Practical No. 2

Rafał Michaluk
rm360179@students.mimuw.edu.pl

March 21, 2022

1 Part 1

a)

$y_w = 1$ if and only if $w = o$, so everything but y_o zeros out. It leaves us with the right hand value.

b)

$$J(v_c, o, U) = -logP(O = 0|C = c)$$

$$P(O = o|C = c) = \frac{exp(u_o^T v_c)}{\sum_{w \in VOC} exp(u_w^T v_c)}$$

Let's simplify:

$$J(v_c, o, U) = -log(\frac{exp(u_o^T v_c)}{\sum_{w \in VOC} exp(u_w^T v_c)}) = -u_o^T v_c + log(\sum_{w \in VOC} exp(u_w^T v_c))$$

We will replace $v_c[i]$ with x and consider it a variable. Then we'll calculate a derivative of J with respect to x

$$J(v_c, o, U)' = (-u_o^T v_c + log(\sum_{w \in VOC} exp(u_w^T v_c)))' = -u_o[i] + (\sum_{w \in VOC} exp(u_w^T v_c))^{-1} \sum_{w \in VOC} exp(u_w^T v_c) u_w[i]$$

$$= -u_o[i] + \sum_{w \in VOC} P(O = w|C = c) u_w[i] = (U\hat{y} - Uy)[i]$$

or a vector version

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

c)

$$J(v_c, o, U) = -u_o^T v_c + log(\sum_{w \in VOC} exp(u_w^T v_c))$$

For some k we will replace $u_k[i]$ with x and consider it a variable. Then we'll calculate a derivative of J with respect to x

If $k = o$:

$$J(v_c, o, U)' = -v_c[i] + (\sum_{w \in VOC} exp(u_w^T v_c))^{-1} exp(u_k^T v_c) v_c[i] = -v_c[i] + \hat{y}[k] v_c[i]$$

If $k \neq o$:

$$J(v_c, o, U)' = 0 + \hat{y}[k] v_c[i]$$

These ifs can be simplified to:

$$J(v_c, o, U)' = -v_c[i] y[k] + \hat{y}[k] v_c[i] = v_c[i](\hat{y}[k] - y[k])$$

or a vector version:

$$(\hat{y}[k] - y[k]) v_c$$

We can also write derivatives to u_k for all k at the same time using matrices:

$$\frac{\partial J(v_c, o, U)}{\partial U} = v_c(\hat{y} - y)^T$$

d)

$$\sigma(x) = \frac{1}{1 + exp(-x)}$$

$$\sigma(x)' = ((1 + exp(-x))^{-1})' = (-1)(1 + exp(-x))^{-2} exp(-x)(-1)$$

$$= \frac{exp(-x)}{(1 + exp(-x))^2} = \frac{1}{1 + exp(-x)} \frac{exp(-x)}{1 + exp(-x)} = \sigma(x) \frac{1 + exp(-x) - 1}{1 + exp(-x)} = \sigma(x)(1 - \sigma(x))$$

$$\sigma(x)' = \sigma(x)(1 - \sigma(x))$$

e)

$$J(v_c, o, U) = -log(\sigma(u_o^T v_c)) - \sum_k^K log(\sigma(-u_k^T v_c))$$

We will replace $v_c[i]$ with x and consider it a variable. Then we'll calculate a derivative of J with respect to x.

$$J(v_c, o, U)' = (-1)\sigma(u_o^T v_c)^{-1} \sigma(u_o^T v_c)' u_o[i] - \sum_k^K \sigma(-u_k^T v_c)^{-1} \sigma(-u_k^T v_c)'(-1) u_k[i]$$

$$\sum_{k}^{K}[u_k[i](1-\sigma(-u_k^T v_c))] - (1-\sigma(u_o^T v_c)u_o[i]$$

Or in vector version:

$$\sum_{k}^{K}[u_k(1-\sigma(-u_k^T v_c))] - (1-\sigma(u_o^T v_c)u_o$$

For some $p \neq o$, we will replace $u_p[i]$ with x and consider it a variable. Then we'll calculate a derivative of J with respect to x.

$$J(v_c, o, U)' = (-log(\sigma(-u_p^T v_c)))' = (-1)\sigma(-u_p^T v_c)\sigma(-u_p^T v_c)'(-1)v_c[i] = v_c[i](1-\sigma(-u_p^T v_c))$$

or in vector version:

$$v_c(1-\sigma(-u_p^T v_c))$$

We will replace $u_o[i]$ with x and consider it a variable. Then we'll calculate a derivative of J with respect to x.

$$J(v_c, o, U)' = (-1)\sigma(u_o^T v_c)^{-1}\sigma(u_o^T v_c)' v_c[i] = (-1)v_c[i](1-\sigma(u_o^T v_c))$$

or in vector version

$$-v_c(1-\sigma(u_o v_c))$$

It's faster, because we don't need to use the entire matrix U to calculate gradients over v_c, only some K random columns.

f)

$$\frac{\partial J(v_c, w_{t-m}, ..., w_{t+m}, U)}{\partial U} = \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial J(v_c, w_{t+j}, U)}{\partial U}$$

$$\frac{\partial J(v_c, w_{t-m}, ..., w_{t+m}, U)}{\partial v_c} = \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial J(v_c, w_{t+j}, U)}{\partial v_c}$$

If $w \neq c$

$$\frac{\partial J(v_c, w_{t-m}, ..., w_{t+m}, U)}{\partial v_w} = \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial J(v_c, w_{t+j}, U)}{\partial v_w} = 0$$

because v_w doesn't occur in loss then

summary

$$\frac{\partial J_{naive}(v_c, o, U)}{\partial v_c} = U(\hat{y} - y)$$

$$\frac{\partial J(v_c, o, U)}{\partial U} = v_c(\hat{y} - y)^T$$

$$\sigma(x)' = \sigma(x)(1 - \sigma(x))$$

$$\frac{\partial J_{neg}(v_c, o, U)}{\partial v_c} = \sum_k^K [u_k(1 - \sigma(-u_k^T v_c))] - (1 - \sigma(u_o^T v_c)u_o$$

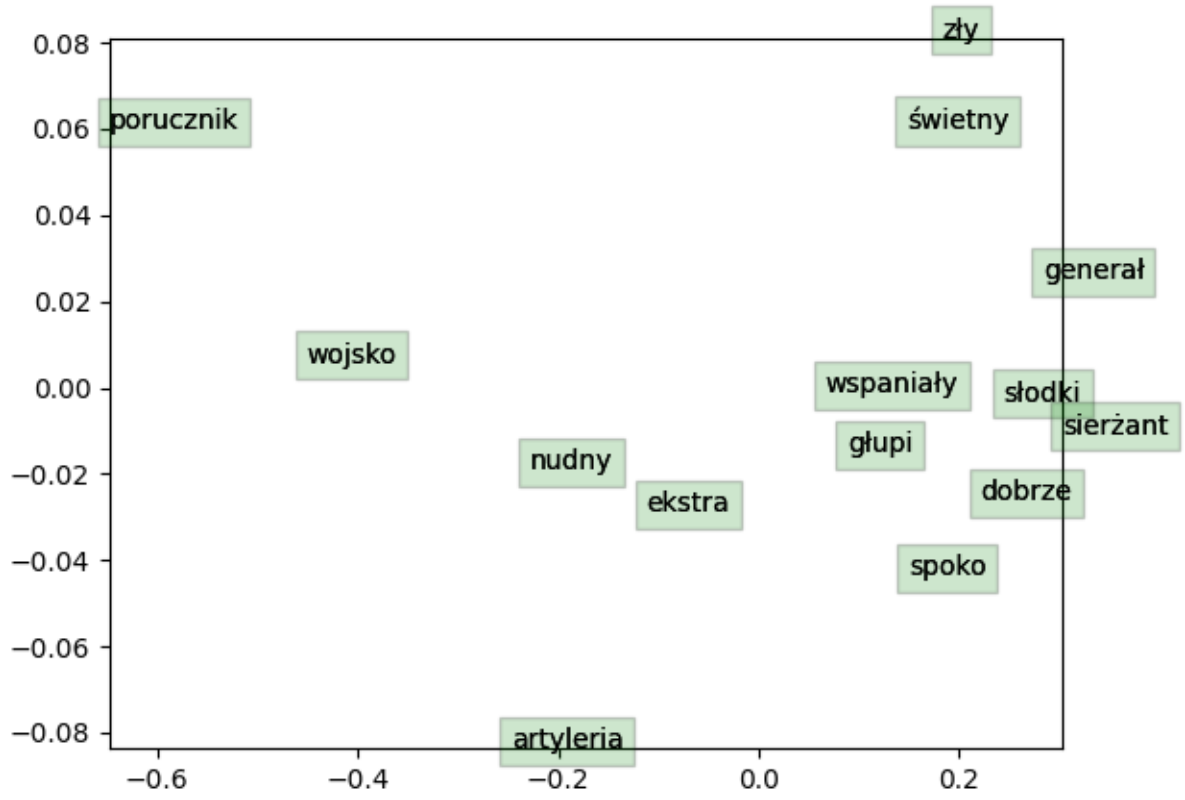
$p \neq o$:

$$\frac{\partial J_{neg}(v_c, o, U)}{\partial u_p} = v_c(1 - \sigma(-u_p^T v_c))$$

$$\frac{\partial J_{neg}(v_c, o, U)}{\partial u_o} = -v_c(1 - \sigma(u_o^T v_c))$$

2 Part 2

Word embeddings from word2vec after applying svd to reduce the dimensonality, these words are pretty similar, because the ranges displayed are very small. Zły is close to świetny, they have similar role in statements, the same with nudny and ekstra. For some reason generał/sierżant is far from porucznik. There are many masculine adjectives close to each other.



3 Part 3

a)

Momentum averages the the gradients of previous steps, where the older the gradient, the less important. It makes the algorithm move in one, more "stable" direction instead of jumping everywhere. This can help in moving directly into minimum instead of jumping indirectly and oscillate (it means also less steps).

b)

The more iterations, the larger m will be, which will make learning slower and slower, which may help finding the minimum more precisely. It also balances dimensions with very high gradients.

c)

L2 makes the parameters go towards zero, which prevents training very complicated models, so it prevents overfitting. Citing the authors, they've improved regularization in Adam

by decoupling the weight decay from the gradient-based update. They simply moved the regularization term directly to weight update

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
$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$

d)

Question 1: It would defeat the purpose of dropout (which is supposed to reduce overfitting and make the network more "robust" and not rely on single neurons). Leaving the neurons disabled during testing would be very similar to just reducing the size of the network. Also evaluation would be unpredictable, because random neurons are being removed.

Question 2: From Bayesian theory perspective, if the dropout is applied during interference, predictions are viewed as sampling from a probabilistic distribution. It means that if we sample multiple times, we could see what the distribution looks like. (in Bayesian networks all predictions are distributions of probability, not single values like in classical NN).