CS224n Assignment 2

Written Answers

Your Name Here

November 14, 2025

1 Understanding Word2Vec

(a)

Since the true distribution y is a one-hot vector, we have $y_o = 1$ for the true outside word o and $y_w = 0$ for all $w \neq o$. Therefore,

$$-\sum_{w \in \text{Vocab}} y_w \log \hat{y}_w = -y_o \log \hat{y}_o = -\log \hat{y}_o.$$

(b)

(i)

We define the naive-softmax loss for a center word c and the true outside word o as

$$J = -\log P(O = o \mid C = c).$$

The skip-gram model uses a softmax distribution:

$$P(O = w \mid C = c) = \frac{e^{u_w^\top v_c}}{\sum_{w'} e^{u_{w'}^\top v_c}}.$$

Substituting this into the loss gives

$$J = -\log \left(\frac{e^{u_o^\top v_c}}{\sum_w e^{u_w^\top v_c}} \right).$$

Using the logarithm identity $\log \frac{a}{b} = \log a - \log b$, we obtain

$$J = -\left(u_o^\top v_c - \log \sum_w e^{u_w^\top v_c} \right) = -u_o^\top v_c + \log \sum_w e^{u_w^\top v_c}.$$

Thus the naive-softmax loss can be written as

$$J = -u_o^\top v_c + \log \sum_w e^{u_w^\top v_c}.$$

(ii)

We have from part (i)

$$\frac{\partial J}{\partial v_c} = U(\hat{y} - y).$$

Hence the gradient is zero exactly when

$$U(\hat{y} - y) = 0.$$

In particular, if there is no non-zero vector in the null space of U that equals $\hat{y} - y$, then this implies $\hat{y} - y = 0$, i.e.

$$\hat{y} = y,$$

meaning the predicted distribution equals the true (one-hot) distribution. In practice one therefore interprets the zero-gradient condition as the model having correctly predicted the outside word.

(iii)

From part (i) we have

$$\frac{\partial J}{\partial v_c} = U(\hat{y} - y) = \sum_w \hat{y}_w u_w - u_o.$$

In gradient descent the update is

$$v_c \leftarrow v_c - \eta \frac{\partial J}{\partial v_c} = v_c - \eta\Big(\sum_w \hat{y}_w u_w\Big) + \eta u_o.$$

Thus the update decomposes into two intuitive forces:

- $-\eta \sum_w \hat{y}_w u_w$: a *repulsive* term that moves v_c away from the model's current predicted average of outside-word vectors. The contribution of each u_w is weighted by $\hat{y}_w$, so words that the model currently (incorrectly) assigns high probability exert a stronger push.
- $+\eta u_o$: an *attractive* term that pulls v_c toward the true outside-word vector u_o, increasing their inner product and thus the predicted probability of the correct outside word.

In summary, subtracting the gradient updates v_c by simultaneously pulling it toward the true outside word and pushing it away from incorrectly predicted words (with strength proportional to their predicted probabilities). When $\hat{y}$ concentrates on the correct word ($\hat{y}_o \approx 1$), these forces vanish and the update becomes small.

(c)

The L_2 normalization of a vector u is $\tilde{u} = u/\|u\|_2$, which preserves the direction of u but removes its magnitude. Whether this loses useful information depends on whether the downstream task relies on vector length.

When L_2 normalization *takes away* useful information. If the downstream classifier uses the *sum* or the raw dot-product of embeddings to make decisions (for example, a binary classifier that sums word embeddings of a phrase and classifies based on the sign or magnitude

of the sum), then the original norms of word vectors encode useful signals such as emphasis, intensity, frequency, or confidence. In particular, if $u_x = \alpha u_y$ with $\alpha > 0$, then

$$\frac{u_x}{\|u_x\|} = \frac{u_y}{\|u_y\|},$$

so normalization makes x and y indistinguishable and thus removes the scalar information α that could be informative for the task.

When L_2 normalization does *not* remove useful information. If the downstream task depends only on vector directions (for example using cosine similarity or nearest-neighbor retrieval with cosine), then normalization preserves the relevant information. Also, if the downstream model can learn to reintroduce appropriate scaling (e.g., via trainable scaling parameters), normalization may not hurt and can even stabilize learning by removing scale-related noise.

Conclusion. L_2 normalization discards magnitude information. It harms tasks that exploit vector norms (such as classifiers relying on sums or raw magnitudes), but is harmless or beneficial when only directions matter (e.g., cosine-based retrieval) or when downstream models can recover scale.

(d)

Since U consists of the outside word vectors as its columns,

$$U = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_{|V|} \\ | & | & & | \end{bmatrix},$$

its gradient with respect to the loss is simply the matrix whose columns are the partial derivatives with respect to each u_w:

$$\frac{\partial J(v_c, o, U)}{\partial U} = \begin{bmatrix} \frac{\partial J}{\partial u_1} & \frac{\partial J}{\partial u_2} & \cdots & \frac{\partial J}{\partial u_{|V|}} \end{bmatrix}.$$

(e)

The loss can be written as

$$J = -u_o^\top v_c + \log \sum_w e^{u_w^\top v_c}.$$

For a fixed index w, the derivative of the first term is $-v_c$ when $w = o$ and 0 otherwise. The derivative of the second term is

$$\frac{\partial}{\partial u_w} \log \Big(\sum_{w'} e^{u_{w'}^\top v_c} \Big) = \frac{e^{u_w^\top v_c}}{\sum_{w'} e^{u_{w'}^\top v_c}} v_c = \hat{y}_w \, v_c.$$

Therefore, for each w,

$$\frac{\partial J}{\partial u_w} = \hat{y}_w v_c - \begin{cases} v_c, & w = o, \\ 0, & w \neq o. \end{cases}$$

Equivalently,

$$\boxed{\frac{\partial J}{\partial u_w} = (\hat{y}_w - y_w) \, v_c}$$

so in particular

$$\frac{\partial J}{\partial u_o} = (\hat{y}_o - 1) v_c, \qquad \frac{\partial J}{\partial u_w} = \hat{y}_w v_c \ (w \neq o).$$

2 Machine Learning & Neural Networks

(a)

(i)

Answer here.

(ii)

Answer here.

(b)

(i)

Answer here.

(ii)

Answer here.

3 Neural Dependency Parsing

(a)

Answer here.

(b)

Answer here.

(c)

Answer here.

(d)

Answer here.

(e)

Write about your model, training, and UAS scores.

(f)

i. Answer here.

ii. Answer here.

iii. Answer here.

iv. Answer here.

(g)

Answer here.