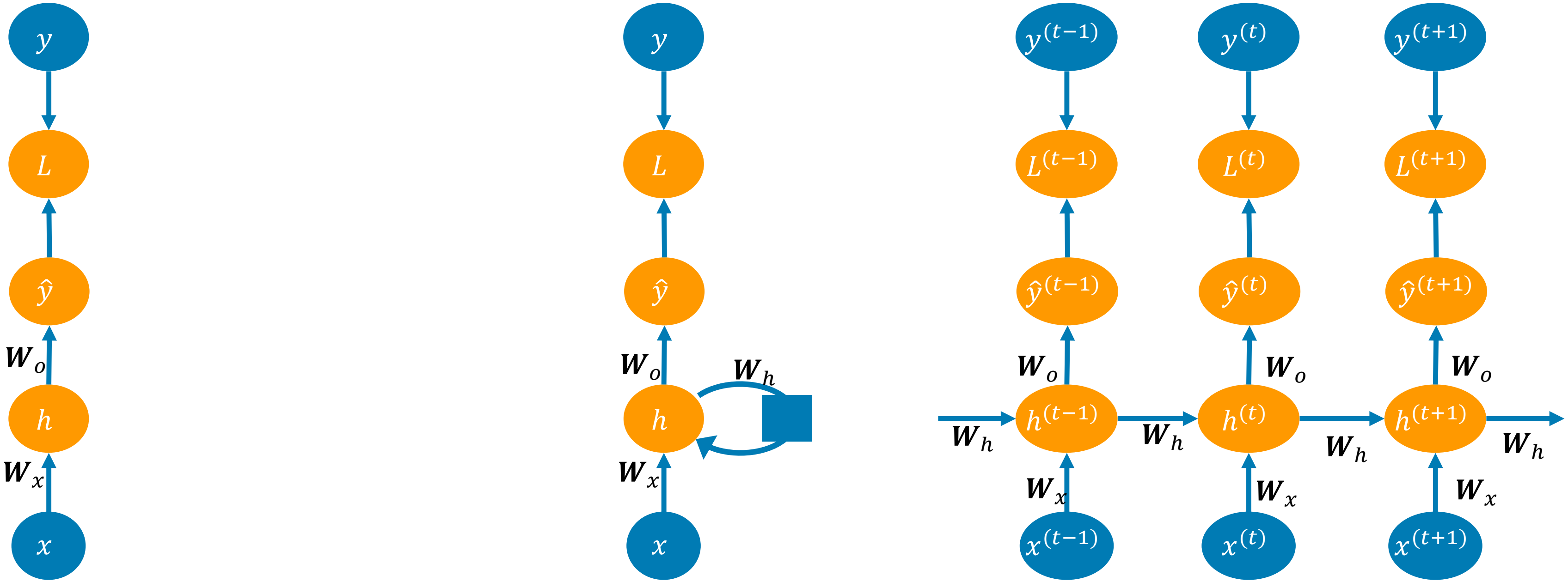
VL Deep Learning for Natural Language Processing

12. Recurrent Neural Networks

Prof. Dr. Ralf Krestel

AG Information Profiling and Retrieval

Recurrent Neural Networks



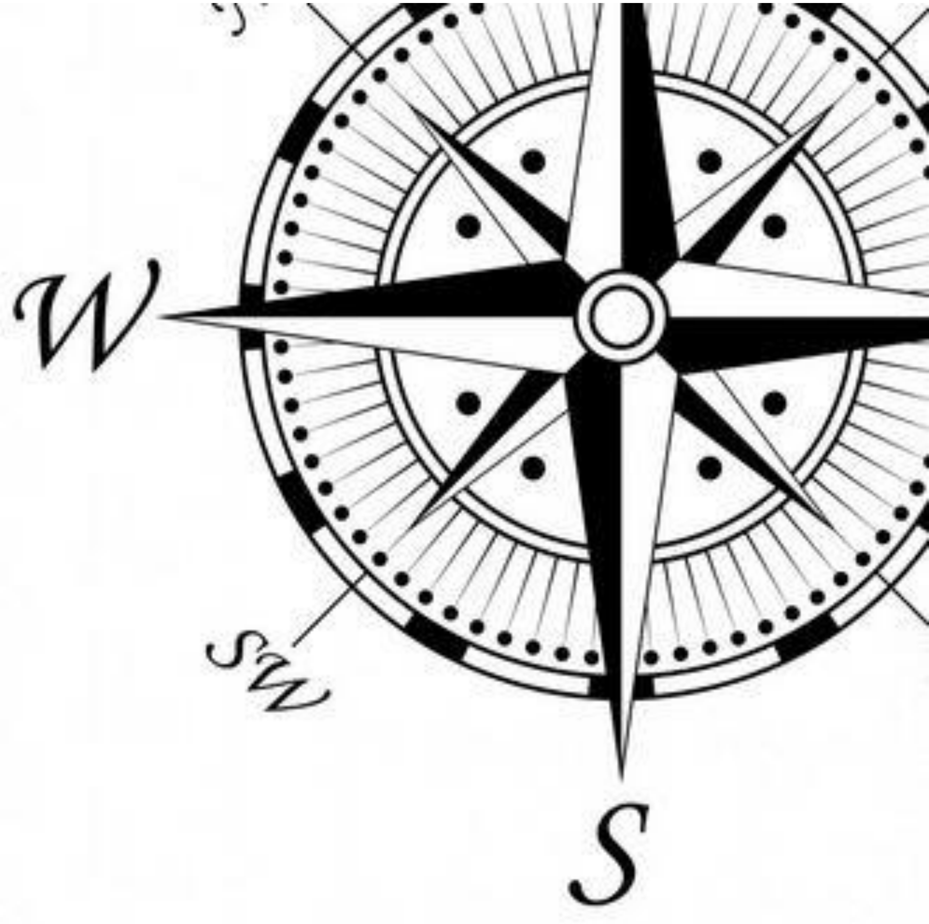
Learning Goals for this Chapter

- Adapt task description so that RNNs can be used to solve the problem
- Understand BPTT
- Explain different kinds of RNNs and how they work
- Implement and evaluate a simple RNN model

- Relevant chapters:
  - P6.2
  - S6 (2021) https://www.youtube.com/watch?v=0LixFSa7yts

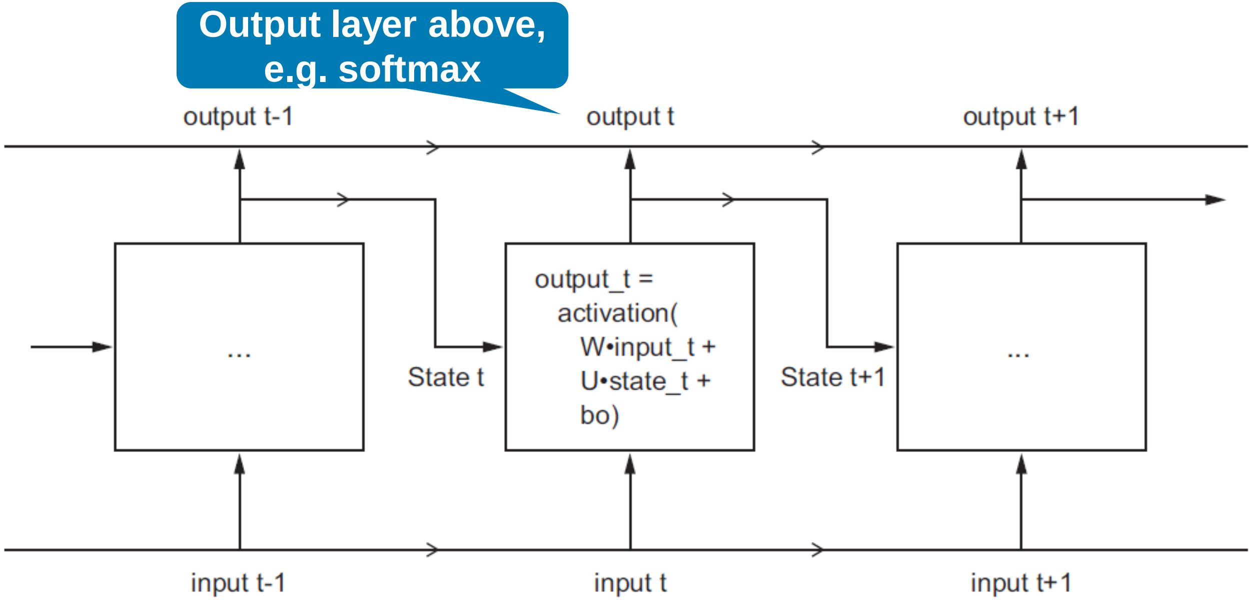
Topics Today

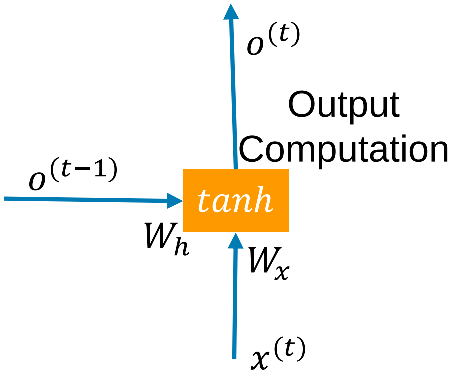
1. **Recurrent Neural Networks (RNN)**
2. Backprop Through Time (BPTT)
3. Different RNN Types
4. A Simple RNN



Rolled-Out Layer

- Output of layer at timestep t
  - $h^{(t)} = \tanh(W_h h^{(t-1)} + W_x x^{(t)} + b_h)$



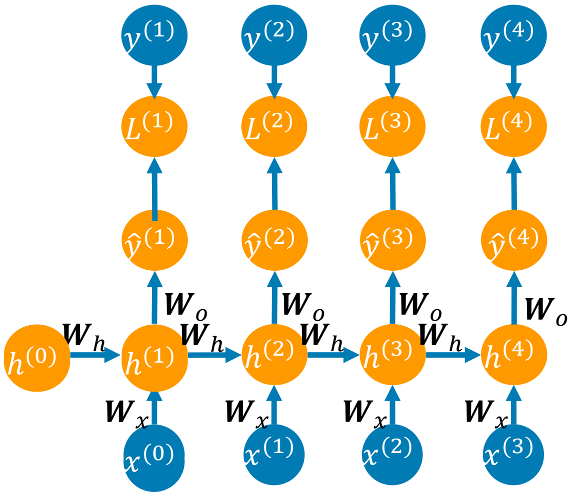


Forward Computation

- $L^{(t)}(\theta) = CE\left(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{y}^{(t)}\right) = -\sum_{j=1}^{|V|} y_j^{(t)} log \hat{y}_j^{(t)}$

- Total loss is averaged:

$$L(\theta) = \frac{1}{T}\sum_{t=1}^{T} L^{(t)}(\theta)$$



Forward Computation Implementation

```
import numpy as np
timesteps = 100
input_features = 32
output_features = 64
inputs = np.random.random((timesteps, input_features))
h_t = np.zeros((output_features,))
W_x = np.random.random((output_features, input_features))
W_h = np.random.random((output_features, output_features))
b_h = np.random.random((output_features,))
successive_outputs = []
for input_t in inputs:
       output_t = np.tanh(np.dot(W_x, input_t) + np.dot(W_h, h_t) + b_h)
       successive_outputs.append(output_t)
       h_t = output_t
final_output_sequence = np.concatenate(successive_outputs, axis=0)
```

Components of an input sequence

Here: Input is random noise

Initial state = 0-vector

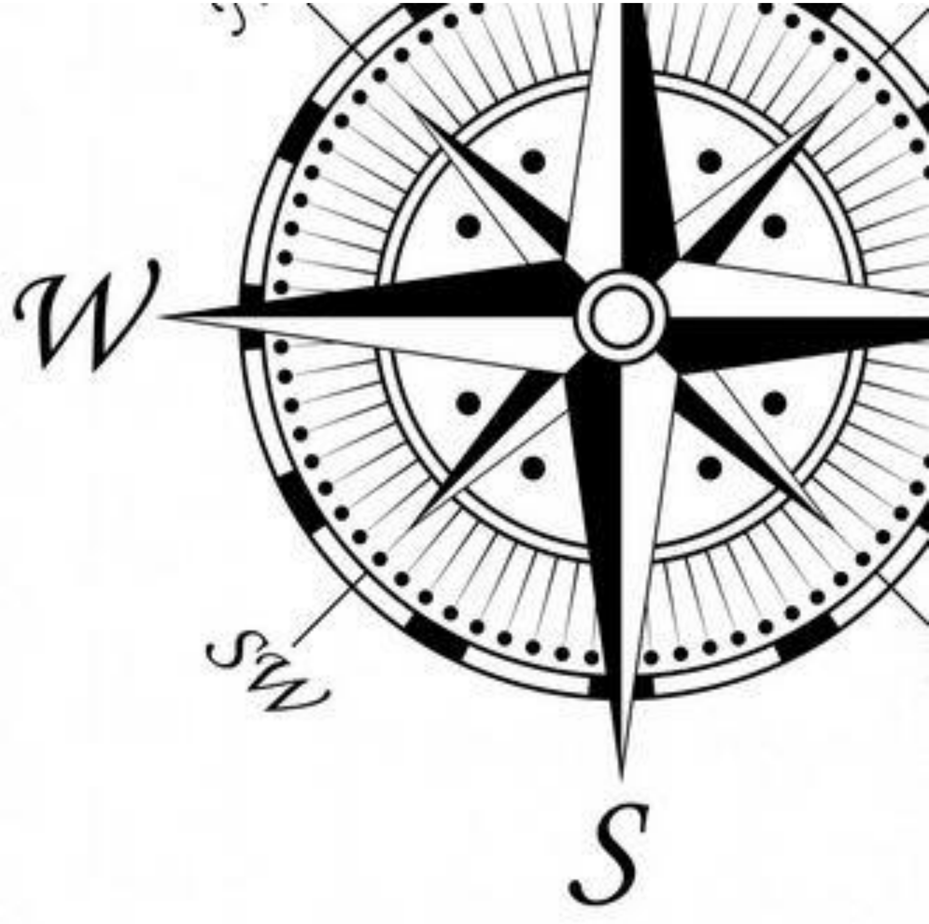
Also the weight matrices are randomly initialized

output_t, sufficient, since it contains information about the whole sequence

Output is a 2d-tensor of shape (timesteps, output_features)

Topics Today

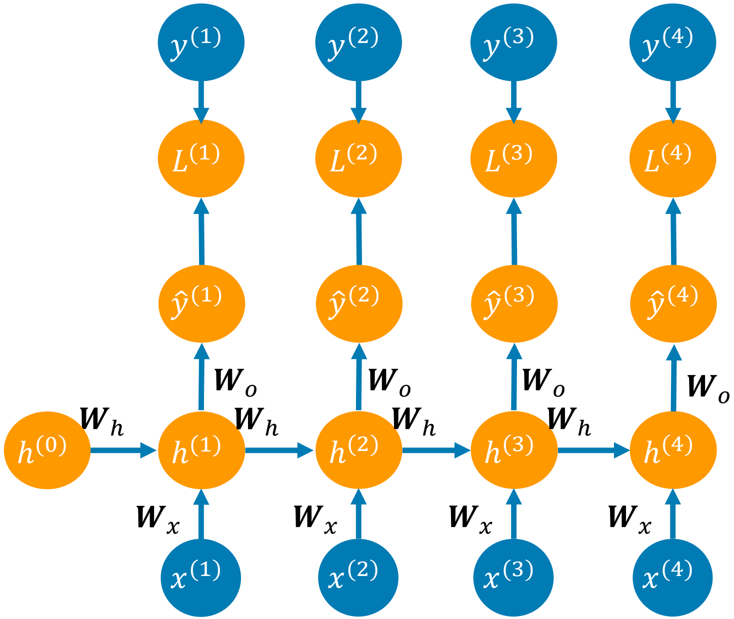
1. Recurrent Neural Networks (RNN)
2. **Backprop Through Time (BPTT)**
3. Different RNN Types
4. A Simple RNN



Backward Computation I

- **Backpropagation through time (BPTT)**
- Given: Multi-variable function $f(x,y)$ and two functions with one variable $x(t)$ and $y(t)$, then this is the multi-variable chain rule

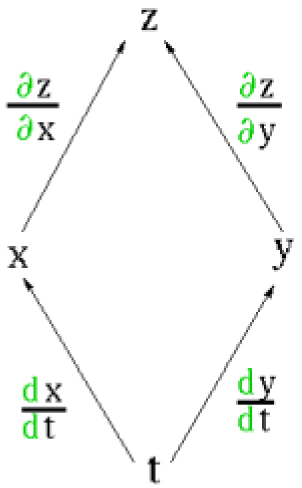
$$\frac{d}{dt} f\big(x(t), y(t)\big) = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$



Multi-Variable Chain Rule: Example

- Let $z = x^2 y - y^2$ where x and y are parametrized as $x = t^2$ and $y = 2t$
- Then

$$
\begin{aligned}
\frac{dz}{dt} &= \frac{\partial z}{\partial x}\frac{dx}{dt} + \frac{\partial z}{\partial y}\frac{dy}{dt} \\
&= (2xy)(2t) + (x^2 - 2y)(2) \\
&= (2t^2 \cdot 2t)(2t) + \big((t^2)^2 - 2(2t)\big)(2) \\
&= 8t^4 + 2t^4 - 8t \\
&= 10t^4 - 8t
\end{aligned}
$$



Backward Computation II

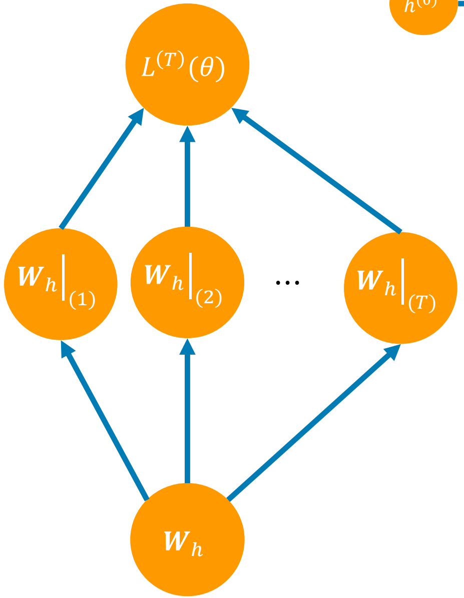
$$\frac{d}{dt}f\big(x(t),y(t)\big) = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

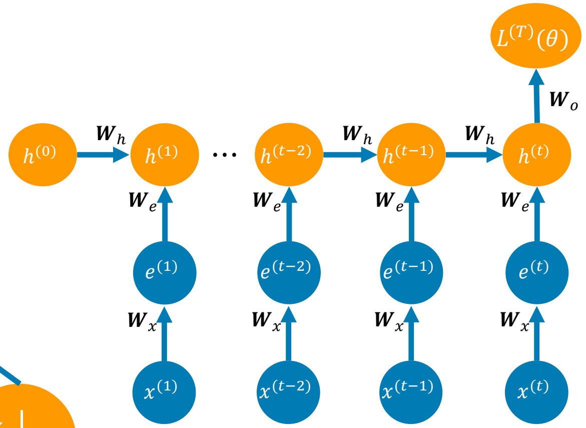
$$L(\theta) = \frac{1}{T}\sum_{t=1}^{T} L^{(t)}(\theta)$$

- Derivation of the loss fuction $L^{(t)}(\theta)$ with respect to repeating $\boldsymbol{W}_h$

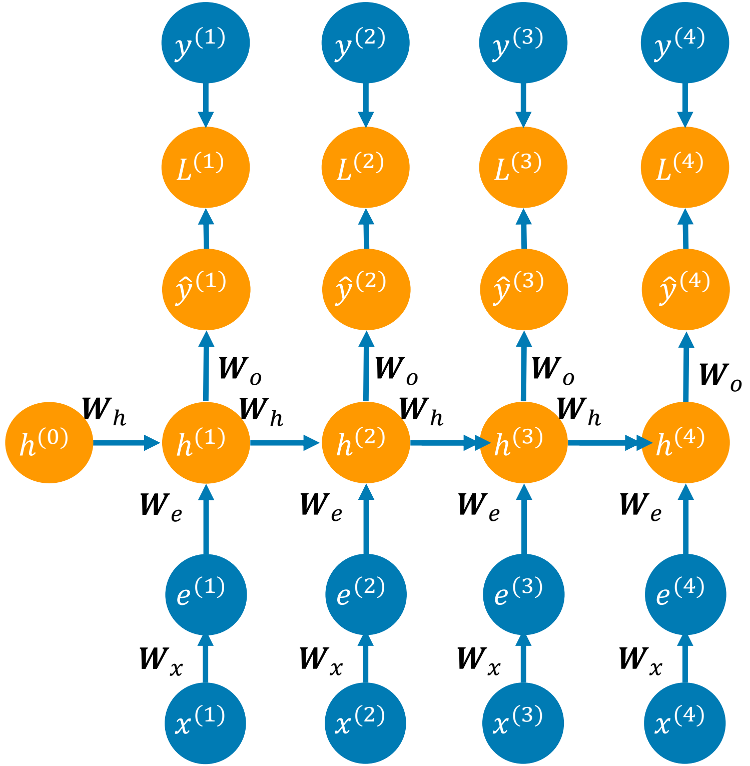
$$\frac{\partial L^{(T)}}{\partial \boldsymbol{W}_h} = \sum_{t=1}^{T}\Big(\frac{\partial L^{(T)}}{\partial \boldsymbol{W}_h}\Big|_{(t)} \cdot \underbrace{\frac{\partial \boldsymbol{W}_h|_{(t)}}{\partial \boldsymbol{W}_h}}_{=1}\Big)$$

$$= \sum_{t=1}^{T}\frac{\partial L^{(T)}}{\partial \boldsymbol{W}_h}\Big|_{(t)}$$





Back Propagation Through Time (BPTT) I



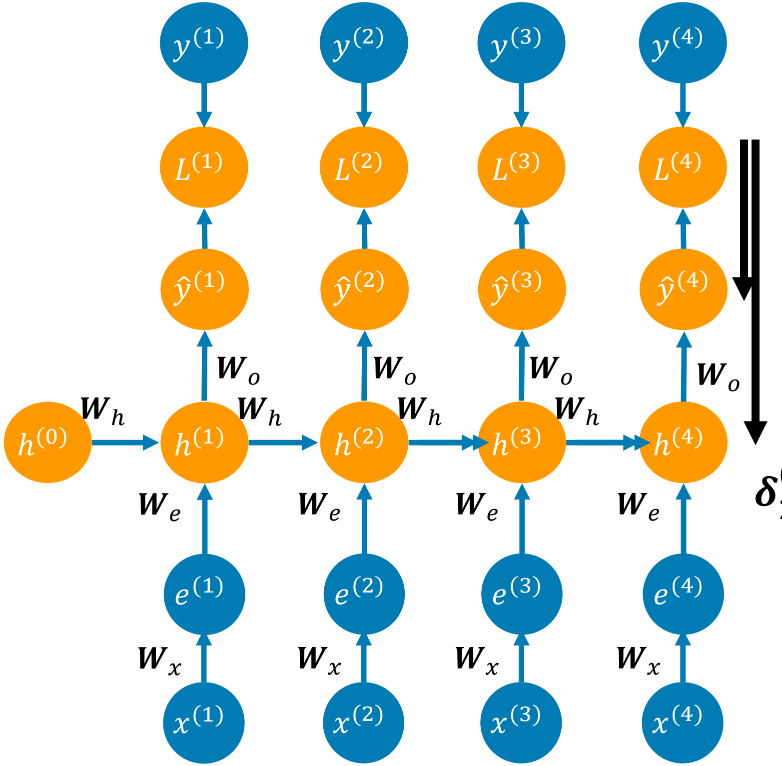
$$L^{(t)}(\theta) = CE\left(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{y}^{(t)}\right) = -\sum_{j=1}^{|V|} y_j^{(t)} log \hat{y}_j^{(t)}$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax(\underbrace{\boldsymbol{h}^{(t)}\boldsymbol{W}_o + \boldsymbol{b}_o}_{\boldsymbol{z}_1^{(t)}})$$

$$\boldsymbol{h}^{(t)} = sigmoid\left(\underbrace{\boldsymbol{h}^{(t-1)}\boldsymbol{W}_h + \boldsymbol{e}^{(t)}\boldsymbol{W}_e + \boldsymbol{b}_h}_{\boldsymbol{z}_2^{(t)}}\right)$$

$$\boldsymbol{e}^{(t)} = \boldsymbol{x}^{(t)}\boldsymbol{W}_x$$

Back Propagation Through Time (BPTT) II



Error signal

$$L^{(t)}(\theta) = CE\left(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{y}^{(t)}\right) = -\sum_{j=1}^{|V|} y_j^{(t)} log \hat{y}_j^{(t)}$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax(\underbrace{\boldsymbol{h}^{(t)}\boldsymbol{W}_o + \boldsymbol{b}_o}_{\boldsymbol{z}_1^{(t)}})$$

$$\boldsymbol{h}^{(t)} = sigmoid\left(\underbrace{\boldsymbol{h}^{(t-1)}\boldsymbol{W}_h + \boldsymbol{e}^{(t)}\boldsymbol{W}_e + \boldsymbol{b}_h}_{\boldsymbol{z}_2^{(t)}}\right)$$

$$\boldsymbol{e}^{(t)} = \boldsymbol{x}^{(t)}\boldsymbol{W}_x$$

$$\boldsymbol{\delta}_1^{(t)} = \frac{\partial L^{(t)}}{\partial \boldsymbol{z}_1^{(t)}} = \hat{\boldsymbol{y}}^{(t)} - \boldsymbol{y}^{(t)}$$

$$\boldsymbol{\delta}_2^{(t)} = \frac{\partial L^{(t)}}{\partial \boldsymbol{z}_2^{(t)}} = \frac{\partial L^{(t)}}{\partial \boldsymbol{z}_1^{(t)}} \frac{\partial \boldsymbol{z}_1^{(t)}}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{z}_2^{(t)}} = \boldsymbol{\delta}_1^{(t)} \boldsymbol{W}_o^T \odot \sigma'(\boldsymbol{z}_2^{(t)})$$

Gradients of an RNN I

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{W}_o} = \frac{\partial L^{(t)}}{\partial \boldsymbol{z}_1^{(t)}} \frac{\partial \boldsymbol{z}_1^{(t)}}{\partial \boldsymbol{W}_o} = \boldsymbol{\delta}_1^{(t)} \otimes \boldsymbol{h}^{(t)}$$

Outer product

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{b}_o} = \boldsymbol{\delta}_1^{(t)}$$

$$\left.\frac{\partial L^{(t)}}{\partial \boldsymbol{W}_e}\right|_{(t)} = \frac{\partial L^{(t)}}{\partial \boldsymbol{z}_2^{(t)}} \frac{\partial \boldsymbol{z}_2^{(t)}}{\partial \boldsymbol{W}_o} = \boldsymbol{\delta}_2^{(t)} \otimes \boldsymbol{e}^{(t)}$$

$$\left.\frac{\partial L^{(t)}}{\partial \boldsymbol{W}_h}\right|_{(t)} = \frac{\partial L^{(t)}}{\partial \boldsymbol{z}_2^{(t)}} \frac{\partial \boldsymbol{z}_2^{(t)}}{\partial \boldsymbol{W}_o} = \boldsymbol{\delta}_2^{(t)} \otimes \boldsymbol{h}^{(t-1)}$$

$$\left.\frac{\partial L^{(t)}}{\partial \boldsymbol{b}_h}\right|_{(t)} = \frac{\partial L^{(t)}}{\partial \boldsymbol{z}_2^{(t)}} \frac{\partial \boldsymbol{z}_2^{(t)}}{\partial \boldsymbol{b}_h} = \boldsymbol{\delta}_2^{(t)}$$

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{W}_{x_{x(t)}}} = \frac{\partial L^{(t)}}{\partial \boldsymbol{z}_2^{(t)}} \frac{\partial \boldsymbol{z}_2^{(t)}}{\partial \boldsymbol{W}_{x_{x(t)}}} = \boldsymbol{\delta}_2^{(t)} \otimes \boldsymbol{W}_e$$

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \frac{\partial L^{(t)}}{\partial \boldsymbol{z}_2^{(t-1)}} \frac{\partial \boldsymbol{z}_2^{(t-1)}}{\partial \boldsymbol{h}^{(t-1)}} = \boldsymbol{\delta}_2^{(t-1)} \otimes \boldsymbol{W}_h$$

$$\boldsymbol{\delta}_1^{(t)} = \frac{\partial L^{(t)}}{\partial \boldsymbol{z}_1^{(t)}} = \hat{\boldsymbol{y}}^{(t)} - \boldsymbol{y}^{(t)}$$

Hadamard product

$$\boldsymbol{\delta}_2^{(t)} = \frac{\partial L^{(t)}}{\partial \boldsymbol{z}_2^{(t)}} = \frac{\partial L^{(t)}}{\partial \boldsymbol{z}_1^{(t)}} \frac{\partial \boldsymbol{z}_1^{(t)}}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{z}_2^{(t)}} = \boldsymbol{\delta}_1^{(t)} \boldsymbol{W}_o^T \odot \sigma'(\boldsymbol{z}_2^{(t)})$$

$$L^{(t)}(\theta) = CE\left(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{y}^{(t)}\right) = -\sum_{j=1}^{|V|} y_j^{(t)} log \hat{y}_j^{(t)}$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax(\underbrace{\boldsymbol{h}^{(t)}\boldsymbol{W}_o + \boldsymbol{b}_o}_{\boldsymbol{z}_1^{(t)}})$$

$$\boldsymbol{h}^{(t)} = sigmoid\left(\underbrace{\boldsymbol{h}^{(t-1)}\boldsymbol{W}_h + \boldsymbol{e}^{(t)}\boldsymbol{W}_e + \boldsymbol{b}_h}_{\boldsymbol{z}_2^{(t)}}\right)$$

$$\boldsymbol{e}^{(t)} = \boldsymbol{x}^{(t)}\boldsymbol{W}_x$$

Gradients of an RNN II

$$\left.\frac{\partial L^{(t)}}{\partial \boldsymbol{W}_e}\right|_{(t-1)} = (\boldsymbol{e}^{(t-1)})^T(\boldsymbol{\delta}_3^{(t-1)} \odot \sigma'(\boldsymbol{z}_2^{(t-1)}))$$

$$\left.\frac{\partial L^{(t)}}{\partial \boldsymbol{W}_h}\right|_{(t-1)} = (\boldsymbol{h}^{(t-2)})^T(\boldsymbol{\delta}_3^{(t-1)} \odot \sigma'(\boldsymbol{z}_2^{(t-1)}))$$

$$\left.\frac{\partial L^{(t)}}{\partial \boldsymbol{b}_h}\right|_{(t-1)} = \boldsymbol{\delta}_3^{(t-1)} \odot \sigma'(\boldsymbol{z}_2^{(t-1)})$$

$$\left.\frac{\partial L^{(t)}}{\partial \boldsymbol{W}_{x_{x(t-1)}}}\right|_{(t-1)} = \boldsymbol{\delta}_3^{(t-1)} \odot \sigma'(\boldsymbol{z}_2^{(t-1)})\boldsymbol{W}_e^T$$

$$\boldsymbol{\delta}_1^{(t)} = \frac{\partial L^{(t)}}{\partial \mathbf{z}_1^{(t)}} = \hat{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}$$

$$\boldsymbol{\delta}_3^{(t-1)} = \boldsymbol{\delta}_1^{(t-1)}\boldsymbol{W}_o^T$$

$$L^{(t)}(\theta) = CE\left(\hat{\boldsymbol{y}}^{(t)}, \boldsymbol{y}^{(t)}\right) = -\sum_{j=1}^{|V|} y_j^{(t)} log \hat{y}_j^{(t)}$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax\underbrace{(\boldsymbol{h}^{(t)}\boldsymbol{W}_o + \boldsymbol{b}_o)}_{\boldsymbol{z}_1^{(t)}}$$

$$\boldsymbol{h}^{(t)} = sigmoid\underbrace{\left(\boldsymbol{h}^{(t-1)}\boldsymbol{W}_h + \boldsymbol{e}^{(t)}\boldsymbol{W}_e + \boldsymbol{b}_h\right)}_{\boldsymbol{z}_2^{(t)}}$$

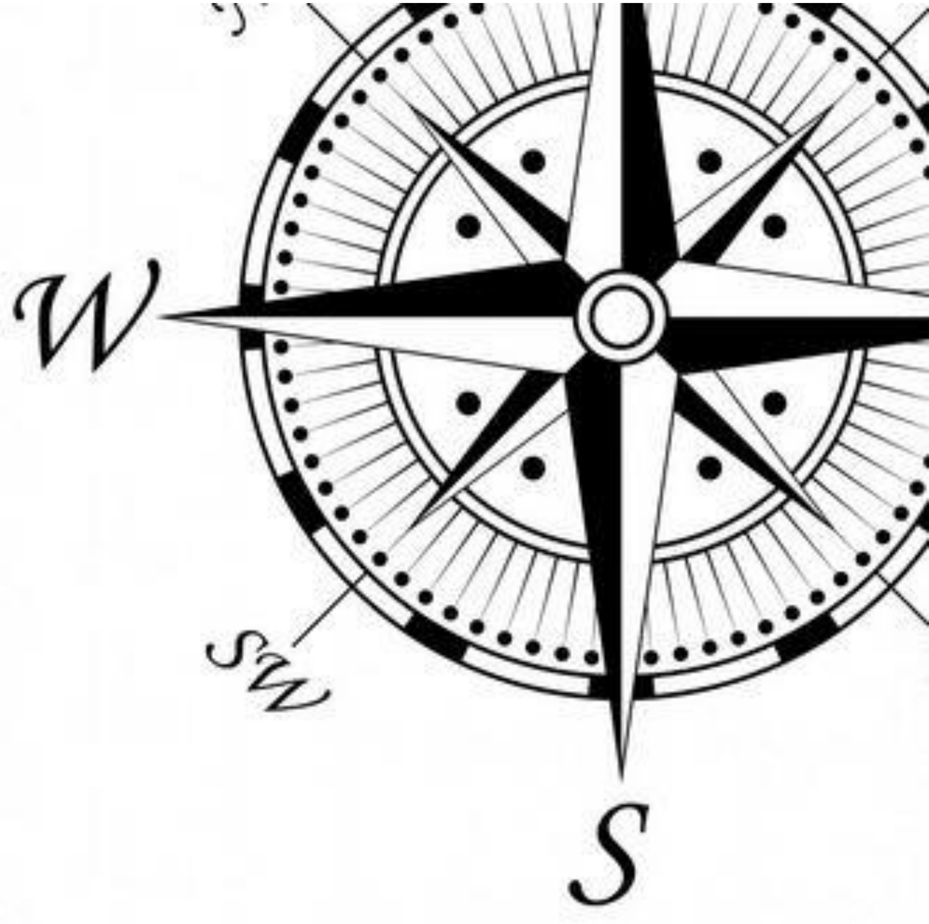
$$\boldsymbol{e}^{(t)} = \boldsymbol{x}^{(t)}\boldsymbol{W}_x$$

Tutorials/Blogposts/Videos…

- https://github.com/go2carter/nn-learn/blob/master/grad-deriv-tex/rnn-grad-deriv.pdf
- https://mmuratarat.github.io/2019-02-07/bptt-of-rnn
- https://arxiv.org/pdf/1610.02583.pdf
- https://www.youtube.com/watch?v=nFTQ7kHQWtc
- https://www.youtube.com/watch?v=q4mVeRLitsU

Topics Today

1. Recurrent Neural Networks (RNN)
2. Backprop Through Time (BPTT)
3. **Different RNN Types**
4. A Simple RNN

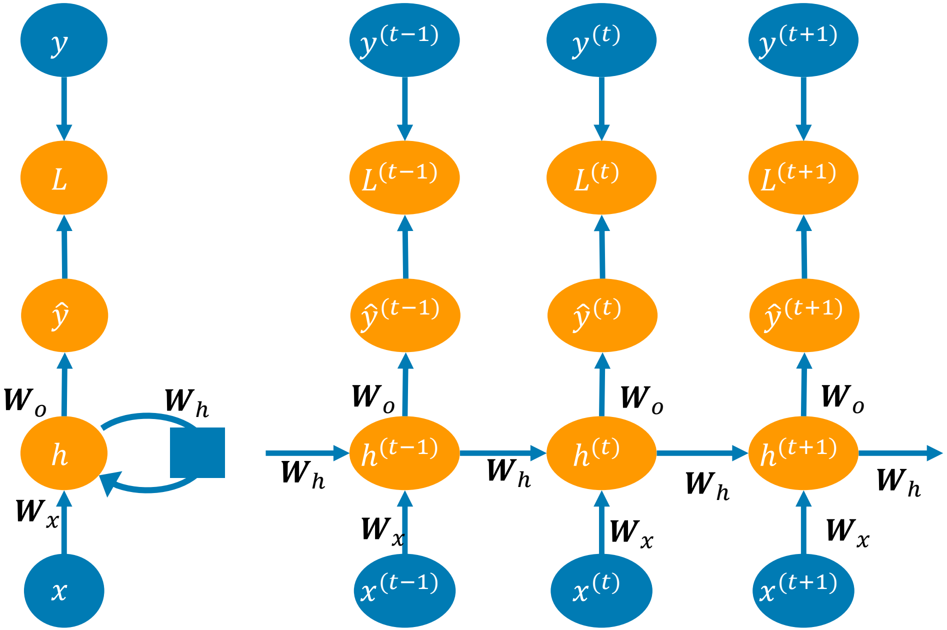


Categorization

- RNNs can be categorized based on their input and output.
  - **Many-to-many**
    - Standard RNN
    - Sequence-to-Sequence
  - **Many-to-one**
    - Summary
  - **One-to-many**
    - Generative models
  - **(One-to-one)**
    - Standard NN

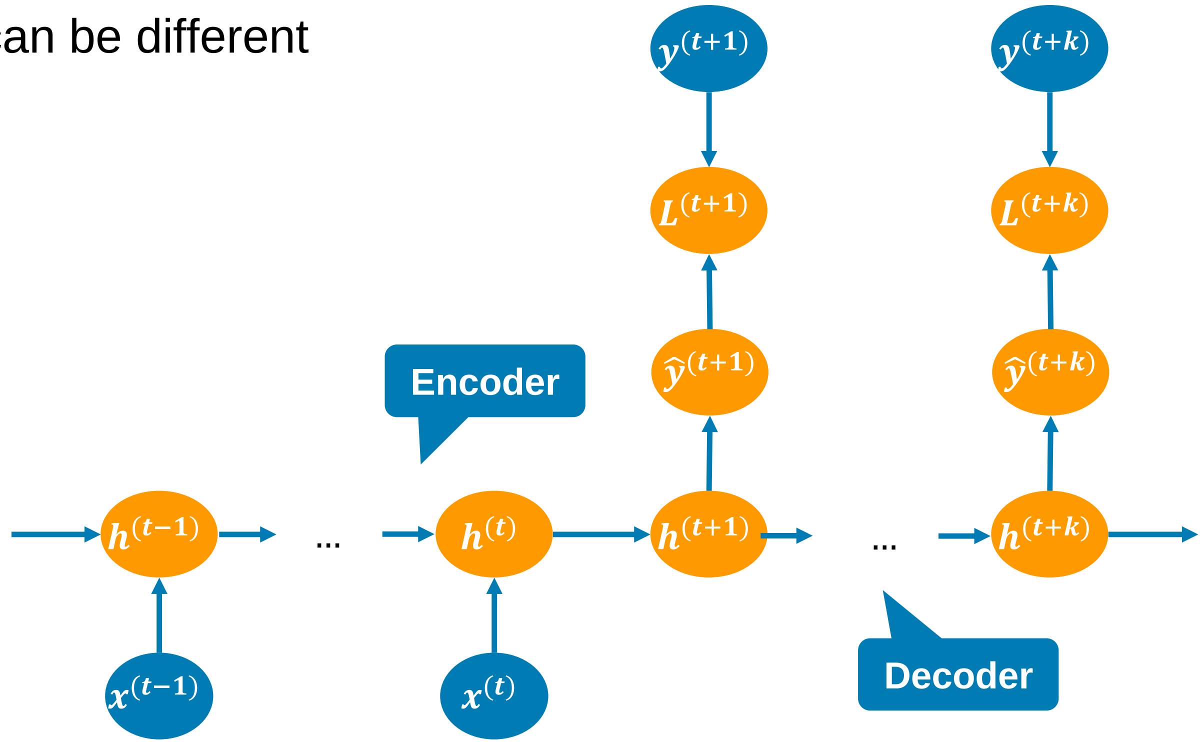
Many-to-Many: Standard RNN

- RNNs are very powerful
  - Turing-complete
- Through the hidden layer, access to all information from the past
- Input length equals output length
  - At each time step t there is an input $x^{(t)}$ and an output $y^{(t)}$



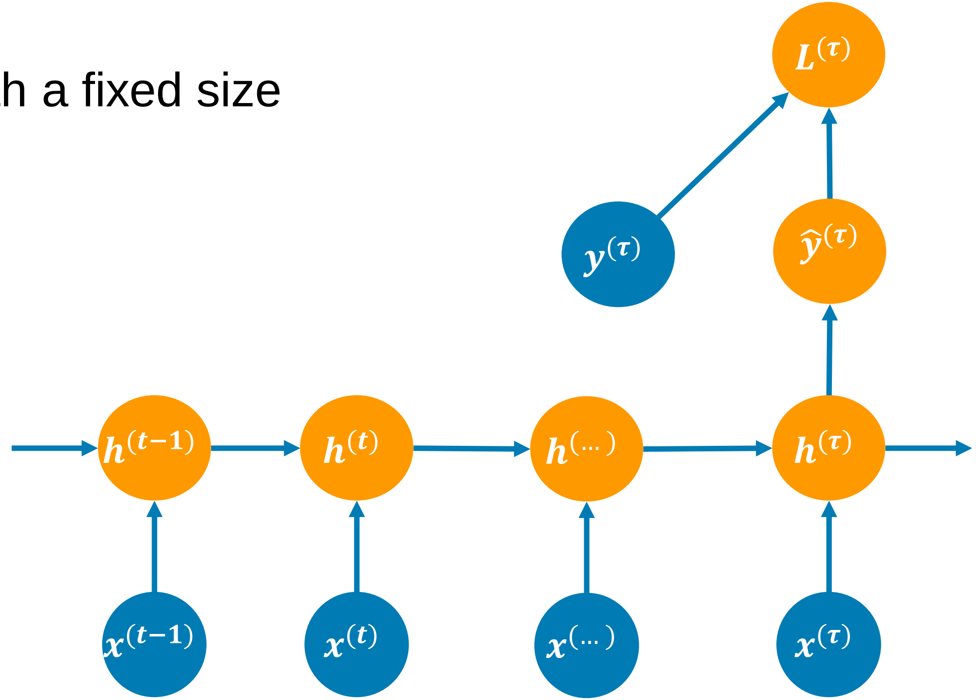
Many-to-Many: Sequence-to-Sequence

- Input and output length can be different
- E.g. machine translation



Many-to-One: Summary

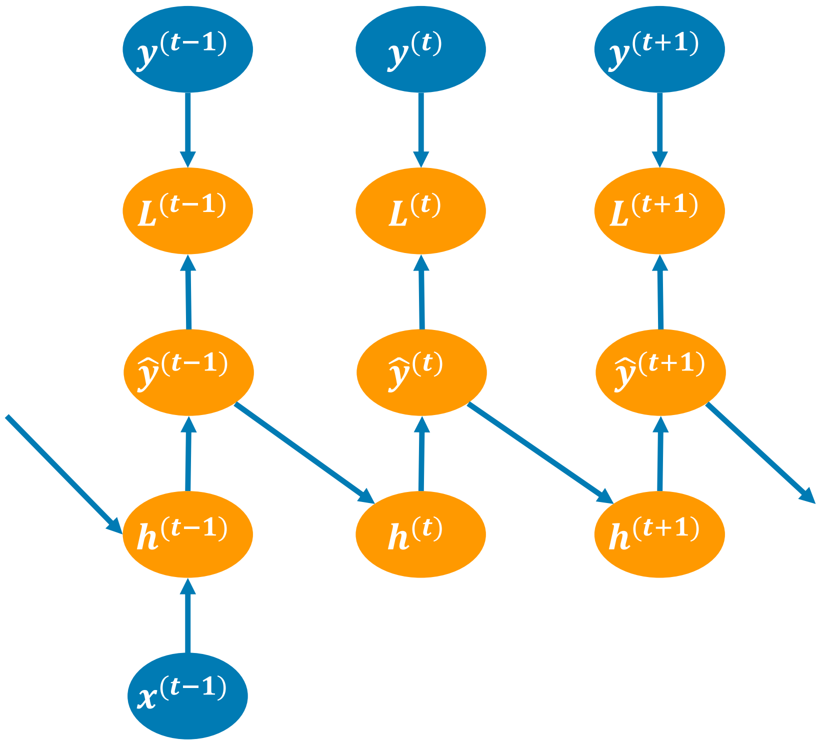
- Can summarize a sequence
  - Representation of a sequence with a fixed size
- E.g. sentiment analyse



- As input for further layers
  - Output will be learnt by backprop
- Here with ist own target value

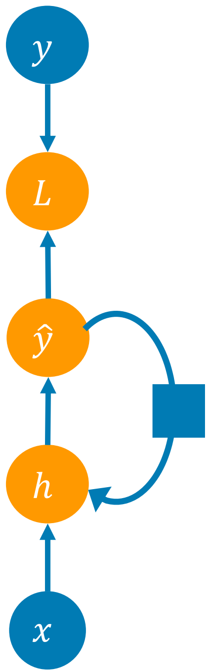
One-to-Many: Generative Models

- E.g. music generation
- Input could be, e.g., an integer, to decide on the style
  - Input could also be empty
    $\boldsymbol{x} = \emptyset$



Many/One-to-Many: RNN Variants

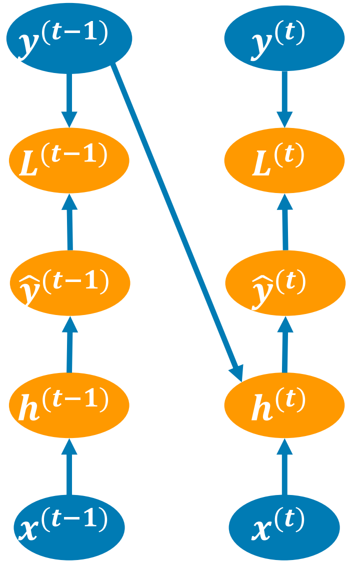
- Less powerful
  - Only output is transmitted
- Only indirect connection of previous hidden layer to current hidden layer
  - Via output layer
    - Typically less dimensions than hidden layer
- Potentially simpler to train
  - Parallelization



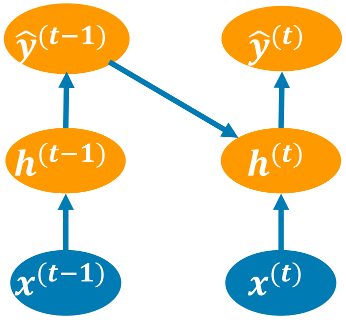
Many/One-to-Many: Teacher Forcing

- Method for RNNs to learn from ground-truth
- When the model is deployed, the ground-truth is approximated by the output

For training



For testing



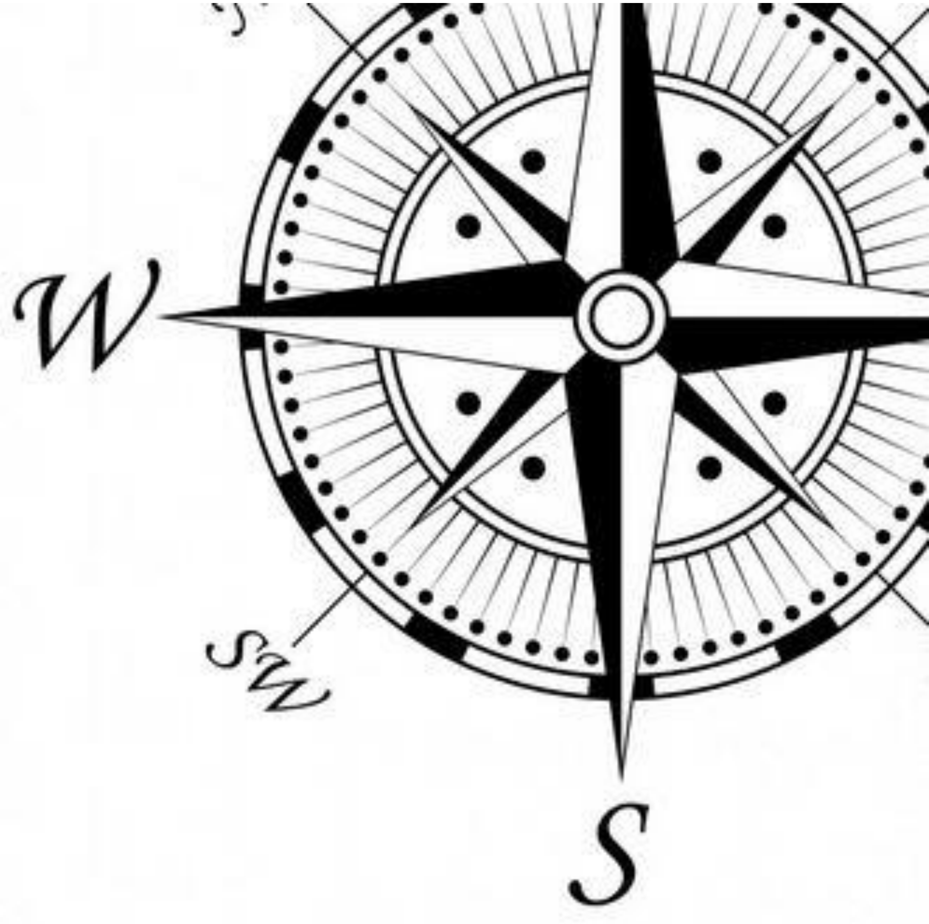
Recurrent Layer

- Think about suitable scenarios for the presented RNNs variants.
- What are advantages and disadvantages of teacher forcing?

Start 5 4 3 2 1 End

Topics Today

1. Recurrent Neural Networks (RNN)
2. Backprop Through Time (BPTT)
3. Different RNN Types
4. **A Simple RNN**



Simple RNN in Keras

```
from keras.models import Sequential
from keras.layers import Embedding, SimpleRNN
model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32, return_sequences=True))
model.summary()
```

```
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
embedding_22 (Embedding)     (None, None, 32)          320000
________________________________________________________________
simplernn_10 (SimpleRNN)     (None, 32)                2080
================================================================
Total params: 322,080
Trainable params: 322,080
Non-trainable params: 0
```

Stacked Simple RNN in Keras

```
model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_24 (Embedding)     (None, None, 32)          320000
_________________________________________________________________
simplernn_12 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simplernn_13 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simplernn_14 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simplernn_15 (SimpleRNN)     (None, 32)                2080
=================================================================
Total params: 328,320
Trainable params: 328,320
Non-trainable params: 0
```

IMDB Example: Preprocessing

```
from keras.datasets import imdb
from keras.preprocessing import sequence
max_features = 10000
maxlen = 500
batch_size = 32
print('Loading data...')
(input_train, y_train), (input_test, y_test) =
imdb.load_data(num_words=max_features)
print(len(input_train), 'train sequences')
print(len(input_test), 'test sequences')
print('Pad sequences (samples x time)')
input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
print('input_train shape:', input_train.shape)
print('input_test shape:', input_test.shape)
```

Batch processing, i.e. multiple sequences simultaneously

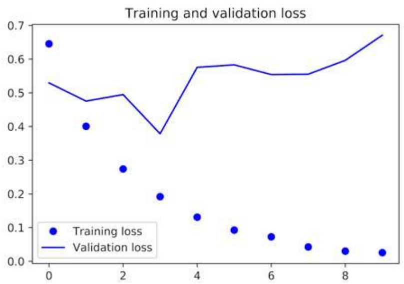
IMDB Example: Model

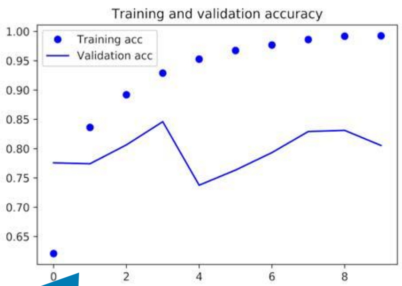
```
from keras.layers import Dense
model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])
history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=batch_size,
                    validation_split=0.2)
```

IMDB Example: Validation

```
import matplotlib.pyplot as plt
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```





Test accuracy without RNN: 88%;
With RNN only 85%;

Exercise

- Implement the previous example
  - Stack RNN-layers
  - Play around with other hyperparameters

Start 10 9 8 7 6 5 4 3 2 1 End

Learning Goals for this Chapter

- Adapt task description so that RNNs can be used to solve the problem
- Understand BPTT
- Explain different kinds of RNNs and how they work
- Implement and evaluate a simple RNN model

- Relevant chapters:
  - P6.2
  - S6 (2021) https://www.youtube.com/watch?v=0LixFSa7yts