University of Pittsburgh

ECE 2195: Special Topics – Computers Machine Learning

Deep Learning – More models!

Mai Abdelhakim, PhD

Assistant Professor of ECE

Swanson School of Engineering

University of Pittsburgh

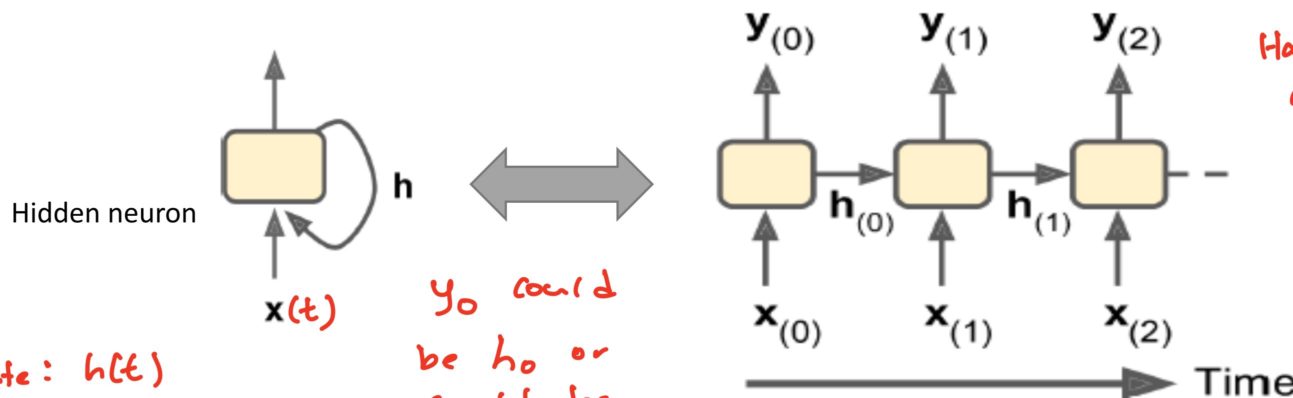
maia@pitt.edu

Recurrent Neural Networks

Typical NN (feedforward)

RNN

- RNN processes **sequential** data
- Has recurrent connection
  - Type of neural network that has an internal loop
  - Has memory of previous state to update next state
- State of hidden neuron at time step t, denoted $h(t)$, is function of input at time step t and state at the previous time $t-1$:
  $h(t) = f(h(t-1), x(t))$ --- f is activation function e.g. tanh



Has a memory about what happened in previous time step

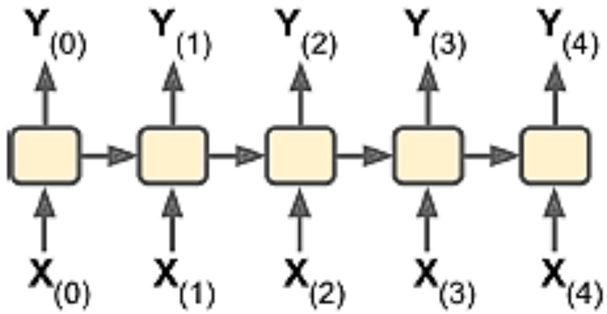
State: $h(t)$

y_0 could be h_0 or could be different

$h_1 = f(h_0, x_1)$
etc.

Different Design Patterns (Architectures 1)
Sequence to sequence network

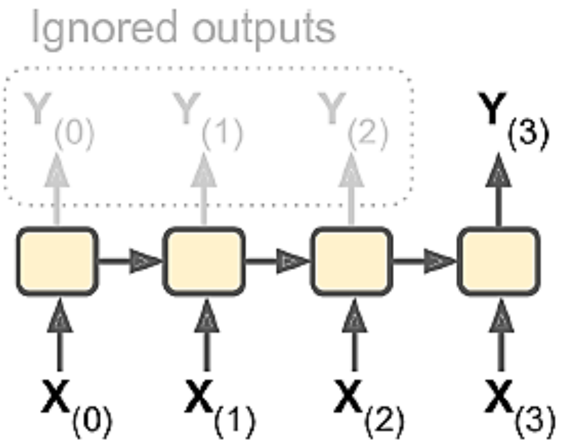
- RNN can have sequence of input and produce sequence of output
  - **Sequence to sequence network**
- Applications:
  - Word-by-word translation
  - feed the RNN with prices over last N days, and it outputs the prices shifted by one day into the future (i.e., from N – 1 days ago to tomorrow)



Ref.: Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow

Different Design Patterns (Architectures 2)
Sequence to vector network

- RNN can have sequence at input & produce a single output (represemted as a vector)
  - **Sequence to vector network**

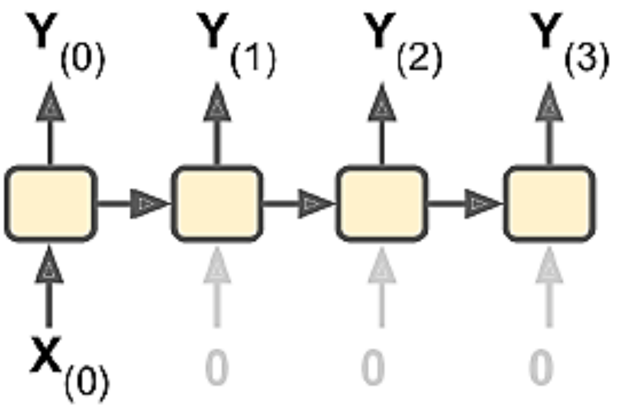


Ref.: Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow

- Applications:
  - Review classification
    - Input is a sequence of words corresponding to a **movie review**
    - Output is a prediction of whether the **review positive or negative**
  - Identify topic of sentence
  - Predict next word

Different Design Patterns (Architectures 3)
Vector to sequence network

- RNN can have a single input vector and produce an output sequence
    - Vector to sequence network
- For example, image captioning
    - Input **image**
    - Output a **caption** for that image
        - Caption is a sequence of words



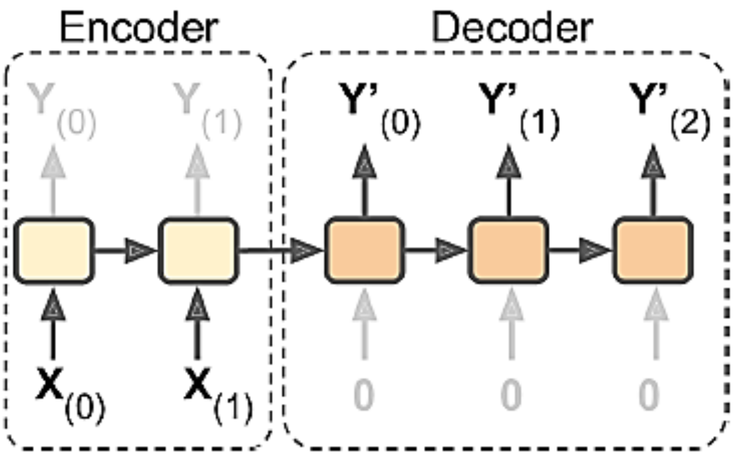
Ref.: Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow

Different Design Patterns (Architectures 4)
Sequence to sequence of different length

- RNN can take input sequence, and produce output sequence of different length

- Example: translation a sentence from one language to another

  - Get whole sentence before translation
  - Works better than translating word by word using traditional sequence to sequence



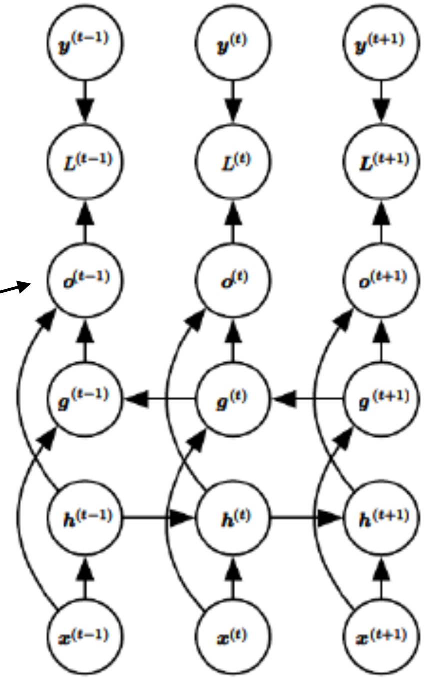
Ref.: Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow

- Composed on encoder then decoder:
  - Encoder: Input sequence to vector
  - Decoder: vector to sequence

Different Design Patterns (Architecture 5) Bidirectional RNN

- One hidden units propagates information forward in time
- Other hidden unit propagate information backward in time

Has summary information of past and future



- Example: Natural language processing
  - Speech recognition: understand after hearing future words

"she stop by yesterday"

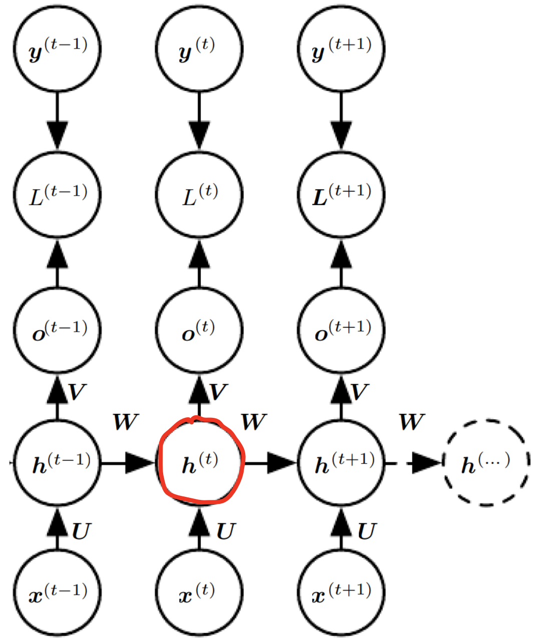
if it only cares about what happened before, would be corrected to stops

BUT - if future information is propagated back, it could be corrected to stopped

Unfolding Basic RNN

Prior hidden state

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)}),
\end{aligned}
$$

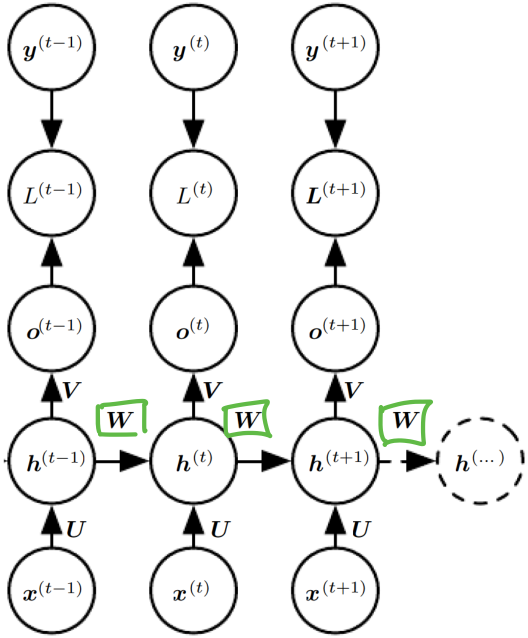


- Parameters are shared across timesteps

$$h(t) = \tanh\left(b + W h^{(t-1)} + U x^{(t)}\right)$$

Unfolding Basic RNN

$$\begin{aligned}\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)},\\ \boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}),\\ \boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)},\\ \hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)}),\end{aligned}$$



- Parameters are shared across timesteps applying same weights to different times in sequence
  - For simplicity consider no non-linear activation and focus on the node states

$$\boldsymbol{h}^{(t)} = \boldsymbol{W}^{\top}\boldsymbol{h}^{(t-1)} \quad \Rightarrow \quad \boldsymbol{h}^{(t)} = \left(\boldsymbol{W}^{t}\right)^{\top}\boldsymbol{h}^{(0)}$$

Gradient will vanish or explode due to applying sample weight several times

$$\boldsymbol{h}^{(t)} = \boldsymbol{W}^\top \boldsymbol{h}^{(t-1)} \Rightarrow \boldsymbol{h}^{(t)} = \left(\boldsymbol{W}^t\right)^\top \boldsymbol{h}^{(0)}$$

- $\frac{\partial h^{(t)}}{\partial h^{(0)}} = \frac{\partial h^{(t)}}{\partial h^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial h^{(t-2)}} \dots \frac{\partial h^{(1)}}{\partial h^{(0)}} = W^T W^T \dots W^T$

$\Rightarrow$ vanish or explode

Slopes get very small (vanish) or very big (explode) if we consider a long sequence !!

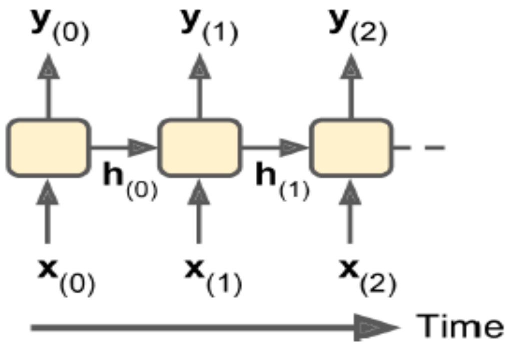
Hard to Capture Long Term Dependencies

- Gradient propagated over many steps will vanish or explode
  - This makes it harder to capture the long-term dependencies

- Computation has multiplication with weight multiple times
  - could vanish (small values) or gets very large (explode).

$$h^{(t)} = W^\top h^{(t-1)} \Rightarrow h^{(t)} = \left(W^t\right)^\top h^{(0)}$$

- **Implication:** Not learning from long sequence
- **Some solutions:** normalize weights, clip large gradient, skip connections,..



Could we modify the processing in the neuron to solve this problems?

LSTM: Long Short-Term Memory Model

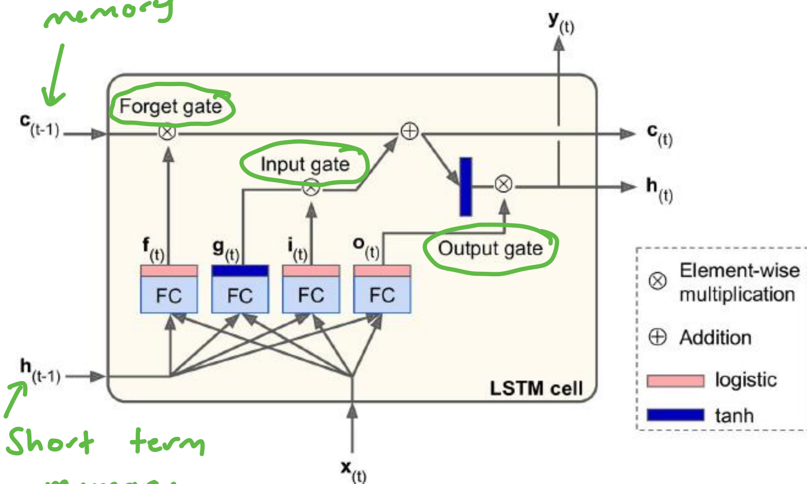
- Modified version of vanilla (i.e, basic) RNN

- Idea : not all information in a long sequence is important

  - E.g. long review – only fews words can tell if the review is good or bad

- LSTM tries to learn what information to keep in the memory and what to discard

Long term memory



Short term memory

Reading chapter 14.: Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow

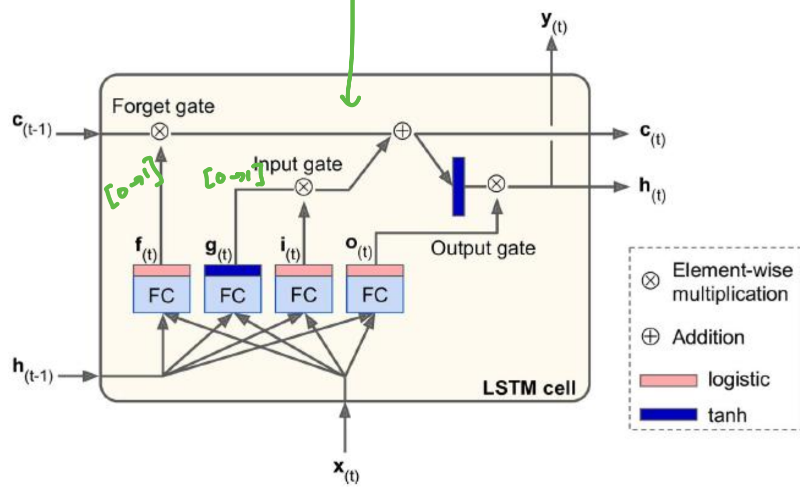
LSTM: Long Short-Term Memory Model

- **Objective**: learn when to include important input, when to consider long term memory, and when to shut off the output.
- Has two states c and h
  - Short term state h
  - Long term state c
- Network learns what to store in the long-term state & what to discard
  - Forget gate control which part of the long-term state to be discarded

- Input gate controls the part of input that should be added to the long-term state

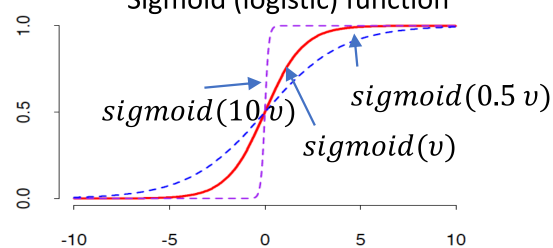
- Output of the state is controlled by the output gate

Is input important enough to be added to long term memory?



Reading chapter 14.: Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow

Sigmoid (logistic) function



LSTM: Long Short-Term Memory Model

$$\mathbf{i}_{(t)} = \sigma(\mathbf{W}_{xi}{}^T \mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^T \mathbf{h}_{(t-1)} + \mathbf{b}_i)$$

Input/update gate

$$\mathbf{f}_{(t)} = \sigma(\mathbf{W}_{xf}{}^T \mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^T \mathbf{h}_{(t-1)} + \mathbf{b}_f)$$

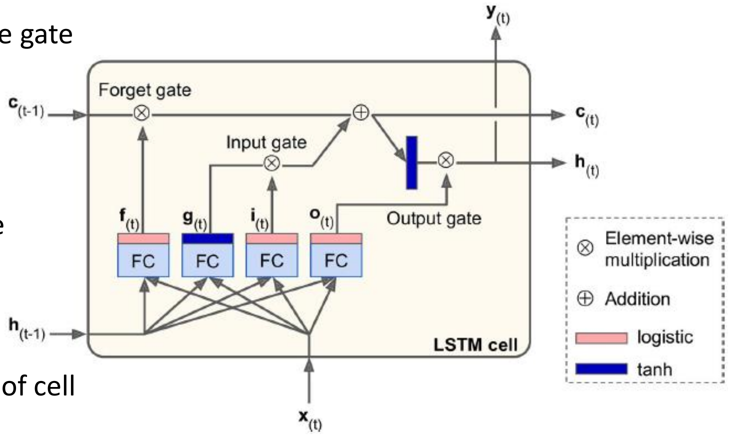
Forget gate

$$\mathbf{o}_{(t)} = \sigma(\mathbf{W}_{xo}{}^T \mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^T \mathbf{h}_{(t-1)} + \mathbf{b}_o)$$

Output gate

$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}{}^T \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^T \mathbf{h}_{(t-1)} + \mathbf{b}_g)$$

State of cell



$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

Long term state: how much to remember previous state, how much to include the current state

$$\mathbf{y}_{(t)} = \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})$$

How much to output from this state

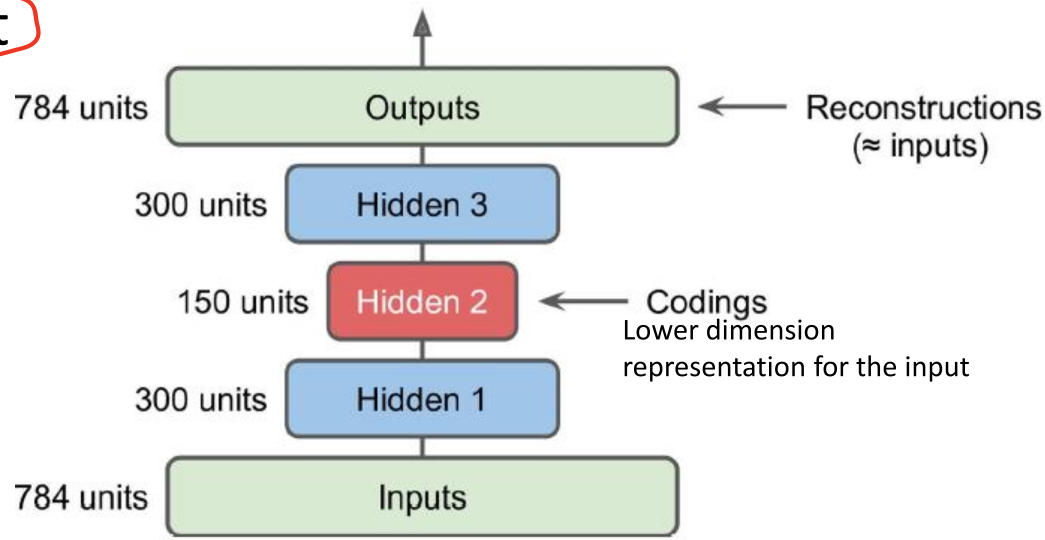
- Keras implementation
  - https://www.tensorflow.org/guide/keras/rnn

```
# Add a LSTM layer with 128 internal units.
model.add(layers.LSTM(128))
```

Another Deep Learning Model - Autoencoders

- Final output = input
  - Unsupervised

- **Applications:**
  - **denoising**
  - **dimensionality reduction**



Stacked Autoencoder: has multiple hidden layers

Typically symmetrical with regards to the central hidden layer (the coding layer).

Ref: chapter 15, "Hands-On Machine Learning with Scikit-Learn and TensorFlow", by Géron

Resources

- Deep learning book
  - https://www.deeplearningbook.org/lecture_slides.html