PyTorch for .NET Developers with Azure Machine Learning

Adnan Masood, PhD

Microsoft Azure
+AI Conference
CO-PRODUCED BY
Microsoft & DEVintersection

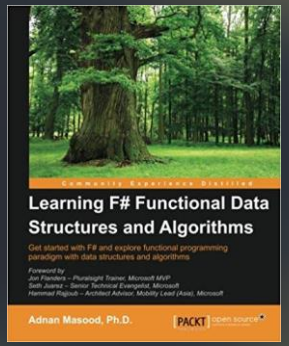
About the Speaker

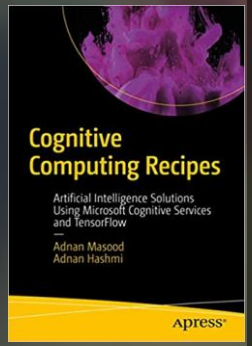
Microsoft® MVP Most Valuable Professional

Adnan Masood, Ph.D. is an Artificial Intelligence and Machine Learning researcher, software architect, and Microsoft MVP (Most Valuable Professional) for Artificial Intelligence. As Chief Architect of AI and Machine Learning, at UST Global, he collaborates with Stanford Artificial Intelligence Lab, and MIT AI Lab for building enterprise solutions

Author of Amazon bestseller in programming languages, **"Functional Programming with F#",** Dr. Masood teaches Data Science at Park University, and has taught Windows Communication Foundation (WCF) courses at the University of California, San Diego. He is a regular speaker to various academic and technology conferences (WICT, DevIntersection, IEEE-HST, IASA, and DevConnections), local code camps, and user groups. He also volunteers as STEM (Science Technology, Engineering and Math) robotics coach for elementary and middle school students

A strong believer in giving back to the community, Dr. Masood is a co-founder and president of the Pasadena .NET Developers group, co-organizer of Tampa Bay Data Science Group, and Irvine Programmer meetup. His recent talk at Women in Technology Conference (WICT) Denver highlighted the importance of diversity in STEM and technology areas, and was featured by variety of news outlets.





Andrej Karpathy

@karpathy

Follow

I've been using PyTorch a few months now and I've never felt better. I have more energy. My skin is clearer. My eye sight has improved.

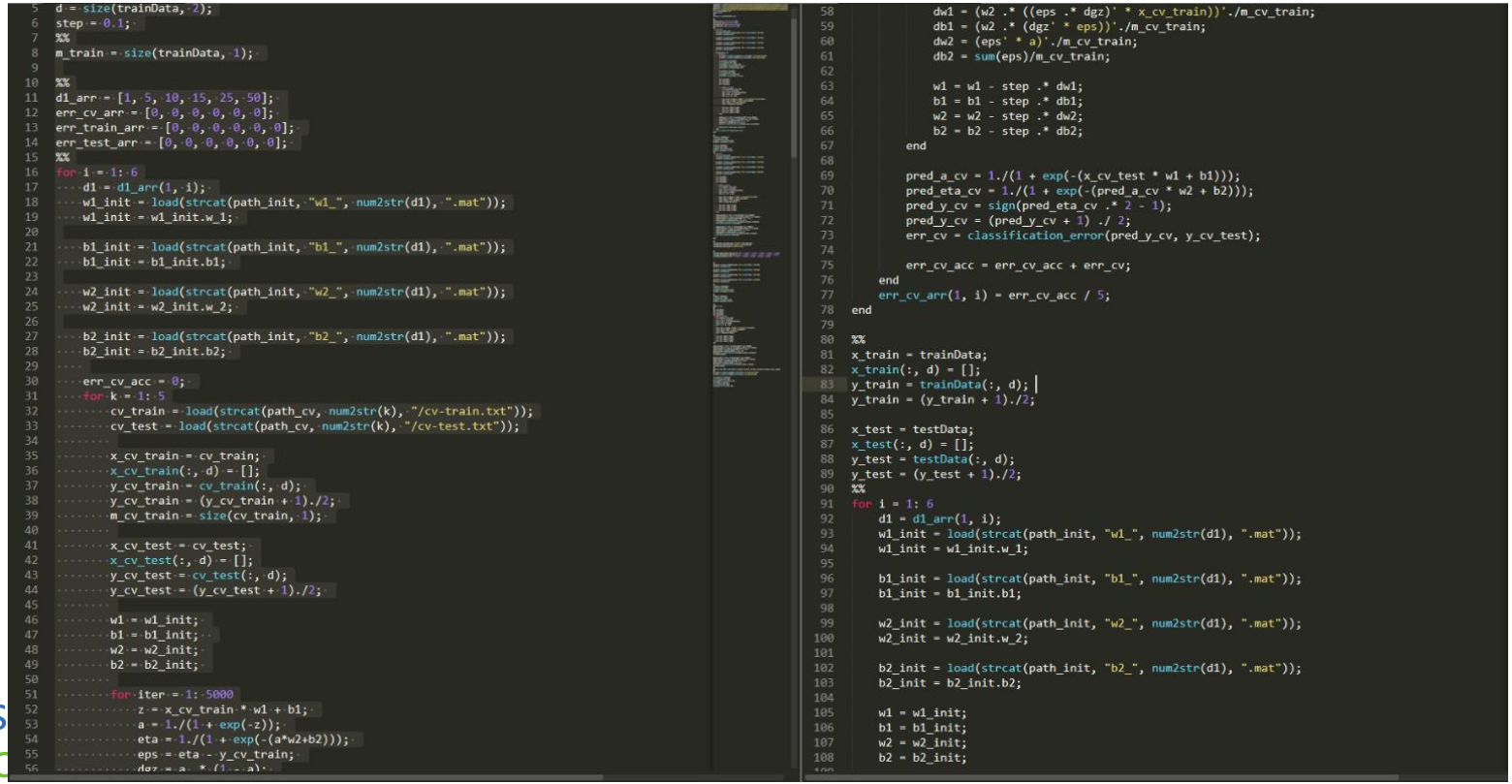
11:56 AM - 26 May 2017

399 Retweets **1,532** Likes

PyTorch for .NET Developers with Azure Machine Learning

- **PyTorch is a Python-based scientific computing package that uses the power of modern GPUS. It is quickly becoming a preferred deep learning research platform built to provide maximum flexibility and speed using dynamic graph.**
- **In this talk we will show how Pytorch can be used on Azure using Azure Machine Learning service, Data Science Virtual Machine, Azure Notebooks, and Visual Studio Code Tools for AI. We will train PyTorch models with Azure Machine Learning service, and how we can use the ONNX runtime inside of our .NET applications. ONNX is a open format to represent deep learning models that is supported by various frameworks and tools. This format makes it easier to interoperate between frameworks and to maximize the reach of your hardware optimization investments.**

Because writing ML code from scratch sucks (part 1)



```
d = size(trainData, 2);
step = 0.1;
%%
m_train = size(trainData, 1);

%%
d1_arr = [1, 5, 10, 15, 25, 50];
err_cv_arr = [0, 0, 0, 0, 0, 0];
err_train_arr = [0, 0, 0, 0, 0, 0];
err_test_arr = [0, 0, 0, 0, 0, 0];
%%
for i = 1: 6
    d1 = d1_arr(1, i);
    w1_init = load(strcat(path_init, "w1_", num2str(d1), ".mat"));
    w1_init = w1_init.w_1;

    b1_init = load(strcat(path_init, "b1_", num2str(d1), ".mat"));
    b1_init = b1_init.b1;

    w2_init = load(strcat(path_init, "w2_", num2str(d1), ".mat"));
    w2_init = w2_init.w_2;

    b2_init = load(strcat(path_init, "b2_", num2str(d1), ".mat"));
    b2_init = b2_init.b2;

    err_cv_acc = 0;
    for k = 1: 5
        cv_train = load(strcat(path_cv, num2str(k), "/cv-train.txt"));
        cv_test = load(strcat(path_cv, num2str(k), "/cv-test.txt"));

        x_cv_train = cv_train;
        x_cv_train(:, d) = [];
        y_cv_train = cv_train(:, d);
        y_cv_train = (y_cv_train + 1)./2;
        m_cv_train = size(cv_train, 1);

        x_cv_test = cv_test;
        x_cv_test(:, d) = [];
        y_cv_test = cv_test(:, d);
        y_cv_test = (y_cv_test + 1)./2;

        w1 = w1_init;
        b1 = b1_init;
        w2 = w2_init;
        b2 = b2_init;

        for iter = 1: 5000
            z = x_cv_train * w1 + b1;
            a = 1./(1 + exp(-z));
            eta = 1./(1 + exp(-(a*w2+b2)));
            eps = eta - y_cv_train;
            dgz = a .* (1 - a);
            dw1 = (w2 .* ((eps .* dgz)' * x_cv_train))'./m_cv_train;
            db1 = (w2 .* (dgz' * eps))'./m_cv_train;
            dw2 = (eps' * a)'./m_cv_train;
            db2 = sum(eps)/m_cv_train;

            w1 = w1 - step .* dw1;
            b1 = b1 - step .* db1;
            w2 = w2 - step .* dw2;
            b2 = b2 - step .* db2;
        end

        pred_a_cv = 1./(1 + exp(-(x_cv_test * w1 + b1)));
        pred_eta_cv = 1./(1 + exp(-(pred_a_cv * w2 + b2)));
        pred_y_cv = sign(pred_eta_cv .* 2 - 1);
        pred_y_cv = (pred_y_cv + 1) ./ 2;
        err_cv = classification_error(pred_y_cv, y_cv_test);

        err_cv_acc = err_cv_acc + err_cv;
    end
    err_cv_arr(1, i) = err_cv_acc / 5;
end

%%
x_train = trainData;
x_train(:, d) = [];
y_train = trainData(:, d);
y_train = (y_train + 1)./2;

x_test = testData;
x_test(:, d) = [];
y_test = testData(:, d);
y_test = (y_test + 1)./2;
%%
for i = 1: 6
    d1 = d1_arr(1, i);
    w1_init = load(strcat(path_init, "w1_", num2str(d1), ".mat"));
    w1_init = w1_init.w_1;

    b1_init = load(strcat(path_init, "b1_", num2str(d1), ".mat"));
    b1_init = b1_init.b1;

    w2_init = load(strcat(path_init, "w2_", num2str(d1), ".mat"));
    w2_init = w2_init.w_2;

    b2_init = load(strcat(path_init, "b2_", num2str(d1), ".mat"));
    b2_init = b2_init.b2;

    w1 = w1_init;
    b1 = b1_init;
    w2 = w2_init;
    b2 = b2_init;
```

Because writing ML code from scratch sucks (part 2)

```
    w2 = w2_init;
    b2 = b2_init;

    for iter = 1: 5000
        z = x_train * w1 + b1;
        a = 1./(1 + exp(-z));
        eta = 1./(1 + exp(-(a*w2+b2)));
        eps = eta - y_train;
        dgz = a .* (1 - a);

        dw1 = (w2 .* ((eps .* dgz)' * x_train))'./m_train;
        db1 = (w2 .* (dgz' * eps))'./m_train;
        dw2 = (eps' * a)'./m_train;
        db2 = sum(eps)/m_train;

        w1 = w1 - step .* dw1;
        b1 = b1 - step .* db1;
        w2 = w2 - step .* dw2;
        b2 = b2 - step .* db2;
    end

    pred_a_train = 1./(1 + exp(-(x_train * w1 + b1)));
    pred_eta_train = 1./(1 + exp(-(pred_a_train * w2 + b2)));
    pred_y_train = sign(pred_eta_train .* 2 - 1);
    pred_y_train = (pred_y_train + 1) ./ 2;
    err_train = classification_error(pred_y_train, y_train);
    err_train_arr(1, i) = err_train;

    pred_a_test = 1./(1 + exp(-(x_test * w1 + b1)));
    pred_eta_test = 1./(1 + exp(-(pred_a_test * w2 + b2)));
    pred_y_test = sign(pred_eta_test .* 2 - 1);
    pred_y_test = (pred_y_test + 1) ./ 2;
    err_test = classification_error(pred_y_test, y_test);
    err_test_arr(1, i) = err_test;

end

%%
plot(d1_arr, err_train_arr, 'color', 'b'); hold on;
plot(d1_arr, err_test_arr, 'color', 'r'); hold on;
plot(d1_arr, err_cv_arr, 'color', 'g');


%%
disp(err_train_arr); hold on; %0.3960    0.1440    0.1120    0.0760    0.0640    0.0360
disp(err_test_arr); hold on; %0.3939    0.2066    0.1726    0.1563    0.1538    0.1544
disp(err_cv_arr); %0.3960    0.1840    0.1640    0.1000    0.1520    0.1440



%%
w1_init = load(strcat(path_init, "w1_", num2str(5), ".mat"));
```

```

b1_init = load(strcat(path_init, "b1_", num2str(5), ".mat"));
b1_init = b1_init.b1;

w2_init = load(strcat(path_init, "w2_", num2str(5), ".mat"));
w2_init = w2_init.w_2;

b2_init = load(strcat(path_init, "b2_", num2str(5), ".mat"));
b2_init = b2_init.b2;

%%
x_train = trainData;
x_train(:, d) = [];
y_train = trainData(:, d);
y_train = (y_train + 1)./2;

%%
x_test = testData;
x_test(:, d) = [];
y_test = testData(:, d);
y_test = (y_test + 1)./2;

%%
step = 0.1;
%%
w1 = w1_init;
b1 = b1_init;
w2 = w2_init;
b2 = b2_init;
for iter = 1: 5000
    z = x_train * w1 + b1;
    a = 1./(1 + exp(-z));
    eta = 1./(1 + exp(-(a*w2+b2)));
    eps = eta - y_train;
    dgz = a .* (1 - a);

    dw1 = (w2 .* ((eps .* dgz)' * x_train))'./m_train;
    db1 = (w2 .* (dgz' * eps))'./m_train;
    dw2 = (eps' * a)'./m_train;
    db2 = sum(eps)/m_train;

    w1 = w1 - step .* dw1;
    b1 = b1 - step .* db1;
    w2 = w2 - step .* dw2;
    b2 = b2 - step .* db2;
end

pred_a_train = 1./(1 + exp(-(x_train * w1 + b1)));
pred_eta_train = 1./(1 + exp(-(pred_a_train * w2 + b2)));
pred_y_train = sign(pred_eta_train .* 2 - 1);
pred_y_train = (pred_y_train + 1) ./ 2;
```

Because writing ML code from scratch sucks (part 3)

```
pred_y_train = sign(pred_eta_train .* 2 - 1);
pred_y_train = (pred_y_train + 1) ./ 2;
err_train = classification_error(pred_y_train, y_train);
disp(err_train);

pred_a_test = 1./(1 + exp(-(x_test * w1 + b1)));
pred_eta_test = 1./(1 + exp(-(pred_a_test * w2 + b2)));
pred_y_test = sign(pred_eta_test .* 2 - 1);
pred_y_test = (pred_y_test + 1) ./ 2;
err_test = classification_error(pred_y_test, y_test);
disp(err_test);
%%
[w1, b1, w2, b2] = nn(x_train, y_train, w1_init, b1_init, w2_init, bw_init, step, iter);
%%
cv_train = load(strcat(path, num2str(1), "/cv-train.txt"));
cv_test = load(strcat(path, num2str(1), "/cv-test.txt"));

x_cv_train = cv_train;
x_cv_train(:, d) = [];
y_cv_train = cv_train(:, d);
x_cv_test = cv_test;
x_cv_test(:, d) = [];
y_cv_test = cv_test(:, d);
```

Contrast with PyTorch

```python
# Fully connected neural network with one hidden layer
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

```python
model = NeuralNet(input_size, hidden_size, num_classes).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Move tensors to the configured device
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Deep learning frameworks

- Modern tools make it easy to implement neural networks
- Often used components
  - Linear, convolution, recurrent layers etc.
- Many frameworks available: Torch (2002), Theano (2011), Caffe (2014), TensorFlow (2015), PyTorch (2016)

Libraries for Deep Learning

- **Torch (Lua):**
  - http://torch.ch/
- **PyTorch (Python)**
  - http://pytorch.org/
- **TensorFlow (Python and C++):**
  - https://www.tensorflow.org/
- **Theano (Python)**
  - http://deeplearning.net/software/theano/
  - No longer maintained
- **Keras, PaddlePaddle, CNTK**

What is TensorFlow?

- **Open source software library for numerical computation using data flow graphs**
- **Developed by Google Brain Team for machine learning and deep learning and made open-source**
- **TensorFlow provides an extensive suite of functions and classes that allow users to build various models from scratch**

These slides are adapted from the following Stanford lectures:

https://web.stanford.edu/class/cs20si/2017/lectures/slides_01.pdf

https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

What's a tensor?

- **Formally, tensors are multilinear maps from vector spaces to the real numbers**
- **Think of them as n-dimensional array, with 0-d tensors being scalars, 1-d tensor vectors, 2-d tensor matrices, etc**

PyTorch

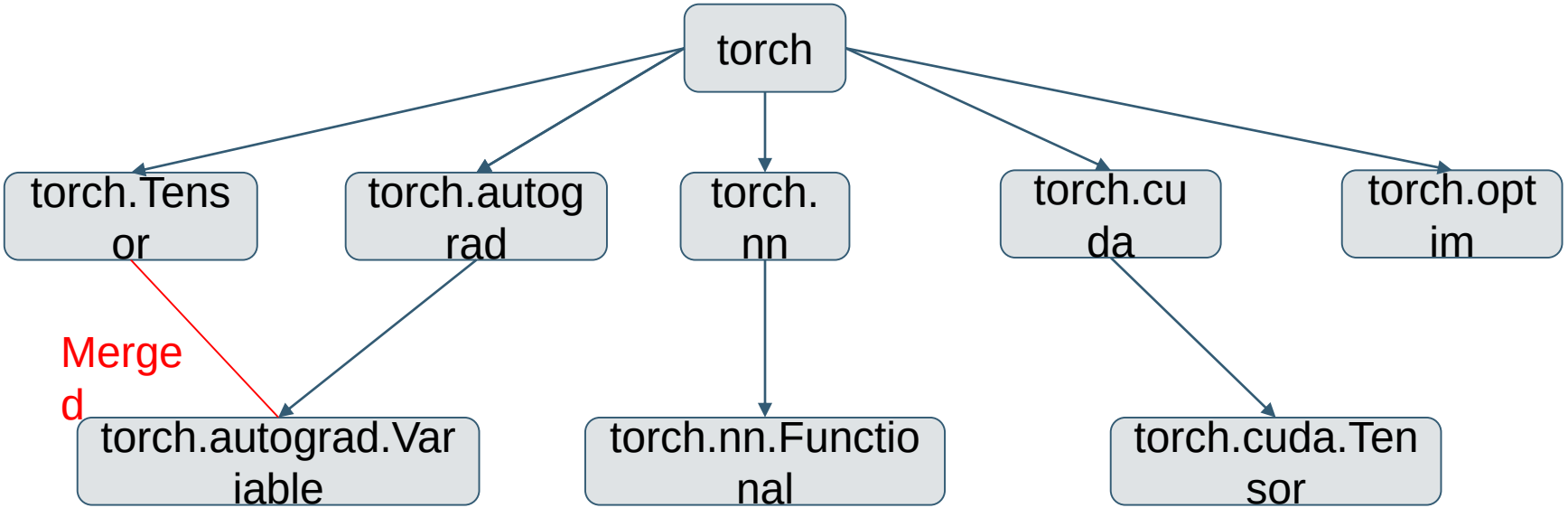
Pytorch

- Fast tensor computation (like numpy) with strong GPU support
- Deep learning research platform that provides maximum flexibility and speed
- Dynamic graphs and automatic differentiation

PyTorch

- Based on Torch, a scientific computing library for Lua
- Developed by FAIR
- Main features are the built-in computational graph and built-in GPU acceleration

Structure of PyTorch library



Basic Concepts

torch.Tensor - similar to numpy.array

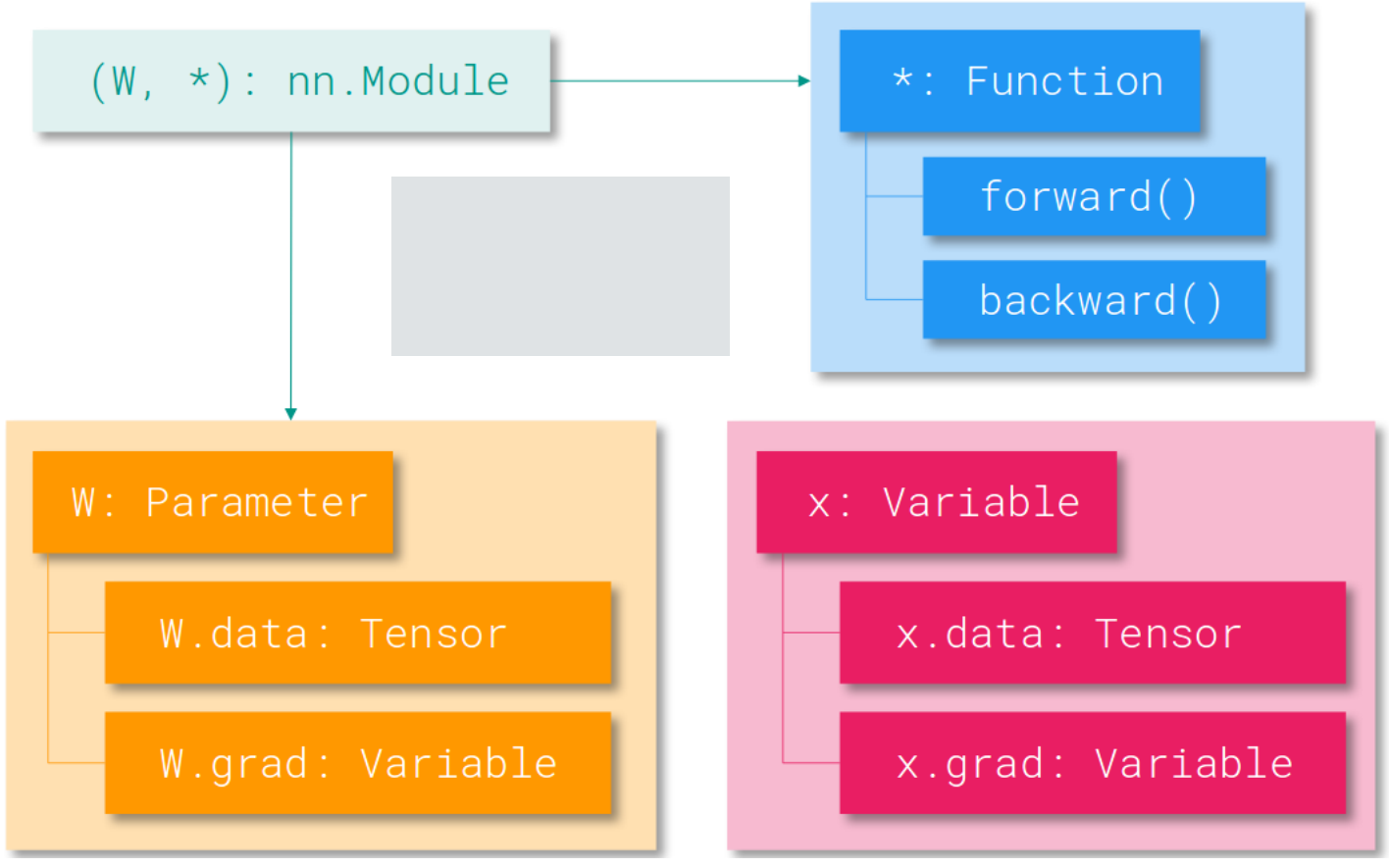
autograd.Variable - wraps a *Tensor* and enables auto differentiation

autograd.Function - operate on *Variables*, implements forward and backward

nn.Parameter - contain *Parameters* and define functions on input *Variables*

nn.Module - contain *Parameters* and define functions on input *Variables*

Basic Concepts



Basic Concepts

```
import torch
x = torch.rand(10, 5)
y = torch.rand(10,5)
z = x * y
```

Matrix Multiplication in Python

```
def matrixmult (A, B):
    rows_A = len(A)
    cols_A = len(A[0])
    rows_B = len(B)
    cols_B = len(B[0])

    if cols_A != rows_B:
      print "Cannot multiply the two matrices. Incorrect dimensions."
      return

    # Create the result matrix
    # Dimensions would be rows_A x cols_B
    C = [[0 for row in range(cols_B)] for col in range(rows_A)]
    print C

    for i in range(rows_A):
        for j in range(cols_B):
            for k in range(cols_A):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

http://stackoverflow.com/questions/10508021/matrix-

Matrix Multiplication in Numpy

```
>>> a=np.array([[1,2],[3,4]])
>>> b=np.array([[11,12],[13,14]])
>>> np.dot(a,b)
array([[37, 40],
       [85, 92]])
```

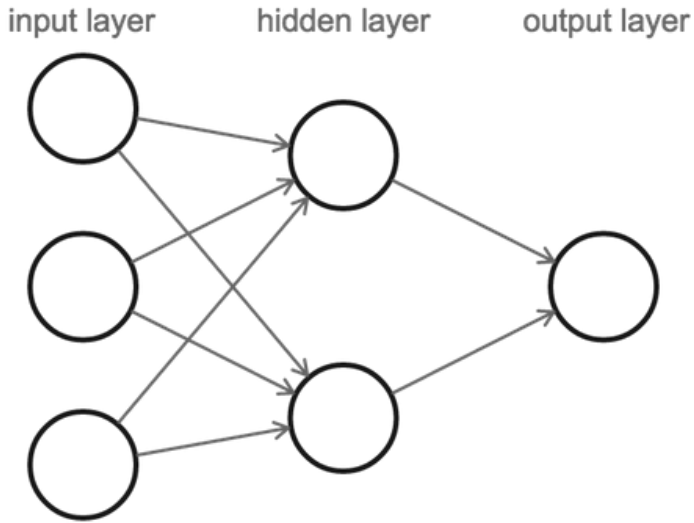
Defining a Neural Net in PyTorch

```python
# Fully connected neural network with one hidden layer
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

```python
model = NeuralNet(input_size, hidden_size, num_classes).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Move tensors to the configured device
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Network definition

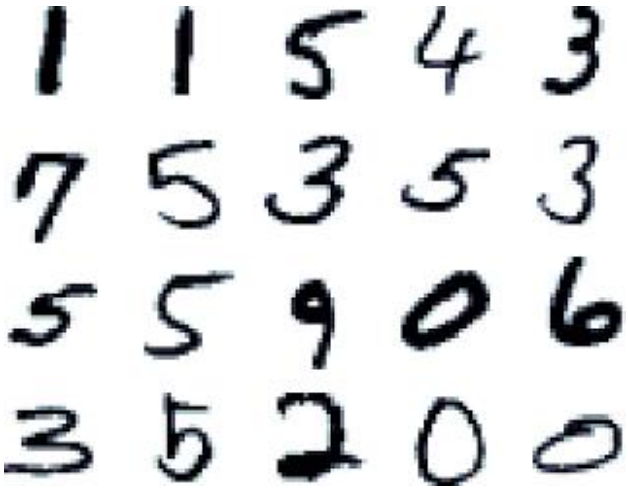
```
class TwoLayerNet(torch.nn.Module):
  def __init__(self, D_in, H, D_out):
    super(TwoLayerNet, self).__init__()
    self.linear1 = torch.nn.Linear(D_in, H)
    self.linear2 = torch.nn.Linear(H, D_out)
    self.relu = torch.nn.ReLU(inplace=True)

  def forward(self, x):
    h_relu = self.linear1(x)
    h_relu = self.relu(h_relu)
    y_pred = self.linear2(h_relu)
return y_pred
```



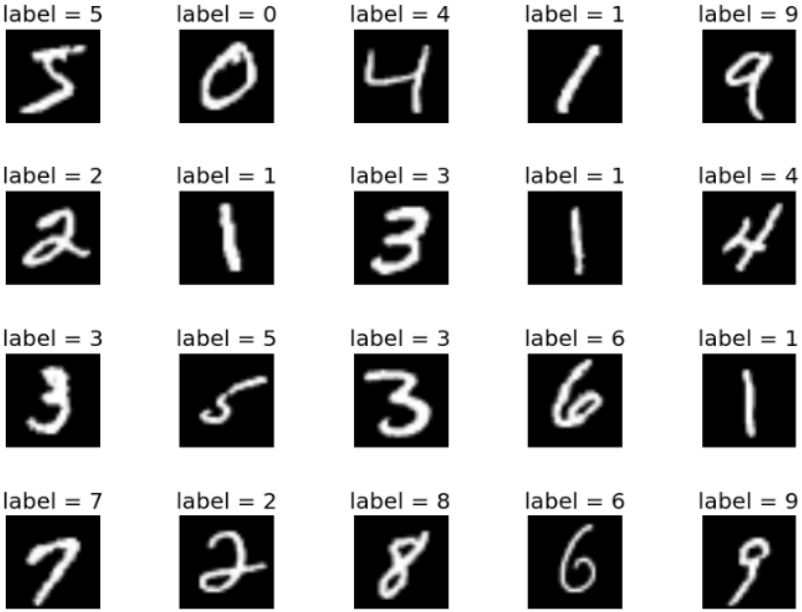
Optimize the Network

```
N, D_in, H, D_out = 64, 1000, 100, 10
model = TwoLayerNet(D_in, H, D_out)
loss_fn = torch.nn.MSELoss(size_average=False)
optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)

for t in range(500):
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x)
    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
optimizer.step()
```

Classifying Handwritten Digits

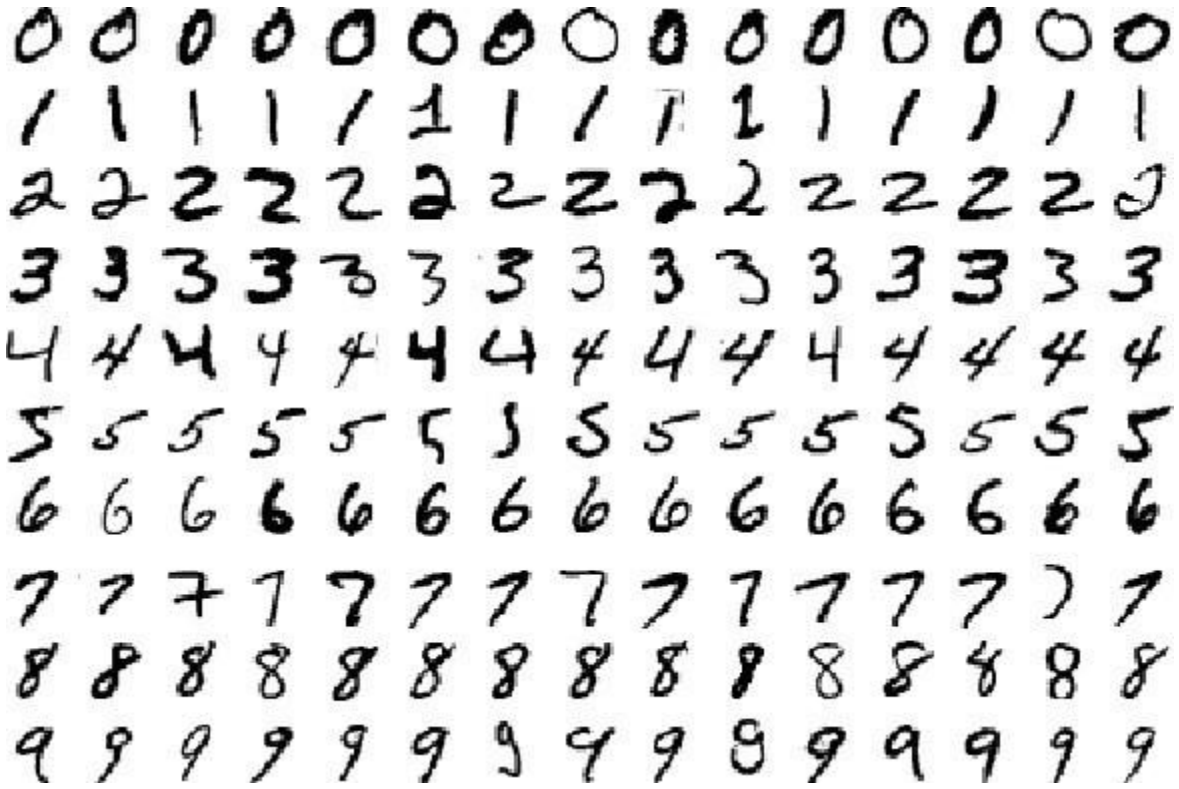


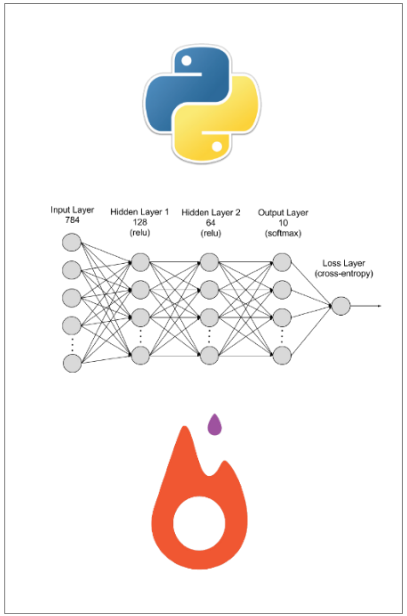
- Input Image
- Conv 5x5, 1/10
- MaxPool 2x2 + RELU
- Conv 5x5, 10/20
- MaxPool 2x2 + RELU
- Conv 5x5, 10/20
- Dropout , p = 0.5
- Linear, 320/50
- Linear, 50/10
- Softmax



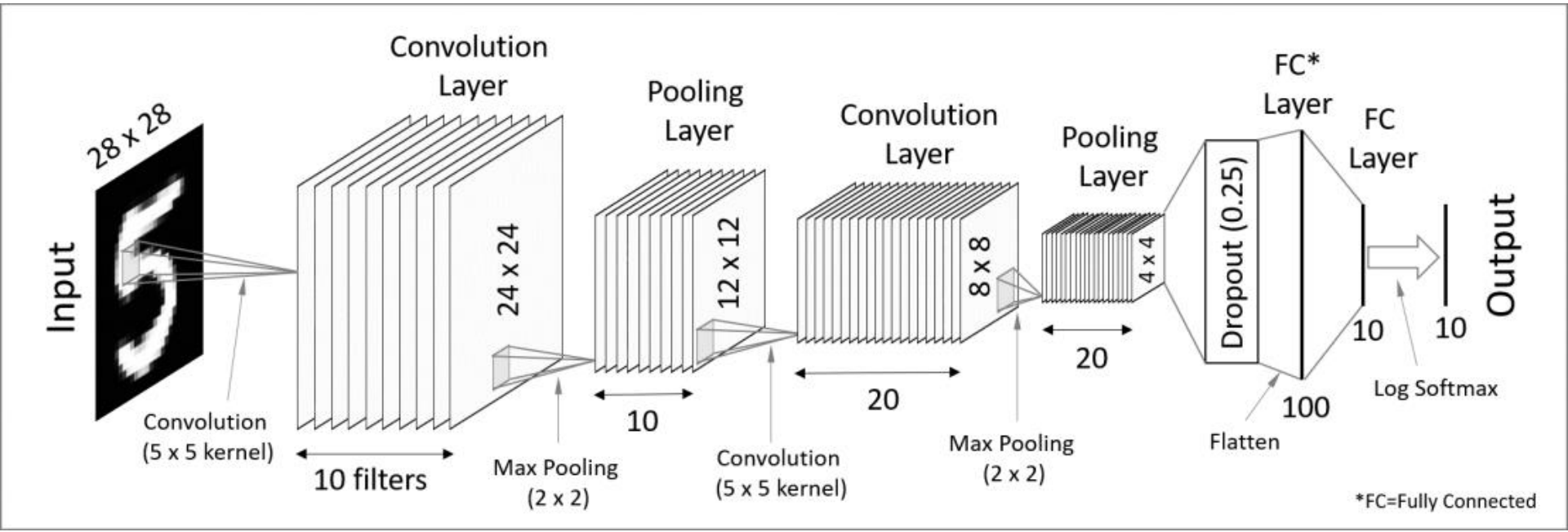
is the new

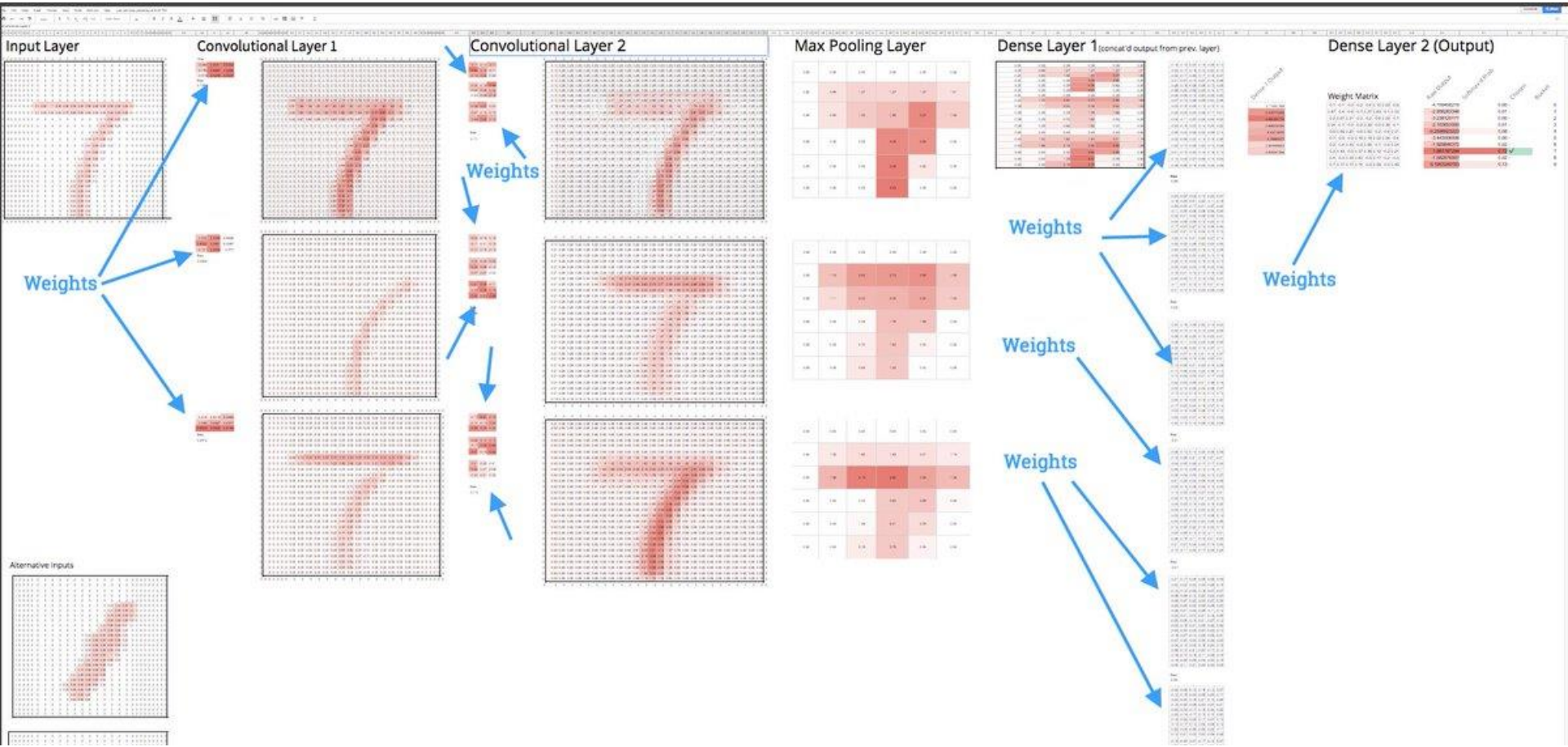
```
>>> print("Hello World!")
Hello World!
```




Input Layer
784
Hidden Layer 1
128
(relu)
Hidden Layer 2
64
(relu)
Output Layer
10
(softmax)
Loss Layer
(cross-entropy)

2


Input
28 x 28
Convolution
(5 x 5 kernel)
Convolution
Layer
24 x 24
10 filters
Pooling
Layer
12 x 12
Max Pooling
(2 x 2)
10
Convolution
Layer
8 x 8
Convolution
(5 x 5 kernel)
20
Pooling
Layer
4 x 4
Max Pooling
(2 x 2)
20
Dropout (0.25)
FC*
Layer
Flatten
100
FC
Layer
10
Log Softmax
10
Output
*FC=Fully Connected


Input Layer
Convolutional Layer 1
Convolutional Layer 2
Max Pooling Layer
Dense Layer 1 (concat'd output from prev. layer)
Dense Layer 2 (Output)
Weight Matrix
Weights
Weights
Weights
Weights
Weights
Weights
Alternative Inputs

Define a CNN Network Class

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

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolutio
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

Questions?

- adnanmasood@gmail.com
- https://twitter.com/adnanmasood
- https://www.linkedin.com/in/adnano

Please use EventsXD to fill out a session evaluation.

Thank you!

Microsoft Azure
+ AI Conference
CO-PRODUCED BY
Microsoft & DEVintersection