Deep Learning - 2019

Recurrent Neural Networks

Prof. Avishek Anand

What we learnt so far ?

- Linearization of input might not be the desired option where locality matters
- Convolutional Neural Networks
  - Filters that exploit parameter sharing, sparsity and translation invariance
  - Pooling layers that reduce resolution without compromising representation quality

- CNNs have been at the forefront of vision related tasks
  - Also used in text, graphs
  - Large datasets and deeper models have fuelled success

- Deeper layers are better but have to be careful about
  - Vanishing and exploding gradients
  - Better optimization techniques help
  - Residual connections and Residual Networks for effective deeper convnets

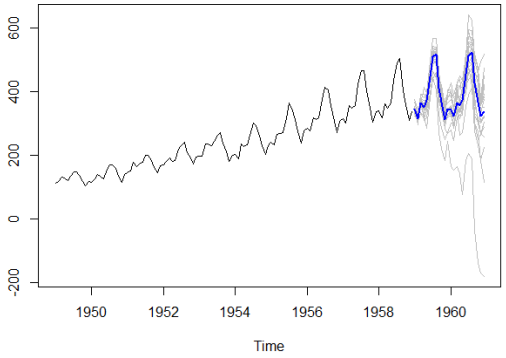
- Today we talk about another common input type where sequential information is involved

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

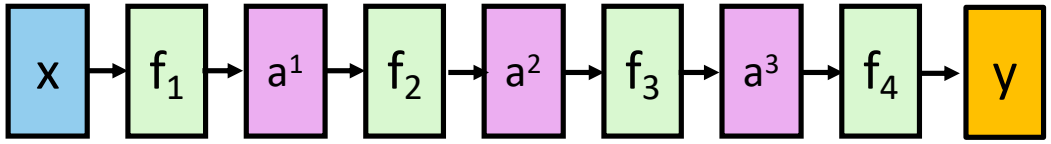
The cat crossed the street, while eating cheese, that was ____

The cat ____ the street that was flat



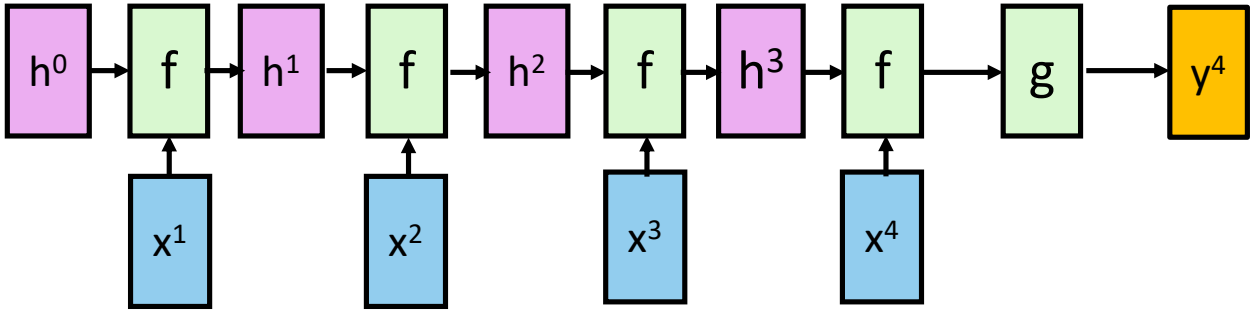
Feedforward vs Recurrent Nets

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



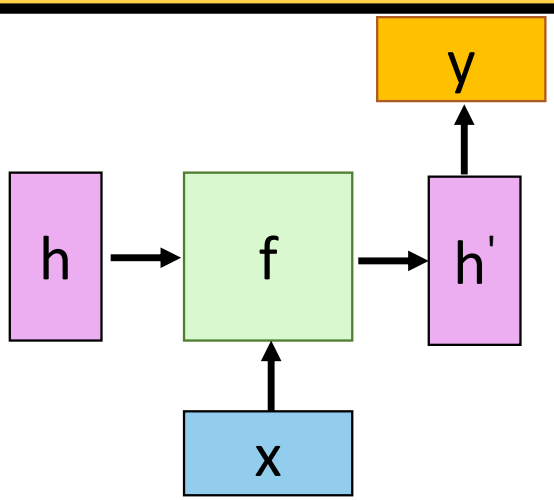
t is layer

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

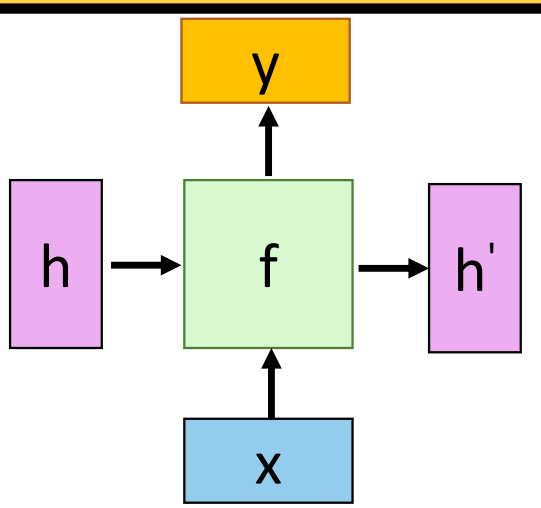


t is time step

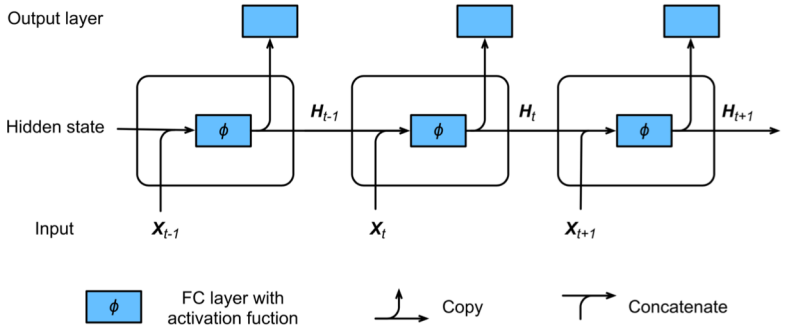
RNN Unit Computation



Note, y is computed from h'



$$h' = \sigma_{\text{softmax}}\left(W \begin{bmatrix} h \\ x \end{bmatrix} \right) \qquad y = \sigma\left(W^{o} h' \right)$$

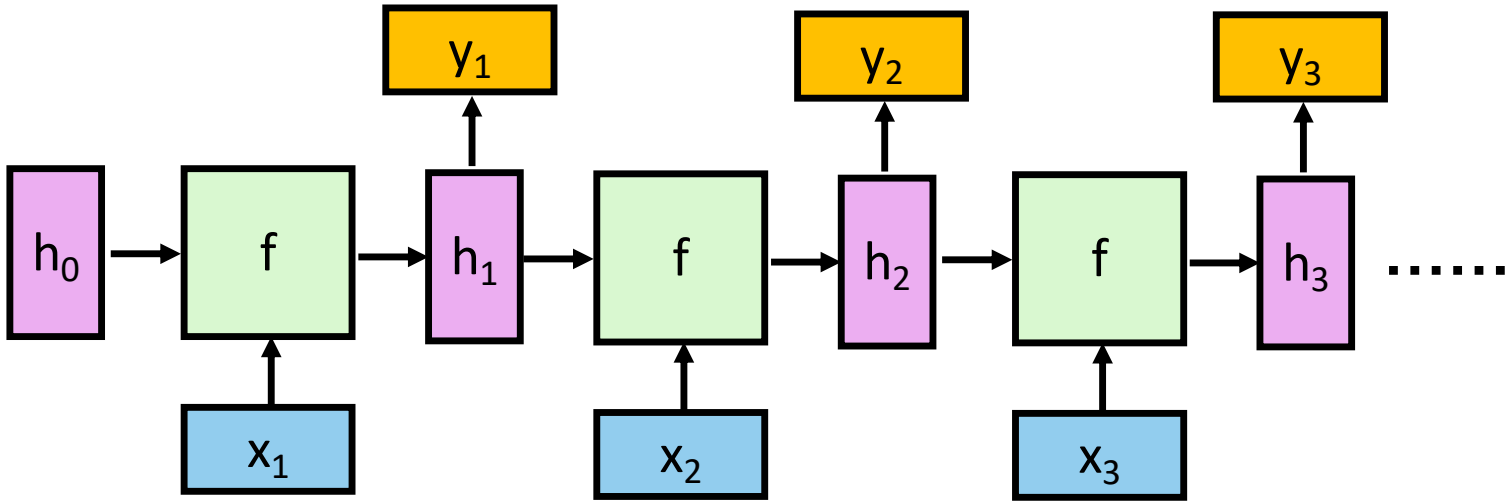


RNN Cell Diagram

Recurrent Neural Network

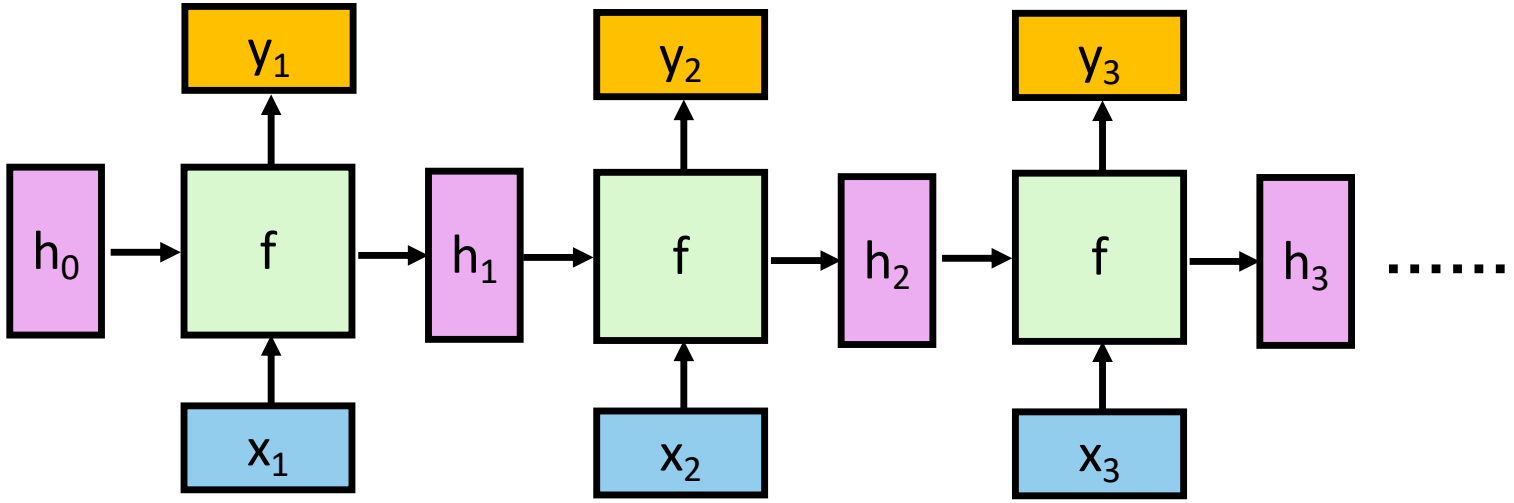
- Given function f: h', y = f(h,x)

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.

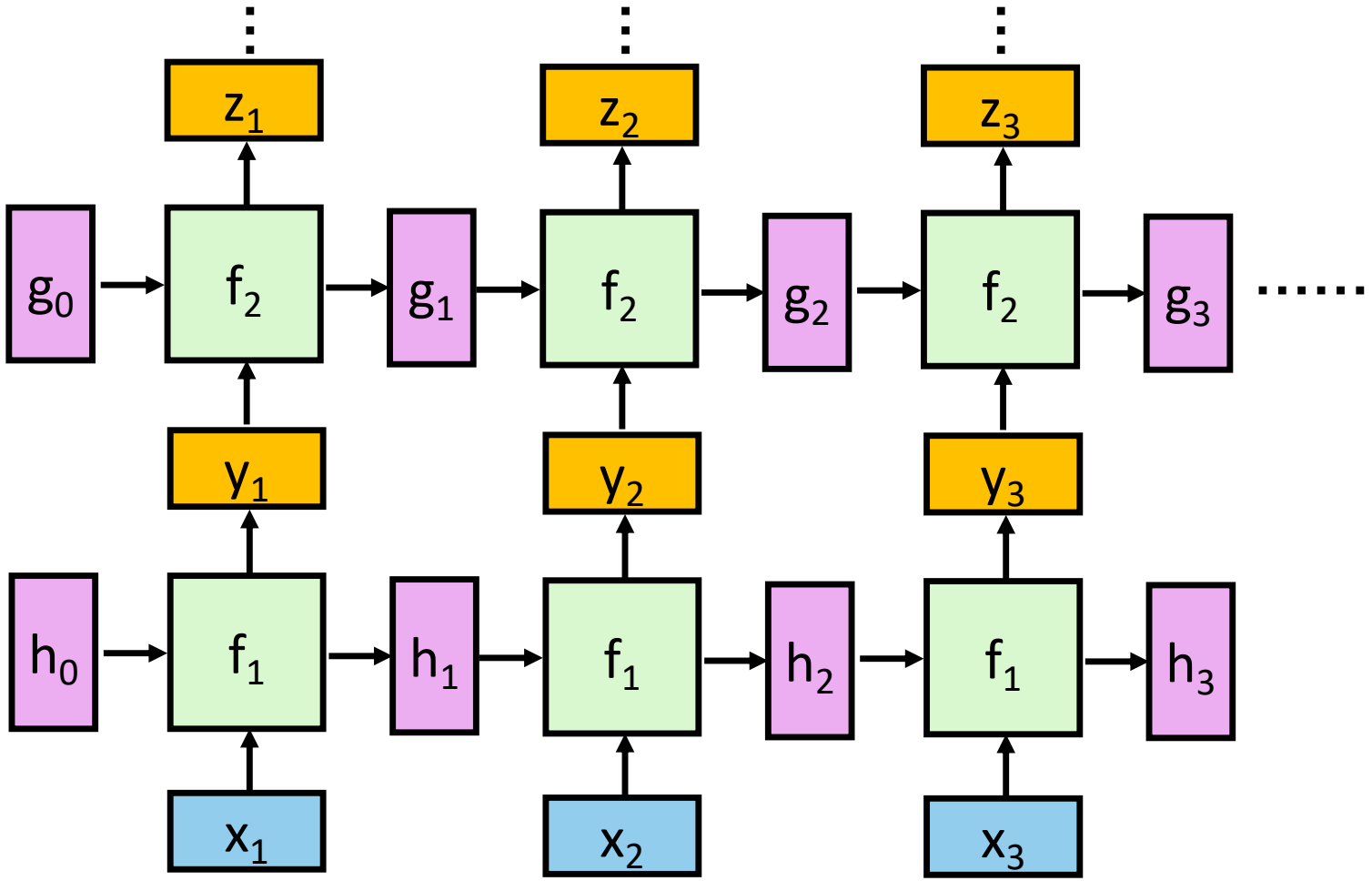
Forward Pass



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

Weights are shared

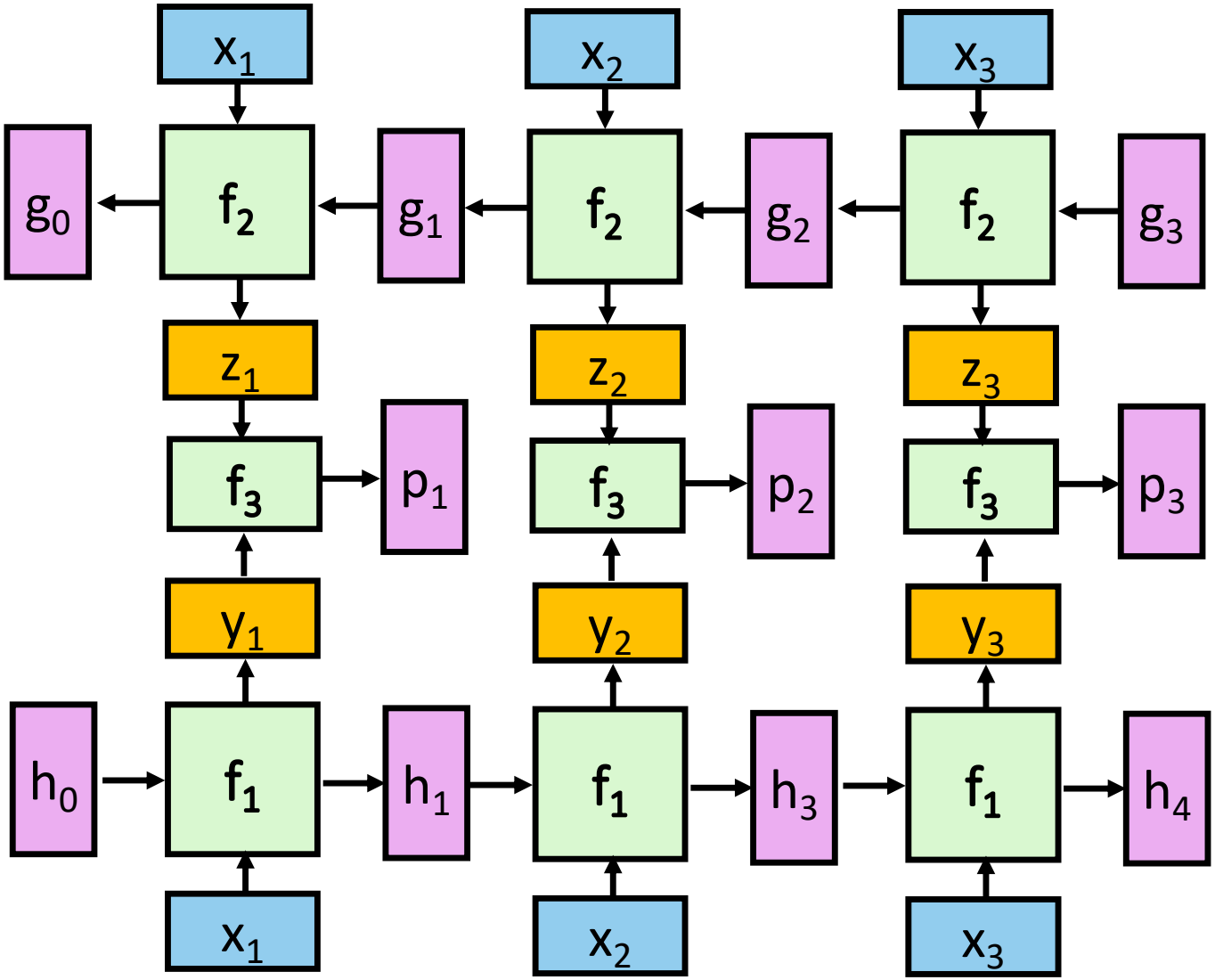
Deep RNN



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

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

Bi- Directional RNN



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

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

$p=f_3(y,z)$

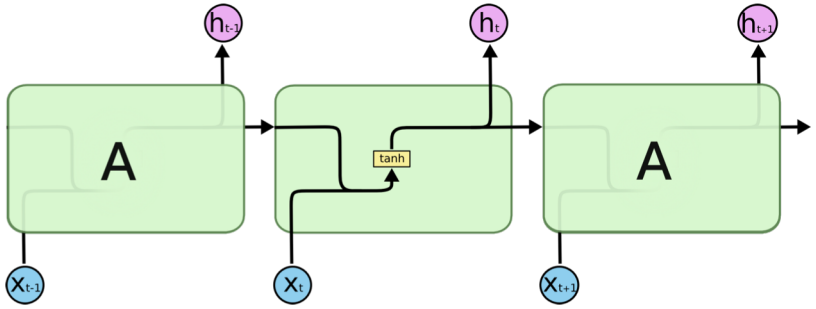
The cat ____ the street that was flat

Problems with vanilla RNNs

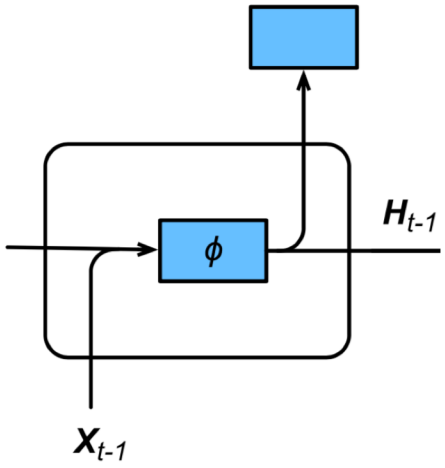
- Inability to capture long-term dependencies
  - When dealing with a time series, it tends to forget old information. When there is a distant relationship of unknown length, we wish to have a “memory” to it.

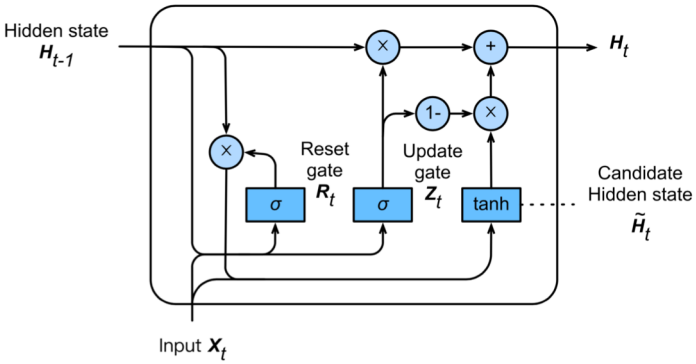
- Vanishing and Exploding gradients
  - Weights either become zero or explode due to products of partial differentials (refer lecture on CNNs)

- Slow inference

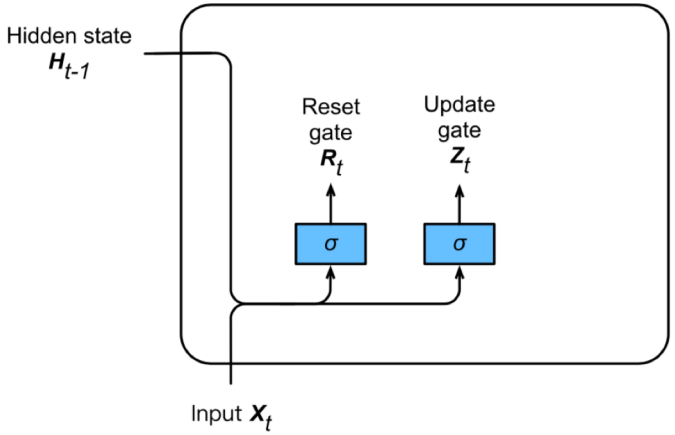


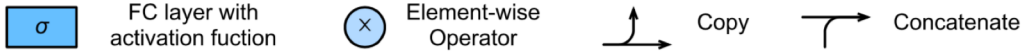
Gated Recurrent Unit





Reset and Update Gates





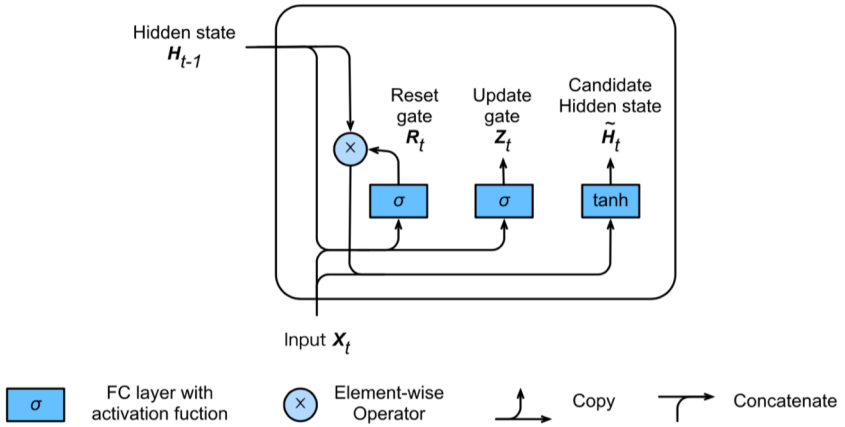
GRUs have the following two distinguishing features:

- 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 $\mathbf{R}_t$ are close to 1 we recover a conventional deep RNN.
- For all entries of $\mathbf{R}_t$ that are close to 0 the hidden state is the result of an MLP with $\mathbf{X}_t$ 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).



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

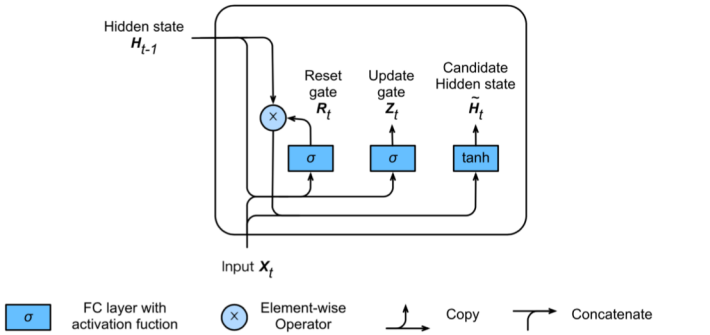
$$\tilde{\mathbf{H}}_t = \tanh\left(\mathbf{X}_t \mathbf{W}_{xh} + \left(\mathbf{R}_t \odot \mathbf{H}_{t-1}\right) \mathbf{W}_{hh} + \mathbf{b}_h\right)$$

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

Forget 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.

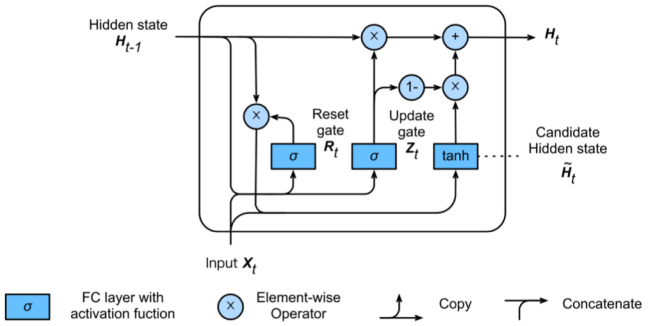
$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t$$



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

- 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

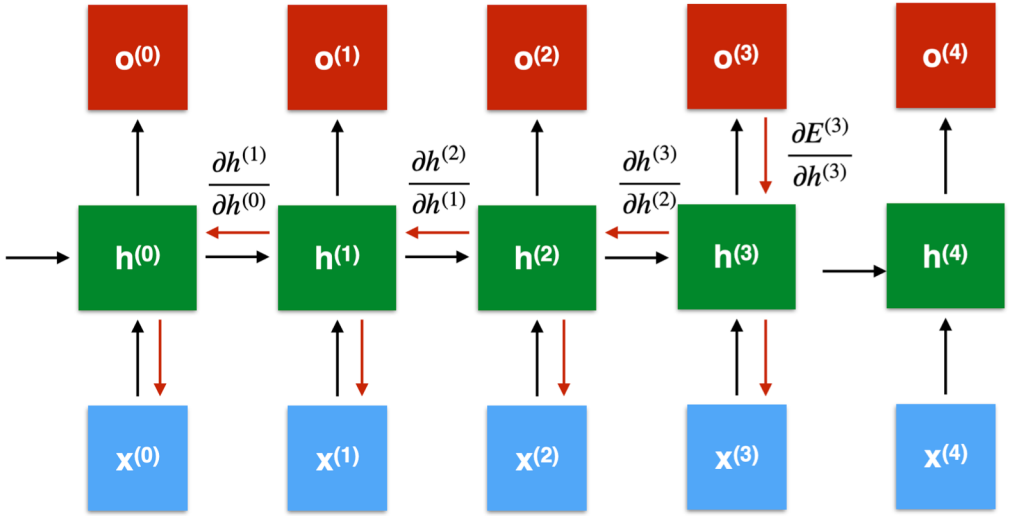
- Whenever it is close to 1
  - the new latent state H_t approaches the candidate latent state $\tilde{H}_t$.
  - Helps cope with the vanishing gradient problem in RNNs and better capture dependencies for time series with large time step distances.



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

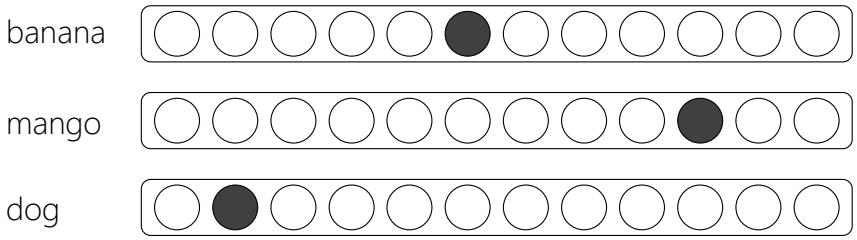
- The weight updates are computed for each copy in the unfolded network, then summed (or averaged) and then applied to the RNN weights



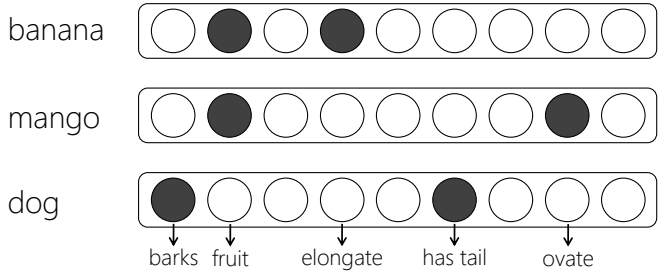
Word Representations

- Document representation issues:
  - High-dimensionality
  - In vector space, this is a sparse vector with one 1 and a lot of zeros.
  - 1-Hot representation: [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]

- **Problem**
  - **Dimensionality high: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)**
  - **Vocabulary Mismatch**



Local representations (1-hot)



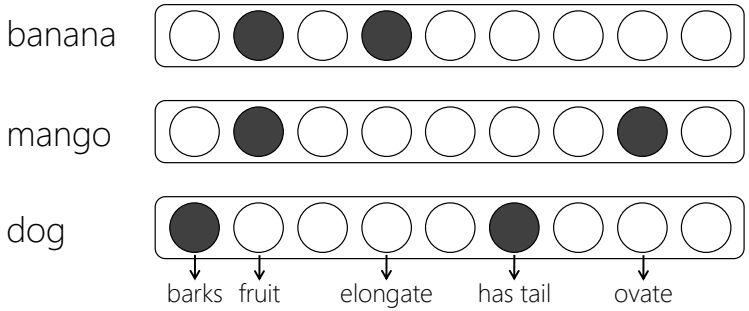
Distributed representations

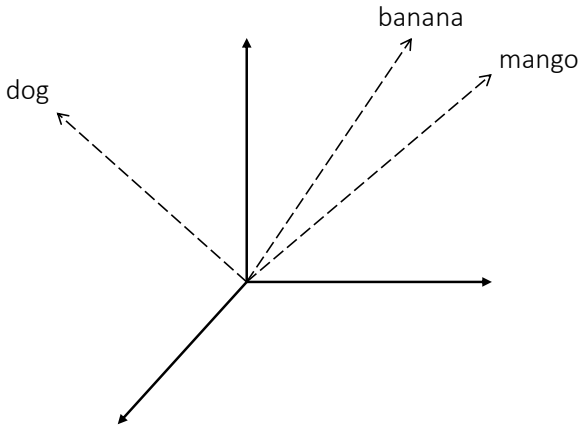
Word Representations

- You can get a lot of value by representing a word by means of its neighbors
  - "You shall know a word by the company it keeps" — (J. R. Firth 1957: 11)

...government debt problems turning into banking crises as has happened in...
.....saying that Europe needs unified banking regulation to replace the hodgepodge...

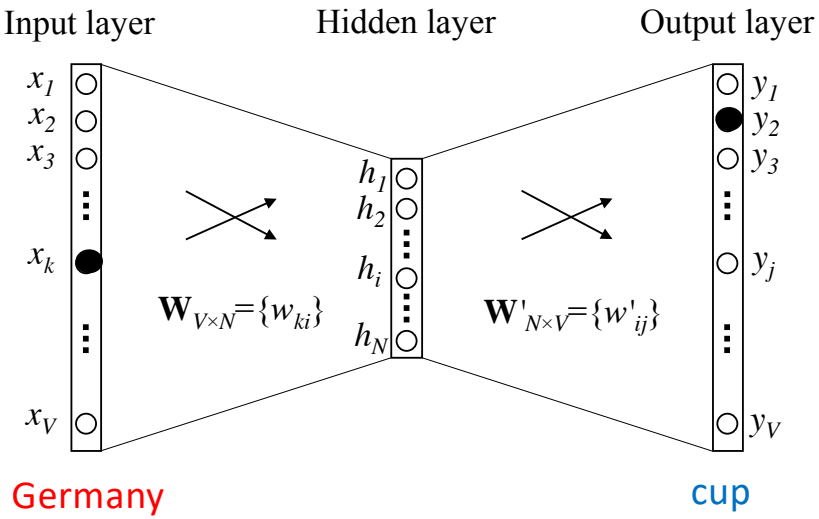
↖ These words will represent *banking* ↗





Word Embeddings

- Considers learning word embeddings as a prediction t '
- Training context : "germany wins cup in brazil"
- **Continuous Bag of Words Model (CBOW):**
  - Guess the central word in the context
  - "germany wins X in brazil"
  - Training data (Germany, cup), (wins, cup), (brazil, cup)
- **Skip-gram**
  - Guess the context given the central word
  - "X X cup X X"
  - Training data (cup, Germany), (cup, wins), (cup, brazil)



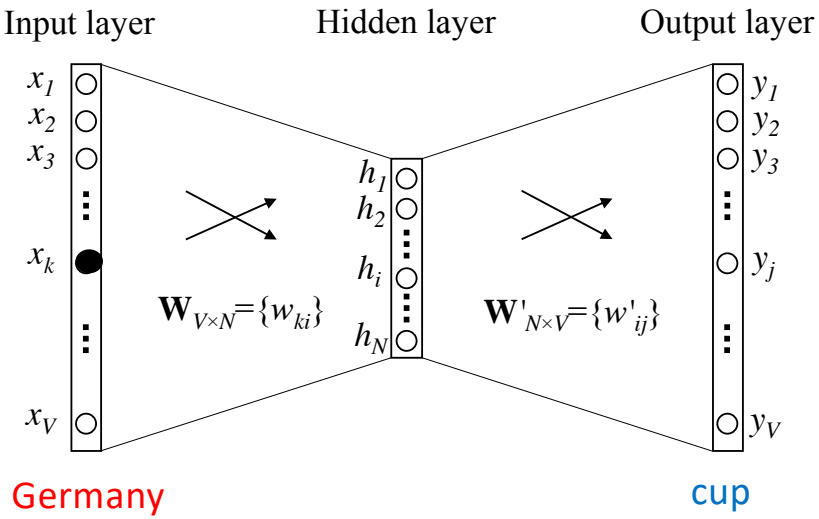
Is it a regression of classification problem ?

Word Representations

- Input and Output dimensions are size V (vocabulary), hidden layer N << V

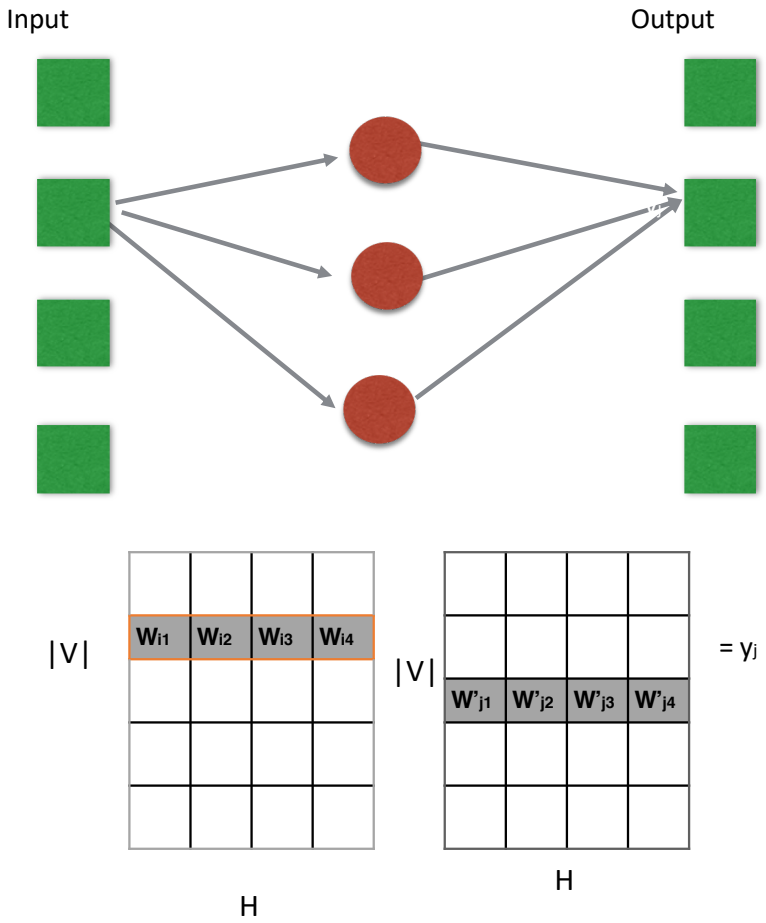
- Input representation : **h** in the hidden space
- Output representation : **h.W'**

- Central question: how do we learn the parameters of this model (weights) given training examples from text
- What is cosine distance between output and input representation ?

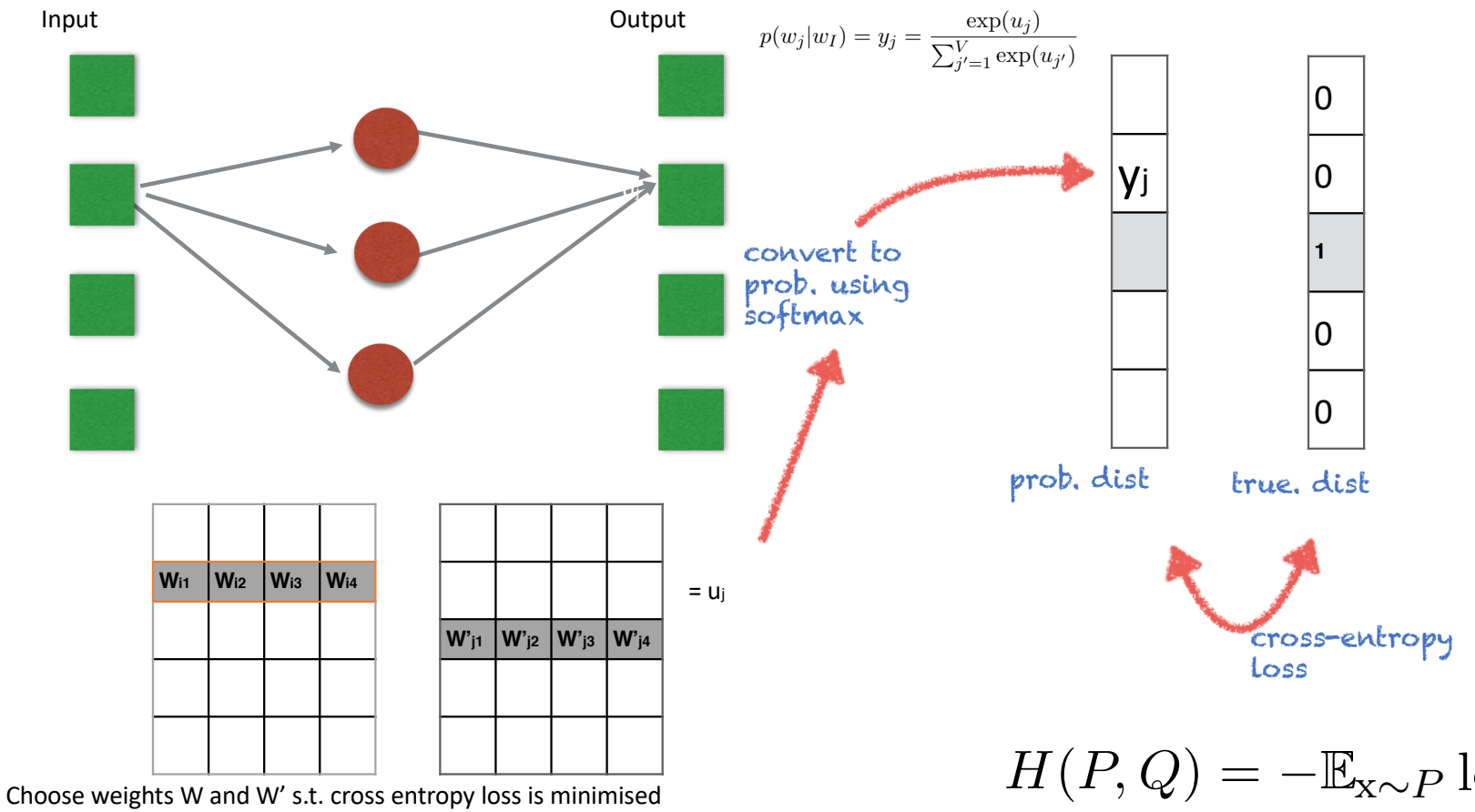


Word Representations

- Cosine similarity at each output node is the measure of similarity w.r.t the given configuration of weights (prarams)
- Loss = comparison between the expected output and current *output distributions*
  - Construct a distribution (softmax function)
  - Use cross-entropy for loss

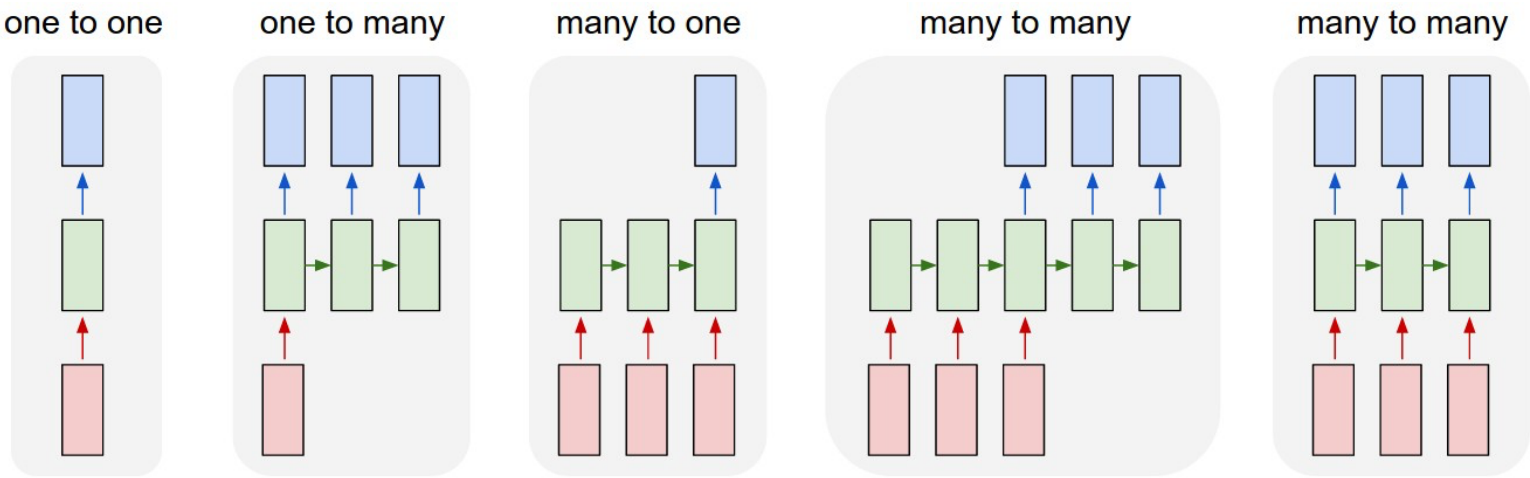


Forward Pass



$$H(P,Q) = -\mathbb{E}_{x\sim P} \log Q(x)$$

Different Scenarios



Sentiment Classification

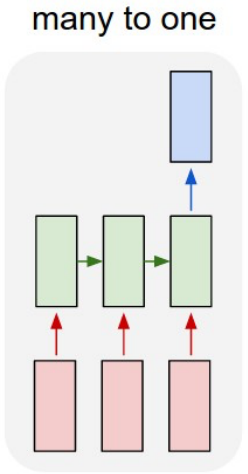
Machine Translation, Language Modeling

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}

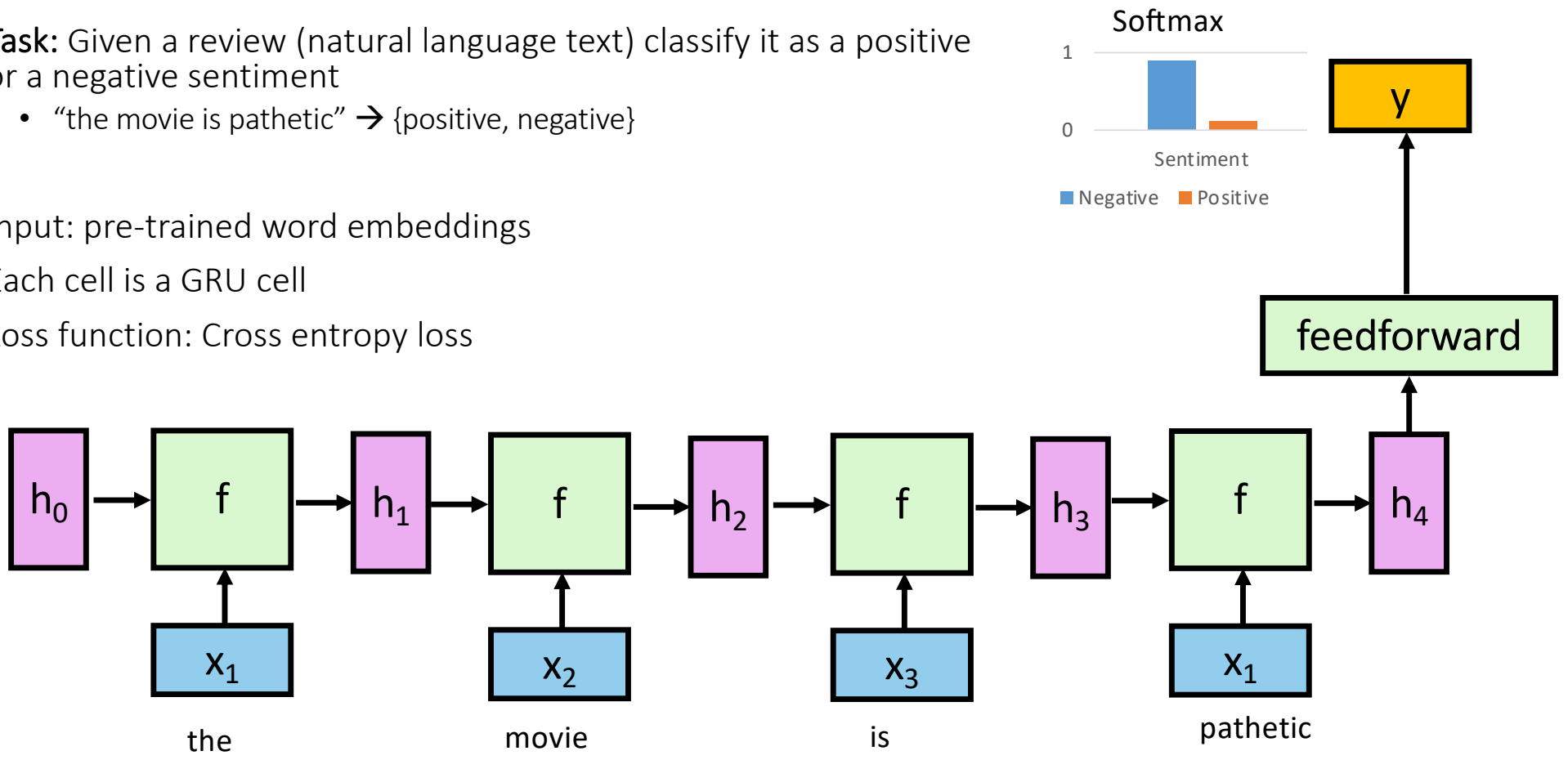
- Input: pre-trained word embeddings
- Each cell is a GRU cell
- Loss function: Cross entropy loss



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/