Bidirectional RNN

Very → how are you?
well → good, etc.

The future can provide useful information as well.

Standard RNN

bidirectional RNN
Bidirectional RNNs predict the current output $y_t$ using the past and the future:

$$\tilde{h}_t = f(x_t, h_{t-1} ; \tilde{W}_h)$$

$$\tilde{h}_t = f(x_t, h_{t+1} ; \tilde{W}_h)$$

$$h_t = \begin{bmatrix} \tilde{h}_t \\ \tilde{h}_t \end{bmatrix}$$

$$g_t = g(h_t, \omega_0)$$

Useful for non-causal task where context is useful, e.g. NLP tagging task (NER, POS, etc.), change point detection in time series, translation.

BTT needs to be modified to account for future dependencies.
Deep RNNs (typically used when "LSTMs" or "RNNs" are mentioned)

- train layer-wise, by computing all outputs from layer $d$ so have inputs for layer $d+1$
- modify BIT to account for dependencies between layers
Sequence-to-Sequence Modeling (2014 used for very good machine translation, popularized deep LSTM seq-to-seq model)

Sutskever et al. “Sequence to Sequence Learning with Neural Networks”

Goal: given input sequence of tokens \((x_1, \ldots, x_T)\)

predict output sequence of tokens \((y_1, \ldots, y_{T'})\)

Canonical example: Machine Translation

Issue: \(T\) may be different than \(T'\) so the many-to-many design paradigm for RNNs may not fit

Soln: encoder-decoder architecture

*many-to-one* RNN

*one-to-many* RNN
Sequence-to-Sequence architecture

Encoder

Decoder

(START)

Very

Well

and

you

<END>

Very

Well

and

you

(teacher forcing > only during training)
Training:

- train the encoder & decoder simultaneously (using BPTT) so that given the input sequence $(x_{1:7} \ldots \rightarrow x_T)$ and output sequence $(y_{1:7} \ldots \rightarrow y_{T-1})$:
  
  - encoder learns an informative $h_T$ for decoding
  
  - decoder learns how to, given $h_T$ and a special $<$START$>$ token, generate the correct output followed by an $<$END$>$ token.

- Since the output $\hat{y}_t$ depends on $\hat{y}_{t-1}$ we can have very slow training if $\hat{y}_t$ is inaccurate at the start of the sequence, because the errors propagate. Instead, use Teacher forcing: use $y_{t-1}$ (the ground truth) as the input to predict $\hat{y}_t$ at training time.
As our main inputs: \((X_1, \ldots, X_T)\) and \((\text{START}, y_1, \ldots, y_T')\)

and as training outputs: \((y_1, \ldots, y_T', \text{END})\)
Inference

- Given input sequence \((x_1, \ldots, x_T)\) use encoder to obtain \(h_T\)
- Initialize the decoder with hidden state \(h_T\) and feed in the \(\text{<START>}\) token
- At each time, feed \(y_{t-1}\) into the decoder to predict the next token and take \(y_t\) to be the most likely token
Details

- use deep LSTMs (GRUs) in the encoder and decoder
- reverse the order of the source sequence to introduce shorter length dependencies w/ the target sequence

Muy bien y to Very well and you

to y bien Muy Very well and you

- use "beam decoding" in practice (see Sutskever et al.)
Recap course:
- basic prob theory & optim theory needed for ML
- several stochastic first-order algos for ML (SGD, AdaGrad, Adam, Rmsprop)
- basic convex machine learning models (esp. logistic regression)
- practice w/ implementing these algos & seeing impact of hyperparam choices
- kernel learning for for nonlinear ML w/ potentially infinite data features
- fully nonlinear ML using NNs (tradeoffs inherent in NNs: no global minimum, hyperparam size is larger issue, training time)
- Different NN architectures for different problem classes:
  - Fully connected (general probes); autoencoders
  - CNNs for vision; several popular architectures
  - RNNs for sequential data
- Mitigation strategies for exploding & vanishing gradients
Topics we didn't cover:

- Gaussian Processes
- attention & transformers (canonical architecture) for NLP
- graph NNs
- generative models (GANs and VAEs)
- deep RL
- online learning > bandits