Exercises on text classification with Multi-Layer Perceptrons (MLPs)

Ion Androutsopoulos, 2024–25

Submit as a group of 2–3 members (unless specified otherwise in the lectures) a report (max. 10 pages, PDF format) for exercises 9 and 10. You may optionally submit also exercise 12 for extra bonus (your report may then be up to 15 pages). Include in your report all the required information, especially experimental results. Do not include code in the report, but include a link to a Colab notebook containing your code. Make sure to divide fairly the work of your group to its members and describe in your report the contribution of each member. The contribution of each member will also be checked during the oral examination of your submission. For delayed submissions, one point will be subtracted per day of delay.

1. In the XOR MLP of slide 10, for each one of the possible four input vectors, calculate the corresponding new feature vector that the two hidden neurons produce, i.e., the vector that contains the outputs of the two AND gates. Show that the new feature vectors are linearly separable (hence they can now be separated by the neuron of the last layer).

Answer: Let $\langle x_1, x_2 \rangle$ be the input vector, with $1, -1$ denoting True or False, respectively, as on slide 10. Let n_1, n_2 be the outputs of the top and bottom AND gates, respectively, of slide 10. The two possible input vectors for which the XOR output is False, i.e., $\langle x_1 = -1, x_2 = -1 \rangle$ and $\langle x_1 = 1, x_2 = 1 \rangle$ are both mapped to the new point $\langle n_1, n_2 \rangle = \langle -1, -1 \rangle$. The two possible input vectors for which the XOR output is True, i.e., $\langle x_1 = -1, x_2 = 1 \rangle$ and $\langle x_1 = 1, x_2 = -1 \rangle$ are mapped to the new points $\langle n_1, n_2 \rangle = \langle -1, 1 \rangle$ and $\langle n_1, n_2 \rangle = \langle 1, -1 \rangle$, respectively. The three new points (one False, two True) are now linearly separable.

2. Show that a Perceptron (single neuron) with a sigmoid activation function is a linear separator.

Answer: For each input, the Perceptron produces the output $\sigma(\vec{w} \cdot \vec{x}) = \sigma(w_n x_n + \cdots + w_1 x_1 + w_0)$, assuming that we always set $x_0 = 1$, where $\vec{w}$ is the weights vector the Perceptron has learned, $\vec{x}$ is the feature vector of the input, and σ the sigmoid function. We classify the input in the positive class if $\sigma(\vec{w} \cdot \vec{x}) \geq 0.5$, and in the negative class if $\sigma(\vec{w} \cdot \vec{x}) < 0.5$. Since $\sigma(z) \geq 0.5$ if and only if $z \geq 0$, we classify in the positive class the inputs for which $\vec{w} \cdot \vec{x} \geq 0$, i.e., the inputs whose feature vectors $\vec{x}$ are on the hyper-plane $\vec{w} \cdot \vec{x} = 0$ or in the positive semi-space of the hyper-plane $\vec{w} \cdot \vec{x} = 0$ ("above" the hyper-plane), whereas we classify in the negative class any other input. Hence, the separating hyper-surface is the hyper-plane $\vec{w} \cdot \vec{x} = 0$, which is a linear separator.

3. (a) Show that a two-level network of Perceptrons (like the one we used to implement the XOR gate on slide 10) can learn any logical function. As on slide 10: we use 1 to represent True (T) and -1 to represent False (F); we use a sign activation function in all neurons; we select appropriate weights and threshold values (or bias terms) to make the neurons of the first layer behave as AND gates and the neuron of the second layer behave as an OR gate.

X	*Y*	*Z*	*C*
F	F	F	T
F	F	T	T
F	T	F	T
F	T	T	F
T	F	F	F
T	F	T	T
T	T	F	F
T	T	T	F

Answer: Every logical function can be defined by a truth table. For example, the table on the right defines a logical function of three variables (X, Y, Z) with response C. In the first layer, we use as many neurons as the number of rows of the truth table where C = T (4 neurons in our example). Each neuron of the first layer takes as inputs the variables (X, Y, Z). We make sure that each neuron of the first layer works as an AND gate that fires (outputs T) when the variables have the values of a raw (different per neuron) of the truth table where

C = T and only then. In our example, the neuron for the first row of the truth table could have weights set to –0.3 for all of its three inputs and threshold set to 0.8, so that it would fire if and only if all three inputs are F (–1). The first-layer neuron for the second row of the truth table, could have weights set to –0.3 for X and Y, but set to +0.3 for Z, and threshold set to 0.8, so that it would fire if and only if X = –1 (F), Y = –1 (F), Z = 1 (T). Similarly, for the other first-layer neurons for the rows of the truth table where C = T. Here, for the single neuron of the output layer to operate as an OR gate, we could set all of its input weights to 0.5 and its threshold to –1.5.

(b) Explain why an MLP like the one of the previous question, with appropriate weights and thresholds, can represent (hence, also learn in principle) every binary classification training dataset of Boolean features, provided that it has enough neurons in the first layer and there are no inconsistent training instances, but may not perform well on unseen (test) instances.

X	*Y*	*Z*	*C*
F	F	F	T
F	F	T	T
T	T	F	F
T	T	T	F

Answer: The training instances can be represented by a (possibly incomplete) truth table like the one on the right, where we assume that we now have only 4 training instances coming from the truth table of the example of the previous question (here we have the first two and the last two rows of the table of the previous question). Again, we construct an MLP with two layers, as in the previous question. Here the first layer will have only two neurons (acting as AND gates), since we only have two positive (C = T) training instances. The MLP will respond correctly when the values of X, Y, Z correspond to one of the 4 rows of the table on the right (training instances). In effect, the MLP has memorized the two positive training instances for which it responds C = T, and responds C = F in all other cases. However, the MLP will behave incorrectly in all the other lines of the truth table of the previous question, which might be included in the test data. Concretely, it will wrongly respond C = F for the combinations of X, Y, Z values that correspond to the third row from top and third row from bottom in the table of the previous question.

4. (i) We wish to train a (single) Perceptron to separate the instances of the two classes (black and white dots, inside and outside of a circle) of the figure on the right. There are only two (real-valued) features, corresponding to the two axes. Explain why the Perceptron cannot learn to correctly separate the two classes using the current two features.

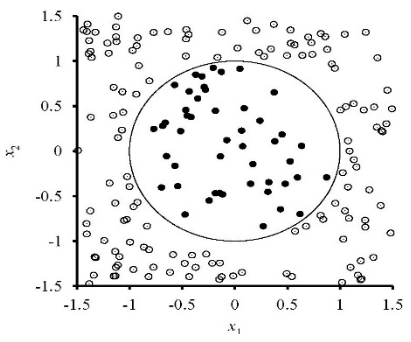


Image from the book of Russel & Norvig; see references in the slides.

Answer: The Perceptron is a linear classifier, i.e., it learns a point (for one feature), a straight line (for two features), a plane (for three features), or more generally a hyper-plane (for more features), and classifies unseen instances by examining if they fall above or below the hyper-plane. The dataset of the figure is not linearly separable with the current two features, i.e., there is no straight line that separates the black from the white dots. Hence, a single Perceptron cannot learn to separate the two classes with the current features.

(ii) Propose a mapping from the feature vector of each instance to a single real number (a single real-valued feature), so that the new (single) feature will allow the Perceptron to correctly separate the two classes.

Answer: We can represent each instance by its distance from the center of the circle. Then all the instances will be along the axis of the new, single feature (distance from the center), the black dots will be on the left of the value that corresponds to the radius (approximately 1), and the white dots will be on the right of the radius value. With the new (single feature) *representation, the classes are linearly separable, hence the Perceptron can learn to correctly separate them.*

5. (i) Two students are discussing how the Perceptron (single neuron) relates to a logistic regression classifier. The first student claims that a (binary) logistic regression classifier is the same as a (single neuron) Perceptron with a sigmoid activation function. To support her view, she wrote down the formulae that compute the output of the Perceptron and the probability that the logistic regression classifier assigns to the positive class, in both cases given an input vector $\vec{x}$. Write down the formulae. What do they show?

Answer: The output of the Perceptron with a sigmoid activation function is:

$$\Phi(\sum w_l x_l) = \Phi(\vec{w} \cdot \vec{x}) = 1/(1+e^{-\vec{w} \cdot \vec{x}})$$

The probability that the logistic regression classifier assigns to the positive class is:

$$P(c_+ \mid \vec{x}) = 1/(1+e^{-\vec{w} \cdot \vec{x}})$$

The formulae show that if we use the same weights $\vec{w}$, the output of the Perceptron will be the same as the probability of the positive class of logistic regression, which seems to agree with the claim of the first student.

(ii) The second student, however, responded that the Perceptron and logistic regression learn different weights, even if they use the same training dataset, the same initial weights, and the same optimizer. To support her claim, she wrote down the weight update rules of the Perceptron (with sigmoid activation function, slide 14) and logistic regression (with stochastic gradient ascent). Write the update rules. What do they show?

Answer: The weight update rule of the Perceptron (with sigmoid activation function) is:

$$w_l \leftarrow w_l + \eta \cdot \Phi(\vec{w} \cdot \vec{x}^{(i)}) \cdot (1 - \Phi(\vec{w} \cdot \vec{x}^{(i)})) \cdot (t^{(i)} - \Phi(\vec{w} \cdot \vec{x}^{(i)})) \cdot x_l^{(i)}$$

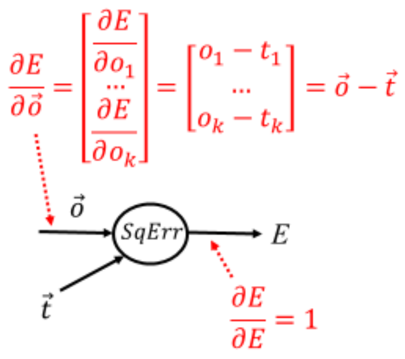
The weight update rule of logistic regression (with stochastic gradient ascent, without regularization) is:

$$w_l \leftarrow w_l + \eta \cdot [t^{(i)} - P(c_+ \mid \vec{x}^{(i)})] \cdot x_l^{(i)}$$

The update rules are indeed different. This is because the Perceptron that we considered in the slides tries to minimize the squared error loss, whereas logistic regression tries to minimize the cross-entropy (or to maximize the conditional log-likelihood) of the training data. Hence, the second student is right, that in general the Perceptron will learn different weights than logistic regression. However, if we used the cross-entropy loss in the Perceptron too (and the same regularization and optimizer), we would come up with the same update rules, which would agree with the first student's opinion.

6. Show that without activation functions, a multi-layer neural network is equivalent to applying a linear transformation to the input, i.e., the output can be written as $\vec{o} = W\vec{x} + b$, where W is a weights matrix, $b \in \mathbb{R}$ is a bias term, and $\vec{x} \epsilon \mathbb{R}^n$ is the input feature vector.

7. Confirm the computation of $\frac{\partial E}{\partial \vec{o}}$ in the computation graph of slide 29.



$\frac{\partial E}{\partial \vec{o}} = \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \cdots \\ \frac{\partial E}{\partial o_k} \end{bmatrix} = \begin{bmatrix} o_1 - t_1 \\ \cdots \\ o_k - t_k \end{bmatrix} = \vec{o} - \vec{t}$
$\vec{o}$
$SqErr$
E
$\vec{t}$
$\frac{\partial E}{\partial E} = 1$

Answer: The gradient that we need to compute is:

$$\frac{\partial E}{\partial \vec{o}} = \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \cdots \\ \frac{\partial E}{\partial o_i} \\ \cdots \\ \frac{\partial E}{\partial o_k} \end{bmatrix}$$

Let us consider separately a single derivative $\frac{\partial E}{\partial o_i}$ *(a single element of the gradient):*

$$\frac{\partial E}{\partial o_i} = \frac{\partial}{\partial o_i} \sum_{j=1}^{k} \frac{1}{2}\left(t_j - o_j\right)^2 = \frac{\partial}{\partial o_i}\frac{1}{2}(t_i - o_i)^2 = \frac{1}{2} \cdot 2 \cdot (t_i - o_i) \cdot \frac{\partial}{\partial o_i}(t_i - o_i)$$
$$= (t_i - o_i) \cdot (-1) = (o_i - t_i)$$

Hence:

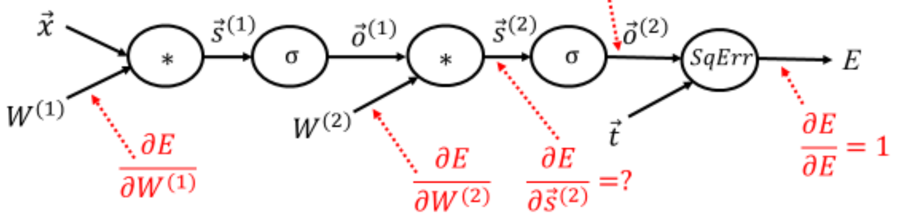
$$\frac{\partial E}{\partial \vec{o}} = \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \cdots \\ \frac{\partial E}{\partial o_i} \\ \cdots \\ \frac{\partial E}{\partial o_k} \end{bmatrix} = \begin{bmatrix} o_1 - t_1 \\ \cdots \\ o_i - t_i \\ \cdots \\ o_k - t_k \end{bmatrix} = \vec{o} - \vec{t}$$

Note: We do not need to compute $\frac{\partial E}{\partial \vec{t}}$*, because we do not update* $\vec{t}$ *(the correct prediction).*

8. (i) Compute the gradient $\frac{\partial E}{\partial \vec{o}^{(2)}}$ in the network with the following computation graph.

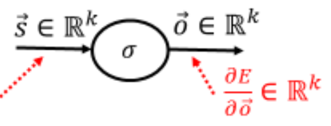
Answer: The gradient $\frac{\partial E}{\partial \vec{o}^{(2)}}$ *is computed as in Exercise 7.*

$$\frac{\partial E}{\partial \vec{o}^{(2)}} = \begin{bmatrix} \frac{\partial E}{\partial o_1^{(2)}} \\ \cdots \\ \frac{\partial E}{\partial o_{k_2}^{(2)}} \end{bmatrix} = \begin{bmatrix} o_1^{(2)} - t_1 \\ \cdots \\ o_{k_2}^{(2)} - t_{k_2} \end{bmatrix} = \vec{o}^{(2)} - \vec{t}$$



$\vec{x}$
$W^{(1)}$
$*$
$\vec{s}^{(1)}$
σ
$\vec{o}^{(1)}$
$*$
$W^{(2)}$
$\vec{s}^{(2)}$
σ
$\vec{o}^{(2)}$
SqErr
$\vec{t}$
E
$\frac{\partial E}{\partial W^{(1)}}$
$\frac{\partial E}{\partial W^{(2)}}$
$\frac{\partial E}{\partial \vec{s}^{(2)}} = ?$
$\frac{\partial E}{\partial E} = 1$

(ii) Show that for a sigmoid node $\sigma(\vec{s}) = \vec{o}$, $\frac{\partial E}{\partial \vec{s}}$ can be computed as follows, where J is the Jacobian matrix.[1]



$\vec{s} \in \mathbb{R}^k$
σ
$\vec{o} \in \mathbb{R}^k$
$\frac{\partial E}{\partial \vec{o}} \in \mathbb{R}^k$

$$\frac{\partial E}{\partial \vec{s}} = \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \vdots \\ \frac{\partial E}{\partial s_i} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial \sigma(s_1)}{\partial s_1} & \frac{\partial \sigma(s_2)}{\partial s_1} & \cdots & \frac{\partial \sigma(s_k)}{\partial s_1} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \sigma(s_1)}{\partial s_i} & \frac{\partial \sigma(s_2)}{\partial s_i} & \cdots & \frac{\partial \sigma(s_k)}{\partial s_i} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \sigma(s_1)}{\partial s_k} & \frac{\partial \sigma(s_2)}{\partial s_k} & \cdots & \frac{\partial \sigma(s_k)}{\partial s_k} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \vdots \\ \frac{\partial E}{\partial o_i} \\ \vdots \\ \frac{\partial E}{\partial o_k} \end{bmatrix} = J^T \frac{\partial E}{\partial \vec{o}}$$

$$= \begin{bmatrix} \sigma(s_1)(1-\sigma(s_1)) & 0 & \cdots & 0 \\ 0 & \sigma(s_2)(1-\sigma(s_2)) & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \sigma(s_k)(1-\sigma(s_k)) \end{bmatrix} \frac{\partial E}{\partial \vec{o}}$$

Answer: The gradient that we need to compute is:

$$\frac{\partial E}{\partial \vec{s}} = \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \vdots \\ \frac{\partial E}{\partial s_i} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix}$$

[1] See https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant.

Let us consider separately a single derivative $\frac{\partial E}{\partial s_i}$ *(a single element of the gradient). By the chain rule of derivatives, we obtain:*

$$\frac{\partial E}{\partial s_i} = \sum_{j=1}^{k} \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial s_i}$$

However, each s_i *affects only* $o_i = \sigma(s_i)$. *It does not affect any other* $o_j = \sigma(s_j)$, *for* $j \neq i$. *Hence,* $\frac{\partial o_j}{\partial s_i} = 0$ *for* $j \neq i$, *and we obtain:*

$$\frac{\partial E}{\partial s_i} = \frac{\partial E}{\partial o_i} \frac{\partial o_i}{\partial s_i} = \frac{\partial E}{\partial o_i} \frac{\partial \sigma(s_i)}{\partial s_i} = \frac{\partial E}{\partial o_i} \sigma(s_i)\left(1 - \sigma(s_i)\right)$$

where we have use the property of the sigmoid that $\frac{d\sigma(x)}{dx} = \sigma(x)\left(1 - \sigma(x)\right)$.

Therefore:

$$\frac{\partial E}{\partial \vec{s}} = \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \vdots \\ \frac{\partial E}{\partial s_i} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial o_1} \frac{\partial \sigma(s_1)}{\partial s_1} \\ \vdots \\ \frac{\partial E}{\partial o_i} \frac{\partial \sigma(s_i)}{\partial s_i} \\ \vdots \\ \frac{\partial E}{\partial o_k} \frac{\partial \sigma(s_k)}{\partial s_k} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial o_1} \sigma(s_1)\left(1 - \sigma(s_1)\right) \\ \vdots \\ \frac{\partial E}{\partial o_i} \sigma(s_i)\left(1 - \sigma(s_i)\right) \\ \vdots \\ \frac{\partial E}{\partial o_k} \sigma(s_k)\left(1 - \sigma(s_k)\right) \end{bmatrix}$$

The latter can also be written as:

$$\frac{\partial E}{\partial \vec{s}} = \begin{bmatrix} \frac{\partial \sigma(s_1)}{\partial s_1} & 0 & \dots & 0 \\ 0 & \frac{\partial \sigma(s_2)}{\partial s_2} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \frac{\partial \sigma(s_k)}{\partial s_k} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \frac{\partial E}{\partial o_2} \\ \vdots \\ \frac{\partial E}{\partial o_k} \end{bmatrix} =$$

$$= \begin{bmatrix} \sigma(s_1)\left(1 - \sigma(s_1)\right) & 0 & \dots & 0 \\ 0 & \sigma(s_2)\left(1 - \sigma(s_2)\right) & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \sigma(s_k)\left(1 - \sigma(s_k)\right) \end{bmatrix} \frac{\partial E}{\partial \vec{o}}$$

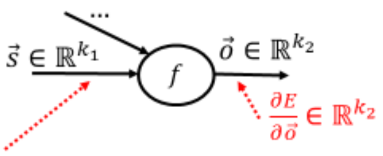
More generally, it can be written as:

$$\frac{\partial E}{\partial \vec{s}} = \begin{bmatrix} \frac{\partial \sigma(s_1)}{\partial s_1} & \frac{\partial \sigma(s_2)}{\partial s_1} & \dots & \frac{\partial \sigma(s_k)}{\partial s_1} \\ \frac{\partial \sigma(s_1)}{\partial s_2} & \frac{\partial \sigma(s_2)}{\partial s_2} & \dots & \frac{\partial \sigma(s_k)}{\partial s_2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial \sigma(s_1)}{\partial s_k} & \frac{\partial \sigma(s_2)}{\partial s_k} & \dots & \frac{\partial \sigma(s_k)}{\partial s_k} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \frac{\partial E}{\partial o_2} \\ \vdots \\ \frac{\partial E}{\partial o_k} \end{bmatrix} = J^T \frac{\partial E}{\partial \vec{o}}$$

where J is the Jacobian matrix:

$$J = \begin{bmatrix} \frac{\partial\sigma(s_1)}{\partial s_1} & \frac{\partial\sigma(s_1)}{\partial s_2} & \cdots & \frac{\partial\sigma(s_1)}{\partial s_k} \\ \frac{\partial\sigma(s_2)}{\partial s_1} & \frac{\partial\sigma(s_2)}{\partial s_2} & \cdots & \frac{\partial\sigma(s_2)}{\partial s_k} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial\sigma(s_k)}{\partial s_1} & \frac{\partial\sigma(s_k)}{\partial s_2} & \cdots & \frac{\partial\sigma(s_k)}{\partial s_k} \end{bmatrix}$$

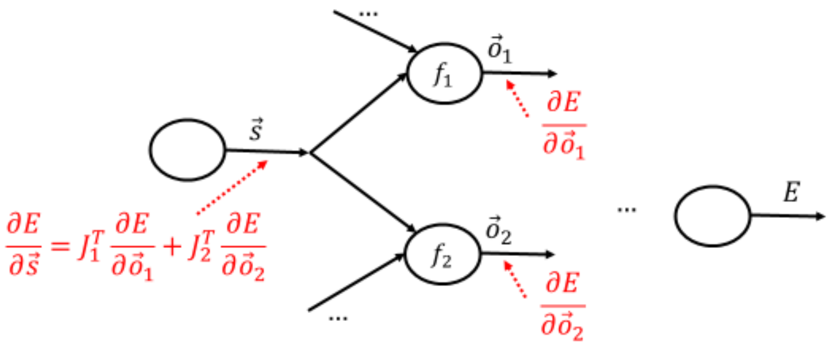
The latter applies more generally. For a node that computes $f(\vec{s},\ldots) = \vec{o}$, we can compute $\frac{\partial E}{\partial \vec{s}}$ as follows (provided that $\vec{s}$ is fed only to the f node):



…
$\vec{s} \in \mathbb{R}^{k_1}$
f
$\vec{o} \in \mathbb{R}^{k_2}$
$\frac{\partial E}{\partial \vec{o}} \in \mathbb{R}^{k_2}$

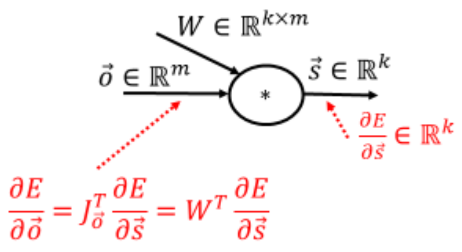
$$\frac{\partial E}{\partial \vec{s}} = \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \vdots \\ \frac{\partial E}{\partial s_i} \\ \vdots \\ \frac{\partial E}{\partial s_{k_1}} \end{bmatrix} = \begin{bmatrix} \frac{\partial o_1}{\partial s_1} & \frac{\partial o_2}{\partial s_1} & \cdots & \frac{\partial o_{k_2}}{\partial s_1} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial o_1}{\partial s_i} & \frac{\partial o_2}{\partial s_i} & \cdots & \frac{\partial o_{k_2}}{\partial s_i} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial o_1}{\partial s_{k_1}} & \frac{\partial o_2}{\partial s_{k_1}} & \cdots & \frac{\partial o_{k_2}}{\partial s_{k_1}} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \vdots \\ \frac{\partial E}{\partial o_i} \\ \vdots \\ \frac{\partial E}{\partial o_{k_2}} \end{bmatrix} = J^T \frac{\partial E}{\partial \vec{o}}$$

(Check that this is also true for $\frac{\partial E}{\partial \vec{o}}$ in exercise 7.)
If $\vec{s}$ is fed to two (or more) nodes f_1, f_2, we have to add the gradients for $\frac{\partial E}{\partial \vec{s}}$ that we get from f_1, f_2:



…
f_1
$\vec{o}_1$
$\frac{\partial E}{\partial \vec{o}_1}$
$\vec{s}$
$\frac{\partial E}{\partial \vec{s}} = J_1^T \frac{\partial E}{\partial \vec{o}_1} + J_2^T \frac{\partial E}{\partial \vec{o}_2}$
…
E
f_2
$\vec{o}_2$
$\frac{\partial E}{\partial \vec{o}_2}$
…

(iii) Show that for a matrix-vector multiplication node $W\vec{o} = \vec{s}$, $\frac{\partial E}{\partial \vec{o}}$ can be computed as follows:



$W \in \mathbb{R}^{k\times m}$
$\vec{o} \in \mathbb{R}^m$
$*$
$\vec{s} \in \mathbb{R}^k$
$\frac{\partial E}{\partial \vec{s}} \in \mathbb{R}^k$
$\frac{\partial E}{\partial \vec{o}} = J_{\vec{o}}^T \frac{\partial E}{\partial \vec{s}} = W^T \frac{\partial E}{\partial \vec{s}}$

Answer:

$$\vec{s} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \cdots \\ s_k \end{bmatrix} = W\vec{o} = \begin{bmatrix} w_{1,1} & w_{2,1} & \cdots & w_{m,1} \\ w_{1,2} & w_{2,2} & \cdots & w_{m,2} \\ w_{1,3} & w_{2,3} & \cdots & w_{m,3} \\ \cdots & \cdots & \cdots & \cdots \\ w_{1,k} & w_{2,k} & \cdots & w_{m,k} \end{bmatrix} \begin{bmatrix} o_1 \\ o_2 \\ o_3 \\ \cdots \\ o_m \end{bmatrix} = \begin{bmatrix} w_{1,1}o_1 + w_{2,1}o_2 + \cdots + w_{m,1}o_m \\ w_{1,2}o_1 + w_{2,2}o_2 + \cdots + w_{m,2}o_m \\ w_{1,3}o_1 + w_{2,3}o_2 + \cdots + w_{m,3}o_m \\ \cdots \\ w_{1,k}o_1 + w_{2,k}o_2 + \cdots + w_{m,k}o_m \end{bmatrix}$$

The gradient that we need to compute is:

$$\frac{\partial E}{\partial \vec{o}} = \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \vdots \\ \frac{\partial E}{\partial o_i} \\ \vdots \\ \frac{\partial E}{\partial o_m} \end{bmatrix}$$

Let us consider separately a single derivative $\frac{\partial E}{\partial o_i}$ *(a single element of the gradient). By the chain rule of derivatives, we obtain:*

$$\frac{\partial E}{\partial o_i} = \sum_{j=1}^{k} \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial o_i}$$

According to the equations for $\vec{s} = W\vec{o}$ *above:*

$$s_j = w_{1,j}o_1 + w_{2,j}o_2 + \cdots + w_{i,j}o_i + \cdots + w_{m,j}o_m$$

Hence:

$$\frac{\partial s_j}{\partial o_i} = w_{i,j}$$

Therefore:

$$\frac{\partial E}{\partial o_i} = \sum_{j=1}^{k} \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial o_i} = \sum_{j=1}^{k} \frac{\partial E}{\partial s_j} w_{i,j}$$

which can also be written as:

$$\frac{\partial E}{\partial o_i} = \begin{bmatrix} \frac{\partial s_1}{\partial o_i} & \frac{\partial s_2}{\partial o_i} & \cdots & \frac{\partial s_k}{\partial o_i} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \frac{\partial E}{\partial s_2} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix} = \begin{bmatrix} w_{i,1} & w_{i,2} & \cdots & w_{i,k} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \frac{\partial E}{\partial s_2} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix}$$

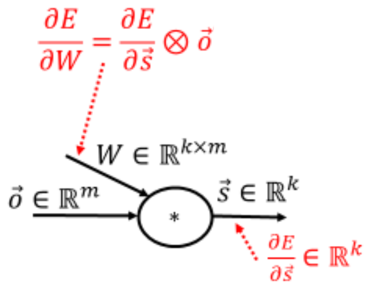
Hence, for the overall gradient:

$$\frac{\partial E}{\partial \vec{o}} = \begin{bmatrix} \frac{\partial E}{\partial o_1} \\ \frac{\partial E}{\partial o_2} \\ \vdots \\ \frac{\partial E}{\partial o_i} \\ \vdots \\ \frac{\partial E}{\partial o_m} \end{bmatrix} = \begin{bmatrix} \frac{\partial s_1}{\partial o_1} & \frac{\partial s_2}{\partial o_1} & \cdots & \frac{\partial s_k}{\partial o_1} \\ \frac{\partial s_1}{\partial o_2} & \frac{\partial s_2}{\partial o_2} & \cdots & \frac{\partial s_k}{\partial o_2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial s_1}{\partial o_i} & \frac{\partial s_2}{\partial o_i} & \cdots & \frac{\partial s_k}{\partial o_i} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial s_1}{\partial o_m} & \frac{\partial s_2}{\partial o_m} & \cdots & \frac{\partial s_k}{\partial o_m} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \frac{\partial E}{\partial s_2} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,k} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ w_{i,1} & w_{i,2} & \cdots & w_{i,k} \\ \vdots & \vdots & \vdots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,k} \end{bmatrix} \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \frac{\partial E}{\partial s_2} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix} =$$

$$J_{\vec{o}}^T \frac{\partial E}{\partial \vec{s}} = W^T \frac{\partial E}{\partial \vec{s}}$$

Note: We prefer to use matrix operators, which can be efficiently computed using highly optimized algorithms and GPUs, rather than relying on our own for-loops (e.g., in our own Python scripts) to compute individual elements of matrices, which is much slower.

(iv) Show that for a matrix-vector multiplication node $W\vec{o} = \vec{s}$, $\frac{\partial E}{\partial W} = \frac{\partial E}{\partial \vec{s}} \otimes \vec{o}$, where $\otimes$ denotes the outer product.[2]


$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial \vec{s}} \otimes \vec{o}$
$W \in \mathbb{R}^{k\times m}$
$\vec{o} \in \mathbb{R}^m$
$*$
$\vec{s} \in \mathbb{R}^k$
$\frac{\partial E}{\partial \vec{s}} \in \mathbb{R}^k$

Answer: Recall that we use the following notation for the elements of W:

$$W = \begin{bmatrix} w_{1,1} & w_{2,1} & \cdots & w_{m,1} \\ w_{1,2} & w_{2,2} & \cdots & w_{m,2} \\ w_{1,3} & w_{2,3} & \cdots & w_{m,3} \\ \cdots & \cdots & \cdots & \cdots \\ w_{1,k} & w_{2,k} & \cdots & w_{m,k} \end{bmatrix}$$

The gradient that we need to compute is:

[2] See https://en.wikipedia.org/wiki/Matrix_multiplication#Outer_product.

$$\frac{\partial E}{\partial W} = \begin{bmatrix} \frac{\partial E}{\partial w_{1,1}} & \frac{\partial E}{\partial w_{2,1}} & \cdots & \frac{\partial E}{\partial w_{m,1}} \\ \frac{\partial E}{\partial w_{1,2}} & \frac{\partial E}{\partial w_{2,2}} & \cdots & \frac{\partial E}{\partial w_{m,2}} \\ \frac{\partial E}{\partial w_{1,3}} & \frac{\partial E}{\partial w_{2,3}} & \cdots & \frac{\partial E}{\partial w_{m,3}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial E}{\partial w_{1,k}} & \frac{\partial E}{\partial w_{2,k}} & \cdots & \frac{\partial E}{\partial w_{m,k}} \end{bmatrix}$$

Let us consider separately a single derivative $\frac{\partial E}{\partial w_{i,j}}$ *(a single element of the gradient). By the chain rule of derivatives, we obtain:*

$$\frac{\partial E}{\partial w_{i,j}} = \sum_{l=1}^{k} \frac{\partial E}{\partial s_l} \frac{\partial s_l}{\partial w_{i,j}}$$

According to the equations for $\vec{s} = W\vec{o}$ *in part (iii) of the exercise:*

$$s_l = w_{1,l}o_1 + w_{2,l}o_2 + \cdots + w_{i,l}o_i + \cdots + w_{m,l}o_m$$

Hence:

$$\frac{\partial s_l}{\partial w_{i,j}} = 0, \text{for } l \neq j$$

and:

$$\frac{\partial E}{\partial w_{i,j}} = \sum_{l=1}^{k} \frac{\partial E}{\partial s_l} \frac{\partial s_l}{\partial w_{i,j}} = \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial w_{i,j}}$$

Given that:

$$s_j = w_{1,j}o_1 + w_{2,j}o_2 + \cdots + w_{i,j}o_i + \cdots + w_{m,j}o_m$$

we obtain:

$$\frac{\partial s_j}{\partial w_{i,j}} = o_i$$

Hence:

$$\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial s_j} \frac{\partial s_j}{\partial w_{i,j}} = \frac{\partial E}{\partial s_j} o_i$$

Going back to the overall gradient:

$$\frac{\partial E}{\partial W} = \begin{bmatrix} \frac{\partial E}{\partial w_{1,1}} & \frac{\partial E}{\partial w_{2,1}} & \cdots & \frac{\partial E}{\partial w_{m,1}} \\ \frac{\partial E}{\partial w_{1,2}} & \frac{\partial E}{\partial w_{2,2}} & \cdots & \frac{\partial E}{\partial w_{m,2}} \\ \frac{\partial E}{\partial w_{1,3}} & \frac{\partial E}{\partial w_{2,1}} & \cdots & \frac{\partial E}{\partial w_{m,3}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial E}{\partial w_{1,k}} & \frac{\partial E}{\partial w_{2,k}} & \cdots & \frac{\partial E}{\partial w_{m,k}} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial s_1} o_1 & \frac{\partial E}{\partial s_1} o_2 & \cdots & \frac{\partial E}{\partial s_1} o_m \\ \frac{\partial E}{\partial s_2} o_1 & \frac{\partial E}{\partial s_2} o_2 & \cdots & \frac{\partial E}{\partial s_2} o_m \\ \frac{\partial E}{\partial s_3} o_1 & \frac{\partial E}{\partial s_3} o_2 & \cdots & \frac{\partial E}{\partial s_3} o_m \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial E}{\partial s_k} o_1 & \frac{\partial E}{\partial s_k} o_2 & \cdots & \frac{\partial E}{\partial w_{s_k}} o_m \end{bmatrix} =$$

$$= \begin{bmatrix} \frac{\partial E}{\partial s_1} \\ \frac{\partial E}{\partial s_2} \\ \frac{\partial E}{\partial s_3} \\ \vdots \\ \frac{\partial E}{\partial s_k} \end{bmatrix} \begin{bmatrix} o_1 & o_2 & \cdots & o_m \end{bmatrix} = \frac{\partial E}{\partial \vec{s}} \otimes \vec{o}$$

9. Repeat exercise 11 of Part 2 (text classification with mostly linear classifiers), now using an MLP classifier implemented (by you) in Keras/TensorFlow or PyTorch.[3] You may use different features in the MLP classifier than the ones you used in exercise 11 of Part 2. Tune the hyper-parameters (e.g., number of hidden layers, dropout probability) on the development subset of your dataset. Monitor the performance of the MLP on the development subset during training to decide how many epochs to use. Include experimental results of a baseline majority classifier, as well as experimental results of your best classifier from exercise 11 of Part 2, now treated as a second baseline. Include in your report:

- Curves showing the loss on training and development data as a function of epochs (slide 49).
- Precision, recall, F1, precision-recall AUC scores, for each class and classifier, separately for the training, development, and test subsets, as in exercise 11 of Part 2.
- Macro-averaged precision, recall, F1, precision-recall AUC scores (averaging the corresponding scores of the previous bullet over the classes), for each classifier, separately for the training, development, and test subsets, as in exercise 11 of Part 2.
- A short description of the methods and datasets you used, including statistics about the datasets (e.g., average document length, number of training/dev/test documents, vocabulary size) and a description of the preprocessing steps that you performed.

You may optionally wish to try ensembles. One possibility is to use *k* separate MLP classifiers, corresponding to your *k* best checkpoints (*k* best epochs in terms of development loss), and aggregate their decisions by majority voting. Another possibility is to use temporal averaging, i.e., use a single MLP classifier, whose weights are the average of the weights of the *k* best checkpoints.

10. Develop a part-of-speech (POS) tagger for one of the languages of the Universal Dependencies treebanks (http://universaldependencies.org/), using an MLP (implemented by you) operating on windows of words (slides 35–36). Consider only the words, sentences, and POS tags of the treebanks (not the dependencies or other annotations). Use Keras/TensorFlow

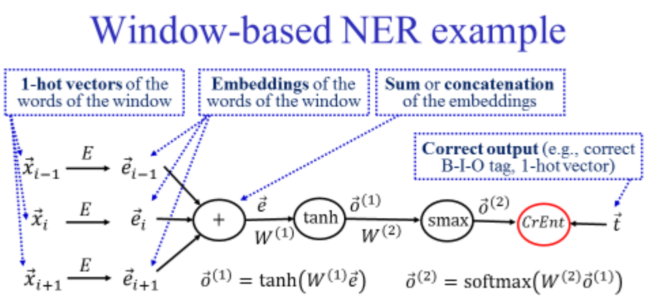
[3] See http://keras.io/, https://www.tensorflow.org/, http://pytorch.org/.

or PyTorch to implement the MLP. You may use any types of word features you prefer, but it is recommended to use pre-trained word embeddings. Make sure that you use separate training, development, and test subsets. Tune the hyper-parameters (e.g., number of hidden layers, dropout probability) on the development subset. Monitor the performance of the MLP on the development subset during training to decide how many epochs to use. Include experimental results of a baseline that tags each word with the most frequent tag it had in the training data; for words that were not encountered in the training data, the baseline should return the most frequent tag (over all words) of the training data. Include in your report:

- Curves showing the loss on training and development data as a function of epochs (slide 49).
- Precision, recall, F1, precision-recall AUC scores, for each class (tag) and classifier, separately for the training, development, and test subsets, as in exercise 11 of Part 2.
- Macro-averaged precision, recall, F1, precision-recall AUC scores (averaging the corresponding scores of the previous bullet over the classes), for each classifier, separately for the training, development, and test subsets, as in exercise 11 of Part 2.
- A short description of the methods and datasets you used, including statistics about the datasets (e.g., average sentence length, number of training/dev/test sentences and words, vocabulary size) and a description of the preprocessing steps.

You may optionally wish to try ensembles, as in exercise 9 above.

11. (a) We use the window-based neural network named entity recognizer (NER) of the slide on the right, with 300-dimensional word embeddings, to recognize three types of named entities (persons, organizations, locations). We use B-I-O tags (BPerson, IPerson, BOrganization etc., with a single O tag). The size of the vocabulary is $|V| = 100{,}000$. The "+" node concatenates the embeddings of the three words in the window. The hidden layer contains 500 neurons (with *tanh* activation functions). What are the dimensions of matrices $E, W^{(1)}, W^{(2)}$? Fully justify your answers.


Window-based NER example
1-hot vectors of the words of the window
Embeddings of the words of the window
Sum or concatenation of the embeddings
Correct output (e.g., correct B-I-O tag, 1-hot vector)
$\vec{x}_{i-1}$
$\vec{x}_i$
$\vec{x}_{i+1}$
E
$\vec{e}_{i-1}$
$\vec{e}_i$
$\vec{e}_{i+1}$
+
$\vec{e}$
$W^{(1)}$
tanh
$\vec{o}^{(1)}$
$W^{(2)}$
smax
$\vec{o}^{(2)}$
CrEnt
$\vec{t}$
$\vec{o}^{(1)} = \tanh(W^{(1)}\vec{e})$
$\vec{o}^{(2)} = \text{softmax}(W^{(2)}\vec{o}^{(1)})$

12 [**optional**] Repeat exercise 3 of Part 1 (*n*-gram language models and context-aware spelling correction) now using an MLP language model, instead of an *n*-gram language model. The MLP takes as input the concatenation of the word embeddings of the *n* previous words, and outputs a probability distribution over the vocabulary as a prediction for the next word. Compare the WER and CER scores you obtain using the MLP language model to those you had obtained with the bigram and trigram language models of Part 1.