Data Science and Advanced Programming — Lecture 10

Modeling Sequence Data with RNNs, LSTMs

Simon Scheidegger
Department of Economics, University of Lausanne, Switzerland

November 17th, 2025 | 12:30 - 16:00 | Internef 263

Road-Map

- This lecture:
  - Deep Learning cont'd
  - More advanced topics:
  - Recurrent neural networks and beyond.
- Throughout lectures - hands-on:
  - Basics on Tensorflow & Keras
  - Examples related to the day's topics in Tensorflow

Keras & Tensorflow Basics

- tensorflow.org
- Keras API: https://www.tensorflow.org/guide/keras/sequential_model
- Fun data sets to play with: https://www.kaggle.com/datasets
- Some "clean" data to play with: https://archive.ics.uci.edu/ml/index.php
- Help for debugging — Tensorboard: https://www.tensorflow.org/tensorboard

A Gentle First Example

- Let's look at the notebook: demo/03_Gentle_DNN.ipynb.
- This Notebook contains all the basic functionality from a theoretical point of view.
- 2 simple examples, one regression, and one classification.

Action Required

Look at the test functions on the right-hand side*. Pick three of those test functions (from Genz 1987).

- Approximate a 2-dimensional function stated below with Neural Nets based 10,50,100,500 points randomly sampled from $[0,1]^2$. Compute the average and maximum error.
- The errors should be computed by generating 1,000 uniformly distributed random test points from within the computational domain.
- Plot the maximum and average error as a function of the number of sample points.
- Repeat the same for 5-dimensional and 10-dimensional functions. Is there anything particular you observe?

*Choose the parameters w and c in meaningful ways.

$$\text{oscillatory: } f_1(x) = \cos\left(2\pi w_1 + \sum_{i=1}^{d} c_i x_i\right),$$

$$\text{product peak: } f_2(x) = \prod_{i=1}^{d} (c_i^{-2} + (x_i - w_i)^2)^{-1},$$

$$\text{corner peak: } f_3(x) = \left(1 + \sum_{i=1}^{d} c_i x_i\right)^{-(d+1)},$$

$$\text{Gaussian: } f_4(x) = \exp\left(-\sum_{i=1}^{d} c_i^2 \cdot (x_i - w_i)^2\right),$$

$$\text{continuous: } f_5(x) = \exp\left(-\sum_{i=1}^{d} c_i \cdot |x_i - w_i|\right),$$

$$\text{discontinuous: } f_6(x) = \begin{cases} 0, & \text{if } x_1 > w_1 \text{ or } x_2 > w_2, \\ \exp\left(\sum_{i=1}^{d} c_i x_i\right), & \text{otherwise.} \end{cases}$$

Varying test functions can be obtained by altering the parameters $c = (c_1, \ldots, c_n)$ and $w = (w_1, \ldots, w_n)$. We chose these parameters randomly from $[0, 1]$. Similarly to Barthelmann et al. [2000], we normalized the c_i such that $\sum_{i=1}^{d} c_i = b_j$, with b_j depending on d, f_j according to

j	1	2	3	4	5	6
b_j	1.5 d		1.85	7.03	20.4	4.3

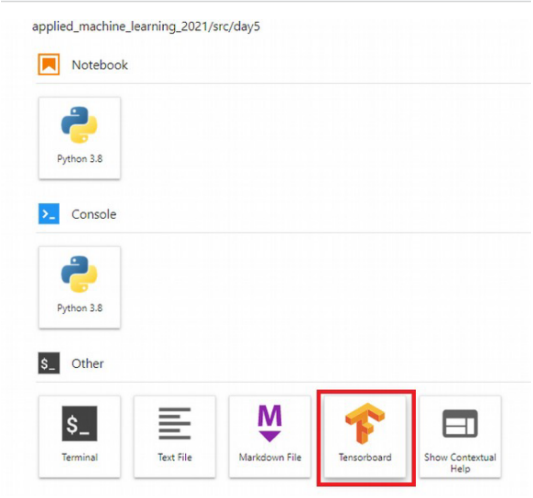
Furthermore, we normalized the w_i such that $\sum_{i=1}^{d} w_i = 1$.

Action Required (II)

- Play with the architecture:
  - Number of hidden layers.
  - activation functions.
  - choice of the stochastic gradient descent algorithm.
  - Monitor the performance with respect to the architecture.

A Semi-Comprehensive TF Tour

- `demo/04_TF_tour.ipynb`
- 5 examples (incl. Kaggle data set from Lending Club)
- Tensorboard
  - On Nuvolos: In-cell it won't work in JupyterLab.
  - Once you've run all the cells, go to the launcher, click TensorBoard, and you should be good to go.
  - Right after this, a new tensorboard tab should show up that contains the expected.



Action Required

- Focus on the example with the Kaggle data set.
  - Play with the architecture.
  - Number of hidden layers.
  - activation functions.
  - choice of the stochastic gradient descent algorithm.
  - Monitor the performance with respect to the architecture.

  Try to use Tensorboard.

Some Personal Take-Away

- Swish activation is the "best" if you need smooth and deep.
- Multiple of 2 for network (training speed).
- Smaller learning rate with deeper networks.
- Batch normalization for speed.
- Glorot initialization.
- Custom layers for custom models (https://www.tensorflow.org/tutorials/customization/custom_layers).

A Break — Mountains

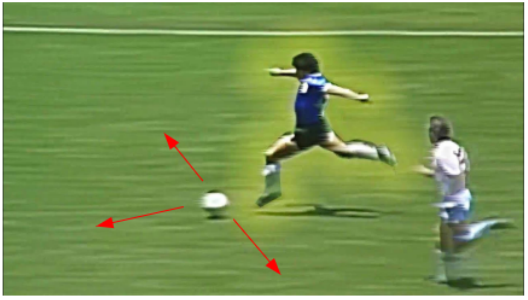
Beyond Vanilla DNN

- Not all applications are plain vanilla deep neural nets.
- There exist situations where more intricate architectures are needed.
- Examples:
  - Time-series comparisons, such as estimating how closely related two documents or two stock tickers are.
  - Sequence-to-sequence learning, such as decoding an English sentence into French.
  - Sentiment analysis, such as classifying the sentiment of tweets or movie reviews as positive or negative.
  - Time-series forecasting, such as predicting the future weather at a certain location, given recent weather data.

Example

Given a picture of a ball, can we predict where it will go?

Example



Given a picture of a ball, can we predict where it will go?

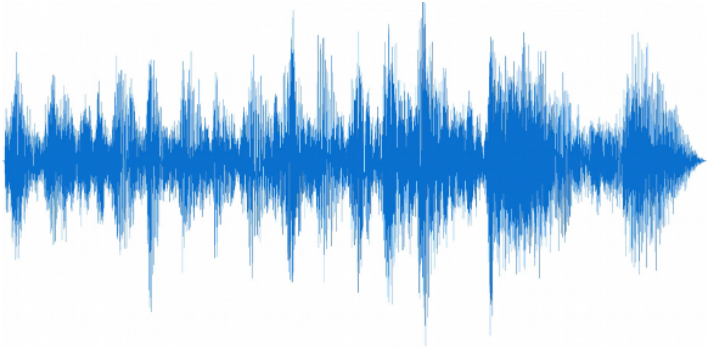
A Sequence Modeling Problem: Predict the Next Word

"Today, we are having a class on deep _____"

A Sequence Modeling Problem: Predict the Next Word

"Today, we are having a class on deep learning"

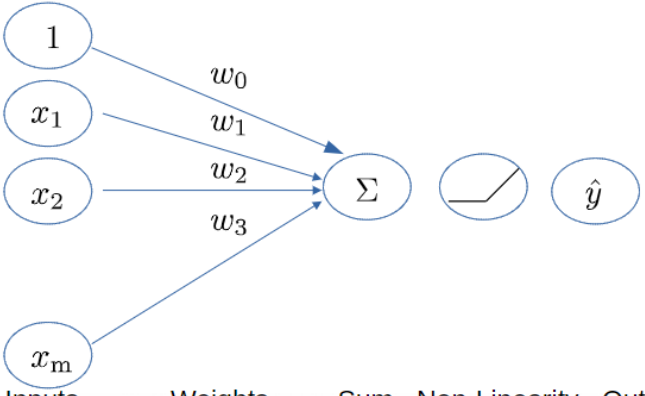
A Sequence Modeling Problem: Audio



A Sequence Modeling Problem: Beethoven

https://www.youtube.com/watch?v=Rvj30blscqw

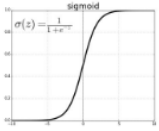
The Perceptron Revisited



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i\right)$$

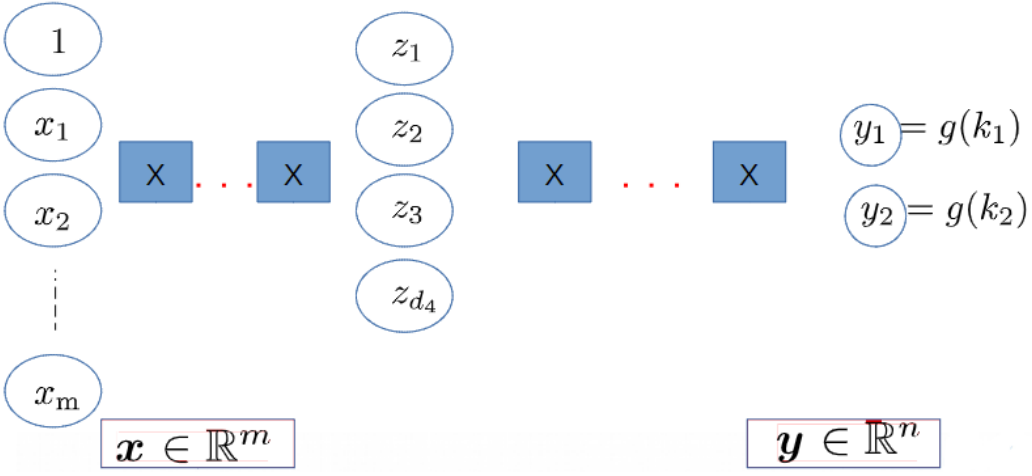
Activation Functions
e.g. sigmoid function

$$\mathrm{g(z)}{=}\sigma(z) = \frac{1}{1+e^{-z}}$$



$\Longrightarrow$ Bias term allows you to shift your activation function to the left or the right

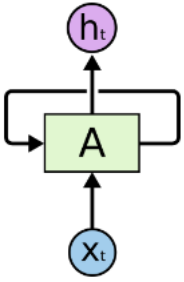
Feed-Forward Nets Revisited



Recurrent Neural Nets

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

- To model sequences, we need to
  - Handle variable-length sequences.
  - Track long-term dependencies.
  - Maintain information about the order.
  - Share parameters across the sequence.



- Recurrent Neural Networks (RNN) are an approach to sequence modeling problems (Rumelhart et al. (1986)).
- More specifically, given an observation sequence $x = \{x_1, x_2, \ldots, x_T\}$ and its corresponding label $y = \{y_1, y_2, \ldots, y_T\}$, we want to learn a map $f : x \rightarrow y$.

RNN

- RNNs are a family of neural networks for processing sequential data.
- A RNN is a neural network that is specialized for processing a sequence of values $x^{(1)}, \ldots, x^{(\tau)}$.
- Unfold the computational graph of a dynamical system:

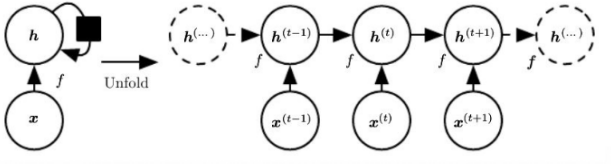


Fig. from Goodfellow et al. (2016)

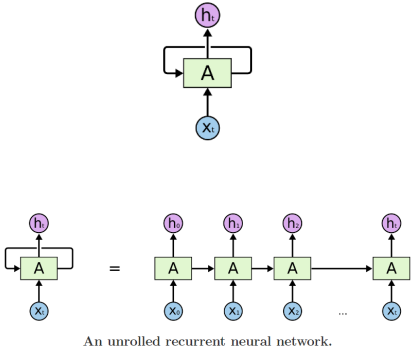
Preview on RNN

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

- Recurrent layers use their own output as input.
  - In Figure: A is a recurrent cell
- Introduce history or time dependency in NNs.
- The only way to efficiently train them is to unroll them.

$$\boldsymbol{h}^{(t)} = f\left(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}\right)$$

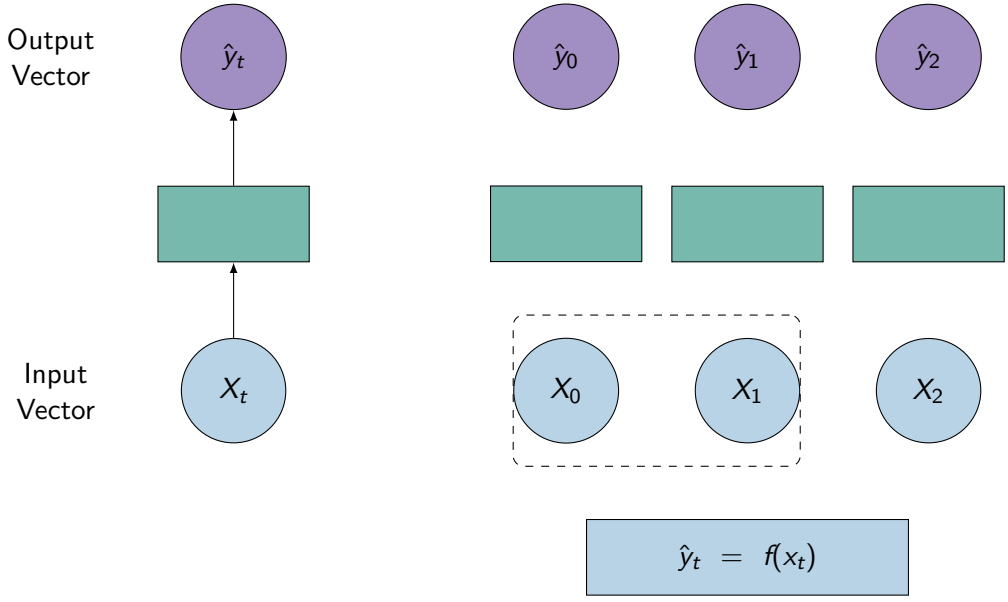
Cell state | Function (parameterized) | Old state | Input vector at time t



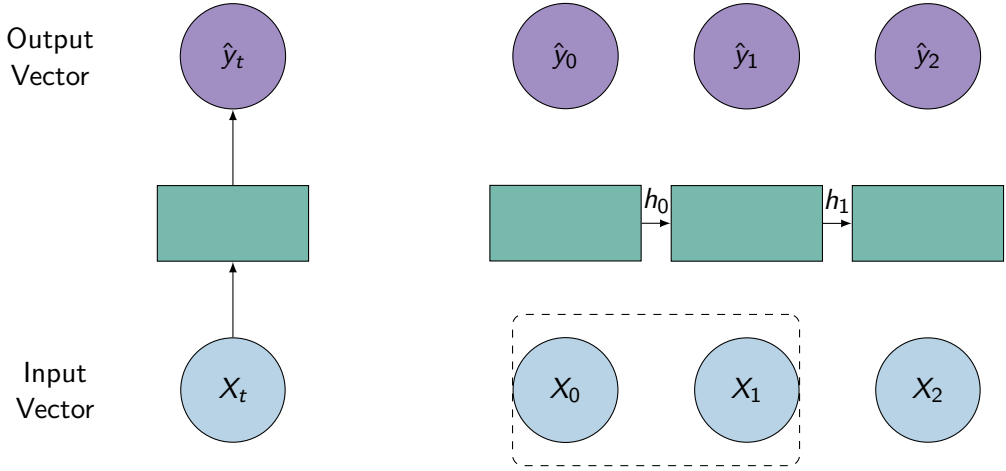
An unrolled recurrent neural network.

Feed-Forward Nets Revisited

Handling Individual Time Steps

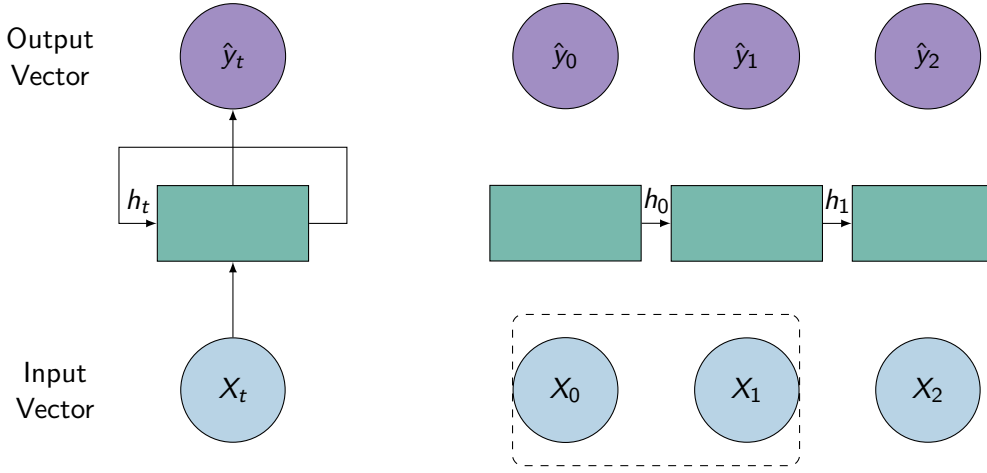


Neurons With Recurrence



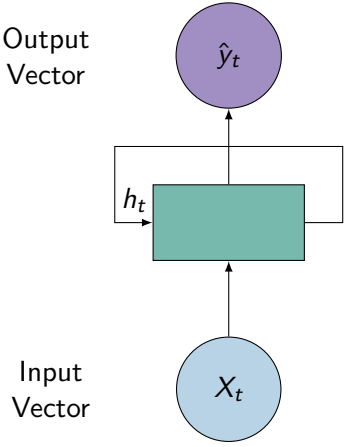
$$\underbrace{\hat{y}_t}_{\text{output}} = f\left(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{past memory}}\right)$$

Neurons With Recurrence



$$\underbrace{\hat{y}_t}_{\text{output}} = f\underbrace{(x_t}_{\text{input}}, \underbrace{h_{t-1})}_{\text{past memory}}$$

Recurrent Neural Networks



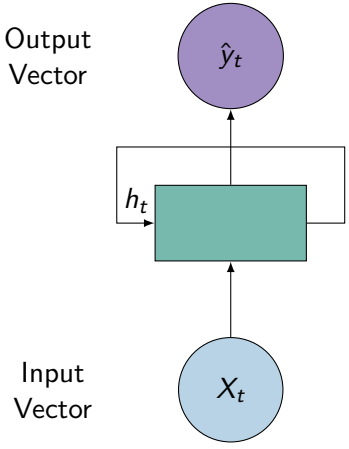
- Apply a recurrence relation at every time step to process a sequence:

$$\underbrace{h_t}_{\text{cell state}} = \underbrace{f_W}_{\substack{\text{function with} \\ \text{weights W}}} \left(\underbrace{x_t}_{\text{input}}, \underbrace{h_{t-1}}_{\text{old state}} \right)$$

- cell state function input old state with weights W
- Note: the same function and set of parameters are used at every time step.

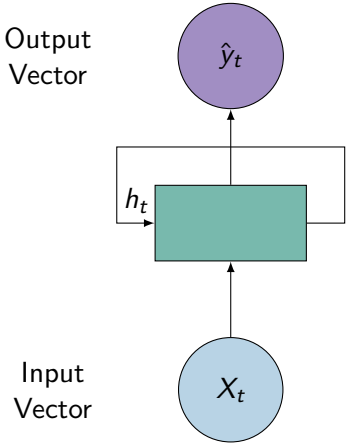
RNNs have a **cell state** that is updated **at each time step** as a sequence is proceeded.

RNN Intuition



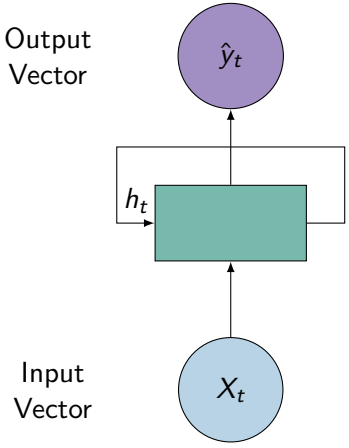

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```

RNN Intuition



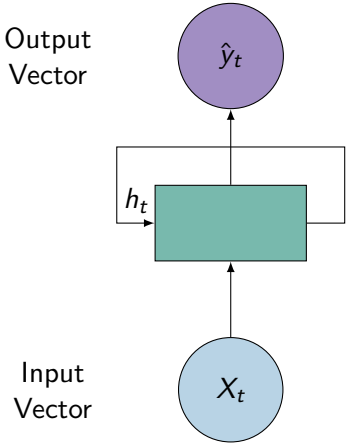

```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```

RNN Intuition




```
my_rnn = RNN()
hidden_state = [0, 0, 0, 0]

sentence = ["I", "love", "recurrent", "neural"]

for word in sentence:
    prediction, hidden_state = my_rnn(word, hidden_state)

next_word_prediction = prediction
# >>> "networks!"
```

RNN State Update And Output



- Output vector

$$\hat{\boldsymbol{y}}_t = W_{hy}^T h_t$$

- Update hidden state

$$\mathrm{h}_t = \tanh\left(\boldsymbol{W}_{\boldsymbol{hh}}^{\boldsymbol{T}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}}^{\boldsymbol{T}} x_t\right)$$

- Input vector

$$x_t$$

RNN In One Slide

- RNN models a dynamic system, where the hidden (cell) state h_t is not only dependent on the current observation x, but also relies on the previous hidden state h.
- More specifically, we can represent h_t as $\mathbf{h}_t = \mathbf{f}(\mathbf{h}_{t\text{-}1}, \mathbf{x}_t)$ (Eq. 1) where f is a nonlinear (time-invariant) mapping.
- Thus, h_t contains information about the whole sequence, which can be inferred from the recursive definition in $\mathrm{Eq.1}$.
- In other words, RNN can use the hidden variables as a memory to capture long term information from Figare 1: It is a RNN example corresponding extended RNN moil a sequence.
- Prediction at the time step t : z_t

$$\mathbf{h}_t = \tanh\left(W_{hh}\mathbf{h}_{t-1} + W_{xh}\mathbf{x}_t + \mathbf{b}_\mathbf{h}\right)$$

$$z_t = \mathrm{softmax}\left(W_{hz}\mathbf{h}_t + \mathbf{b}_z\right)$$

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\sum y_t \log z_t$$

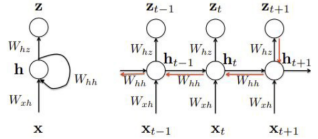
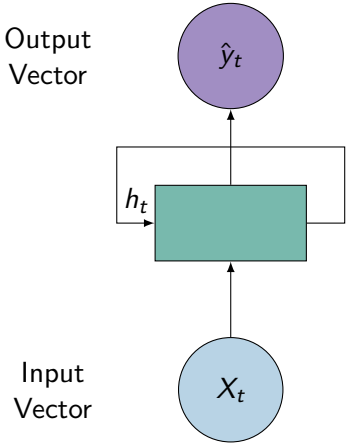


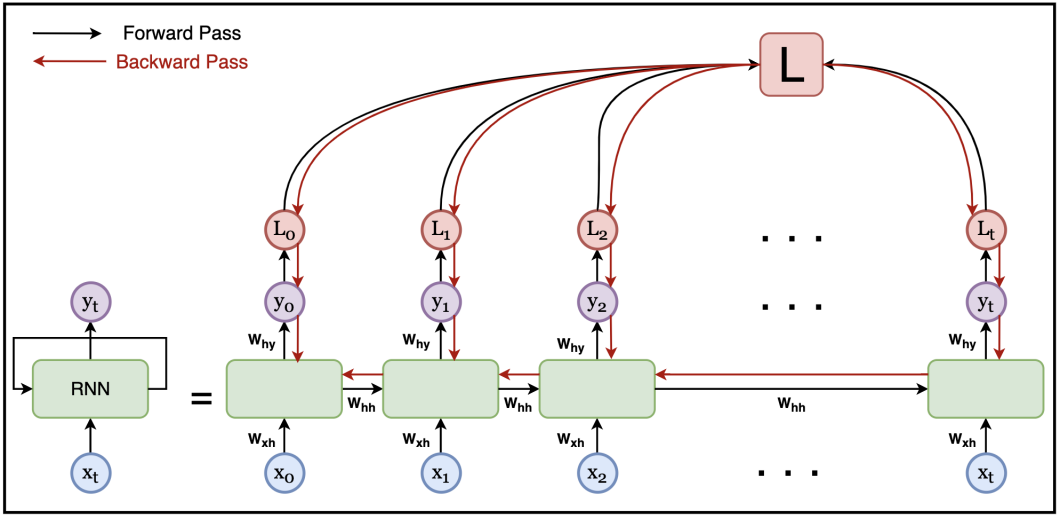
Figure 1: It is a RNN example: the left recursive description for RNNs, and the right is the corresponding extended RNN model in a time sequential manner.

RNN: Computation Graph Across Time



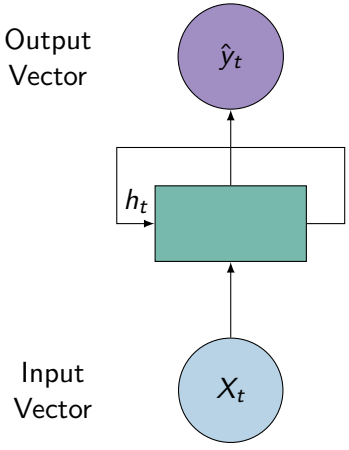
$\rightarrow$ represent as computational graph unrolled across time.

Back-Propagation Through Time



Re-use same weight matrices at every time step!

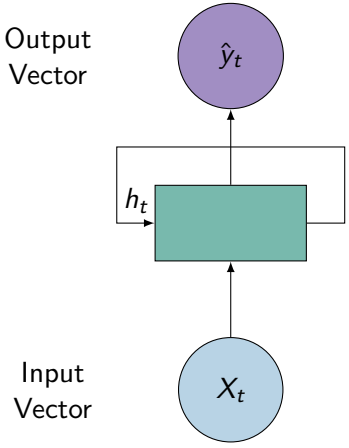
RNN From Scratch & Tensorflow

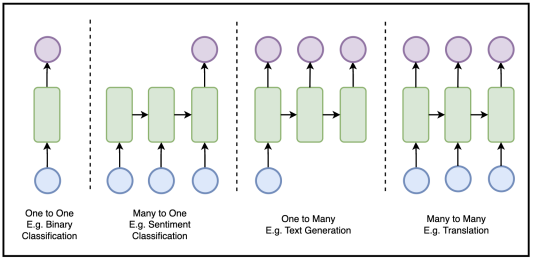



```python
class MyRNNCell(tf.keras.layers.Layer):
    def __init__(self, rnn_units, input_dim, output_dim):
        super(MyRNNCell, self).__init__()

        # Initialize weight matrices
        self.W_xh = self.add_weight([rnn_units, input_dim])
        self.W_hh = self.add_weight([rnn_units, rnn_units])
        self.W_hy = self.add_weight([output_dim, rnn_units])

        # Initialize hidden state to zeros
        self.h = tf.zeros([rnn_units, 1])

    def call(self, x):
        # Update the hidden state
        self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

        # Compute the output
        output = self.W_hy * self.h

        # Return the current output and hidden state
        return output, self.h
```

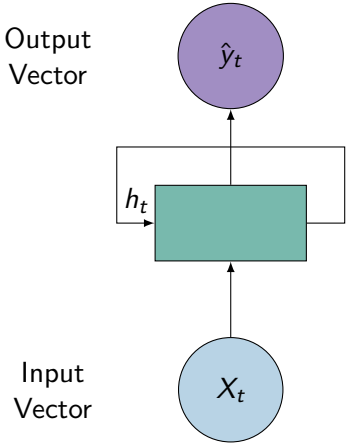
```python
tf.keras.layers.SimpleRNN(rnn_units)
```

RNN Intuition





Sequence Modeling — Design Criteria

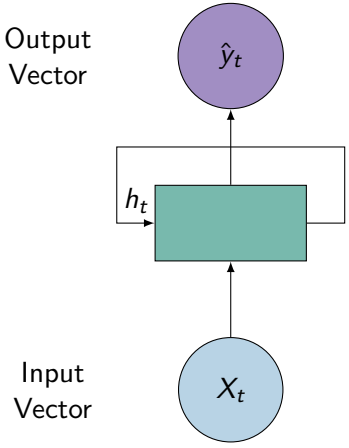


Recall: to model sequences, we need to:

- Handle variable-length sequences.
- Track long-term dependencies.
- Maintain information about order.
- Share parameters across the sequence.

$\rightarrow$ Recurrent Neural Networks (RNNs) meet these sequence modeling design criteria.

Handle Variable Sequence Lengths



The food was great.

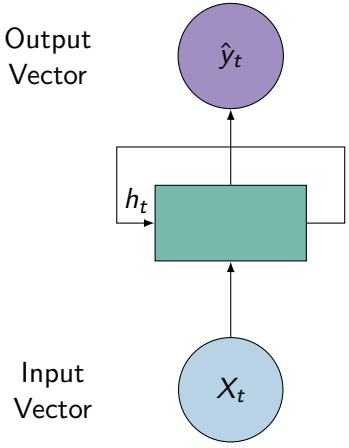
vs.

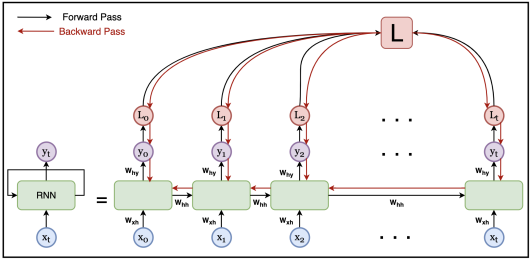
We visited a Pizzeria for lunch.

vs.

We were hungry because we went for sport before eating.

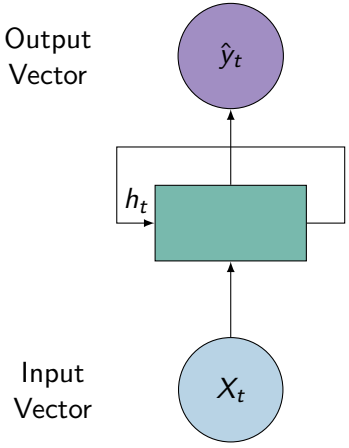
Backpropagation Through Time

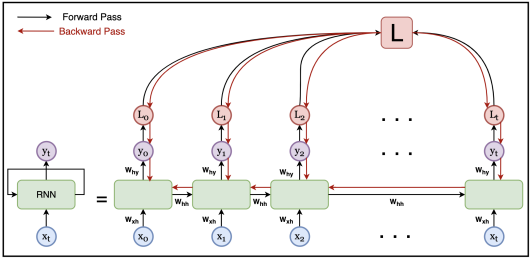




Computing the gradient wrt. h_0 involves many factors of $\mathrm{W}_{\mathrm{hh}}+$ repeated gradient computation!

Backpropagation Through Time





Many values > 1:
Exploding gradients

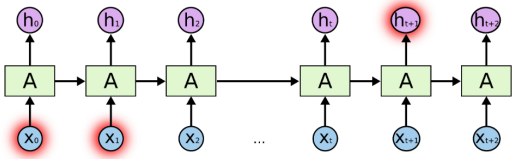
Many values < 1:
Vanishing gradients

Recall: RNN Hard to Train

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

Recurrent blocks suffer from two problems:

- Long-term dependencies do not work well.
  - Difficult to connect two distant parts of the input.
- Magnitude of the signal can get amplified at each recurrent connection.
  - At every time iteration, the gradient can either vanish or explode.
  - Very hard to train them.

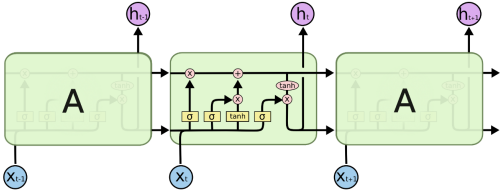


I grew up in England... and I speak fluent ____

Long Short-Term Memory (LSTM)

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

- Hochreiter & Schmidhuber (1997)
- LSTM layers are improved versions of the recurrent layers.
  - They rely on a gated cell to track information throughout many time steps.
  - They can learn long-term dependencies.
  - They can forget.
- They have an internal state and a structure which is composed of four actual layers.
  - Layers labeled with σ are gates which can block or let information flow.



Long Short-Term Memory (LSTM)

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

- The core of LSTM is a memory unit (or cell) c_t which encodes the information of the inputs that have been observed up to that step.
- The memory cell c_t has the same inputs ($\mathbf{h}_{t-1}$ and $\mathbf{x}_t$) and outputs $\mathbf{h}_t$ as a normal recurrent network, but has more gating units which control the information flow.
- The input gate and output gate respectively control the information input to the memory unit and the information output from the unit. More specifically, the output h_t of the LSTM cell can be shut off via the output gate.

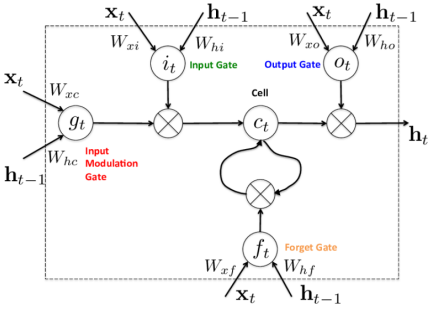
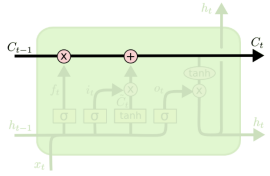


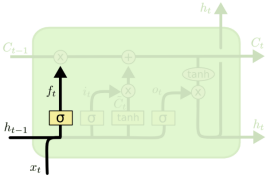
Fig. from G. Chen (2016)

LSTM Forget Gate

http://www.bioinf.jku.at/publications/older/2604.pdf
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- LSTMs follow two paths
  - They update their internal state.
  - They give an output based on the internal state and the input.
- A gate layer σ decides if we should forget an old part of the internal state.
- Something which has to be replaced by new information.



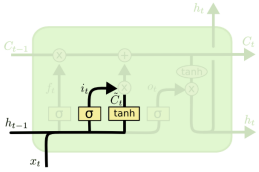


1. **Forget**
2. Store
3. Update
4. Output

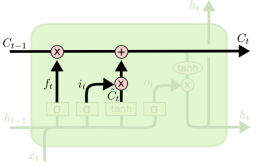
LSTM New State

http://www.bioinf.jku.at/publications/older/2604.pdf
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Once that the layer decided what to forget, it computes:
  - What has to replace it, i, based on the input and the old state.
  - What has to be used to replace it, the candidate value C_t
- The new state C_t can be computed based on the new information.



1. Forget
2. **Store**
3. Update
4. Output

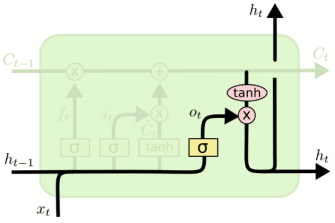


1. Forget
2. Store
3. **Update**
4. Output

LSTM Output

http://www.bioinf.jku.at/publications/older/2604.pdf
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Based on the new state and the input, the layer can produce a result.
  - this is the output.
  - the same value is also passed to the next iteration.
- Why is this so important?
  - Many translation algorithms and voice interpreters are based on small variations of this layer.
- Action required: `demo/05_RNN_intro.ipynb` (see also https://www.tensorflow.org/guide/keras/rnn)



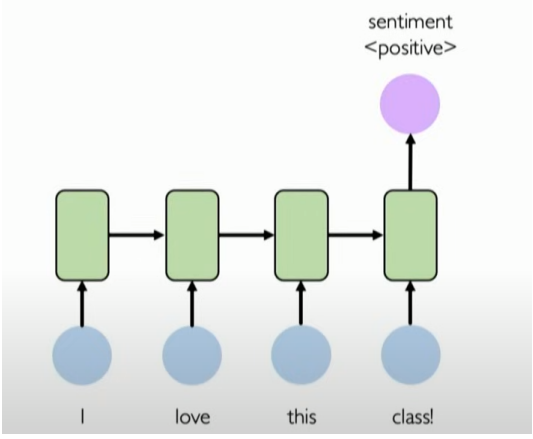
1. Forget
2. Store
3. Update
4. **Output**

Action Required

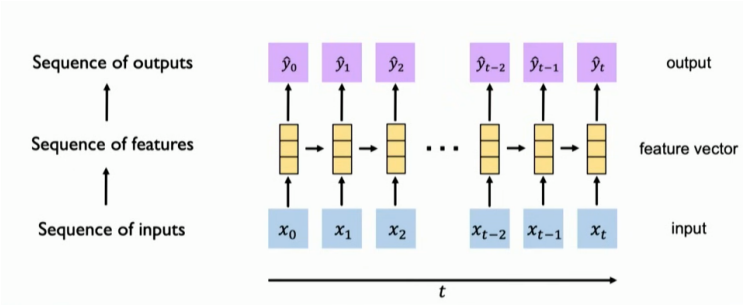
- `demo/05_RNN_intro.ipynb` (Ozone and stock market time series data).
- There is a weather data set from the Max Planck Institute of Biochemistry `https://www.bgc-jena.mpg.de/wetter/`.
- Open the notebook `demo/05b_Weather_data.ipynb`.
- Given this time series (Temperature as a function of time), try to make predictions of various time intervals into the future.

Limitations of Recurrent Models

- Encoding bottleneck
- Slow, no parallelization
- Not long memory (very long sequences cannot be handled, not even by LSTMs)
- Can we go beyond those limitations to process sequential data?



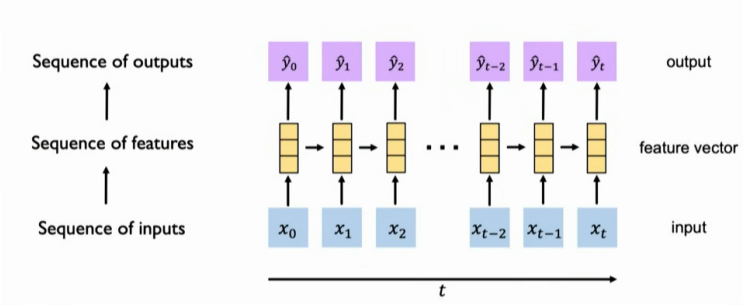
High-level Goal of Sequence Modeling



RNNs: recurrence to model sequence dependencies

- Encoding bottleneck
- Slow, no parallelization
- Not long memory (very long sequences cannot be handled, not even by LSTMs)
- Can we go beyond those limitations to process sequential data?

Desired Capabilities



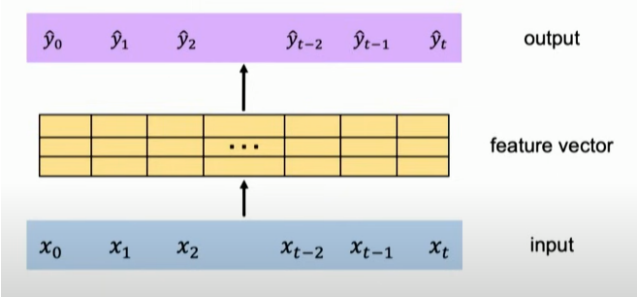
RNNs: recurrence to model sequence dependencies

- Continuous stream
- Parallelization
- Long memory

$\rightarrow$ Can we eliminate the need for recurrence entirely (i.e., no need to process the data time-step by time-step)?

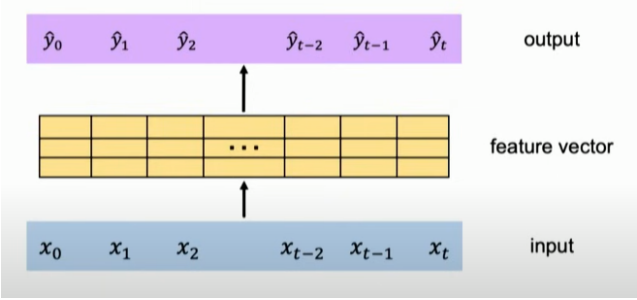
A Naive Approach: Feed everything into a dense network

→ Can we eliminate the need for recurrence entirely (i.e., no need to process the data time-step by time-step)?



- No recurrence (good)
- Not scalable (bad; network would be gigantic)
- No order (bad)
- No long memory (bad)

A Naive Approach: Feed everything into a dense network

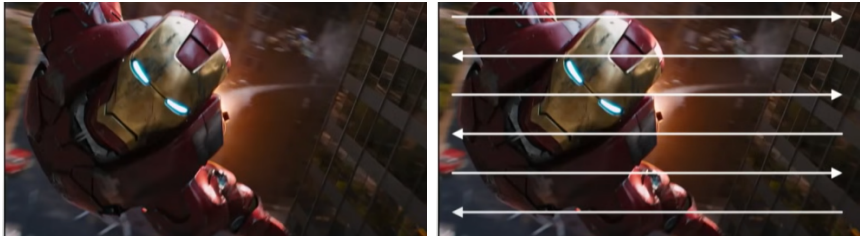


- No recurrence (good)
- Not scalable (bad; network would be gigantic)
- No order (bad)
- No long memory (bad)

$\rightarrow$ Idea: Identify and attend to what is important

Attention is all you Need

Attention: I cannot overstate the importance of this concept. It's the underlying building principle, e.g., of Chat-GPT (Generative Pre-trained Transformers).



Attending to the most important parts of an input.

- Idea: Identify and attend to what is important.
- Naive: Scan (with eyes) over the image.

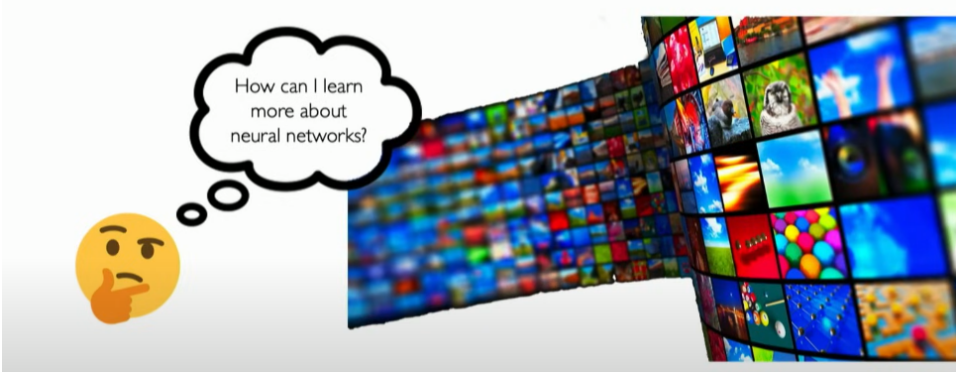
Intuition behind Self-Attention

Attending to the most important parts of an input.

Attending to the most important parts of an input.

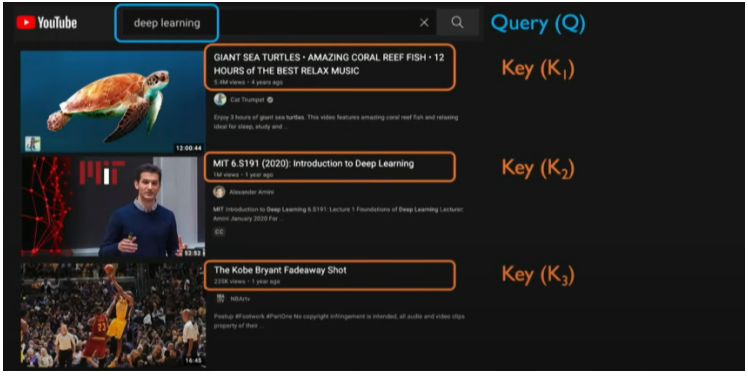
- Identify which parts to attend to (similar to a search problem).
- Extract the features with high attention.

A Simple Example: Search



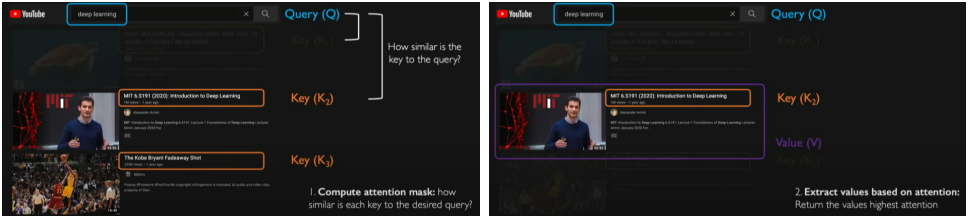
- Search through the internet.
- Trying to find something that matters.

Understand Attention with Search



- You enter a query ("deep learning").
- Find the overlap between your query and each of those titles right the keys in the database.
- We want to compute a metric of similarity and relevance between the queries and the keys.
- How similar are they?

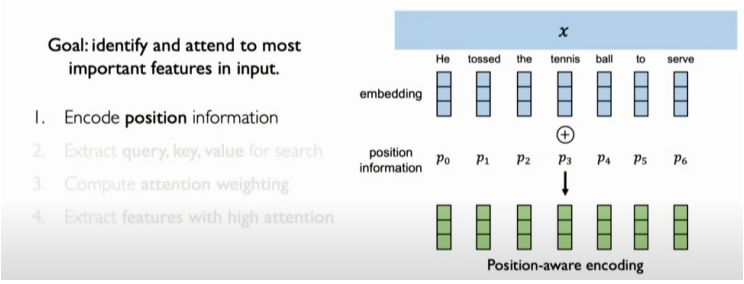
Understand Attention with Search



- The Second option is relevant.
- The Third option is not so relevant.
- Key operation: similarity computation bringing the query and the key together.
- The final step is now that we have identified what key is relevant to extract the relevant information that we want to pay attention to: the video itself.
- We can see this the value.
- Extracted successfully the relevant video.

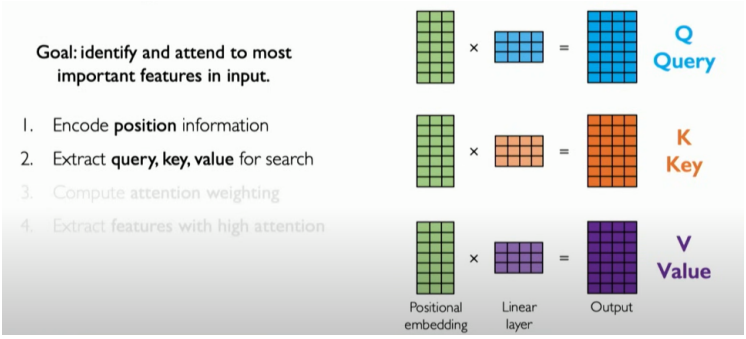
Learning Self-Attention with Neural Networks

→ Data is fed in all at once! Need to encode position information to understand order.



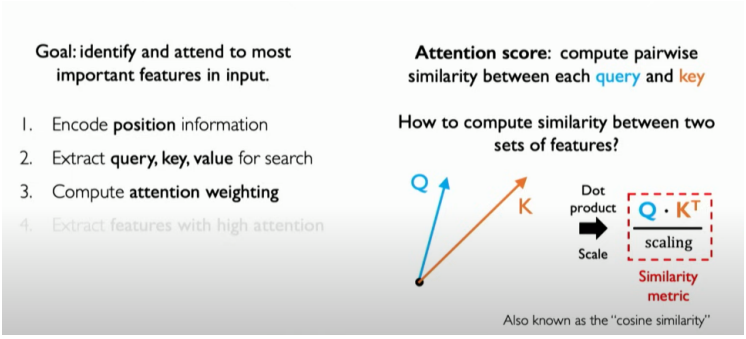
Learning Self-Attention with Neural Networks

→ Three sets of Neural Network weights for Query, Key, and Value.



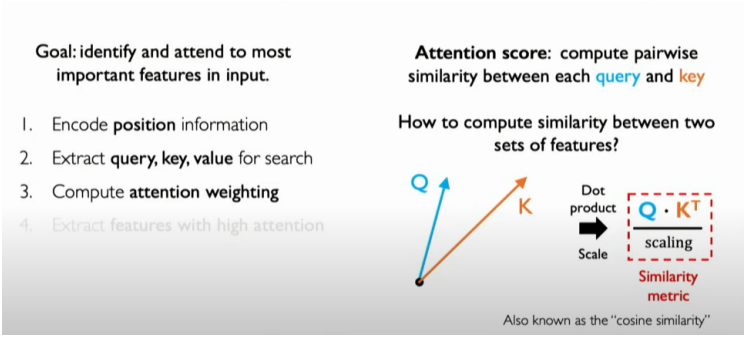
Learning Self-Attention with Neural Networks

→ Three sets of Neural Network weights for Query, Key, and Value.



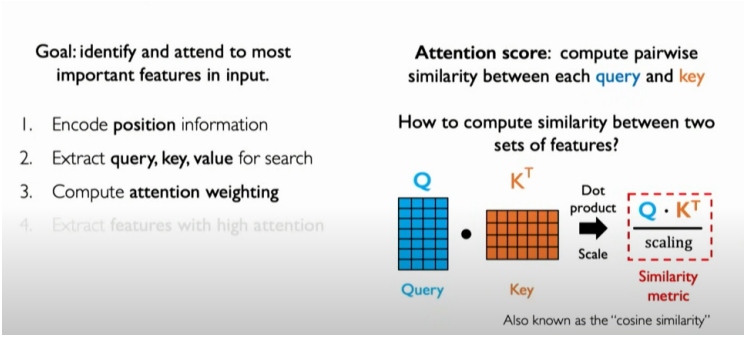
Learning Self-Attention with Neural Networks

→ Three sets of Neural Network weights for Query, Key, and Value.



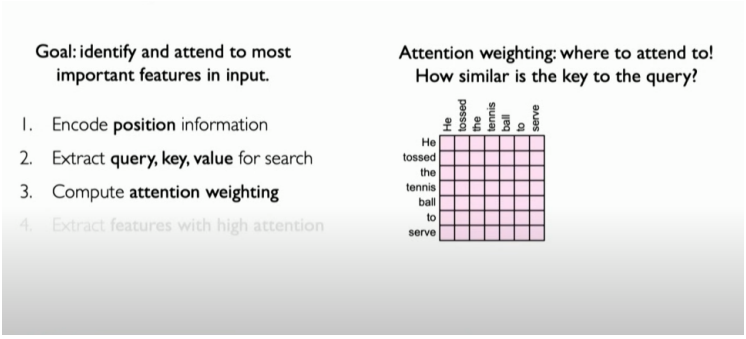
Learning Self-Attention with Neural Networks

→ Three sets of Neural Network weights for Query, Key, and Value.



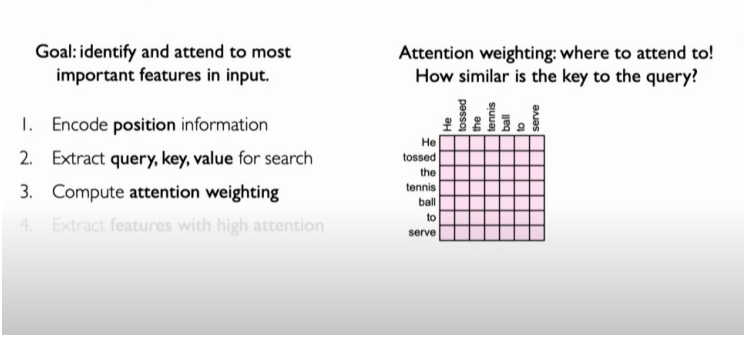
Learning Self-Attention with Neural Networks

→ Three sets of Neural Network weights for Query, Key, and Value.



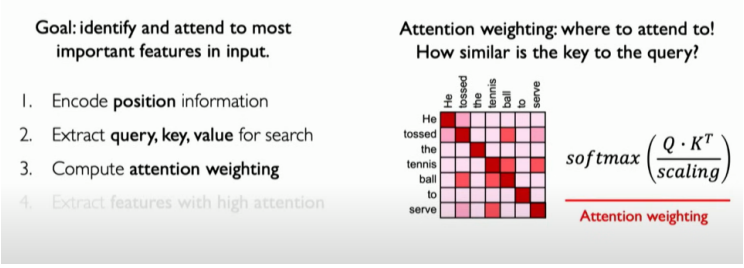
Learning Self-Attention with Neural Networks

→ Three sets of Neural Network weights for Query, Key, and Value.



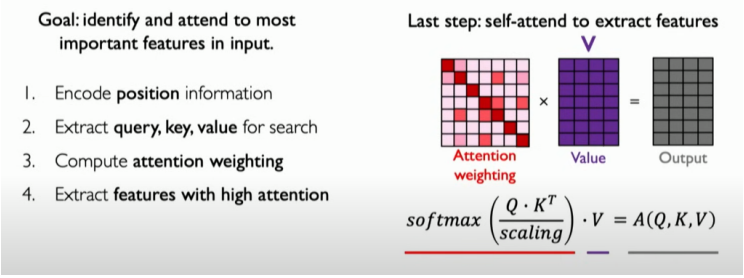
Learning Self-Attention with Neural Networks

→ Three sets of Neural Network weights for Query, Key, and Value.

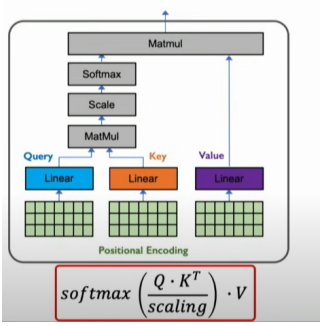


Learning Self-Attention with Neural Networks

→ Three sets of Neural Network weights for Query, Key, and Value.



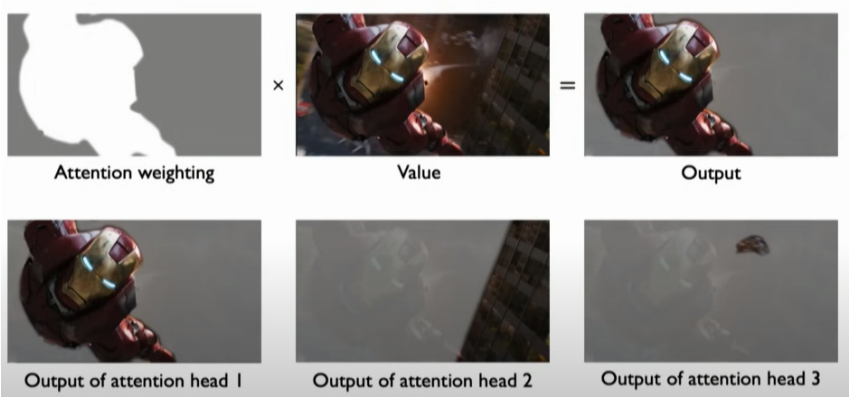
Learning Self-Attention with Neural Networks: Recap



- Goal: identify and attend to the most important features in the input.
- Encode position information.
- Extract query, key, and value for search.
- Compute attention weighting.
- Extract features with high attention.

$\rightarrow$ These operations form a self-attention head that can plug into a larger network. Each head attends to a different part of the input.

Applying Multiple Self-Attention Heads



Self-attention applied

Language Processing

An armchair in the shape of an avocado

BERT, GPT-3

Devlin et al., *NAACL* 2019
Brown et al., *NeurIPS* 2020

Biological Sequences

AlphaFold2

Jumper et al., *Nature* 2021

Computer Vision

Vision Transformers

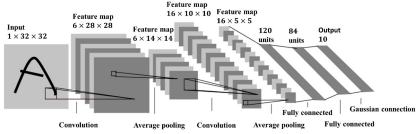
Dosovitskiy et al., *ICLR* 2020

Deep Learning for Sequence Modeling: Summary

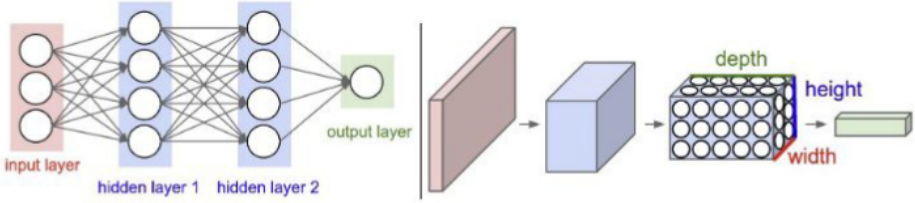
1. RNNs are well suited for sequence modeling tasks.
2. Model sequences via a recurrence relation.
3. Training RNNs with backpropagation through time.
4. Models for music generation, classification, machine translation, and more.
5. Self-attention to model sequences without recurrence.
6. Self-attention is the basis for many large language models.

Convolutional Neural Nets

- Possibly the most successful types of networks.
- Uses sequences of convolutional layers.
- Can be interleaved with pooling operations or fully connected layers.
- Train faster than MLPs.
- Can be used for 2D, 3D or higher dimensions (though 2D are the most common).
- Used for image recognition, object detection, sound analysis, etc.
- There exist more intricate architectures.
- Yann LeCun (1998): http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf



Convolutional Neural Nets

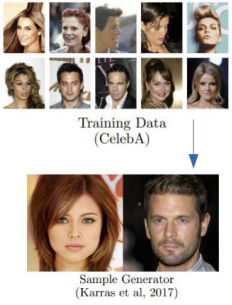


Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Generative Adversarial Nets

- GANs were introduced by Goodfellow et al. (2014) https://arxiv.org/abs/1406.2661.
- The idea is to train a network to generate samples which are indistinguishable from real ones (from the training set).
- The input is a random noise sample (latent space).
- Another network is trained at the same time to distinguish between real and fake samples.

See https://arxiv.org/abs/1701.00160 for a turorial. Upper fig from Goodfellow (2018).



Questions?