Practical No. 2

Adriana Bukała
ab394064@students.mimuw.pl

October 9, 2022

1 Part 1

a)

$$-\sum_{w \epsilon Vocab} y_w \log(\hat{y_w}) = -\sum_{w \epsilon Vocab} \mathbf{1}_{w=o} \log(\hat{y_w}) = -\log(\hat{y_o})$$

b)

$$\nabla_{v_c} \mathbf{J}_{naive-softmax}(v_c, o, U) = \nabla_{v_c} - \frac{\log\left(\exp\left(u_o^\intercal v_c\right)\right)}{\sum_{w \epsilon Vocab} \log\left(\exp\left(u_w^\intercal v_c\right)\right.} =$$

$$-\nabla_{v_c} \log\left(\exp\left(u_o^\intercal v_c\right) + \nabla_{v_c} \sum_{w \epsilon Vocab} \log\left(\exp\left(u_w^\intercal v_c\right) = -\nabla_{v_c} u_o^\intercal v_c + \nabla_{v_c} \log \sum_{w \epsilon Vocab} \exp\left(u_w^\intercal v_c\right) =\right.\right.$$

$$-u_o + \frac{1}{\sum_{w' \epsilon Vocab} \exp\left(u_{w'}^\intercal v_c\right)} \sum_{w \epsilon Vocab} \exp\left(u_w^\intercal v_c\right) u_w = -u_o + \sum_{w \epsilon Vocab} \frac{\exp\left(u_w^\intercal v_c\right)}{\sum_{w' \epsilon Vocab} \exp\left(u_{w'}^\intercal v_c\right)} u_w =$$

$$\sum_{w \epsilon Vocab} u_w(-y_w + \hat{y_w}) = \mathbf{U}^\intercal(\hat{y} - y)$$

c)

$$\nabla_{u_w} \mathbf{J}_{naive-softmax}(v_c, o, U) = \nabla_{u_w} - \frac{\log\left(\exp\left(u_o^\intercal v_c\right)\right.}{\sum_{w' \epsilon Vocab} \log\left(\exp\left(u_{w'}^\intercal v_c\right)\right.} =$$

$$\nabla_{u_w} - u_o^\intercal v_c + \nabla_{u_w} \log \sum_{w' \epsilon Vocab} \exp\left(u_{w'}^\intercal v_c\right) =$$

$$\mathbf{1}_{w=o} v_c + \frac{1}{\sum_{w' \epsilon Vocab} \exp\left(u_{w'}^\intercal v_c\right)} \sum_{w' \epsilon Vocab} \mathbf{1}_{w'=w} \exp\left(u_w^\intercal v_c\right) v_c =$$

$$\sum_{w \epsilon Vocab} (\hat{y_w} - y_w) v_c = (\hat{y} - y)^\intercal v_c$$

d)

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

e)

$$\mathbf{J}_{neg-sample}(v_c, o, U) = -\log\sigma(u_o^\intercal v_c) - \sum_{k\epsilon K}\log\sigma(-(u_k^\intercal v_c))$$

$$\nabla_{v_c}\mathbf{J}_{neg-sample}(v_c, o, U) = -\frac{1}{\sigma(u_o^\intercal v_c)}\sigma(u_o^\intercal v_c)(1-\sigma(u_o^\intercal v_c))u_o$$
$$+\sum_{k\epsilon K}\frac{1}{\sigma(-u_k^\intercal v_c)}\sigma(-u_k^\intercal v_c)(1-\sigma(-u_k^\intercal v_c))u_k =$$
$$(\sigma(u_o^\intercal v_c)-1)u_o + \sum_{k\epsilon K}(1-\sigma(-u_k^\intercal v_c))u_k =$$
$$(\sigma(u_o^\intercal v_c)-1)u_o - \sum_{k\epsilon K}(\sigma(-u_k^\intercal v_c)-1)u_k$$

$$\nabla_{u_o}\mathbf{J}_{neg-sample}(v_c, o, U) = (\sigma(u_o^\intercal v_c)-1)v_c$$

$$\nabla_{u_k}\mathbf{J}_{neg-sample}(v_c, o, U) = \sum_{k\epsilon K}(\sigma(-u_k^\intercal v_c)-1)v_c$$

This loss is much more efficient to compute than naive softmax loss, because it's not using the whole vocabulary, but only K drawn samples (words).

f)

$$\frac{\partial\mathbf{J}_{skip-gram}(v_c, w_{t-m}, ..., w_{t+m}, \mathbf{U})}{\partial\mathbf{U}} = \partial\mathbf{U}\sum_j\mathbf{J}_{skip-gram}(v_c, w_{t+j}, \mathbf{U}) =$$
$$\sum_j\partial\mathbf{U}\mathbf{J}_{skip-gram}(v_c, w_{t+j}, \mathbf{U})$$

$$\frac{\partial\mathbf{J}_{skip-gram}(v_c, w_{t-m}, ..., w_{t+m}, \mathbf{U})}{\partial v_c} = \partial v_c\sum_j\mathbf{J}_{skip-gram}(v_c, w_{t+j}, \mathbf{U}) =$$
$$\sum_j\partial v_c\mathbf{J}_{skip-gram}(v_c, w_{t+j}, \mathbf{U})$$

$$\frac{\partial\mathbf{J}_{skip-gram}(v_c, w_{t-m}, ..., w_{t+m}, \mathbf{U})}{\partial v_w} = \partial v_w\sum_j\mathbf{J}_{skip-gram}(v_c, w_{t+j}, \mathbf{U}) = \mathbf{0}$$

when w != c

2 Part 2

c)

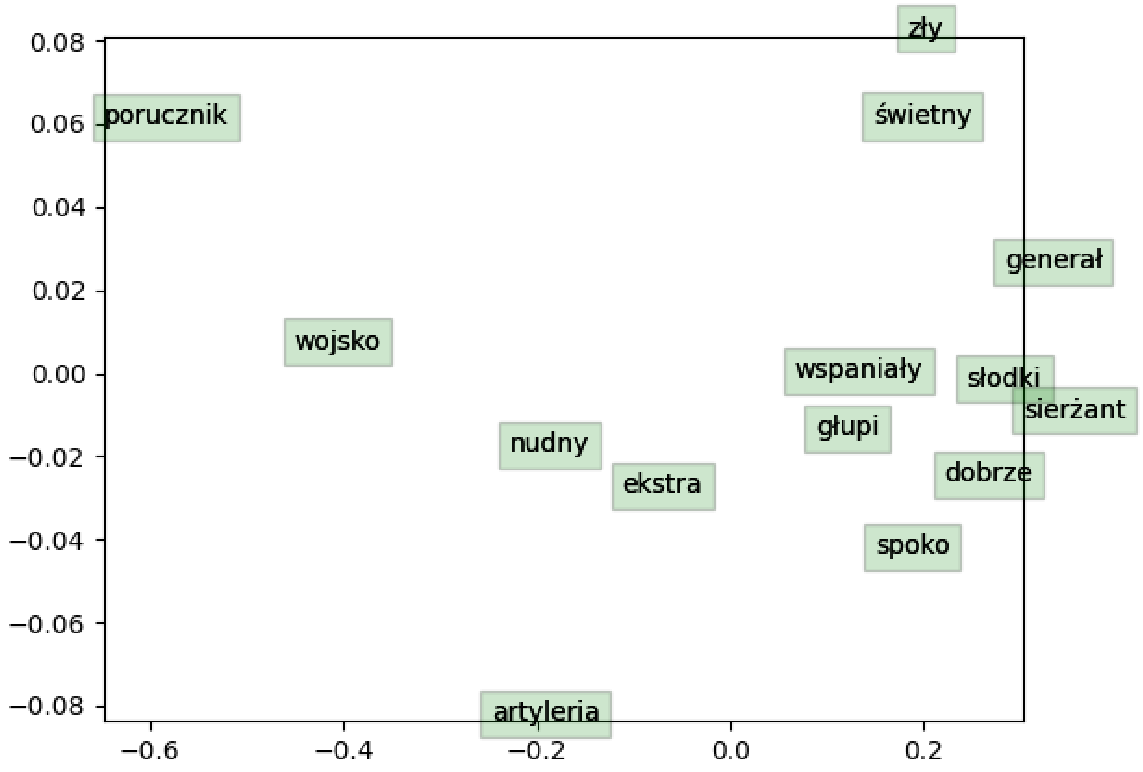


Figure 1: 2D visualization of word vectors after 40k epochs of training

Let's categorize visualized words in two types: 1) adjectives and adverbs, 2) military vocabulary. Words from first category are intuitively assigned to a suitable cluster, i.e. their meaning was well captured by skip-gram model, although some of them are closer to their antonyms than synonyms (for example "świetny" seems to be more similar to "zły" than "wspaniały" or "ekstra"). Grip of meaning of words from the second group is not quite there, what is probably due to rareness of these words, so that model didn't have enough data (used context) or time to learn similarity between them.

3 Part 3

a)

Without using momentum, update step is controlled completely by the current minibatch. In the ideal world, each minibatch would be a good approximation of the whole dataset, so each update step would lead to better and better understanding (generalization) of used data. But often that's not a case, meaning that minibatch usually approximates only a part of dataset, contains outliers and so on. That could lead to bad generalization or "walking" back and forth between learning representations of examples from different classes. Momentum introduces weighted average (controlled by hyperparameter β_1) between current gradient and rolling average of previous gradients. Since β_1 is usually a high number (0.9), influence of current gradient is decreased, so overall variance of updates is decreased as well. Lower variance stabilizes learning process, so that model could learn good data representation without "giving" too much attention to outliers (which are increasing variance of whole dataset).

b)

Dividing updates by $\sqrt{v}$ (moving average of magnitudes of gradients) results in decrease of final updates for parameters with high gradients and increase of final updates of parameters with low gradients. Since gradient is computed starting from the last layer, early layers usually have low gradients. So scaling updates with $\sqrt{v}$ would increase their magnitude, leading to better efficiency of the model, since early layers are vital for achieving good generalization.

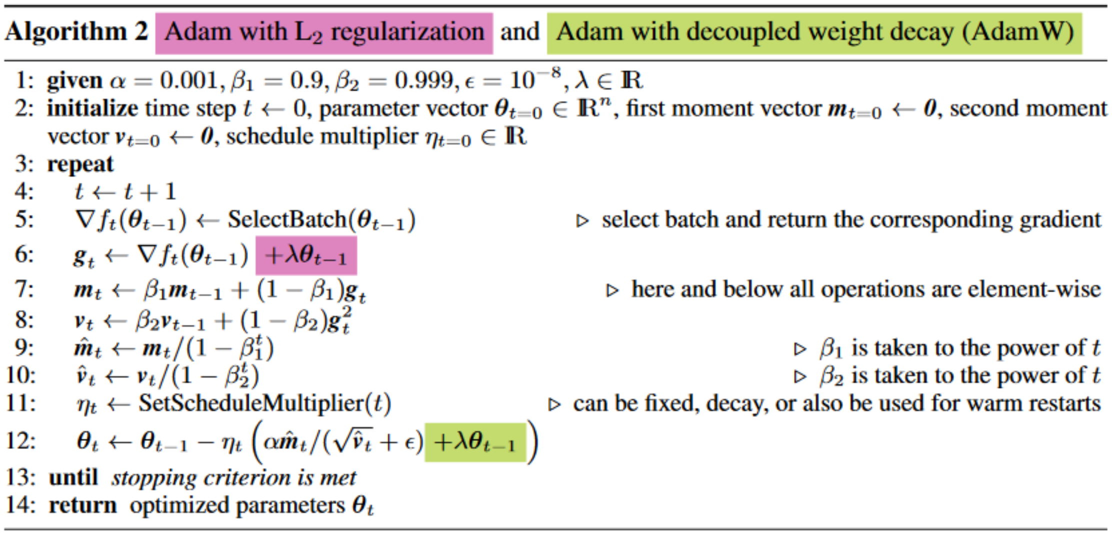
*c)

$L2$ regularization is helpful, because it reduces the risk of overfitting, which is often the case in low-data settings, since model doesn't have enough samples to generalize the data.

Adam optimizer, being an adaptive scheme, characterizes with *adapting* parameter updates to magnitude of their gradients (as explained in the previous subsection). That means that regularization factor from step 6 in Figure 2 would be scaled by $\sqrt{v}$ in step 12 as well, i.e.:

$$\theta_t \leftarrow \theta_{t-1} - \eta_t(\alpha \frac{\beta_1 m_{t-1} + (1-\beta_1)(\nabla_{f_t}(\theta_{t-1}) + \lambda\theta_{t-1})}{1-\beta_1^t}) \frac{1}{\sqrt{\hat{v}_t} + \epsilon}$$

If gradient of certain weight is large, corresponding v is large as well, so $\sqrt{v}$ downscales it. But that means that weights with large gradient are regularized less than weights with low gradient, so $L2$ regularization doesn't work as intended. And weight decay as well, since it's often seen as identical. That being said, authors of AdamW proposed decoupling weight decay from the optimization schemes (marked with green box on Figure 2). Separation of weight decay and adaptive gradient scheme restores the initial idea behind weight decay.



Algorithm 2 Adam with L$_2$ regularization and Adam with decoupled weight decay (AdamW)

```
1:  given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
2:  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 $\eta_{t=0} \in \mathbb{R}$
3:  repeat
4:      $t \leftarrow t + 1$
5:      $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$          ▷ select batch and return the corresponding gradient
6:      $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$ $+\lambda\boldsymbol{\theta}_{t-1}$
7:      $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$          ▷ here and below all operations are element-wise
8:      $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
9:      $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t/(1 - \beta_1^t)$          ▷ $\beta_1$ is taken to the power of $t$
10:     $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t/(1 - \beta_2^t)$          ▷ $\beta_2$ is taken to the power of $t$
11:     $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$          ▷ can be fixed, decay, or also be used for warm restarts
12:     $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) +\lambda\boldsymbol{\theta}_{t-1}\right)$
13: until stopping criterion is met
14: return optimized parameters $\boldsymbol{\theta}_t$
```

Figure 2: Comparison of Adam with $L2$ regularization and decoupled weight decay (AdamW)

d)

Generally speaking, purpose of training the machine learning model is to achieve the best possible generalization (understanding) of the data, while evaluation phase is for measuring performance of the model. Whole idea of dropout is to somehow force neurons to learn different features, so that in total it would result in good data generalization. With dropout only part of the potential of the model is used, so using it during evaluation would lead to not using all the available knowledge (neurons). That obviously makes no sense, since we wouldn't be measuring performance of the whole model, but just a (random!) part of it.

* Objective of using Bayesian theory in machine learning is to estimate so-called *a posteriori* distribution given as:

$$P(\theta|X,Y) = \frac{P(Y|\theta,X) * P(\theta)}{P(Y|X)}$$

where θ denotes parameters of the model, X independent variables, Y dependent variable and $P(\theta)$ is *a priori* distribution (i.e. prior beliefs about modelled subject). However, computing the posterior could be computationally exhaustive or ambiguous, since it's hard to say, how to estimate the probability of already existing data ($P(Y|X)$). Instead of computing all these distributions, one could try to approximate them using sampling methods such as Monte Carlo Markov Chain (MCMC), i.e. compute posterior on sample of the data and estimate true posterior based on it. Whole idea of sampling is to choose k examples from the data and learn them, while ignoring the rest, i.e. setting all $n - k$ non-chosen ones to zero, just as when applying dropout to neurons. It's worth noticing that Bayesian theory has application in uncertainty estimation, so this transition between Bayesian theory and dropout makes uncertainty estimation in neural networks less computationally exhaustive.