From multiplication to convolutional networks

How do ML with Theano

Today’s Talk

- A motivating problem
- Understanding a model based framework
- Theano
  - Linear Regression
  - Logistic Regression
  - Net
  - Modern Net
  - Convolutional Net

Follow along

Tutorial code at:

https://github.com/Newmu/Theano-Tutorials

Data at:

http://yann.lecun.com/exdb/mnist/

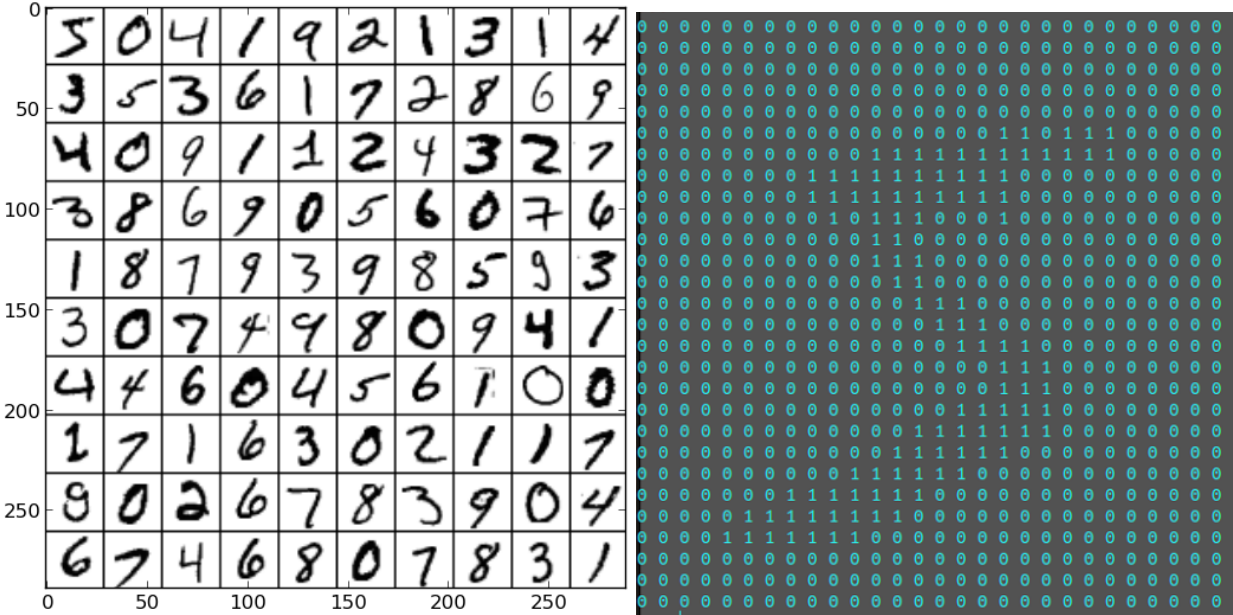
Slides at:

http://goo.gl/vuBQfe

A motivating problem

How do we program a computer to recognize a picture of a handwritten digit as a 0-9?

What could we do?



A dataset - MNIST

What if we have 60,000 of these images and their label?

```
X = images
Y = labels
X = (60000 x 784) #matrix (list of lists)
Y = (60000)       #vector (list)
```

Given X as input, predict Y

An idea

For each image, find the “most similar” image and guess that as the label.

```
def similarity(image, images):
    similarities = []
    for img in images:
        distance = sqrt(sum(square(image - img)))
        sim = 1 / distance
        similarities.append(sim)
    return similarities


def ocr(image, images, labels):
    similarities = similarity(image, images)
    return labels[argmax(similarities)]
```

An idea

For each image, find the “most similar” image and guess that as the label.

```
def similarity(image, images):
    similarities = []
    for img in images:
        distance = sqrt(sum(square(image - img)))
        sim = 1 / distance
        similarities.append(sim)
    return similarities


def ocr(image, images, labels):
    similarities = similarity(image, images)
    return labels[argmax(similarities)]
```

KNearestNeighbors ~95% accuracy

Trying things

Make some functions computing relevant information for solving the problem

```
def ocr(image):
    if n_closed_loops(image) == 1 and n_vertical_strokes == 0:
        return 0
    elif n_vertical_strokes(image) == 1 and width(image) <= 10:
        return 1
    elif ....
    else: ???
```

What we can code

Make some functions computing relevant information for solving the problem

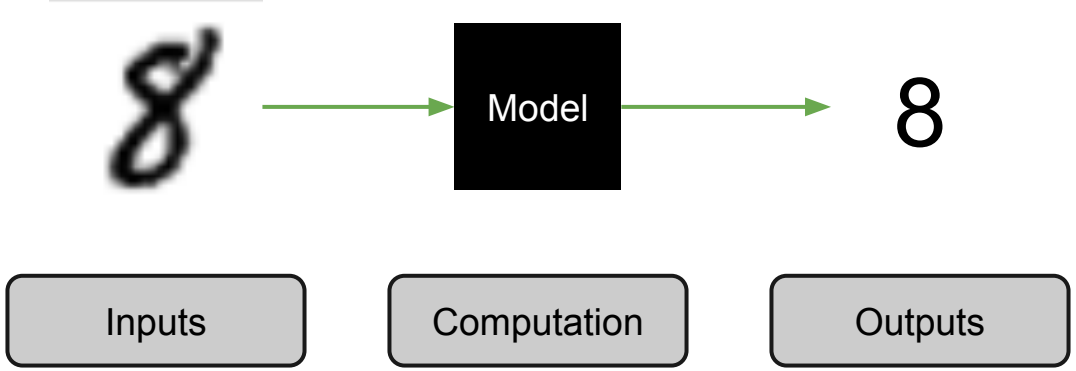
```
def ocr(image):
    if n_closed_loops(image) == 1 and n_vertical_strokes == 0:
        return 0
    elif n_vertical_strokes(image) == 1 and width(image) <= 10:
        return 1
    elif ....
    else: ???
```

feature engineering

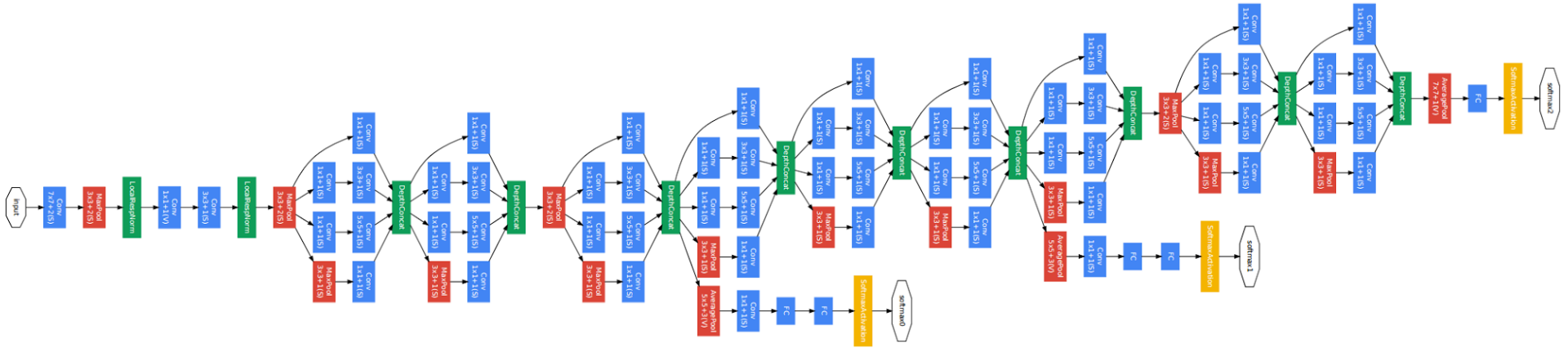
What we can code

Hard coded rules are brittle and often aren't obvious or apparent for many problems.

```
def ocr(image):
    if n_closed_loops(image) == 1 and n_vertical_strokes == 0:
        return 0
    elif n_vertical_strokes(image) == 1 and width(image) <= 10:
        return 1
    elif ....
    else: ???
```

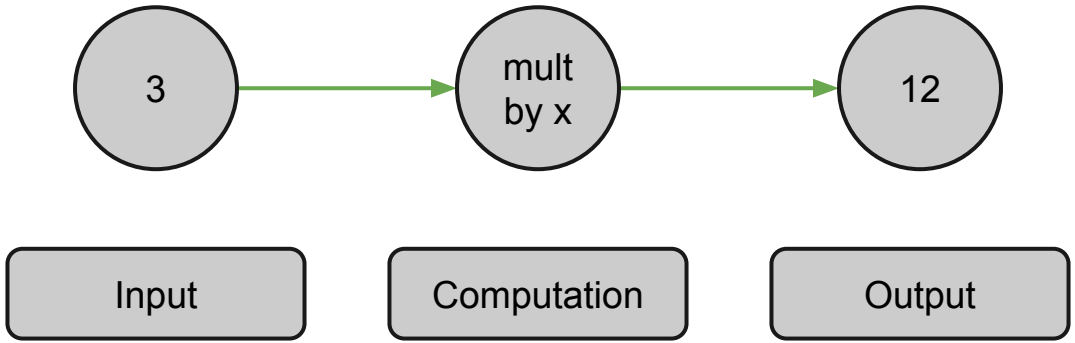


A Machine Learning Framework



from arXiv:1409.4842v1 [cs.CV] 17 Sep 2014

A … model? - GoogLeNet



A very simple model

```
1  import theano
2  from theano import tensor as T
3
4  a = T.scalar()
5  b = T.scalar()
6
7  y = a * b
8
9  multiply = theano.function(inputs=[a, b], outputs=y)
10
11 print multiply(1, 2) #2
12 print multiply(3, 3) #9
13
```

Theano intro

```
1  import theano
2  from theano import tensor as T
3
4  a = T.scalar()
5  b = T.scalar()
6
7  y = a * b
8
9  multiply = theano.function(inputs=[a, b], outputs=y)
10
11 print multiply(1, 2) #2
12 print multiply(3, 3) #9
13
```

imports

Theano intro

```
1  import theano
2  from theano import tensor as T
3
4  a = T.scalar()
5  b = T.scalar()
6
7  y = a * b
8
9  multiply = theano.function(inputs=[a, b], outputs=y)
10
11 print multiply(1, 2) #2
12 print multiply(3, 3) #9
13
```

imports

theano symbolic variable initialization

Theano intro

```
1  import theano
2  from theano import tensor as T
3
4  a = T.scalar()
5  b = T.scalar()
6
7  y = a * b
8
9  multiply = theano.function(inputs=[a, b], outputs=y)
10
11 print multiply(1, 2) #2
12 print multiply(3, 3) #9
13
```

imports

theano symbolic variable initialization

our model

Theano intro

```
1  import theano
2  from theano import tensor as T
3
4  a = T.scalar()
5  b = T.scalar()
6
7  y = a * b
8
9  multiply = theano.function(inputs=[a, b], outputs=y)
10
11 print multiply(1, 2) #2
12 print multiply(3, 3) #9
13
```

imports

theano symbolic variable initialization

our model

compiling to a python function

Theano intro

```
1  import theano
2  from theano import tensor as T
3
4  a = T.scalar()
5  b = T.scalar()
6
7  y = a * b
8
9  multiply = theano.function(inputs=[a, b], outputs=y)
10
11 print multiply(1, 2) #2
12 print multiply(3, 3) #9
13
```

imports

theano symbolic variable initialization

our model

compiling to a python function

usage

Theano intro

```python
import theano
from theano import tensor as T
import numpy as np

trX = np.linspace(-1, 1, 101)
trY = 2 * trX + np.random.randn(*trX.shape) * 0.33

X = T.scalar()
Y = T.scalar()

def model(X, w):
    return X * w

w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
y = model(X, w)

cost = T.mean(T.sqr(y - Y))
gradient = T.grad(cost=cost, wrt=w)
updates = [[w, w - gradient * 0.01]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)

for i in range(100):
    for x, y in zip(trX, trY):
        train(x, y)

```

Theano

imports

```
import theano
from theano import tensor as T
import numpy as np

trX = np.linspace(-1, 1, 101)
trY = 2 * trX + np.random.randn(*trX.shape) * 0.33

X = T.scalar()
Y = T.scalar()

def model(X, w):
    return X * w

w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
y = model(X, w)

cost = T.mean(T.sqr(y - Y))
gradient = T.grad(cost=cost, wrt=w)
updates = [[w, w - gradient * 0.01]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)

for i in range(100):
    for x, y in zip(trX, trY):
        train(x, y)
```

Theano

```python
import theano
from theano import tensor as T
import numpy as np

trX = np.linspace(-1, 1, 101)
trY = 2 * trX + np.random.randn(*trX.shape) * 0.33

X = T.scalar()
Y = T.scalar()

def model(X, w):
    return X * w

w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
y = model(X, w)

cost = T.mean(T.sqr(y - Y))
gradient = T.grad(cost=cost, wrt=w)
updates = [[w, w - gradient * 0.01]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)

for i in range(100):
    for x, y in zip(trX, trY):
        train(x, y)
```

imports

training data generation

Theano

```python
import theano
from theano import tensor as T
import numpy as np

trX = np.linspace(-1, 1, 101)
trY = 2 * trX + np.random.randn(*trX.shape) * 0.33

X = T.scalar()
Y = T.scalar()

def model(X, w):
    return X * w

w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
y = model(X, w)

cost = T.mean(T.sqr(y - Y))
gradient = T.grad(cost=cost, wrt=w)
updates = [[w, w - gradient * 0.01]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)

for i in range(100):
    for x, y in zip(trX, trY):
        train(x, y)
```

imports

training data generation

symbolic variable initialization

Theano

```python
import theano
from theano import tensor as T
import numpy as np

trX = np.linspace(-1, 1, 101)
trY = 2 * trX + np.random.randn(*trX.shape) * 0.33

X = T.scalar()
Y = T.scalar()

def model(X, w):
    return X * w

w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
y = model(X, w)

cost = T.mean(T.sqr(y - Y))
gradient = T.grad(cost=cost, wrt=w)
updates = [[w, w - gradient * 0.01]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)

for i in range(100):
    for x, y in zip(trX, trY):
        train(x, y)
```

imports

training data generation

symbolic variable initialization

our model

Theano

```
1  import theano
2  from theano import tensor as T
3  import numpy as np
4
5  trX = np.linspace(-1, 1, 101)
6  trY = 2 * trX + np.random.randn(*trX.shape) * 0.33
7
8  X = T.scalar()
9  Y = T.scalar()
10
11 def model(X, w):
12     return X * w
13
14 w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
15 y = model(X, w)
16
17 cost = T.mean(T.sqr(y - Y))
18 gradient = T.grad(cost=cost, wrt=w)
19 updates = [[w, w - gradient * 0.01]]
20
21 train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)
22
23 for i in range(100):
24     for x, y in zip(trX, trY):
25         train(x, y)
26
```

imports

training data generation

symbolic variable initialization

our model

model parameter initialization

Theano

```
1  import theano
2  from theano import tensor as T
3  import numpy as np
4
5  trX = np.linspace(-1, 1, 101)
6  trY = 2 * trX + np.random.randn(*trX.shape) * 0.33
7
8  X = T.scalar()
9  Y = T.scalar()
10
11 def model(X, w):
12     return X * w
13
14 w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
15 y = model(X, w)
16
17 cost = T.mean(T.sqr(y - Y))
18 gradient = T.grad(cost=cost, wrt=w)
19 updates = [[w, w - gradient * 0.01]]
20
21 train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)
22
23 for i in range(100):
24     for x, y in zip(trX, trY):
25         train(x, y)
26
```

imports

training data generation

symbolic variable initialization

our model

model parameter initialization

metric to be optimized by model

Theano

```python
import theano
from theano import tensor as T
import numpy as np

trX = np.linspace(-1, 1, 101)
trY = 2 * trX + np.random.randn(*trX.shape) * 0.33

X = T.scalar()
Y = T.scalar()

def model(X, w):
    return X * w

w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
y = model(X, w)

cost = T.mean(T.sqr(y - Y))
gradient = T.grad(cost=cost, wrt=w)
updates = [[w, w - gradient * 0.01]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)

for i in range(100):
    for x, y in zip(trX, trY):
        train(x, y)
```

imports

training data generation

symbolic variable initialization

our model

model parameter initialization

metric to be optimized by model

learning signal for parameter(s)

Theano

```
1  import theano
2  from theano import tensor as T
3  import numpy as np
4
5  trX = np.linspace(-1, 1, 101)
6  trY = 2 * trX + np.random.randn(*trX.shape) * 0.33
7
8  X = T.scalar()
9  Y = T.scalar()
10
11 def model(X, w):
12     return X * w
13
14 w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
15 y = model(X, w)
16
17 cost = T.mean(T.sqr(y - Y))
18 gradient = T.grad(cost=cost, wrt=w)
19 updates = [[w, w - gradient * 0.01]]
20
21 train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)
22
23 for i in range(100):
24     for x, y in zip(trX, trY):
25         train(x, y)
26
```

imports

training data generation

symbolic variable initialization

our model

model parameter initialization

metric to be optimized by model

learning signal for parameter(s)

how to change parameter based on learning signal

Theano

```
1  import theano
2  from theano import tensor as T
3  import numpy as np
4
5  trX = np.linspace(-1, 1, 101)
6  trY = 2 * trX + np.random.randn(*trX.shape) * 0.33
7
8  X = T.scalar()
9  Y = T.scalar()
10
11 def model(X, w):
12     return X * w
13
14 w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
15 y = model(X, w)
16
17 cost = T.mean(T.sqr(y - Y))
18 gradient = T.grad(cost=cost, wrt=w)
19 updates = [[w, w - gradient * 0.01]]
20
21 train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)
22
23 for i in range(100):
24     for x, y in zip(trX, trY):
25         train(x, y)
26
```

imports

training data generation

symbolic variable initialization

our model

model parameter initialization

metric to be optimized by model
learning signal for parameter(s)
how to change parameter based on learning signal

compiling to a python function

Theano

```
1  import theano
2  from theano import tensor as T
3  import numpy as np
4
5  trX = np.linspace(-1, 1, 101)
6  trY = 2 * trX + np.random.randn(*trX.shape) * 0.33
7
8  X = T.scalar()
9  Y = T.scalar()
10
11 def model(X, w):
12     return X * w
13
14 w = theano.shared(np.asarray(0., dtype=theano.config.floatX))
15 y = model(X, w)
16
17 cost = T.mean(T.sqr(y - Y))
18 gradient = T.grad(cost=cost, wrt=w)
19 updates = [[w, w - gradient * 0.01]]
20
21 train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)
22
23 for i in range(100):
24     for x, y in zip(trX, trY):
25         train(x, y)
26
```

imports

training data generation

symbolic variable initialization

our model

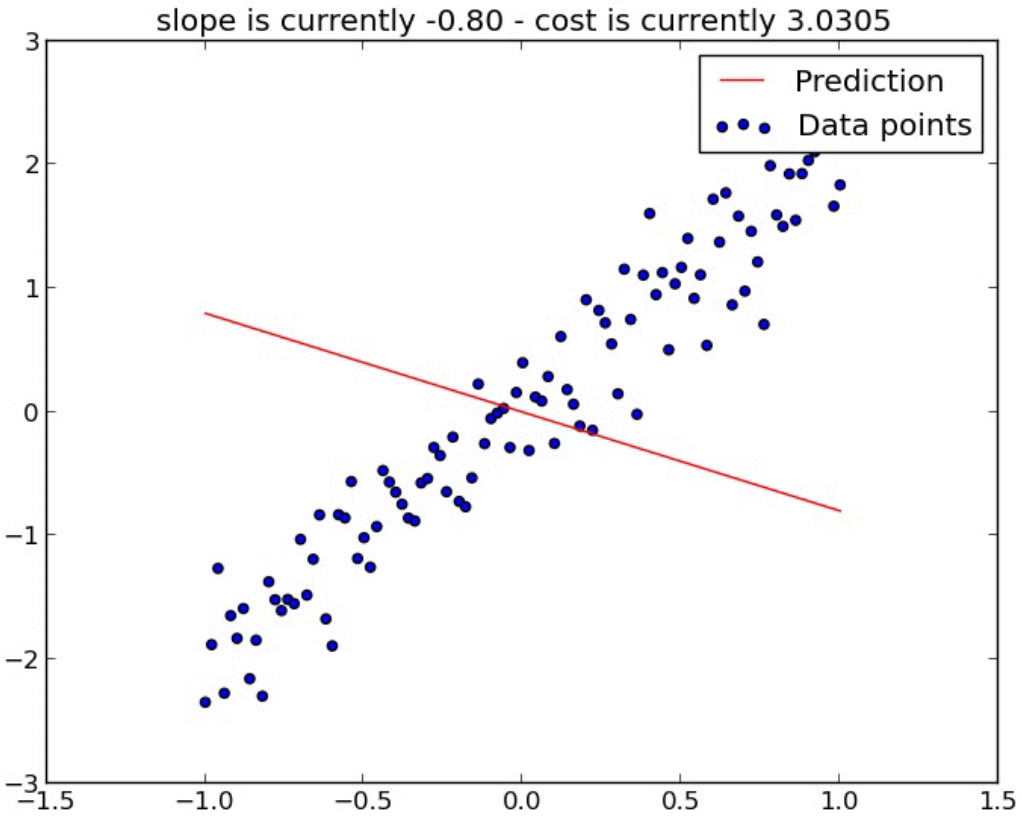
model parameter initialization

metric to be optimized by model
learning signal for parameter(s)
how to change parameter based on learning signal

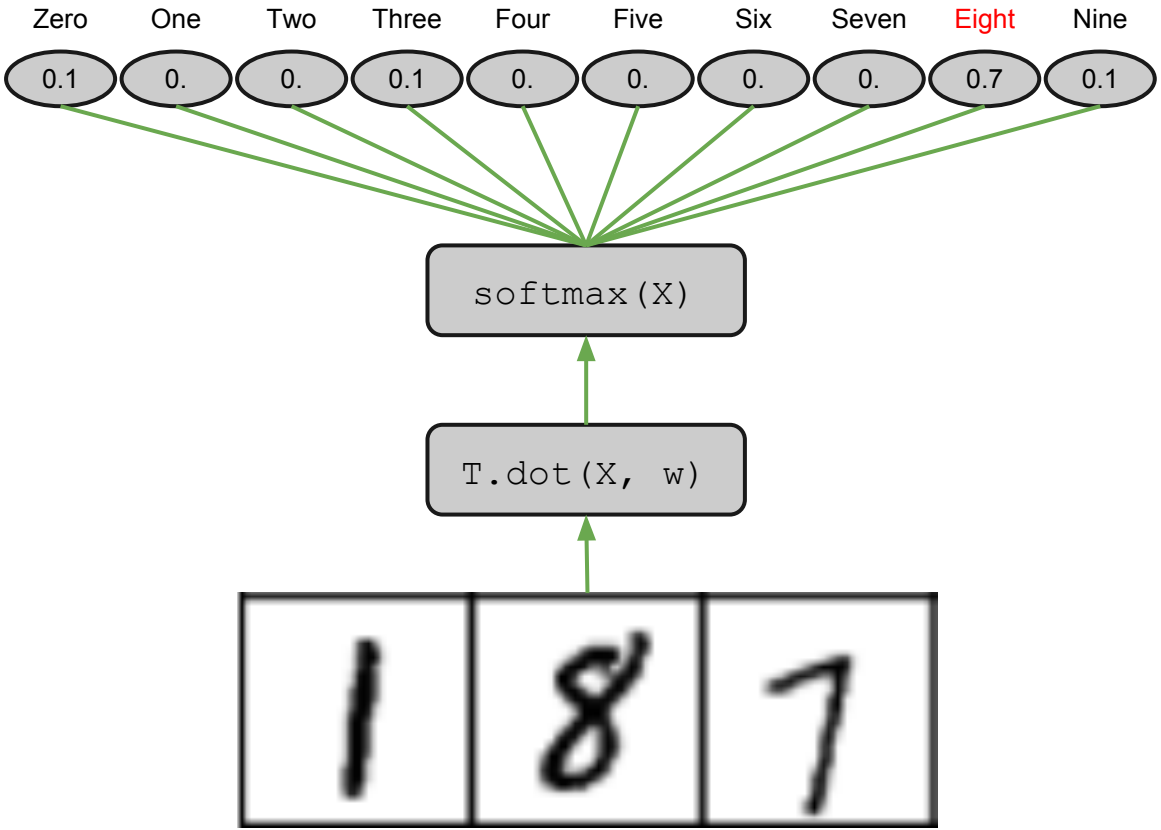
compiling to a python function

iterate through data 100 times and train model
on each example of input, output pairs

Theano



Theano doing its thing



Logistic Regression

```python
import theano
from theano import tensor as T
import numpy as np
from load import mnist

def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def model(X, w):
    return T.nnet.softmax(T.dot(X, w))

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w = init_weights((784, 10))

py_x = model(X, w)
y_pred = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
gradient = T.grad(cost=cost, wrt=w)
update = [[w, w - gradient * 0.05]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=update, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_pred, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print i, np.mean(np.argmax(teY, axis=1) == predict(teX))

```

Back to Theano

```python
import theano
from theano import tensor as T
import numpy as np
from load import mnist

def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def model(X, w):
    return T.nnet.softmax(T.dot(X, w))

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w = init_weights((784, 10))

py_x = model(X, w)
y_pred = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
gradient = T.grad(cost=cost, wrt=w)
update = [[w, w - gradient * 0.05]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=update, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_pred, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print i, np.mean(np.argmax(teY, axis=1) == predict(teX))

```

convert to correct dtype

Back to Theano

```python
import theano
from theano import tensor as T
import numpy as np
from load import mnist

def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def model(X, w):
    return T.nnet.softmax(T.dot(X, w))

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w = init_weights((784, 10))

py_x = model(X, w)
y_pred = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
gradient = T.grad(cost=cost, wrt=w)
update = [[w, w - gradient * 0.05]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=update, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_pred, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print i, np.mean(np.argmax(teY, axis=1) == predict(teX))

```

convert to correct dtype

initialize model parameters

Back to Theano

```python
import theano
from theano import tensor as T
import numpy as np
from load import mnist

def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def model(X, w):
    return T.nnet.softmax(T.dot(X, w))

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w = init_weights((784, 10))

py_x = model(X, w)
y_pred = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
gradient = T.grad(cost=cost, wrt=w)
update = [[w, w - gradient * 0.05]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=update, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_pred, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print i, np.mean(np.argmax(teY, axis=1) == predict(teX))

```

convert to correct dtype

initialize model parameters

our model in matrix format

Back to Theano

```python
import theano
from theano import tensor as T
import numpy as np
from load import mnist

def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def model(X, w):
    return T.nnet.softmax(T.dot(X, w))

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w = init_weights((784, 10))

py_x = model(X, w)
y_pred = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
gradient = T.grad(cost=cost, wrt=w)
update = [[w, w - gradient * 0.05]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=update, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_pred, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print i, np.mean(np.argmax(teY, axis=1) == predict(teX))
```

convert to correct dtype

initialize model parameters

our model in matrix format

loading data matrices

Back to Theano

```python
import theano
from theano import tensor as T
import numpy as np
from load import mnist

def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def model(X, w):
    return T.nnet.softmax(T.dot(X, w))

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w = init_weights((784, 10))

py_x = model(X, w)
y_pred = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
gradient = T.grad(cost=cost, wrt=w)
update = [[w, w - gradient * 0.05]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=update, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_pred, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print i, np.mean(np.argmax(teY, axis=1) == predict(teX))
```

convert to correct dtype

initialize model parameters

our model in matrix format

loading data matrices

now matrix types

Back to Theano

```python
import theano
from theano import tensor as T
import numpy as np
from load import mnist

def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def model(X, w):
    return T.nnet.softmax(T.dot(X, w))

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w = init_weights((784, 10))

py_x = model(X, w)
y_pred = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
gradient = T.grad(cost=cost, wrt=w)
update = [[w, w - gradient * 0.05]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=update, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_pred, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print i, np.mean(np.argmax(teY, axis=1) == predict(teX))
```

convert to correct dtype

initialize model parameters

our model in matrix format

loading data matrices

now matrix types

probability outputs and maxima predictions

Back to Theano

```python
import theano
from theano import tensor as T
import numpy as np
from load import mnist

def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def model(X, w):
    return T.nnet.softmax(T.dot(X, w))

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w = init_weights((784, 10))

py_x = model(X, w)
y_pred = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
gradient = T.grad(cost=cost, wrt=w)
update = [[w, w - gradient * 0.05]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=update, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_pred, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print i, np.mean(np.argmax(teY, axis=1) == predict(teX))
```

convert to correct dtype

initialize model parameters

our model in matrix format

loading data matrices

now matrix types

probability outputs and maxima predictions

classification metric to optimize

Back to Theano

```python
import theano
from theano import tensor as T
import numpy as np
from load import mnist

def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def model(X, w):
    return T.nnet.softmax(T.dot(X, w))

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w = init_weights((784, 10))

py_x = model(X, w)
y_pred = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
gradient = T.grad(cost=cost, wrt=w)
update = [[w, w - gradient * 0.05]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=update, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_pred, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print i, np.mean(np.argmax(teY, axis=1) == predict(teX))

```

convert to correct dtype

initialize model parameters

our model in matrix format

loading data matrices

now matrix types

probability outputs and maxima predictions

classification metric to optimize

compile prediction function

Back to Theano

```python
import theano
from theano import tensor as T
import numpy as np
from load import mnist

def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def model(X, w):
    return T.nnet.softmax(T.dot(X, w))

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w = init_weights((784, 10))

py_x = model(X, w)
y_pred = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
gradient = T.grad(cost=cost, wrt=w)
update = [[w, w - gradient * 0.05]]

train = theano.function(inputs=[X, Y], outputs=cost, updates=update, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_pred, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print i, np.mean(np.argmax(teY, axis=1) == predict(teX))
```

convert to correct dtype

initialize model parameters

our model in matrix format

loading data matrices

now matrix types

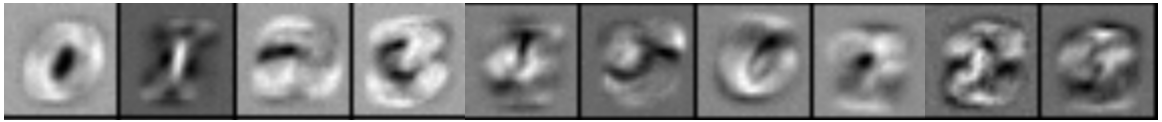
probability outputs and maxima predictions

classification metric to optimize

compile prediction function

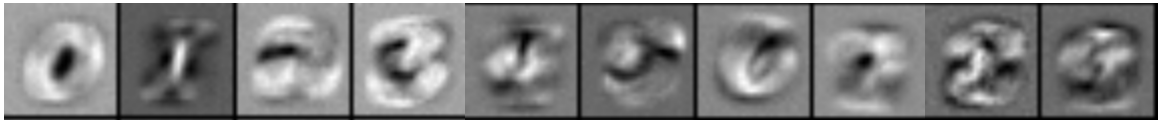
train on mini-batches of 128 examples

Back to Theano



0 1 2 3 4 5 6 7 8 9

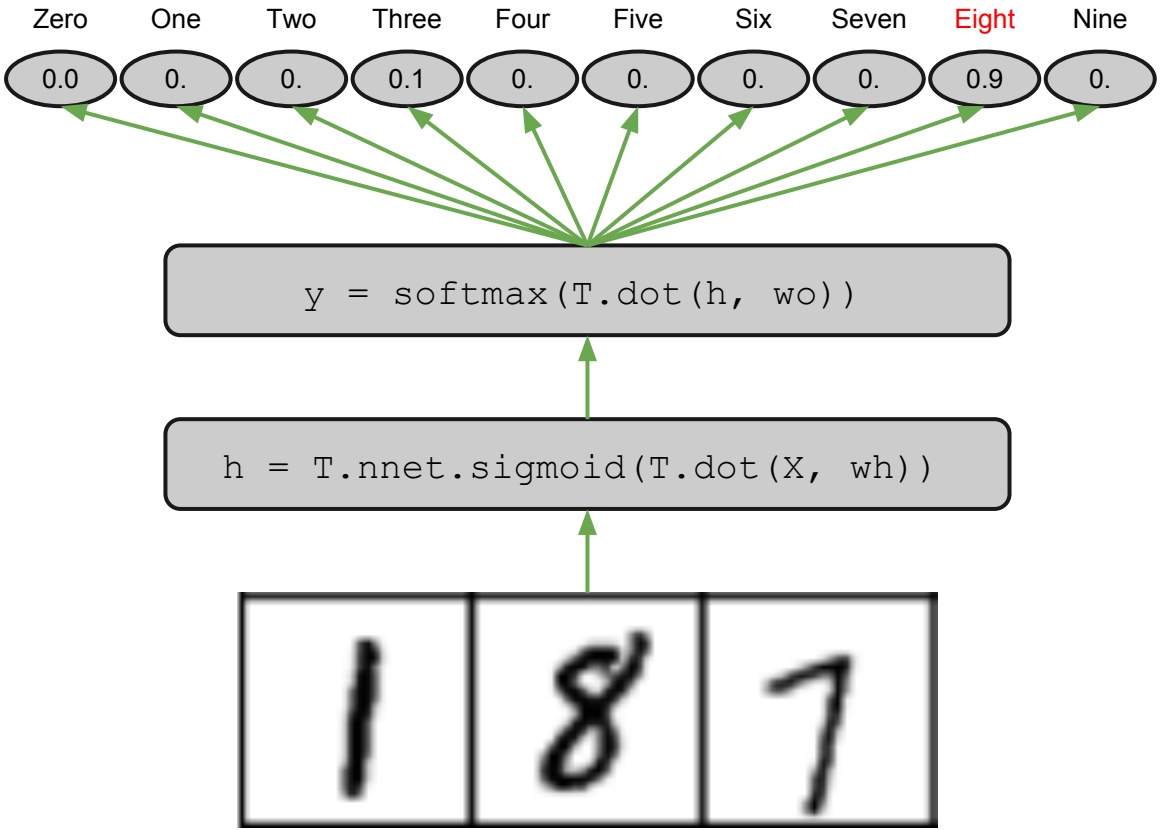
What it learns



0 1 2 3 4 5 6 7 8 9

Test Accuracy: 92.5%

What it learns



An “old” net (circa 2000)

```
def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def sgd(cost, params, lr=0.05):
    grads = T.grad(cost=cost, wrt=params)
    updates = []
    for p, g in zip(params, grads):
        updates.append([p, p - g * lr])
    return updates

def model(X, w_h, w_o):
    h = T.nnet.sigmoid(T.dot(X, w_h))
    pyx = T.nnet.softmax(T.dot(h, w_o))
    return pyx

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w_h = init_weights((784, 625))
w_o = init_weights((625, 10))

py_x = model(X, w_h, w_o)
y_x = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
params = [w_h, w_o]
updates = sgd(cost, params)

train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_x, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print np.mean(np.argmax(teY, axis=1) == predict(teX))
```

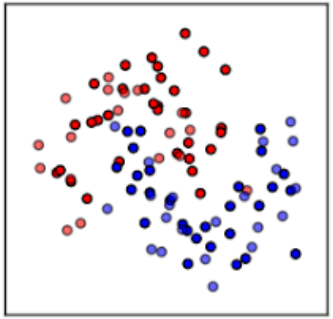
A “old” net in Theano

```python
def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def sgd(cost, params, lr=0.05):
    grads = T.grad(cost=cost, wrt=params)
    updates = []
    for p, g in zip(params, grads):
        updates.append([p, p - g * lr])
    return updates

def model(X, w_h, w_o):
    h = T.nnet.sigmoid(T.dot(X, w_h))
    pyx = T.nnet.softmax(T.dot(h, w_o))
    return pyx

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w_h = init_weights((784, 625))
w_o = init_weights((625, 10))

py_x = model(X, w_h, w_o)
y_x = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
params = [w_h, w_o]
updates = sgd(cost, params)

train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_x, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print np.mean(np.argmax(teY, axis=1) == predict(teX))
```

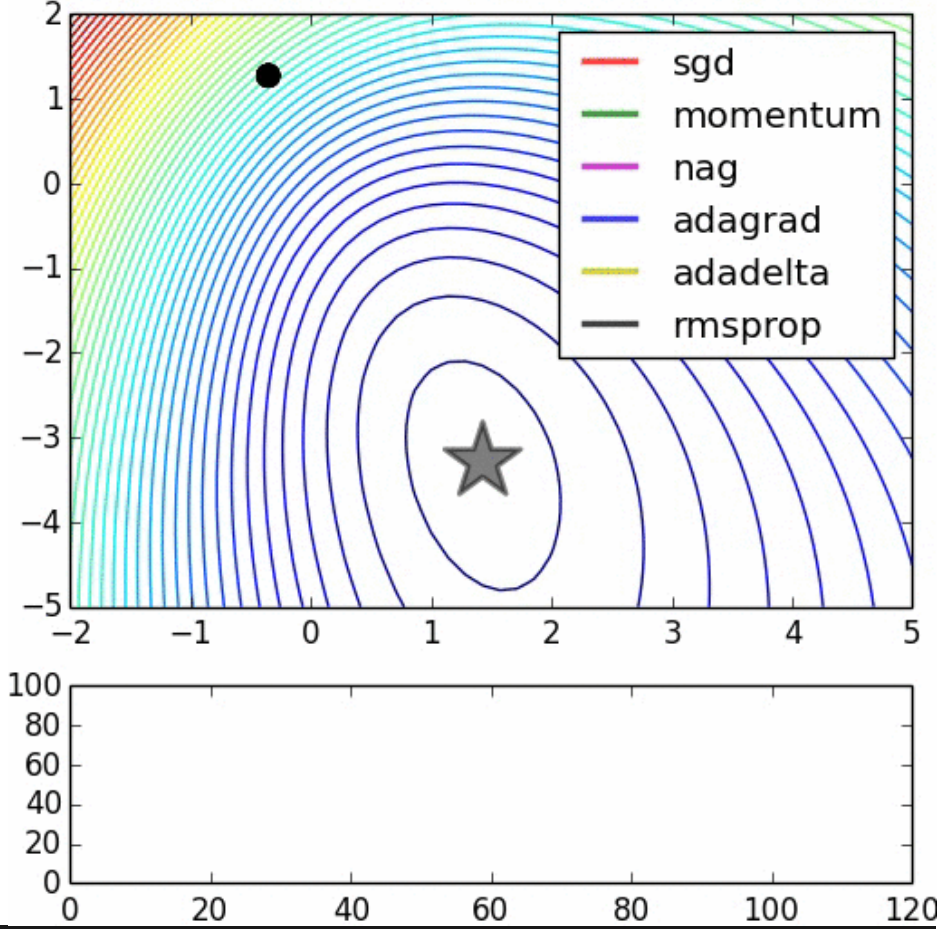
generalize to compute gradient descent on all model parameters

A “old” net in Theano

Understanding SGD



2D moons dataset
courtesy of scikit-learn

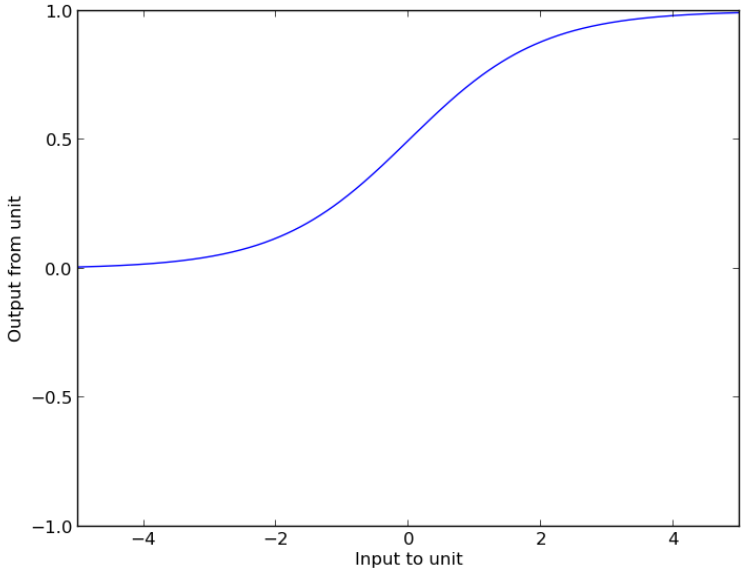


```python
def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def sgd(cost, params, lr=0.05):
    grads = T.grad(cost=cost, wrt=params)
    updates = []
    for p, g in zip(params, grads):
        updates.append([p, p - g * lr])
    return updates

def model(X, w_h, w_o):
    h = T.nnet.sigmoid(T.dot(X, w_h))
    pyx = T.nnet.softmax(T.dot(h, w_o))
    return pyx

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w_h = init_weights((784, 625))
w_o = init_weights((625, 10))

py_x = model(X, w_h, w_o)
y_x = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
params = [w_h, w_o]
updates = sgd(cost, params)

train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_x, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print np.mean(np.argmax(teY, axis=1) == predict(teX))
```

generalize to compute gradient descent on all model parameters

2 layers of computation
input -> hidden (sigmoid)
hidden -> output (softmax)

A “old” net in Theano



Understanding Sigmoid Units

```python
def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def sgd(cost, params, lr=0.05):
    grads = T.grad(cost=cost, wrt=params)
    updates = []
    for p, g in zip(params, grads):
        updates.append([p, p - g * lr])
    return updates

def model(X, w_h, w_o):
    h = T.nnet.sigmoid(T.dot(X, w_h))
    pyx = T.nnet.softmax(T.dot(h, w_o))
    return pyx

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w_h = init_weights((784, 625))
w_o = init_weights((625, 10))

py_x = model(X, w_h, w_o)
y_x = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
params = [w_h, w_o]
updates = sgd(cost, params)

train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_x, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print np.mean(np.argmax(teY, axis=1) == predict(teX))
```

generalize to compute gradient descent on all model parameters

2 layers of computation
input -> hidden (sigmoid)
hidden -> output (softmax)

initialize both weight matrices

A “old” net in Theano

```python
def floatX(X):
    return np.asarray(X, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.01))

def sgd(cost, params, lr=0.05):
    grads = T.grad(cost=cost, wrt=params)
    updates = []
    for p, g in zip(params, grads):
        updates.append([p, p - g * lr])
    return updates

def model(X, w_h, w_o):
    h = T.nnet.sigmoid(T.dot(X, w_h))
    pyx = T.nnet.softmax(T.dot(h, w_o))
    return pyx

trX, teX, trY, teY = mnist(onehot=True)

X = T.fmatrix()
Y = T.fmatrix()

w_h = init_weights((784, 625))
w_o = init_weights((625, 10))

py_x = model(X, w_h, w_o)
y_x = T.argmax(py_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(py_x, Y))
params = [w_h, w_o]
updates = sgd(cost, params)

train = theano.function(inputs=[X, Y], outputs=cost, updates=updates, allow_input_downcast=True)
predict = theano.function(inputs=[X], outputs=y_x, allow_input_downcast=True)

for i in range(100):
    for start, end in zip(range(0, len(trX), 128), range(128, len(trX), 128)):
        cost = train(trX[start:end], trY[start:end])
    print np.mean(np.argmax(teY, axis=1) == predict(teX))
```

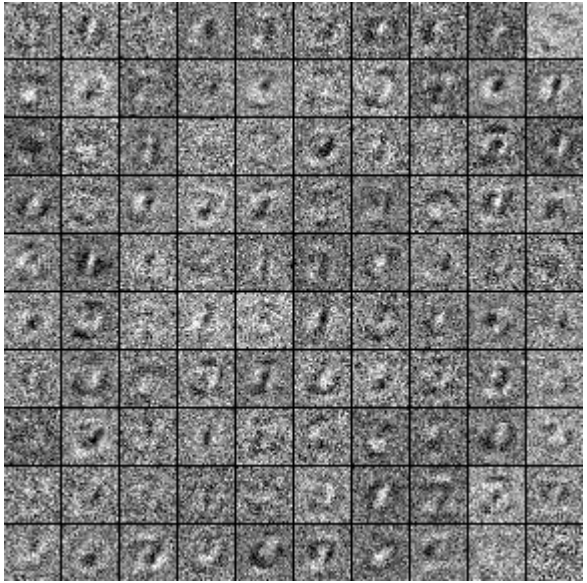
generalize to compute gradient descent on all model parameters

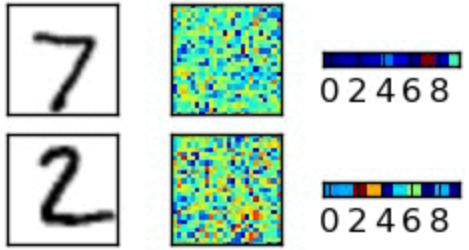
2 layers of computation
input -> hidden (sigmoid)
hidden -> output (softmax)

initialize both weight matrices

updated version of updates

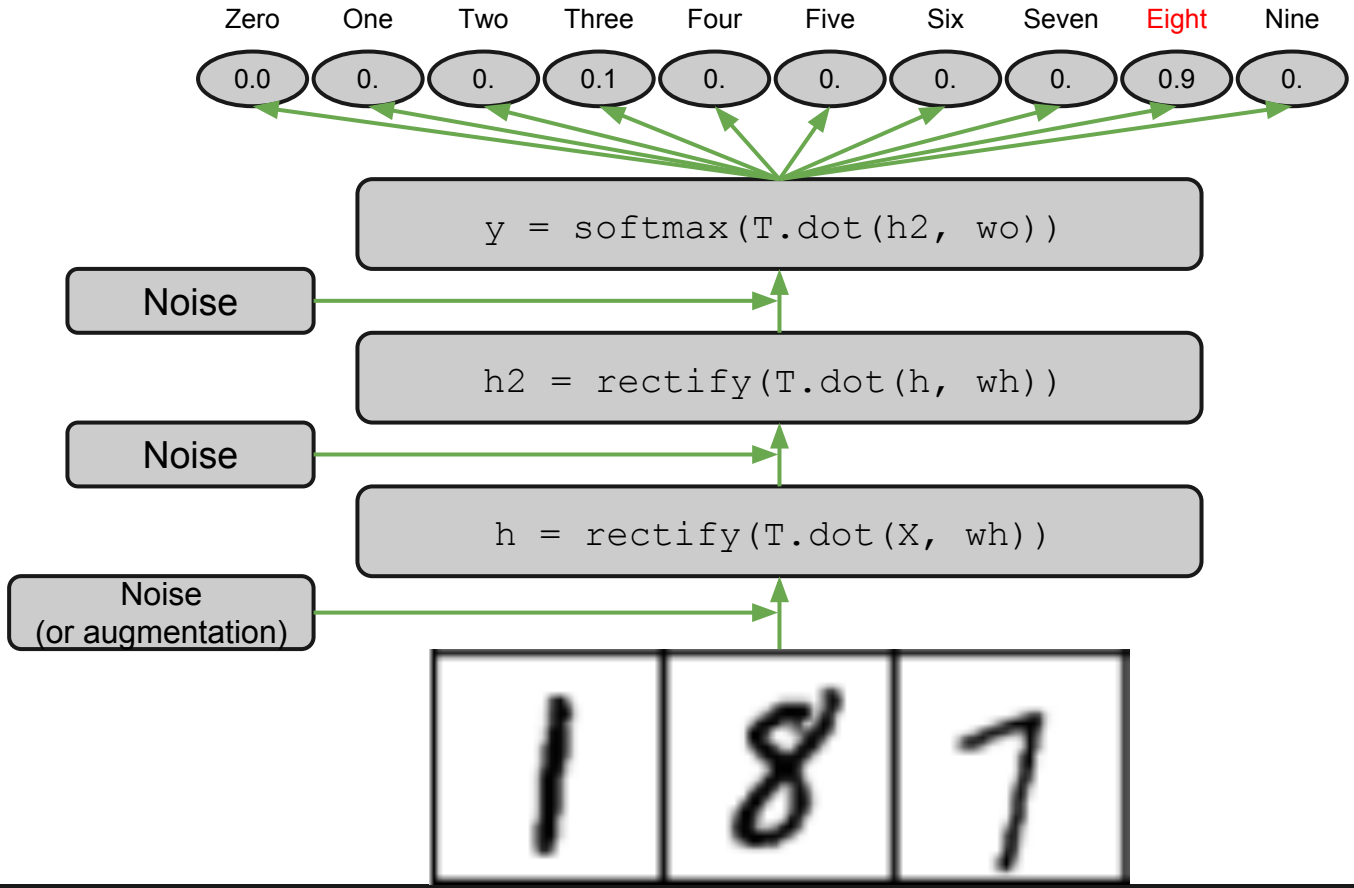
A “old” net in Theano





Test Accuracy: 98.4%

What an “old” net learns



A “modern” net - 2012+

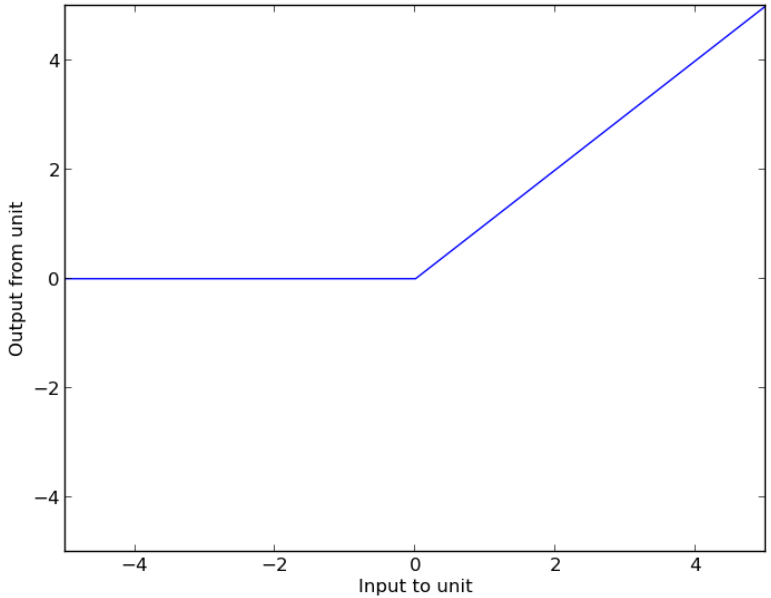
```python

def rectify(X):
    return T.maximum(X, 0.)

def softmax(X):
    e_x = T.exp(X - X.max(axis=1).dimshuffle(0, 'x'))
    return e_x / e_x.sum(axis=1).dimshuffle(0, 'x')

def RMSprop(cost, params, lr=0.001, rho=0.9, epsilon=1e-6):
    grads = T.grad(cost=cost, wrt=params)
    updates = []
    for p, g in zip(params, grads):
        acc = theano.shared(p.get_value() * 0.)
        acc_new = rho * acc + (1 - rho) * g ** 2
        gradient_scaling = T.sqrt(acc_new + epsilon)
        g = g / gradient_scaling
        updates.append((acc, acc_new))
        updates.append((p, p - lr * g))
    return updates

def dropout(X, p=0.):
    if p > 0:
        retain_prob = 1 - p
        X *= srng.binomial(X.shape, p=retain_prob, dtype=theano.config.floatX)
        X /= retain_prob
    return X

def model(X, w_h, w_h2, w_o, p_drop_input, p_drop_hidden):
    X = dropout(X, p_drop_input)
    h = rectify(T.dot(X, w_h))

    h = dropout(h, p_drop_hidden)
    h2 = rectify(T.dot(h, w_h2))

    h2 = dropout(h2, p_drop_hidden)
    py_x = softmax(T.dot(h2, w_o))
    return h, h2, py_x

```

A “modern” net in Theano

```python

def rectify(X):
    return T.maximum(X, 0.)

def softmax(X):
    e_x = T.exp(X - X.max(axis=1).dimshuffle(0, 'x'))
    return e_x / e_x.sum(axis=1).dimshuffle(0, 'x')

def RMSprop(cost, params, lr=0.001, rho=0.9, epsilon=1e-6):
    grads = T.grad(cost=cost, wrt=params)
    updates = []
    for p, g in zip(params, grads):
        acc = theano.shared(p.get_value() * 0.)
        acc_new = rho * acc + (1 - rho) * g ** 2
        gradient_scaling = T.sqrt(acc_new + epsilon)
        g = g / gradient_scaling
        updates.append((acc, acc_new))
        updates.append((p, p - lr * g))
    return updates

def dropout(X, p=0.):
    if p > 0:
        retain_prob = 1 - p
        X *= srng.binomial(X.shape, p=retain_prob, dtype=theano.config.floatX)
        X /= retain_prob
    return X

def model(X, w_h, w_h2, w_o, p_drop_input, p_drop_hidden):
    X = dropout(X, p_drop_input)
    h = rectify(T.dot(X, w_h))

    h = dropout(h, p_drop_hidden)
    h2 = rectify(T.dot(h, w_h2))

    h2 = dropout(h2, p_drop_hidden)
    py_x = softmax(T.dot(h2, w_o))
    return h, h2, py_x

```

rectifier

A “modern” net in Theano



Understanding rectifier units

```python
def rectify(X):
    return T.maximum(X, 0.)

def softmax(X):
    e_x = T.exp(X - X.max(axis=1).dimshuffle(0, 'x'))
    return e_x / e_x.sum(axis=1).dimshuffle(0, 'x')

def RMSprop(cost, params, lr=0.001, rho=0.9, epsilon=1e-6):
    grads = T.grad(cost=cost, wrt=params)
    updates = []
    for p, g in zip(params, grads):
        acc = theano.shared(p.get_value() * 0.)
        acc_new = rho * acc + (1 - rho) * g ** 2
        gradient_scaling = T.sqrt(acc_new + epsilon)
        g = g / gradient_scaling
        updates.append((acc, acc_new))
        updates.append((p, p - lr * g))
    return updates

def dropout(X, p=0.):
    if p > 0:
        retain_prob = 1 - p
        X *= srng.binomial(X.shape, p=retain_prob, dtype=theano.config.floatX)
        X /= retain_prob
    return X

def model(X, w_h, w_h2, w_o, p_drop_input, p_drop_hidden):
    X = dropout(X, p_drop_input)
    h = rectify(T.dot(X, w_h))

    h = dropout(h, p_drop_hidden)
    h2 = rectify(T.dot(h, w_h2))

    h2 = dropout(h2, p_drop_hidden)
    py_x = softmax(T.dot(h2, w_o))
    return h, h2, py_x
```

rectifier

numerically stable softmax

A “modern” net in Theano

```python
def rectify(X):
    return T.maximum(X, 0.)

def softmax(X):
    e_x = T.exp(X - X.max(axis=1).dimshuffle(0, 'x'))
    return e_x / e_x.sum(axis=1).dimshuffle(0, 'x')

def RMSprop(cost, params, lr=0.001, rho=0.9, epsilon=1e-6):
    grads = T.grad(cost=cost, wrt=params)
    updates = []
    for p, g in zip(params, grads):
        acc = theano.shared(p.get_value() * 0.)
        acc_new = rho * acc + (1 - rho) * g ** 2
        gradient_scaling = T.sqrt(acc_new + epsilon)
        g = g / gradient_scaling
        updates.append((acc, acc_new))
        updates.append((p, p - lr * g))
    return updates

def dropout(X, p=0.):
    if p > 0:
        retain_prob = 1 - p
        X *= srng.binomial(X.shape, p=retain_prob, dtype=theano.config.floatX)
        X /= retain_prob
    return X

def model(X, w_h, w_h2, w_o, p_drop_input, p_drop_hidden):
    X = dropout(X, p_drop_input)
    h = rectify(T.dot(X, w_h))

    h = dropout(h, p_drop_hidden)
    h2 = rectify(T.dot(h, w_h2))

    h2 = dropout(h2, p_drop_hidden)
    py_x = softmax(T.dot(h2, w_o))
    return h, h2, py_x
```

rectifier

numerically stable softmax

a running average of the magnitude of the gradient

A “modern” net in Theano

```python
def rectify(X):
    return T.maximum(X, 0.)

def softmax(X):
    e_x = T.exp(X - X.max(axis=1).dimshuffle(0, 'x'))
    return e_x / e_x.sum(axis=1).dimshuffle(0, 'x')

def RMSprop(cost, params, lr=0.001, rho=0.9, epsilon=1e-6):
    grads = T.grad(cost=cost, wrt=params)
    updates = []
    for p, g in zip(params, grads):
        acc = theano.shared(p.get_value() * 0.)
        acc_new = rho * acc + (1 - rho) * g ** 2
        gradient_scaling = T.sqrt(acc_new + epsilon)
        g = g / gradient_scaling
        updates.append((acc, acc_new))
        updates.append((p, p - lr * g))
    return updates

def dropout(X, p=0.):
    if p > 0:
        retain_prob = 1 - p
        X *= srng.binomial(X.shape, p=retain_prob, dtype=theano.config.floatX)
        X /= retain_prob
    return X

def model(X, w_h, w_h2, w_o, p_drop_input, p_drop_hidden):
    X = dropout(X, p_drop_input)
    h = rectify(T.dot(X, w_h))

    h = dropout(h, p_drop_hidden)
    h2 = rectify(T.dot(h, w_h2))

    h2 = dropout(h2, p_drop_hidden)
    py_x = softmax(T.dot(h2, w_o))
    return h, h2, py_x
```

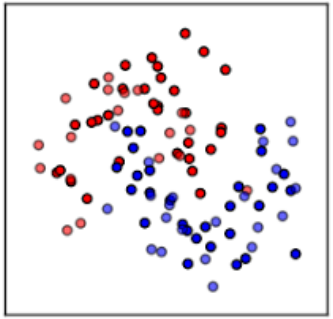
rectifier

numerically stable softmax

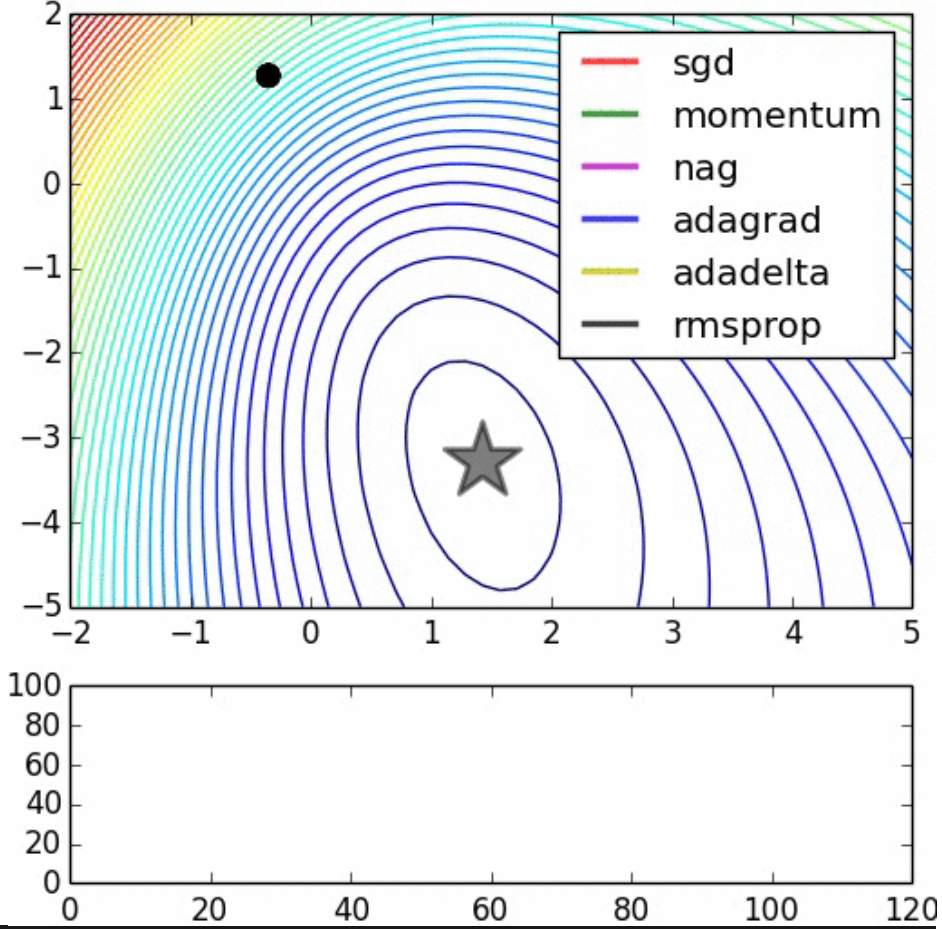
a running average of the magnitude of the gradient

scale the gradient based on running average

A “modern” net in Theano



2D moons dataset
courtesy of scikit-learn



Understanding RMSprop

```
14
15 def rectify(X):
16     return T.maximum(X, 0.)
17
18 def softmax(X):
19     e_x = T.exp(X - X.max(axis=1).dimshuffle(0, 'x'))
20     return e_x / e_x.sum(axis=1).dimshuffle(0, 'x')
21
22 def RMSprop(cost, params, lr=0.001, rho=0.9, epsilon=1e-6):
23     grads = T.grad(cost=cost, wrt=params)
24     updates = []
25     for p, g in zip(params, grads):
26         acc = theano.shared(p.get_value() * 0.)
27         acc_new = rho * acc + (1 - rho) * g ** 2
28         gradient_scaling = T.sqrt(acc_new + epsilon)
29         g = g / gradient_scaling
30         updates.append((acc, acc_new))
31         updates.append((p, p - lr * g))
32     return updates
33
34 def dropout(X, p=0.):
35     if p > 0:
36         retain_prob = 1 - p
37         X *= srng.binomial(X.shape, p=retain_prob, dtype=theano.config.floatX)
38         X /= retain_prob
39     return X
40
41 def model(X, w_h, w_h2, w_o, p_drop_input, p_drop_hidden):
42     X = dropout(X, p_drop_input)
43     h = rectify(T.dot(X, w_h))
44
45     h = dropout(h, p_drop_hidden)
46     h2 = rectify(T.dot(h, w_h2))
47
48     h2 = dropout(h2, p_drop_hidden)
49     py_x = softmax(T.dot(h2, w_o))
50     return h, h2, py_x
51
```

rectifier

numerically stable softmax

a running average of the magnitude of the gradient

scale the gradient based on running average

randomly drop values and scale rest

A “modern” net in Theano

```
14
15 def rectify(X):
16     return T.maximum(X, 0.)
17
18 def softmax(X):
19     e_x = T.exp(X - X.max(axis=1).dimshuffle(0, 'x'))
20     return e_x / e_x.sum(axis=1).dimshuffle(0, 'x')
21
22 def RMSprop(cost, params, lr=0.001, rho=0.9, epsilon=1e-6):
23     grads = T.grad(cost=cost, wrt=params)
24     updates = []
25     for p, g in zip(params, grads):
26         acc = theano.shared(p.get_value() * 0.)
27         acc_new = rho * acc + (1 - rho) * g ** 2
28         gradient_scaling = T.sqrt(acc_new + epsilon)
29         g = g / gradient_scaling
30         updates.append((acc, acc_new))
31         updates.append((p, p - lr * g))
32     return updates
33
34 def dropout(X, p=0.):
35     if p > 0:
36         retain_prob = 1 - p
37         X *= srng.binomial(X.shape, p=retain_prob, dtype=theano.config.floatX)
38         X /= retain_prob
39     return X
40
41 def model(X, w_h, w_h2, w_o, p_drop_input, p_drop_hidden):
42     X = dropout(X, p_drop_input)
43     h = rectify(T.dot(X, w_h))
44
45     h = dropout(h, p_drop_hidden)
46     h2 = rectify(T.dot(h, w_h2))
47
48     h2 = dropout(h2, p_drop_hidden)
49     py_x = softmax(T.dot(h2, w_o))
50     return h, h2, py_x
51
```

rectifier

numerically stable softmax

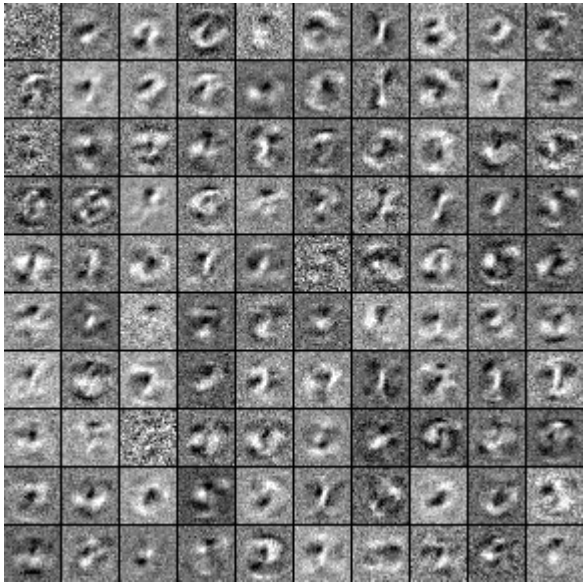
a running average of the magnitude of the gradient

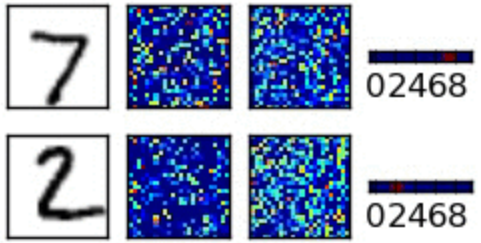
scale the gradient based on running average

randomly drop values and scale rest

Noise injected into model
rectifiers now used
2 hidden layers

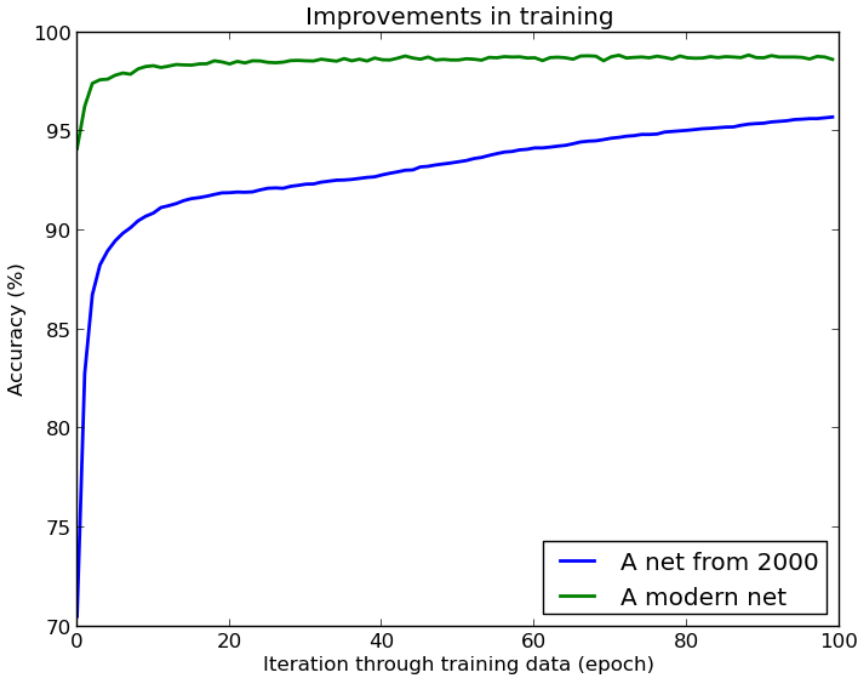
A “modern” net in Theano



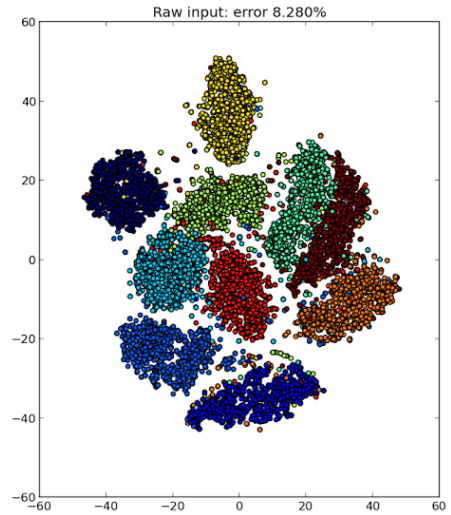


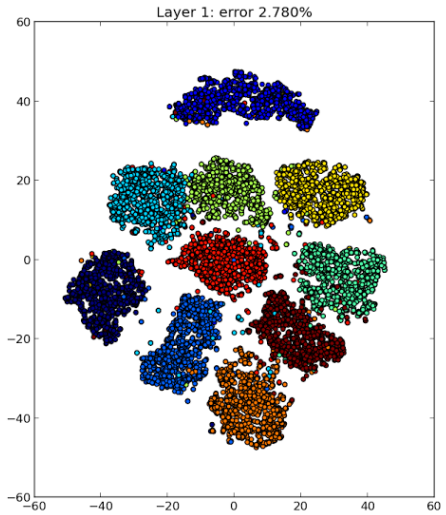
Test Accuracy: 99.0%

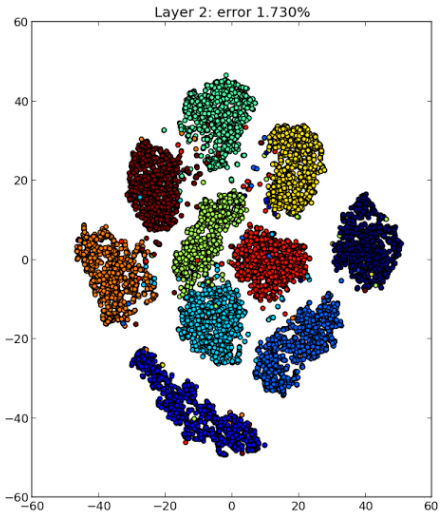
What a “modern” net learns

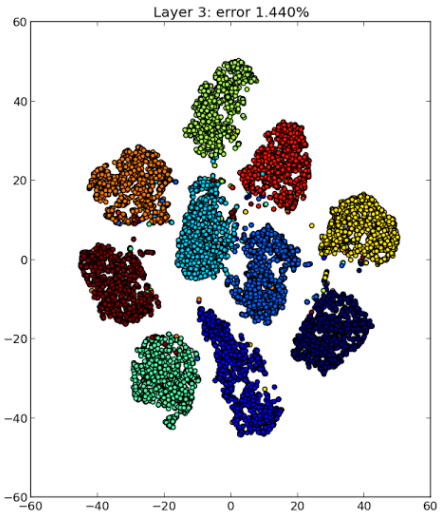


Quantifying the difference

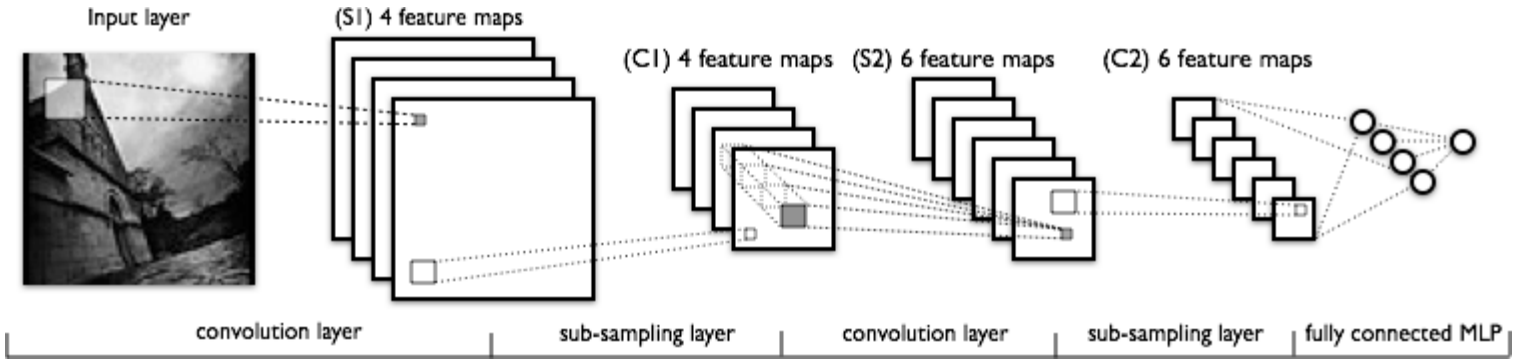








What a “modern” net is doing



from deeplearning.net

Convolutional Networks

```
def model(X, w, w2, w3, w4, p_drop_conv, p_drop_hidden):
    l1a = rectify(conv2d(X, w, border_mode='full'))
    l1 = max_pool_2d(l1a, (2, 2))
    l1 = dropout(l1, p_drop_conv)

    l2a = rectify(conv2d(l1, w2))
    l2 = max_pool_2d(l2a, (2, 2))
    l2 = dropout(l2, p_drop_conv)

    l3a = rectify(conv2d(l2, w3))
    l3b = max_pool_2d(l3a, (2, 2))
    l3 = T.flatten(l3b, outdim=2)
    l3 = dropout(l3, p_drop_conv)

    l4 = rectify(T.dot(l3, w4))
    l4 = dropout(l4, p_drop_hidden)

    pyx = softmax(T.dot(l4, w_o))
    return l1, l2, l3, l4, pyx

trX, teX, trY, teY = mnist(onehot=True)

trX = trX.reshape(-1, 1, 28, 28)
teX = teX.reshape(-1, 1, 28, 28)

X = T.ftensor4()
Y = T.fmatrix()

w = init_weights((32, 1, 3, 3))
w2 = init_weights((64, 32, 3, 3))
w3 = init_weights((128, 64, 3, 3))
w4 = init_weights((128 * 3 * 3, 625))
w_o = init_weights((625, 10))

noise_l1, noise_l2, noise_l3, noise_l4, noise_py_x = model(X, w, w2, w3, w4, 0.2, 0.5)
l1, l2, l3, l4, py_x = model(X, w, w2, w3, w4, 0., 0.)
y_x = T.argmax(py_x, axis=1)
```

A convolutional network in Theano

```python
def model(X, w, w2, w3, w4, p_drop_conv, p_drop_hidden):
    l1a = rectify(conv2d(X, w, border_mode='full'))
    l1 = max_pool_2d(l1a, (2, 2))
    l1 = dropout(l1, p_drop_conv)

    l2a = rectify(conv2d(l1, w2))
    l2 = max_pool_2d(l2a, (2, 2))
    l2 = dropout(l2, p_drop_conv)

    l3a = rectify(conv2d(l2, w3))
    l3b = max_pool_2d(l3a, (2, 2))
    l3 = T.flatten(l3b, outdim=2)
    l3 = dropout(l3, p_drop_conv)

    l4 = rectify(T.dot(l3, w4))
    l4 = dropout(l4, p_drop_hidden)

    pyx = softmax(T.dot(l4, w_o))
    return l1, l2, l3, l4, pyx

trX, teX, trY, teY = mnist(onehot=True)

trX = trX.reshape(-1, 1, 28, 28)
teX = teX.reshape(-1, 1, 28, 28)

X = T.ftensor4()
Y = T.fmatrix()

w = init_weights((32, 1, 3, 3))
w2 = init_weights((64, 32, 3, 3))
w3 = init_weights((128, 64, 3, 3))
w4 = init_weights((128 * 3 * 3, 625))
w_o = init_weights((625, 10))

noise_l1, noise_l2, noise_l3, noise_l4, noise_py_x = model(X, w, w2, w3, w4, 0.2, 0.5)
l1, l2, l3, l4, py_x = model(X, w, w2, w3, w4, 0., 0.)
y_x = T.argmax(py_x, axis=1)
```

a “block” of computation conv -> activate -> pool -> noise

A convolutional network in Theano

```
46 def model(X, w, w2, w3, w4, p_drop_conv, p_drop_hidden):
47     l1a = rectify(conv2d(X, w, border_mode='full'))
48     l1 = max_pool_2d(l1a, (2, 2))
49     l1 = dropout(l1, p_drop_conv)
50
51     l2a = rectify(conv2d(l1, w2))
52     l2 = max_pool_2d(l2a, (2, 2))
53     l2 = dropout(l2, p_drop_conv)
54
55     l3a = rectify(conv2d(l2, w3))
56     l3b = max_pool_2d(l3a, (2, 2))
57     l3 = T.flatten(l3b, outdim=2)
58     l3 = dropout(l3, p_drop_conv)
59
60     l4 = rectify(T.dot(l3, w4))
61     l4 = dropout(l4, p_drop_hidden)
62
63     pyx = softmax(T.dot(l4, w_o))
64     return l1, l2, l3, l4, pyx
65
66 trX, teX, trY, teY = mnist(onehot=True)
67
68 trX = trX.reshape(-1, 1, 28, 28)
69 teX = teX.reshape(-1, 1, 28, 28)
70
71 X = T.ftensor4()
72 Y = T.fmatrix()
73
74 w = init_weights((32, 1, 3, 3))
75 w2 = init_weights((64, 32, 3, 3))
76 w3 = init_weights((128, 64, 3, 3))
77 w4 = init_weights((128 * 3 * 3, 625))
78 w_o = init_weights((625, 10))
79
80 noise_l1, noise_l2, noise_l3, noise_l4, noise_py_x = model(X, w, w2, w3, w4, 0.2, 0.5)
81 l1, l2, l3, l4, py_x = model(X, w, w2, w3, w4, 0., 0.)
82 y_x = T.argmax(py_x, axis=1)
83
84
```

a “block” of computation conv -> activate -> pool -> noise

convert from 4tensor to normal matrix

A convolutional network in Theano

```
def model(X, w, w2, w3, w4, p_drop_conv, p_drop_hidden):
    l1a = rectify(conv2d(X, w, border_mode='full'))
    l1 = max_pool_2d(l1a, (2, 2))
    l1 = dropout(l1, p_drop_conv)

    l2a = rectify(conv2d(l1, w2))
    l2 = max_pool_2d(l2a, (2, 2))
    l2 = dropout(l2, p_drop_conv)

    l3a = rectify(conv2d(l2, w3))
    l3b = max_pool_2d(l3a, (2, 2))
    l3 = T.flatten(l3b, outdim=2)
    l3 = dropout(l3, p_drop_conv)

    l4 = rectify(T.dot(l3, w4))
    l4 = dropout(l4, p_drop_hidden)

    pyx = softmax(T.dot(l4, w_o))
    return l1, l2, l3, l4, pyx

trX, teX, trY, teY = mnist(onehot=True)

trX = trX.reshape(-1, 1, 28, 28)
teX = teX.reshape(-1, 1, 28, 28)

X = T.ftensor4()
Y = T.fmatrix()

w = init_weights((32, 1, 3, 3))
w2 = init_weights((64, 32, 3, 3))
w3 = init_weights((128, 64, 3, 3))
w4 = init_weights((128 * 3 * 3, 625))
w_o = init_weights((625, 10))

noise_l1, noise_l2, noise_l3, noise_l4, noise_py_x = model(X, w, w2, w3, w4, 0.2, 0.5)
l1, l2, l3, l4, py_x = model(X, w, w2, w3, w4, 0., 0.)
y_x = T.argmax(py_x, axis=1)
```

a “block” of computation conv -> activate -> pool -> noise

convert from 4tensor to normal matrix

reshape into conv 4tensor (b, c, 0, 1) format

A convolutional network in Theano

```
46 def model(X, w, w2, w3, w4, p_drop_conv, p_drop_hidden):
47     l1a = rectify(conv2d(X, w, border_mode='full'))
48     l1 = max_pool_2d(l1a, (2, 2))
49     l1 = dropout(l1, p_drop_conv)
50
51     l2a = rectify(conv2d(l1, w2))
52     l2 = max_pool_2d(l2a, (2, 2))
53     l2 = dropout(l2, p_drop_conv)
54
55     l3a = rectify(conv2d(l2, w3))
56     l3b = max_pool_2d(l3a, (2, 2))
57     l3 = T.flatten(l3b, outdim=2)
58     l3 = dropout(l3, p_drop_conv)
59
60     l4 = rectify(T.dot(l3, w4))
61     l4 = dropout(l4, p_drop_hidden)
62
63     pyx = softmax(T.dot(l4, w_o))
64     return l1, l2, l3, l4, pyx
65
66 trX, teX, trY, teY = mnist(onehot=True)
67
68 trX = trX.reshape(-1, 1, 28, 28)
69 teX = teX.reshape(-1, 1, 28, 28)
70
71 X = T.ftensor4()
72 Y = T.fmatrix()
73
74 w = init_weights((32, 1, 3, 3))
75 w2 = init_weights((64, 32, 3, 3))
76 w3 = init_weights((128, 64, 3, 3))
77 w4 = init_weights((128 * 3 * 3, 625))
78 w_o = init_weights((625, 10))
79
80 noise_l1, noise_l2, noise_l3, noise_l4, noise_py_x = model(X, w, w2, w3, w4, 0.2, 0.5)
81 l1, l2, l3, l4, py_x = model(X, w, w2, w3, w4, 0., 0.)
82 y_x = T.argmax(py_x, axis=1)
83
84
```

a “block” of computation conv -> activate -> pool -> noise

convert from 4tensor to normal matrix

reshape into conv 4tensor (b, c, 0, 1) format

now 4tensor for conv instead of matrix

A convolutional network in Theano

```
def model(X, w, w2, w3, w4, p_drop_conv, p_drop_hidden):
    l1a = rectify(conv2d(X, w, border_mode='full'))
    l1 = max_pool_2d(l1a, (2, 2))
    l1 = dropout(l1, p_drop_conv)

    l2a = rectify(conv2d(l1, w2))
    l2 = max_pool_2d(l2a, (2, 2))
    l2 = dropout(l2, p_drop_conv)

    l3a = rectify(conv2d(l2, w3))
    l3b = max_pool_2d(l3a, (2, 2))
    l3 = T.flatten(l3b, outdim=2)
    l3 = dropout(l3, p_drop_conv)

    l4 = rectify(T.dot(l3, w4))
    l4 = dropout(l4, p_drop_hidden)

    pyx = softmax(T.dot(l4, w_o))
    return l1, l2, l3, l4, pyx

trX, teX, trY, teY = mnist(onehot=True)

trX = trX.reshape(-1, 1, 28, 28)
teX = teX.reshape(-1, 1, 28, 28)

X = T.ftensor4()
Y = T.fmatrix()

w = init_weights((32, 1, 3, 3))
w2 = init_weights((64, 32, 3, 3))
w3 = init_weights((128, 64, 3, 3))
w4 = init_weights((128 * 3 * 3, 625))
w_o = init_weights((625, 10))

noise_l1, noise_l2, noise_l3, noise_l4, noise_py_x = model(X, w, w2, w3, w4, 0.2, 0.5)
l1, l2, l3, l4, py_x = model(X, w, w2, w3, w4, 0., 0.)
y_x = T.argmax(py_x, axis=1)
```

a “block” of computation conv -> activate -> pool -> noise

convert from 4tensor to normal matrix

reshape into conv 4tensor (b, c, 0, 1) format

now 4tensor for conv instead of matrix

conv weights (n_kernels, n_channels, kernel_w, kerbel_h)

A convolutional network in Theano

```
46 def model(X, w, w2, w3, w4, p_drop_conv, p_drop_hidden):
47     l1a = rectify(conv2d(X, w, border_mode='full'))
48     l1 = max_pool_2d(l1a, (2, 2))
49     l1 = dropout(l1, p_drop_conv)
50
51     l2a = rectify(conv2d(l1, w2))
52     l2 = max_pool_2d(l2a, (2, 2))
53     l2 = dropout(l2, p_drop_conv)
54
55     l3a = rectify(conv2d(l2, w3))
56     l3b = max_pool_2d(l3a, (2, 2))
57     l3 = T.flatten(l3b, outdim=2)
58     l3 = dropout(l3, p_drop_conv)
59
60     l4 = rectify(T.dot(l3, w4))
61     l4 = dropout(l4, p_drop_hidden)
62
63     pyx = softmax(T.dot(l4, w_o))
64     return l1, l2, l3, l4, pyx
65
66 trX, teX, trY, teY = mnist(onehot=True)
67
68 trX = trX.reshape(-1, 1, 28, 28)
69 teX = teX.reshape(-1, 1, 28, 28)
70
71 X = T.ftensor4()
72 Y = T.fmatrix()
73
74 w = init_weights((32, 1, 3, 3))
75 w2 = init_weights((64, 32, 3, 3))
76 w3 = init_weights((128, 64, 3, 3))
77 w4 = init_weights((128 * 3 * 3, 625))
78 w_o = init_weights((625, 10))
79
80 noise_l1, noise_l2, noise_l3, noise_l4, noise_py_x = model(X, w, w2, w3, w4, 0.2, 0.5)
81 l1, l2, l3, l4, py_x = model(X, w, w2, w3, w4, 0., 0.)
82 y_x = T.argmax(py_x, axis=1)
83
84
```

a “block” of computation conv -> activate -> pool -> noise

convert from 4tensor to normal matrix

reshape into conv 4tensor (b, c, 0, 1) format

now 4tensor for conv instead of matrix

conv weights (n_kernels, n_channels, kernel_w, kerbel_h)

highest conv layer has 128 filters and a 3x3 grid of responses

A convolutional network in Theano

```
def model(X, w, w2, w3, w4, p_drop_conv, p_drop_hidden):
    l1a = rectify(conv2d(X, w, border_mode='full'))
    l1 = max_pool_2d(l1a, (2, 2))
    l1 = dropout(l1, p_drop_conv)

    l2a = rectify(conv2d(l1, w2))
    l2 = max_pool_2d(l2a, (2, 2))
    l2 = dropout(l2, p_drop_conv)

    l3a = rectify(conv2d(l2, w3))
    l3b = max_pool_2d(l3a, (2, 2))
    l3 = T.flatten(l3b, outdim=2)
    l3 = dropout(l3, p_drop_conv)

    l4 = rectify(T.dot(l3, w4))
    l4 = dropout(l4, p_drop_hidden)

    pyx = softmax(T.dot(l4, w_o))
    return l1, l2, l3, l4, pyx

trX, teX, trY, teY = mnist(onehot=True)

trX = trX.reshape(-1, 1, 28, 28)
teX = teX.reshape(-1, 1, 28, 28)

X = T.ftensor4()
Y = T.fmatrix()

w = init_weights((32, 1, 3, 3))
w2 = init_weights((64, 32, 3, 3))
w3 = init_weights((128, 64, 3, 3))
w4 = init_weights((128 * 3 * 3, 625))
w_o = init_weights((625, 10))

noise_l1, noise_l2, noise_l3, noise_l4, noise_py_x = model(X, w, w2, w3, w4, 0.2, 0.5)
l1, l2, l3, l4, py_x = model(X, w, w2, w3, w4, 0., 0.)
y_x = T.argmax(py_x, axis=1)
```

a “block” of computation conv -> activate -> pool -> noise

convert from 4tensor to normal matrix

reshape into conv 4tensor (b, c, 0, 1) format

now 4tensor for conv instead of matrix

conv weights (n_kernels, n_channels, kernel_w, kerbel_h)

highest conv layer has 128 filters and a 3x3 grid of responses

noise during training

A convolutional network in Theano

```
def model(X, w, w2, w3, w4, p_drop_conv, p_drop_hidden):
    l1a = rectify(conv2d(X, w, border_mode='full'))
    l1 = max_pool_2d(l1a, (2, 2))
    l1 = dropout(l1, p_drop_conv)

    l2a = rectify(conv2d(l1, w2))
    l2 = max_pool_2d(l2a, (2, 2))
    l2 = dropout(l2, p_drop_conv)

    l3a = rectify(conv2d(l2, w3))
    l3b = max_pool_2d(l3a, (2, 2))
    l3 = T.flatten(l3b, outdim=2)
    l3 = dropout(l3, p_drop_conv)

    l4 = rectify(T.dot(l3, w4))
    l4 = dropout(l4, p_drop_hidden)

    pyx = softmax(T.dot(l4, w_o))
    return l1, l2, l3, l4, pyx

trX, teX, trY, teY = mnist(onehot=True)

trX = trX.reshape(-1, 1, 28, 28)
teX = teX.reshape(-1, 1, 28, 28)

X = T.ftensor4()
Y = T.fmatrix()

w = init_weights((32, 1, 3, 3))
w2 = init_weights((64, 32, 3, 3))
w3 = init_weights((128, 64, 3, 3))
w4 = init_weights((128 * 3 * 3, 625))
w_o = init_weights((625, 10))

noise_l1, noise_l2, noise_l3, noise_l4, noise_py_x = model(X, w, w2, w3, w4, 0.2, 0.5)
l1, l2, l3, l4, py_x = model(X, w, w2, w3, w4, 0., 0.)
y_x = T.argmax(py_x, axis=1)
```

a “block” of computation conv -> activate -> pool -> noise

convert from 4tensor to normal matrix

reshape into conv 4tensor (b, c, 0, 1) format

now 4tensor for conv instead of matrix

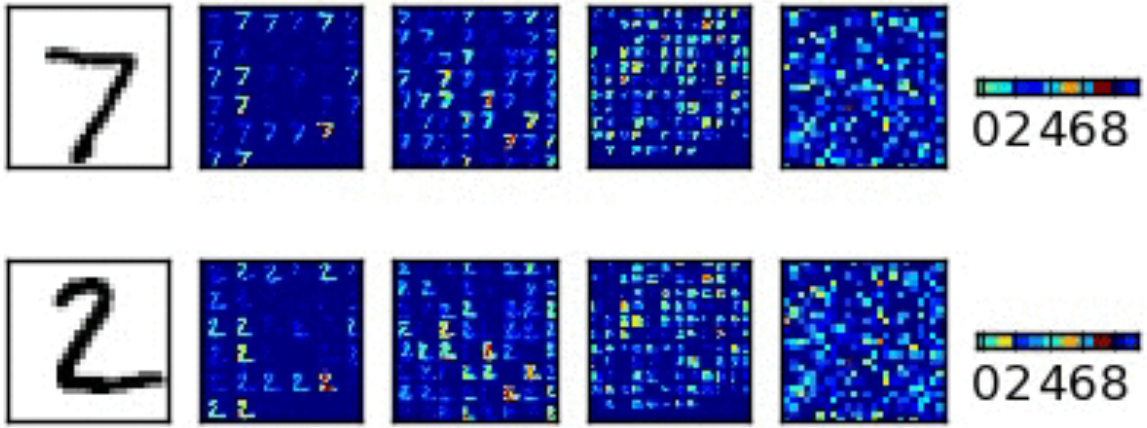
conv weights (n_kernels, n_channels, kernel_w, kerbel_h)

highest conv layer has 128 filters and a 3x3 grid of responses

noise during training

no noise for prediction

A convolutional network in Theano



Test Accuracy: 99.5%

What a convolutional network learns

Takeaways

- A few tricks are needed to get good results
  - Noise important for regularization
  - Rectifiers for faster, better, learning
  - Don’t use SGD - lots of cheap simple improvements

- Models need room to compute.
- If your data has structure, your model should respect it.

Resources

- More in-depth theano tutorials
  - http://www.deeplearning.net/tutorial/
- Theano docs
  - http://www.deeplearning.net/software/theano/library/
- Community
  - http://www.reddit.com/r/machinelearning

A plug

Keep up to date with indico:

https://indico1.typeform.com/to/DgN5SP

Questions?