For this project , I did two experiments on a Linear Model in PyTorch. For the first experience, I wanted to study the effect of the activation function. First, I got the results using sigmoid as the activation function. And then, I tried the same model with ReLU as the activation function.

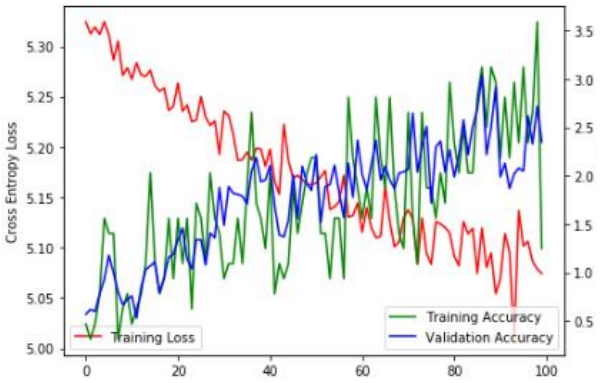
1) The input to the sigmoid function is transformed into a value between 0 and 1. Inputs that are much larger than one are transformed to 1. Values much smaller than 0 are snapped to 0. A problem with this function is that it saturates. It means that high values snap to 1, and small values snap to -1 or 0. Further, it is really sensitive to changes around the mid-point of its input, such as 0.5. Once saturated, it becomes challenging for the learning algorithm to continue to adapt the weights to improve the performance of the model. I wanted to see whether this issue of saturation is affecting our image classification problem or not. So I decided to use the function which can provide more sensitivity to the activation sum input and avoid easy saturation. The solution is to use ReLU. This function is a simple calculation that returns the value provided as input directly, or the value 0 if the input is 0 or less.

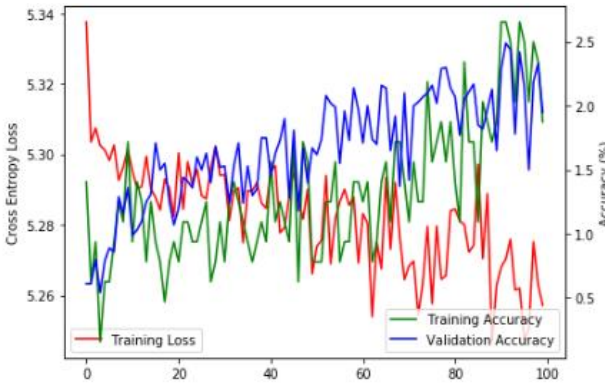
Data:
Training Data Shape: (640, 12288)
Training Labels Shape: (640,)
Validation Data Shape: (10000, 12288)
Validation Labels Shape: (10000,)

Neural Network Architecture:
dense = torch.nn.Linear(in_features=num_features, out_features=num_classes)
1) activation = torch.nn.LogSigmoid(),
2) self.activation = torch.nn.ReLU()
batch_limit = 20, batch_size = 32
num_features = 64*64*3, num_classes = 200
max_iter = 100, model = LinearModel
optimizer = torch.optim.SGD
criterion = torch.nn.CrossEntropyLoss()

As shown above, I used the same training and validation set, the same features and classes, and also the same loss function (CrossEntropy). I used the suggested tiny-imagenet-200 package. The training set contains 200 different classes, and for each of these classes, the package includes 500 images. The validation set contains 50 images for each of the 200 classes. We are using a flattened format that is suitable for training. The only difference between the two models is the activation function. This way, I can make sure I am just evaluating the effect of using different activation functions. The control model that I started with uses Sigmoid as the activation function. Then I change the activation function to ReLU.

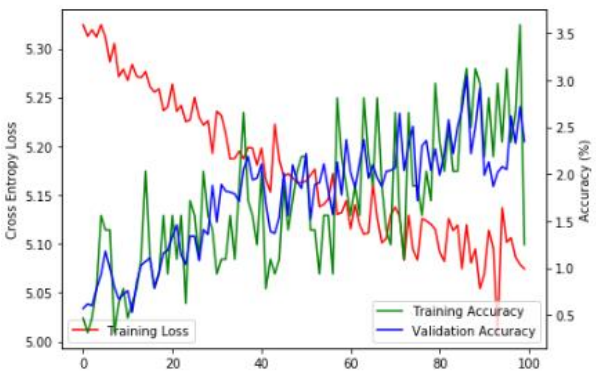
The left image below belongs to the control model. The right belongs to the experimented model:

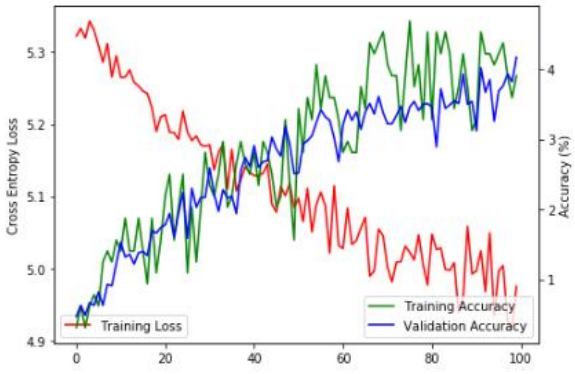




There are several differences that we can discuss on these two models. First, as we can see, using RLU, the training error is less. Also, the validation accuracy and training accuracy are more or le ss similar in the control model with the sigmoid function. Nevertheless, when using ReLU, valida tion accuracy is even better than training accuracy, which seems excellent. However, checking th e exact values, we can observe that overall accuracy, for both training set and validation set are hi gher when we are using sigmoid as the activation function. So using ReLU did not improve the o verall accuracy of training and validation, and for this image classification problem, sigmoid is p erforming well as the activation function.

2) As the second experiment, I decided to check the effect of using the different optimizers.
As we know, optimization is a process of searching for parameters that minimize the loss (here w e use CrossEntropyLoss as our loss function). The data and all classes and features are the same a s above. I used sigmoid as the activation function. So both models we are using in this section to experiment are using sigmoid activation function. The only difference between the control model and the experimental model is that for the control model, I used SGD as the optimizer, and for th e latter, I used Adadelta.
SGD performs a parameter update for each training example. It performs one update at a time. D ue to these frequent updates, parameter updates have high variance and cause the loss function to fluctuate to different intensities and lead to unstable convergence. For this reason, I decided to ex periment with another optimizer: Adadelta. This optimizer tends to remove the learning Rate pro blem. So in order to use this optimizer, we do not need to set a default learning Rate.
The results are shown below. The left image belongs to the control model (SGD optimizer), and t he right image belongs to the experimental model (Adadelta optimizer).





As we can see, the accuracy of experimental model is much higher than the control model. And i t's considerable for both training accuracy and validation accuracy. For some values, for exampl e 60-80 range the accuracy of experimental model is even higher than 4. But for the control mod el, the highest accuracy belongs to 95-100 range which is less than 3.5. All in all, as we can see, Adadelta improved the accuracy very well.