Neural Network Training session

Afshin Sadeghi
Firas Kassawat

Based on The KG lab SS 2019

Table of Content

PyTorch Installation

Operating System: Linux
Install miniconda from anaconda website:

https://docs.conda.io/en/latest/miniconda.html

Miniconda

	Windows	Mac OS X	Linux
Python 3.7	64-bit (exe installer) 32-bit (exe installer)	64-bit (bash installer) 64-bit (.pkg installer)	64-bit (bash installer) 32-bit (bash installer)
Python 2.7	64-bit (exe installer) 32-bit (exe installer)	64-bit (bash installer) 64-bit (.pkg installer)	64-bit (bash installer) 32-bit (bash installer)

Installation instructions

PyTorch Installation

Install pytorch using anaconda

PyTorch Build	Stable (1.0)	Preview (Nightly)			
Your OS	Linux	Mac	Windows		
Package	Conda	Pip	LibTorch	Source	
Language	Python 2.7	Python 3.5	Python 3.6	Python 3.7	C++
CUDA	8.0	9.0	10.0	None	
Run this Command:	conda install pytorch-cpu torchvision-cpu -c pytorch				

PyTorch Tensors and Operations on Tensors

A `torch.Tensor` is a multi-dimensional matrix containing elements of a single data type.
In the tensor definition we can say it resides in CPU or GPU.
It can be defined from a Python list or from a numpy array. (default is torch.FloatTensor or `float32`)

```
>>> torch.tensor([[1., -1.], [1., -1.]])
tensor([[ 1.0000, -1.0000],
        [ 1.0000, -1.0000]])
>>> torch.tensor(np.array([[1, 2, 3], [4, 5, 6]]))
tensor([[ 1,  2,  3],
        [ 4,  5,  6]])
```

PyTorch Tensors and Operations on Tensors

Defining the type of tensor:

```
>>> tensor = torch.ones((2,), dtype=torch.float64)
>>> tensor.new_full((3, 4), 3.141592)
tensor([[ 3.1416,  3.1416,  3.1416,  3.1416],
        [ 3.1416,  3.1416,  3.1416,  3.1416],
        [ 3.1416,  3.1416,  3.1416,  3.1416]], dtype=torch.float64)
```

PyTorch Tensors and Operations on Tensors

- Indexing and slicing notation:

```
>>> x = torch.tensor([[1, 2, 3], [4, 5, 6]])
>>> print(x[1][2])
tensor(6)
```

- For a tensor containing a single value: item()

```
>>> x = torch.tensor([[1]])
>>> x
tensor([[ 1]])
>>> x.item()
1
```

PyTorch Tensors and Operations on Tensors

- Creation can be by making a tensor of ones or zeroes:

```
>>> tensor = torch.tensor((), dtype=torch.float64)
>>> tensor.new_zeros((2, 3))
tensor([[ 0.,  0.,  0.],
        [ 0.,  0.,  0.]], dtype=torch.float64)
```

- Moving tesor to CPU or a core in GPU:

```
>>> cuda0 = torch.device('cuda:0')
>>> torch.ones([2, 4], dtype=torch.float64, device=cuda0)
tensor([[ 1.0000,  1.0000,  1.0000,  1.0000],
        [ 1.0000,  1.0000,  1.0000,  1.0000]], dtype=torch.float64, device='cuda:0')
```

tensor.to('cpu') (tensor here is the variable name)

PyTorch: Automatic Differentiation of Tensors

https://pytorch.org/docs/stable/notes/autograd.html

- By adding .autograd to tensor definition
- Variable()
- nn.Parameter()

$$Tanh(W_h * h^T + W_z * x^T)$$

$$\begin{cases} i2h = w_x.x^T \\ h2h = w_h.h^T \\ next_h = i2h + h2h \\ next_h = tanh(next_h) \end{cases}$$

A graph is created on the fly

W_h h W_x x

```
from torch.autograd import Variable

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

https://blog.algorithmia.com/exploring-the-deep-learning-framework-pytorch/

Practice in class

20 minutes:

Practice what you have learnt!

Use the Pytorch Tutorial website and do the followings:

Installation, tensors, operation on tensors, auto-diff, using cpu/gpu.

https://bit.ly/2xABi3f

https://bit.ly/2UuEoRk

PyTorch: Implementing a linear regression

torch.nn

`torch.nn.Module` :Base class for all neural network modules. For example:

torch.nn.`Conv2d to define a convolutional layer`

torch.nn.`Linear to define a linear regression layer`

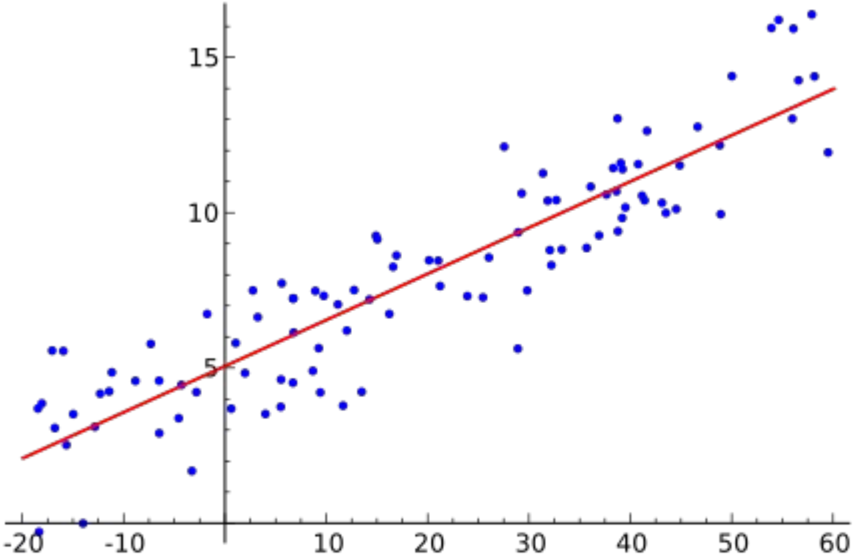
torch.nn.`Embedding to define` lookup table that stores embeddings of a fixed dictionary and size.

Learning material online: https://pytorch.org/docs/stable/nn.html?highlight=torch%20nn#module-torch.nn

`torch.nn.Parameter` : A module that contains Tensor and is associated with a module class. When a Parameter is associated with a module as a model attribute, it is added to the parameter list automatically and can be accessed using the 'parameters' iterator of pytorch.

PyTorch: Implementing a linear regression

- Creating and training linear regression model in PyTorch:
-
- Input: A set of N data samples in the form of (x,y)
-
    - x: input feature,
    - y: label/target.
    -
- Output: Trained linear model (Wx + b) to predict labels for unseen data
- Evaluation: Mean Square Error



https://hackernoon.com/linear-regression-in-x-minutes-using-pytorch-8eec49f6a0e2

PyTorch: Defining a Network and learning the associated Classes

- **Creating Models in PyTorch:**
  - Create a Class (LinearRegressionModel)
  - Declare your Forward Pass
  - Set the HyperParameters (input_dim, output_dim)

Nn.linear (Ax+b)

```python
class LinearRegressionModel(nn.Module):

    def __init__(self, input_dim, output_dim):

        super(LinearRegressionModel, self).__init__()
        # Calling Super Class's constructor
        self.linear = nn.Linear(input_dim, output_dim)
        # nn.linear is defined in nn.Module

    def forward(self, x):
        # Here the forward pass is simply a linear function

        out = self.linear(x)
        return out

input_dim = 1
output_dim = 1
```

https://hackernoon.com/linear-regression-in-x-minutes-using-pytorch-8eec49f6a0e2

PyTorch: Defining a Network and learning the associated Classes

Steps

1. Create instance of model
2. Select Loss Criterion
3. Choose Hyper Parameters

```
model = LinearRegressionModel(input_dim,output_dim)

criterion = nn.MSELoss()# Mean Squared Loss
l_rate = 0.01
optimiser = torch.optim.SGD(model.parameters(), lr = l_rate)
#Stochastic Gradient Descent

epochs = 2000
```

https://hackernoon.com/linear-regression-in-x-minutes-using-pytorch-8eec49f6a0e2

PyTorch: Defining a Network and learning the associated Classes

Training the Model in batch:

Zero_grad : clear grads

Forward pass (making prediction)

Criterion of optimizer(calculate error)

Back propagation(update weights)

Update step

```
for epoch in range(epochs):

    epoch +=1
    #increase the number of epochs by 1 every time

    inputs = Variable(torch.from_numpy(x_train))
    labels = Variable(torch.from_numpy(y_correct))

    #clear grads as discussed in prev post

    optimiser.zero_grad()

    #forward to get predicted values

    outputs = model.forward(inputs)
    loss = criterion(outputs, labels)
    loss.backward()# back props
    optimiser.step()# update the parameters
    print('epoch {}, loss {}'.format(epoch,loss.data[0]))
```

https://hackernoon.com/linear-regression-in-x-minutes-using-pytorch-8eec49f6a0e2

PyTorch: Defining a Network and learning the associated Classes

Finally, Print the Predicted Values

```
predicted
=model.forward(Variable(torch.from_numpy(x_train))).data.numpy()

plt.plot(x_train, y_correct, 'go', label = 'from data', alpha = .5)
plt.plot(x_train, predicted, label = 'prediction', alpha = 0.5)
plt.legend()
plt.show()
print(model.state_dict())
```

https://hackernoon.com/linear-regression-in-x-minutes-using-pytorch-8eec49f6a0e2

Practice in class

Students do LR in class

Questions?

Next Session?

LSTM, Relu, Concatenation and multiplication

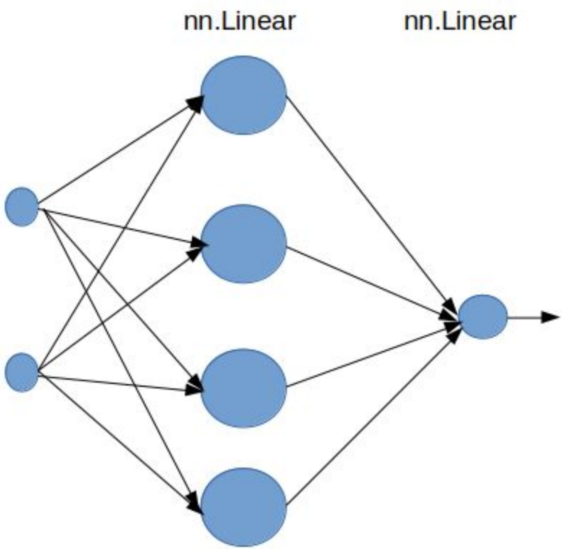
Attention scores

PyTorch: Exercise for home 1

Create Neural Networks with (one, two and three) hidden layers and one output layer:

- Each node is a linear function(Wx+b)
- nn.Linear can be a single node or a layer
- Hidden layer activation function
  - Tanh, sin, sigmoid, etc.
- Output node activation function
  - Linear function
- Dataset:
  - Experiment on 10 UCI regression dataset:
  - https://archive.ics.uci.edu/ml/datasets.html
- Loss function: MSE loss, Cross entropy loss
- Evaluation: Mean Square Error (Do experiments 20 times and report mean and variance of MSE)

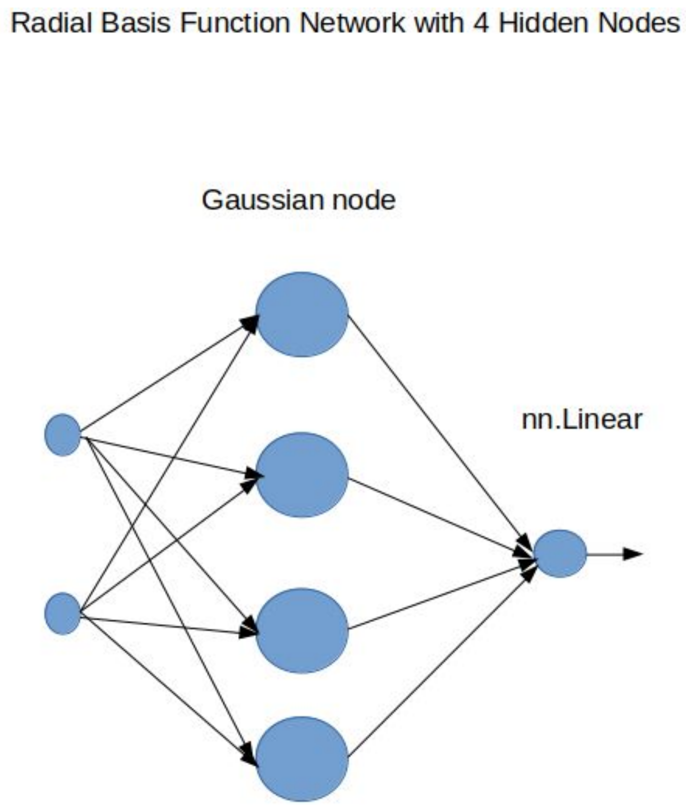
Single Hidden Layer Neural Network with 4 Hidden Nodes



PyTorch: Exercise for home 2

Create Radial Basis Function (RBF) Neural Network with one hidden layer and an output layer:

- Each node is a RBF node
- g(x) = exp(-b||x - m||)
- Output node activation function
  - Linear function
- Dataset:
  - Experiment on 10 UCI regression dataset:
  - https://archive.ics.uci.edu/ml/datasets.html
- Loss function: MSE loss, Cross entropy loss
- Evaluation: Mean Square Error (Do experiments 20 times and report mean and variance of MSE)



References:

https://docs.conda.io/en/latest/miniconda.html
https://pytorch.org/docs/stable/tensors.html#torch.Tensor
https://pytorch.org
https://hackernoon.com/linear-regression-in-x-minutes-using-pytorch-8eec49f6a0e2
https://blog.algorithmia.com/exploring-the-deep-learning-framework-pytorch/