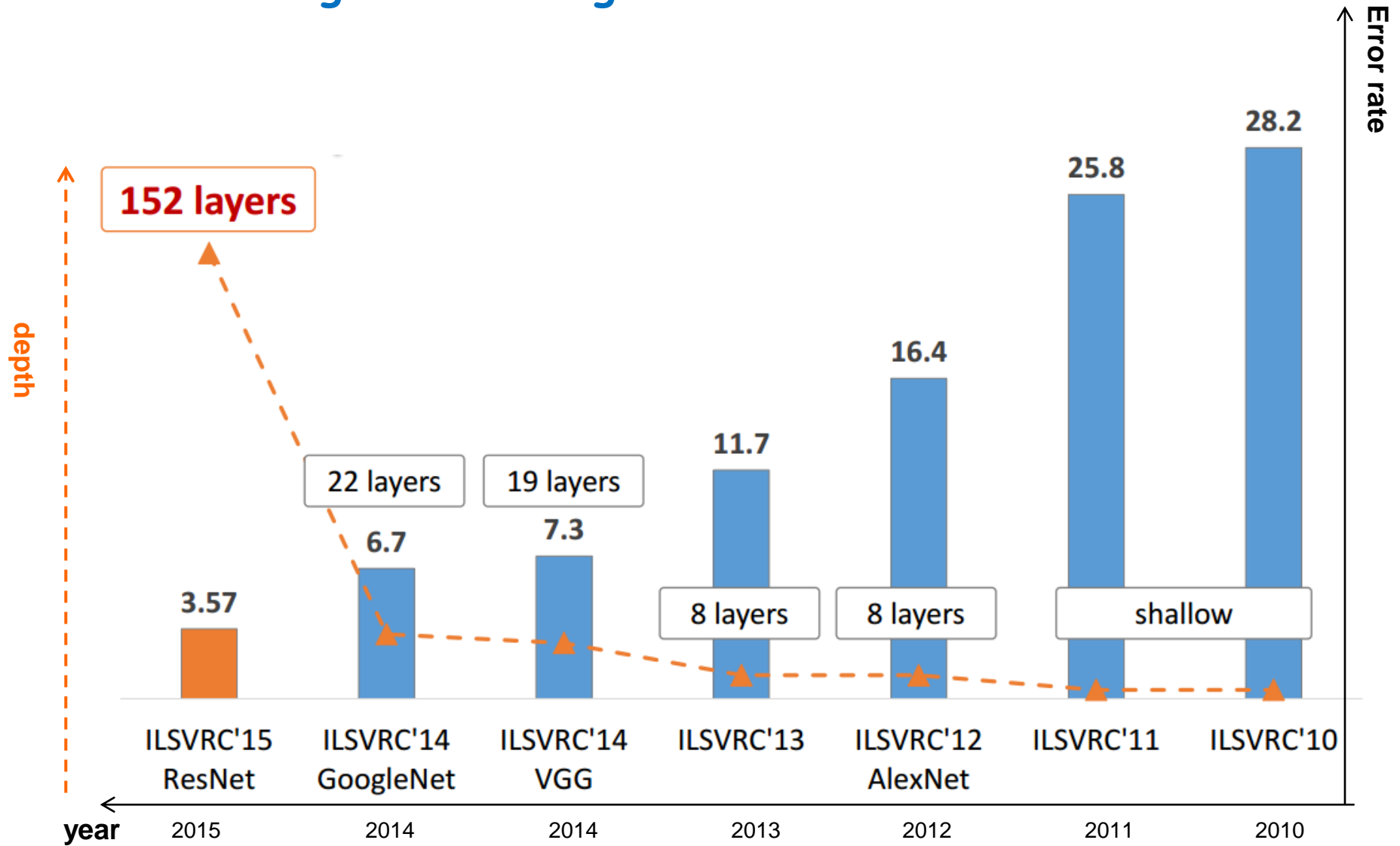
sick@zhaw.ch

Topics

- **Famous CNN architectures**
 - LeNet, AlexNet, GoogleNet, VGG, Microsoft ResNet
- **Introduction of recurrent neural networks (RNN)**
 - possible use cases
 - memory in recurrent NN
- **Working with an RNN**
 - stepping through an RNN
 - loss construction in an RNN
 - RNNs in Keras
- **Tricks of the trade**
 - common and different tricks to train deep CNNs and RNNs

Modern CNN architectures

Review of ImageNet winning CNN architectures



Going deeper is easy – or not?



The challenge is to design a network in which the gradient can reach all the layers of a network which might be dozens, or even hundreds of layers deep.

This was achieved by some recent improvements, such as ReLU and batch normalization, and by designing the architecture in a way which allows the gradient to reach deep layer, e.g. by additional skip connections.

LeNet-5 1998: first CNN for ZIP code recognition

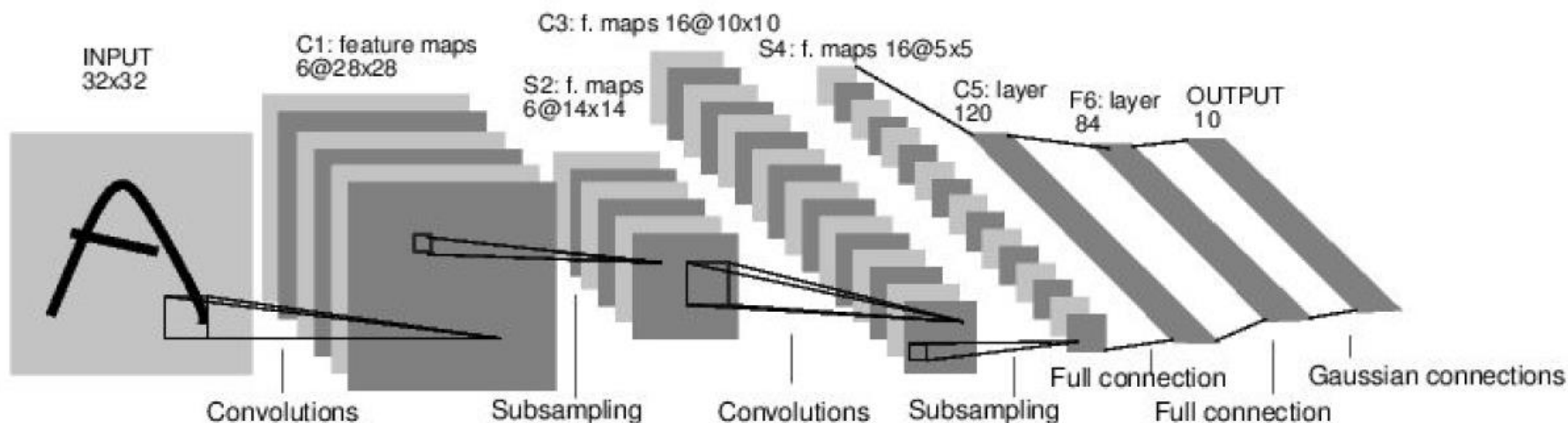
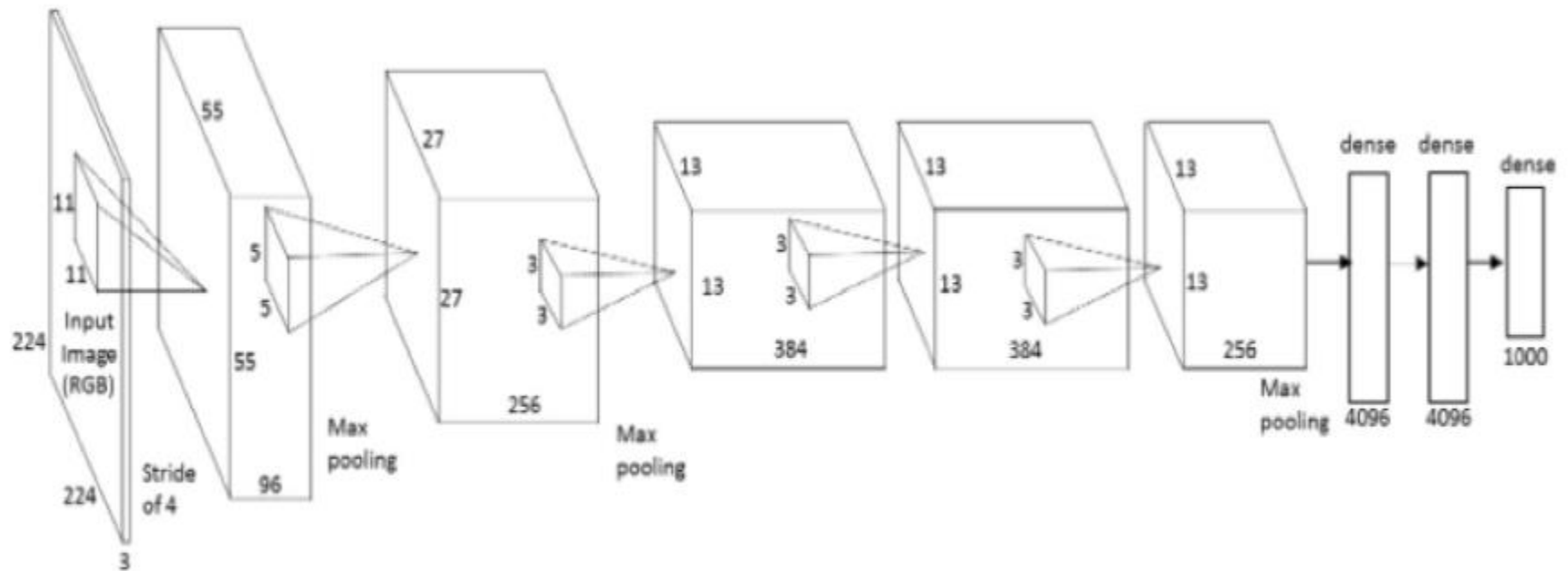


Image credits: <http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>

Conv filters were 5×5 , applied at stride 1
Subsampling (Pooling) layers were 2×2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

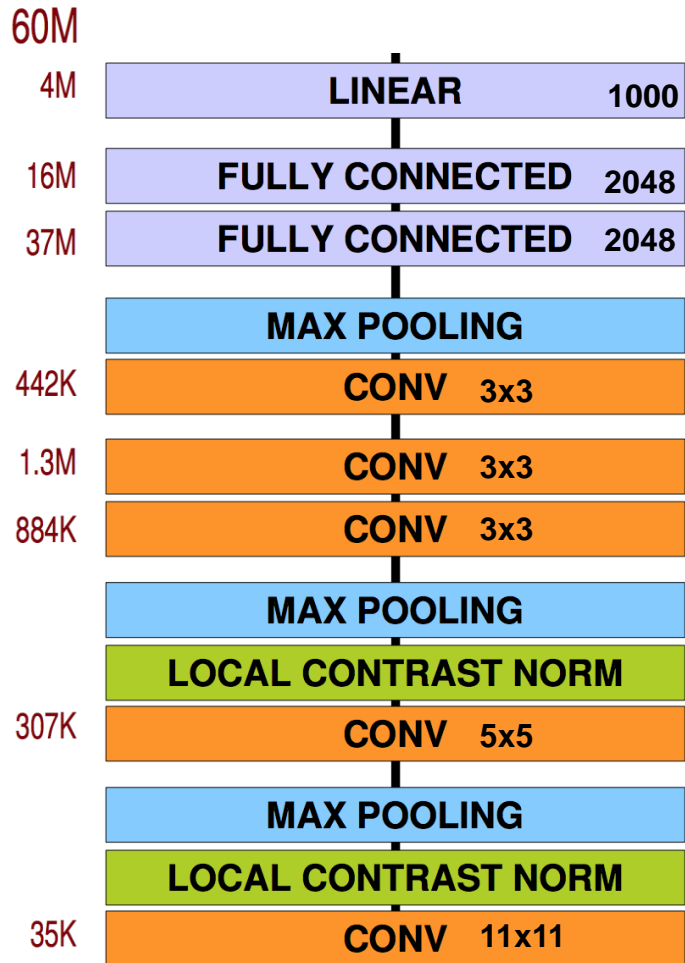
<http://yann.lecun.com/exdb/lenet/index.html>

"AlexNet", 2012 winner of the imageNet challenge



Architecture of AlexNet

"AlexNet", 2012 winner of the imageNet challenge



Seminal paper. 26.2% error → 16.5%

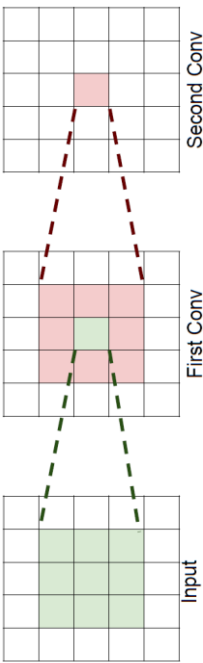
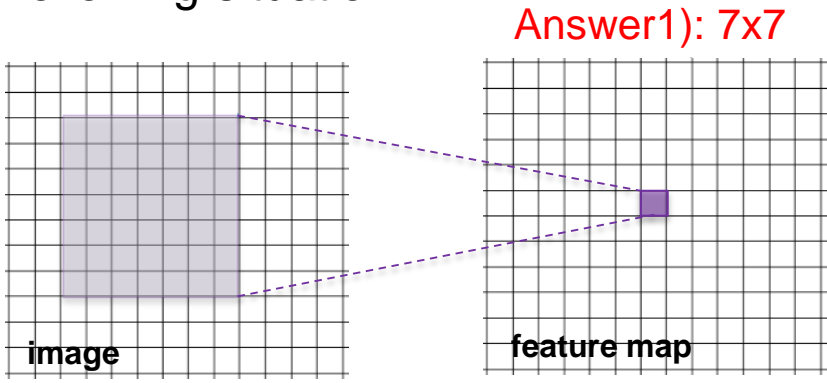
- Dropout
- ReLU instead of sigmoid
- Used data augmentation techniques
- 5 conv layer (filter: 11x11, 5x5, 3x3, 3x3, 3x3)
- Parallelisation on two GPUs
- Local Response Normalization (not used anymore)

The trend in modern CNN architectures goes to small filters

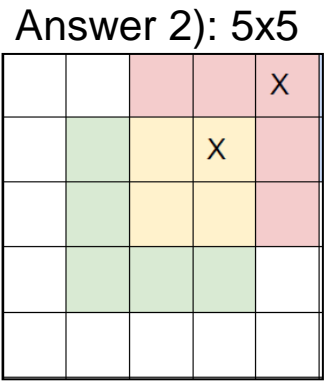
Why do modern architectures use very small filters?

Determine the receptive field in the following situation:

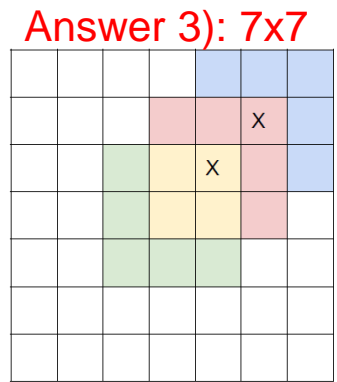
1) Suppose we have **one**
7x7 conv layers (stride 1)
49 weights



2) Suppose we **stack two**
3x3 conv layers (stride 1)



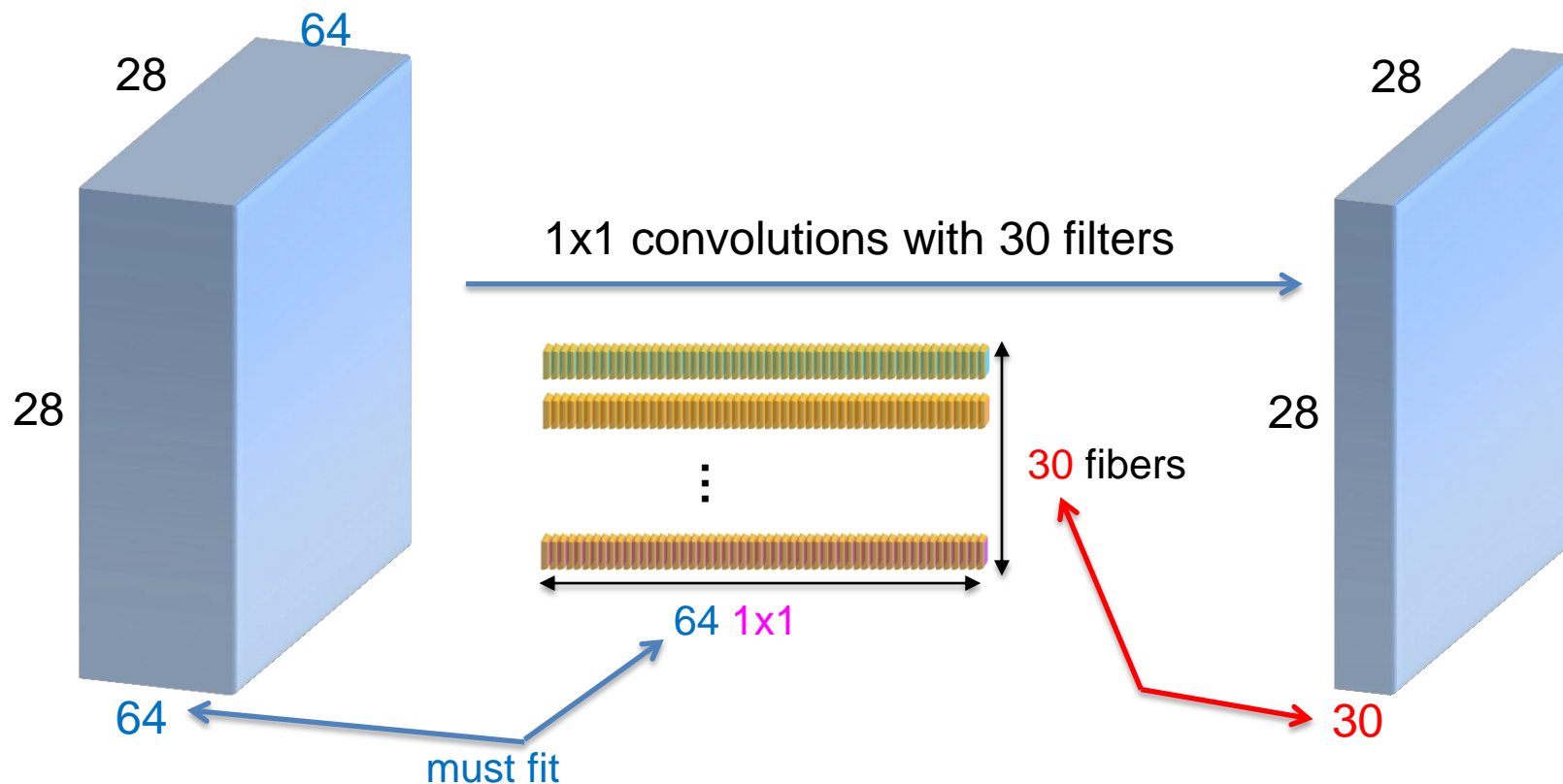
3) Suppose we **stack three**
3x3 conv layers (stride 1)
3*9=27 weights



We need less weights for the same receptive field when stacking small filters!

Go to the extreme: What is about filter size 1?

1x1 convolutions act only in depth dimension

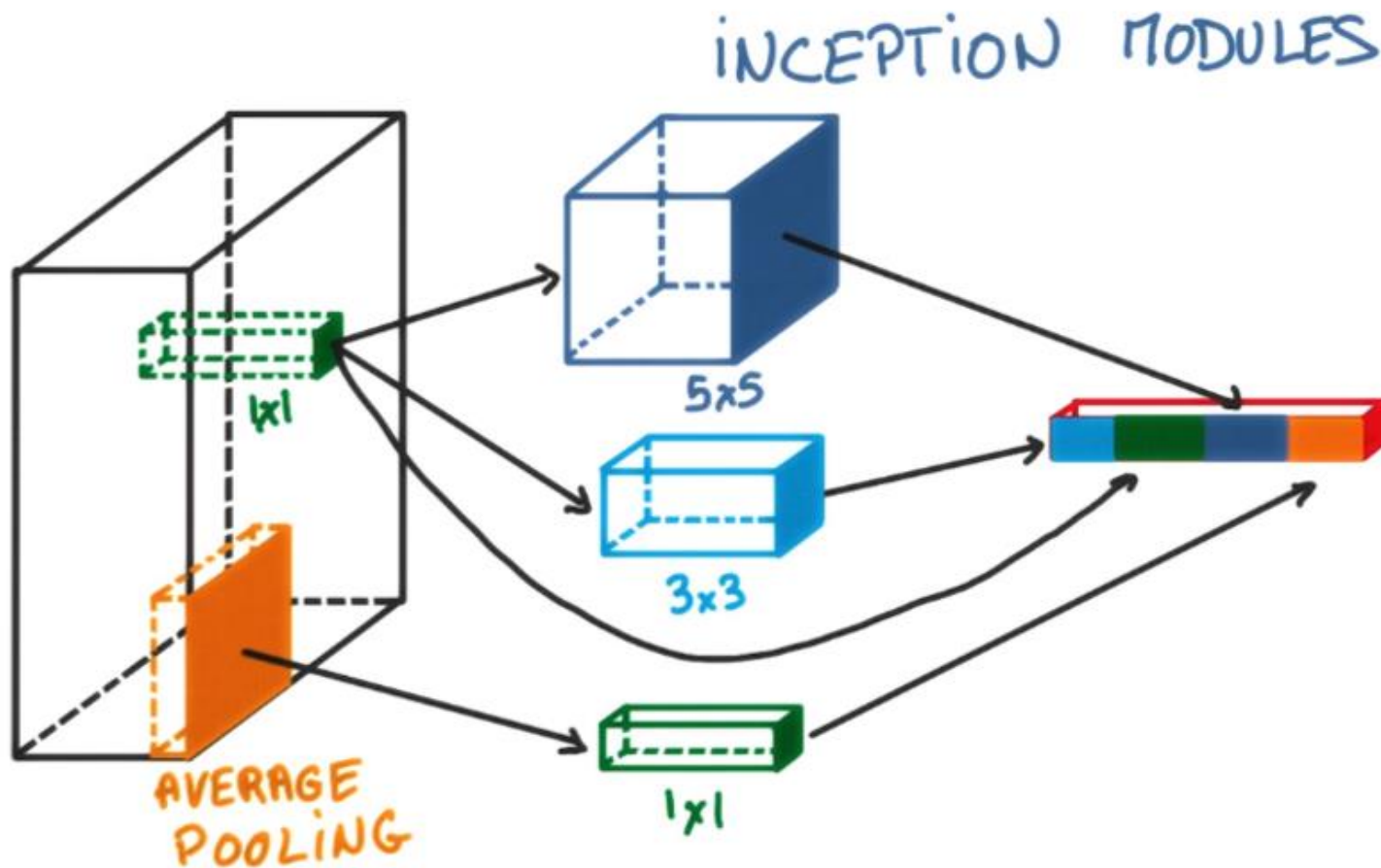


1x1 convolution act along a “fiber” in depth dimension across the channels.

→ efficient way to reduce/change the depth dimension

→ simultaneously introduce more non-linearity

The idea of inception modules



Between two layers just **do several operations in parallel**: pooling and 1×1 conv, and 3×3 and 5×5 . “same”-conv and concatenate them together. Benefit: total number of parameters is small, yet performance better.

How to concatenate tensors of different dimensions?

DepthConcat needs to make the tensor the same in all, but the depth dimension:

To deal with different output dimensions, the largest spatial dimension is select and **zero-padding around the smaller dimensions is added**.

Pseudo code for an example:

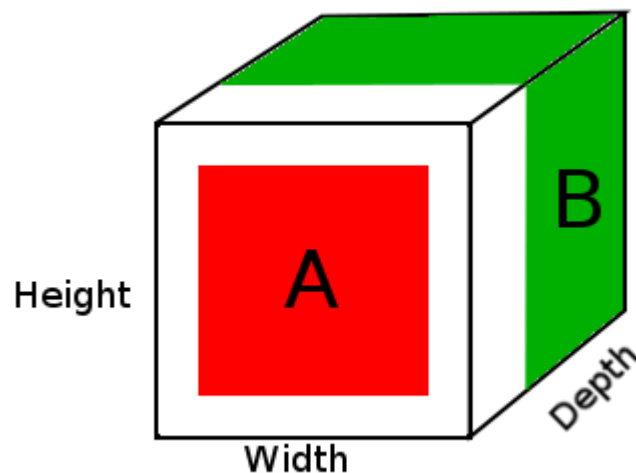
```
A = tensor of size (14, 14, 2)
```

```
B = tensor of size (16, 16, 3)
```

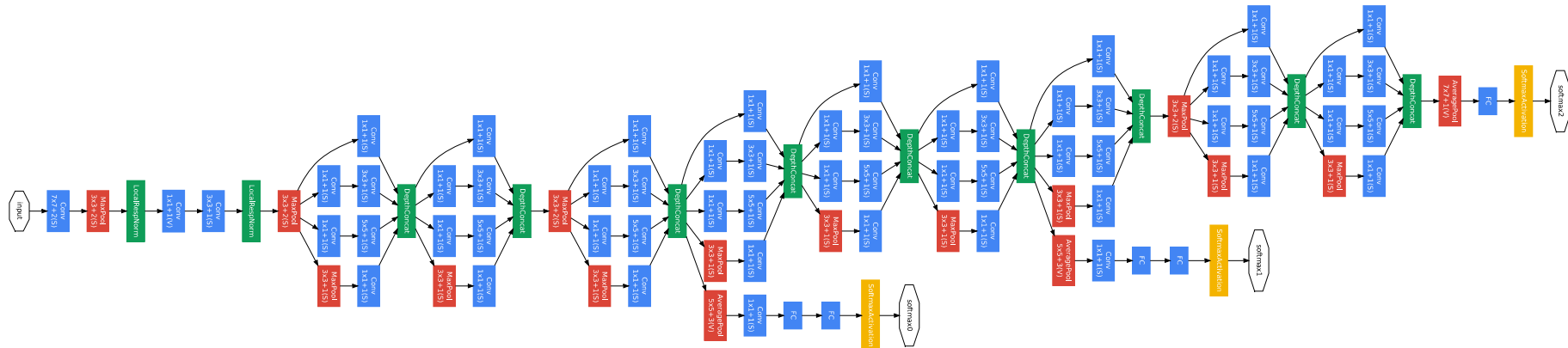
```
result = DepthConcat([A, B])
```

where result will have a
height of 16,
width of 16 and a
depth of 5 (2 + 3)

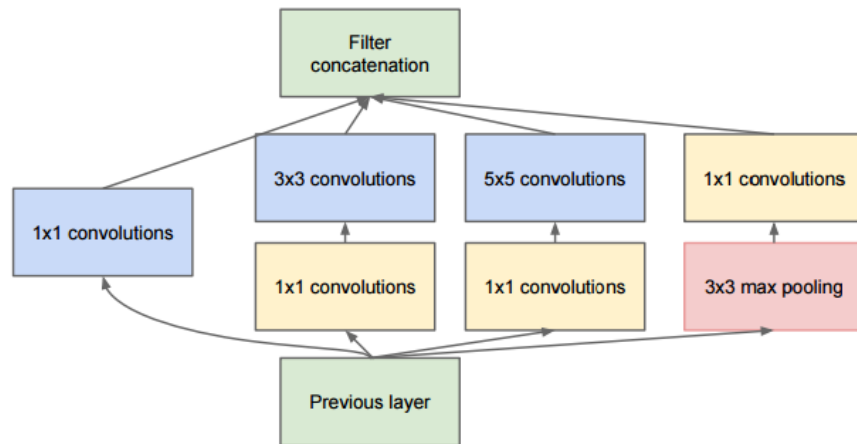
Usually depth gets large!
Do 1x1 conv afterwards!



Winning architecture (GoogLeNet, 2014)

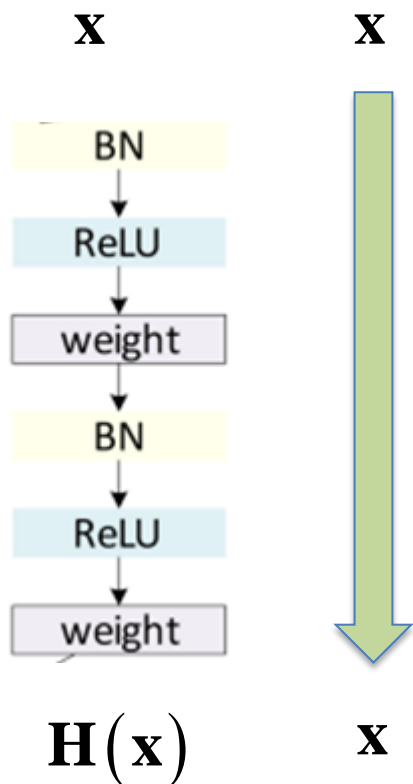


The inception module (convolutions and maxpooling)



Few parameters, hard to train.
Comments see [here](#)

Highway Networks: providing a highway for the gradient



Idea: Use nonlinear transform T to determine how much of the output \mathbf{y} is produced by H or the identity mapping. Technically we do that by:

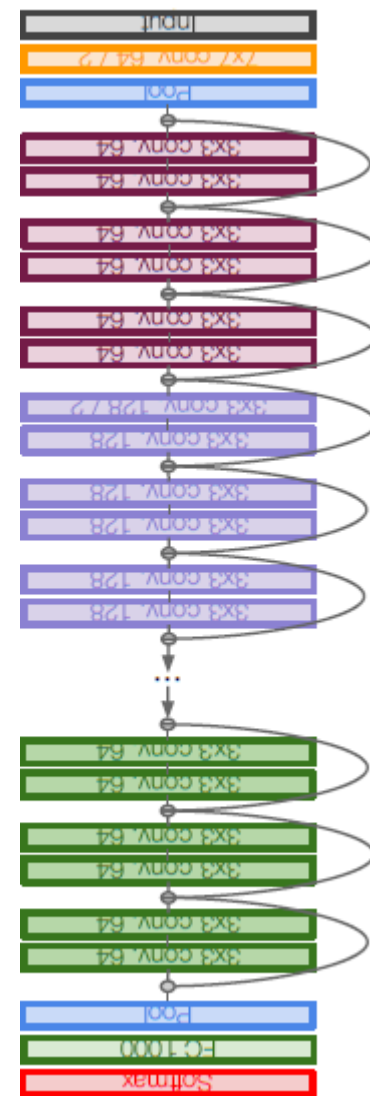
$$\mathbf{y} = H(\mathbf{x}, \mathbf{W}_H) \cdot T(\mathbf{x}, \mathbf{W}_T) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W}_T)).$$

Special case:

$$\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W}_T) = 0 \\ H(\mathbf{x}, \mathbf{W}_H), & \text{if } T(\mathbf{x}, \mathbf{W}_T) = 1 \end{cases}$$

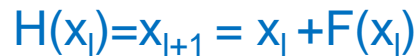
This opens a highway for the gradient:

$$\frac{d\mathbf{y}}{d\mathbf{x}} = \begin{cases} \mathbf{I}, & \text{if } T(\mathbf{x}, \mathbf{W}_T) = 0, \\ H'(\mathbf{x}, \mathbf{W}_H), & \text{if } T(\mathbf{x}, \mathbf{W}_T) = 1. \end{cases}$$



152
layers

- add shortcut connections every two
- all 3x3 conv (almost)



This deep architecture could still be trained, since the gradients can skip layers which diminish the gradient!



“Oxford Net” or “VGG Net” 2014 2nd place

- 2nd place in the imageNet challenge
- More **traditional**, easier to train
- More weights than GoogLeNet
- Small pooling
- **Stacked 3x3 convolutions before maxpooling**
-> **large receptive field**
- no strides (stride 1)
- ReLU after conv. and FC (batchnorm was not used)
- Pre-trained model is available

<http://arxiv.org/abs/1409.1556>



wiederkehrend

Recurrent Neural Networks

Resources

- Many figures are taken from the following resources:
 - **Deep Learning Book** chap10
 - <http://www.deeplearningbook.org/contents/rnn.html>
 - Other online DL courses
 - Lecture on RNN: http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf
 - Video to CS231n <https://www.youtube.com/watch?v=iX5V1WpxxkY>
 - [CS 598 LAZ](#) Lecture 2 and 3 on RNN
 - Blog Posts
 - Karpathy, May 2015: The unreasonable effectiveness of Recurrent Neural Networks <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
 - Colah, August 2015: Understanding LSTM Networks <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Teaser: Recurrent NN for language translation



Speech Recognition Breakthrough for the Spoken, Translated Word

2012: [Microsoft Chief Research Officer Rick Rashid demonstrates](#) breakthrough in DL based translation that converts his spoken English words into computer-generated Chinese language.

2017: Language translator is available as [mobile app](#)

Teaser: Recurrent NN for Image Captioning



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."



"boy is doing backflip on wakeboard."



"a young boy is holding a baseball bat."



"a cat is sitting on a couch with a remote control."

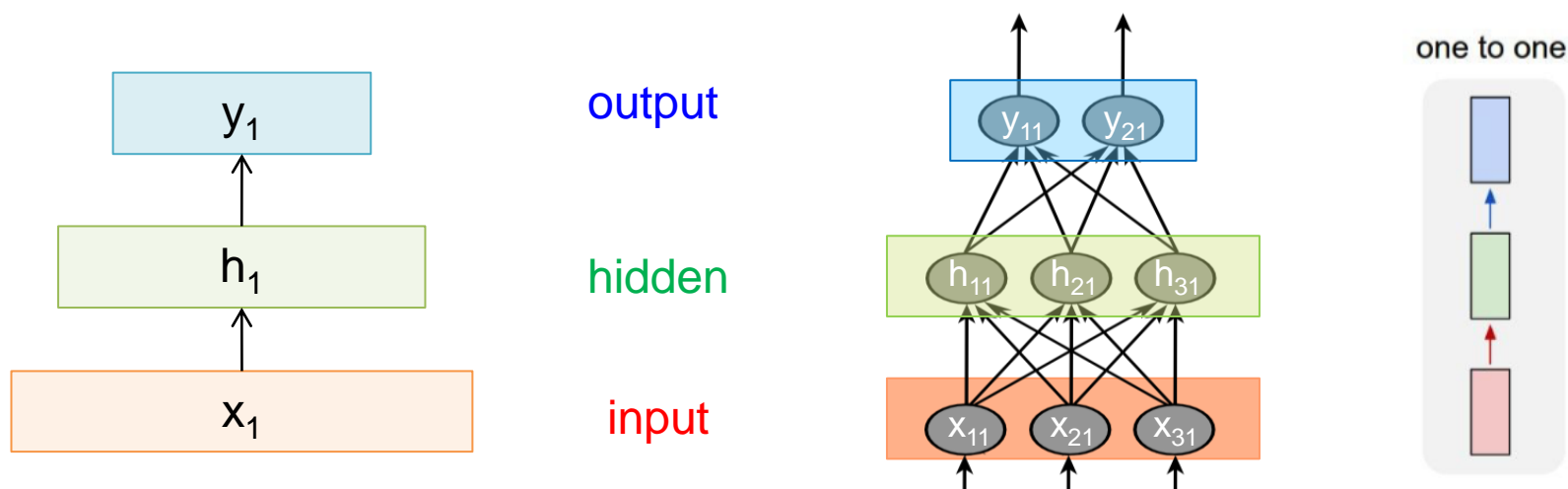


"a woman holding a teddy bear in front of a mirror."



"a horse is standing in the middle of a road."

Task for a simple feed forward network



Classification task:
“happily surprised” or “angrily surprised”?

Architecture type: fcNN, CNN (preferred)

hidden layer: #hierarchies (complexity)

Size of hidden layer \sim #features needed



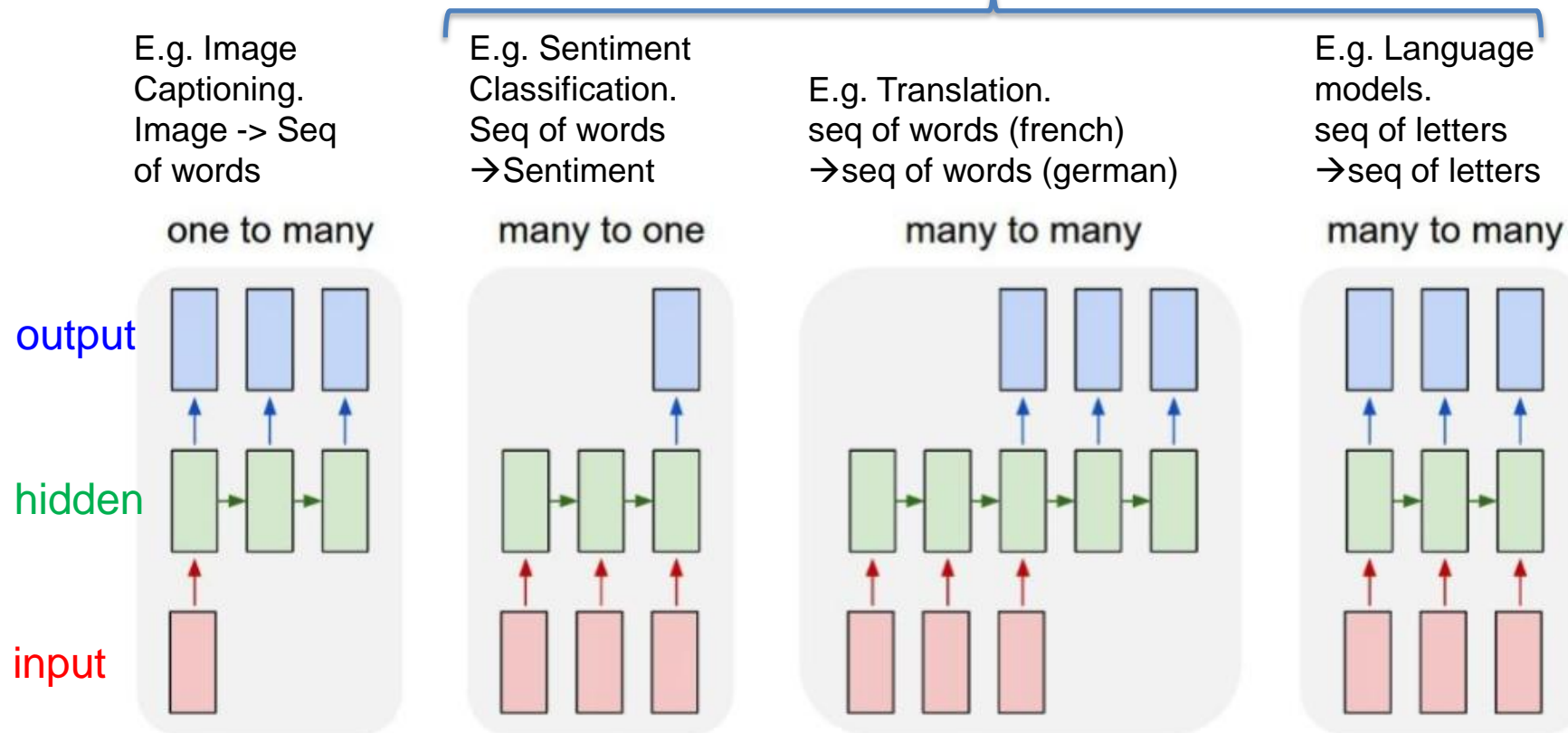
To apply a trained CNN to new images we would resize the images to the expected input size of the CNN

Motivation for recurrent neural networks

Recurrent neural networks (RNN) are used to model sequences.

RNN can handle inputs and outputs with flexible length.

To learn with sequence data we need a **memory** about the seen former parts.



Can DL identify a spam e-mail?

Important Notice

Please note that starting from October 30, 2015 we will be introducing new online banking authentication procedures in order to protect the information of our online banking users.

You are required to confirm your personal details with us as you will not be able access our online service until this has been done. As you're already registered for online banking all you need to do is to confirm your online banking details.

Get Started

Once you've completed this process you'll be able to have full access to our online banking service.

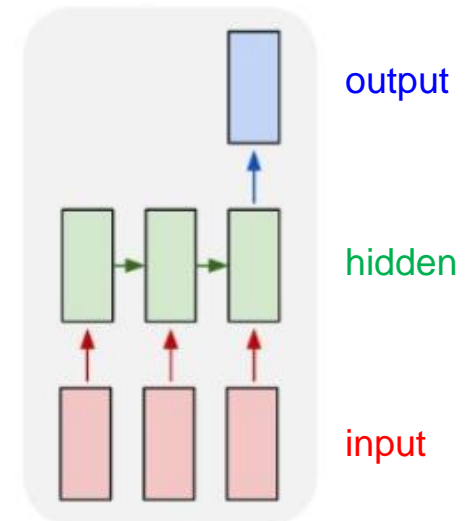
Best regards,
Natwest Online Banking Team

Guten Tag, Sick Beate (sick)

siehe Anhang
gescanntes Dokument.:
[http://dildosatisfaction.com/Rechnungs-Details-98773504333/Sick Beate \(sick\)](http://dildosatisfaction.com/Rechnungs-Details-98773504333/Sick Beate (sick))

Mit freundlichen GrÃ¼Ãen

many to one



Challenges:

- 1) We need to find a numeric representation of sentences (embedding)
- 2) We need to be able to handle inputs of different length

Can DL model score-development during reading an essay?

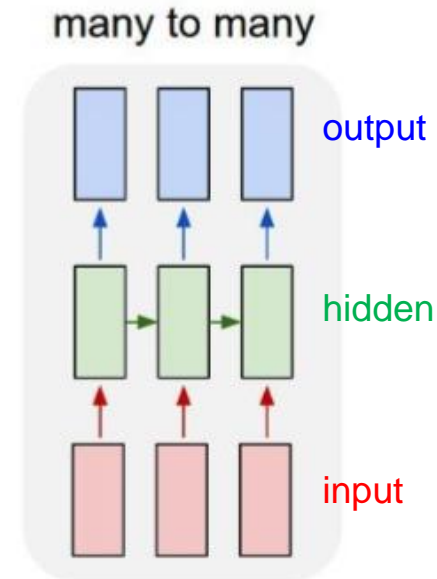
Imagine you are a lecturer on “literature”.

Each year you have to score essays on Schiller’s Bell.

Can DL help you doing the job?

Example essay:

Schiller’s best-known poetic work, “The Song of the Bell” is in the form of a ballad. The poem’s genesis occurred when Schiller visited a bell foundry at Rudolstadt. The poem has a narrator, the master bell maker, who shows his apprentices how casting a bell is similar to living the various phases of life. Schiller’s bell stands for harmony, peace, and the possibility of creating a better society....



Challenges:

- 1) We need to find a numeric representation of sentences (embedding)
- 2) We need to be able to handle inputs of different length

Can a DL model be used to write an essay?

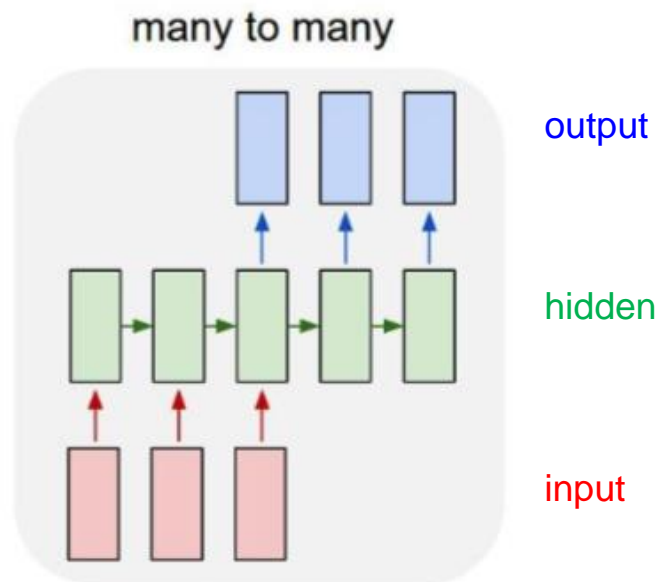
Imagine you are a lazy student

You have to write the 10⁶th essay on a topic.

What to do without copying - can DL help?

Idea use former essays to train a DL model predicting the words for the next sentence based on the words in the 4 last sentences.

Start with a couple of sentences (e.g. 10) and let the trained model do the rest 😊



Challenges:

- 1) We need to find a numeric representation of words (embedding)
- 2) We need to be able to handle inputs of different length

Challenge 1: How to represent text as numeric vector?

- Count vectors or “bag of words”
 - Determine vocabulary (or alphabet)
 - Represent each sentence (word) in a document as corresponding count vector

Example:

Sentence 1: “The cat sat on the hat”

Sentence 2: “The dog ate the cat and the hat”

Count vector matrix:

	the	cat	sat	on	hat	dog	ate	and
1	2	1	1	1	1	0	0	0
2	3	1	0	0	1	1	1	1

Now we can represent each sentence as a numeric vector of the same length!

Remark: You might also want to check out the more sophisticated “word2vec” methods -> ask google.

“word2vec encodes words to higher dimensional space that provides semantic relationships that we can manipulate as vectors”

Recurrent NN have memory

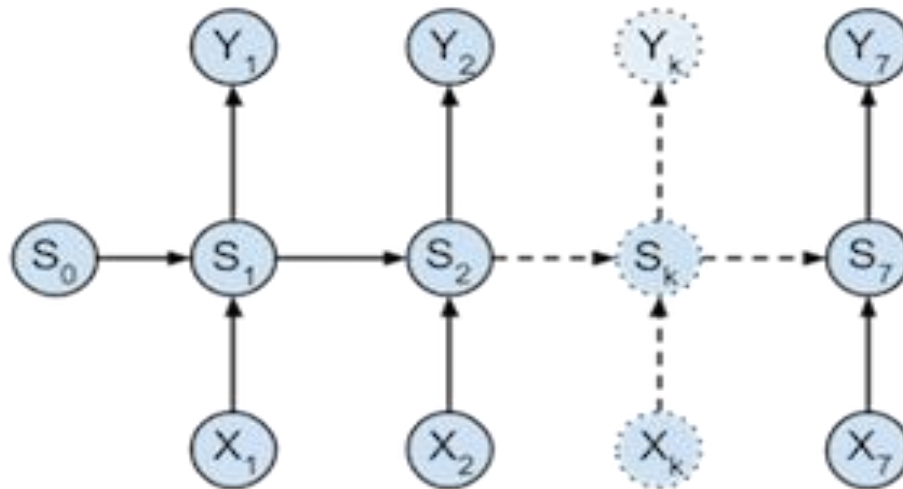
Challenge 2: How to handle input of different lengths?

Each text (e.g. e-mail) can have a different number of sentences!

By reading from sentence to sentence our believe in different categories (e.g. spam or not-spam) can change and we want to update our classification.

We need a model which can **memorize the information from former inputs**.

Output: $y_1 = \text{prob}_{\text{spam}}$, $y_2 = \text{prob}_{\text{spam}}$,



Hidden memory State:

S_1 =state after first sentence

S_2 =state after first sentence

...

Input: $x_1 = \text{vector}_{\text{sentence1}}$, $x_2 = \text{vector}_{\text{sentence2}}$,

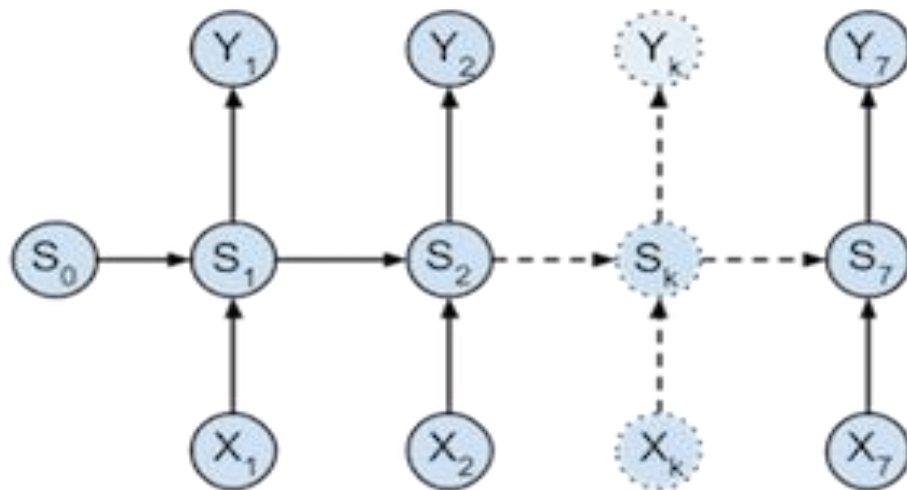
How to choose the dimensions of the hidden state?

The input \mathbf{x} is a vector of the size of our vocabulary.

The output \mathbf{y} is a vector of size 2 (spam/no-spam or passed/failed).

The hidden state \mathbf{h} (or \mathbf{S}) should have enough capacity to capture different concepts of spams (e.g. fraud, sex, conference) – we could choose a vector of length 3.

Output: $y_1=(p_1, p_2)_{t1}$ $y_2=(p_1, p_2)_{t2}$



Hidden memory State:
 S_1 =state after first sentence
 S_2 =state after first sentence
...

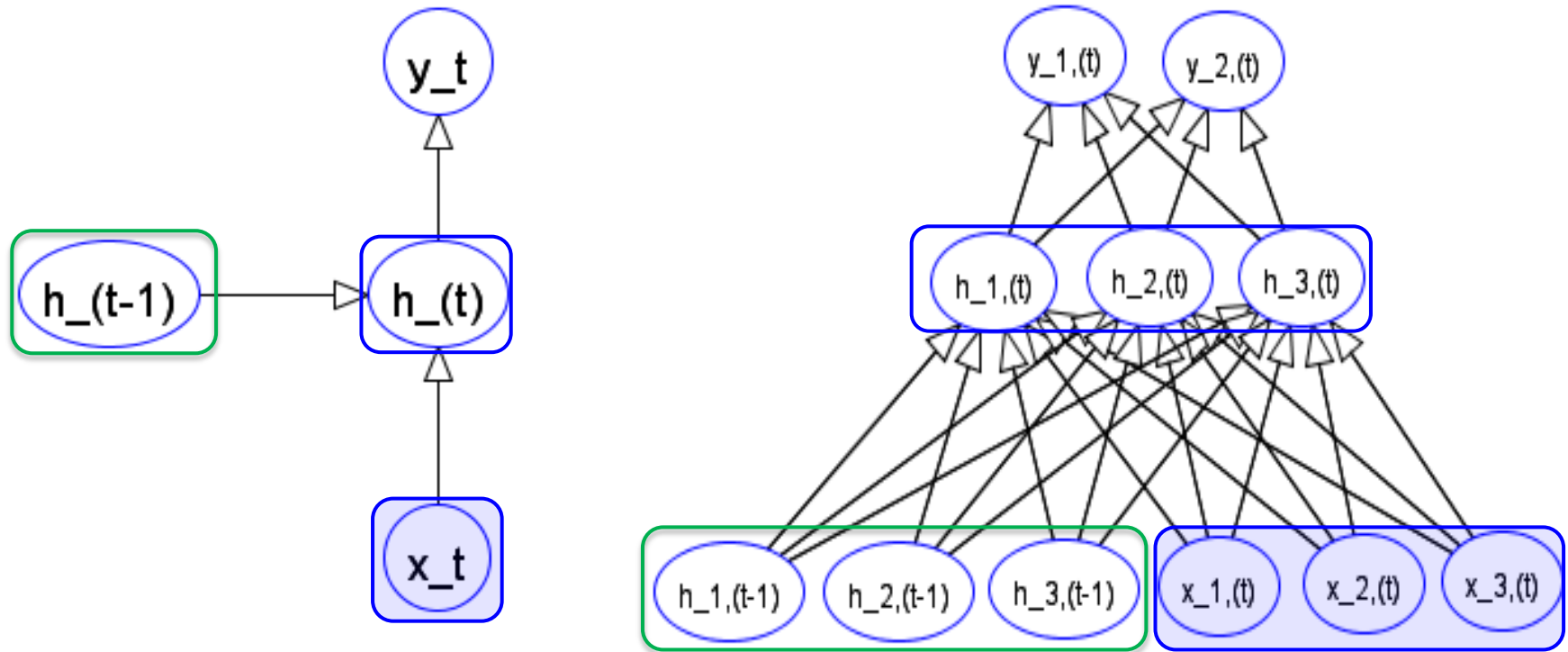
Input: x_1 =vector_{sentence1}, x_2 =vector_{sentence2},

Two representations of a RNN at time t

\mathbf{x} has 3 components – a very small vocabulary ;-)

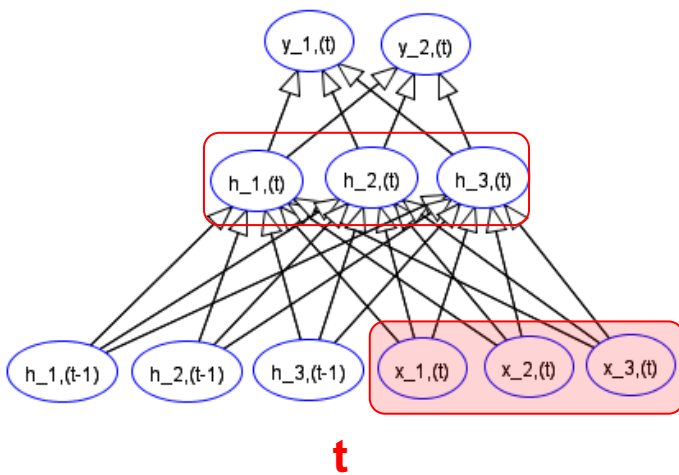
\mathbf{y} has 2 components (spam/no-spam for mails, passed/failed for scoring essays).

\mathbf{h} 3 components to capture abstract concepts (e.g. fraud/sex/conference for emails, or copied/boring/original for essays) and is initialized at $t=0$ with $(0,0,0)$.

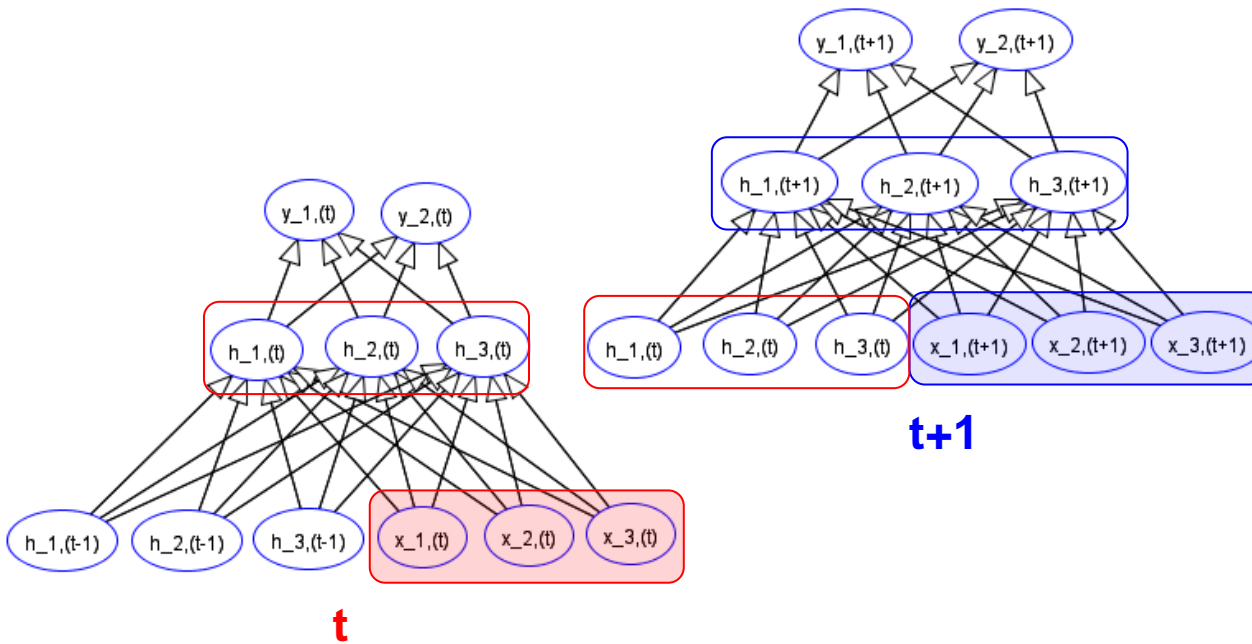


Stepping through an RNN

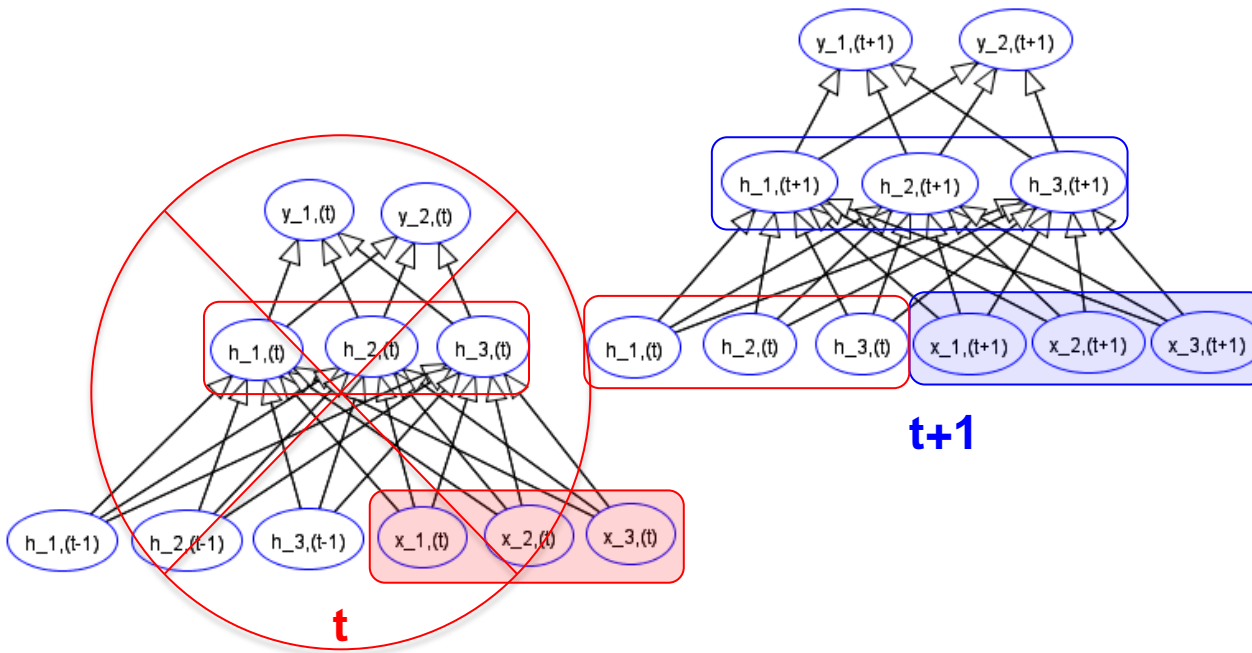
A simple RNN at 3 successive time steps



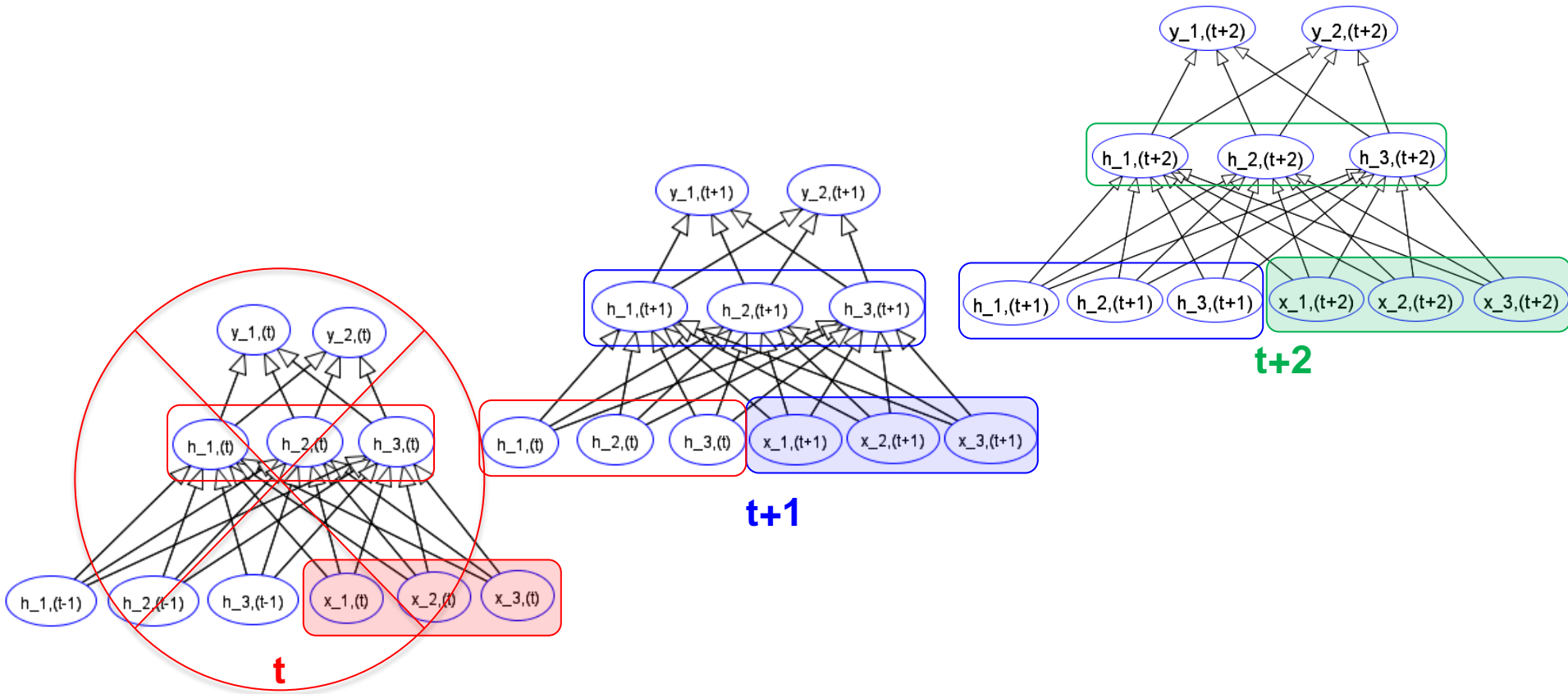
A simple RNN at 3 successive time steps



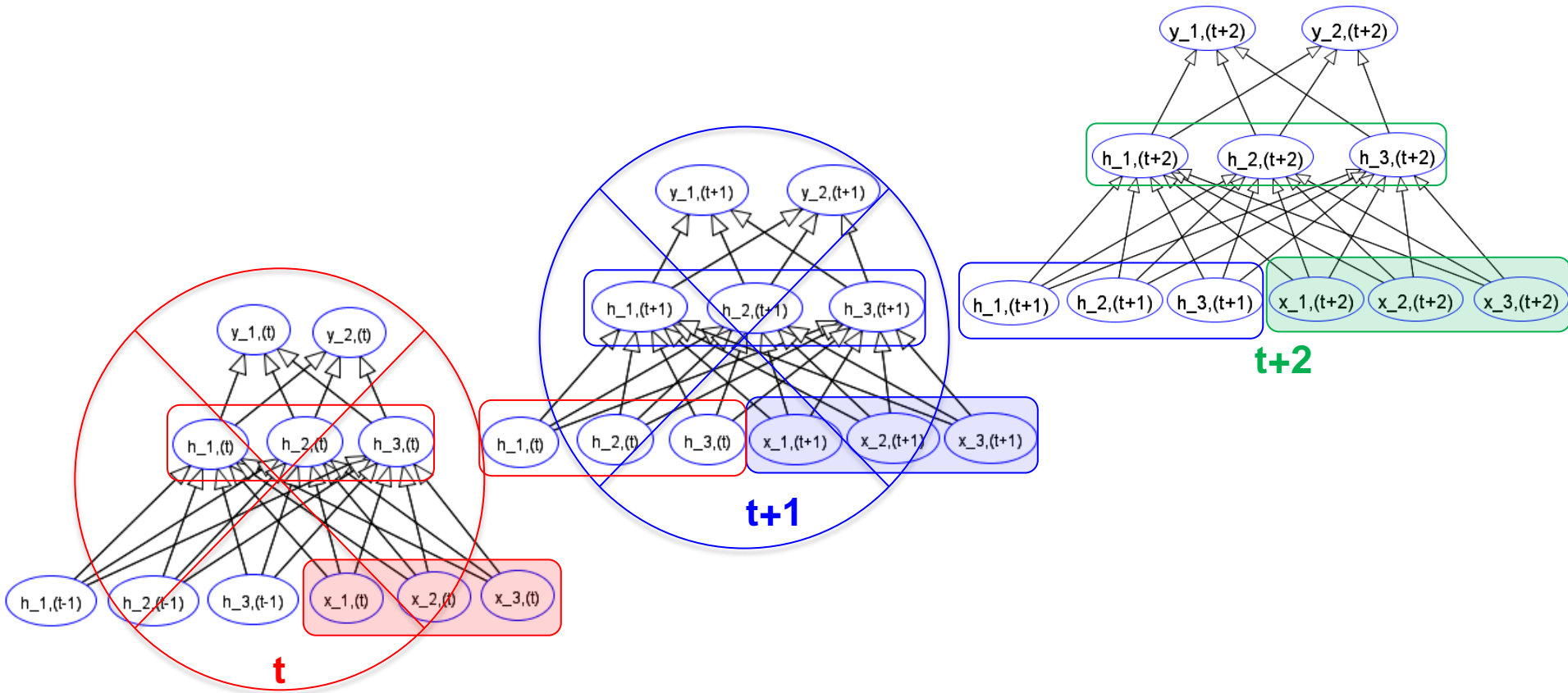
A simple RNN at 3 successive time steps



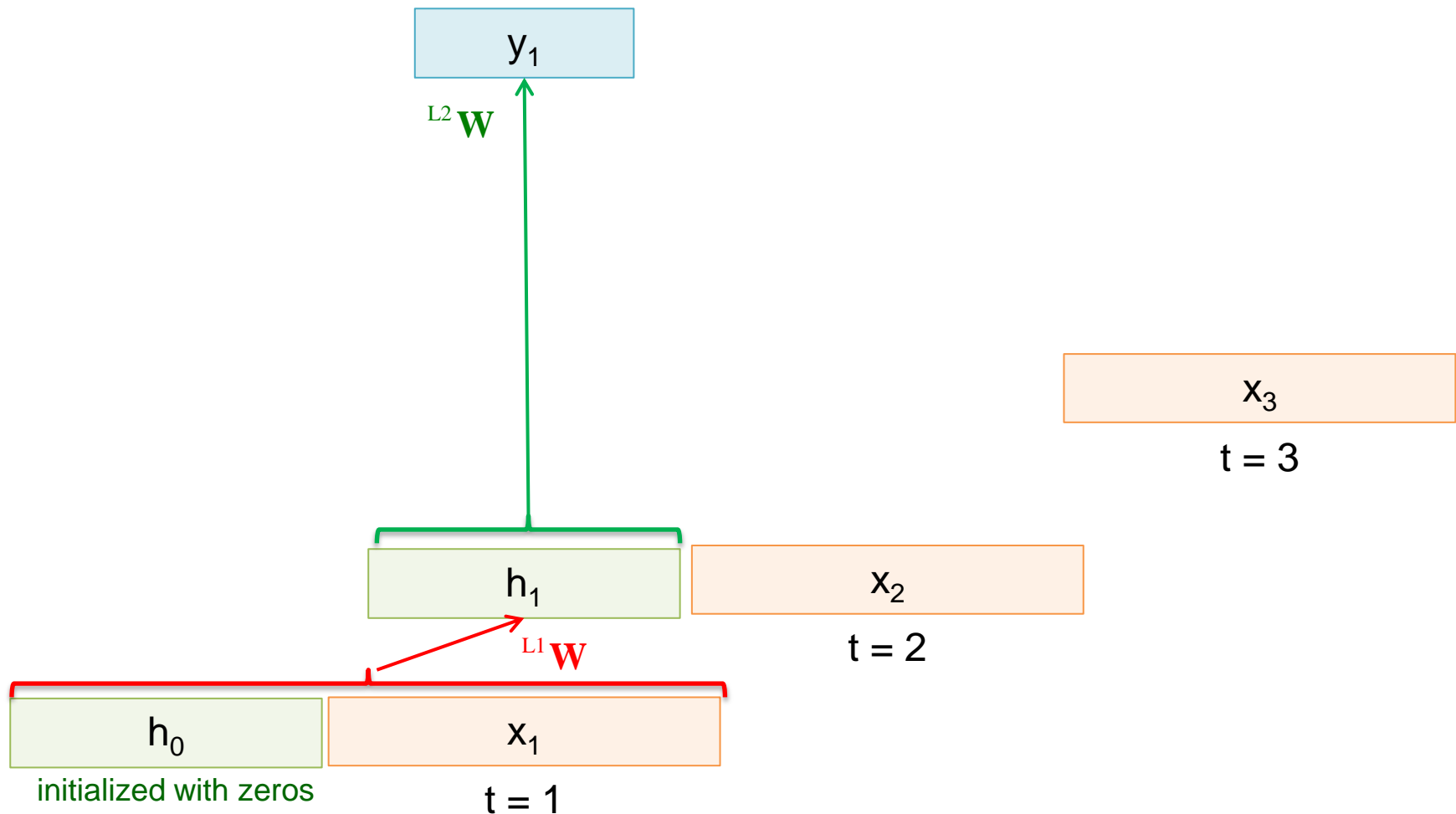
A simple RNN at 3 successive time steps



A simple RNN at 3 successive time steps

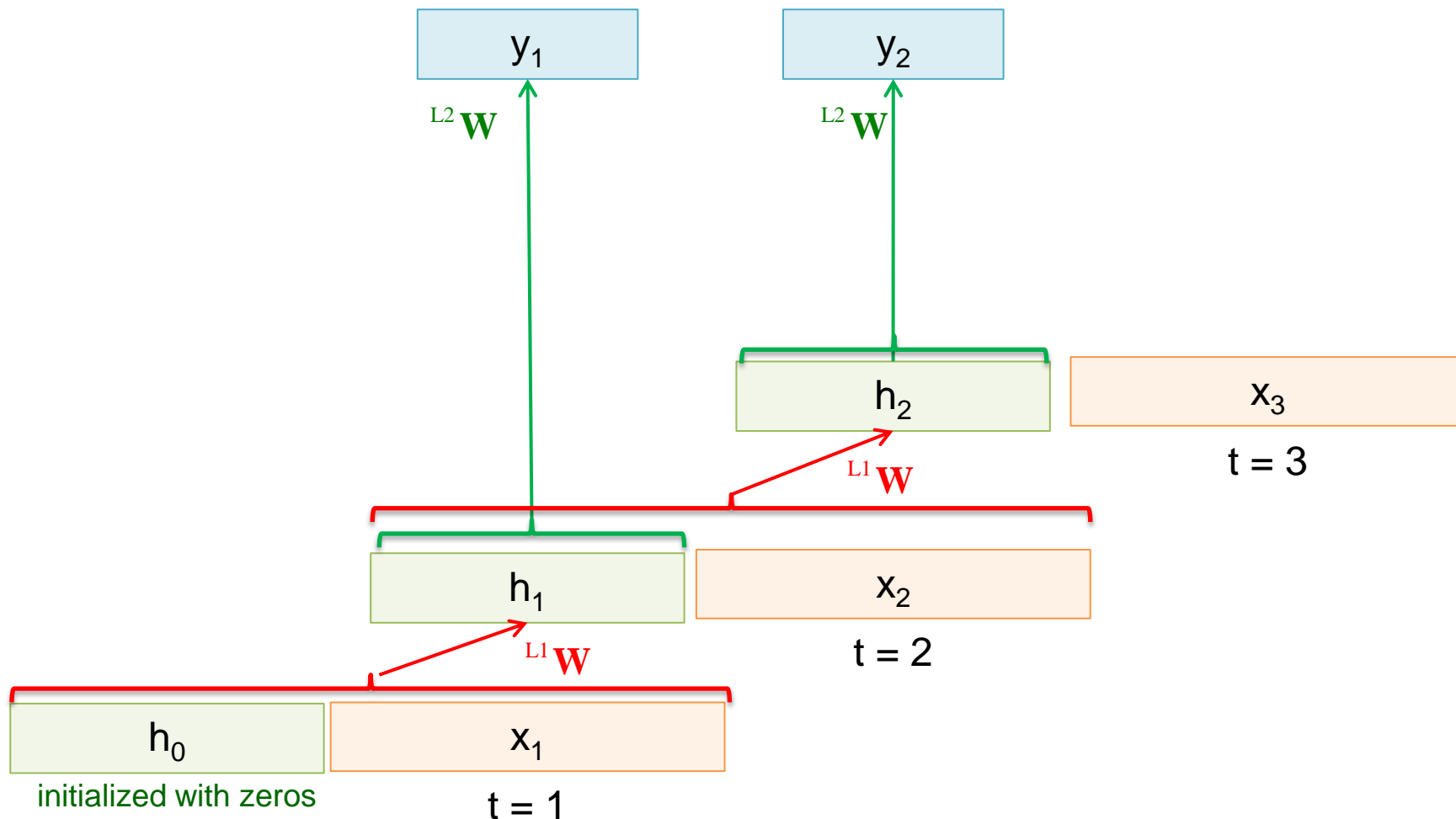


An RNN shares weights across all time steps



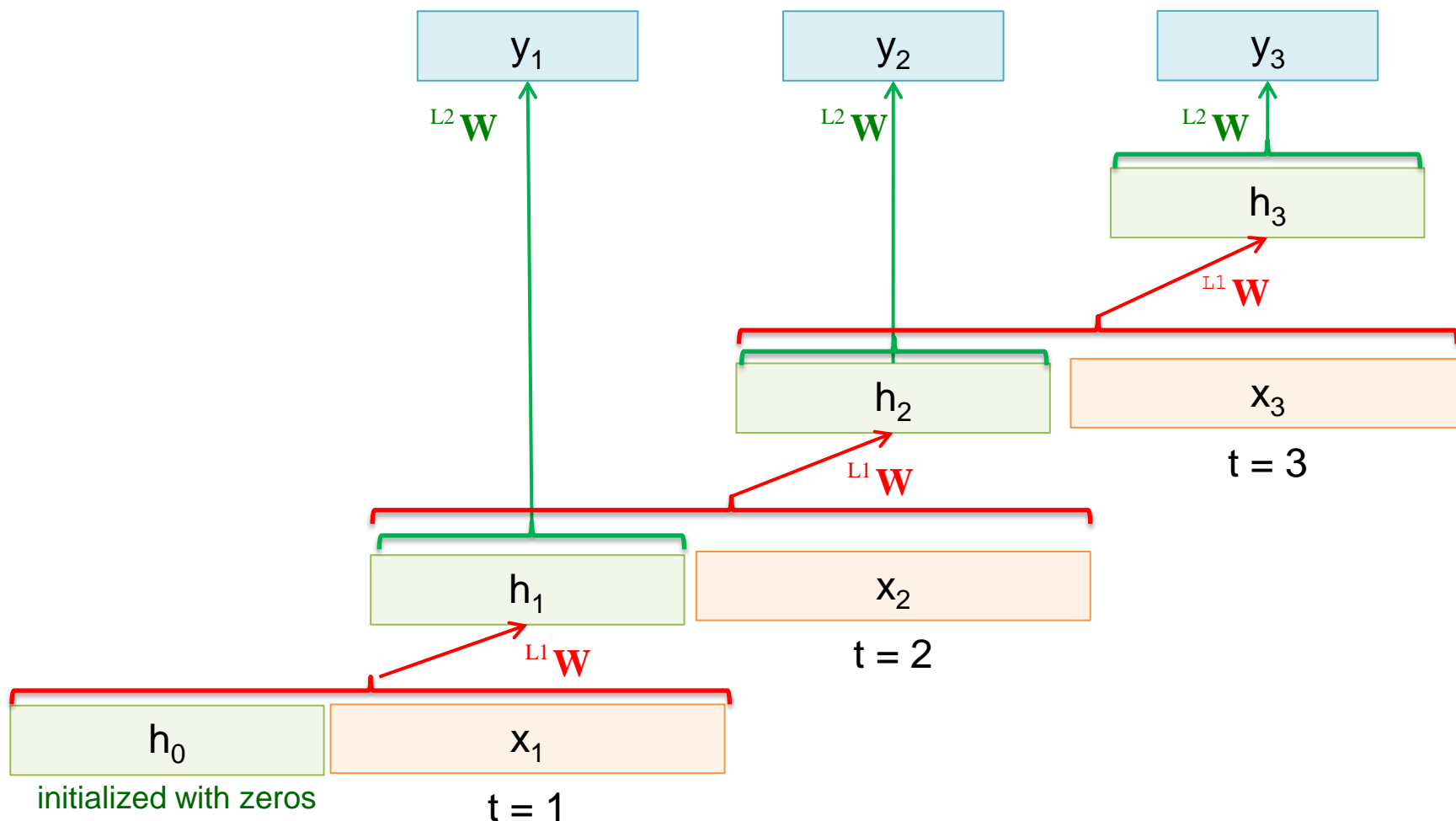
Imagine a trained RNN with fixed weight matrices in layer 1 and layer 2.

An RNN shares weights across all time steps



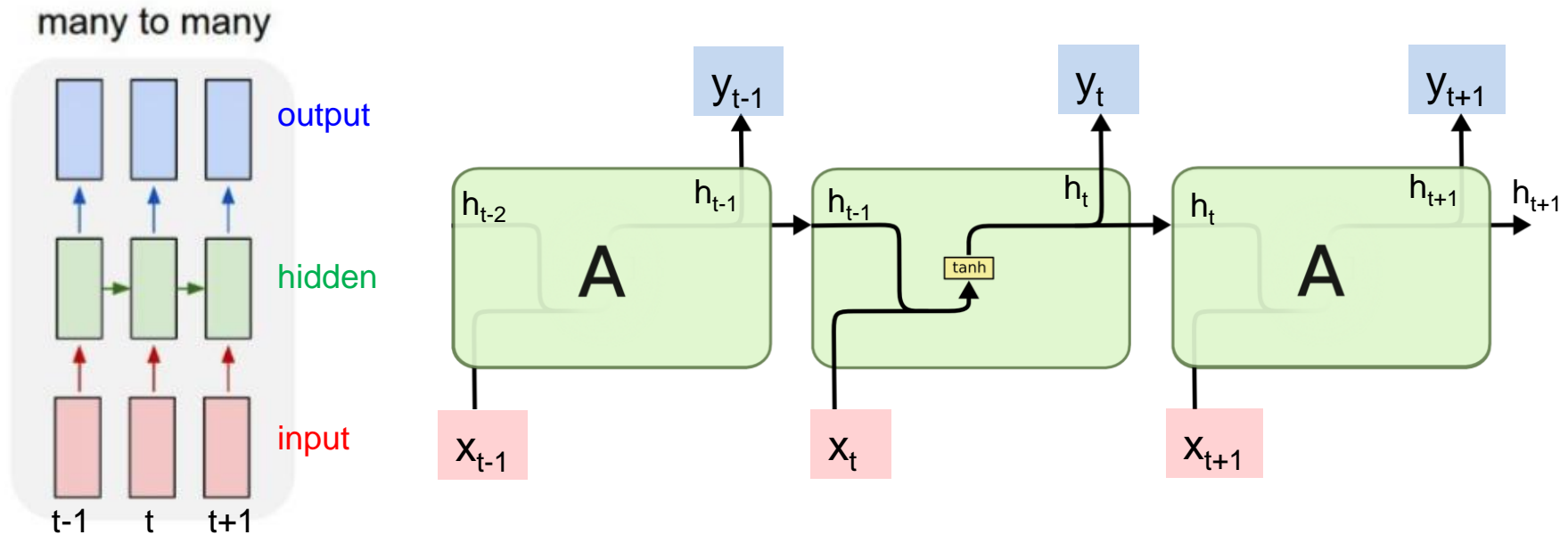
We use at each time step the same weight matrices between the layers!

An RNN shares weights across all time steps



The length of the input sequence can have arbitrary length.
We just reuse (keras: distribute) the same NN for each instance in the sequence!
Therefore it is called recurrent network!!

Using diagrams to represent an RNN

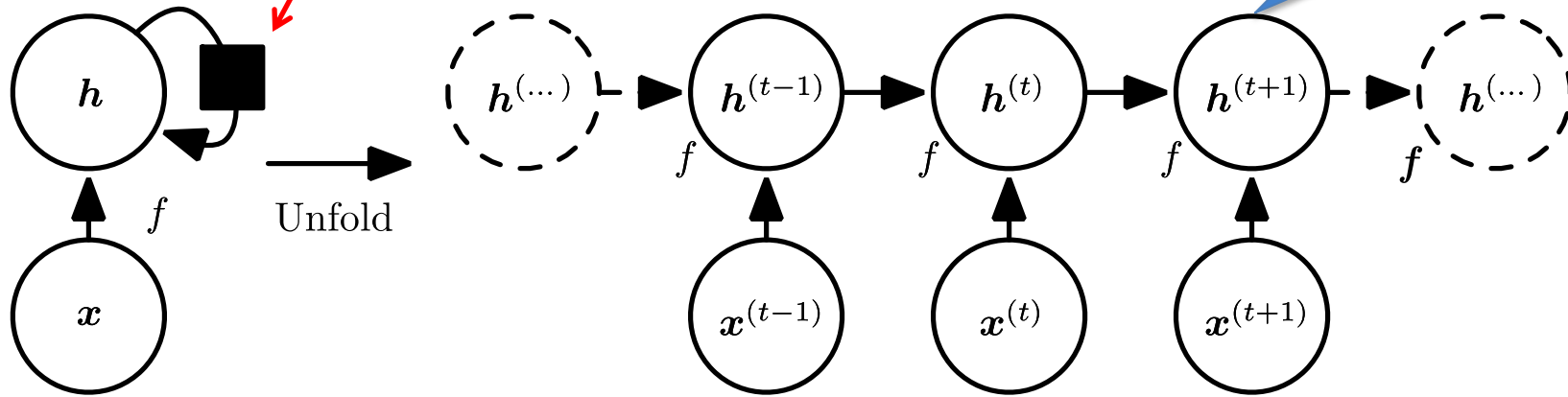


$$\mathbf{A} = f_{\mathbf{W}}(\mathbf{h}_{t-1}, \mathbf{x}_t) = \tanh([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W} + \mathbf{b})$$

Using a circuit diagram to represent an RNN

Use same weights
in each time step

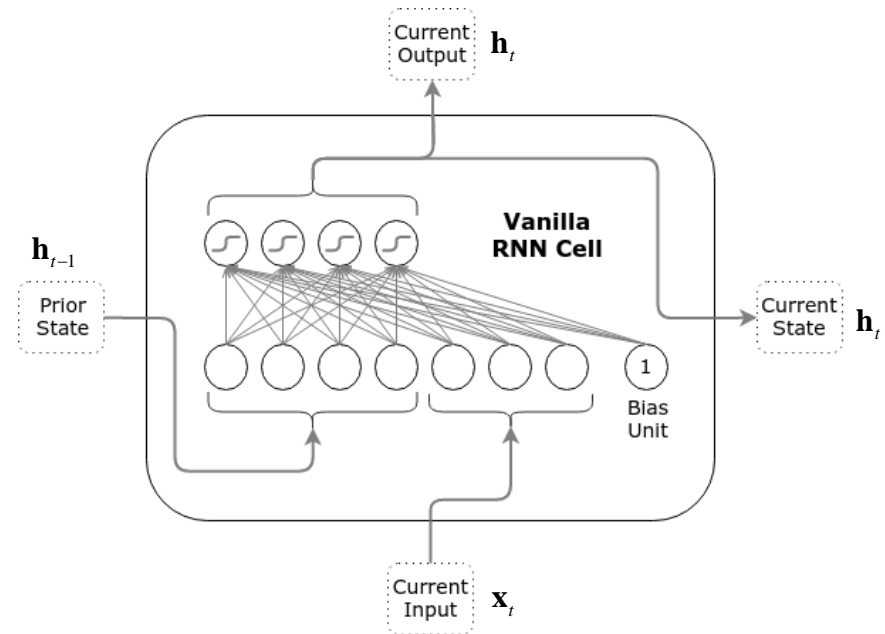
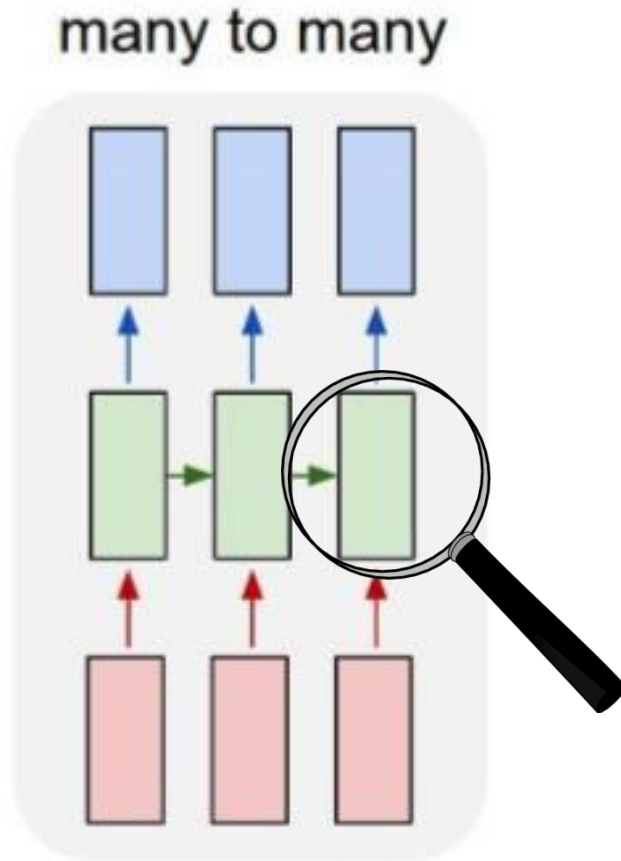
$h^{(t)}$ summarizes /
abstraction



Left: Circuit Diagram (black square delay of one time step)

Right: Unrolled / unfolded

Looking into a RNN “cell”



$$\text{output} = \mathbf{h}_t = \tanh([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W} + \mathbf{b})$$

Loss construction in an RNN

Determine the loss contribution of instance 1

mini-batch of size M=8

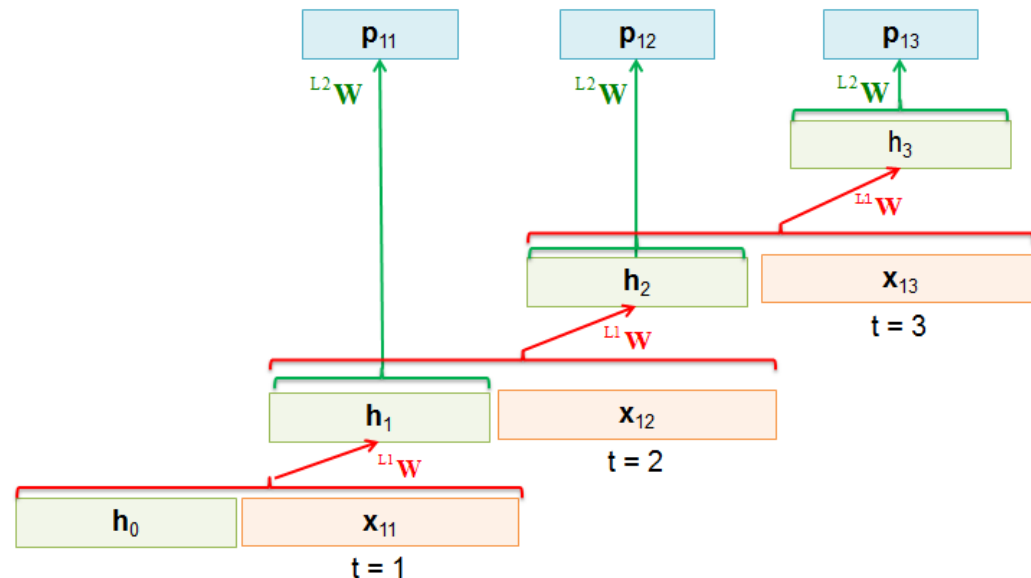
train data input (S=len(seq)=3):

instance_id	seq_t1	seq_t2	seq_t3
1	\mathbf{x}_{11}	\mathbf{x}_{12}	\mathbf{x}_{13}
2	\mathbf{x}_{21}	\mathbf{x}_{22}	\mathbf{x}_{23}
3	\mathbf{x}_{31}	\mathbf{x}_{32}	\mathbf{x}_{33}
⋮	⋮	⋮	⋮
8	\mathbf{x}_{81}	\mathbf{x}_{82}	\mathbf{x}_{83}

train data target (2 classes, K=2):

instance id	y t1	y t2	y t3
1	(1,0)	(1,0)	(0,1)
2	(0,1)	(1,0)	(0,1)
3	(0,1)	(0,1)	-1
⋮	⋮	⋮	⋮
8	(1,0)	(1,0)	(1,0)

instance 1:



x-entropy is used to determine distance between 2-dim p-vector and 2-dim y-vector at each of the 3 positions in the sequence:

$$\text{Loss_1} = \sum_{s=1}^3 \left(- \sum_{k=1}^2 y_{1sk} \cdot \log(p_{1sk}) \right)$$

Determine the loss contribution of instance 2

mini-batch of size M=8

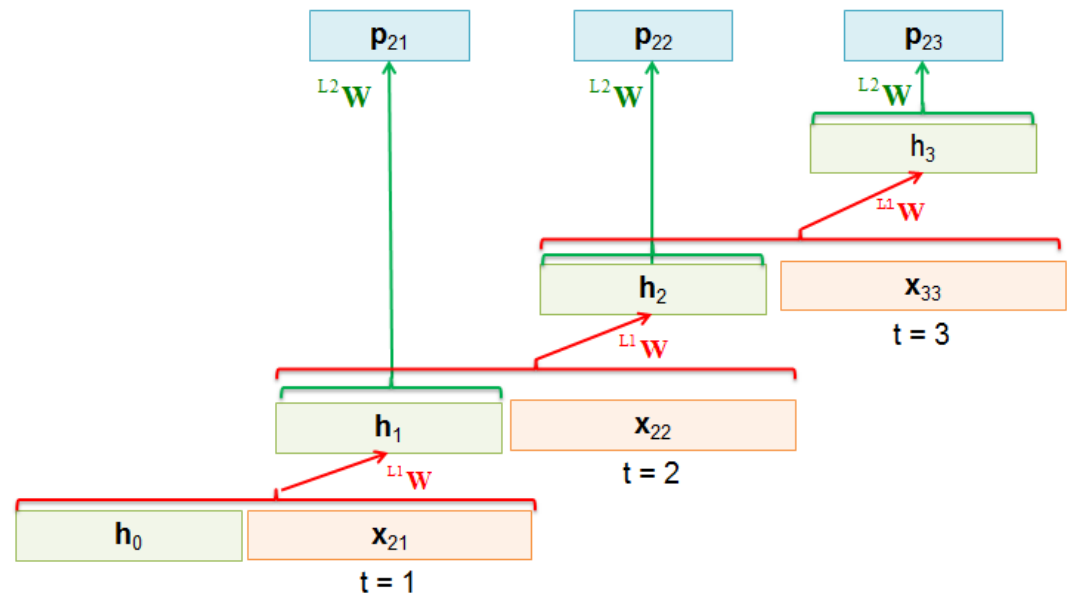
train data input (S=len(seq)=3):

instance_id	seq_t1	seq_t2	seq_t3
1	\mathbf{x}_{11}	\mathbf{x}_{12}	\mathbf{x}_{13}
2	\mathbf{x}_{21}	\mathbf{x}_{22}	\mathbf{x}_{23}
3	\mathbf{x}_{31}	\mathbf{x}_{32}	\mathbf{x}_{33}
⋮	⋮	⋮	⋮
8	\mathbf{x}_{81}	\mathbf{x}_{82}	\mathbf{x}_{83}

train data target (2 classes, K=2):

instance_id	y_t1	y_t2	y_t3
1	(1,0)	(1,0)	(0,1)
2	(0,1)	(1,0)	(0,1)
3	(0,1)	(0,1)	-1
⋮	⋮	⋮	⋮
8	(1,0)	(1,0)	(1,0)

instance 2:



x-entropy is used to determine distance between 2-dim p-vector and 2-dim y-vector at each of the 3 positions in the sequence:

$$\text{Loss_2} = \sum_{s=1}^3 \left(- \sum_{k=1}^2 y_{2sk} \cdot \log(p_{2sk}) \right)$$

Determine the loss of the whole mini-batch

mini-batch of size M=8

train data input (S=len(seq)=3):

instance_id	seq_t1	seq_t2	seq_t3
1	\mathbf{x}_{11}	\mathbf{x}_{12}	\mathbf{x}_{13}
2	\mathbf{x}_{21}	\mathbf{x}_{22}	\mathbf{x}_{23}
3	\mathbf{x}_{31}	\mathbf{x}_{32}	\mathbf{x}_{33}
⋮	⋮	⋮	⋮
8	\mathbf{x}_{81}	\mathbf{x}_{82}	\mathbf{x}_{83}

train data target (2 classes, K=2):

instance_id	y_t1	y_t2	y_t3
1	(1,0)	(1,0)	(0,1)
2	(0,1)	(1,0)	(0,1)
3	(0,1)	(0,1)	-1
⋮	⋮	⋮	⋮
8	(1,0)	(1,0)	(1,0)

Cost C or Loss is given by the cross-entropy averaged over all instances in mini-batch:

$$\text{Loss} = \frac{1}{8} \sum_{m=1}^8 \text{Loss}_m$$

$$\text{Loss} = \frac{1}{8} \sum_{m=1}^8 \left[\sum_{s=1}^3 \left(- \sum_{k=1}^2 y_{\text{msk}} \cdot \log(p_{\text{msk}}) \right) \right]$$

Based on the mini-batch loss the weights in the two weight matrices of layer 1 and layer 2 are updated.

A simple forward pass



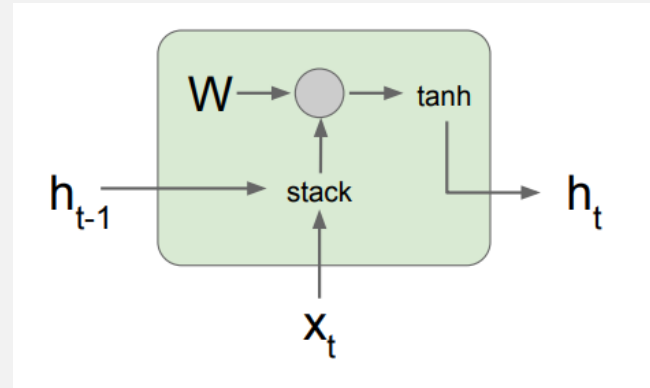
- Given the hidden state at t-1, the input x at t and the weight matrix as:

$$\mathbf{h}_{t-1} = \begin{pmatrix} 0 & 1 \end{pmatrix}$$

$$\mathbf{x}_t = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

$$\mathbf{W} = \begin{pmatrix} 1.5 & -2 \\ 0 & 1 \\ -1 & 0.5 \\ 2 & 0 \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} 0 & 0 \end{pmatrix}$$



$$\mathbf{A} = f_{\mathbf{W}}(\mathbf{h}_{t-1}, \mathbf{x}_t) = \tanh([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W} + \mathbf{b}) \dots$$

- Calculate the activation A of the hidden state \mathbf{h}_t at time t.

Solution

$$h_{t-1} = (0, 1)$$

$$x_t = (1, 0)$$

$$W = \begin{pmatrix} 1.5 & -2 \\ 0 & 1 \\ -1 & 1/2 \\ 2 & 0 \end{pmatrix}$$

$$(0, 1, 1, 0) \begin{pmatrix} 1.5 & -2 \\ 0 & 1 \\ -1 & 1/2 \\ 2 & 0 \end{pmatrix} = (-1, 1.8)$$

$$\Rightarrow h_t = \text{tanh}((-1, 1.8)) \approx (-0.76, 0.91)$$

RNN in Keras

```
from keras.layers import Activation, Dense, SimpleRNN, TimeDistributed
```

```
model = keras.models.Sequential()
```

```
model.add(SimpleRNN(6, batch_input_shape=(None, 50, 3), return_sequences=True))
```

```
model.add(TimeDistributed(Dense(2)))
```

```
model.add(Activation('softmax'))
```

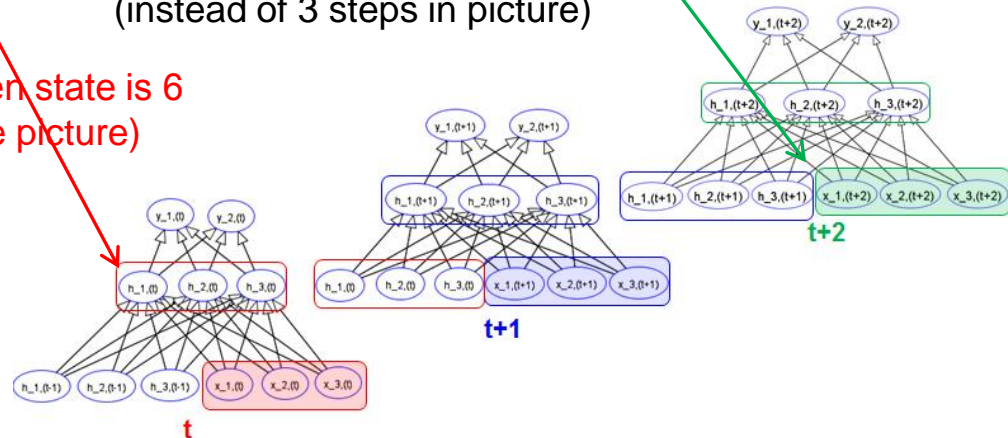
```
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
```

at each step we use hidden state as input to a fcNN with 2 output nodes

length of input vector at each time step is here 3

each training sequence consists out of 50 elements (instead of 3 steps in picture)






“capacity” of hidden state is 6 (instead of 3 in the picture)



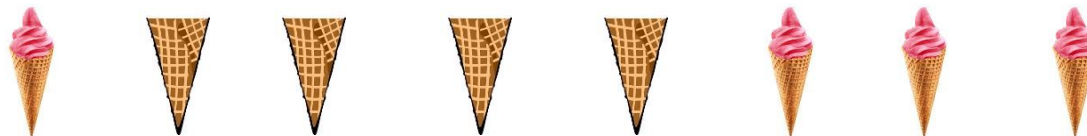


Example: The ice cream store

Toy Example: Ice Cream Store

y store can sell icecream		(1,0)	ice cream available			
		(0,1)	store has run out of ice cream			
x weather		(1,0,0)		(0,1,0)		(0,0,1)

Sample



Complicated order policy we don't know

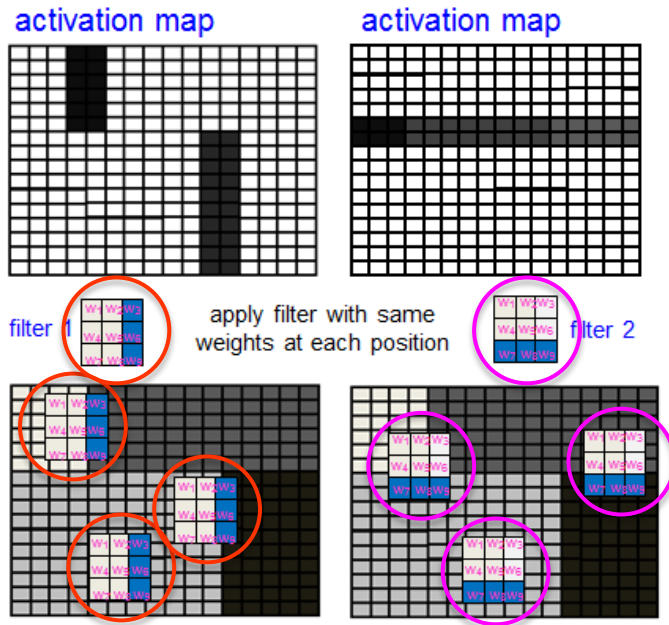


Days

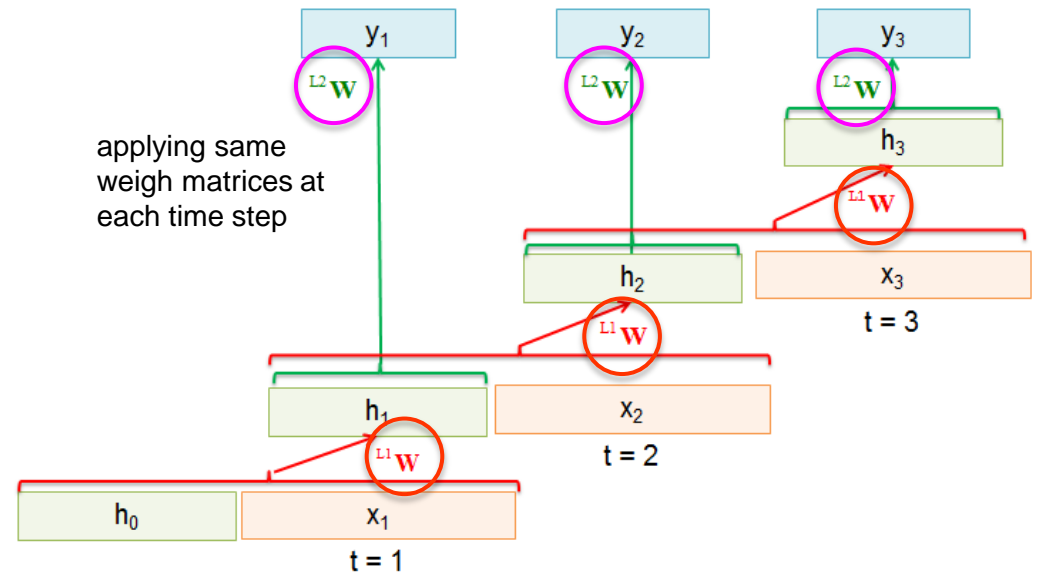
Train a RNN and predict if ice is available based on weather in last couple of days

Common tricks in RNN & CNN and some differences

CNN and Recurrent Network share weights



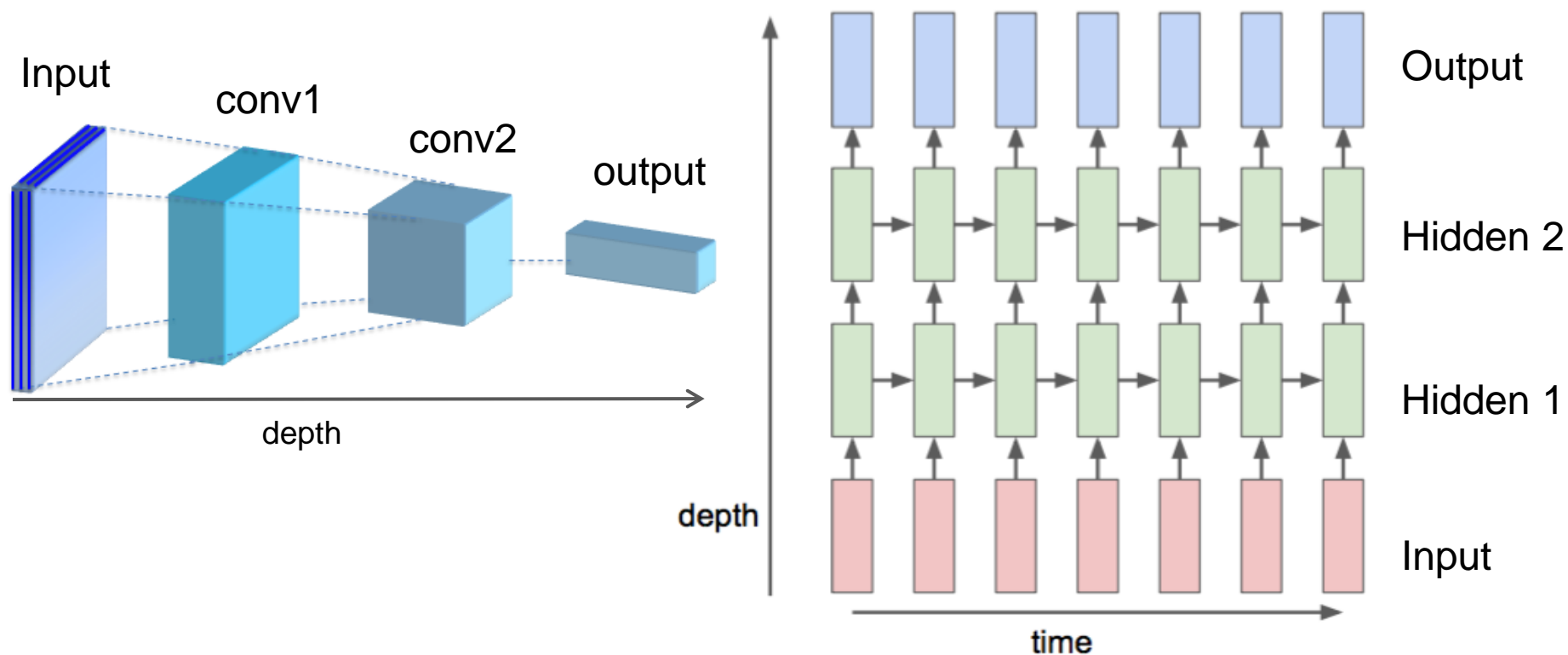
CNN share weights between different local regions of the image



RNN share weights between time steps

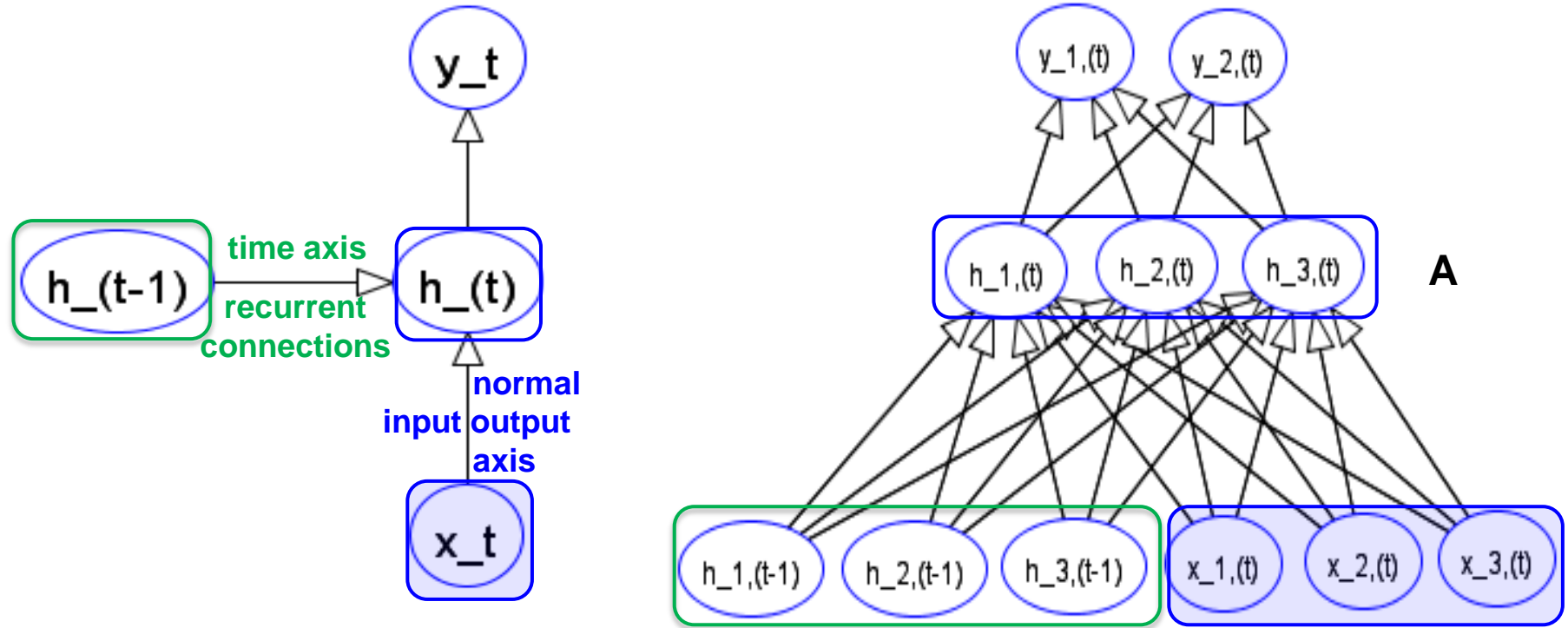
Remark: no weight sharing in fully connected NN

Also in RNN we can go deep for hierarchical features



Usually we see only 1-4 hidden layers in an RNN compared to usually 4-100 stacked hidden convolutional blocks in CNNs.

Dropout in recurrent architectures allow to choose different different dropout rates for recurrent and normal connections



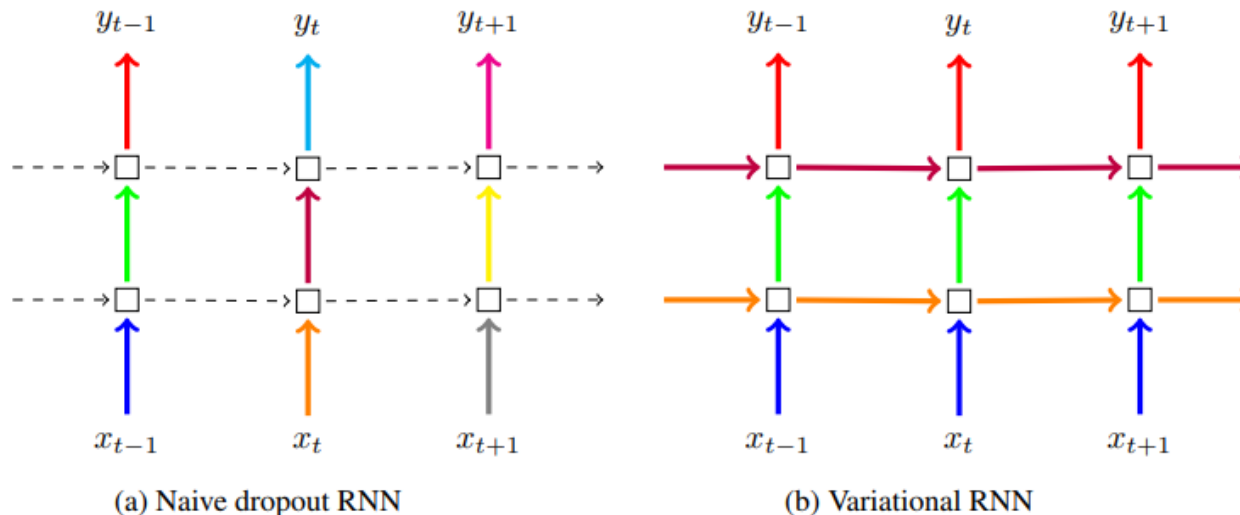
$$\mathbf{A} = f_{\mathbf{W}}(\mathbf{h}_{t-1}, \mathbf{x}_t) = \tanh([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W} + \mathbf{b}) = \tanh(\mathbf{h}_{t-1} \cdot \mathbf{W}_h + \mathbf{x}_t \cdot \mathbf{W}_x + \mathbf{b})$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_h \\ \mathbf{W}_x \end{pmatrix}$$

Dimensions in example: \mathbf{W} :6x3, \mathbf{W}_h :3x3, \mathbf{W}_x :3x3

Dropout in recurrent architectures

It is important to **use identical dropout masks** (marked by arrows with same color) **at different time steps** in recurrent architectures like GRU or LSTM.



same arrow
color indicates
identical dropout
mask

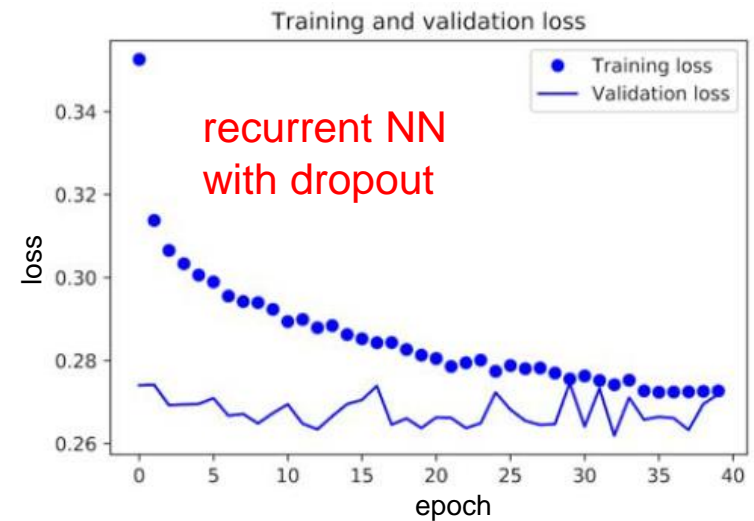
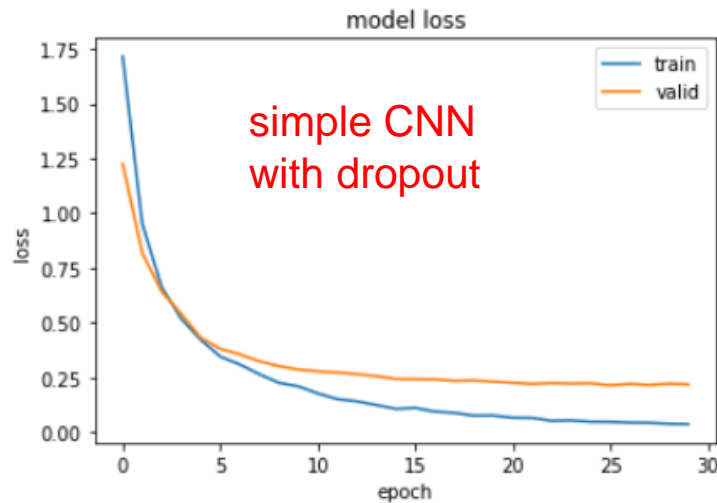
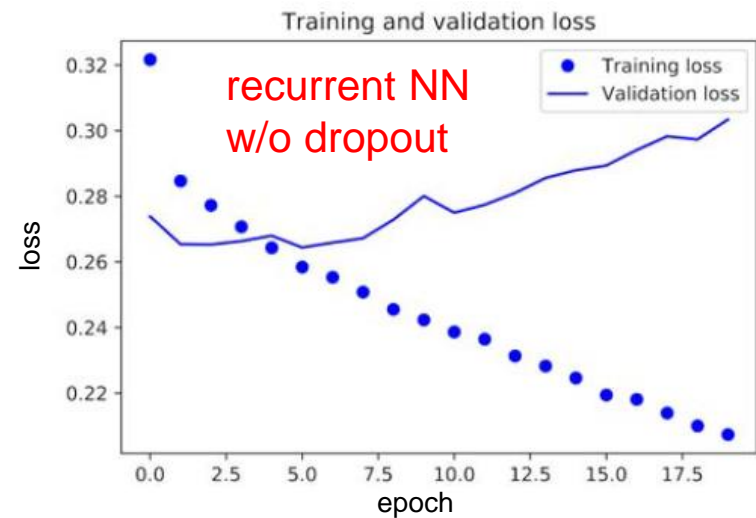
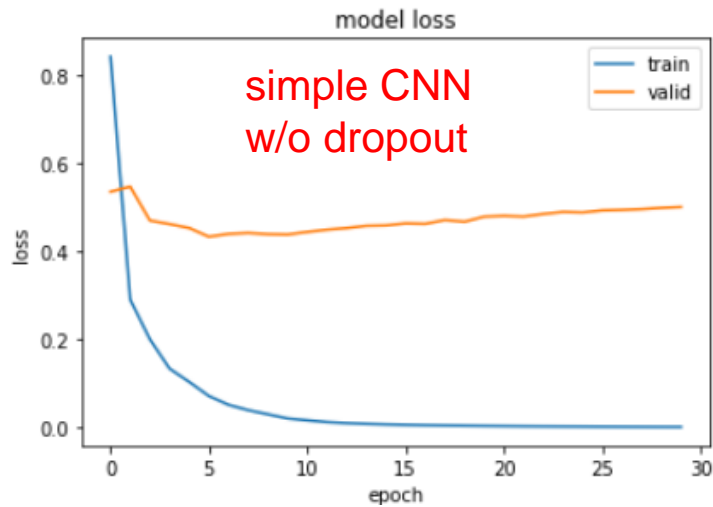
Figure 1: **Depiction of the dropout technique following our Bayesian interpretation (right) compared to the standard technique in the field (left).** Each square represents an RNN unit, with horizontal arrows representing time dependence (recurrent connections). Vertical arrows represent the input and output to each RNN unit. Coloured connections represent dropped-out inputs, with different colours corresponding to different dropout masks. Dashed lines correspond to standard connections with no dropout. Current techniques (naive dropout, left) use different masks at different time steps, with no dropout on the recurrent layers. The proposed technique (Variational RNN, right) uses the same dropout mask at each time step, including the recurrent layers.

[Gal2016](#)

In keras:

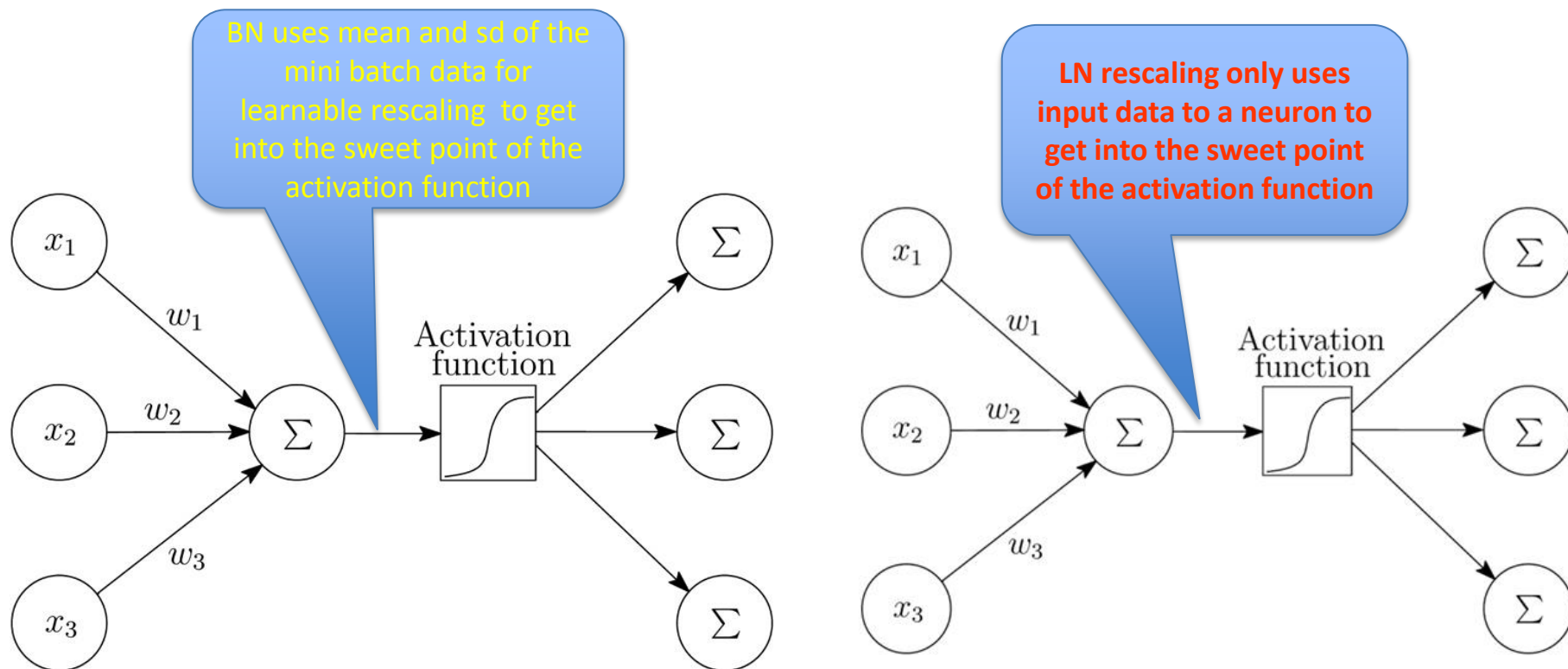
```
model.add(layers.GRU(32, dropout=0.2, recurrent_dropout=0.2, input_shape=(None, ...)))
```


Dropout can fight overfitting in CNN and recurrent NN



Batchnormalization is crucial to train deep CNNs

Layernormalization is beneficial in RNN: LN \neq BN



Applying BN to RNN would not take into account the recurrent architecture of the NN over which statistics of the input to a neuron might change considerable within the same mini batch. In LN the mean and variance from all of the summed inputs to the neurons in a layer on a single training case are used for normalization .