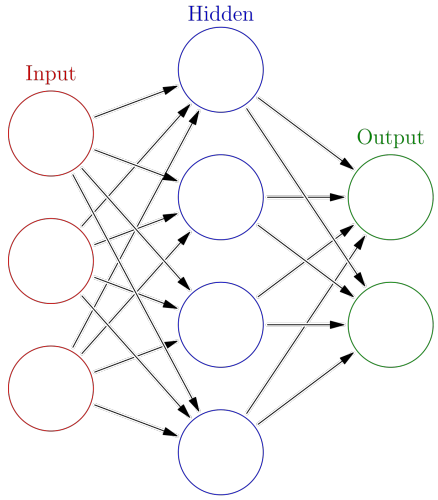

image from Wikipedia

CS440/ECE448 Fall 2023

Assignment 10: Neural Nets and PyTorch

Due date: Monday November 13th, 11:59pm

The goal of this assignment is to employ neural networks, nonlinear and multi-layer extensions of the linear perceptron, to classify images into four categories: ship, automobile, dog, or frog. That is, your ultimate goal is to create a classifier that can tell what each picture depicts.

You will improve your network from MP9 using modern techniques such as changing the activation function, changing the network architecture, or changing other initialization details.

You will be using the PyTorch and NumPy libraries to implement these models. The PyTorch library will do most of the heavy lifting for you, but it is still up to you to implement the right high-level instructions to train the model.

You will need to consult the PyTorch documentation, linked multiple times on this page, to help you with implementation details. **We strongly recommend this PyTorch Tutorial, specifically the Training a Classifier section since it walks you thorugh building a classifier very similar to the one in this assignment.** There are also other guides out there such as this one.

Template package and submission

The template package contains your starter code and also your training and development datasets (packed into a single file).

In the code package, you should see these files:

- **reader.py** - This file is responsible for reading in the data set. It creates a giant NumPy array of feature vectors corresponding to each image.
- **mp10.py** - This is the main file that starts the program, and computes the accuracy, and confusion matrix using your implementation.
- **neuralnet.py** - This is the file where you will be doing all of your work.

Modify only neuralnet.py.

Run `python3 mp10.py -h` in your terminal to find more about how to run the program.

There is one optional (ungraded) submission part where you can compete on a leaderboard with your best network using **neuralnet_leaderboard.py**. The binary files **net.model** and **param_dict.state** are generated when you run mp10.py --part 3 and are used to load in your best model submission, whose architecture must be defined in **neuralnet_leaderboard.py** This will be the model used to get your score on the leaderboard. The leaderboard is sorted by your performance on the hidden test set, but this correlates well with the dev set performance. The leaderboard is only for bragging rights and is worth no points.

You will need to import torch and numpy. Otherwise, you should use only modules from the standard python library. **Do not use torchvision**.

The autograder doesn't have a GPU, so it **will fail** if you attempt to use CUDA.

Dataset

The dataset consists of 3000 **31x31** colored (RGB) images (a modified subset of the CIFAR-10 dataset, provided by Alex Krizhevsky). This set is split for you into 2250 training examples (which are a mostly balanced sample of cars, boats, frogs and dogs) and 750 development examples.

The function load_dataset() in **reader.py** will unpack the dataset file, returning images and labels for the training and development sets. Each of these items is a Tensor.

Modern Network

In this part, you will try to improve your performance by employing modern machine learning techniques. These include, but are not limited to, the following:

1. **Choice of activation function**: Some possible candidates include Tanh, ELU, softplus, and LeakyReLU. You may find that choosing the right activation function will lead to significantly faster convergence, improved performance overall, or even both.
2. **L_2 Regularization:** Regularization is when you try to improve your model's ability to generalize to unseen examples. One commonly used form is L_2 regularization. Let $\mathcal{R}(W)$ be the empirical risk (mean loss). You can implement L2 regularization by adding an additional term that penalizes the norm of the weights. More precisely, your new empirical risk becomes

   $$\mathcal{R}(W) := \mathcal{R}(W) + \lambda \sum_{W \in P} \|W\|_2^2$$

   where P is the set of all your parameters and λ (usually small) is some hyperparameter you choose. There are several other techniques besides L_2 regularization for improving the generalization of your model, such as dropout or batch normalization.
3. **Network Depth and Width:** The sort of network you implemented in part 1 is a **two-layer network** because it uses two weight matrices. Sometimes it helps performance to add more hidden units or add more weight matrices to obtain greater representation power and make training easier.
4. **Using Convolutional Neural Networks:** While it is possible to obtain nice results with traditional multilayer perceptrons, when doing image classification tasks it is often best to use convolutional neural networks, which are tailored specifically to signal processing tasks such as image

recognition. See if you can improve your results using convolutional layers in your network. Note: The input to a CNN layer must be 4-dimensional: (batch_size, channel_num, height, width). Since your original input to the network is 2D, you'll need to reshape it before feeding it to the CNN layer.

Try to make your classification accuracy as high as possible, subject to the constraint that you may use at most **500,000 total parameters**. This means that if you take every floating point value in all of your weights including bias terms, i.e. as returned by this pytorch utility function, you only use at most 500,000 floating point values.

You should be able to get an accuracy of at least **0.79** on the development set.

Some tips

If you're using a convolutional net, you need to reshape your data in the forward() method, and not the fit() method. The autograder will call your forward function on data with shape (N,2883). That's probably not what your CNN is expecting. It's very helpful to print out the shape of key objects before/after each layer when trying to debug dimension issues. (the `.view()` method from tensors is very useful here)

Apparently it's still possible to be using a 32-bit environment. This may be ok. However, be aware that recent versions of PyTorch are optimized for a 64-bit environment and that is what Gradescope is using.

Ensure to use data standardization, as done in MP9.

The leaderboard

For your own enjoyment, we have provided also an anonymous leaderboard for this MP. Even after you have full points on the MP, you may wish to try even more things to improve your performance on the hidden test set by tuning your network better, training it for longer, using dropouts, data augmentations, etc. For the leaderboard, you can submit the **net.model** and **state_dict.state** created with your best trained model (after running mp10.py --part 3) alongside **neuralnet_leaderboard.py** (note that these need not implement the same thing, as you could wish to do fancy things that would be too slow for the autograder). We will not train this specific network on the autograder, so if you wish to go wild with augmentations, costly transformations, more complex architectures, you're welcome to do so. Just do not exceed the 500k parameter limit, or your entry will be invalid. Also, please do not use additional external data for the sake of fairness (i.e. using a resnet backbone trained on ImageNet would be very unfair and counterproductive).