



# Recurrent Neural Networks



# Deep Learning

- We've used Neural Networks to solve Classification and Regression problems, but we still haven't seen how Neural Networks can deal with sequence information.
- For this we use Recurrent Neural Networks



# Deep Learning

- RNN Theory
- Basic Manual RNN
- Vanishing Gradients
- LSTM and GRU Units
- Time Series with RNN
- Time Series Exercise / Solutions
- Word2Vec



# Let's get started!



# Recurrent Neural Networks Theory



# Deep Learning

- Examples of Sequences
  - Time Series Data (Sales)
  - Sentences
  - Audio
  - Car Trajectories
  - Music



# Deep Learning

- Let's imagine a sequence:
  - [1,2,3,4,5,6]
- Would you be able to predict a similar sequence shifted one time step into the future?
  - [2,3,4,5,6,7]



# Deep Learning

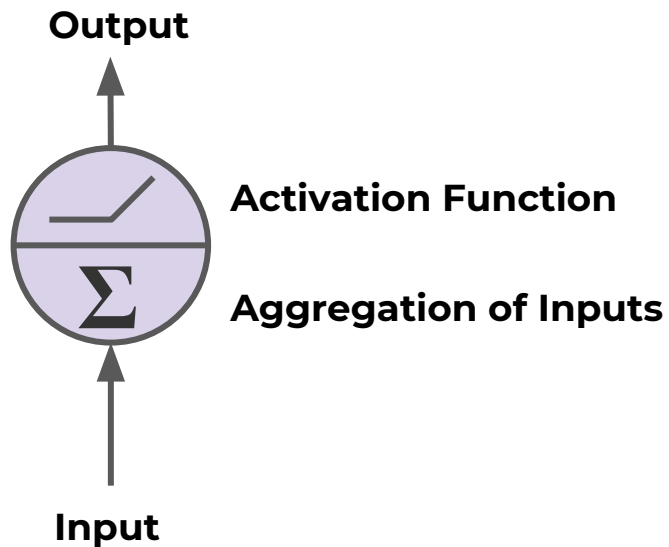
- Let's imagine a sequence:
  - [1,2,3,4,5,6]
- Would you be able to predict a similar sequence shifted one time step into the future?
  - [2,3,4,5,6,7]





# Deep Learning

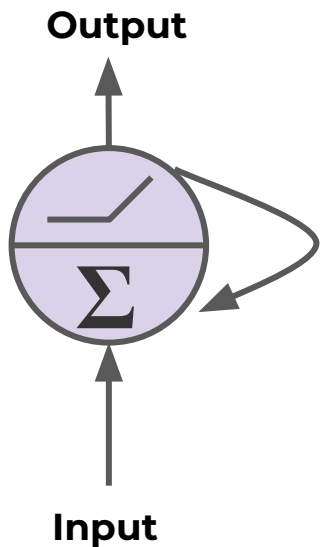
- Normal Neuron in Feed Forward Network





# Deep Learning

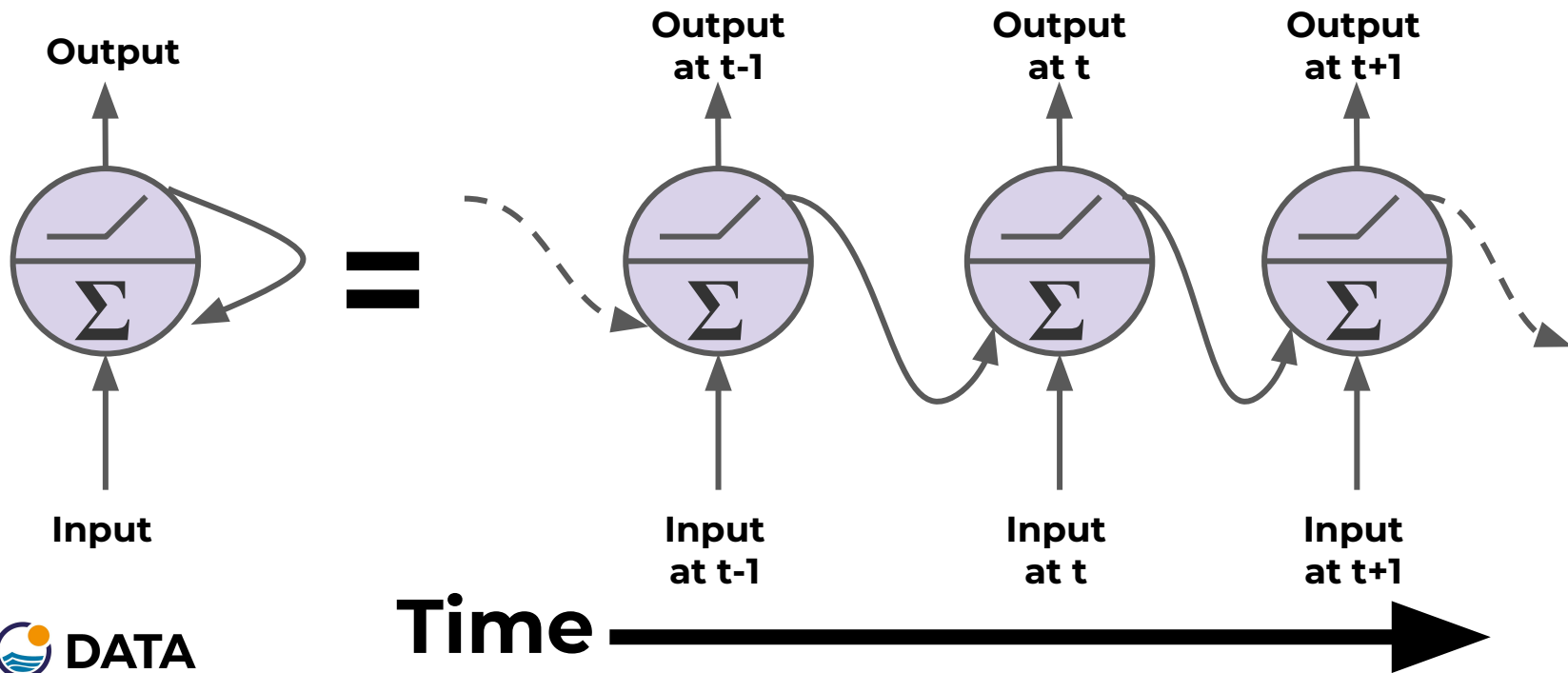
- Recurrent Neuron - Sends output back to itself!
  - Let's see what this looks like over time!





# Deep Learning

- Recurrent Neuron





# Deep Learning

- Cells that are a function of inputs from previous time steps are also known as *memory cells*.
- RNN are also flexible in their inputs and outputs, for both sequences and single vector values.



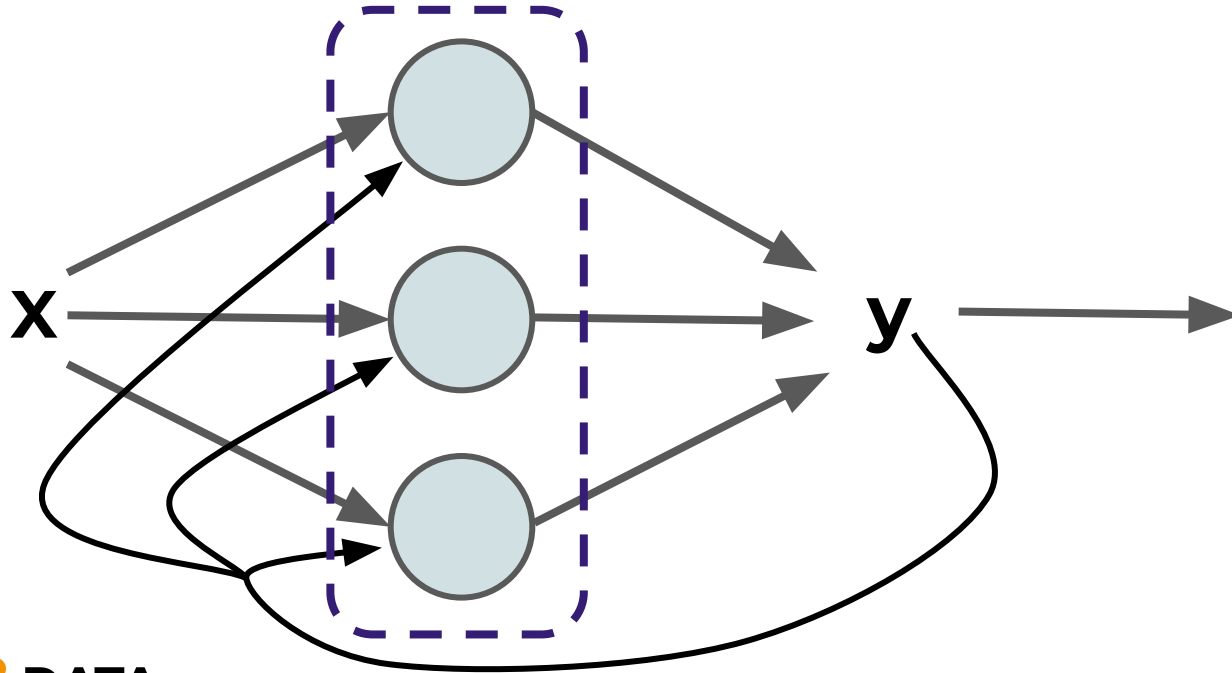
# Deep Learning

- We can also create entire layers of Recurrent Neurons...



# Deep Learning

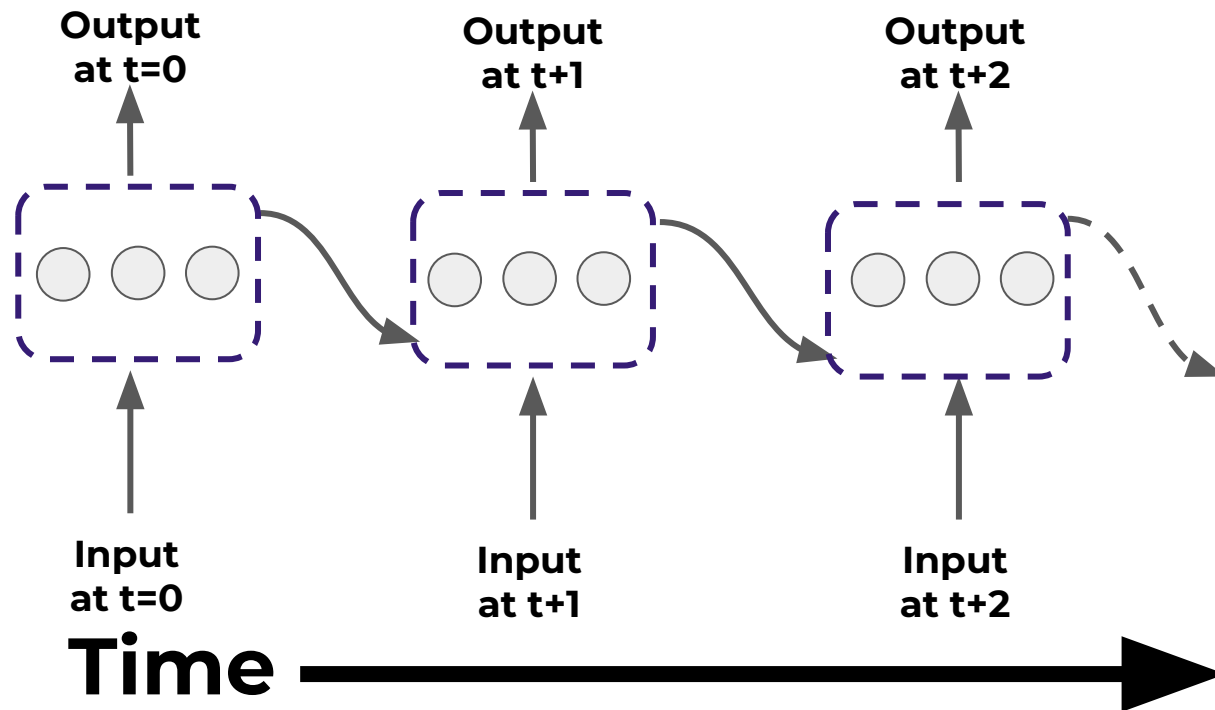
- RNN Layer with 3 Neurons:





# Deep Learning

- “Unrolled” layer.





# Deep Learning

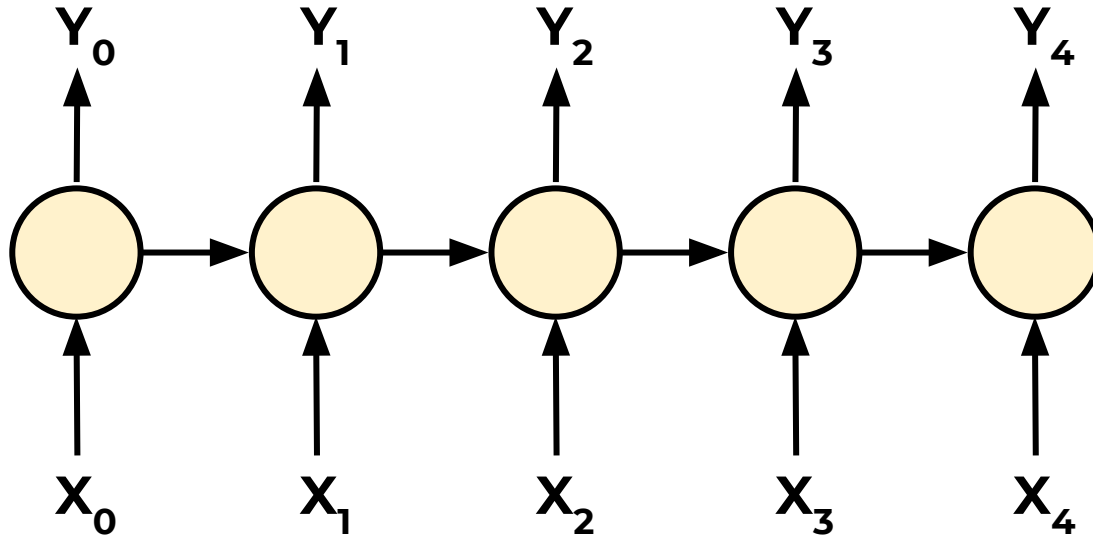
- RNN are also very flexible in their inputs and outputs.
- Let's see a few examples.





# Deep Learning

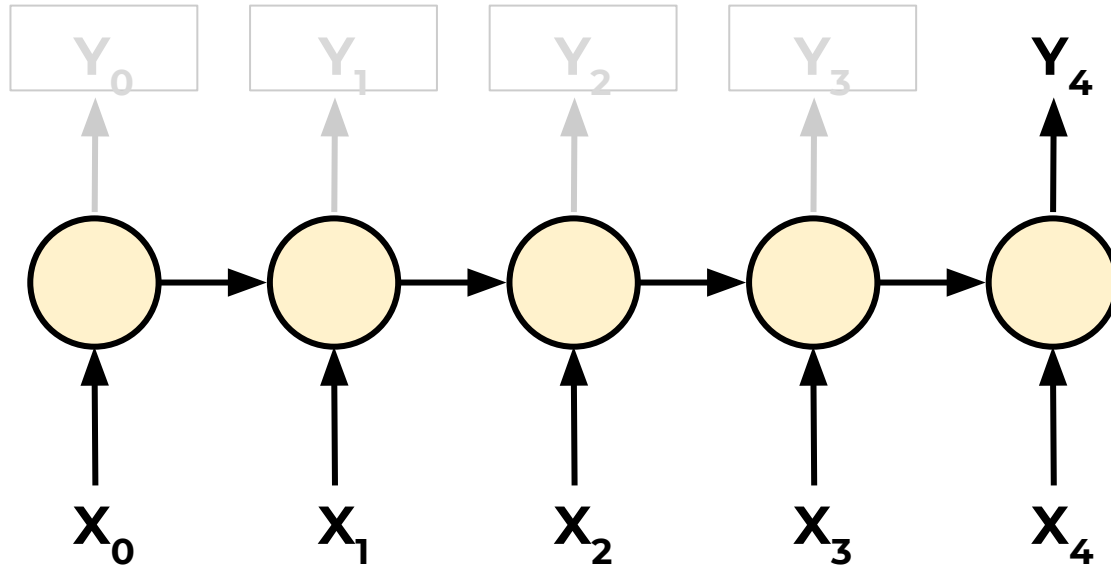
- Sequence to Sequence





# Deep Learning

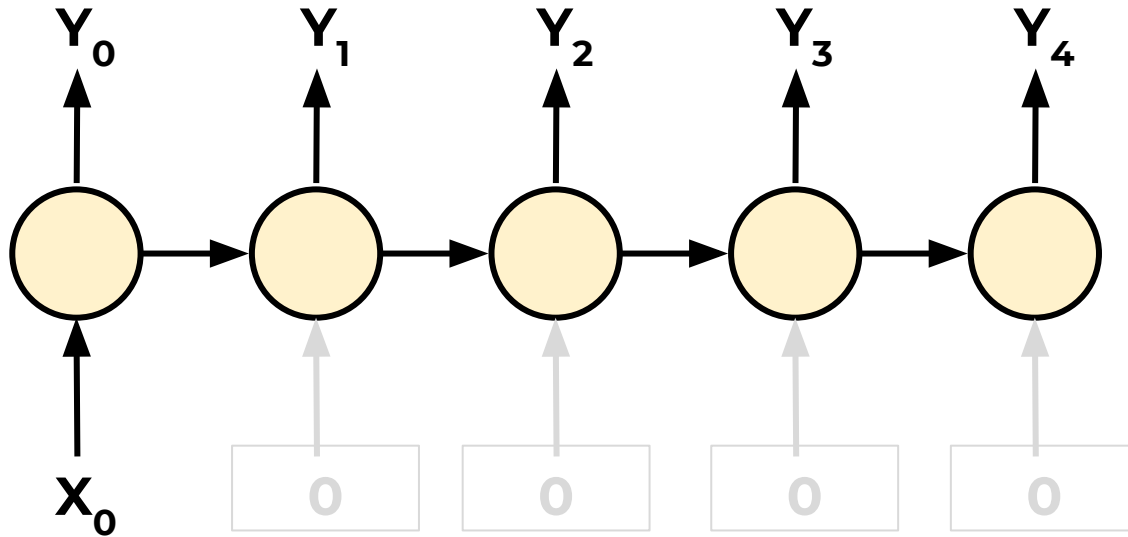
- Sequence to Vector





# Deep Learning

- Vector to Sequence





# Deep Learning

- Let's explore how we could build a simple RNN model in TensorFlow manually.
- Afterwards we'll see how to use TensorFlow's built in RNN API classes!



# Manual RNN with TF



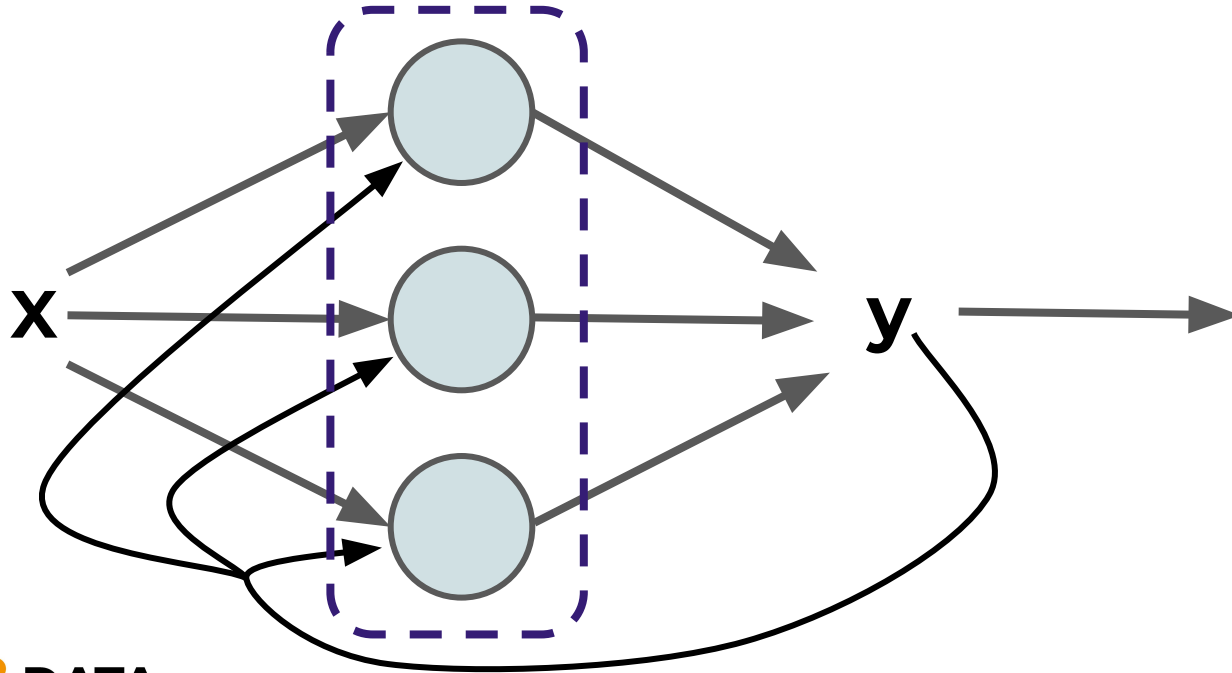
# Deep Learning

- In this lecture we'll manually create a 3 neuron RNN layer with TensorFlow.
- The main idea to focus on here is the input format of the data.
- Let's quickly review what we will create.



# Deep Learning

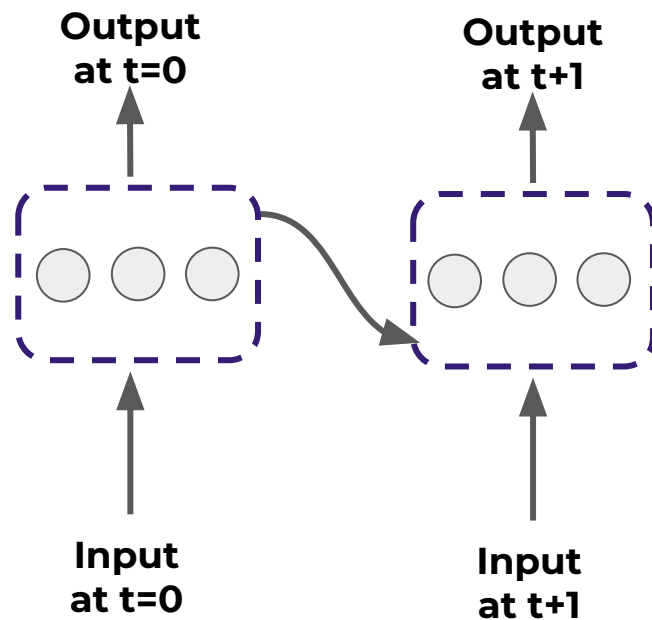
- We'll construct the following RNN Layer:





# Deep Learning

- “Unrolled” layer.







# Deep Learning

- We'll start by running the RNN for 2 batches of data,  $t=0$  and  $t=1$
- Each Recurrent Neuron has 2 sets of weights:
  - $W_x$  for input weights on  $X$
  - $W_y$  for weights on output of original  $X$



## Example of RNN Data

t=0	t=1	t=2	t=3	t=4
[ The,	brown,	fox,	is,	quick]
[ The,	red,	fox,	jumped,	high]

words\_in\_dataset[0] = [The, The]

words\_in\_dataset[1] = [brown, red]

words\_in\_dataset[2] = [fox, fox]

words\_in\_dataset[3] = [is, jumped]

words\_in\_dataset[4] = [quick, high]

num\_batches = 5, batch\_size = 2, time\_steps = 5



# Vanishing Gradients



# Deep Learning

- Backpropagation goes backwards from the output to the input layer, propagating the error gradient.
- For deeper networks issues can arise from backpropagation, vanishing and exploding gradients!



# Deep Learning

- As you go back to the “lower” layers, gradients often get smaller, eventually causing weights to never change at lower levels.
- The opposite can also occur, gradients explode on the way back, causing issues.



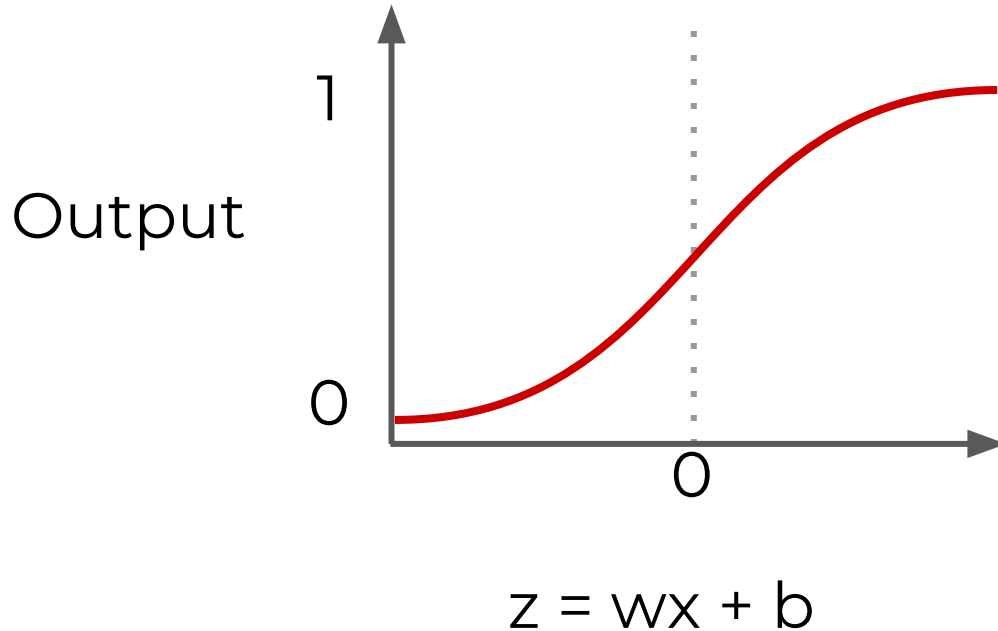
# Deep Learning

- Let's discuss why this might occur and how we can fix it.
- Then in the next lecture we'll discuss how these issues specifically affect RNN and how to use LSTM and GRU to fix them.



# Deep Learning

- Why does this happen?

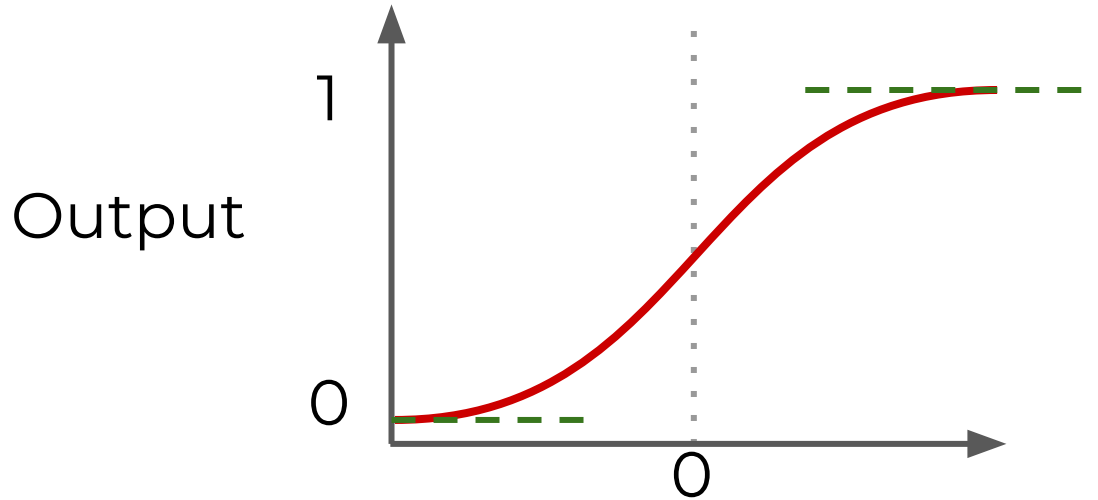


$$f(x) = \frac{1}{1 + e^{-(x)}}$$



# Deep Learning

- Why does this happen?



$$f(x) = \frac{1}{1 + e^{-(x)}}$$

$$z = wx + b$$

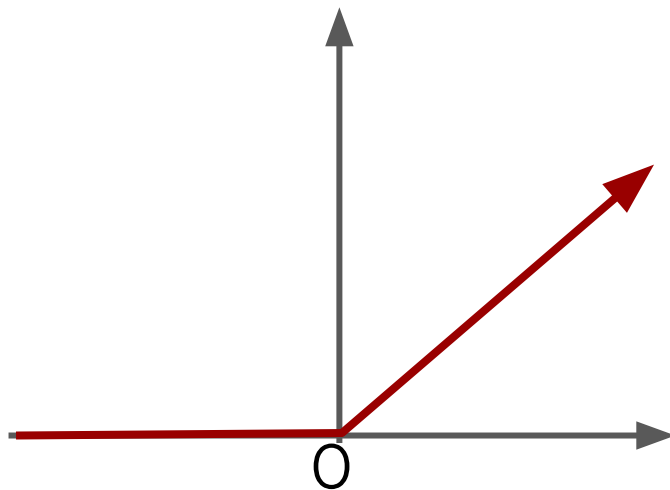




# Deep Learning

- Using Different Activation Functions

Output



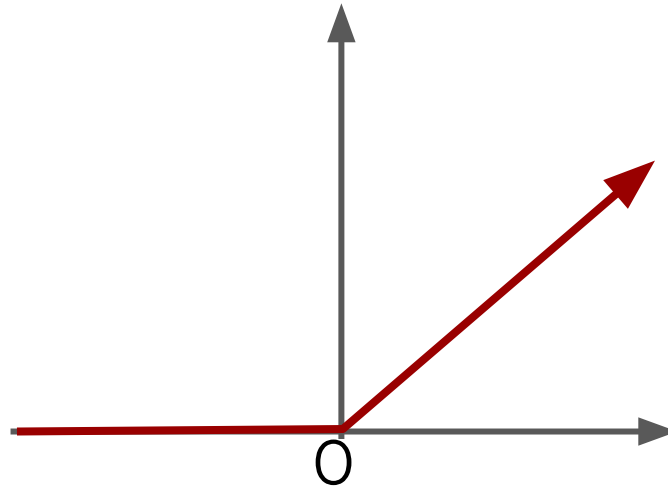
$$z = wx + b$$



# Deep Learning

- The ReLu doesn't saturate positive values.

Output



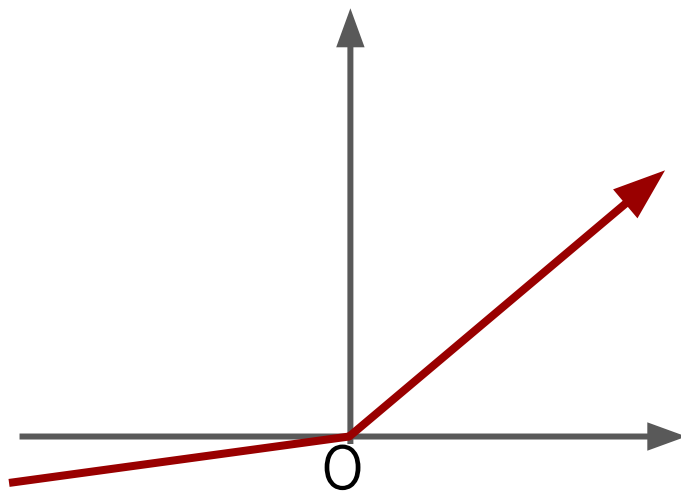
$$z = wx + b$$



# Deep Learning

- “Leaky” ReLU

Output

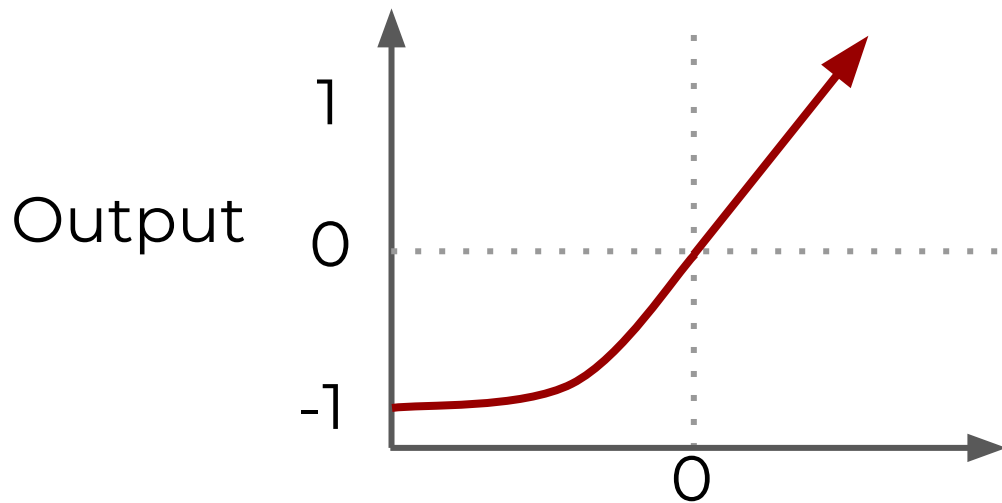


$$z = wx + b$$



# Deep Learning

- Exponential Linear Unit (ELU)



$$z = wx + b$$



# Deep Learning

- Another solution is to perform batch normalization, where your model will normalize each batch using the batch mean and standard deviation.



# Deep Learning

- Apart from batch normalization, researchers have also used “gradient clipping”, where gradients are cut off before reaching a predetermined limit (e.g. cut off gradients to be between -1 and 1)



# Deep Learning

- RNN for Time Series present their own gradient challenges, let's explore special neuron units that help fix these issues!



# LSTM and GRU





# Deep Learning

- Many of the solutions previously presented for vanishing gradients can also apply to RNN: different activation functions, batch normalizations, etc...
- However because of the length of time series input, these could slow down training



## Deep Learning

- A possible solution would be to just shorten the time steps used for prediction, but this makes the model worse at predicting longer trends.



# Deep Learning

- Another issue RNN face is that after awhile the network will begin to “forget” the first inputs, as information is lost at each step going through the RNN.
- We need some sort of “long-term memory” for our networks.



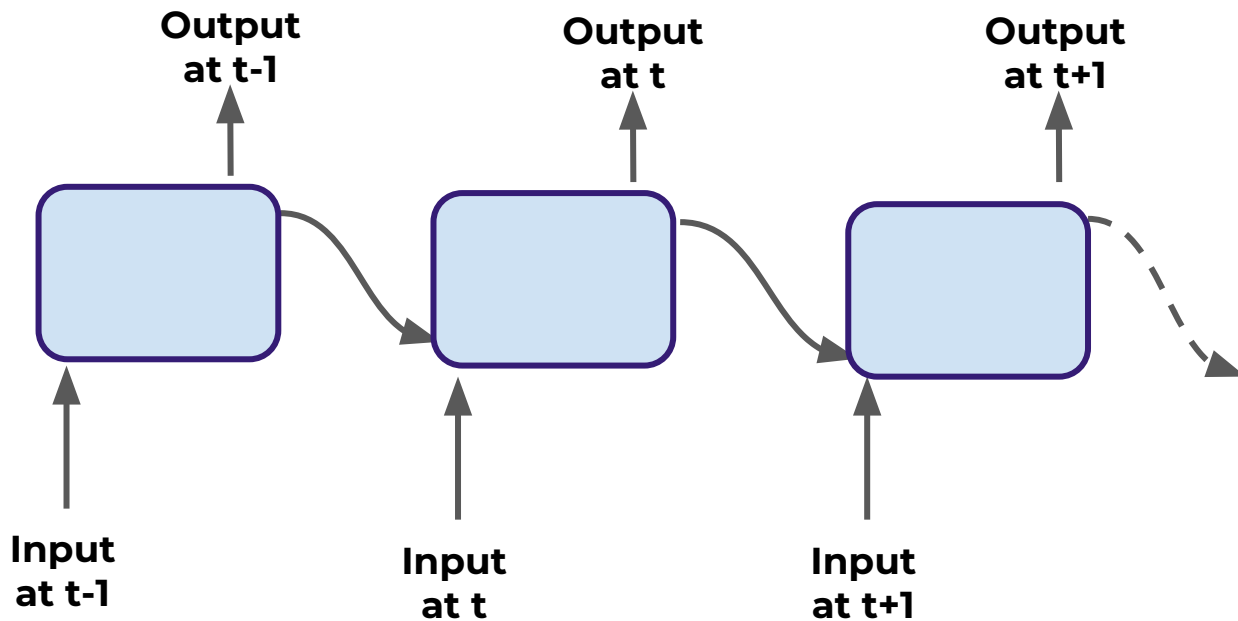
# Deep Learning

- The LSTM (Long Short-Term Memory) cell was created to help address these RNN issues.
- Let's go through how an LSTM cell works!



# Deep Learning

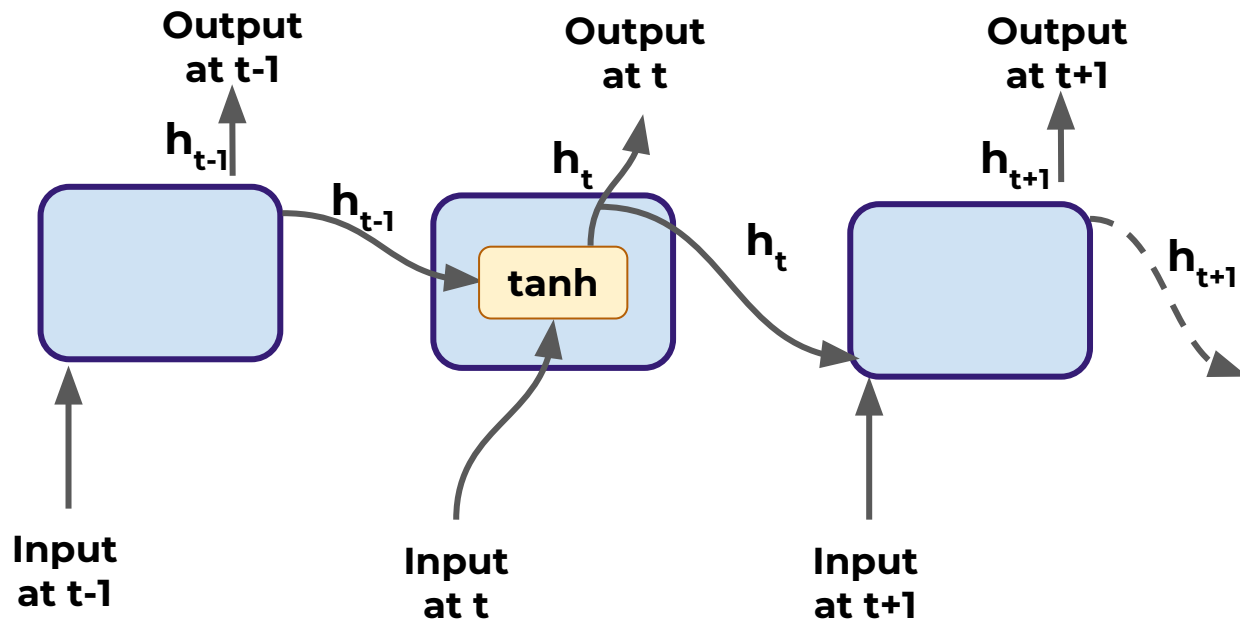
- A typical RNN





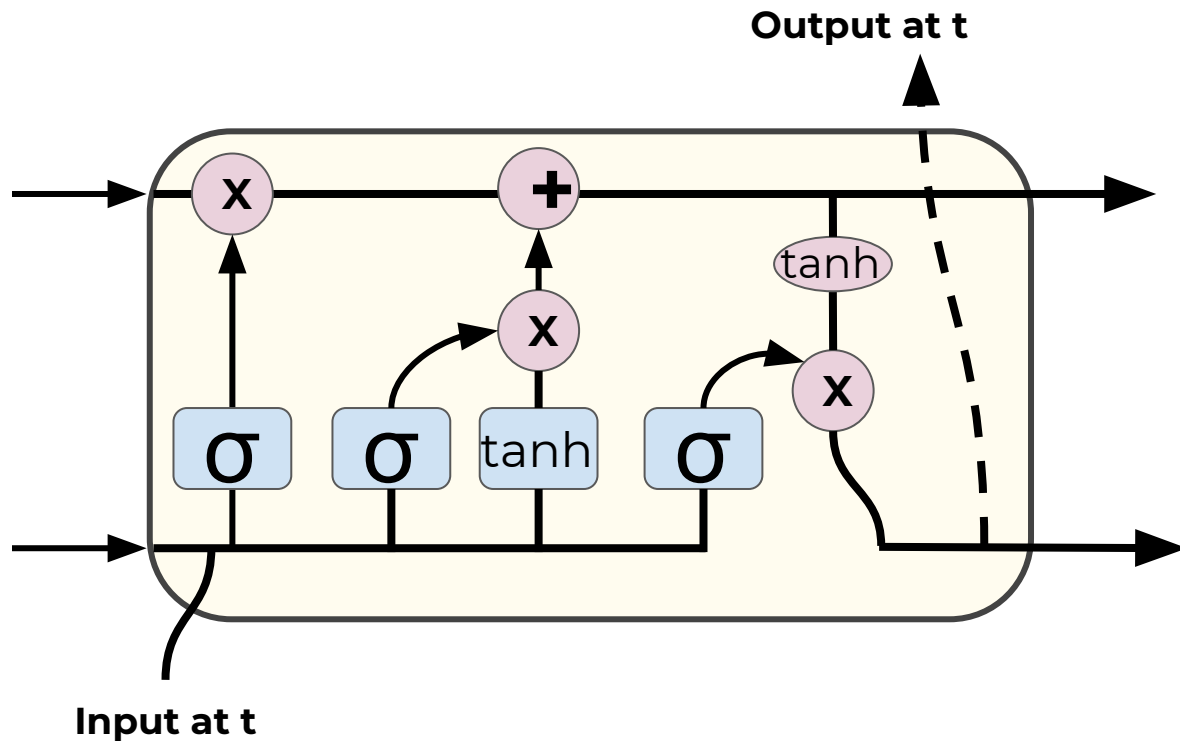
# Deep Learning

- A typical RNN cell





# An LSTM Cell

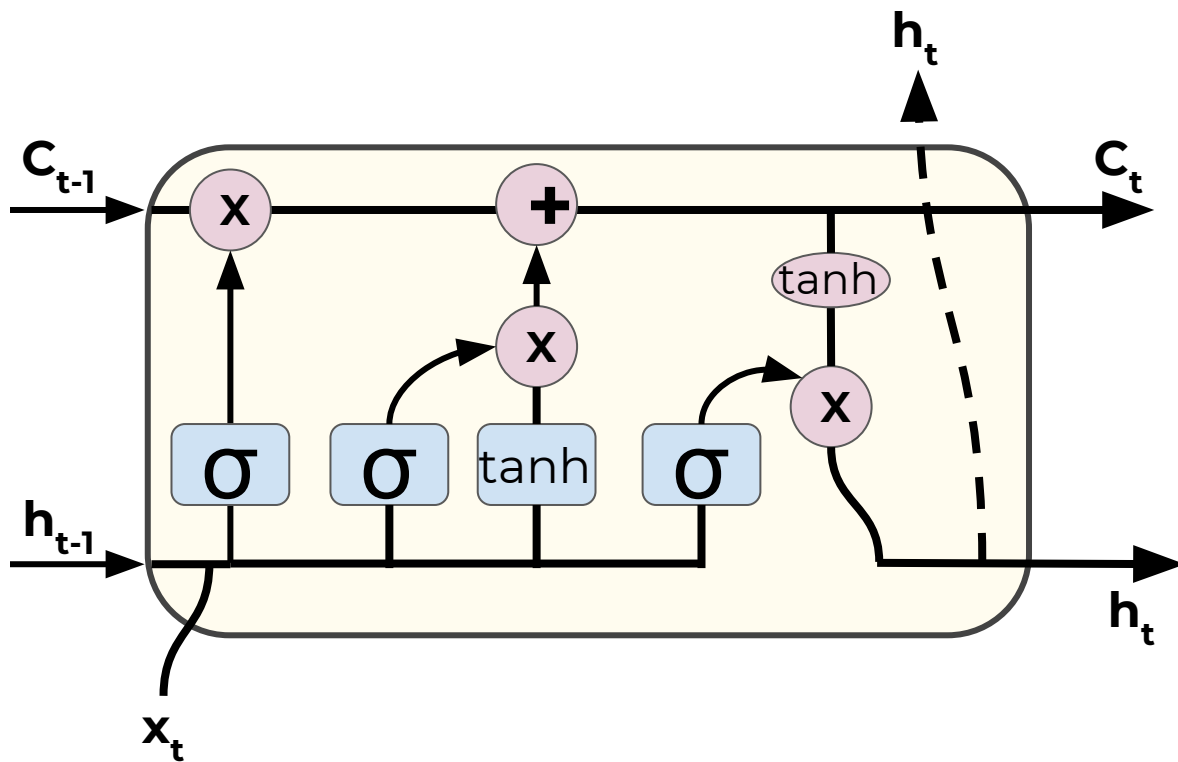


Here we can see the entire LSTM cell.

Let's go through the process!



## An LSTM Cell



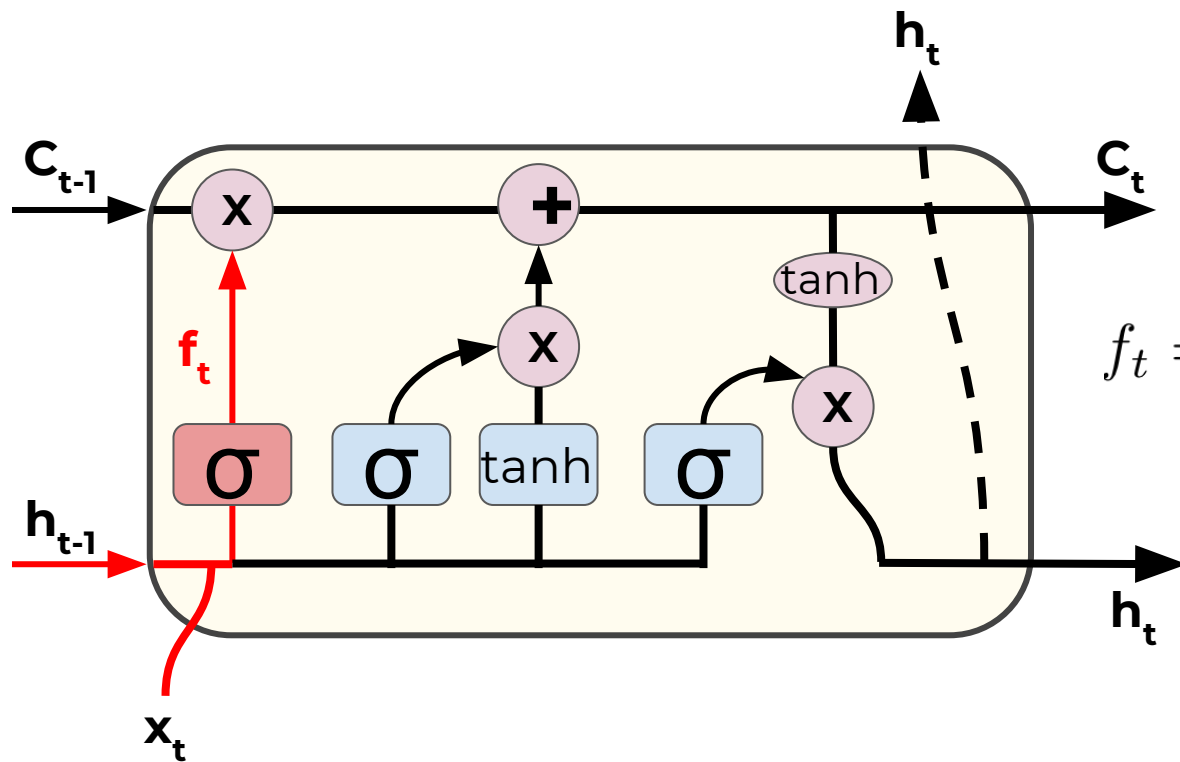
Here we can see the entire LSTM cell.

Let's go through the process!





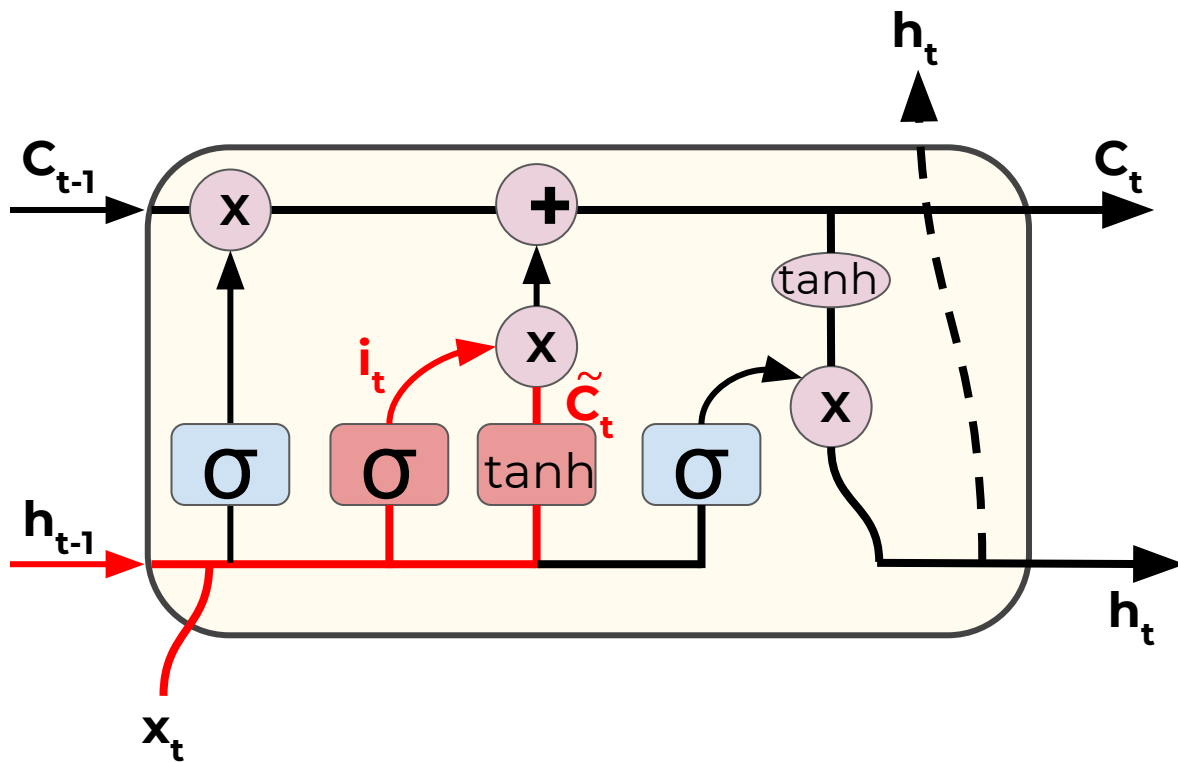
# An LSTM Cell



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



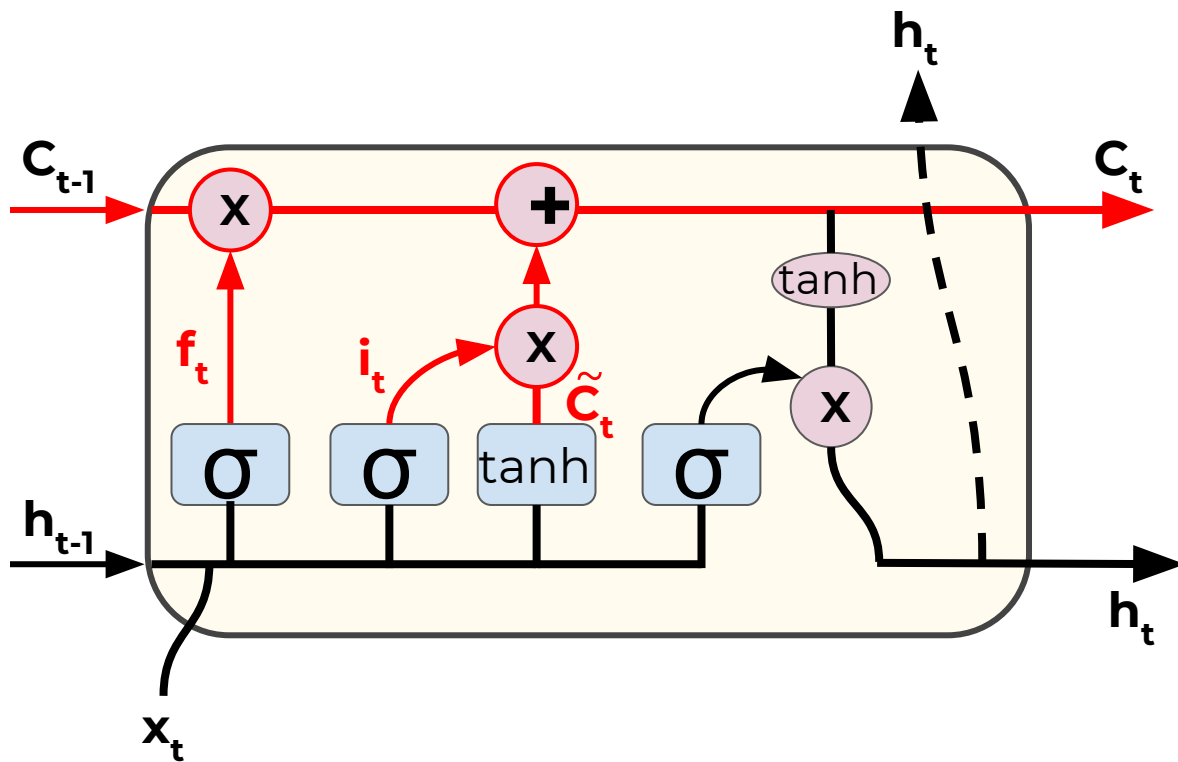
# An LSTM Cell



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



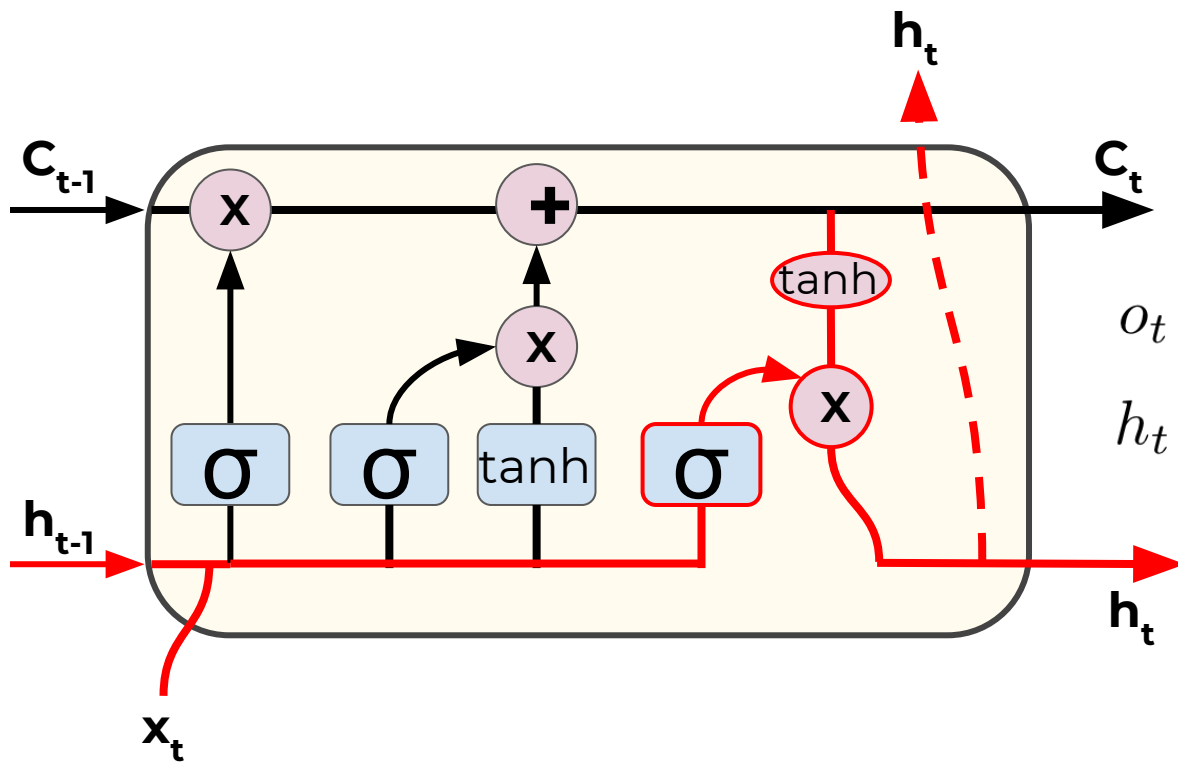
# An LSTM Cell



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



# An LSTM Cell

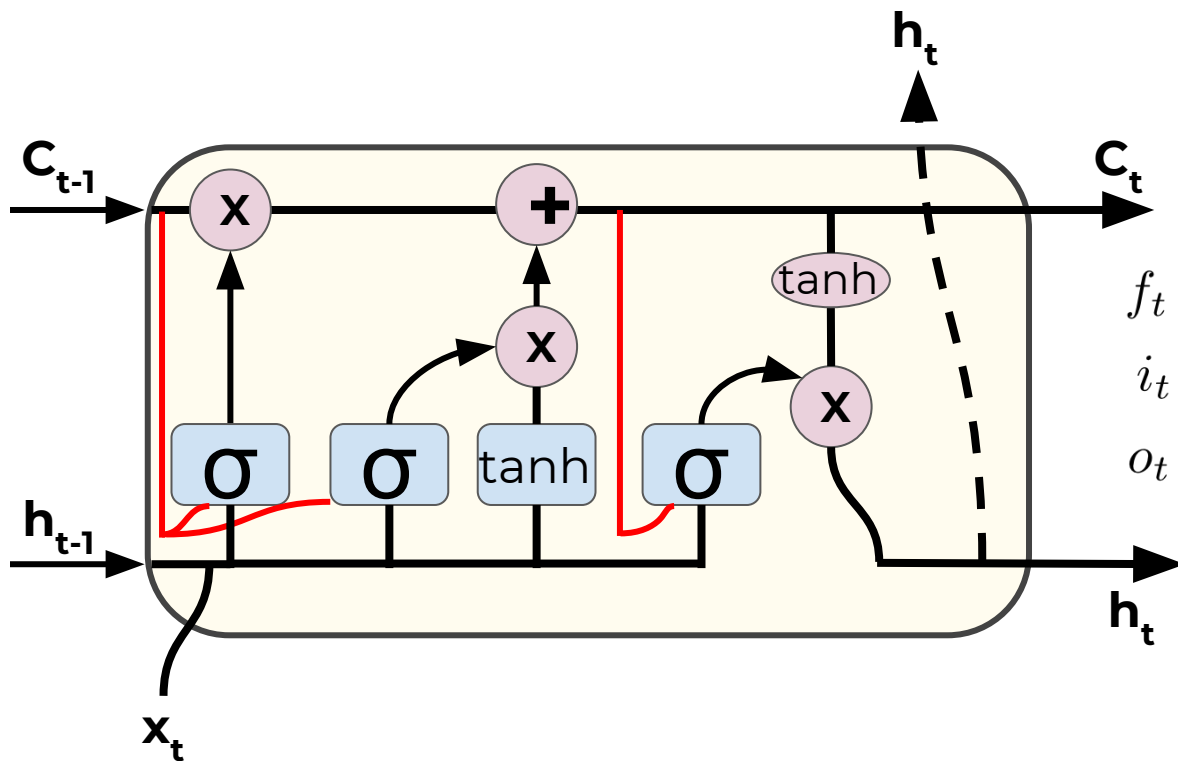


$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

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



# An LSTM Cell with “peepholes”



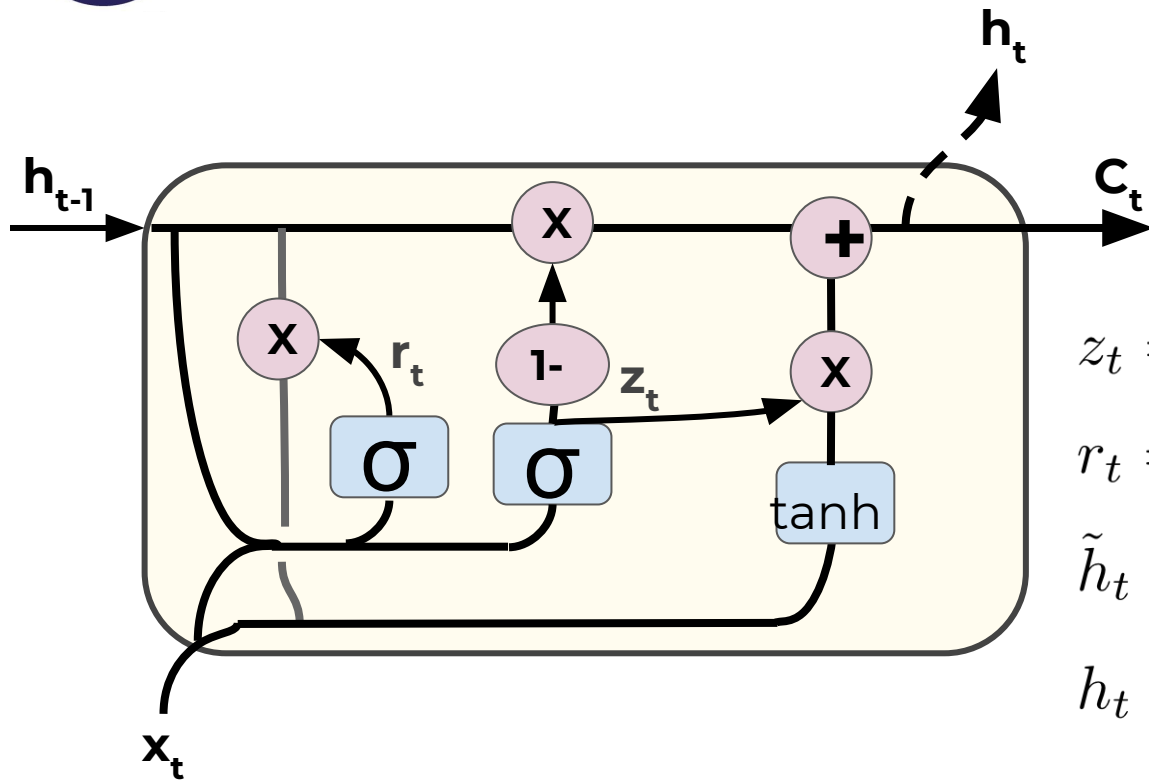
$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

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

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



# Gated Recurrent Unit (GRU)



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$



# Deep Learning

- Fortunately TensorFlow comes with these neuron models built into a nice API, making it easy to swap them in and out.
- Up next we'll explore using this TensorFlow RNN API for Time Series prediction and generation!



# RNN with TF API





## Deep Learning

- Now that we understand various possible improvements for RNN, let's use TensorFlow built-in `tf.nn` function API to solve sequence problems!



# Deep Learning

- Recall our original sequence thought exercise:
  - $[1, 2, 3, 4, 5, 6]$
- Can we predict the sequence shifted one time step forward?
  - $[2, 3, 4, 5, 6, 7]$



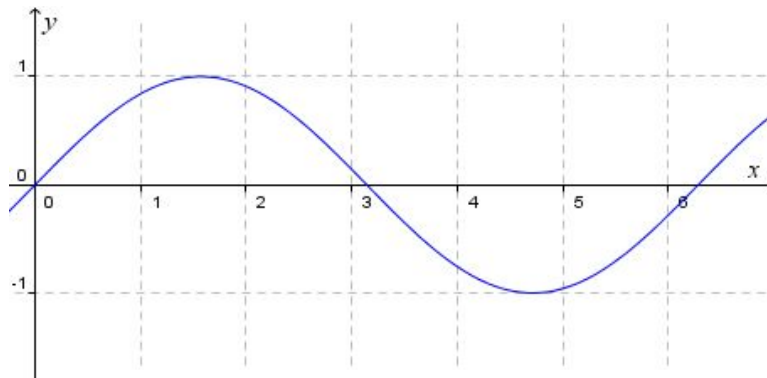
# Deep Learning

- What about this time series?
  - $[0, 0.84, 0.91, 0.14, -0.75, -0.96, -0.28]$



# Deep Learning

- What about this time series?
  - $[0, 0.84, 0.91, 0.14, -0.75, -0.96, -0.28]$
- It's actually just  $\sin(x)$ :
  - $[0.84, 0.91, 0.14, -0.75, -0.96, -0.28, 0.65]$





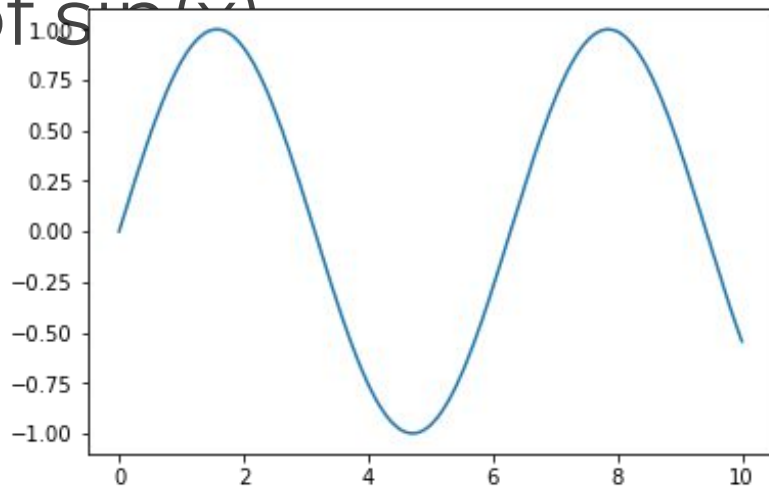
# Deep Learning

- We'll start by creating a RNN that attempts to predict a time series shifted over 1 unit into the future.
- Then we'll attempt to generate new sequences with a seed series.



# Deep Learning

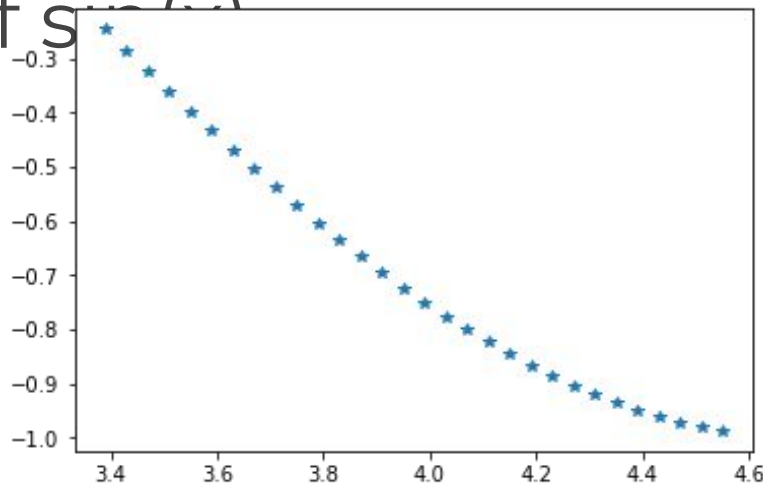
- We'll first create a simple class to generate  $\sin(x)$  and also grab random batches of  $\sin(x)$





# Deep Learning

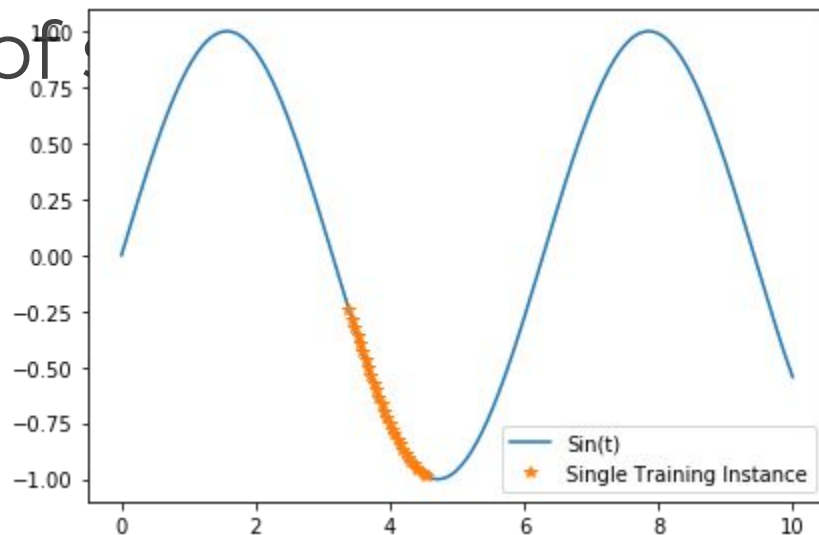
- We'll first create a simple class to generate  $\sin(x)$  and also grab random batches of  $\sin(x)$





# Deep Learning

- We'll first create a simple class to generate  $\sin(x)$  and also grab random batches of

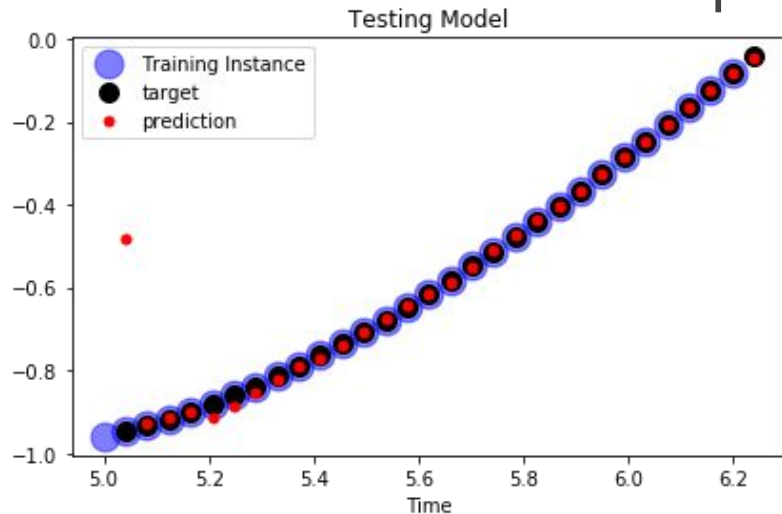






# Deep Learning

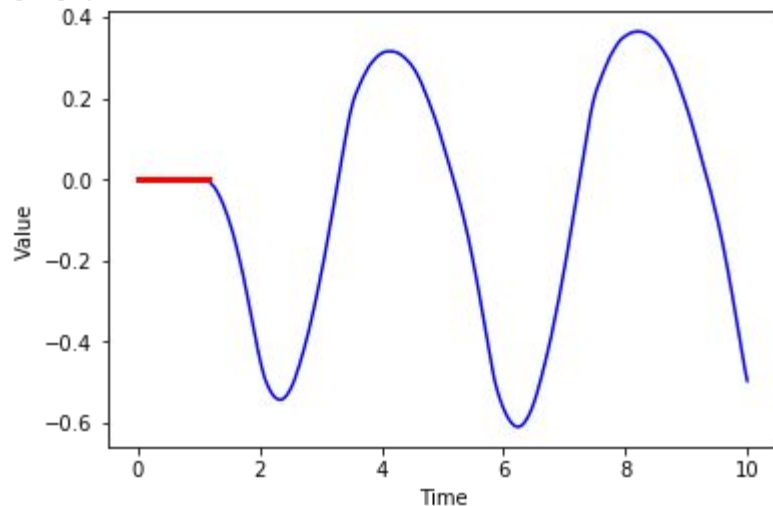
- Then the trained model will be given a time series and attempt to predict a time series shifted one time step ahead





# Deep Learning

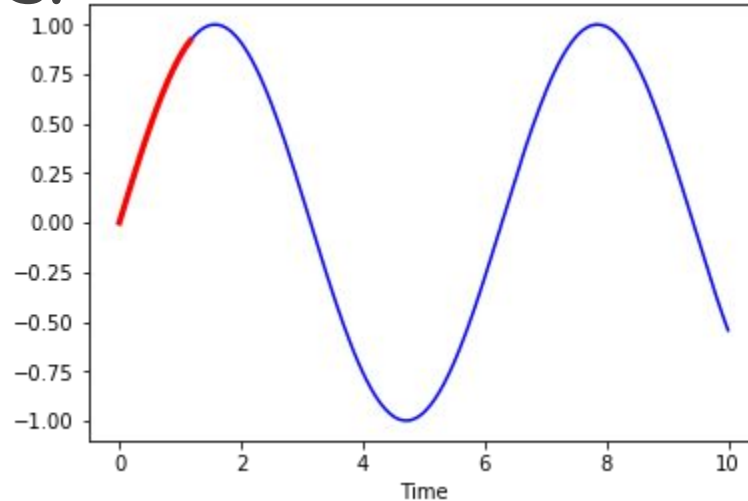
- Afterwards we'll use the same model to generate much longer time series given a seed series.





# Deep Learning

- Afterwards we'll use the same model to generate much longer time series given a seed series.





# Time Series Exercise



# Time Series Exercise Solutions



# Quick Note on Word2Vec



## Deep Learning

- Optional series of lectures describing Word2Vec with TensorFlow.
- Recommend you check out gensim library if you are further interested in Word2Vec.



# Word2Vec





# Deep Learning

- Now that we understand how to work with time series of data, let's take a look at another common series data source, words.
- For example a sentence can be:
  - ["Hi", "how", "are", "you"]



# Deep Learning

- In “classic” NLP , words are typically replaced by numbers indicating some frequency relationship to their documents.
- In doing this, we lose information about the relationship between the words themselves.



# Deep Learning

- Count-Based
  - Frequency of words in corpus
- Predictive Based
  - Neighboring words are predicted based on a vector space



# Deep Learning

- Let's explore one of Neural Network's most famous use cases in natural language processing:
  - The Word2Vec model created by Mikolov et al.



# Deep Learning

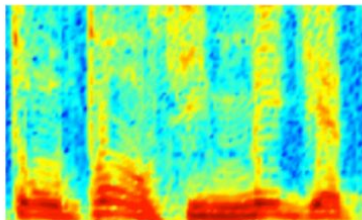
- The goal of the Word2Vec model is to learn word embeddings by modeling each word as a vector in n-dimensional space.
- But why use word-embeddings?



# Deep Learning

## Representation of Data

### AUDIO



Audio Spectrogram

DENSE

### IMAGES

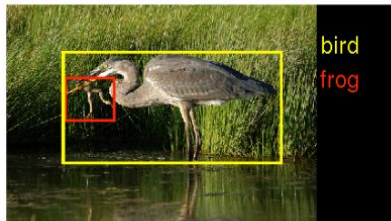


Image pixels

DENSE

### TEXT

0	0	0	0.2	0	0.7	0	0	0	...	...
---	---	---	-----	---	-----	---	---	---	-----	-----

Word, context, or  
document vectors

SPARSE



# Deep Learning

- Word2Vec creates vector spaced models that represent (embed) words in a continuous vector space.
- With words represented as vectors we can perform vector mathematics on words (e.g. check similarity, add/subtract vectors)



# Deep Learning

- At the start of training each embedding is random, but through backpropagation the model will adjust the value of each word vector in the given number of dimensions.
- More dimensions means more training time, but also more “information” per



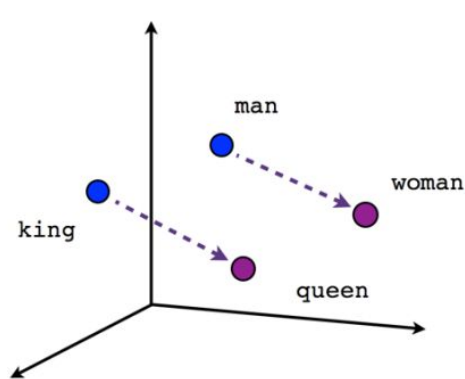


# Deep Learning

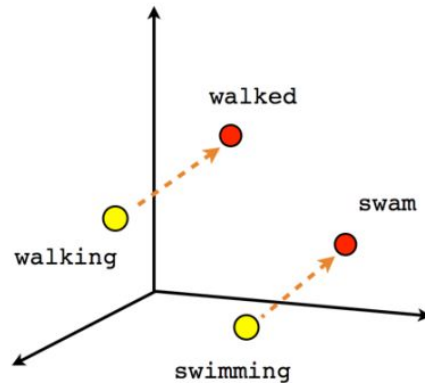
- Similar words will find their vectors closer together.
- Even more impressive, the model may produce axes that represent concepts, such as gender, verbs, singular vs plural, etc...



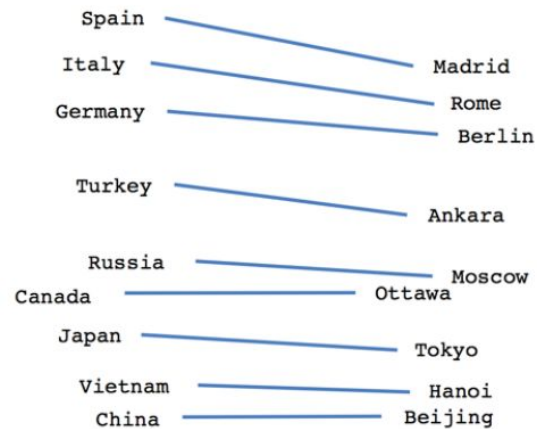
# Deep Learning



Male-Female



Verb tense



Country-Capital



# Deep Learning

- Similar words will find their vectors closer together.
- Even more impressive, the model may produce axes that represent concepts, such as gender, verbs, singular vs plural, etc...



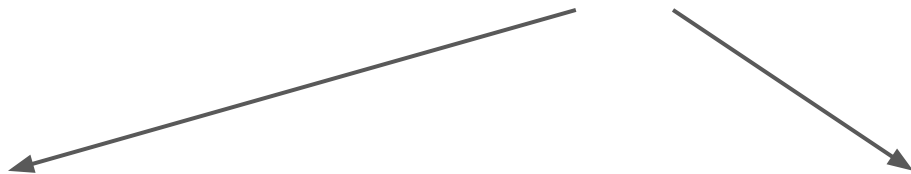
# Deep Learning

- Prediction Target
- Skip-Gram Model
  - The dog chews the bone
  - Typically better for larger data sets
- CBOW (Continuous Bag of Words)
  - The dog chews the bone
  - Typically better for smaller data sets



# Deep Learning

- The dog chews the  $w_t = ?$



bone

vs

[book, car, house, sun, ... guitar  
]

Target word

Noise words

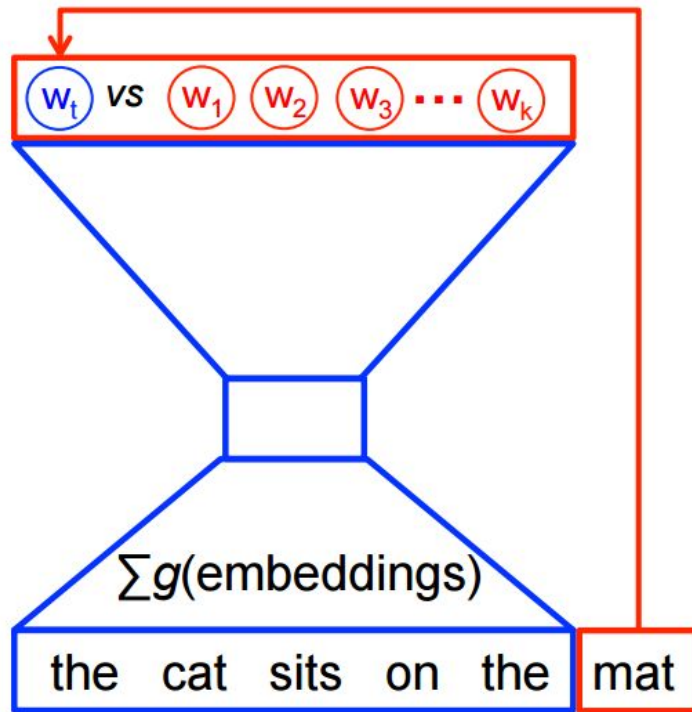


# Deep Learning

Noise classifier

Hidden layer

Projection layer





# Deep Learning

- Noise-Contrastive Training
- Target word is predicted by maximizing
  - $J_{\text{NEG}} = \log Q_{\theta}(D=1|w_t, h) + k_{n \sim P_{\text{noise}}} E [\log Q_{\theta}(D=0|w_n, h)]$
- $Q_{\theta}(D=1|w_t, h)$  is binary logistic regression is the probability that the word  $w_t$  is in the context  $h$  in the dataset  $D$  parameterized by  $\theta$ .



# Deep Learning

- Noise-Contrastive Training
- Target word is predicted by maximizing
  - $J_{\text{NEG}} = \log Q_{\theta}(D=1|w_t, h) + k_{n \sim P_{\text{noise}}} E [\log Q_{\theta}(D=0|w_n, h)]$
- $w_n$  are  $k$  words drawn from noise distribution





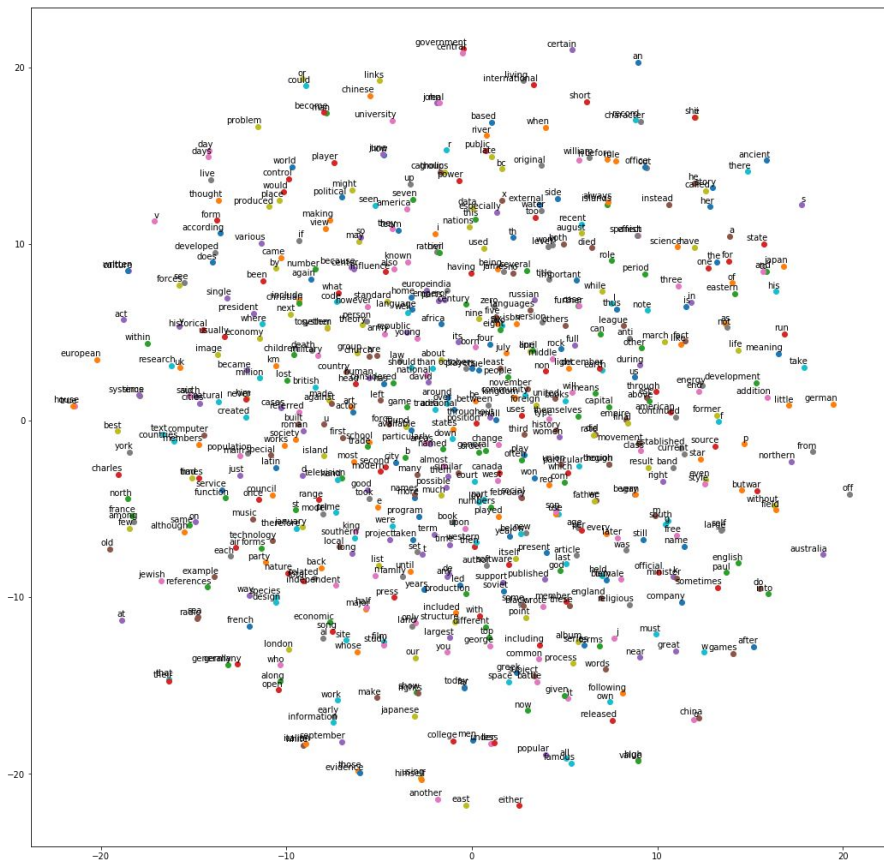
# Deep Learning

- Noise-Contrastive Training
- Target word is predicted by maximizing
  - $J_{\text{NEG}} = \log Q_{\theta}(D=1|w_t, h) + k_{n \sim P_{\text{noise}}} E [\log Q_{\theta}(D=0|w_n, h)]$
- The goal is to assign high probability to correct words and low probability to noise words.



## Deep Learning

- Once we have vectors for each word we can visualize relationships by reducing the dimensions from 150 to 2 using t-Distributed Stochastic Neighbor Embedding .





# Let's get started!



# Word2Vec Code Along



# Deep Learning

- We will be using the TensorFlow Documentation example implementation of Word2Vec.
- We will be referring to the provided notebook for blocks of code often!



# Deep Learning

- If Word2Vec is something that interests you further, check out the gensim library for Python, it has a much simpler to use API for Word2Vec and additional functionality!