Recurrent Neural Networks (RNNs)

Rosie Martinez, Frank Burkholder

Objectives

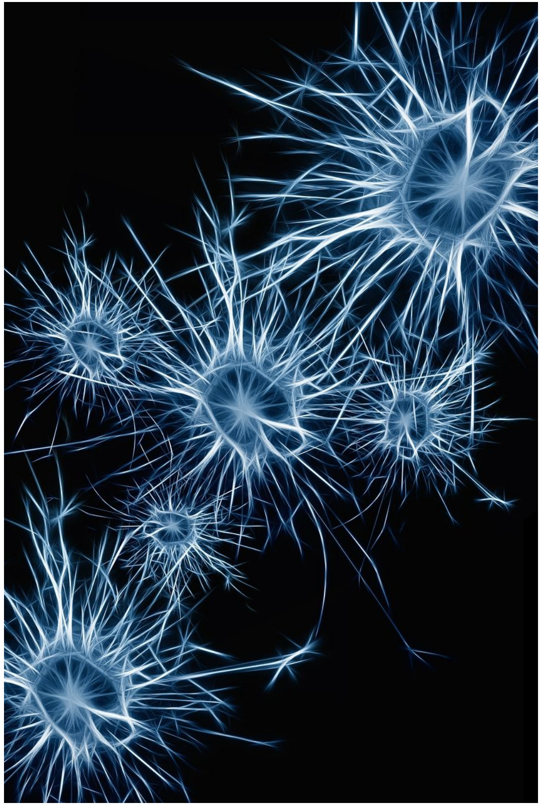
- Introduce RNNs
    - Why are RNNs good for sequential analysis?
- Time Series (common application for RNNs)
- State important differences between MLPs and RNNs
- State a common shortcoming of “vanilla” RNNs
- Explain, at a high-level, what an LSTM is (assignment)
- Predict stock price using an LSTM (assignment)

Recurrent Neural Networks

Models based on the connection of simple computational units, loosely analogous to neurons in the human brain.

What distinguishes RNNs is that connections between neurons can form a directed cycle. This gives an RNN the ability to maintain a state based on previous inputs. So it can model temporal, sequential behavior.

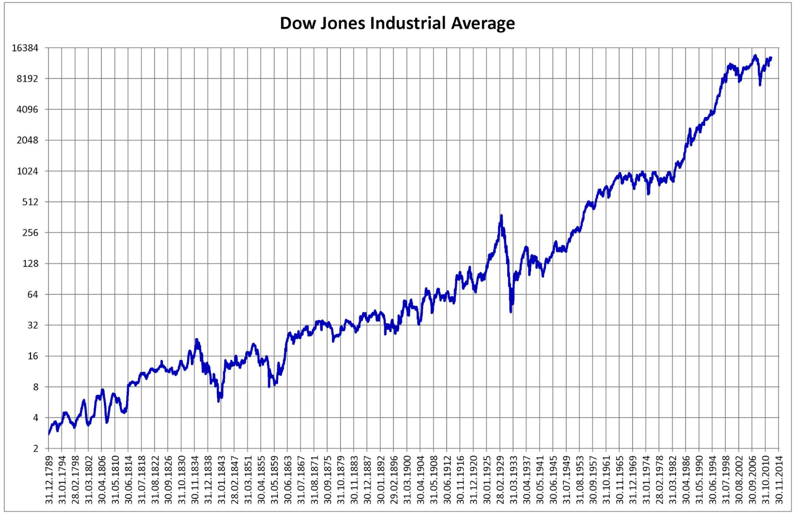
RNN use cases:
Pattern recognition: handwriting, captioning images
Sequential data: speech recognition, stock price prediction, and generating text and news stories



Time Series

A **time series** is a series of data points indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence taken at successive *equally spaced* points in time. The data are ordered (are not independent of each other).

examples: heights of ocean tides, the daily closing value of the DJIA



Not regular intervals, but in this case could put everything on a 16th note time interval

Time Series Analysis (Forecasting)

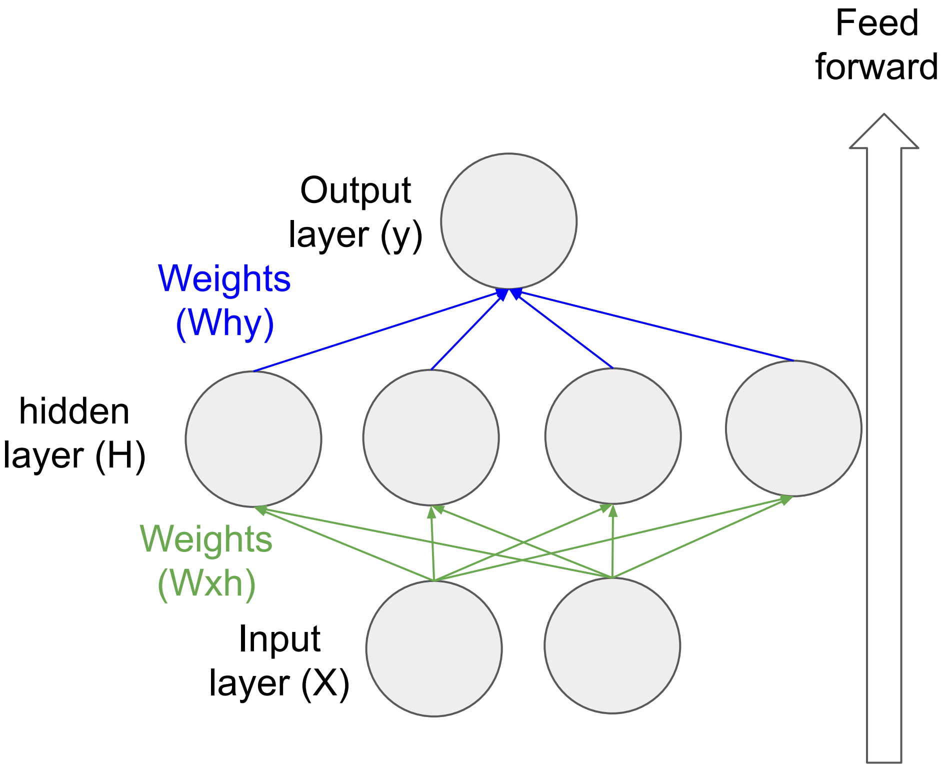
Recurrent neural networks (specifically LSTMs) are shown to outperform the classical approach to forecasting, ARIMA.

See arXiv paper.

However, you often don't need to go to such exotic approaches.
Depending on how you featurize your data, you can use a conventional technique.

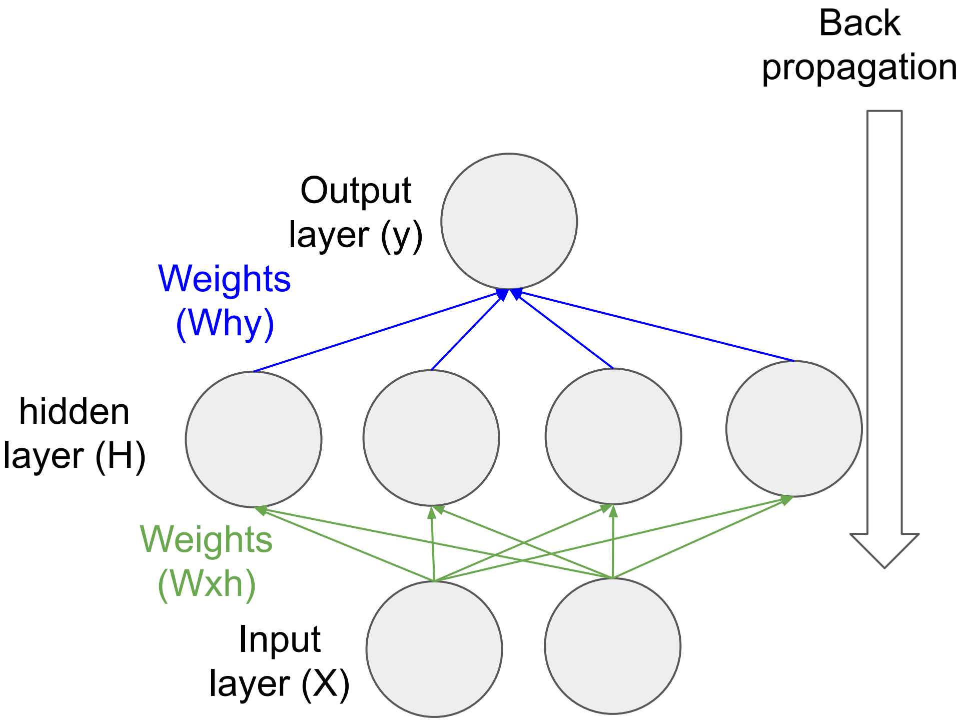
Multi-layer perceptron

- Maps input data to corresponding outputs. Calculation *feeds forward* through the network.
- Nodes are arranged in layers.
- Nodes have values for any input given the sum of inputs to the node and an *activation function* that transforms the sum to a non-linear output.
- The “learning” in the network is held by the trained values of the connections (the weights). These weights start with random values.

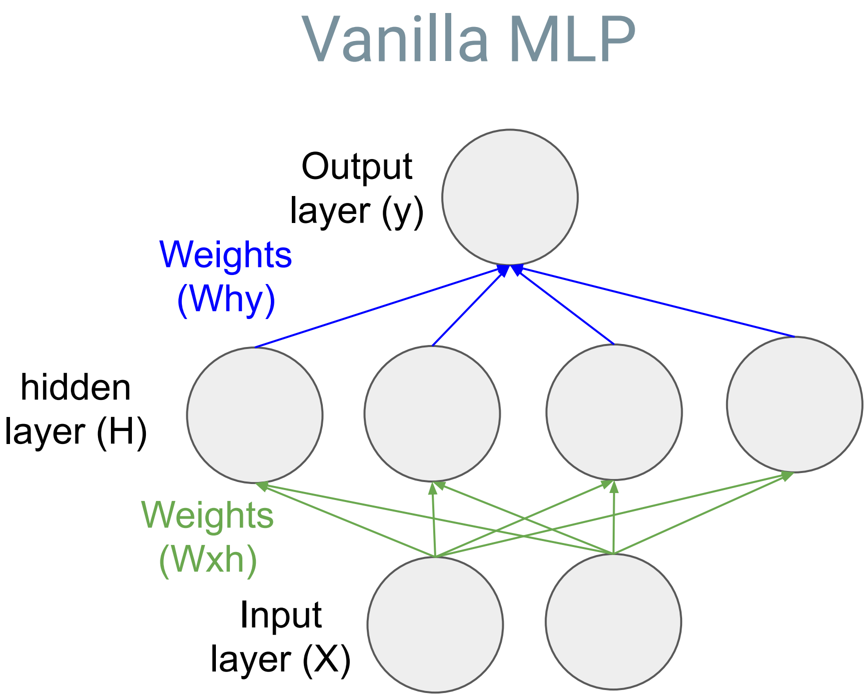


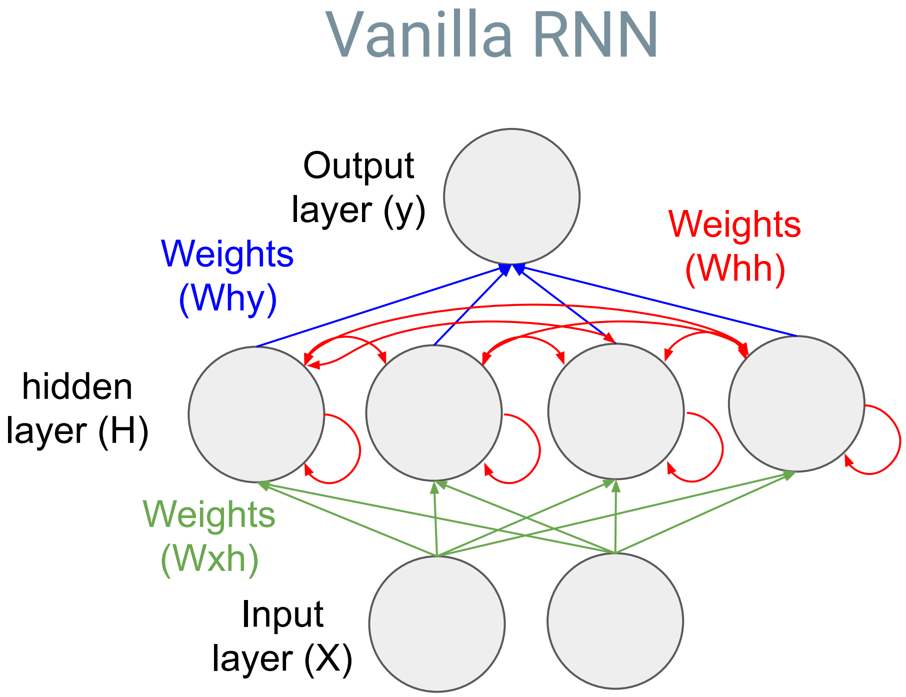
Review: Multilayer perceptron - learning the weights

- After feed forward yields a predicted output ($\mathbf{y}_p$), its difference (loss) is calculated from the real output ($\mathbf{y}$).
- Back propagation estimates how much the loss varies due to each weight in the network (the gradient).
- Gradient descent uses the gradient and learning rate to tweak each of the weights so that the network predicts a little better next time.
- When the total loss reaches an acceptable level, stop. Now it's trained and ready to predict.



Comparing the simplest version of these neural nets

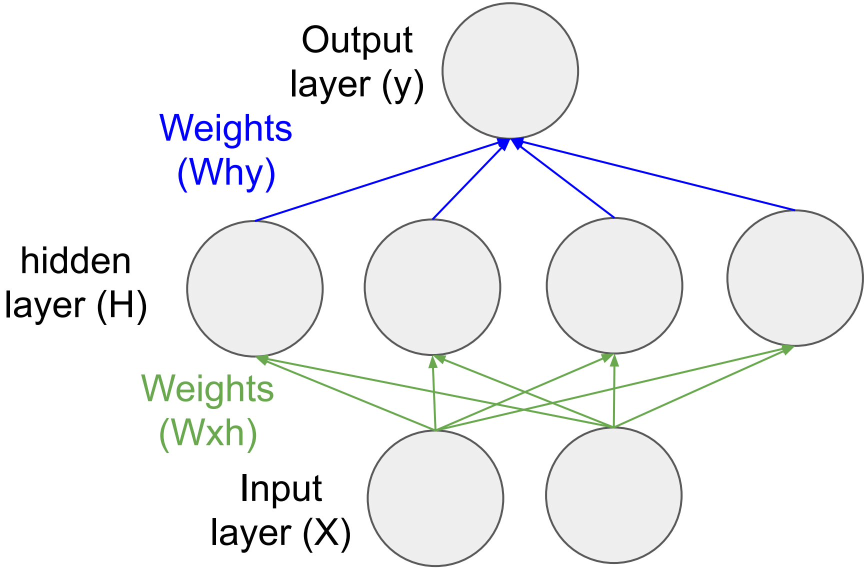




A double arrow indicates a weight in each direction (2 weights).

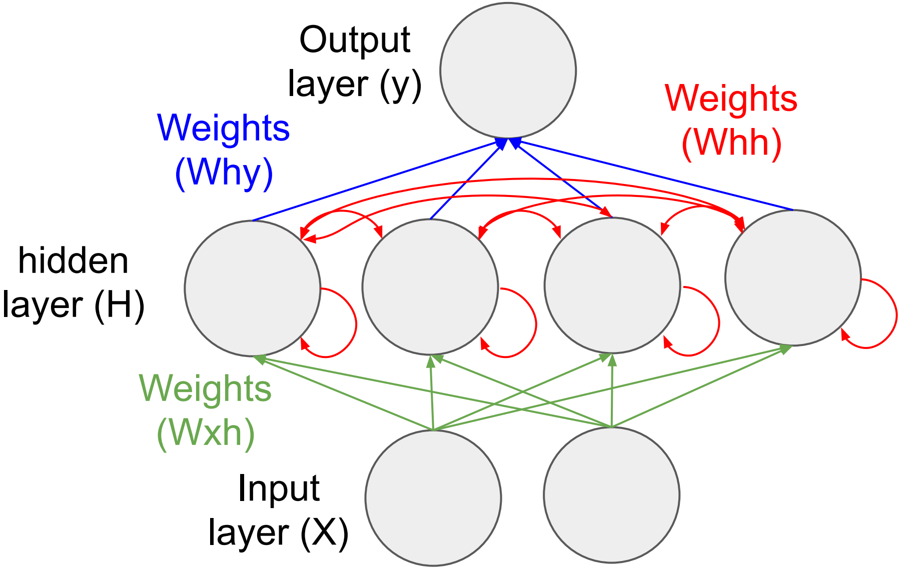
So how many weights in each architecture?

Vanilla MLP



Why =
Wxh =

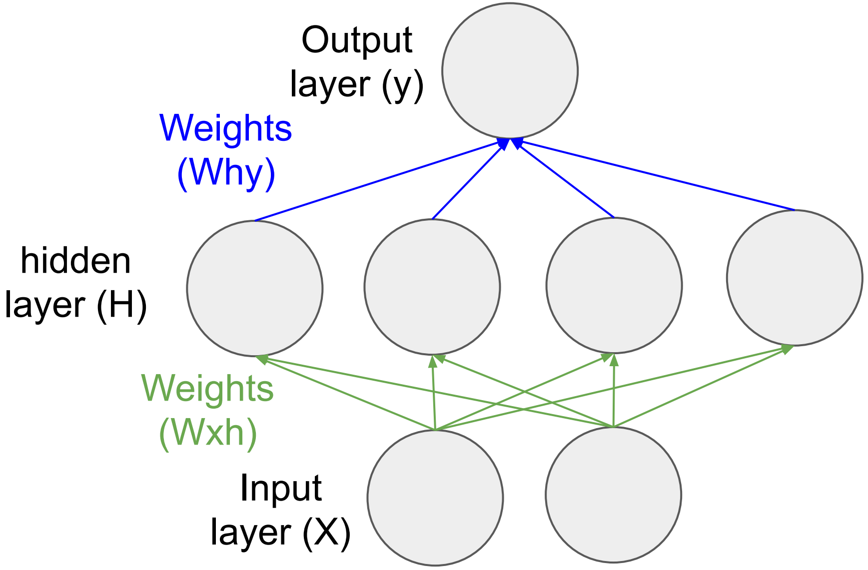
Vanilla RNN



Why =
Whh =
Wxh =

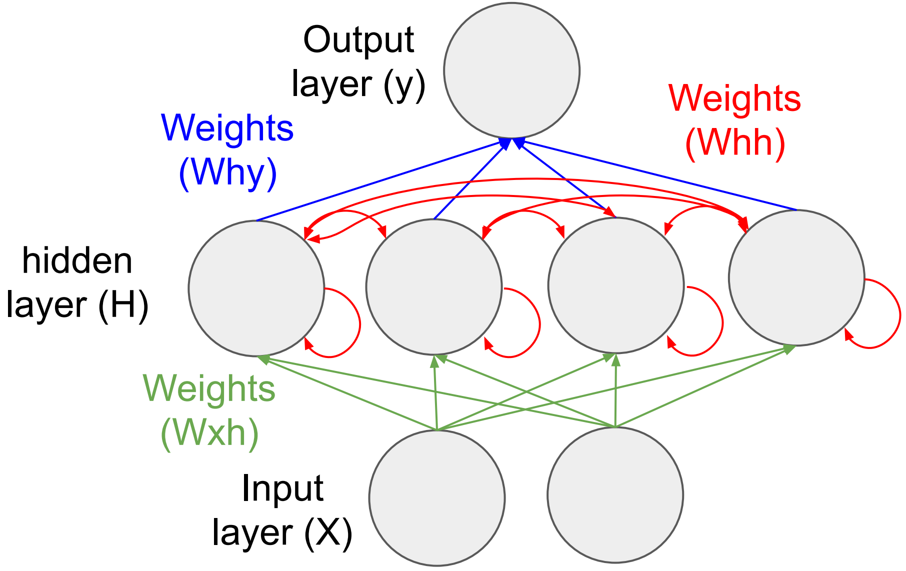
So how many weights in each architecture?

Vanilla MLP



Why = 4
Wxh = 8

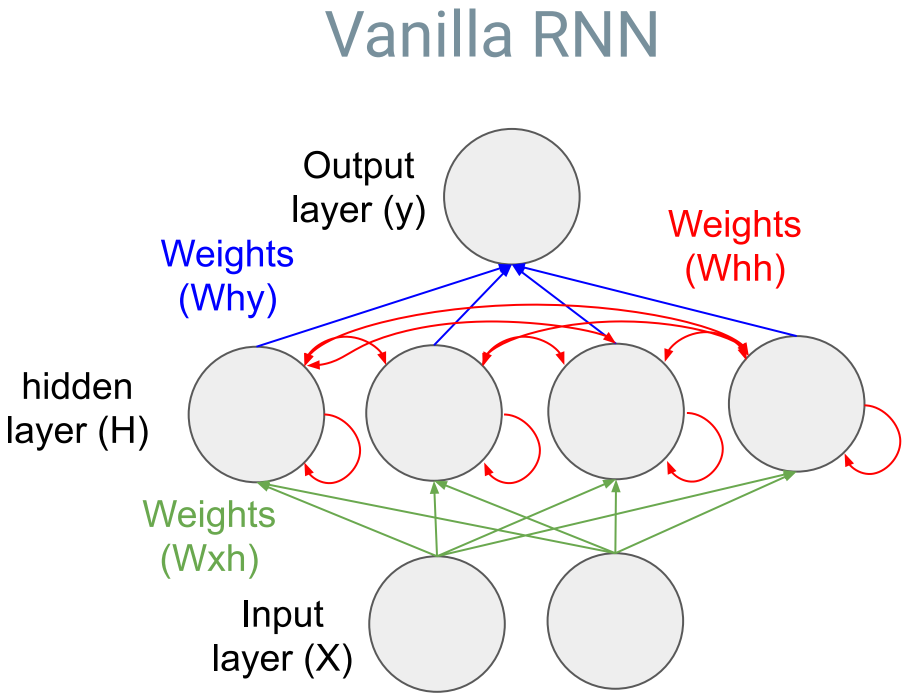
Vanilla RNN



Why = 4
Whh = 16
Wxh = 8

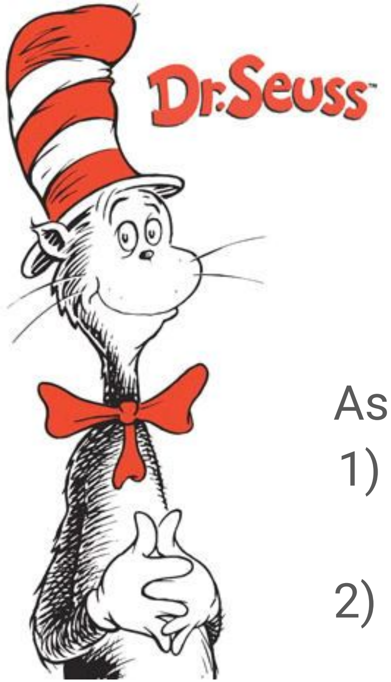
Benefit of the intra - layer recurrent connections

- The previous state of a node in a recurrent hidden layer (**H_prev** for coding purposes) can affect the value of itself or other nodes in the layer in the present time (it's a directed cycle).
- This gives the net the ability to model sequential data.
- Feedforward and backpropagation work the same way.
- Learn **Whh** like all the other weights. In a trained model all the weights are fixed. It's the activations of the nodes that changes with changes in sequence.



Exercise: RNN - text is sequential data

Use the min-char-rnn.py code to learn Dr. Seuss. As the model trains it will eventually write some new books!

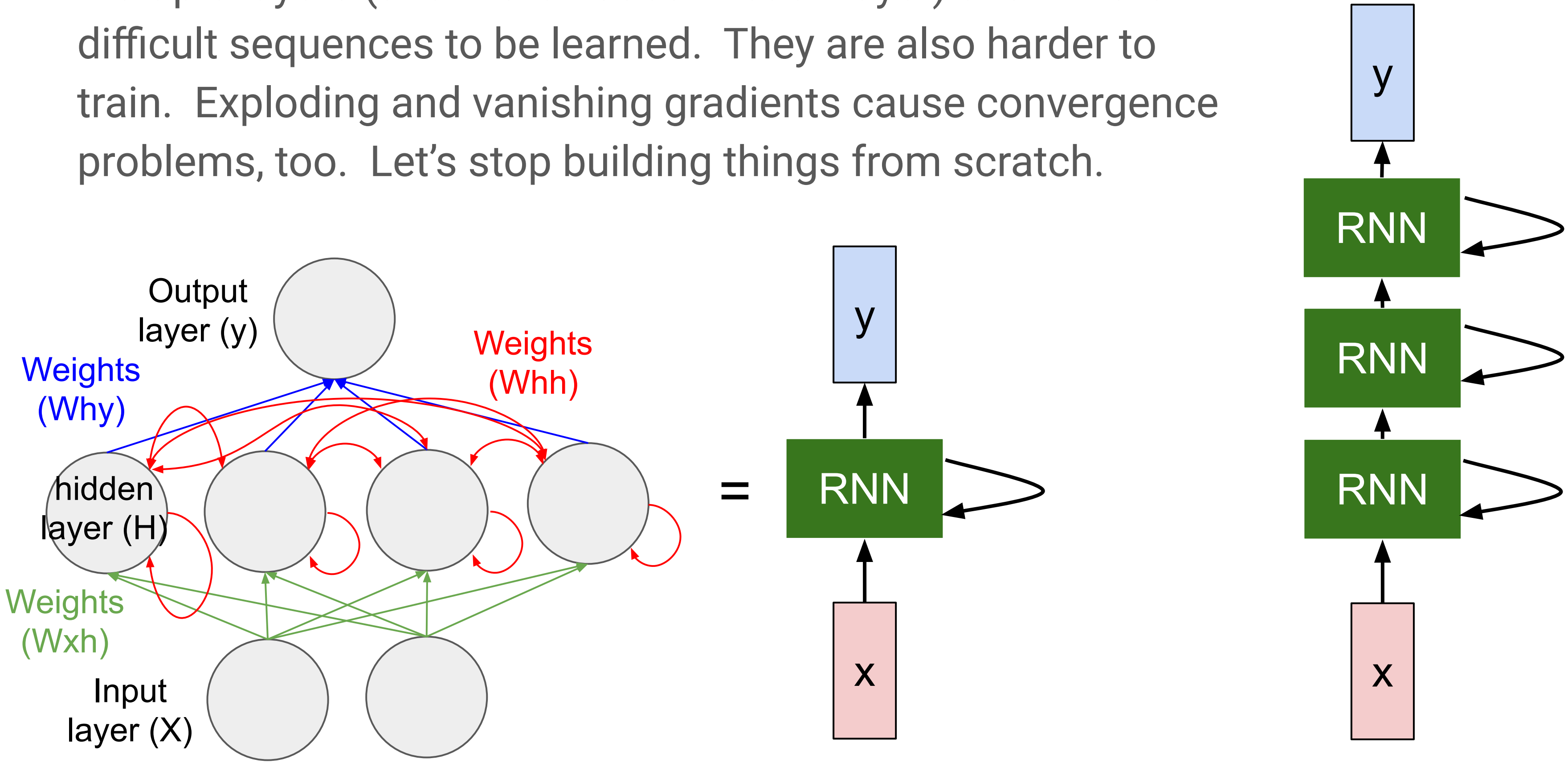


As you play with the model, try to answer the following questions:

1) What is the model's architecture?
   (#inputs, #layers, activation functions, #outputs)
2) How is the model predicting new characters?

Moving into multilayer RNNs

Multiple layers (and more nodes in each layer) allow more difficult sequences to be learned. They are also harder to train. Exploding and vanishing gradients cause convergence problems, too. Let's stop building things from scratch.



Keras

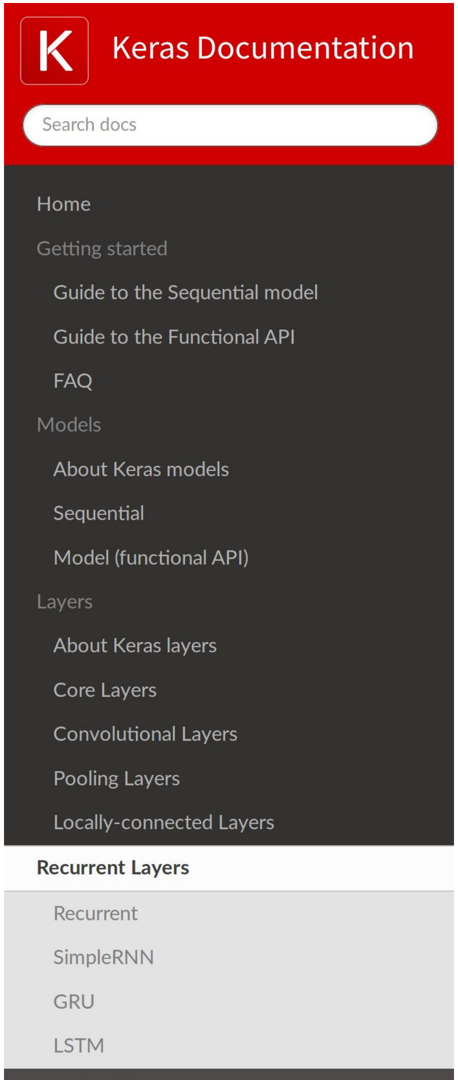
Keras is a high-level neural networks API, written in Python and capable of running on top of either TensorFlow, CNTK or Theano.

It’s one of Tensorflow’s default APIs.

We use it in the DSI for capstone projects. MLPs, CNNs, RNNs, some Reinforcement Learning too.

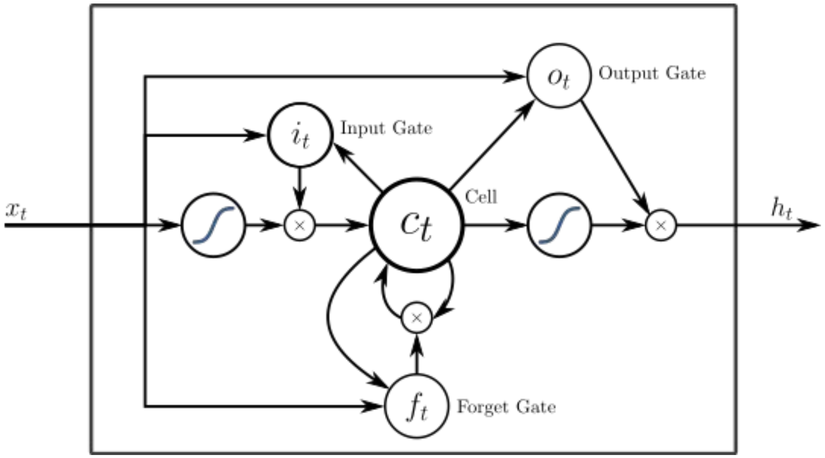
Available recurrent layers:

- Recurrent
- SimpleRNN
- Long Short-Term Memory (LSTM)
- Gated Recurrent Unit (GRU)



LSTM

- Long short-term memory (LSTM) is an architecture (an artificial neural network) proposed in 1997.
- LSTM network is well-suited … when there are time lags of unknown size and bound between important events.
- LSTM practical applications: natural language text compression, handwriting recognition, speech recognition, translation. (See Wikipedia)
- Your assignment will ask you to describe how one works, and then you'll use them in Keras to predict stock price!



Attribute

RNN resources

Andrej Karpathy's NN course, Karpathy Github
Ian Goodfellow's Deep Learning book
Iamtrask's blog
Christopher Olah's blog
Jakob Aungiers's blog, Aungiers Github

Github page for Frank's Meetup!

Past capstones:
Seinfeld Chat Bot: Matt Devor
Artificial Music Composition: Erin Desmond

Objectives

- Introduce RNNs
  - Why are RNNs good for sequential analysis?
- Time Series (common application for RNNs)
- State important differences between MLPs and RNNs
- State a common shortcoming of “vanilla” RNNs
- Explain, at a high-level, what an LSTM is (assignment)
- Predict stock price using an LSTM (assignment)