P¿1[0]

DS-GA.1008 Deep Learning April 7, 2017

Solutions to Assignment 2

Team: Peanut Butter *Due: Tuesday, April 4*

Problem 1 - Batch Normalization

- Write down the expression of mean and variance of scaler features. If we want transform them into zero mean and unit standard deviation, how do we normalize the feature?

  Solution: Assume that we are implementing a BN module with d-dimensional input $x = (x_1, x_2, \ldots, x_d)$ and we choose a mini-batch of size m.

  We need to normalize each scalar feature independantly. Thus for each dimension k, we have

$$\mu_B \implies \frac{1}{m} \cdot \sum_{i=1}^{m} x_i^k$$

$$\sigma_B^2 \implies \frac{1}{m} \sum_{i=1}^{m} (x_i^k)^2$$

$$\hat{x_i} = \frac{x_i - \sigma_B^2}{\sqrt{\sigma_B^2 + \epsilon)}}$$

- Write down the output of BN in terms of γk and β_k

  Solution:

$$BN_{\gamma,\beta}(x_i) = \gamma(\hat{x_i}) + \beta$$

  Lets represent $BN_{\gamma,\beta}$ as y and E as the energy that needs to be backpropogated through the network. Using chain rule, we have

$\frac{\partial E}{\partial \hat{x_i}} = \frac{\partial E}{\partial y_i} \cdot \gamma$
$\frac{\partial E}{\partial \sigma_B^2} = \sum_{i=1}^{m} \frac{\partial E}{\partial \hat{x_i}}(x_i - \mu_B) \cdot \frac{-1}{2}(\sigma_B^2 + \epsilon)^{\frac{-3}{2}}$
$\frac{\partial E}{\partial \mu_B} = (\sum_{i=1}^{m} \frac{\partial E}{\partial \hat{x_i}} \cdot \frac{-1}{\sqrt{\sigma_B^2+\epsilon}}) + \frac{\partial E}{\partial \sigma_B^2} \cdot \frac{\sum_{i=1}^{m} -2\dot{(}x_i-\mu_B)}{m}$
$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial \hat{x_i}} \cdot \frac{1}{\sqrt{\sigma_B^2+\epsilon}} + \frac{\partial E}{\partial \sigma_B^2} \cdot \frac{2\dot{(}x_i-\mu_B)}{m} + \frac{\partial E}{\partial \mu_B} \cdot \frac{1}{m}$

$\frac{\partial E}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial E}{\partial y_i} \cdot \hat{x_i}$
$\frac{\partial E}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial E}{\partial y_i}$

DS-GA.1008 Deep Learning April 7, 2017

Solutions to Assignment 2

Team: Peanut Butter *Due: Tuesday, April 4*

Problem 2 - Convolution

1. How many values will be generated if we forward propagate the image over the given convolution kernel?

   Solution:
   The number of values generated at the end of a forward propagation of an image of size $W_1 * H_1$ is given by
   $$W_2 = (W_1 - k_W)/s + 1$$
   $$H_2 = (H_1 - k_H)/s + 1$$
   where k_W is the kernel width, k_H is the kernel height, and s is the stride.
   In our case $W_1 = H_1 = 5$ and $k_W = k_H = 3$ and s = 1. So, the output size $W_2 = H_2 = (5-3)/1+1 = 3$.
   Therefore, the number of values generated = **3 * 3 = 9** $\square$

2. Calculate these values

   Solution:
   Convolution between 2 functions f and g is defined as follows -
   $$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2].g[x-n_1][y-n_2]$$
   Forward propagation is calculated by convolving the image with the kernel in "valid" mode. The result of such a convolution is as follows -
   $$\begin{bmatrix} 158 & 183 & 172 \\ 229 & 237 & 238 \\ 195 & 232 & 244 \end{bmatrix}$$ $\square$

3. Suppose the gradient backpropagated from the layers above this layer is a 33 matrix of all 1s. Write down the value of the gradient (with respect to the input) backpropagated out of this layer.

   Solution:
   Backward propagation can be achieved by performing a "full" convolution on the gradient of the previous layer and the flipped kernel (rotate the kernel by 190 degrees). The result is as

follows -

$$\begin{bmatrix} 3 & 11 & 14 & 11 & 3 \\ 5 & 20 & 32 & 27 & 12 \\ 10 & 25 & 39 & 29 & 14 \\ 7 & 14 & 25 & 18 & 11 \\ 5 & 5 & 7 & 2 & 2 \end{bmatrix} \qquad \square$$

DS-GA.1008 Deep Learning April 7, 2017

Solutions to Assignment 2

Team: Peanut Butter *Due: Tuesday, April 4*

Problem 3 - Variants of pooling

1. List three different kinds of pooling, and their corresponding module implemented in PyTorch.

   Solution:

   These are the 3 types of pooling that are implemented in PyTorch.

Index	**Type of Pooling**	**Corresponding 2D module**
1	Max pooling	nn.MaxPool2d
2	L_p pooling	nn.LPPool2d
3	Average pooling	nn.AvgPool2d

Table 1: PyTorch implementations of pooling

   There are also "adaptive" implementations of max and average pooling like AdaptiveMaxPool2d and AdaptiveAvgPool2d. □

2. Write down the mathematical forms of these three pooling modules.

   Solution:
   Pooling is performed over a vector v that results in a scalar *f(v)*. The different types of pooling are defined as follows -

   - Max Pooling: $f(v) = Max_i(v_i)$
   - L_p Pooling: $f(v) = (\sum_i v_i^p)^{\frac{1}{p}}$
   - Average Pooling:$f(v) = \frac{1}{P}\sum_{i=1}^{P} v_i$

   □

3. Pick one of the pooling listed and describe the reason for incorporating it into a deep learning system.

   Solution:

   - **Max pooling** helps achieve spatial invariance by reducing the resolution of the feature maps, thereby obtaining more compact representations, and better robustness to noise and clutter.

- Invariance to local translation helps us recognize the presence or absence of a feature than recognizing *exactly* where it is.
- Reducing the size of the feature maps also makes the network computationally more efficient.
- Specifically, a feature f_m, a Bernoulli variable of variance $\sigma_m^2 = (1-(1-\alpha)^P)(1-\alpha)^P$, whose variance can be shown to be increasing first, then decreasing.
- The increase of the variance can play against the better separation of the expectations of the max-pooled feature activation, when parameter values α_1 and α_2 are too close for the two classes.
- Therefore, Max pooling is particularly well suited to the separation of features that are very sparse (i.e., have a very low probability of being active simultaneously)

$\square$

DS-GA.1008 Deep Learning	April 7, 2017

Solutions to Assignment 2

Team: Peanut Butter *Due: Tuesday, April 4*

Problem 4 - t-SNE

1. What is the crowding problem and how does t-SNE alleviate it? Give details.

   Solution: The crowding problem arises when we try to accommodate datapoints of a higher-dimensional space into a two dimensional data plane. This can result in the area of the two dimensional map available to accommodate moderately distant data points will not be nearly large enough compared with the area available for nearby datapoints, and is known as the crowding problem.

   t-SNE alleviates this problem by,

   - Converting distances into probabilities using a Gaussian distribution in the high dimensional space
   - Using a probability distribution that has much heavier tails than a Gaussian to convert distances into probabilities in the lower dimensions.

   This allows the moderate distances in the higher dimension space to be modeled into a much larger distance in the lower dimension space, thus eliminating the mis-representation of moderately dissimilar data points. □

2. Gradient of the t-SNE cost function.

$$C = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$
$$= \sum_i \sum_j p_{ij} \log p_{ij} - p_{ij} \log q_{ij}$$

Let $d_{ij} = ||y_i - y_j||$, and,

$$Z = \sum_{k \neq l} (1 + d_{kl}^2)^{-1}$$

If we fix, y_i, only pariwise distances that change are d_{ij} and d_{ji} $\forall$ j.

Hence,

$$\frac{\partial C}{\partial y_i} = \sum_j \Big(\frac{\partial C}{\partial d_{ij}} + \frac{\partial C}{\partial d_{ji}} \Big) (y_i - y_j).$$
$$= 2 \sum_j \frac{\partial C}{\partial d_{ij}} (y_i - y_j)$$

$$\frac{\partial C}{\partial d_{ij}} = \sum_{k \neq l} p_{kl} \frac{\partial (\log q_{kl})}{\partial d_{ij}}$$
$$= \sum_{k \neq l} p_{kl} \frac{\partial (\log q_{kl} Z - \log Z)}{\partial d_{ij}}$$
$$= \sum_{k \neq l} p_{kl} \left(\frac{1}{q_{kl} Z} \frac{\partial ((1 + d_{kl}^2)^{-1})}{\partial d_{ij}} - \frac{1}{Z} \frac{\partial Z}{\partial d_{ij}} \right)$$

$((1 + d_{kl}^2)^{-1})$is non-zero only when $k = i$ and $l = j$. Hence,

$$\frac{\partial C}{\partial d_{ij}} = 2 \frac{p_{ij}}{q_{ij} Z} (1 + d_{ij}^2)^{-2} - 2 \sum_{k \neq l} p_{kl} \frac{(1 + d_{ij}^2)^{-2}}{Z}.$$

For $\sum_{k \neq l} p_{kl} = 1$, hence,

$$\frac{\partial C}{\partial d_{ij}} = 2 p_{ij} (1 + d_{ij}^2)^(- 1) - 2 q_{ij} (1 + d_{ij}^2)^{-1}$$
$$= 2 (p_{ij} - q_{ij}) (1 + d_{ij}^2)^{-1}.$$

Substituting into the earlier value for $\frac{\partial C}{\partial d_{ij}}$, we get,

$$\frac{\partial C}{\partial d_{ij}} = 4 \sum_j (p_{ij} - q_{ij}) (1 + ||y_i - y_j||^2)^{-1} (y_i - y_j).$$

DS-GA.1008 Deep Learning April 7, 2017

Solutions to Assignment 2

Team: Peanut Butter *Due: Tuesday, April 4*

Problem 5 - Sentence Classification

1. **ConvNet**

   (a) Design a one layer ConvNet which first maps the sentence to a vector of length 5 (with the help of convolution and pooling), then feeds this vector to fully connected layer with softmax to get the probability values for possible 3 classes.

   Solution:

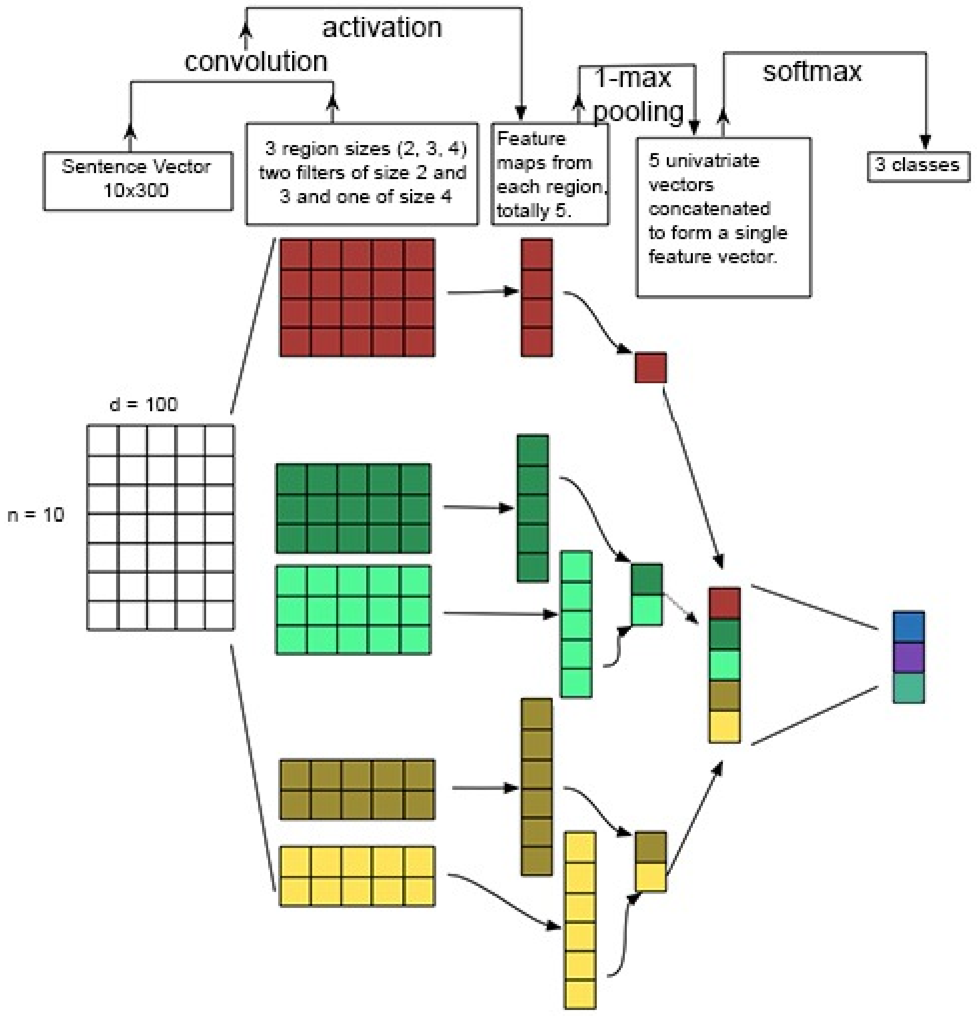
   i. Use 5 convolution filters, each of which take the input matrix of size [10x300]. The kernel sizes to be used are:
      - 100 x 2
      - 100 x 2
      - 100 x 3
      - 100 x 3
      - 100 x 4

   ii. Apply 1-max pooling on each result from the convolution kernels and concatenate them to get a feature vector of size 5.

   iii. Feed this flattened feature vector to a fully connected layer that then maps to 3 outputs.

   iv. Apply softmax on the outputs to obtain the predicted class.

   See illustration below.


activation
convolution
1-max pooling
softmax
Sentence Vector 10x300
3 region sizes (2, 3, 4) two filters of size 2 and 3 and one of size 4
Feature maps from each region, totally 5.
5 univatriate vectors concatenated to form a single feature vector.
3 classes
d = 100
n = 10

□

(b) Clearly mention the sizes for your input, kernel, outputs at each step.

Solution:

i. Input 10 words, each of size 100, which is a [10x300] matrix.
ii. Use 5 convolution layers, described as below.
   - Filter 1: 100 x 2, outputs a 100 x 9 vector upon convolution
   - Filter 2: 100 x 2, outputs a 100 x 9 vector upon convolution
   - Filter 3: 100 x 3, output a 100 x 8 vector upon convolution
   - Filter 4: 100 x 3, outputs a 100 x 8 vector upon convolution

- Filter 5: 100 x 4, outputs a 100 x 7 vector upon convolution

iii. Use 1-max pooling on the output each of the 5 convolution layers obtained from the above step. This would result in 5 individual values, which can be concatenated to for a [1x5] vector.

iv. Send the [1x5] vector into a fully connected layer, which then produces 3 output.

v. After applying softmax on the 3 outputs, we will be left with 1 class, which is the prediction made by this CNN.

□

(c) Please describe the effect of small filter size vs large filter size during the convolution. What would be your approach to select the filter sizes for classification task?

Solution: A typical filter in a convolution is not more than 5x5 in size. Stacking convolution layers with smaller filters as opposed to larger filters helps expose stronger input features with fewer parameters.

To obtain a 7x7 view of the input layer, we can do one of the 2 things

i. Perform a single convolution with 7x7 filters: By doing this, we would have a linear computation over the input using 7x7 filters. If the number of channels is C, this would amount to $49C^2$ parameters.

ii. Have 3 convolution layers with 3x3 filters: This is more expressive because of the nonlinearities that each layer introduces. If the number of channels is C, having 3 layers of filter size 3 will result in $27C^2$ parameters which is way fewer parameters in comparison.

Having smaller filter sizes, along with right padding, helps maintain spatial resolution by allowing us to stack up several conv layers of the same input size as output, as we see in Resnets and VGG, thereby allowing us to build very deep networks.
In our case, since the network was allowed to be only 1 layer deep, we experimented with different convolution filter sizes, and 10 worked better than 5.

□

2. **RNN**

(a) How simple RNN which is trained for language modeling, can be used to get the sentence vector?

Solution: There are multiple ways to get the sentence vector from an RNN which is trained for language modelling. If we were to called the sentence vector as a sentence embedding of constant length, we could simply consider a concatenation of hidden vectors as a vector which encodes the sentence representation. In language modelling the output vector dimension is same as the input vector dimension. □

(b) Design simple RNN which first maps the sentence to a vector of length 50 (with the help of convolution and pooling), then feeds this vector to fully connected layer with softmax to get the probability values for possible 4 classes.

Solution:

i. Input with sequence length 50
ii. Embedding layer with dimension d, along with vocabulary size of v
iii. Input to rnn is the same size as the embedding layer, d. The output from the rnn is of size 50*n, where n is the number of recurrent units.
iv. Decoder layer 1 is a Linear layer with sigmoid non linearity which takes as input 50*n and outputs k units
v. Decoder layer 2 is a Linear layer with sigmoid non linearity which takes an input size k and outputs 4 units. The output of this layer acts as the sentence vector representation which is further fed into a Softmax layer to output number of classes.

□

(c) Clearly mention the sizes of all the RNN components such as your input vector, hidden layer weight matrix, hidden state vecotrs, cell state vector, output layers.

Solution: In our experiments with the IMDB dataset, we implemented the above designed RNN as below -

i. Input vector - Sentence of 50 words
ii. Embedding dimension - d = 300
iii. Number of tokens 74555 (size of the vocabulary)
iv. hidden layer weight matrix [1x50x50]. We used 50 GRU units and the sentence size was set to 50.
v. Fully connected layer-1 mapping [50x50 $\rightarrow$ 500]
vi. Fully connected layer-2 mapping [500 $\rightarrow$ 2]

□

3. **Extra Credit - fastText experiments**

Solution:

We performed experiments on the Large Movie Review Dataset to classify reviews [1]. We compared fastText with the CNN and RNN architectures that we came up with. The final architecture used a sequence length of 30 and embedding size of 300.

CNN Architecture
input (30 words long sequence)
(embedding): Embedding(74555, 300)
(conv1):Conv1d(30, 10, kernel_size=(10,))
(maxpool1):MaxPool1d(kernel_size=2)
(fc1): Linear (1450 $\rightarrow$ 600)
(fc2): Linear (600 $\rightarrow$2)

Table 2: CNN used for sentence classification

RNN Architecture
input (30 words long sequence)
(encoder): Embedding(74555, 300)
(rnn): GRU(300, 50, bias=False)
(decoder1): Linear (1500 $\rightarrow$ 500)
(decoder2): Linear (500 $\rightarrow$2)

Table 3: RNN used for sentence classification

FastText implements a variant of skipgram, and tries to capture information about each word. This is where character-based n-grams come in. This is a fast way to explore sentence semantics and works much better than just looking at word embeddings as a whole, which is what our sentence classification models implement.
CNN with just one convnet layer struggled to learn and did not give accuracies over 50%. We experimented with different filter sizes and input sequence lengths but it was too noisy.
RNN does relatively better during cross validation and achieved a cross validation accuarcy of 75% as opposed to the CNN. It however did not generalize well on the train set. Below is a tabulated comparison of results.

The experiments were run for 30 epochs, with batch size 64. For CNN, an LR of .00001 was used and for RNN, an LR of .01 was used.

Experiment	Precison	Recall	Accuracy
CNN	0.51	0.51	12685/25014
RNN	0.5	0.5	12593/25014
fastText	0.859	0.859	-

□

4. **Extra Credit - Combining CNN and RNN for sentence classification**

We know that CNN is a useful feature extractor and RNN captures information from sequences effectively. So, we can chain them in order to create a sequence of useful features which can then be classified.
We can create a general architecture of combinations of CNN and RNN using a convolutional feature extractor applied on the input, then some recurrent network on top of the CNNs output, then a fully connected layer on RNNs output and finally a softmax layer. A simple architecture for an input sequence of length 50 can be designed like this -

```
RNNModelCNN (
(encoder): Embedding(74555, 300)
(conv1): Conv1d(50, 50, kernel_size=(3,), stride=(1,))
(maxpool1): MaxPool1d( kernel_size=(2,))
(rnn): GRU(149, 50, bias=False)
(decoder1): Linear (2500 -> 500)
(decoder2): Linear (500 -> 2)
)
```

DS-GA.1008 Deep Learning April 7, 2017

Solutions to Assignment 2

Team: Peanut Butter *Due: Tuesday, April 4*

Problem 6 - Language Modeling

We chose to experiment on the Penn Treebank dataset. For most of our experiments we chose to include all the words in the vocabulary (Instead of set

- Experiments with RNN cells :

RNN - Type	Train Ppl	Valid Ppl	Test Ppl
LSTM	19.18	118.98	115.95
GRU	207.8	347	-
RNN with Tanh	77.8	143.80	-
RNN with Relu	61.72	147.17	139.24

  Different RNN cell types were compared with a constant embedding size of 250, with 250 hidden layers and a sequence length of 40. As expected, LSTM outperformed other RNN types. One reason could be that a sequence length of 40 is too long for the gradients to propagate across the network. Although the gradient exploding problem has been taken care by the clipping gradients after every epoch, we realized that the low performance in GRU/RNN could be a case of vanishing gradients which has not been taken care.

- Experiments with Embedding size Increasing the embedding layer size consistently increased the performance, although that came at the cost of training time. Also, we noticed that the number of hidden layers have to be of comparable dimensions as the embedding layer dimension. Since the vocabulary includes all the words in the test set, we use only training perplexity as measure of comparison against an LSTM with sequence length 40.

Embedding layer size	Train ppl
250	41.89
300	17.38
500	15.54
1500	10.79

- Experiments on increasing the number of hidden units gave a gradual increase in performance, but was significantly slower. Beyond 750 units, however, there was no performance gain seen, and often it led to deteroration. The below experiments are the results from running on a learning rate of 5 on a RNN.

Number of hidden units	Train ppl	Time per epoch (s)
500	125.9	84
750	119.7	121
1000	125	219

- Using Pretrained embeddings
  In order to estimate if the embedding layer is getting properly trained, we used pretrained embeddings of dimension 300 and compared the performance of an LSTM network where embeddings of same dimension are trained. We downloaded pretrained GloVe vectors which was trained over the Wikipedia 2014 dataset with 6 Billion tokens and a vocabulary set of 400k words. We did a sideways comparison of how the training perplexity varies across the two models over epochs. It was surprising to see that the network where embeddings are trained performed on par with the network which used pretrained embeddings. Using pretrained embedding did provide an improvement in perplexity in the initial few epochs, However it was very marginal. Also, an LSTM network with 300 hidden layers and sequence length of 40 quickly caught up with the pretrained embedding model.

Epoch 1	trained LSTM	Pretrained w. LSTM
1	154.5	142.54
2	132.87	127.05
3	136.62	132.37
4	131.22	124.29

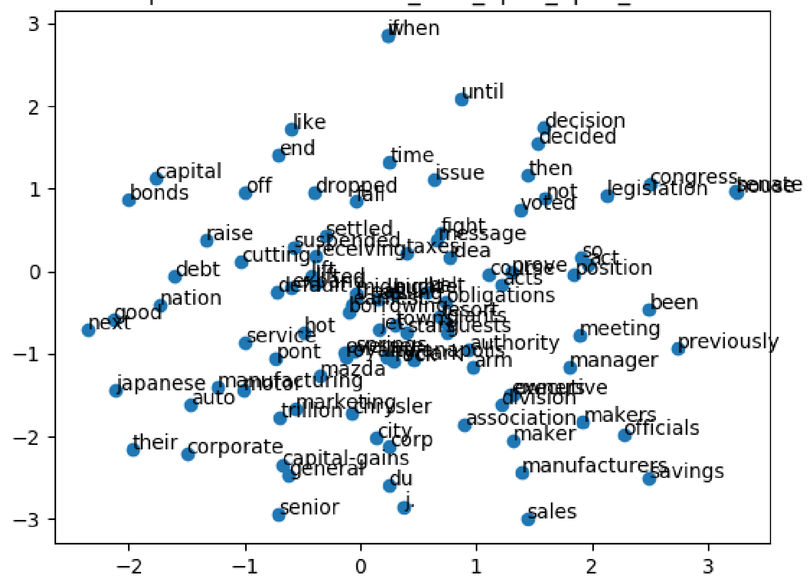
- Experiments with layers
  We fixed the RNN type to be RNN Tanh and used these parameters. We set the size of embedding to 650, number of hidden units to 650, used a dropout of 0.5 and ran it for epochs 40. We experimented with 1, 2 and 5 layers. We observed that with increased number of layers, the network learnt more slowly, resulting in larger test perplexities at the end of 40 epochs.
  We also observed that the gradients quickly decreased to 0 as the number of layers increased. The takeaway is that 0 initializations don't work as the number of layers increase. IRNN(RNN hidden weights are initialized with identity instead of 0s) can be explored in order to fix this which performs similar to LSTMs.

1 TSNE visualization of the best model

The below is the TSNe visualization of the first 100 words in the vocab. The embedding vectors were reduced to 2 components the using tSNE with a learning rate of 30 and and early exaggeration parameter of 10. The resulting KL divergence was 0.476746 after 275 iterations. From the plot, we can see that similar word groups are clustered together:

- has, was
- years, decades, ago
- cancer, symptoms
- up, to
- and, with, that
- director, conglomerate, group, board
- researchers, publishing

TSNE plot for model em600_lstm_bp50_ep10_100words



References

[1] Large Movie Review Dataset
http://ai.stanford.edu/~amaas/data/sentiment/

[2] Enriching Word Vectors with Subword Information *https://arxiv.org/abs/1607.04606*

[3] A Simple Way to Initialize Recurrent Networks of Rectified Linear Units *http://arxiv.org/pdf/1504.00941v2.pdf*