Introduction to Deep Learning – 67822

Or Tal

Theoretical Part

1. Show that the composition of linear functions is a linear function. Show that the composition of affine transformations remains an affine function.

Given n linear transformations: $f_i(I) = W_i I$, assuming that the dimensions match, the composition of these linear transformations would then be: $f_n\left(f_{n-1}\left(\cdots\left(f_1(I)\right)\right)\right) = W_n W_{n-1} \cdots W_1 I = \widehat{W} I \Rightarrow$ the composition is a linear transformation of the input.

Similarly, for the affine case: $f_i(I) = W_i I + b_i$, the composition of n transformations would then be:

$$f_n\left(f_{n-1}\left(\cdots\left(f_1(I)\right)\right)\right) = W_n(W_{n-1}(\cdots(W_2(W_1 I + b_1) + b_2)\cdots) + b_{n-1}) + b_n =$$

$$W_n W_{n-1} \cdots W_1 I + W_n(W_{n-1}(\cdots(W_3(W_2 b_1 + b_2) + b_3)\cdots) + b_{n-1}) = \widehat{W} I + \hat{b}$$

$\Rightarrow$ The composition is an affine transformation of the input.

2. The calculus behind the Gradient Descent method:
   a. What is the stopping condition of this iterative scheme?

Given a multi-variable function $f(\cdot)$ which is defined and differentiable in some region in which wish to maximize/minimize its value.

Gradient Descent optimization iteratively defines: $x^{(n+1)} = x^{(n)} + sgn \cdot \alpha \nabla f(x^{(n)})$ where $sgn = \begin{cases} 1 \; maximize \\ -1 \; minimize \end{cases}$, and α is a step-size parameter.

The stopping condition for this iterative process would then be $|\nabla f(x^{(n)})| < \varepsilon$, for some small $\varepsilon > 0$.

$$f(x + dx) = f(x) + \nabla f(x) \cdot dx + dx^T \cdot H(x) \cdot dx + O(\|dx\|^3),$$

$$H_{ij}(x) = \frac{\partial^2 f}{\partial x_i \partial x_j}(x)$$

b. Use the second-order multivariate Taylor theorem,

to derive the conditions for classifying a stationary point as local maximum or minimum.

Assuming that $f: \mathbb{R}^m \to \mathbb{R}^n$.

Let $k = \max(m, n)$, point $x \in \mathbb{R}^m$ is a critical point if it satisfies one if the following:

- The rank of the derivative matrix $\nabla f(x)$ is less than k.
- The gradient vectors $\nabla f_1(x), \cdots, \nabla f_n(x)$ are linearly dependent.

We could then classify these stationary points by using the Hessian matrix $H(x)$ eigenvalues:

- If all eigenvalues are positive, then $f(x)$ is a local minimum.
- If all eigenvalues are negative, then $f(x)$ is a local maximum
- If some eigenvalues are zero, we would need to use a higher order approximation to classify the stationary point.
- Else $f(x)$ is a saddle point.

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 350 is not 348, but 2 (since 0 is 360..). Write your answer in a tensorflow-codable form.

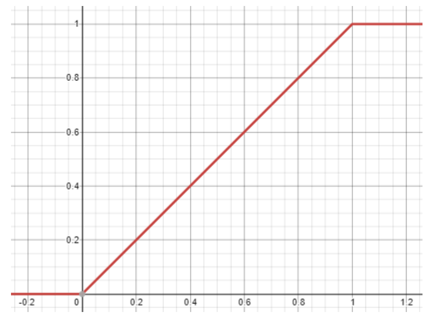
The code below assumes that we would like to have $0 \leq y\_pred \leq 360$ hence if $y\_pred > 360$ or $y\_pred < 0$ we would take the modulus of the angle into account for the error prediction.

```
def angular_loss(y_true, y_pred):
    """
    function to calculate mean absolute loss between angles
    :param y_true: vector of ground truth values,
                   assuming values in range [0,360] (including floats)
    :param y_pred: vector of predictions, would be taken as modulus 360 angle
    :return: mean absolute loss between angles
    """
    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.cast(y_pred, tf.float32)
    shift = (360 - y_true) % 360    # calculate a shift vector
    y_pred = (y_pred + shift) % 360
    loss = tf.where(y_pred < 180, y_pred, 360 - y_pred)
    return tf.reduce_mean(loss)
```

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].

Cybenko's theorem states that for every monotonic continuous function where $\sigma(\infty) = 1,\ \sigma(-\infty) = 0$ there exists a function family $f(x) = \sum_{i=1}^{n} \alpha_i \sigma(w_i \cdot x + b_i)$ that is a dense set in $C([0,1])$.

Hornik's theorem expands Cybenko's theorem to every bounded function.



Consider the following 'ramp' function: $g(x) = \sigma(x) - \sigma(x-1)$, where $\sigma \coloneqq ReLU$.

$g(x)$ is continuous up to finite number of discontinuity points, and is monotonic.
in addition $g(-\infty) = 0, g(\infty) = 1 \Rightarrow$ the conditions for the mentioned Theorem holds therefore the family: $f(x) = \sum_{i=1}^{n} \alpha_i g(w_i \cdot x + b_i)$ is dense in $C([0,1])$.

We may further notice that

$$f(x) = \sum_{i=1}^{n} \alpha_i (\sigma(w_i \cdot x + b_i) - \sigma(w_i \cdot (x-1) + b_i) = \sum_{i=1}^{n} \alpha_i \sigma(w_i \cdot x + b_i) + \sum_{i=1}^{n} -\alpha_i \sigma(w_i \cdot x + \hat{b}_i)$$

Define $\forall i \in \{n+1, \cdots, 2n\}$: $\alpha_i = -\alpha_{i-n},\ b_i = \hat{b}_{i-n}$, we would then get:

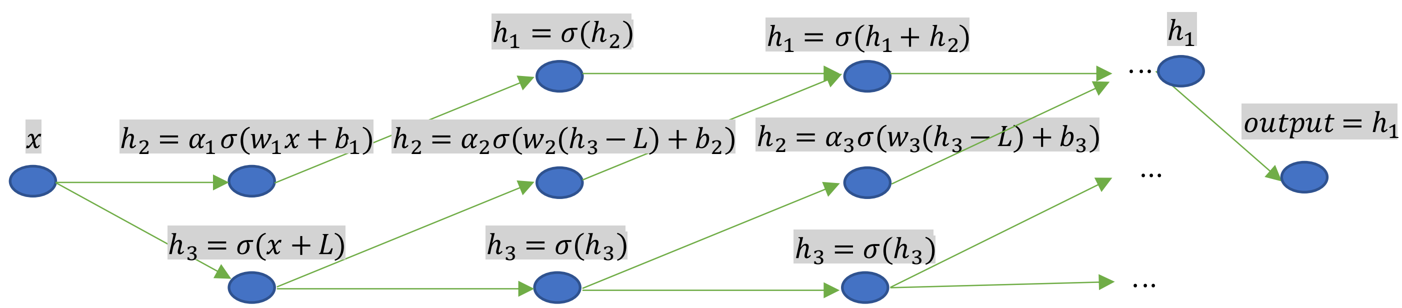
$$f(x) = \sum_{i=1}^{n} \alpha_i g(w_i \cdot x + b_i) = \sum_{i=1}^{2n} \alpha_i \sigma(w_i \cdot x + b_i)$$

$\Rightarrow$A ReLU based function family which is a dense set in $C([0,1])$, as requested.

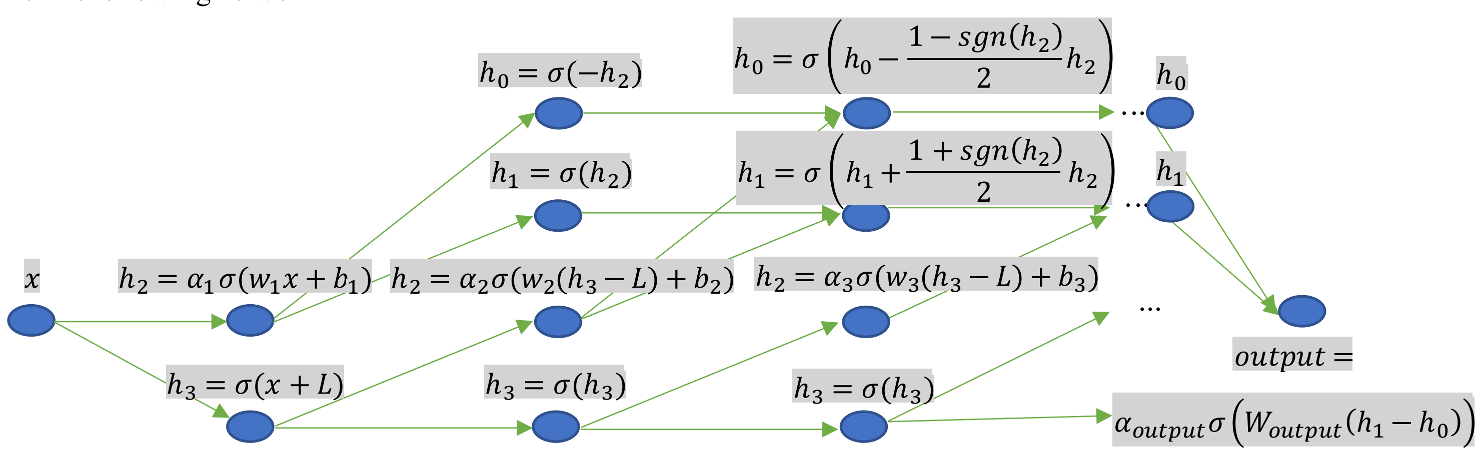
5. Generalize the construction of a deep network that expresses a shallow network in O(N) neurons, that we saw in class, to signed functions.

A shallow network expresses the following function: $\sum_{i=1}^{n} \alpha_i \sigma(W_i X + b_i)$ where W_i, b_i are the i'th node parameters, σ is a relu activation function, and α_i is the corresponding weight the output assigns to the i'th node output.

We've seen in class that in case that $\forall i$: $\alpha_i > 0$, the following deep network scheme indeed describes the above function:



Assuming each node has an activation output by default, allowing some α_i to be ≤ 0 could result a negative value in h_1 arm's ReLU activation input therefore forcing the value to be zero. Following the principle in the above scheme we could offer the following network:



Where arm h_0 keeps track of the negative contributions to the final sum, arm h_1 keeps track of the positive contributions to the final sum, and the output is being calculated by $\alpha_{output} \sigma\left(W_{output}(h_1 - h_0)\right)$ where $\alpha_{output} = sgn(h_1 - h_0)$ and $W_{output} = sgn(h_1 - h_0) \cdot I$.

In a more formal way, by using $O(n)$ nodes this network calculates:

$$\sum_{j \in [n]:\, \alpha_j > 0} \alpha_j \sigma\left(W_j X + b_j\right) - \sum_{i \in [n]:\, \alpha_i \leq 0} (-\alpha_i) \sigma(W_i X + b_i) = \sum_{i=1}^{n} \alpha_i \sigma(W_i X + b_i)$$

As requested.

Practical Part

Model Training:

In this exercise I have trained FC networks with one or two outputs with various hyperparameters configurations.
I have used mean absolute error for the single output configurations training and binary cross entropy for the double output configurations training.
During the experiments I have made, I found out that a FC network with ReLU activations, with a last layer sigmoid activation in the single output case and an output Softmax layer in the double output case, had the best performance from the tested combinations of $\{sigmoid, relu\}^n$ activations.
I have considered several evaluation matrices options (loss, precision, recall and accuracy) as several training paradigms, keeping the model that performed best on evaluating the specific metric during the training step.
I then construct a committee wrapper model, taking 5 models that had the highest accuracy over the test set, and output the committee prediction result (mean prediction for all 5 models) – as learned in a previous course, this ensemble method is commonly used when wanting to improve a model's prediction when trained on relatively small datasets.

Data Preprocessing:

First, I encode the data into numerical representations – for each 9-mer in the dataset, I construct a one-hot vector for each letter of the 9-mer input sequence, and then concatenate all vectors into one vector representing the 9-mer sequence. Labels are saved as $[1]$ or $[1, 0]$ for the positive case and $[0]$ or $[0, 1]$ for the negative case, matching single or double output networks.
Second, I divide the data 95/5 for train/test sets, validation set is randomly chosen from the train set in each training, and it is 10% of the entire data (train/val/test = 85/10/5).
The validation set is used for model evaluation while training and the test set is used to compare the various models.

Batch Sampling:

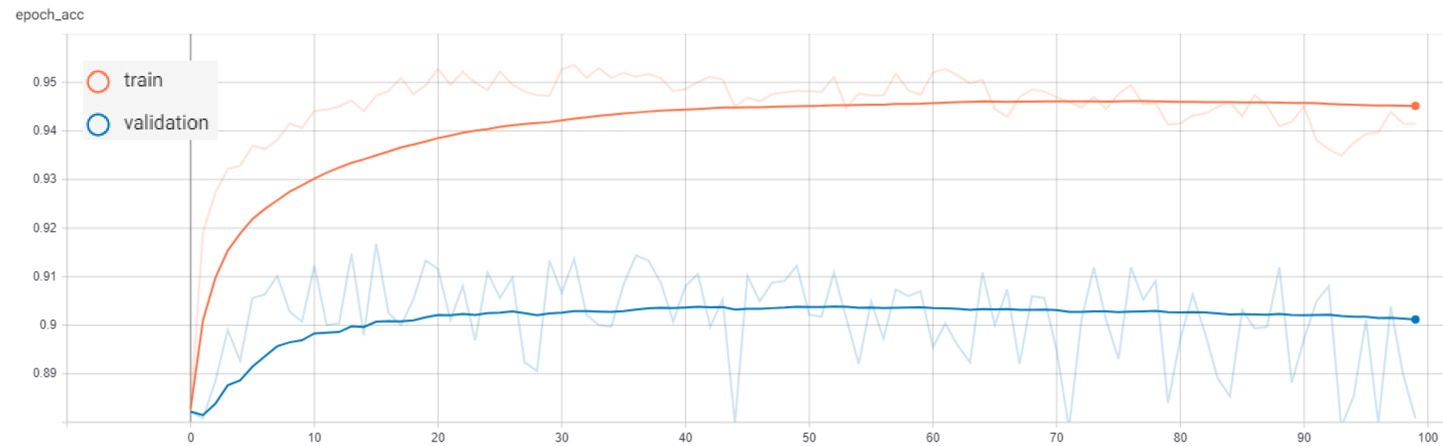
The given dataset is highly imbalanced ($pos: neg \approx 1: 8$), hence I've created a generator class to sample equal number of positive and negative examples in each training batch, randomly sampling positive examples, and sampling in order (the shuffled) negative examples.

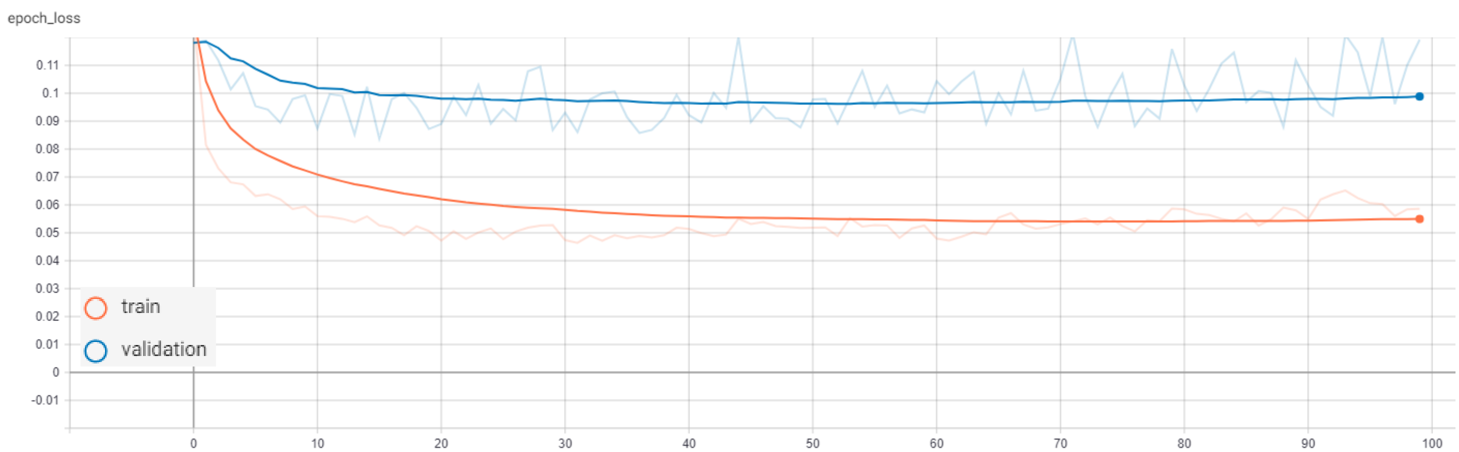
Results

The model that achieved the best accuracy on the test set was:

model: mlp_v2_o1_1, accuracy = 0.9261744966442953

```
Nodes per layer:         [100, 100, 100, 100, 50]
Activations:             ['relu', 'relu', 'relu', 'relu', 'relu']
Num outputs:             1
Num epochs in training:  100
Learning rate:           0.001
Evaluation metric:       recall
```
← metric used for model evaluation over the validation set during training





The top 5 scoring models (constructed the committee from these) accuracy on the test set:

model: mlp_v2_o1_1, accuracy = 0.9261744966442953

model: mlp_v2_o1_8, accuracy = 0.912751677852349

model: mlp_v2_o1_3, accuracy = 0.9060402684563759

model: mlp_v2_o1_4, accuracy = 0.9026845637583892

model: mlp_v0_o1_3, accuracy = 0.8993288590604027

Committee achieved accuracy: 0.9328859060402684

The Top 5 Positive classified Peptides sorted by score：

index：820, 9-mer：LLFNKVTLA, probability：0.9999997854232788
index：1, 9-mer：FVFLVLLPL, probability：0.9999924659729004
index：1059, 9-mer：VVFLHVTYV, probability：0.9999923229217529
index：1219, 9-mer：FIAGLIAIV, probability：0.9999687433242798
index：268, 9-mer：YLQPRTFLL, probability：0.9999364137649536

Attached Files

1. Code files - all files used for training and evaluation
   - data_generator.py
   - models.py
   - train.py
   - detect_cov_pos_from_fasta.py
   - gen_dset_from_txt_files.py
2. Results summary txt files
3. README with cse username

How to run files

1. Preprocess – from cmd:
   python gen_dset_from_txt_files.py --neg <negative_txt_path> --pos <positive_txt_path> --out_dir <output_path>
2. Train – from cmd：
   python train.py --train <train_npy_path> --test <test_npy_path> --w_dir <weights_dir_path> --name <model_name(opt)>
3. Detect positive covid19 peptides in .fasta file – from cmd：
   python detect_cov_pos_from_fasta.py --dst <fasta_path> --w_dir <base_weights_dir_path>