ME491(B) Active Learning #3
Programming of Neural Network

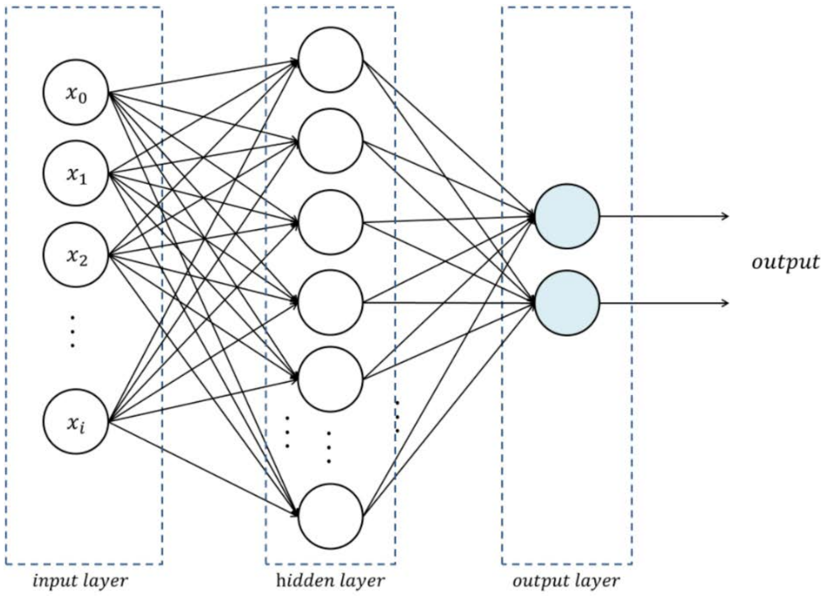
2020.09.21 (Mon)

TA : Jonghwi Kim (stkimjh@kaist.ac.kr)

Neural network

- **Multi-layer perceptron**

- The basic structure / fully connected layers



CLASS `torch.nn.Linear(in_features: int, out_features: int, bias: bool = True)` [SOURCE]

Applies a linear transformation to the incoming data: $y = xA^T + b$

Parameters

- **in_features** – size of each input sample
- **out_features** – size of each output sample
- **bias** – If set to `False`, the layer will not learn an additive bias. Default: `True`

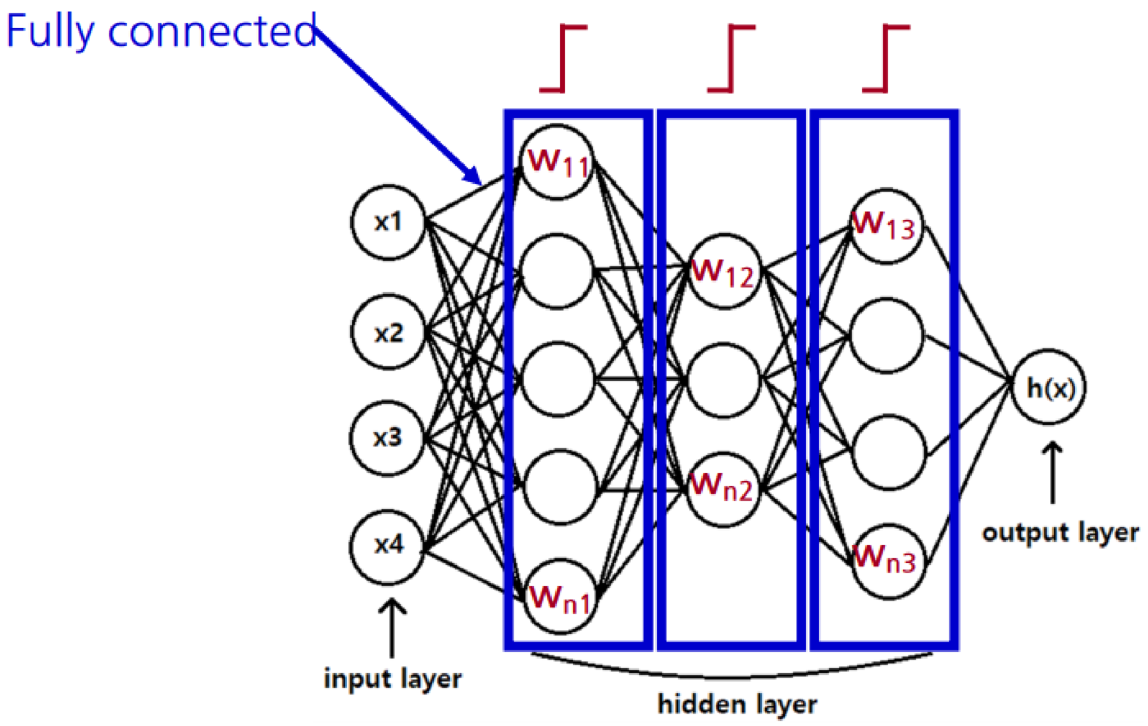
```python
class Net(torch.nn.Module):
    def __init__(self, n_feature, n_hidden, n_output):
        super(Net, self).__init__()
        # hidden layer
        self.hidden = torch.nn.Linear(n_feature, n_hidden)
        # output layer
        self.out = torch.nn.Linear(n_hidden, n_output)

    def forward(self, x):
        # activation function for hidden layer
        x = F.softmax(self.hidden(x),1)
        x = self.out(x)
        return x

net = Net(n_feature=2, n_hidden=10, n_output=2)
```

Source : https://pytorch.org/docs/stable/generated/torch.nn.Linear.html

Neural network

- **Multi-layer perceptron**

- If you want to use more layers and activation functions, try this.
- Also the network should be combined with CUDA.




```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 50)
        self.fc1_drop = nn.Dropout(0.2)
        self.fc2 = nn.Linear(50, 50)
        self.fc2_drop = nn.Dropout(0.2)
        self.fc3 = nn.Linear(50, 10)

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = F.relu(self.fc1(x))
        x = self.fc1_drop(x)
        x = F.relu(self.fc2(x))
        x = self.fc2_drop(x)
        return F.log_softmax(self.fc3(x))

model = Net()
if cuda:
    model.cuda()
```

Optimizer and loss function

- **Optimizer**

- Find the weights of the network that minimizes the loss.
- Gradient descent method-based optimization
- SGD, Adam, RMSProp…

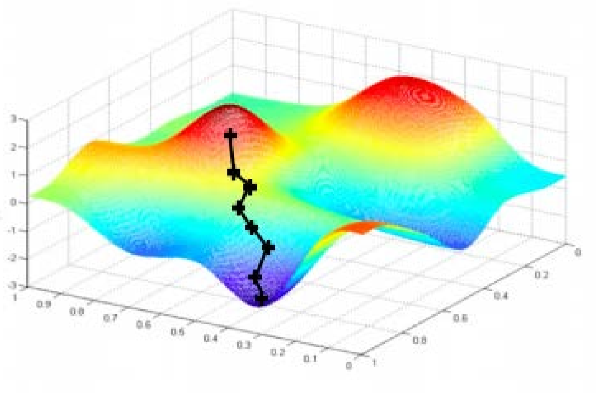
```
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

- **Loss function**

- The objective function that
- MSE loss, Cross-entropy loss, NLL Loss…

```
loss = F.nll_loss(output, target)
```

```
loss_func = torch.nn.MSELoss()
```



Main

- **Train**

```
def train(epoch, log_interval=100):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        if cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data) # Forwawrd
        loss = F.nll_loss(output, target) # Loss calculation
        loss.backward() # Back-propagation
        optimizer.step() # Apply gradient
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data[0]))
```

Main

- **Validate**

```
def validate(loss_vector, accuracy_vector):
    model.eval()
    val_loss, correct = 0, 0
    for data, target in validation_loader:
        if cuda:
            data, target = data.cuda(), target.cuda()
        data, target = Variable(data, volatile=True), Variable(target)
        output = model(data)
        val_loss += F.nll_loss(output, target).data[0]
        pred = output.data.max(1)[1] # get the index of the max log-probability
        correct += pred.eq(target.data).cpu().sum()

    val_loss /= len(validation_loader)
    loss_vector.append(val_loss)

    accuracy = 100. * correct / len(validation_loader.dataset)
    accuracy_vector.append(accuracy)

    print('\nValidation set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        val_loss, correct, len(validation_loader.dataset), accuracy))
```

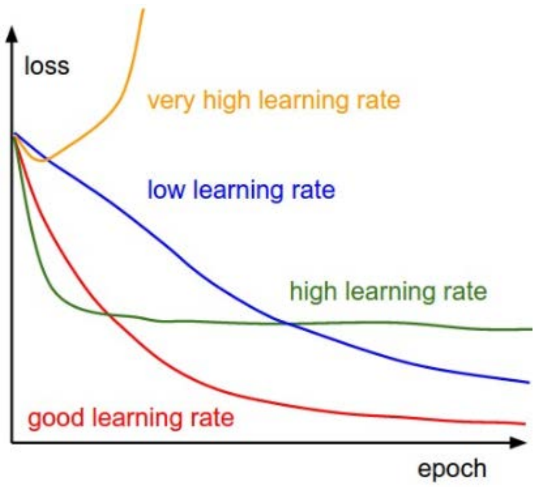
Main

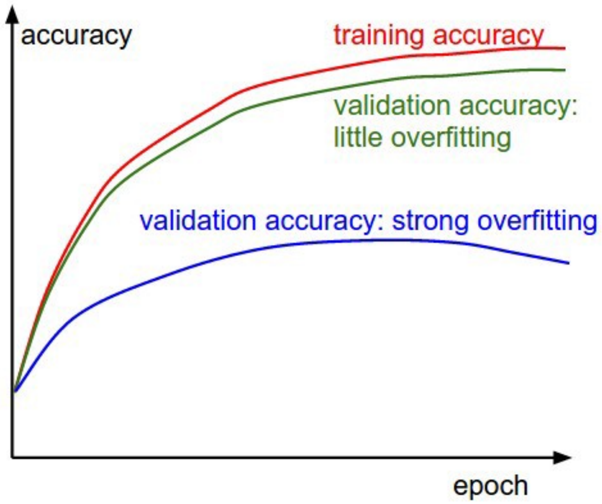
- **Main function**

```python
epochs = 10

lossv, accv = [], []
for epoch in range(1, epochs + 1):
    train(epoch)
    validate(lossv, accv)
```

Validation

- **Diagnose performance of trained neural network**

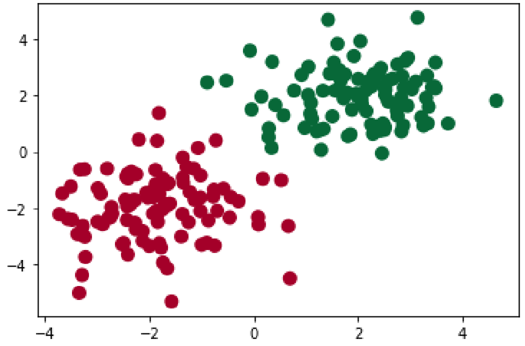




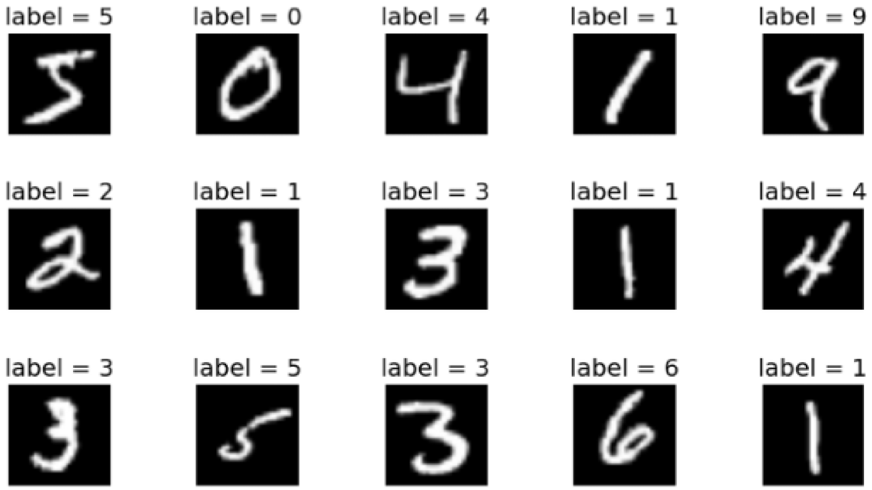
Source : Stanford Univ. CS231n Convolutional Neural Networks for Visual Recognition

Code session

1. **Classify data of 2 classes with 2 instances (x, y)**



2. **MNIST dataset (handwritten digits classification)**



Google Colab

- Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.
- With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.
- You can use the computing services for a maximum of 12 hours at a time.
- After 12 hours, a different virtual machine will be assigned.
- If you want to save trained weights you should mount your own Google Drive by following link.

https://colab.research.google.com/notebooks/io.ipynb

Check this github link!

https://github.com/Jong2/ME491_3rd_week_NN

1. Create your google account if you don't have one.
2. https://colab.research.google.com
3. Create a new Python3 notebook
4. File > Open notebook > GITHUB > find "Jong2" > repository "ME491_3rd_week_NN" > select .ipynb file
5. Copy to Drive

6. Runtime > Change runtime type > GPU