Deep Learning – Exercise 1

Alex Goldovsky | 317074557
Guy Orlinsky | 318421716

Practical

2.

Opening notes:
Except for the results reported below, while investigating the training data, we also noticed that there were quiet a few examples of peptides that appeared as both negative and positive.
Also we actually haven't had to do anything special to avoid non-trivial solutions(maybe except for shuffling the data), we first tried oversampling, and also weighted scoring but at the end we noticed that the network doesn't converge to trivial solution even without these methods, and dropped the attempt to tackle this problem.
In addition, all the parameters were finetuned on the **validation set** and the results shown here are on the **test set**. We mention it since it isn't shown in the code attached.
Finally, all the result are averaged over 30 runs with different random seeds to avoid bias towards a specific seed.

2.a

We chose to represent each 9-mer i.e. peptide as a 200 features vector.
20 features are frequencies of each of the amino acid the peptide contain,
The other 180 features are one-hot features of the peptide's amino acids identity together with their position, for every amino acid and possible position (0-8 index) we have created a feature.
Frequencies:

	9-mer Peptide	freq_A	freq_C	freq_D	freq_E	freq_F	freq_G	freq_H	...	freq_P	freq_Q	freq_R	freq_S	freq_T	freq_V	freq_W_Y
0	MFVFLVLLP	0.0	0.0	0.0	0.0	0.222222	0.0	0.0	...	0.111111	0.0	0.0	0.000000	0.0	0.222222	0.0
1	FVFLVLLPL	0.0	0.0	0.0	0.0	0.222222	0.0	0.0	...	0.111111	0.0	0.0	0.000000	0.0	0.222222	0.0
2	VFLVLLPLV	0.0	0.0	0.0	0.0	0.111111	0.0	0.0	...	0.111111	0.0	0.0	0.000000	0.0	0.333333	0.0
3	FLVLLPLVS	0.0	0.0	0.0	0.0	0.111111	0.0	0.0	...	0.111111	0.0	0.0	0.111111	0.0	0.222222	0.0

One-hot features of Amino acid identity and position:

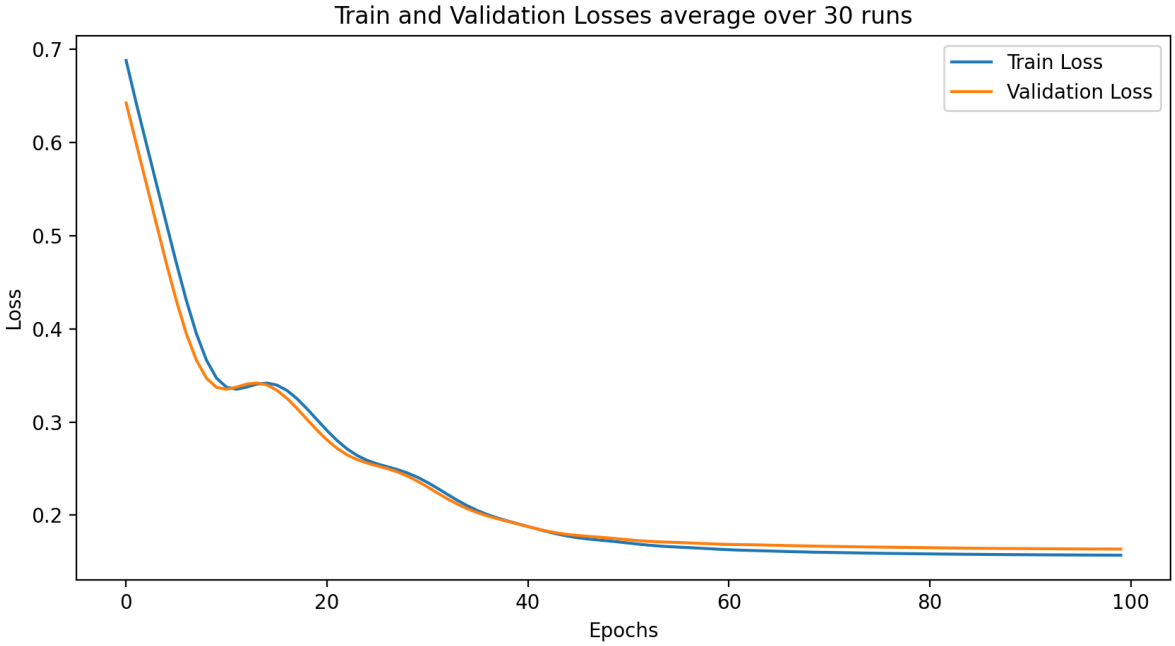
	col_0_A	col_0_C	col_0_D	col_0_E	col_0_F	col_0_G	col_0_H	col_0_I	...	col_8_P	col_8_Q	col_8_R	col_8_S	col_8_T	col_8_V	col_8_W_Y
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0

At first, we tried a quick and dirty approach, the simplest features possible, which are one-hot vectors of the amino acids, which didn't give good results which were measured using the accuracy, precision, and recall.
After reading online we came to understanding that the position of the amino-acids is a strong indicator, as well as the frequency in the peptide.

We also tried using amino-acids embeddings using the *bio-embeddings* python package, at last we have decided to stick to the representation mentioned above, due to its simplicity and superior results.

2.b

The network's input data under the chosen representation is 200.
20 for each amino acid frequency, and 20*9 for each amino acid and possible position combination.
Using the suggested network architecture, we noticed that the training is longer, which isn't a surprise, and we got a hint of slight overfitting issue.
we expected to see a bigger overfitting problem, which at first we did see, when we tried to use oversampling to prevent the network from converging to trivial solutions, as was suggested in the exercise definition.
However then we tried to fit a model without oversampling. Suprisingly we found out that it seems that the dimension does not actually cause very big overfitting, as seen in the following plot, compared to other architectures we have tried (explained later).



Using a threshold of 0.45 for prediction we got the following result on the test set (average of 30 runs):

```
Accuracy: 0.9231841881896448
Precision: 0.6560253793959727
Recall: 0.6201783723522852
F1 Score: 0.6371311073300716
```

3.c

To tackle the high computational complexity, and the small overfitting problem, we came up with the following network architecture:

Input layer, tanh activation layer, 7-dimensional hidden layer, and a single output layer.
Using this architecture, we got a good generalization and on many different random seeds we got a test loss very close to the train loss, the following plot shows the test-train loss plot:

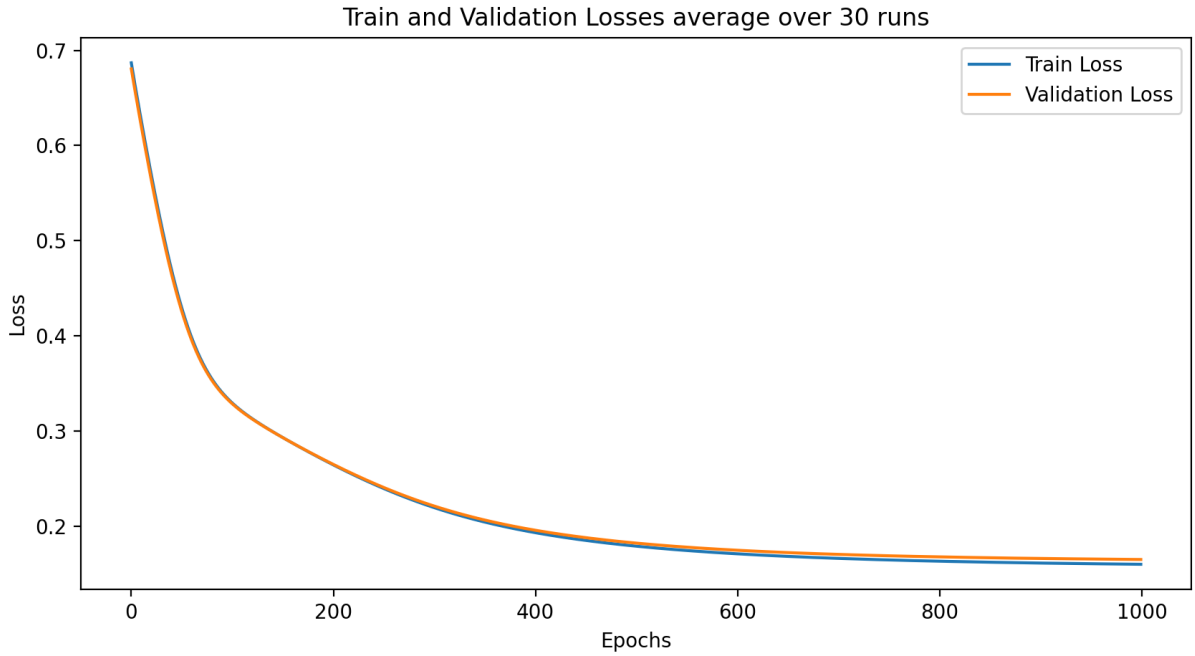
Using this architecture, with a threshold of 0.45, on the test set we have got the following results (average of 30 runs with different seeds):

```
Accuracy: 0.9256093124772644
Precision: 0.6742534697275449
Recall: 0.61371237458194
F1 Score: 0.6419876566187362
```

Which are very slightly better.
Regarding the considerations for the design of the architecture, we first thought that bringing down the dimension of the network would solve the problems seen.
Using trial and error, of course **on the validation set! (And not the test set)**, we found the parameters that seemed optimal(rounded it up to the close 5 for simplicity).
The main hyperparameters we needed to decide upon, were: hidden layers, number of epochs, learning rate, and the threshold for prediction.
The number of epochs was fairly simple to find, we used the train test-loss to look for convergence, same with the learning rate.
For the threshold we looked for one that would maximize the F1 score, as we didn't get instructions for avoiding false positives or false negatives.
As for the hidden layers, we considered tanh and Relu, and got better results with tanh.

2.d

Removing the non-linear operators from the network seemed to have a little effect, if any, the results were a bit worse. See the plot below for the train-test loss



Using this architecture with other hyper-parameters same as the one with no linear layers we got the following metrics:

```
Accuracy: 0.9236813386686068
Precision: 0.6685857802880435
Recall: 0.5924191750278707
F1 Score: 0.6278832215827053
```

As mentioned, the results are a little bit worse, which is seen by the lower metrics values.

2.e

Using different random seeds, we got different result which all seemed to contain the peptide: **NLREFVFKN**. Almost all of the time we got the peptide **LLIVNNATN**, and lastly, the third most frequent was **FDNPVLPFN**. There were another very frequent one or two peptides, but we were only asked about 3.

Theoretical

1.

Linear functions:

Recall the definition of a linear function:

A function $f: \mathbb{R}^n \to \mathbb{R}^m$ is linear if there exists a matrix $A \in \mathbb{R}^{m \times n}$ such that for all $x \in \mathbb{R}^n$, $f(x) = Ax$.

Show that the composition of linear functions is a linear function.

Let $x_n \in \mathbb{R}^n, x_m \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{l \times m}$,

and $f: \mathbb{R}^n \to \mathbb{R}^m, g: \mathbb{R}^m \to \mathbb{R}^l$ such that:

and $f(x) = Ax_n, g(x) = Bx_m$ linear functions.

$$g\big(f(x)\big) = g(Ax) = B(Ax) = (BA)_{[l \times n]} x_n$$

Hence the composition is of the form of linear function $h: \mathbb{R}^n \to \mathbb{R}^l$ with that matrix:

$$C \in \mathbb{R}^{l \times n}, \forall x \in \mathbb{R}^n: \ h(x) = Cx = (BA)x = g \circ f(x)$$

Affine transformations:

Recall the definition of an affine function:

A function $f: \mathbb{R}^n \to \mathbb{R}^m$ is affine if there exists a matrix $A \in \mathbb{R}^{m \times n}$ and some vector $b \in \mathbb{R}^m$, such that for all $x \in \mathbb{R}^n$, $f(x) = Ax$+b.

Show that the composition of affine functions is an affine function.

Let $x_n \in \mathbb{R}^n, x_m, a \in \mathbb{R}^m, b \in \mathbb{R}^l, A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{l \times m}$,

and $f: \mathbb{R}^n \to \mathbb{R}^m, g: \mathbb{R}^m \to \mathbb{R}^l$ such that:

$f(x) = Ax_n + a, g(x) = Bx_m + b$ affine functions.

$$g\big(f(x)\big) = g(Ax + a) = B(Ax + a) + b = B(Ax) + Ba + b = (BA)_{[l \times n]}x + \underbrace{Ba + b}_{c \in \mathbb{R}^l}$$

Hence the composition is of the form of affine transformation with:

$$C \in \mathbb{R}^{l \times n}, c \in \mathbb{R}^l, \forall x \in \mathbb{R}^n: \ h(x) = Cx + c = (BA)x + (Ba + b) = g \circ f(x)$$

2.

2.a

$$f(x, y) = 10(x - 1)^2 + (y + 1)^2/10$$

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} 20(x - 1) \\ \frac{1}{5}(y + 1) \end{pmatrix}$$

And the gradient update equation is:

$$x^{(k+1)} = x^{(k)} - \gamma\left(20\left(x^{(k)} - 1\right)\right)$$
$$y^{(k+1)} = y^{(k)} - \gamma\left(\frac{1}{5}\left(y^{(k)} + 1\right)\right)$$

2.b

Using the hint:

$$x^{(k+1)} = x^{(k)} - 20\gamma x^{(k)} + \gamma = (1 - 20\gamma)x^{(k)} + 20\gamma$$
$$y^{(k+1)} = y^{(k)} - \frac{1}{5}\gamma y^{(k)} - \gamma = \left(1 - \frac{1}{5}\gamma\right)y^{(k)} - \frac{1}{5}\gamma$$

Denote the "true solution" by x^*, denote the error in the k-th iteration by $e^{(k)} = x^{(k)} - x^*$.

Find the error between two iterations:

The gradient update rule:

$$x^{(k+1)} = Ax^{(k)} + B = x^{(k+1)} = A\left(e^{(k)} + x^*\right) + B = Ae^{(k)} + Ax^* + B \overset{(*)}{=} Ae^{(k)} + x^*$$

$(*)$ - since x^* is the true solution and a fixed point, applying the gradient update rule to it won't change it

We get:

$$x^{(k+1)} = Ae^{(k)} + x^* \Leftrightarrow \underbrace{x^{(k+1)} - x^*}_{e^{(k+1)}} = Ae^{(k)}$$

And overall:

$$e^{(k+1)} = Ae^{(k)}$$

So, for the error to contract we need to ask for:

$$|A| < 1$$

Applying for x we require:

$$|1 - 20\gamma| < 1 \Leftrightarrow -1 < 1 - 20\gamma < 1 \Leftrightarrow 0 < 20\gamma < 2 \Leftrightarrow 0 < \gamma < \frac{1}{10}$$

Applying for x we require:

$$\left|1 - \frac{1}{5}\gamma\right| < 1 \Leftrightarrow -1 < 1 - \frac{1}{5}\gamma < 1 \Leftrightarrow 0 < \frac{1}{5}\gamma < 2 \Leftrightarrow 0 < \gamma < 10$$

We of course take the more restrictive bound and get that the maximal γ is:

$$\boxed{\gamma = \frac{1}{10}}$$

If we exceed this learning rate the gradient update rule would oscillate or even diverge, and we won't be able to guarantee to find the optimal solution.

2.c

As seen in 2.b, the value that determines this is $|A|$ which is required to be less than 1.

2.d

The variable that takes longer to converge is y.

We can see it using the same term $|A|$.

Let us choose a value close to $\frac{1}{10}$.

$$\left|1 - 20\left(\frac{1}{10} - \epsilon\right)\right| \sim |1 - 2 + 20\epsilon| \sim 1 - 20\epsilon$$

$$\left|1 - \frac{1}{5}\left(\frac{1}{10} - \epsilon\right)\right| = \left|1 - \frac{1}{50} + \frac{1}{5}\epsilon\right| = \left|\frac{49}{50} + \frac{1}{5}\epsilon\right|$$

For example, choosing $\epsilon = \frac{1}{1000}$ would give us:

$$1 - 20\epsilon = 1 - \frac{1}{50} = \frac{49}{50} < 1 - \frac{1}{5}\left(\frac{1}{10} - \epsilon\right) = \frac{49}{50} + \frac{1}{5000}$$

Since for y $|A|$ is closer to 1 the error would decrease slower, causing slower convergence.

3.

Function Loss $(\theta, \hat{\theta})$:

    If $(|\theta - \hat{\theta}| > 180)$:

        Return $\min(\theta, \hat{\theta}) + (360 - \max(\theta, \hat{\theta}))$

    Else:

        Return $|\theta - \hat{\theta}|$

The idea is – we always want the length of the sorter arc between the two angles.

4.

In all the sections the answer is of course relevant only for the points the derivative actually exists.

4.a

$$\frac{\partial}{\partial x} f(x+y, 2x, z)$$

Define $u: \mathbb{R}^3 \to \mathbb{R}^3$

$$u(x,y,z) = (u_1, u_2, u_3) = (x+y, 2x, z)$$

$$\frac{\partial f(u_1,u_2,u_3)}{\partial x} \overset{chain\ rule}{=} \frac{\partial f}{\partial \boldsymbol{u}} \cdot \frac{\partial \boldsymbol{u}}{\partial x} = \left(\frac{\partial f}{\partial u_1}, \frac{\partial f}{\partial u_2}, \frac{\partial f}{\partial u_3}\right) \cdot \begin{pmatrix} \frac{\partial u_1}{\partial x} \\ \frac{\partial u_2}{\partial x} \\ \frac{\partial u_3}{\partial x} \end{pmatrix} = \frac{\partial f}{\partial u_1}\frac{\partial u_1}{\partial x} + \frac{\partial f}{\partial u_2}\frac{\partial u_2}{\partial x} + \frac{\partial f}{\partial u_3}\frac{\partial u_3}{\partial x} =$$

$$= \frac{\partial f}{\partial (x+y)} \cdot 1 + \frac{\partial f}{\partial (2x)} \cdot 2 + \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial x} = \frac{\partial f}{\partial (x+y)} + 2 \cdot \frac{\partial f}{\partial (2x)}$$

Overall:

$$\boxed{\frac{\partial f(x+y, 2x, z)}{\partial x} = \frac{\partial f}{\partial (x+y)} + 2 \cdot \frac{\partial f}{\partial (2x)}}$$

4.b

$$\boxed{f_1\left(f_2\left(\dots f_n(x)\right)\right)' \overset{chain\ rule}{=} f_1'\left(f_2\left(\dots f_n(x)\right)\right) \cdot f_2'\left(f_3\left(\dots f_n(x)\right)\right) \cdot \dots \cdot f_{n-1}'\left(f_n(x)\right) \cdot f_n'(x)}$$

4.c

$$f_1\left(x, f_2\left(x, \dots f_{n-1}\left(x, f_n(x)\right)\right)\right)'$$

x is the only variable.

Denote:

$$y_i = f_{i+1}\left(x, \dots f_{n-1}\left(x, f_n(x)\right)\right)$$

$$\frac{df_1}{dx} = \frac{\partial f_1}{\partial x} + \frac{\partial f_1}{\partial y_1} \cdot \frac{dy_1}{dx} = \frac{\partial f_1}{\partial x} + \frac{\partial f_1}{\partial y_1} \cdot \frac{d}{dx} f_2\left(x, f_3\left(\dots f_{n-1}\left(x, f_n(x)\right)\right)\right)$$

$$= \frac{\partial f_1}{\partial x} + \frac{\partial f_1}{\partial y_1} \cdot \left(\frac{\partial f_2}{\partial x} + \frac{\partial f_2}{\partial y_2} \cdot \frac{d}{dx} f_3\left(x, f_4\left(\dots f_{n-1}\left(x, f_n(x)\right)\right)\right)\right) =$$

$$= \frac{\partial f_1}{\partial x} + \frac{\partial f_1}{\partial y_1} \cdot \left(\frac{\partial f_2}{\partial x} + \frac{\partial f_2}{\partial y_2} \cdot \left(\frac{\partial f_3}{\partial x} + \dots \left(\frac{\partial f_{n-1}}{\partial x} + \frac{\partial f_{n-1}}{\partial x} \cdot f_n'(x)\right)\right)\right)$$

Overall:

$$\boxed{f_1\big(x, f_2(x,\dots)\big)' = \frac{\partial f_1}{\partial x} + \frac{\partial f_1}{\partial f_2}\cdot\left(\frac{\partial f_2}{\partial x} + \frac{\partial f_2}{\partial f_3}\cdot\left(\frac{\partial f_3}{\partial x} + \cdots\left(\frac{\partial f_{n-1}}{\partial x} + \frac{\partial f_{n-1}}{\partial x}\cdot f_n'(x)\right)\right)\right)}$$

4.d

$$f\left(x + g\big(x + h(x)\big)\right)' = f'\left(x + g\big(x + h(x)\big)\right)\cdot\left(x + g\big(x + h(x)\big)\right)' =$$
$$= f'\left(x + g\big(x + h(x)\big)\right)\cdot\big(1 + g'\big(x + h(x)\big)\cdot\big(x + h(x)\big)'\big) =$$
$$= f'\left(x + g\big(x + h(x)\big)\right)\cdot\Big(1 + g'\big(x + h(x)\big)\cdot\big(1 + h'(x)\big)\Big)$$

Overall:

$$\boxed{f\left(x + g\big(x + h(x)\big)\right)' = f'\left(x + g\big(x + h(x)\big)\right)\cdot\Big(1 + g'\big(x + h(x)\big)\cdot\big(1 + h'(x)\big)\Big)}$$