

# RNN's and LSTM's

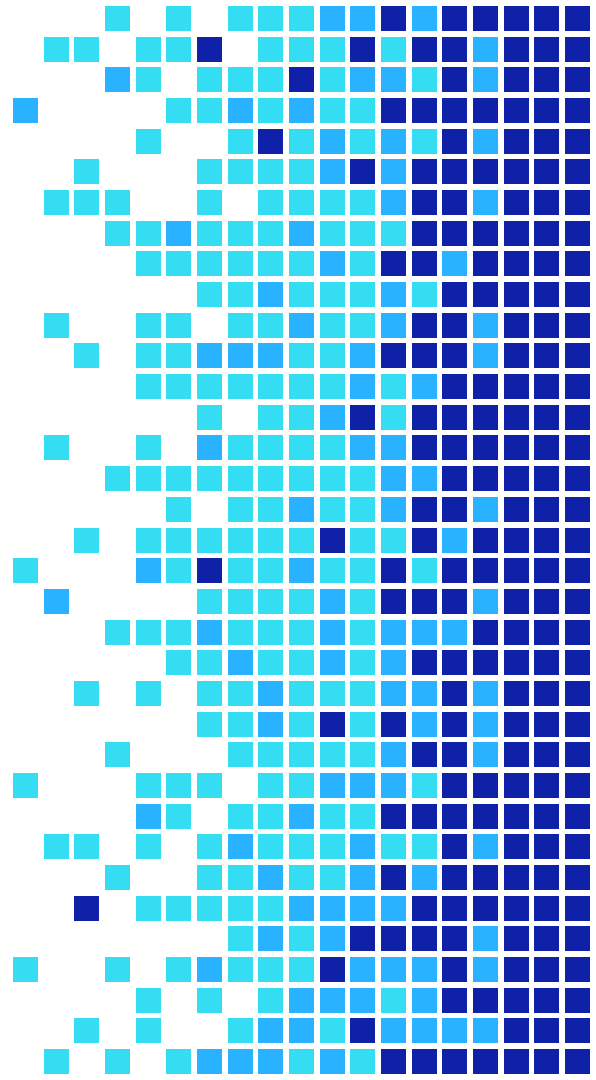
Advanced Track Workshop #5

Anonymous Feedback: [tinyurl.com/w21advtrackfb5](https://tinyurl.com/w21advtrackfb5)

GitHub:

[github.com/uclaacmai/advanced-track-winter21](https://github.com/uclaacmai/advanced-track-winter21)

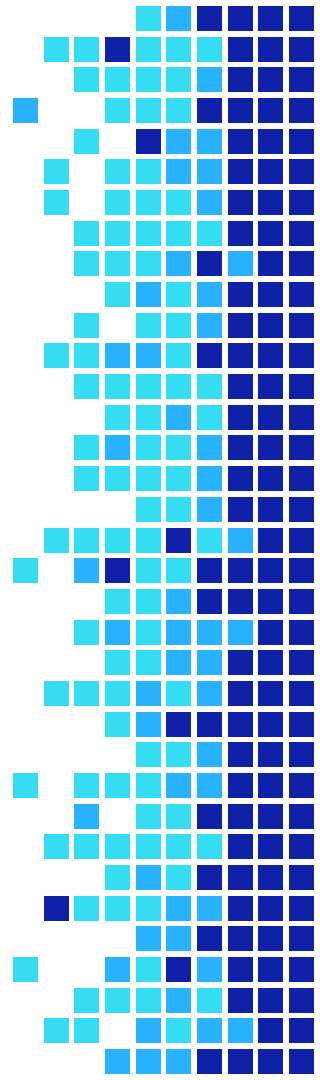
Attendance Code: [skijumping](#)



# Today's Content

- Word Embeddings
- Intro to Recurrent Neural Networks
  - Motivation
  - Concepts
  - Applications

# 1. Word Embeddings



## Example Problem

The baseball player ran over to catch \_\_\_\_.

The grizzly bear goes to the river to catch \_\_\_\_\_.

What are some logical answers?

What clues did you use?

How much context do we need?

# Implications of the Example: Word Embeddings

The grizzly bear goes to the river to catch \_\_\_\_\_.

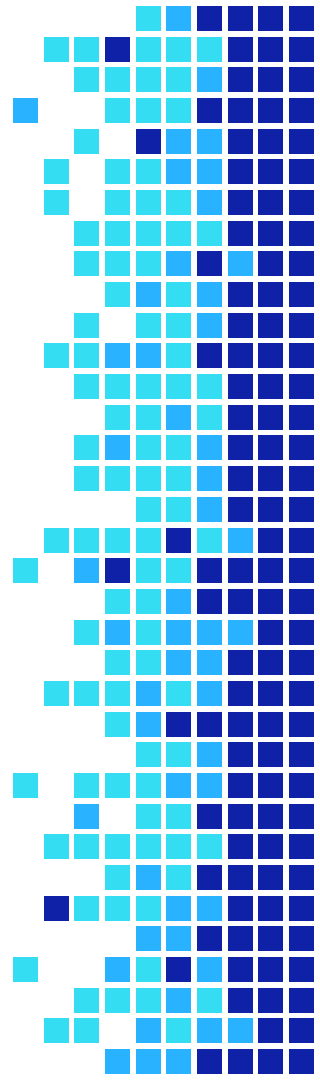
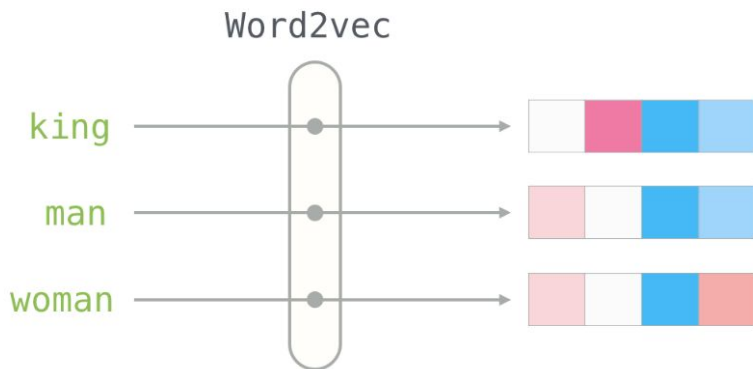
- If we want our network to predict what word should fill in the blank, we need to feed it the every other word in the sentence somehow
- Computers don't read natural language, so how are we going to feed our network the input words?
  - Discuss amongst each other how we might feed the network the input



acm.ai

# Word Embeddings

- We need to provide a numerical encoding for each word in our vocabulary
  - Vocabulary refers to the set of all possible input words to our model (sometimes uncommon words in a corpus are omitted from the vocabulary for simplicity)
- Introducing Word2Vec: the standard for building pre-trained word embeddings with supervised learning!



# Word Embeddings

- But it still looks like we're passing natural language to the computer, how does this solve anything?
- The inputs to word2vec are "one-hot encoded" vectors
  - If your vocabulary has 100 words, each input to word2vec would be a 100 x 1 vector, each of which has 99 entries of "0" and a single entry of "1"

Rome Paris word V

Rome = [1, 0, 0, 0, 0, 0, ..., 0]

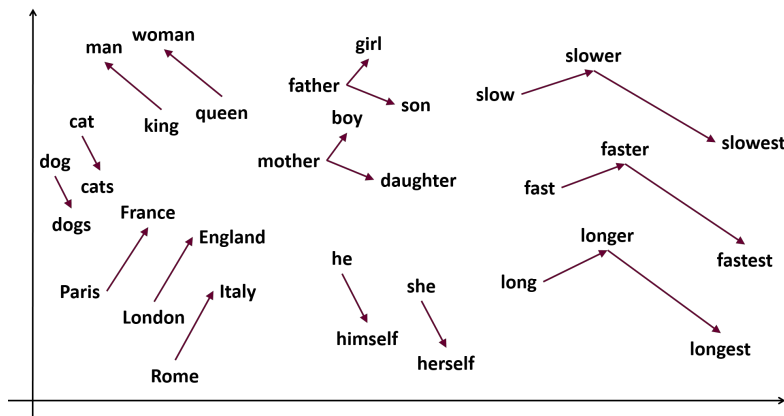
Paris = [0, 1, 0, 0, 0, 0, ..., 0]

Italy = [0, 0, 1, 0, 0, 0, ..., 0]

France = [0, 0, 0, 1, 0, 0, ..., 0]

# Word Embeddings

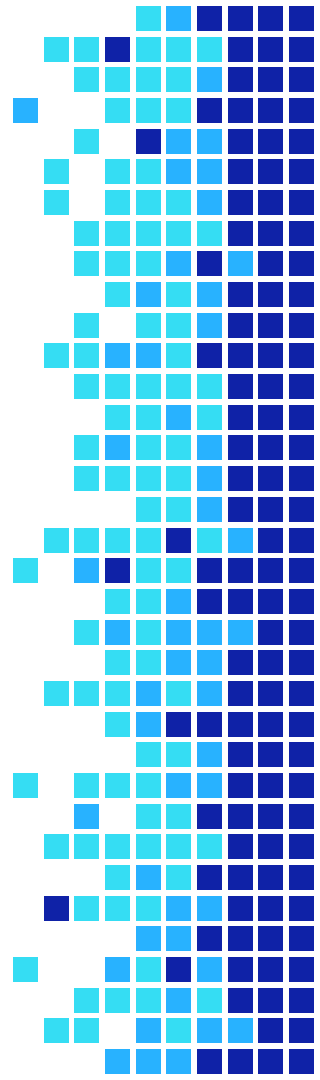
- Because the input words are all one-hot encoded, they are all orthogonal to each other
  - This basically means that the numerical representations of the words have no relationship with each other, which is a problem
- The idea of word2vec is to generate encodings for these words that capture the relationships between similar words





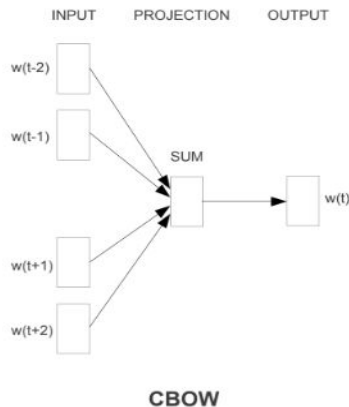
# Word Embeddings

- Thankfully, the word2vec architecture follows a simple fully-connected neural network architecture that we're already familiar with, but what does the output space look like?
  - Word2vec has 2 flavors: Continuous Bag-Of-Words (CBOW) and Skip-Gram
  - The input and output spaces depend on which flavor we adopt



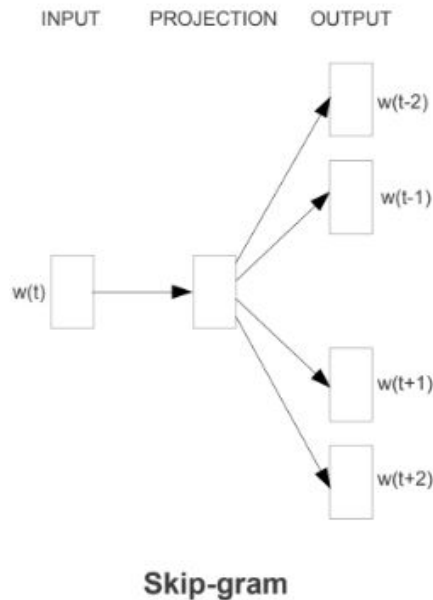
# Word Embeddings (CBOW)

- The grizzly bear goes to the river to catch **fish**.
  - In our original example, we wanted the network to predict that the word "fish" belonged at the end of the sentence.
- In CBOW, we do exactly this.
  - Our input to word2vec would be each of the words "The", "grizzly", "bear", "goes", "to", "the", "river", "to", "catch", all presented as one-hot encoded vectors. CBOW will then output a single word that it think belongs in the blank



# Word Embeddings (Skip-Gram)

- The grizzly bear goes to the river to catch **fish**.
  - In our original example, we wanted the network to predict that the word “fish” belonged at the end of the sentence.
- In Skip-Gram, we do the OPPOSITE
  - Our input to word2vec would be each of the word “fish” presented as one-hot encoded vectors. Skip-gram will then try to predict the surrounding context words





# Word Embeddings: Results

- The idea was to generate vector encodings for each of the words in our vocabulary that would capture the relationship between similar words. This can have some interesting results.
- The common example is to look at the embeddings for the words "king", "man", and "woman". What word do you think the vector "king - man + woman" would most closely represent?
  - Discuss this amongst each other

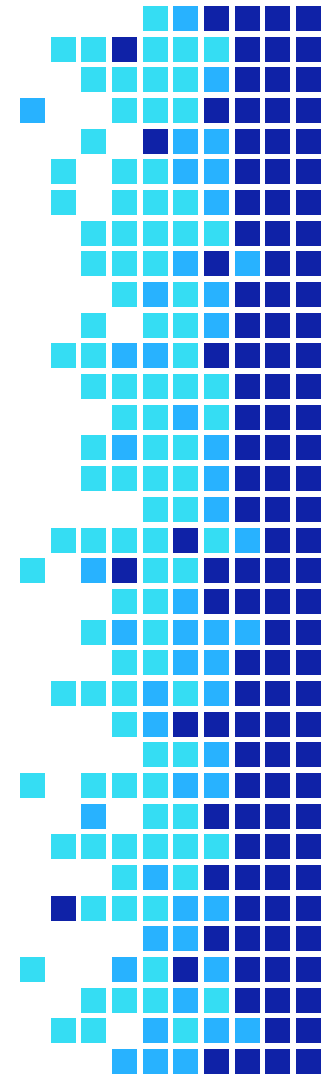
"king"



"Man"



"Woman"

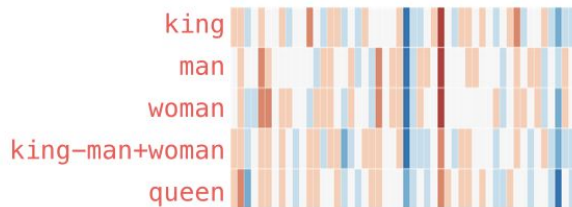




# Word Embeddings: Results

- Your intuition was likely correct: it would most closely represent the word “queen” in natural language
- This example illustrates how word embeddings are essential to generating information-rich encodings for words in natural language processing tasks
  - We can’t just feed our models one-hot encoded vectors. That would be extremely inefficient and the one-hot encoded vectors don’t capture the underlying relationships present in natural language

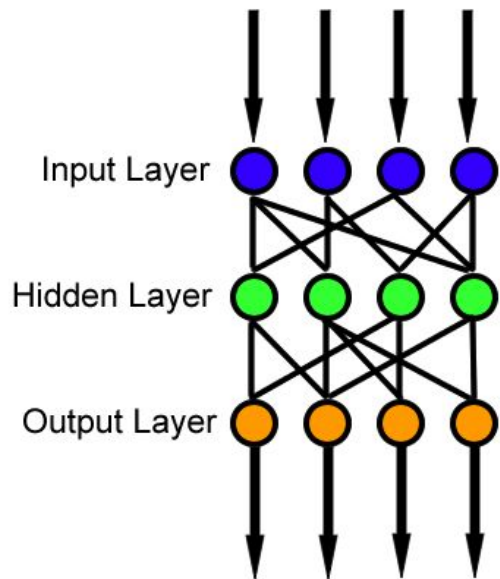
king - man + woman  $\approx$  queen



## 2. Motivation for RNNs

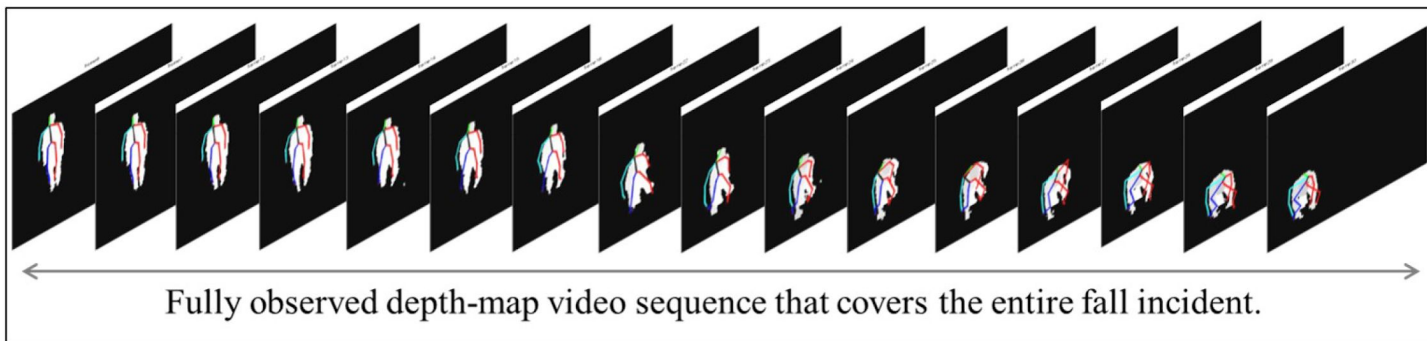
# What we've done so far

- What do all of the models we've covered so far have in common?
  - They are all **feedforward** networks.
  - The input is **non-sequential**.
  - The data is **static** and the input space is **independent** of other elements
- What type of data would result in the data not being independent?



# What can we change?

- Input **Sequences**





# Why are sequences useful?

- Information about past input can inform current predictions (Recurrence Relation)
  - e.g. ball bouncing: where will ball bounce next?
- Forms of Sequence Data
  - Audio
  - Text
- Think of some examples of sequential data

# 3. Concepts

## Structure

- Before:

**input  $\rightarrow$  hidden  $\rightarrow$  output**

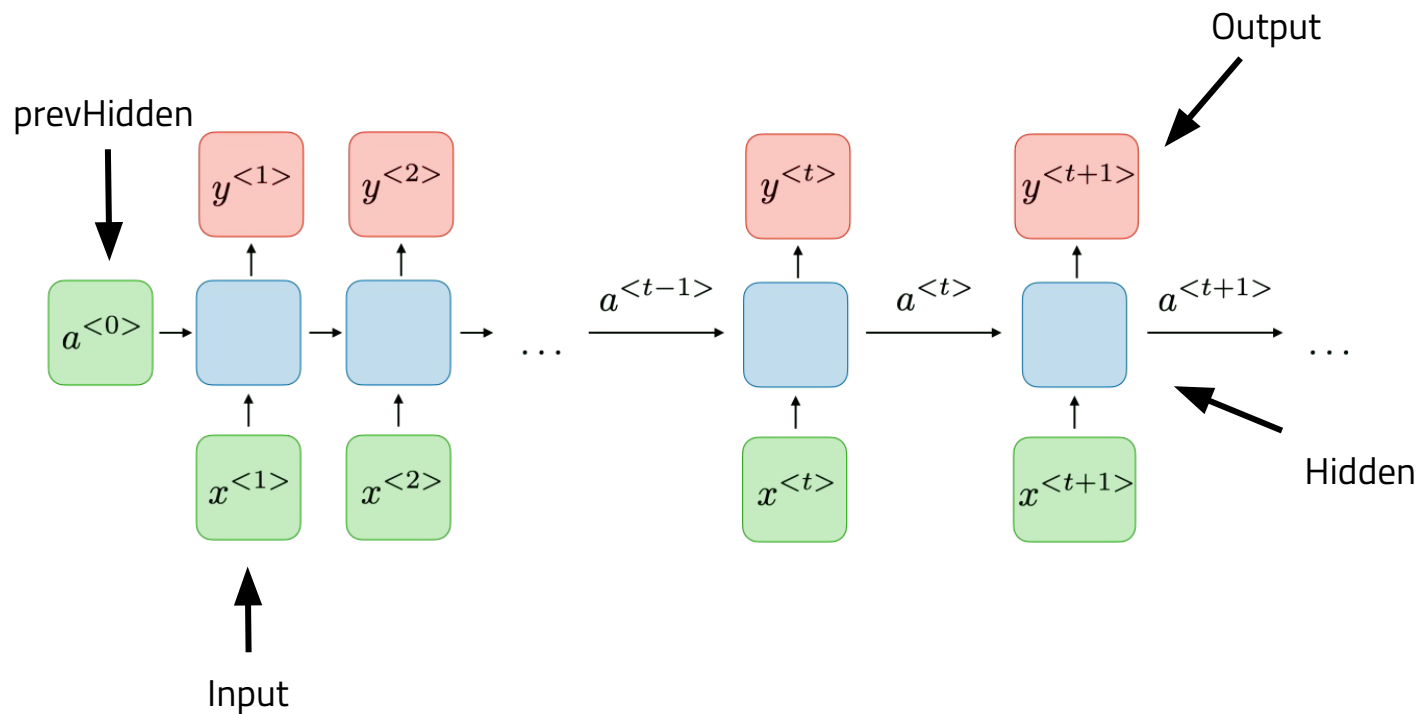
- RNN:

**(input+prevHidden)  $\rightarrow$  hidden  $\rightarrow$  output**



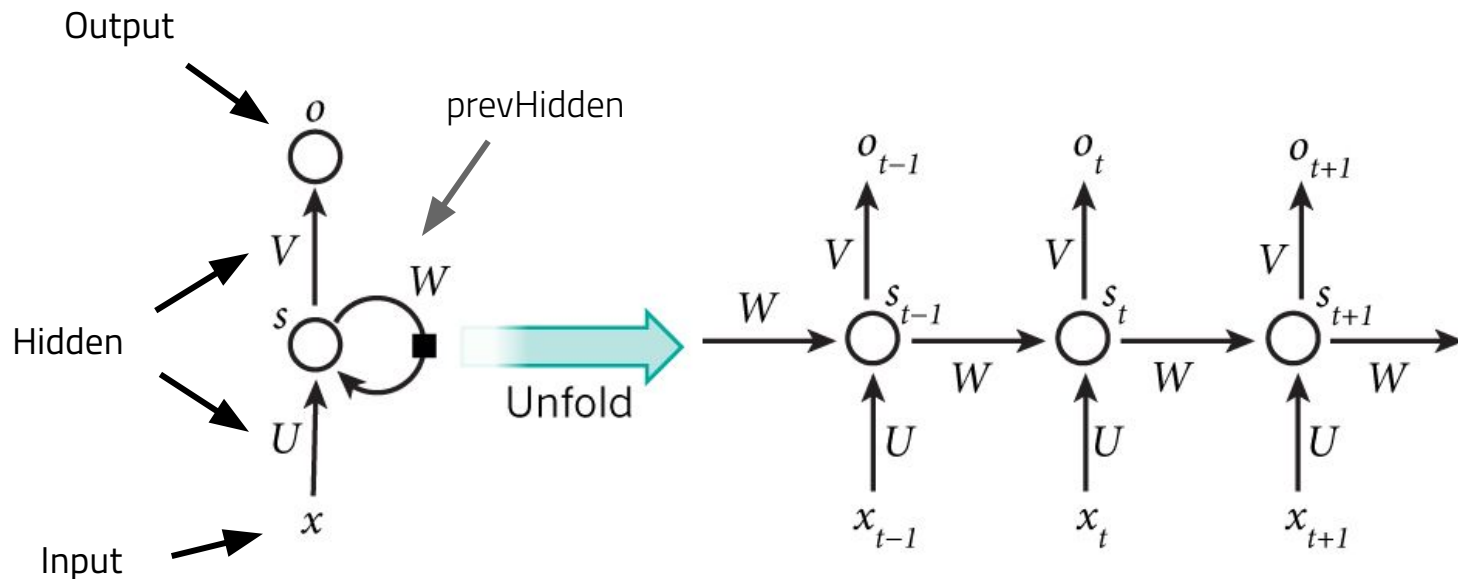
Concatenate, not add

# A Recurrent Neural Network





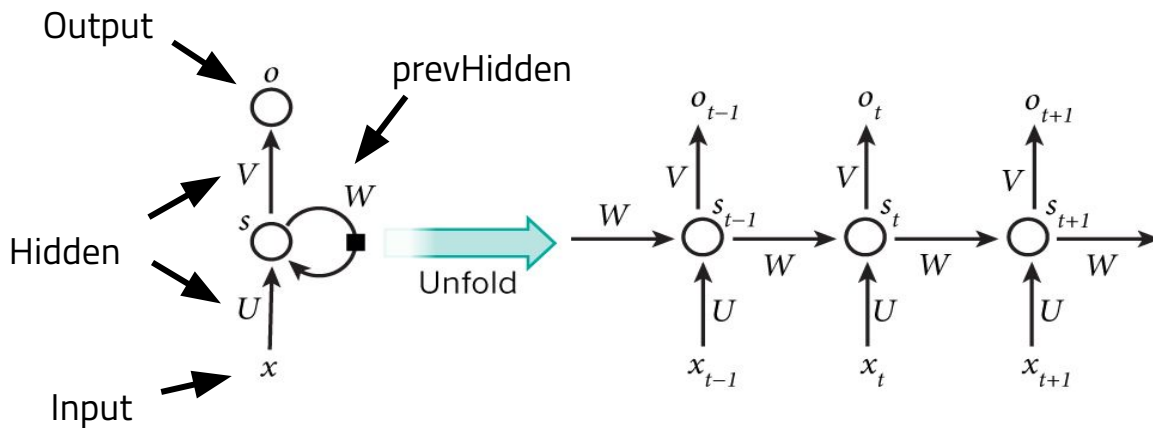
# A Recurrent Neural Network



RNN shares the same parameters ( $U$ ,  $V$ ,  $W$ ) across all steps. This reflects the fact that we are performing the same task at each step, just with different inputs. This greatly reduces the total number of parameters we need to learn.



# A Recurrent Neural Network



- The parameters ( $U$ ,  $V$ ,  $W$ ) are shared across EVERY time step of the RNN
- The “unwrapped” visual of the RNN is very helpful for first understanding the architecture, but it’s important to understand that the visual on the left is a much more proper and accurate depiction
  - Understanding the visual on the left makes it a lot easier to see why these ( $U$ ,  $V$ ,  $W$ ) parameters are the same across all time steps
  - Shared parameters are utilized in nearly all proper neural network architectures, as they save a lot on space and time complexity

# Properties of RNN

## Advantages

- Possibility of processing input of any length
- Model size not increasing with size of input
- Computation takes into account historical information
- Weights are shared across time

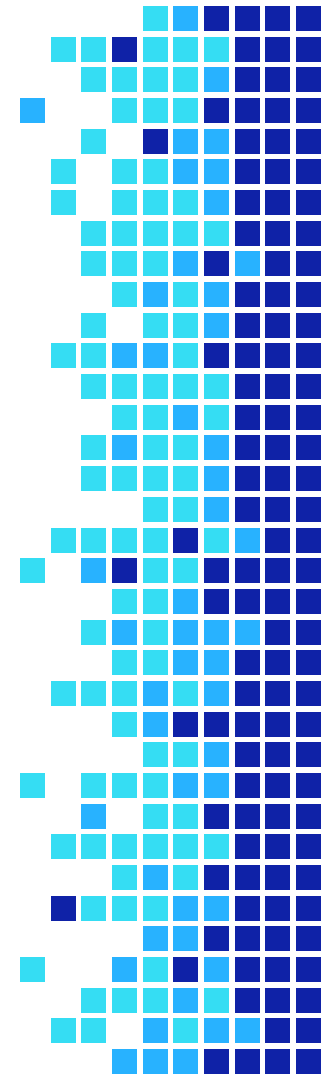
## Drawbacks

- Computation being slow
- Difficulty of accessing information from a long time ago
- Cannot consider any future input for the current state
- Vanishing gradient problem becomes very apparent over large time deltas

# Poll

Which task is RNN best suited for?

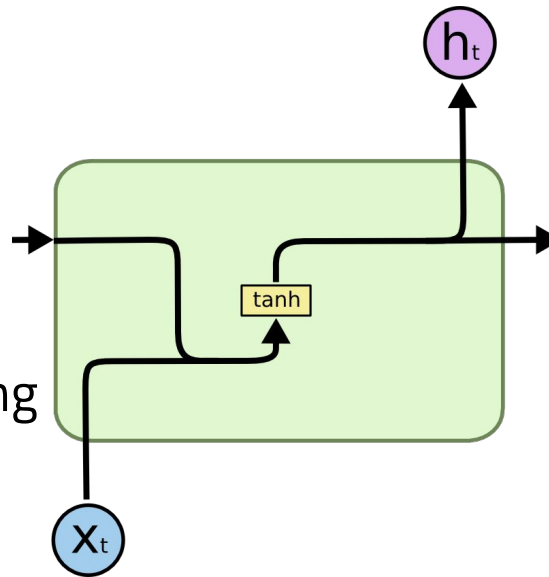
- A. Predicting whether a tumor is benign or malignant
- B. Predicting the housing prices in Boston
- C. Predicting the next word in a sentence
- D. Image classification





# Short Term Memory

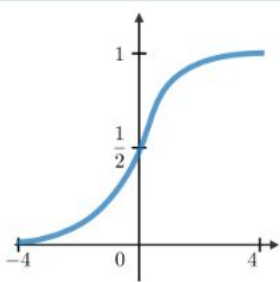
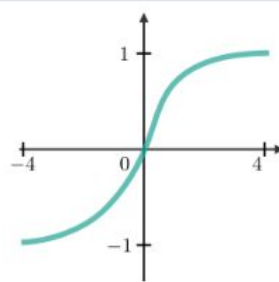
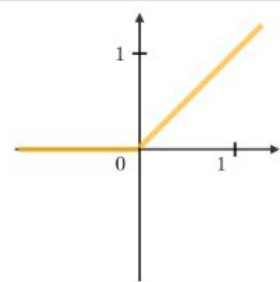
- As RNN processes more steps, it has trouble retaining information from previous steps
- Vanishing Gradient Problem: gradient becomes smaller when backpropagating through time
- RNN doesn't learn long-range dependencies across time steps

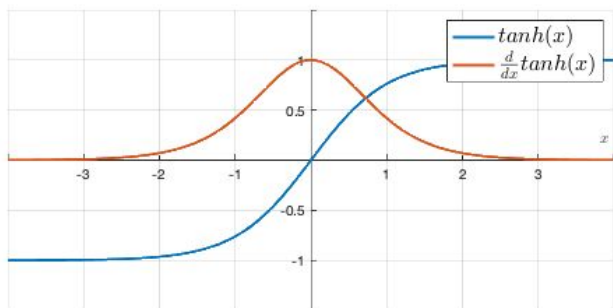


# Recall: Vanishing Gradient

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		



# 4. LSTM

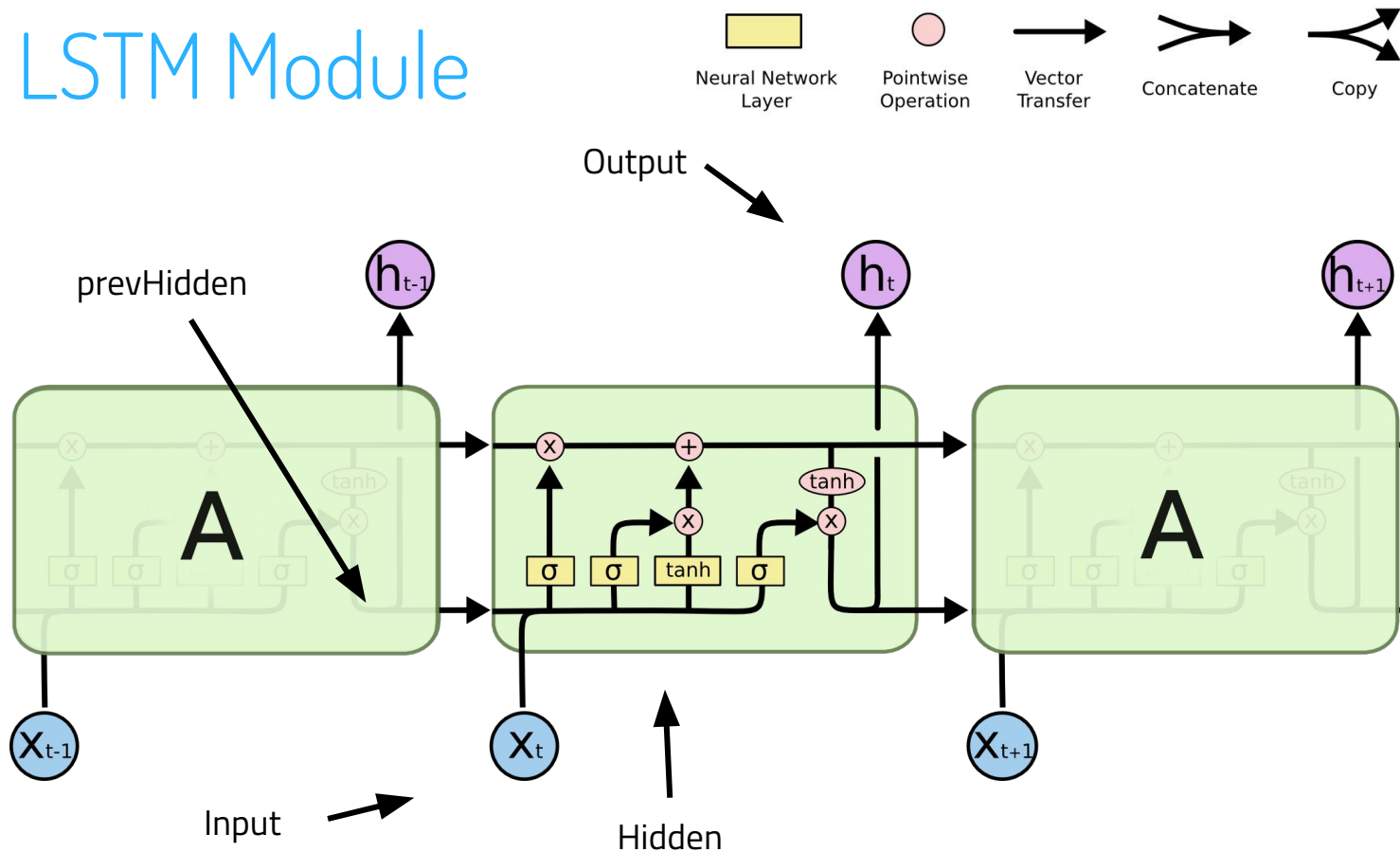
# LSTM

- LSTM (Long Short Term Memory)
- Solves the problem of long term memory
- Most commonly used type of RNN architecture

The following explanation is adapted from:

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

# LSTM Module



# LSTM Processes

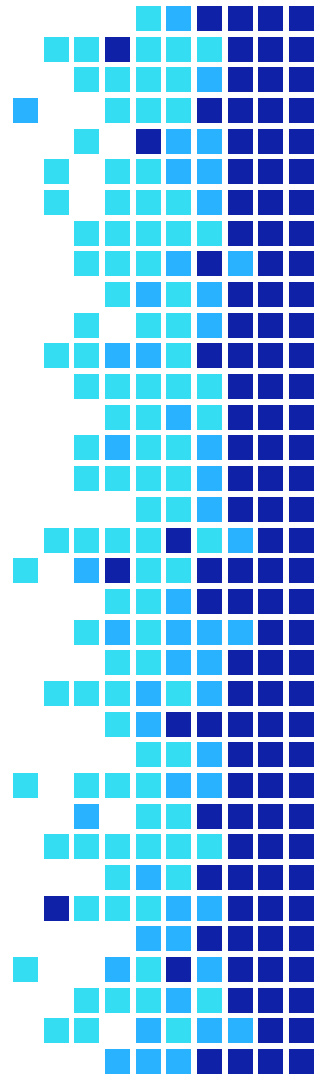
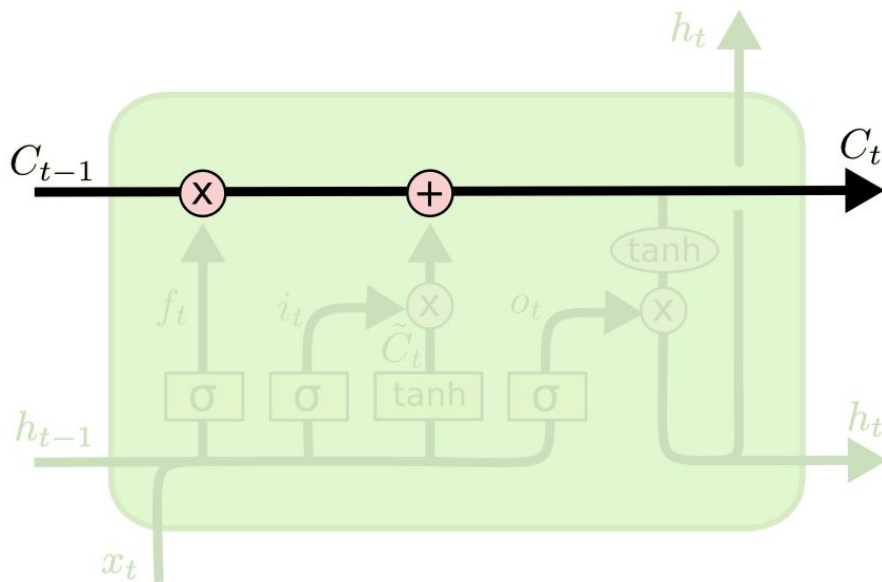
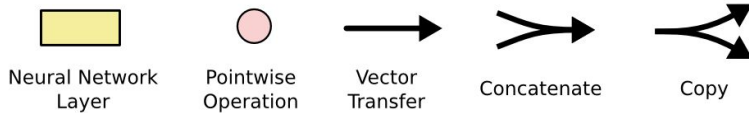
Forget → Store → Update → Output

- Forget
  - Compute what information you want to forget from the LSTM memory
- Store
  - Compute what information you want to add to the LSTM memory
- Update
  - Update the cell state with the results from “forget” and “store”
- Output
  - Deduce the output for the current time step



acm.ai

# LSTM Cell

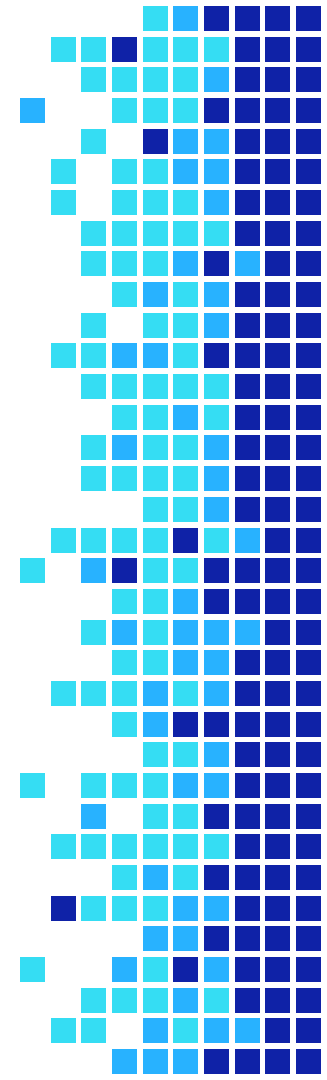




# acm.ai

## LSTM Cell

- You can think of the cell as a “conveyor belt” that stores all of the “memory” from previous time steps
- Over each time step, the cell state can be updated with “gates” that process the input and the previous hidden layer output
- The diagram on the previous slide shows the 2 gates that filter/add information to the cell state, marked with the 2 red circles





# LSTM Forget Gate

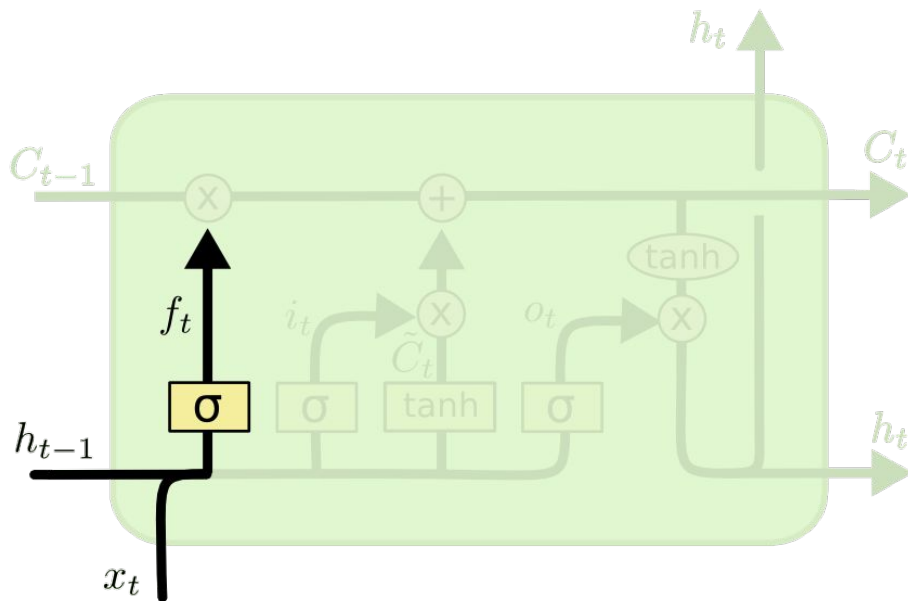
Neural Network  
Layer

Pointwise  
Operation

Vector  
Transfer

Concatenate

Copy



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

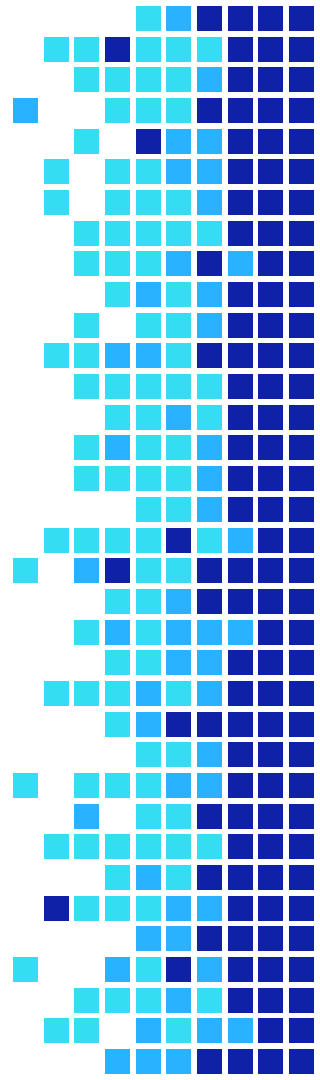


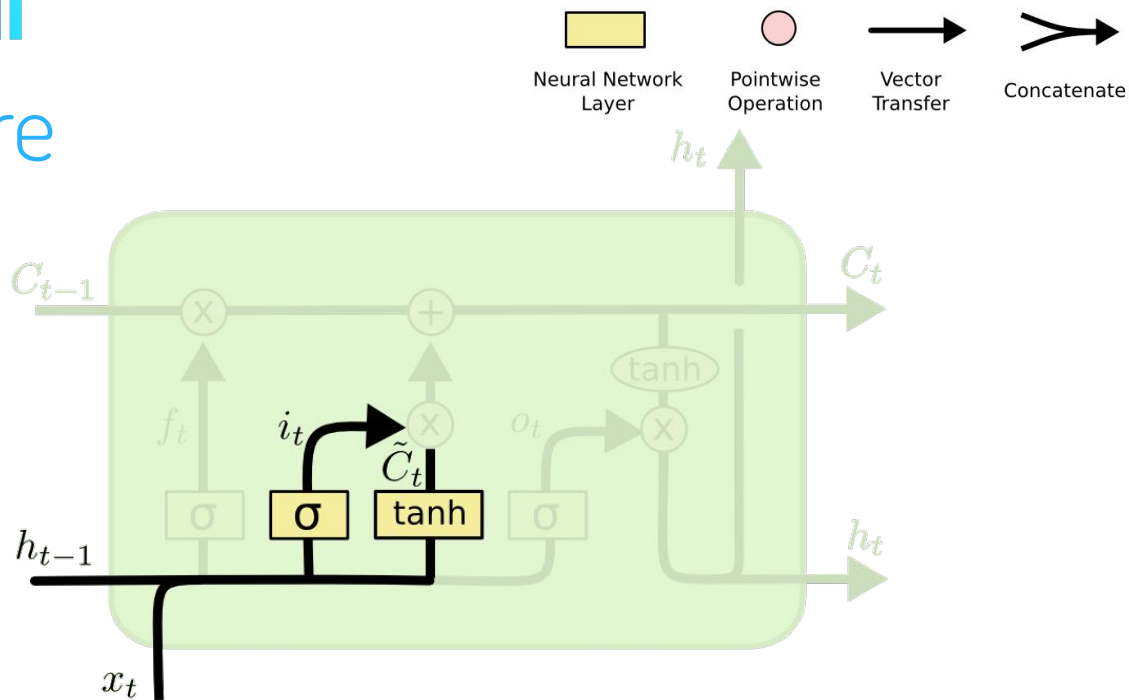
# LSTM Forget

**Forget** → Store → Update → Output

- The application of the sigmoid function (maps to (0,1)) on the hidden state acts as a sort of filter as to what it deems of enough importance to pass onto the cell state
  - '1' = keep this completely
  - '0' = forget this completely
- This result is then passed into the Update phase to eventually remove unwanted "memory" from the LSTM cell state

$$f_t * C_{t-1}$$





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

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

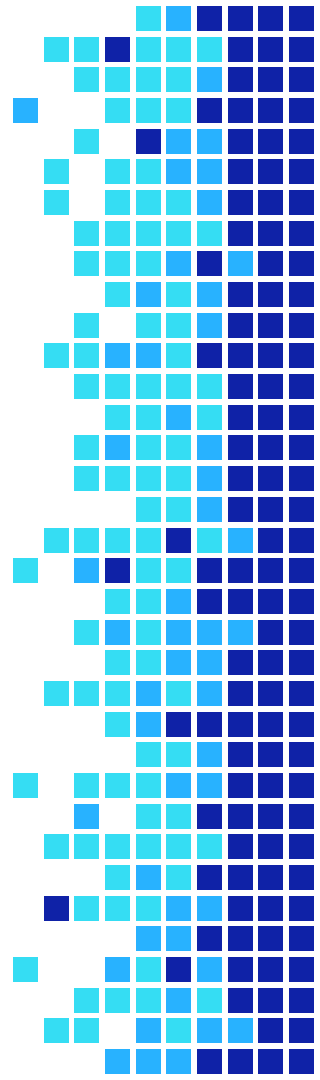


# LSTM Store

Forget → **Store** → Update → Output

- Input passed through a sigmoid layer, effectively deciding which values are to be updated.
- “Tanh” layer creates a vector of candidate values
- Both of the outputs from these subprocesses are then combined to Update the state in the next step, which will add new “memory” to the LSTM cell state

$$i_t * \tilde{C}_t$$





acm.ai

# LSTM Update

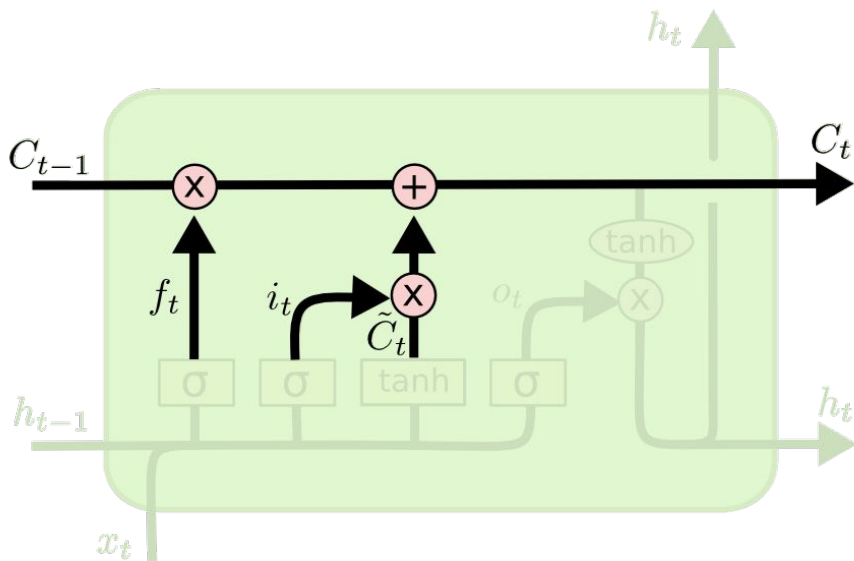
Neural Network  
Layer

Pointwise  
Operation

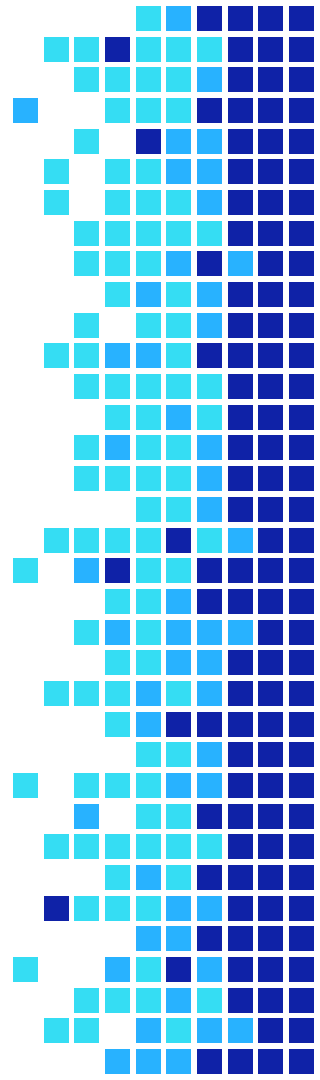
Vector  
Transfer

Concatenate

Copy



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



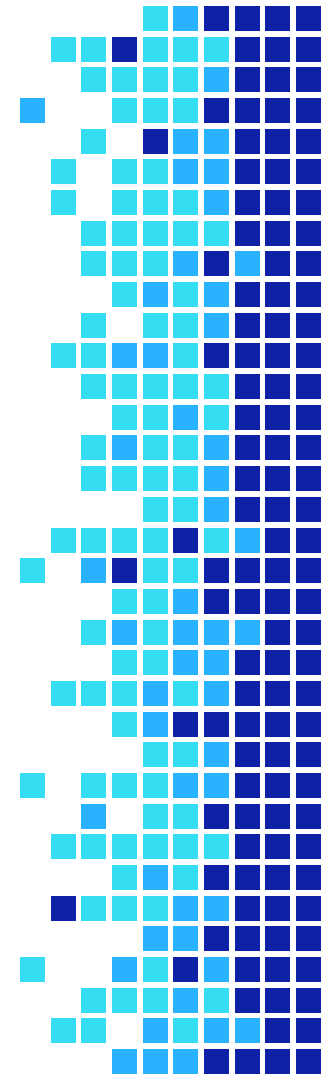


acm.ai

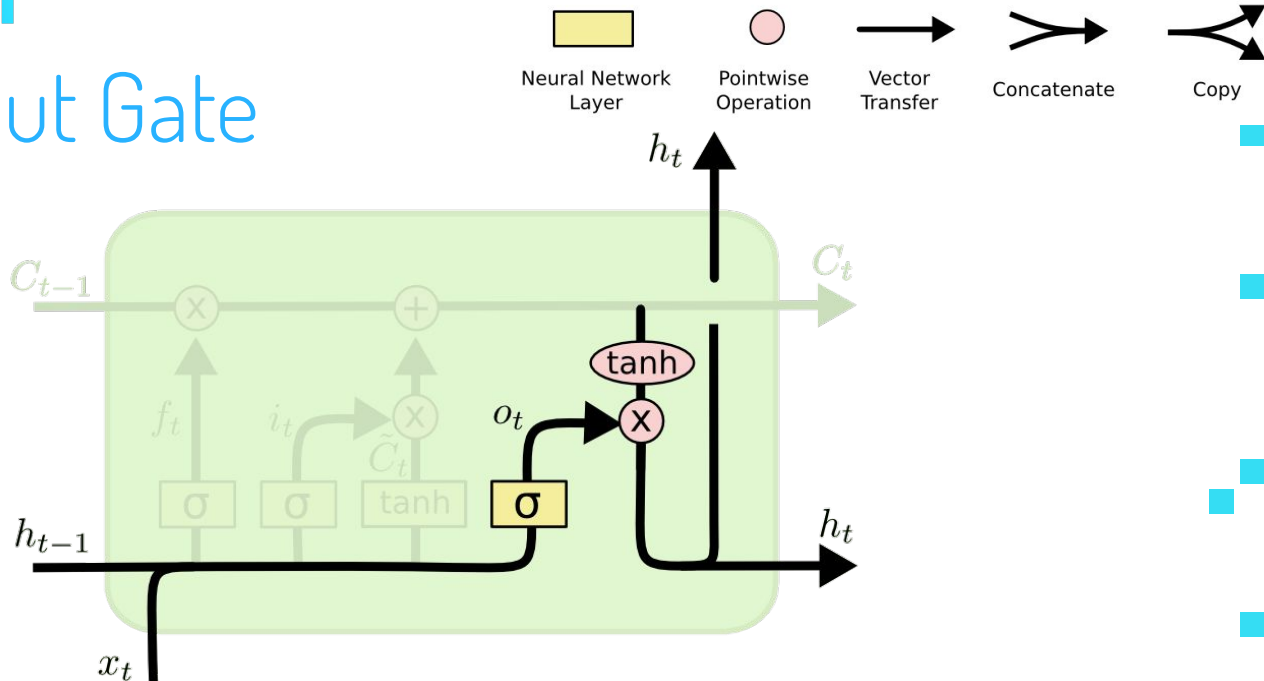
# LSTM Update

Forget → Store → **Update** → Output

- Apply what was computed from the forget and store phases to the output
  - This output then acts as the incoming cell state to the next cell in the recurrent layer
- Reminder: The trainable parameters in the LSTM are shared across all time steps, just like the trainable parameters in the standard RNN model we saw at the beginning of the workshop!



# LSTM Output Gate



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

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

# LSTM Output

Forget → Store → Update → **Output**

- First, filter the output with the sigmoid layer
- Then the tanh layer is applied to the cell state which (maps to  $(-1, 1)$ )
  - These results are then multiplied and passed on as both the output for this cell and the hidden state of the next cell



# Summary

- Forget gate - acts on the old stored cell state  $c(t-1)$  of the previous cell
- Store/Update gate - acts on the new values computed by the current cell
- Output gate - acts on the  $c(t)$  value computed by the current cell
- Trainable parameters:  **$W_f, b_f, W_i, b_i, W_o, b_o$**

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

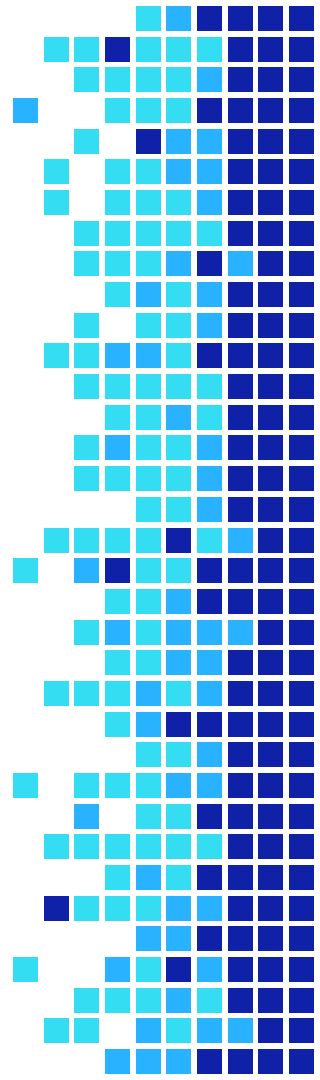
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

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

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

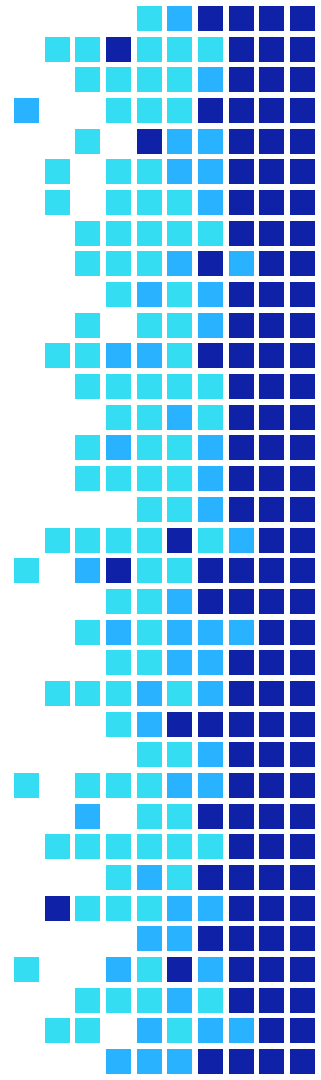
$$h_t = o_t * \tanh(C_t)$$



# Poll

The values of each of the gates is computed using :

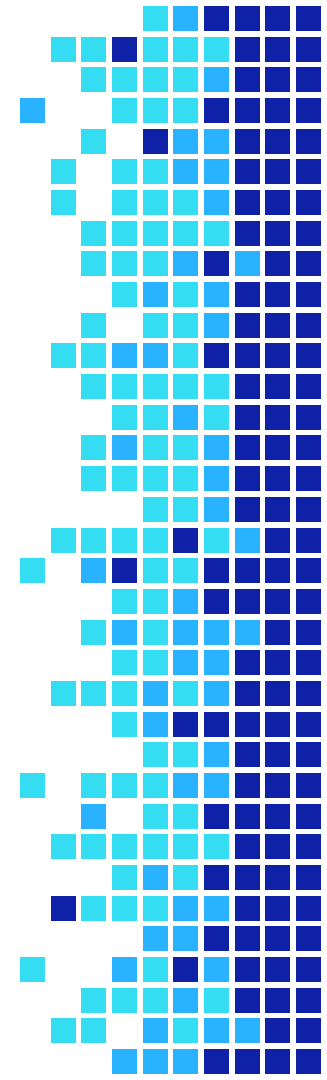
- A. The previous cell state and the current cell state
- B. They are constants shared across the entire network
- C. The previous cell output and current cell input
- D. The previous cell state and the current cell input
- E. None of the above



## 5. Applications of RNNs

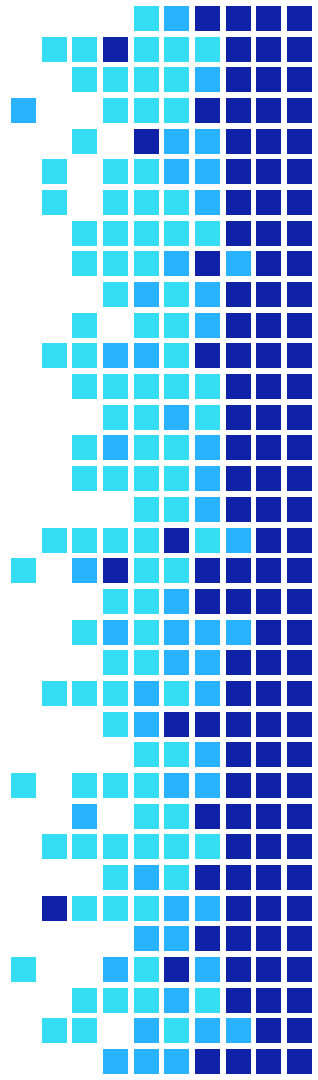
# LSTMs/RNNs used in NLP

- NLP = Natural Language Processing
  - Speech Recognition
  - Language Translation
  - Sentiment analysis

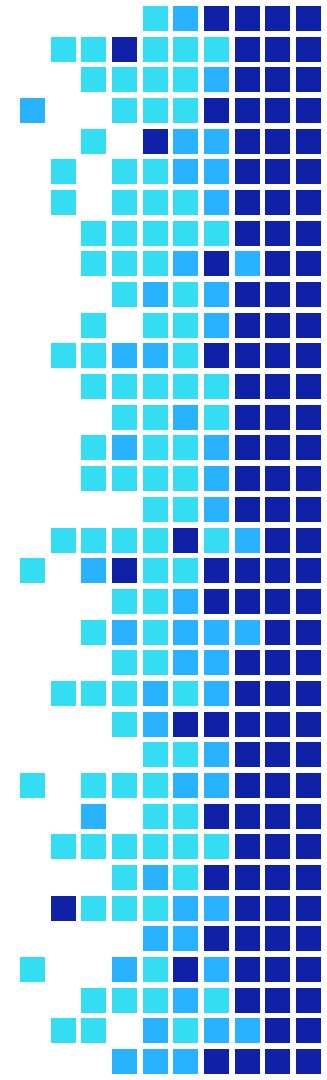


# The current *state* of NLP: GPT-3

- 175 billion parameters
- Trained on 45 TB of text data
- <https://openai.com/blog/openai-api/>
- Very cool writing samples:
  - <https://www.gwern.net/GPT-3>
- Primarily uses **Transformers**



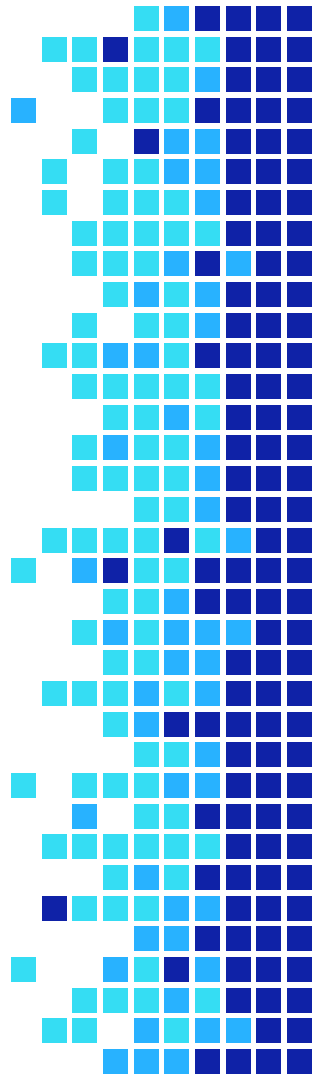
# 6. Transformers!



# Problems with RNNs

- Vanishing Gradient
- Long Term Memory loss
- Cannot be parallelized

[https://towardsdatascience.com/  
the-fall-of-rnn-lstm-2d1594c74ce0](https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0)



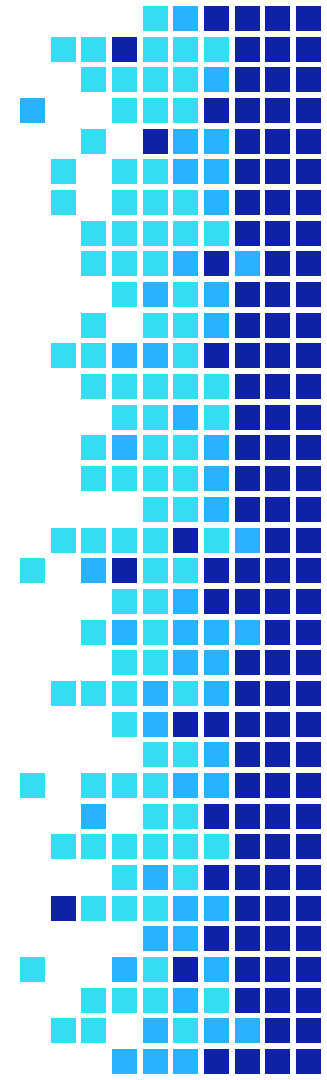
# Attention is All You Need

- Paper: <https://arxiv.org/abs/1706.03762> (2017)
- Illustrated Explanation:  
<https://jalammar.github.io/illustrated-transformer/>



# Discussion

1. Describe one-hot encodings
2. What kind of data are recurrence models most useful for?
3. Can you use recurrence with convolutional models?



# Discussion Answers

- <https://tinyurl.com/advworksheet5>

# Conda environment setup

Please read through and complete the steps on [this](#) document to make sure you're all set up for the project we'll be doing toward the end of Advanced Track. It should only take ~20 minutes and will teach you valuable foundational knowledge that every computer/data scientist should be familiar with. Please try to do it by the sixth workshop, which is when we'll begin using machine learning tools hands on. We'll also likely hold a short optional session where we'll go through this setup.



“

*Questions?*

## Other Resources

- [RNN Effectiveness](#)
- [RNNs Explained](#)
- [LSTMs Explained](#)
- [Attention Explained with Visuals](#)
- [Illustrated Transformers](#)

# Thank you all for coming!

**Anonymous Feedback:** [tinyurl.com/w21advtrackfb5](https://tinyurl.com/w21advtrackfb5)

**Office hours : Thursdays, 9-10pm** (Right after this workshop!)  
on the ACM discord - AI voice channel

**Facebook Group:** [www.facebook.com/groups/uclaacmai/](https://www.facebook.com/groups/uclaacmai/)