DAT560 Generative AI

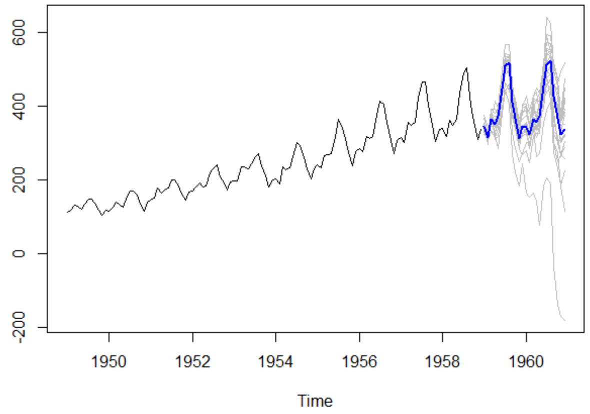
Recurrent Neural Networks

Modelling sequential Data

- What is sequential data ?
  - Data where the order matters – dependencies
- Language Modelling
  - Predict the next word
  - Which word comes next ? "the boy crossed the ____ "
    - Highly likely : *road, street, highway*
    - Less likely : pizza, sugar,..
  - If you understand language well you can generate sensible statements that are grammatically correct and capture world knowledge
- Time Series forecasting

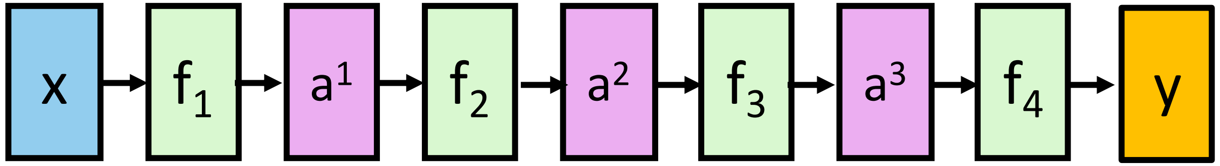
The cat crossed the street, while eating cheese, that was ____

The cat ____ the street that was flat



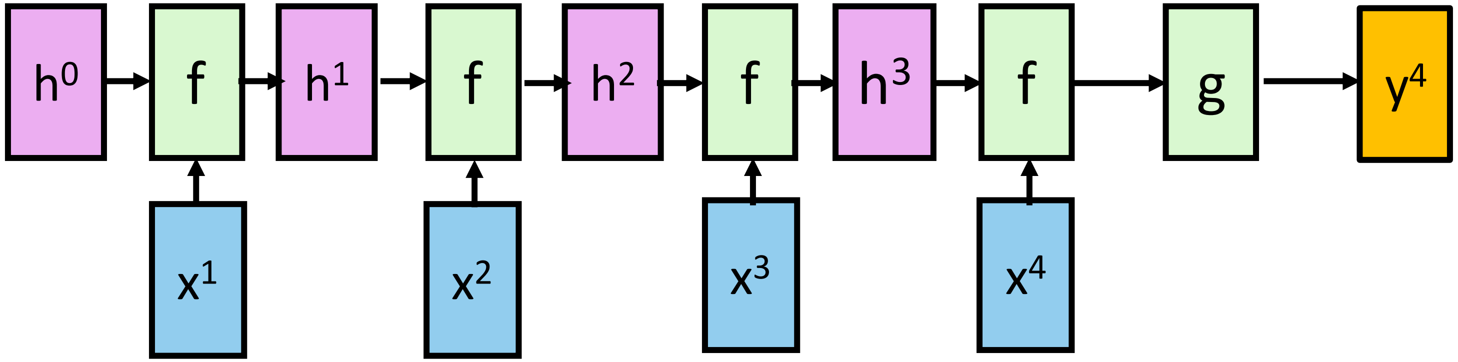
Feedforward vs Recurrent Nets

1. Feedforward network does not have input at each step
2. Feedforward network has different parameters for each layer



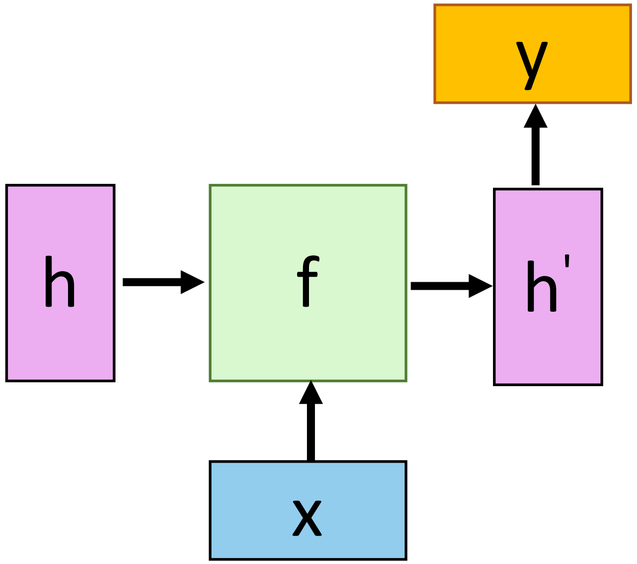
t is a layer

$$a^t = f_t(a^{t-1}) = \sigma(W^t a^{t-1} + b^t)$$

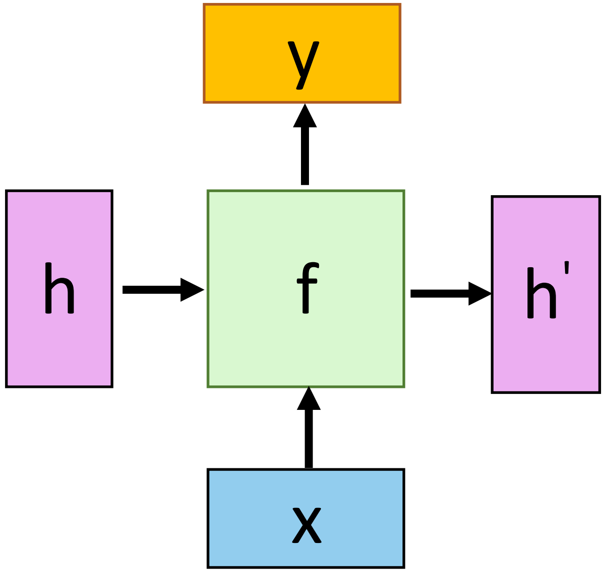


t is time step

RNN Unit Computation

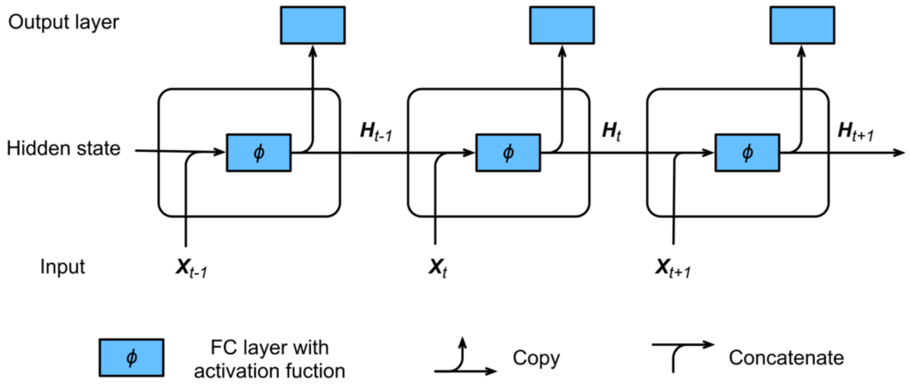


Note, y is computed from h’



$$h' = \underset{\text{sigmoid}}{\sigma}\left(W h + U x \right)$$

$$y = \underset{\text{softmax}}{\sigma}\left(W^{o} h' \right)$$

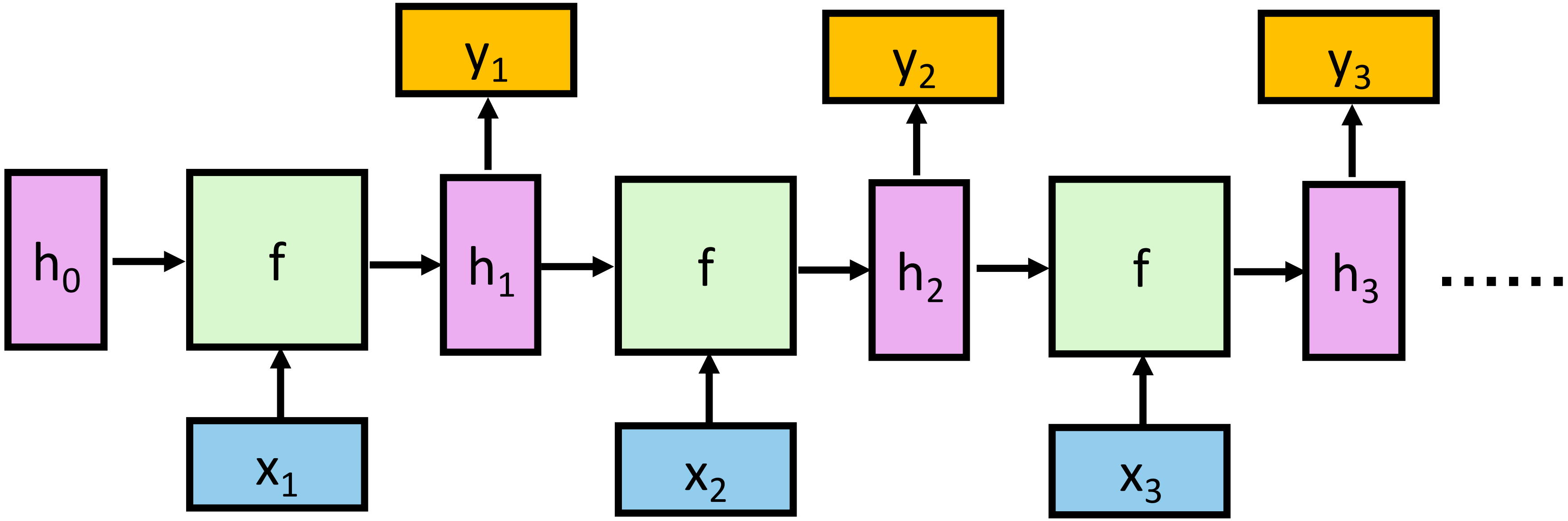


RNN Cell Diagram

Recurrent Neural Network

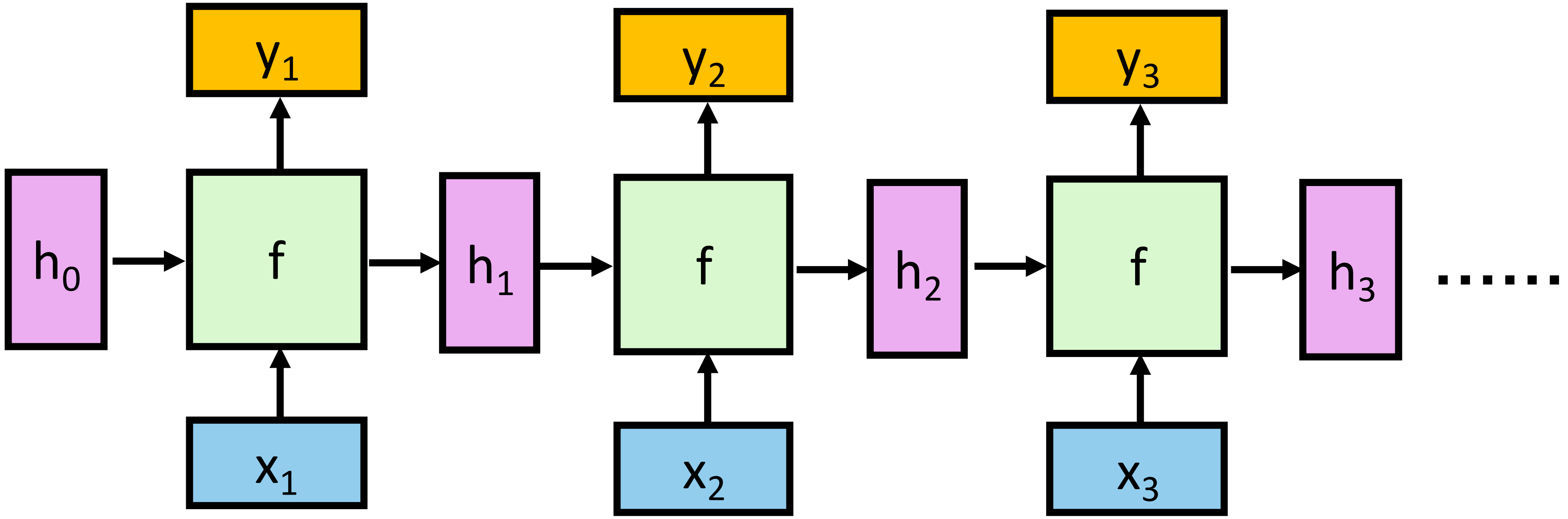
- Given function f: $h', y = f(h,x)$

h and h' are vectors with the same dimension



No matter how long the input/output sequence is, we only need one function f. If f's are different, then it becomes a feedforward NN. This may be treated as another compression from fully connected network.

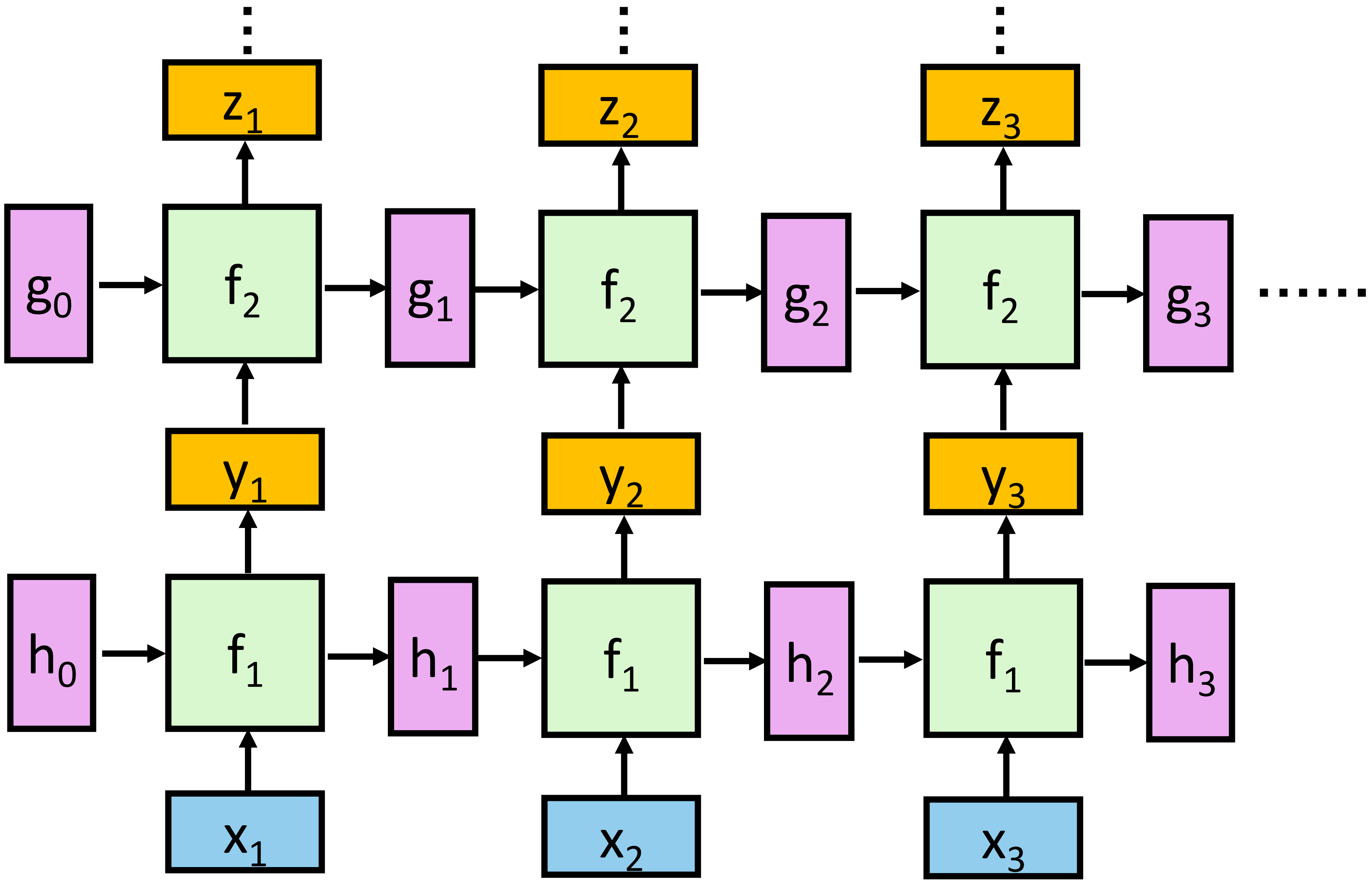
Forward Pass



$$h_1 = f(W^T h_0 + U^T x_1)$$
$$h_2 = f\left(W^T f(W^T h_0 + U^T x_1) + U^T x_2\right)$$
$$h_3 = f\left(W^T f\left(W^T f(W^T h_0 + U^T x_1) + U^T x_2\right) + U^T x_3\right)$$

Weights are shared

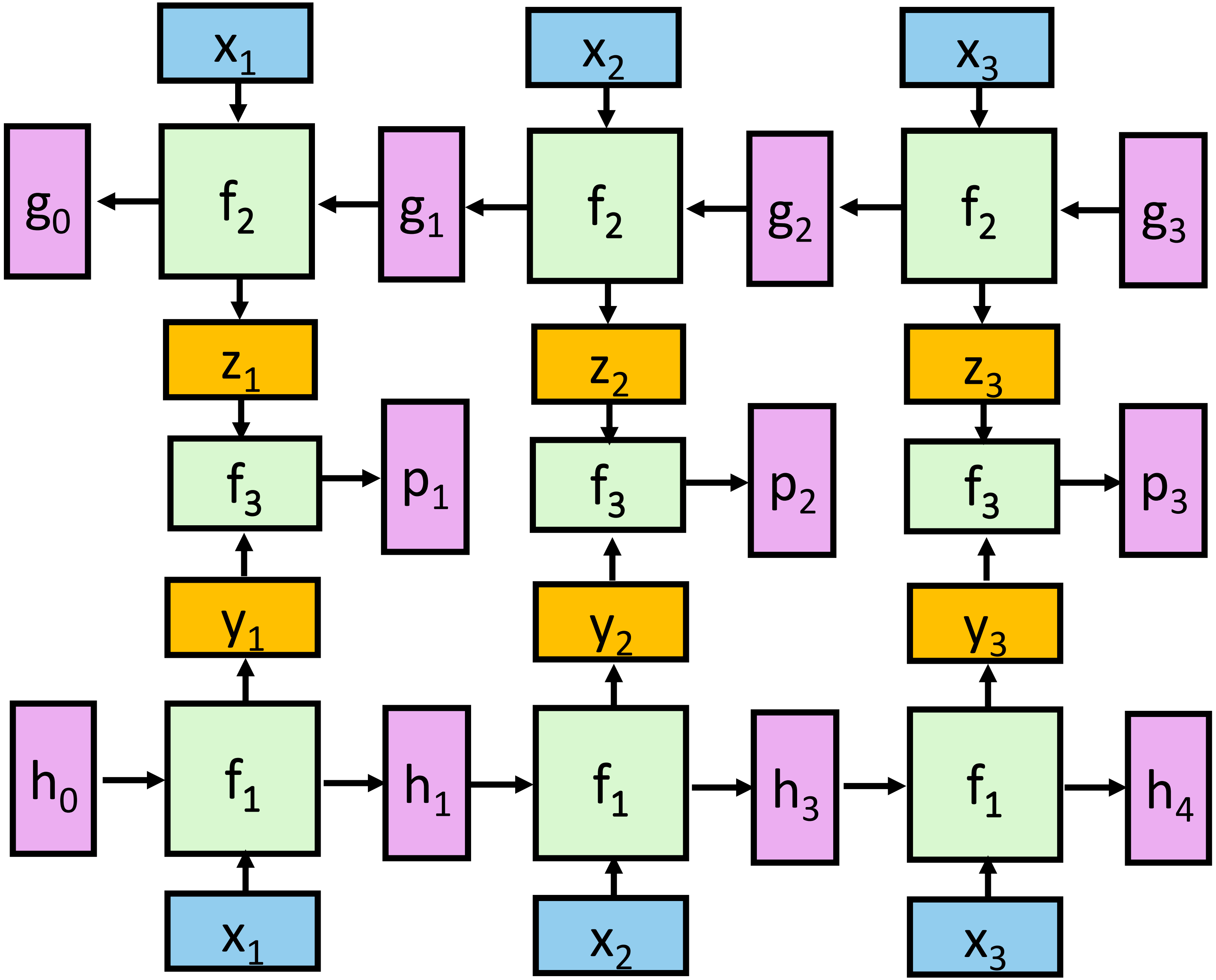
Deep RNN



$h', y = f_1(h, x)$

$g', z = f_2(g, y)$

Bi- Directional RNN



$y,h=f_1(x,h)$

$z,g = f_2(g,x)$

$p=f_3(y,z)$

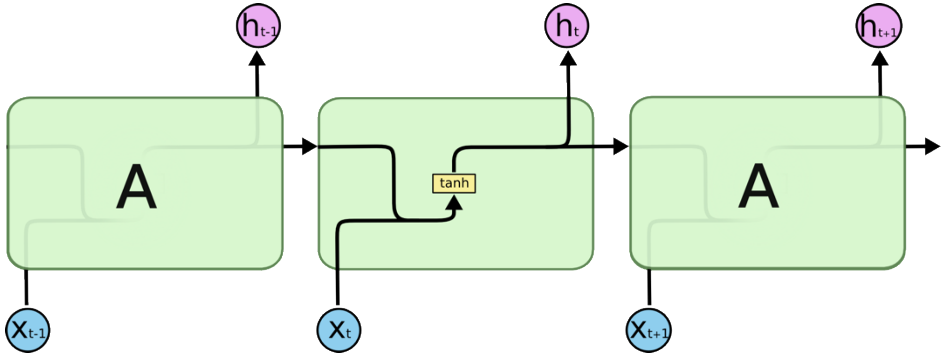
The cat ____ the street that was flat

Problems with vanilla RNNs

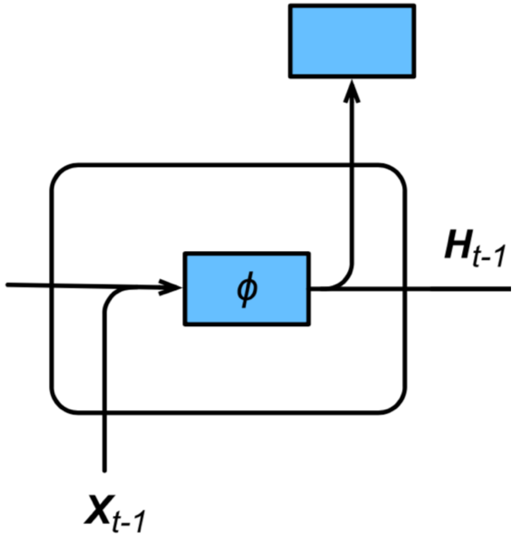
- Inability to capture long-term dependencies
  - When dealing with a time series, it tends to forget old information. When there is a distant relationship of unknown length, we wish to have a “memory” to it.

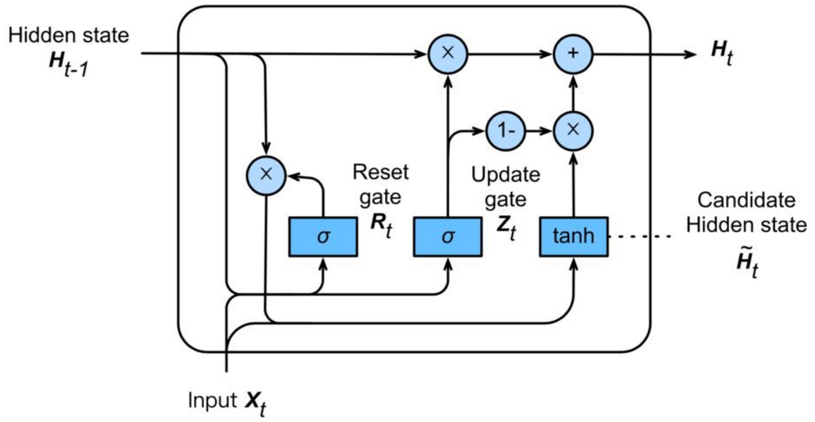
- Vanishing and Exploding gradients
  - Weights either become zero or explode due to products of partial differentials

- Slow inference

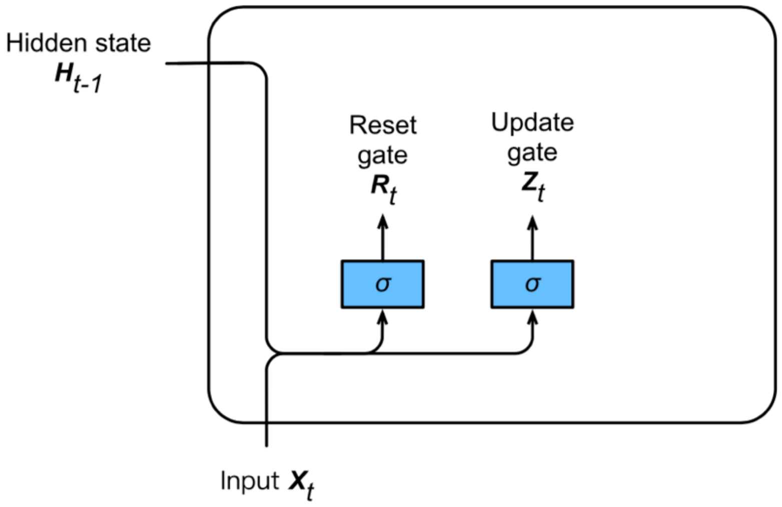


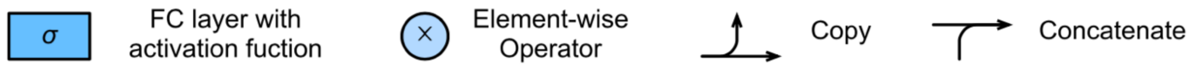
Gated Recurrent Unit





Reset and Update Gates





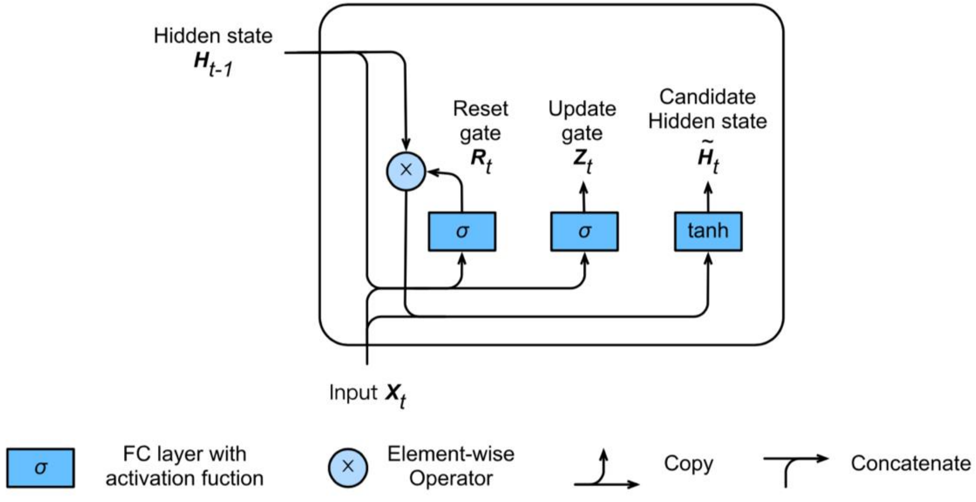
GRUs have the following two distinguishing features:

- Reset gates help capture short-term dependencies in time series.
- Update gates help capture long-term dependencies in time series.

$$\mathbf{R}_t = \sigma\left(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \mathbf{b}_r\right)$$
$$\mathbf{Z}_t = \sigma\left(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \mathbf{b}_z\right)$$

Reset gate

- If we want to be able to reduce the influence of previous states
  - multiply $\mathbf{H}_{t-1}$ with $\mathbf{R}_t$ elementwise
- Whenever the entries in R_t are close to 1 we recover a conventional deep RNN.
- For all entries of R_t that are close to 0 the hidden state is the result of an MLP with X_t as input
- Any pre-existing hidden state is thus 'reset' to defaults. This leads to the following candidate for a new hidden state (it is a candidate since we still need to incorporate the action of the update gate).



$$\mathbf{H}_t = \tanh\left(\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{H}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h\right)$$

$$\tilde{\mathbf{H}}_t = \tanh\left(\mathbf{X}_t \mathbf{W}_{xh} + \left(\mathbf{R}_t \odot \mathbf{H}_{t-1}\right) \mathbf{W}_{hh} + \mathbf{b}_h\right)$$

Nonlinearity (Tanh) to ensure that the values of the hidden state (−1, 1)

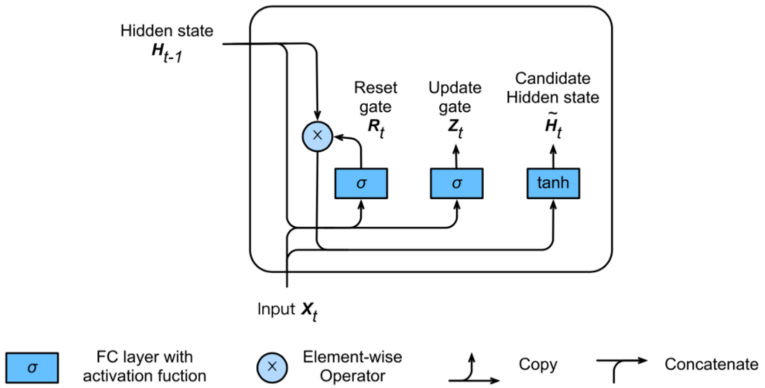
Update gate

- Determines the extent to which the new state H_t is just the old state and by how much the new candidate state is used.

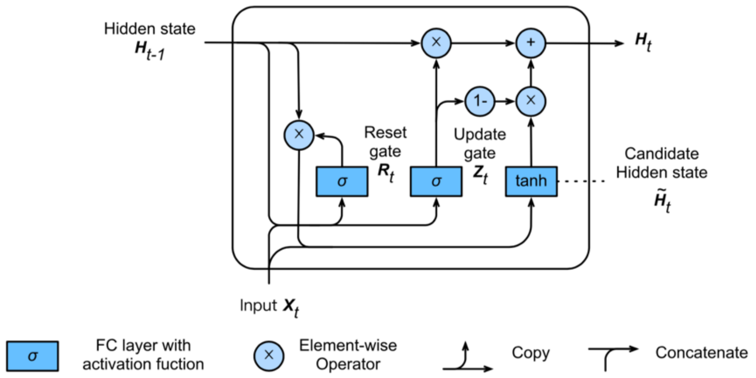
$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t$$

- Whenever the update gate is close to 1
  - we simply retain the old state.
  - In this case the information from X_t is essentially ignored, effectively skipping time step t in the dependency chain

- Whenever it is close to 0
  - the new latent state H_t approaches the candidate latent state $\tilde{H}_t$.
  - Helps cope with the vanishing gradient problem in RNNs and better capture dependencies for time series with large time step distances.



$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t$$

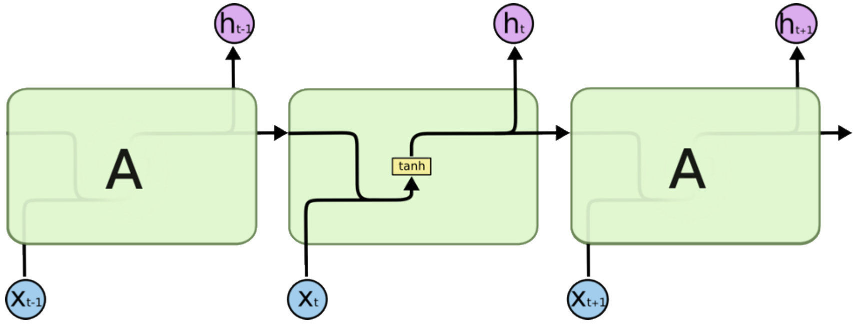


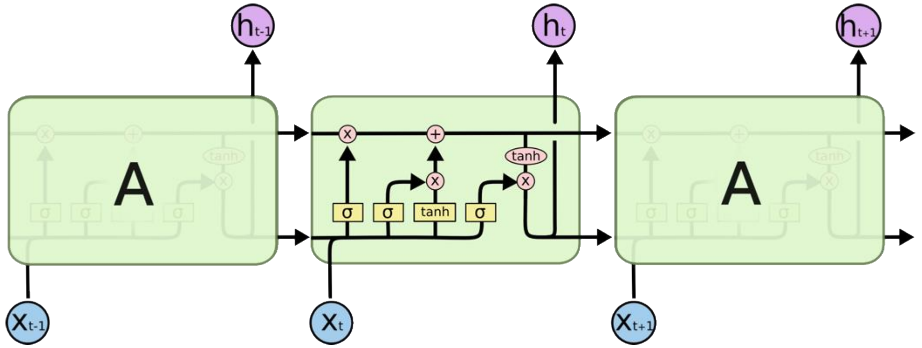
LSTM Networks

- Long Short Term Memory networks – usually just called "LSTMs"
- LSTM was first proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber (Neural Computation. 9 (8): 1735–1780.)
  - Submission to NIPS was rejected in 1997!
- It was used by (until Transformers came in)
  - Microsoft for conversational speech recognition
  - Google for machine translation
  - Apple for siri (on iphones)
  - IBM, Baidu, Samsung and so on…

LSTM Networks

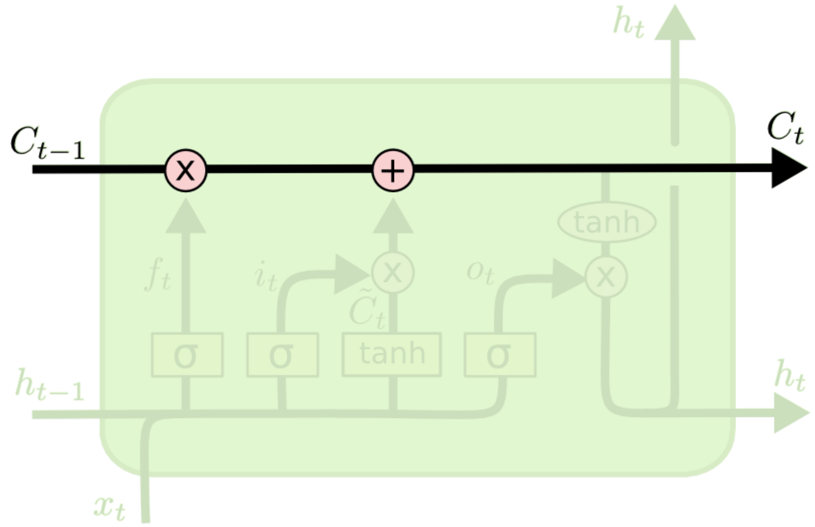
- LSTMs are designed to overcome vanishing gradient problems of RNN





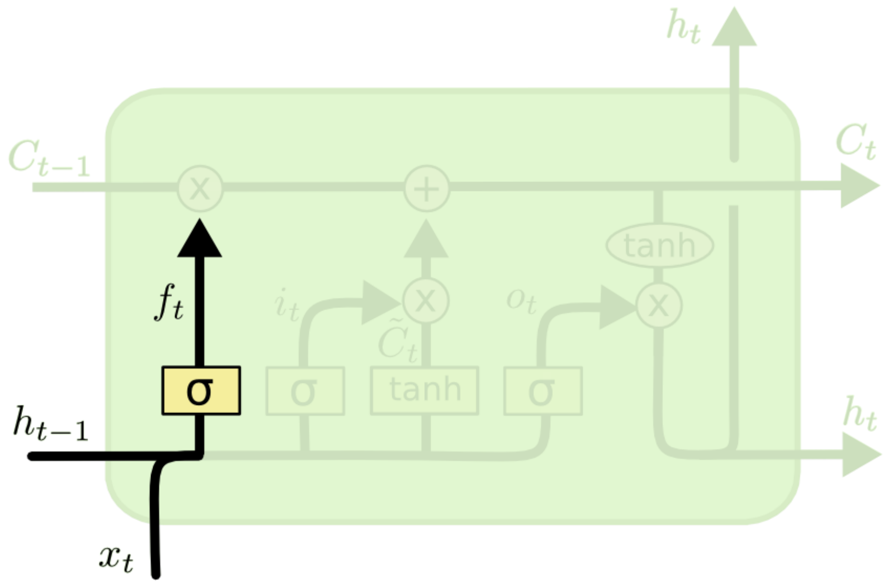
The Core Idea Behind LSTMs

- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
- The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



Step-by-Step LSTM Walk Through

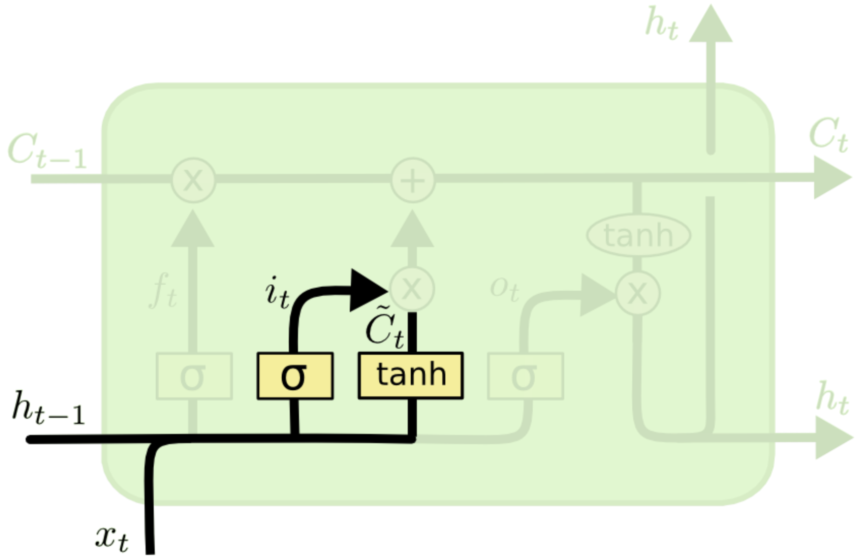
- Forget gate: Decides what information to throw away from the cell state
- *f_t is between 0 and 1 due to sigmoid*
- 1 represents “completely keep this” while a 0 represents “completely get rid of this.”



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

Step-by-Step LSTM Walk Through

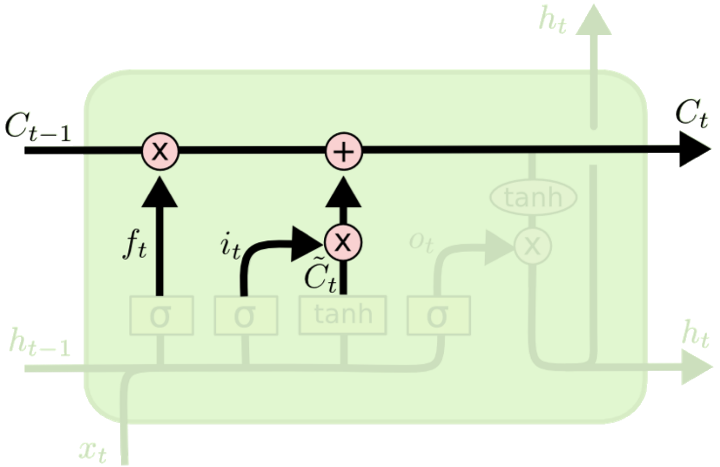
- Input gate: Decides what new information to store in the cell state.
- This has two parts.
  - a sigmoid layer called the "input gate layer"
  - a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Step-by-Step LSTM Walk Through

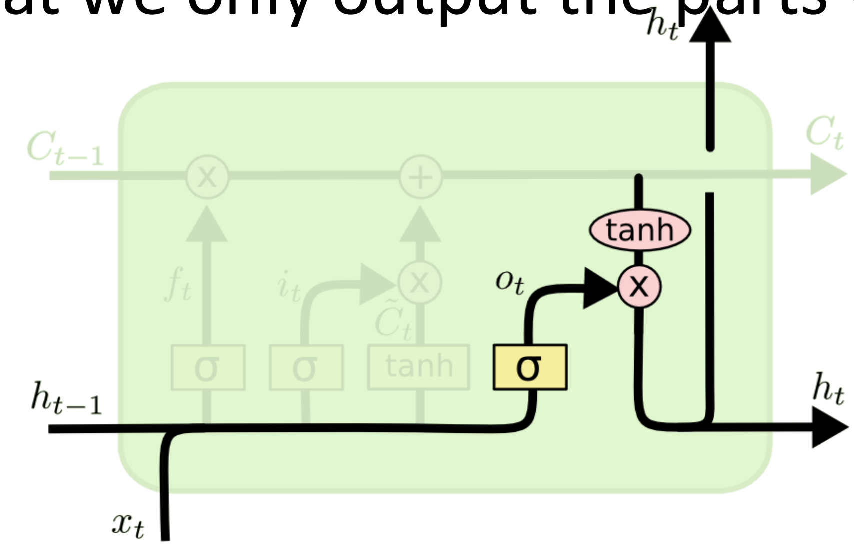
- Next step is to update the old cell state, C_{t-1}, into the new cell state C_t
- We multiply the old state by forget gate output f_t, Then we add $i_t * \tilde{C}_t$.



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

Step-by-Step LSTM Walk Through

- Finally the output will be based on our cell state, but will be a filtered version.
- First, we run a sigmoid layer which decides what parts of the cell state we're going to output.
- Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



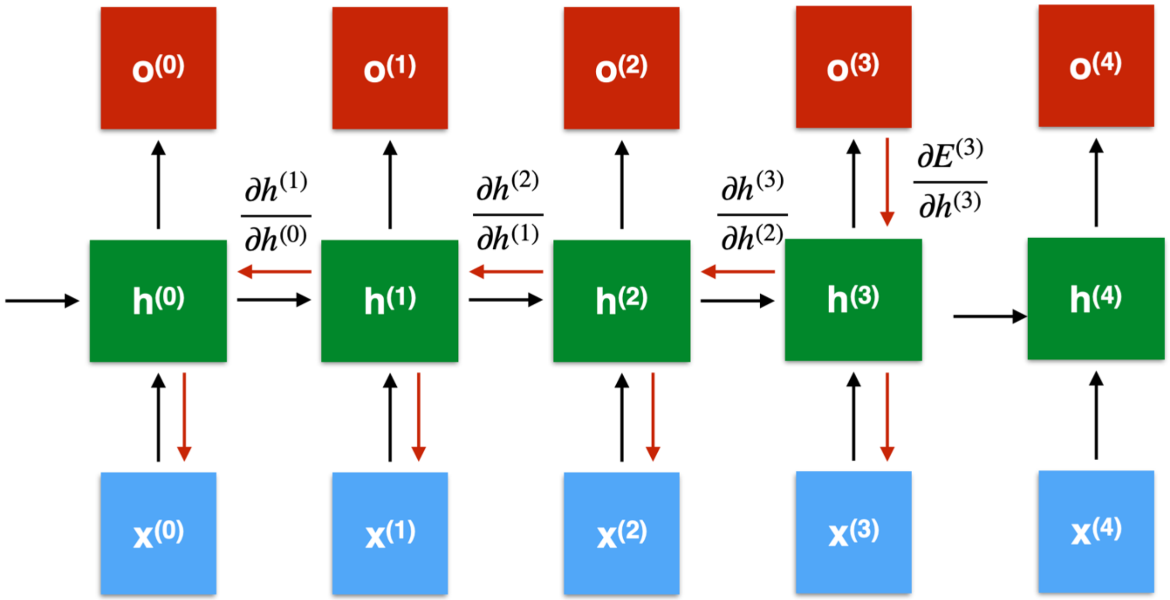
$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

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

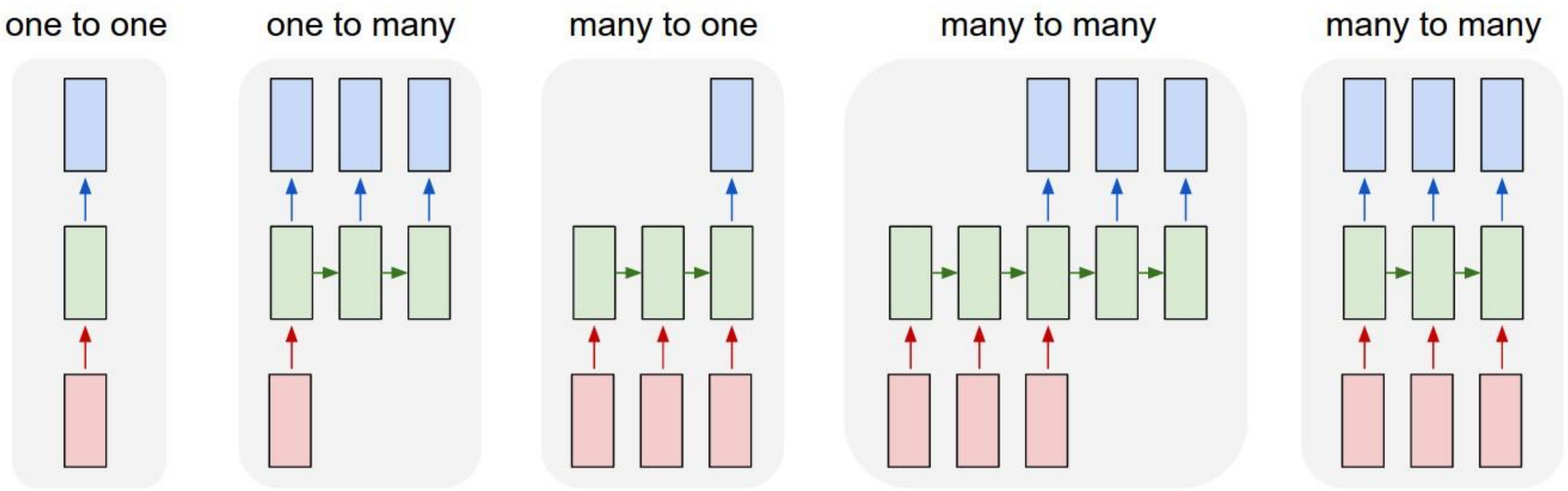
Back Propagation through Time

- One of the methods used to train RNNs
- The unfolded network (used during forward pass) is treated as one big feed-forward network
- This unfolded network accepts the whole time series as input

- The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and then applied to the RNN weights



Different Scenarios



Sentiment Classification

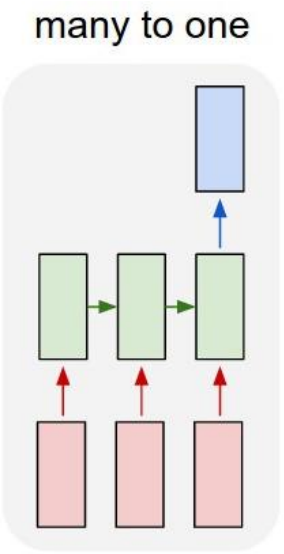
Machine Translation, Language Modeling

POS tagging

Sentiment Classification

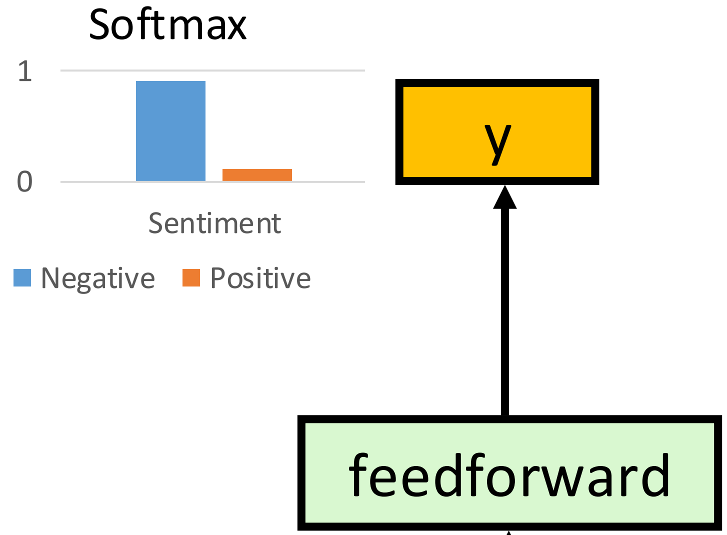
- **Task:** Given a review (natural language text) classify it as a positive or a negative sentiment
  - "the movie is pathetic" → {positive, negative}

- Input: pre-trained word embeddings
- Each cell is a GRU cell
- Loss function: Cross entropy loss

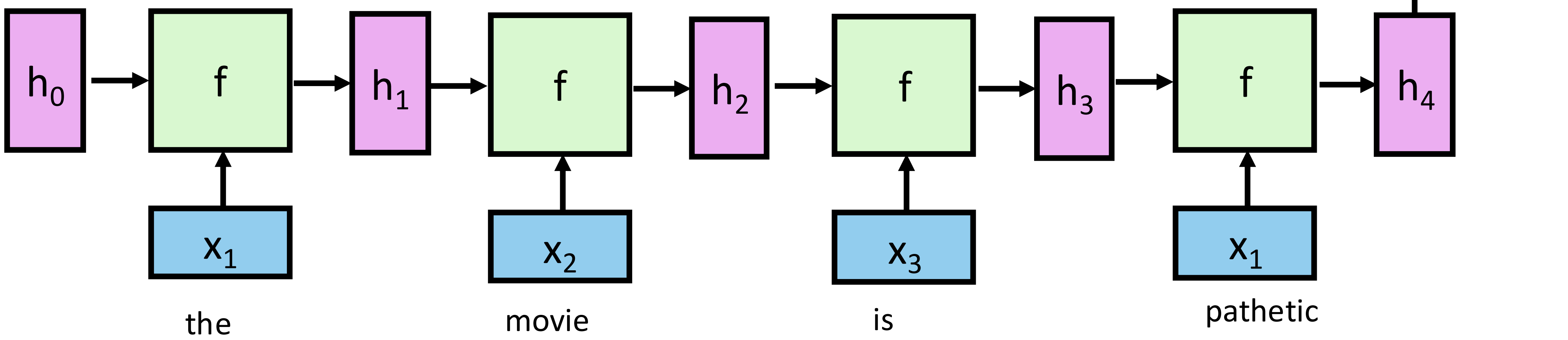


Simple RNN for Sentiment Classification

- **Task:** Given a review (natural language text) classify it as a positive or a negative sentiment
  - “the movie is pathetic” → {positive, negative}



- Input: pre-trained word embeddings
- Each cell is a GRU cell
- Loss function: Cross entropy loss



References

- Luis Serrano, A Friendly Introduction to Recurrent Neural Networks, https://www.youtube.com/watch?v=UNmqTiOnRfg, Aug. 2018
- Brandon Rohrer, Recurrent Neural Networks (RNN) and Long, Short-Term Memory (LSTM), https://www.youtube.com/watch?v=WCUNPb-5EYI, Jun. 2017
- Denny Britz, Recurrent Neural Networks Tutorial, http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/, Sept. 2015 (Implementation)
- Colah's blog, Understanding LSTM Networks, http://colah.github.io/posts/2015-08-Understanding-LSTMs/, Aug. 2015

- https://distill.pub/2019/memorization-in-rnns/
- http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Slides credit

- Avishek Anand