Sequential Data Processing (RNNs, LSTMs)

Machine Learning Decal

Hosted by Machine Learning at Berkeley

Agenda

Motivation

Recurrent Neural Networks (RNNs)

RNN Demo

Long Short-Term Memory Networks (LSTMs)

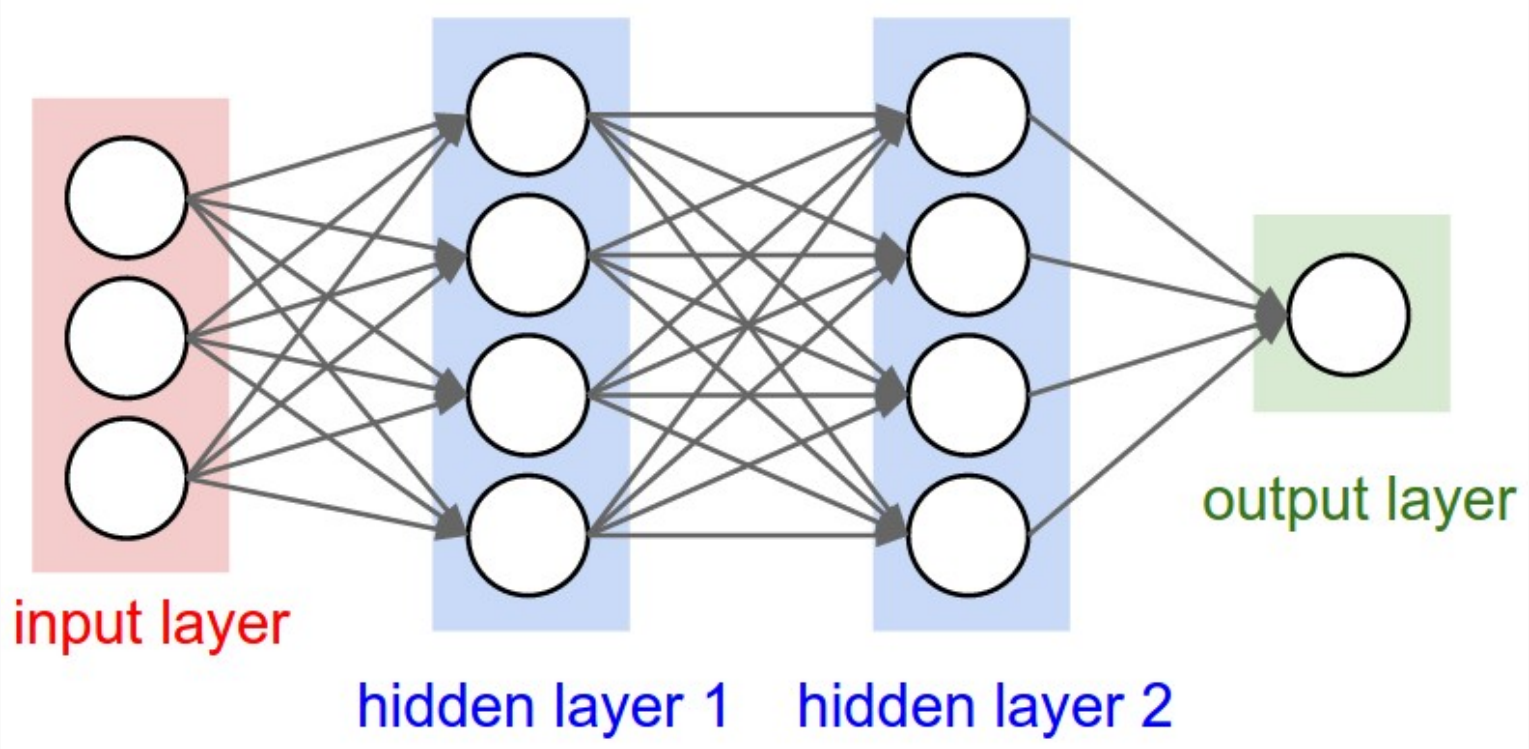
LSTM Demo 1

Momentum

LSTM Demo 2

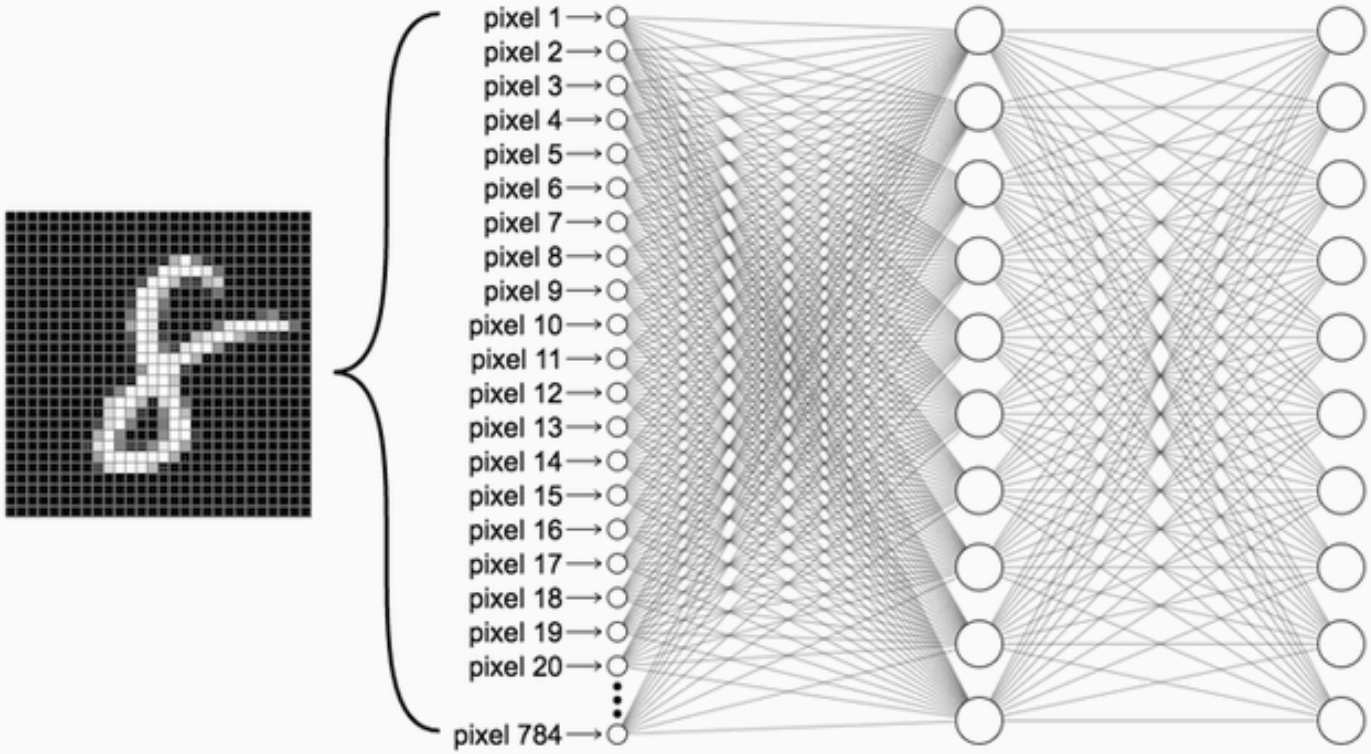
Motivation

Feed Forward Networks



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

Feed Forward Networks



Feed Forward Networks

Feed forward neural networks are *universal function approximators*, meaning that any function from $\mathbb{R}^n$ to $\mathbb{R}^m$ can be represented *arbitrarily well* by some neural network.

A Motivating Example

Say we could predict tomorrow's stock prices given daily price readings...

A Motivating Example

Say we could predict tomorrow's stock prices given daily price readings...

We could answer questions such as:

- What will Apple's stock price be tomorrow?

A Motivating Example

Say we could predict tomorrow's stock prices given daily price readings...

We could answer questions such as:

- What will Apple's stock price be tomorrow?
- What will Apple's stock price be in one month?
- Is now a good time to sell Facebook stocks?
- When will Google's stock peak?

A Motivating Example

Say we could predict tomorrow's stock prices given daily price readings...

We could answer questions such as:

- What will Apple's stock price be tomorrow?
- What will Apple's stock price be in one month?
- Is now a good time to sell Facebook stocks?
- When will Google's stock peak?

Can we design a feed forward network for this task?

What should our network know?

Example Stock Prices Over Time

Apple:

Facebook:

Google:

A Motivating Example

Conclusions:

- Knowing just the price the previous day is not enough.

A Motivating Example

Conclusions:

- Knowing just the price the previous day is not enough.
- We somehow need context regarding the stock.

A Motivating Example

Conclusions:

- Knowing just the price the previous day is not enough.
- We somehow need context regarding the stock.
- Recent price trends are relevant as well.

A Motivating Example

Conclusions:

- Knowing just the price the previous day is not enough.
- We somehow need context regarding the stock.
- Recent price trends are relevant as well.
- We could leverage the fact that we we are given daily readings.

A Motivating Example

Conclusions:

- Knowing just the price the previous day is not enough.
- We somehow need context regarding the stock.
- Recent price trends are relevant as well.
- We could leverage the fact that we we are given daily readings.

Ideally our model can consider not only the previous day's price but information further in the past.

Shortcomings of Feed Forward Neural Networks

Basic feed forward networks generally struggle with time series problems.

Shortcomings of Feed Forward Neural Networks

Basic feed forward networks generally struggle with time series problems.

- No support for variable-length sequences.

Shortcomings of Feed Forward Neural Networks

Basic feed forward networks generally struggle with time series problems.

- No support for variable-length sequences.
- No notion of 'order' of the inputs.

Shortcomings of Feed Forward Neural Networks

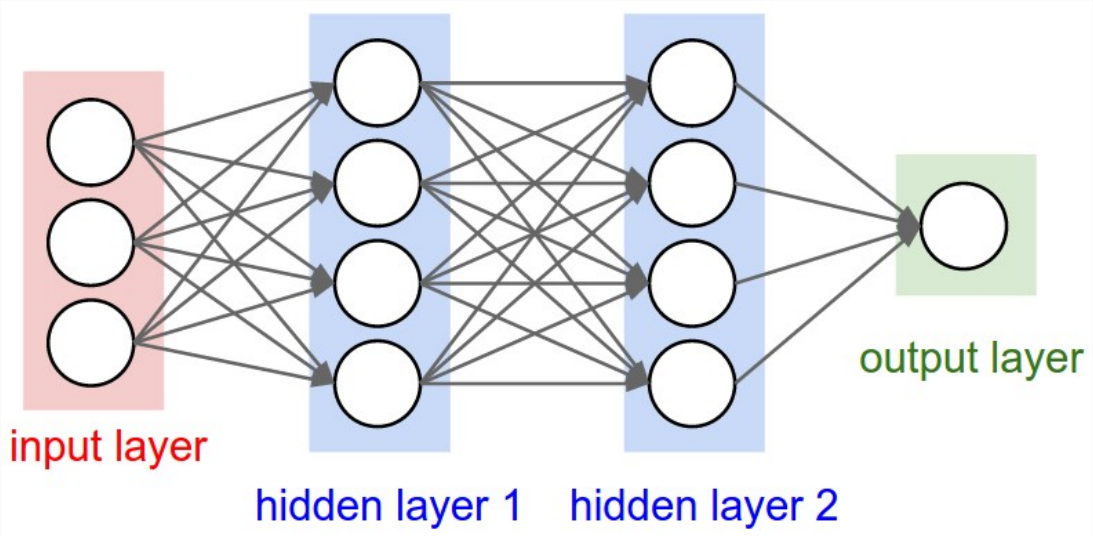
Basic feed forward networks generally struggle with time series problems.

- No support for variable-length sequences.
- No notion of 'order' of the inputs.

Shortcomings of Feed Forward Neural Networks

Basic feed forward networks generally struggle with time series problems.

- No support for variable-length sequences.
- No notion of 'order' of the inputs.



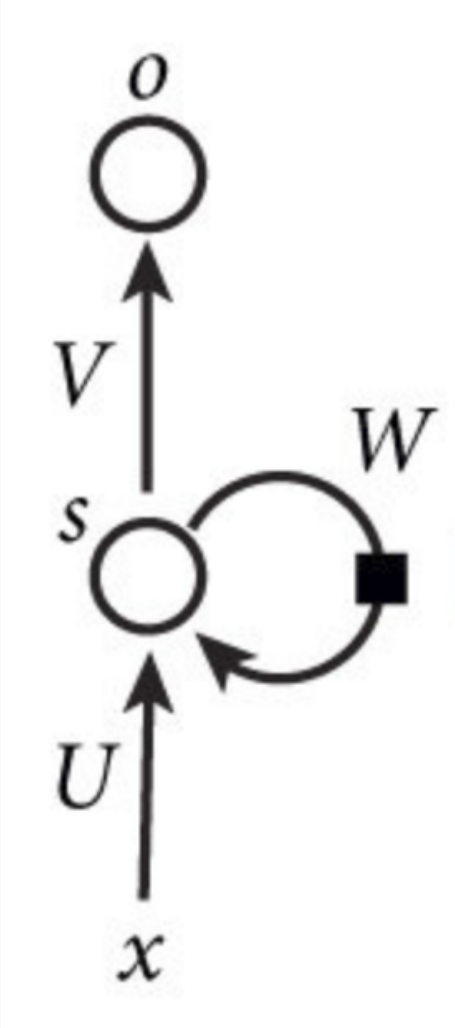
Example: Language

Example: Videos

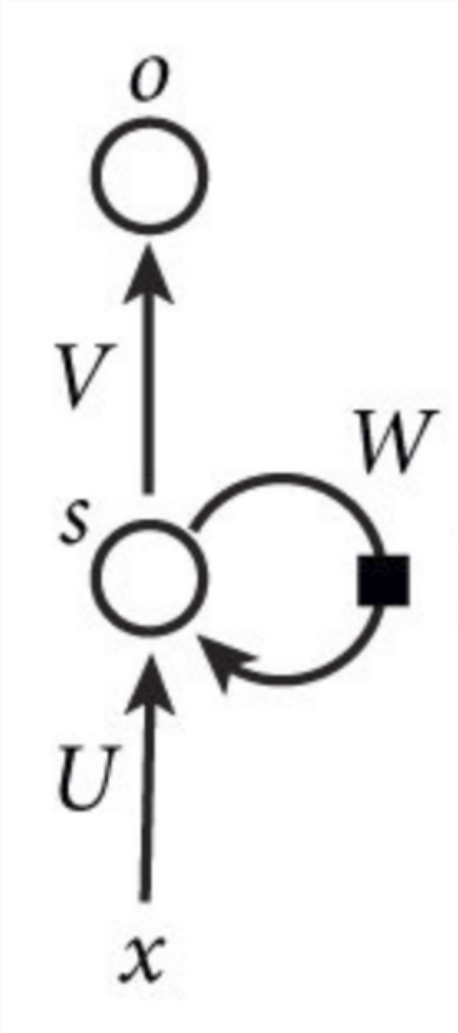
https://www.youtube.com/watch?v=pW6nZXeWIGM

Recurrent Neural Networks (RNNs)

Internal States



Update Equations



$$h^{(t)} = \tanh\left(b + Wh^{(t-1)} + Ux^{(t)}\right)$$

$$o^{(t)} = c + Vh^{(t)}$$

Backprop?

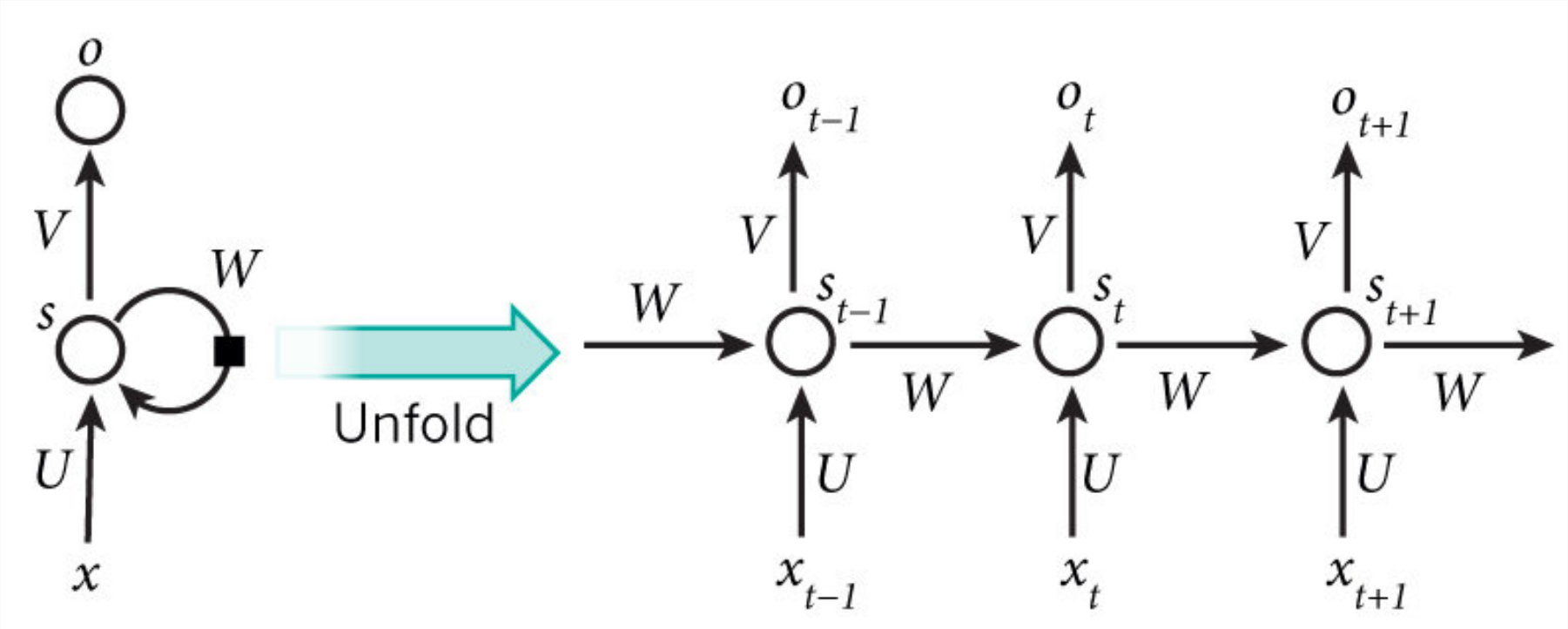
Remember that backpropagation requires no cycles in the computation graph.

Backprop?

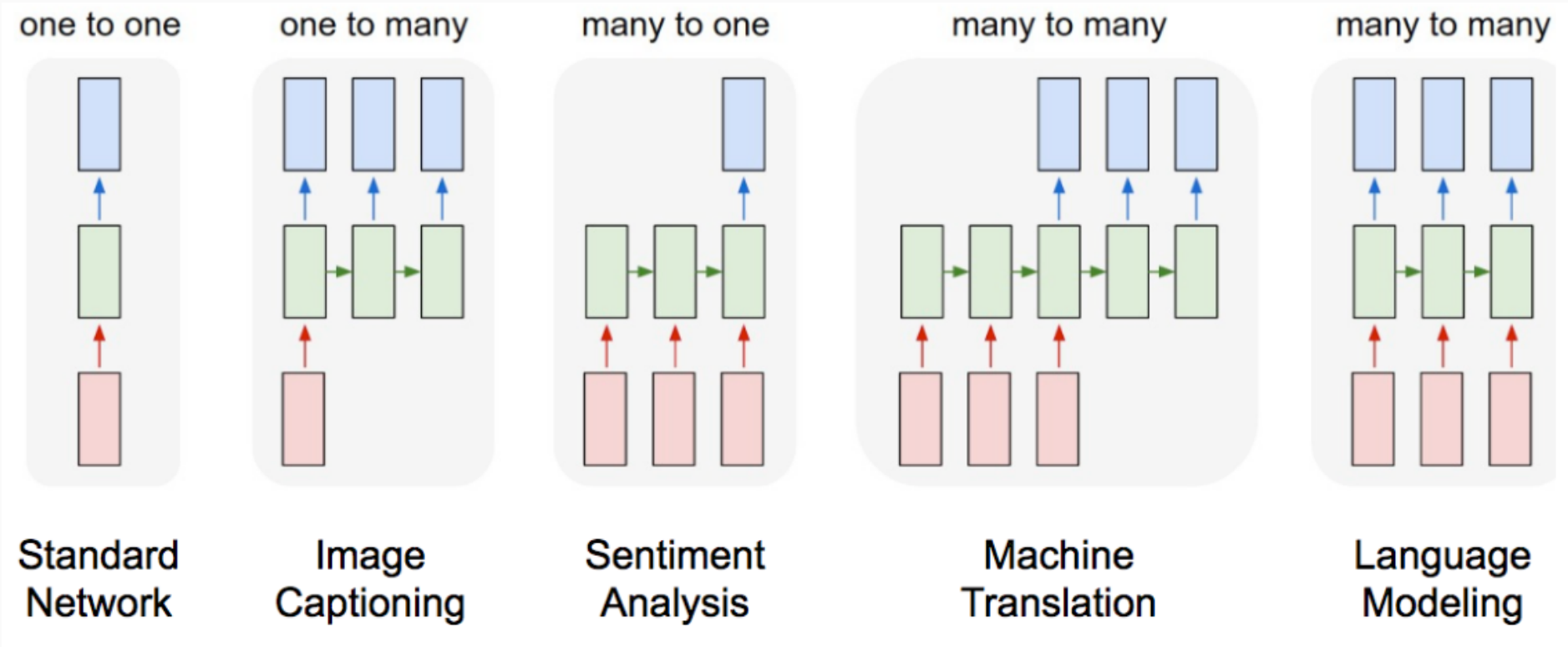
Remember that backpropagation requires no cycles in the computation graph.

Does backprop work for networks with recurrence?

Unrolling the Graph



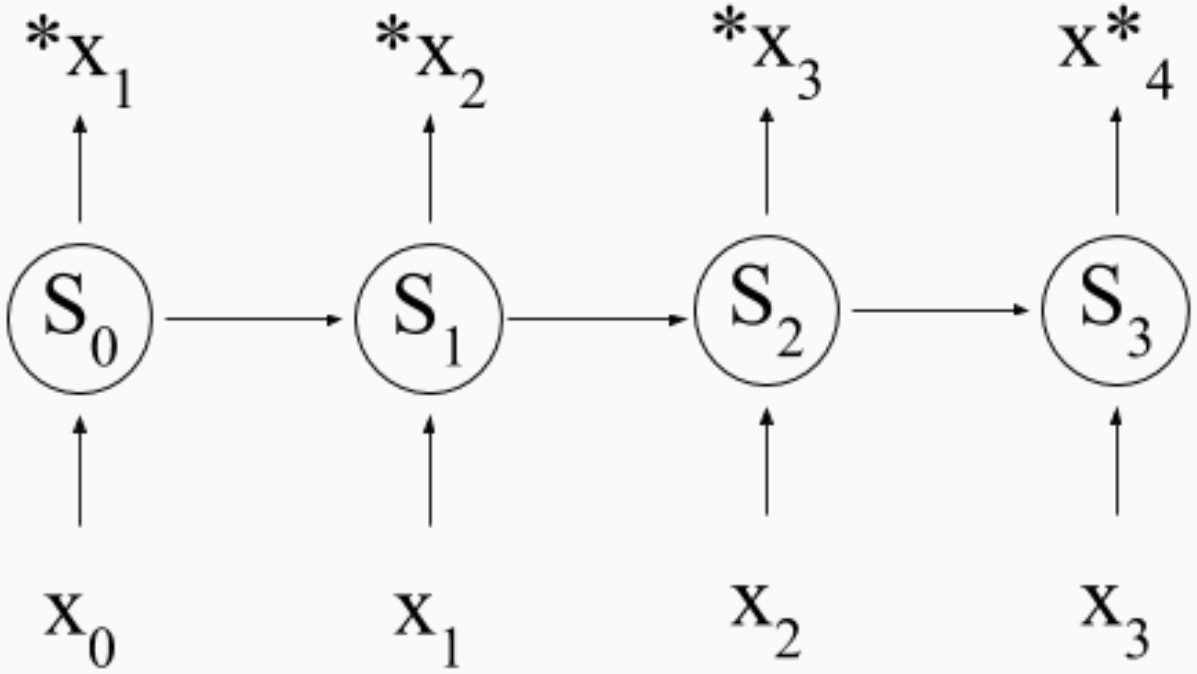
Uses


one to one
one to many
many to one
many to many
many to many
Standard Network
Image Captioning
Sentiment Analysis
Machine Translation
Language Modeling

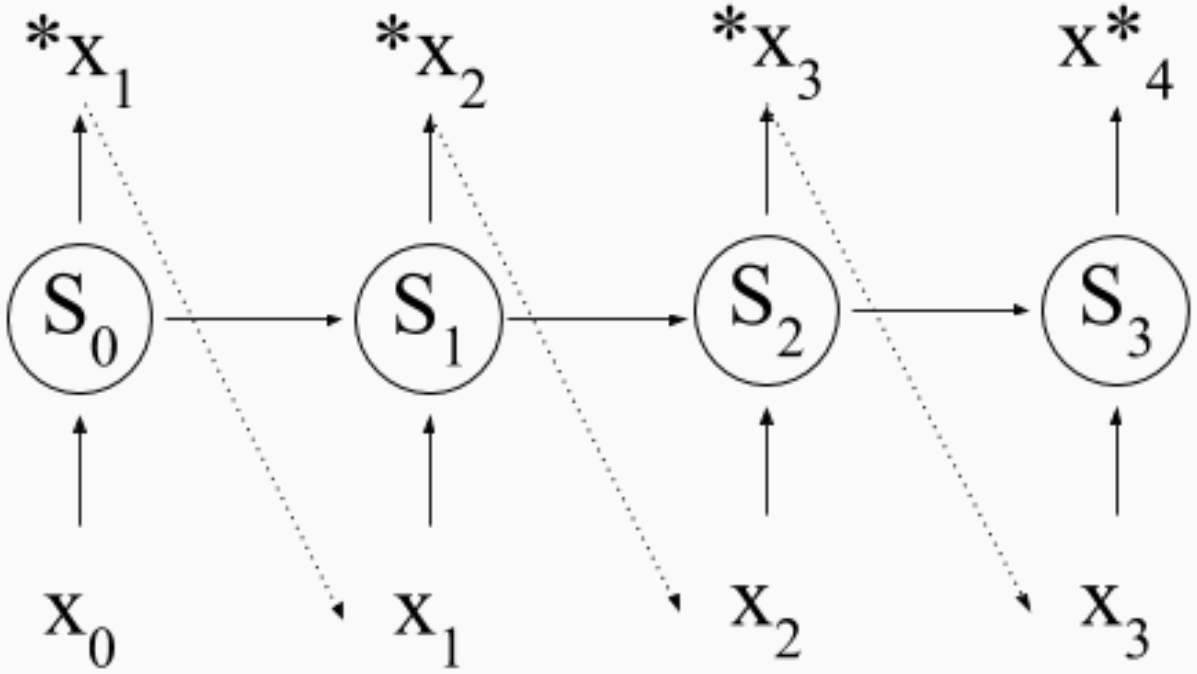
Sequence Modeling

Sequence modeling is the task of predicting the "next item" given current and previous tokens.
(e.g., Bitcoin price prediction, language modeling).

Sequence Modeling

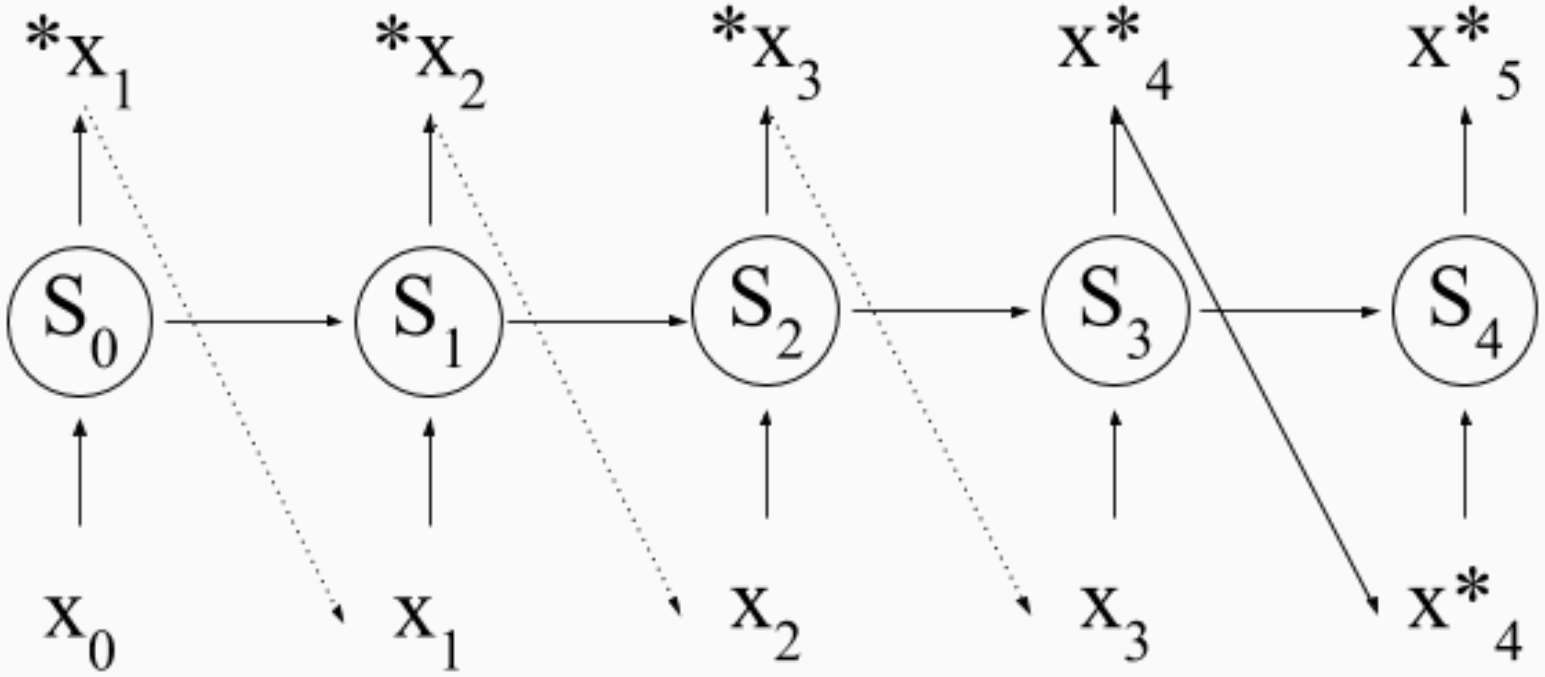


Teacher Forcing



Do we use the previously predicted value or the ground truth value as input?

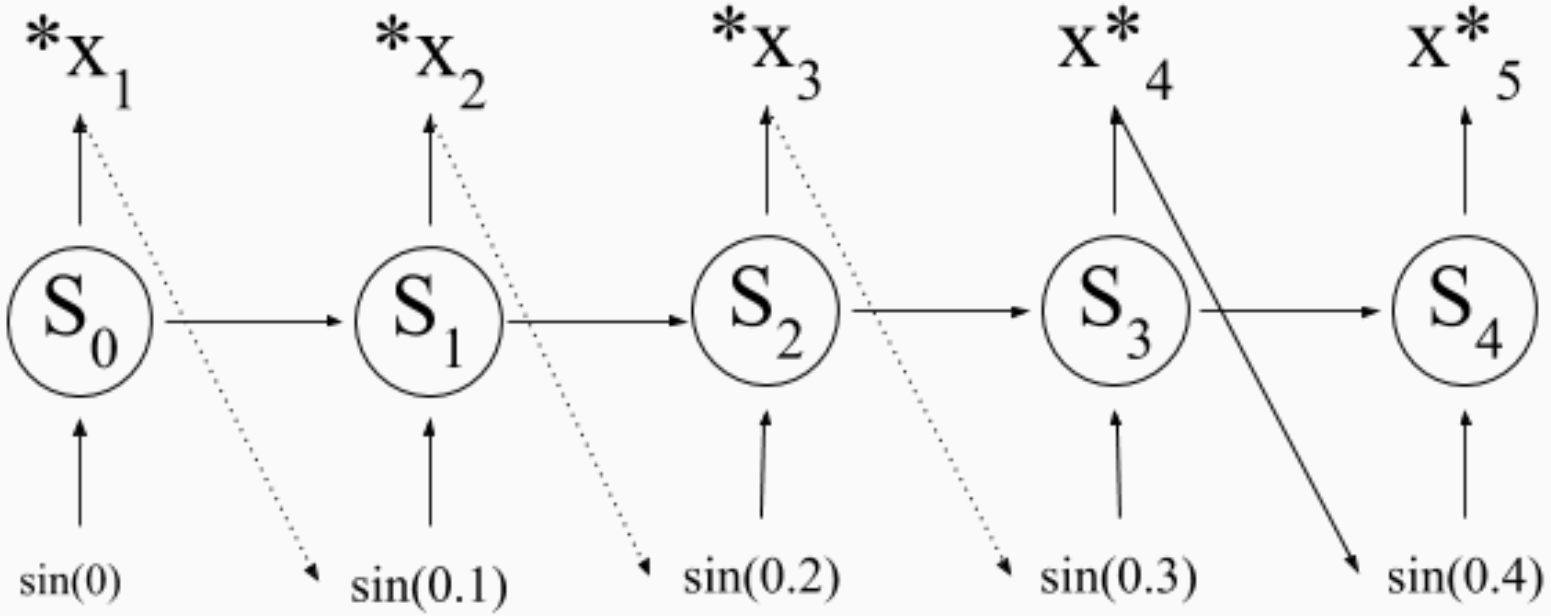
Extrapolation!



Continuously feed the output of the model back in to generate future points.

RNN Demo

Modeling a Sine Wave

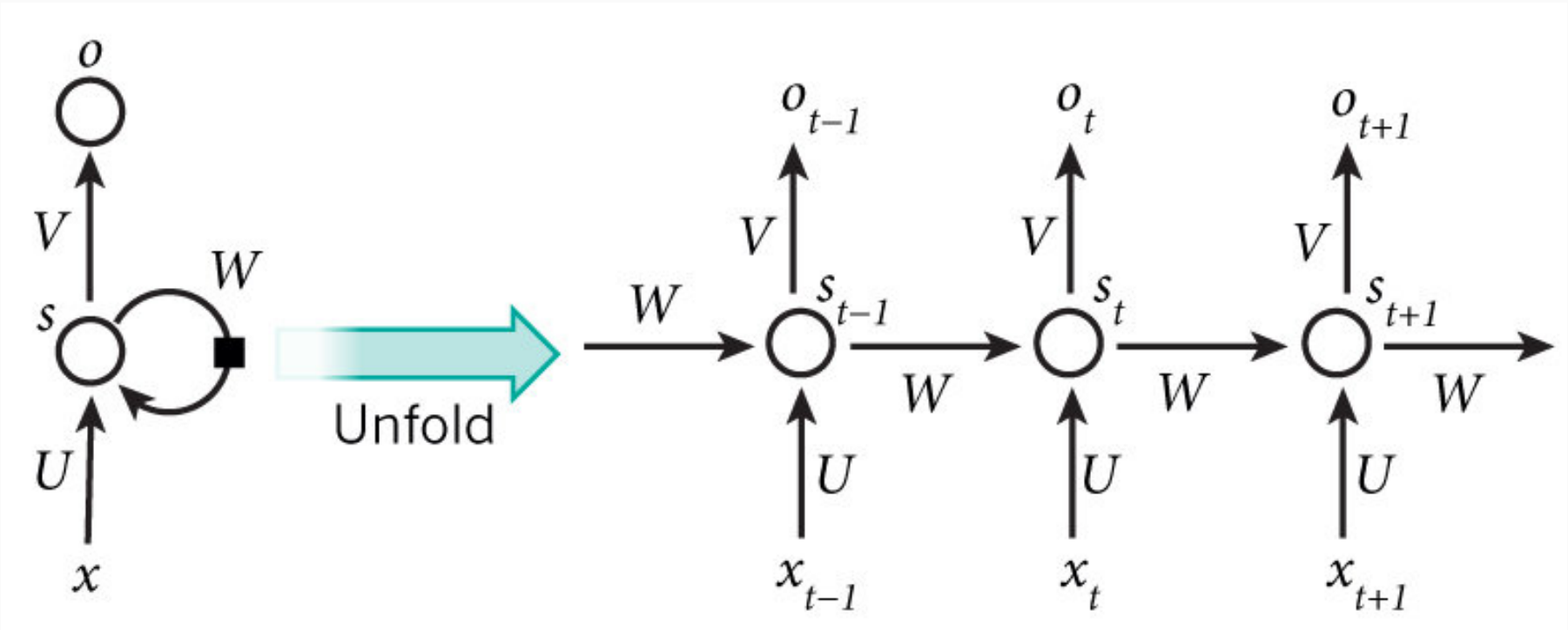


Input and Output of Model

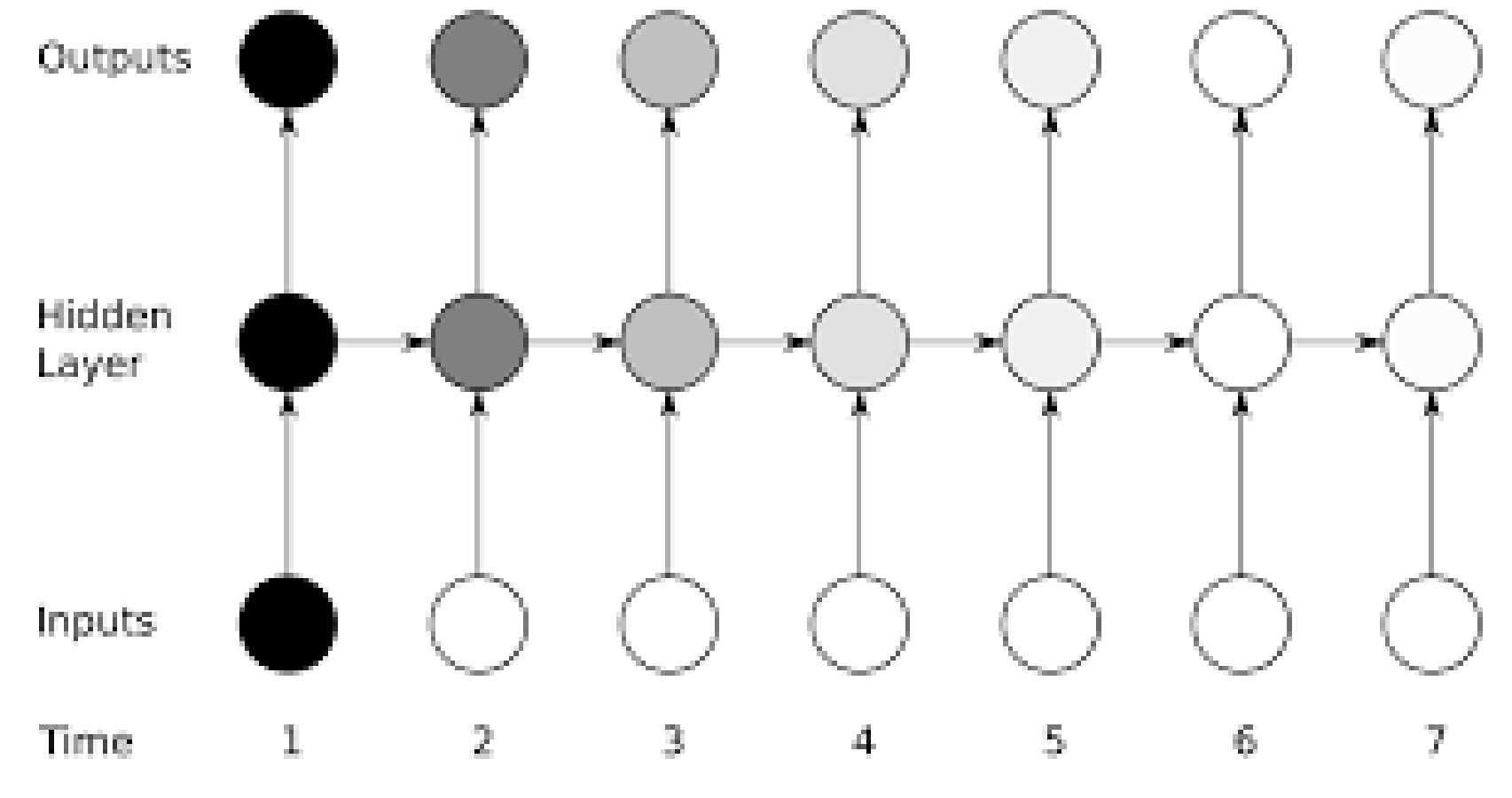
Input	y_0	y_1	...	y_{n-1}
Output	$*y_1$	$*y_2$	...	$*y_n$
Target	y_1	y_2	...	y_n

Long Short-Term Memory Networks (LSTMs)

Vanishing Gradient Problem

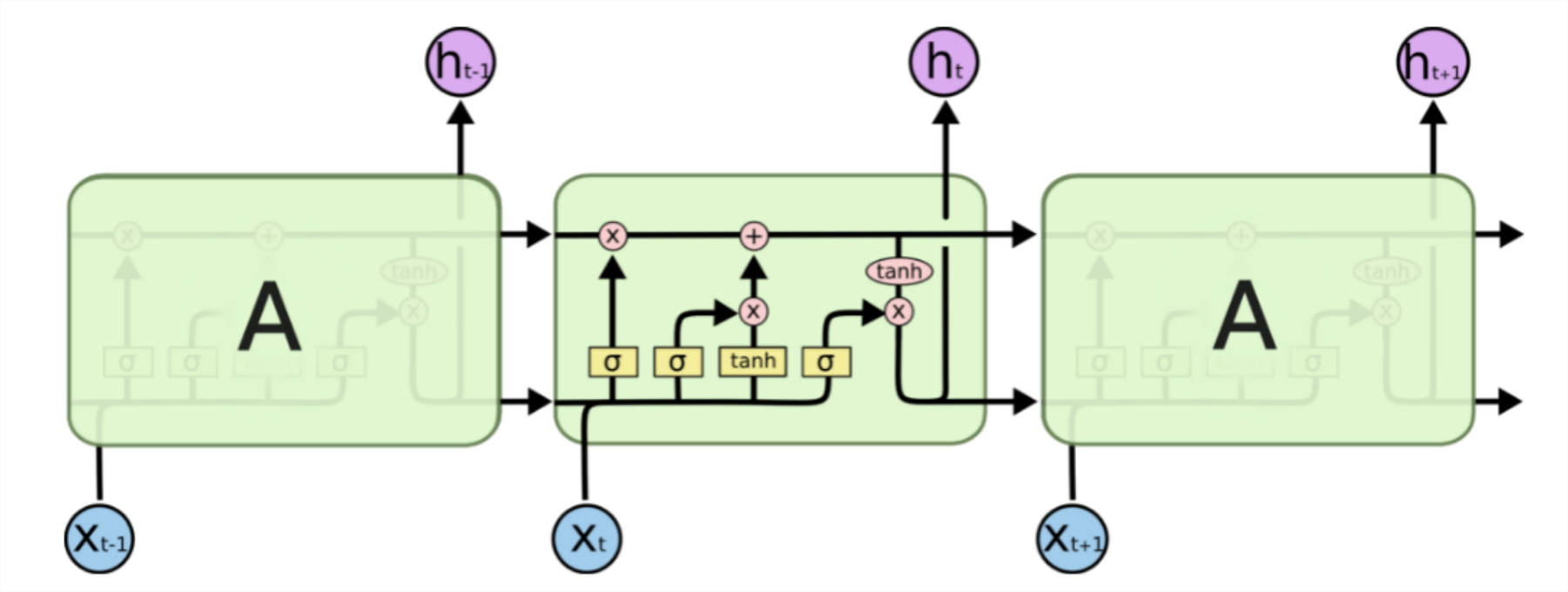


Vanishing Gradient Problem



As we change the input weight matrix U, signal from earlier timesteps exponentially decays through time.

LSTMs: A Solution to Vanishing Gradient

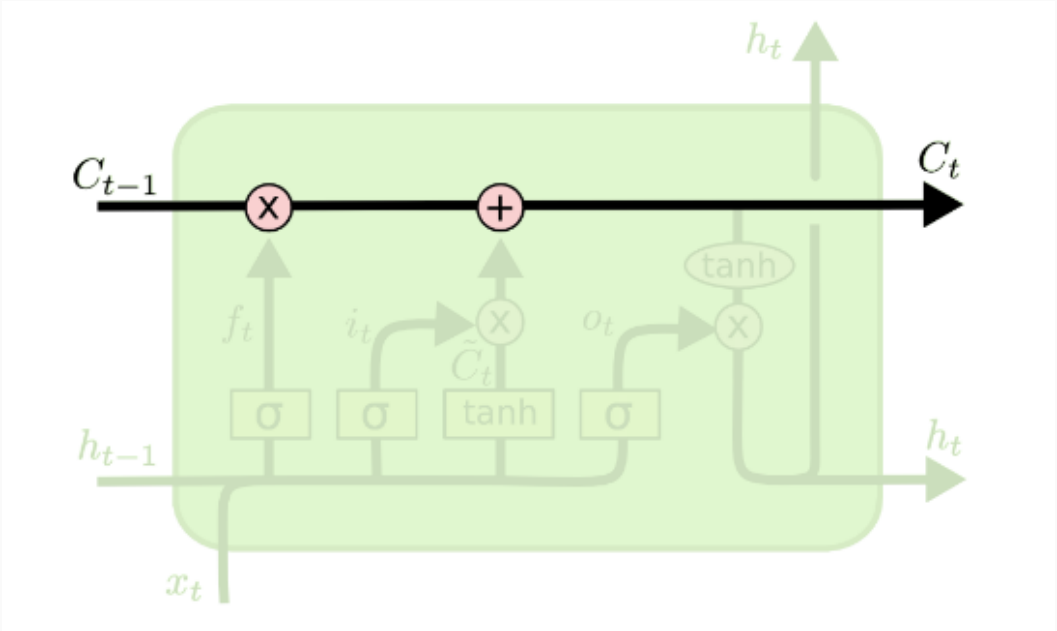


Fundamental Idea

Why does the LSTM avoid the vanishing gradient issue?

Fundamental Idea

Why does the LSTM avoid the vanishing gradient issue?



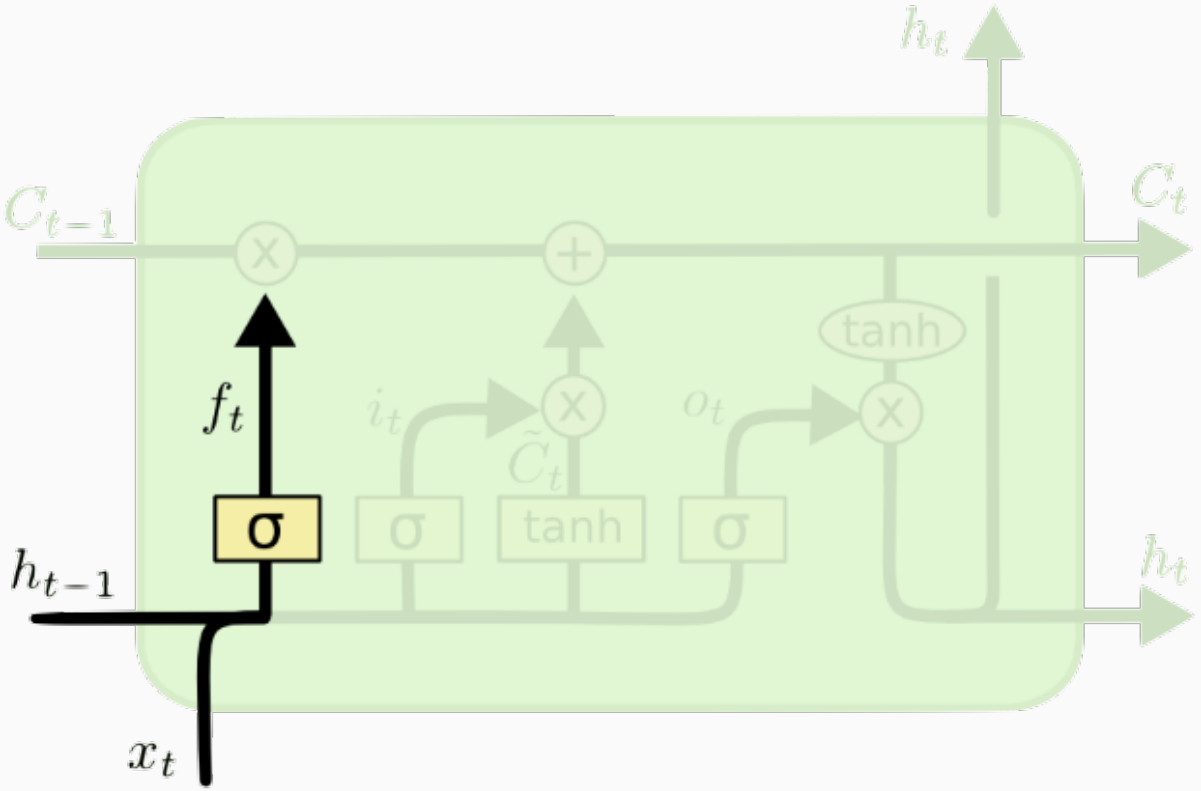
The cell state is a single, continuous 'information highway' across the entire calculation that does not have any matrix multiplications!

LSTM Parts

LSTMs contain five parts:

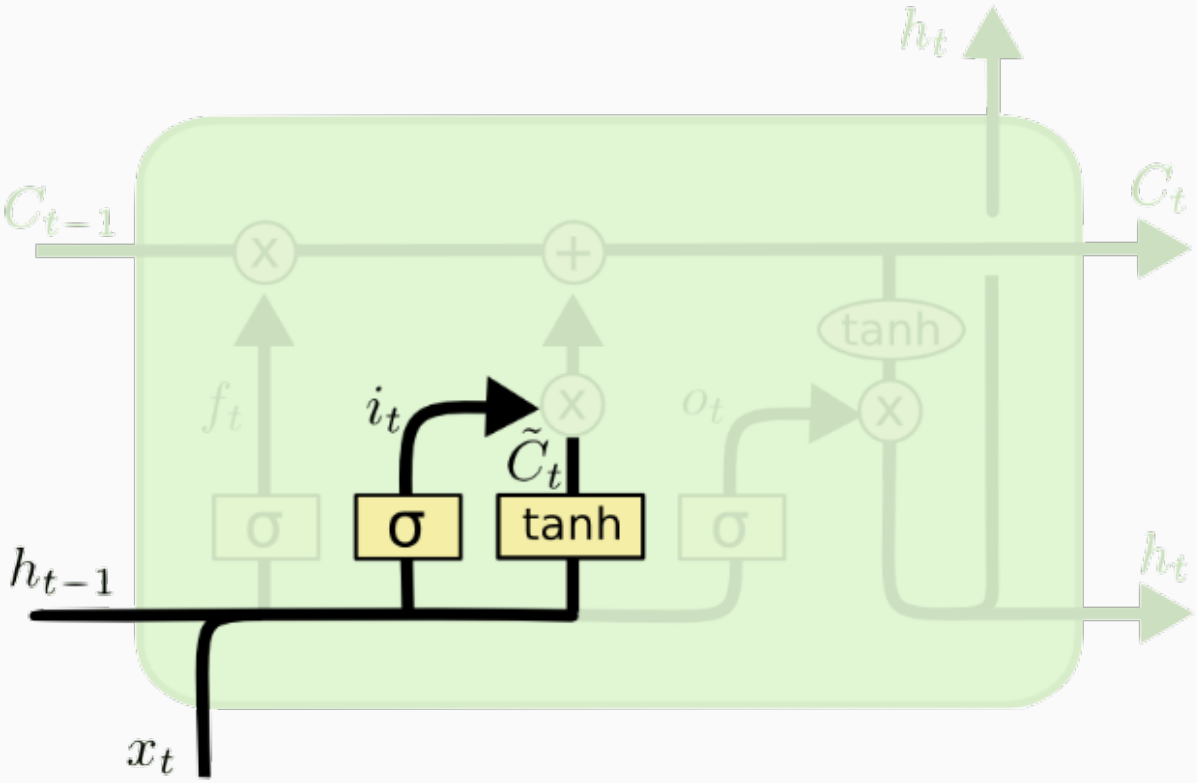
- A cell state.
- A forget gate.
- An input gate.
- An update gate.
- An output gate.

The Forget Gate Layer



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

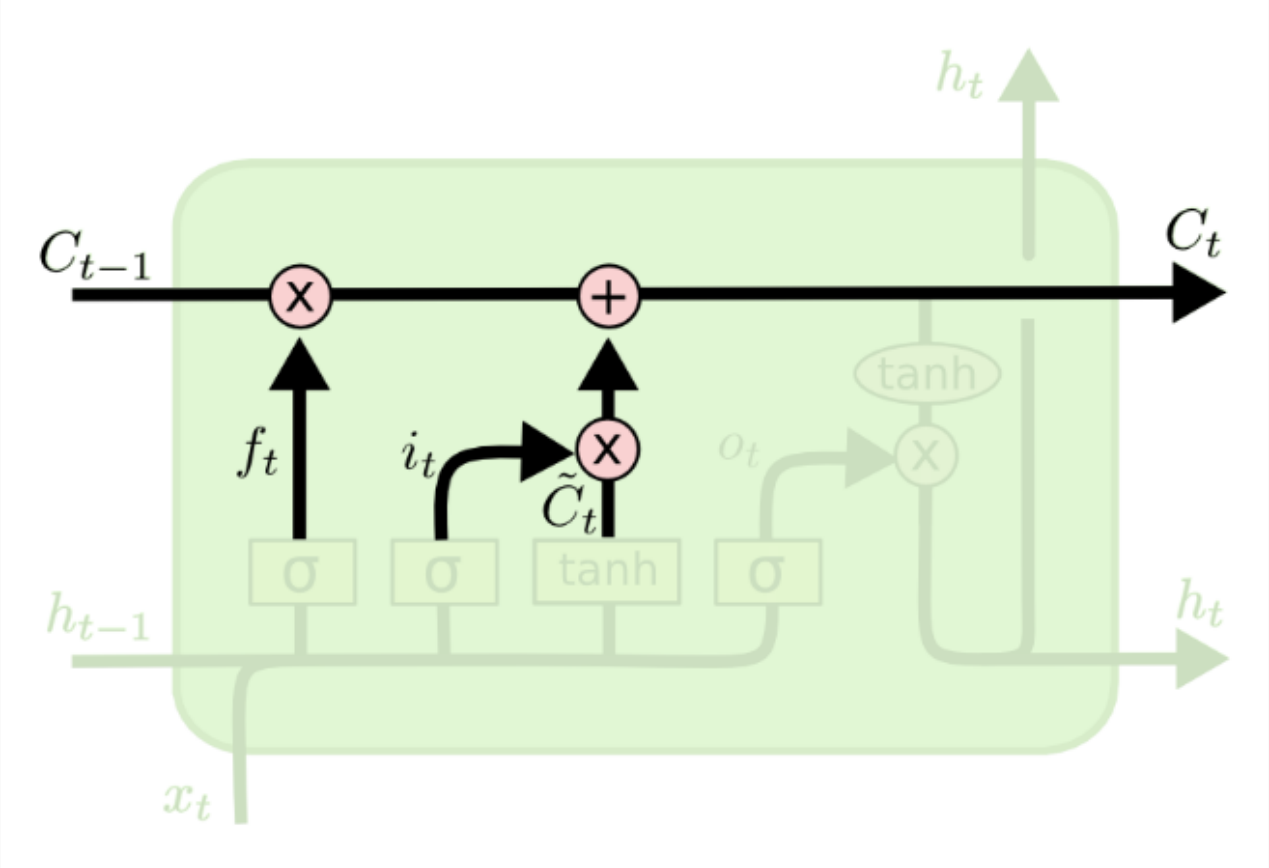
The Input Gate Layer



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

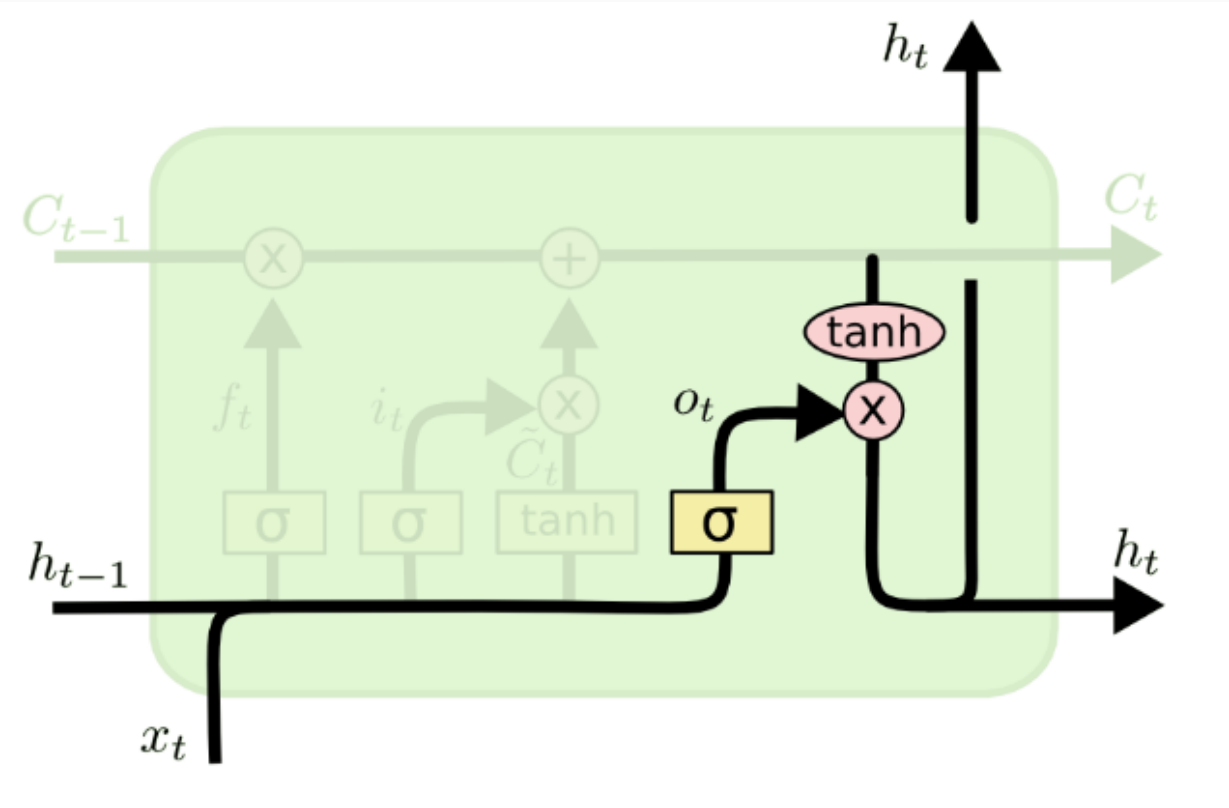
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

The Update Gate Layer



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

The Output Gate Layer



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

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

LSTM Recap

LSTM Recap

- Determine which parts of the cell state should be forgotten ('forget mask').

LSTM Recap

- Determine which parts of the cell state should be forgotten ('forget mask').
- Determine how ('cell state candidate') and to what extent ('update mask') you want to add to the cell state.

LSTM Recap

- Determine which parts of the cell state should be forgotten ('forget mask').
- Determine how ('cell state candidate') and to what extent ('update mask') you want to add to the cell state.
- Apply the 'forget mask' to the cell state. Then add the 'cell state candidate' multiplied by the 'update mask'.

LSTM Recap

- Determine which parts of the cell state should be forgotten ('forget mask').
- Determine how ('cell state candidate') and to what extent ('update mask') you want to add to the cell state.
- Apply the 'forget mask' to the cell state. Then add the 'cell state candidate' multiplied by the 'update mask'.
- Determine which parts of the cell state you want to be present in the hidden state ('hidden mask').

LSTM Recap

- Determine which parts of the cell state should be forgotten ('forget mask').
- Determine how ('cell state candidate') and to what extent ('update mask') you want to add to the cell state.
- Apply the 'forget mask' to the cell state. Then add the 'cell state candidate' multiplied by the 'update mask'.
- Determine which parts of the cell state you want to be present in the hidden state ('hidden mask').
- Set the hidden state to the tanh activated cell state multiplied by the 'hidden mask'. Output the hidden state.

LSTM Demo 1

Revisiting the Sine Wave

When we used an RNN we fixed the starting position for all sine waves to $x = 0$.

Revisiting the Sine Wave

When we used an RNN we fixed the starting position for all sine waves to $x = 0$.

Will the same model work if we randomize the starting position for our sine wave?

Revisiting the Sine Wave

When we used an RNN we fixed the starting position for all sine waves to $x = 0$.

Will the same model work if we randomize the starting position for our sine wave?

It turns out fixing the initial position allowed us to optimize the initial hidden state in a way that internalized this consistency.

Revisiting the Sine Wave

When we used an RNN we fixed the starting position for all sine waves to $x = 0$.

Will the same model work if we randomize the starting position for our sine wave?

It turns out fixing the initial position allowed us to optimize the initial hidden state in a way that internalized this consistency.

What do we do now?

Revisiting the Sine Wave

We know that an RNN can accurately model a sine wave if its initial hidden state reflects the *phase* of the sine wave.

Revisiting the Sine Wave

We know that an RNN can accurately model a sine wave if its initial hidden state reflects the *phase* of the sine wave.

Perhaps if we gave our model the capability of learning this phase from part of the wave, we could then use a similar RNN to model the rest of the sine wave!

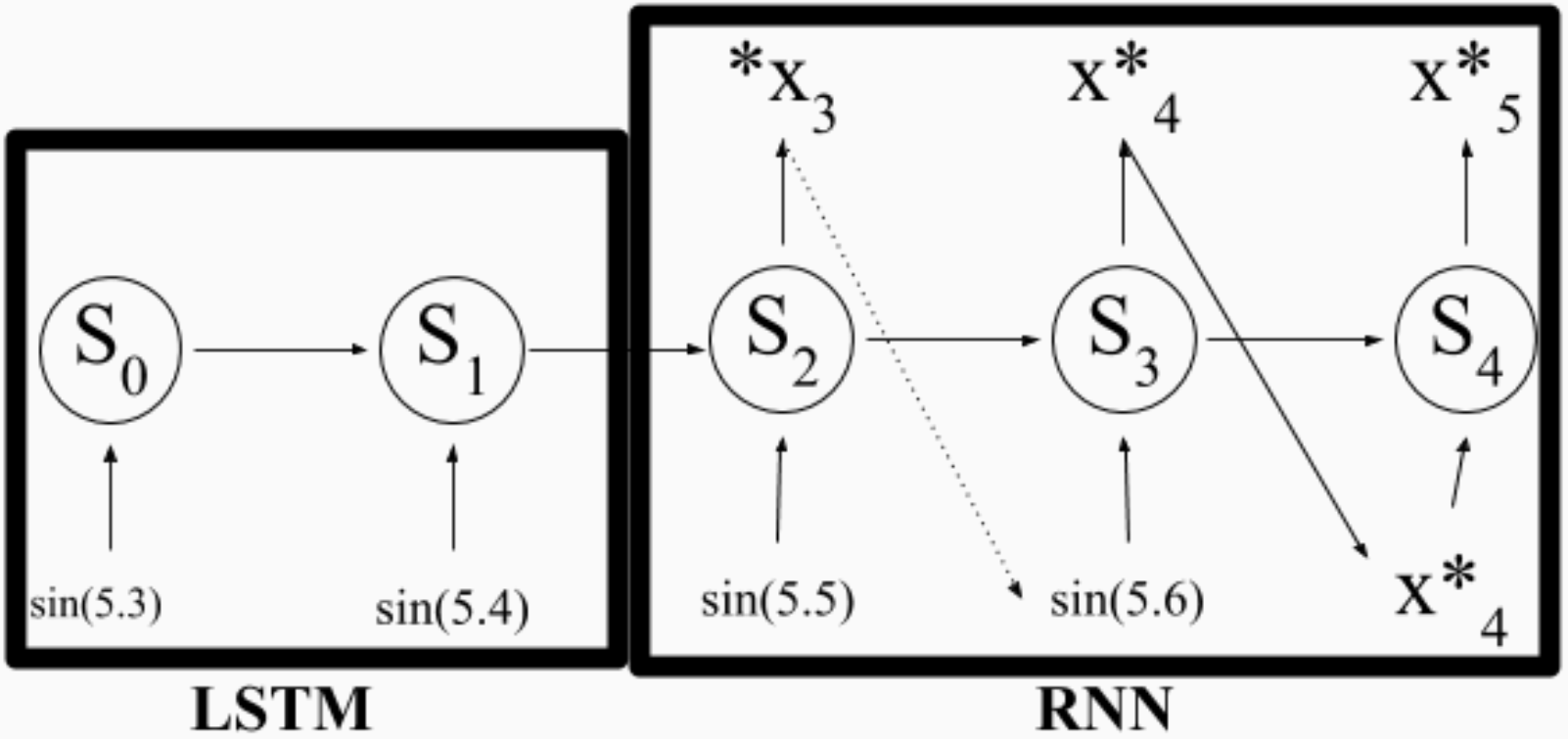
Revisiting the Sine Wave

We know that an RNN can accurately model a sine wave if its initial hidden state reflects the *phase* of the sine wave.

Perhaps if we gave our model the capability of learning this phase from part of the wave, we could then use a similar RNN to model the rest of the sine wave!

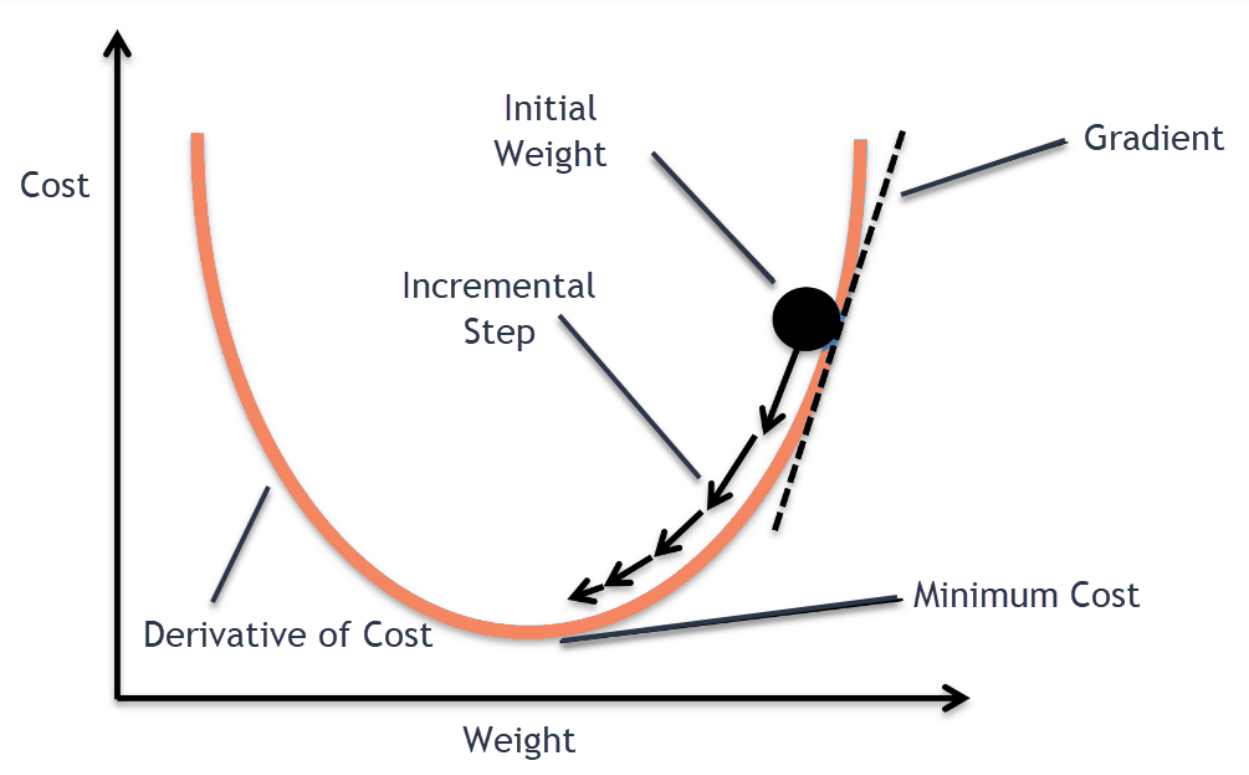
Idea: Pass the first half of the wave through an LSTM to encode the phase, or whatever information it thinks might be relevant. Then use the earlier process on the rest of the wave with a RNN.

LSTM Encoder + RNN

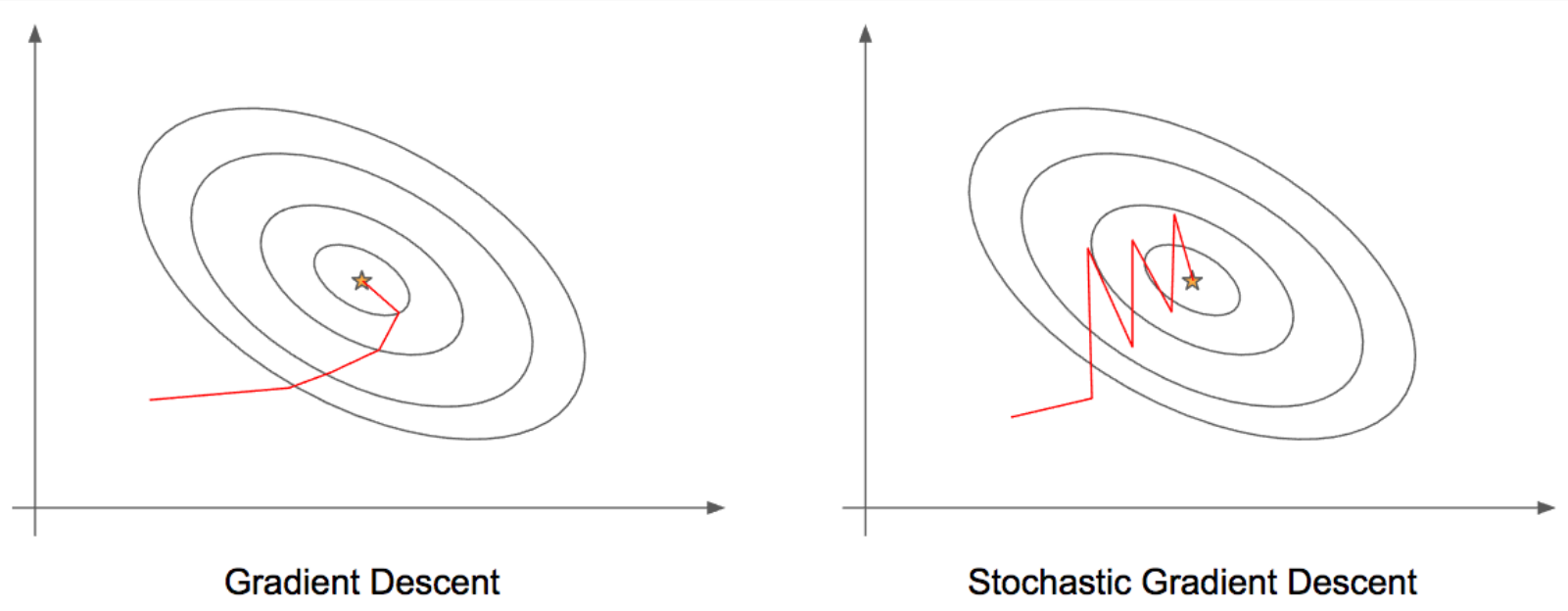


Momentum

Ideal Gradient Descent



Stochastic Gradient Descent



Enhancing Gradient Descent

- LSTMs can be difficult to train because the weights must be applicable to any input at any possible time step.

Enhancing Gradient Descent

- LSTMs can be difficult to train because the weights must be applicable to any input at any possible time step.
- Basic gradient descent only considers the current location, and is thus sensitive to specific data points.

Enhancing Gradient Descent

- LSTMs can be difficult to train because the weights must be applicable to any input at any possible time step.
- Basic gradient descent only considers the current location, and is thus sensitive to specific data points.
- A good model should aggregate feedback as opposed to reacting to all feedback independently.

An Idea

Every time we update our weights, we believe we are moving in the optimal direction.

An Idea

Every time we update our weights, we believe we are moving in the optimal direction.

In typical mini batch learning, this direction entirely changes as each batch.

An Idea

Every time we update our weights, we believe we are moving in the optimal direction.

In typical mini batch learning, this direction entirely changes as each batch.

We still want to update weights frequently, ruling out massive batch sizes.

An Idea

Every time we update our weights, we believe we are moving in the optimal direction.

In typical mini batch learning, this direction entirely changes as each batch.

We still want to update weights frequently, ruling out massive batch sizes.

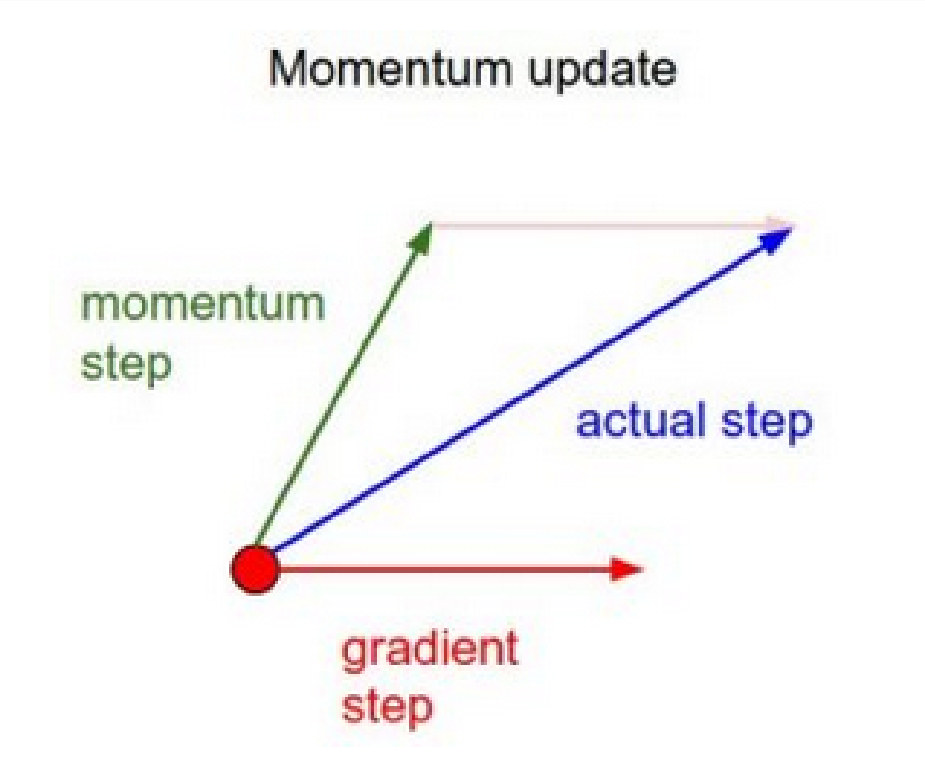
What if instead of exclusively considering the previous batch we consider previous data as well?

Momentum

Use batch gradient to update *how we update* weights.

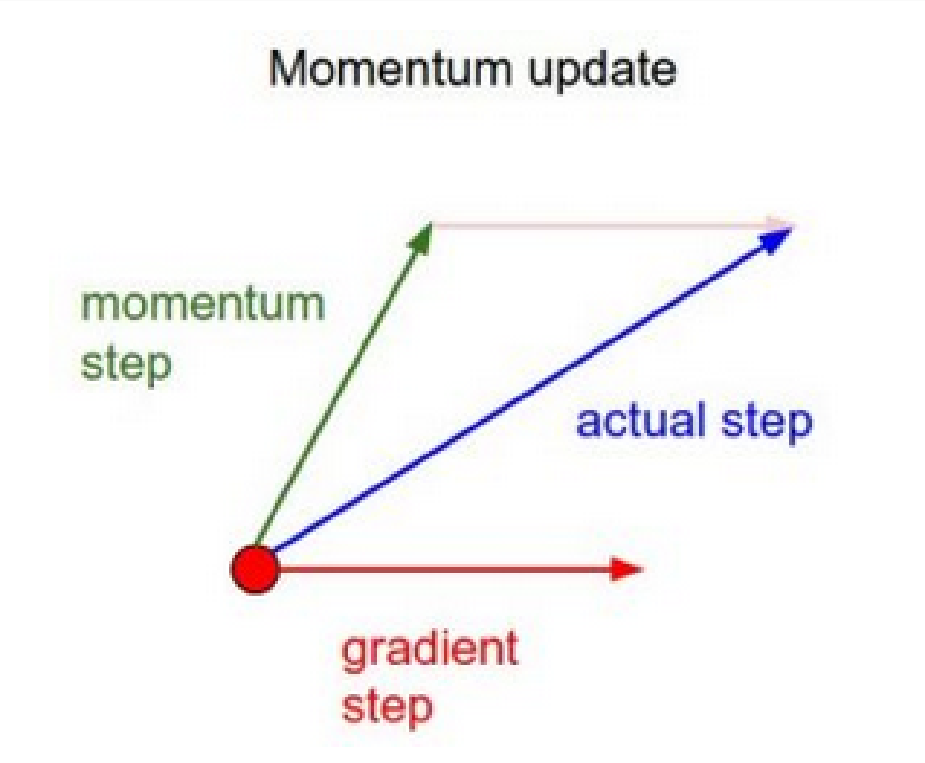
Momentum

Use batch gradient to update *how we update* weights.



Momentum

Use batch gradient to update *how we update* weights.



$$z^{(t)} = \beta z^{(t-1)} + (1-\beta)\nabla_\theta C(\theta; X, y)$$

$$w^{(t)} = w^{(t-1)} - \alpha z^{(t)}$$

Key Benefits

- Allows for aggregation without having to use large batch sizes.

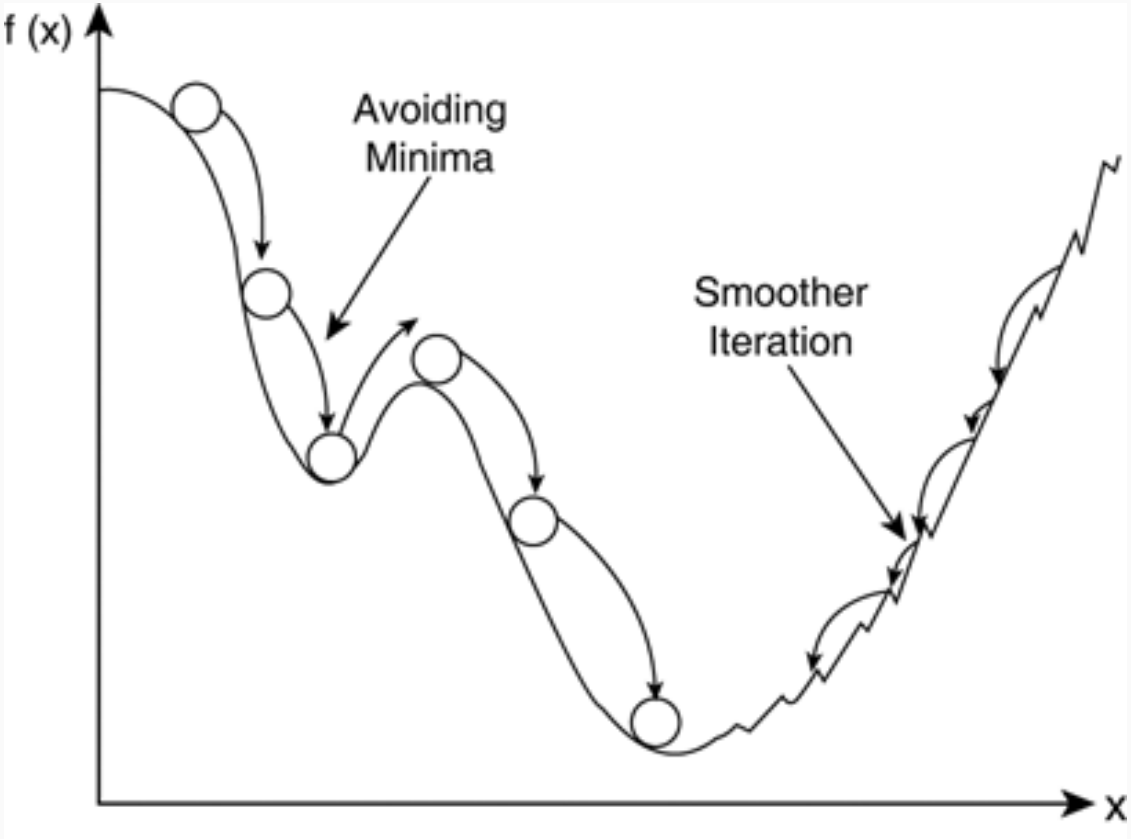
Key Benefits

- Allows for aggregation without having to use large batch sizes.
- Smoother iteration to reduce sensitivity to variable loss surfaces.

Key Benefits

- Allows for aggregation without having to use large batch sizes.
- Smoother iteration to reduce sensitivity to variable loss surfaces.
- Escape poor local minimums.

Optimization Methods



Other Enhanced Gradient Descent Algorithms

Most all neural networks benefit from some form of enhanced Gradient Descent, not only LSTM's.

Other Enhanced Gradient Descent Algorithms

Most all neural networks benefit from some form of enhanced Gradient Descent, not only LSTM's.

Some popular optimization algorithms used:

- Adam
- RMSProp
- Adagrad
- Adadelta
- Nesterov Momentum

All are fundamentally based in Gradient Descent!

LSTM Demo 2

Text Generation

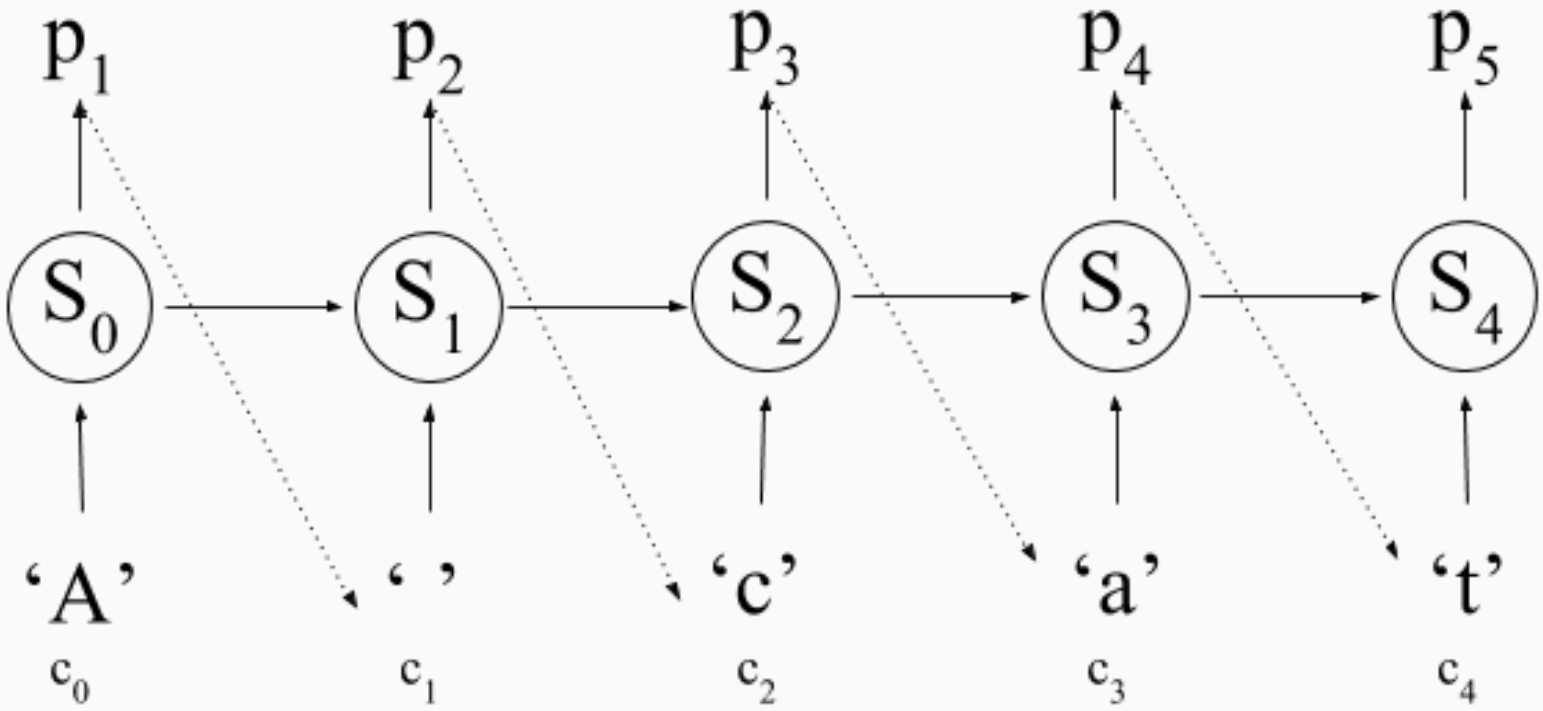
Now we are going to try to generate novel text using an LSTM.

Text Generation

Now we are going to try to generate novel text using an LSTM.

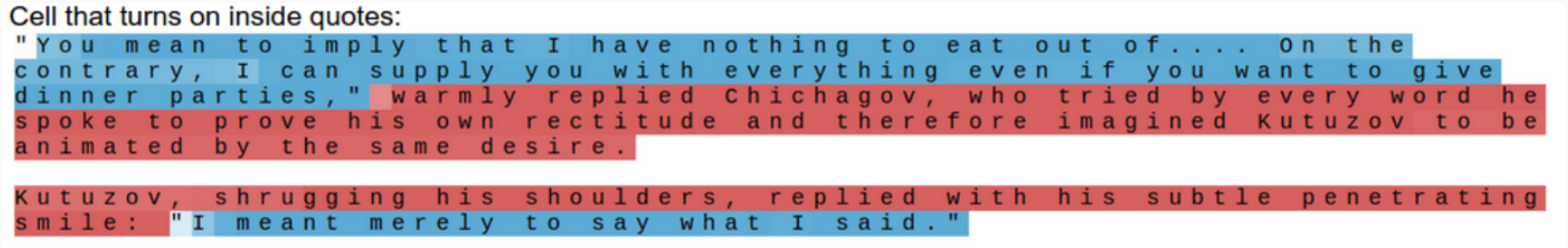
The idea is that we let the model 'read' a lot of existing english text, so that it can learn how to model the language and reproduce similar samples.

Model Architecture



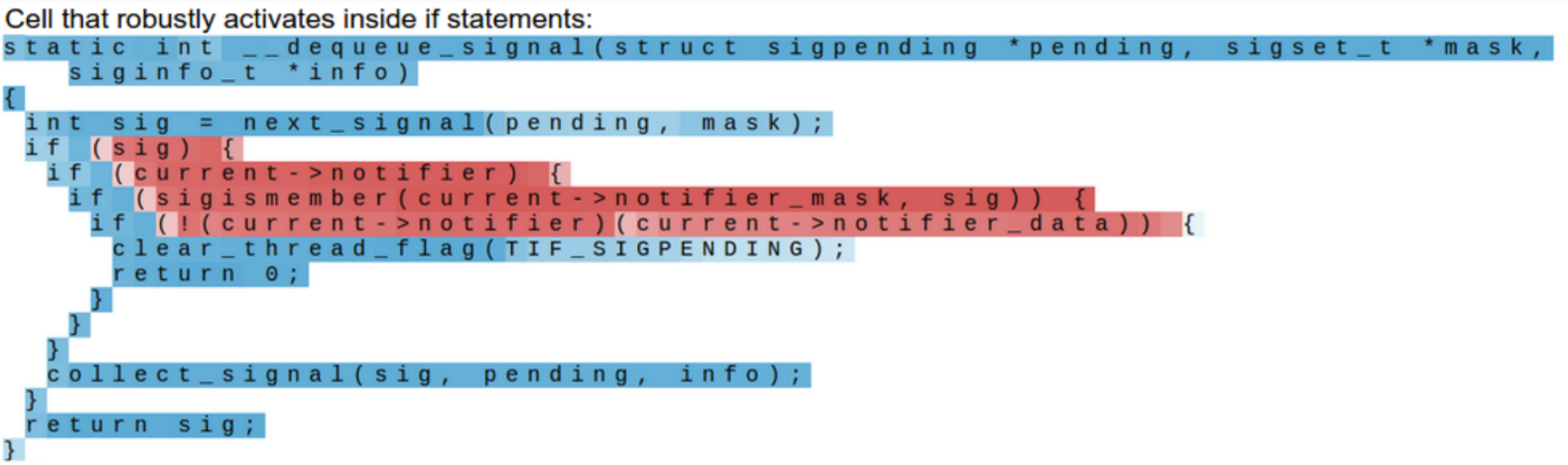
Note: p_i represents a probability distribution spanning all possible characters for character i.

Analyzing LSTMs

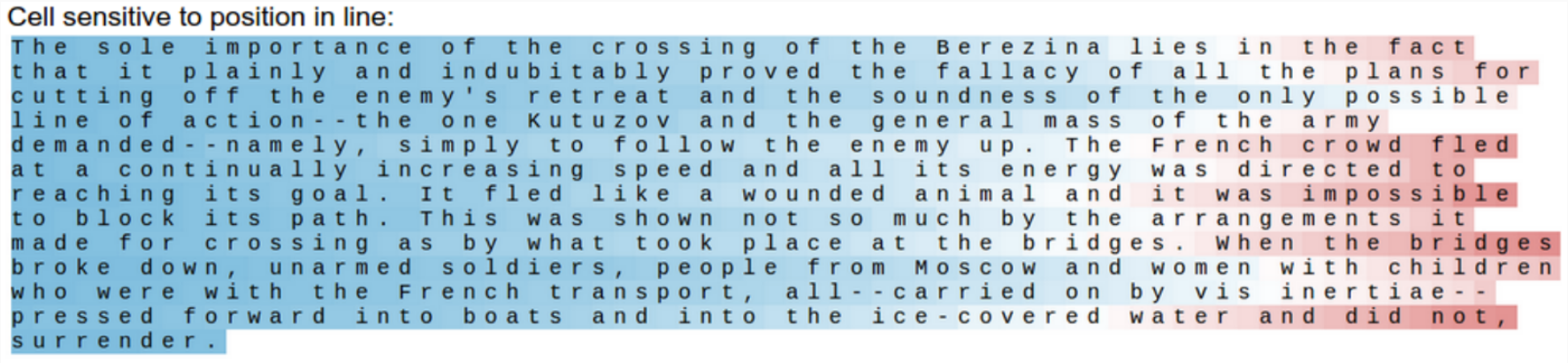


LSTMs keep track of interesting information!

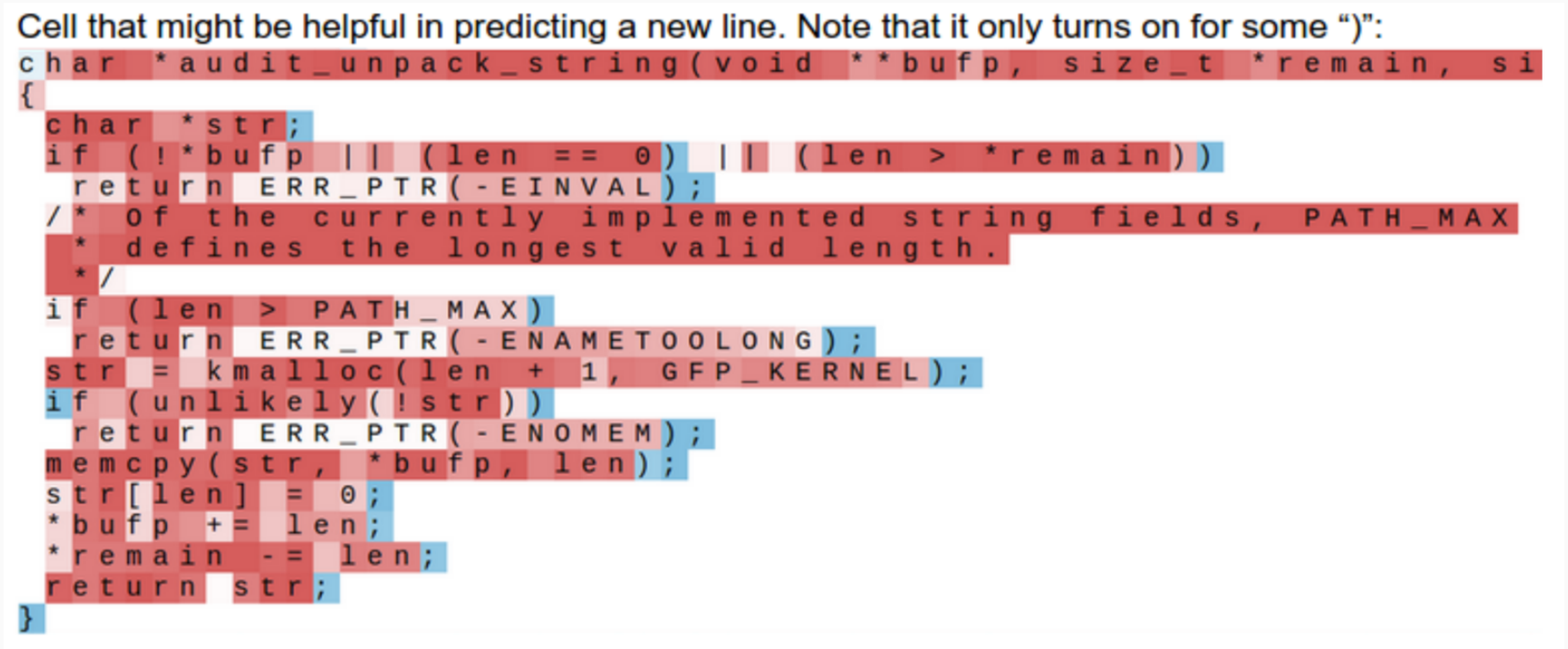
Analyzing LSTMs



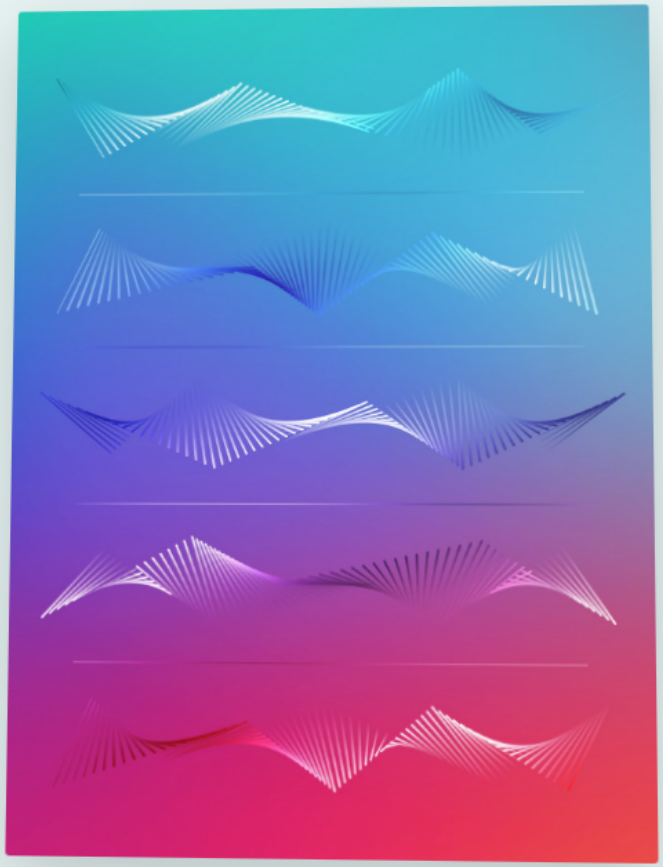
Analyzing LSTMs



Analyzing LSTMs



Analyzing LSTMs



APRIL 6, 2017

Unsupervised Sentiment Neuron

We've developed an unsupervised system which learns an excellent representation of sentiment, despite being trained only to predict the next character in the text of Amazon reviews.

A linear model using this representation achieves state-of-the-art sentiment analysis accuracy on a small but extensively-studied dataset, the Stanford Sentiment Treebank (we get 91.8% accuracy versus the previous best of 90.2%), and can match the performance of previous supervised systems using 30-100x fewer labeled examples. Our representation also contains a distinct "sentiment neuron" which contains almost all of the sentiment signal.

VIEW ON GITHUB

VIEW ON ARXIV

READ MORE