TTIC 31110
Speech Technologies

May 7, 2020

Question from last time

We defined a feedforward neural network as any vector function $f(\mathbf{x})$ of a vector input $\mathbf{x}$ that can be written as a composition of layers of a particular form:

$$
\begin{aligned}
f(\mathbf{x}) &= \mathbf{y}^L \\
\mathbf{y}^l &= \sigma_l(\mathbf{W}_l \mathbf{y}^{l-1} + \mathbf{b}^l) \\
\mathbf{y}^0 &= \mathbf{x}
\end{aligned}
$$

And we implied that σ_1 is always applied elementwise. But that is not true for the softmax activation function.

Outline

Hybrid models for speech recognition using recurrent neural networks

Recall: Hybrid HMM/NNs

Typical approach:

1. Train a frame-based discriminative classifier of sub-word units (e.g. phones, phone states, triphone states) given some labeled training data, $c^* = f_c(\mathbf{o})$, where c is the class and $\mathbf{o}$ is a frame feature vector
2. The output is a posterior probability $p(c|\mathbf{o})$
3. Convert $p(c|\mathbf{o})$ to something like an observation model (a "likelihood"): $p(\mathbf{o}|c) \propto \frac{p(c|\mathbf{o})}{p(c)}$
4. Use the result in place of the observation model in an HMM
5. Most popular type of frame classifier by far: neural network

Issues with DNNs/CNNs in hybrid models

- To handle acoustic context effects well, need to use large window of input around current frame $\Longrightarrow$ a lot of parameters!
- To handle phonetic context effects, we use context-dependent phonetic states $\Longrightarrow$ a lot of parameters!
- We need to do this because the NN operates on each frame independently: We “forget” the past

Issues with DNNs/CNNs in hybrid models

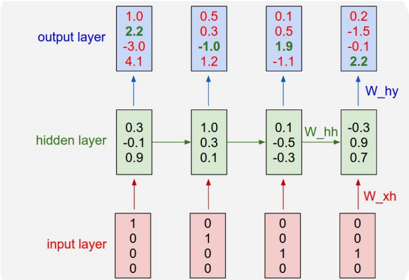
- To handle acoustic context effects well, need to use large window of input around current frame $\Longrightarrow$ a lot of parameters!
- To handle phonetic context effects, we use context-dependent phonetic states $\Longrightarrow$ a lot of parameters!
- We need to do this because the NN operates on each frame independently: We "forget" the past
- An alternative solution: recurrent neural networks (RNNs)
- RNNs have become ubiquitous in speech recognition, all other speech technologies, NLP and many other sequence processing tasks

Issues with DNNs/CNNs in hybrid models

- To handle acoustic context effects well, need to use large window of input around current frame $\Longrightarrow$ a lot of parameters!
- To handle phonetic context effects, we use context-dependent phonetic states $\Longrightarrow$ a lot of parameters!
- We need to do this because the NN operates on each frame independently: We "forget" the past
- An alternative solution: recurrent neural networks (RNNs)
- RNNs have become ubiquitous in speech recognition, all other speech technologies, NLP and many other sequence processing tasks
- (They may eventually get replaced with transformers)

Recurrent neural networks

An RNN is a neural network that maintains a state vector in each frame, i.e. "remembers" the past



In RNN acoustic model, input = acoustic frame, output = state posteriors

$$
\begin{aligned}
\mathbf{h}_t &= \sigma_h(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \\
\mathbf{y}_t &= \sigma_y(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y)
\end{aligned}
$$

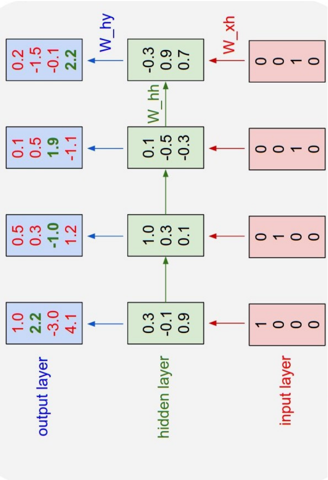
(All lower-case variables are vectors; upper-case are matrices)

Recurrent neural networks

Can think of RNNs as rotated feedforward neural networks...

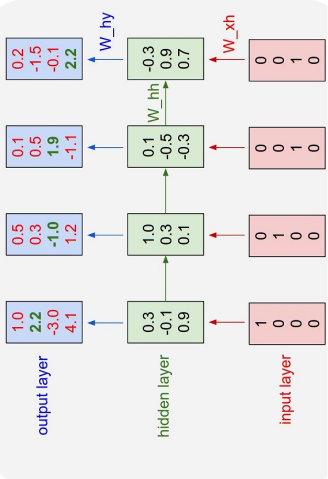
Recurrent neural networks

Can think of RNNs as rotated feedforward neural networks...



Recurrent neural networks

Can think of RNNs as rotated feedforward neural networks...



Except:

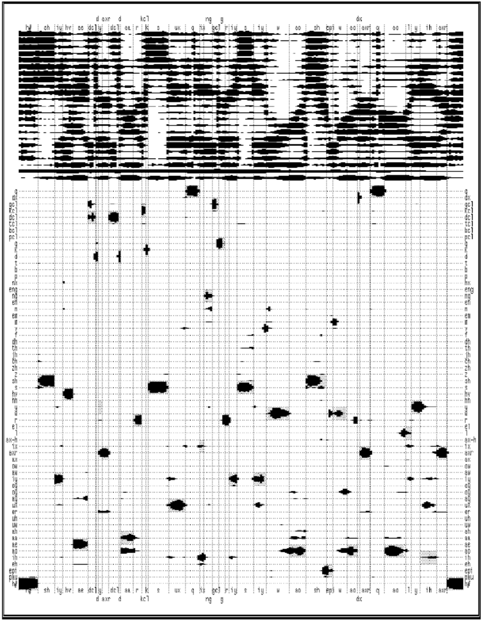
- "Layers" correspond to time steps
- Parameters (weights) are shared across "layers"
- Each "layer" has an additional input and output
- The number of layers is determined by the length (number of time steps) of the input

Using RNNs in hybrid HMM/NN models

If trained with cross-entropy loss, the outputs approximate posterior probabilities of labels (e.g., HMM states):

$$y_t^i \approx p(q_t = i | \mathbf{x}_1, \ldots, \mathbf{x}_t)$$

Example RNN phone posteriors



Robinson *et al.*,
"The use of recurrent neural networks in continuous speech recognition,"
in Lee *et al.*, eds., *Automatic Speech and Speaker Recognition: Advanced Topics*.
Kluwer Academic Publishers, 1995.

Using RNNs in hybrid HMM/NN models

If trained with cross-entropy loss, the outputs approximate posterior probabilities of labels (e.g., HMM states):

$$y_t^i \approx p(q_t = i | \mathbf{x}_1, \ldots, \mathbf{x}_t)$$

Using RNNs in hybrid HMM/NN models

If trained with cross-entropy loss, the outputs approximate posterior probabilities of labels (e.g., HMM states):

$$y_t^i \approx p(q_t = i | \mathbf{x}_1, \ldots, \mathbf{x}_t)$$

Given this we can attempt to scale as in HMM/DNNs:

$$\begin{aligned}
p(\mathbf{x}_{1:T} | q_{1:T}) &= \frac{p(q_{1:T} | \mathbf{x}_{1:T}) p(\mathbf{x}_{1:T})}{p(q_{1:T})} \\
&\approx \propto \prod_{t=1}^{t} \frac{p(q_t | \mathbf{x}_{1:T})}{p(q_t)} \\
&\propto \prod_{t=1}^{t} \frac{p(q_t | \mathbf{x}_{1:t})}{p(q_t)}
\end{aligned}$$

Using RNNs in hybrid HMM/NN models

If trained with cross-entropy loss, the outputs approximate posterior probabilities of labels (e.g., HMM states):

$$y_t^i \approx p(q_t = i | \mathbf{x}_1, \ldots, \mathbf{x}_t)$$

Given this we can attempt to scale as in HMM/DNNs:

$$\begin{aligned} p(\mathbf{x}_{1:T} | q_{1:T}) &= \frac{p(q_{1:T} | \mathbf{x}_{1:T}) p(\mathbf{x}_{1:T})}{p(q_{1:T})} \\ &\approx \propto \prod_{t=1}^{t} \frac{p(q_t | \mathbf{x}_{1:T})}{p(q_t)} \\ &\propto \prod_{t=1}^{t} \frac{p(q_t | \mathbf{x}_{1:t})}{p(q_t)} \end{aligned}$$

This is making an assumption that wasn't needed in HMM/DNNs! But we'll proceed anyway...

Training RNNs in hybrid HMM/NN models

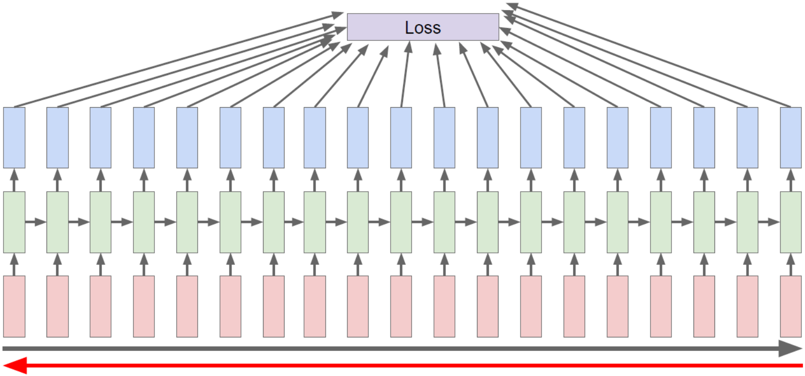
Typically trained with cross-entropy loss (log loss):

$$\begin{aligned}\ell_{CE} &= -\sum_t \sum_c y_{t,c} \log f_{t,c}(\mathbf{x}_{1:t}) \\ &= -\sum_t \log f_{t,c_t^*}(\mathbf{x}_{1:t})\end{aligned}$$

where:

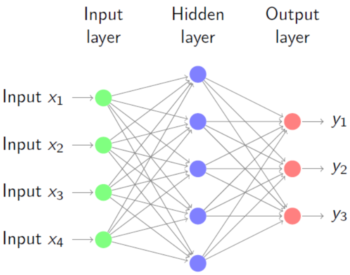
$\mathbf{x}_{1:t}$ is input sequence up to time t for one sequence in training set
$y_{t,c} = 1$ if ground-truth label at time $t = c$, 0 otherwise
$f_{t,c}(\mathbf{x}_{1:t})$ is our estimate of $p(c|\mathbf{x}_{1:t})$
c_t^* is ground-truth label at time t

Training RNNs



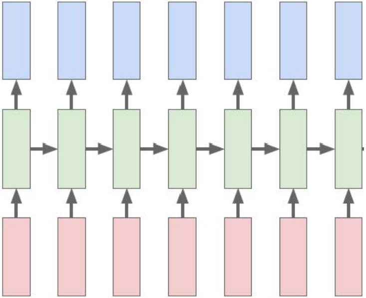
Recall: Gradient descent for feedforward NNs

- Computing gradients of loss with respect to each weight is done via backpropagation (chain rule)
- To compute gradient with respect to a lower-layer weight, we need gradient with respect to higher-layer outputs
- So backpropagation proceeds from the "top" (deepest) layer to the "bottom" (input) layer



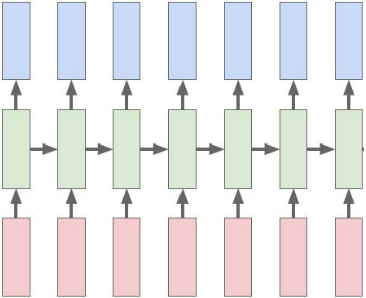
Gradient descent for RNNs: Backpropagation through time

For RNNs, somewhat like feedforward NNs but with some twists



Gradient descent for RNNs: Backpropagation through time

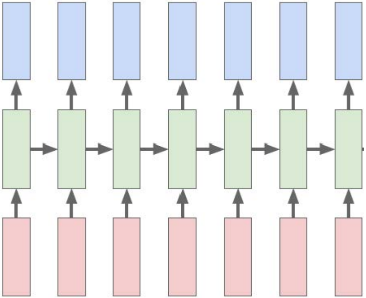
For RNNs, somewhat like feedforward NNs but with some twists



- For a given training sequence, RNN is like a feedforward NN with as many layers as time steps

Gradient descent for RNNs: Backpropagation through time

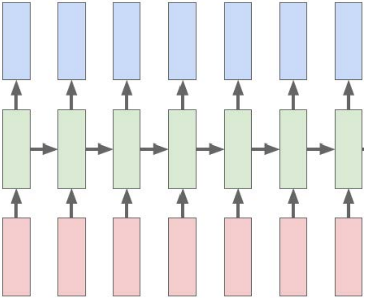
For RNNs, somewhat like feedforward NNs but with some twists



- For a given training sequence, RNN is like a feedforward NN with as many layers as time steps
- ... with lots of shared parameters

Gradient descent for RNNs: Backpropagation through time

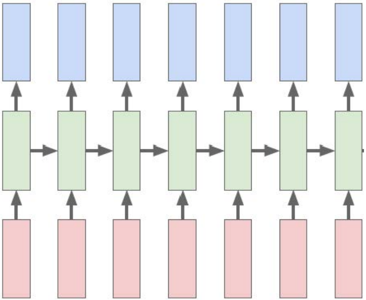
For RNNs, somewhat like feedforward NNs but with some twists



- For a given training sequence, RNN is like a feedforward NN with as many layers as time steps
- ... with lots of shared parameters
- ... and with an extra input vector and output vector per layer

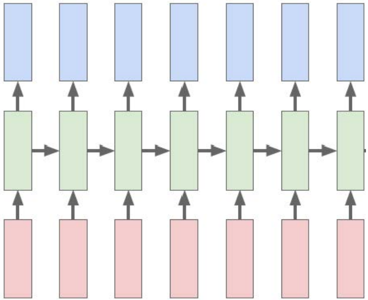
Backpropagation through time

For RNNs, somewhat like feedforward NNs but with some twists



Backpropagation through time

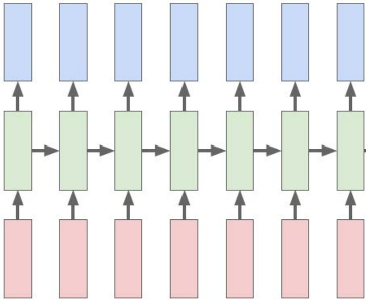
For RNNs, somewhat like feedforward NNs but with some twists



- Loss is a sum of losses over all time steps, e.g.
  $\ell_{CE} = -\sum_t \log f_{t,c_t*}(\mathbf{x}_{1:t})$

Backpropagation through time

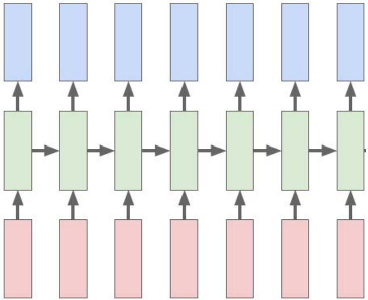
For RNNs, somewhat like feedforward NNs but with some twists



- Loss is a sum of losses over all time steps, e.g. $\ell_{CE} = -\sum_t \log f_{t,c_t*}(\mathbf{x}_{1:t})$
- So gradient w.r.t. a given weight is a sum of gradients of loss for a given time step $-\log f_{t,c_t^*}(\mathbf{x}_{1:t})$

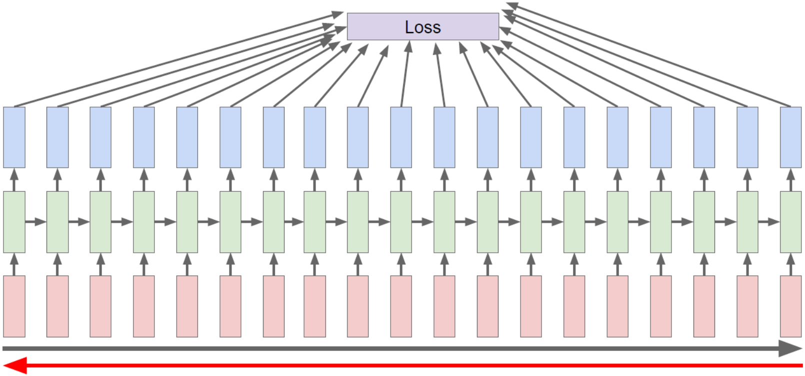
Backpropagation through time

For RNNs, somewhat like feedforward NNs but with some twists



- Loss is a sum of losses over all time steps, e.g. $\ell_{CE} = -\sum_t \log f_{t,c_t*}(\mathbf{x}_{1:t})$
- So gradient w.r.t. a given weight is a sum of gradients of loss for a given time step $-\log f_{t,c_t^*}(\mathbf{x}_{1:t})$
- For each time step, backpropagate through the "feedforward" network we've just imagined

Backpropagation through time (BPTT)



Truncated backpropagation through time

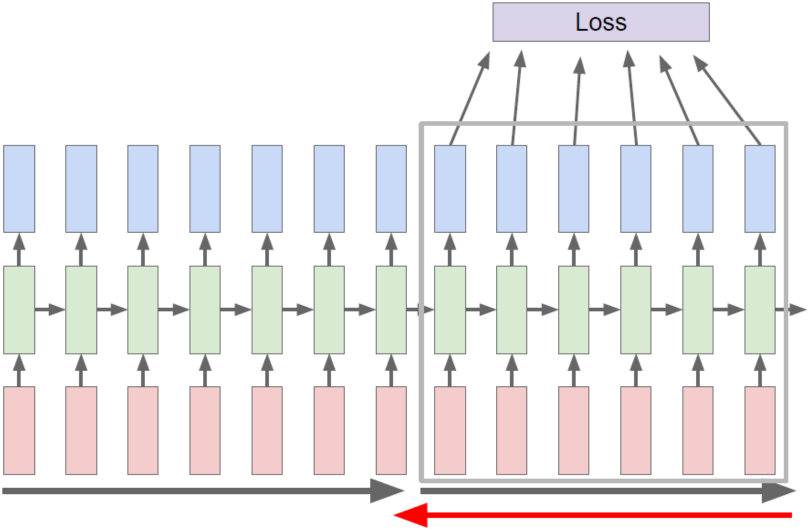
Backpropagating through large number of time steps can be computationally infeasible

Truncated backpropagation through time

Backpropagating through large number of time steps can be computationally infeasible

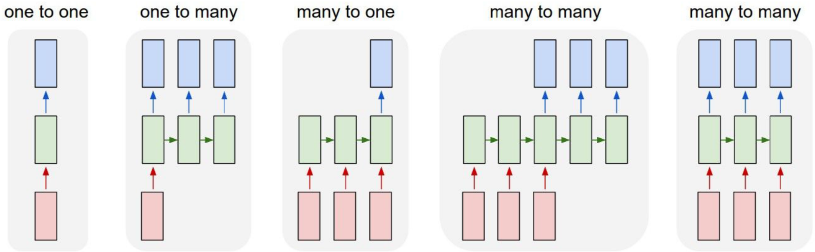
- Truncated BPTT: Divide up the sequence into (possibly overlapping) subsequences of length K
- Compute the activations as usual, but backpropagate only through the K steps of each subsequence

Truncated BPTT



A zoo of RNN structures

There are a number of other ways of using RNNs...

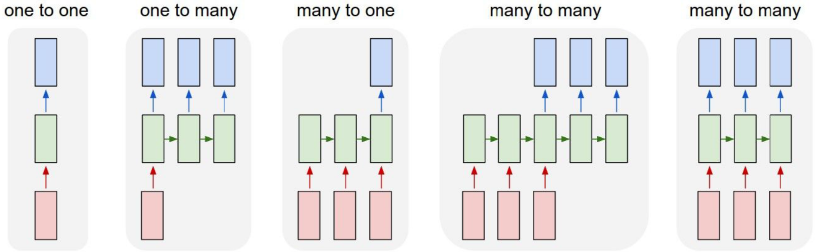


[Andrej Karpathy]

- The first network is non-recurrent
- The last is the type of RNN used in hybrid HMM/NNs

A zoo of RNN structures

There are a number of other ways of using RNNs...

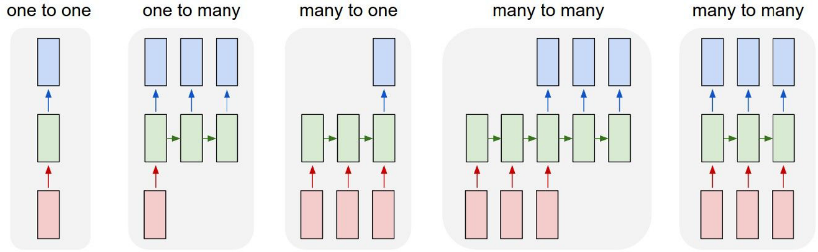


[Andrej Karpathy]

- One to many: E.g., image captioning (input = image, output = sequence of words)
- Many to one: Sequence classification (text sentiment classification, speaker identification, ...)

A zoo of RNN structures

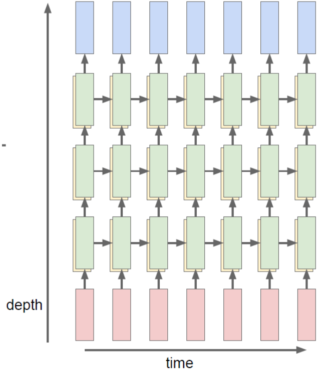
There are a number of other ways of using RNNs...



[Andrej Karpathy]

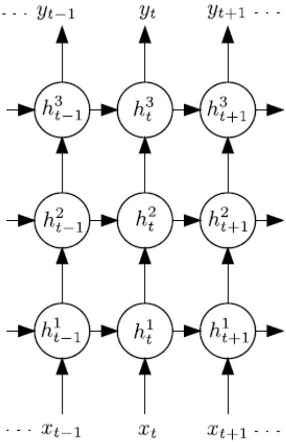
- Unaligned many to many = encoder-decoder RNN = "sequence-to-sequence" model
- Machine translation, speech recognition, ... (we'll talk about this soon)

Extensions: Deep recurrent neural networks



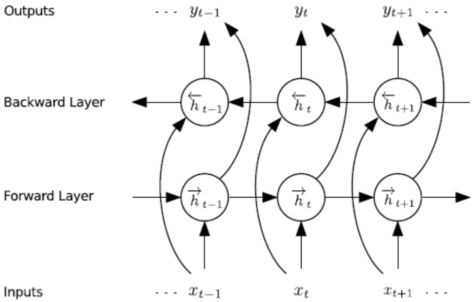
$$h_t^n = \sigma(W_{h^n h^{n-1}} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n)$$

Extensions: Deep recurrent neural networks



$$\mathbf{h}_t^n = \sigma(\mathbf{W}_{h^n h^{n-1}} \mathbf{h}_t^{n-1} + \mathbf{W}_{h^n h^n} \mathbf{h}_{t-1}^n + \mathbf{b}_h^n)$$

Extensions: Bidirectional RNNs



$$\mathbf{y}_t \quad = \quad \sigma(\mathbf{W}_{\overrightarrow{h}y}\overrightarrow{\mathbf{h}}_t + \mathbf{W}_{\overleftarrow{h}y}\overleftarrow{\mathbf{h}}_t + \mathbf{b}_y)$$

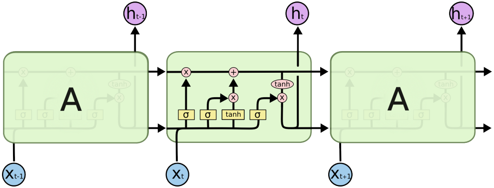
Training RNNs can be very challenging...

- Backpropagation involves many multiplications of the same weight matrix
- Depending on the matrix value at a given iteration, this can lead to either exploding or vanishing gradients
- Exploding gradients often avoided through **gradient clipping**
- For vanishing gradients, a modification of RNNs is typically used

Extensions: Long short-term memory networks

[Hochreiter+ 1997]

LSTMs: A way of selectively "remembering" and "forgetting"

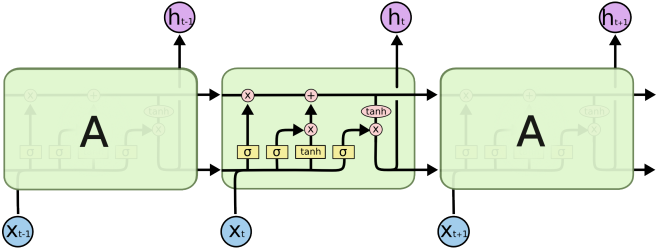


$$
\begin{aligned}
h_t &= o_t \odot \tanh(c_t), \quad \text{where} \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad \text{(output gate)} \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad \text{(cell memory)} \\
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad \text{(input gate)} \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad \text{(forget gate)}
\end{aligned}
$$

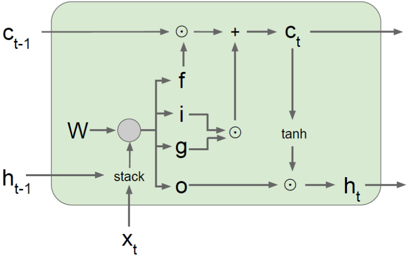
These are the hidden state, output gate, cell activation, input gate, and forget gate

Extensions: LSTMs (2)

There are many variants of LSTMs...



LSTMs



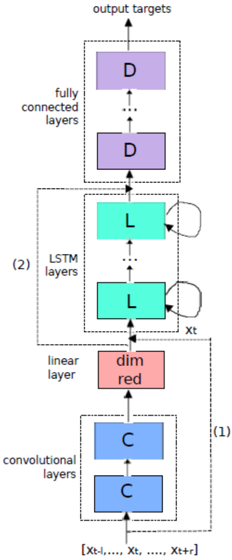
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Other types of gated RNNs: gated recurrent units (GRUs), LSTM variants

Extensions: Google's "CLDNN"



Practical observations about hybrid HMM/NNs

Some of the details seem to have limited practical impact...

- Scaling by the state priors
- HMM transition probabilities (why?)

Practical observations about hybrid HMM/NNs

Some of the details seem to have limited practical impact...

- Scaling by the state priors
- HMM transition probabilities (why?)
  - Typical range of continuous densities is much larger than probabilities of discrete transitions
  - (Inaccurate) conditional independence assumption of HMMs

$\Longrightarrow$ The NN classifier is doing most of the work, while the HMM is largely enforcing that only allowed state sequences are hypothesized

End-to-end RNNs for speech recognition

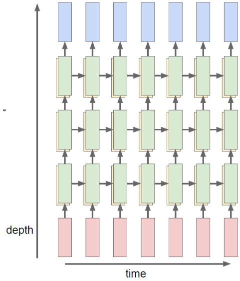
Hybrid models involve a lot of machinery...

- Train a simple HMM/GMM recognizer
- Run Viterbi with HMM/GMM to get per-frame state labels
- Train neural network frame classifier
- Scale classifier outputs by state priors to get scaled HMM observation model
- Train HMM/NN
- And some of these steps are not completely necessary, but it's not clear why...

Can we train an RNN to map directly from acoustic input sequence to output text sequence?

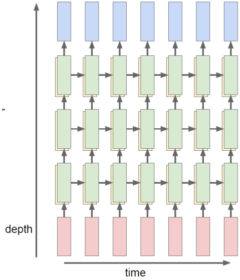
End-to-end RNNs for speech recognition

We could train an RNN to output a text label (word, character) at each frame



End-to-end RNNs for speech recognition

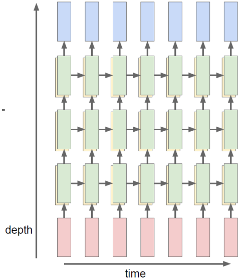
We could train an RNN to output a text label (word, character) at each frame



- But what does this mean? Words and characters usually span many frames of speech

End-to-end RNNs for speech recognition

We could train an RNN to output a text label (word, character) at each frame



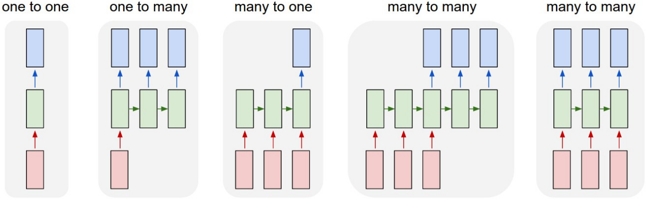
- But what does this mean? Words and characters usually span many frames of speech
- And the alignment between frames and characters is not simple or even monotonic

End-to-end RNNs for speech recognition

Two typical approaches:

- Encoder-decoder ("sequence-to-sequence") models
- Connectionist temporal classification (CTC)

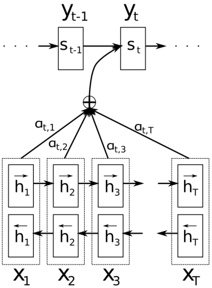
Encoder-decoder RNNs ("sequence-to-sequence")



- Can be trained directly to optimize $p(y|x)$ without aligning the input and output
- Input (acoustic frame) and output (characters) don't have to operate at the same rate!
- Introduced for machine translation [Cho+ 2014, Sutskever+ 2014]

Attention models

Basic encoder-decoder models must represent the entire input with a single vector



- In attention models, each decoder state depends on a weighted combination of encoder states (a "context vector")
- These weights are an "attention vector"
- The attention vector is itself a function of the input and output, with learned parameters

Summary so far

- RNNs are very popular in both hybrid HMM/NN and end-to-end models
- RNN training is typically done via (truncated) backpropagation through time
- LSTMs (or other gating mechanisms) are used to avoid vanishing gradients
- End-to-end RNNs for speech tasks are typically either attention encoder-decoders or connectionist temporal classification-based