CS244d Assignment 2

Denny Britz

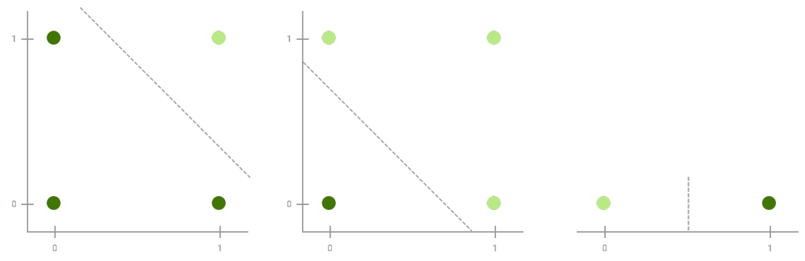
July 2015

0 Warmup: Boolean Logic

(a)

$A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B) = (A \vee B) \wedge \neg(A \wedge B)$.

A linear classifier is able to represent an operation if the classes can be separated by a linear boundary. Figure 1 shows how this is possible for AND, OR and NOT.



(a) AND Operator (b) OR Operator (c) NOT Operator

Figure 1: Classifying Boolean Operators. Red circles represent a class of 0. Green circles represent a class of 1. The dashed line shows a linear classifier separating the classes.

(b)

For the AND operator we can set $b_i = -1.5$ and $w_{i1} = w_{i2} = 1$.

For the OR operator we can set $b_i = -0.5$ and $w_{i1} = w_{i2} = 1$.

For the NOT operator we can set $b_i = 0.5$ and $w_{i1} = -1$.

One can verify each of the above by writing down the truth table.

0.1 (c)

This part of the assignment is done in the iPython notebook.

1 Deep Networks for Named Entity Recognition

To keep the notation consistent let's define the layer inputs and activations as follows:

$$
\begin{align}
a_1 &= z_1 = x^{(t)} \tag{1}\\
z_2 &= Wa_1 + b_1 \tag{2}\\
a_2 &= \tanh(z_2) = h \tag{3}\\
z_3 &= Ua_2 + b_2 \tag{4}\\
a_3 &= \text{softmax}(z_3) = \hat{y} \tag{5}
\end{align}
$$

The dimensions are $a_1, z_1 \in \mathbb{R}^{150}$ and $z_2, a_2 \in \mathbb{R}^{100}$ and $z_3, a_3 \in \mathbb{R}^5$.

(a)

We make use of the fact that the derivative of the cross entropy loss $J(\theta)$ for a softmax function is given by $\frac{\partial J}{\partial \theta} = (\hat{y} - y)$. We derived this fact in problem set 1.

$$
\begin{align}
\frac{\partial J}{\partial U} &= \frac{\partial J}{\partial z3}\frac{\partial z_3}{\partial U} \tag{6}\\
&= (a_3 - y)\frac{\partial z_3}{\partial U} \tag{7}\\
&= (a_3 - y)a_2^T = (\hat{y} - y)a_2^T \tag{8}\\
&= \delta_3 a_2^T \tag{9}
\end{align}
$$

We confirm that $\frac{\partial J}{\partial U} \in \mathbb{R}^{5x100}$.

$$
\begin{align}
\frac{\partial J}{\partial b_2} &= \frac{\partial J}{\partial z3}\frac{\partial z_3}{\partial b_2} \tag{10}\\
&= (a_3 - y)\frac{\partial z_3}{\partial b_2} \tag{11}\\
&= (a_3 - y) = (\hat{y} - y) \tag{12}\\
&= \delta_3 \tag{13}
\end{align}
$$

We confirm that $\frac{\partial J}{\partial b_2} \in \mathbb{R}^5$.

Let's remember that $\tanh'(x) = 1 - \tanh^2(x)$. From the backpropagation formula we know that:

$$\begin{aligned}
\frac{\partial J}{\partial W} &= \delta_2 a_1^T &(14)\\
&= ((1 - \tanh^2(z_2)) \circ (U^T \delta_3) a_1^T &(15)\\
&= ((1 - \tanh^2(z_2)) \circ (U^T (\hat{y} - y)) a_1^T &(16)
\end{aligned}$$

We confirm that $\frac{\partial J}{\partial W} \in \mathbb{R}^{100x150}$.

$$\begin{aligned}
\frac{\partial J}{\partial b_1} &= \delta_2 &(17)\\
&= ((1 - \tanh^2(z_2)) \circ (U^T \delta_3) &(18)\\
&= ((1 - \tanh^2(z_2)) \circ (U^T (\hat{y} - y)) &(19)
\end{aligned}$$

We confirm that $\frac{\partial J}{\partial b_1} \in \mathbb{R}^{100}$

$$\begin{aligned}
\frac{\partial J}{\partial L_1} &= W^T \delta_2 &(20)\\
&= W^T((1 - \tanh^2(z_2)) \circ (U^T (\hat{y} - y)) &(21)
\end{aligned}$$

We confirm that $\frac{\partial J}{\partial L_1} \in \mathbb{R}^{150}$.

(b)

Since $\frac{\partial J_{reg}}{\partial b_1} = \frac{\partial J_{reg}}{\partial b_2} = \frac{\partial J_{reg}}{\partial L_i} = 0$ the gradients for b_1, b_2 and L_i stay the same.

$$\begin{aligned}
\frac{\partial J_{reg}}{\partial W} &= \lambda W &(22)\\
\frac{\partial J_{reg}}{\partial U} &= \lambda U &(23)
\end{aligned}$$

We simply add the above to $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial I}$ from section 1.

(c)

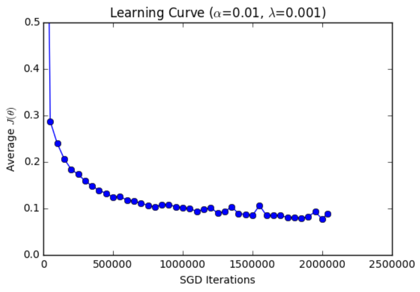
Done in the iPython notebook.

(d)

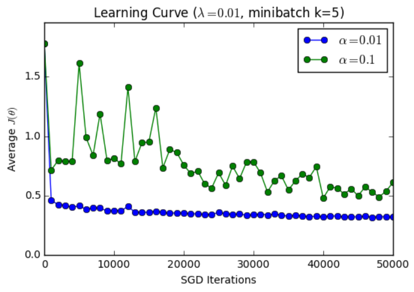
Done in the iPython notebook. For my model I used $\alpha = 0.01$, $\lambda = 0.001$, a 100-dimensional hidden layer, and mini-batch sizes of $k = 10$. I iterated over the training set 20 times. I did not use annealing for the learning rate.

(e)

Done in the iPython notebook. The figure below shows the learing plots.



(a) Best Learning Curve



(b) Comparison Learning Curve

(f)

Done in the iPython notebook. Table 1 shows the performance of the best model.

Mean precision	88.62%
Mean recall	70.59%
Mean F1	78.30%

Table 1: Performance evaluation of the best model.

1.1 Deep Networks: Probing Neuron Responses

(a)

Table 2 shows the top-10 word lists for the center word on 5 hidden layer neurons. We can see that some of the neurons try to encode high-level concepts such as countries, institutions or adverbs.

Neuron 2	malaysia, germany, berlin, britain, ocean, norway, ireland, pakistan, iraq, korea
Neuron 7	have, which, worker, help, measures, ;, what, sector, that, how
Neuron 24	suddenly, permanently, immediately, foster, there, forth, she, temporarily, behalf, thereby
Neuron 35	educational, salaries, membership, health, ministry, legacy, approval, assistance, credit, instruction
Neuron 67	admission, achievement, additional, initial, extra, arithmetic, example, answer, endorsement, instruction

Table 2: Top-10 word lists for the center word on 5 hidden layer neurons.

(b)

Table 3 shows the top-10 word lists for the center word on the model output. We see that the neurons learn general concepts related to locations, organizations and people. For example, the concept of people names for the PER label.

LOC	lanka, england, korea, berlin, ireland, egypt, britain, iraq, pakistan, italy
MISC	german, turkish, danish, korean, brazilian, olympic, israeli, italian, belgian, english
ORG	zenith, inc., liverpool, cdu, microsoft, reuters, ajax, commons, inc, corp
PER	wept, jim, scott, stephen, martin, herb, trembling, sarah, roger, pat

Table 3: Top-10 word lists for the center word on the model output.

(c)

Table 4 shows the top-10 word lists for the first word on the model output. The models seems to learn concepts that commonly prefix the classes. For example directional modifiers for LOC, first names for PER and company prefixes for ORG.

LOC	beneath, near, st., mount, lake, west, native, san, santa, cape
MISC	county, golf, oak, serie, exotic, fair, grand, lakes, tour, windows
ORG	la, cdu, f.c., securities, cooperation, enterprise, &, v, moody, dream
PER	stephen, matt, d., pat, m., john, a., michael, roger, peter

Table 4: Top-10 word lists for the first word on the model output.

2 Recurrent Neural Networks: Language Modeling

2.1 (a)

Below I dropped the (t) superscript for the sake of brevity.

$$
\begin{aligned}
2^J &= 2^{-\sum_{j=1}^{|V|} y_j \log \hat{y}_j} && (24)\\
&= \frac{1}{\prod_{j=1}^{|V|} 2^{y_j \log \hat{y}_j}} && (25)\\
&= \frac{1}{\prod_{j=1}^{|V|} 2^{\log \hat{y}_j^{y_j}}} && (26)\\
&= \frac{1}{\prod_{j=1}^{|V|} \hat{y}_j^{y_j}} && (27)\\
&= \frac{1}{\sum_{j=1}^{|V|} y_j \cdot \hat{y}_j} && (28)\\
&= PP\left(\hat{y}, y\right) && (29)
\end{aligned}
$$

In the last step we used the fact that y_j is a one-hot vector and thus the sum and the product are identical. Since taking something to exponent of 2 is a monotonic transformation, minimizing the cross entropy also minimizes the perplexity.

Summed over all examples we can see that $2^{\frac{1}{|N|}\sum_t J^{(t)}} = \sqrt[N]{\prod_t PP\left(\hat{y}, y\right)}$. Thus, the arithmetic mean of the cross entropy is the geometric mean of the perplexity.

In the case of completely random predictions we would predict $\frac{1}{|V|}$ for each word. This would yield a cross entropy of $-\log_2 \frac{1}{|V|}$:

$$
\begin{aligned}
-\log_2 \frac{1}{2000} &= 10.9657842847 && (30)\\
-\log_2 \frac{1}{10000} &= 13.2877123795 && (31)
\end{aligned}
$$

2.2 (b)

Let's introduce some notation that will be useful for the coming problems. Let:

$$z_1^{(t)} = Hh^{(t-1)} + Lx^{(t)} \tag{32}$$
$$z_2^{(t)} = Uh^{(t)} \tag{33}$$

We also define the δ terms for a given time step t. Note that at time step t there will be a delta term with respect to all the previous time steps. For example, $\delta_1^{(t)}$ is the delta term at time t with respect to time step 1. Let:

$$\delta_x^{(t)} = \frac{\partial J^{(t)}}{\partial z_1^{(x)}} \tag{34}$$

Let's now derive the gradients for the model parameters:

$$\begin{aligned}
\frac{\partial J^{(t)}}{\partial U} &= \frac{\partial J^{(t)}}{\partial z_2^{(t)}} \frac{\partial z_2^{(t)}}{\partial U} && (35)\\
&= (\hat{y}^{(t)} - y^{(t)}) \frac{\partial z_2^{(t)}}{\partial U} && (36)\\
&= (\hat{y}^{(t)} - y^{(t)}) h^{(t)T} && (37)\\
& && (38)
\end{aligned}$$

By the backpropagation rule, let $\delta_t^{(t)} = ((U^T(\hat{y}^{(t)} - y^{(t)})) \circ \sigma'(z_1^{(t)}))$.

$$\frac{\partial J^{(t)}}{\partial L} = \delta_t^{(t)} \left(x^{(t)}\right)^T \tag{39}$$
$$\left.\frac{\partial J^{(t)}}{\partial H}\right|_{(t)} = \delta_t^{(t)} \left(h^{(t-1)}\right)^T \tag{40}$$
$$\frac{\partial J^{(t)}}{\partial h^{(t-1)}} = H^T \delta_t^{(t)} \tag{41}$$

Note that since $x^{(t)}$ is a one-hot vector, $\frac{\partial J^{(t)}}{\partial L_{x^{(t)}}}$ is simply $\delta_t^{(t)}$.

2.3 (c)

In order to calculate $\delta_{(t-1)}^{(t)}$ we can backpropagate as follows:

$$\delta^{(t)}_{(t-1)} = (H^T \delta^{(t)}_t) \circ \sigma'(z_1^{(t-1)}) \tag{42}$$

Calculating the gradients is easy now. We note that the gradients at time $(t-1)$ only depend on the above delta term and the previous hidden layer value. By backpropagation:

$$\frac{\partial J^{(t)}}{\partial L_{x^{(t-1)}}} = \delta^{(t)}_{(t-1)} \tag{43}$$

$$\frac{\partial J^{(t)}}{\partial H}\bigg|_{(t-1)} = \delta^{(t)}_{(t-1)} \left(h^{(t-2)}\right)^T \tag{44}$$

2.4 (d)

Let's assume the cost of matrix multiplication for AB with $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ is $O(mnp)$. We also assume that our vocabulary $|V|$ is much larger than the dimensionality of our hidden layers D_h.

Then, for forward propagation we get:

$$O(D_h^2 + |V|D_h) = O(|V|D_h) \tag{45}$$

Backpropagating one step in time:

$$O(2|V|D_h + D_h^2) = O(|V|D_h) \tag{46}$$

Backpropagating τ steps in time, just adds one more calculation for $\delta^{(t)}_{(t-1)}$ and a gradient of complexity D_h^2:

$$O(|V|D_h + \tau D_h^2) == O(|V|D_h) \tag{47}$$

$|V|D_h$ is the slowest operation in the above. Thus, the computation time is heavily depended on the size of the vocabulary.

2.5 (e)

Done in the iPython notebook.

2.6 (f)

The hyperparameters for my network are $\alpha = 0.005$, bptt $= 3$. I used a SGD with a random training schedule and one pass over the complete dataset (56,522 iterations). The unadjusted perplexity score is 125.696 and the adjusted perplexity score is 222.597.

I could probably do better by further turning the parameters and using and using annealing for the learning rate. However, training takes several hours and I did not have time to run more experiments.

2.7 (g)

Three examples sentences my RNN generated are, with their cross-entropy loss:

- The all american in debt does exchange the company (63.0133)
- In the materials (25.339)
- These receive documents with any sign (48.11)