Foundations of Generative AI

CS 6180
Nadim Saad
Fall 2025
HW 1

Problem Set 1

Important:

- Write your name as well as your NU ID on your assignment. Please number your problems.
- Submit both results and your code.
- Give complete answers. Do not just give the final answer; instead show steps you went through to get there and explain what you are doing. Do not leave out critical intermediate steps.
- This assignment must be submitted electronically through Gradescope by September 19th 2025 (Friday) by 11:59 PM.

1 Word2Vec

Vectors can be used to represent words. Why do we represent words as vectors? To better compute with them. We saw in class that one-hot vectors can be used so that every word is given by a different vector, but we encode no meaningful notion of similarity or other relationship. This is why we started looking at the Word2Vec model which relies on the idea that "a word is known by the company it keeps".

We are given a corpus of text $\mathcal{C}$ and are interested in learning which words may appear in the surrounding of other words. Specifically, given a some specific words o and c, we want to predict the probability $P(O = o|C = c)$ that word o appears within the contextual window of c denoted by $\mathcal{W}_c$, i.e. o is an 'outside' word for c.

Recall that for each word w, we choose to learn two $d-$dimensional vectors $\vec{u}_w$, the 'outside' vector when w is an outside word, and $\vec{v}_w$, the 'center' vector when w is the center word. We assume that every word in our vocabulary is matched to an integer k so that $\vec{u}_k$ and $\vec{v}_k$ are the 'outside' and 'center' vectors, respecitvely, for the word indexed by k. We store the $\vec{u}$ and $\vec{v}$ vectors for all the words in our vocabulary in two matrices, U and V such that the k^{th} columns of U and V are $\vec{u}_k$ and $\vec{v}_k$, respecitvely.

We model the probability $P(O = o|C = c)$ using the softmax function,

$$P(O = o|C = c) = \frac{\exp\left(\vec{u}_o^T \vec{v}_c\right)}{\sum_{w \in \text{Vocab}} \exp\left(\vec{u}_w^T \vec{v}_c\right)}.$$

Recall from class that we seek to find the $\vec{u}$ and $\vec{v}$ vectors by minimizing the negative log-likelihood function (or maximizing the log-likelihood function) given by,

$$-\sum_{c \in \mathcal{C}} \sum_{o \in \mathcal{W}_c} \log P(O = o|C = c).$$

For the sake of simplicity, in this assignment we will focus on a single pair of words c and o so that the loss is given by,

$$\mathcal{L}(\vec{v}_c, o, U) = -\log P(O = o|C = c).$$

1. Consider a particular center word c and a particular outside word o. We define $\vec{y}$ to be a one-hot vector with a 1 for the true outside word o and 0 everywhere else, and $\vec{\hat{y}}$ to be a vector where each element contains the probabilities $P(O = o|C = c)$. Both vectors $\vec{y}$ and $\vec{\hat{y}}$ are vectors with length equal to the number of words in the vocabulary.

   We will prove that the loss function given in 1 can also be seen as the cross-entropy between $\vec{y}$ and $\vec{\hat{y}}$. Show that,

   $$-\sum_{w \in \text{Vocab}} y_w \log \hat{y}_w = \mathcal{L}(\vec{v}_c, o, U) = -\log P(O = o|C = c).$$

   You can describe your reasoning in words.

2. To train this model and figure out the optimal outside and center vectors, we need to compute the gradients of the loss function with respect to each word vectors. Show that,

   $$\nabla_{\vec{v}_c} \mathcal{L} = U\vec{\hat{y}} - U\vec{y}.$$

   Hint:

   - Start by simplifying the expression of the loss function so that it's easier to compute its derivative.
   - Explain why $U\vec{y} = \vec{u}_o$.
   - Explain why $U\vec{\hat{y}} = \sum_{w \in \text{Vocab}} \hat{y}_w \vec{u}_w$.

3. The gradient you found is the difference between the two terms. Provide an interpretation of how each of these terms improves the word vector when this gradient is subtracted from the word vector $\vec{v}_c$.

4. Given a single pair of words c and o, show that,

   $$\nabla_{\vec{u}_w} \mathcal{L} = \begin{cases} \hat{y}_w \vec{v}_c & \text{if } w \neq o, \\ (\hat{y}_o - 1)\vec{v}_c & \text{if } w = o. \end{cases} \tag{1}$$

   What is $\nabla_U \mathcal{L}$ equal to?

5. In many downstream applications using word embeddings, L^2 normalized vectors (e.g. $\vec{u}/\|u\|_2$ where $\vec{u}/\|u\|_2 = \sqrt{\sum_i u_i^2}$) are used instead of their raw forms. Let's consider a hypothetical downstream task of binary classification of phrases as being positive or negative, where you decide the sign based on the sum of individual embeddings of the words. When would L^2 normalization take away useful information for the downstream task? When would it not?

2 Adam Optimizer and Dropout

Stochastic Gradient Descent (SGD) can be used to train a deep learning model with parameters θ by minimizing the loss function $\mathcal{L}(\theta)$. Recall that SGD's update rule is given by,

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha \nabla_{\vec{\theta}} \mathcal{L}(\vec{\theta}^{(t)}).$$

SGD has some limitations. First, It can take a long time to converge or not even converge. For instance, if α is too small, the learning will be very slow, and α is large, it might even diverge. Second, every parameter is updated with the same step size, regardless of how frequently it appears or how steep the gradients are. Third, in stochastic settings, gradient estimates are noisy and SGD may jitter around minima without settling.

Adam Optimization extends SGD using some steps that we explore in this problem.

1. Momentum Adam first incorporates a rolling average of the gradient, denoted by m, leading to the update rule,

$$m^{(t+1)} = \beta_1 m^{(t)} + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta^{(t)}),$$
$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha m^{(t+1)},$$

where β_1 is a hyperparameter between 0 and 1 (often set to 0.9). The initial value of m is given by $m^{(0)} = 0$.

We will implement both SGD and SGD with momentum on $f(\vec{\theta}) = \frac{1}{2}\left(100\theta_1^2 + \theta_2^2\right)$ with $\vec{\theta} = \begin{bmatrix}\theta_1 \\ \theta_2\end{bmatrix}$. We choose a convergence tolerance of $\epsilon = 0.001$, an initial of $\vec{\theta}^{(0)} = \begin{bmatrix}1 \\ 1\end{bmatrix}$ and continue updating the parameters for at most $i_{max} = 1000$. How do the below settings perform:

- SGD with $\alpha = 0.1$.
- SGD with $\alpha = 0.005$.
- SGD with momentum with $\alpha = 0.1$ and $\beta_1 = 0.9$.

Did all scenarios lead to the optimal solution? How many iterations did the algorithms require for convergence (if they converged)?

2. Explain intuitively how using m stops the updates from varying as much and why this low variance may be helpful to learning, overall.

3. Adam then extends the idea of momentum with the trick of adaptive learning rates by keeping track of v, a rolling average of the magnitudes of the gradients,

$$m^{(t+1)} = \beta_1 m^{(t)} + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta^{(t)}),$$
$$v^{(t+1)} = \beta_2 v^{(t)} + (1-\beta_2)\left(\nabla_\theta \mathcal{L}(\theta^{(t)}) \star \nabla_\theta \mathcal{L}(\theta^{(t)})\right),$$
$$\theta^{(t+1)} = \theta^{(t)} - \alpha \frac{m^{(t+1)}}{\sqrt{v^{(t+1)}}},$$

where $\vec{x} \star \vec{y}$ computes an element wise multiplication of the two vectors $\vec{x}$ and $\vec{y}$, i.e. if $\vec{x} = \begin{bmatrix}x_1 \\ x_2 \\ \vdots \\ x_n\end{bmatrix}$ and $\vec{y} = \begin{bmatrix}y_1 \\ y_2 \\ \vdots \\ y_n\end{bmatrix}$ then $\vec{x} \star \vec{y} = \begin{bmatrix}x_1 y_1 \\ x_2 y_2 \\ \vdots \\ x_n y_n\end{bmatrix}$. Note the division between $m^{(t+1)}$ and $\sqrt{v^{(t+1)}}$ is also performed element wise.

Notice that $m^{(t+1)}$ is divided by $\sqrt{v^{(t+1)}}$. Which of the model parameters will get larger updates? How might this help with the learning?

4. Dropout is a regularization technique. During training, dropout randomly sets units in the hidden layer $\vec{h}$, with k elements, to zero with probability p_{drop} (dropping different units at each mini batch), and then multiplies $\vec{h}$ by a constant γ. This can be summarized as,

$$\vec{h}_{drop} = \gamma \vec{d} \star \vec{h},$$

where $\vec{d}$ is a vector of the same size as $\vec{h}$. The elements of $\vec{d}$ are randomly generated: an entry is 0 with probability p_{drop} and 1 with probability $1 - p_{drop}$.

(a) γ is chosen such that the expected value of $\vec{h}_{drop}$ is $\vec{h}$, that is,

$$E_{p_{drop}}\left[\left(\vec{h}_{drop}\right)_i\right] = \vec{h}_i,$$

for all $i = 1, 2, ..., k$. Show that $\gamma = \frac{1}{1-p_{drop}}$.

Hint: What are the possible values of $\left(\vec{h}_{drop}\right)_i$ and their associated probabilities?

(b) Does the expression for γ in the previous part make sense? Note that the dropout technique removes some neurons randomly, lowering the average activation.

(c) Why should dropout be applied during training and NOT be applied during evaluation?