Intro to deep learning 2

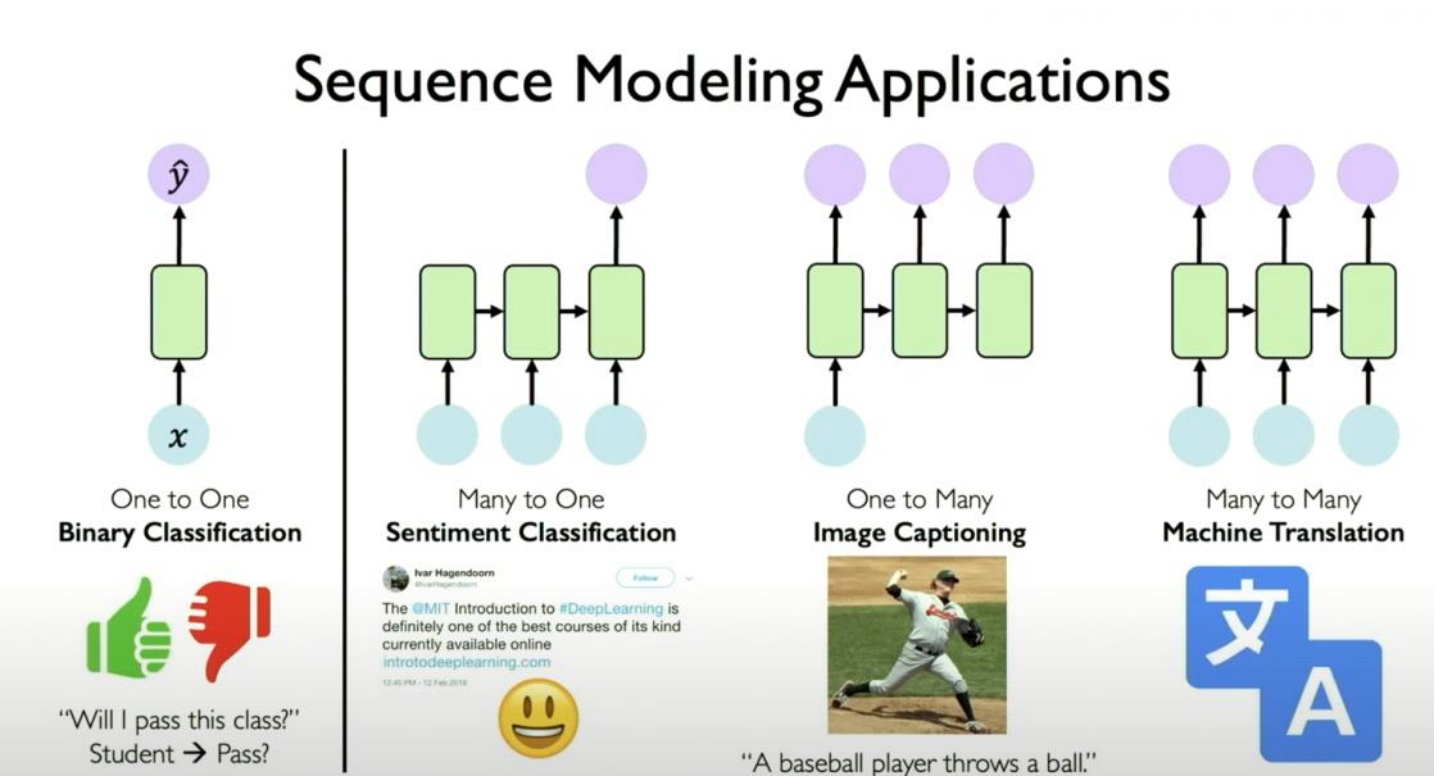
Wednesday, August 20, 2025 12:46 PM

Lecture 2

Deep sequence modeling lecture

Sequential data and seq modeling sequential data is any form of data with a dependence of some degree over time up to now. Examples: DNA, market data, audio waves, texts….

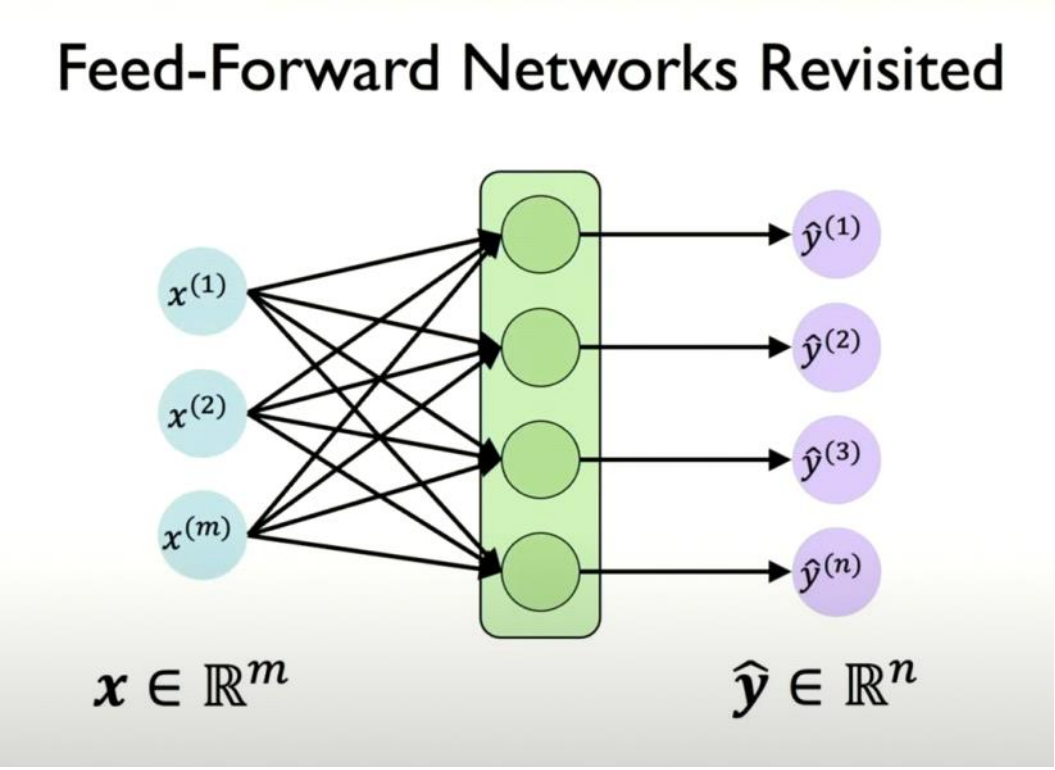
Various types of modeling: one to one, many to one (sentiment analysis), one to many (video generation), many to many (translation of sequences of words). There are many other architectures and criteria used but the main ones are in this lecture



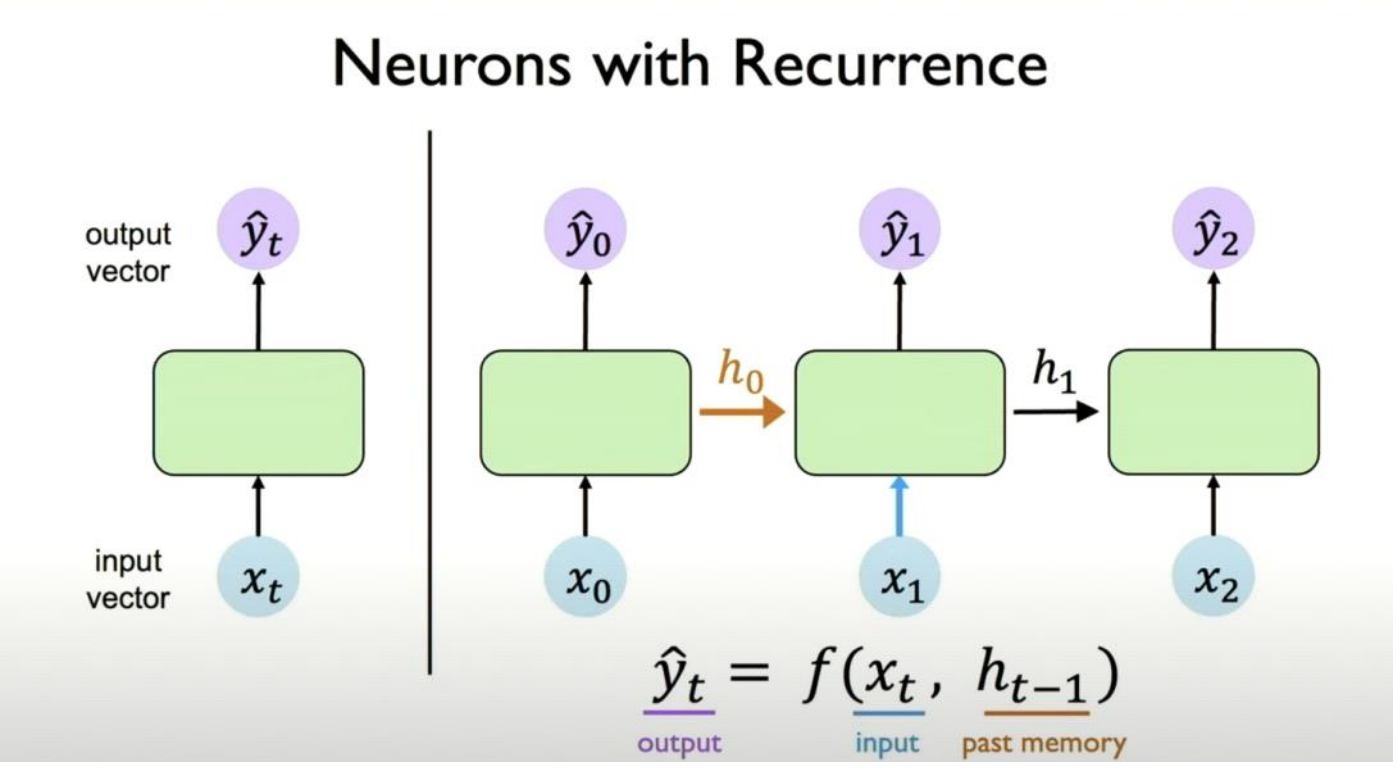
How is this done with NN?

Neurons with recurrence, revisiting the neuron

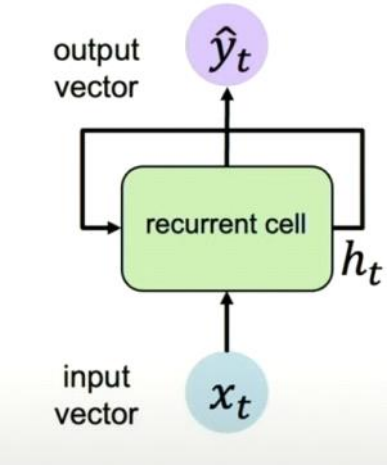
FFN as we know it. This does not give you any sequence, but it is a static input



We build out the same FNN where there is a relationship between time steps using a variable h_t called state of the network. We are capturing what happens previously in time steps t-1 and is passed forward



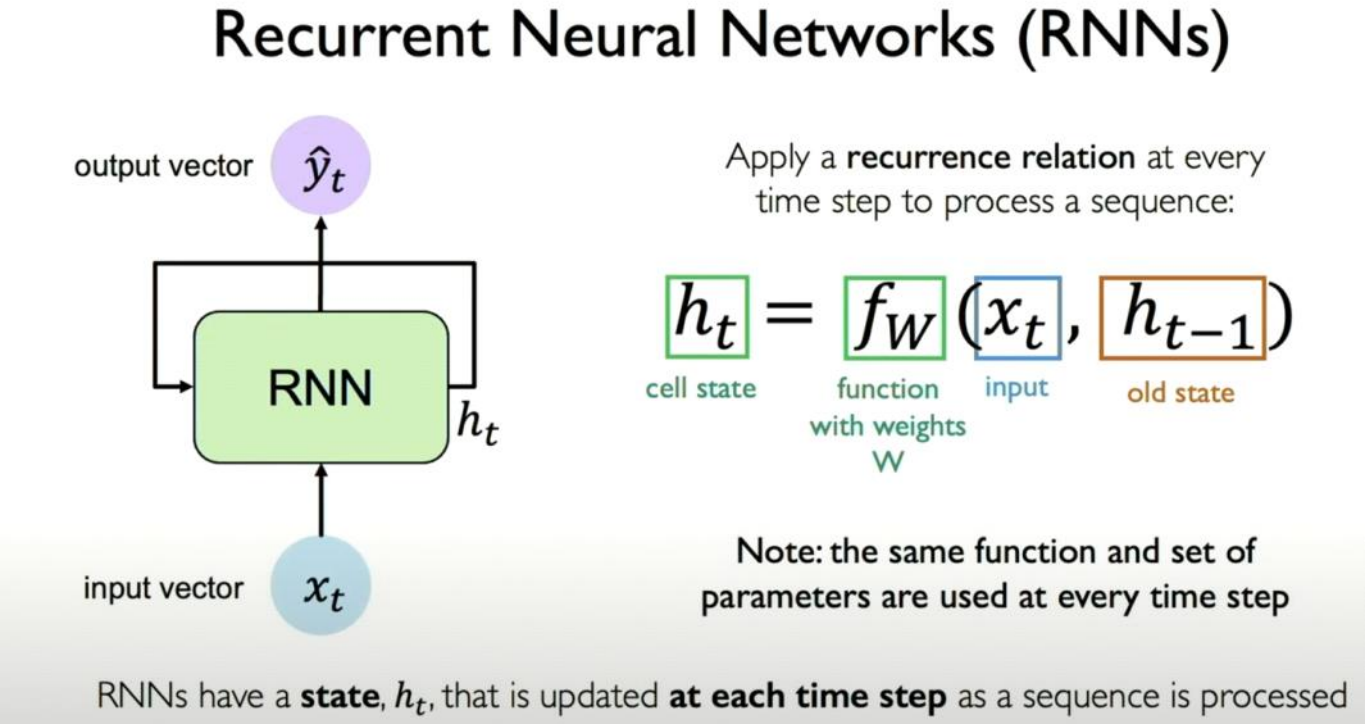
All wrapped up



This is in few words a recurrent neural network RNN

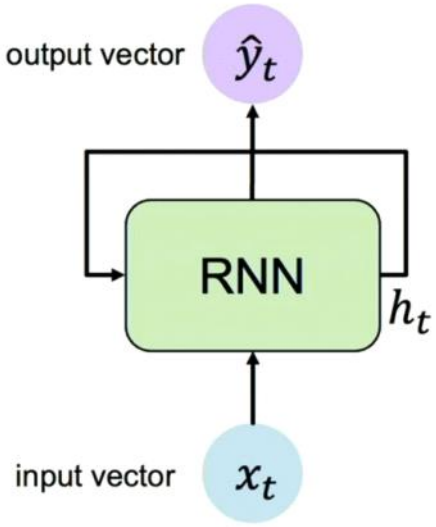
RNNs

RNNS have states h_t updated at each time step. The relation is given below in the image. The set of weights is both function of h and the information x



How the output is generated is similar to the other networks, but we have an extra equation for weight matrices to update h. We will also see later how are the various activation functions can be used

RNN State Update and Output



Output Vector

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

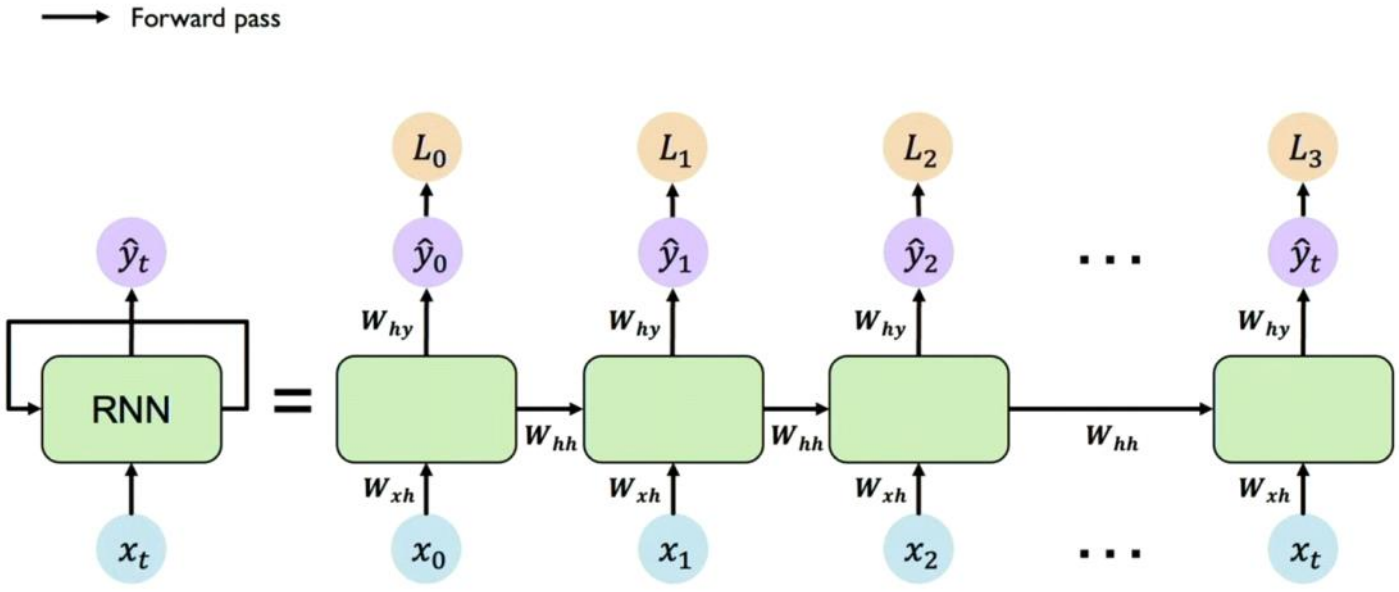
Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{\boldsymbol{hh}}^{\boldsymbol{T}} h_{t-1} + \boldsymbol{W}_{\boldsymbol{xh}}^{\boldsymbol{T}} x_t)$$

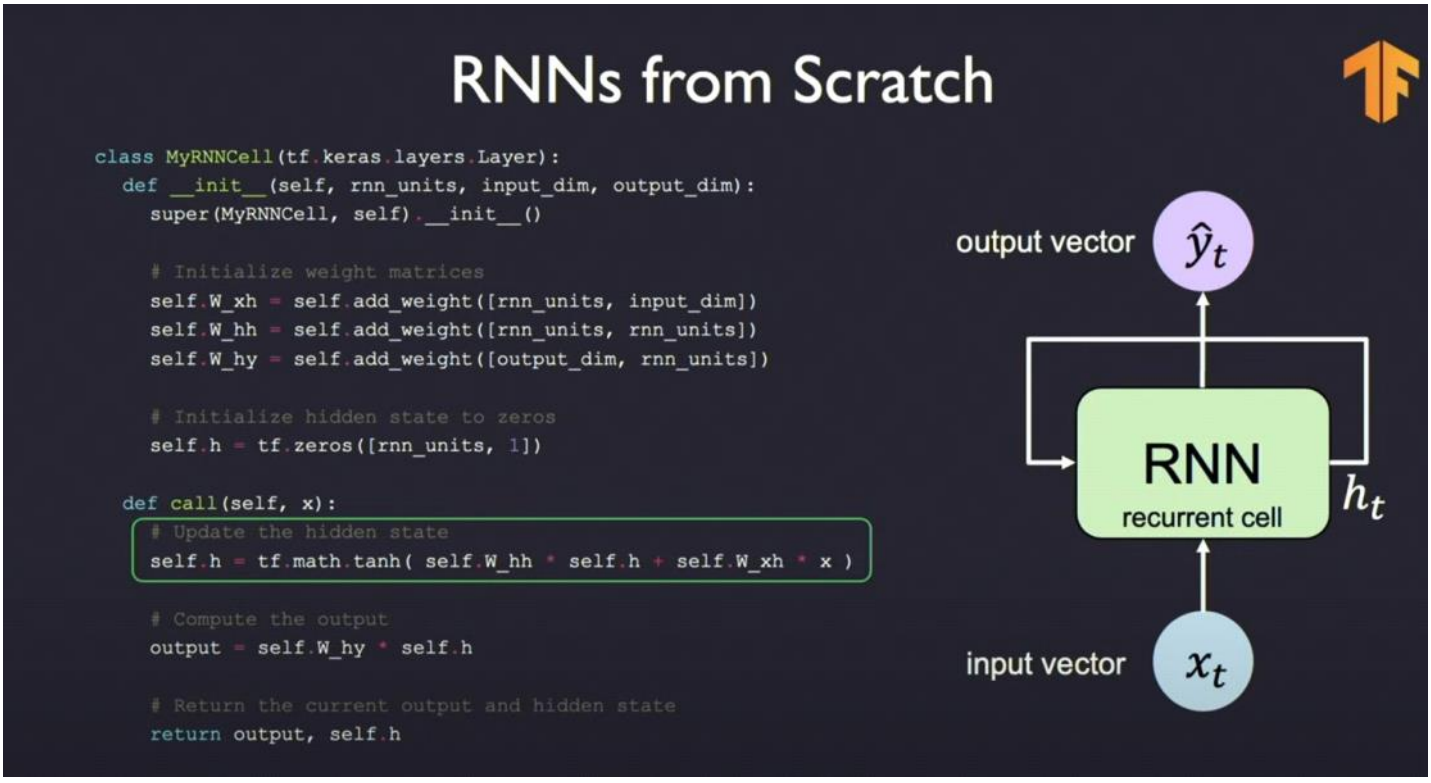
Input Vector

$$x_t$$

With this way of doing things over time steps, we have a loss at each step. Note all the weight matrices KEEP being updated over time steps. Below a computational graph



We can handle the losses by summing them together.
The pseudocode looks like below

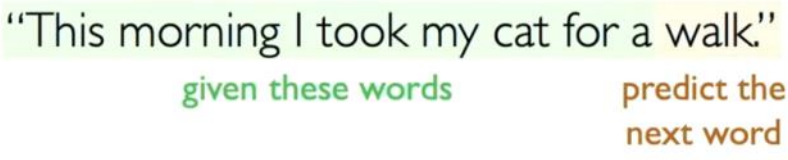


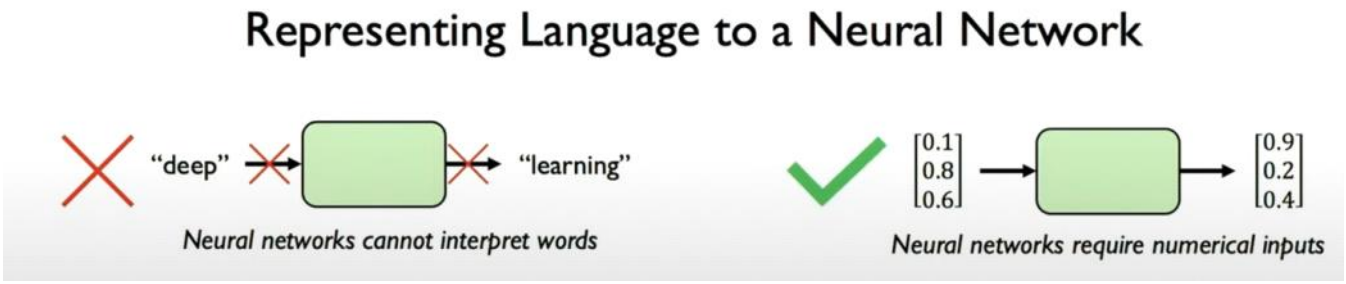
Design criteria

What unique aspects we need the NN to capture to handle well the sequence data? Sequences have various lengths, long-term dependencies, specific order, and share parameters across the sequence. RNN cover those criteria with their architecture

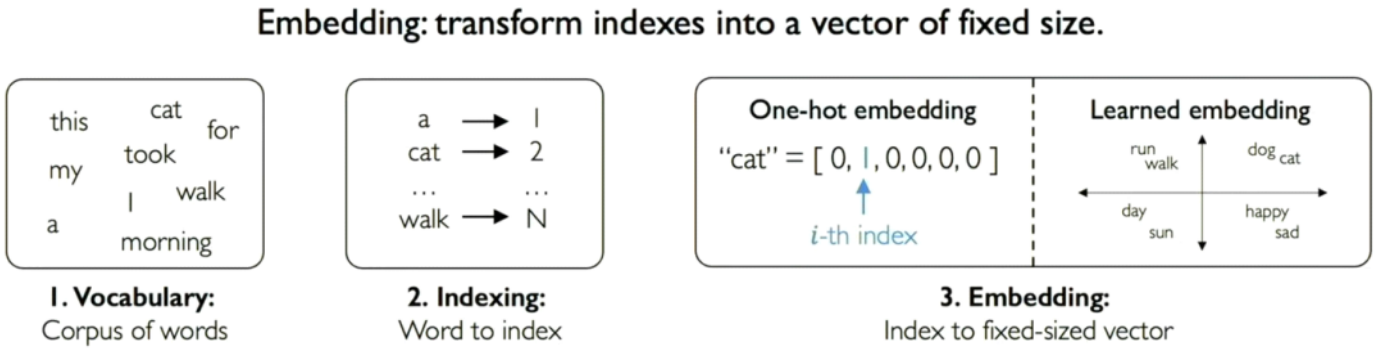
Example problem: next word prediction

A neural network is a numerical operator. How do I feed this sequence data to the network? We transform the input in a numerical vector of fixed size



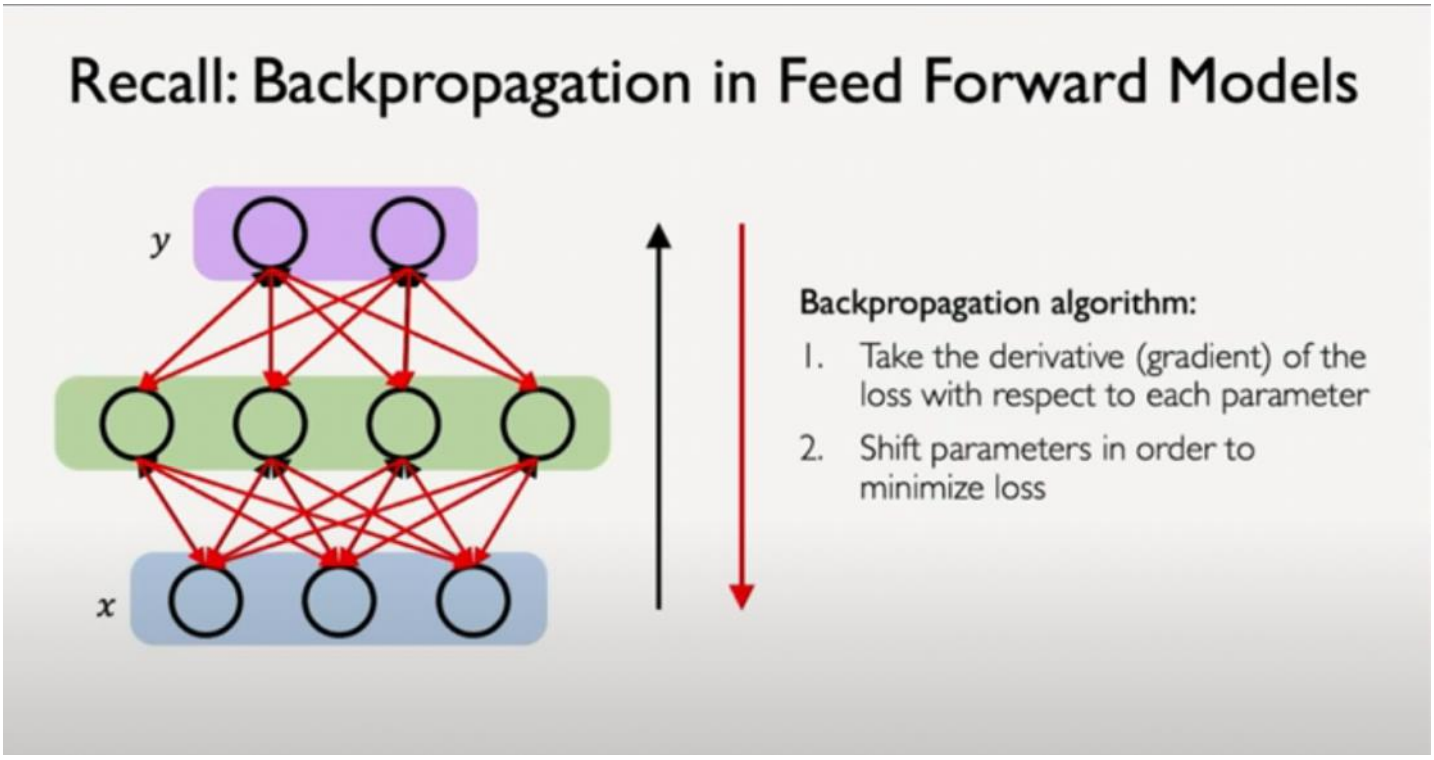


There are various ways to do this. A one-hot index with a huge vector encoding each word. Another way is to learn an embedding projecting words into a vector space (word2vec)

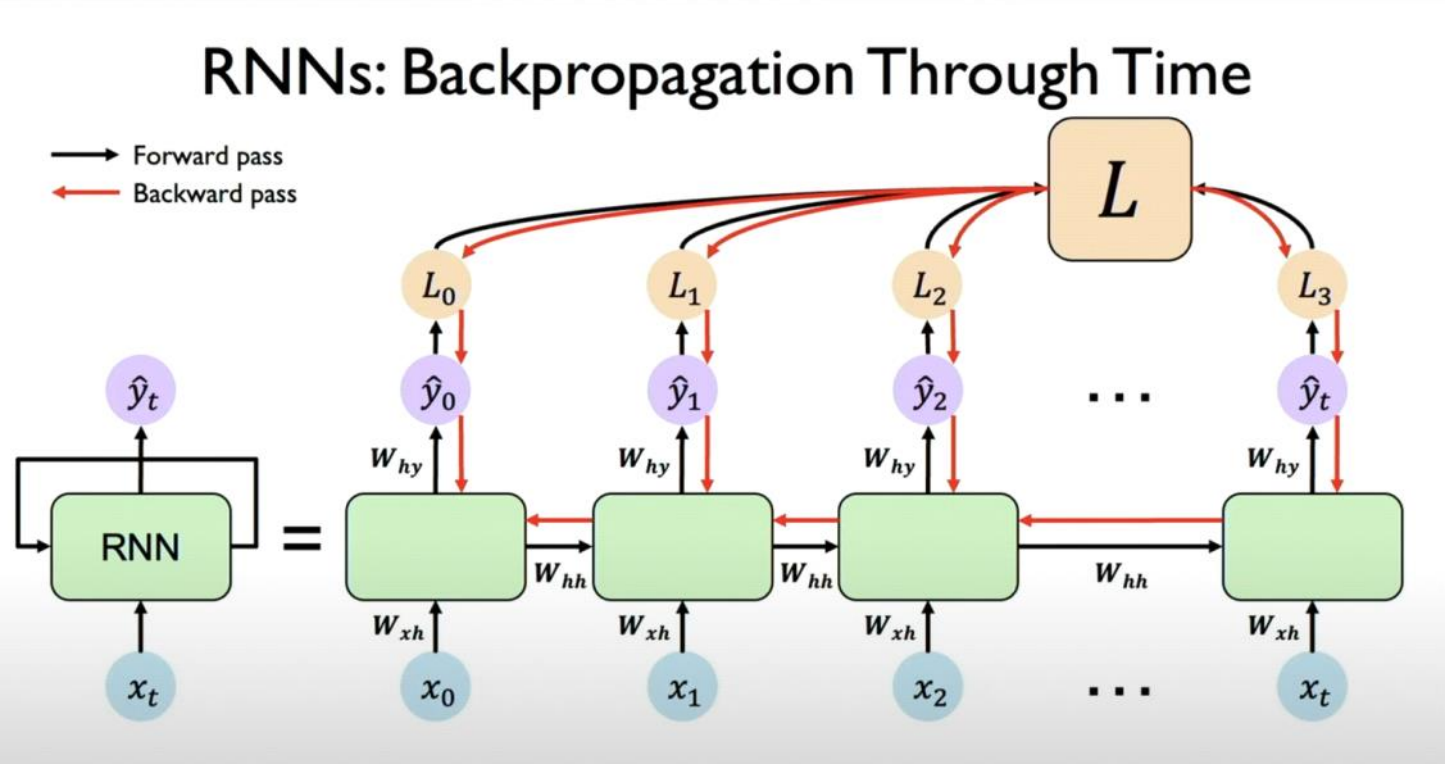


Backpropagation through time BPTT

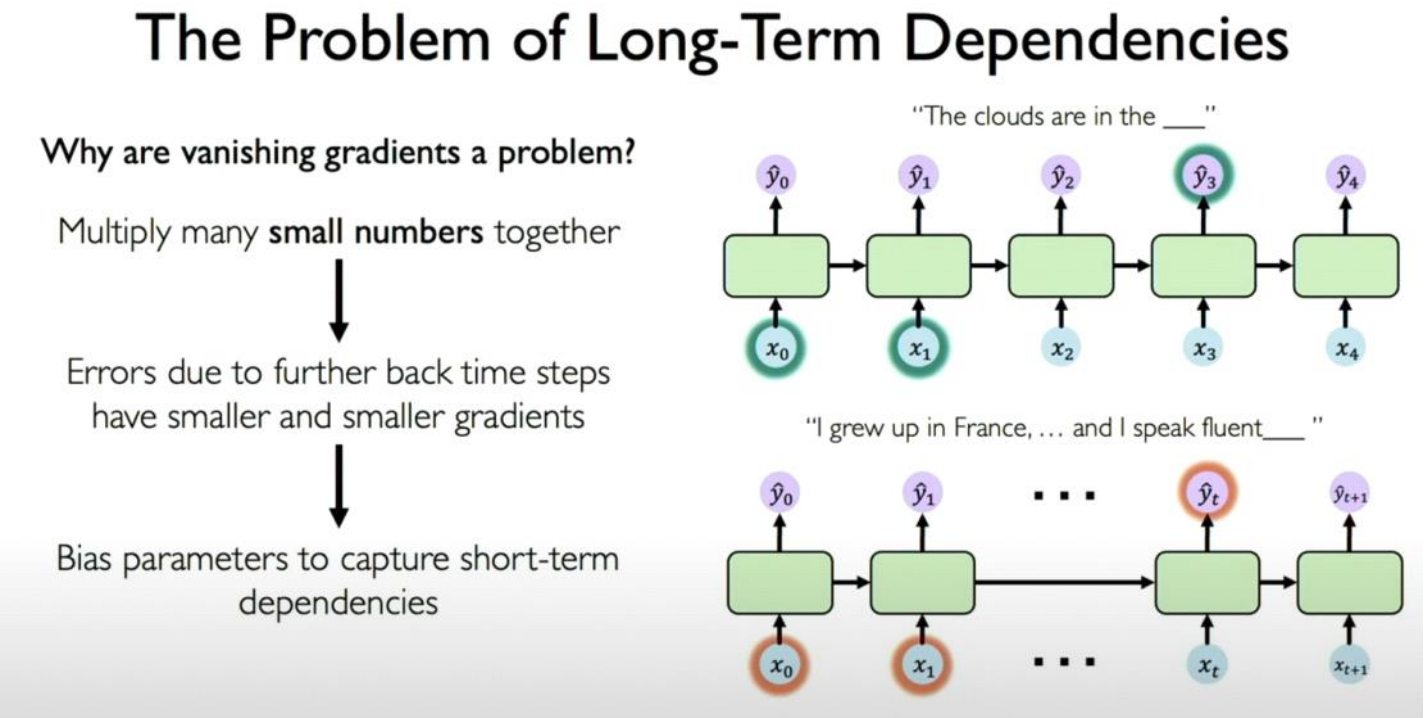
How the training is done this time! In FNN we propagate the gradient to adjust the parameters and minimize the loss.



Now our loss is computed over time steps and added at the end. We need to backpropagate through each single loss the weight matrix for the output,
and backpropagate through each time step of the networks. For each loss you thus continue through a different nr of time steps with the backpropagation

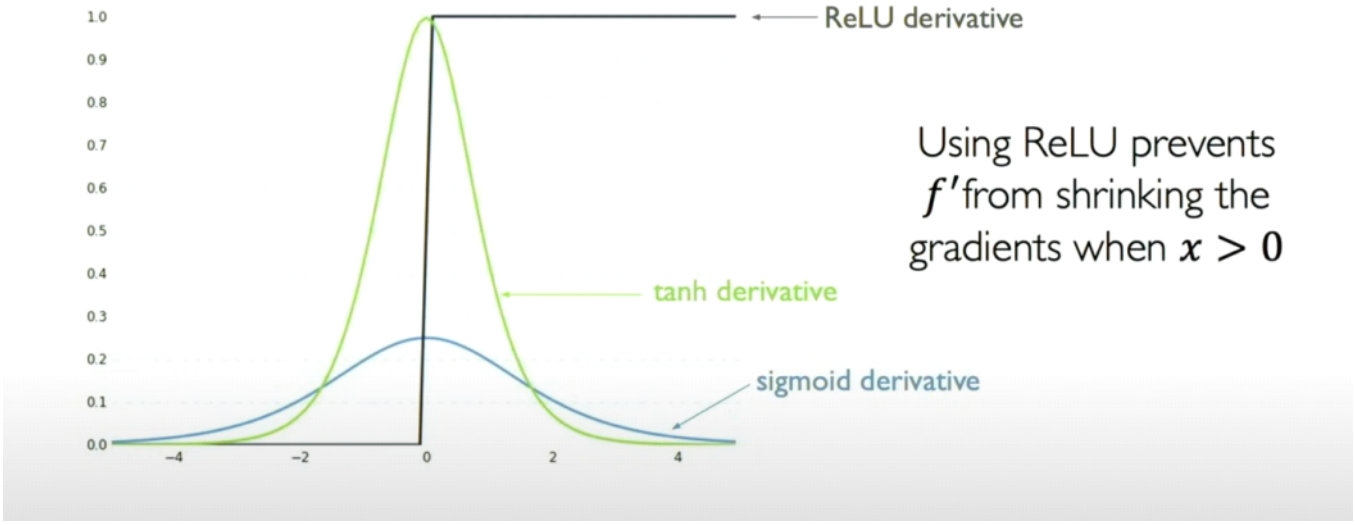


Gradients can become too big or small and there are techniques to solve the problem. Gradient clipping for large values, or network architecture, weight initialization and choice of activation fun for vanishing gradients.
Vanishing gradients are in reality diminishing the capacity of modeling of the network because with very long sequences you have a hard time keep track of all the time sequence



RELU activation is also good for this problem because the derivative helps reducing the shrinking when x is positive

Trick #1: Activation Functions



Initialize weights which reduce shrinking

Trick #2: Parameter Initialization

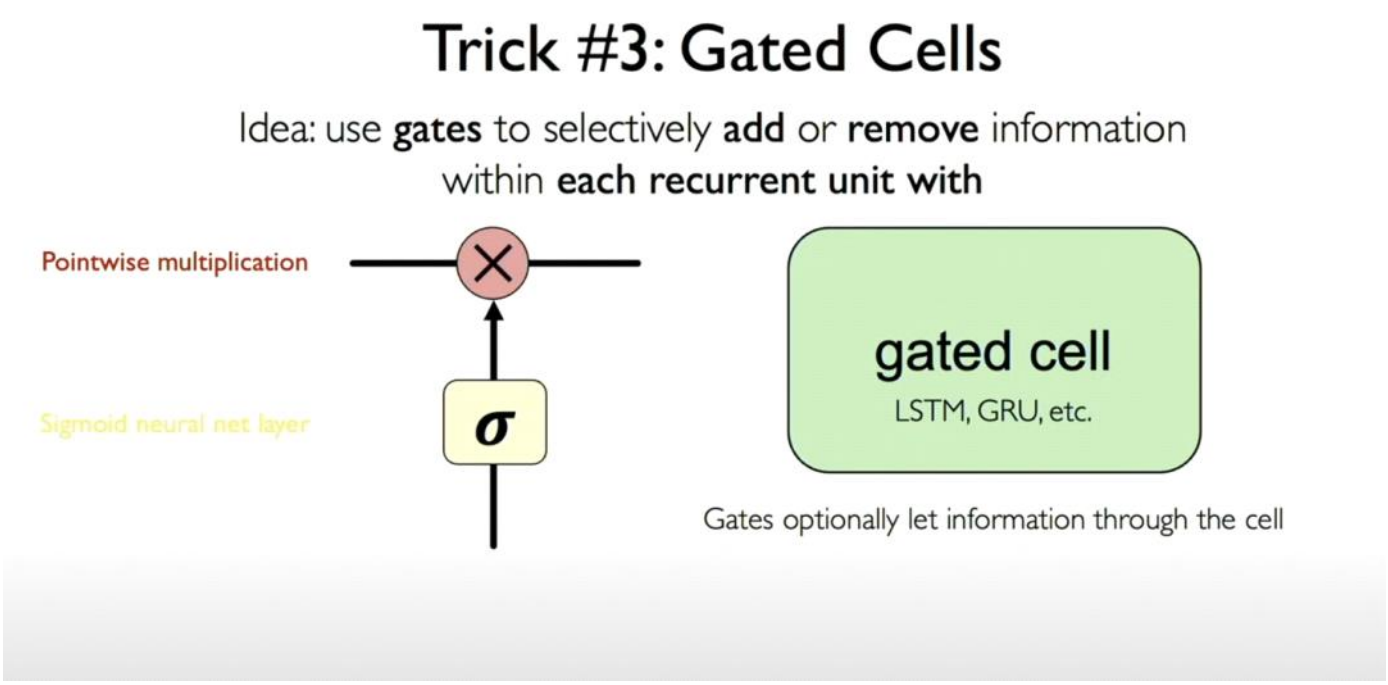
Initialize **weights** to identity matrix

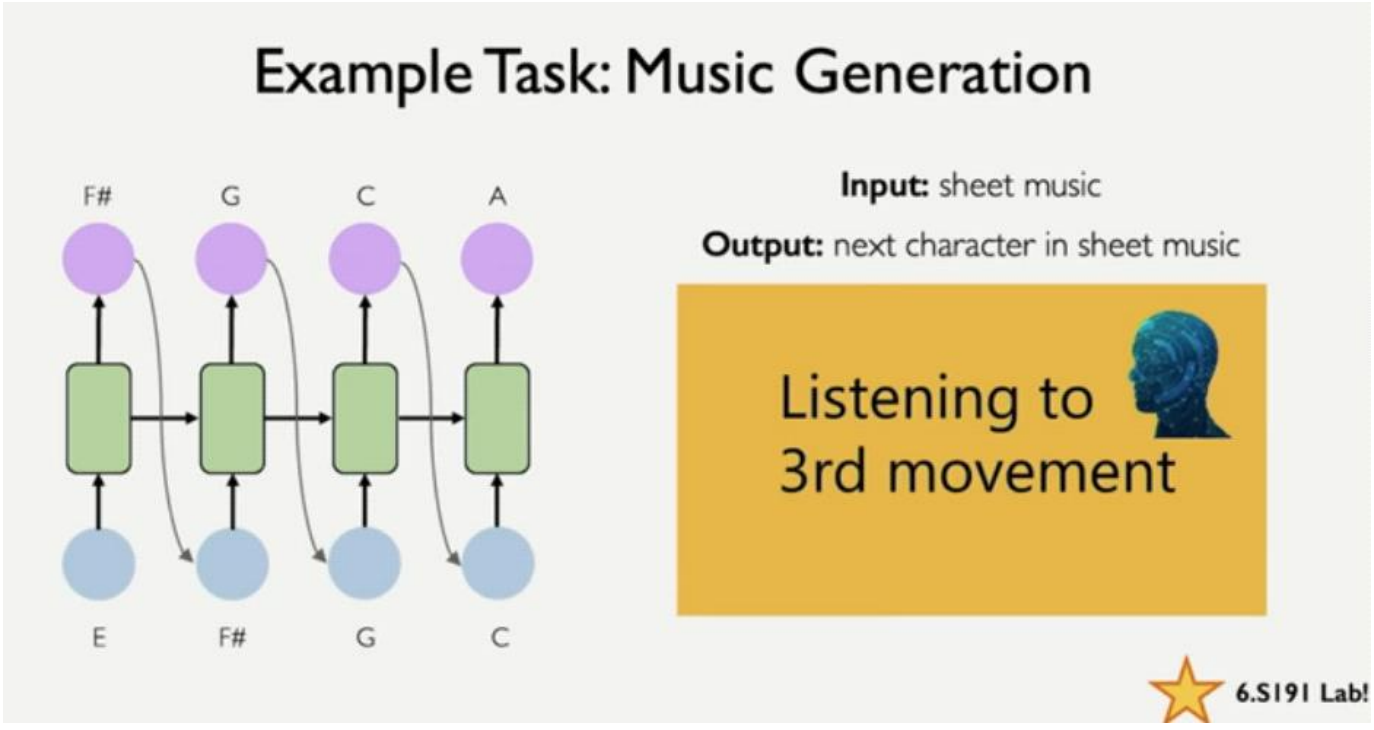
Initialize **biases** to zero

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

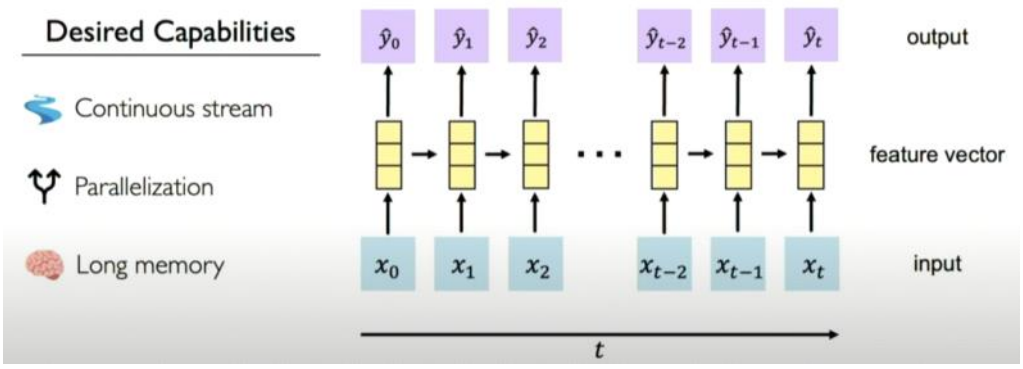
This helps prevent the weights from shrinking to zero.

Keep or remove specific informations from the product with gating cells. Gating is for example employed in NN models such as LSTM (Long-Short term memory neural network). The update of cell state is such that an additional layer decides how to keep or ignore some of the information. The update to the cell state is then more intelligent and keeps necessary information. All this is part of the backpropagation of gradient.



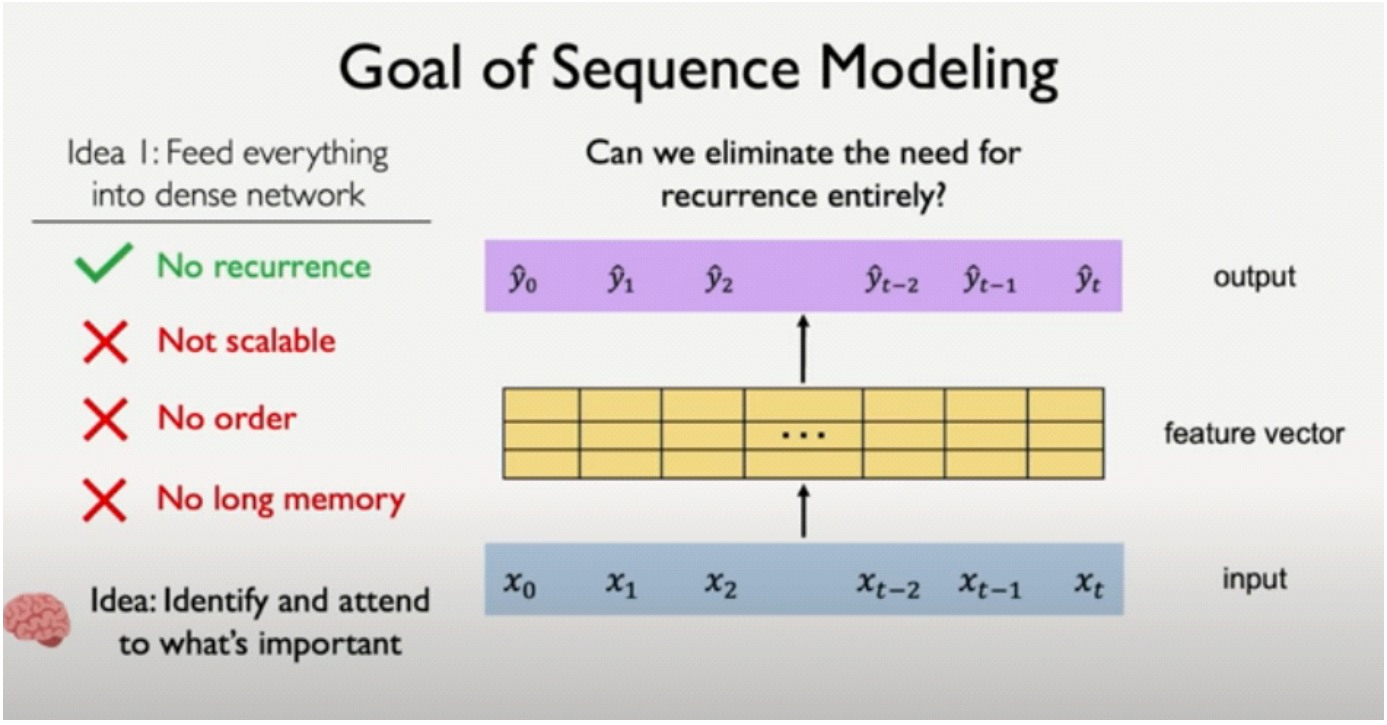


RNN are not always the answers because they have a bottleneck in the size of the hidden state. It is slow (not parallelized), it does not have real long term memory
We want models able to handle a continuous stream of information, a model efficient with large memory capacity

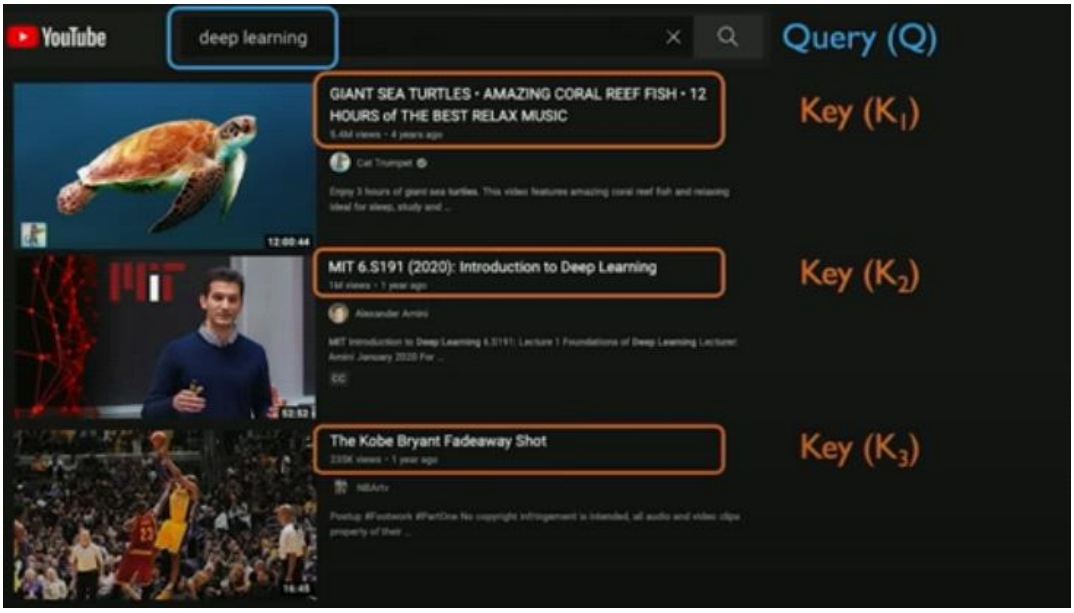


One could think about eliminating recurrence and concatenate all the data at once to generate output at once. This eliminates all notions of sequence and the scalability is not existing because we have a huge input. The memory capacity is also lost. Another idea is to learn a NN model which sees which parts of a sequence are important and convey info we should be capturing and learning. This is the core **idea of ATTENTION. The TRANSFORMER is a type of NN where attention is at the foundation of its mechanism.**

ATTENTION - SELF ATTENTION

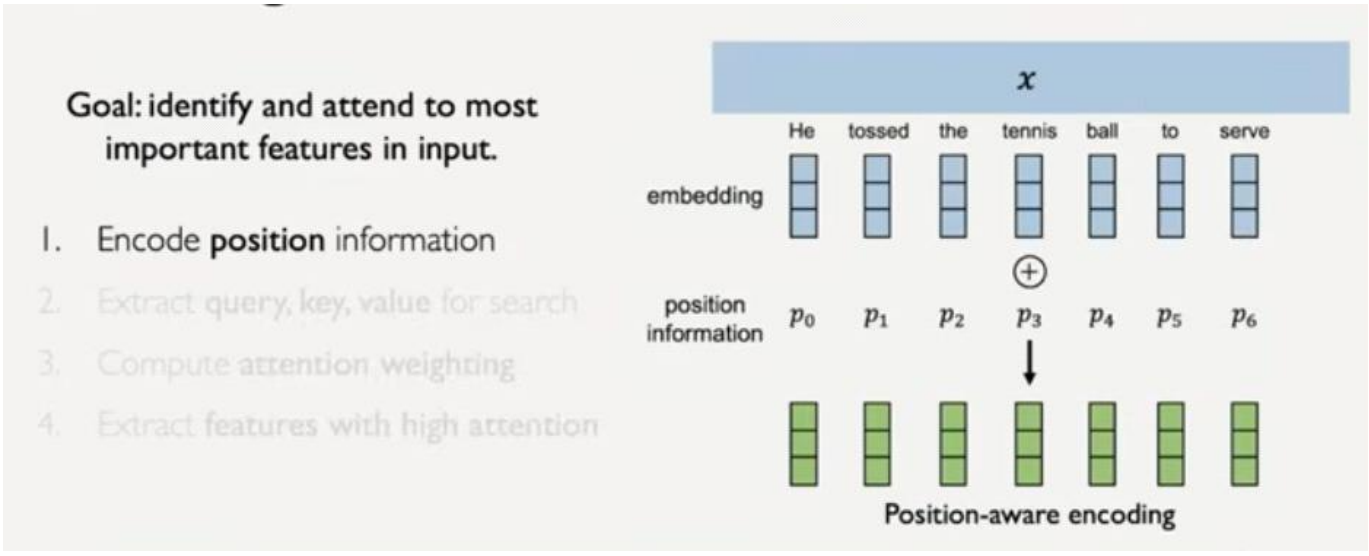


You need to find in the data the important parts. This is similar to a search problem. Let's look at how this work. For your query you want to find the most similar
Set of features, a key. The most similar becomes your value. For example

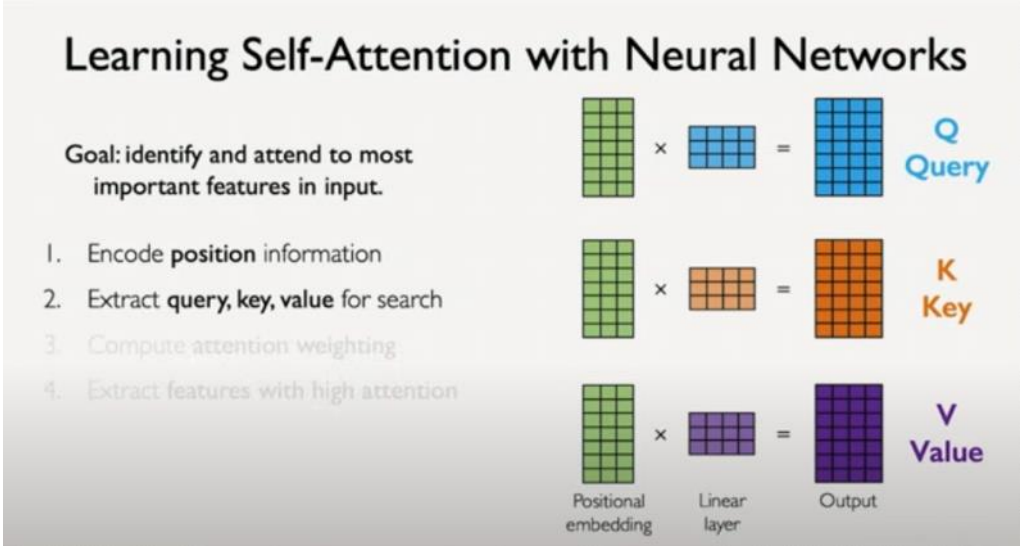


In practice: Note we have gone past the recurrence need, we have instead all data at once. We want to extract the most important features.

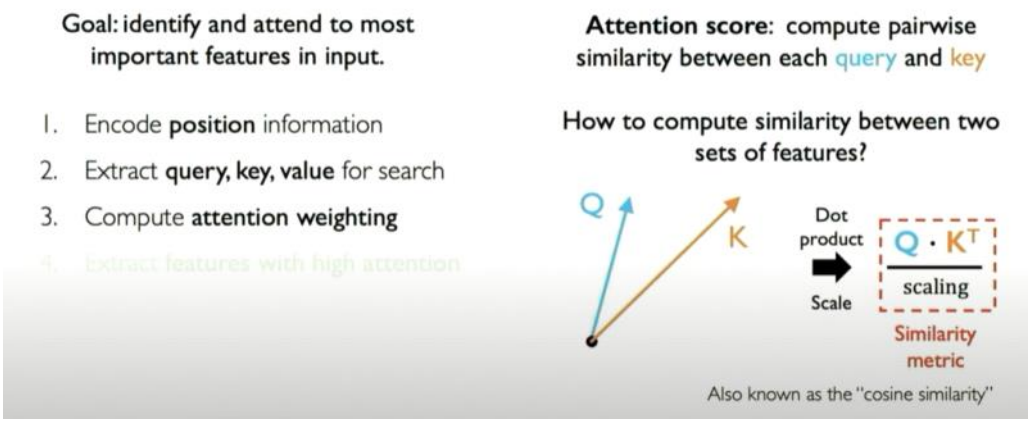
First of all you need to encode the information about order and position. In self attention and transformer we use a position aware encoding with positional embeddings



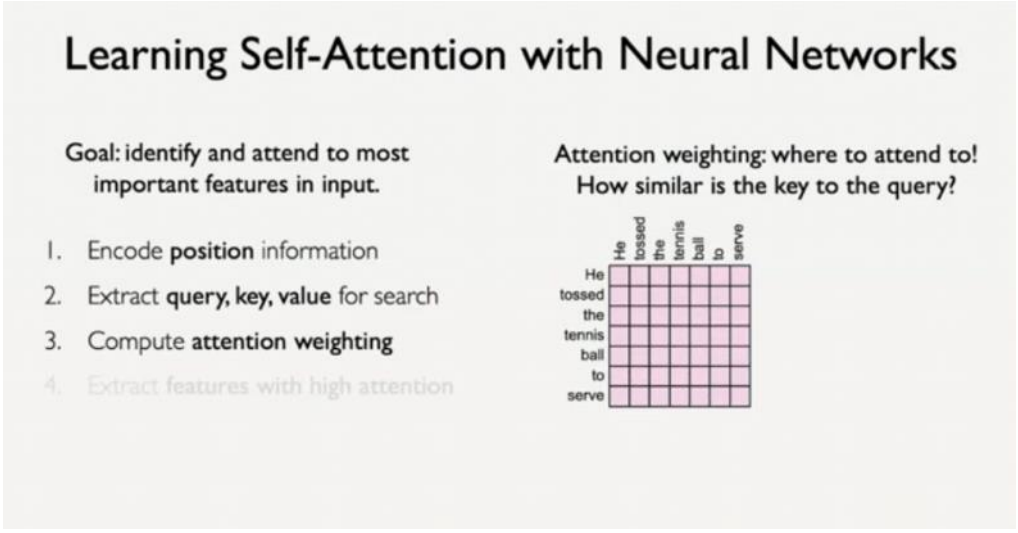
Data is fed all at once and also preserve positions now. Afterwards you need to extract the Query Q, Key K and Value V for search.
This is done learning three NN layers, which are three sets of matrices. Those are done with three NN layers on the same positional embedding.

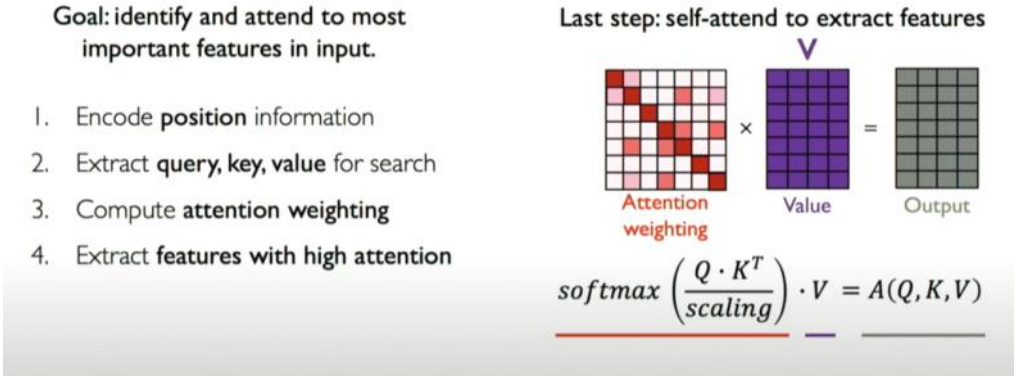


The attention score is done with the pairwise similarity of Query and Key. Here you are effectively comparing which
data features are important.
Similarity is computed with their dot product (cosine similarity). We apply a softmax function to the scaled similarity
so we have probabilities. The final step
is now to take the attention weighting and multiplying it by the Value.
So we identify what is important and **extract** relevant features (self attendance step).

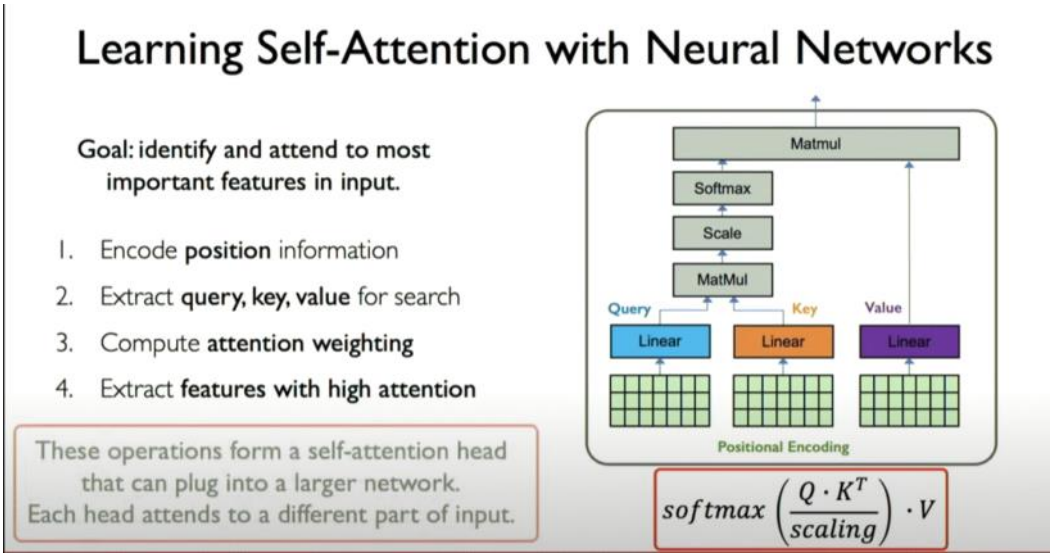


What does it all mean?





Wrapping up



You can have multiple attention heads which extract different sets of features that are relevant in the data

Applications examples