

UiO : Department of Informatics
University of Oslo

IN5400 Machine Learning for Image Analysis

Lecture 10: Recurrent Neural Networks

March 22 , 2022

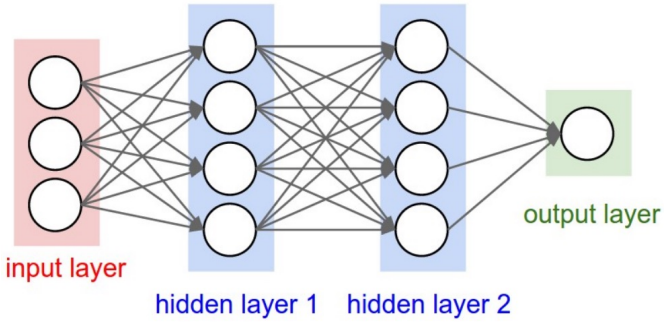
Dhananjay Tomar

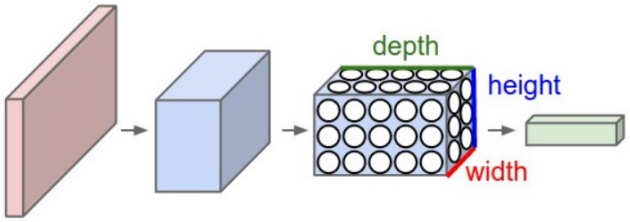
Outline

- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNNs
- Training Recurrent Neural Networks
- Challenges in training: Exploding or vanishing gradients
- LSTM and GRU
- Multilayer and bidirectional RNNs
- Image captioning (mandatory 2)

Overview

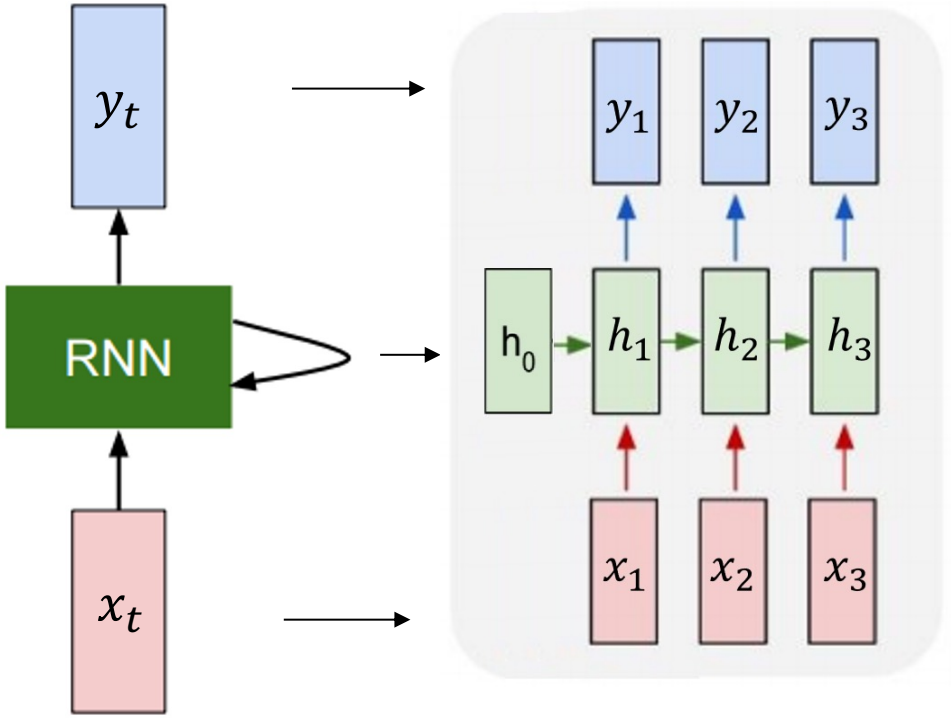
- What have we learnt so far?
  - Fully connected neural networks
  - Convolutional neural networks (CNNs)
- What are their limitations?
  - Processing data with unknown length
- Typical applications of recurrent networks (many types of sequence data):
  - Speech recognition
  - Music generation
  - Sentiment analysis (e.g., +ve vs –ve reviews)
  - Text analysis and translation





Recurrent Neural Network (RNN)

- Given a time series of input data $X = \{X_1, \ldots\ldots X_N\}$ (text, images, time series)
- X_t is typically a vector
- Compute hidden state H_t
- Generate output Y_t

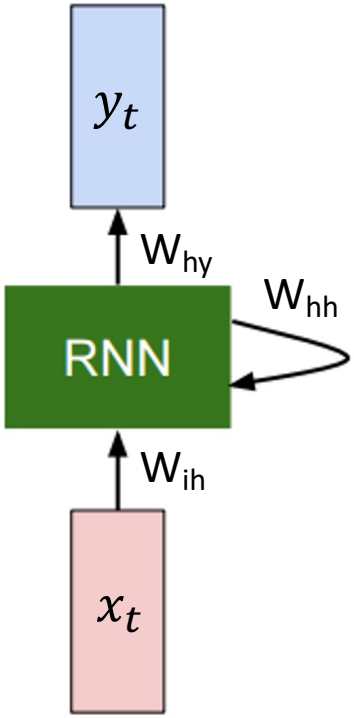


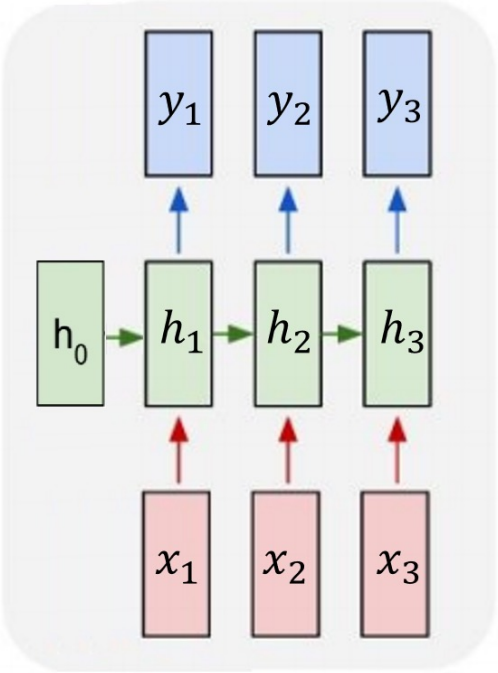
(Vanilla) Recurrent Neural Network

- Input vector: x_t
- Weight matrices: W_{ih}, W_{hh}, W_{hy}
- Vanilla RNN update:
  - Hidden state vector:

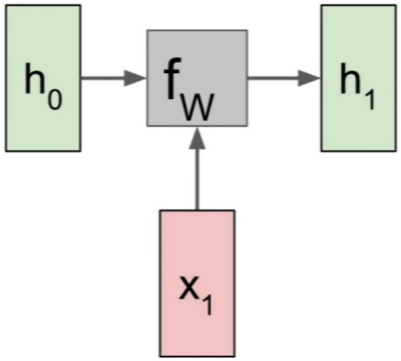
    $$H_t = \tanh(W_{ih}x_t + W_{hh}H_{t-1} + b)$$

  - Output: $y_t = g(W_{hy}H_t + b)$ where g is an activation function
- **Remark:** We often concatenate W_{hh} and W_{ih} into W, and multiply with the concatenation of H_{t-1} and x_t.

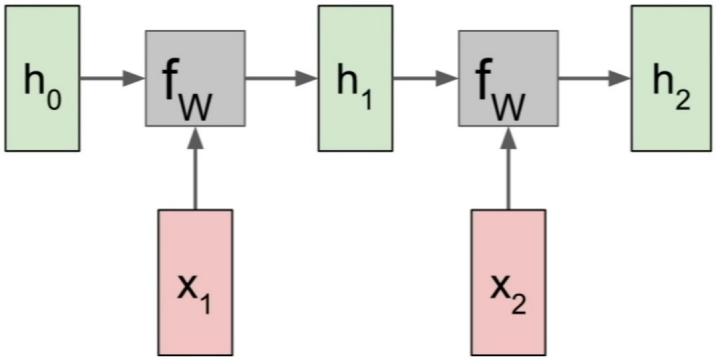




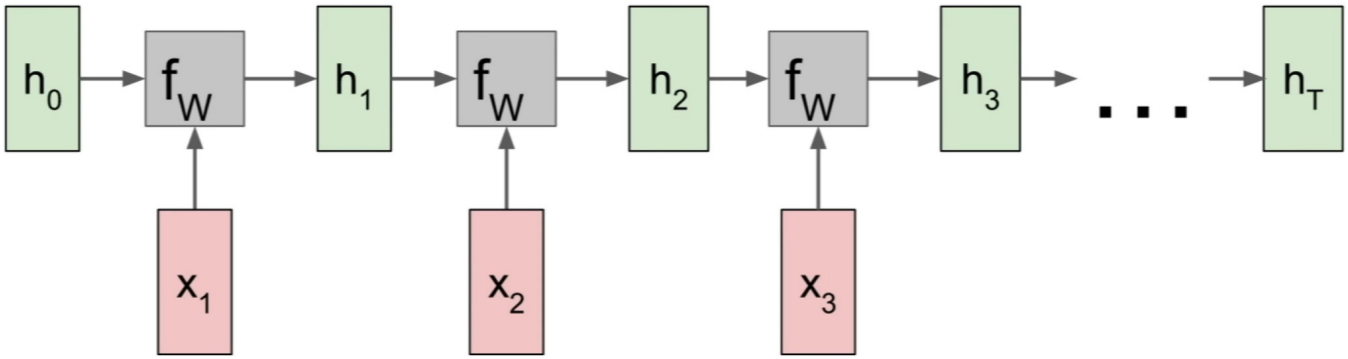
RNN: Computational Graph



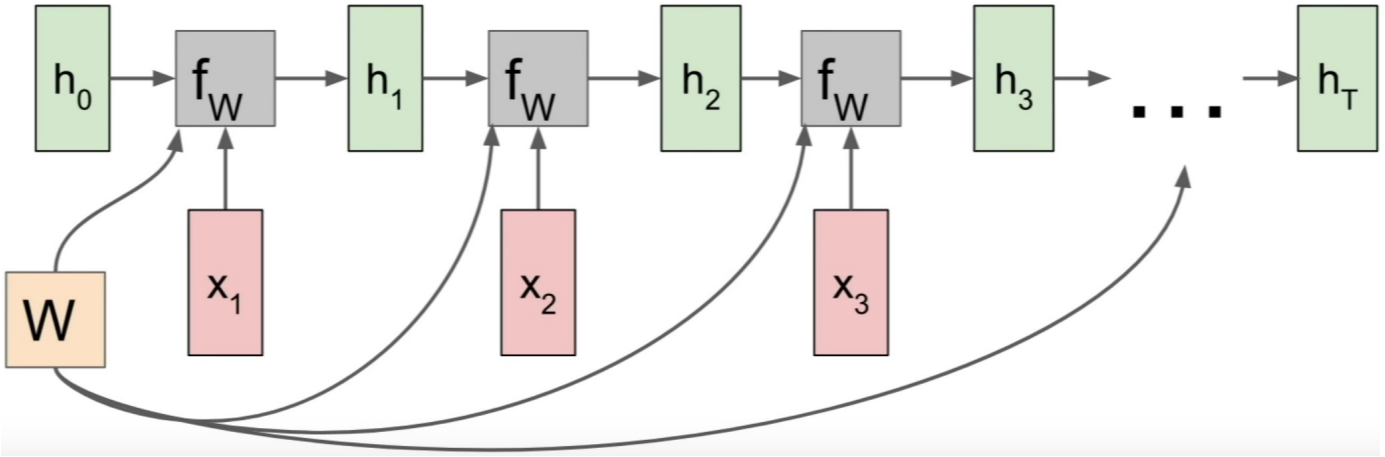
RNN: Computational Graph



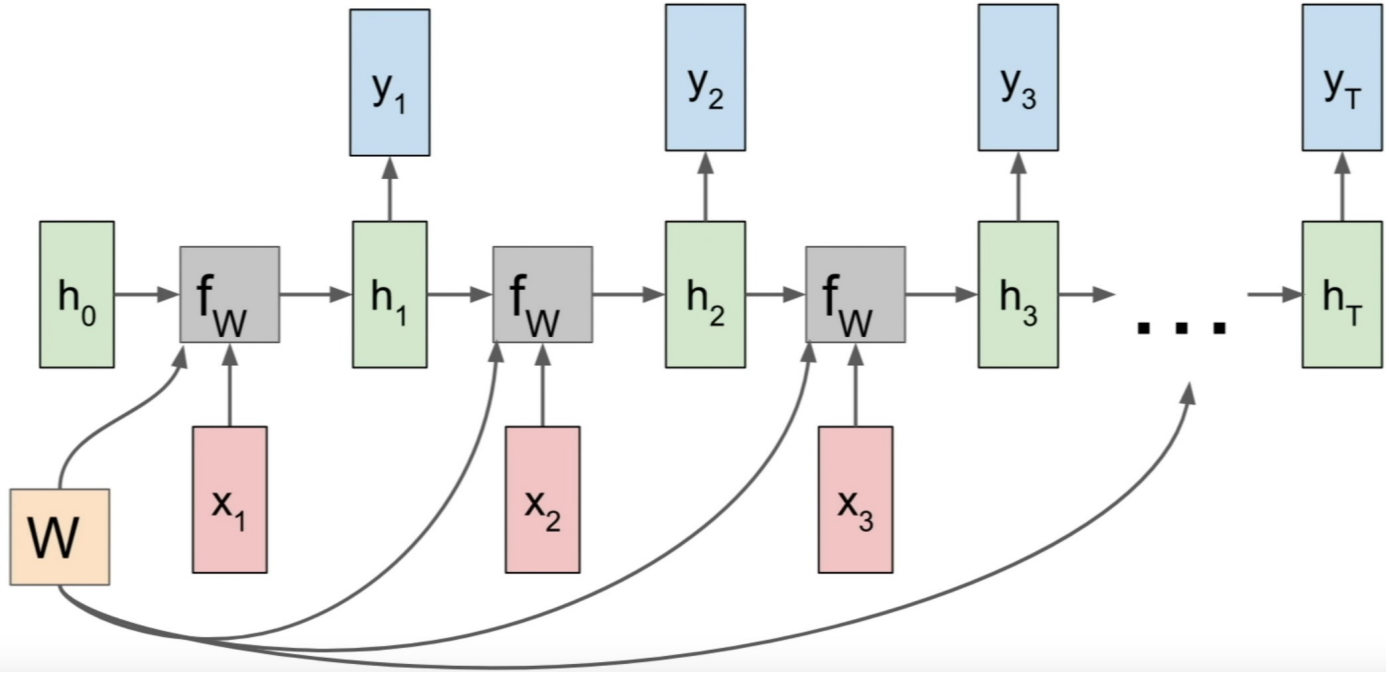
RNN: Computational Graph



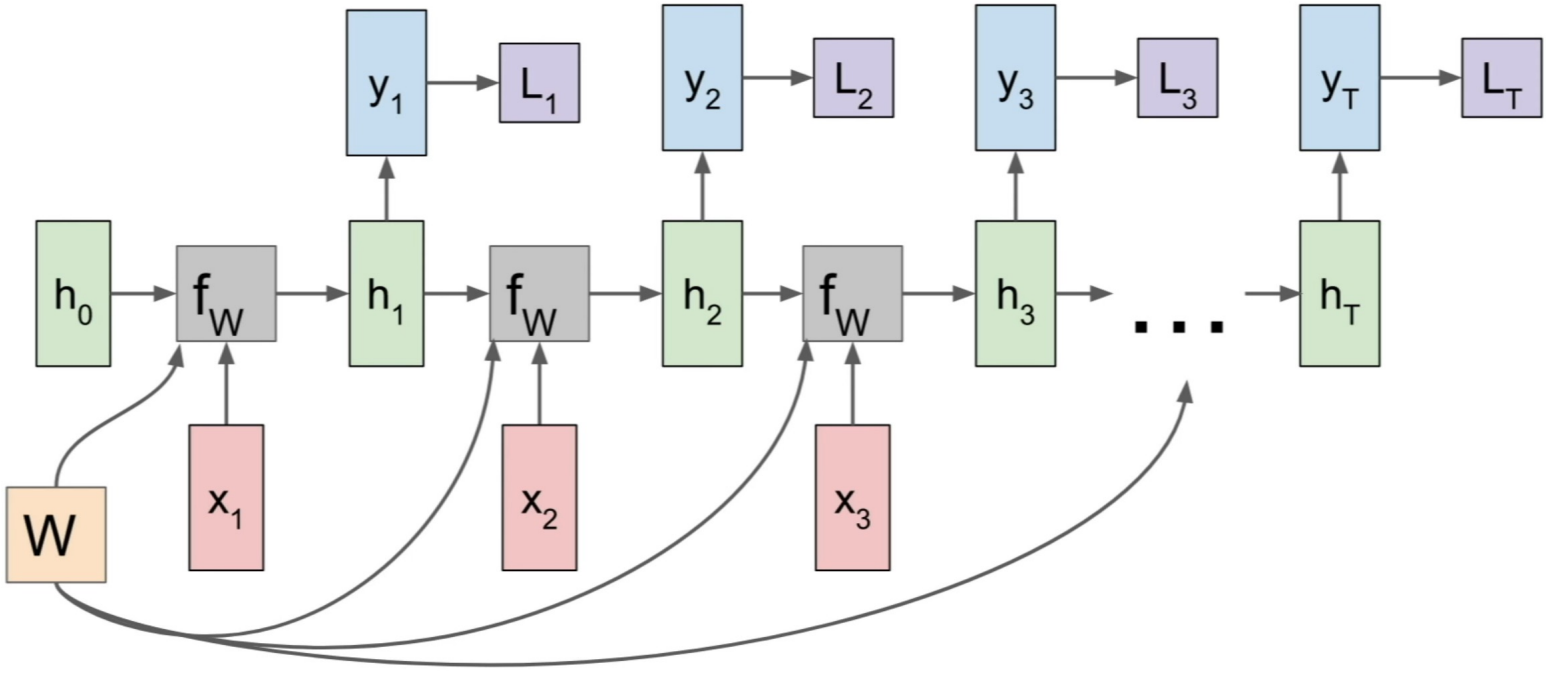
RNN: Computational Graph



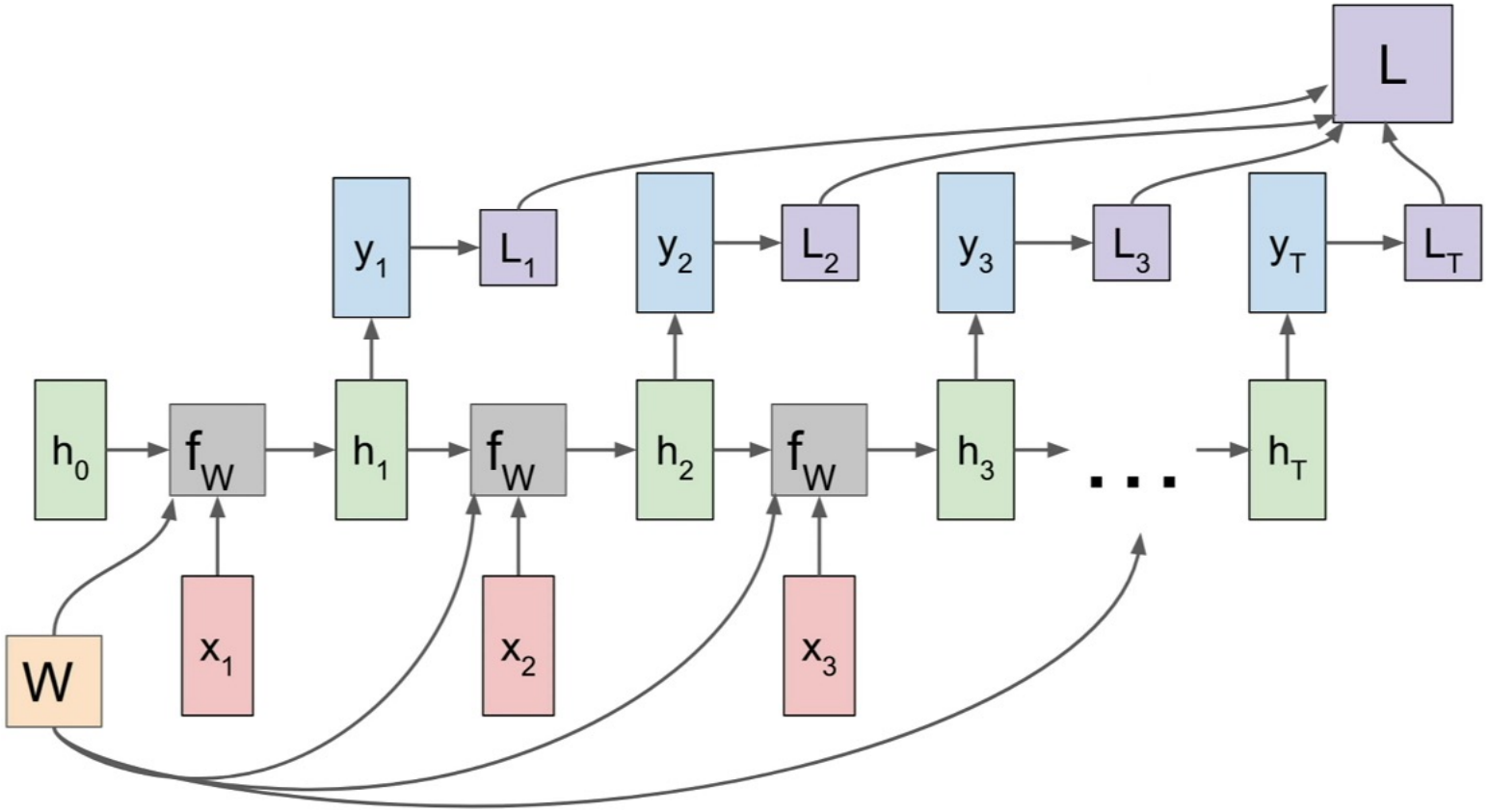
RNN: Computational Graph



RNN: Computational Graph

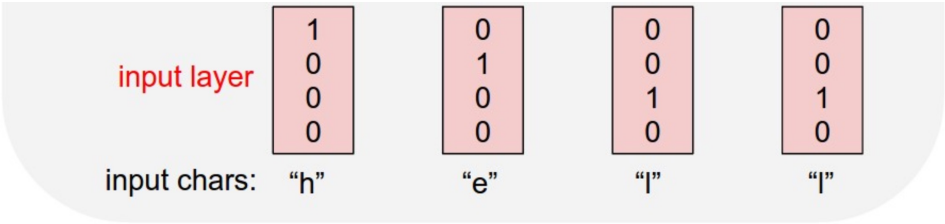


RNN: Computational Graph



Example: Language model

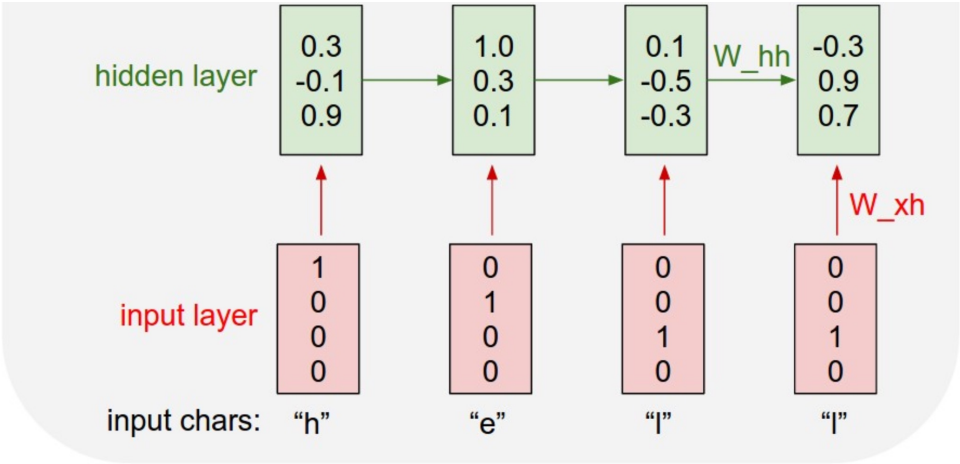
- Task: Predicting the next character
- Training sequence: “hello”
- Vocabulary:
  - [h, e, l, o]
- Encoding: One-hot



RNN: Predicting the next character

- Task: Predicting the next character
- Training sequence: “hello”
- Vocabulary:
  - [h, e, l, o]
- Encoding: One-hot
- Model:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

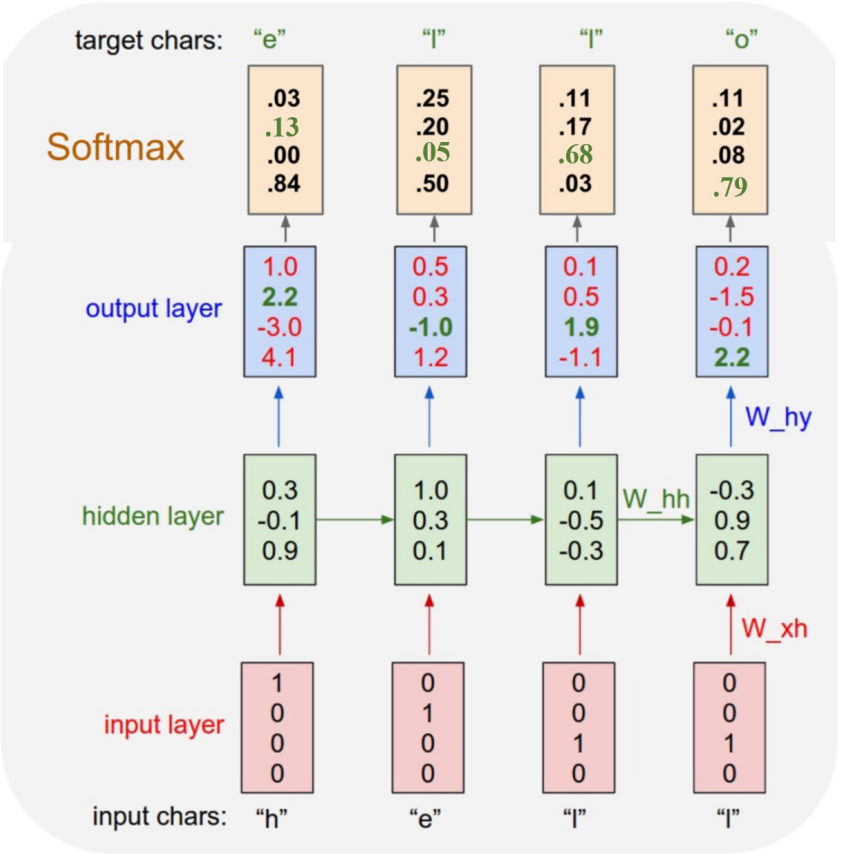


RNN: Predicting the next character

- Task: Predicting the next character
- Training sequence: “hello”
- Vocabulary:
  - [h, e, l, o]
- Encoding: One-hot
- Model:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

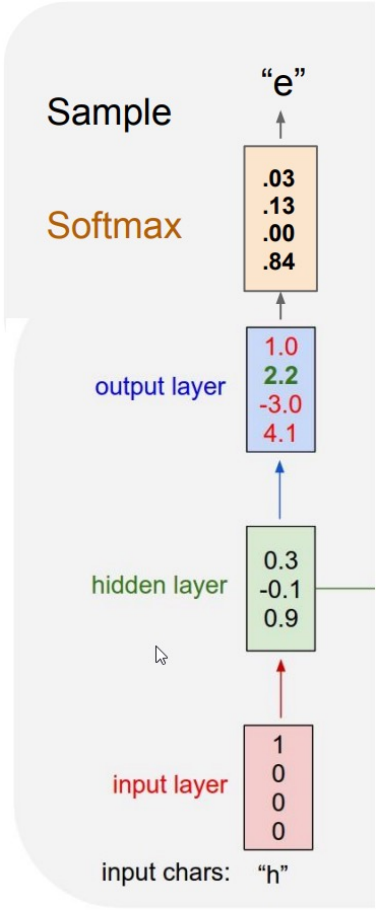
$$y_t = W_{hy}h_t$$



RNN: Predicting the next character

- Vocabulary:
  - [h, e, l, o]

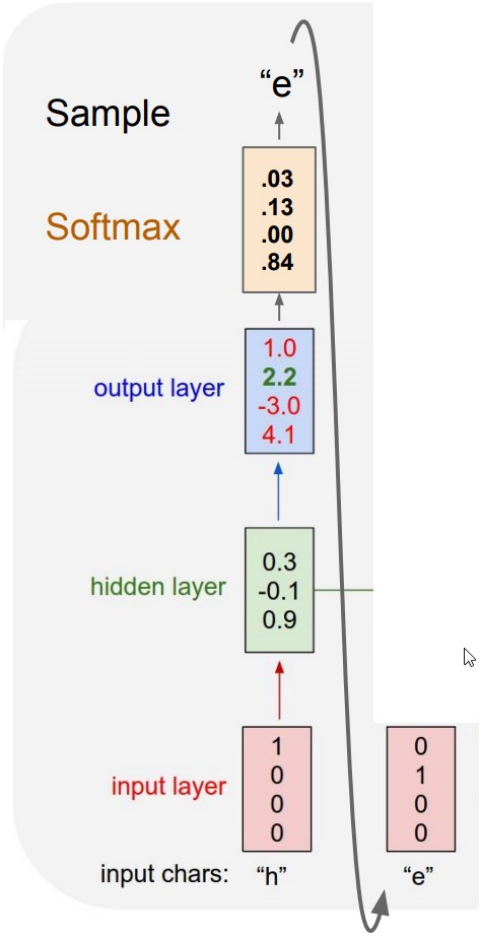
- At test time, we can sample from the model to synthesize new text.



RNN: Predicting the next character

- Vocabulary:
  - [h, e, l, o]

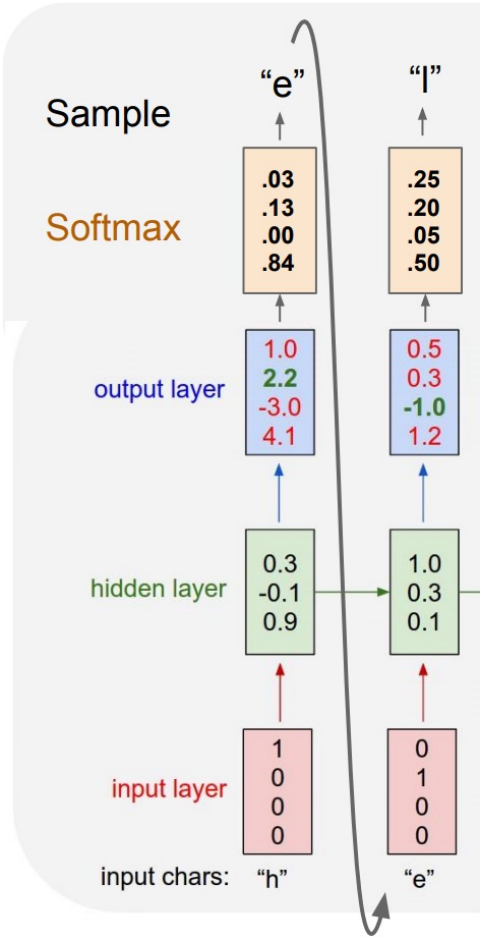
- At test time, we can sample from the model to synthesize new text.



RNN: Predicting the next character

- Vocabulary:
  - [h, e, l, o]

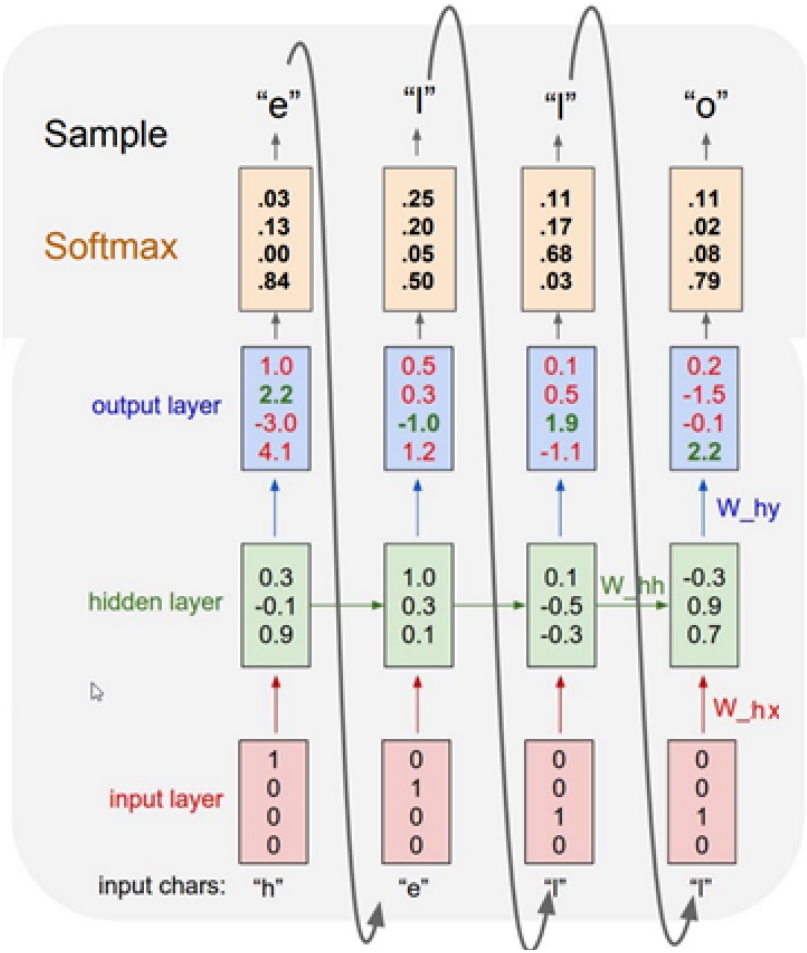
- At test time, we can sample from the model to synthesize new text.



RNN: Predicting the next character

- Vocabulary:
  - [h, e, l, o]

- At test time, we can sample from the model to synthesize new text.

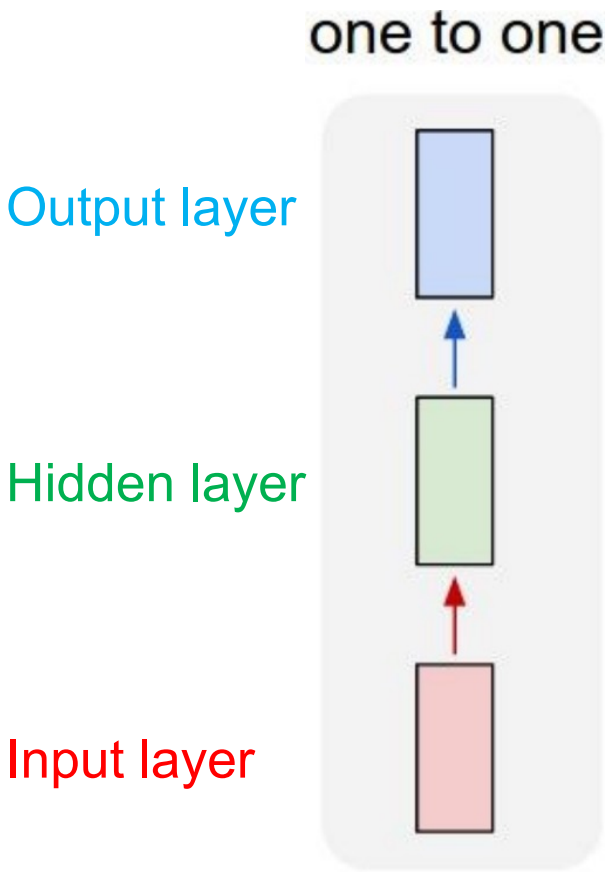


Progress

- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNNs
- Training Recurrent Neural Networks
- Challenges in training: Exploding or vanishing gradients
- LSTM and GRU
- Multilayer and bidirectional RNNs
- Image captioning (mandatory 2)

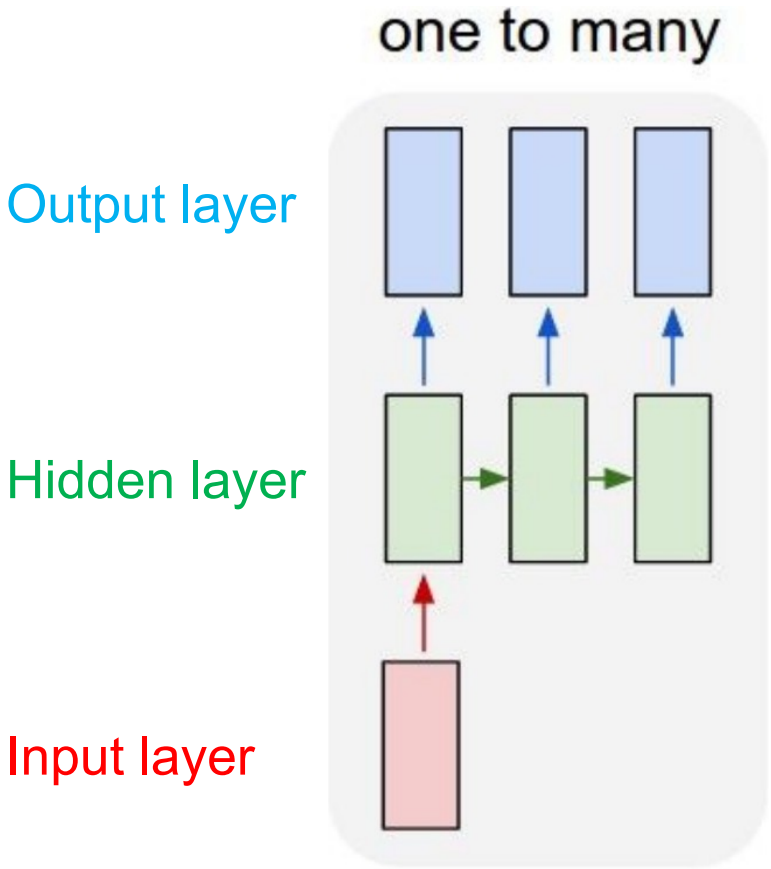
RNN: One-to-one

- Normal feed forward neural network
- One input - one output (classification or regression)



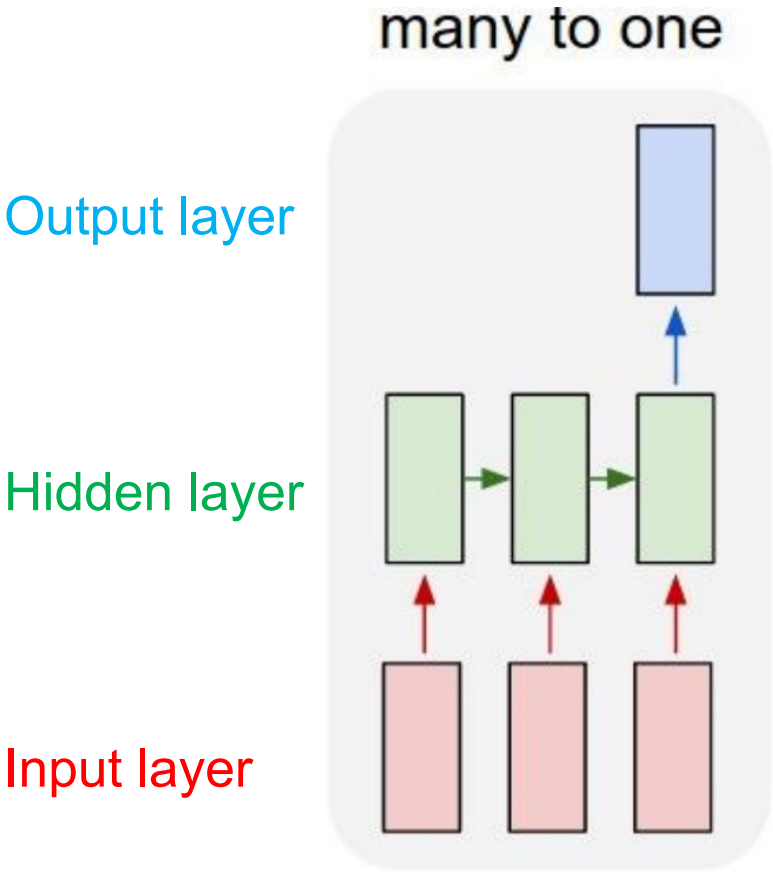
RNN: One-to-many

- Example: Image Captioning
- Input: One Image
- Output: A sequence of words



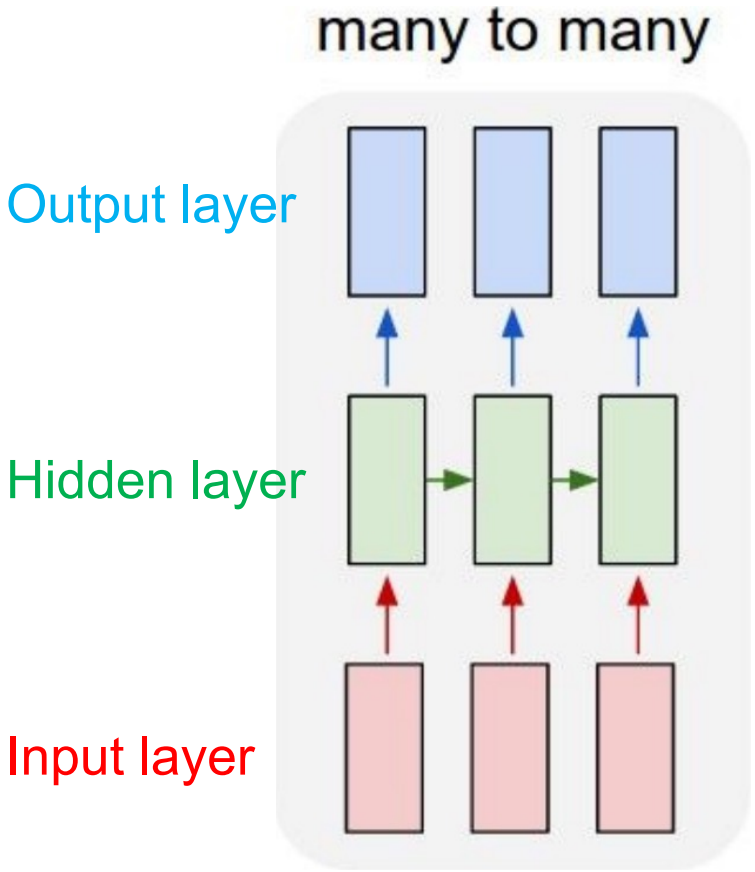
RNN: Many-to-one

- Example: Sentiment Analysis
- Input: A series of words / characters
- Output: A sentiment category



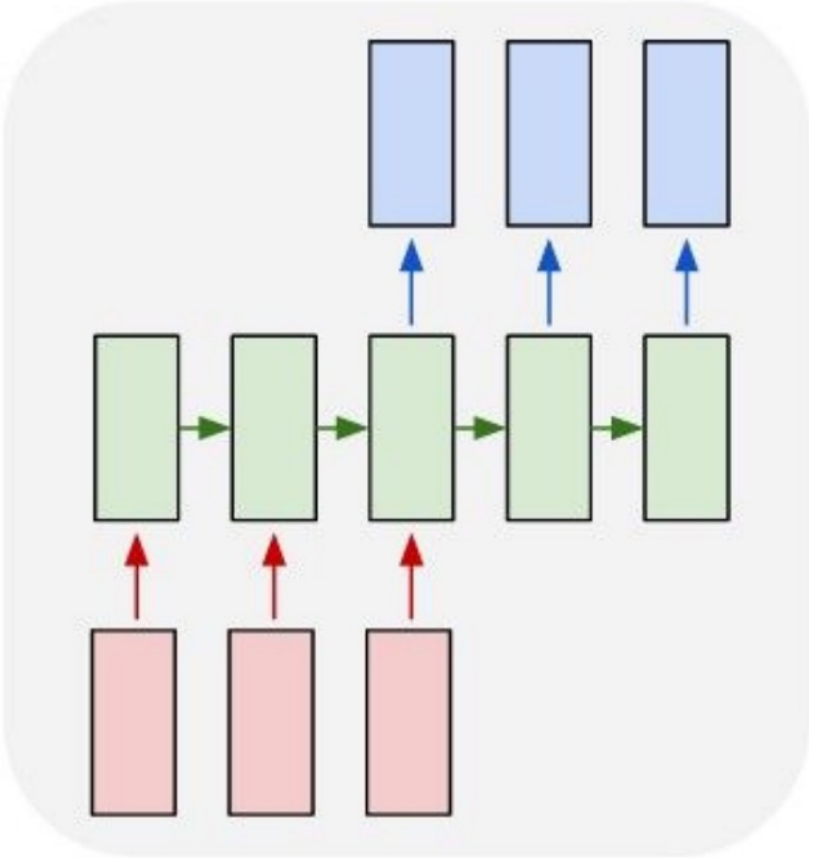
RNN: Many-to-many

- Example: Part-of-speech (POS) tagging / grammatical tagging
- Input: A sequence of words
- Output: A category for each word



RNN: Many-to-many (encoder-decoder)

- Example: Neural Machine Translation
- Note that the inputs and outputs can have different lengths here.

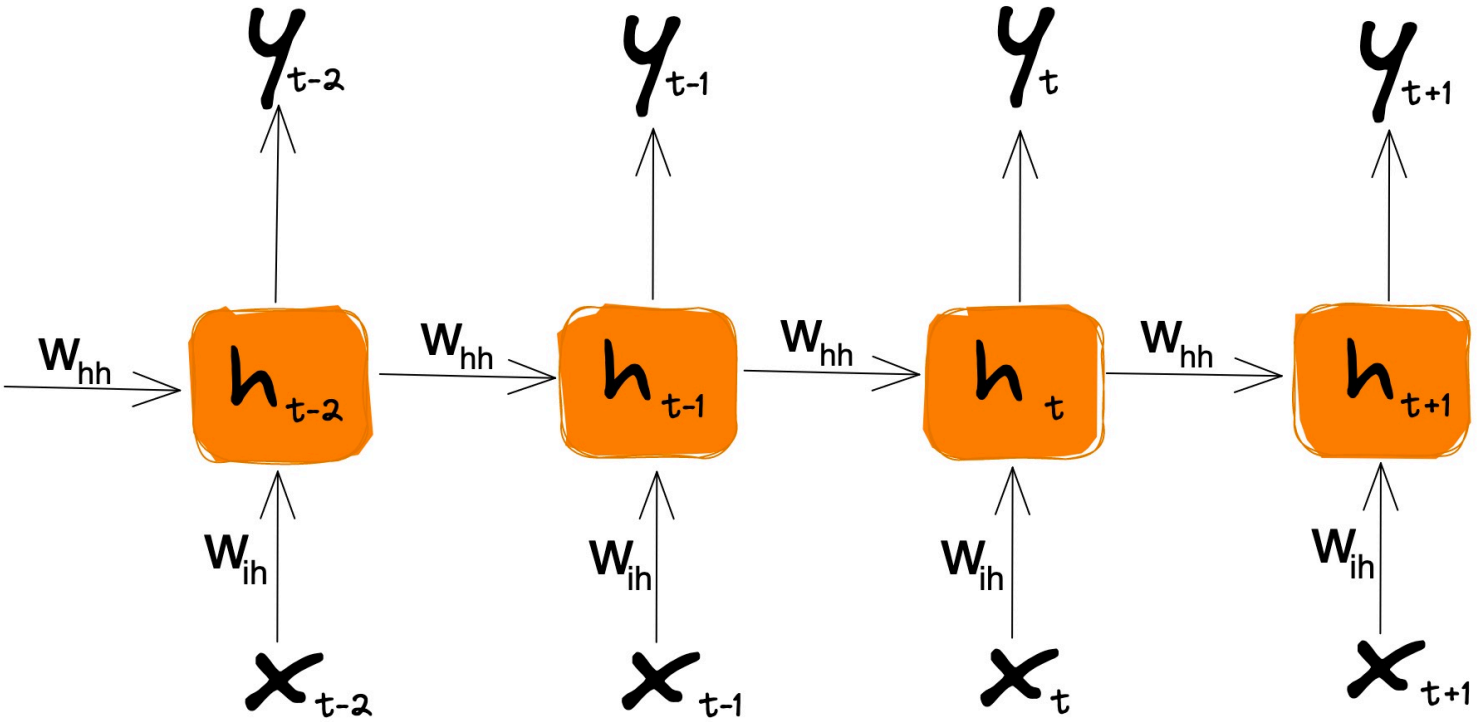


Progress

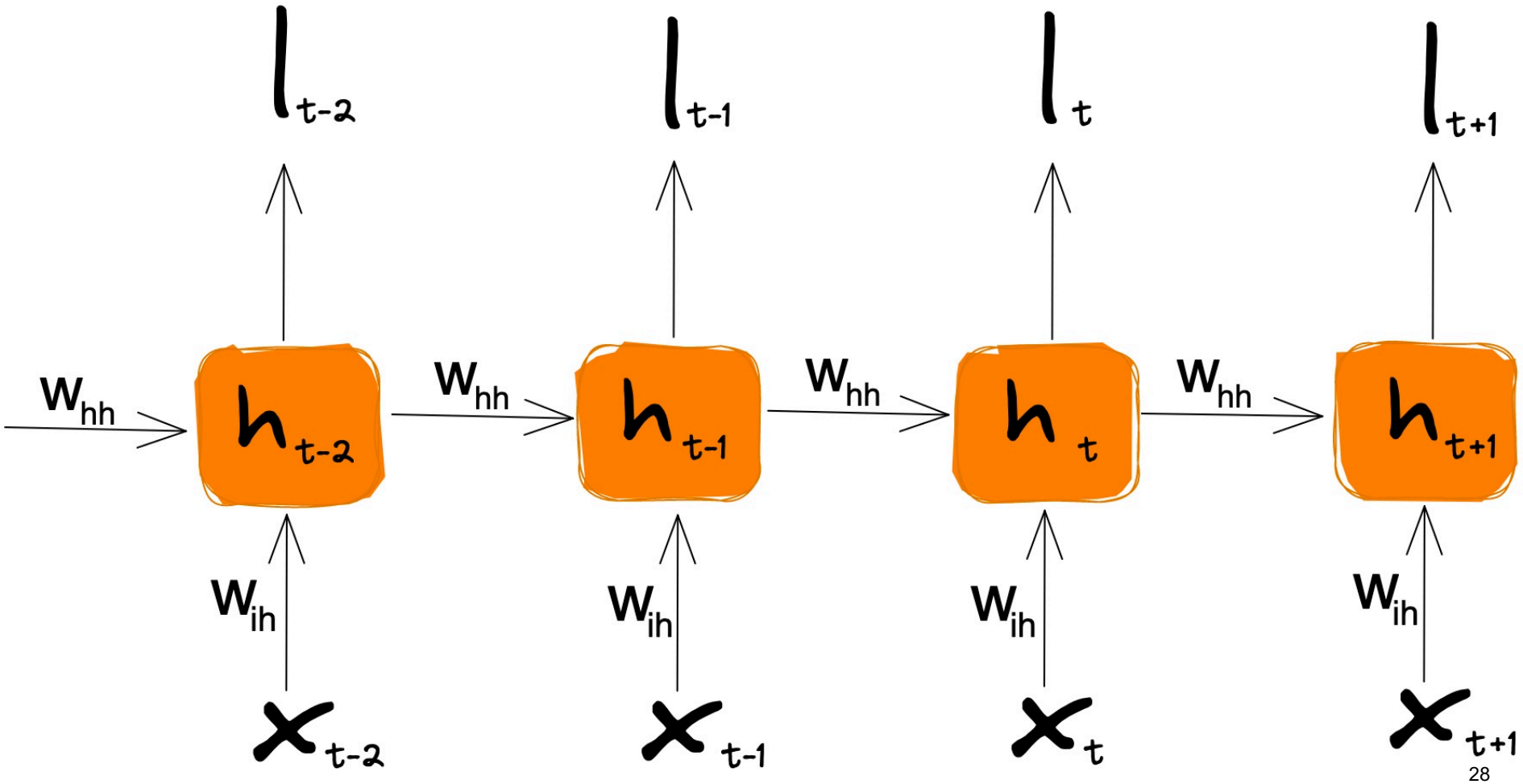
- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNNs
- Training Recurrent Neural Networks
- Challenges in training: Exploding or vanishing gradients
- LSTM and GRU
- Multilayer and bidirectional RNNs
- Image captioning (mandatory 2)

Training RNNs: Backpropagation Through Time

- RNNs are trained using Backpropagation through time (BPTT).
- BPTT steps:
  - Unroll the RNN in the time dimension (…t-2, t-1, t, t+1…).
  - Apply backpropagation on the unrolled RNN by treating it as a regular feed-forward network.

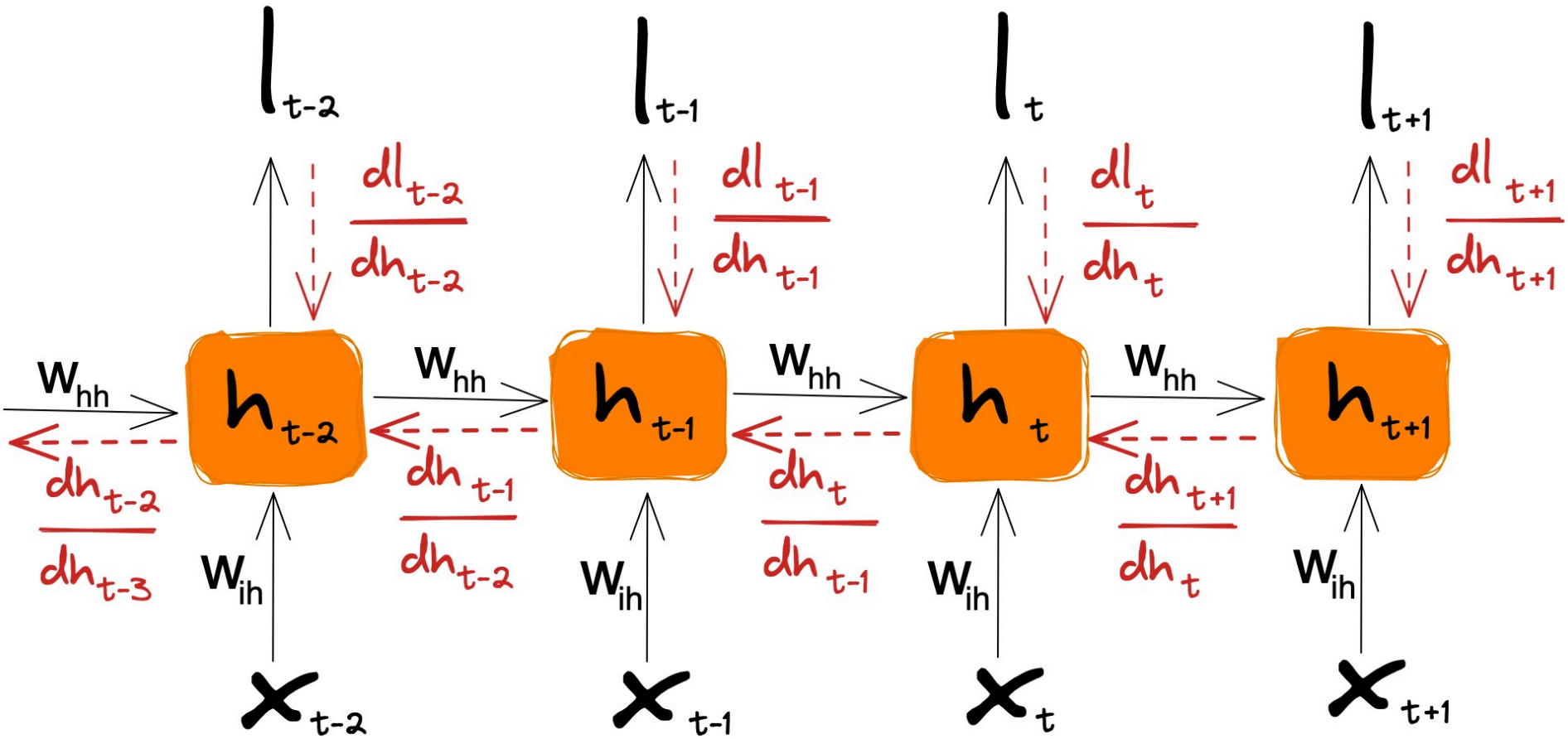


Training RNNs: Backpropagation Through Time



Training RNNs: Backpropagation Through Time

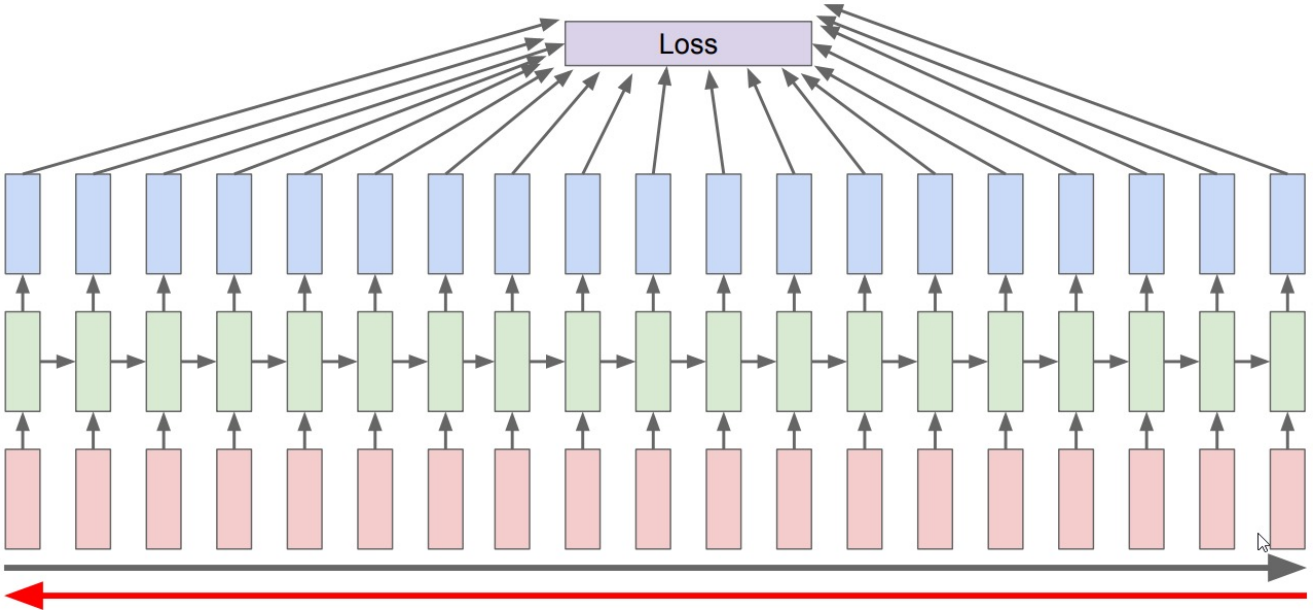
- The gradients are then backpropagated as shown through the red arrows.
- Notice how weights at a time step k receive gradient from all the time steps t where t > k, i.e., all the time steps ahead of k.



Training RNNs: Backpropagation Through Time

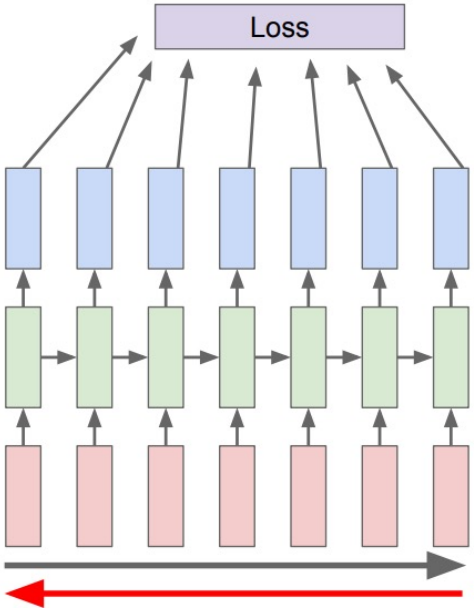
Challenge:

- To calculate gradients, you must keep the internal states of RNN in the memory until you get the backpropagating gradient.
- Then, what if you are reading a book with 100 million characters?



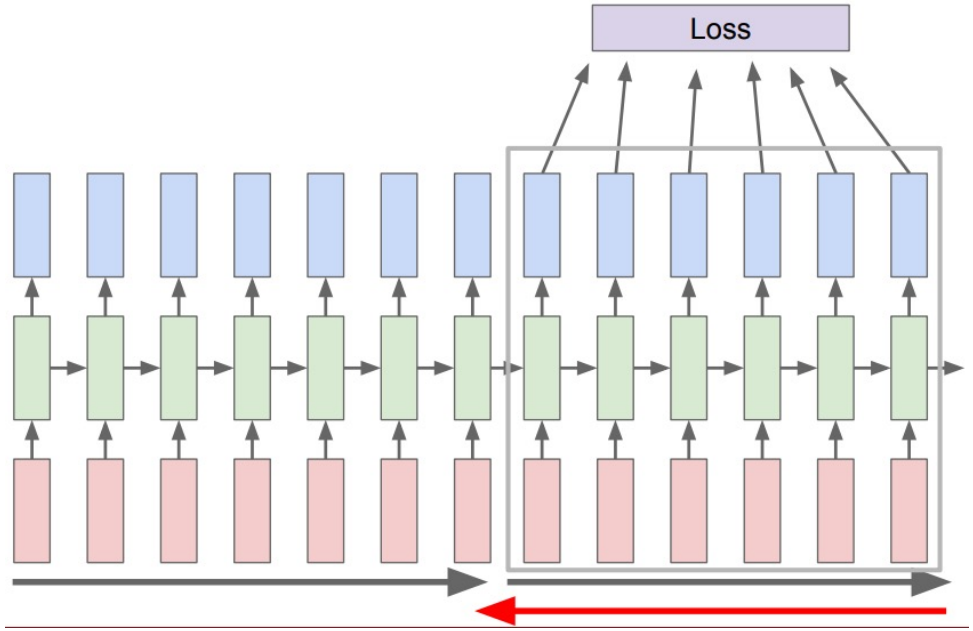
Training RNNs: Truncated Backpropagation Through Time

- The solution is to stop at some point and update the weight as if we were were done. This algorithm is called Truncated BPTT as we truncate the the unrolled RNN and apply BPTT on what we get.
- Here, BPTT is applied on whatever has been computed so far.



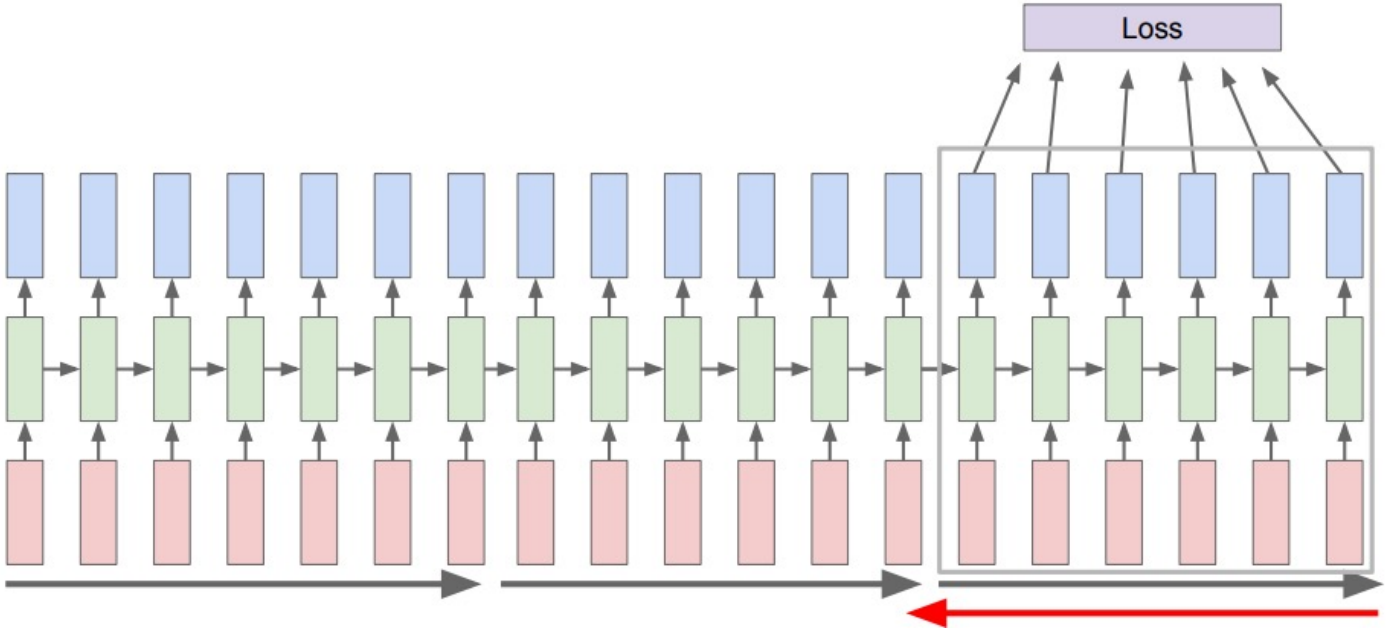
Training RNNs: Truncated Backpropagation Through Time

- We unroll the RNN further and again stop after a few time steps.
- The BPTT is now applied on the newly unrolled steps (and not on the previous ones).



Training RNNs: Truncated Backpropagation Through Time

- And this continues until has RNN has been completely unrolled for all the time steps.



Training RNNs: Truncated Backpropagation Through Time

- Advantages:
  - Reduces the memory requirement
  - Faster parameter updates

- Disadvantage:
  - Not able to capture dependencies longer than the truncated length.

Progress

- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNNs
- Training Recurrent Neural Networks
- Challenges in training: Exploding or vanishing gradients
- LSTM and GRU
- Multilayer and bidirectional RNNs
- Image captioning (mandatory 2)

Challenges in training RNNs

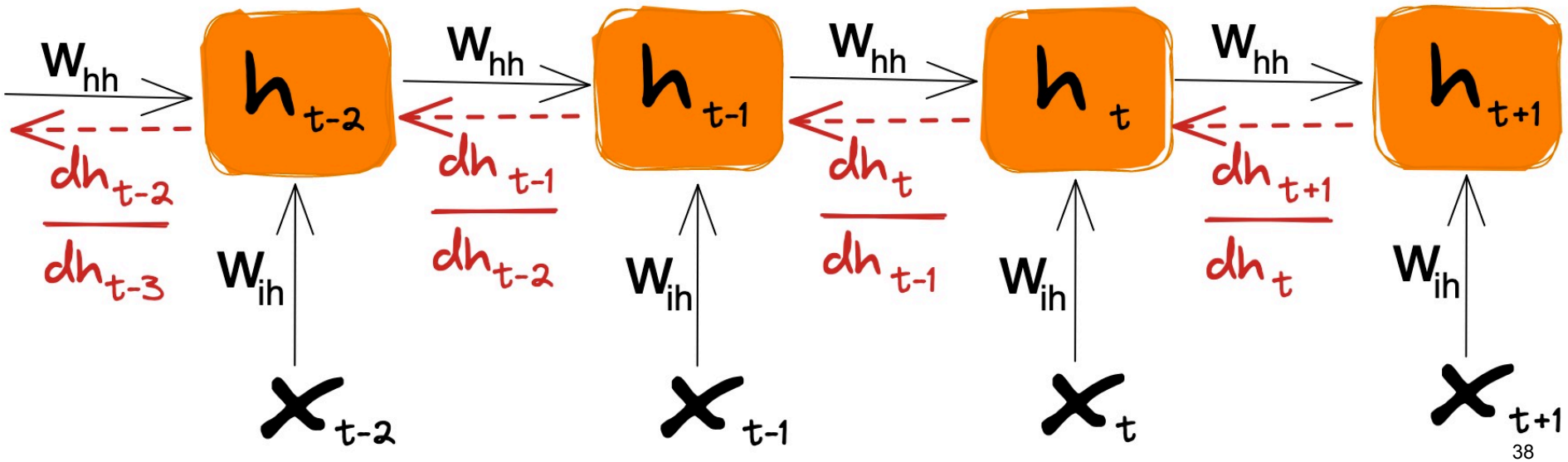
- Exploding gradients
- Vanishing gradients
- Preserve long-range dependencies

Challenges: Exploding or vanishing gradients

- The basic intuition behind vanishing or exploding gradients is pretty simple:
  - If you take a real number < 1 and keep multiplying it by itself, the result will approach 0.
    - E.g. $(0.7)^{10} \approx 0.028$ and $(0.7)^{100} \approx 0$
  - If you take a real number > 1 and keep multiplying it by itself, the result will approach ∞.
    - E.g. $(7)^{10} = 282475249$ and $(7)^{100} \approx 3.23 \times 10^{84}$

Challenges: Exploding or vanishing gradients

- Notice how gradients pass through w_{hh} to go from one time step to the previous one.
- Let's see mathematically how gradients for $\mathbf{w_{hh}}$ flow from a *time step* ***t*** to a *time step* ***k***.



Challenges: Exploding or vanishing gradients

- Let the hidden state at time step t be h_t and loss be L.
- Recall the equation for calculating h_t is

$$h_t = W_{hh}\sigma(h_{t-1}) + W_{ih}x_t + b \quad (1)$$

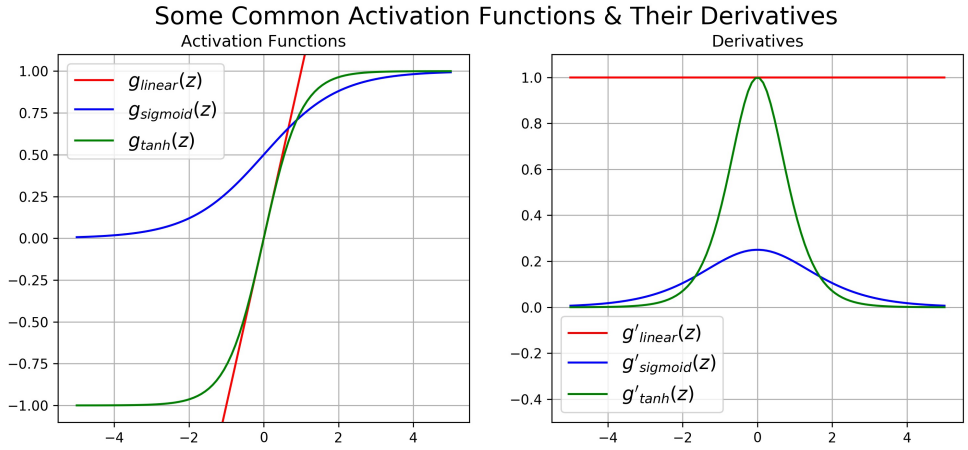
- We're interested in calculating the gradients received by W_{hh} at time step k. Using chain rule, we have
- $$\frac{dL}{dW_{hh}^{(k)}} = \frac{dL}{dh_t}\frac{dh_t}{dh_{t-1}}\frac{dh_{t-1}}{dh_{t-2}}\cdots\frac{dh_{k+1}}{dh_k}\frac{dh_k}{dW_{hh}^{(k)}} \quad (2)$$
- $$\frac{dL}{dW_{hh}^{(k)}} = \frac{dL}{dh_t}\prod_{i=k+1}^{t}\frac{dh_i}{dh_{i-1}}\frac{dh_k}{dW_{hh}^{(k)}} \quad (3)$$
- $$\prod_{i=k+1}^{t}\frac{dh_i}{dh_{i-1}} = \prod_{i=k+1}^{t}\mathrm{diag}(\sigma'(h_{i-1}))W_{hh}^{T} \quad (4)$$
  - where σ′ is the derivative of σ, ***diag*** converts vector to a diagonal matrix and superscript T on W represents transpose of W.

Challenges: Exploding or vanishing gradients

- $\frac{dL}{dW_{hh}^{(k)}} = \frac{dL}{dh_t} \prod_{i=k+1}^{t} \frac{dh_i}{dh_{i-1}} \frac{dh_k}{dW_{hh}^{(k)}}$ (3)
- $\prod_{i=k+1}^{t} \frac{dh_i}{dh_{i-1}} = \prod_{i=k+1}^{t} diag(\sigma'(h_{i-1}))W_{hh}^{T}$ (4)
- Equations (3) and (4) tell us that the gradient from the loss function at time step t is multiplied by σ' and the matrix W_{hh} (t-k) times to get the gradients received by W_{hh} at time step k.
- For simplicity, let's replace W_{hh}^{T} with W and $diag(\sigma'(h_{i-1}))$ with σ' (we ignore h_{i-1} for now).
- The product (equation (4)) then becomes $\sigma'^{n}W^{n}$ for BPTT through n time steps.
- Let's analyse both the terms in the product.

Challenges: Exploding or vanishing gradients

- σ'^{n} will lead to vanishing gradients if σ' < 1 for a lot of h_{i-1} values and exploding gradients if it is > 1 (recall the basic intuition).
- For σ = ReLU, the derivative has no upper bound so the gradients may potentially explode depending on the values of logits. Logits could also be < 1 in which case the gradients would vanish.
- For σ = sigmoid or tanh, the derivatives have an upper bound of 0.25 and 1 respectively so they can't lead to exploding gradients but can help gradients vanish.



Challenges: Exploding or vanishing gradients

- To analyse W, we will assume that its eigendecomposition exists. You should be familiar with it from MAT1110 / MAT1120.
- Decompose square matrix W as $W = U\Sigma U^{-1}$ where U's columns are the eigenvectors of W and Σ is a diagonal matrix whose diagonal elements are the corresponding eigenvalues.

Challenges: Exploding or vanishing gradients

- $W^n = (U\Sigma U^{-1})(U\Sigma U^{-1})(U\Sigma U^{-1})...(U\Sigma U^{-1})$
- $W^n = (U\Sigma I\Sigma I\Sigma I...\Sigma U^{-1})$ because $UU^{-1} = I$ (I=Identity matrix) by definition
- $W^n = (U\Sigma\Sigma\Sigma...\Sigma U^{-1}) = U\Sigma^n U^{-1}$
- What this means is that the growth or decay of W's multiplication with itself is dictated by its eigenvalues.
- If the *spectral radius* (magnitude of the eigenvalue with the highest absolute value) of W is > 1, i.e., **max(np.abs(Σ)) > 1** then the matrix will explode and if it's < 1, the matrix will decay.

Challenges: Exploding or vanishing gradients

$$W^n = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix}^n \begin{bmatrix} d_1 & e_1 & f_1 \\ d_2 & e_2 & f_2 \\ d_3 & e_3 & f_3 \end{bmatrix}$$

$$W^n = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{bmatrix} \lambda_1^n & 0 & 0 \\ 0 & \lambda_2^n & 0 \\ 0 & 0 & \lambda_3^n \end{bmatrix} \begin{bmatrix} d_1 & e_1 & f_1 \\ d_2 & e_2 & f_2 \\ d_3 & e_3 & f_3 \end{bmatrix}$$

$$W^n = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{bmatrix} \lambda_1^n d_1 & \lambda_1^n e_1 & \lambda_1^n f_1 \\ \lambda_2^n d_2 & \lambda_2^n e_2 & \lambda_2^n f_2 \\ \lambda_3^n d_3 & \lambda_3^n e_3 & \lambda_3^n f_3 \end{bmatrix}$$

If only $|\lambda_1| > 1$ then $\lambda_2^n = \lambda_3^n = 0$

Challenges: Exploding or vanishing gradients

$$W^n = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{bmatrix} \lambda_1^n d_1 & \lambda_1^n e_1 & \lambda_1^n f_1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$W^n = \begin{bmatrix} \lambda_1^n d_1 a_1 & \lambda_1^n e_1 a_1 & \lambda_1^n f_1 a_1 \\ \lambda_1^n d_1 a_2 & \lambda_1^n d_1 a_2 & \lambda_1^n d_1 a_2 \\ \lambda_1^n d_1 a_3 & \lambda_1^n d_1 a_3 & \lambda_1^n d_1 a_3 \end{bmatrix}$$

All the gradients explode even if just one eigenvalue's magnitude is greater than one.

Exploding gradients: Gradient clipping

- To avoid exploding gradients, clip them if they are larger than a threshold.
- Two common approaches: clipping-by-value or clipping-by-norm.
- Let **g** be a single gradient and **gg** be a vector of gradients
- **Clipping-by-value**: If g > threshold, **g = threshold**.
- **Clipping-by-norm**: If l^2norm(gg) > threshold, for each g in gg apply

$$g = g * \frac{\text{threshold}}{l^2\text{norm}(gg)}$$

Orthogonal Initialisation

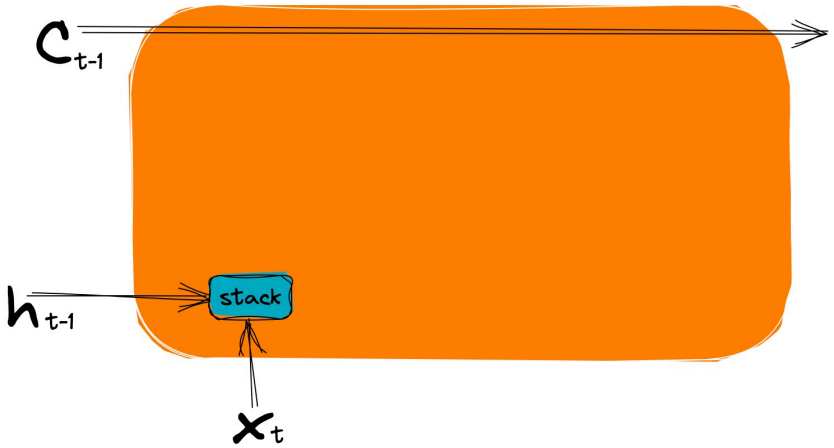
- One way to partially prevent gradients from vanishing or exploding in the beginning of the training is to initialise the matrix with orthogonal matrices.
- Orthogonal matrices have a nice property that all their eigenvalues are 1. Therefore, W^n neither explodes nor decays.
- But as BPTT updates the weight matrices over and over, the matrices might not remain orthogonal.

Progress

- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNNs
- Training Recurrent Neural Networks
- Challenges in training: Exploding or vanishing gradients
- LSTM and GRU
- Multilayer and bidirectional RNNs
- Image captioning (mandatory 2)

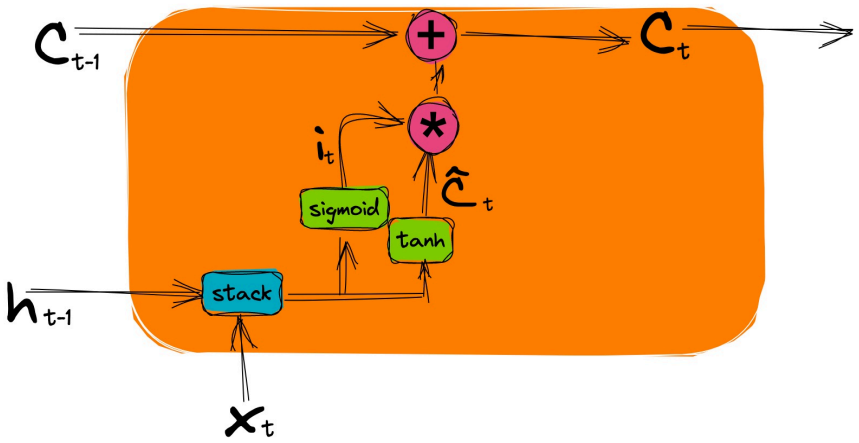
LSTM Networks

- One way to prevent vanishing gradients and preserve long-term dependencies is to use LSTM (Long Short Term Memory) networks.
- The main idea in LSTM's design is to maintain a cell state C which serves as memory. Each node (an LSTM cell) passes this cell state to itself across all the time steps.
- The other components are mostly gates that modify the cell state and generate output of the cell using the cell state.



LSTM Networks: Input Gate

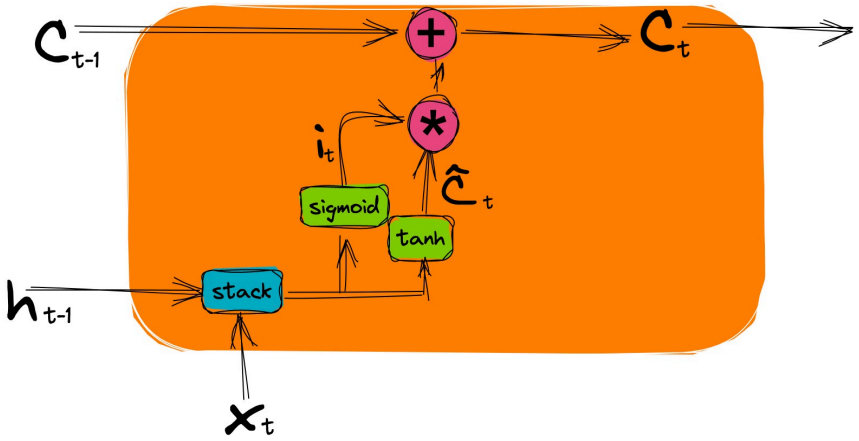
- Input gate controls what fraction of the new memory is added to the old one.
- To modify the cell state, first a new candidate cell state is generated as $\hat{C} = \tanh(W_c[h_{t-1}x_t] + b_c)$
- Then the input gate decides how much of this new candidate should be added to the memory. Input gate's value is calculated as $i_t = \text{sigmoid}(W_i[h_{t-1}x_t] + b_i)$



LSTM Networks: Output Gate

- The cell state of LSTM is internal. Other layers and cells don't have access to it. It's NOT cell's output.
- To get the output, first a candidate output is generated by simply passing the cell state through tanh: $\hat{\mathbf{h}}_\mathbf{t} = \tanh(C_t)$
- Then the output gate decides how much of this value are we going to output. Output gate's value is calculated as

$$\mathbf{o}_\mathbf{t} = \mathbf{sigmoid}(\mathbf{W}_\mathbf{o}[\mathbf{h}_{\mathbf{t}-1}\mathbf{x}_\mathbf{t}] + \mathbf{b}_\mathbf{o})$$

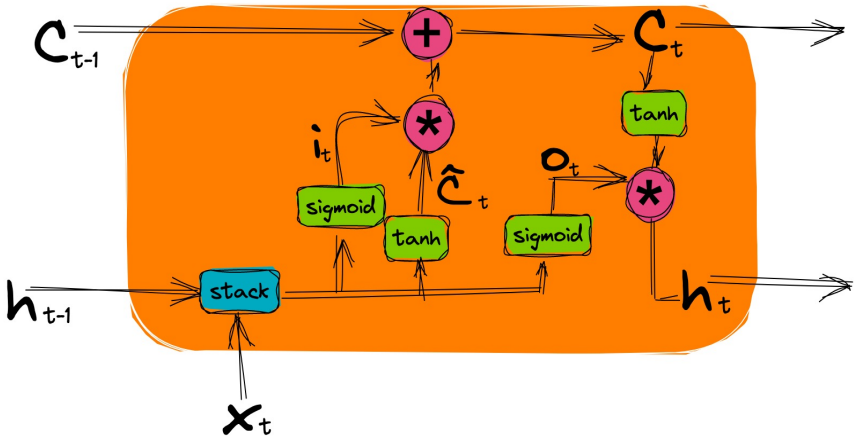


LSTM Networks: Output Gate

- $\hat{h}_t = \tanh(C_t)$
- $o_t = \text{sigmoid}(W_o[h_{t-1}x_t] + b_o)$

- The output of the cell is then

$$h_t = o_t \odot \hat{h}_t = o_t \odot \tanh(C_t)$$

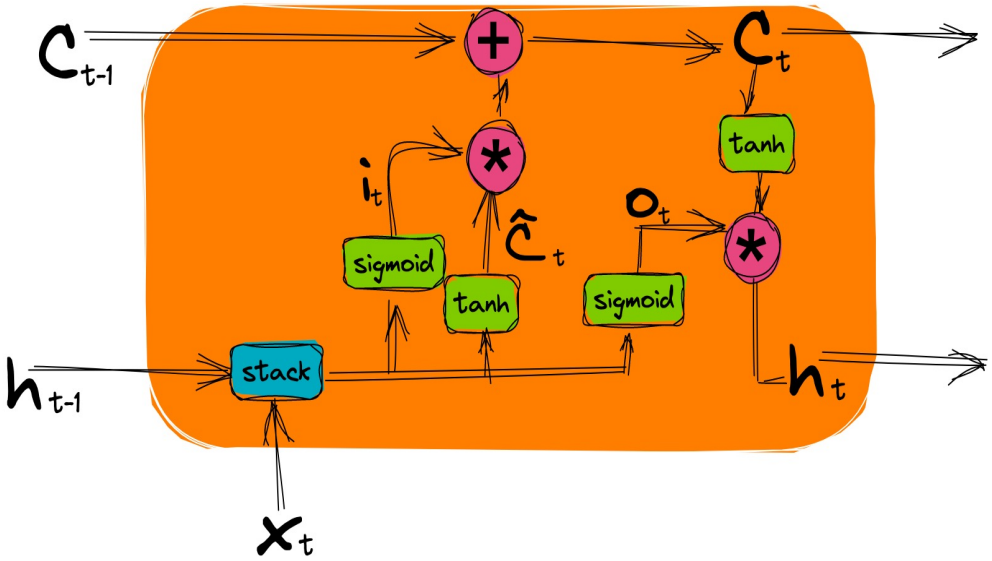


LSTM Networks: First (original) Formulation

- In the first LSTM paper, the cell state was updated as

$$C_t = C_{t-1} + i_t \odot \hat{C}$$

- This formulation ensured that the gradients could flow through the cell state without vanishing or exploding because $\frac{dC_t}{dC_{t-1}} = 1$
- So, $\frac{dC_t}{dC_k} = \frac{dL}{dh_t}\frac{dh_t}{dC_t}\frac{dC_t}{dC_{t-1}}\frac{dC_{t-1}}{dC_{t-2}}\cdots\frac{dC_{k+1}}{dC_k} = \frac{dL}{dh_t}\frac{dh_t}{dC_t} * 1 * 1 \cdots * 1$



LSTM Networks: Forget Gate

- LSTM's formulation with the forget gate(added a few years later) is the standard formulation.
- Forget gate decides how of the old cell state to erase. This allows LSTM to forget whatever it finds useless. Just like other gates, forget gate is also calculated as $\mathbf{f_t = sigmoid(W_f[h_{t-1}x_t] + b_f)}$
- The cell state in this standard formulation is then calculated as

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}$$

- The derivative of the cell state is not 1 anymore but practically LSTM with a forget gate performs better than the one without. In fact, forget gate has been found to be the most important gate in LSTM.
  - Greff, Klaus, et al. "LSTM: A search space odyssey." *IEEE transactions on neural networks and learning systems*
  - Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures." *ICML 2015*

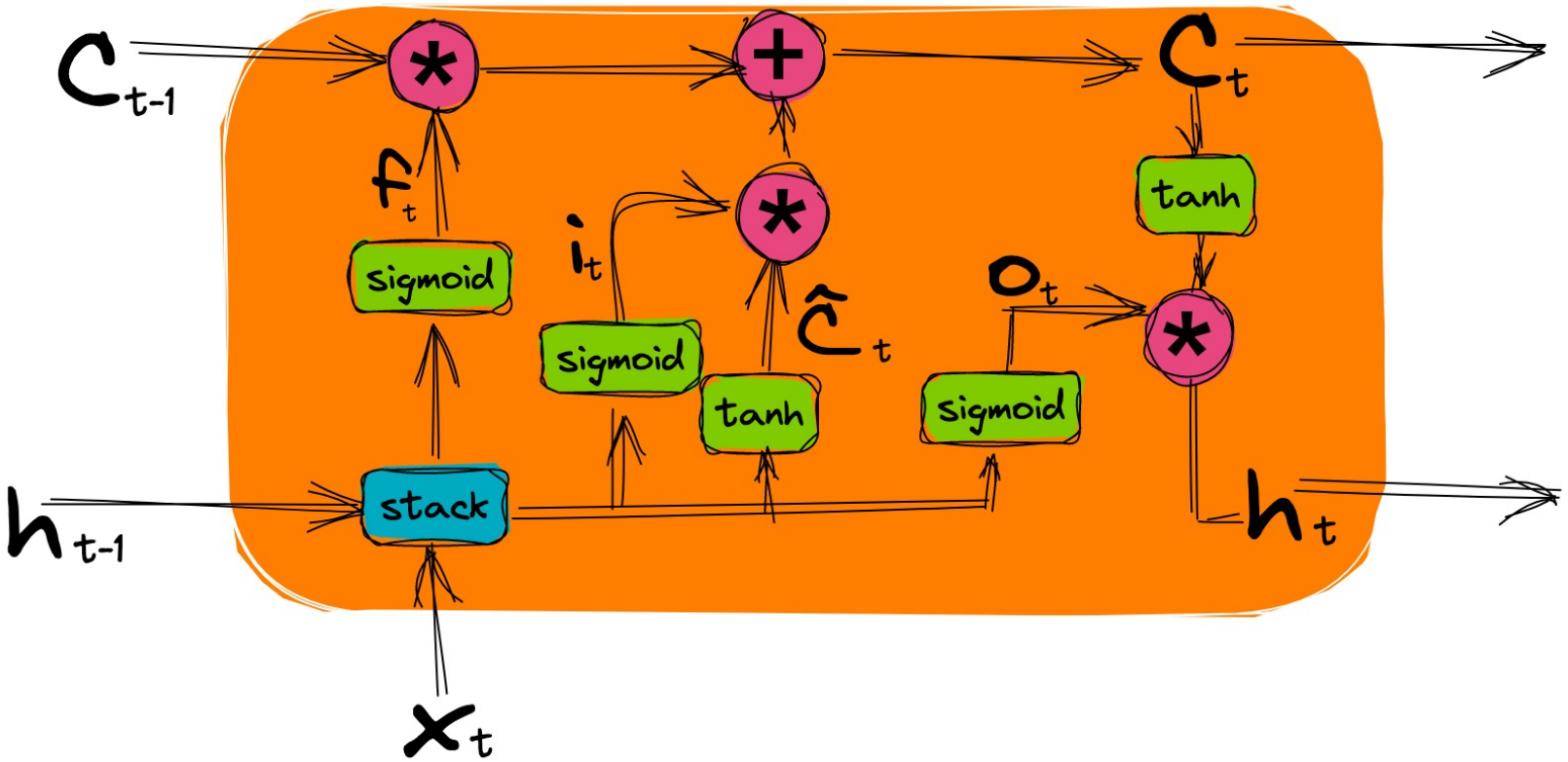
LSTM Networks: Forget Gate Initialisation

- Since forget gate can interrupt gradient flow through the cell state, a common trick in initialising forget gate is to set its bias to a positive value greater than 1.
- This results in forget gate being open (having value 1) and gradients flowing through the cell state uninterrupted initially.

Standard LSTM Network

- $i_t = \text{sigmoid}(W_i[h_{t-1}x_t] + b_i)$
- $o_t = \text{sigmoid}(W_o[h_{t-1}x_t] + b_o)$
- $f_t = \text{sigmoid}(W_f[h_{t-1}x_t] + b_f)$
- $\hat{C} = \tanh(W_c[h_{t-1}x_t] + b_c)$

- $C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}$
- $h_t = o_t \odot \tanh(C_t)$



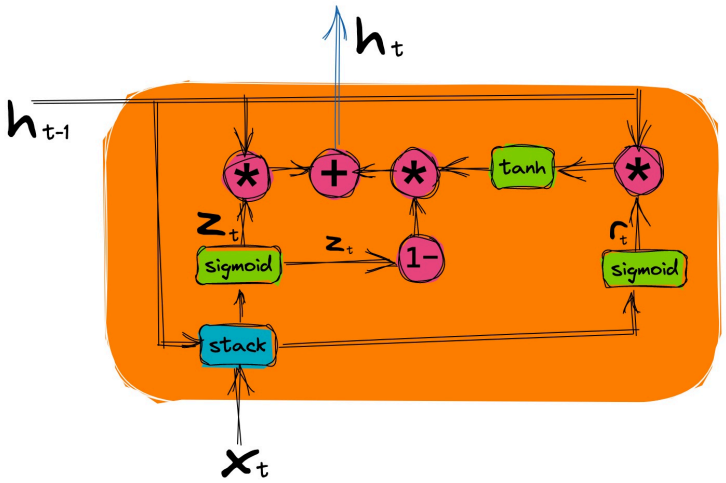
LSTM Success Highlights

- 2015: Google started using LSTMs for speech recognition. Google blog link
- 2016: Google started using LSTMs for translation. Google blog link
- 2017: Facebook started using LSTMs for translation. Facebook blog link
- 2017-2018: OpenAI's OpenAI Five's heroes in Dota2 were LSTM networks. OpenAI blog link
- 2019: DeepMind's AlphaStar for Starcraft II uses LSTM at core. DeepMind blog link
- And many more.

GRU: Gated Recurrent Unit

- It's a variant of LSTM. It's simpler and smaller (less parameters) than LSTM.
- It combines input gate and forget gate into a single update gate.
- It also merges the cell state **C** and the hidden state **h**.

- Update gate: $z_t = \text{sigmoid}(W_z[h_{t-1}x_t] + b_z)$
- Reset gate: $r_t = \text{sigmoid}(W_r[h_{t-1}x_t] + b_r)$
- Proposed activation: $\hat{h}_t = \tanh(W_h[(r_t \odot h_{t-1})x_t] + b_h)$
- Final output activation: $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \hat{h}_t$

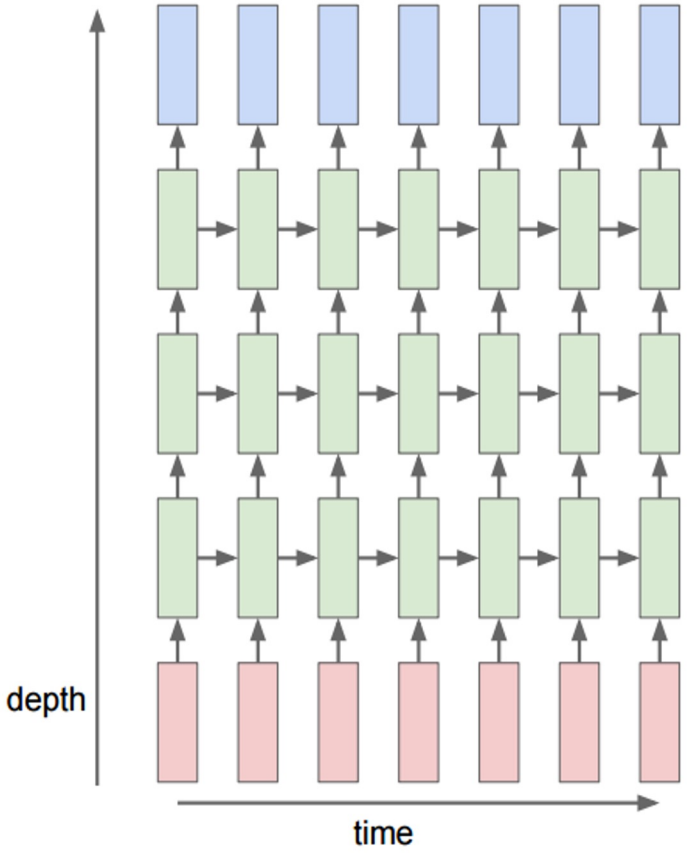


Progress

- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNNs
- Training Recurrent Neural Networks
- Challenges in training: Exploding or vanishing gradients
- LSTM and GRU
- Multilayer and bidirectional RNNs
- Image captioning (mandatory 2)

Multi-layer Recurrent Neural Networks

- Multi-layer RNNs can be used to enhance model complexity.
- Stacking layers creates higher level feature representation.
- Normally, only 2 or 3 layers deep. More complex relationships are learnt in time.



Bidirectional RNNs

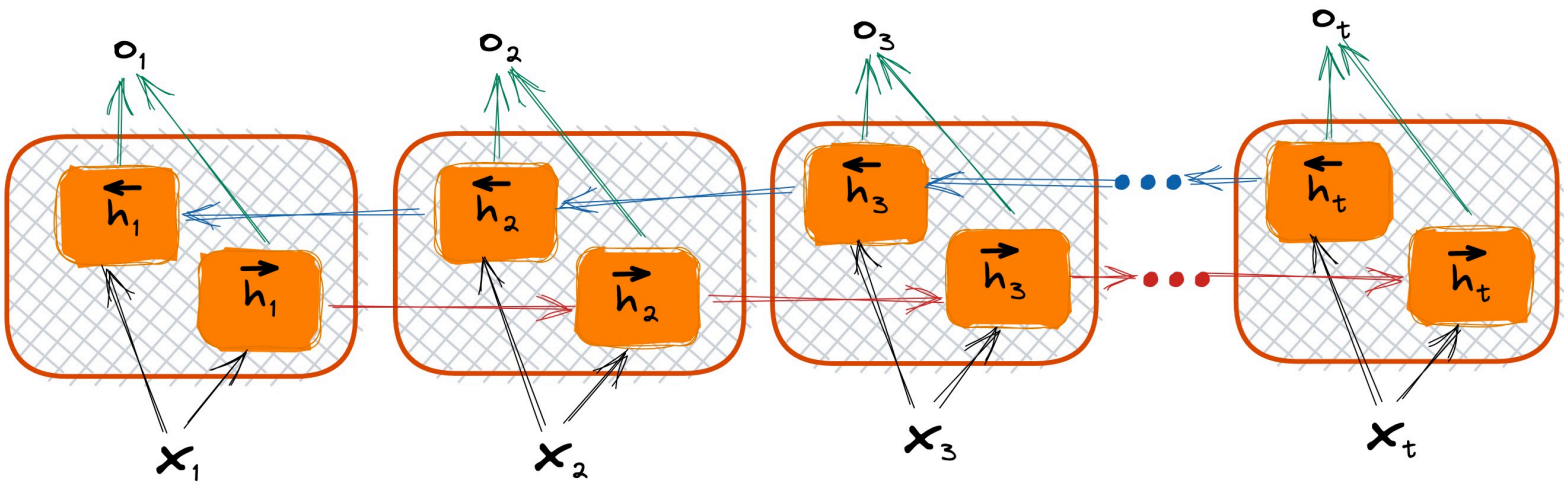
- For many applications, looking at the "future" tokens can be helpful.
- Bidirectional RNNs traverse the input in both directions.
- Concatenate the hidden states from both directions.
- Hidden blocks in each direction can be vanilla RNNs, LSTMs, GRUs etc.
- Speech recognition, handwriting recognition, etc.

$$\vec{h}_t = f\left(\vec{W}_{hx}x_t + \vec{W}_{hh}\vec{h}_{t-1}\right)$$

$$\overleftarrow{h}_t = f\left(\overleftarrow{W}_{hx}x_t + \overleftarrow{W}_{hh}\overleftarrow{h}_{t+1}\right)$$

$$h_t = \left[\vec{h}_t, \overleftarrow{h}_t\right]$$

$$y_t = g(W_{hy}h_t + b)$$

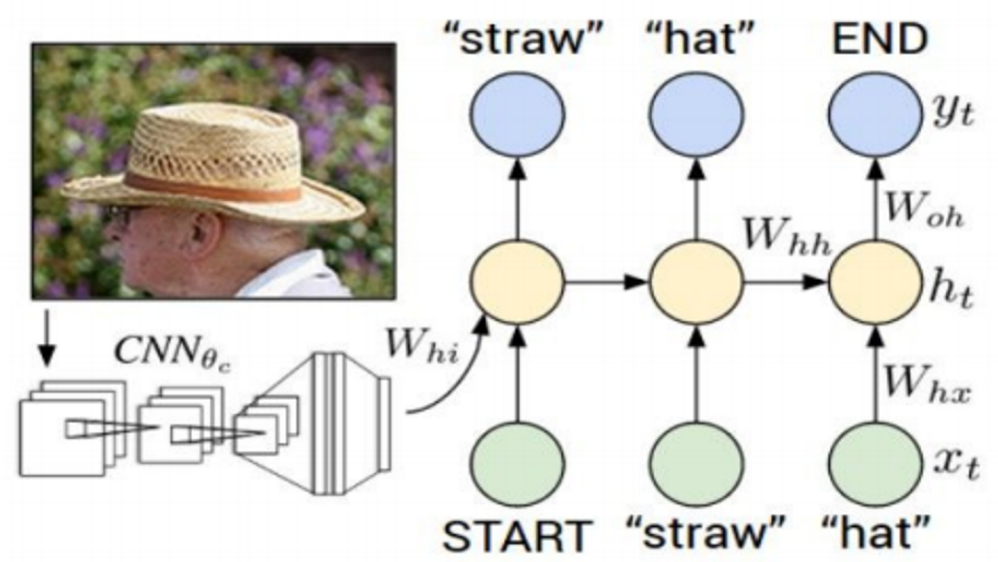


Progress

- Overview (Feed forward and convolution neural networks)
- Vanilla Recurrent Neural Network (RNN)
- Input-output structure of RNNs
- Training Recurrent Neural Networks
- Challenges in training: Exploding or vanishing gradients
- LSTM and GRU
- Multilayer and bidirectional RNNs
- Image captioning (mandatory 2)

Image Captioning

- Combine a CNN with an RNN to generate descriptive text for an image.
- Train an RNN to interpret high level features of a CNN



Deep Visual-Semantic Alignments for Generating Image Descriptions

Image Captioning

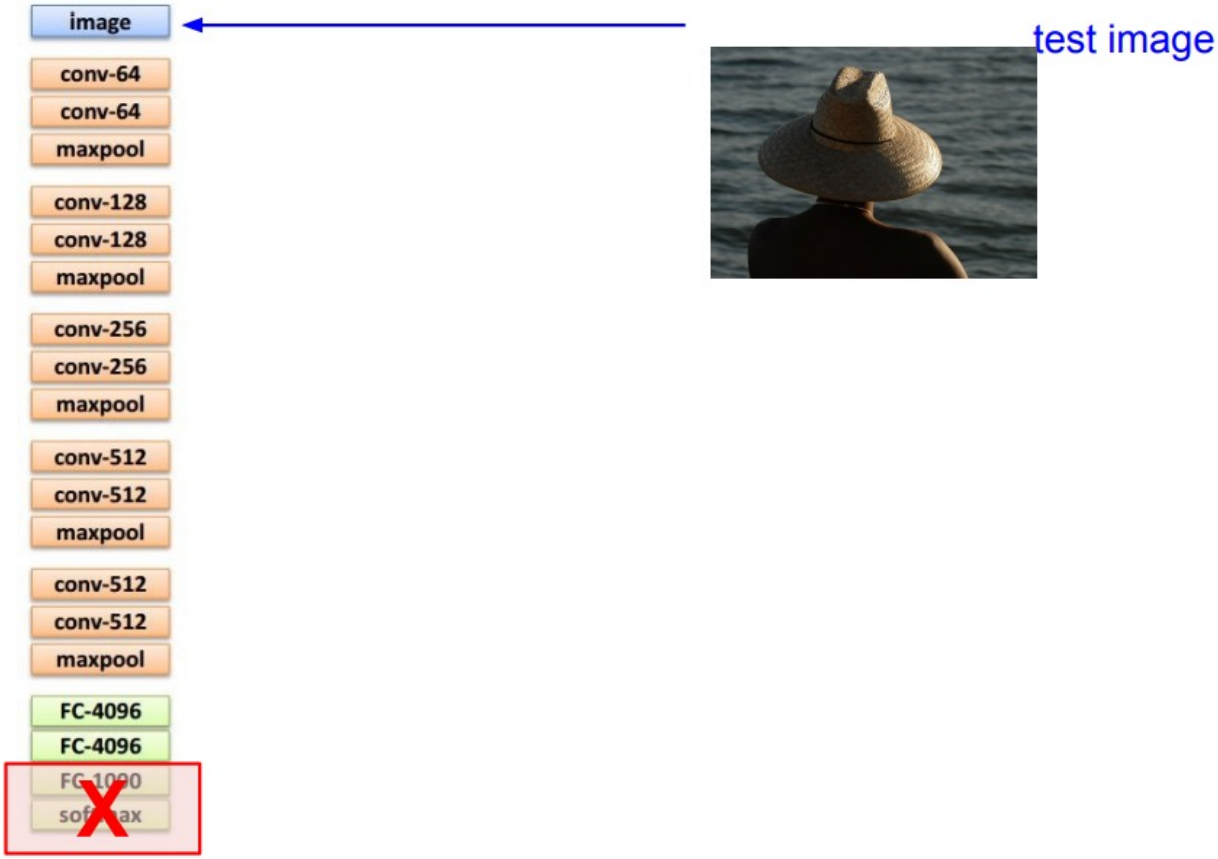
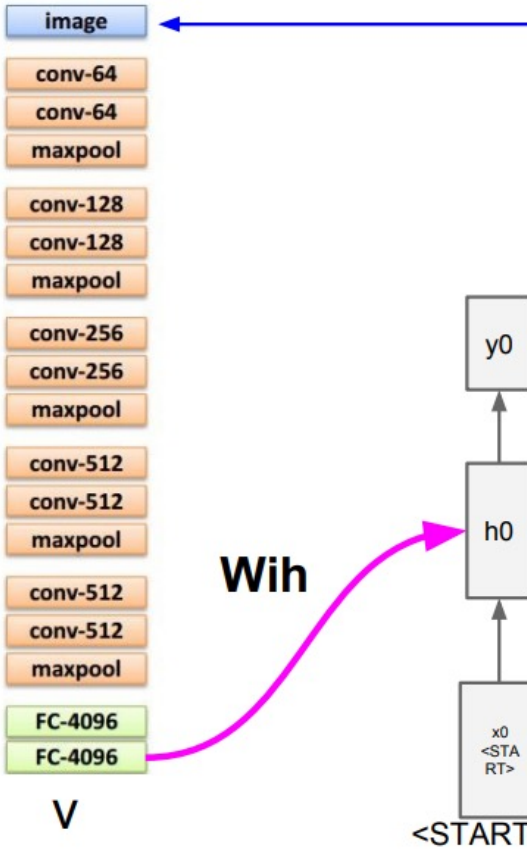


Image Captioning





test image

before:

$h = \tanh(W_{xh} * x + W_{hh} * h)$

now:

$h = \tanh(W_{xh} * x + W_{hh} * h + \mathbf{W_{ih}} * \mathbf{v})$

Image Captioning

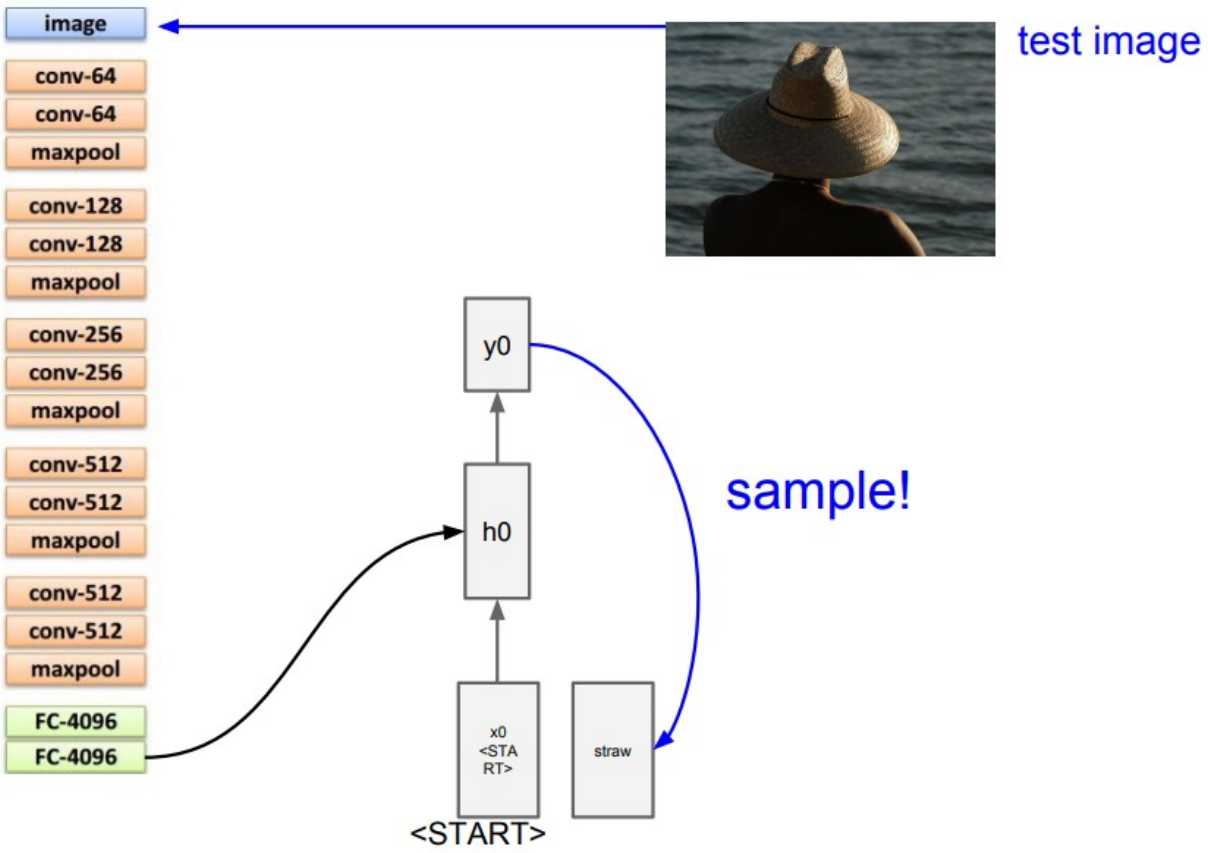


Image Captioning

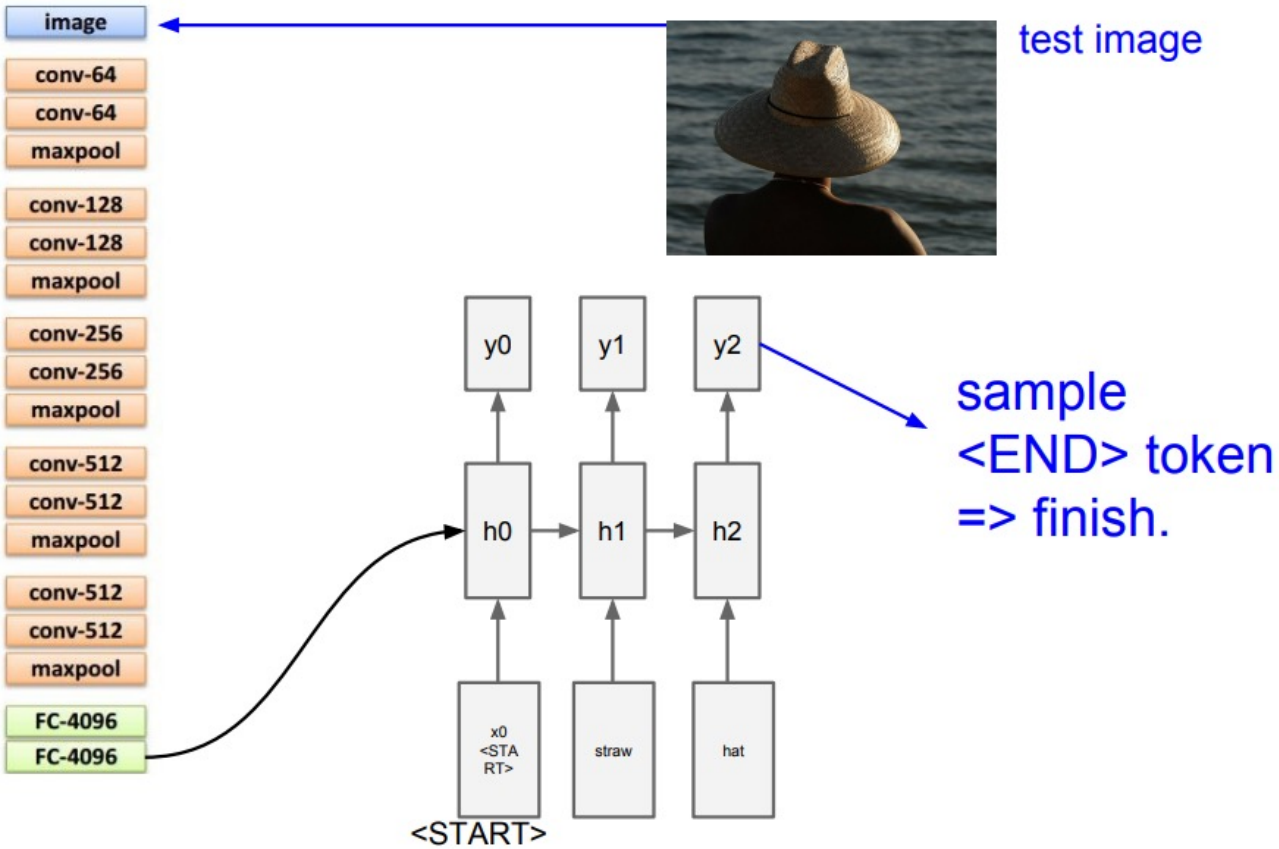


Image Captioning: Results



A cat sitting on a suitcase on the floor



A cat is sitting on a tree branch



A dog is running in the grass with a frisbee



A white teddy bear sitting in the grass



Two people walking on the beach with surfboards



A tennis player in action on the court



Two giraffes standing in a grassy field



A man riding a dirt bike on a dirt track

NeuralTalk2

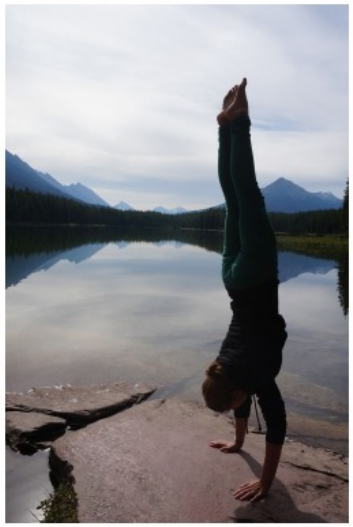
Image Captioning: Failures



A woman is holding a cat in her hand



A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard



A bird is perched on a tree branch



A man in a baseball uniform throwing a ball

NeuralTalk2

Image Captioning: Measuring Quality of a caption

- We can measure the performance of the model by using a criterion for similarity between the true caption and the generated caption.
- One such measure is the BLEU score (**Bi**lingual **e**valuation **u**nderstudy) see .e.g. https://en.wikipedia.org/wiki/BLEU.
- BLEU is a modified precision measure that handles sequences of different length. It combines counts of unigrams, bigrams, and n-grams into one score 0-100%.
- BLEU will basically compare n-grams (n=1-4 for instance) of the generated caption with the n-grams of the target captions and give a score.
- Read more here: https://towardsdatascience.com/bleu-bilingual-evaluation-understudy-2b4eab9bcfd1

Readings

- A blog post explaining LSTMs - https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Stanford's lecture on RNNs:
  - cs231n: Lecture 10 | Recurrent Neural Networks
- An awesome blog post from Andrej Karpathy on RNNs and Language Modelling:
  - The Unreasonable Effectiveness of Recurrent Neural Networks
- If you would like learn about transformers, language modelling, etc. then check out **IN 5550** Neural methods in natural language processing.
- Additional material for mandatory 2:
  - BLEU score: https://towardsdatascience.com/bleu-bilingual-evaluation-understudy-2b4eab9bcfd1
  - METEOR score: https://machinelearninginterview.com/topics/machine-learning/meteor-for-machine-translation/
  - Attention mechanism:
    - Basic Intuition: https://www.youtube.com/watch?v=SysgYptB198
    - Attention model: https://www.youtube.com/watch?v=quoGRI-1l0A
    - https://machinelearningmastery.com/the-attention-mechanism-from-scratch/