Graves 2013, "Generating Sequences with Recurrent Neural Networks"

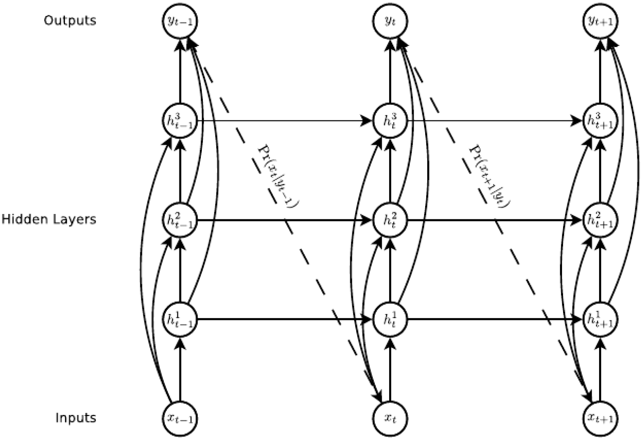
Johannes Bausch and Jack Kamm

14 November 2017

Outline

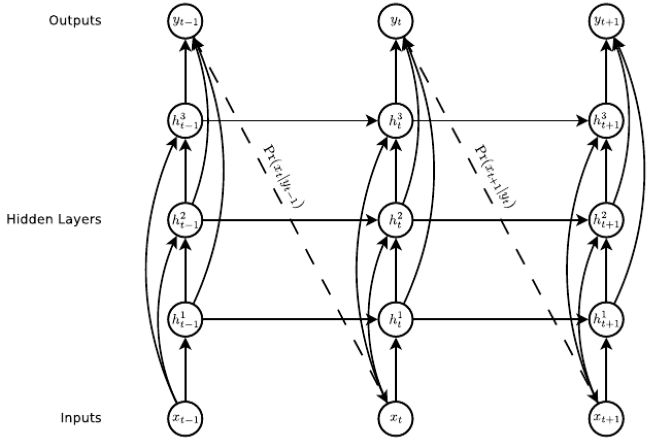
Table of Contents

Recurrent neural networks



Input x_t, hidden layers h_t^n, output y_t, generative model $\mathbb{P}(x_{t+1} \mid y_t)$

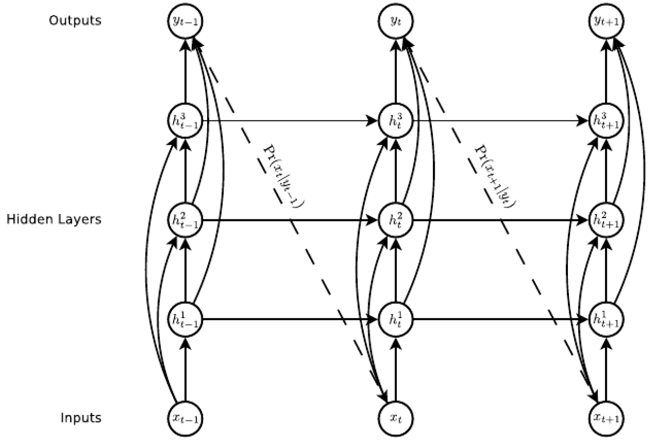
Recurrent neural networks



h_t^n = nonlinear link $\circ$ affine combo of x_t, h_{t-1}^n, h_t^{n-1}

$$h_t^n = \mathcal{H}(W_{ih^n} x_t + W_{h^{n-1}h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n)$$

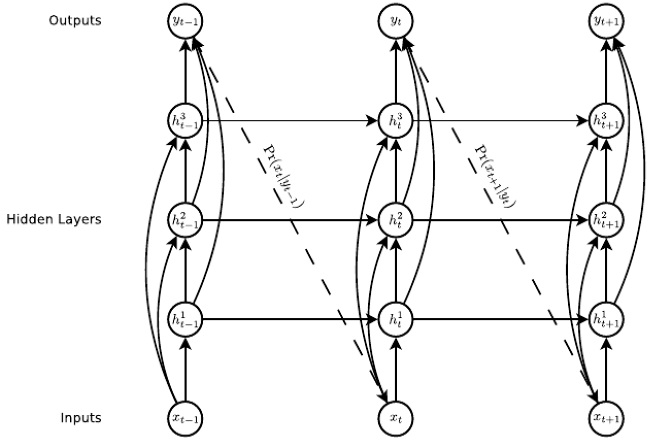
Recurrent neural networks



y_t = nonlinear link ∘ affine combo of h_t^n

$$y_t = \mathcal{Y}(b_y + \sum_{n=1}^{N} W_{h^n y} h_t^n)$$

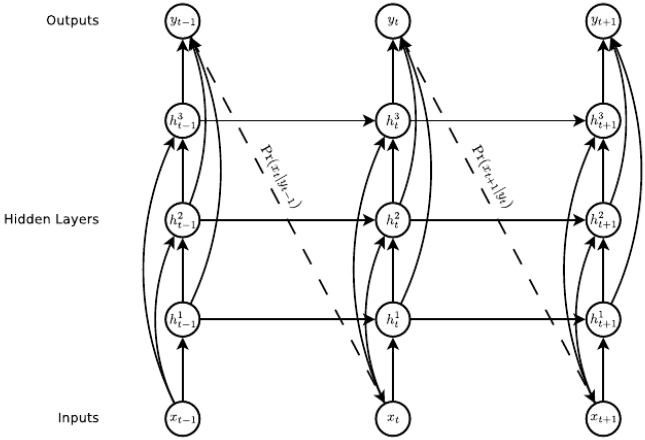
Recurrent neural networks



Train by maximizing likelihood of generative model:

$$\mathbb{P}(\mathbf{x}) = \prod_{t=1}^{T} \mathbb{P}(x_t \mid y_{t-1})$$

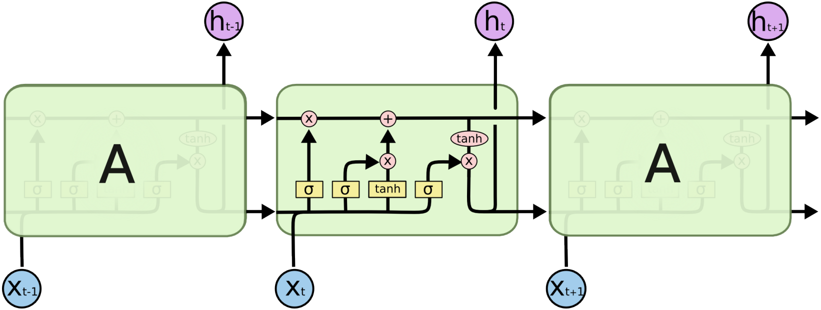
Recurrent neural networks



Compute $\nabla_\Theta \log \mathbb{P}_\Theta(\mathbf{x})$ by "truncated backpropagation through time"

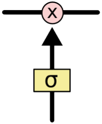
- i.e., reverse chain-rule + "clip" exploding derivatives

Long short term memory[1]

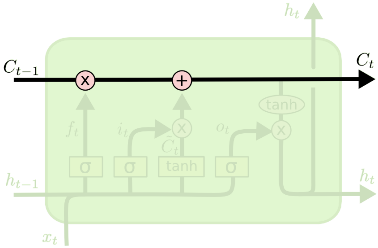


Information passes through a series of "gates"

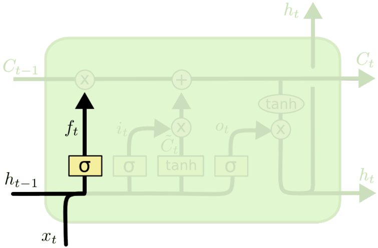
- "Gate" = multiplication with sigmoid
  - $\sigma = 0 \Rightarrow$ "let nothing thru"
  - $\sigma = 1 \Rightarrow$ "let all thru"



[1] graphics have slight differences with Graves 2013; they are taken from: https://colah.github.io/posts/2015-08-Understanding-LSTMs

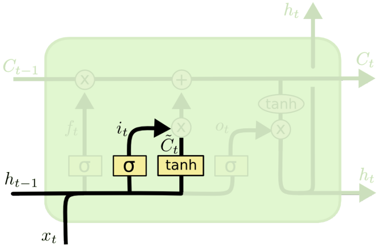


C_t = "cell state" = flows horizontally across LSTM units



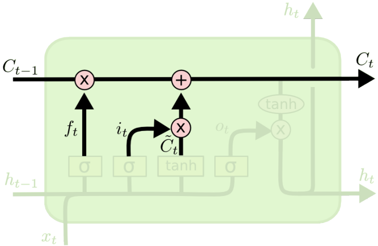
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

f_t = "forget gate" = gate to forget information from C_{t-1}



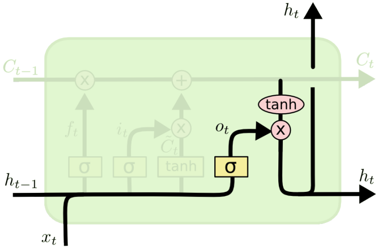
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

i_t = "input gate" = gate to add information from h_{t-1} and x_t to C_t



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Update C_t using f_t and i_t

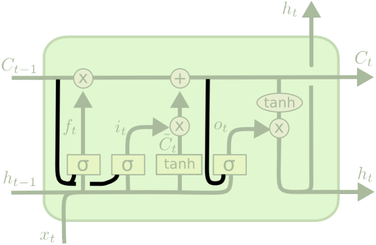


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

o_t = "output gate" = gate to output information to cell above/right

Many variations on LSTM exist. Here is the one used in Graves 2013:



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Notice the extra "peephole" connections from C to f, i, o

Table of Contents

Text Prediction

Basic framework for text prediction
$y_t = \mathbb{P}(x_{t+1})$
Per-character vs per-word prediction

Penn Treebank Test Set

Penn Treebank
Perplexity? BPC?
Regularization schemes

Wikipedia Experiments

- Wikipedia experiments
- `karpathy.github.io/2015/05/21/rnn-effectiveness` has some nice visualizations for understanding what's going on, e.g. what the inner neurons represent

Table of Contents

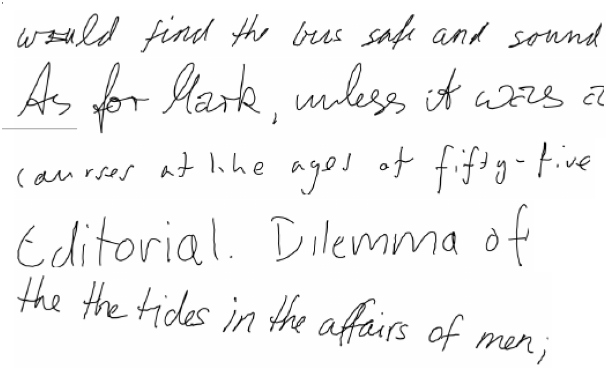


Figure: Training samples from IAM handwriting database

Input $x_t = (\text{pos}_t - \text{pos}_{t-1}) \times (\text{is-end-of-stroke}) \in \mathbb{R}^2 \times \{0, 1\}$

Mixture density network

$$
\begin{aligned}
x_t &\in \mathbb{R} \times \mathbb{R} \times \{0, 1\} \\
y_t &= (e_t, \{\pi_t^j, \mu_t^j, \sigma_t^j, \rho_t^j\}_{j=1}^M) \\
\mathbb{P}(x_{t+1} \mid y_t) &= \sum_{j=1}^{M} \pi_t^j \mathcal{N}(x_{t+1} \mid \mu_t^j, \sigma_t^j, \rho_t^j) \begin{cases} e_t & \text{if } (x_{t+1})_3 = 1 \\ 1 - e_t & \text{else} \end{cases}
\end{aligned}
$$

- e_t = end-of-stroke prob = $\frac{1}{1+\exp(\hat{e}_t)}$
- π_t^j = mixture prob = $\frac{\exp(\hat{\pi}_t^j)}{\sum_{j'} \exp(\hat{\pi}_t^{j'})}$
- μ_t^j = component mean = $\hat{\mu}_t^j$
- σ_t^j = component variance = $\exp(\hat{\sigma}_t^j)$
- ρ_t^j = component x/y correlation = $\tanh(\hat{\rho}_t^j)$

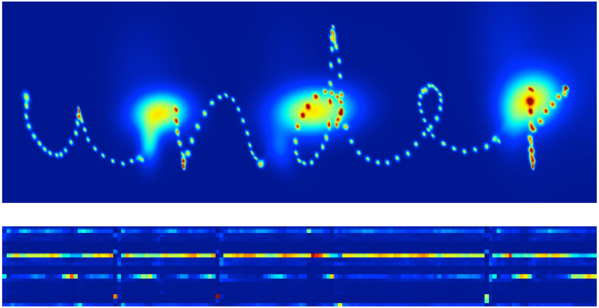


Figure: Mixture density outputs as word "under" is written

- Blobs = Predictions at end of strokes for first point in next stroke
- Lower panel = component weights

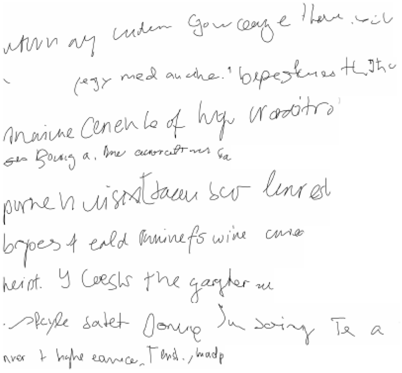


Figure: Samples generated by prediction network (700-timesteps each). Network has learned strokes, some characters, and even short words (e.g. "of").

- 3 hidden layers, each with 400 LSTM cells
- 3.4×10^6 parameters total
- Trained with rmsprop and adaptive weight noise
- Output derivatives $\frac{\partial \log \mathbb{P}(\mathbf{x})}{\partial \hat{y}_t}$ clipped to be in $[-100, 100]$; LSTM derivatives clipped within $[-10, 10]$
  - Clipping vital for numerical stability

Handwriting synthesis

- Generate handwriting for a given text
- Main challenge: aligning continuous handwriting sequence with discrete text
- Main idea: bottom hidden layer maintains a distribution of the current position in the text ("soft window")

Synthesis Network

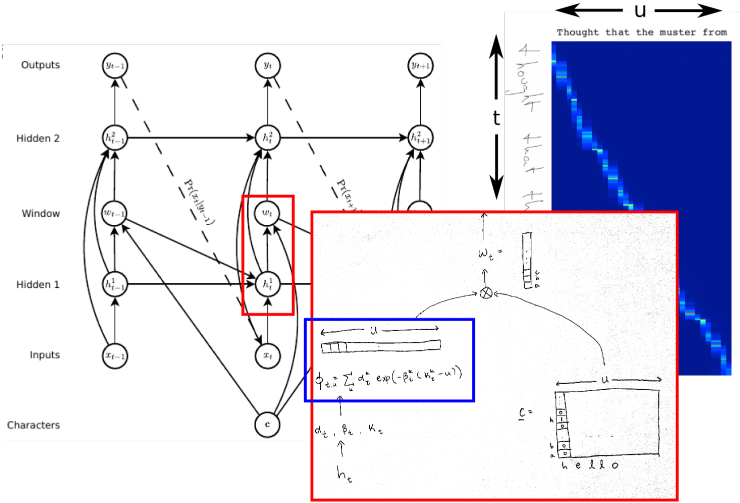
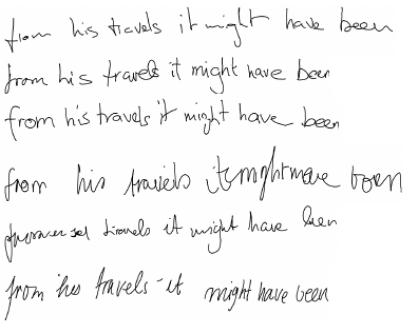
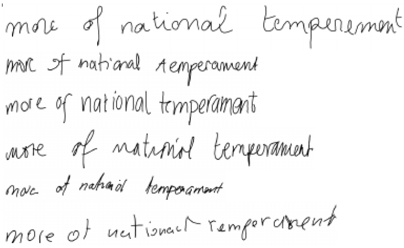


Figure: $\phi_t(h_t^1)$ = distn over positions; $w_t = \mathbf{c}\phi_t$ = distn over characters

Unbiased sampling





- Sample from $\mathbb{P}(\mathbf{x} \mid \mathbf{c})$ by iteratively sampling from $\mathbb{P}(x_{t+1} \mid y_t)$
- Stop when $\phi(t, U+1) > \max_{u \leq U} \phi(t, u)$
- First line of each block is real; subsequent lines generated by network

Biased sampling

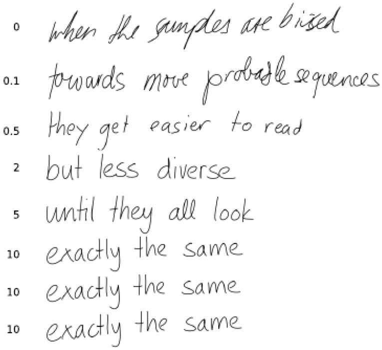


Figure: what is the bias term mean here

- Decreasing the variance of the output sequences leads to neater handwriting

Primed sampling