SIDDHARTH SONI - B20AI041

Deep Learning
Programming Assignment - 1

Question 1

Introduction

Dataset: CIFAR10 (imported using pytorch torchvision dataset library).

- Converted to tensors,
- gray scaling performed,
- no normalization performed

50k examples, 32x32 size, 1 channel (Gray) : trainset

10k examples, 32x32 size, 1 channel (Gray) : testset

10 image classes

After train-validation split: train_set size: 40000 val_set size: 10000

Dataloader creation:

- batch_params: {'batch_size': 1024, 'num_workers': 0, 'pin_memory': True}
- Train loader : Shuffle = True
- Val, test loader : Shuffle = False

Activation Functions Comparison

Activation functions utilized:

1. Sigmoid

2. Tanh
3. ReLU
4. Leaky ReLU

Model architecture:

```
Sequential(
  (0): Sequential(
    (0): Linear(in_features=1024, out_features=256, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.0, inplace=False)
  )
  (1): Sequential(
    (0): Linear(in_features=256, out_features=256, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.0, inplace=False)
  )
  (2): Sequential(
    (0): Linear(in_features=256, out_features=256, bias=True)
    (1): Sigmoid()
    (2): Dropout(p=0.0, inplace=False)
  )
  (3): Sequential(
    (0): Linear(in_features=256, out_features=10, bias=True)
    (1): Softmax(dim=1)
  )
)
```

Training Params:

#epochs = 10

Optimizer = Adam

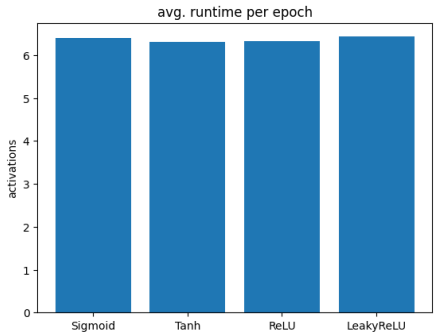
Learning rate = 1e-3

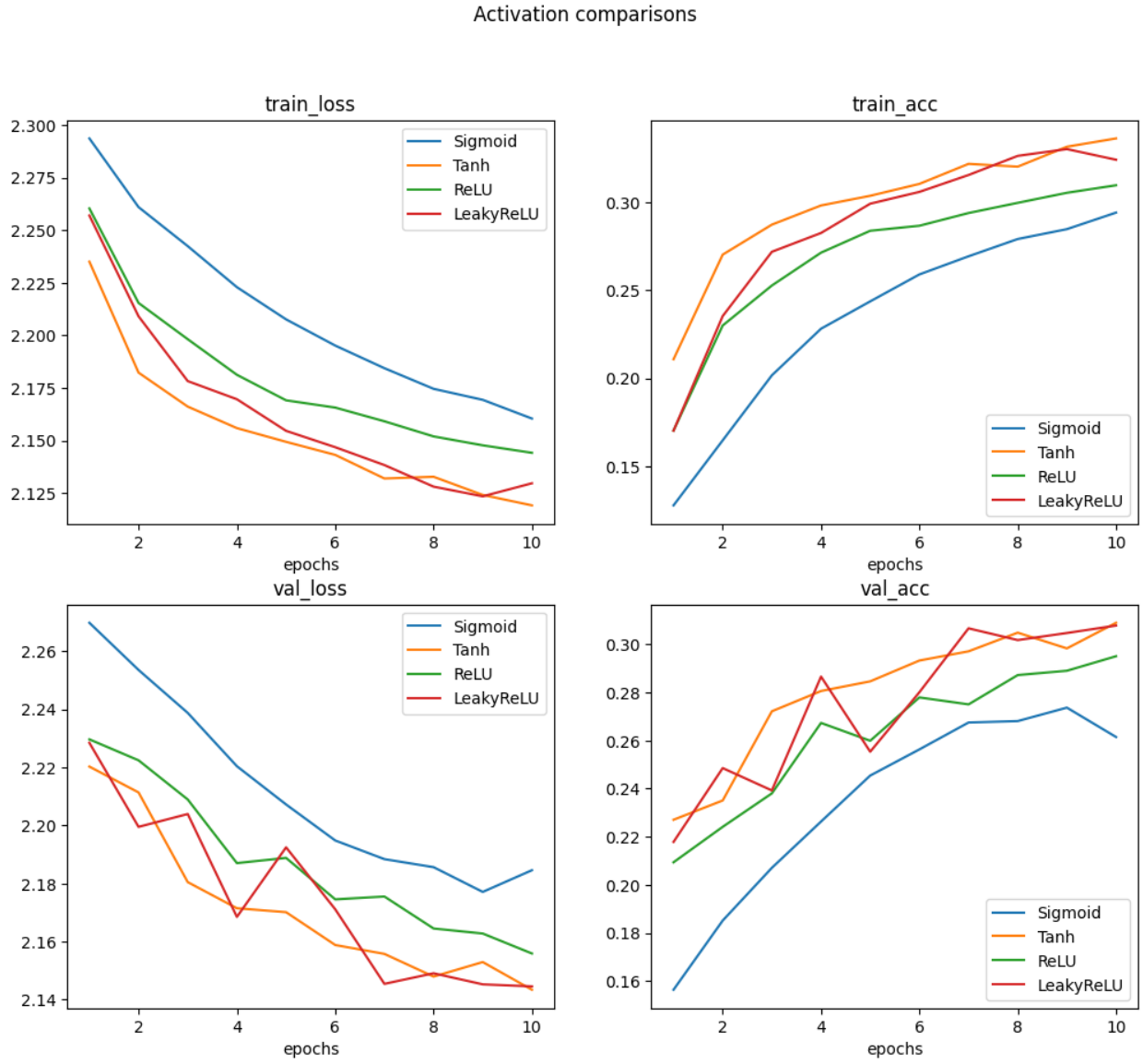
Loss function = Cross Entropy Loss

Activations : variable

hidden layers = 5

Observations





Inferences

Since the architecture is same for all gradients, therefore the weight updation only depends on the backpropagation gradients.

The Sigmoid function's gradient distribution range is smallest [0, ~0.25]. This implies that the expected value of gradient is even smaller. This means that cumulative gradient is the product of smallest values among all activations and therefore the weight updation is least efficient.

The Tanh function's gradient distribution is spread across a wider range of [0, 1] which implies that the expected value of gradient is larger. Hence, the cumulative gradient is a product of larger values as compared to sigmoid and therefore weight updation is more efficient than sigmoid.

The ReLU function has range {0, 1} => weights either update very efficiently or don't update whatsoever.

Leaky ReLU has gradients in range {0.01, 1}. Similar to ReLU, weights update better compared to sigmoid. But non-zero gradient in the negative plane => smaller learning occurs.

Runtime comparison: gradient calculation is fastest in ReLU and Leaky ReLU. Therefore, the runtimes are slightly smaller as compared to sigmoid and Tanh.

Vanishing Gradient Problem

Activation functions utilized:

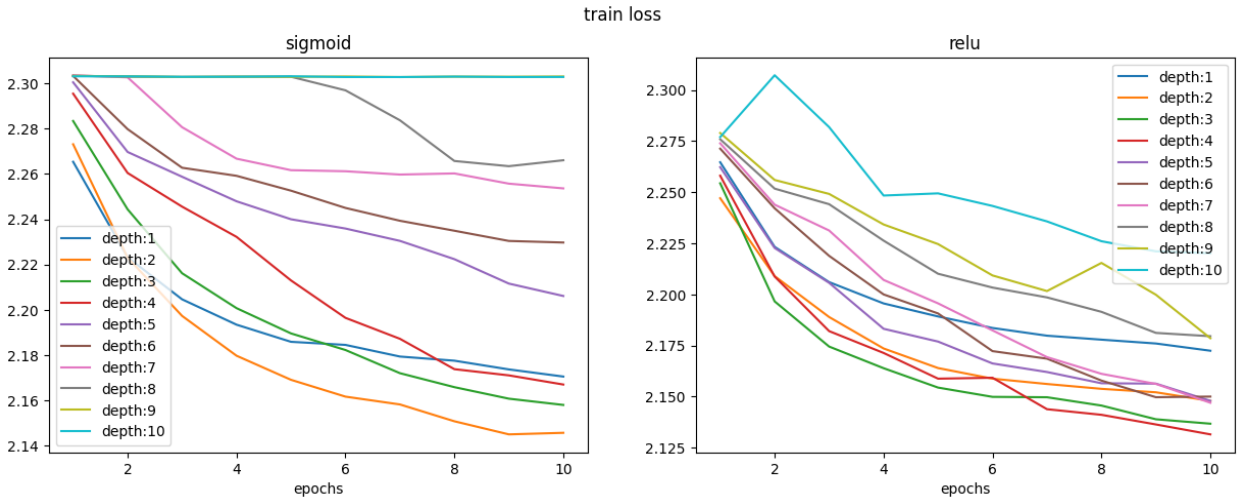
1. Sigmoid
2. ReLU

Model Architecture and training parameters:

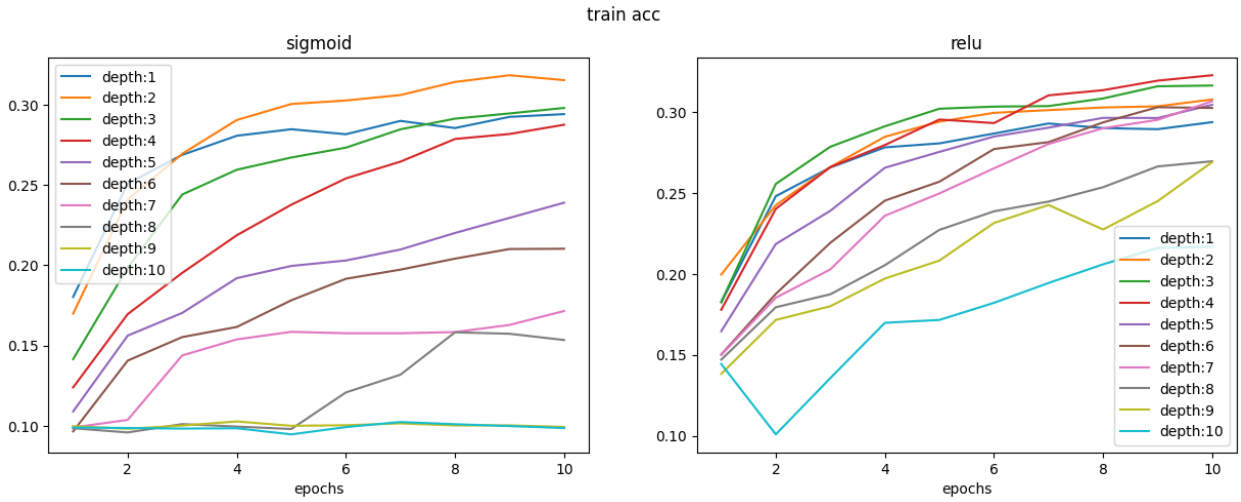
- Same as above, but the number of hidden layers are varied.

Observations

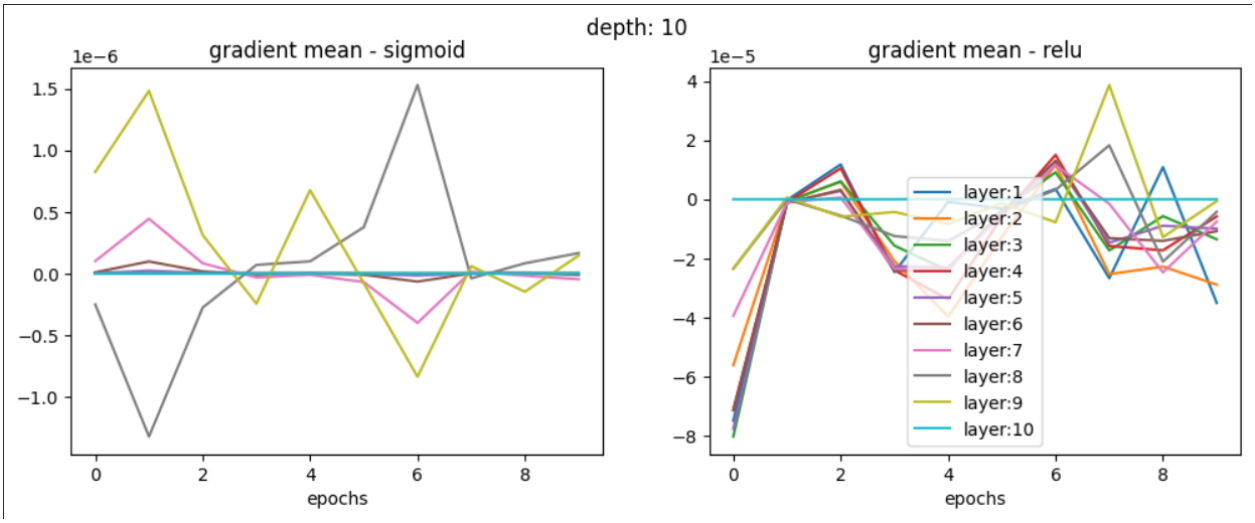
- Training loss comparison over changing depths



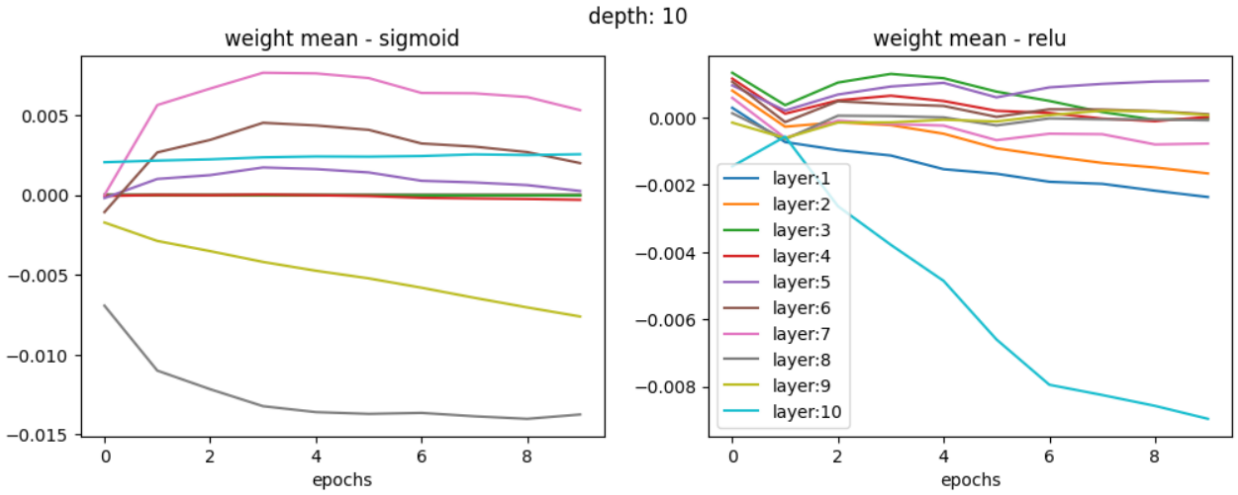
- Training accuracy comparison over changing depths



- Gradient mean comparison for layers over changing depths



- Weight mean comparison for layers over changing depths



Inferences

Since the architecture is same for all gradients, therefore the weight updation only depends on the backpropagation gradients.

Loss updation slows down as depth increases. Result for depth 1 will be the same for both activations because depth 1 => 0 HL and only 1 output layer exists => activation functions are not in use.(except the softmax activation used for classification)

For sigmoid the updation decreases exponentially to the extent that at depth 9 and 10, the loss is not even updating. Meanwhile loss updation is consistent in ReLU for all depths. This implies that since the only difference is backpropagation, sigmoid gradients vanish to 0 because loss is not updating whereas the ReLU gradient update is the same for all depth changes i.e, no gradients are vanishing.

The theory is also supported by the fact that mean gradients are visibly updated only for 3 last layers in sigmoid and is ~0 for all other layers. But for ReLU we can see that the mean of gradients keeps on updating from last to first layer significantly and does not vanish.

Question 2

Introduction

Dataset: Gurumukhi (imported using pytorch torchvision datasets ImageFolder library).

- Converted to tensors,
- gray scaling performed,
- no normalization performed

1k examples, 32x32 size, 1 channel (Gray) : trainset

178 examples, 32x32 size, 1 channel (Gray) : testset

10 image classes

After train-validation split: train_set size: 900 val_set size: 100

Dataloader creation:

- batch_params: {'batch_size': 1000, 'num_workers': 0, 'pin_memory': True}
- Train loader : Shuffle = True
- Val, test loader : Shuffle = False

Regularization Comparison

Regularization Techniques utilized:

1. L1 norm
2. L2 norm
3. Dropout

Model Architecture:

```
MLP(
  (activation): ReLU()
  (network): Sequential(
    (0): Sequential(
      (0): Linear(in_features=1024, out_features=256, bias=True)
      (1): ReLU()
      (2): Dropout(p=0.0, inplace=False)
    )
    (1): Sequential(
      (0): Linear(in_features=256, out_features=10, bias=True)
      (1): Softmax(dim=1)
    )
  )
)
```

Training Params:

#epochs = 150

Optimizer = Adam

Learning rate = 1e-3

Loss function = Cross Entropy Loss

Activations : ReLU

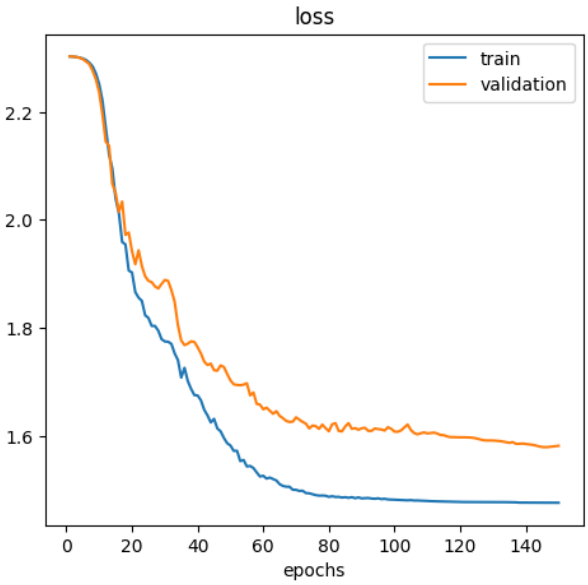
hidden layers = 6

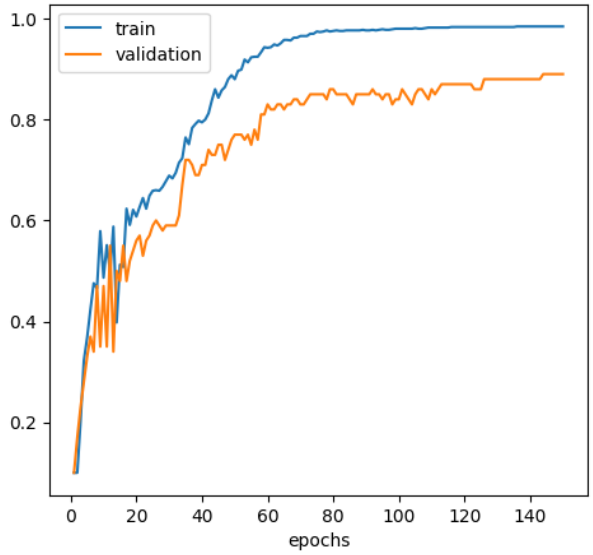
#neurons in each layer = 256

- l1_Regularization_coefficient = 1e-3
- l2Regularization_coefficient = 1
- Dropout probability = 0.2

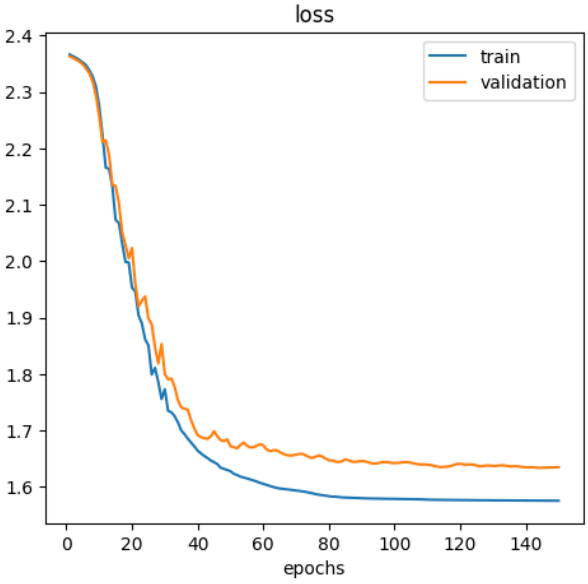
Observations

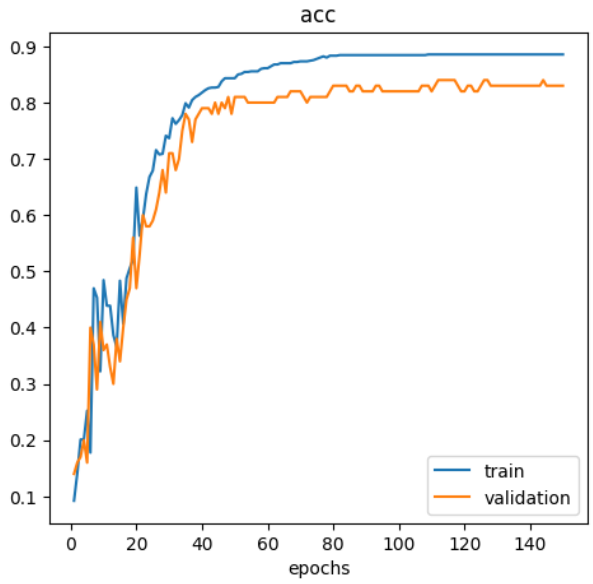
no regularization

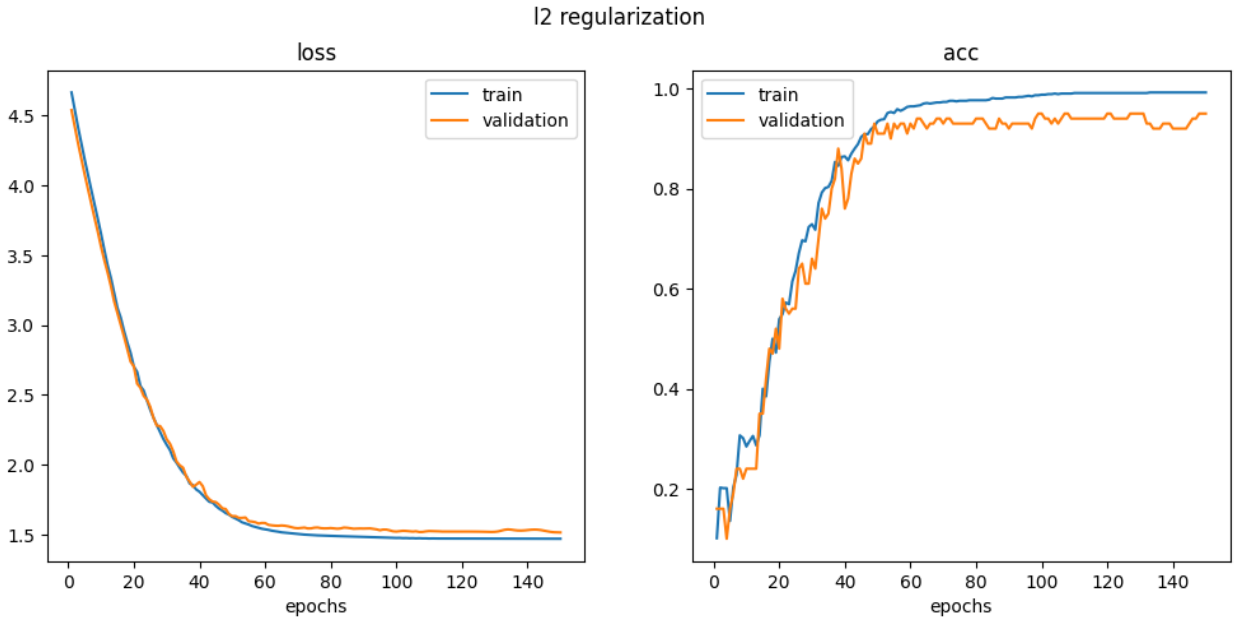




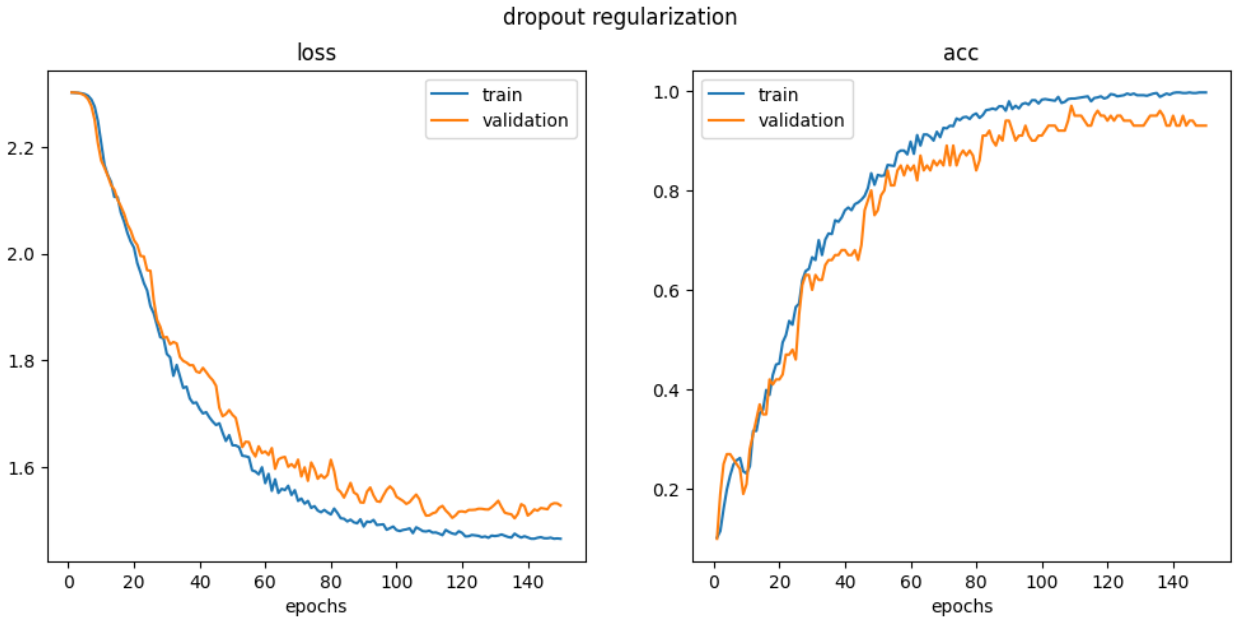
l1 regularization






l2 regularization
loss
train
validation
epochs
acc
train
validation
epochs


l2 regularization
loss
train
validation
epochs
acc
train
validation
epochs



Inferences

As we can see from the baseline graph(6HL + 1output layer), train acc. >> validation acc. => data is overfitting due to more depth than necessary.

Comparison : Dropout > l2 > l1

Dropout not only decreases the gap between train and validation, but it also increases the overall accuracy of the model.

L2 decreases the gap between train and validation to a smaller extent, but it also increases overall accuracy of the model to some extent.

L1 tries to decrease the gap of train and validation but all decrease the accuracy of the model, hence the worst among all for this model.

Since L1 works on feature selection and L2 works on codependent data and since the dataset is of numbers where feature selection is not as important understanding high dependance of pixel values that result in a number shape, therefore the result is justified.