Deep Learning: Models for Sequence Data (RNN and LSTM)

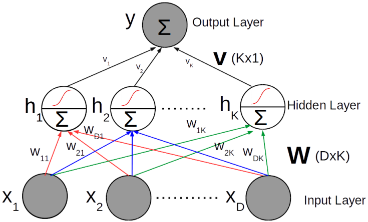
Piyush Rai

Machine Learning (CS771A)

Nov 4, 2016

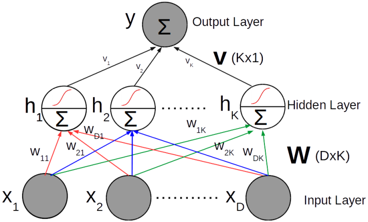
Recap: Feedforward Neural Network

- Consists of an input layer, one or more hidden layers, and an output layer

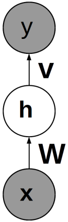


Recap: Feedforward Neural Network

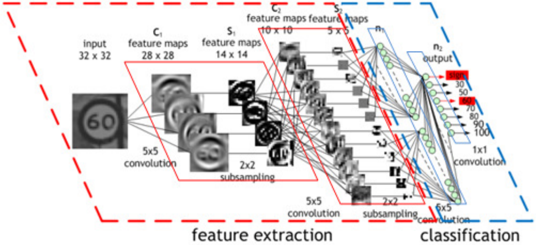
- Consists of an input layer, one or more hidden layers, and an output layer



- A "macro" view of the above (note: $\boldsymbol{x} = [x_1, \ldots, x_D], \boldsymbol{h} = [h_1, \ldots, h_K]$)

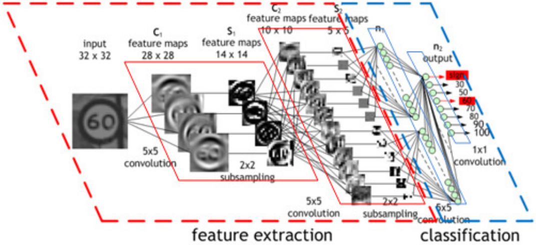


Recap: Convolutional Neural Network



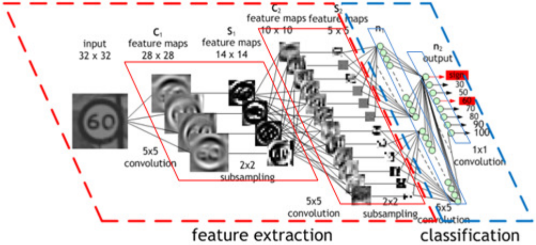
- Special type of feedforward neural nets (local connectivity + weight sharing)

Recap: Convolutional Neural Network



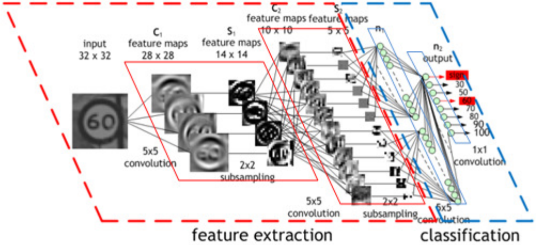
- Special type of feedforward neural nets (local connectivity + weight sharing)
- Each layer uses a set of "filters" (basically, weights to be learned) which can detect specific features. Filters are like basis/dictionary (PCA analogy)

Recap: Convolutional Neural Network



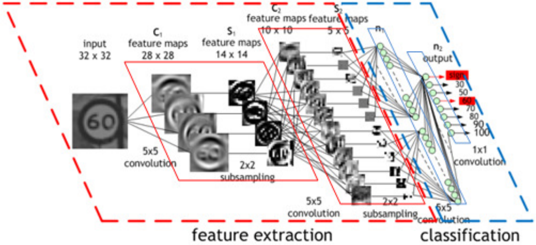
- Special type of feedforward neural nets (local connectivity + weight sharing)
- Each layer uses a set of "filters" (basically, weights to be learned) which can detect specific features. Filters are like basis/dictionary (PCA analogy)
- Each filter is convolved over entire input to produce a feature map

Recap: Convolutional Neural Network



- Special type of feedforward neural nets (local connectivity + weight sharing)
- Each layer uses a set of "filters" (basically, weights to be learned) which can detect specific features. Filters are like basis/dictionary (PCA analogy)
- Each filter is convolved over entire input to produce a feature map
- Nonlinearity and pooling and applied after each convolution layer

Recap: Convolutional Neural Network

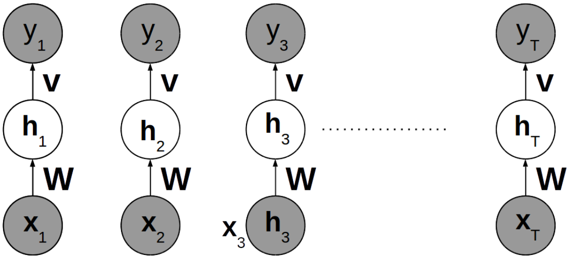


- Special type of feedforward neural nets (local connectivity + weight sharing)
- Each layer uses a set of "filters" (basically, weights to be learned) which can detect specific features. Filters are like basis/dictionary (PCA analogy)
- Each filter is convolved over entire input to produce a feature map
- Nonlinearity and pooling and applied after each convolution layer
- Last layer (one that connects to outputs) is fully connected

Deep Neural Networks for Modeling Sequence Data

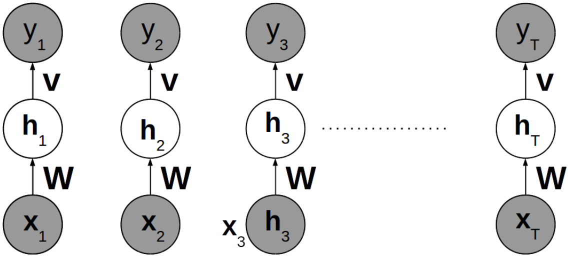
Limitation of Feedforward Neural Nets

- FFNN can't take into account the sequential structure in the data
- For a sequence of observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$, their corresponding hidden units (states) $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T$ are assumed independent of each other



Limitation of Feedforward Neural Nets

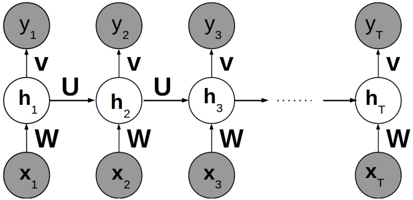
- FFNN can't take into account the sequential structure in the data
- For a sequence of observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$, their corresponding hidden units (states) $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T$ are assumed independent of each other



- Not idea for sequential data, e.g., sentence/paragraph/document (sequence of words), video (sequence of frames), etc.

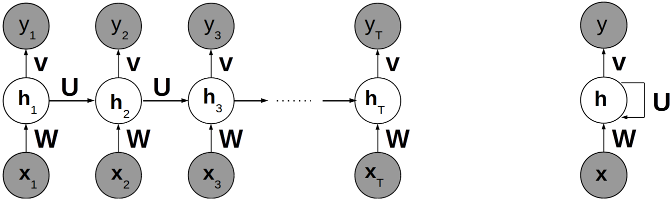
Recurrent Neural Nets (RNN)

- Hidden state at each step depends on the hidden state of the previous



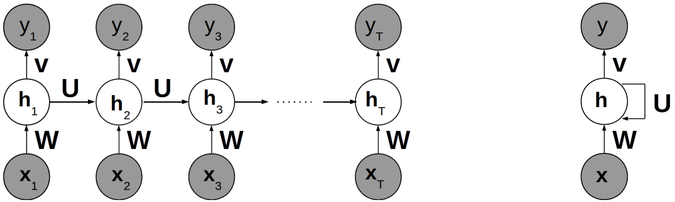
Recurrent Neural Nets (RNN)

- Hidden state at each step depends on the hidden state of the previous



Recurrent Neural Nets (RNN)

- Hidden state at each step depends on the hidden state of the previous



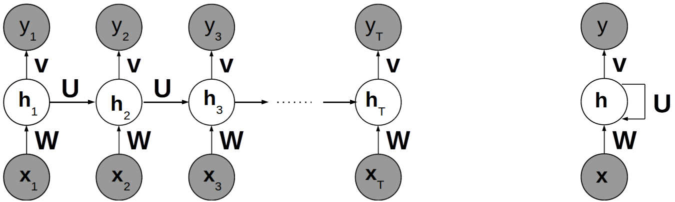
- Each hidden state is typically defined as

$$\boldsymbol{h}_t = f(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

where $\mathbf{U}$ is like a transition matrix and f is some nonlin. fn. (e.g., tanh)

Recurrent Neural Nets (RNN)

- Hidden state at each step depends on the hidden state of the previous



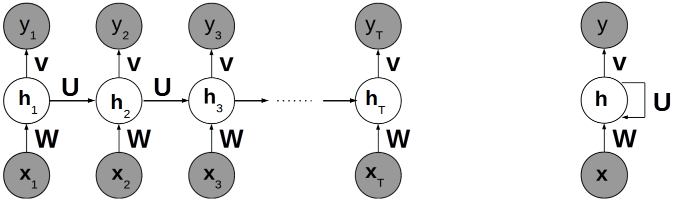
- Each hidden state is typically defined as

$$\boldsymbol{h}_t = f(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

  where $\mathbf{U}$ is like a transition matrix and f is some nonlin. fn. (e.g., tanh)
- Now $\boldsymbol{h}_t$ acts as a memory. Helps us remember what happened up to step t

Recurrent Neural Nets (RNN)

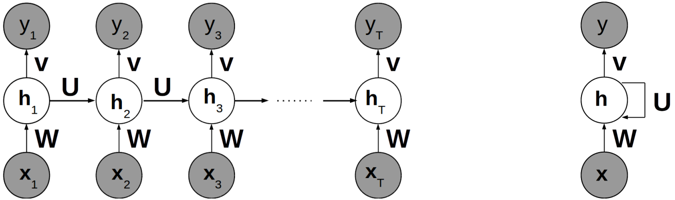
- Hidden state at each step depends on the hidden state of the previous



- Each hidden state is typically defined as
$$\boldsymbol{h}_t = f(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$
where $\mathbf{U}$ is like a transition matrix and f is some nonlin. fn. (e.g., tanh)
- Now $\boldsymbol{h}_t$ acts as a memory. Helps us remember what happened up to step t
- Note: Unlike sequence data models such as HMM where each state is discrete, RNN states are continuous-valued

Recurrent Neural Nets (RNN)

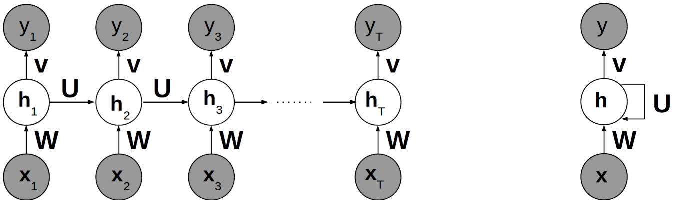
- Hidden state at each step depends on the hidden state of the previous



- Each hidden state is typically defined as
$$\boldsymbol{h}_t = f(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$
where $\mathbf{U}$ is like a transition matrix and f is some nonlin. fn. (e.g., tanh)
- Now $\boldsymbol{h}_t$ acts as a memory. Helps us remember what happened up to step t
- Note: Unlike sequence data models such as HMM where each state is discrete, RNN states are continuous-valued (in that sense, RNNs are similar to Linear-Gaussian models like Kalman Filters which have continuous states)

Recurrent Neural Nets (RNN)

- Hidden state at each step depends on the hidden state of the previous



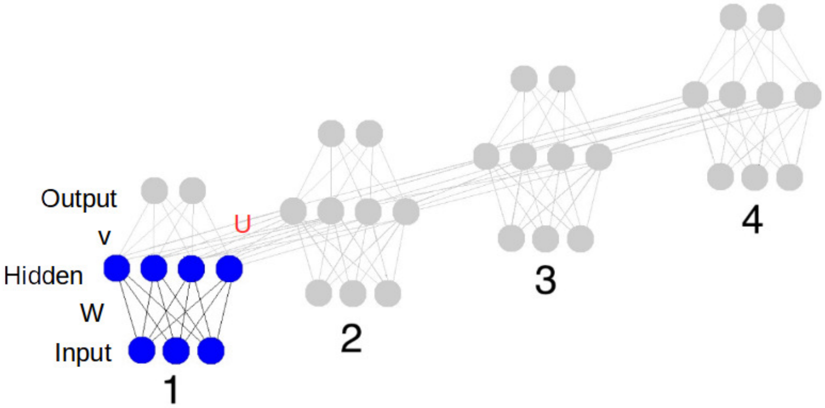
- Each hidden state is typically defined as

$$\boldsymbol{h}_t = f(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

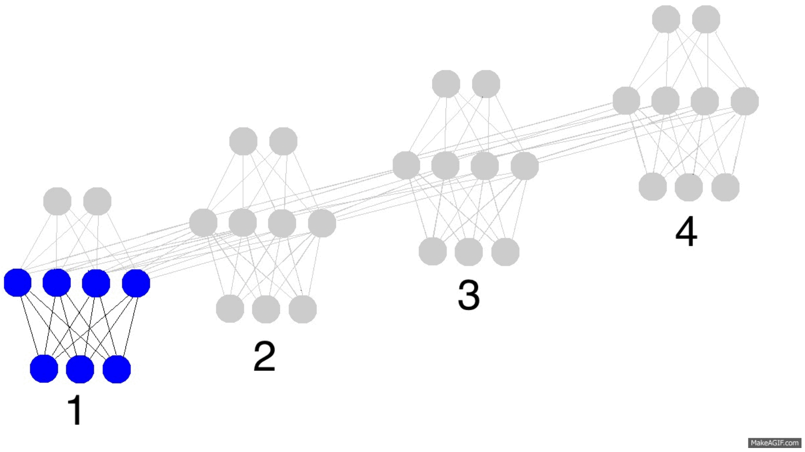
  where $\mathbf{U}$ is like a transition matrix and f is some nonlin. fn. (e.g., tanh)
- Now $\boldsymbol{h}_t$ acts as a memory. Helps us remember what happened up to step t
- Note: Unlike sequence data models such as HMM where each state is discrete, RNN states are continuous-valued (in that sense, RNNs are similar to Linear-Gaussian models like Kalman Filters which have continuous states)
- RNNs can also be extended to have more than one hidden layer

Recurrent Neural Nets (RNN)

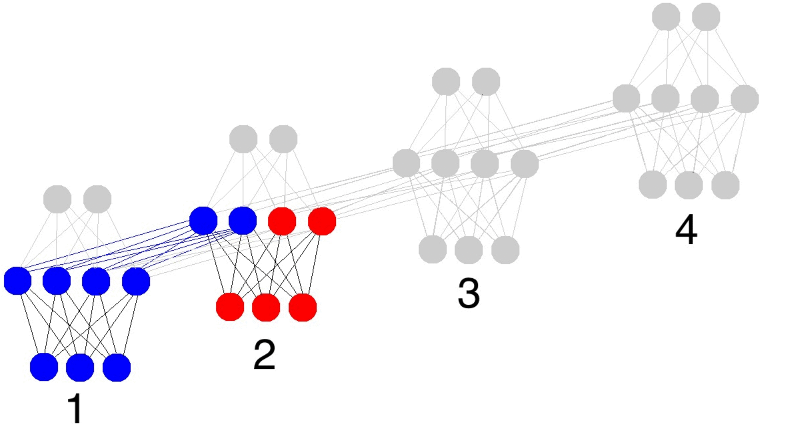
- A more "micro" view of RNN (the transition matrix **U** connects the hidden states across observations, propagating information along the sequence)



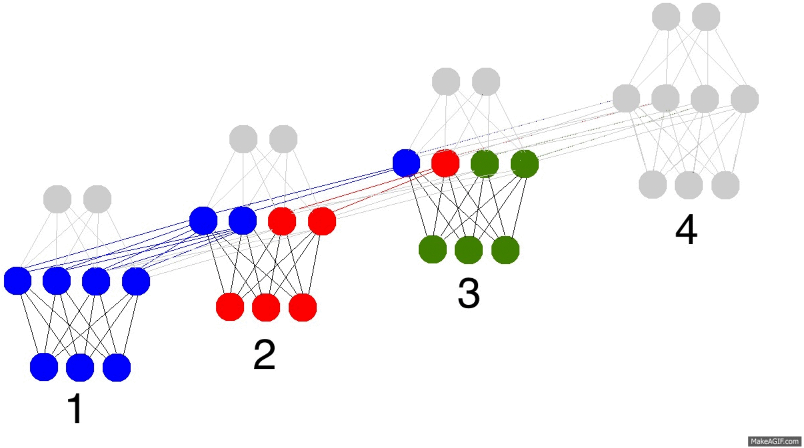
RNN in Action..



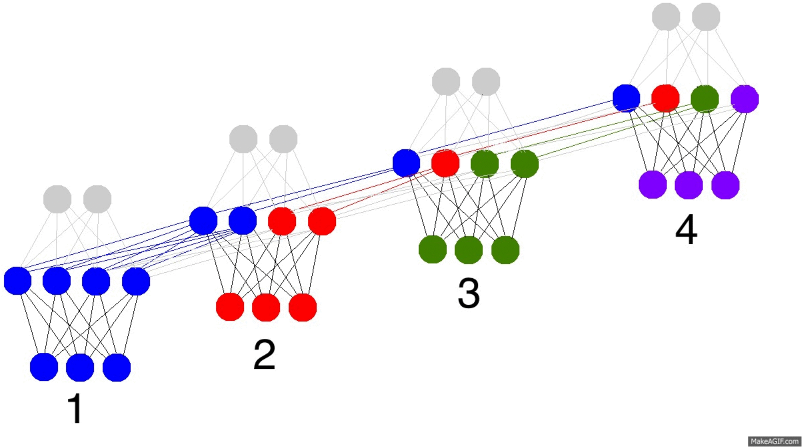
RNN in Action..



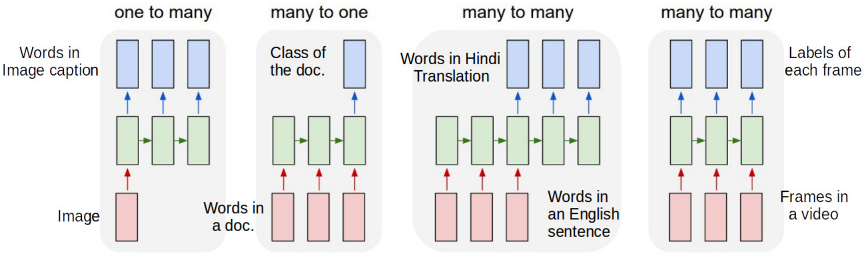
RNN in Action..



RNN in Action..

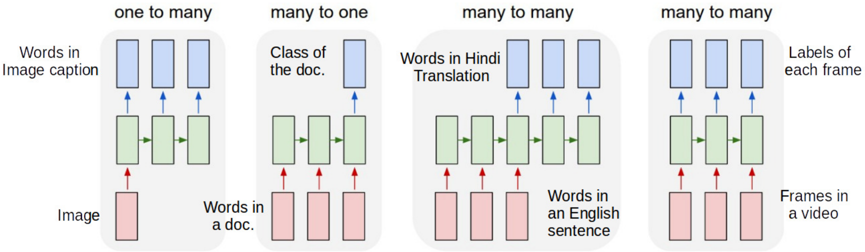


RNN: Applications



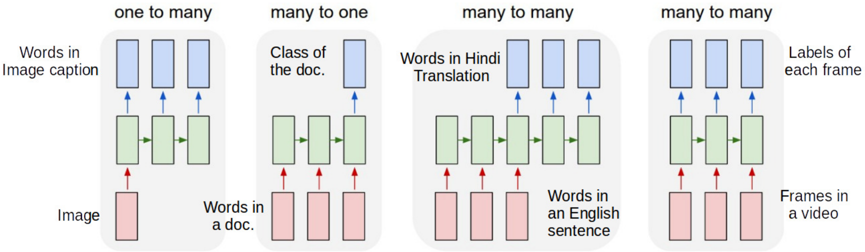
- RNNs are widely applicable and are also very flexible.

RNN: Applications



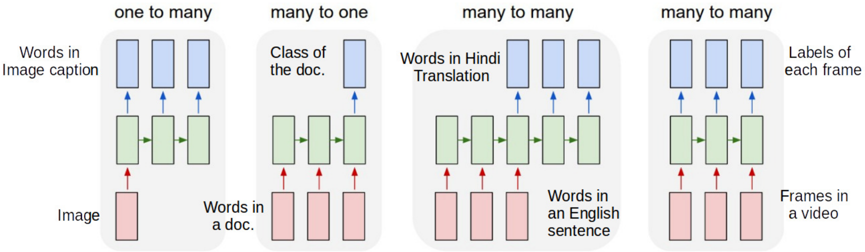
- RNNs are widely applicable and are also very flexible. E.g.,

RNN: Applications



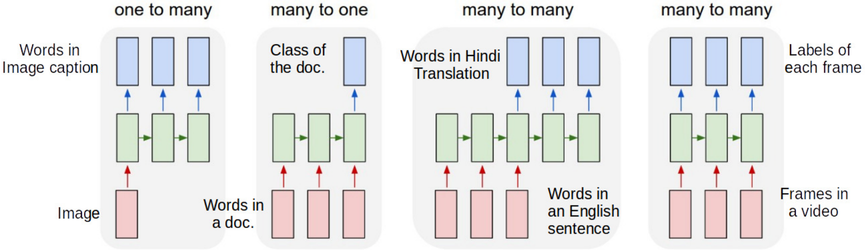
- RNNs are widely applicable and are also very flexible. E.g.,
  - Input, output, or both, can be sequences (possibly of different lengths)

RNN: Applications



- RNNs are widely applicable and are also very flexible. E.g.,
  - Input, output, or both, can be sequences (possibly of different lengths)
  - Different inputs (and different outputs) need not be of the same length

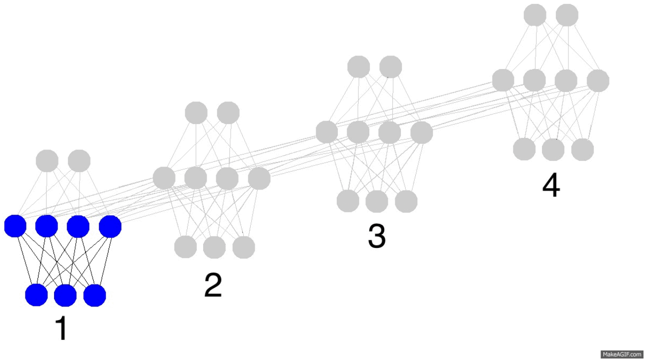
RNN: Applications



- RNNs are widely applicable and are also very flexible. E.g.,
  - Input, output, or both, can be sequences (possibly of different lengths)
  - Different inputs (and different outputs) need not be of the same length
  - Regardless of the length of the input sequence, RNN will learn a fixed size embedding for the input sequence

Training RNN

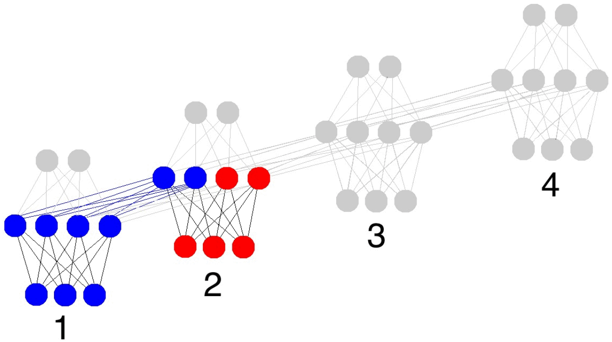
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

Training RNN

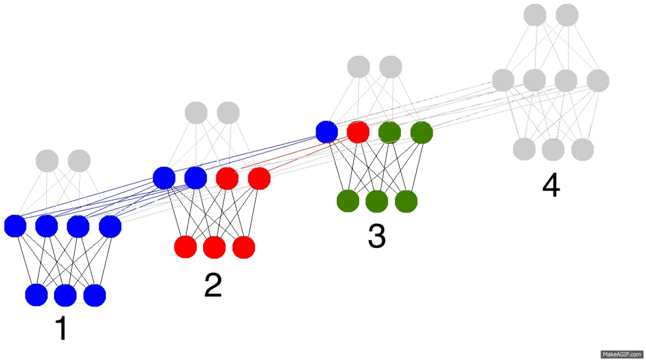
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

Training RNN

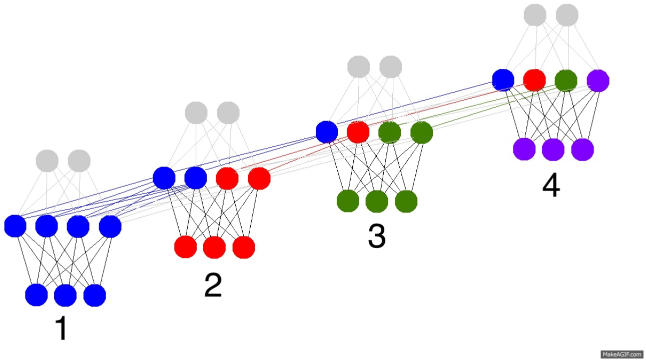
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

Training RNN

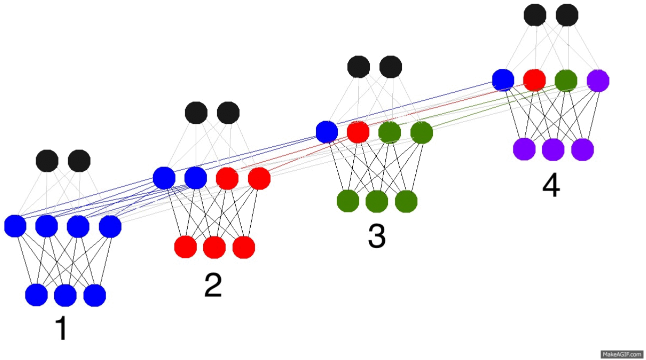
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

Training RNN

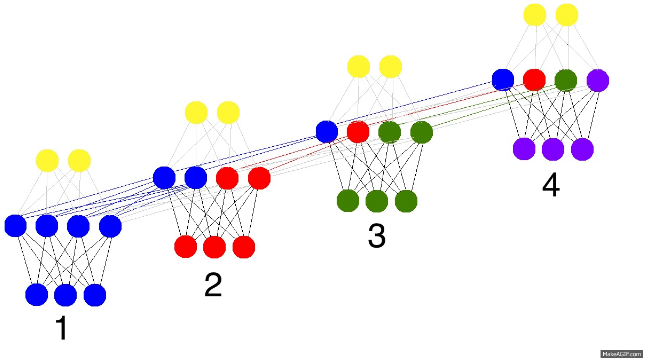
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

Training RNN

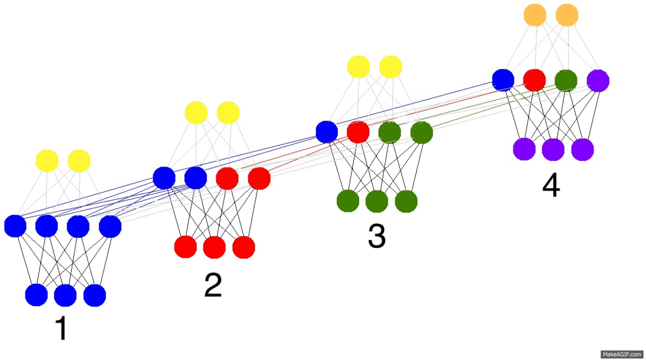
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

Training RNN

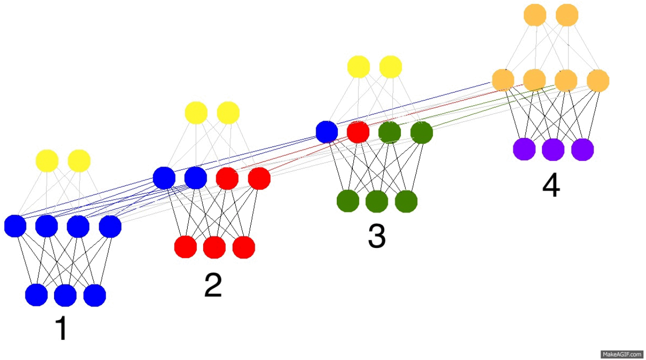
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

Training RNN

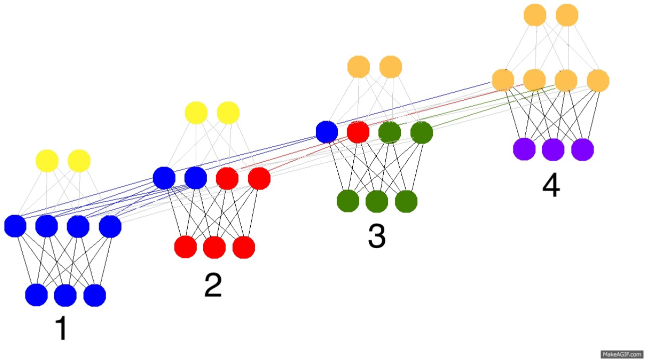
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

Training RNN

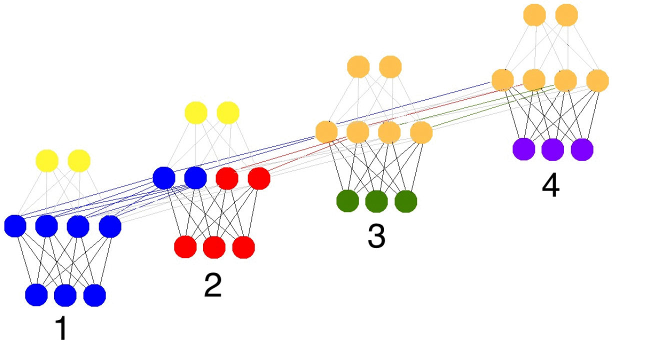
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

Training RNN

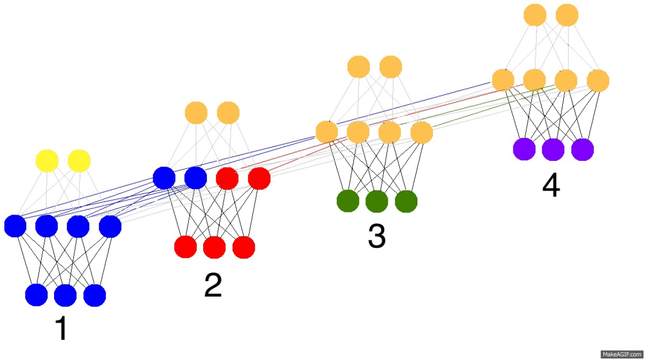
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

Training RNN

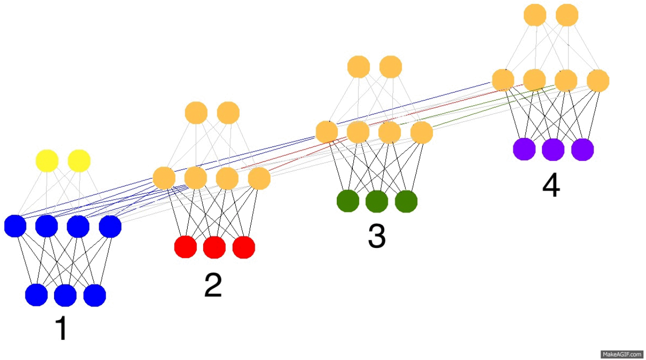
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

Training RNN

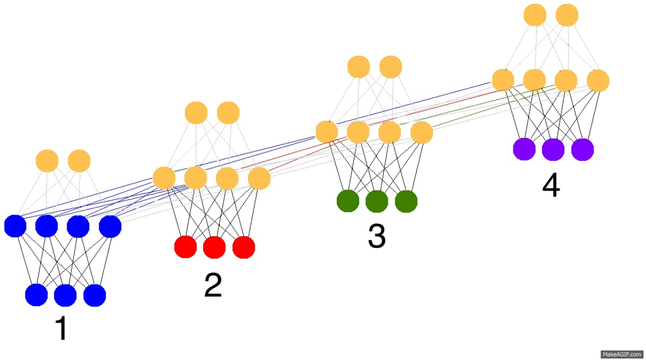
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

Training RNN

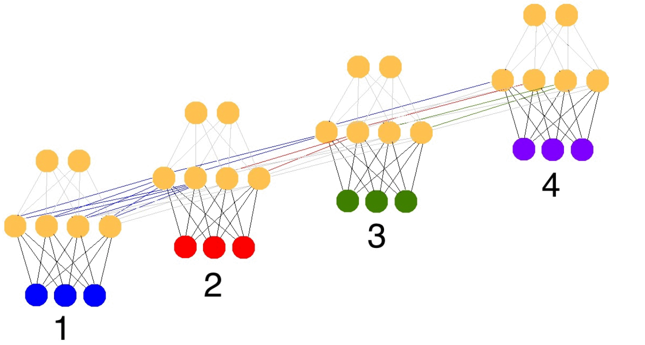
- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



- Black: Prediction, Yellow: Error, Orange: Gradients

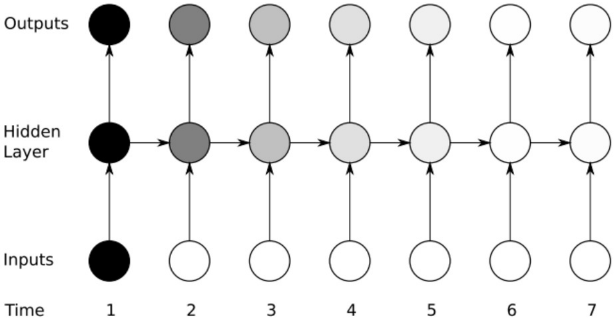
Training RNN

- Trained using Backpropagation Through Time (forward propagate from step 1 to end, and then backward propagate from end to step 1)
- Think of the time-dimension as another hidden layer and then it is just like standard backpropagation for feedforward neural nets



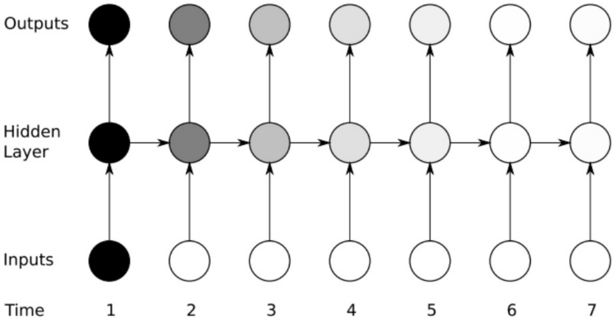
- Black: Prediction, Yellow: Error, Orange: Gradients

RNN: Vanishing/Exploding Gradients Problem



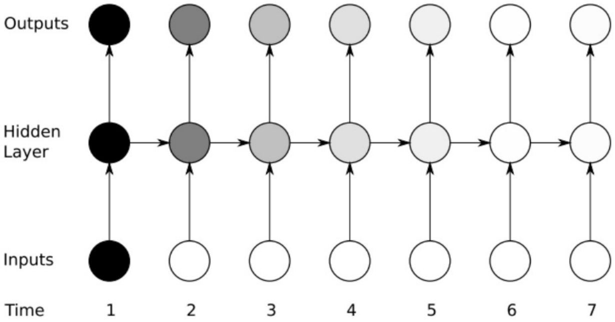
- Sensitivity of hidden states and outputs on a given input becomes weaker as we move away from it along the sequence (weak memory)

RNN: Vanishing/Exploding Gradients Problem



- Sensitivity of hidden states and outputs on a given input becomes weaker as we move away from it along the sequence (weak memory)
- New inputs "overwrite" the activations of previous hidden states

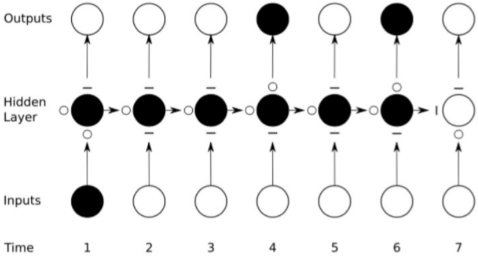
RNN: Vanishing/Exploding Gradients Problem



- Sensitivity of hidden states and outputs on a given input becomes weaker as we move away from it along the sequence (weak memory)
- New inputs "overwrite" the activations of previous hidden states
- Repeated multiplications can cause the gradients to vanish or explode

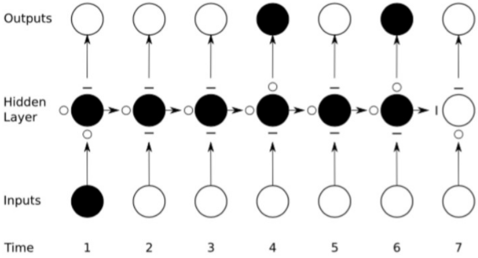
Capturing Long-Range Dependencies

- Idea: Augment the hidden states with gates (with parameters to be learned)
- These gates can help us remember and forget information "selectively"



Capturing Long-Range Dependencies

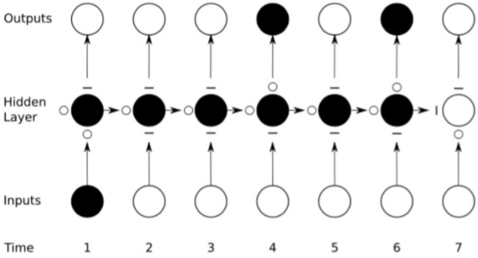
- Idea: Augment the hidden states with gates (with parameters to be learned)
- These gates can help us remember and forget information "selectively"



- The hidden states have 3 type of gates
  - Input (bottom), Forget (left), Output (top)

Capturing Long-Range Dependencies

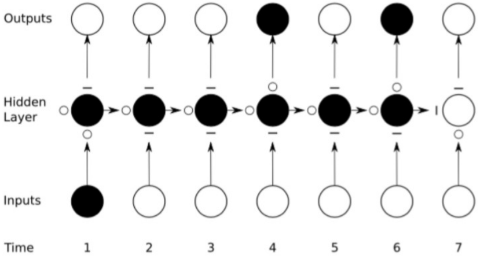
- Idea: Augment the hidden states with gates (with parameters to be learned)
- These gates can help us remember and forget information "selectively"



- The hidden states have 3 type of gates
  - Input (bottom), Forget (left), Output (top)
- Open gate denoted by 'o', closed gate denoted by '-'

Capturing Long-Range Dependencies

- Idea: Augment the hidden states with gates (with parameters to be learned)
- These gates can help us remember and forget information "selectively"



- The hidden states have 3 type of gates
  - Input (bottom), Forget (left), Output (top)
- Open gate denoted by 'o', closed gate denoted by '-'
- LSTM (Hochreiter and Schmidhuber, mid-90s): **Long Short-Term Memory** is one such idea

Long Short-Term Memory (LSTM)

- Essentially an RNN, except that the hidden states are computed differently

Long Short-Term Memory (LSTM)

- Essentially an RNN, except that the hidden states are computed differently
- Recall that RNN computes the hidden states as

$$\boldsymbol{h}_t = \tanh(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

Long Short-Term Memory (LSTM)

- Essentially an RNN, except that the hidden states are computed differently
- Recall that RNN computes the hidden states as

$$\boldsymbol{h}_t = \tanh(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

- For RNN: State update is multiplicative (weak memory and gradient issues)

Long Short-Term Memory (LSTM)

- Essentially an RNN, except that the hidden states are computed differently
- Recall that RNN computes the hidden states as

$$\boldsymbol{h}_t = \tanh(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

- For RNN: State update is multiplicative (weak memory and gradient issues)
- In contrast, LSTM maintains a "context" C_t and computes hidden states as

Long Short-Term Memory (LSTM)

- Essentially an RNN, except that the hidden states are computed differently
- Recall that RNN computes the hidden states as

$$\boldsymbol{h}_t = \tanh(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

- For RNN: State update is multiplicative (weak memory and gradient issues)
- In contrast, LSTM maintains a "context" C_t and computes hidden states as

$$\hat{C}_t = \tanh(\mathbf{W}^c\boldsymbol{x}_t + \mathbf{U}^c\boldsymbol{h}_{t-1}) \quad \text{("local" context, only up to immediately preceding state)}$$

Long Short-Term Memory (LSTM)

- Essentially an RNN, except that the hidden states are computed differently
- Recall that RNN computes the hidden states as

$$\boldsymbol{h}_t = \tanh(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

- For RNN: State update is multiplicative (weak memory and gradient issues)
- In contrast, LSTM maintains a "context" C_t and computes hidden states as

$$\begin{aligned} \hat{C}_t &= \tanh(\mathbf{W}^c\boldsymbol{x}_t + \mathbf{U}^c\boldsymbol{h}_{t-1}) && (\text{"local" context, only up to immediately preceding state}) \\ i_t &= \sigma(\mathbf{W}^i\boldsymbol{x}_t + \mathbf{U}^i\boldsymbol{h}_{t-1}) && (\text{how much to take in the local context}) \end{aligned}$$

Long Short-Term Memory (LSTM)

- Essentially an RNN, except that the hidden states are computed differently
- Recall that RNN computes the hidden states as

$$\boldsymbol{h}_t = \tanh(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

- For RNN: State update is multiplicative (weak memory and gradient issues)
- In contrast, LSTM maintains a "context" C_t and computes hidden states as

$$\begin{aligned}
\hat{C}_t &= \tanh(\mathbf{W}^c\boldsymbol{x}_t + \mathbf{U}^c\boldsymbol{h}_{t-1}) && (\text{"local" context, only up to immediately preceding state}) \\
{\color{blue}i_t} &= \sigma(\mathbf{W}^i\boldsymbol{x}_t + \mathbf{U}^i\boldsymbol{h}_{t-1}) && (\text{how much to take } {\color{blue}\text{in}} \text{ the local context}) \\
{\color{red}f_t} &= \sigma(\mathbf{W}^f\boldsymbol{x}_t + \mathbf{U}^f\boldsymbol{h}_{t-1}) && (\text{how much to } {\color{red}\text{forget}} \text{ the previous context})
\end{aligned}$$

Long Short-Term Memory (LSTM)

- Essentially an RNN, except that the hidden states are computed differently
- Recall that RNN computes the hidden states as

$$\boldsymbol{h}_t = \tanh(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

- For RNN: State update is multiplicative (weak memory and gradient issues)
- In contrast, LSTM maintains a "context" C_t and computes hidden states as

$$\begin{array}{rcll} \hat{C}_t & = & \tanh(\mathbf{W}^c\boldsymbol{x}_t + \mathbf{U}^c\boldsymbol{h}_{t-1}) & (\text{"local" context, only up to immediately preceding state}) \\ {\color{blue}i_t} & = & \sigma(\mathbf{W}^i\boldsymbol{x}_t + \mathbf{U}^i\boldsymbol{h}_{t-1}) & (\text{how much to take } {\color{blue}\text{in}} \text{ the local context}) \\ {\color{red}f_t} & = & \sigma(\mathbf{W}^f\boldsymbol{x}_t + \mathbf{U}^f\boldsymbol{h}_{t-1}) & (\text{how much to } {\color{red}\text{forget}} \text{ the previous context}) \\ {\color{green}o_t} & = & \sigma(\mathbf{W}^o\boldsymbol{x}_t + \mathbf{U}^o\boldsymbol{h}_{t-1}) & (\text{how much to } {\color{green}\text{output}}) \end{array}$$

Long Short-Term Memory (LSTM)

- Essentially an RNN, except that the hidden states are computed differently
- Recall that RNN computes the hidden states as

$$\boldsymbol{h}_t = \tanh(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

- For RNN: State update is multiplicative (weak memory and gradient issues)
- In contrast, LSTM maintains a "context" C_t and computes hidden states as

$$\begin{aligned}
\hat{C}_t &= \tanh(\mathbf{W}^c\boldsymbol{x}_t + \mathbf{U}^c\boldsymbol{h}_{t-1}) && \text{(“local” context, only up to immediately preceding state)}\\
{\color{blue}i_t} &= \sigma(\mathbf{W}^i\boldsymbol{x}_t + \mathbf{U}^i\boldsymbol{h}_{t-1}) && \text{(how much to take {\color{blue}in} the local context)}\\
{\color{red}f_t} &= \sigma(\mathbf{W}^f\boldsymbol{x}_t + \mathbf{U}^f\boldsymbol{h}_{t-1}) && \text{(how much to {\color{red}forget} the previous context)}\\
{\color{green}o_t} &= \sigma(\mathbf{W}^o\boldsymbol{x}_t + \mathbf{U}^o\boldsymbol{h}_{t-1}) && \text{(how much to {\color{green}output})}\\
C_t &= C_{t-1} \odot {\color{red}f_t} + \hat{C}_t \odot {\color{blue}i_t} && \text{(a modulated additive update for context)}
\end{aligned}$$

Long Short-Term Memory (LSTM)

- Essentially an RNN, except that the hidden states are computed differently
- Recall that RNN computes the hidden states as

$$\boldsymbol{h}_t = \tanh(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

- For RNN: State update is multiplicative (weak memory and gradient issues)
- In contrast, LSTM maintains a "context" C_t and computes hidden states as

$$\hat{C}_t = \tanh(\mathbf{W}^c\boldsymbol{x}_t + \mathbf{U}^c\boldsymbol{h}_{t-1}) \quad \text{("local" context, only up to immediately preceding state)}$$

$$i_t = \sigma(\mathbf{W}^i\boldsymbol{x}_t + \mathbf{U}^i\boldsymbol{h}_{t-1}) \quad \text{(how much to take in the local context)}$$

$$f_t = \sigma(\mathbf{W}^f\boldsymbol{x}_t + \mathbf{U}^f\boldsymbol{h}_{t-1}) \quad \text{(how much to forget the previous context)}$$

$$o_t = \sigma(\mathbf{W}^o\boldsymbol{x}_t + \mathbf{U}^o\boldsymbol{h}_{t-1}) \quad \text{(how much to output)}$$

$$C_t = C_{t-1} \odot f_t + \hat{C}_t \odot i_t \quad \text{(a modulated additive update for context)}$$

$$h_t = \tanh(C_t) \odot o_t \quad \text{(transform context into state and selectively output)}$$

Long Short-Term Memory (LSTM)

- Essentially an RNN, except that the hidden states are computed differently
- Recall that RNN computes the hidden states as
$$\boldsymbol{h}_t = \tanh(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$
- For RNN: State update is multiplicative (weak memory and gradient issues)
- In contrast, LSTM maintains a "context" C_t and computes hidden states as

$$
\begin{aligned}
\hat{C}_t &= \tanh(\mathbf{W}^c\boldsymbol{x}_t + \mathbf{U}^c\boldsymbol{h}_{t-1}) && \text{(``local'' context, only up to immediately preceding state)}\\
{\color{blue}i_t} &= \sigma(\mathbf{W}^i\boldsymbol{x}_t + \mathbf{U}^i\boldsymbol{h}_{t-1}) && \text{(how much to take {\color{blue}in} the local context)}\\
{\color{red}f_t} &= \sigma(\mathbf{W}^f\boldsymbol{x}_t + \mathbf{U}^f\boldsymbol{h}_{t-1}) && \text{(how much to {\color{red}forget} the previous context)}\\
{\color{green}o_t} &= \sigma(\mathbf{W}^o\boldsymbol{x}_t + \mathbf{U}^o\boldsymbol{h}_{t-1}) && \text{(how much to {\color{green}output})}\\
C_t &= C_{t-1} \odot {\color{red}f_t} + \hat{C}_t \odot {\color{blue}i_t} && \text{(a modulated additive update for context)}\\
h_t &= \tanh(C_t) \odot {\color{green}o_t} && \text{(transform context into state and selectively output)}
\end{aligned}
$$

- Note: $\odot$ represents elementwise vector product. Also, state updates now additive, not multiplicative. Training using backpropagation through time.

Long Short-Term Memory (LSTM)

- Essentially an RNN, except that the hidden states are computed differently
- Recall that RNN computes the hidden states as

$$\boldsymbol{h}_t = \tanh(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1})$$

- For RNN: State update is multiplicative (weak memory and gradient issues)
- In contrast, LSTM maintains a "context" C_t and computes hidden states as

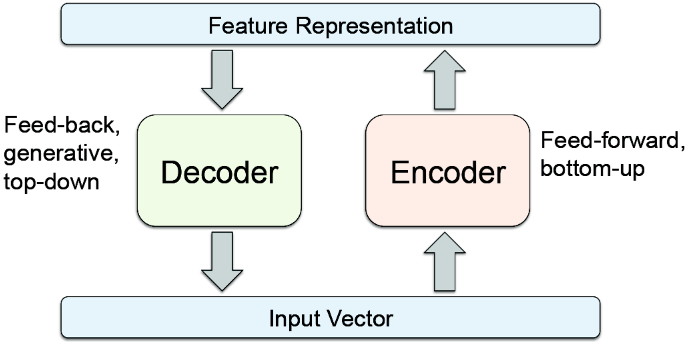
$$\begin{aligned}
\hat{C}_t &= \tanh(\mathbf{W}^c\boldsymbol{x}_t + \mathbf{U}^c\boldsymbol{h}_{t-1}) && \text{("local" context, only up to immediately preceding state)} \\
{\color{blue}i_t} &= \sigma(\mathbf{W}^i\boldsymbol{x}_t + \mathbf{U}^i\boldsymbol{h}_{t-1}) && \text{(how much to take {\color{blue}in} the local context)} \\
{\color{red}f_t} &= \sigma(\mathbf{W}^f\boldsymbol{x}_t + \mathbf{U}^f\boldsymbol{h}_{t-1}) && \text{(how much to {\color{red}forget} the previous context)} \\
{\color{green}o_t} &= \sigma(\mathbf{W}^o\boldsymbol{x}_t + \mathbf{U}^o\boldsymbol{h}_{t-1}) && \text{(how much to {\color{green}output})} \\
C_t &= C_{t-1} \odot {\color{red}f_t} + \hat{C}_t \odot {\color{blue}i_t} && \text{(a modulated additive update for context)} \\
h_t &= \tanh(C_t) \odot {\color{green}o_t} && \text{(transform context into state and selectively output)}
\end{aligned}$$

- Note: $\odot$ represents elementwise vector product. Also, state updates now additive, not multiplicative. Training using backpropagation through time.
- Many variants of LSTM exists, e.g., using C_{t-1} in local computations, Gated Recurrent Units (GRU), etc. Mostly minor variations of basic LSTM above

Neural Nets for Unsupervised Learning

Autoencoder

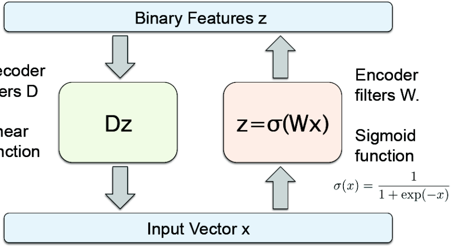
- A neural net for unsupervised feature extraction
- Basic principle: Learns an encoding of the inputs so as to recover the original input from the encodings as well as possible



- Also used to initialize deep learning models (layer-by-layer pre-training)

Autoencoder: An Example

- Real-valued inputs, binary-valued encodings



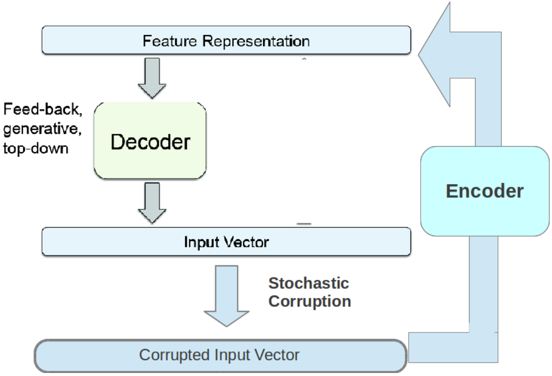
- Sigmoid encoder (parameter matrix W), linear decoder (parameter matrix D), learned via:

$$\arg\min_{D,W} E(D,W) = \sum_{n=1}^{N} ||D\mathbf{z}_n - \mathbf{x}_n||^2 = \sum_{n=1}^{N} ||D\sigma(W\mathbf{x}_n) - \mathbf{x}_n||^2$$

- If encoder is also linear, then autoencoder is equivalent to PCA

Denoising Autoencoders

- Idea: introduce stochastic corruption to the input; e.g.:
  - Hide some features
  - Add gaussian noise



Summary

- Looked at feedforward neural networks and extensions such as CNN

Summary

- Looked at feedforward neural networks and extensions such as CNN
- Looked at (deep) neural nets (RNN/LSTM) for learning from sequential data
  - Methods like RNN and LSTM are widely used for learning from such data
  - Modeling and retaining context is important when modeling sequential data (desirable to have a "memory module" of some sort as in LSTMs)

Summary

- Looked at feedforward neural networks and extensions such as CNN
- Looked at (deep) neural nets (RNN/LSTM) for learning from sequential data
  - Methods like RNN and LSTM are widely used for learning from such data
  - Modeling and retaining context is important when modeling sequential data (desirable to have a "memory module" of some sort as in LSTMs)
- Looked at Autoencoder - Neural network for unsupervised feature extraction

Summary

- Looked at feedforward neural networks and extensions such as CNN
- Looked at (deep) neural nets (RNN/LSTM) for learning from sequential data
    - Methods like RNN and LSTM are widely used for learning from such data
    - Modeling and retaining context is important when modeling sequential data (desirable to have a "memory module" of some sort as in LSTMs)
- Looked at Autoencoder - Neural network for unsupervised feature extraction
- Didn't discuss some other popular methods, e.g., deep generative models, but these are based on similar underlying principles