CS 182 Lecture 10: Recurrent Neural Networks

6/30

applications: classifying sentiment, recognizing phonemes, classifying video activity

dealing with variable-size inputs → zero-pad (simple)

one input per layer

$z^\ell = W^\ell \bar{a}^{\ell-1} + b^\ell \quad (\bar{a}^{\ell-1} = \begin{bmatrix} a^{\ell-1} \\ x_{i,t} \end{bmatrix})$ ↳ each layer: activations of prev. layer, inputs to layer.

have a "pretend layer" w/ $a^\ell = 0$ to start chain.

RNN: W^ℓ is the same for all layers (and b^ℓ). ↳ parameter-sharing; "extends a standard NN along the time dimension".

Minor backprop change: instead of "overwriting" gradients at ℓ for $\ell+1$, ADD to loss. $d\ell/dO_f \mathrel{+}= df/dO_f$ &.

analogous to function chaining → AKA "accumulate" the gradient.

applications: generating text caption for image; predicting sequence of future video frames; generating audio sequence

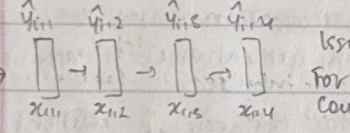
dealing with variable-size outputs → one output per layer ↙ "decoder"

→ separate loss for every layer $z^\ell = W^\ell a^{\ell-1} + b^\ell$, $a^\ell = \sigma(z^\ell)$ → $\hat{y}_\ell = f(a^\ell)$. e.g. simply linear + softmax

Reminder: $\delta_{xf} = \frac{df}{dx_f} = \frac{df}{dx}\delta_{yf}$ $(\delta_{yf} = \frac{df}{dy_f})$ $\quad \mathcal{L}(\hat{y}_1:\tau) = \sum_\ell \mathcal{L}_\ell(\hat{y}_\ell)$.

For variable-size output: $\delta_{yf} = \delta'_{yf} + \delta^2_{yf}$ (for nodes with >1 descendents (separate losses, all but final layer activations))

Input and outputs at each step? combine the two concepts.



Issue: RNNs are super deep, essentially, unable to retain memory. For lots of layers, can run into vanishing gradient problem. Could also have exploding gradient, could fix w/ clipping.

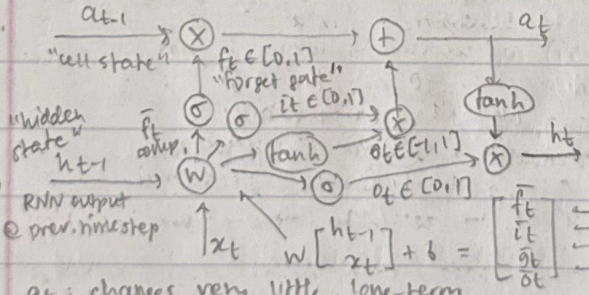
"RNN dynamics" $\quad \bar{a}_{t-1} = \begin{bmatrix} a_{t-1} \\ x_t \end{bmatrix}$

$a_t = g(a_{t-1}, x_t)$ where $z_t = W\bar{a}_{t-1} + b$ $\quad a_t = \sigma(z_t)$ w/ Jacobian: $\frac{da}{da_{t-1}}$, should $\approx I$. (to remember)

to "remember" → copy previous activation

to "forget" → overwrite it w/ current input.

to and behold, LSTM. (long short-term memory)



$a_t = a_{t-1} f_t + g_t$ so $\frac{da_i}{da_{t-1,i}} = f_t \in [0,1]$

trying to preserve linearity of a_t through "scapegoating" h_t with nonlinearities. Therefore we maintain

a_t: changes very little, long-term.

h_t: changes all the time, short-term.

GRU: LSTM w/o $\bar{o}_t$.

takes longer to tune hyperparameters

most RNNs in practice look like this, because there are strong dependencies b/w outputs (like in text generation). Yesterday's output → today's input!

Training ideas → outputs are shifted inputs (as "tokens"). Earlier mistakes can compound...

Can use scheduled sampling to regularize (model can see its own inputs and can act more reasonably).

Different ways to "use" RNNs?

one → one

many → many → stackable → can incorporate convolutions → can be bidirectional (e.g. speech recognition).

RNN encoders → process inputs prior to feeding them RNN layers (ex: images).

RNN decoders → process outputs right after feeding them RNN layers (less common)

Cool examples:
- → Shakespeare text gen
- → Fake LaTeX gen for math
- → OpenAI GPT-2 gen text (using Transformer).