1. Monolingual embeddings

See code.

2. Multilingual word embeddings

Using the distributivity of the inner product (w.r.t to the Frobenius norm):

$$\begin{aligned} \|WX - Y\|_F^2 &= \langle WX - Y, WX - Y\rangle_F \\ &= \|WX\|_F^2 + \|Y\|_F^2 - 2\langle WX, Y\rangle_F \end{aligned} \tag{1}$$

With

$$\|WX\|_F^2 = \mathrm{Tr}((WX)^*(WX)) \overset{(a)}{=} \mathrm{Tr}(X^\top W^\top WX) \overset{(b)}{=} \mathrm{Tr}(X^\top X) = \mathrm{Tr}(X^*X) = \|X\|_F^2$$

where $X^* = \overline{X^\top} = X^\top$ as we are dealing with real matrices (a). (b) comes from the orthogonality of W. The same derivations hold for Y. Hence (1) can be written as:

$$\|WX - Y\|_F^2 = \|X\|_F^2 + \|Y\|_F^2 - 2\langle WX, Y\rangle_F \tag{2}$$

As the sum $\|X\|_F^2 + \|Y\|_F^2$ does not depend on W, the optimization problem can be simplified as follows:

$$W^* = \underset{W \in O_d(\mathbb{R})}{\mathrm{argmax}} \langle WX, Y\rangle_F \tag{3}$$

Then, using the fact that $\mathrm{Tr}(AB) = \mathrm{Tr}(BA)$ with $A = X^\top$ and $B = W^\top Y$:

$$\langle WX, Y\rangle_F = \mathrm{Tr}((WX)^*Y) = \mathrm{Tr}(X^\top W^\top Y) = \mathrm{Tr}(W^\top YX^\top)$$

yields (3) to be equivalent to:

$$W^* = \underset{W \in O_d(\mathbb{R})}{\mathrm{argmax}} \langle W, YX^\top\rangle_F \tag{4}$$

Now we can introduce the Singular-Value Decomposition of $YX^\top$.

$$W^* = \underset{W \in O_d(\mathbb{R})}{\mathrm{argmax}} \langle W, YX^\top\rangle_F = \underset{W \in O_d(\mathbb{R})}{\mathrm{argmax}} \langle W, U\Sigma V^T\rangle_F \tag{5}$$

where, if $X, Y \in \mathbb{R}^{d\times p}$ (and thus $YX^\top \in \mathbb{R}^{d\times d}$); $U, V \in \mathbb{O}_d(\mathbb{R})$ and $(\Sigma)_{ij} = \sigma_i \mathbb{1}_{\{i=j\}}$ with $\sigma_i > 0 \,\forall\, i \in \{1, ..., d\}$. Now, using basic properties of the trace we have that:

$$\begin{aligned} \langle W, U\Sigma V^T\rangle_F &= \mathrm{Tr}([W^\top U][\Sigma V^\top]) \\ &= \mathrm{Tr}([\Sigma][V^\top W^\top U]) \\ &= \mathrm{Tr}(V^\top W^\top U\Sigma) \\ &= \mathrm{Tr}((U^\top WV)^\top \Sigma) \\ &:= \mathrm{Tr}(Z^\top \Sigma) \\ &= \sum_{i,j} Z_{ij}\Sigma_{ij} \\ &\overset{(*)}{=} \sum_i Z_{ii}\sigma_i \end{aligned}$$

As $Z := U^\top WV$ is a product of orthogonal matrix, it is still an d-dimensional orthogonal matrix whose rows' coefficients (and columns' coefficients) sum to 1. So in any case $Z_{ii} \leq 1 \, \forall \, i \in \{1, ..., d\}$. As maximizing in W is equivalent to maximizing in Z (as Z is a linear transformation of W), we have that (*) is maximal if and only if $Z_{ii} = 1 \, \forall \, i$, that is to say $Z^* = U^\top WV = I_{d \times d} := I$. This directly implies that:

$$W^* = UV^\top \tag{6}$$

3. Sentence classification with BoV

Data set	LogReg (idf-w)	LogReg (mean)	SVM (mean)
Train set	**0.494**	0.486	0.474
Dev set	0.432	0.443	**0.444**

Table 1: Accuracy on train and dev sets for logistic regression with average of word vectors ("LogReg (mean)"); logistic regression with weighted-average ("LogReg (idf-w)"); SVM with average of word vectors ("SVM (mean)")

SVM is very slighty better on dev set and overfits less. I also tried XgBoost (which is very promising in general) but it couldn't reach 0.4 accuracy on dev set.

4. Deep Learning models for classification

Question 4.1

I chose the categorical-cross entropy. Let $(x_i, y_i) \overset{\text{iid}}{\sim} P$ with P the data's distribution which is unknown. When dealing with Neural Networks one may output, for each sample, a vector of probability whose length matches the number of classes to predict, and compare it to the ground truth labels. The last layer `Dense(5, activation='sigmoid')` combined with categorical-cross entropy loss allows to do so. Namely, we estimate the cross-entropy as the following as P is unknown :

$$H(P, Q) = -\mathbb{E}_P[\log(Q(y|x)] \approx -\frac{1}{N} \sum_{i=1}^{N} \log q(y = y_i|x_i) \tag{7}$$

where N is the number of training samples. We could also write:

$$-\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K=5} \mathbb{1}_{y_i=j} \log q(y = y_i|x_i) \tag{8}$$

With:

$$q(y = y_i|x_i) = \frac{e^{w_i^\top x + b_i}}{\sum_{j=1}^{K=5} e^{w_j^\top x + b_j}} \tag{9}$$

where w_i, b_i's are learned using back-propagation and SGD in a Deep Learning framework.

Question 4.2

In order to give some elements of comparison between question 4.2 ad 4.3, I didn't choose here to adapt a lot the initial architecture. Indeed, for this question I used a LSTM network with 64 hidden units and dropout rates both equal to 0.2. This network was trained on embedded sentences. This embedding has been produced thanks to the `keras.preprocessing.text.one_hot` and `keras.preprocessing.sequence.pad_sequences` which just consist in hashing and then padding the words belonging to the sentences. As mentioned before, I used categorical-cross entropy as loss function, Adam solver and accuracy as classification metric (batch size is set to 8). Figure 1 clearly illustrates the LSTM's overfitting from 2 epochs. Reducing the number of hidden units led to underfitting. The final question aims at producing better accuracy on dev test. Nevertheless, as we'll see now, overfitting and poor performance still remain with other more complex networks.

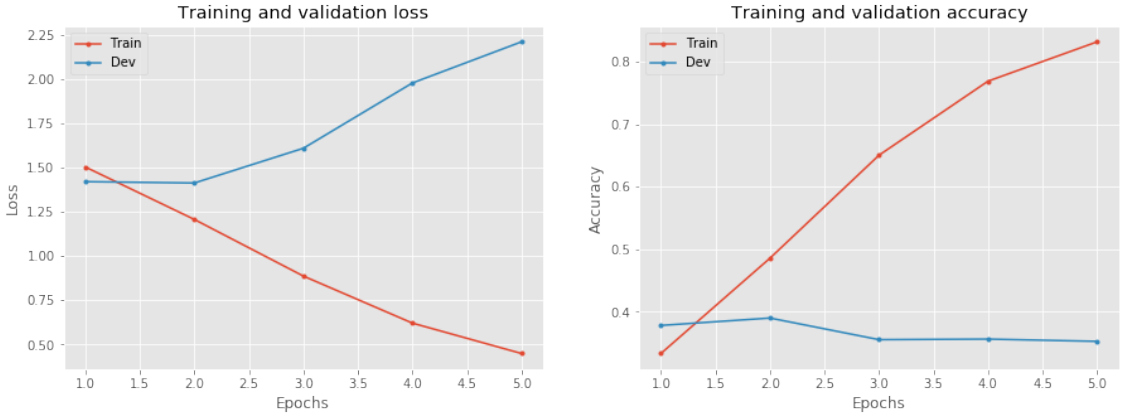

Figure 1: Train/dev results w.r.t the number of epochs

Question 4.3

Looking at the previous LSTM's poor performance despite the number of hidden units, batch size or whatever parameter to be tuned (keeping the architecture fixed), I first decided to change the encoder using pre-trained look-up tables. Namely, the embedding layer was now composed of weights representing, for each word, it's 300d-vector. To do so I alternatively chose FastText and Glove embeddings. However, the dev accuracy was never exceeding 0.4. Thus, I tested several architectures like DCNN (Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. "A convolutional neural network for modelling sentences."), LSTM and Bidirectional LSTM (Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. "Improved semantic representations from tree-structured long short-term memory networks.") and at last BLSTM-2DCNN (Peng et al. 2016. "Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling."). Obviously, to save computational resources, I adapted these architectures in reducing the number of parameters. Figure 2 is the result of my implementation of BLSTM-2DCNN for 30 epochs with Adam solver whose learning rate was forced to 10^{-5} as 10^{-3} rapidly led to a plateau (also observed here from 10 epochs...). Performance and overfitting are even worse. Actually, I always encountered such behavior from the different networks implemented. Moreover, the simple hasing technique used in question 4.2

was the only embedding technique that led to reaching 0.4 dev accuracy. The predictions of BLSTM-2DCNN are available in the zip file but won't be as good as the logistic regression or SVM. I also tried to see on GitHub how people were loading SST-1 dataset and I consequently provided little modifications on sentences during loading which led to improving the dev accuracy by about 0.1 in average. At last, as in Peng et al. (2016), I tried to train the parameters from glove or FastText in order to gain performance. It was computationally burdensome with slightly better improvement.

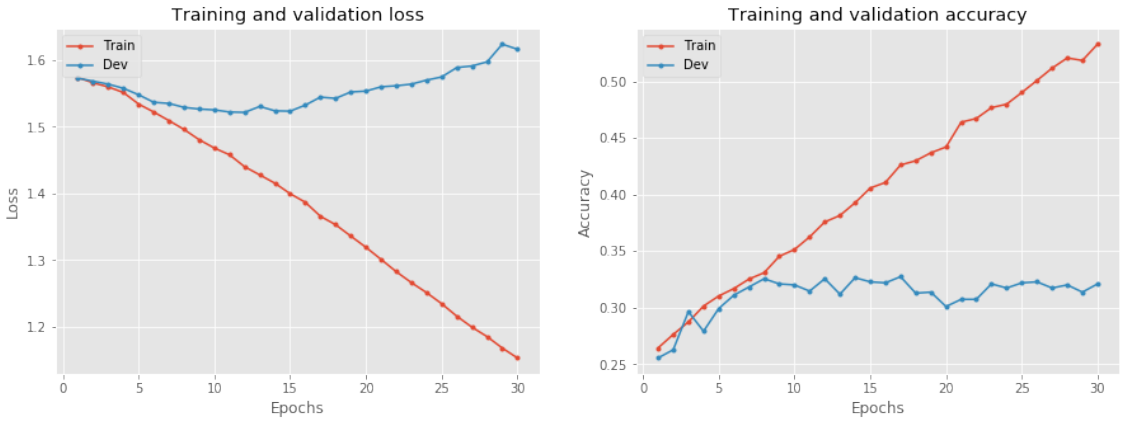

Figure 2: Train/dev results w.r.t the number of epochs