McGill **Artificial Intelligence** Society

Lecture 5: Recurrent Neural Networks and sequential problems

Announcements

Assignment 4

- Due next week

Deliverable 3

- Due next week

Today's Lesson Plan

Recap of feed-forward neural networks

What kind of problems can vanilla NNs not solve?

Recurrent Neural Networks

Vanishing gradient problem

Long Short Term Memory

Shortcomings of RNNs (and NNs in general!)

Demo

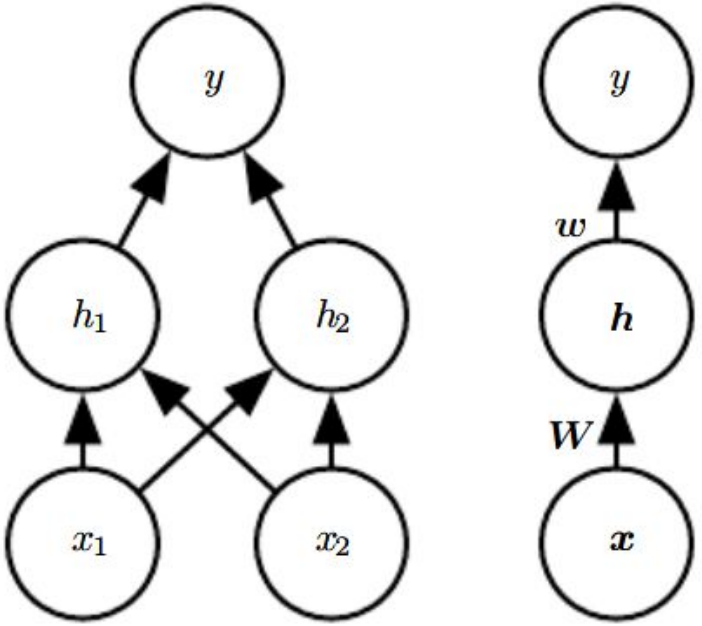
Recap: Feed-forward Neural Networks

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b.$$

Output of a single hidden layer neural network with rectified linear activation.

RHS is condensed version of LHS.

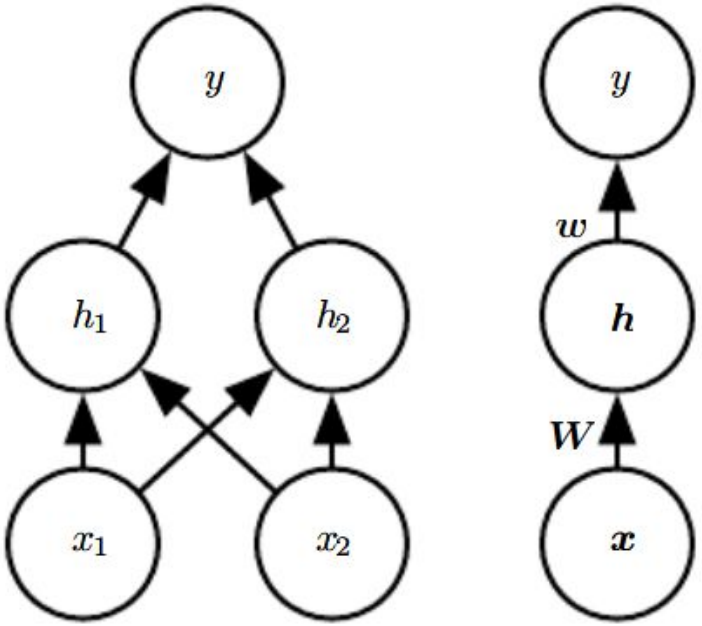
Why is it called feed-forward?



Taken from http://www.deeplearningbook.org/contents/mlp.html

Shortcomings of non-sequential neural networks

- No notion of “time” in a vanilla neural network
- fixed length input/output



Solution: Recurrency and sharing parameters

In vanilla neural network, each input has its own parameter (elements of W).

- “I went to Nepal in 2009” and “In 2009,I went to Nepal.”
- Every input (word) has its own weight

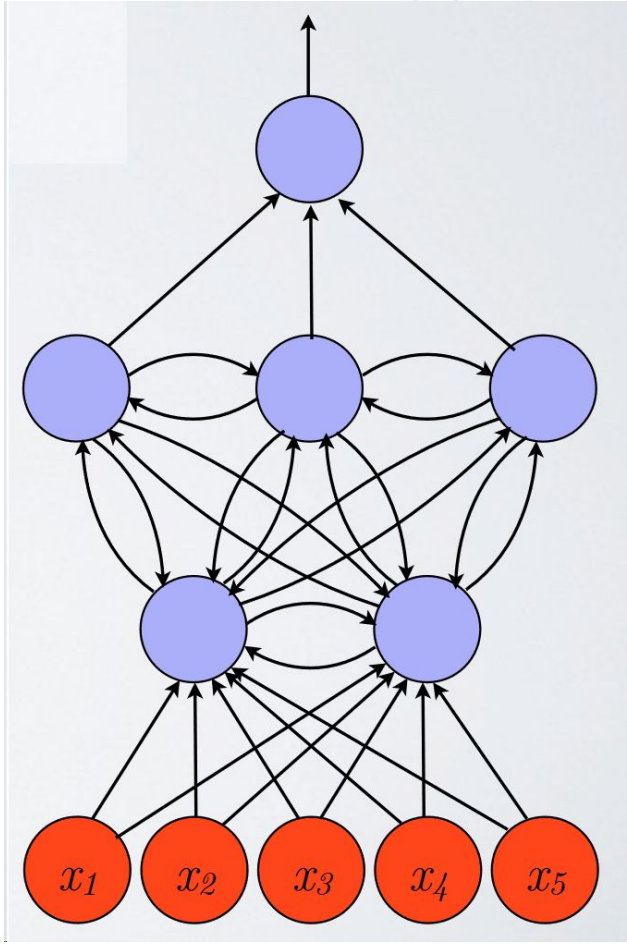
Recurrent neural network shares weights across multiple time steps

Refer to inputs of a sequential problem as x_t, with t being the time-step index from 1 … T.

Recurrency and Cycles

What does recurrency and cycles in our computation graph allow us to do?

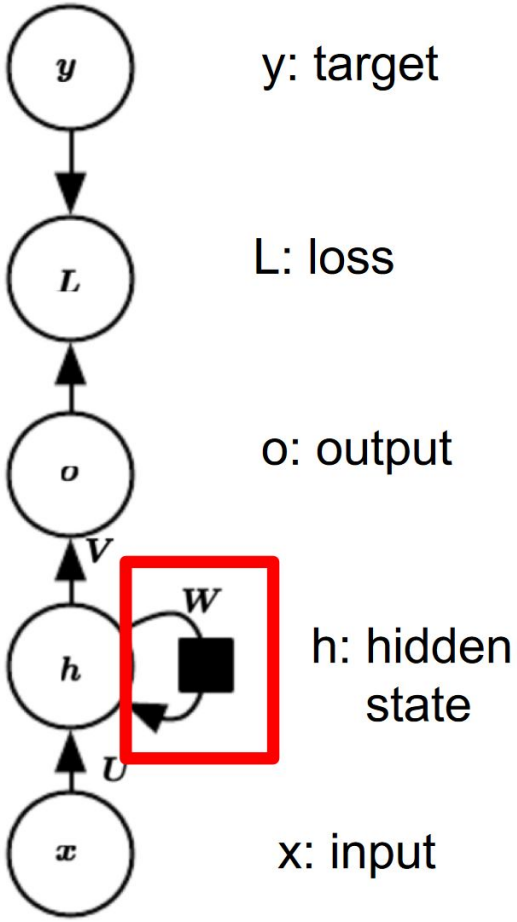
- Keep internal state throughout time steps
- Allow us to work with time-series data
- Capture oscillatory patterns
- Ignore parts of input sequence



How do we incorporate a sequential flow of information?

What kind of cycles do we use?

- Time delay cycles - information from this state is sent to next state
- There is a “remembered” state that is passed forward in the sequence. We call this the *hidden state* of an RNN (h_t).
- We can use h_t as input into some downstream model (ie. for machine translation, distribution over target dictionary)



Slide adapted from Herke Van Hoof’s - COMP 551 Lecture 13 on Dimensionality Reduction

RNNs continued.

What does this allow us to do?

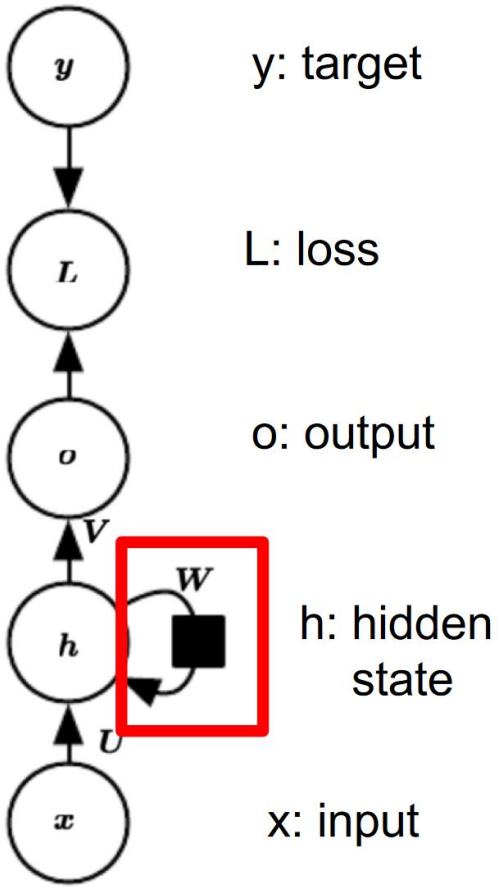
- Can view an RNN as having a ***hidden state*** *h_t* that changes over time
- *h_t* represents **'useful information'** from past inputs

$$\boldsymbol{h_{t+1} = f(h_t, x_t)}$$

Often, $h_{t+1} = \sigma(Wh_t + Ux_t + b_h)$, where sigma is the non-linearity, b is bias

- h_t used to make predictions:

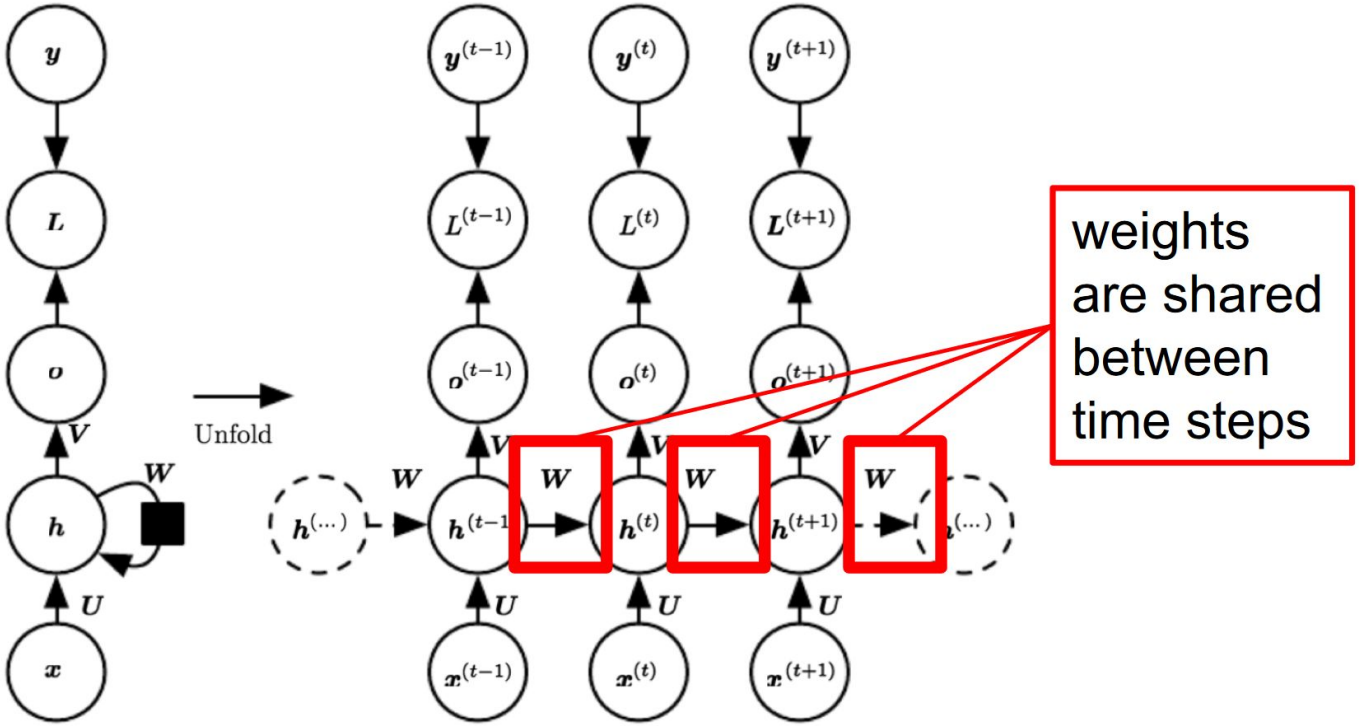
$$o_t = \sigma(Vh_t + b_o)$$



Taken from Ryan Lowe/Joelle Pineau's COMP 551 slides

Unfolding the computational graph

This is actually what's happening during an RNN computation - we continually predict the next hidden state, and use this for predictions. But how do we train this model?



How do we train RNNs?

Same as feed-forward networks, we use back-propagation on our "unrolled" computation graph.

Use chain rule to calculate partial derivatives

SAME as feed-forward neural networks! We call this **Backpropagation through time (BPTT)**.

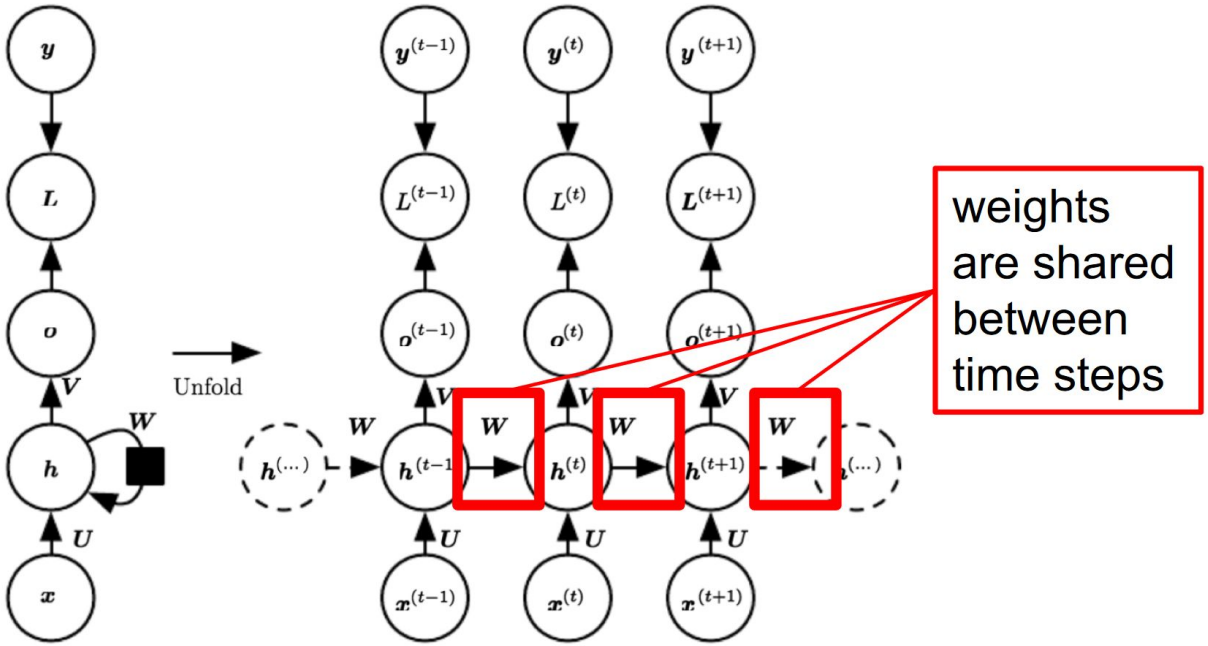
Problem with long-term dependencies

Example: Language modelling.

- “I grew up in France, [long sentence here], I speak fluent ____” -> Model trying to predict next word.

In practice - very hard for RNNs to learn long-term dependencies.

Why is this?



Problem with long-term dependencies cont.

Remember: transition between hidden states uses the same weight **w** for all time steps.

Can cause gradients to either **explode** or **vanish** - imagine multiplying a scalar by itself *n* number of times. What happens? Similar with matrices!

If you're interested in eigendecomposition view on this, check out the 551 slides here

If you're interested in a mathematical view of this check out this from U of T

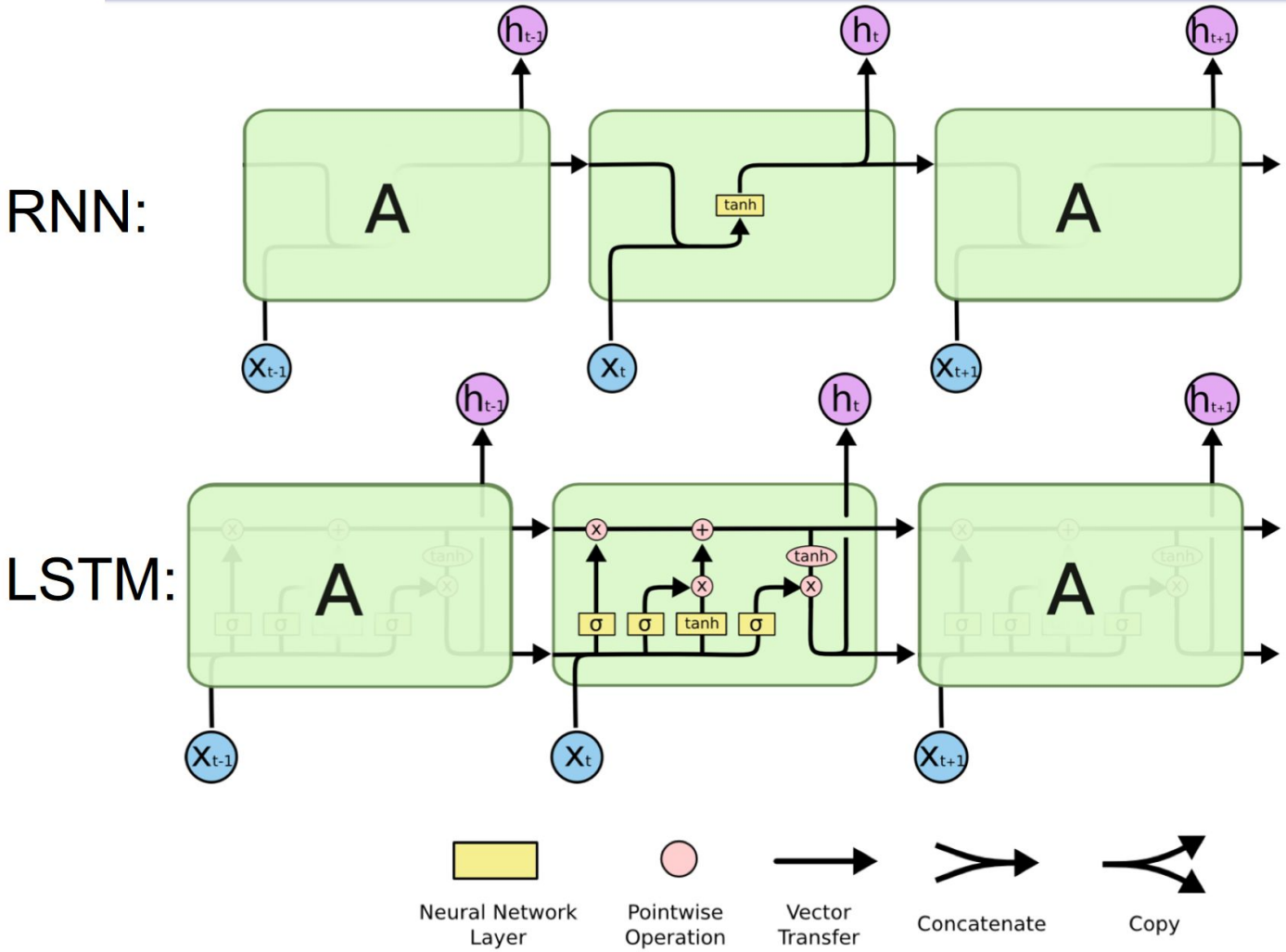
Fixing vanishing/exploding gradients

- Clip gradients:
  - if |gradient| > max_val :
    gradient = value * sign(gradient)

- More complex RNN architecture to include non-multiplicative interactions.

Note: next few slides (15 - 22) all taken from Ryan Lowe/Herke Van Hoof/Joelle Pineau's 551 slides.

Long Short-Term Memory (LSTM)



LSTM components

Governed by this set of equations below

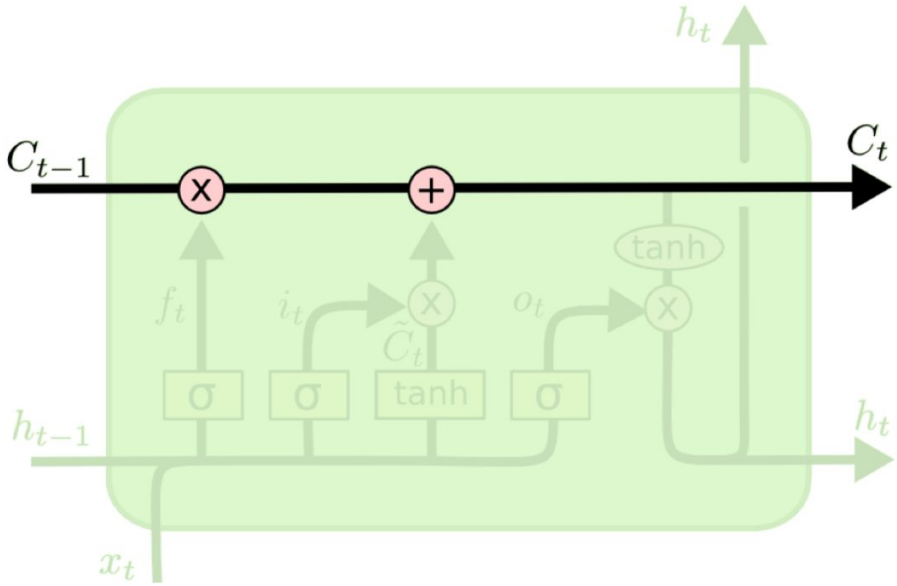
- Has two type of “states”, cell state and the previous hidden state we mentioned

$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right) \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right) \\
h_t &= o_t * \tanh\left(C_t\right)
\end{aligned}
$$

Slides 16-21 Taken from Herke Van Hoof’s COMP551 Slides Lecture 13 - Dimensionality Reduction

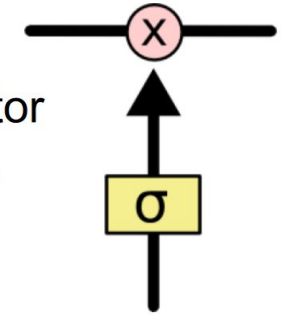
Long Short-Term Memory (LSTM)

- Core idea: the **cell state**, an 'information highway'
- Cell state is updated **additively** based on input, rather than **multiplicatively** => *less prone to exploding/ vanishing gradients*



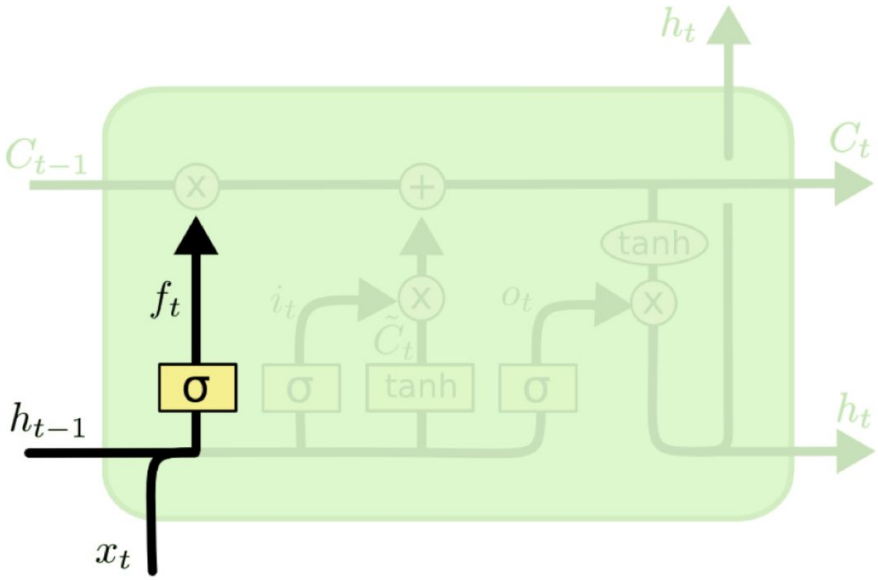
Long Short-Term Memory (LSTM)

- Cell state vs hidden state (roughly)
  - Hidden state: what info from past do I need **to make my next prediction**?
  - Cell state: what info from past might I need **to make future predictions**?
- For regular RNN, hidden state plays both of these roles
- LSTM uses a set of '**gates**' to control information flow
  - Gate = sigmoid layer + elem.-wise multiplication. Gives vector of numbers between [0,1] that determine how much of each component to let through



Long Short-Term Memory (LSTM)

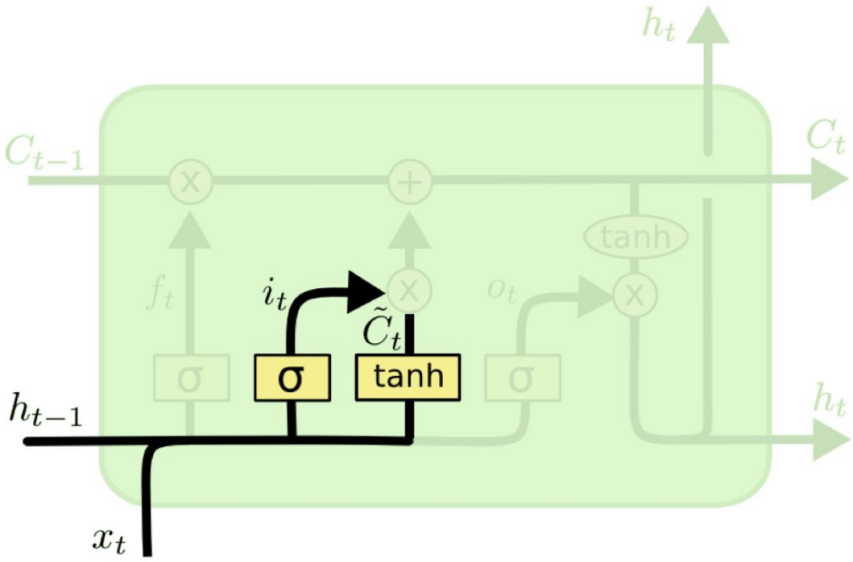
- **Forget gate**: how much information do we want to keep from the previous cell state?



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Long Short-Term Memory (LSTM)

- **Input gate**: what information from the current input (and previous hidden state) do we want to transfer to the cell state?

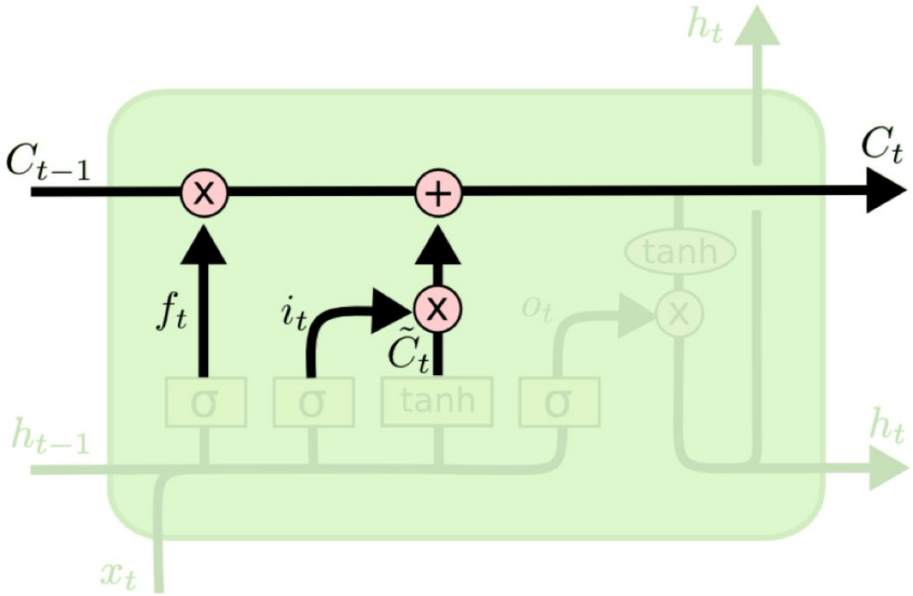


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Long Short-Term Memory (LSTM)

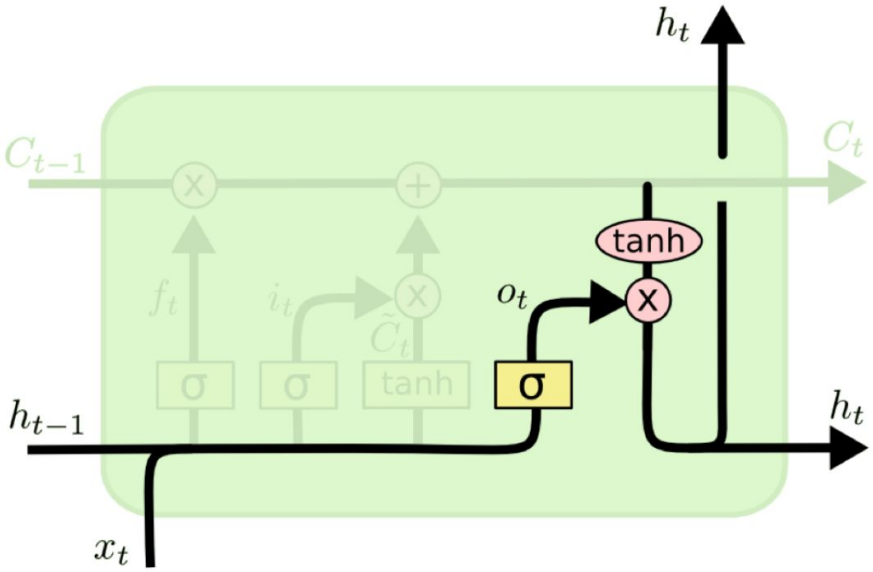
- Cell state updated as an **additive linear combination** of old cell state and processed input



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Long Short-Term Memory (LSTM)

- **Output gate:** what information from the cell state do we need to make the next prediction?



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM components cont.

Still interested? For a more in-depth explanation of RNNs and LSTMs check out:

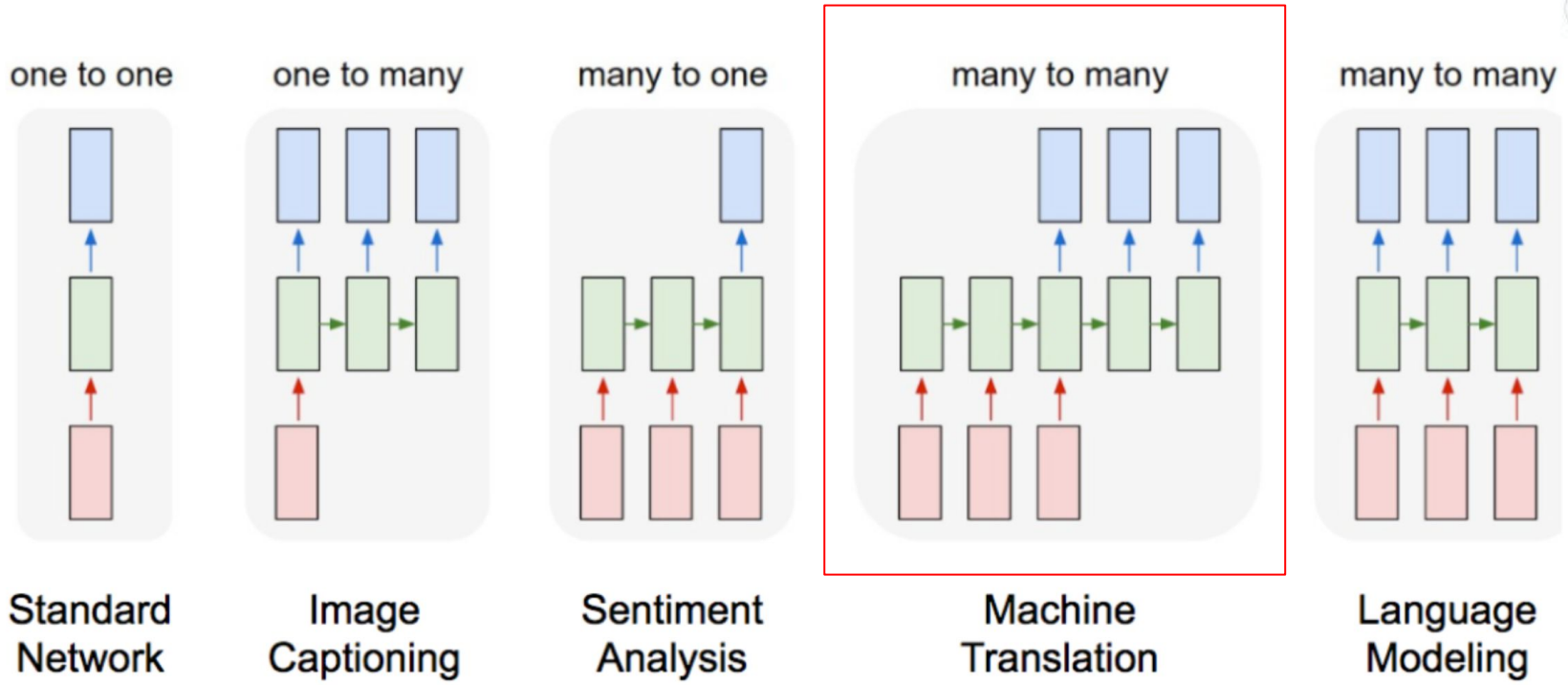
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

What are some things we can do with RNNs?

5 minute break

How do we apply this to a downstream task?

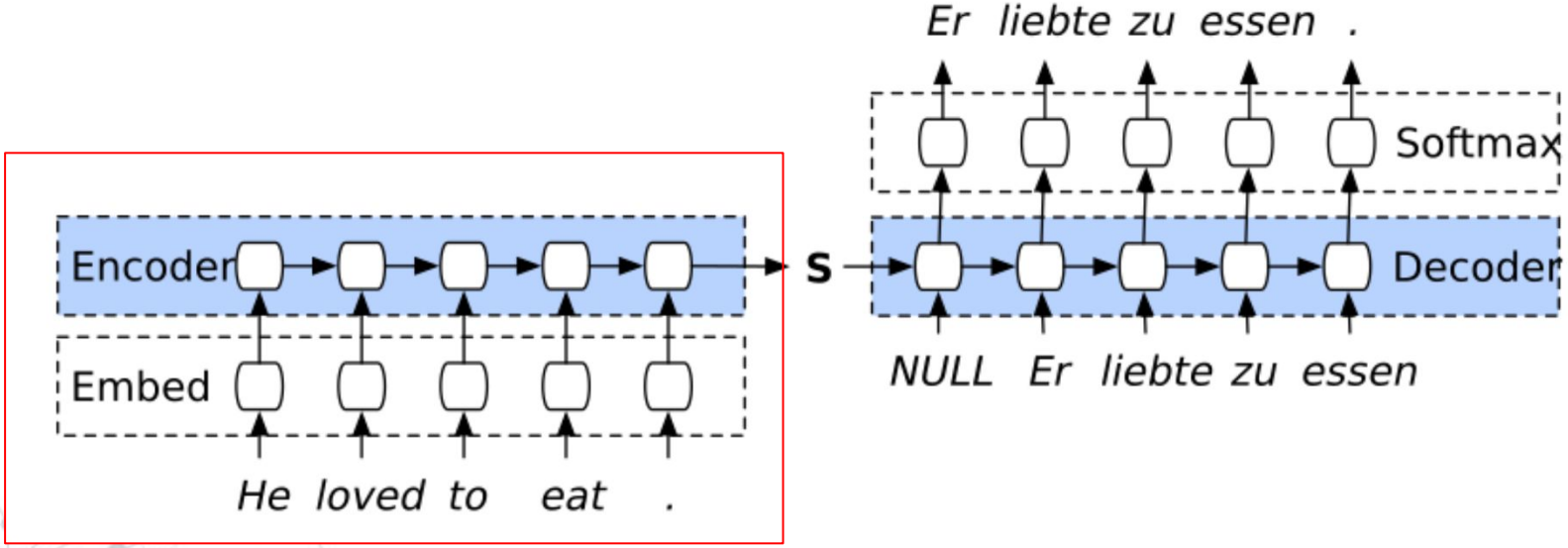
Sequential nature of RNNs allow us to apply it to many tasks, for example...



Sequence to sequence models

One of the most popular approaches towards sequential data are Sequence to Sequence (Sutskever et al, 2014.) models.

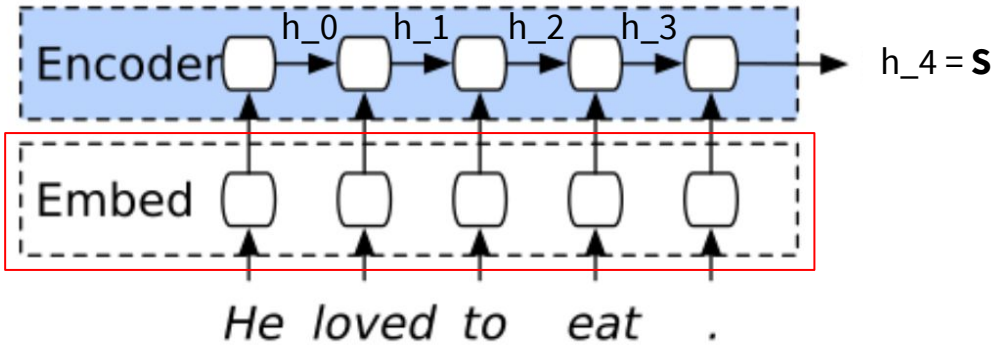
Follows an encoder-decoder architecture.



Encoders with RNNs

Representation learning for sequences

- Builds a representation of input sequence with the hidden state.
- First embeds each input from a word index into something in R^n, where n is embedding size.
- Then passes this embedding to the RNN, RNN generates a hidden state
- Sequentially takes in input and has a final “representation” of entire sentence in final hidden state (h_T).
- How could we encode variable length sentences?



A quick aside on embeddings

Vector representations of a certain word.

Used for capturing context. Why do we need them?

ie. "*have a good day*" vs "*have a great day*" - barely any difference in meaning.

We can map each word in the combined vocab into the following dictionary:

0 = have, 1 = a, 2 = good, 3 = great, 4 = day

One-hot encodings of each will tell us that (in terms of euclidean distance) *have* and *day* are as dissimilar as *great* and *good* - which is not true!

We need a way of capturing similarity in vector space!

Word embeddings cont.

We do this with multiple methods

- frequency based methods (how often does a word show up in this set of documents? or similar words appear close to each other)
- prediction based methods (solve the prediction problem of given a context, predict a word / given a word, predict a context, then use the weights)

Allows us to do cool things in embedding space like

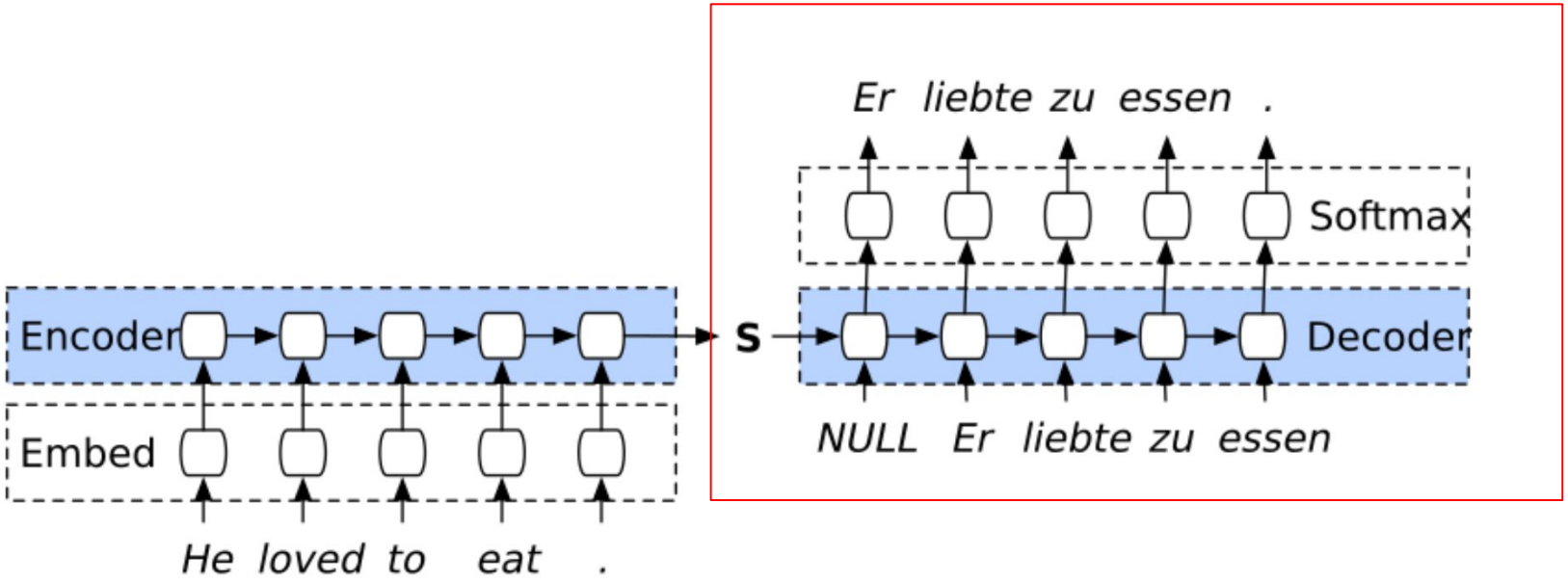
embed(king) - embed(man) + embed(woman) = embed(Queen)

If you're interested in learning more about embeddings, check this article out

Back to seq-to-seq

Now that we have representations of each word, we feed those in to an RNN to find a combined representation of the entire ***sentence***.

How do we use this to “decode” the input sentence into another language?



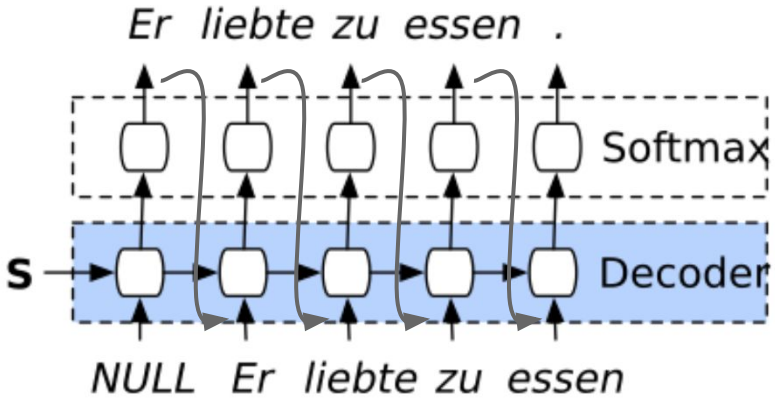
Decoders

We use this representation of our sentence to output a probability distribution over our target vocab

We pick the max of this distribution as the output

And feed this output (or maybe not???) as the input to the next step

How do we generate variable length sequences?



RNN Demo

Extensions

Encoding entire translated sequence from one vector - super impressive!!!

- This doesn't really work in practice too well…
- while LSTMs help with the vanishing gradients problem, it's still very hard to remember context from so long ago

It would be nice if at every step we could see all the previous encoded words and do something with those hidden states….

Attention mechanisms

Use all previous hidden states to predict decoded output at timestep t.

2014 -

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

2017 -

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

2018 -

TRANSFORMER-XL: ATTENTIVE LANGUAGE MODELS
BEYOND A FIXED-LENGTH CONTEXT

Zihang Dai*[1], Zhilin Yang*[2], Yiming Yang[1], Jaime Carbonell[1],
Quoc V. Le[2], Ruslan Salakhutdinov[1]
[1]Carnegie Mellon University, [2]Google Brain
{dzihang,yiming,jgc,rsalakhu}@cs.cmu.edu, {zhiliny,qvl}@google.com

MORE attention and MORE parameters == BETTER performance!!!
If you're interested check out the papers!

Where do current NLP models fail?

It seems like state of the art language models do very well in certain tasks!

One big area where modern NLP (and deep learning in general) fails - common sense reasoning.

Winograd Schema Challenge

SYSTEM PROMPT (HUMAN-WRITTEN)	*In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*
MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)	The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Taken from https://blog.openai.com/better-language-models/

Thanks!

Any questions?

Reminders:

BEER AFTER

Homework 4 Due next week!

Deliverable 2 should be submitted - Deliverable 3 due in 2 weeks!

Fill in feedback form (https://goo.gl/forms/GDhcRQUr5HmYfpNg2). We have quality feedback so far, but not a lot!