Zurich University of Applied Sciences

Modelling sequence data and using ideas from CNN & Recurrent Neural Networks

Beate Sick

sick@zhaw.ch

Remark: Much of the material has been developed together with Elvis Murina and Oliver Dürr

Topics

- **Famous tricks in challenge winning CNN architectures**
  - Inductive Bias, weight sharing, Inception moduls, gradient highway via skip-connections
- **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

Review of ImageNet winning CNN architectures

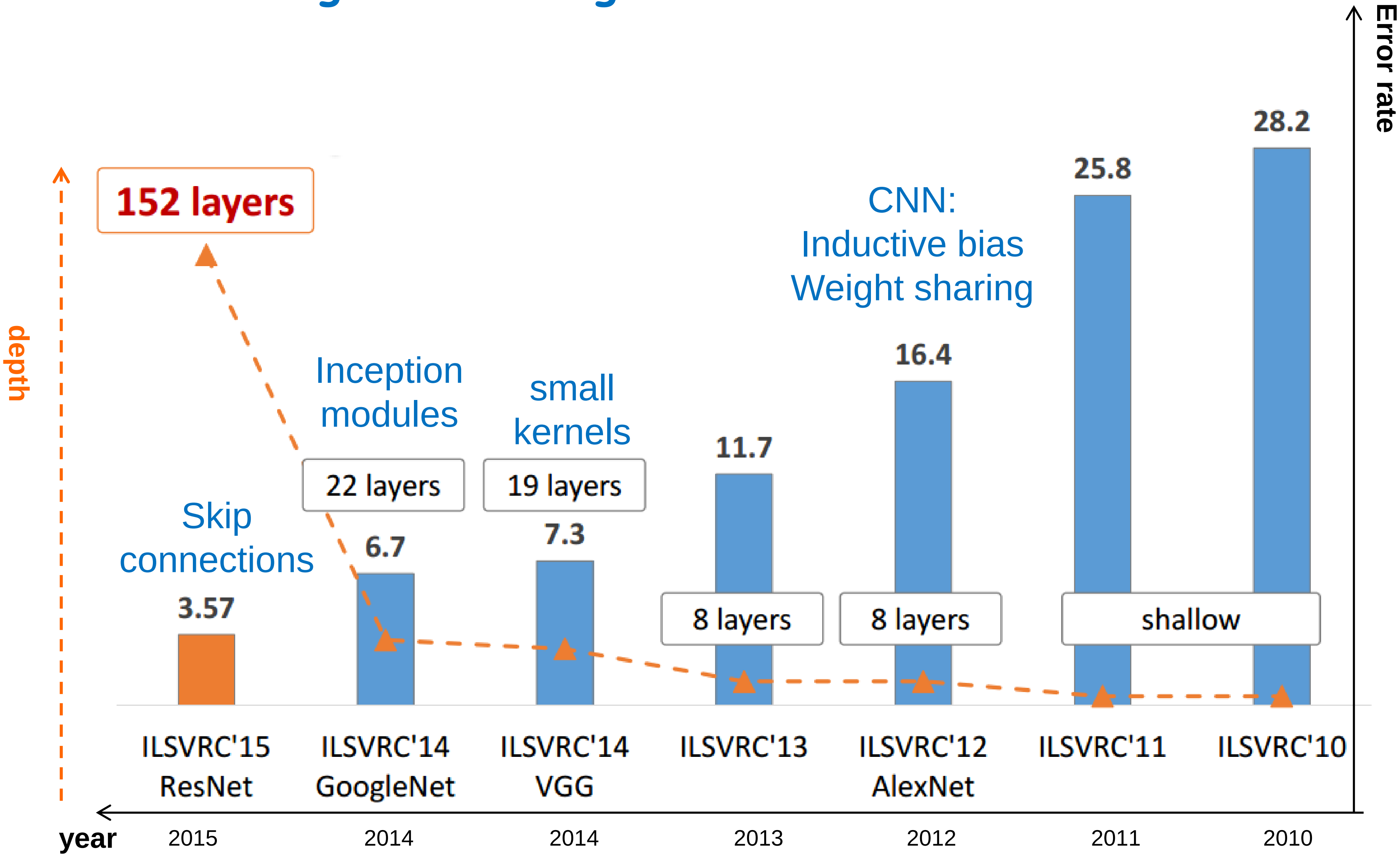


Image credits (modified): K.He, X.Zhang, S.Ren, J.Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015

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.

"Oxford Net" or "VGG Net" 2014 2nd place

- 2nd place in the imageNet challenge
- More traditional, easier to train
- Small pooling
- Stacked 3x3 convolutions before maxpooling

  -> large receptive field
- ReLU after conv. and FC (batchnorm was not used)
- More weights than GoogLeNet

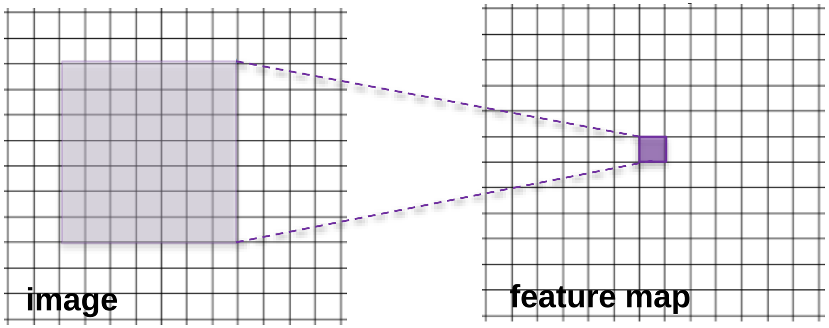
http://arxiv.org/abs/1409.1556

The trend in modern CNN architectures goes to small filters

How best to achieve a receptive field of 7x7 pixels?

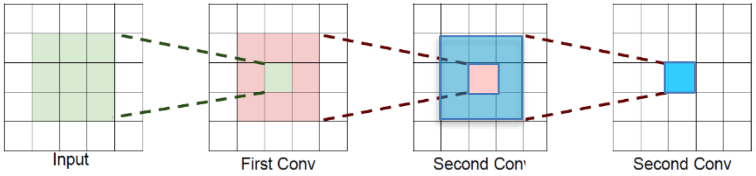
1) Working with **one** 7x7 conv layers (stride 1)

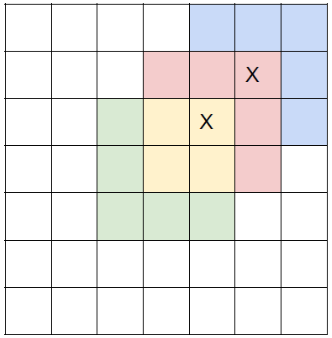
49 weights



2) Working with **three** 3x3 conv layers (stride 1)

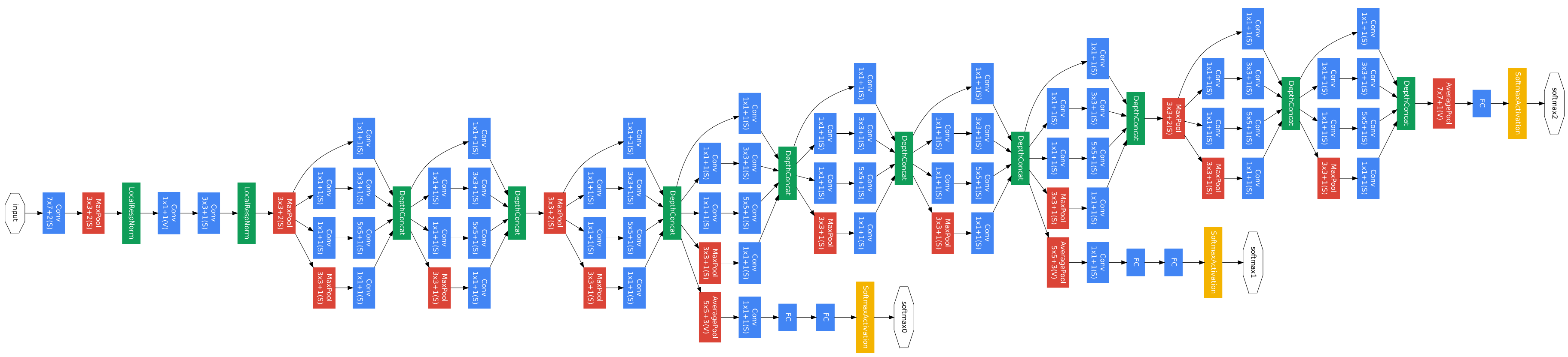
3*9=27 weights



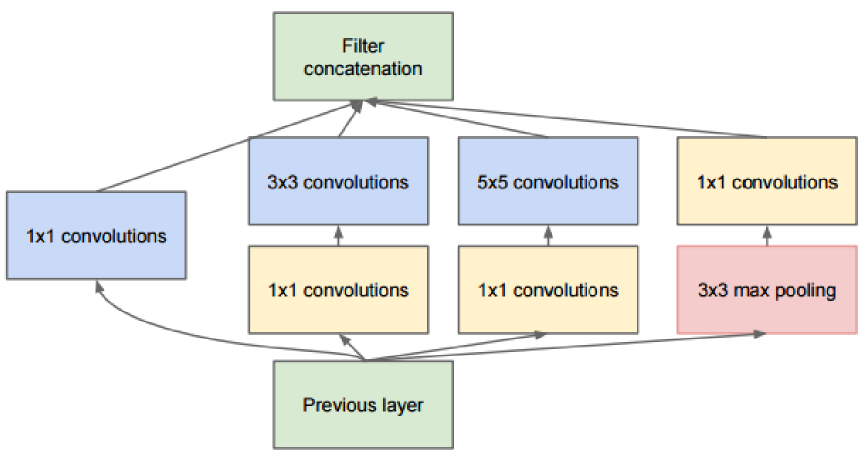


When working stacking conv-layers with small kernels we can achieve with less weights and more non-linear combinations the same receptive field

Winning architecture (GoogLeNet, 2014)



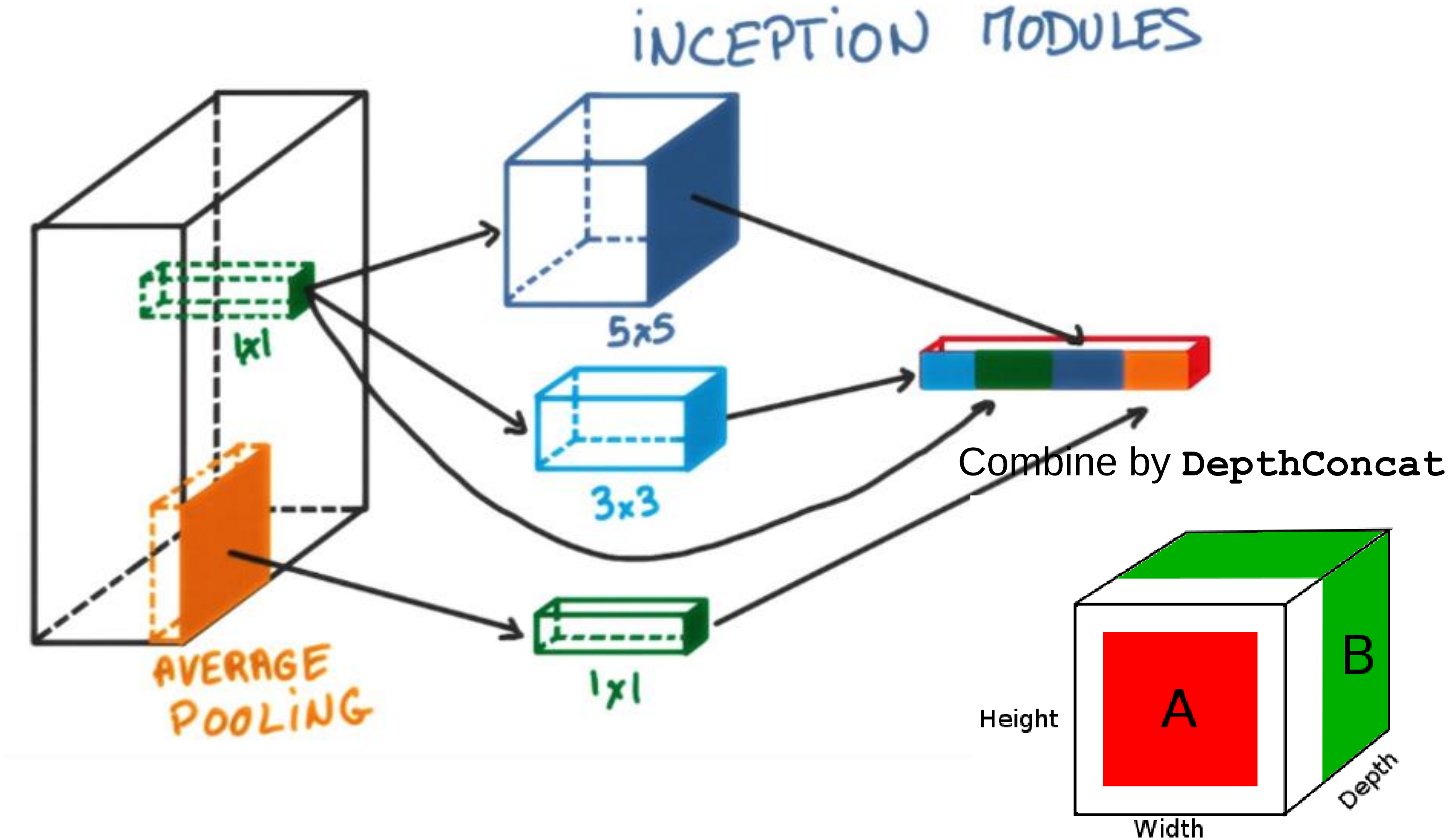
The inception module:
use in 1 layer in parallel different kernels and combine their results



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

[Going deeper with convolution http://arxiv.org/abs/1409.4842]

The idea of inception modules



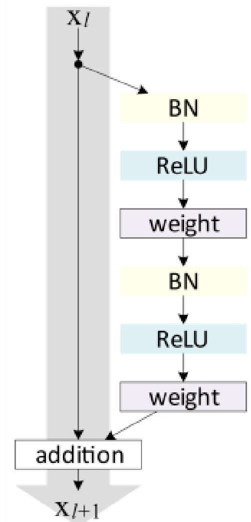
Slide (modified) credit: Google MOOC on tensorflow

“ResNet” from Microsoft 2015 winner of imageNet

152 layers

ResNet basic design (VGG-style)

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



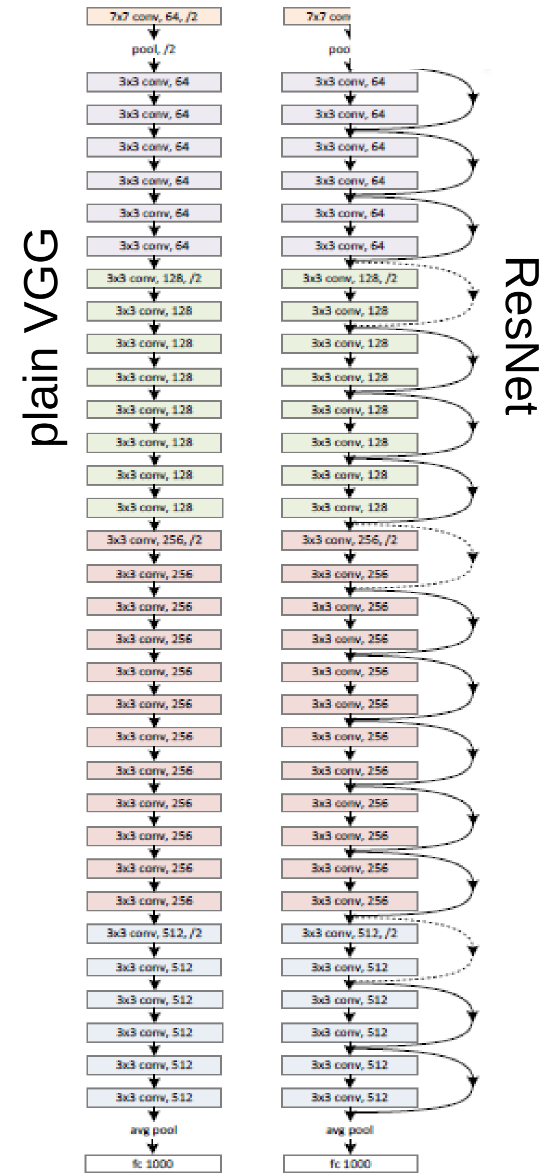
$F(x_l)$

$H(x_l)=x_{l+1} = x_l + F(x_l)$

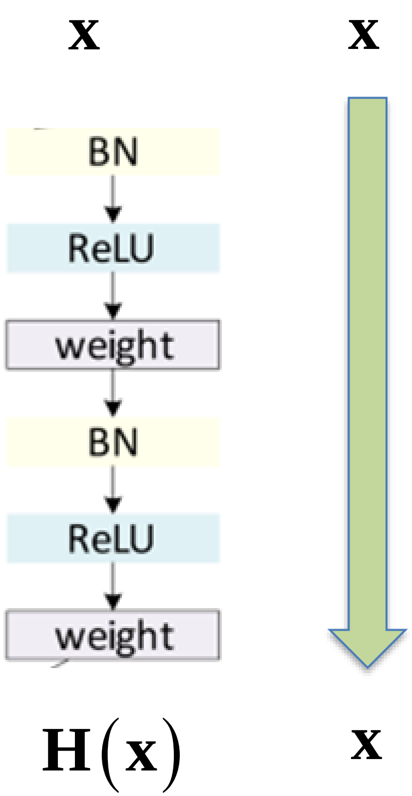
152 layers:
Why does this train at all?

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

F(x) is called “residual” since it only learns the “delta” which is needed to add to x to get H(x)



Highway Networks with skip connections: providing a highway for the gradient



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

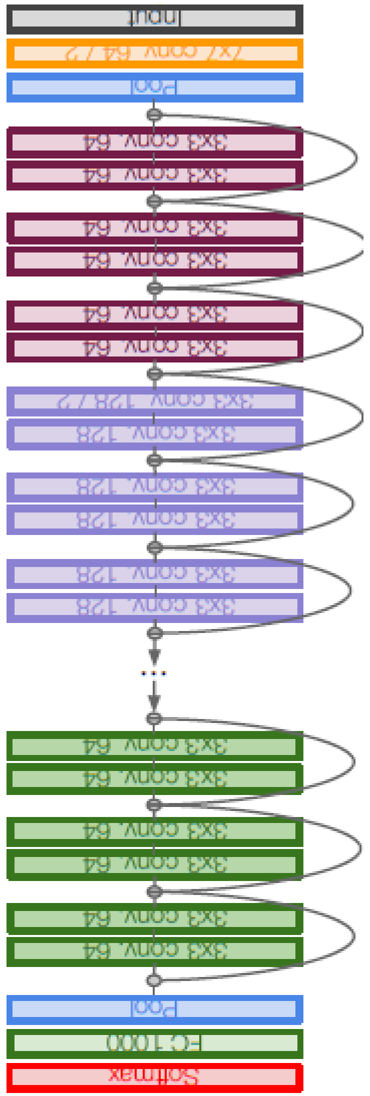
$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T})).$$

Special case:

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

This opens a highway for the gradient:

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

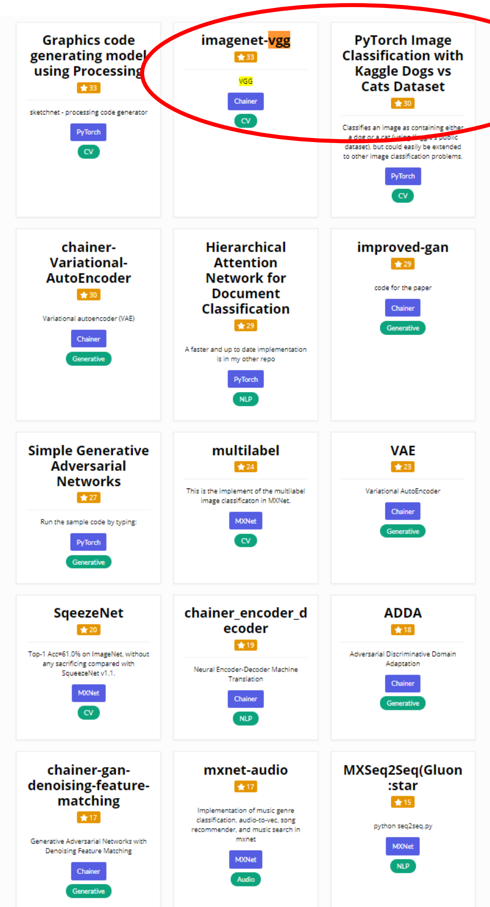


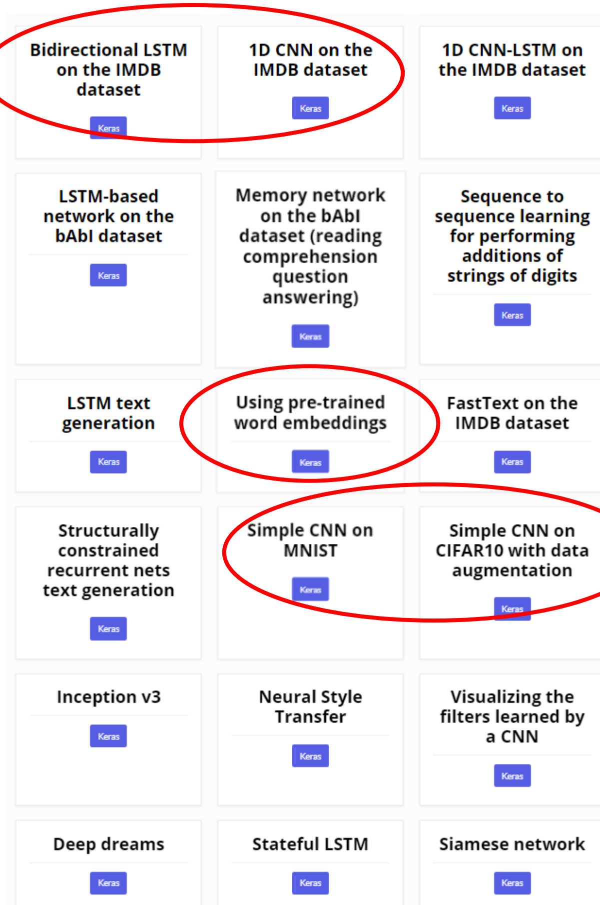
Srivastava, Greff, Schmidhuber 2014, “Highway Networks” https://arxiv.org/pdf/1505.00387.pdf

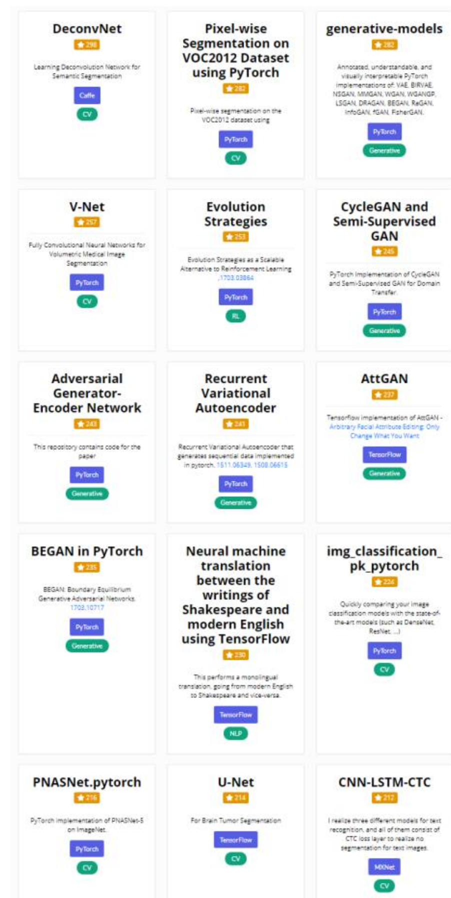
Model zoo: many pretrained NN are out there

https://modelzoo.co/

Base pretrained models and datasets in pytorch (MNIST, SVHN, CIFAR10, CIFAR100, STL10, AlexNet, VGG16, VGG19, ResNet, Inception, SqueezeNet)

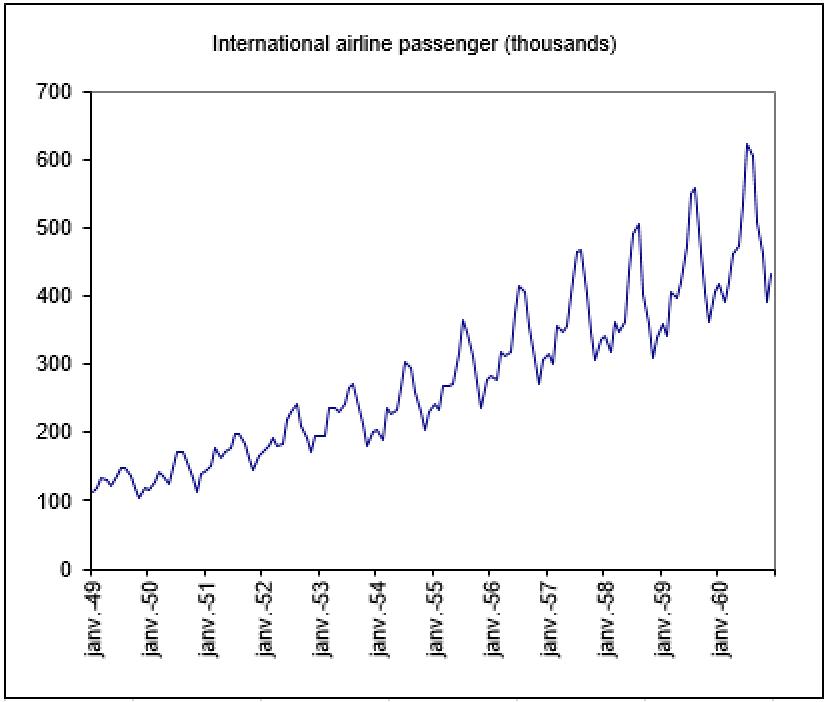


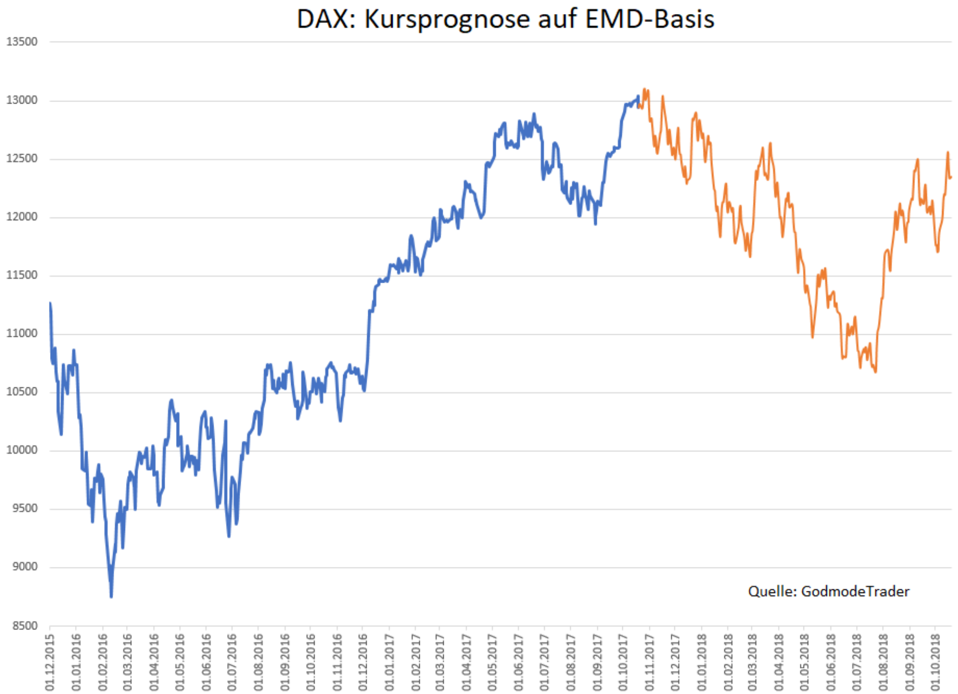




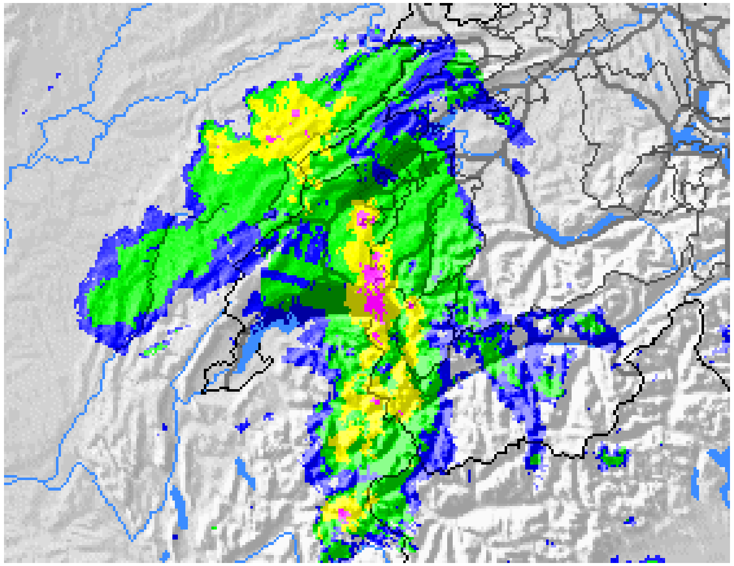
Sequence Data

Example Sequence Data: time-series





Example Sequence Data: videos



https://www.metradar.ch/2009_exp/pc/service.php

https://www.pinterest.ch/pin/460704236854535539/

Example sequence data: speech 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]

Example Sequence Data: text

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)

Mit freundlichen GrÃ¼ÃŸe

Modeling sequence data

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

E.g. Image Captioning. Image -> Seq of words

E.g. Sentiment Classification. Seq of words →Sentiment

E.g. Translation. seq of words (french) →seq of words (german)

E.g. Language models. seq of letters →seq of letters

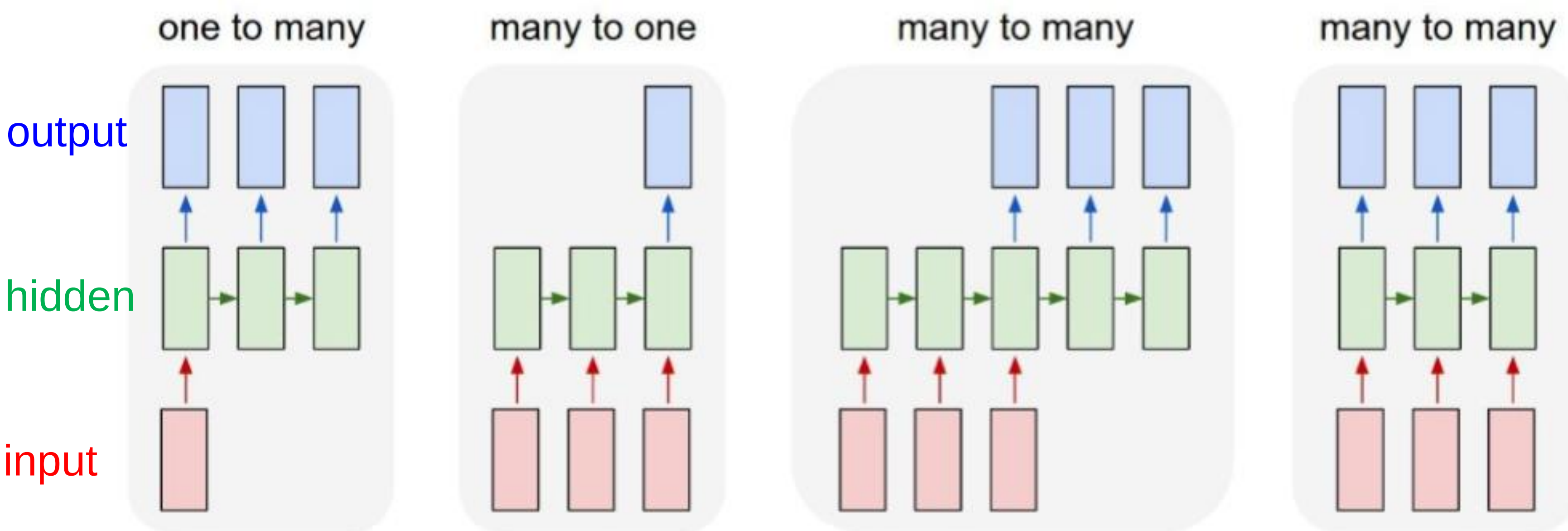
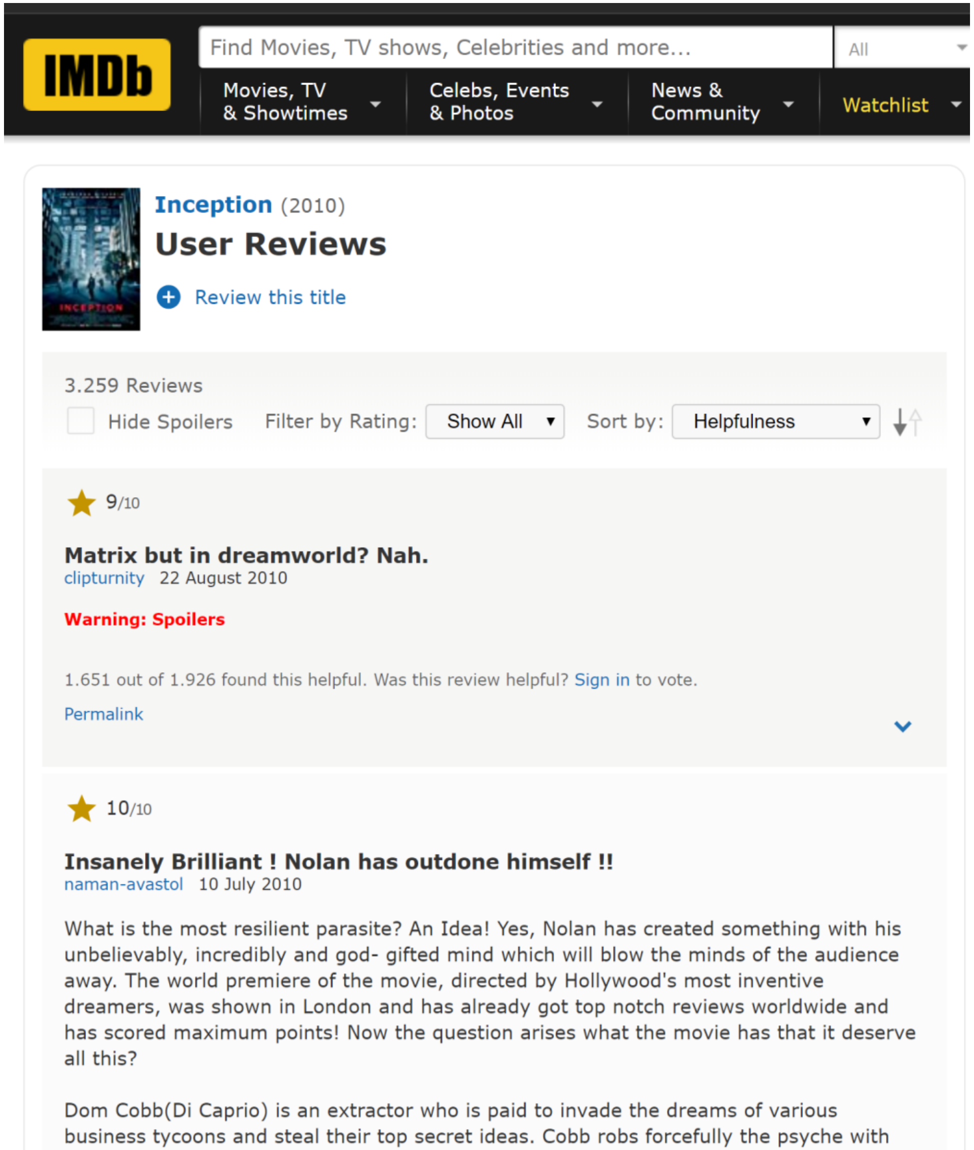


Illustration: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Possible task: Sentiment analysis with movie reviews



	review	sentiment
0	I went and saw this movie last night after bei...	1
1	Actor turned director Bill Paxton follows up h...	1
2	As a recreational golfer with some knowledge o...	1
3	I saw this film in a sneak preview, and it is ...	1
4	Bill Paxton has taken the true story of the 19...	1

Challenges:

1) We need to find a numeric representation of words (e.g. bag of words, or embedding)

2) We need to be able to handle inputs of different length.

Bag of words: Ignoring word order

- Count vectors or “bag of words”
  - Determine vocabulary (or alphabet, or word, or token)

Example:

Document 1: “The cat sat on the hat”

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

Bag of words (=word count vector):

document	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

Represent document as word count vector ignoring order of words (token).

It allows to represent each sentence as a numeric vector of the same length!

It can be seen as feature-vector and can be used for traditional classifiers as RF.

Getting Bag of words in sklearn: ignoring word order

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(max_features=6000, min_df=5, max_df=0.7)

X = vectorizer.fit_transform(documents).toarray()
```

consider only the 6000 most frequent words

take only words that appear in at least 5 different reviews

ignore all words that appear in more than 70% of all reviews

Get from text to ordered numeric vector
Step 1) Tokenize text, 1-hot-encoding

- Determine the size N of the relevant vocabulary
- Each token (word) is represented by a numeric value between 0 and N-1
- Corresponding 1-hot-encoded representations have length N

Example: Look at vocabulary with N=9 to tokenize 2 text-samples

Vocabulary: `{'the': 1, 'cat': 2, 'sat': 3, 'on': 4, 'mat': 5, 'dog': 6, 'ate': 7, 'my': 8, 'homework': 9}`

Text-sample: "The cat sat on the hat" "The dog ate my homework"

Tokenized: `[1, 2, 3, 4, 1, 5]` `[1, 6, 7, 8, 9]`

1-hot:

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

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

Tokenizing in keras

Creates a tokenizer, configured to only take into account the 1,000 most common words

```
from keras.preprocessing.text import Tokenizer

samples = ['The cat sat on the mat.', 'The dog ate my homework.']

tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(samples)

sequences = tokenizer.texts_to_sequences(samples)

one_hot_results = tokenizer.texts_to_matrix(samples, mode='binary')

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

Builds the word index

Turns strings into lists of integer indices

You could also directly get the one-hot binary representations. Vectorization modes other than one-hot encoding are supported by this tokenizer.

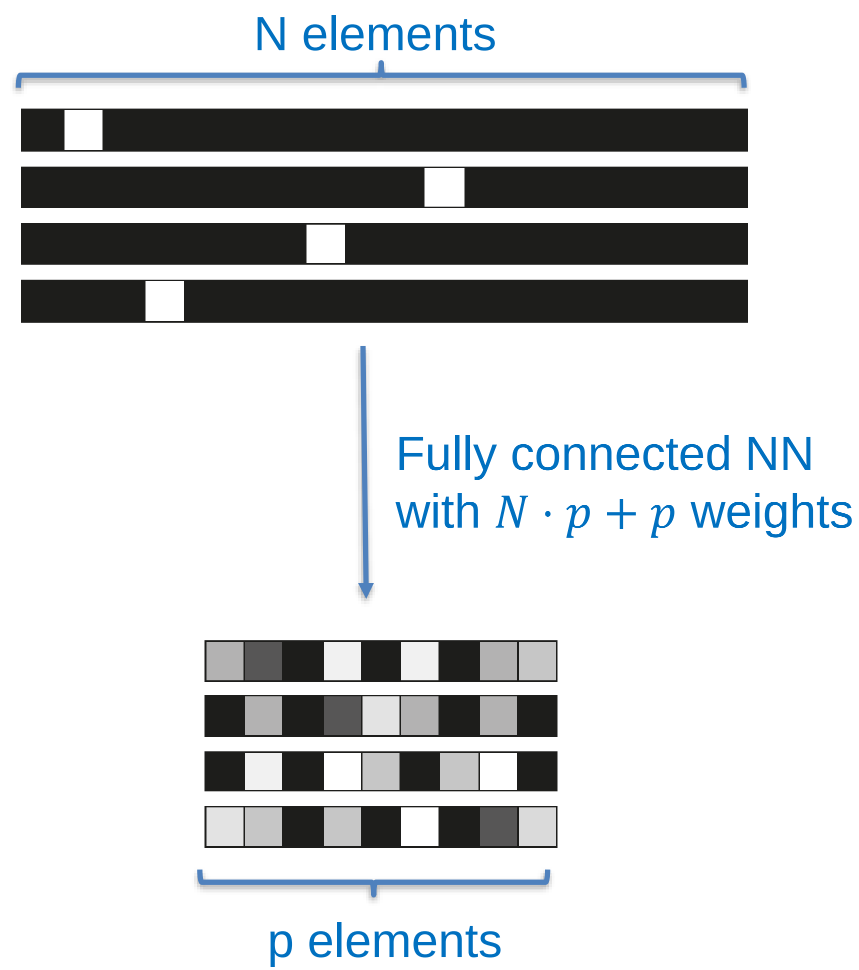
How you can recover the word index that was computed

Credits: F. Chollet’s Book: DL with Python

Go from 1-hot-encodings to word embeddings

1-hot encodings
- Based on vocabulary of size N
- sparse: one 1 and N-1 zeros
- High-dim: vector-length = N

Word embedding's are
- Dense
- Low-dimensional: vector-length = p
- Learned from data via fcNN N→p



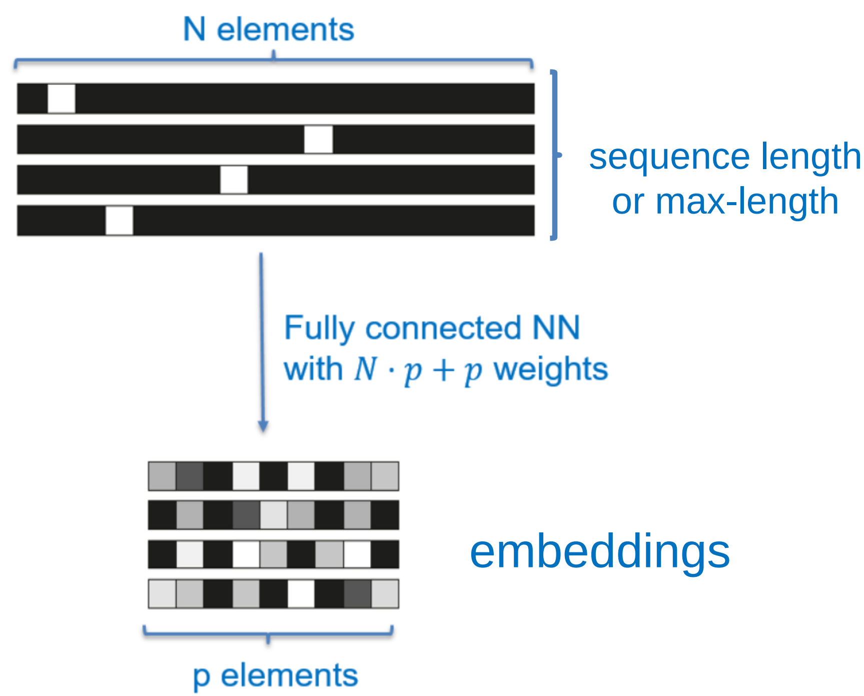
Wordembedding layer in keras

```
from keras.layers import Embedding

embedding_layer = Embedding(1000, 64)
```

The Embedding layer takes at least two arguments: the number of possible tokens (here, 1,000: 1 + maximum word index) and the dimensionality of the embeddings (here, 64).

The embedding layer returns a 3D floating-point tensor of shape `(samples, sequence_length, embedding_dimensionality)`.

- Flatten results in vector of length: `seq-length*p`
- Pool over sequence length results in vector of length: `p`
- Input to RNN layer or a 1D convolution layer



A simple text classification NN using flattened embeddings

Specifies the maximum input length to the Embedding layer so you can later flatten the embedded inputs. After the Embedding layer, the activations have shape (samples, maxlen, 8).

Flattens the 3D tensor of embeddings into a 2D tensor of shape (samples, maxlen * 8)

Adds the classifier on top

```
from keras.models import Sequential
from keras.layers import Flatten, Dense

model = Sequential()
model.add(Embedding(10000, 8, input_length=maxlen))

model.add(Flatten())

model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_split=0.2)
```

A text classification NN using pooled embeddings

```
from keras.models import Sequential
from keras.layers import Dense, Embedding,GlobalAveragePooling1D,Dropout

EMBEDDING_DIM = 30

model = Sequential()
model.add(Embedding(vocab_size, EMBEDDING_DIM, input_length=(None)))
model.add(GlobalAveragePooling1D())
model.add(Dropout(0.5))
model.add(Dense(20, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, None, 30)          1868910   
_________________________________________________________________
global_average_pooling1d_1 ( (None, 30)                0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 30)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 20)                620       
_________________________________________________________________
dropout_2 (Dropout)          (None, 20)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 21        
=================================================================
```

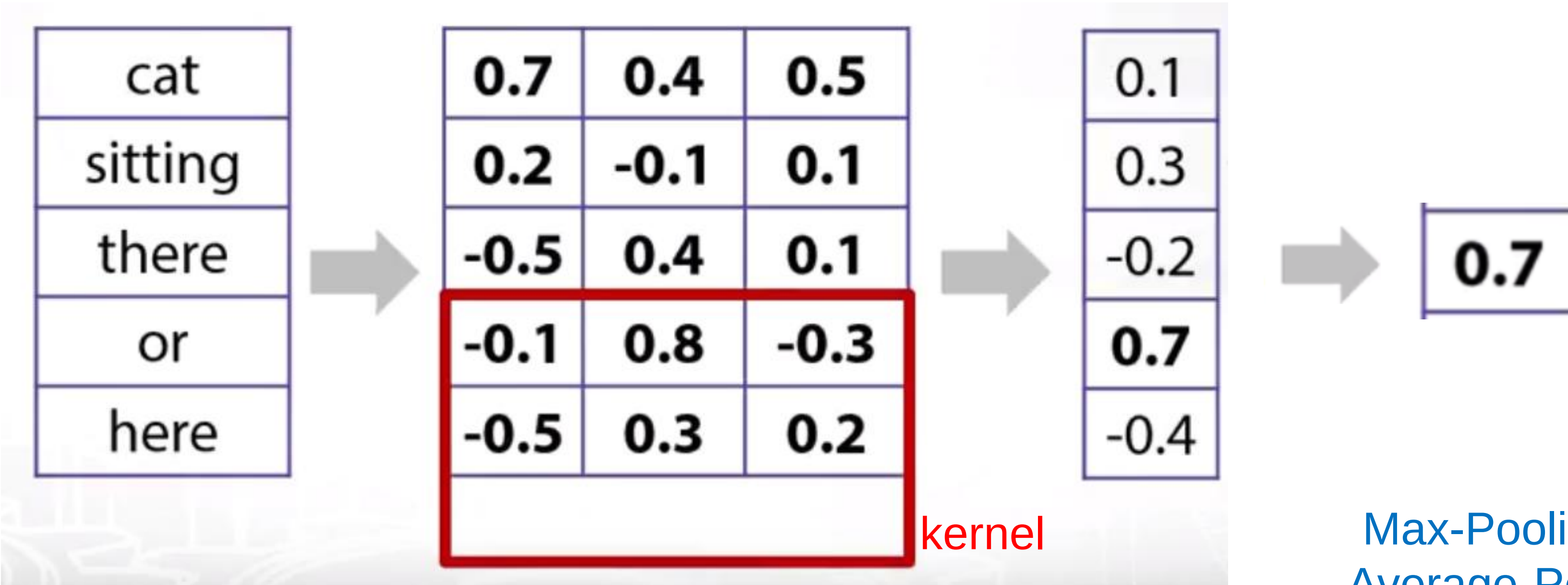
Applying 1D convolution embedding sequences

Sequence of tokens

Sequence of embeddings

Sequence of convolution results

Max-pooling result



Credits: https://www.youtube.com/watch?v=wNBaNhvL4pg

Width of kernel = embedding dimension
Height of kernel = #neighboring tokes

Max-Pooling or Average-Pooling makes result independent of sequence length

We slide kernel only in direction → 1D convolution

A text classification NN feeding embeddings to 1D conv

```
from keras.models import Model
from keras.layers import Input, Dense, Concatenate, Dropout, Embedding, Conv1D,GlobalMaxPooling1D,GlobalAv
eragePooling1D
EMBEDDING_DIM = 30


a = Input(shape=(max_length,))
x = Embedding(vocab_size, EMBEDDING_DIM)(a)
x1 = Conv1D(filters=50,kernel_size=(3),activation="relu",padding="same")(x)
x2 = Conv1D(filters=50,kernel_size=(5),activation="relu",padding="same")(x)
x3 = Conv1D(filters=50,kernel_size=(7),activation="relu",padding="same")(x)

g1 = GlobalAveragePooling1D()(x1)
g2 = GlobalAveragePooling1D()(x2)
g3 = GlobalAveragePooling1D()(x3)
conc= Concatenate()([g1,g2,g3])
conc = Dropout(0.3)(conc)
conc = Dense(50, activation='relu')(conc)
conc = Dropout(0.3)(conc)
out= Dense(1, activation='sigmoid')(conc)
model = Model(inputs=a, outputs=out)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

Inception idea:
Use several filter-heights in parallel.

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/

Recurrent NN have memory

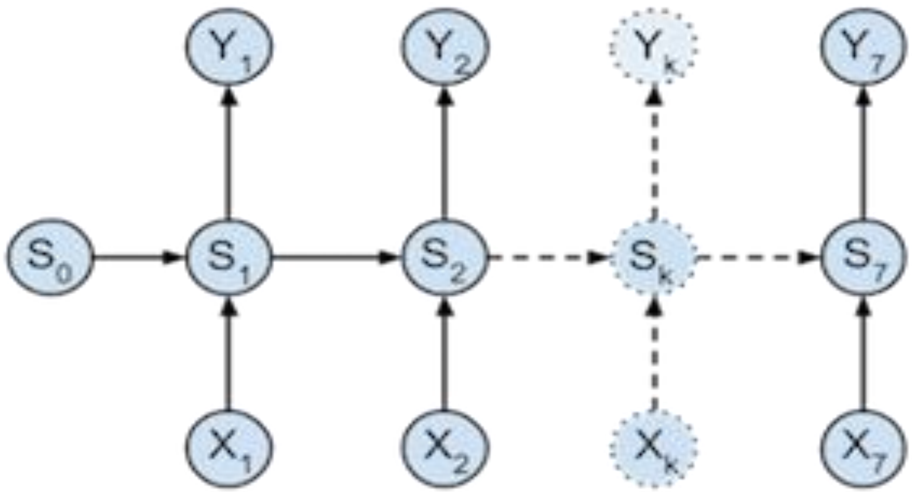
Challenge 2: How to handle input of different lengths?

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=prob$_{spam}$, y_2=prob$_{spam}$, ….



Hidden memory State:

S_1=state after first sentence

S_2=state after first sentence

…

Input: x_1=vector$_{sentence1}$, x_2=vector$_{sentence2}$, ….

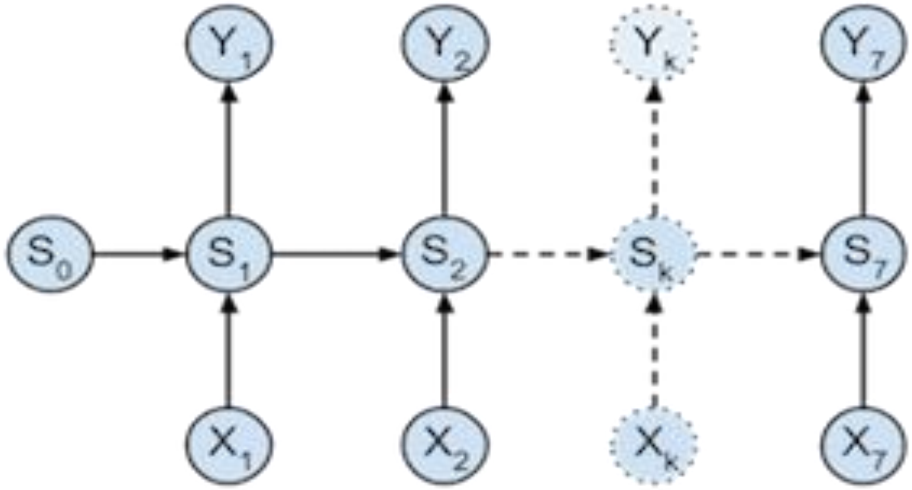
How to choose the dimensions of the hidden state?

The input **x** is a vector of the size of our vocabulary.

The output **y** is a vector of size 2 (spam/no-spam or passed/failed).

The hidden state **h** (or **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
…

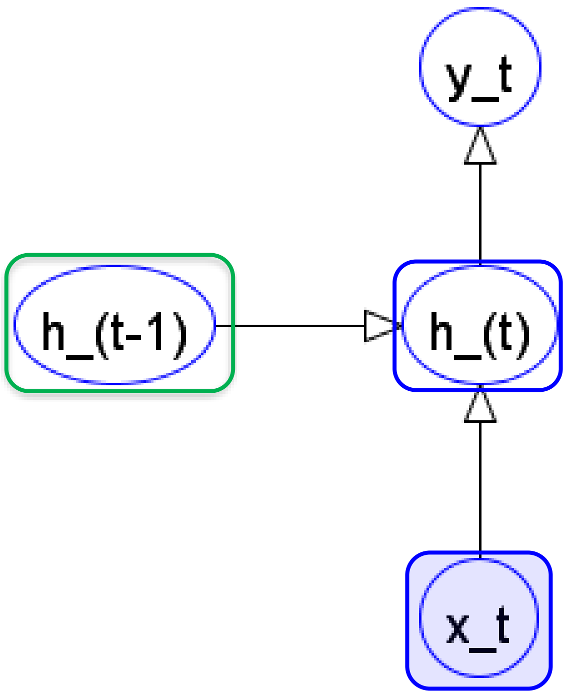
Input: $x_1=\text{vector}_{\text{sentence1}}$, $x_2=\text{vector}_{\text{sentence2}}$, ….

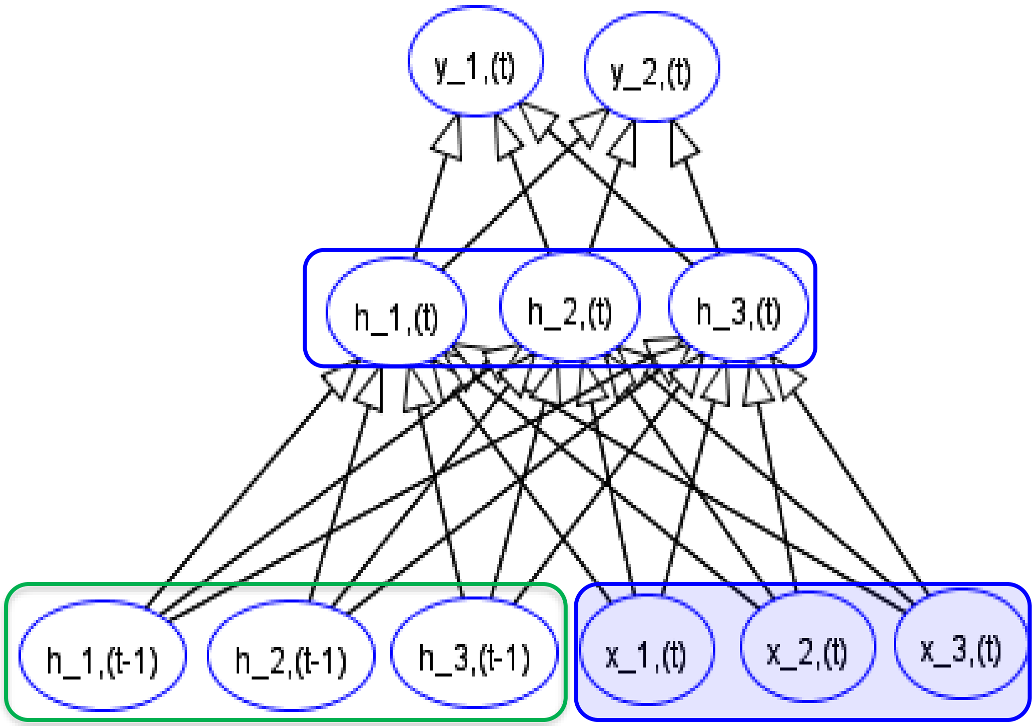
Two representations of a RNN at time t

x has 3 components – a very small vocabulary ;-)

y has 2 components (spam/no-spom for mails, passed/failed for scoring essays).

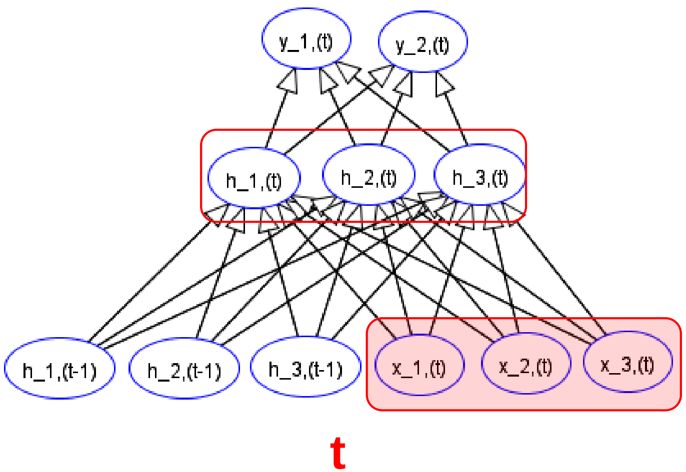
h 3 components to capture abstract concepts (e.g. fraud/sex/conference for emails, or copied/boring/original for essays) and is initialized at t=0 with (0,0,0).



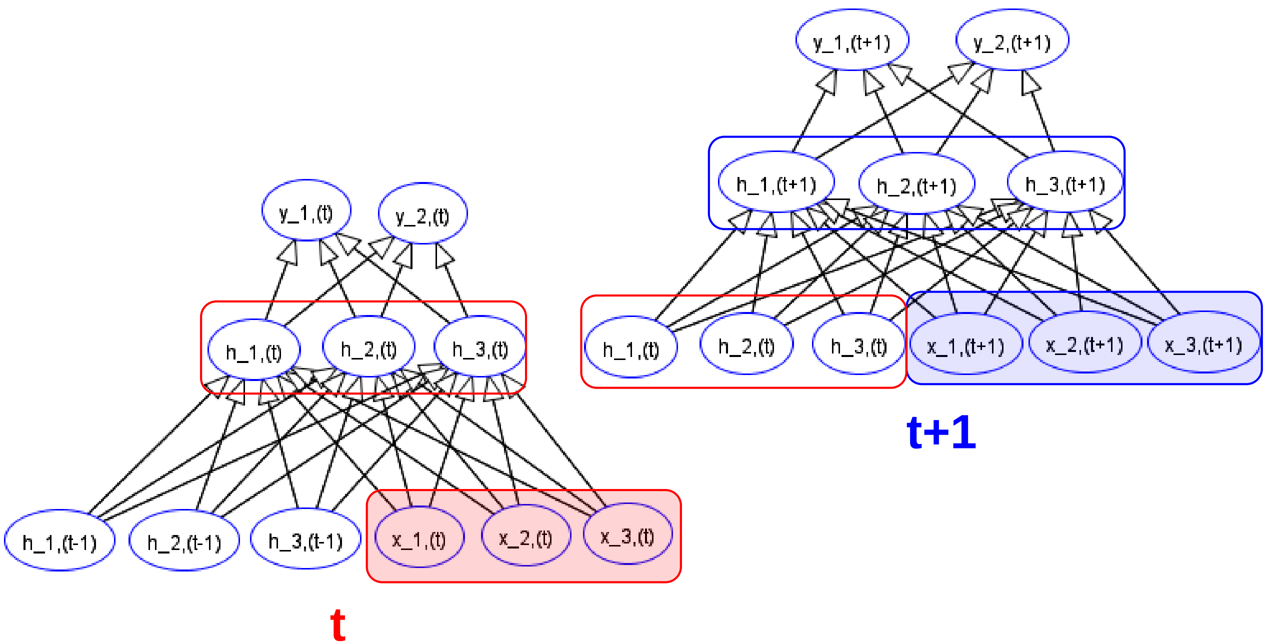


Stepping through an RNN

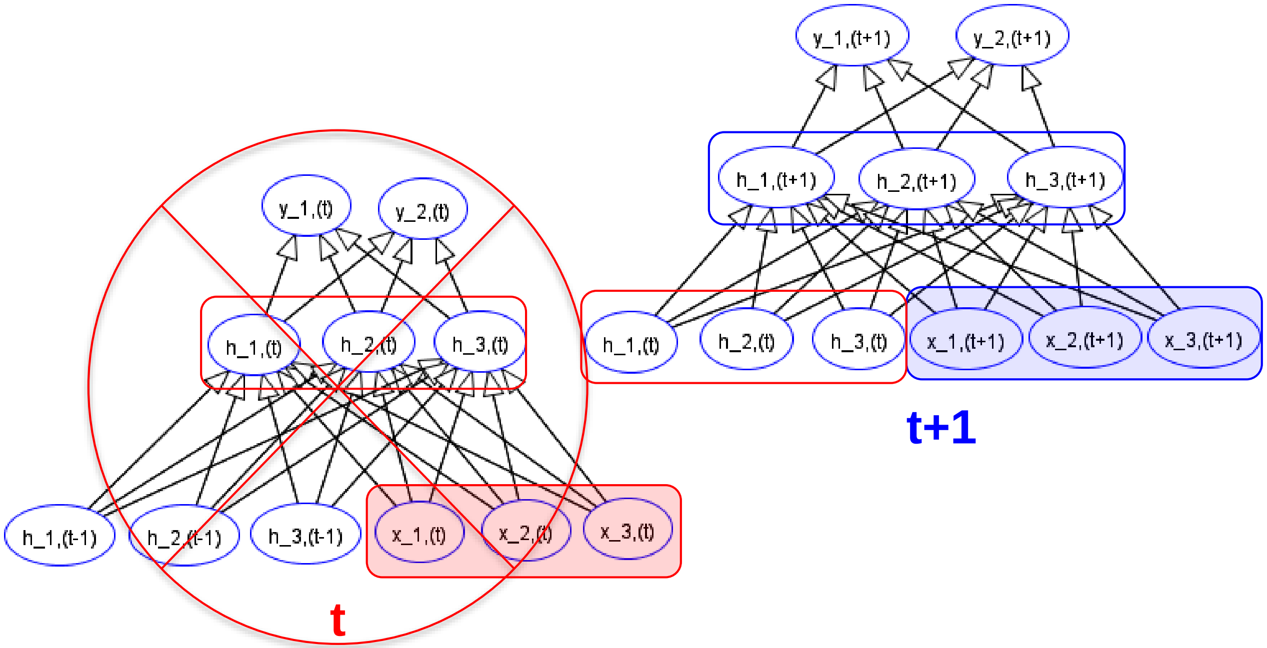
A simple RNN at 3 successive time steps



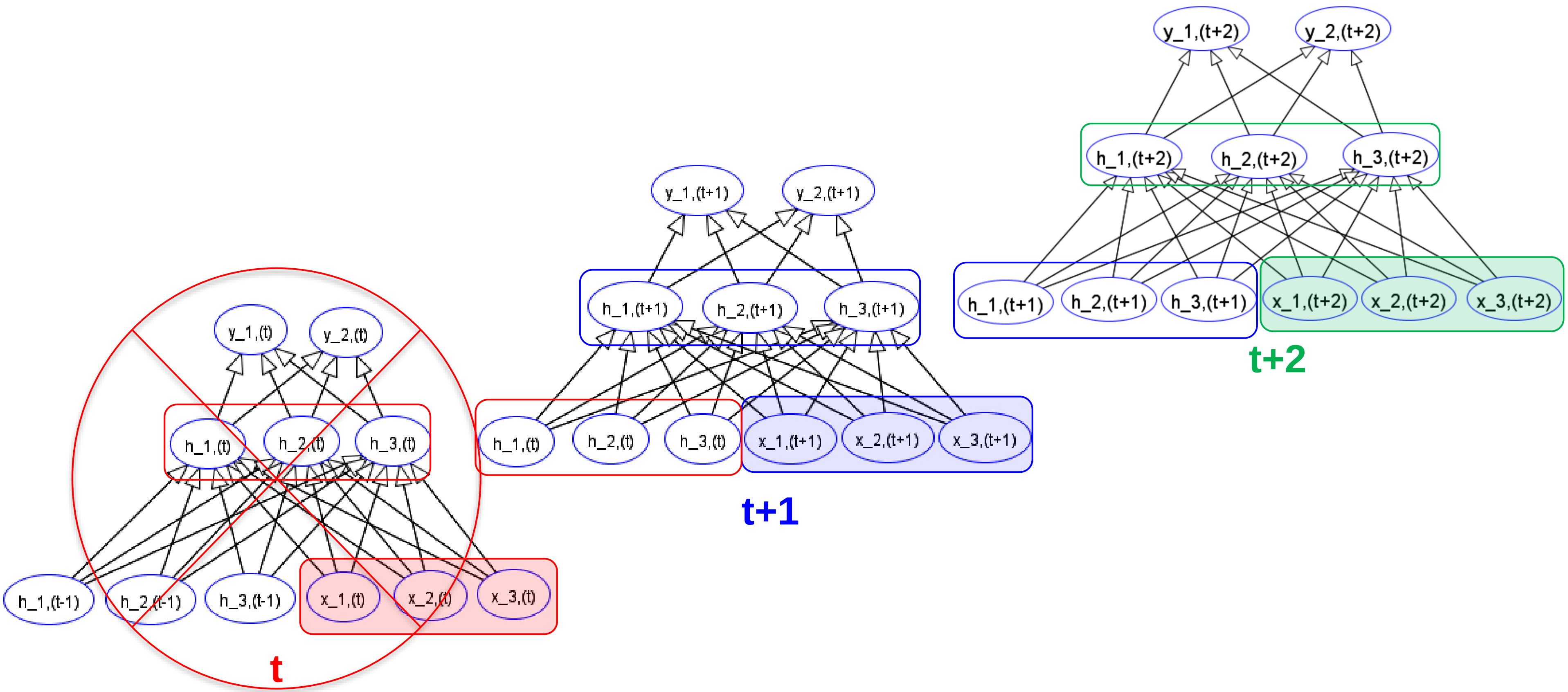
A simple RNN at 3 successive time steps



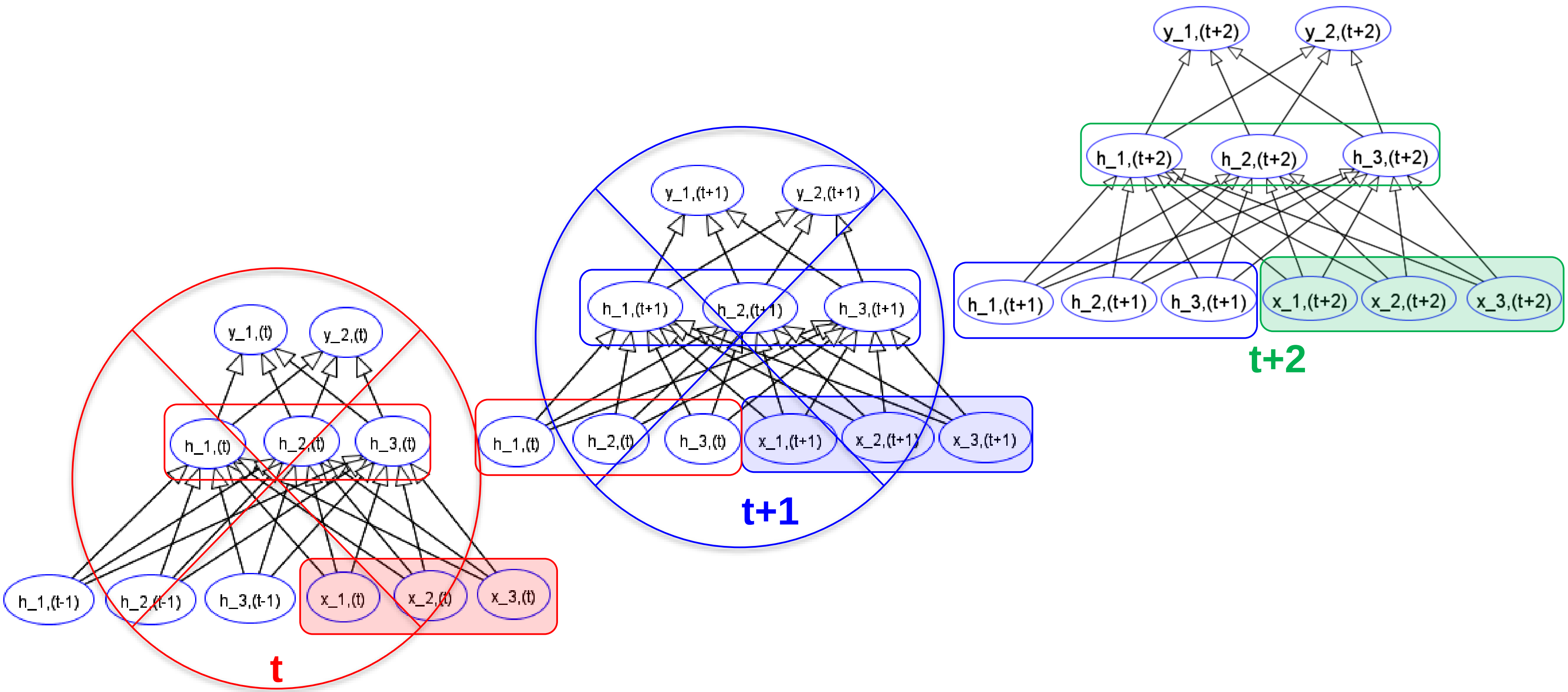
A simple RNN at 3 successive time steps



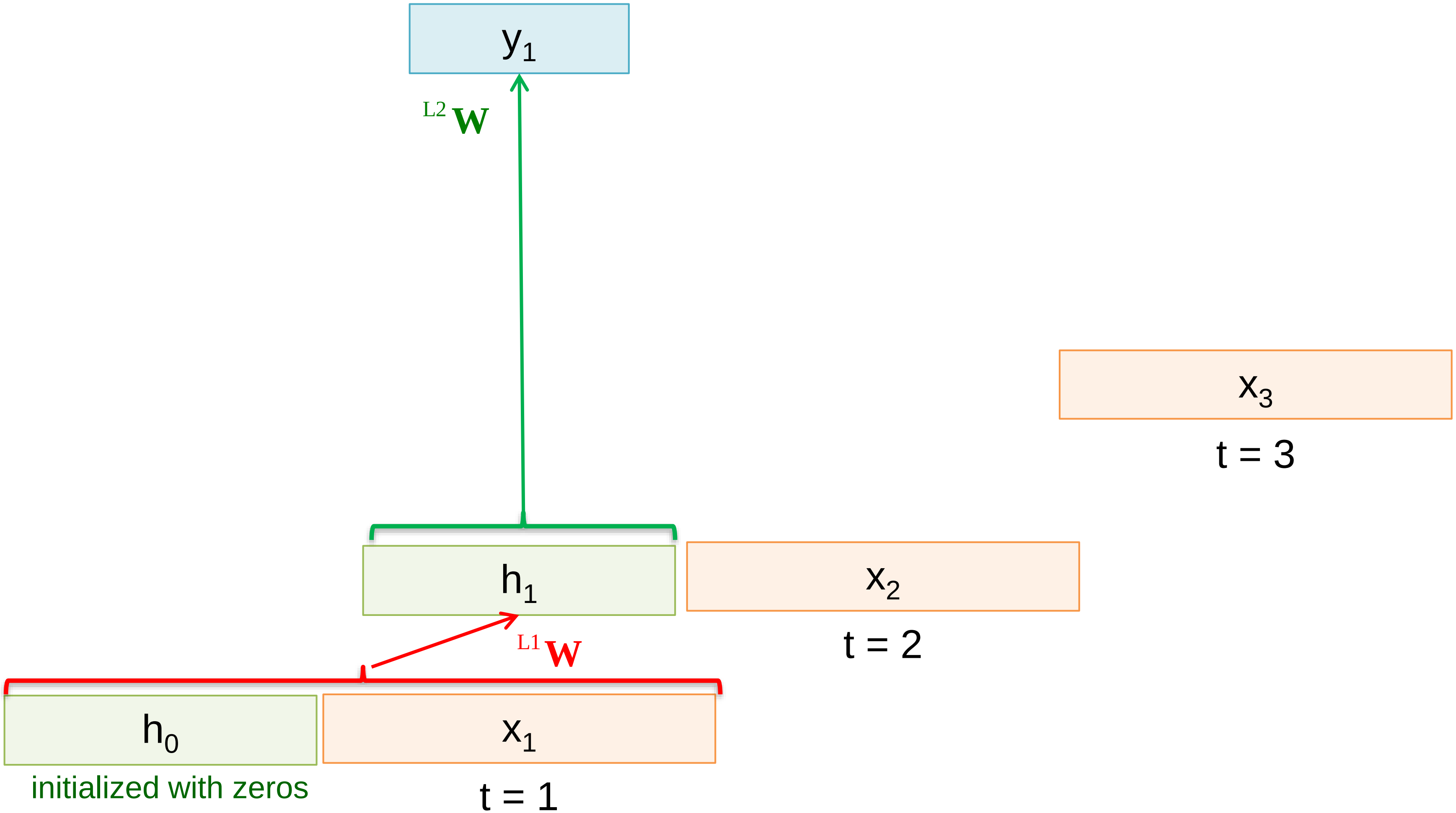
A simple RNN at 3 successive time steps



A simple RNN at 3 successive time steps

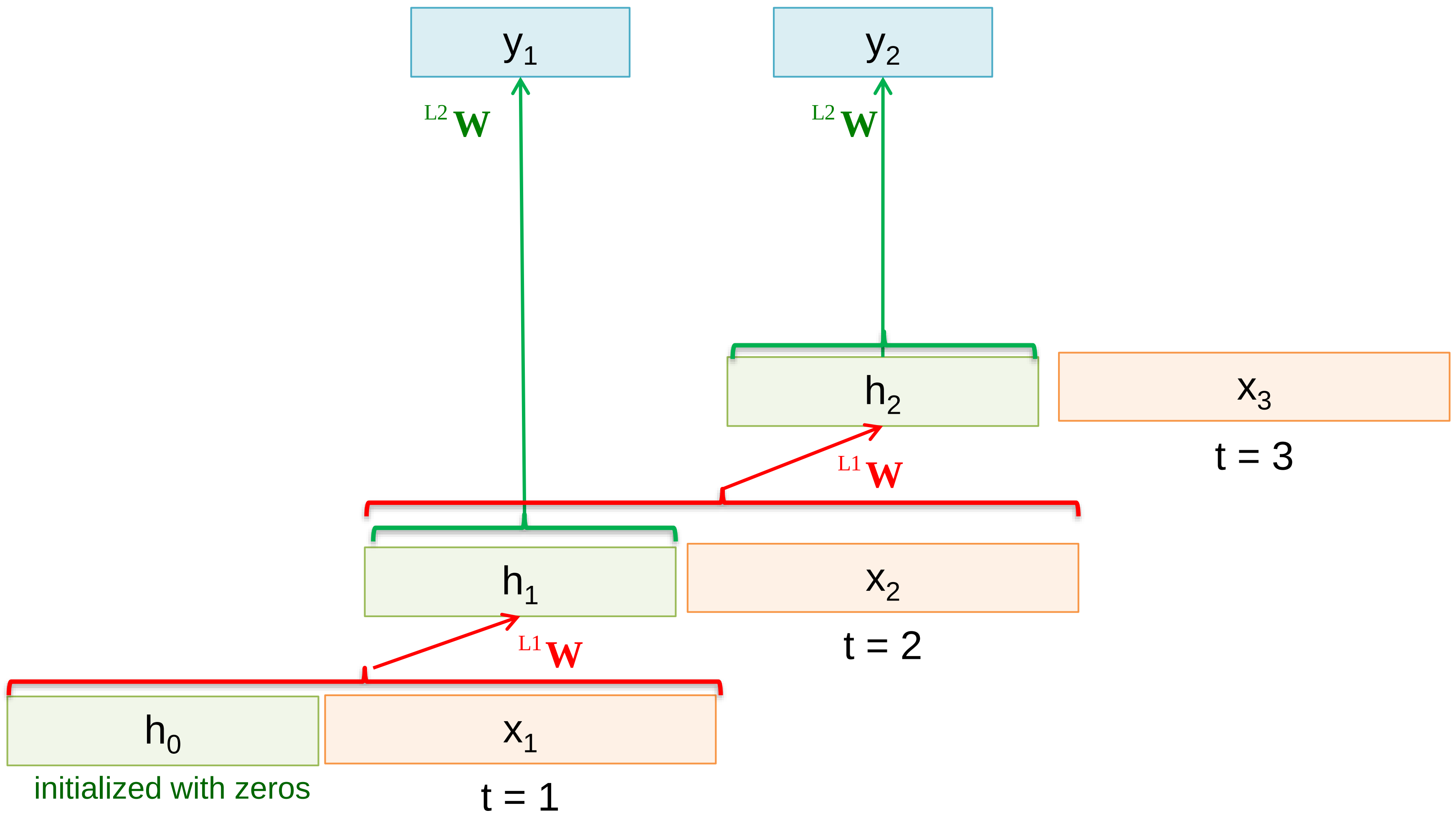


An RNN shares weights across all time steps



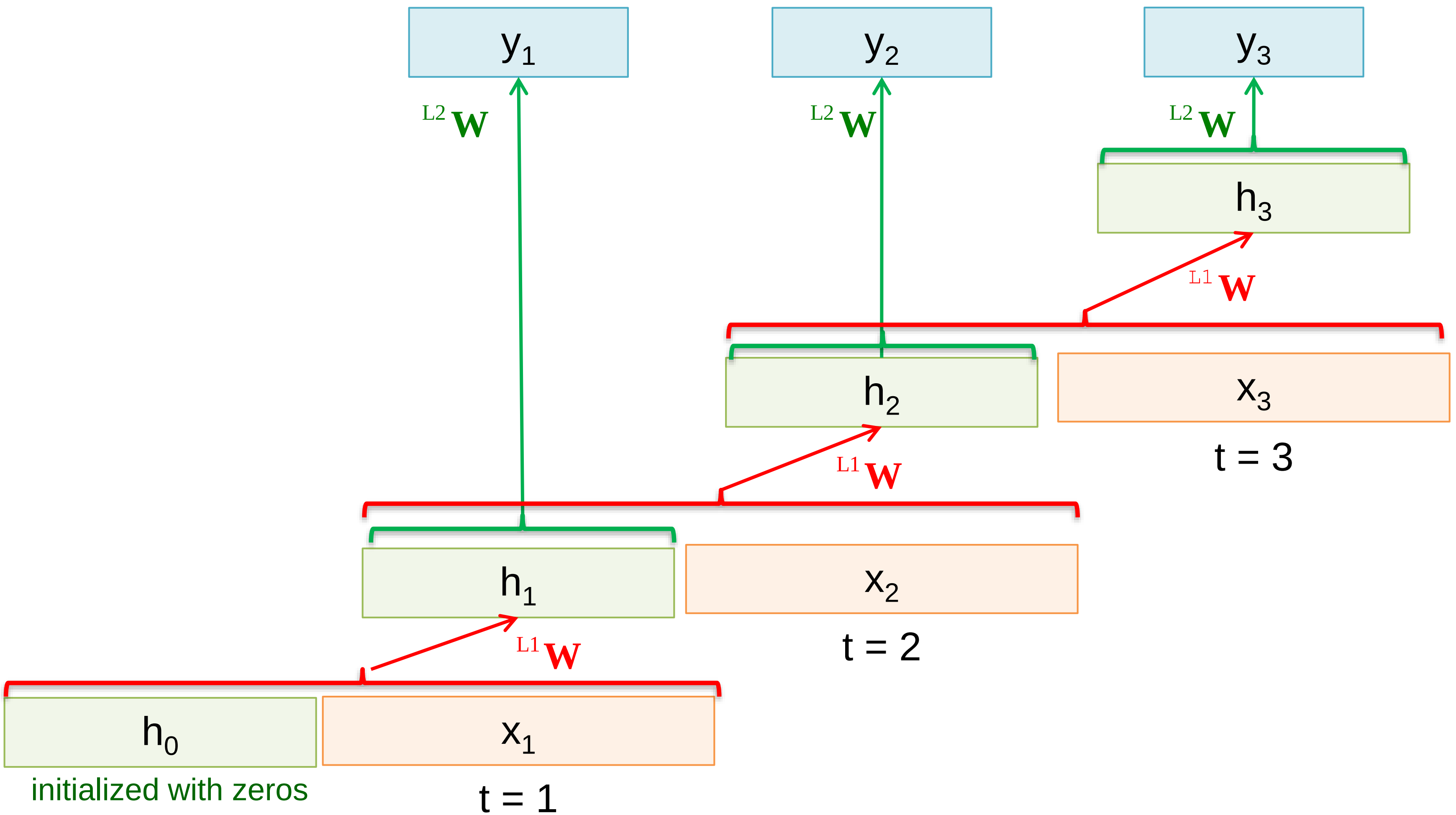
Imagine a trained RNN with fixed weight matrices in layer 1 and layer 2.

An RNN shares weights across all time steps



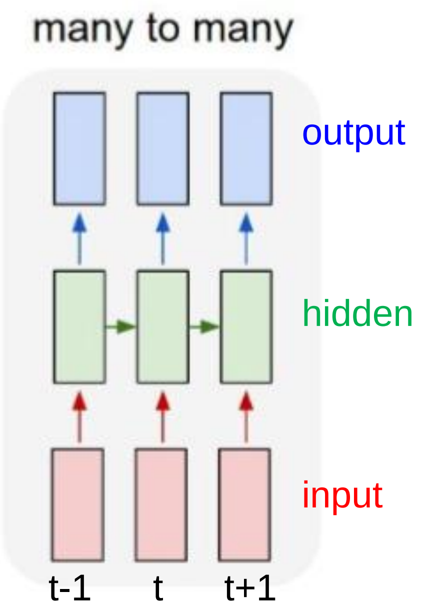
We use at each time step the same weight matrices between the layers!

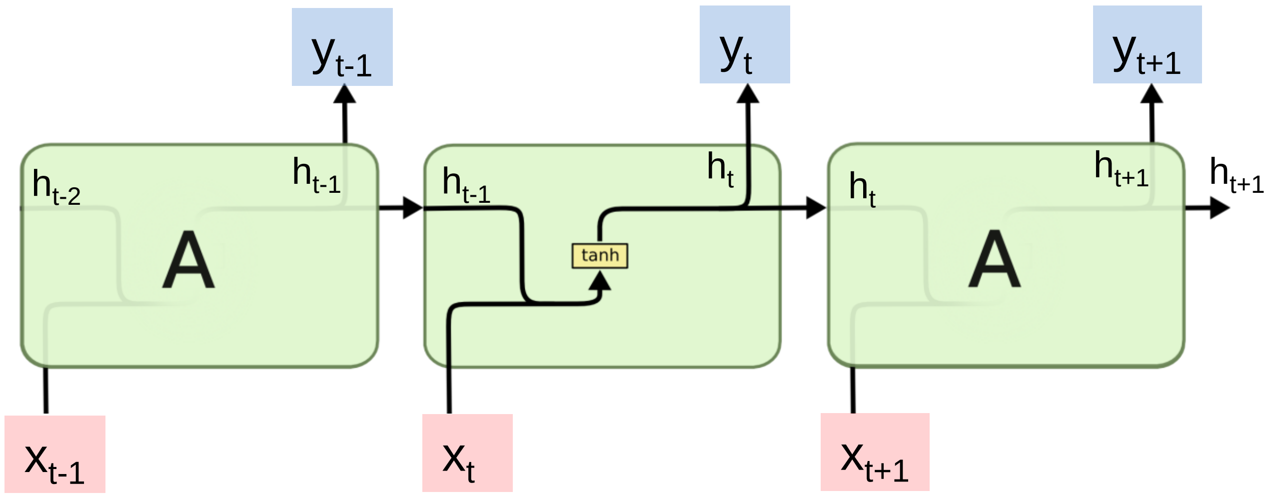
An RNN shares weights across all time steps



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

Using diagrams to represent an RNN

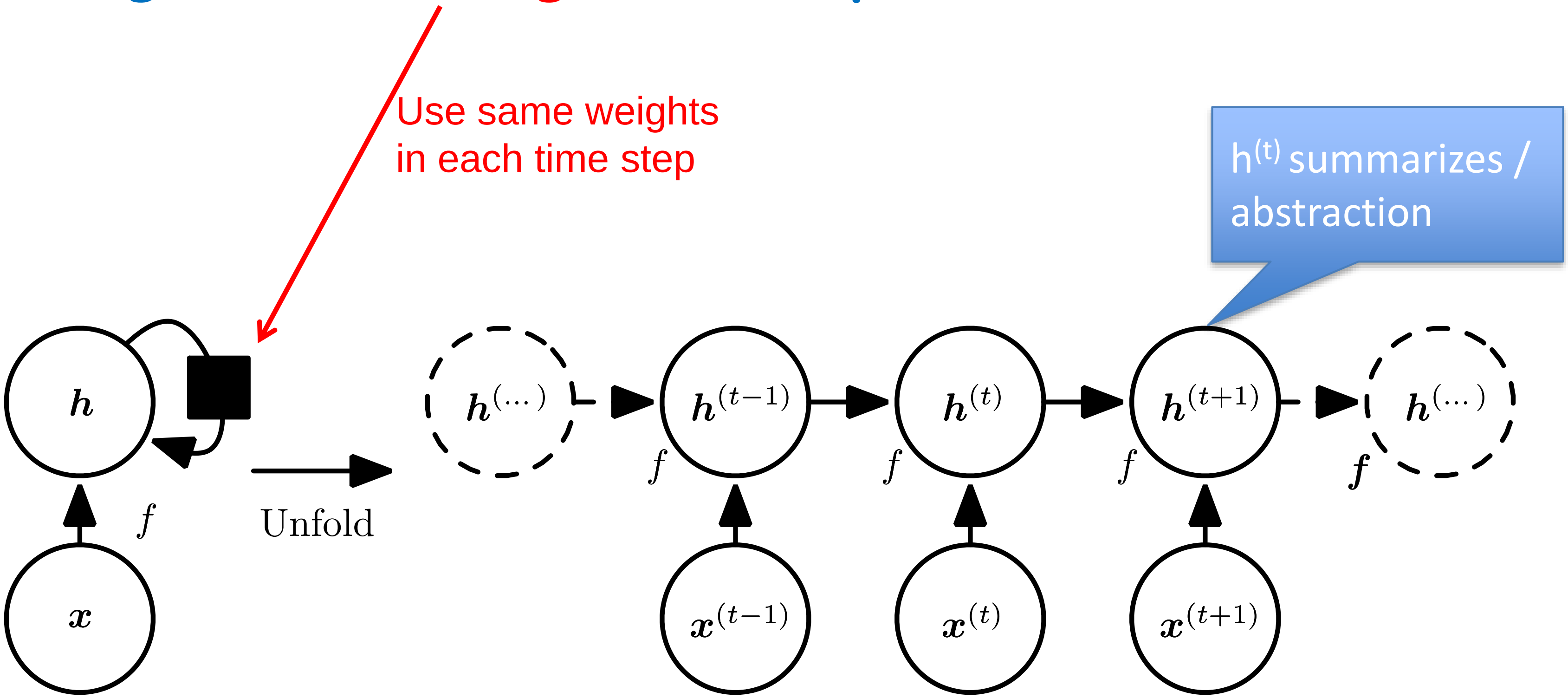




$$\mathbf{A} = f_{\mathbf{W}}(\mathbf{h}_{t-1}, \mathbf{x}_t) = \tanh\left([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W} + \mathbf{b}\right)$$

Illustration (adapted): http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Using a circuit diagram to represent an RNN

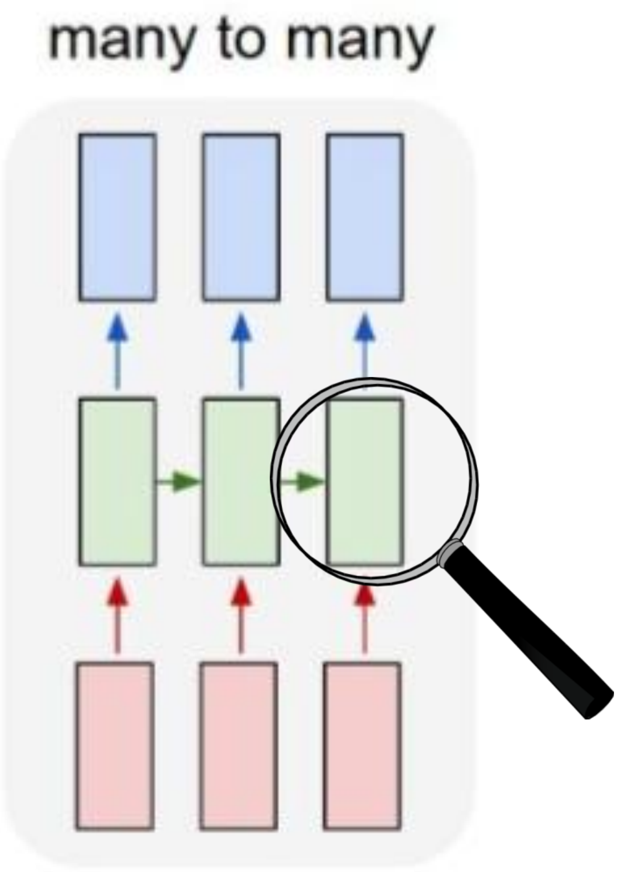


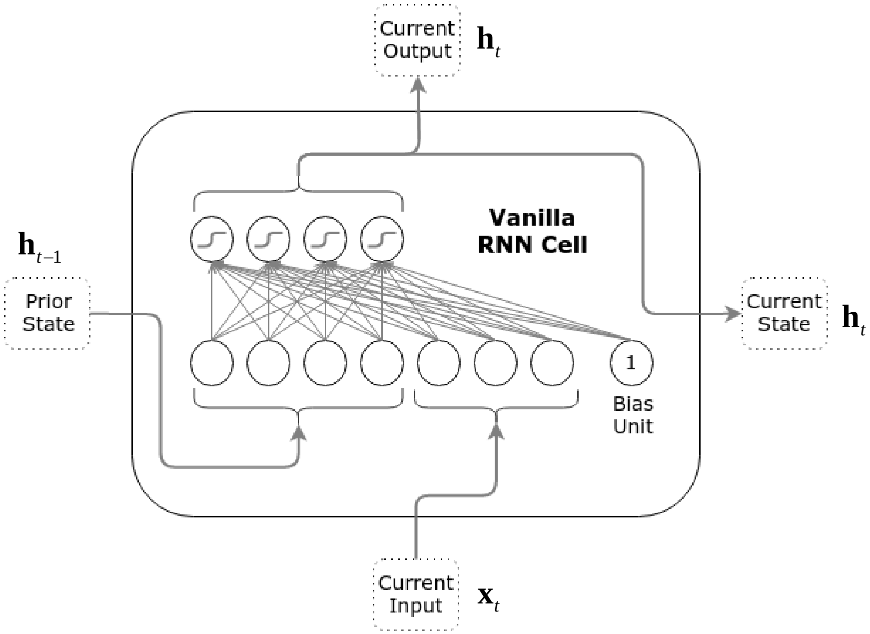
Left: Circuit Diagram (black square delay of one time step)

Right: Unrolled / unfolded

Illustration: http://www.deeplearningbook.org/

Looking into a RNN "cell"





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

image credits: https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html

Loss construction in an RNN

Determine the loss contribution of instance 1

mini-batch of size M=8

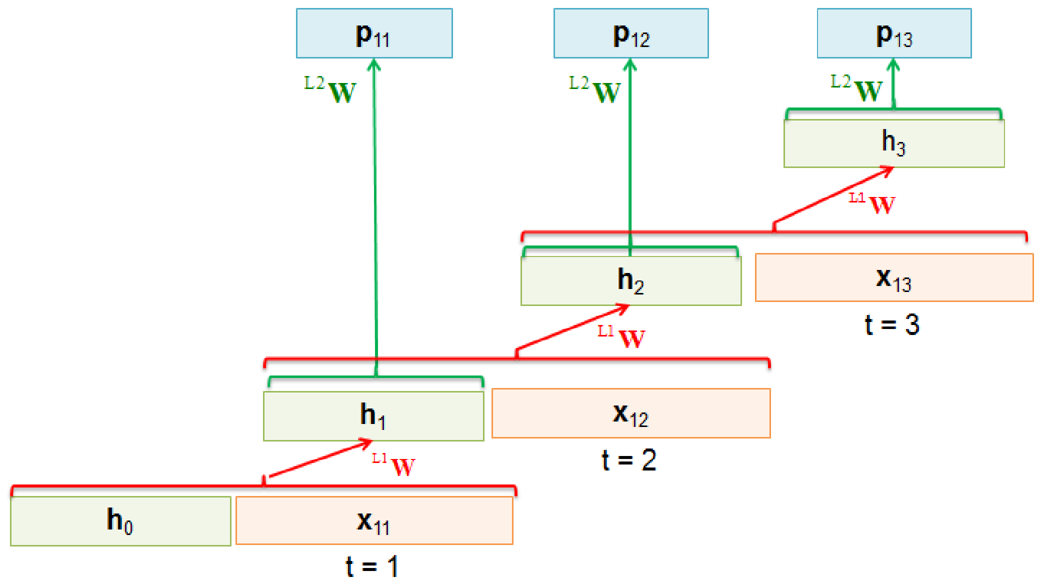
train data input (S=len(seq)=3):

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:

$$\boxed{\text{Loss\_1}} = \sum_{s=1}^{3}\left(-\sum_{k=1}^{2} y_{1sk}\cdot\log\left(p_{1sk}\right)\right)$$

Determine the loss contribution of instance 2

mini-batch of size M=8

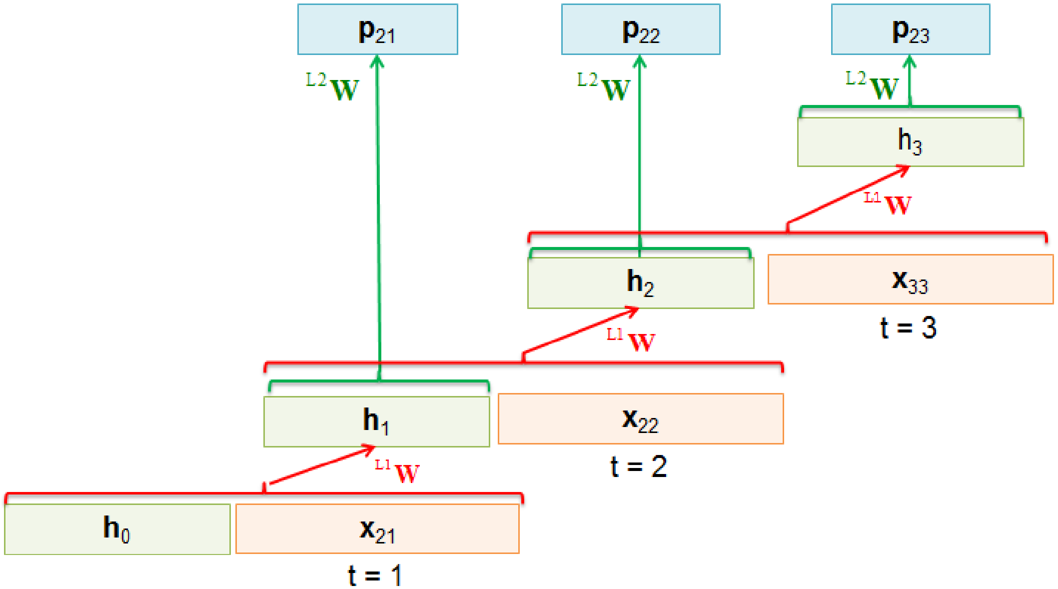
train data input (S=len(seq)=3):

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:

$$\boxed{\text{Loss\_2}} = \sum_{s=1}^{3} \left(-\sum_{k=1}^{2} y_{2sk} \cdot \log \left(p_{2sk} \right) \right)$$

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)

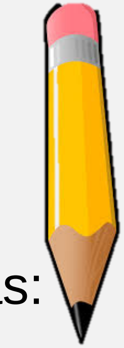
Cost C or Loss is given by the cross-entropy averaged over all instances in mini-batch:

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

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

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



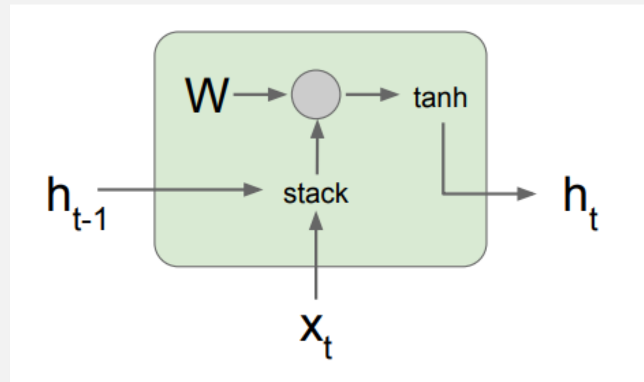
- Given the hidden state at t-1, the input x at t and the weight matrix as:

$$h_{t-1} = \begin{pmatrix} 0, & 1 \end{pmatrix}$$

$$x_t = \begin{pmatrix} 1, & 0 \end{pmatrix}$$

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

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



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

- Calculate the activation A of the hidden state h_t at time t.

Solution

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

$$x_t = (1, 0)$$

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

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

$$\Rightarrow h_t = th((-1, 1.5)) \simeq (-0.76, 0.91)$$

RNN in Keras

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

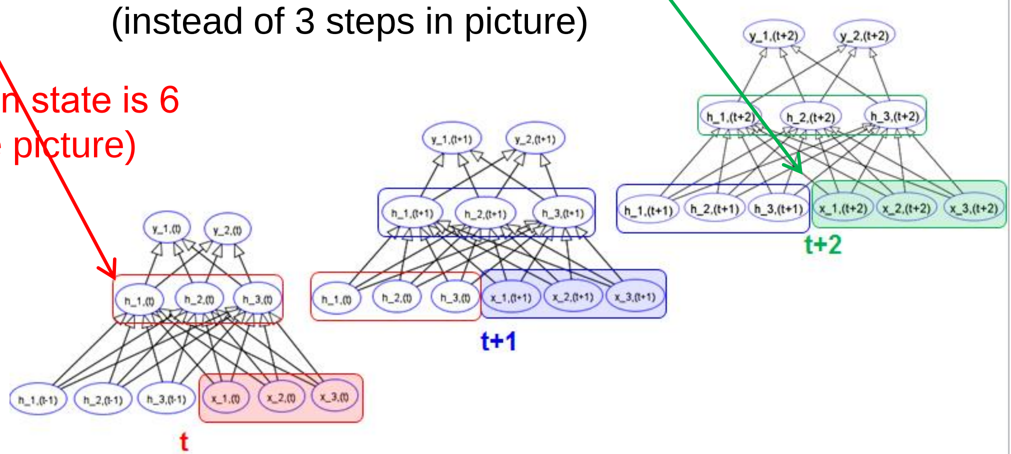
```
model = keras.models.Sequential()

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

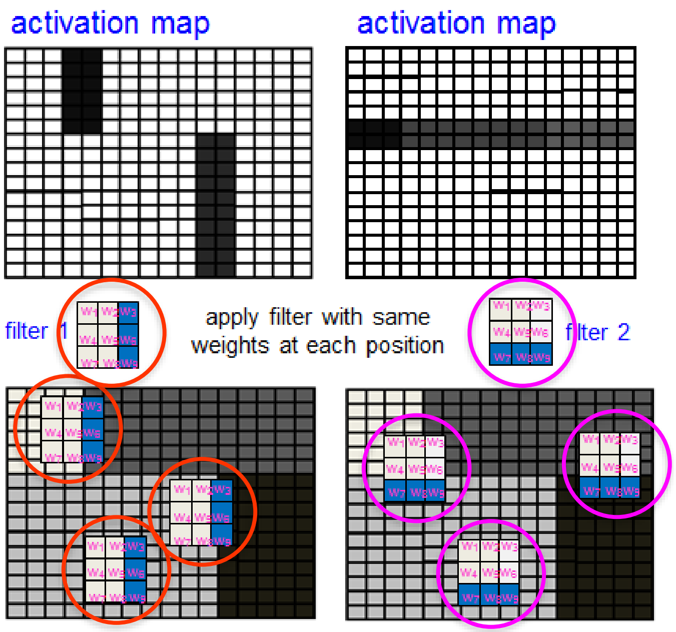
each training sequence consists out of 50 elements (instead of 3 steps in picture)

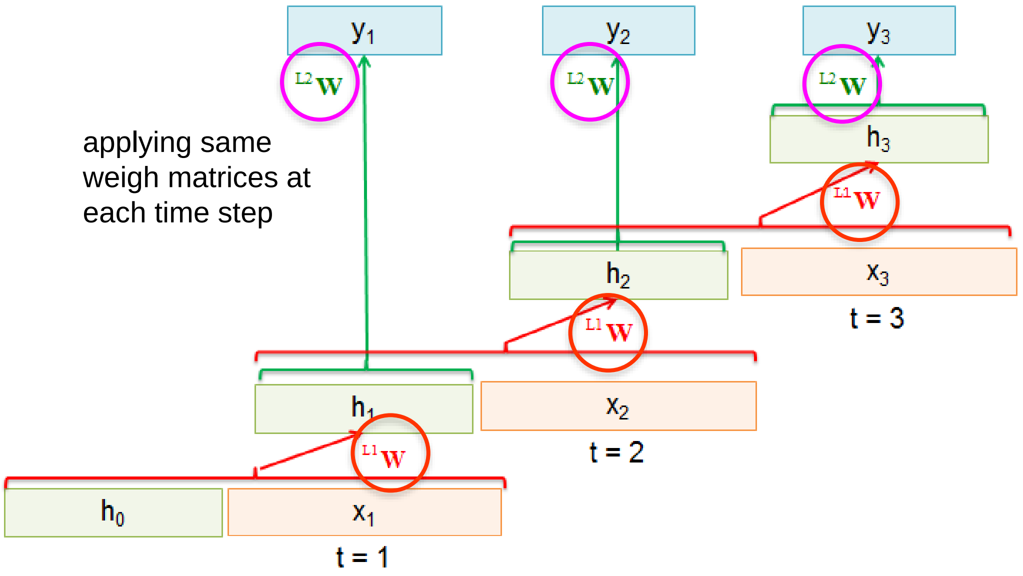
“capacity” of hidden state is 6 (instead of 3 in the picture)



Common tricks in RNN & CNN and some differences

CNN and Recurrent Network share weights



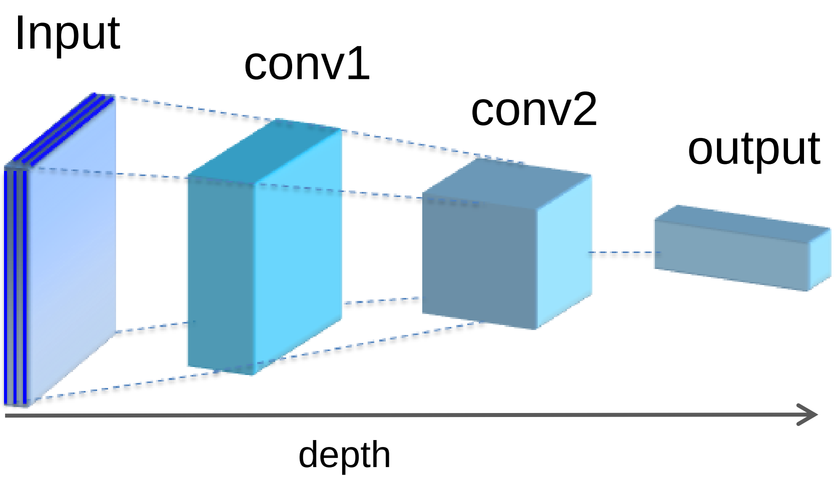


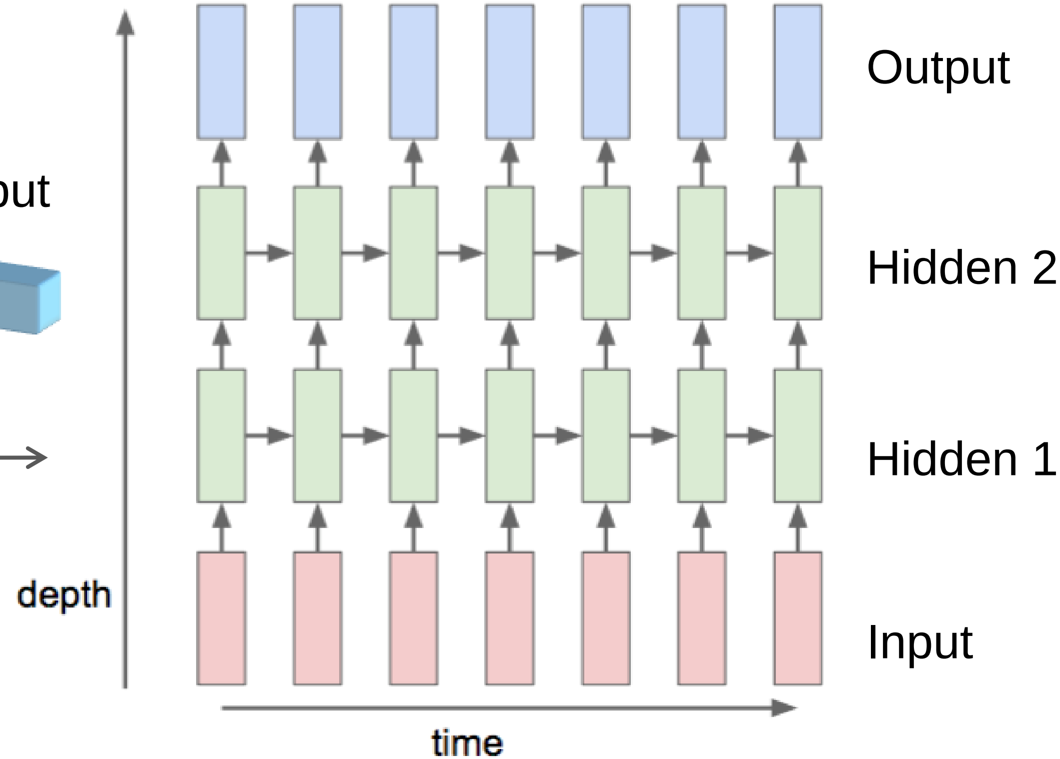
CNN share weights between different local regions of the image

RNN share weights between time steps

Remark: no weight sharing in fully connected NN

Also in RNN we can go deep for hierarchical features

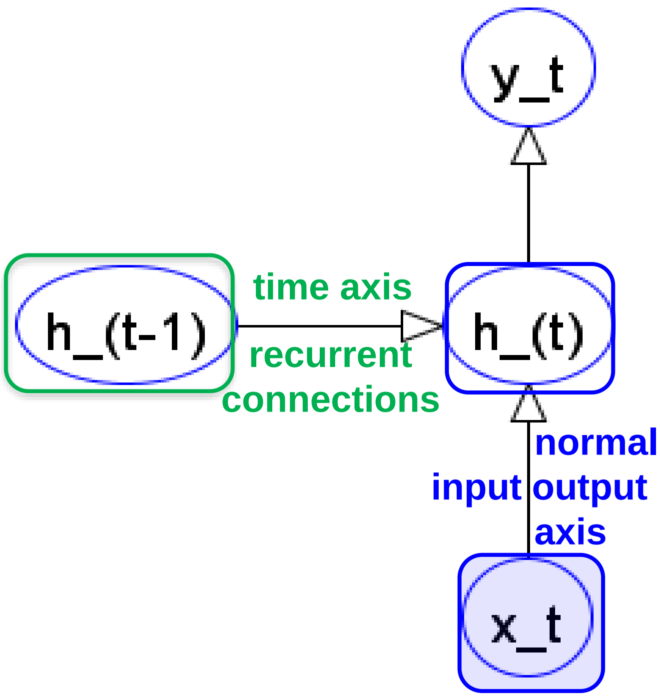


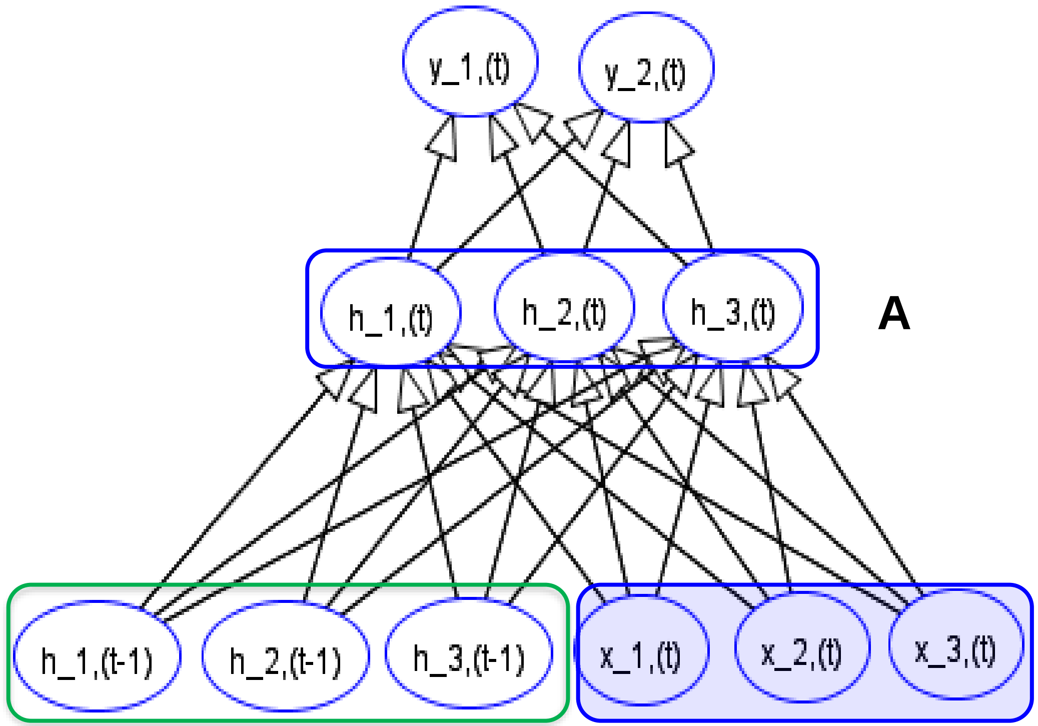


Usually we see only 1-4 hidden layers in an RNN compared to usually 4-100 stacked hidden convolutional blocks in CNNs.

Illustration: http://www.deeplearningbook.org/ and : http://karpathy.github.io/2015/05/21/rnn-effectiveness/

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





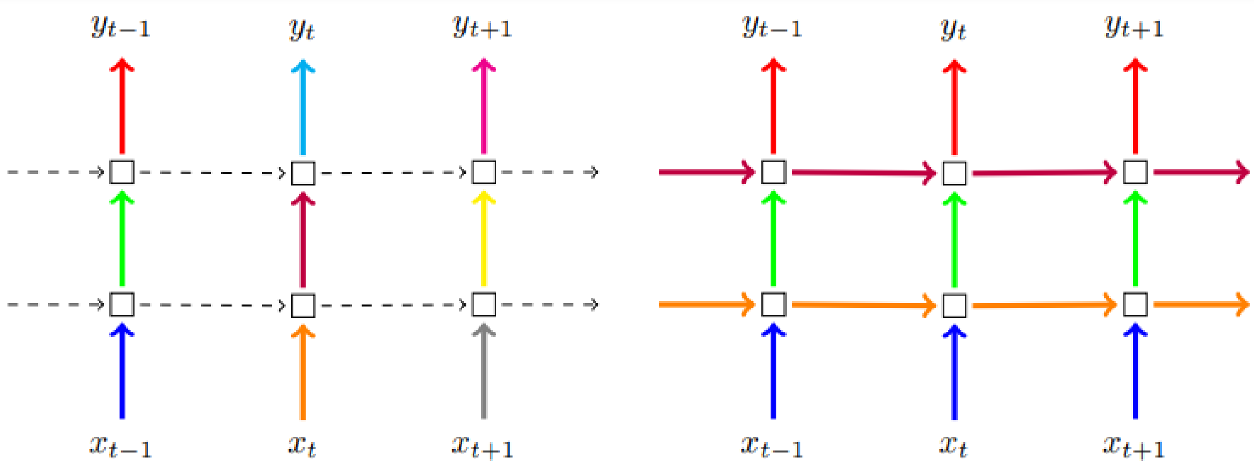
$$\mathbf{A} = f_{\mathbf{W}}(\mathbf{h}_{t-1}, \mathbf{x}_t) = \tanh\left([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W} + \mathbf{b}\right) = \tanh\left(\mathbf{h}_{t-1} \cdot \mathbf{W}_h + \mathbf{x}_t \cdot \mathbf{W}_x + \mathbf{b}\right)$$

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

Dimensions in example: **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

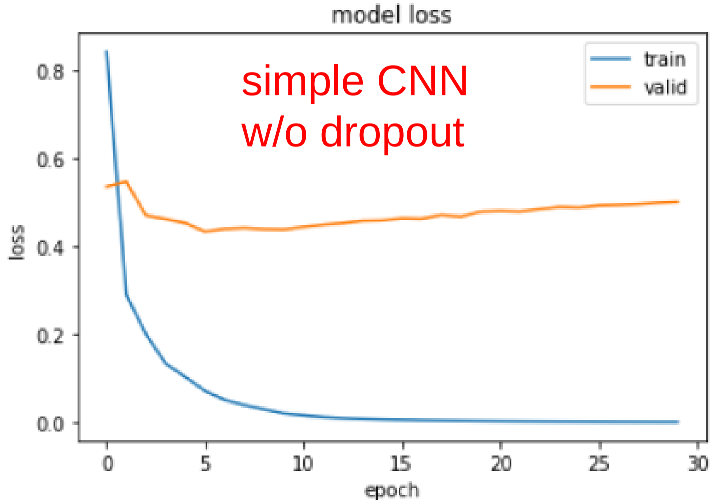
Figure 1: **Depiction of the dropout technique following our Bayesian interpretation (right) compared to the standard technique in the field (left).** Each square represents an RNN unit, with horizontal arrows representing time dependence (recurrent connections). Vertical arrows represent the input and 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.

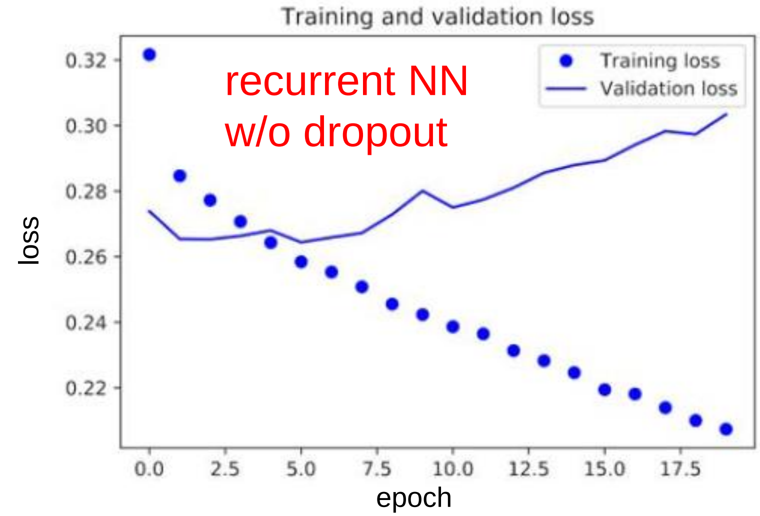
Gal2016

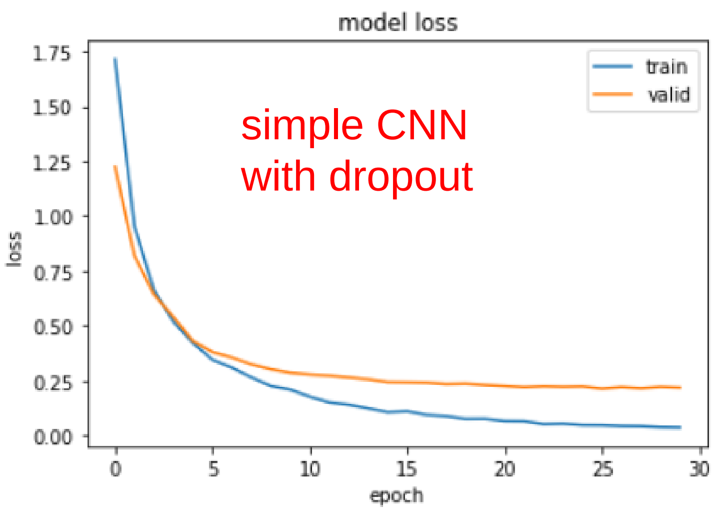
In keras:

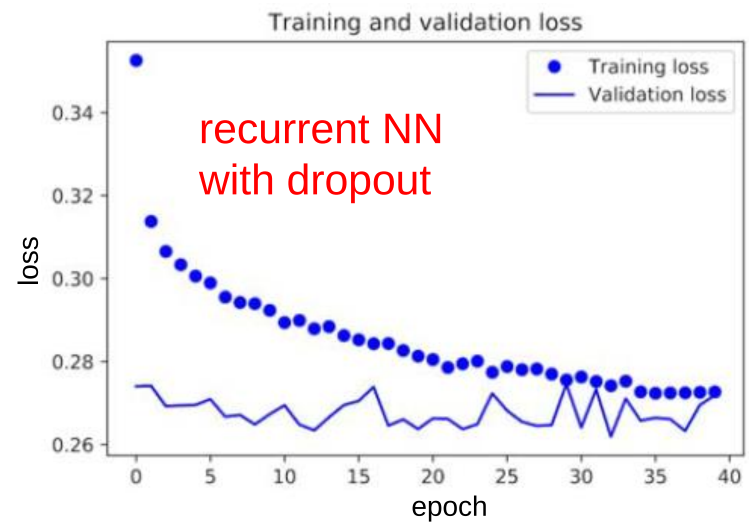
```
model.add(layers.GRU(32, dropout=0.2, recurrent_dropout=0.2, input_shape=(None, …)))
```

Dropout can fight overfitting in CNN and recurrent NN

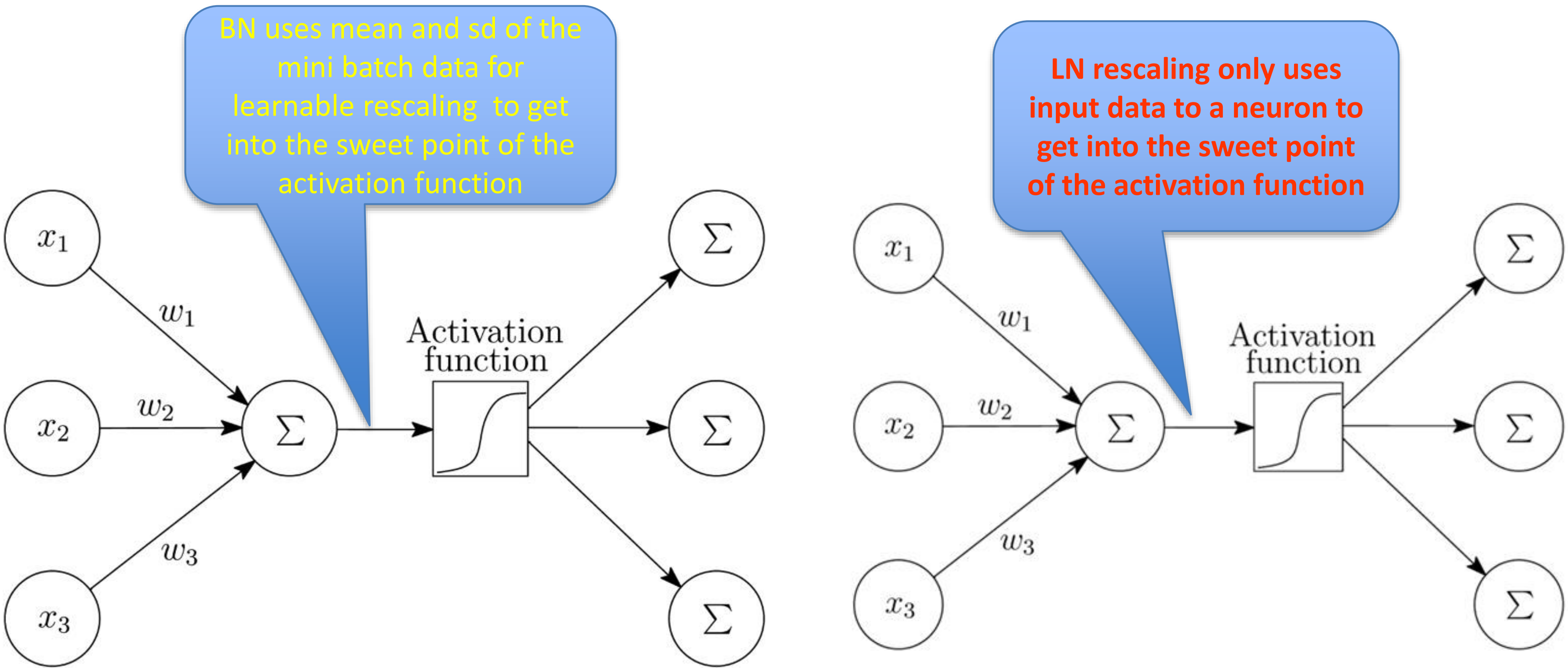








Batchnormalization is crucial to train deep CNNs
Layernormalization is benefial in RNN: LN $\neq$ BN



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 .

For details on layer nomarlization see [Ba.Kiros.Hinton2016]