FVE3011

NN, DNN for MNIST

담당교수: 최 학남 (xncui@inha.ac.kr)

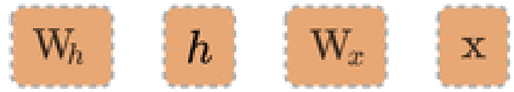
인하대학교

Contents

- Optimizers
- Neural Network for MNIST
- Deep Neural Network for MNIST
  - Xavier , drop out

Back-propagation with graph

A graph is created on the fly

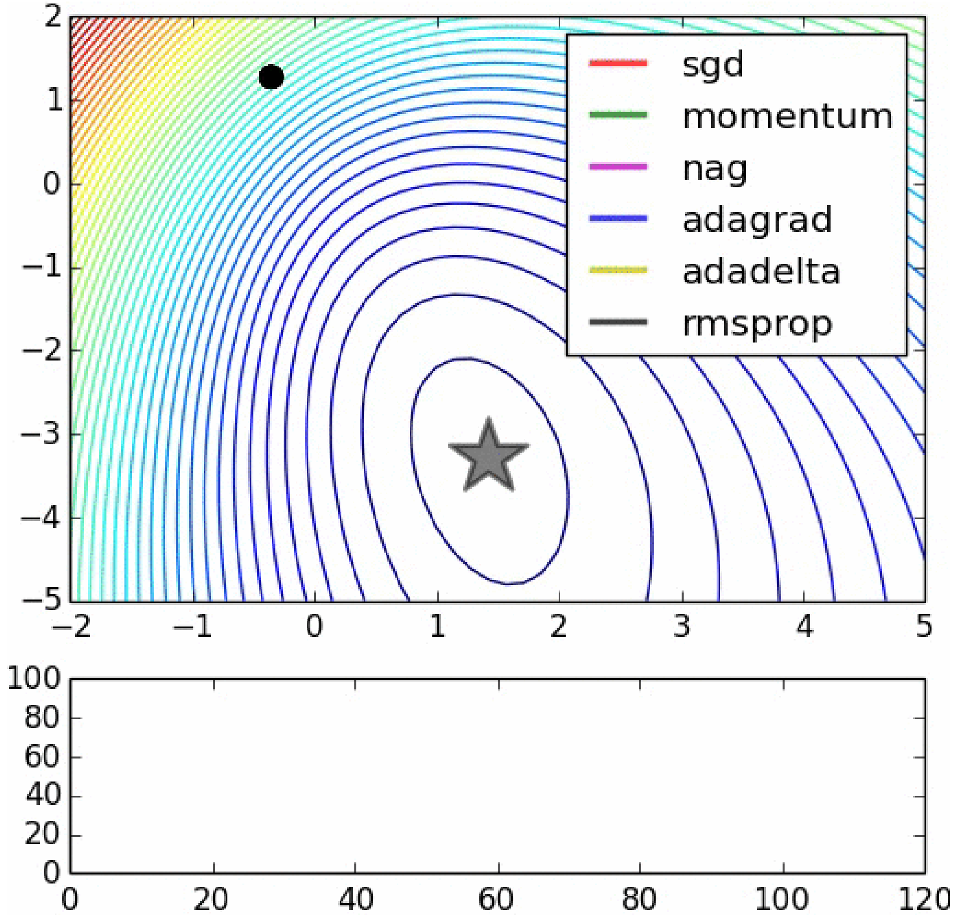



```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

Optimizers

- AdadeltaOptimizer
- AdagradOptimizer
- AdagradDAOptimizer
- MomentumOptimizer
- **AdamOptimizer**
- FtrlOptimizer
- ProximalGradientDescentOptimizer
- ProximalAdagradOptimizer
- RMSPropOptimizer

Optimizers

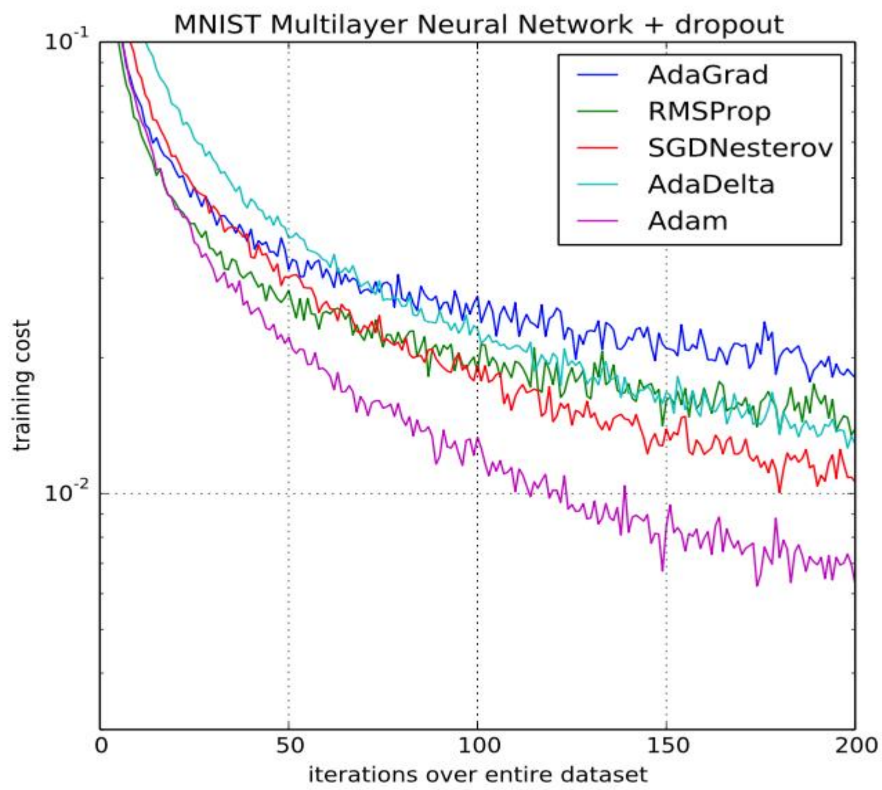


http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Optimizers

ADAM: a method for stochastic optimization

[Kingma et al. 2015]



Use Adam Optimizer

```
# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

Softmax classifier for MNIST

```
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# MNIST dataset
mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)

# dataset loader
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True)

# model
model = torch.nn.Linear(784, 10, bias=True)

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

Softmax classifier for MNIST

```
# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(mnist_train) // batch_size

    for i, (batch_xs, batch_ys) in enumerate(data_loader):
        # reshape input image into [batch_size by 784]
        X = Variable(batch_xs.view(-1, 28 * 28))
        Y = Variable(batch_ys)    # label is not one-hot encoded

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    print("[Epoch: {:>4}] cost = {:>.9}".format(epoch + 1, avg_cost.data[0]))

print('Learning Finished!')
```

```
Epoch: 0001 cost = 5.888845987
Epoch: 0002 cost = 1.860620173
Epoch: 0003 cost = 1.159035648
Epoch: 0004 cost = 0.892340870
Epoch: 0005 cost = 0.751155428
Epoch: 0006 cost = 0.662484806
Epoch: 0007 cost = 0.601544010
Epoch: 0008 cost = 0.556526115
Epoch: 0009 cost = 0.521186961
Epoch: 0010 cost = 0.493068354
Epoch: 0011 cost = 0.469686249
Epoch: 0012 cost = 0.449967254
Epoch: 0013 cost = 0.433519321
Epoch: 0014 cost = 0.419000337
Epoch: 0015 cost = 0.406490815
Learning Finished!
Accuracy: 0.9035
```

Softmax classifier for MNIST

```python
# Test model and check accuracy
X_test = Variable(mnist_test.test_data.view(-1, 28 * 28).float())
Y_test = Variable(mnist_test.test_labels)

prediction = model(X_test)
correct_prediction = (torch.max(prediction.data, 1)[1] == Y_test.data)
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy)

# Get one and predict
r = random.randint(0, len(mnist_test) - 1)
X_single_data = Variable(mnist_test.test_data[r:r + 1].view(-1, 28 * 28).float())
Y_single_data = Variable(mnist_test.test_labels[r:r + 1])

print("Label: ", Y_single_data.data)
single_prediction = model(X_single_data)
print("Prediction: ", torch.max(single_prediction.data, 1)[1])
```

NN for MNIST

```
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# MNIST dataset
mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)

# dataset loader
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True)
# nn layers
linear1 = torch.nn.Linear(784, 256, bias=True)
linear2 = torch.nn.Linear(256, 256, bias=True)
linear3 = torch.nn.Linear(256, 10, bias=True)
relu = torch.nn.ReLU()
```

NN for MNIST

```python
# model
model = torch.nn.Sequential(linear1, relu, linear2, relu, linear3)

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(mnist_train) // batch_size

    for i, (batch_xs, batch_ys) in enumerate(data_loader):
        # reshape input image into [batch_size by 784]
        X = Variable(batch_xs.view(-1, 28 * 28))
        Y = Variable(batch_ys)    # label is not one-hot encoded

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    print("[Epoch: {:>4}] cost = {:>.9}".format(epoch + 1, avg_cost.data[0]))

print('Learning Finished!')
```

NN for MNIST

```
# Test model and check accuracy
X_test = Variable(mnist_test.test_data.view(-1, 28 * 28).float())
Y_test = Variable(mnist_test.test_labels)

prediction = model(X_test)
correct_prediction = (torch.max(prediction.data, 1)[1] == Y_test.data)
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy)
```

Accuracy: tensor(0.9740)

```
# Get one and predict
r = random.randint(0, len(mnist_test) - 1)
X_single_data = Variable(mnist_test.test_data[r:r + 1].view(-1, 28 * 28).float())
Y_single_data = Variable(mnist_test.test_labels[r:r + 1])

print("Label: ", Y_single_data.data)
single_prediction = model(X_single_data)
print("Prediction: ", torch.max(single_prediction.data, 1)[1])
```

Xavier for MNIST

```
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# MNIST dataset
mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)

# dataset loader
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True)
# nn layers
linear1 = torch.nn.Linear(784, 256, bias=True)
linear2 = torch.nn.Linear(256, 256, bias=True)
linear3 = torch.nn.Linear(256, 10, bias=True)
relu = torch.nn.ReLU()

# xavier initializer
torch.nn.init.xavier_uniform(linear1.weight)
torch.nn.init.xavier_uniform(linear2.weight)
torch.nn.init.xavier_uniform(linear3.weight)
```

Xavier for MNIST

```
# model
model = torch.nn.Sequential(linear1, relu, linear2, relu, linear3)

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(mnist_train) // batch_size

    for i, (batch_xs, batch_ys) in enumerate(data_loader):
        # reshape input image into [batch_size by 784]
        X = Variable(batch_xs.view(-1, 28 * 28))
        Y = Variable(batch_ys)    # label is not one-hot encoded

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    print("[Epoch: {:>4}] cost = {:>.9}".format(epoch + 1, avg_cost.data[0]))

print('Learning Finished!')
```

Epoch: 0001 cost = 0.301498963
Epoch: 0002 cost = 0.107252513
Epoch: 0003 cost = 0.064888892
Epoch: 0004 cost = 0.044463030
Epoch: 0005 cost = 0.029951642
Epoch: 0006 cost = 0.020663404
Epoch: 0007 cost = 0.015853033
Epoch: 0008 cost = 0.011764387
Epoch: 0009 cost = 0.008598264
Epoch: 0010 cost = 0.007383116
Epoch: 0011 cost = 0.006839140
Epoch: 0012 cost = 0.004672963
Epoch: 0013 cost = 0.003979437
Epoch: 0014 cost = 0.002714260
Epoch: 0015 cost = 0.004707661
Learning Finished!
Accuracy: **0.9783**

Xavier for MNIST

```
# Test model and check accuracy
X_test = Variable(mnist_test.test_data.view(-1, 28 * 28).float())
Y_test = Variable(mnist_test.test_labels)

prediction = model(X_test)
correct_prediction = (torch.max(prediction.data, 1)[1] == Y_test.data)
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy)

# Get one and predict
r = random.randint(0, len(mnist_test) - 1)
X_single_data = Variable(mnist_test.test_data[r:r + 1].view(-1, 28 * 28).float())
Y_single_data = Variable(mnist_test.test_labels[r:r + 1])

print("Label: ", Y_single_data.data)
single_prediction = model(X_single_data)
print("Prediction: ", torch.max(single_prediction.data, 1)[1])
```

Xavier for MNIST

Epoch: 0001 cost = 141.207671860
Epoch: 0002 cost = 38.788445864
Epoch: 0003 cost = 23.977515479
Epoch: 0004 cost = 16.315132428
Epoch: 0005 cost = 11.702554882
Epoch: 0006 cost = 8.573139748
Epoch: 0007 cost = 6.370995680
Epoch: 0008 cost = 4.537178684
Epoch: 0009 cost = 3.216900532
Epoch: 0010 cost = 2.329708954
Epoch: 0011 cost = 1.715552875
Epoch: 0012 cost = 1.189857912
Epoch: 0013 cost = 0.820965160
Epoch: 0014 cost = 0.624131458
Epoch: 0015 cost = 0.454633765
Learning Finished!
Accuracy: **0.9455 (normal dist)**

Epoch: 0001 cost = 0.301498963
Epoch: 0002 cost = 0.107252513
Epoch: 0003 cost = 0.064888892
Epoch: 0004 cost = 0.044463030
Epoch: 0005 cost = 0.029951642
Epoch: 0006 cost = 0.020663404
Epoch: 0007 cost = 0.015853033
Epoch: 0008 cost = 0.011764387
Epoch: 0009 cost = 0.008598264
Epoch: 0010 cost = 0.007383116
Epoch: 0011 cost = 0.006839140
Epoch: 0012 cost = 0.004672963
Epoch: 0013 cost = 0.003979437
Epoch: 0014 cost = 0.002714260
Epoch: 0015 cost = 0.004707661
Learning Finished!
Accuracy: **0.9783 (xavier)**

Deep NN for MNIST

```python
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# MNIST dataset
mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)

# dataset loader
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True)
# nn layers
linear1 = torch.nn.Linear(784, 512, bias=True)
linear2 = torch.nn.Linear(512, 512, bias=True)
linear3 = torch.nn.Linear(512, 512, bias=True)
linear4 = torch.nn.Linear(512, 512, bias=True)
linear5 = torch.nn.Linear(512, 10, bias=True)
relu = torch.nn.ReLU()

# xavier initializer
torch.nn.init.xavier_uniform(linear1.weight)
torch.nn.init.xavier_uniform(linear2.weight)
torch.nn.init.xavier_uniform(linear3.weight)
torch.nn.init.xavier_uniform(linear4.weight)
torch.nn.init.xavier_uniform(linear5.weight)
```

Deep NN for MNIST

```python
# model
model = torch.nn.Sequential(linear1, relu,
                            linear2, relu,
                            linear3, relu,
                            linear4, relu,
                            linear5)

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(mnist_train) // batch_size

    for i, (batch_xs, batch_ys) in enumerate(data_loader):
        # reshape input image into [batch_size by 784]
        X = Variable(batch_xs.view(-1, 28 * 28))
        Y = Variable(batch_ys)    # label is not one-hot encoded

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    print("[Epoch: {:>4}] cost = {:>.9}".format(epoch + 1, avg_cost.data[0]))

print('Learning Finished!')
```

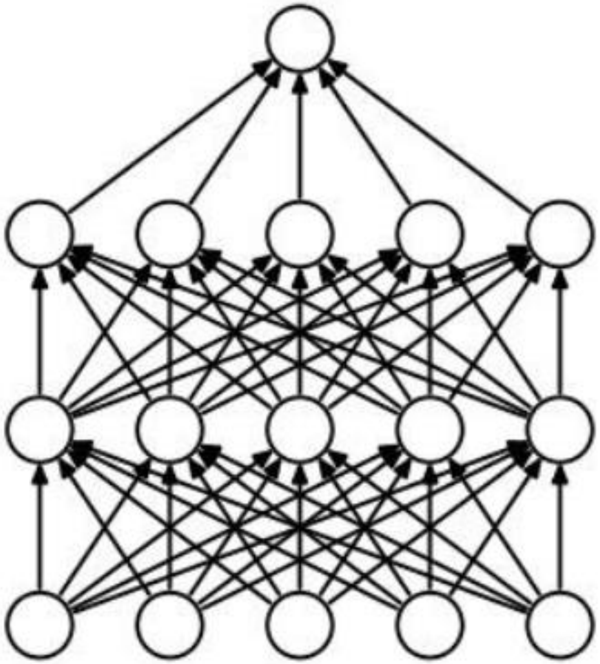
Epoch: 0001 cost = 0.266061549
Epoch: 0002 cost = 0.080796588
Epoch: 0003 cost = 0.049075800
Epoch: 0004 cost = 0.034772298
Epoch: 0005 cost = 0.024780529
Epoch: 0006 cost = 0.017072763
Epoch: 0007 cost = 0.014031383
Epoch: 0008 cost = 0.013763446
Epoch: 0009 cost = 0.009164047
Epoch: 0010 cost = 0.008291388
Epoch: 0011 cost = 0.007319742
Epoch: 0012 cost = 0.006434021
Epoch: 0013 cost = 0.005684378
Epoch: 0014 cost = 0.004781207
Epoch: 0015 cost = 0.004342310
Learning Finished!
Accuracy: **0.9742**

Deep NN for MNIST

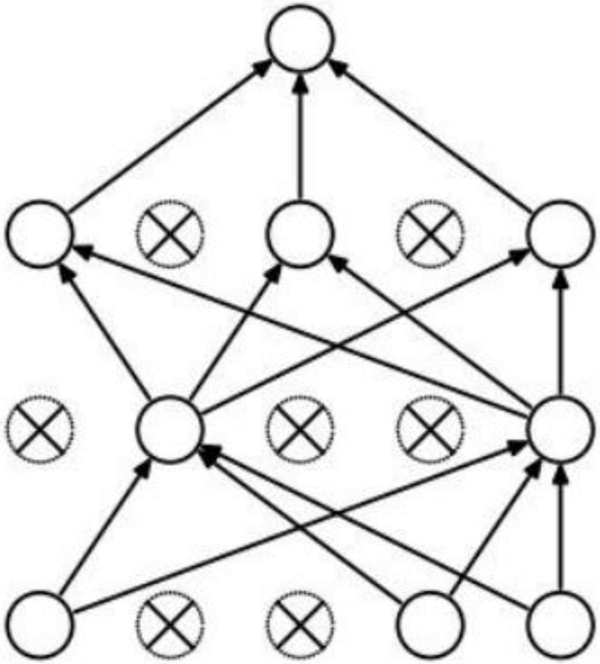
```
# Test model and check accuracy
X_test = Variable(mnist_test.test_data.view(-1, 28 * 28).float())
Y_test = Variable(mnist_test.test_labels)

prediction = model(X_test)
correct_prediction = (torch.max(prediction.data, 1)[1] == Y_test.data)
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy)

# Get one and predict
r = random.randint(0, len(mnist_test) - 1)
X_single_data = Variable(mnist_test.test_data[r:r + 1].view(-1, 28 * 28).float())
Y_single_data = Variable(mnist_test.test_labels[r:r + 1])

print("Label: ", Y_single_data.data)
single_prediction = model(X_single_data)
print("Prediction: ", torch.max(single_prediction.data, 1)[1])
```

Regularization: Dropout

“Randomly set some neurons to zero in the forward pass”



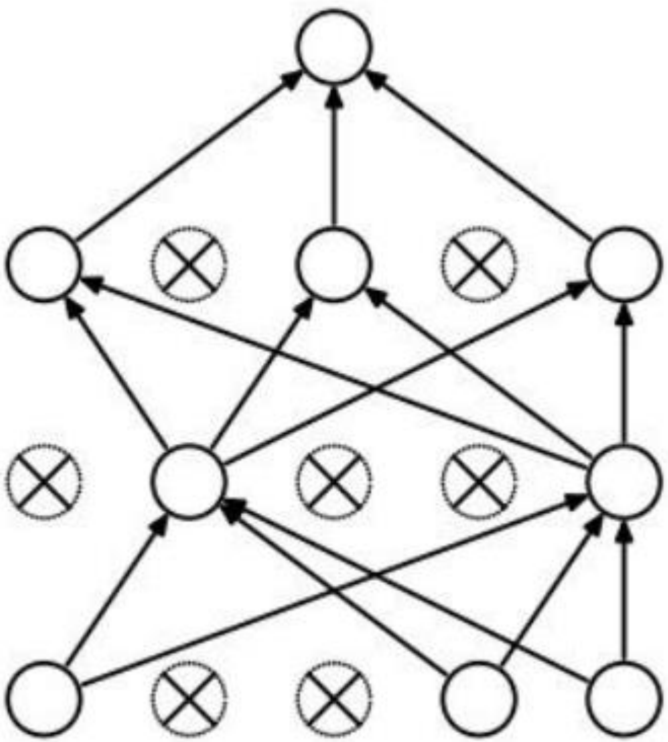
(a) Standard Neural Net



(b) After applying dropout.

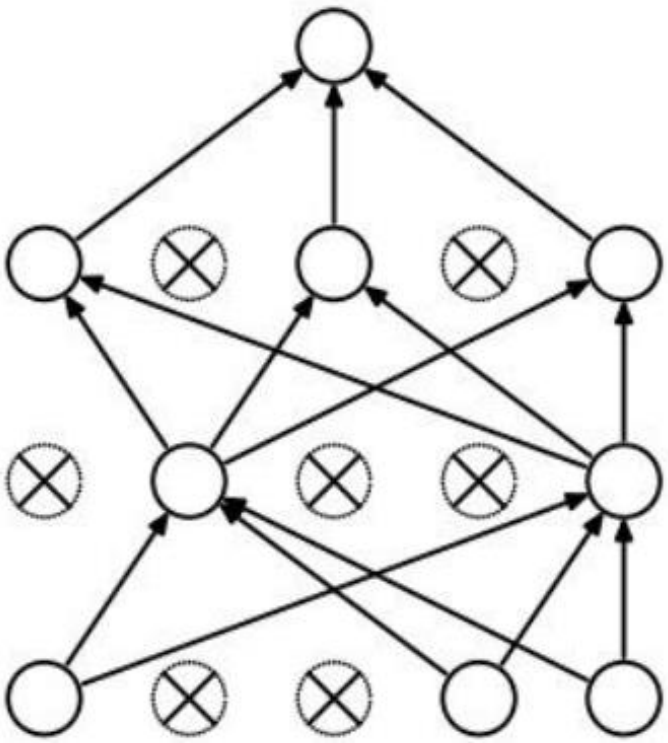
Regularization: Dropout

Question: how could this possible be a good idea?



Regularization: Dropout

Question: how could this possible be a good idea?



- Forces the network to have a **redundant representation**

Deep NN with Dropout for MNIST

```python
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
keep_prob = 0.7

# MNIST dataset
mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)

# dataset loader
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True)
```

```python
# nn layers
linear1 = torch.nn.Linear(784, 512, bias=True)
linear2 = torch.nn.Linear(512, 512, bias=True)
linear3 = torch.nn.Linear(512, 512, bias=True)
linear4 = torch.nn.Linear(512, 512, bias=True)
linear5 = torch.nn.Linear(512, 10, bias=True)

relu = torch.nn.ReLU()
# p is the probability of being dropped in PyTorch
dropout = torch.nn.Dropout(p=1 - keep_prob)

# xavier initializer
torch.nn.init.xavier_uniform(linear1.weight)
torch.nn.init.xavier_uniform(linear2.weight)
torch.nn.init.xavier_uniform(linear3.weight)
torch.nn.init.xavier_uniform(linear4.weight)
torch.nn.init.xavier_uniform(linear5.weight)
```

Deep NN with Dropout for MNIST

```python
# model
model = torch.nn.Sequential(linear1, relu, dropout,
                            linear2, relu, dropout,
                            linear3, relu, dropout,
                            linear4, relu, dropout,
                            linear5)

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss()    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(mnist_train) // batch_size

    for i, (batch_xs, batch_ys) in enumerate(data_loader):
        # reshape input image into [batch_size by 784]
        X = Variable(batch_xs.view(-1, 28 * 28))
        Y = Variable(batch_ys)    # label is not one-hot encoded

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    print("[Epoch: {:>4}] cost = {:>.9}".format(epoch + 1, avg_cost.data[0]))

print('Learning Finished!')
```

```python
# Test model and check accuracy
model.eval()    # set the model to evaluation mode (dropout=False)

X_test = Variable(mnist_test.test_data.view(-1, 28 * 28).float())
Y_test = Variable(mnist_test.test_labels)

prediction = model(X_test)
correct_prediction = (torch.max(prediction.data, 1)[1] == Y_test.data)
accuracy = correct_prediction.float().mean()
print('Accuracy:', accuracy)

# Get one and predict
r = random.randint(0, len(mnist_test) - 1)
X_single_data = Variable(mnist_test.test_data[r:r + 1].view(-1, 28 * 28).float())
Y_single_data = Variable(mnist_test.test_labels[r:r + 1])

print("Label: ", Y_single_data.data)
single_prediction = model(X_single_data)
print("Prediction: ", torch.max(single_prediction.data, 1)[1])
```

Epoch: 0013 cost = 0.047231930
Epoch: 0014 cost = 0.041290121
Epoch: 0015 cost = 0.043621063
Learning Finished!
Accuracy: 0.9804!!

Summary

- Softmax VS Neural Nets for MNIST, 90% and 94.5%
- Xavier initialization: 97.8%
- Deep Neural Nets with Dropout: 98%
- Adam and other optimizers