Recurrent neural networks

IN5400 — Machine Learning for Image Analysis

Anne Solberg

07.04.2021

University of Oslo

Outline

- Motivation
- Vanilla Recurrent Neural Networks
- Input-output structure
- Training recurrent networks
- LSTM cells
- GRU cells
- Short on text prediction
- RNNs and CNNs for image captioning
- Learning goals

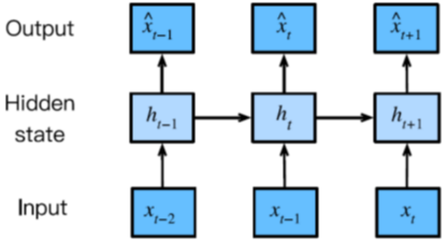
Introduction and motivation

Overview

- Models we have learnt so far:
  - Fully connected neural networks
  - Convolutional neural networks
- When do these models not work well?
  - Processing data with unknown length (time series data, text sequences, image sequences).
- Typical applications of recurrent networks (many types of sequence data):
  - Speech recognition
  - Music generation
  - Sentiment analysis (e.g. rate a text)
  - Video processing
  - Text analysis and translation

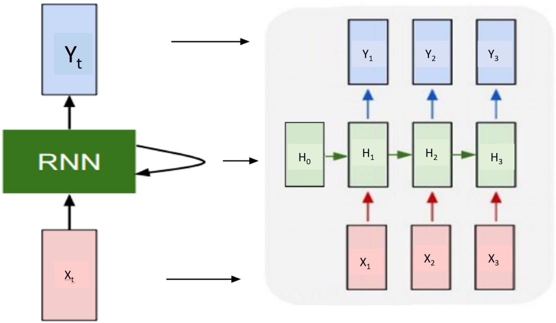
Brief introduction to time series

- Given a time series of measurements $\{x_1, ...x_N\}$
- Suppose we want to predict x_{t+1} given $\{x_1, ...x_t\}$
- If x_{t+1} only depends of previous values $(\{x_1, ...x_t\})$, we say that $\mathbf{x}$ is **causal**.
- In probabilistic terms, we can state $x_{t+1} \sim P(x_{t+1}|x_t, ...x_1)$.
- If x_{t+1} only depends on x_t, this is simplified to $x_{t+1} \sim P(x_{t+1}|x_t)$.
- Assume that the estimated $\hat{x}_{t+1}$ depends on some unobserved latent variable state describe by h_t.
- We will see these latent states also in the nodes in a recurrent network.



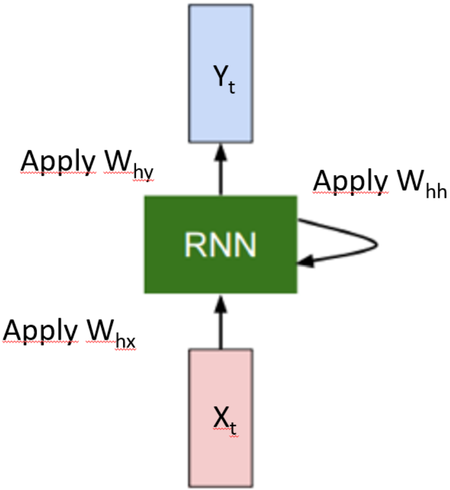
Recurrent neural network (RNN)

- Given a time series of input data $X = \{X_1, ...X_N\}$ (text, images, time series)
- X_t is typically a vector, and $\mathbf{X}$ a matrix
- Estimate output Y_t given $\{X_t\}$ and the hidden state vector H_t.
- Update state to get $H_t = f(H_{t-1}, X_t)$.
- For each time:
  - Take a new input
  - Update the state
  - Reuse weights
  - Compute a new output

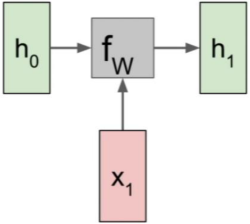


Vanilla Recurrent neural network (RNN)

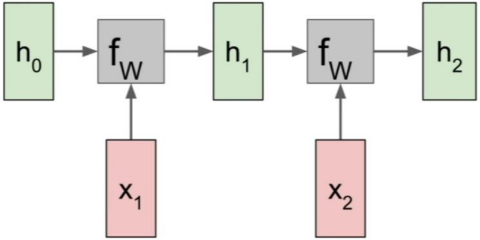
- Input vector X_t.
- Hidden state vector H_t
- Weight matrices W_{hh}, W_{hx}, W_{hy}.
- Vanilla RNN update:
  $H_t = tanh(W_{hh}H_{t-1} + W_{hx}X_t + b)$.
- Output: $Y_t = g(W_{hy}H_t + b)$. Here g() is typically sigmoid or softmax.
- Remark: We often concatenate W_{hh} and W_{hx} into W, and multiply with the concatenation of H_t and X_t.



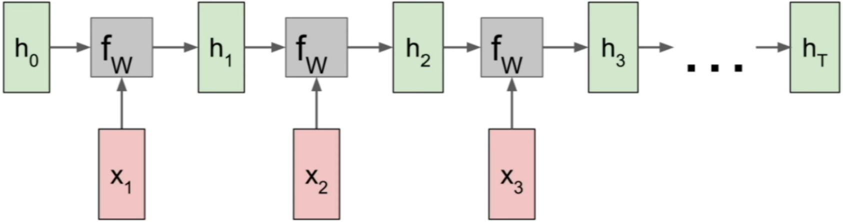
RNN computational graph - first update



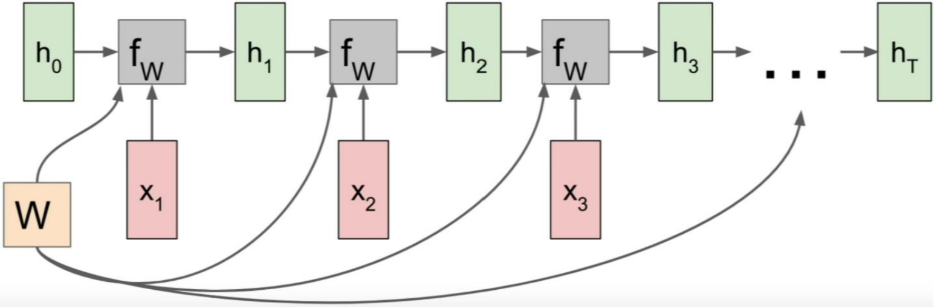
RNN computational graph - second update



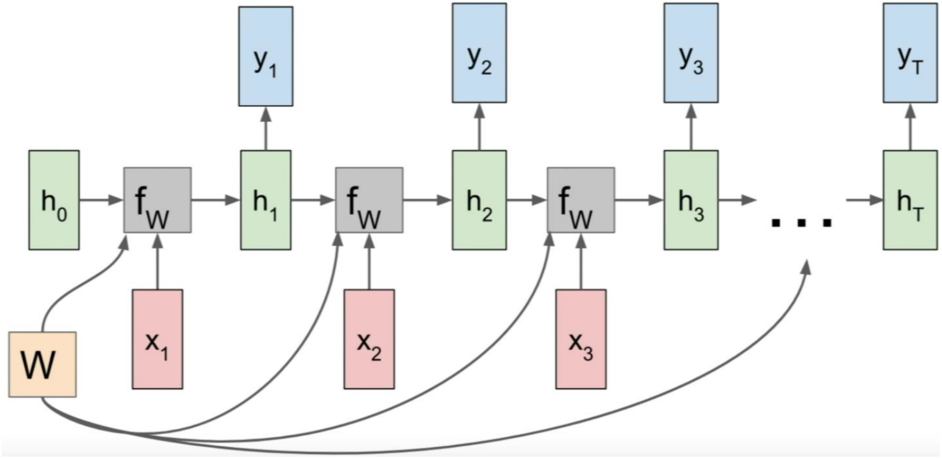
RNN computational graph - all state updates



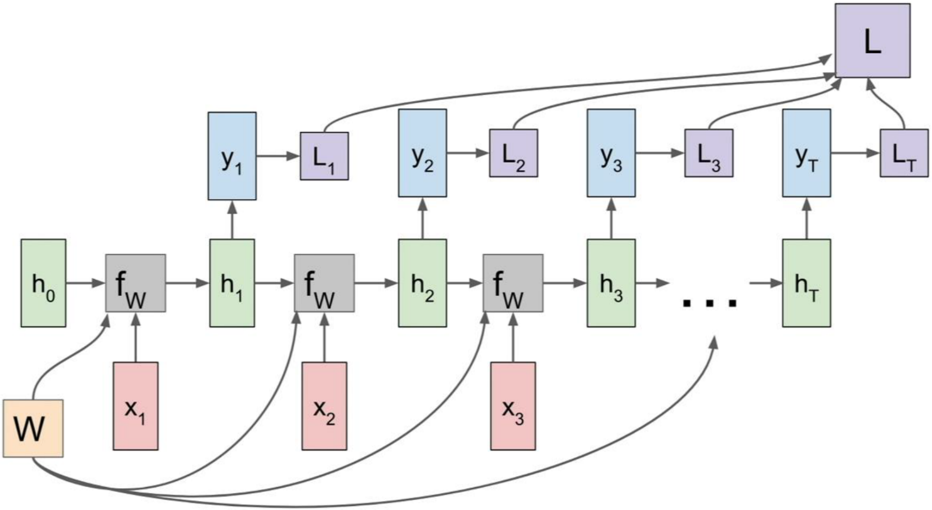
RNN computational graph



RNN computational graph - reuse weights for all steps

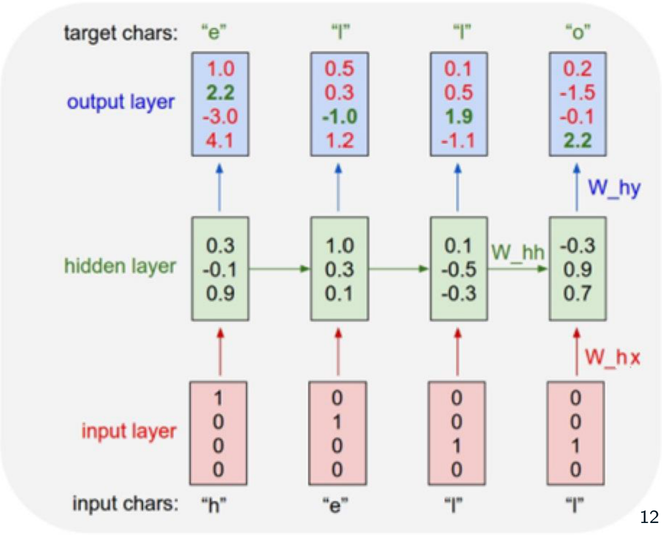


RNN computational graph - Computing loss L



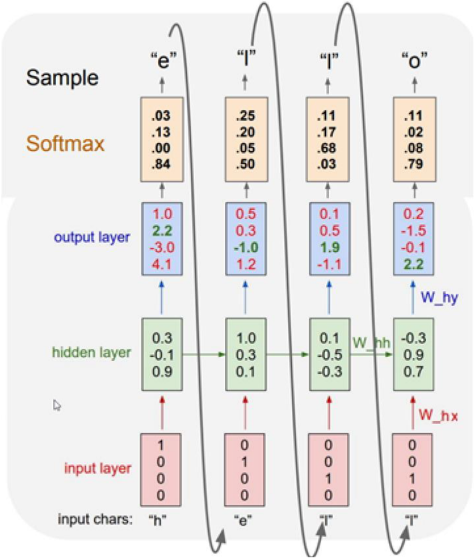
Example - predict next character - training time

- Task: Predict the next character
- Training sequence: 'hello'
- Vocabulary: ['h','e','l','o']
- Encoding: onehot
- For character prediction and onehot labels: softmax used at output layer, cross-entropy loss between the softmax and the onehot-vector of true next character.
- During training: notice that the input at step t+1 is equal to the output at step t, $x_{t+1} = y_t$.



Example: generate new sequences (during test/inference)

- Test time: generate new text by sampling from the softmax "probabilities".
- Sample from softmax by e.g. drawing a random number [0, 1] and assign according to softmax values.
- The sampled character is input to the next time step.
- Note that we do not use just the most probably character from softmax, but sample from the softmax distribution.



Typical text preprocessing

- The example above predicted single characters, we can also predict words.
- If so, typically we preprocess the text.
  1. Read as strings.
  2. Split into tokens (word and symbols like "EOS", "EOF", "UNK" and other special tokens.
  3. Build a vocabulary to map between tokens and numerical indices in the vocabulary.
  4. Convert text into numerical indices.

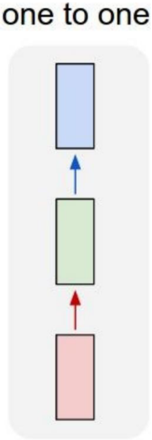
Input/output structure of RNNs

Input/output overview

- One-to-one
- One-to-many
- Many-to-one
- Many-to-many
- Many-to-many (encoder/decoder)

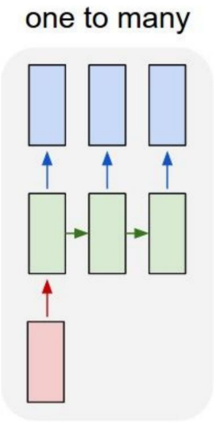
One-to-one models

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



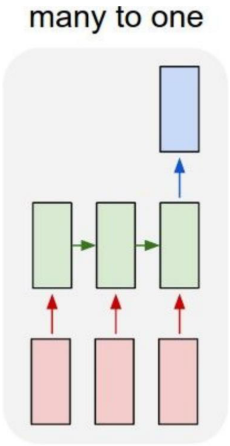
One-to-many models

- Example: Image captioning
- One input image - output: a sequence of words.



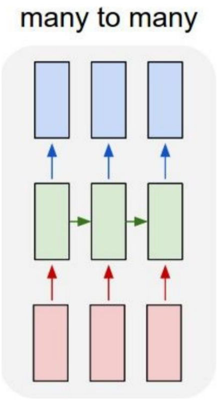
Many-to-one models

- Example: video classification
- Example for text analysis: sentiment classification



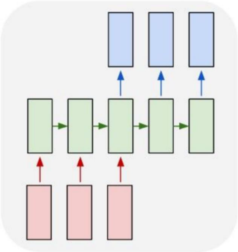
Many-to-Many models

- Example: frame-to-frame video classification



Many-to-Many : encoder-decoder

- Example: text translation
- Note here that the input and the output can have different lengths.



Training RNNs

RNNs and training

- Challenge: preserve long-range dependencies
- Vanilla recurrent networks
  - $H_t = f(W_{hh}H_{t-1} + W_{hx}X_t + b)$
  - If $f = ReLU$ we easily get exploding values or gradients
  - If $f = tanh$ we easily get vanishing gradients, and remembering many steps back is difficult.
- Finite memory will also set limitations.

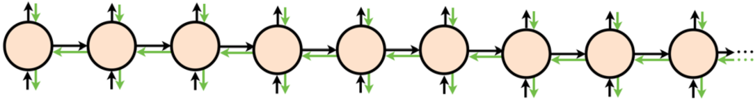
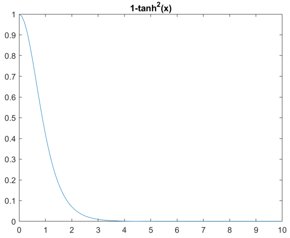


Figure 1: Undirected graph

Exploding or vanishing gradients

- tanh() does not give exploding *values*.
- tanh() an give exploding gradients:
- $H_t = \tanh(W_{hh}H_{t-1} + W_{hx}X_t + b)$
- $\frac{\partial H_t}{\partial H_{t-1}} = \left[1 - \tanh^2(W_{hh}H_{t-1} + W_{hx}X_t + b)\right] W_{hh}$
- Depending on the size of W_{hh}, the gradient can either vanish or explode in time:
- For scalar W_{hh}:
  - If $|W_{hh}| < 1$: vanishing gradients.
  - If $|W_{hh}| > 1$: exploding gradients.
- For matrix W_{hh}:
  - If largest singular value < 1 : vanishing gradients.
  - If largest singular value > 1: exploding gradients.

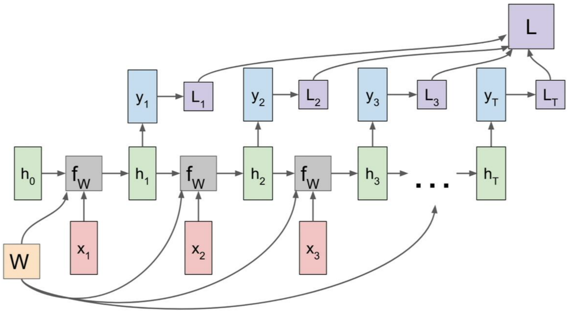


Exploding gradients: Gradient clipping

- To avoid exploding gradients, clip them if they are larger than a threshold.
- Two common approaches: clipping-by-value or clipping-by-norm.
- Clipping-by-value: If $g > \textit{threshold}, g = \textit{threshold}$. ($g$ is the gradient)
- Clipping-by-norm: If $g > \textit{threshold}$, set $g = \textit{threshold}\frac{g}{||g||}$.

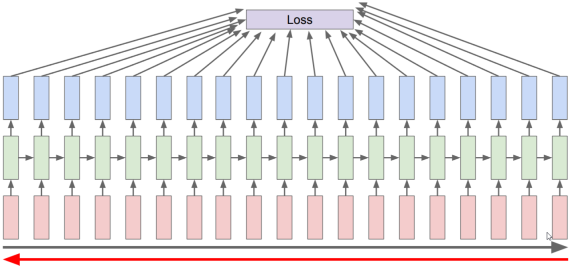
Is vanishing gradients a problem in RNNs?

- RNNs get a fresh input X_t at each time step
- Thus, vanishing gradients is not a big problem
- A bigger challenge is to remember many steps back (we will introduce LSTMs and GRU cells to help this).



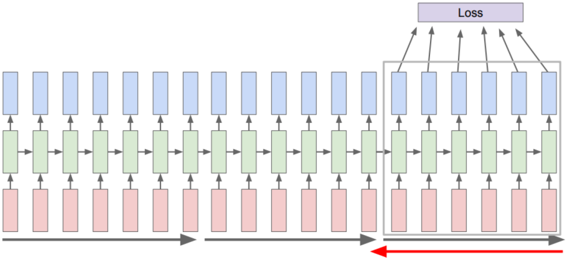
Backpropagation through time (we will not go into computational details)

- Training a recurrent network is done using backpropagation through time.
- To calculate gradients you have to keep your inputs in memory until you get the backpropagated gradient.
- What do you do if you are reading a long book?



Truncated backpropagation through time

- Stop after some steps and update the weights as if you were done.
- Advantages: Reducing the memory requirement, faster parameter updates.
- Disadvantage: Not able to capture longer dependencies then the truncated length.



Alternative hidden state computations - GRU and LSTM cells

Gated Recurrent Units (GRU) main concepts

- Let each hidden state have a memory cell $\tilde{H}_t$.
- This memory can learn to keep important concepts from earlier in the sequence (e.g. if a noun is plural or not in "The cars, that we used to go to the mountains, were dirty").
- Define an update gate that controls how the memory is updated.
- The gate has the same dimension as the hidden state vector and has elements between 0 and 1.
- Note that we use a sigmoid as a "soft" gate to squeeze the input to be betweeen 0 and 1.
- Also introduce a gate to decide how much of the input is used to combine the input with the memory.

Gated Recurrent Units (GRU)

- Gate computations:
- Reset gate: $R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r)$
- Update gate: $Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$
- Candidat hidden state: $\tilde{H}_t = tanh(X_t W_{xr} + (R_t \odot H_{t-1}) W_{hh} + b_h)$
- Hidden state update: $H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$

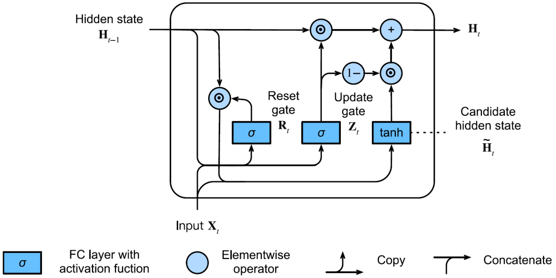


Fig. 9.1.3: Computing the hidden state in a GRU model.

Gated Recurrent Units (GRU) - some observations

- Gate computations:
- Reset gate: $R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r)$
- Update gate: $Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$
- Candidat hidden state: $\tilde{H}_t = tanh(X_t W_{xr} + (R_t \odot H_{t-1}) W_{hh} + b_h)$
- Hidden state update: $H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t$
- If the update gate is close to 1, the hidden state will remember the previous state.
- If both the update and the reset gate are close to 0, the GRU will forget the past.
- In a vanilla RNN, we update the hidden state no matter how useful we find the input.

Long Short Term Memory (LSTM) cells

- The LSTM cells have an additional type of gate: an output gate, and a candidate memory cell.

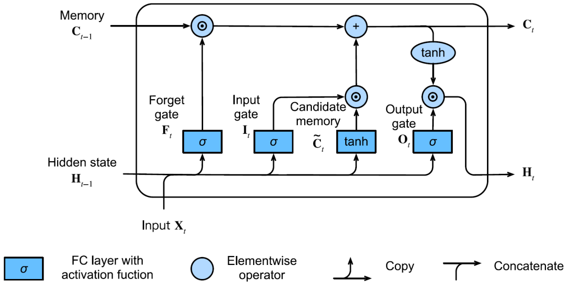


Fig. 9.2.4: Computing the hidden state in an LSTM model.

Long Short Term Memory (LSTM) cell - equations

- Input gate: $I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$
- Forget gate: $F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$
- Output gate: $O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$
- Candidate memory cell $\tilde{C}_t = tanh(X_t W_{xc} + H_{t-1}) W_{hc} + b_h)$
- Memory cell update: $C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$
- Hidden state update: $H_t = O_t \odot tanh(C_t)$

GRU vs. LSTM?

- GRU is simpler than LSTM, have fewer parameters and might train faster

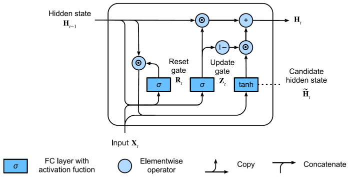


Fig. 9.1.3: Computing the hidden state in a GRU model.

- LSTMs have more power, and for some applications work better.

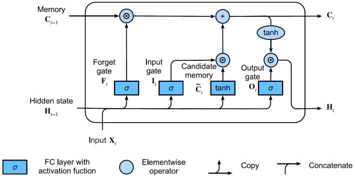


Fig. 9.2.4: Computing the hidden state in an LSTM model.

Multilayer RNNs

- Multilayer RNNs can be used to enhance model complexity
- Stacking layers creates a higher level feature representation.
- Normally max 3 layers are used. More complex relationships are learning in the time dimension.

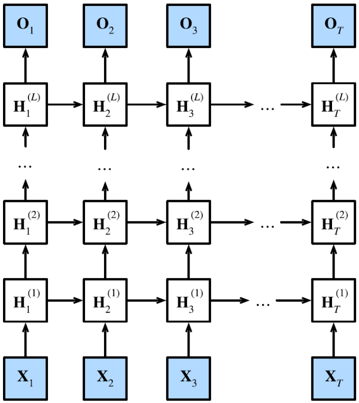


Fig. 9.3.1: Architecture of a deep RNN.

Bidirectional recurrent neural networks

- A standard RNN can only model causal sequences, where the output and hidden states only depend on *past* times.
- For many applications, causality is not a reasonable assumption.
- Example text 1: "Teddy bears are on sale".
- Example text 2: "Teddy Roosevelt was a great president."
- The solution is to use bidirectional RNNs.
- They traverse the hidden states in both time directions and combine the output.
- Concatenate the hidden states in both direction to get H_t and compute the output as $O_t = H_t W_{hq} + b_q$.

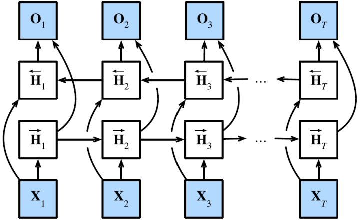


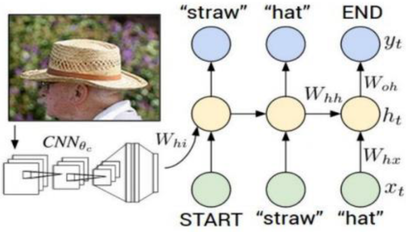
Fig. 9.4.2: Architecture of a bidirectional RNN.

$$\overrightarrow{\mathbf{H}}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh}^{(f)} + \overrightarrow{\mathbf{H}}_{t-1} \mathbf{W}_{hh}^{(f)} + \mathbf{b}_h^{(f)}),$$
$$\overleftarrow{\mathbf{H}}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh}^{(b)} + \overleftarrow{\mathbf{H}}_{t+1} \mathbf{W}_{hh}^{(b)} + \mathbf{b}_h^{(b)}),$$

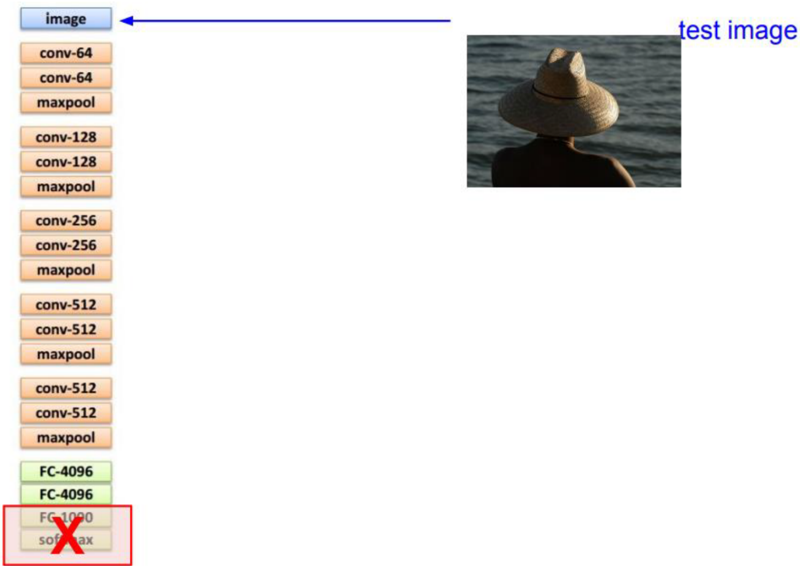
Introducing Mandatory 2 - Image captioning using RNNs

Introduction to image captioning

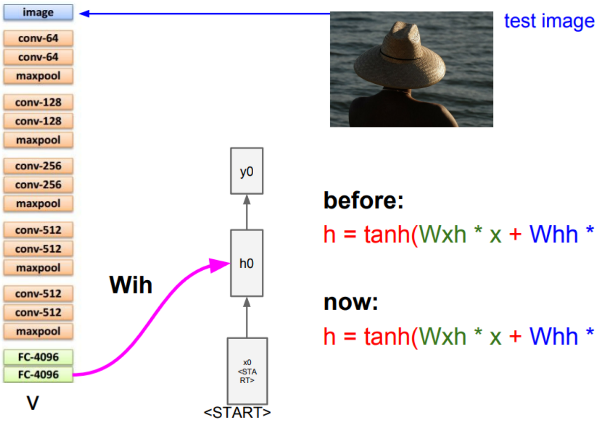
- Combining text RNNs with the output from a CNN.
- RNN input: CNN features.



Introduction to image captioning - CNN features to RNN



Introduction to image captioning - X_0



before:

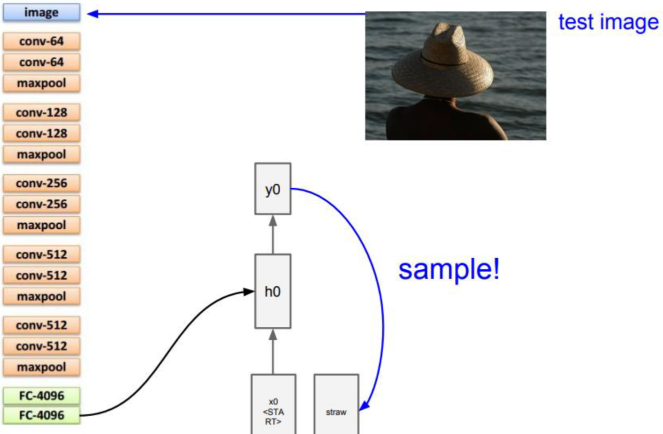
h = tanh(Wxh * x + Whh * h)

now:

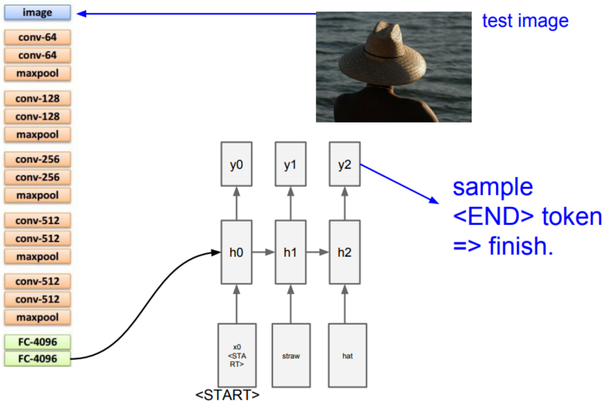
h = tanh(Wxh * x + Whh * h + **Wih * v**)

Introduction to image captioning - generate text/inference

- At training time we compare the true word with the softmax output.
- During inference/text generation, we sample from the softmax distribution to get the input to the next hidden state.



Introduction to image captioning - stopping the sequence.

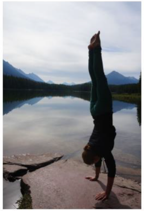


Example of good captions.

Example of not so good captions.

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

Measuring the quality of a caption - BLEU score

- We can measure the performance of the model if we use a criterion for similarity between the true caption and the generated caption e.g. on validation datat.
- One such measure is the BLEU score (Bilingual evalution understudy) see .e.g https://en.wikipedia.org/wiki/BLEU.
- Given two reference sentences like " The cat is on the mat" and "There is a cat on the mat".
- Given one candidate ML translation like "The the cat on cat", we measure a modified precision score between the reference sentences and the ML candidate.
- 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%.
- Read more on e.g. https://towardsdatascience.com/bleu-bilingual-evaluation-understudy-2b4eab9bcfd1

Do you want to do state-of-the-art language modelling

- Currently, language models use transformers and attention.
- If you want to learn, check out IN 5550 Neural methods in natural language processing.

Learning goals RNN

- Vanilla RNNs
- RNN computational graphs
- Input/Output structures
- Challenges in learning.
- GRU and LSTM - basic concepts.
- From Mandatory 2: practical RNNs for image captioning.
- If you want to learn, check out IN 5550 Neural methods in natural language processing.

Questions?