Nowadays, neural network models, especially deep models, have been dominating a majority of machine learning tasks, from face detection and object retrieval to speech recognition and language understanding. In this lab session, we will focus on the classical *Artificial Neural Network (ANN)*, with a toy application in digit recognition. Since implementing an efficient neural network model often requires a lot of effort, in real projects, it is common and highly recommended to make use of existing libraries. There is a number of libraries for deep learning, released recently by big tech companies and laboratories, such as:

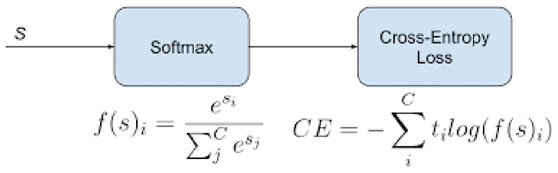
- **Tensorflow** from Google
- **Torch and PyTorch** from Facebook
- **CNTK** from Microsoft
- **Caffe** from UC Berkeley
- **Theano** from University of Montreal

Pytorch seems to be the most popular one at the moment. Thus, we will use Pytorch for the exercises of the Deep Learning course. First, make sure you have finished installing Pytorch as specified at the setup guide.

1 Cross-entropy Loss

In this exercise, we will implement the cross-entropy Loss. Cross-entropy builds upon the idea of entropy from information theory. Cross-entropy loss is used when adjusting model weights during training. The aim is to minimize the loss, i.e, the smaller the loss the better the model. A perfect model has a cross-entropy loss of 0. Normally it is used for multi-class and multi-label classification.

Multi-class classification: Each sample can belong to one of the C classes. The neural network will have C output neurons that can be gathered in a vector s (of scores). The target (ground truth) vector t will be a one-hot vector with a positive class and $C-1$ negative classes. This task is treated as a single classification problem of samples in one of C classes. Usually an activation function (Sigmoid / Softmax) is applied to the scores before the



s
Softmax
Cross-Entropy Loss
$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$
$CE = -\sum_i^C t_i log(f(s)_i)$

Figure 1: Softmax is applied to the scores before the cross-entropy loss computation [3].

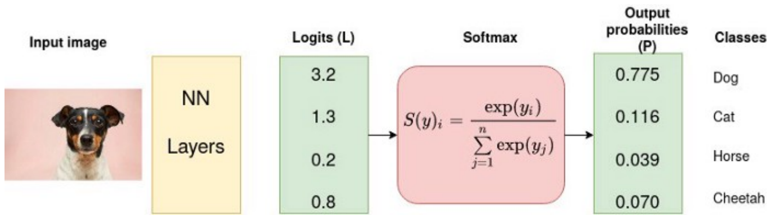


Figure 2: Softmax converts scores (logits) into probabilities [3].

cross-entropy computation (see Fig. 1). With Softmax, your model predicts a vector of probabilities based on the scores (logits) (see e.g., Fig. 2). For instance given the vector of probabilities [0.7, 0.2, 0.1] that sum up to 1, the first entry is the most likely. Softmax converts logits into probabilities. The purpose of the cross-entropy is to take the output probabilities (P) and measure the distance from the truth values (as shown in Fig. 3). We would calculate the cross-entropy loss using the formula given in Fig. 4. The code for this exercise is in the file *Cross Entropy Loss.ipynb*.

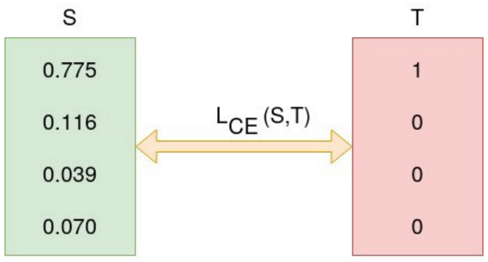


Figure 3: Cross Entropy [3].

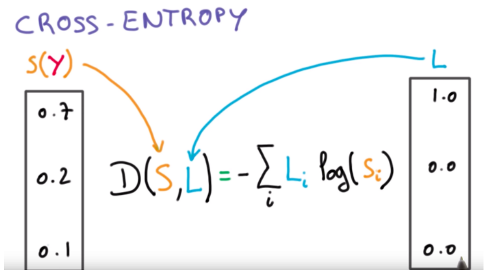


Figure 4: Cross Entropy Formula [3].

Step 1: Create the dataset For this exercise, we will rely on a randomly generated dataset. For the data X, you should generate a distribution of size batch_size $\times$ C that is uniform and gets float values from (uni_s, uni_e). uni_s and uni_e are hyper-parameters. For the labels Y initialize a vector of size batch size that takes random integer values between 0 and C.

Step 2: Compute cross-entropy Compute the cross-entropy loss using the corresponding Pytorch functions.

Step 3: Call the cross-entropy formula Call the function that calculates the cross-entropy loss manually along with the corresponding arguments.

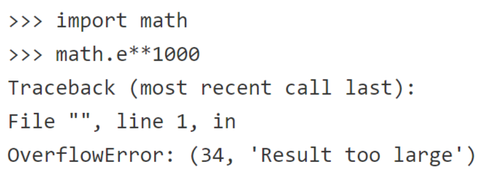
Step 4: Compute the cross-entropy loss manually Use the following two formulas for computing the loss. The first one is the softmax and outputs a probability distribution over the classes

$$f(s_i) = \frac{e^{s_i}}{\sum_{j=1}^{C} e^{s_j}} \tag{1}$$

where s_i is the score for class i normalized by the scores for all other classes (including the class itself). C is the number of classes. The cross-entropy formula is:

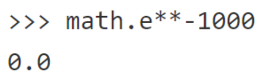
$$\text{Loss} = -\sum_{i=1}^{C} t_i \log(f(s_i)) \tag{2}$$

$f(s_i)$ is the output of our softmax function. It is a prediction, so we can also call it $\hat{y}$. t_i is the ground truth for that class, i.e., the one hot encoded label



```
>>> import math
>>> math.e**1000
Traceback (most recent call last):
File "", line 1, in
OverflowError: (34, 'Result too large')
```

Figure 5: Overflow error [4].



```
>>> math.e**-1000
0.0
```

Figure 6: Numerical stability issues [4].

of the true class, only one entry is one, the rest are zeroes.

Step 5: Compute the cross-entropy loss manually (version 2) For the loss in Eq. 2 in the case that we pass the log inside, the term with the log can be rewritten as:

$$\log\left(\frac{e^{s_i}}{\sum_{j=1}^{C} e^{s_j}}\right) = \log(e^{s_i}) - \log\left(\sum_{j=1}^{C} e^{s_j}\right) = s_i - \log\left(\sum_{j=1}^{C} e^{s_j}\right) \tag{3}$$

Step 6: Compute a numerically stable cross-entropy loss with Pytorch However, one issue with the aforementioned implementations is numerical stability. Why? To illustrate the problem, let's take 2 examples for our s_i sequence of numbers: {1000, 1000, 1000} and {-1000, -1000, -1000}. Due to our amazing mathematical ability, we know that feeding either of these sequences into the softmax function will yield a probability distribution of {1/3, 1/3, 1/3} and the log of 1/3 is a reasonable negative number. Now let's try to calculate one of the terms of the summation in python. We experience overflow and numerical stability issues (see Figs. 5 and 6). To alleviate this issue, we can rely on the logsumexp trick and more details for the exact calculation of the formulas can be found in here. For this step use the last part of Eq. 3 but for numerical stability use the logsumexp function of Pytorch.

Step 7: Compute manually a numerically stable cross-entropy loss The logsumexp trick indicates that:

$$\log(\sum_{j=1}^{C} e^{s_j}) = c + \log(\sum_{j=1}^{C} e^{s_j - c}) \tag{4}$$

where $c = max(s_1, ..., s_C)$. Compute manually a numerically stable cross-entropy loss according to this formula.

Step 8: Compute the percentage loss using 3 digits precision Return True in the case that the Pytorch computed loss and the manually computed loss have a precision of 3 digits and then compute the percentage of the loss between the cross-entropy loss and the manually computed loss for the given number of iterations.

Step 9: Compute the percentage loss for the following two cases

- Values when the uniform distribution takes values between 0 and 1,
- 0 and 100.

Do this for the Pytorch computed cross-entropy loss and for each of the manually computed versions of the cross-entropy loss. Which are your conclusions?

2 Artificial Neural Network with Pytorch

In this exercise, we will implement an exercise similar to the one from the previous lab, but using Pytorch instead of sklearn. The code for this exercise is in the file *MLP.ipynb*. Note that we have added TODO comments for all the lines that code is required from your side.

Step 1: Load dataset Run the code to load the digits dataset, split train/test and add the bias term.

Step 2: Plot image Add code to plot the first image of the training set to visualize what your input look like.

Step 3: Build an ANN model using the Pytorch interface In this step and in the follwing steps, you need to call the functions from the Pytorch interface to build an ANN model to classify the digit images.

- First, start with transforming the data into Pytorch tensors
- Then, add the required arguments to your model that is a linear stack of layers. Use the instructions in the code of the simpleMLP class to construct the network of two linear layers with a relu activation function.
- Enable batching of the training data by using the Pytorch dataloader. Complete also the Dataset class for that.

Step 3: Train the model Train the model by following the instructions in the comments. Conduct forward passes to update the weights of the model using the cross-entropy loss defined in Pytorch. You will use Adam as an optimizer, a learning rate of 0.001, and a hidden layer size of 32. You will train the model for 50 epochs.

Step 4: Evaluate the model Fill in the missing code to perform prediction and evaluation.

Step 5: Visualize the classification result Run the code to visualize the classification results.

Step 6: Run the model with different learning rates. Plot the validation accuracy using different learning rates of [1e-1, 1e-2, 1e-3, 1e-4, 1e-5] by keeping the rest of the hyperparameters fixed as defined at step 3. Note that the e-suffix represents times ten raised to the power. What do you observe?

Step 7: Run the model with different neural network hidden layer sizes. Plot the validation accuracy using different neural network layer hidden sizes [16, 32, 64, 128, 256, 512] by keeping the rest of the hyperparameters fixed as defined at step 3. What do you observe?

Step 8: Run the model with different optimizers Plot the validation accuracy using different optimizers by keeping the rest of the hyperparameters fixed as defined at step 3. The set of optimizers are defined in the ipython notebook. What do you observe?

Step 9: Add L2 and L1 regularization to your model The most popular regularization terms are L2 regularization, which is the sum of squares of all weights in the model, and L1 regularization, which is the sum of the absolute values of all weights in the model. In PyTorch, we could implement regularization pretty easily by adding a term to the loss. After computing the loss, whatever the loss function is, we can iterate the parameters of the model, sum their respective square (for L2) or abs (for L1), and backpropagate. For the regularization use a lambda of 0.001.

Step 10: Repeat steps 3-5 You should do this for the newly defined network in function ComplexMLP, where you should also use the Dropout, and the Batchnorm modules. More details about the network can be found in the ipython notebook.

References

[1] `https://pytorch.org/assets/deep-learning/Deep-Learning-with-PyTorch.pdf`

[2] `https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html`

[3] `https://medium.com/unpackai/cross-entropy-loss-in-ml-d9f22fc11fe0`

[4] `https://blog.feedly.com/tricks-of-the-trade-logsumexp/`

[5] `https://gregorygundersen.com/blog/2020/02/09/log-sum-exp/`