NATURAL LANGUAGE PROCESSING

WITH DEEP LEARNING

University of Guilan

Lecturer: Javad PourMostafa

NLP981

RECAP (POS WITH HMM)

- Imaging $x = x_1, x_2, \cdots, x_n$ be visible words and
- $y = y_1, y_2, \cdots, y_n$ be corresponding hidden tags
- Find the correct formula for Hidden Markov Model

$$p(x, y) = \mathrm{p(x|y)\ p(y)} = \prod_{t=1}^{T} p(x_t | y_t)\, p(y_t |\, y_{t-1})$$

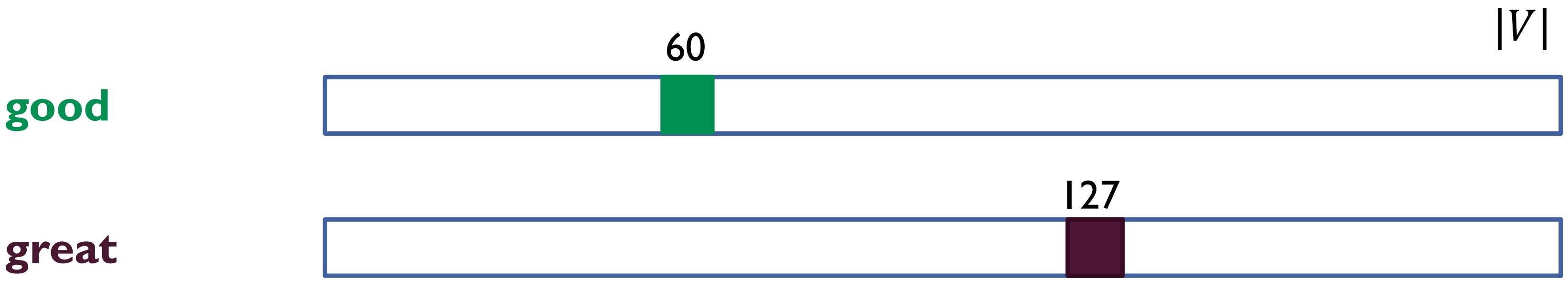
- Finally:
- Using Viterbi algorithm at each time step to find dynamically the most probable sequence of hidden tags.

RECAP (LANGUAGE MODELING)

- Could you remember n-gram Language Modeling (LM)?
- E.g. :
  - Have a good day
- 4-gram
- $P(day|\ Have\ a\ good) = \frac{C(Have\ a\ good\ day)}{C(Have\ a\ good)}$
- Use some smoothing techniques

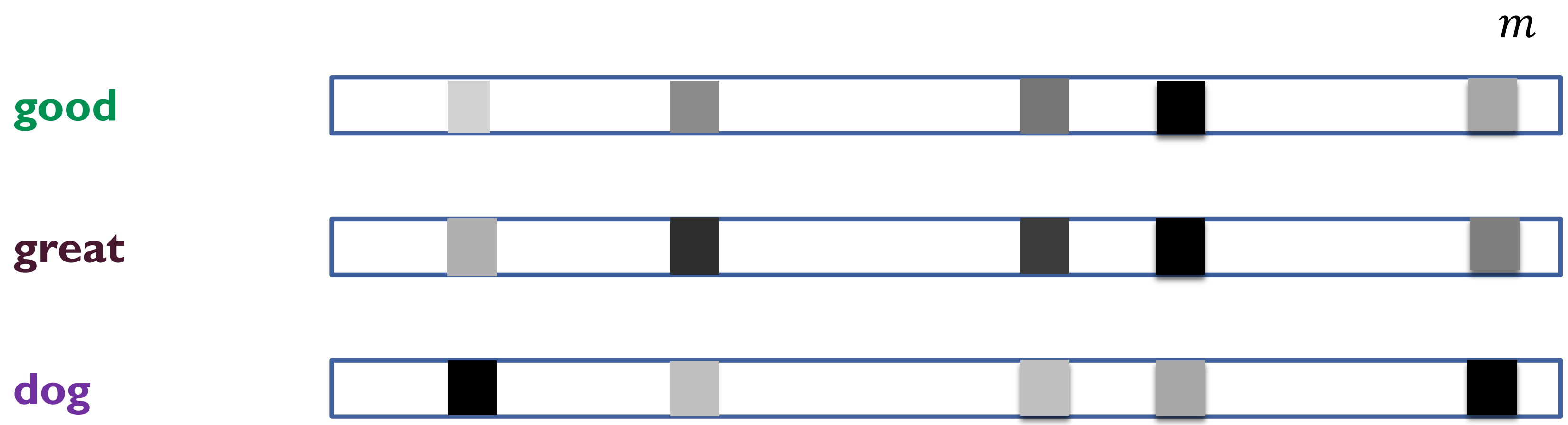
CURSE OF DIMENSIONALITY

- Imagine you have seen the following many times:
  - *Have a **good** day.*
- In contrast, you have not seen the following:
  - *Have a **great** day.*
- What happens than (even with smoothing)?



GENERALIZE BETTER

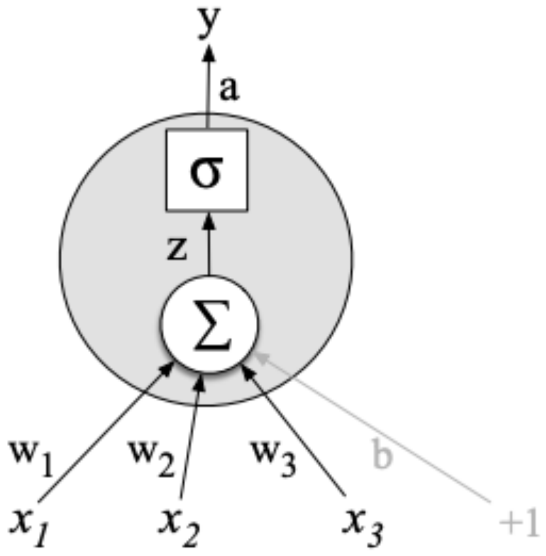
- Knowledge Transferring
- Solution?
  - Learn **Distributed Representations** for words



A Simple Introduction to Neural Networks

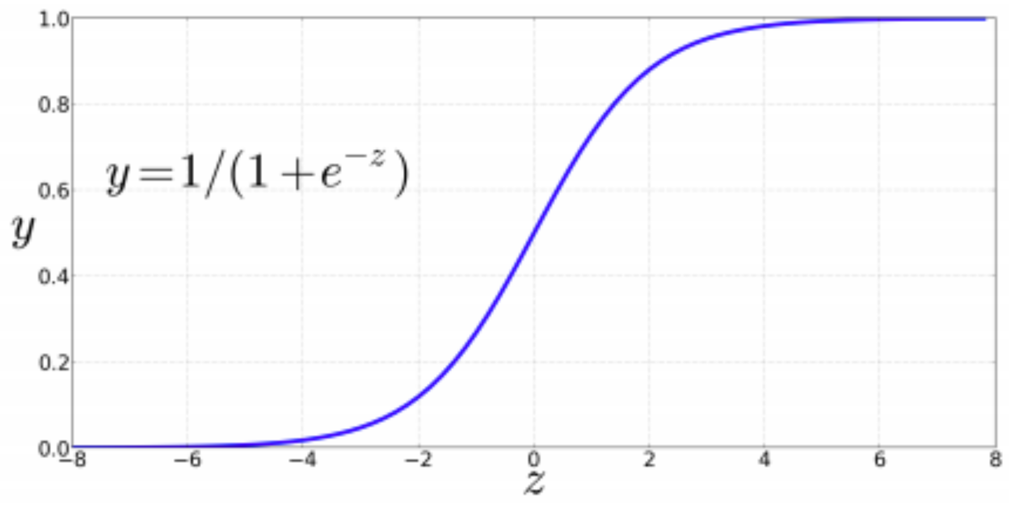
NEURAL NETWORKS

- **Neuron**
  - A single and basic computational unit of a NN
  - Takes inputs, does some math with them, and produce one output
- At its heart:
  - Taking a weighted sum of its inputs
  - With one additional term in the sum **(bias)**
  - $\zeta = b + \sum_i w_i x_i$
  - Apply a non-linear function f to Zeta. **(activation)**
  - $y = f(\zeta)$



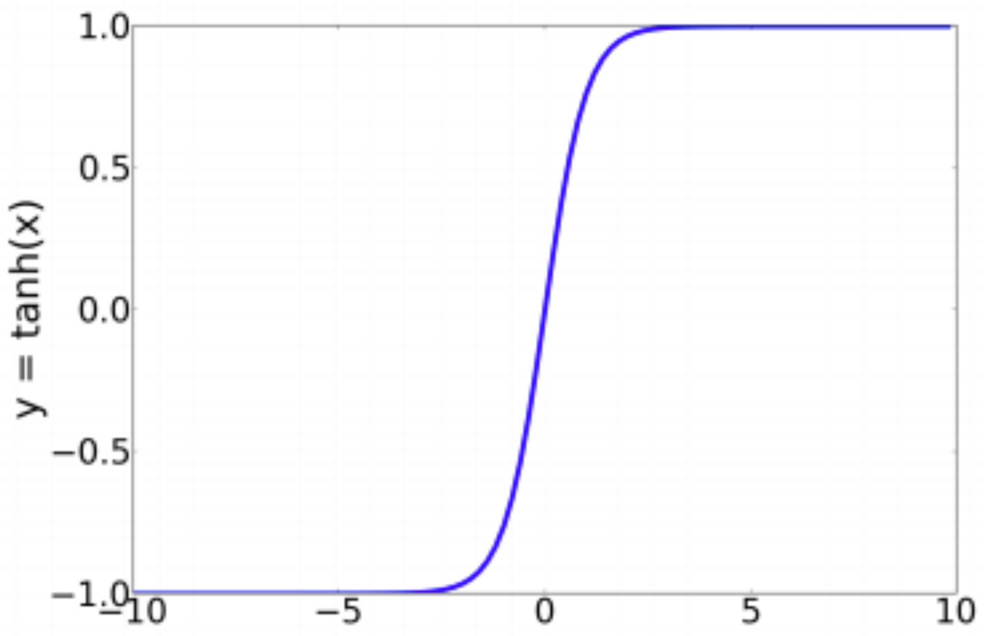
ACTIVATION FUNCTION: SIGMOID

- Maps the output into the range $[0, 1]$
- Useful in Squashing outliers toward 0,1
- $y = \sigma(\zeta) = \frac{1}{1+e^{-\zeta}}$
- $y = \sigma(w.x + b) = \frac{1}{1+e^{-(w.x+b)}}$
- Have saturated problem



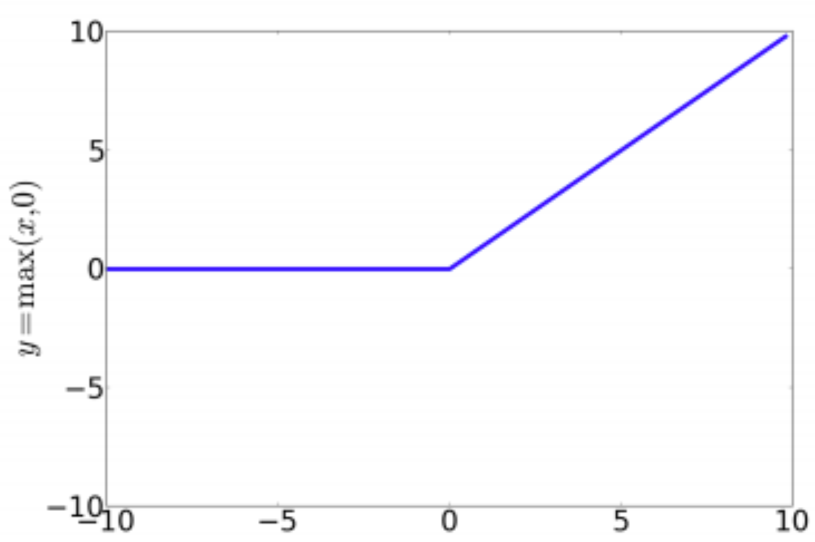
ACTIVATION FUNCTION: TANH

- Sigmoid is not commonly used as AF
- Variant of the sigmoid
- Ranges from $-1 \; to + 1$
- Have saturated problem
- $y = \dfrac{e^{\zeta} - e^{-\zeta}}{e^{\zeta} + e^{-\zeta}}$
- $y = \dfrac{e^{(w.x+b)} - e^{-(w.x+b)}}{e^{(w.x+b)} + e^{-(w.x+b)}}$



ACTIVATION FUNCTION: RELU

- ReLU = Rectifier Linear Unit
- Perhaps the most commonly used
- **Same as x when x is positive, and 0 otherwise**
- $y = \max(x, 0)$
- Rectifier do not have saturated problem!



ACTIVATION FUNCTION: EXAMPLE

- Suppose we have a unit with following weight vector and bias:
- $w = [0.2, 0.3, 0.9]$
- $b = 0.5$
- $x = [0.5, 0.6, 0.1]$
- Find the output with sigmoid activation function:

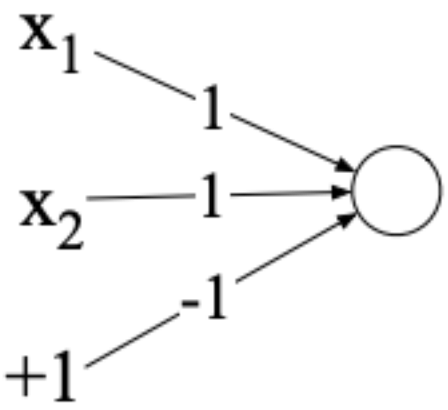
$$y = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}} = \frac{1}{1 + e^{-(.5*.2+.6*.3+.1*.9+.5)}} = e^{-0.87} = .70$$

THE PERCEPTRON

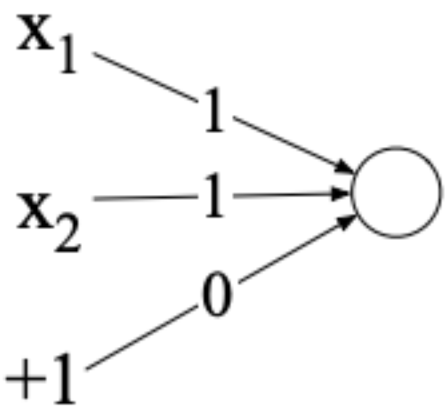
- Is a very simple neural unit
- Binary classifier
- Called threshold function
- Does not have a non-linear activation function
- $y = \begin{cases} 1, & if\ w.x + b > 0 \\ 0, & otherwise \end{cases}$

THE XOR PROBLEM

- Possible to compute the logical AND and OR functions with perceptron
- On the other hand, impossible to compute logical XOR! (Cause it's not linearly separable function)



AND



OR

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

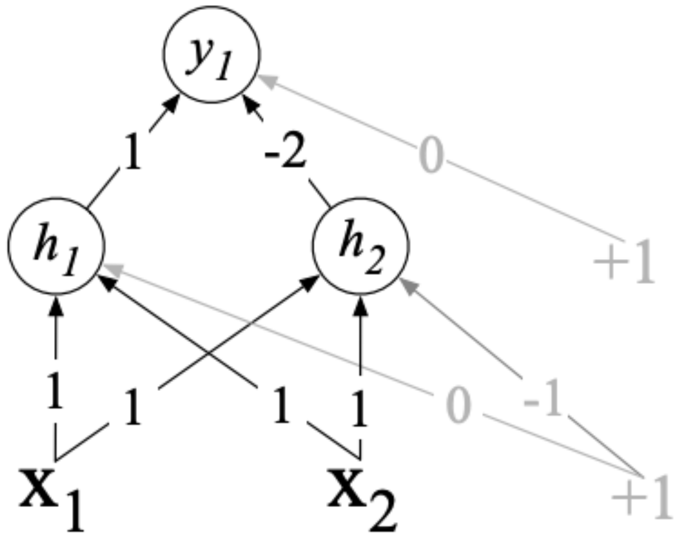
x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

NEURAL NETWORK: THE SOLUTION

- XOR can be calculated by a layered network of units.
- Goodfellow et al. computed XOR
- Two layers of ReLU-based units.
  - Three ReLU units (h_1, h_2, y_1)
- Let's walk through: $[0, 1]$
- Note: In real world, the weights for NN are learned automatically. (How?)
  - **Using the error backpropagation algorithm**

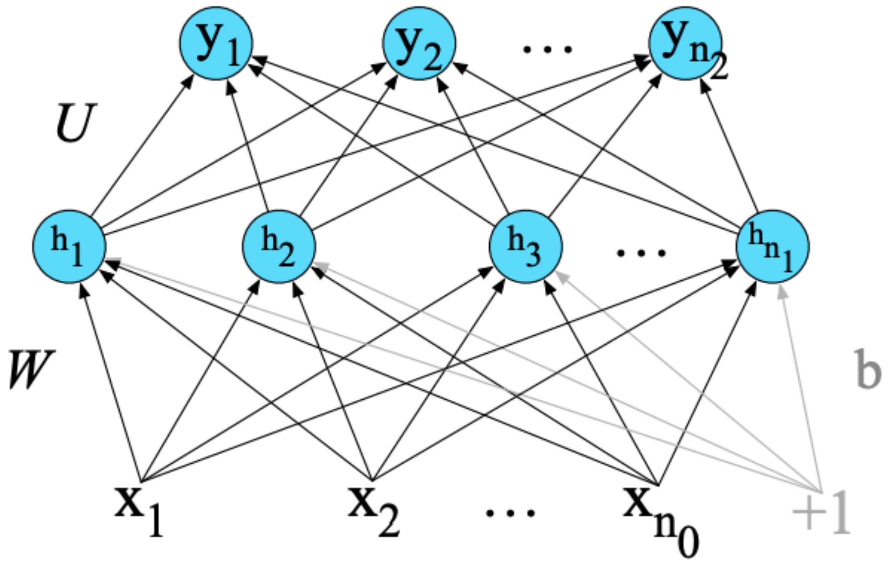


FEED-FORWARD NEURAL NETWORKS

- A multilayer neural network
- Units are connected with no cycles
- The outputs from units in each layer are passed to units in the next higher layer
  - No outputs are passed back to lower layer!
- Are there any NNs which have cycles? Yeah!
- For historical reason feedforward networks, sometimes called
  - Multi-layer Perceptrons (MLPs)
  - But it is a misnomer
  - Perceptrons are linear but feedforward NNs made up of units with not-linearities.

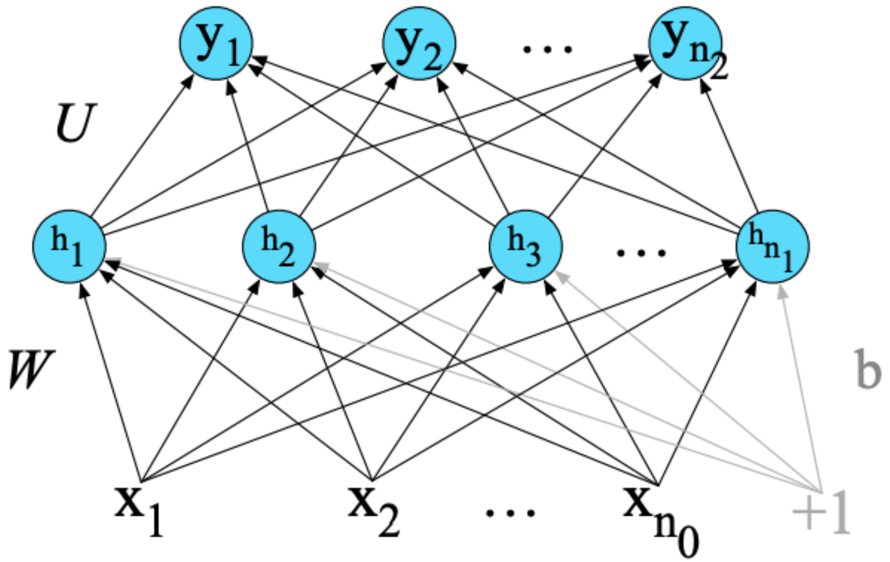
FEED-FORWARD NEURAL NETWORKS (CONT'D)

- Simple feedforward NNs:
  - Input units, Hidden units, Output units
- The **core** of neural networks: **hidden layer**
- In Standard architecture**, each layer is fully-connected.**
- The resulting value **h** forms a representation of **input**.
- It is better to save weights into a single matrix **W** !
- $h = \sigma(Wx + b)$



FEED-FORWARD NEURAL NETWORKS (CONT'D)

- The output units depend on problems, i.e.
  - Sentiment classification
  - Part-of-Speech
- Some models don't include a bias vector b in output
- $\zeta = Uh$
- For example given a vector $\zeta = [0.6 \quad 1.1 \quad -1.5 \quad 1.2 \quad 3.2 \quad -1.1]$
- We need an activation function for normalizing a vector of real values. (softmax)
- Thus, $\text{softmax}(\zeta)$ *is* $[0.055 \;\; 0.090 \;\; 0.0067 \;\; 0.10 \;\; 0.47 \;\; 0.010]$



FEED-FORWARD NEURAL NETWORKS (CONCLUSION)

- Final quotations for a feedforward network with a single hidden layer
  - $h = \sigma(Wx + b)$
  - $\zeta = Uh$
  - $\mathrm{y} = \mathrm{softmax}(\zeta)$

- Let's set up some notation to talk about deep networks (depth>2)
  - $W^{[1]} = \text{ } weight\ matrix\ for\ the\ first\ hidden\ layer$
  - $b^{[1]} = \text{ } bias\ vector\ for\ the\ first\ hidden\ layer$
  - $n_j = \text{ } number\ of\ units\ at\ layer\ j$
  - $g(.) = activation\ function\ (ReLU\ or\ \mathrm{Tanh}\ for\ hidden\ layer\ and\ Softmax\ for\ output)$

TRAINING NEURAL NETWORKS

- **Feedforward** neural net is an **instance** of **supervised machine learning.**
- What does it mean?
  - We know the correct output y for each observation x
- Goal:
  - To learn parameters $W^{[i]}$ and $b^{[i]}$ for each layer i
  - And make **predicted y** for each training observation **as close as** possible to the **true y**.
- Somehow the procedure is like **logistic regression**.

TRAINING NEURAL NETWORKS (CONT'D)

a) A **loss function** is needed.

- which models distance between the system output and gold output
- A common to use is **cross-entropy loss**.

b) To minimize this loss function, we'll use **gradient descent optimization algorithm.**

c) **Knowing Gradient of loss function** (How?)

- Using error **backpropagation** or **reverse differentiation**.
- More complex part of learning about NN

MORE DETAILS ABOUT NNS

- Weights need to be initialized with small random numbers.
- Normalizing input is recommended
- Using regularization techniques
  - Dropout
- Tuning hyperparameters (chosen by algorithm designer)
  - Learning rate
  - Mini-batch size
  - epoch
  - Model architecture

- Using GPU to paralyze computation

Feedforward Neural Language Models

A NEURAL PROBABILISTIC LANGUAGE MODEL (NPLM)

- Bengio's NPLM in 2003 (feedforward NN)

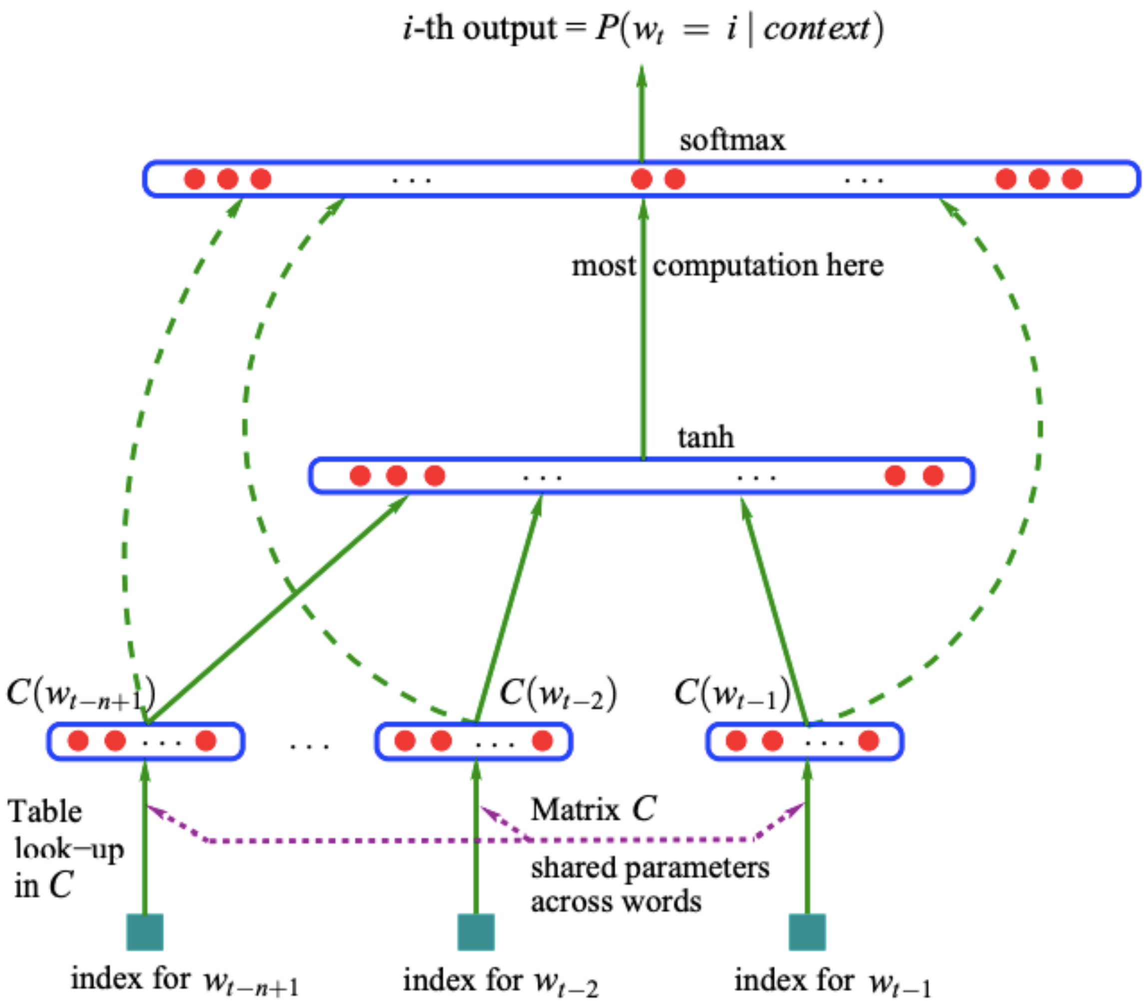
$$x = (C(w_{t-1}), C(w_{t-2}), \cdots, C(w_{t-n+1}))$$

- X is a word feature

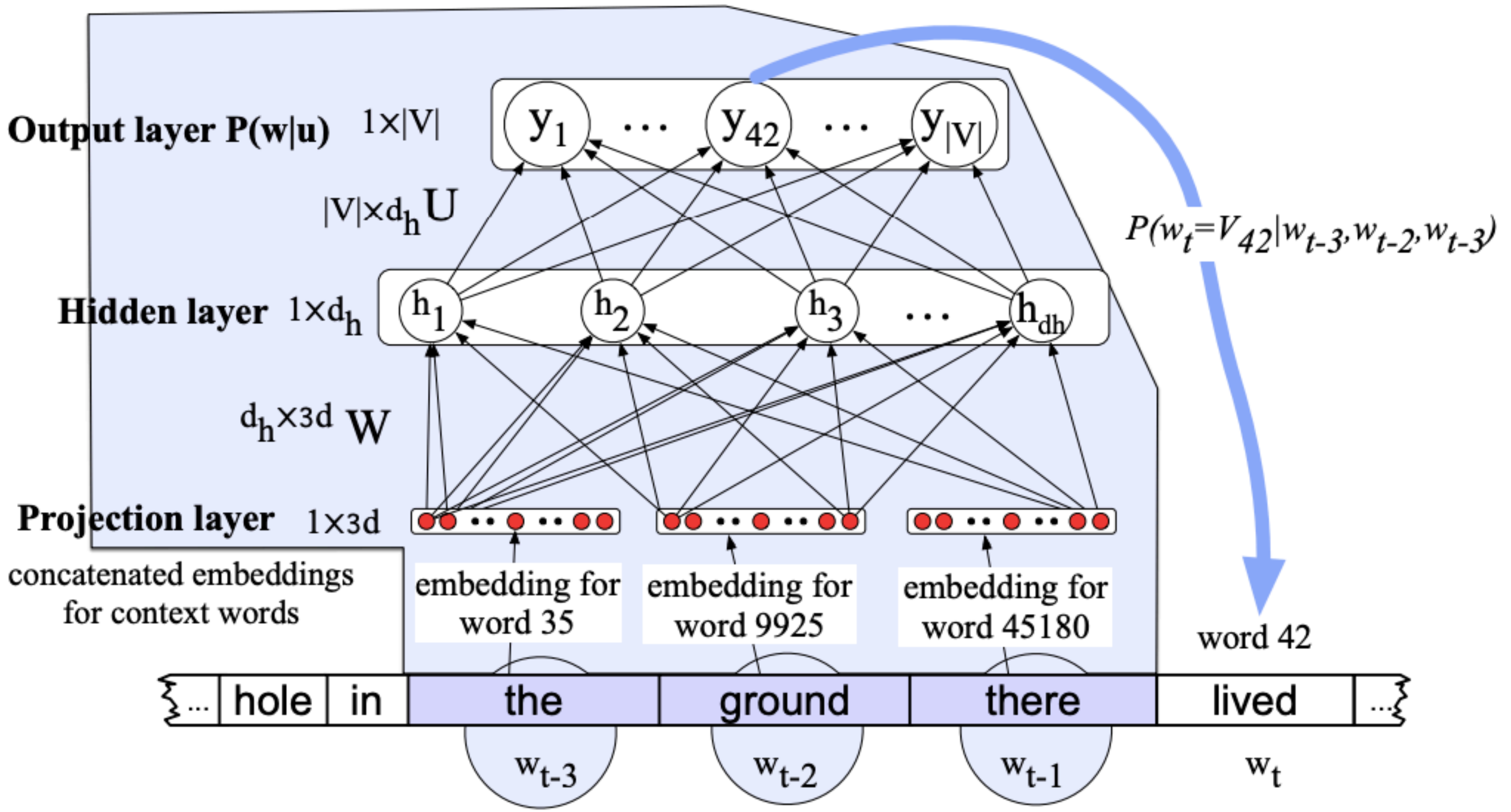
$$y = b + Wx + U\tanh(d + Hx)$$

- y_i are unnormalized probabilities for word i

$$\hat{P}(w_t | w_{t-1}, \cdots w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$



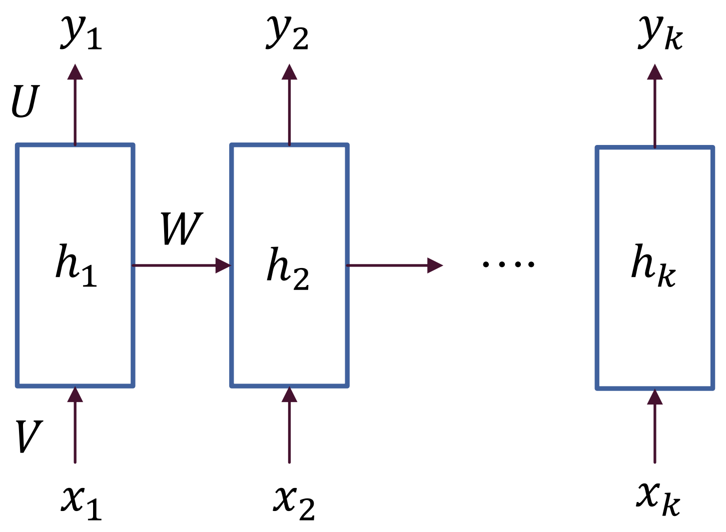
A NEURAL PROBABILISTIC LANGUAGE MODEL (NPLM)



Recurrent Neural Language Models

RECURRENT NEURAL NETWORKS

- **Humans don't start their thinking from scratch every second.**
- Traditional NNs can't do this! (Major shortcoming).
- RNNs address this issue.
- RNN contains a **cycle/loop**
- A RNN is a multiple copies of same network
- Powerful in spoken and written language
- Model the sequences

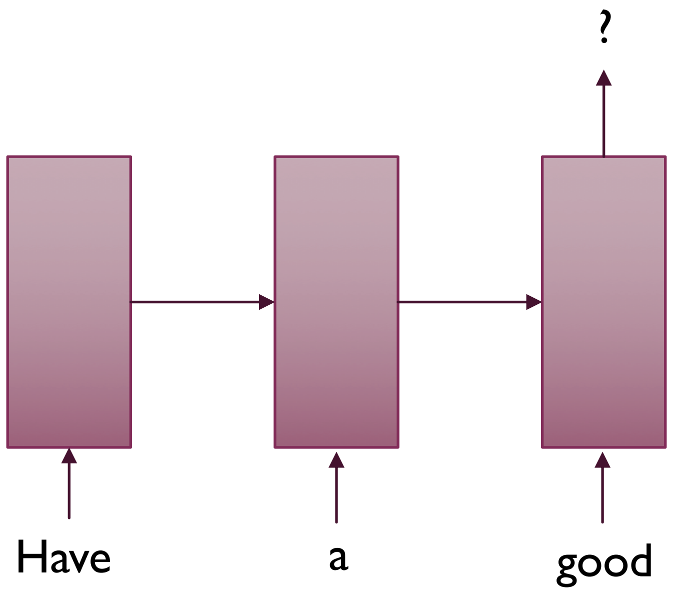


$$h_i = f(wh_{i-1} + Vx_i + b)$$

$$y_i = Uh_i + \tilde{b}$$

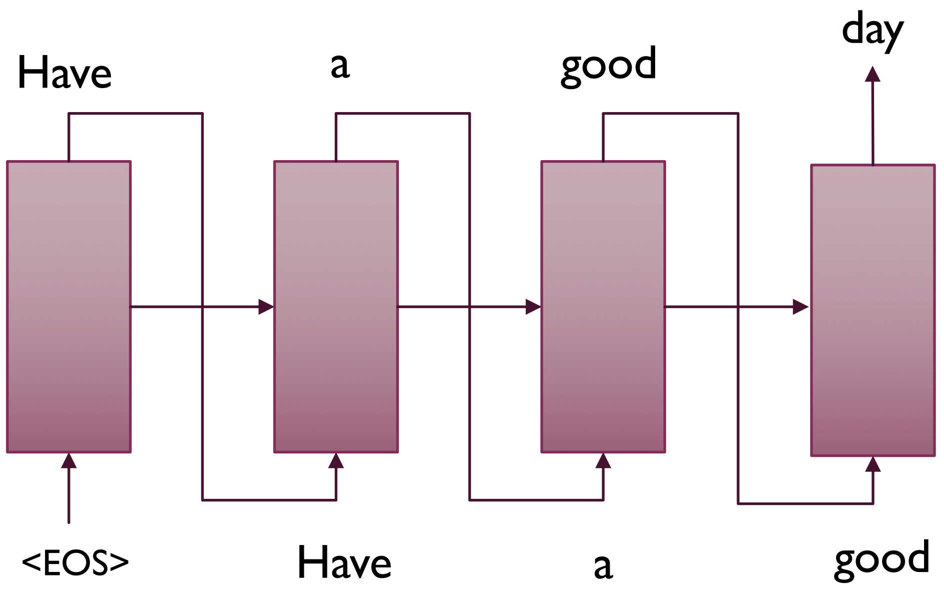
RNN LANGUAGE MODEL

- Predict a next word based on a previous context
- Architecture:
    a. Use the current state output
    b. Apply a linear layer on top
    c. Do **softmax** to get probabilities



USE RNN TO GENERATE LANGUAGE

- Idea:
  - Feed previous output as the next input
  - Take **argmax** at each step (**greedily**) or use **beam search**



COMPARISON RNN LM WITH N-GRAM MODEL

- **RNN-LM** has lower perplexity
- In comparison with **5-gram model with Knesser-Ney smoothing**

Model	# words	PPL
KN5 LM	200K	336
KN5 LM + RNN 90/2	200K	271
KN5 LM	1M	287
KN5 LM + RNN 90/2	1M	225
KN5 LM	6.4M	221
KN5 LM + RNN 250/5	6.4M	156

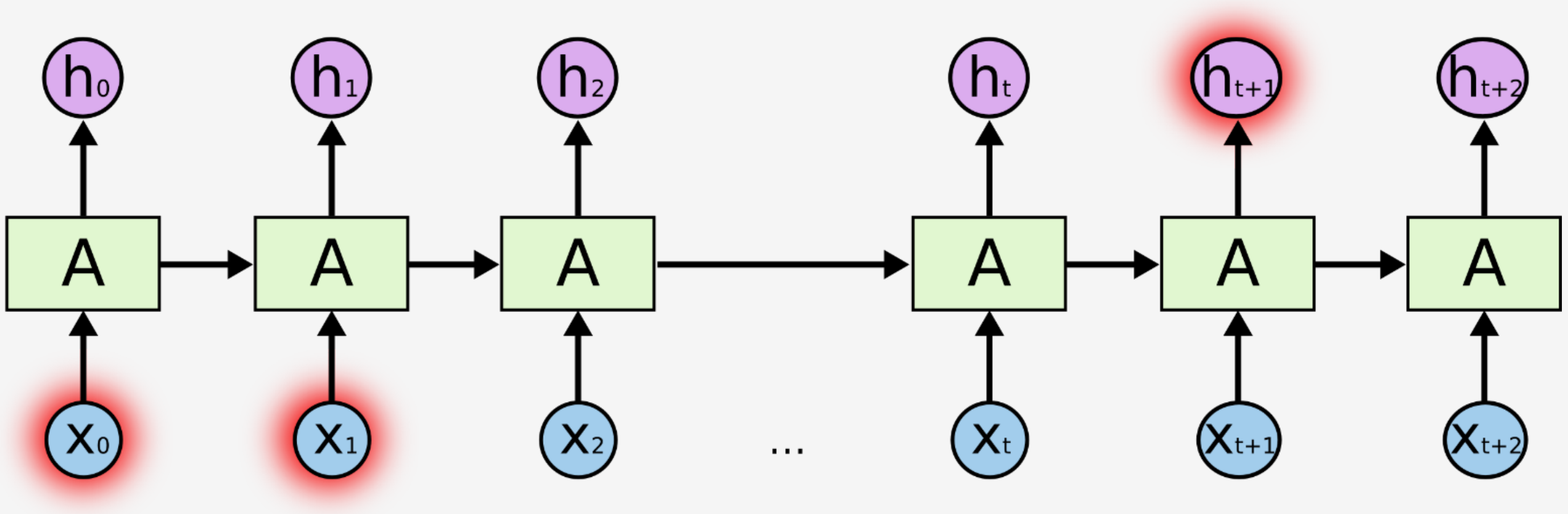
- Afterward, later experiments have shown **char-level RNNs** can be more effective!

SOME NOTES FOR MAKING YOUR OWN LM

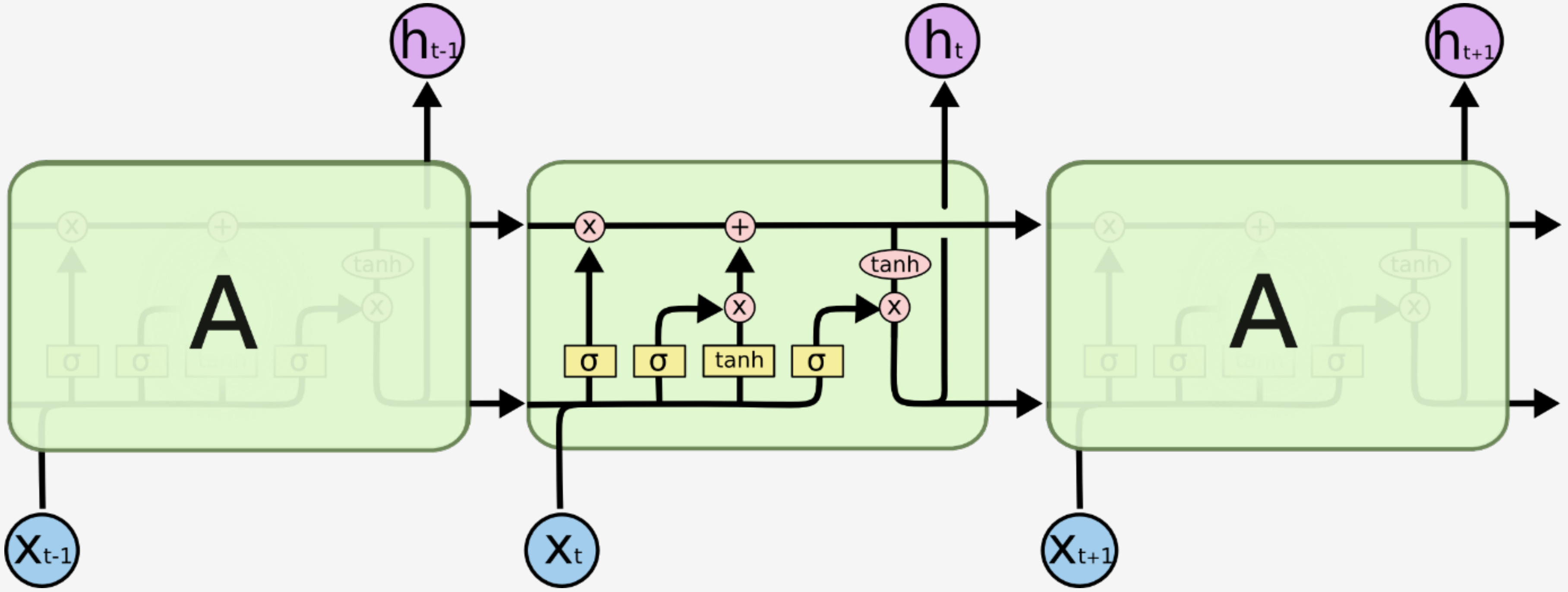
- LSTM or GRU are also used to handle long sequences
- Start with one layer, then go for 3, 4 and so forth
- Use dropout for regularization (https://arxiv.org/pdf/1409.2329.pdf)
- Using TensorFlow to get the advantages of your GPU
- Tune learning rate (like Adam)

LONG SHORT-TERM MEMORY NETWORKS (LSTMS)

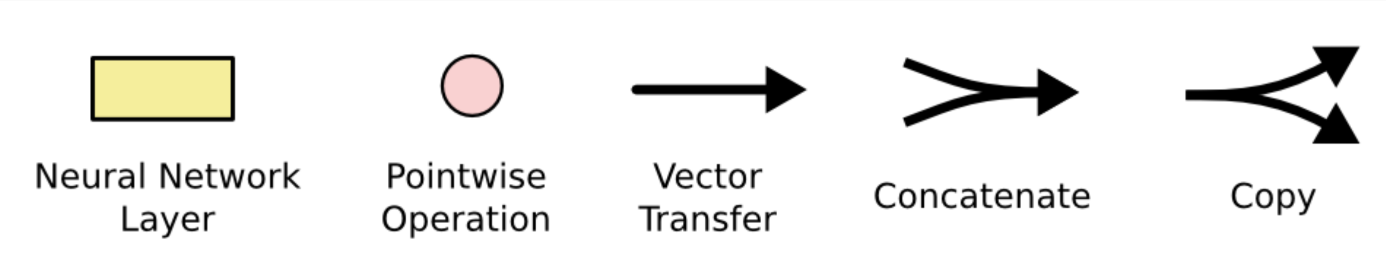
- Solve the problem of **Long-Term Dependencies**
- Prediction Point 🡐 Very Large Gap 🡒 Relevant Information



LSTM (CONT'D)

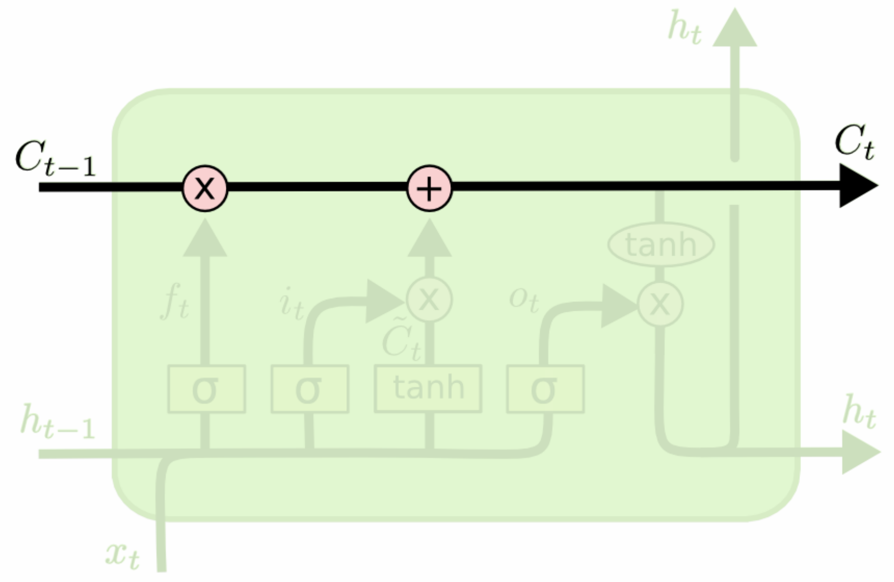


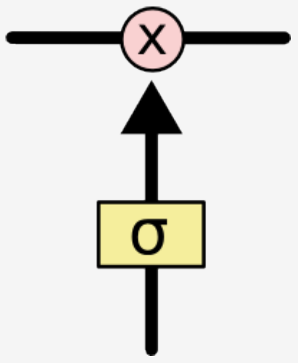
The LSTM contains four interacting layers.



LSTM (CONT'D)

- Idea: The horizontal line through the top of the diagram
- Called cell state / memory
- Remove and add info to the cell state (0/1)
- Regulated by gates
- A gate consists of a sigmoid NN layer and a pointwise multiplication operation.





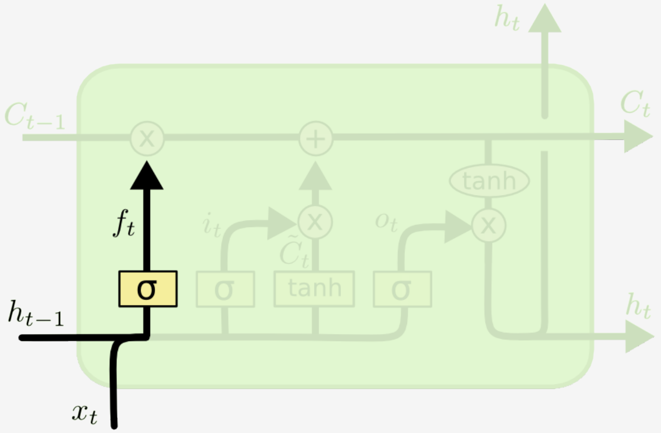
Gate

LSTM (CONT'D)

- Has 3 Gates to protect and control the cell state

1. Forget Gate Layer / Sigmoid Layer:

   Decide to throw away the information



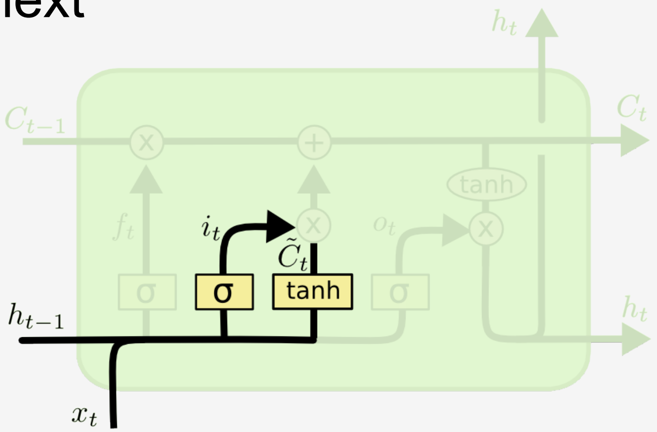
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

2. Input Gate Layer

   Decide which value will be updated next

   1. Input Gate Layer
   2. A tanh layer (squash)

Finally: $i_t * \tilde{C}$

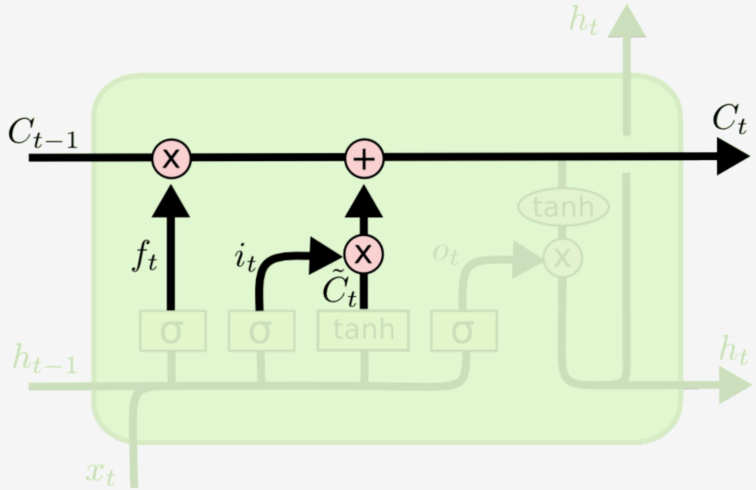


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

LSTM (CONT'D)

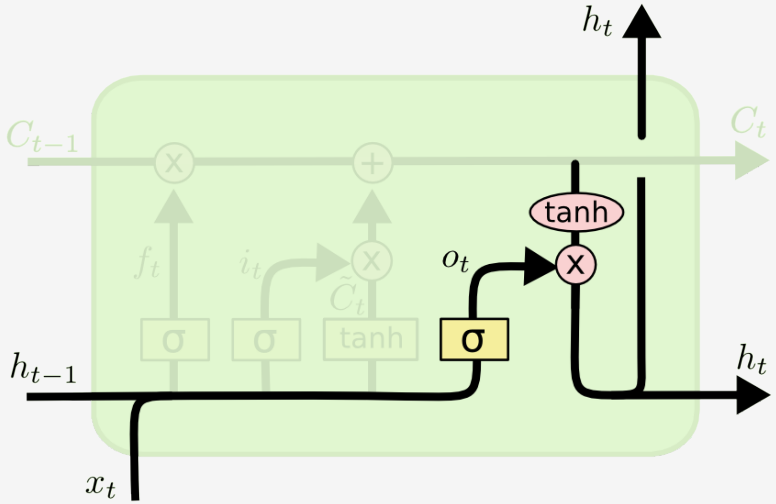
- Cell state up to now



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

3. Output Gate Layer

   A filtered version of output



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

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