Generating Sequences With Recurrent Neural Networks

1 Citation

Graves, Alex. "Generating sequences with recurrent neural networks." arXiv preprint arXiv:1308.0850 (2013).

`https://arxiv.org/pdf/1308.0850.pdf`

2 Abstract

We use Long Short-Term Memory (LSTM) Recurrent Neural Networks (RNN) to predict the next character in a Wikipedia dataset and the next stroke in a handwriting recognition dataset. The result are models that can synthesize text and handwriting.

3 Introduction

To use an RNN for sequence generation, you sample an initial value, feed it through an RNN, sample from the output, and feed the output back in. The sampling ensures randomness. RNNs don't have much memory, so we use LSTMs.

Our datasets are Hutter Prize Wikipedia and IAM Online Handwriting Database.

4 Prediction Network

We use $t = 1...T$ and $n = 1...N$. Our hidden layers are represented by:

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

where H represents the LSTM equations.

Our LSTM equations are:

$$i_t = \sigma(W_{xi} x_t + W_{hi} h_{t-1} + W_{ci} c_{t-1} + b_i)$$
$$f_t = \sigma(W_{xf} x_t + W_{hf} h_{t-1} + W_{cf} c_{t-1} + b_f)$$
$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc} x_t + W_{hc} h_{t-1} + b_c)$$
$$o_t = \sigma(W_{xo} x_t + W_{ho} h_{t-1} + W_{co} c_{t-1} + b_o)$$
$$h_t = o_t \tanh c_t$$

where W_{ci}, W_{cf}, W_{co} are diagonal. We also have bias terms for the i_t, f_t, c_t, and o_t terms, but omit them in the equation above.

We stack multiple LSTM layers.

You need to use gradient clipping to prevent large gradients during training.

Sequences begin with a null vector.

Characters are one-hot encoded.

5 Text Prediction

We do character-level prediction (rather than word-level). Suppose there are K characters. We stack a softmax layer onto our LSTMs.

$$\Pr(x_{t+1} = k | y_t) = y_t^k = \frac{\exp(\hat{y}_k^t)}{\sum_{k'=1}^{K} \exp(\hat{y}_{k'}^t)}$$

Our loss function is:

$$L(\mathbf{x}) = -\sum_{t=1}^{T} \log \Pr(x_{t+1} | y_t) = -\sum_{t=1}^{T} \log y_t^{x_t+1}$$

We use dynamic evaluation, where we update the network weights after responding to a test example.

When training, we use adaptive weight noise to avoid overfitting.

We use two metrics: bits-per-character (BPC), which is the average of $\log_2 \Pr(x_{t+1}, y_t)$ and perplexity, which is 2^{BPW} where BPW is the average bits-per-word.

We use the 1M word Penn Treeback dataset and train both word-level and character-level LSTM networks. The word-level network does a little better on perplexity and BPC.

The Hutter Prize Wikipedia dataset has 100M bytes from English Wikipedia. It includes XML + URLs as well. The input and output to our network is a character (from a set of 205 unicode characters). We train with 7 layers of 700-cell LSTMs (21M weights) that we train with SGD (0.0001) + momentum (0.9) for 4 epochs with gradient clipping to $[-1, 1]$. We use dynamic evaluation at test-time. The model learns to close XML tags and parentheses and generates many real words and realistic URLs. The network does not make sense beyond a few short phrases. The network seems to be heavily influenced by the data that it was most recently trained on (seems to be underfitting).

6 Handwriting Prediction

The IAM-OnDB dataset has online (i.e. sequence of pen (x, y) coordinates instead of image of handwriting) handwriting data. We predict one pen-stroke at a time, which requires a lot of memory because a single letter can span 25 strokes and a line of text can span 700 strokes.

To set the output of our network, we use mixture density networks. The input to our network is a triple (x_1, x_2, x_3) where (x_1, x_2) are the pen coordinates and $x_3 \in \{0, 1\}$ indicates whether the point ends a stroke.

The output of our network is a mixture density network of the form:

$$\hat{y}_t = (\hat{e}_t, \{\hat{w}_j, \hat{\mu}_j, \hat{\sigma}_j, \hat{\rho}_j\}_{j=1}^{M}) = b_y + \sum_{n=1}^{N} W_{h^n y} h_t^n$$

To constrain these values to the proper ranges, we do the following:

$$e_t = \frac{1}{1+\exp(\hat{e}_t)}$$
$$\pi_t^j = \frac{\exp(\hat{\pi}_t^j)}{\sum_{j'=1}^{M}\exp(\hat{\pi}_t^{j'})}$$
$$\mu_t^j = \hat{\mu}_j^t$$
$$\sigma_t^j = \exp(\hat{\sigma}_t^j)$$
$$\rho_t^j = \tanh(\hat{\rho}_t^j)$$

With this, we can define the following probability that we can plug into our loss function:

$$\Pr(x_{t+1}|y_t) = \sum_{j=1}^{M} \pi_t^j N(x_{t+1}|\mu_t^j, \sigma_t^j, \rho_t^j) \begin{cases} e_t & \text{if } (x_{t+1})_3 = 1 \\ 1-e_t & \text{otherwise} \end{cases}$$

where

$$N(x|\mu,\sigma,\rho) = \frac{1}{2\pi\sigma_1, \sigma_2, \sqrt{1-\rho^2}} \exp\frac{-Z}{2(1-\rho^2)}$$

and

$$Z = \frac{(x_1-\mu_1)^2}{\sigma_1^2} + \frac{(x_2-\mu_2)^2}{\sigma_2^2} - \frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2}$$

We normalize the pen stroke offsets to mean zero and unit variance. We set $M = 20$ for our mixture density network.

We use three layers of LSTMs with 400 cells each. We train with adaptive weight noise. We use RMSProp to train the network:

$$\epsilon_i = \frac{\partial L(\mathbf{x})}{\partial w_i}$$
$$n_i = Nn_i + (1-N)\epsilon_i^2$$
$$g_i = Ng_i + (1-N)\epsilon_i$$
$$\triangle_i = D\triangle_i - C\frac{\epsilon_i}{\sqrt{n_i - g_i^2 - L}}$$
$$w_i = w_i + \triangle_i$$

where $(N, D, C, L) = (0.95, 0.9, 0.0001, 0.0001)$,

Output derivatives were clipped to $[-100, 100]$ and LSTM derivatives were clipped to $[-10, 10]$. Despite this, the networks sometimes had numerical problems later in training.

The resulting handwriting samples have good looking letters and even a few short words.

7 Handwriting Synthesis

The previous section describes a network that can take a pen position and end-of-stroke marker and predict where the pen goes next. Now, we want to generate pen strokes based on a sequence of input characters.

The key challenge is that the input and output sequences are different lengths. We tried using an RNN transducer, but that did not get good performance. So, we convolve a "soft window" with the text string and use that as an input instead.

Our network has three hidden layers. Inputs have skip connections to all hidden layers. Hidden layers have skip connections to the output layers. We feed the previous output (i.e. (x_1, x_2, x_3) - pen stroke coordinates and end-of-stroke marker) as input to the next timestep and we also feed in the character sequence $\mathbf{c}$.

For a length U character sequence $\mathbf{c}$ and length T data sequence $\mathbf{x}$, we have:

$$\phi(x, u) = \sum_{k=1}^{K} \alpha_t^k \exp\left(-\beta_t^k (\kappa_t^k - u)^2\right)$$

$$w_t = \sum_{u=1}^{U} \phi(t, u) c_u$$

Intuitively, c_u is the window height at u, κ_t is the window location, β_t is window width, and α_t is the window importance.

We learn the $3K$ window parameters using the first hidden layer of the network:

$$(\hat{\alpha}_t, \hat{\beta}_t, \hat{\kappa}_t) = W_{h^1p} h_t^1 + b_p$$

$$\alpha_t = \exp(\hat{\alpha}_t)$$

$$\beta_t = \exp(\hat{\beta}_t)$$

$$\kappa_t = \kappa_{t-1} + \exp(\hat{\kappa}_t)$$

Notice that the κ_t values are modeled as offsets - this is essential for getting the network to work properly.

We pass the w_t vector to the second and third hidden layer, and also to the first hidden layer at the next timestep. The hidden layer functions are:

$$h_t^1 = H(W_{ih^1} x_t + W_{h^1h^1} h_{t-1}^1 + W_{wh^1} w_{t-1} + b_h^1)$$

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

where H represents the LSTM.

We use the IAM-OnDB dataset again because it has both the online handwriting data and the transcribed character sequences. We use three LSTM layers with 400 cells each and $M = 20$ mixture distribution components and 57 possible characters. We set $K = 10$ for the soft window.

As before, we train with RMSProp and adaptive weight noise.

How do we tell the synthesis network to stop generating outputs? We use the heuristic that when $\phi(t, U+1) > \phi(t, u)$ for $1 \leq u \leq U$, we stop synthesizing inputs. This produces very diverse handwriting outputs, but people we showed it to said it was sometimes hard to read. The model struggled a lot with rare words, indicating that it might be implicitly learning a character language model.

To bias the model towards more readable handwriting, we pick a "probability bias" b and update the mixture standard deviations as follows:

$$\sigma_t^j = \exp(\hat{\sigma}_t^j - b)$$

and the mixture weights as follows:

$$\pi_t^j = \frac{\exp(\hat{\pi}_t^j(1+b))}{\sum_{j'=1}^{M} \exp(\hat{\pi}_t^{j'}(1+b))}$$

The above technique biases the model towards selecting the best component.

We also experimented with "primed sampling", which is where the model mimics a given handwriting sample. So, if we want to prime with character sequence **c** and handwriting sample **x** and then generate handwriting for a different character sequence **s**, we do the following. First, we feed in sequence **c** and generate outputs, but instead of feeding network back into the network, we use **x** instead. Then, we feed in **s** and run the network as usual.

Note that if you combine priming with the variance reduction technique for making handwriting more readable, you could make a system that takes existing handwriting and makes it more readable.

8 Conclusions and Future Work

We've shown how LSTMs can be used to generate sequences. Our convolutional soft window is a helpful in synthesizing handwriting.