Practical No. 2

Emilia Wiśnios
ew407219@students.mimuw.edu.pl

March 20, 2022

Part 1

a)

We know that $\boldsymbol{y}$ is a one-hot vector with a 1 for the true outside word o, and 0 everywhere else. With that information we can say that

$$\begin{aligned} -\sum_{w \in \text{Vocab}} \boldsymbol{y_w} \log(\hat{\boldsymbol{y}}_{\boldsymbol{w}}) &= -\left(y_1 \log(\hat{y}_1) + ... + y_o \log(\hat{y}_o) + ... + y_w \log(\hat{y}_w)\right) \\ &= -\left(0 \cdot (\hat{y}_1) + ... + 1 \cdot \log(\hat{y}_o) + ... + 0 \cdot \log(\hat{y}_w)\right) \\ &= -\log(\hat{y}_o) \end{aligned}$$

■

b)

From the previous subpoint we know that the cross-entropy loss is equivalent to naive-softmax loss for one hot encoded vector. Let's start with finding the derivative of cross-entropy loss with softmax using the chain rule.

$$L = -\sum_i y_i \log(\text{softmax}(y_i))$$

$$\text{softmax}(y_k) = p_k$$

$$\begin{aligned} \frac{\partial}{\partial w_i} L &= -\sum_k y_k \frac{\partial \log(p_k)}{\partial w_i} \\ &= -\sum_k y_k \frac{\partial \log(p_k)}{\partial p_k} \cdot \frac{\partial p_k}{\partial w_i} \\ &= -\sum_k y_k \frac{1}{p_k} \cdot \frac{\partial p_k}{\partial w_k} \end{aligned}$$

Now we have to find the derivative of softmax.

$$\text{softmax}(y_i) = \frac{\exp(y_i)}{\sum_k \exp y_k}$$

$$\frac{\partial p_i}{\partial y_j} = \frac{\partial \frac{\exp(y_i)}{\sum_k \exp(y_k)}}{\partial y_j}$$

For $i = j$:

$$\begin{aligned}
\frac{\partial \frac{\exp(y_i)}{\sum_k \exp(y_k)}}{\partial y_j} &= \frac{\exp(y_i)\sum_k \exp(y_k) - \exp(y_j)\exp(y_i)}{(\sum_k \exp(y_k))^2} \\
&= \frac{\exp(y_i)}{\sum_k \exp(y_k)} \cdot \frac{\sum_k \exp(y_k) - \exp(y_j)}{\sum_k \exp(y_k)} \\
&= p_i(1 - p_j)
\end{aligned}$$

For $i \neq j$:

$$\begin{aligned}
\frac{\partial \frac{\exp(y_i)}{\sum_k \exp(y_k)}}{\partial y_j} &= \frac{0 - \exp(y_j)\exp(y_i)}{(\sum_k \exp(y_k))^2} \\
&= -\frac{\exp(y_j)}{(\sum_k \exp(y_k))^2} \cdot \frac{\exp(y_i)}{(\sum_k \exp(y_k))^2} \\
&= -p_j p_i
\end{aligned}$$

Given that we can come back to our cross-entropy derivative

$$\begin{aligned}
\frac{\partial L}{\partial w_i} &= -\sum_k y_k \frac{1}{p_k} \cdot \frac{\partial p_k}{\partial w_k} \\
&= -y_i(1 - p_i) - \sum_{k \neq i} y_k \frac{1}{p_k}(-p_k \cdot p_i) \\
&= -y_i(1 - p_i) + \sum_{k \neq i} y_k p_i \\
&= p_i \left(y_i + \sum_{k \neq i} y_k \right) - y_i \\
&= p_i - y_i
\end{aligned}$$

The last equality is satisfied because i is one-hot encoded vector, so $\sum_k y_k = 1$.
Now let

$$\theta = U^T v_c.$$

We want to compute the partial derivative of $J_{\text{naive-softmax}}$ with respect to v_c. We've already showed that

$$\frac{\partial L}{\partial \theta} = (p_\theta - y)^T$$

Now, using the chain rule we can compute the final derivative:

$$\begin{aligned}\frac{\partial J_{\text{naive-softmax}}}{\partial v_c} &= \frac{\partial L}{\partial \theta}\frac{\partial \theta}{\partial v_c} \\ &= (p_\theta - y)^T \frac{\partial U^T v_c}{\partial v_c} \\ &= U^T(p_\theta - y)\end{aligned}$$

In our task $p_\theta = \hat{y}$ so

$$\frac{\partial J_{\text{naive-softmax}}}{\partial v_c} = U^T(\hat{y} - y)$$

c)

From the subpoint above

$$\begin{aligned}\frac{\partial J_{\text{naive-softmax}}}{\partial U} &= \frac{\partial L}{\partial \theta}\frac{\partial \theta}{\partial U} \\ &= (\hat{y} - y)^T \frac{\partial U^T v_c}{\partial U} \\ &= (\hat{y} - y)^T v_c\end{aligned}$$

d)

We have

$$\begin{aligned}\frac{d}{dx}\frac{\exp(x)}{\exp(x)+1} &= \frac{\exp(x)\cdot(\exp(x)+1) - \exp(x)^2}{(\exp(x)+1)^2} \\ &= \frac{\exp(x)}{(\exp(x)+1)^2} \\ &= \sigma(x)\cdot\frac{1}{\exp(x)+1} \\ &= \sigma(x)\cdot\frac{\exp(x)+1-\exp(x)}{\exp(x)+1} \\ &= \sigma(x)\cdot(1-\sigma(x))\end{aligned}$$

e)

We have

$$J_{\text{neg-sample}}(v_c, o, U) = -\log(\sigma(u_o^T v_c)) - \sum_k^K \log(\sigma(-u_k^T v_c))$$

Let's start with partial derivative of $J_{\text{neg-sample}}$ with respect to u_o.

$$\begin{aligned}\frac{\partial J_{\text{neg-sample}}}{\partial u_o} &= -\frac{1}{\sigma(u_o^T v_c)}\cdot\sigma(u_o^T v_c)\cdot(1-\sigma(u_o^T v_c))\cdot v_c \\ &= (\sigma(u_o^T v_c) - 1)v_c\end{aligned}$$

For u_k we have:

$$\begin{aligned}\frac{\partial J_{\text{neg-sample}}}{\partial u_k} &= -\frac{1}{\sigma(-u_k^T v_c)} \cdot \sigma(-u_k^T v_c) \cdot (1 - \sigma(u_k^T v_c)) \cdot v_c \\ &= (\sigma(-u_k^T v_c) - 1) v_c, \quad \forall k \in [1, K]\end{aligned}$$

Finally, for v_c we have

$$\begin{aligned}\frac{\partial J_{\text{neg-sample}}}{\partial v_c} &= -\frac{1}{\sigma(u_o^T v_c)} \cdot \sigma(u_o^T v_c) \cdot (1 - \sigma(u_o^T v_c)) \cdot u_o - \sum_k^K -\frac{1}{\sigma(-u_k^T v_c)} \cdot \sigma(-u_k^T v_c) \cdot (1 - \sigma(-u_k^T v_c)) u_k \\ &= (\sigma(u_o^T v_c) - 1) u_o - \sum_k^K (\sigma(-u_k^T v_c) - 1) u_k\end{aligned}$$

The negative sampling loss is more efficient to compute than the naive-softmax loss because in negative sampling loss we don't have to compute whole matrix U for outside vectors – we only calculate vectors for fixed size K.

f)

(i)

$$\frac{\partial J_{\text{skip-gram}}(v_c, w_{t-m}, ..., w_{t+m})}{\partial U} = \sum_{\substack{-m \le j \le m \\ j \neq 0}} \frac{\partial J(v_c, w_{t+j}, U)}{\partial U}$$

(ii)

$$\frac{\partial J_{\text{skip-gram}}(v_c, w_{t-m}, ..., w_{t+m})}{\partial v_c} = \sum_{\substack{-m \le j \le m \\ j \neq 0}} \frac{\partial J(v_c, w_{t+j}, U)}{\partial v_c}$$

(iii)

$$\frac{\partial J_{\text{skip-gram}}(v_c, w_{t-m}, ..., w_{t+m})}{\partial w_c} = 0 \text{ for } w \neq c$$

Part 2

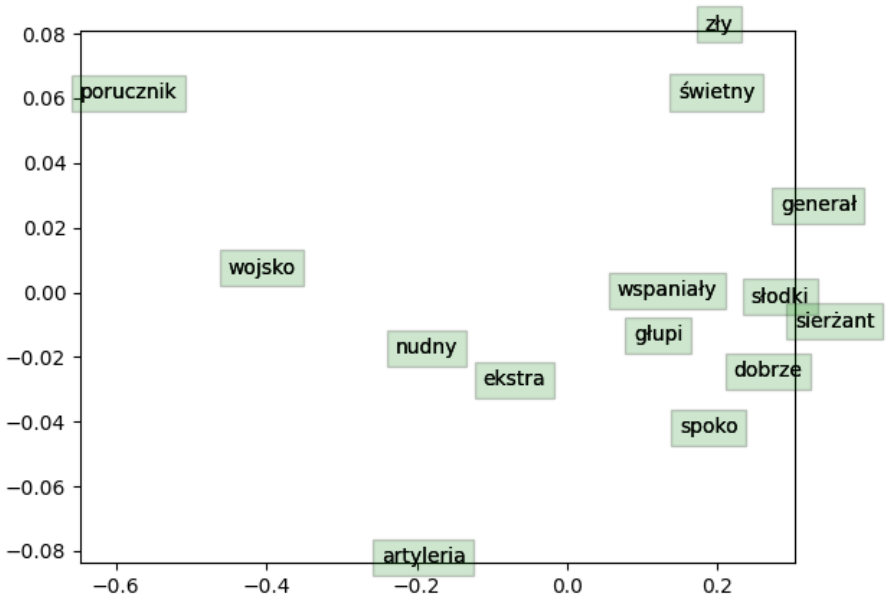

Figure 1: Visualization of word vectors after 40k iterations.

In this task we wanted to plot word vectors for two groups of words – one group of words connected to army (*porucznik, wojsko, artyleria, generał, sierżant*) and second group of words adjectives both positive and negative (*zły, świetny, nudny, ekstra, wspaniały, słodki, głupi, dobrze, spoko*). In the second plot we can see that three of the army words form a cluster (*porucznik, wojsko, artyleria*). Remaining two (*generał, sierżant*) are close to each other, but far from the cluster. Regarding second group of words – positive and negative adjectives are not really separated from each other. There are no visible clusters.

Part 3

a)

Momentum makes the updates smoother. It generally helps pointing the gradients towards the long term direction. It handles noisy gradients better – in consequence we're closer to the actual derivative.
Overall, momentum would reduce oscillations of gradient and speed up convergence.

b)

Division by $\sqrt{v}$ normalizes the parameters updates – the parameters with the smallest on average gradients will get the larger update. It helps parameters learn at similar pace and

speeds up convergence.

c)

L2 regularization can be helpful in low-data setting, because it can prevent overfitting (it leads to small coefficients).
Authors of AdamW propose adding decouped weight decay.
[...] decoupled weight decay regularizes all weights with the same rate λ, effectively regularizing weights x with large s more than standard L2 regularization

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Figure 2: AdamW pseudocode from *Decoupled Weight Decay Regularization*

d)

1. We use to improve robustness of the model and avoid overfitting on training set. During evaluation we want to use that robust model to check its performance.

2. Given a neural network with dropout (in testing), we obtain different output every forward pass for the same output. Linked paper shows that those passes are equivalent to Monte-Carlo sampling. Mean and variance of obtained outputs provides network's output and uncertainty.