Intro to deep learning - exercise 1

Guy Kornblit, 308224948; Mohamad Salama, 318983384

Programming Task

In this exercise we trained a deep neural network (multi-layered perceptron) to identify peptides that an HLA allele (HLA_A0201) detects. Train data contains ~3K positive examples and ~24.5K negative example. Each peptide consists of 9 amino acids (of 20 types).

We will address main topics that appeared in the process of research, compare models, and apply the best one on the test data of Covid19 peptides.

Preprocess stage:

1. **One hot encoding -** we noticed that each peptide could have multiple amino-acids of the same type, and that the order of appearance of each acid is important. So, we created one-hot encoding for every location on the peptide (9) to all the acids (20), resulting a feature matrix of 180 features.
2. **Split to train-validation-test -** To compare between tested models, we separated the train data in this manner - 0.8% train, 0.1% validation and 0.1% test.
3. **Working with Imbalanced data -** Finding out that the positive examples were a minority class (approximately 10% of the train data), we implemented the following steps:
    a. Stratification - keeps the ratio of positive examples in train, validation and test sets.
    b. Bigger train batch size (1024) - ensuring that the minority class is represented in each batch.
    c. Additional steps - weighted loss and oversampling will be discussed in improvements sections.

Additional general (fixed) configuration

1. **Loss** - we used Binary cross entropy, which enforces a single output neuron.
2. **Optimization** - We fixed an amount of 50 Epochs for all networks, and used the Early stopping module by TensorFlow, that stops after validation loss convergence lasts for at least 10 epochs. Namely, we stop training when gradient optimization exhausted.
3. **Metrics** - Since the problem is binary classification, such that the positive target is the relevant one (and the minority class), we used Recall and Precisions as main metrics in the model evaluation to measure if the model learns the positive class. Binary Accuracy was fixed on 0.5 on every experiment.

In addition, AUC was used to measure the probability of the model to rank a random positive sample higher than a random negative sample.

Baseline model

Architecture - We chose a model with 2 hidden FC layers with Relu activation, and a single output node that correlates with binary classification (detected\not detected). The number of neurons per layer were determined to mimic the structure of the input (180 features encoding 20 features for a general peptide). We used Adam optimizer of a binary cross entropy loss function.

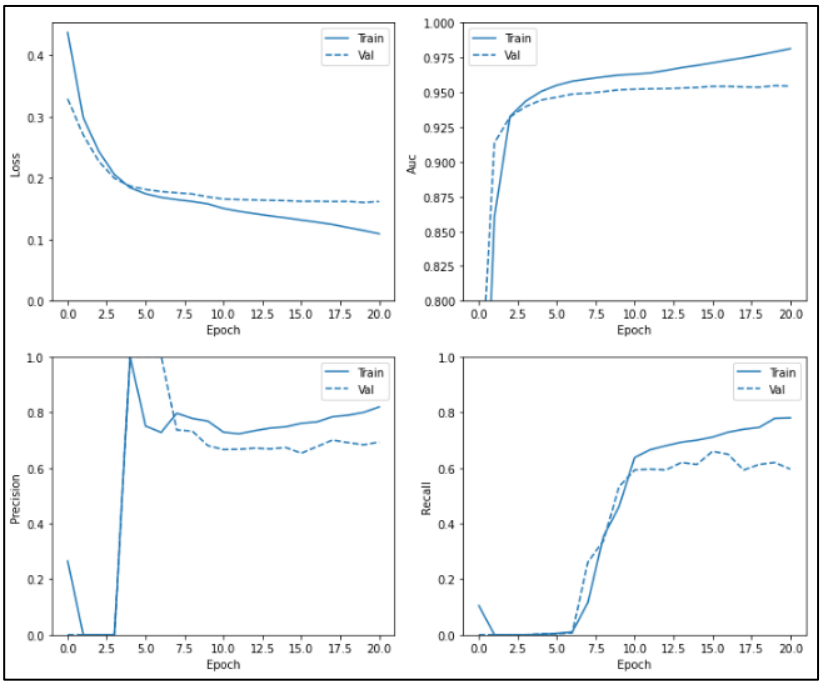
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_54 (Dense)             (None, 180)               32580
_________________________________________________________________
dense_55 (Dense)             (None, 20)                3620
_________________________________________________________________
dense_56 (Dense)             (None, 1)                 21
=================================================================
Total params: 36,221
Trainable params: 36,221
Non-trainable params: 0
```

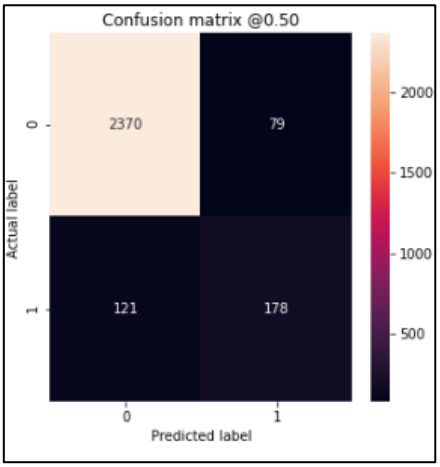
Loss	Binary cross-entropy
Optimizer	Adam
Learning rate	0.001

Model Architecture

Training - We can see by the loss plot that the model tends to slightly overfit the training data before the validation loss converge. In addition, there is a spike in the precision and recall percentages, which caused by rapid optimization throughout the first few epochs. Again, it shows that the train data achieve higher scores in every metric, due to overfit.

Validation Scores	**Loss**	**Accuracy**	**Recall**	**Precision**	**AUC**
	0.16	0.92	0.57	0.66	0.95

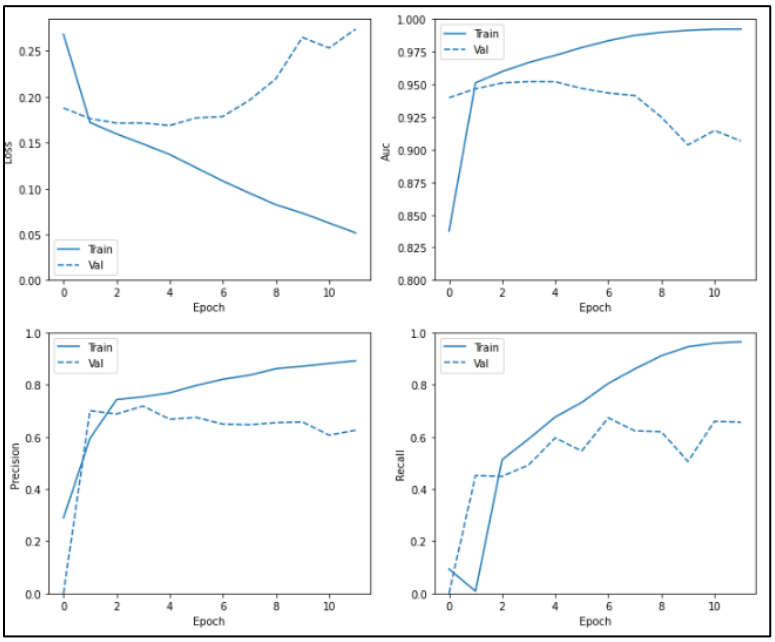


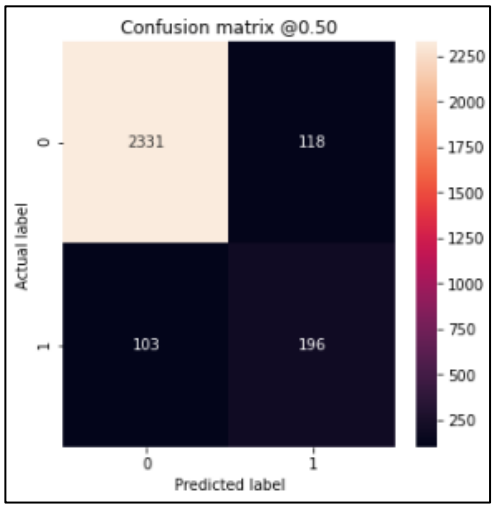


Modifications:

1. **Learning rate** - increasing to 0.01 (ten times the previous rate) resulted with increased pace of overfitting, getting higher validation loss, lower accuracy, and precision, but higher recall (by .07). Namely, the model quickly learned to prefer positive labeling (not that significantly though) but increased its error rate.

Validation Scores	Loss	Accuracy	Recall	Precision	AUC
	0.24	0.91	0.64	0.62	0.91





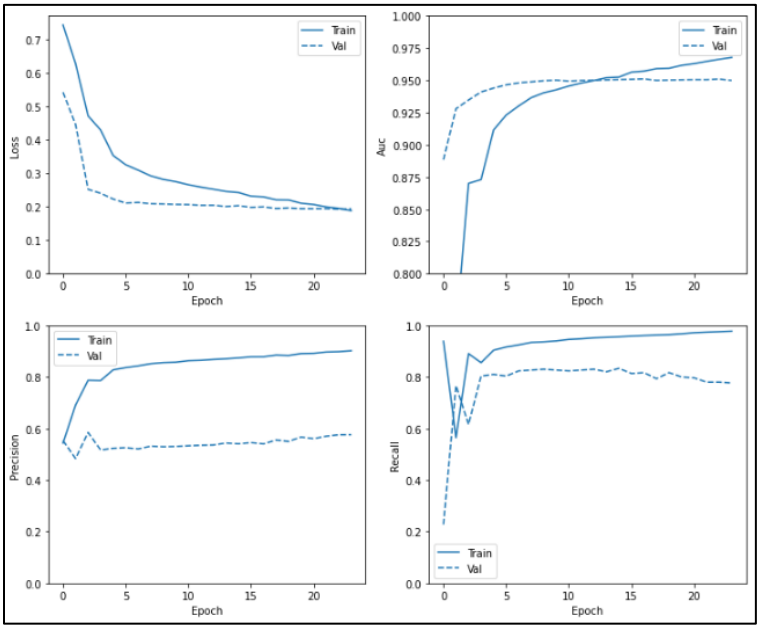
2. **Handling Imbalance between classes.**

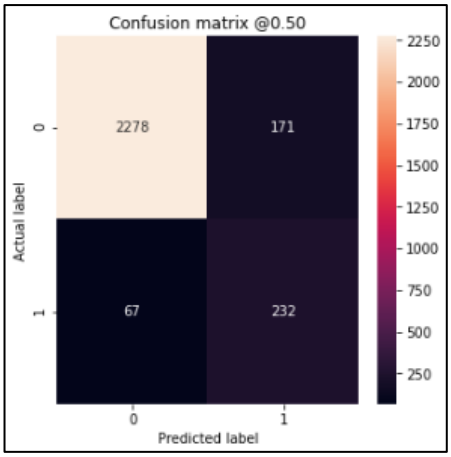
   To improve predictions performance, we tried to compensate class imbalance with operations on the train set and training process. (Learning rate in this part is the default 0.001).

   a. **Oversample -** basic random oversampling of positive examples. Namely, the process generates random duplicates to match the number of the negative examples.

   We can see that within about the same accuracy and precision levels, the Recall score increased.

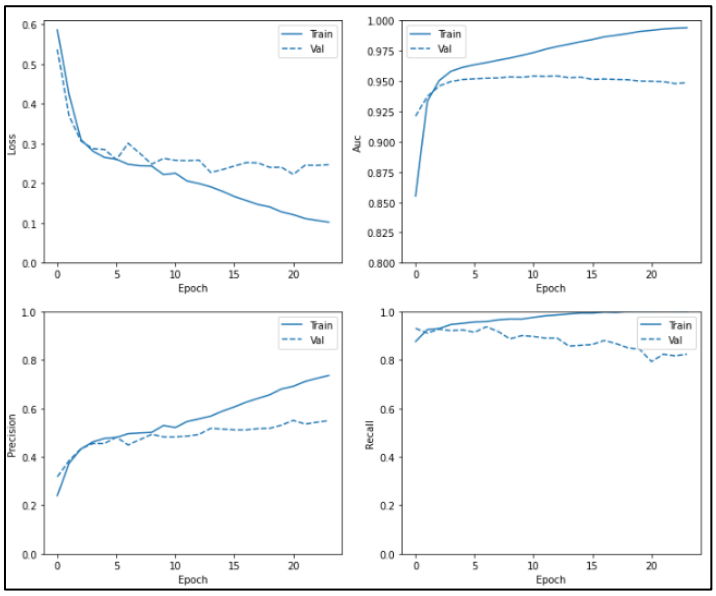
Validation Scores	Loss	Accuracy	Recall	Precision	AUC
	0.19	0.91	0.77	0.57	0.949

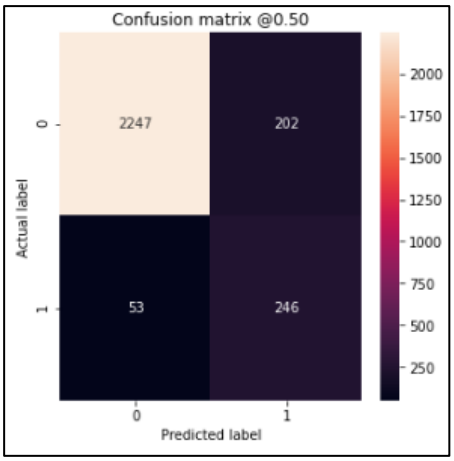




b. **Weighted loss** - weighting the loss function (during training only). This can be useful to tell the model to "pay more attention" to samples from an under-represented class. We used weights determined by $(\frac{1}{\#positives}, \frac{1}{\#negetives})$, scaled by the total amount of examples. Since we multiply the loss values, we no longer can compare loss values between models using this method.

Validation Scores	Loss	Accuracy	Recall	Precision	AUC
	0.24	0.90	0.82	0.549	0.948





In both cases, we can see that the method appears to have a similar, slightly stronger, effect as oversampling. Accuracy and precision are low due to more false positives, but conversely the recall and AUC are higher because the model also found more true positives.

Experiment different Architectures

Architecture (list of number of neurons in each layer + activation functions)	**Optimizer** (lr=0.001 unless stated o.w)	**Validation Loss**	**Validation Accuracy**	**Validation Recall**	**Validation Precision**	**Validation AUC**	#Epochs to val loss convergence (patience=10)
Baseline: 180+ReLu, 20+Relu, 1+Sigmoid.	Adam	0.16	0.92	0.57	0.66	0.95	20
Baseline	Adam (0.01)	0.24	0.91	0.64	0.62	0.91	13
Baseline	SGD	0.38	0.89	0	0	0.69	50 (max)
Baseline, oversample	Adam	0.19	0.91	0.77	0.57	0.949	23
Baseline, weighted Loss	Adam	-	0.90	0.82	0.549	0.948	23
180+Leaky_ReLu, 20+Leaky_ReLu, 1+Sigmoid.	Adam	0.16	0.91	0.56	0.63	0.95	18
As before Over sampled	Adam	0.21	0.89	0.85	0.5	0.94	13
As Before Weighted Loss	Adam	-	0.89	0.87	0.5	0.95	23
As Before. Weighted loss	Adam (0.01)	-	0.91	0.74	0.57	0.92	22
256+LR, 128+LR, 64+LR, 32+LR, 1+Sigmoid.	Adam	0.18	0.91	0.56	0.62	0.94	13
256+LR, 128+LR, 64+LR, 32+LR, 1+Sigmoid.	Adam (0.01)	0.23	0.913	0.5	0.62	0.92	11
As Before Weighted Loss	Adam	-	0.89	0.52	0.82	0.93	15
256+ReLu, 128+ReLu, 64+ReLu, 32+ReLu, 1+Sigmoid.	Adam	0.28	0.92	0.58	0.65	0.89	13
Same as Before Weighted Loss	Adam	0.35	0.91	0.69	0.58	0.90	14

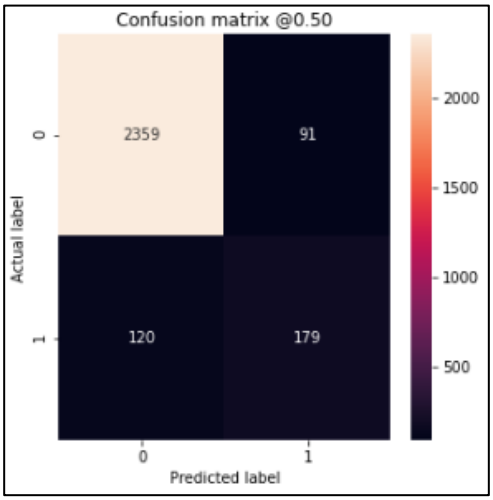
It seems that out tests could not find a sweet spot between a high recall and a reasonable precision. In addition, it seems that oversampling and weighting the loss are equivalent operations in terms of all metrics.

Based on the results, we decided to proceed with the same basic baseline model.

Model evaluation

We chose our baseline model, now we can test it on the held-out test data and check its performance. It seems that the performance is highly like the validation results, probably due to the sampling method of the validation and the test sets (same size, after stratification). Even though, the result seems robust in that manner.

	Loss	Accuracy	Recall	Precision	AUC
Validation Scores	0.16	0.92	0.57	0.66	0.95
Test Scores	0.16	0.923	0.59	0.66	0.95



Predict the detection of peptides from the Spikes of SARS-Cov-2

After training the model on the whole training data, we predict labels for the Covid-19 sequence, after converting it to peptides. Classification returned that 41 of the peptides were detected (3%). These are the 5 top scored peptides by our model:

	peptide	Prob
400	VIRGDEVRQ	0.93973
875	ALLAGTITS	0.92006
187	NLREFVFKN	0.85689
334	LCPFGEVFN	0.84212
299	KCTLKSFTV	0.83915

Theoretical questions

1. Claim: the composition of linear functions is a linear function, and the same applies to affine transformations.

   Proof:

   a. We know that every linear transformation $T: U \to V$ can be represented by a matrix $A \in M_{dim(V)\times dim(U)}(F)$. Namely, for $T_1, \dots, T_n$ linear transformations there are $A_1, \dots, A_n$ corresponding matrices.

   We also that a composition of linear transformations $T_1 \circ \dots \circ T_n$ (while it is well defined) is equivalent to multiplying their representing matrices. Thus, the composition $T_1 \circ \dots \circ T_n$ for every $n \in N$, **is equivalent to** $\boldsymbol{A = A_1 \cdot \dots \cdot A_n}$. Now, A is a matrix and therefore represents another linear transformation.

   b. If T is an Affine transformation, T can be represented by a matrix A such that $T(\vec{x}) = A\vec{x} + \vec{b}$ such that $\vec{b} \neq 0$ and $\vec{x}, \vec{b} \in \mathrm{R}^{\mathrm{d}}$. But we can also write

$$\begin{bmatrix} T(\vec{x}) \\ 1 \end{bmatrix} = \begin{bmatrix} & A & & \vec{b} \\ 0 & \dots & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix}$$

   Namely, we can consider an affine transformation from $dom(T) \to Im(T)$ as a linear transformation between vector spaces, and a representation matrix $A \in M_{dim(Im(T))+1\times dim(dom(T))+1}$. Thus, we get the claim based on the linear case. ■

2. Gradient Descent method.
   a. The stopping condition of the iterative scheme
   $$f(x^{n+1}) = f(x^n) - \alpha \nabla f(x^n)$$
   is when $|f(x^{n+1}) - f(x^n)| < \varepsilon$ for $\varepsilon > 0$ for all n > N for some N. Alternatively, when the difference equals 0, meaning the algorithm reached local minima (by Fermat's law).
   b. A stationary point is a point x, where x is the nth-updated parameters of the loss functions, is a point where $\nabla f(x) = 0$. To classify whether x is a local minima or maxima, we use Taylor second-order approximation -
   $$f(x + \mathrm{dx}) = f(x) + \nabla f(x) \cdot dx + (\mathrm{dx})^T H(\mathrm{x})\mathrm{dx} + O(\|\mathrm{dx}\|^3)$$
   $$= f(x) + (\mathrm{dx})^T H(x)\mathrm{dx} + O(\|\mathrm{dx}\|^3)$$
   Clearly, if -
   1. $f(x + dx) - f(x) = dx^\top \cdot H(x) \cdot dx + O(\|dx\|^3) \geq 0$ then we got x is a local minimum.
   2. $f(x + dx) - f(x) = dx^\top \cdot H(x) \cdot dx + O(\|dx\|^3) \leq 0$ then we got x is a local maximum.

   Now according to the Linear algebra theorem: Let $u^\top Au$ be a quadratic form, let A be a real symmetric matrix, and for i = 1,…, n, let $\det A_i$ be the determinant of the upper left $i \times i$ submatrix of A, then the quadratic form is positive semidefinite iff $\det A_i \geq 0$ for all i. And this theorem immediately give us the negative definiteness as well, if $B = -A$ then $u^\top Au$ is negative semidefinite iff $\det B_i \geq 0$ for all i.

   But also we have $\det B_i = (-1)^i \det A_i$, then we got that $u^\top Au$ is negative semidefinite iff $(-1)^i d_i > 0$, so let A be the Hessian matrix and we will got that if:
      a. If $(-1)^i \det H_i > 0$ for all i then x is a local maximum.
      b. If $\det H_i > 0$ for all i then x is a local minimum.
3. Assume the network is required to predict an angle (0-360 degrees). How will you define a prediction loss which accounts for the circularity of this quantity, i.e., the loss between 2 and 360 is not 358, but 2 (since 0 is 360).

```python
def loss_function(y_pred, y_label):
  loss1 = np.abs(y_pred - y_label) % 360
  loss2 = np.abs(360 - y_pred - y_label) % 360
  return np.min([loss1, loss2])
```

4. Explain why Cybenko and Hornik theorems also imply that linear combinations of translated and dilated RelU functions form a dense set in C[0,1].

Answer: The Cybenko theorem claims that, let σ be a continuous monotone function of the form:

$$\sigma(x) = \begin{cases} 1 & for\ x \to +\infty \\ 0 & for\ x \to -\infty \end{cases}$$

Then the family $f(x) = \sum \alpha_i \sigma(w_i x + b_i)$ is dense in $C[0,1]^n$, and the Hornik theorem extends the theorem to all monotonous bounded functions.

So, we have to construct a bounded, continuous function by linear combination of RELU function,

$f(x)$ = ReLU(x) − ReLU(x-1)

then we got that f is a continuous monotone function of the form:

$$f(x) = \begin{cases} 1 & for\ x \to +\infty \\ 0 & for\ x \to -\infty \end{cases}$$

So we can get the formulas of Cybenko and Hornik also on the RELU function.

5. Generalize the construction of a deep network that expresses a shallow network in O(N) neurons, that we saw in class, to signed functions.

Answer: we added two neurons, the first one in the second layer and the third one in the third layer. This two neurons handle the case when α_i is negative, which in case h_2 is negative we got h_1 is zero (negative values gets zero) and h_3 is non-negative so in the third layer, h_1 sums only h_1 and the positive result (h_2 *or* h_3).