Watson Studio democratizes machine learning and deep learning to accelerate infusion of AI in your business to drive innovation. Watson Studio provides a suite of tools and a collaborative environment for data scientists, developers and domain experts.

(http://cocl.us/pytorch_link_top)

# Linear regression: Checkpoints

## Table of Contents

Model checkpoints are used for resuming training,but you must save more than just the model's `state_dict` . It is important to also save the optimizer's `state_dict` . In this lab we will save multiple components, organize them in a dictionary and use `torch.save()` to serialize the dictionary. We will then load the model resume training.


- Make Some Data
- Create a Linear Regression Object, Data Loader and Criterion Function
- Train the Model and Save Checkpoints
- Resume training model with Checkpoints


Estimated Time Needed: **15 min**

---

## Preparation

We'll need the following libraries, and set the random seed.

In [1]:

```
# Import the libraries and set random seed

from torch import nn
import torch
import numpy as np
import matplotlib.pyplot as plt
from torch import nn,optim
from torch.utils.data import Dataset, DataLoader

torch.manual_seed(1)
```

Out[1]:

```
<torch._C.Generator at 0x1af8520a290>
```

## Make Some Data

First let's create some artificial data, in a dataset class.

In [2]:

```
# Create Data Class

class Data(Dataset):

    # Constructor
    def __init__(self, train = True):
        if train == True:
            self.x = torch.arange(-3, 3, 0.1).view(-1, 1)
            self.f = -3 * self.x + 1
            self.y = self.f + 0.1 * torch.randn(self.x.size())
            self.len = self.x.shape[0]


    # Getter
    def __getitem__(self, index):
        return self.x[index], self.y[index]

    # Get Length
    def __len__(self):
        return self.len
```

We create a training data object:

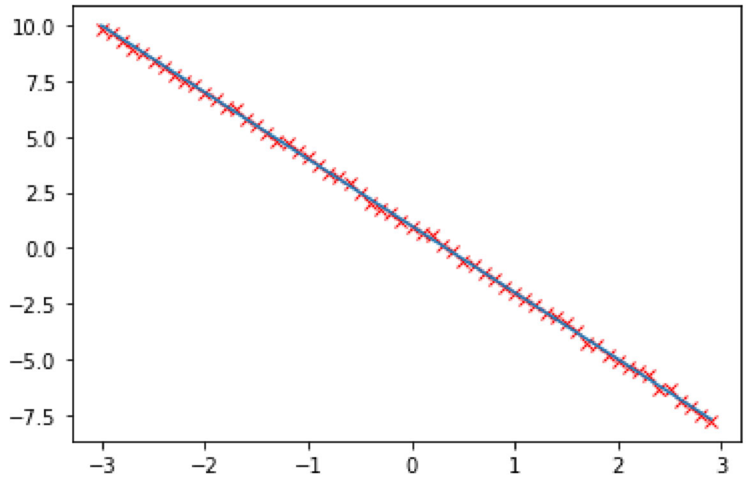
In [3]:

```
#Create train_data object and val_data object
train_data = Data()
```

We overlay the training points in red over the function that generated the data.

In [4]:

```
# Plot the training data points
plt.plot(train_data.x.numpy(), train_data.y.numpy(), 'xr')
plt.plot(train_data.x.numpy(), train_data.f.numpy())
plt.show()
```




# Create a Linear Regression Class, Object, Data Loader, Criterion Function

Create linear regression model class.

In [5]:

```
# Create linear regression model class

from torch import nn

class linear_regression(nn.Module):

    # Constructor
    def __init__(self, input_size, output_size):
        super(linear_regression, self).__init__()
        self.linear = nn.Linear(input_size, output_size)

    # Predition
    def forward(self, x):
        yhat = self.linear(x)
        return yhat
```

Create the model object

In [6]:

```
# Create the model object

model = linear_regression(1, 1)
```

We create the optimizer, the criterion function and a Data Loader object.

In [7]:

```
# Create optimizer, cost function and data loader object

optimizer = optim.SGD(model.parameters(), lr = 0.01)
criterion = nn.MSELoss()
trainloader = DataLoader(dataset = train_data, batch_size = 1)
```

## Train the Model and Save Checkpoints

path to checkpoint and file name

In [8]:

```
checkpoint_path='checkpoint_model.pt'
```

checkpoint dictionary

In [9]:

```
checkpoint={'epoch':None,'model_state_dict':None ,'optimizer_state_dict':None ,'loss': None}
```

Train for three epochs, save checkpoint information. The epoch, model state dictionary, optimizer state dictionary and loss are stored in a python dictionary.

In [10]:

```
epochs=4
LOSS_TRAIN = []
for epoch in range(epochs):
    for x, y in trainloader:
        yhat = model(x)
        loss = criterion(yhat, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_train = criterion(model(train_data.x), train_data.y).item()

        LOSS_TRAIN.append(loss_train)

    checkpoint['epoch']=epoch
    checkpoint['model_state_dict']=model.state_dict()
    checkpoint['optimizer_state_dict']= optimizer.state_dict()
    checkpoint['loss']=loss
    torch.save(checkpoint, checkpoint_path)
```

# Resume training model with Checkpoints

We can load the checkpoint dictionary, using `torch load`

In [11]:

```
checkpoint = torch.load(checkpoint_path)
checkpoint
```

Out[11]:

```
{'epoch': 3,
 'model_state_dict': OrderedDict([('linear.weight', tensor([[-3.0072]])),
              ('linear.bias', tensor([0.9848]))]),
 'optimizer_state_dict': {'state': {},
  'param_groups': [{'lr': 0.01,
    'momentum': 0,
    'dampening': 0,
    'weight_decay': 0,
    'nesterov': False,
    'params': [0, 1]}]},
 'loss': tensor(0.0034, requires_grad=True)}
```

We create a new model with arbitrary model parameter values :

In [12]:

```
model_checkpoint = linear_regression(1,1)
model_checkpoint.state_dict()
```

Out[12]:

```
OrderedDict([('linear.weight', tensor([[-0.8890]])),
             ('linear.bias', tensor([0.7278]))])
```

We load the state dictionary from the checkpoint dictionary into the model .

In [13]:

```
model_checkpoint.load_state_dict(checkpoint['model_state_dict'])
model_checkpoint.state_dict()
```

Out[13]:

```
OrderedDict([('linear.weight', tensor([[-3.0072]])),
             ('linear.bias', tensor([0.9848]))])
```

we create an arbitrary optimizer object

In [14]:

```
optimizer = optim.SGD(model_checkpoint.parameters(), lr = 1)
optimizer.state_dict()
```

Out[14]:

```
{'state': {},
 'param_groups': [{'lr': 1,
   'momentum': 0,
   'dampening': 0,
   'weight_decay': 0,
   'nesterov': False,
   'params': [0, 1]}]}
```

we can update the optimizer object using the optimizer state dictionary from the checkpoints:

In [15]:

```
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
optimizer.state_dict()
```

Out[15]:

```
{'state': {},
 'param_groups': [{'lr': 0.01,
   'momentum': 0,
   'dampening': 0,
   'weight_decay': 0,
   'nesterov': False,
   'params': [0, 1]}]}
```

we load the loss

In [16]:

```
loss =checkpoint['loss']
print('loss:',loss)
```

```
loss: tensor(0.0034, requires_grad=True)
```

we continue training the model

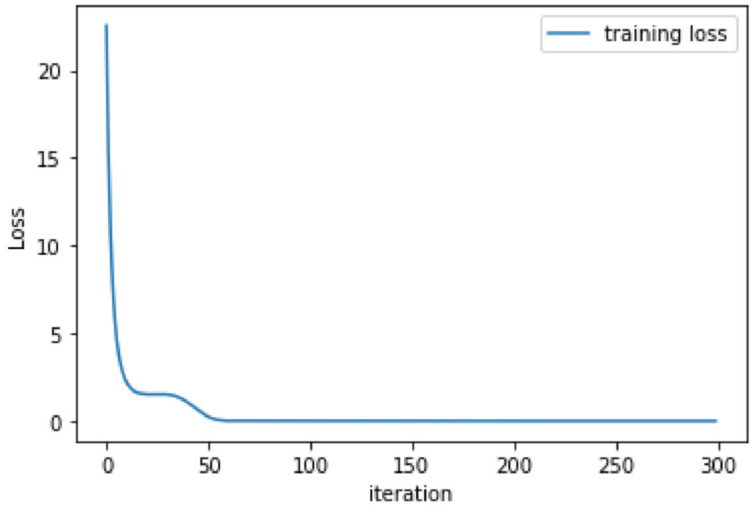
In [17]:

```
for epoch in range(checkpoint['epoch'],epochs):
    for x, y in trainloader:
        yhat = model_checkpoint(x)
        loss = criterion(yhat, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_train = criterion(model_checkpoint(train_data.x), train_data.y).item()

        LOSS_TRAIN.append(loss_train)
```

View the loss for every iteration on the training set

In [18]:

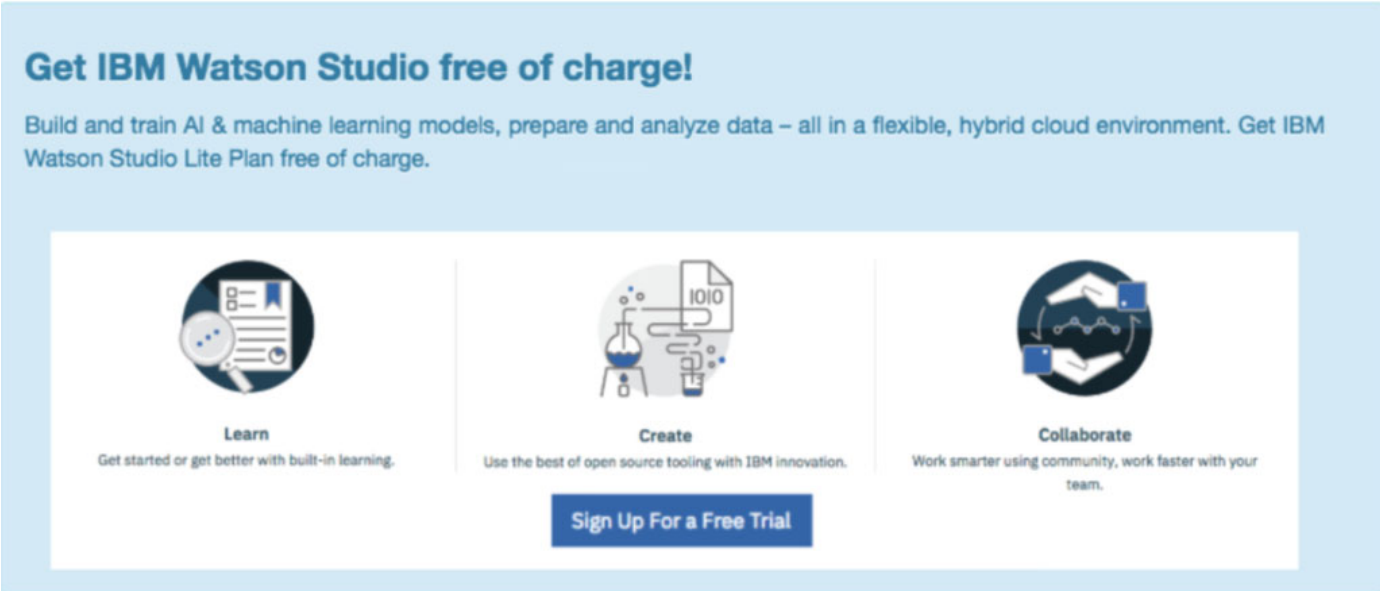
```
# Plot the loss

plt.plot(LOSS_TRAIN, label = 'training loss')
plt.xlabel("iteration")
plt.ylabel("Loss")
plt.legend(loc = 'upper right')
plt.show()
```




We can see the model obtained via early stopping fits the data points much better. For more variations of early stopping see:

Prechelt, Lutz. *"Early stopping-but when?." Neural Networks: Tricks of the trade. Springer, Berlin, Heidelberg, 1998. 55-69*.

Inference




[(http://cocl.us/pytorch_link_bottom)](http://cocl.us/pytorch_link_bottom)

## About the Authors:

[Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/)](https://www.linkedin.com/in/joseph-s-50398b136/) has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Other contributors: [Michelle Carey (https://www.linkedin.com/in/michelleccarey/)](https://www.linkedin.com/in/michelleccarey/), [Mavis Zhou (www.linkedin.com/in/jiahui-mavis-zhou-a4537814a)](www.linkedin.com/in/jiahui-mavis-zhou-a4537814a)

---