TTIC 31110
Speech Technologies

May 12, 2020

Announcements

- HW4 due Monday 5/25 7pm
- Term project timeline, guidelines
  - Carefully read the guidelines under the "Term project" module
  - Let us know if you need help with compute resources

Outline

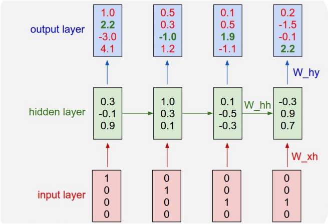
End-to-end recurrent neural networks (RNN) models for speech recognition

Attention models

Connectionist temporal classification (CTC)

Recap: RNNs

A network that maintains a state vector in each frame, i.e. "remembers" the past



[Andrej Karpathy]

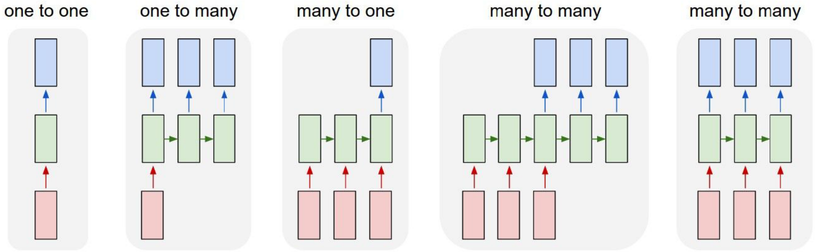
$$\begin{aligned} \mathbf{h}_t &= \sigma_h(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \\ \mathbf{f}_t &= \sigma_y(\mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y) \end{aligned}$$

In hybrid HMM/NN, **input** = acoustics, **output** = state posteriors

- σ_y is a softmax function so as to output a distribution over classes (HMM states)

A zoo of RNN structures

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

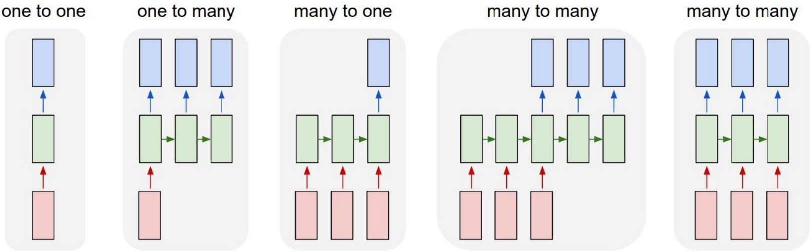


[Andrej Karpathy]

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

A zoo of RNN structures

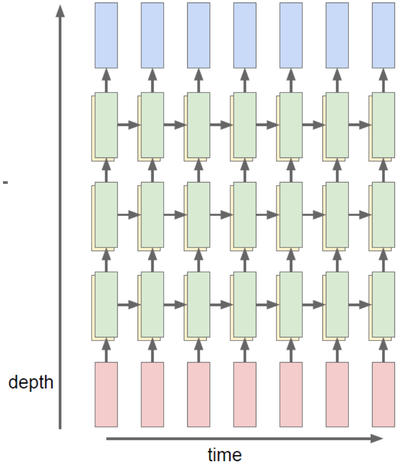
There are a number of other ways of using RNNs...



[Andrej Karpathy]

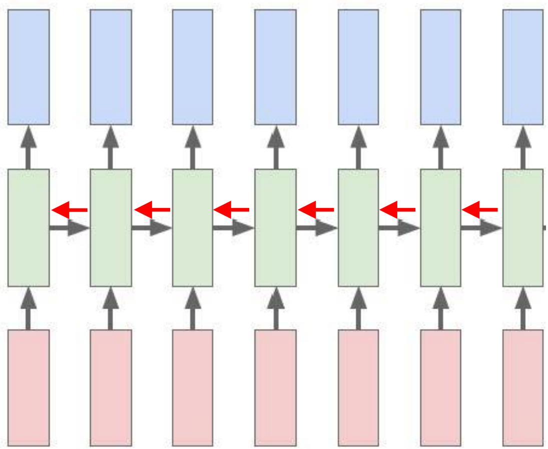
- Unaligned many to many = encoder-decoder RNN = "sequence-to-sequence" model
- Machine translation, speech recognition, ... (we'll talk about this soon)

Recap: Deep recurrent neural networks



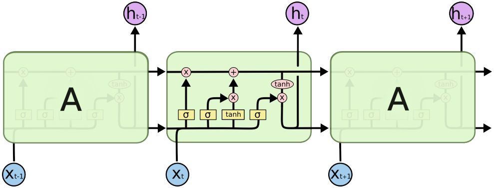
$$\mathbf{h}_t^n = \sigma(\mathbf{W}_{h^n h^{n-1}} \mathbf{h}_t^{n-1} + \mathbf{W}_{h^n h^n} \mathbf{h}_{t-1}^n + \mathbf{b}_h^n)$$

Recap: Bidirectional RNNs



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

Recap: Long short-term memory RNNs



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

End-to-end RNNs for speech recognition

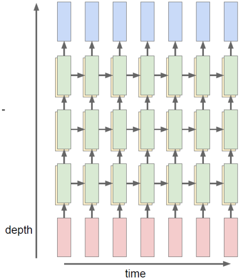
Hybrid HMM/NN models involve a lot of machinery...

- 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

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



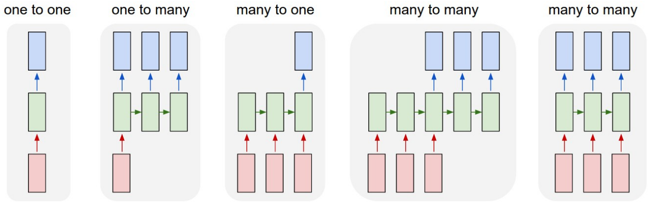
- But what does this mean? Words and characters usually span many frames of speech
- And the alignment between frames and characters is not simple or even monotonic

End-to-end RNNs for speech recognition

Two typical approaches:

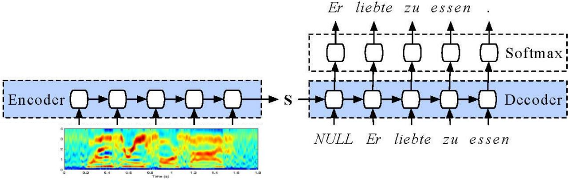
- Encoder-decoder ("sequence-to-sequence") models [Bahdanau+ 2015]
- Connectionist temporal classification (CTC) [Graves+ 2006]

Encoder-decoder RNNs (“sequence-to-sequence”)



- Can be trained directly to optimize a loss on y 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]

Encoder-decoder RNNs in more detail

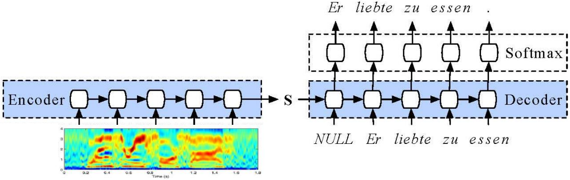


[smerity.com]

"Vanilla" RNN encoder-decoder equations:

$$\begin{aligned}
&\text{Encoder}: \\
\mathbf{h}_t &= \sigma_h(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \\
\mathbf{s} &= \mathbf{h}_T = \mathbf{s}_0 \\
&\text{Decoder}: \\
\mathbf{s}_j &= \sigma_s(\mathbf{W}_{ys}\mathbf{y}_{j-1} + \mathbf{W}_{ss}\mathbf{s}_{j-1} + \mathbf{b}_s) \\
\mathbf{f}_j &= \mathrm{softmax}(\mathbf{W}_{sy}\mathbf{s}_j + \mathbf{b}_y) \\
\hat{y}_j &= \mathrm{argmax}\, \mathbf{f}_j
\end{aligned}$$

Encoder-decoder RNNs in more detail

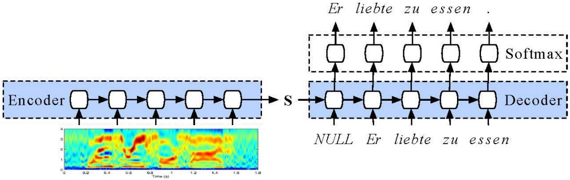


"Vanilla" RNN encoder-decoder equations:

$$
\begin{aligned}
&\text{Decoder}: \\
&\mathbf{s}_j = \sigma_s(\mathbf{W}_{ys}\mathbf{y}_{j-1} + \mathbf{W}_{ss}\mathbf{s}_{j-1} + \mathbf{b}_s) \\
&\mathbf{f}_j = \text{softmax}(\mathbf{W}_{sy}\mathbf{s}_j + \mathbf{b}_y) \\
&\hat{y}_j = \text{argmax}\, \mathbf{f}_j
\end{aligned}
$$

- Here $\mathbf{y}_j$ is a "one-hot" vector representing $\hat{y}_j$
- Interpretation: f_{jd} is the probability of the next word being the d^{th} word in the vocabulary, given the previous words and the acoustic input

Encoder-decoder RNNs in more detail

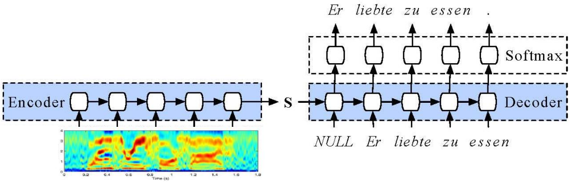


"Vanilla" RNN encoder-decoder equations:

$$
\begin{aligned}
&\text{Decoder :} \\
\mathbf{s}_j &= \sigma_s(\mathbf{W}_{ys}\mathbf{y}_{j-1} + \mathbf{W}_{ss}\mathbf{s}_{j-1} + \mathbf{b}_s) \\
\mathbf{f}_j &= \text{softmax}(\mathbf{W}_{sy}\mathbf{s}_j + \mathbf{b}_y) \\
\hat{y}_j &= \text{argmax}\,\mathbf{f}_j
\end{aligned}
$$

- Typical loss: cross-entropy (log loss) $-\log p(y_{1:J}|\mathbf{x}_{1:T}) = -\sum_{j=1}^{J} \log f_{jd}$
- where $y_{1:J}$ = ground-truth label sequence corresponding to input sequence $\mathbf{x}_{1:T}$ and d is the index of y_j in the vocabulary

Encoder-decoder RNNs in more detail

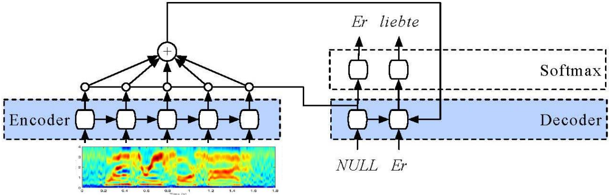


Can be extended to a variety of types of encoder and decoder RNNs

- LSTM/GRU instead of vanilla RNN units
- Deep encoder, deep decoder (less typical)
- Bidirectional encoder

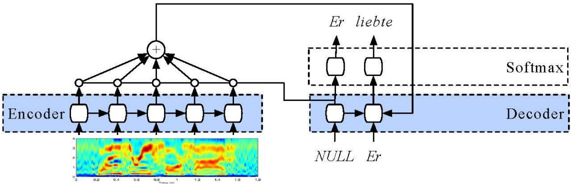
Attention models

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



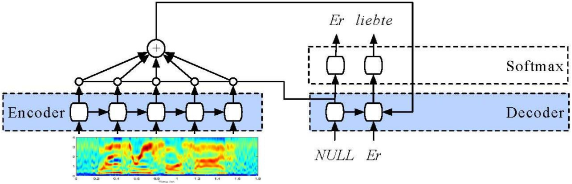
- In attention models, each decoder state depends on a weighted combination of encoder states (a "context vector")
- These weights are an "attention vector"
- The attention vector is itself a function of the input and output, with learned parameters

Attention models: Example



$$
\begin{aligned}
\text{Decoder}: & & \\
\mathbf{s}_j &= \sigma_s(\mathbf{W}_{ys}\mathbf{y}_{j-1} + \mathbf{W}_{ss}\mathbf{s}_{j-1} + \mathbf{b}_s) \\
\mathbf{f}_j &= \mathrm{softmax}(\mathbf{W}_{sy}[\mathbf{c}_j; \mathbf{s}_j] + \mathbf{b}_y) \\
\hat{y}_j &= \mathrm{argmax}\, \mathbf{f}_j \\
\text{Context vector}: & & \\
\mathbf{c}_j &= \sum_{t=1}^{T} \alpha_{jt}\mathbf{h}_t \\
\alpha_j &= \mathrm{softmax}(\mathbf{u}_j) \\
u_{jt} &= \mathbf{h}_t^T \mathbf{s}_j
\end{aligned}
$$

Attention models: Other types



$$
\begin{aligned}
\mathbf{c}_j &= \sum_{t=1}^{T} \alpha_{jt}\mathbf{h}_t \\
\alpha_j &= \mathrm{softmax}(\mathbf{u}_j) \\
\mathbf{u}_{jt} &= \mathbf{h}_t^T \mathbf{s}_j \\
\text{OR } \mathbf{u}_{jt} &= \mathbf{v}^T \tanh(\mathbf{W}_1\mathbf{h}_t + \mathbf{W}_2\mathbf{s}_j + \mathbf{b}_a) \\
\text{OR } \mathbf{u}_{jt} &= \mathbf{v}^T \tanh(\mathbf{W}_1\mathbf{h}_t + \mathbf{W}_2\mathbf{s}_j + \mathbf{W}_f\mathbf{f}_{jt} + \mathbf{b}_a) \\
\text{where } \mathbf{f}_j &= \mathbf{F} * \alpha_{j-1} \text{ and } \mathbf{F} \text{ a learned filter matrix}
\end{aligned}
$$

(The last is sometimes called “location-aware” or “convolutional” attention)

Scheduled sampling [Bengio+ 2015]

During training, what should the input $\mathbf{y}_j$ to the decoder be?

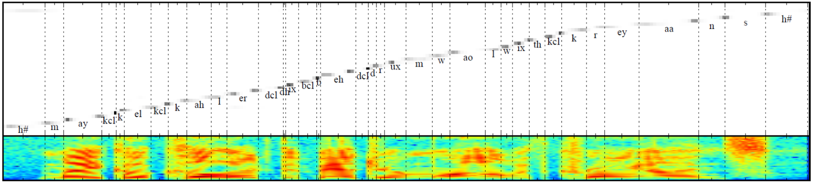
- To match test-time model, should set $\mathbf{y}_j$ to a one-hot vector representing $\hat{y}_{j-1}$
- ... but then the training model would produce garbage output until it is trained well
- Or, we could set $\mathbf{y}_j$ to a one-hot vector representing the ground-truth y_{j-1}
- ... but then we are training and testing with different models
- Scheduled sampling: At each iteration of training, use the ground-truth label with some probability ϵ and the model's previous prediction with probability $1-\epsilon$

Output labels

- Words
- Characters (graphemes)
- Some other sub-word unit? (e.g., "word-pieces")
- The shorter the unit, the larger the vocabulary that can be represented
- ... but the less "memory" the decoder gets to use
- ... and the longer it takes to train

Visualizing attention

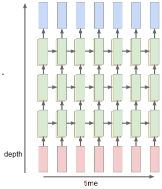
FDHC0_SX209: Michael colored the bedroom wall with crayons.



The alignment between the acoustics and labels is largely monotonic, so maybe attention models are overkill?

Rewind: End-to-end RNNs for speech recognition

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



- But each word/character usually spans many frames of speech
- And the alignment between frames and characters is ambiguous. Consider the word **through**. Which frames does the **g** correspond to?
- And we are not usually given frame labels
- But given the mostly monotonic alignment between frames and labels, maybe we were too dismissive?

Connectionist temporal classification (CTC)

[Graves+ 2006]

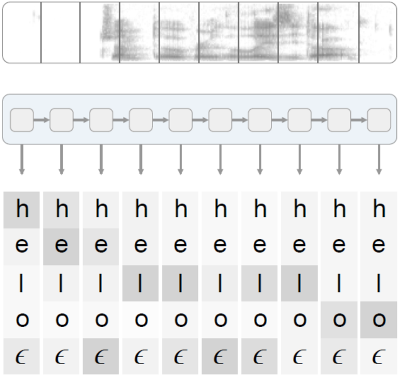
CTC modifies the per-frame RNN labeler idea with two key things:

- An extra "blank" label ϵ
- A mapping from frame-level label sequences to true label sequences

Connectionist temporal classification (CTC)

[Graves+ 2006]

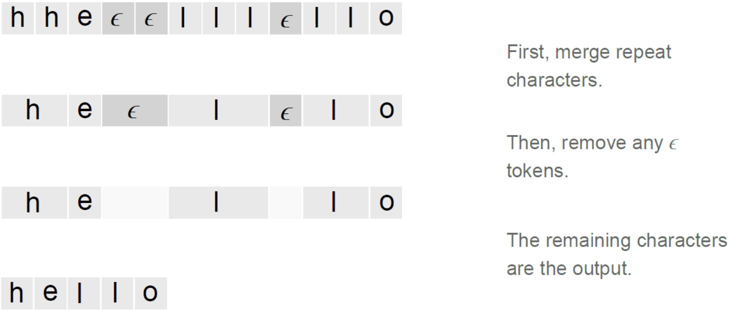
From https://distill.pub/2017/ctc/:



RNN with softmax output layer produces a posterior probability for each label + ϵ

Basic ("greedy") CTC decoding

- RNN with softmax output layer produces a posterior probability for each label + ϵ
- At each time frame, output the most likely frame label
- Finally, map frame labels to "collapsed" label sequence as follows:



CTC training

Given a sequence X of T acoustic frames and a corresponding label sequence Y with $L < T$ labels, e.g. the word **cat**, consider the set of all of the valid frame label sequences ("alignments") $\mathcal{A}_{X,Y}$

Valid Alignments	Invalid Alignments	
ϵ c c ϵ a t	c ϵ c ϵ a t	corresponds to $Y =$ [c, c, a, t]
c c a a t t	c c a a t	has length 5
c a ϵ ϵ ϵ t	c ϵ ϵ ϵ t t	missing the 'a'

CTC training

Given a sequence X of T acoustic frames and a corresponding label sequence Y with $L < T$ labels, e.g. the word **cat**, consider the set of all of the valid frame label sequences ("alignments") $\mathcal{A}_{X,Y}$.

Then the CTC loss is a *marginal log loss*:

$$-\log p(Y|X) = -\log \sum_{A \in \mathcal{A}_{X,Y}} \prod_{t=1}^{T} p(a_t|X)$$

where $p(a_t|X)$ is the softmax output of the RNN at frame t

CTC training

CTC loss:
$-\log p(Y|X) = -\log \sum_{A \in \mathcal{A}_{X,Y}} \prod_{t=1}^{T} p(a_t|X)$

Looks hard to backprop, but it turns out to be equivalent to a forward-backward-like algorithm!