CONVOLUTIONAL NEURAL NETWORK (CNN)

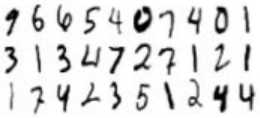
The Street View House Numbers (SVHN) Dataset

SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with mir
formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorp
(over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing di
is obtained from house numbers in Google Street View images.

Overview

- 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10.
- 73257 digits for training, 26032 digits for testing, and 531131 additional, somewhat less difficult samples, to use as extra training dat
- Comes in two formats:

1. Original images with character level bounding boxes.
2. MNIST-like 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides)



Knowledge • 1,075 teams

Digit Recognizer

Wed 25 Jul 2012 — Sat 31 Dec 2016 (9 months to go)

Dashboard

Home
Data
Make a submission

Information
Description
Evaluation
Rules
Tutorial

Forum

Scripts
New Script

Competition Details » Get the Data » Make a submission

Classify handwritten digits using the famous MNIST data

Get started on this competition through Kaggle Scripts

The goal in this competition is to take an image of a handwritten single digit, and determine what that digit is. As the competition progresses, we will release tutorials which explain different machine learning algorithms and help you to get started.

Data Pre-processing

Grayscale image: 2-D

Color image: 3-D

MNIST data: N x 28 x 28

For a full network we flatten it: N x 784

Train

$W \leftarrow W - \eta \nabla J$

Theano and TensorFlow will calculate it for us.

Prediction

- Feedforward
- $z_1 = s(W_0 x)$
- $z_2 = s(W_1 z_1)$
- $z_3 = s(W_2 z_2)$
- $y = s(W_3 z_3)$

- s() = nonlinearity like sigmoid, tanh, relu, softmax
- Last layer is sigmoid for binary, softmax for more than 2 classes

Train and Predict

```
model = Model()

model.fit(X, Y)

Yhat = model.predict(X)
```

Mechanical calculation

Let:
size(x) = N = 5, x(0), ..., x(4) are defined, otherwise 0
size(w) = M = 3, w[j] = 1 for j=0,1,2

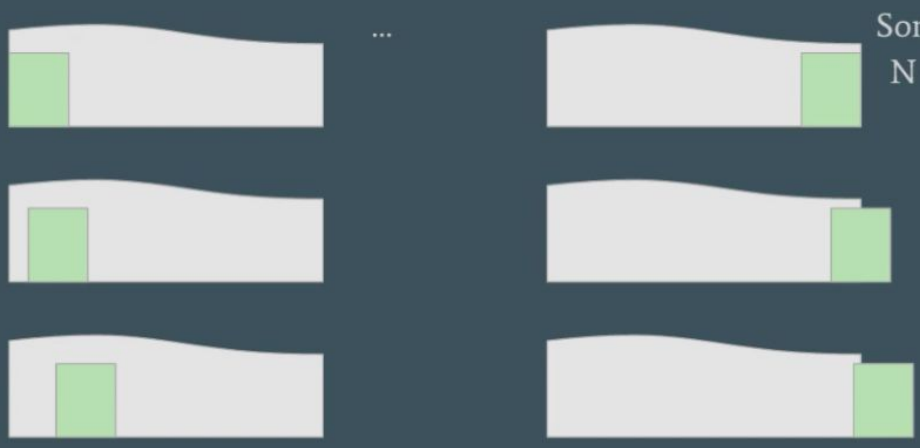
$y[i] = \sum_j w[j]x[i - j]$

y[0] = x[0 - 0] + x[0 - 1] + x[0 - 2] = x[0] + x[-1] + x[-2] = x[0]

y[1] = x[1 - 0] + x[1 - 1] + x[1 - 2] = x[1] + x[0] + x[-1] = x[1] + x[0]

y[2] = x[2 - 0] + x[2 - 1] + x[2 - 2] = x[2] + x[1] + x[0]

In Pictures



Size of output

$y[i,j] = \sum_{ii}\sum_{jj} w[ii,jj]*x[i - ii,j - jj]$

Mathematically, it's -infinity to +infinity (but we can't store that)

The size of X is the size of where X is non-zero.

The size of W is the size of where W is non-zero.

What is the size of Y?

Write your own convolution

Hopefully it helps you understand convolution better! Pseudo-code:

```
n1, n2 = X.shape
m1, m2 = W.shape
for i in xrange(n1):
  for j in xrange(n2):
    for ii in xrange(m1):
      for jj in xrange(m2):
        y[i,j] = w[ii,jj]*x[i - ii,j - jj]
```

Convolution in 2-D

Simply do the same operation in both directions independently

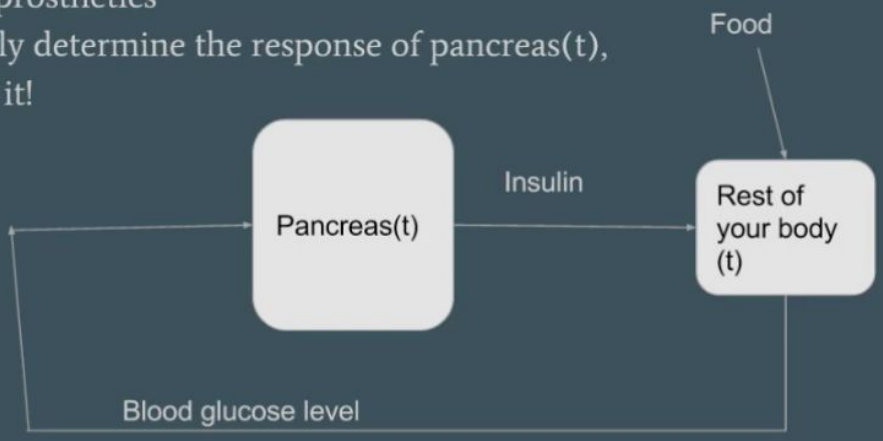
$x[n,m] * w[n,m] = \sum x[n-i,m-j]w[i,j]$

In code (in reality need to make y bigger or ignore out-of-bounds, we'll just use lib):

```
def convolve(x, w):
  y = np.zeros(x.shape)
  for n in xrange(x.shape[0]):
    for m in xrange(x.shape[1]):
      for i in xrange(w.shape[0]):
        for j in xrange(w.shape[1]):
          y[n,m] += x[n-i,m-j]*w[i,j]
```

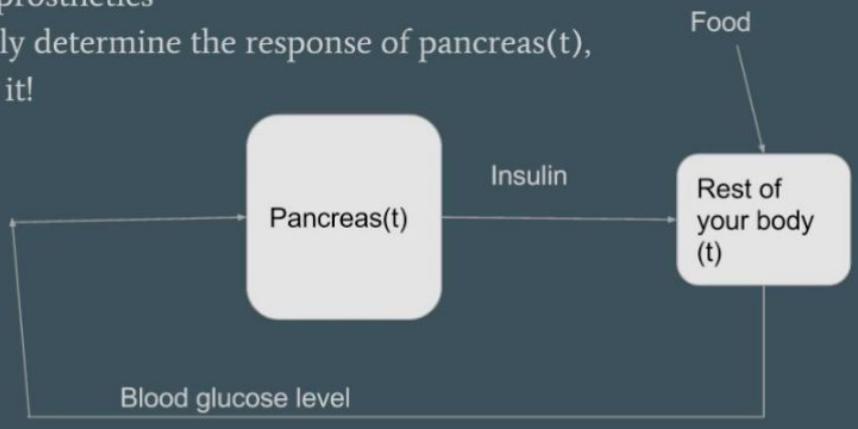
Applications

- Human organ prosthetics
- If we can exactly determine the response of pancreas(t), we can replace it!



Applications

- Human organ prosthetics
- If we can exactly determine the response of pancreas(t), we can replace it!



Impulse Response

$$x(t) * \delta(t) = \int x(t - \tau)\delta(\tau)d\tau = x(t)$$

$$x[n] * \delta[n] = \sum \delta[m]x[n - m] = \sum \delta[0]x[n] = x[n]$$

One important filter

What happens if we convolve with a "delta" function?

In continuous time: $\delta(t) = du/dt$ (where u(t) is a step function)

u(t) = 1 if t >= 0 and 0 otherwise 

So $\delta(t)$ value at t=0 is infinity, but the integral is 1. 

In discrete time:

$\delta[n]$ = 1 if n = 0 and 0 otherwise

One important filter

What happens if we convolve with a "delta" function?

In continuous time: $\delta(t) = du/dt$ (where u(t) is a step function)

u(t) = 1 if t >= 0 and 0 otherwise 

So $\delta(t)$ value at t=0 is infinity, but the integral is 1. 

In discrete time:

$\delta[n]$ = 1 if n = 0 and 0 otherwise

What is convolution?

Continuous:

$x(t) * w(t) = \int x(\tau)w(t - \tau)d\tau = \int w(\tau)x(t - \tau)d\tau$

Discrete (we'll use this one later):

$x[n] * w[n] = \sum x[m]w[n - m] = \sum w[m]x[n - m]$

You can slide the filter across the signal, or slide the signal across the filter, it's the same thing!

Representing filters mathematically

We can generalize this even more:

$\int w(m)x(t - m)dm$

We'll go back to the summation (non-integral form) for image processing so don't worry about the integrals!

$m' = t - m \;/\; m = t - m'$

$\int w(t - m')x(m')dm'$

*** doesn't matter which we consider the filter and which we consider the signal

How would we make an echo?

How would we do that in math?

$x(t) + w_1 x(t - t_1) + w_2 x(t - t_2) + ...$

Where $w_1 < 1$, $w_2 <$ w1, etc. (for this particular filter)

More generally:

$\sum w_m x(t - t_m)$

How would we make an echo?

Length stays the same, but 2 things happen:

1) Add a quieter signal
2) The quieter signal is shifted to the right (forward in time)

What is convolution?

- Convolution is an important operation from signal processing and analysis
- Think of your favorite audio effect - let's say it's an "echo"
- All effects can be thought of as filters, or what we call them in machine learning sometimes, kernels - we'll call it h(t) or w(t)

Mechanical calculation

Let:
size(x) = N = 5, x(0), ..., x(4) are defined, otherwise 0
size(w) = M = 3, w[j] = 1 for j=0,1,2

$y[i] = \sum_j w[j]x[i - j]$

$y[0] = x[0 - 0] + x[0 - 1] + x[0 - 2] = x[0] + x[-1] + x[-2] = x[0]$

$y[1] = x[1 - 0] + x[1 - 1] + x[1 - 2] = x[1] + x[0] + x[-1] = x[1] + x[0]$

$y[2] = x[2 - 0] + x[2 - 1] + x[2 - 2] = x[2] + x[1] + x[0]$

Mechanical Calculation

$y[3] = x[3] + x[2] + x[1]$

$y[4] = x[4] + x[3] + x[2]$

$y[5] = x[5] + x[4] + x[3] = x[4] + x[3]$

$y[6] = x[6] + x[5] + x[4] = x[4]$

$y[7] = x[7] + x[6] + x[5] = 0$

Just to be complete:
$y[-1] = x[-1] + x[-2] + x[-3] = 0$

Mechanical Calculation

$y[0] \ldots y[6]$ have values, therefore its size is 7.

$size(x) = 5$, $size(w) = 3$

$5 + 3 - 1 = 7$

Why Convolution?

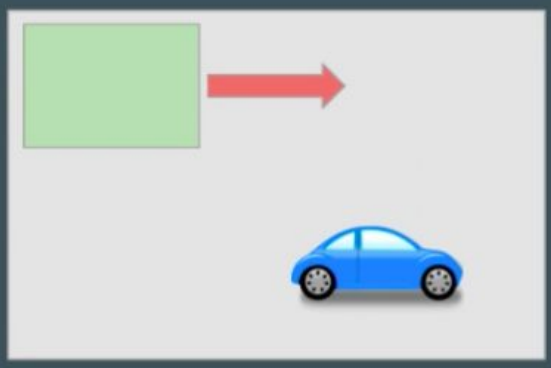
- Images are special types of inputs
- We can see them with our own eyes, so we know what the neural network should be robust to
- Perfect example: translational invariance

Why Convolution?

- We know these are both cars
- But a feedforward neural network has different weights everywhere
- If we only trained it on the left image, it wouldn't recognize the right

Why Convolution?

- What does convolution do?
- Multiplies the same weights everywhere on the image
- Suppose this is a feature finder that identifies the car
- i.e. outputs very +ve # when it sees a car, very -ve # otherwise
- Then it will find a car no matter where it is on the image



Why Convolution?

Recall:

Feedforward neural networks contain multiple layers which allow each layer to find successively complex features

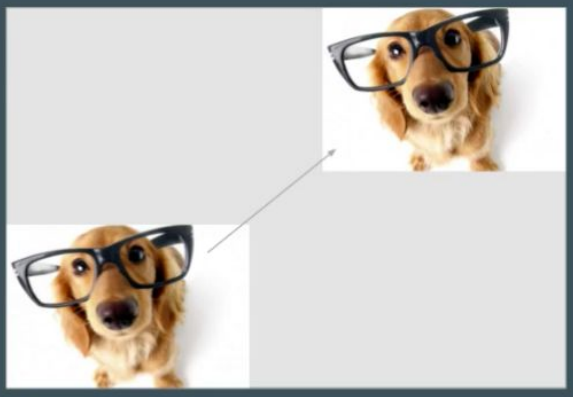
Layers of convolutions behave the same. First layer will find small, simple features anywhere on the image (e.g. a line). Next layer will find more complex features, and so on.

CNNs

- Before, every node in previous layer connected to every node in next layer. No more!
- Inspired by LeCun, 1998. "Gradient-based learning applied to document recognition."
- LeNet model

Why do convolution?

- Think of it as a sliding window or sliding filter
- Doesn't matter where the dog is
- It's still a dog
- "Translational invariance"

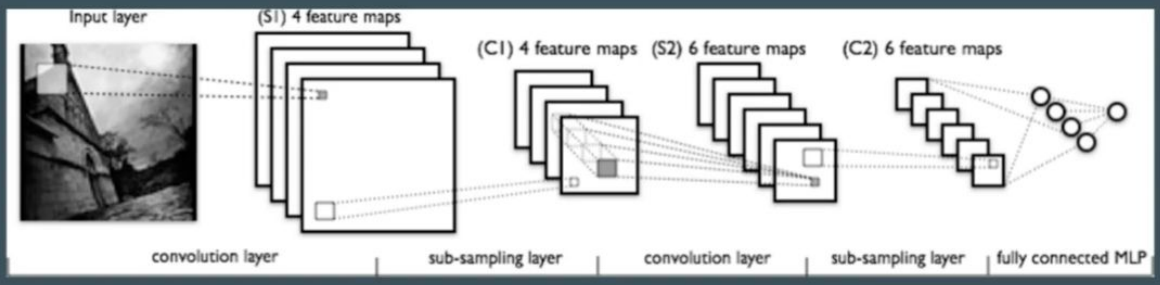


- Question to think about:
- How to solve rotational invariance?

Downsampling (aka Sub-sampling)

- 16kHz is adequate for voice
- Telephone is 8kHz - sounds muffled
- After convolution, we just want to know if a feature was found, so take a neighborhood (square) of data and get the max (called maxpooling)
- Can also use average pooling (but we won't)
- theano.tensor.signal.downsample.max_pool_2d
- tf.nn.max_pool

The architecture



Technicalities

- 4-D tensor input: N x 3 colors x width x height
- 4-D convolution filter / kernel: #feature maps x 3 colors x width x height

- Order of each dimension somewhat arbitrary
- MATLAB will load as 32x32x3xN
- Theano / TensorFlow filters in different order
- Filter size << Image size
- Weight sharing → Better generalization

Technicalities

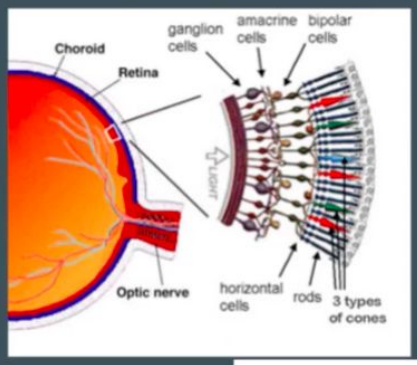
- 4-D tensor input: N x 3 colors x width x height
- 4-D convolution filter / kernel: #feature maps x 3 colors x width x height

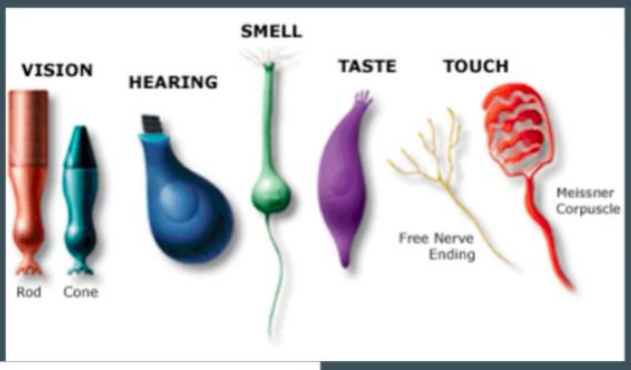
- Order of each dimension somewhat arbitrary
- MATLAB will load as 32x32x3xN
- Theano / TensorFlow filters in different order
- Filter size << Image size
- Weight sharing → Better generalization

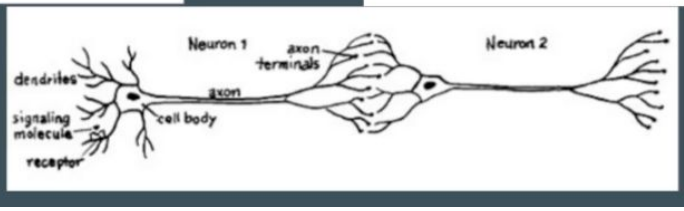
Visual Cortex

- Original inspiration: Modeling the brain
- Now model biological visual system

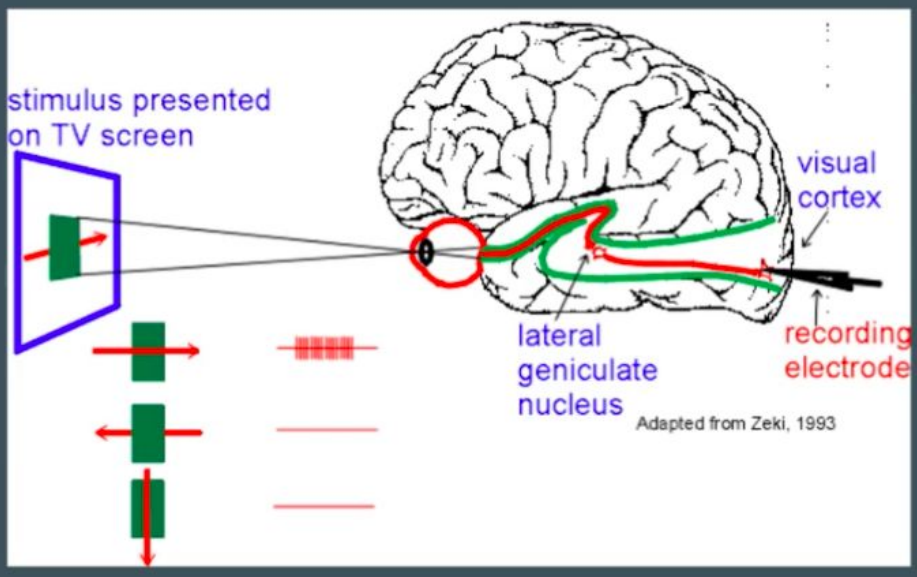
Receptors



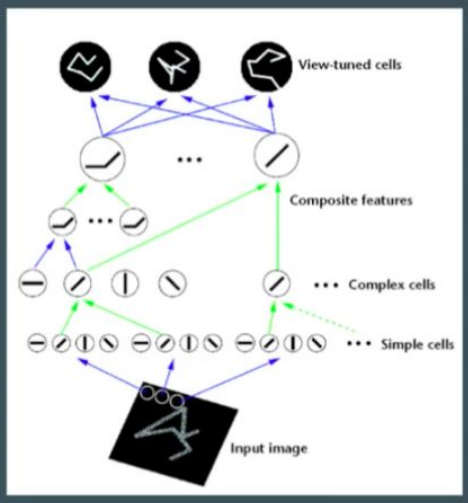


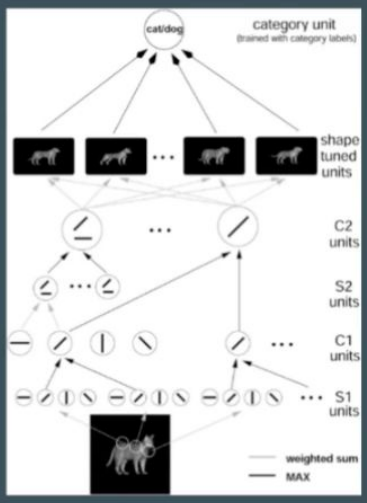


Receptive Field



Hierarchy





Hubel

Hubel, 1968 “Receptive fields and functional architecture of monkey striate cortex”. Journal of Physiology

Wikipedia: Hubel and Wiesel (e.g., Hubel, 1963; Hubel-Wiesel, 1962) advanced the theory that ***receptive fields of cells at one level of the visual system are formed from input by cells at a lower level of the visual system.*** **In this way, small, *simple receptive fields could be combined to form large, complex receptive fields.***

Convolution and Pooling Gradient

- You already know how to derive the gradient for a feedforward neural network of any size from Deep Learning part 1
- What if we add convolution and pooling?

Convolution

$$y(m,n) = \sum_i \sum_j w(i,j)x(m-i,n-j)$$

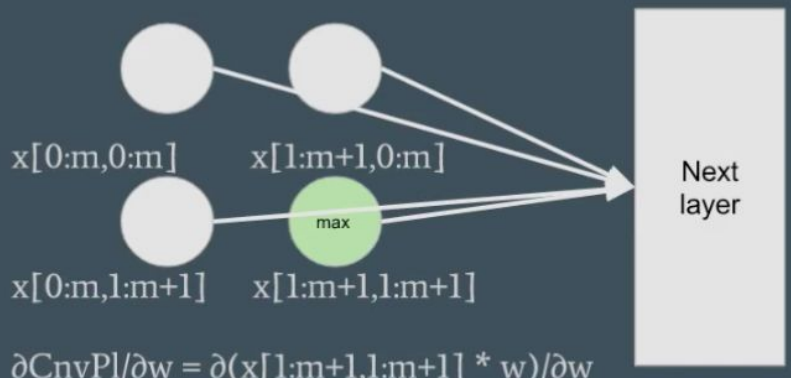
$$\partial y(m,n)/\partial w(i,j) = ?$$

Pooling

- Average pooling is easy, we know how to take the derivative inside a summation already.
- What about max pooling? This is weird, because we are choosing the output of one node and discarding the others.

Max Pooling

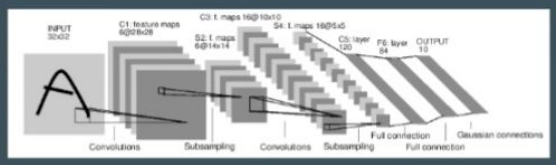
- If it influences an output, it should be updated
- A weight should be updated by the error signal at any nodes it affects



$$\partial CnvPl/\partial w = \partial(x[1:m+1,1:m+1] * w)/\partial w$$

LeNet Remedial

- For some, it's easy to understand the high-level concepts and match them to the code
- We'll review the LeNet architecture in-depth
- It's really just 2 simple steps:



|----------step 1----------|-step 2-|

1) Convolution + Pooling layers (ConvPool)

2) Feedforward neural network layers (deep learning part 1/2)

- If you don't already know step 2 in Theano+TensorFlow you might have trouble with this course!
- Difficult parts: Step 1, Interface between Step 1 and 2

Why 4-D filters?

- Each individual filter is of the same dimensionality as the image (3-D = color x width x height)
- And we just have multiple of them (stack multiple 3-D things together → one 4-D thing)

Shapes

- Now let's talk specific shapes
- Mini-batch of images: N x C x W x H
- Number of samples / color / width / height
- Only one of these will change after convolution
- N can't change
- If we have N images in, we must have N images out
- W and H might change but not necessarily
- Recall: output of convolution can be:
  - New width = old width + filter width - 1
  - New width = old width - filter width + 1
  - New width = old width
- You can configure it!

Shapes

- Color will change!
- Think of color intensity as a feature value
- So what are the 4 dimensions of the filter?

(Num feature maps in, num feature maps out, filter width, filter height)

- Not necessarily in this order (Theano and TensorFlow order them differently)
- At the very first layer, # colors = 3 = num feature maps in
- At every layer after, it's whatever you set num feature maps to

$$\text{Image\_out}[n,k,i,j] = \sum_{i'}\sum_{j'}\sum_{m}\text{Filter}[m,k,i',j']\text{Image\_in}[n,m,i - i', j - j']$$

Mini-Quiz

How many numbers do we need to specify the shape of a convolution filter?

Pause the video until you can think of the answer.

Mini-Quiz Answer

You need to specify 3 numbers to specify the shape of a convolution filter.

feature maps out
Filter width
Filter height

feature maps in is defined by the data itself. Ex. in the first layer, it will be 3 (since there are 3 color channels).

Since neural networks are repeating structures, the # of feature maps out becomes the # of feature maps in of the next layer, and so on.

Mini-Quiz

Shape of image batch = (batch size, color, width, height) = (15, 3, 32, 32)

Filter shape = (3, 10, 5, 5)

Assume we configure convolution so that image size out = image size in

What is the size of the data (image batch) after convolution?

Mini-Quiz Answer

Image batch size after convolution = (15, 10, 32, 32)

N must remain the same (the # of images can't change of course)

Width and height must remain the same (we configured the convolution so that this is so)

The only thing left to change is the number of feature maps, which is now 10.

What's next?

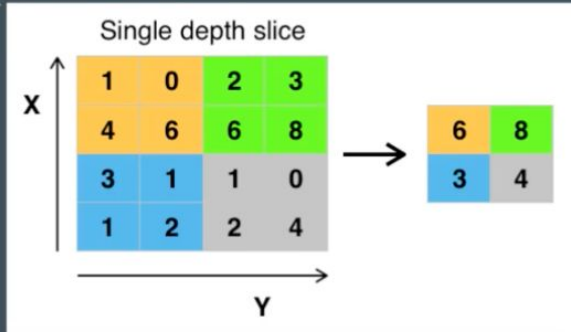
We could chain this to another convolution and its number of input feature maps would be 10.

But we know in the LeNet pooling comes next. Another convolution would be the equivalent of having a poolsize of (1, 1).

Not uncommon, but outside scope of this course.

Pooling

- Pooling is just another word for downsampling
- If we have a poolsize of 2x2, that means we split up the original image into a grid of blocks each of size 2x2
- Maxpooling: take the max of the 4 pixels
- Average pooling: take the average of the 4 pixels
- This diagram shows maxpooling



Mini-Quiz

We know the data is 4-d N x C x W x H

Which of these are affected by pooling?

Mini-Quiz Answer

Only width and height are affected by pooling.

It is a spatial transformation.

Number of images must remain the same.

Number of features must remain the same - we don't want to throw away features.

Mini-Quiz

Suppose we just finished convolution, data is now of size (15, 10, 32, 32)

We now put it through a pooling operation with poolsize = (2, 2)

What is the data size after pooling?

Pause the video until you get the answer.

Mini-Quiz Answer

Draw it on paper if you don't understand it visually.

(15, 10, 16, 16)

Downsample by 2 == that dimension halves in size

ConvPool to Fully Connected Feedforward Layer

- What do we do at the interface between the last ConvPool layer and the first fully connected feedforward layer?
- Data is currently N x C x W x H
- When we pass data through a feedforward net it needs to be of size N x D, each sample is a D-size vector
- How do we fit a 4-d array into a 2-d array?

Answer

- Same as how we dealt with images when we used only feedforward neural networks - flatten()
- In a feedforward net, each input node is treated equally, doesn't matter what color channel or (x, y) coordinate the pixel came from
- The only dimension that should not change is N
- If there are N input images, there must be N output predictions, and correspondingly N target labels
- Theano and Numpy have simple flatten methods
- TensorFlow's is ugly

Answer

- Same as how we dealt with images when we used only feedforward neural networks - flatten()
- In a feedforward net, each input node is treated equally, doesn't matter what color channel or (x, y) coordinate the pixel came from
- The only dimension that should not change is N
- If there are N input images, there must be N output predictions, and correspondingly N target labels
- Theano and Numpy have simple flatten methods
- TensorFlow's is ugly

Mini-Quiz

- Input mini batch is of size (15, 3, 32, 32)
- Convolution is configured to output image same size as input
- 2 ConvPool layers
- 1st ConvPool layer: # feature maps = 10, filter width = 5, filter height = 5, poolsize = (2, 2)
- 2nd ConvPool layer: # feature maps = 8, filter width = 3, filter height = 3, poolsize = (2, 2)
- 1 feedforward layer: # hidden units = 100
- What does the size of the weight matrix at the feedforward layer have to be?
- Pause the video until you get the answer.

Mini-Quiz

- Input mini batch is of size (15, 3, 32, 32)
- Convolution is configured to output image same size as input
- 2 ConvPool layers
- 1st ConvPool layer: # feature maps = 10, filter width = 5, filter height = 5, poolsize = (2, 2)
- 2nd ConvPool layer: # feature maps = 8, filter width = 3, filter height = 3, poolsize = (2, 2)
- 1 feedforward layer: # hidden units = 100
- What does the size of the weight matrix at the feedforward layer have to be?
- Pause the video until you get the answer.

Mini-Quiz Answer

- After 1st convolution: (15, 10, 32, 32) (we already did this)
- After 1st pooling: (15, 10, 16, 16) (we did this too)
- After 2nd convolution: (15, 8, 16, 16)
- After 2nd pooling: (15, 8, 8, 8)
- After flatten: (15, 512)

Therefore, weight matrix must be of shape (512, 100) since the next layer has 100 units

How many?

- Common question:
- How many ConvPool layers should I include?
- How many feedforward layers should I include?
- What filter sizes should I use?
- What number of feature maps should I use?
- These are all hyperparameters - what is best is specific to your dataset
- Deep Learning part 2 discussed random search, grid search, cross-validation to help you do this

Convolution and Downsample

```
from theano.tensor.nnet import conv2d

from theano.tensor.signal import downsample
```

ConvPool

```
def convpool(X, W, b, poolsize=(2, 2)):
  conv_out = conv2d(input=X, filters=W)
  pooled_out = downsample.max_pool_2d(
    input=conv_out,
    ds=poolsize,
    ignore_border=True
  )
  return T.tanh(pooled_out + b.dimshuffle('x', 0, 'x', 'x'))
```

Rearrange Input

```
def rearrange(X):
  # input is (32, 32, 3, N)
  # output is (N, 3, 32, 32)
  N = X.shape[-1]
  out = np.zeros((N, 3, 32, 32), dtype=np.float32)
  for i in xrange(N):
    for j in xrange(3):
      out[i, j, :, :] = X[:, :, j, i]
  return out / 255
```

Filter Size

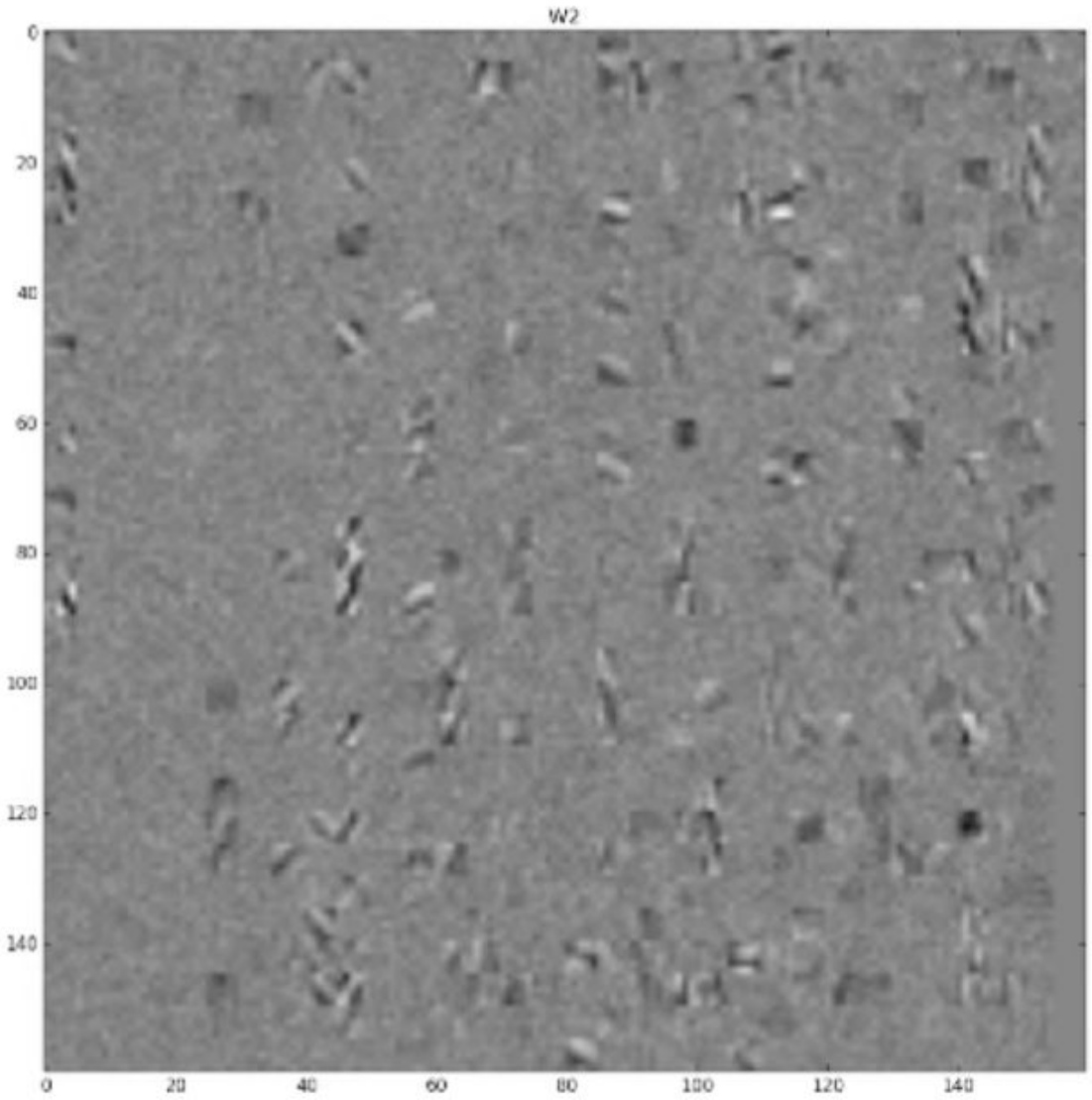
```
# after conv will be of dimension 32 - 5 + 1 = 28
# after downsample 28 / 2 = 14
W1_shape = (20, 3, 5, 5) # (num_feature_maps, num_color_channels, filter_width, filter_height)
W1_init = init_filter(W1_shape, poolsz)
b1_init = np.zeros(W1_shape[0], dtype=np.float32) # one bias per output feature map

# after conv will be of dimension 14 - 5 + 1 = 10
# after downsample 10 / 2 = 5
W2_shape = (50, 20, 5, 5) # (num_feature_maps, old_num_feature_maps, filter_width, filter_height)
W2_init = init_filter(W2_shape, poolsz)
b2_init = np.zeros(W2_shape[0], dtype=np.float32)
```

Vanilla ANN stage

```
W3_init = np.random.randn(W2_shape[0]*5*5, M) /
    np.sqrt(W2_shape[0]*5*5 + M)
b3_init = np.zeros(M, dtype=np.float32)
W4_init = np.random.randn(M, K) / np.sqrt(M + K)
b4_init = np.zeros(K, dtype=np.float32)
```

Forward Pass

```
Z1 = convpool(X, W1, b1)
Z2 = convpool(Z1, W2, b2)
Z3 = relu(Z2.flatten(ndim=2).dot(W3) + b3)
pY = T.nnet.softmax( Z3.dot(W4) + b4 )
```



ConvPool

```
def convpool(X, W, b):
  # just assume pool size is (2,2) because we need to augment it with 1s
  conv_out = tf.nn.conv2d(X, W, strides=[1, 1, 1, 1], padding='SAME')
  conv_out = tf.nn.bias_add(conv_out, b)
  pool_out = tf.nn.max_pool(
    conv_out,
    ksize=[1, 2, 2, 1],
    strides=[1, 2, 2, 1],
    padding='SAME')
  return pool_out
```

Rearrange Inputs

```
def rearrange(X):
  # input is (32, 32, 3, N)
  # output is (N, 32, 32, 3)
  N = X.shape[-1]
  out = np.zeros((N, 32, 32, 3), dtype=np.float32)
  for i in xrange(N):
    for j in xrange(3):
      out[i, :, :, j] = X[:, :, j, i]
  return out / 255
```

Save RAM

```
Xtrain = Xtrain[:73000,]
Ytrain = Ytrain[:73000]
Xtest = Xtest[:26000,]
Ytest = Ytest[:26000]
Ytest_ind = Ytest_ind[:26000,]

# or you can compute using N = N / batch_sz * batch_sz
```

Different order of dimensions in filters

```
# (filter_width, filter_height, num_color_channels, num_feature_maps)
W1_shape = (5, 5, 3, 20)
W1_init = init_filter(W1_shape, poolsz)
b1_init = np.zeros(W1_shape[-1], dtype=np.float32)


# (filter_width, filter_height, old_num_feature_maps, num_feature_maps)
W2_shape = (5, 5, 20, 50)
W2_init = init_filter(W2_shape, poolsz)
b2_init = np.zeros(W2_shape[-1], dtype=np.float32)
```

Vanilla ANN

```
W3_init = np.random.randn(W2_shape[-1]*8*8, M) /
    np.sqrt(W2_shape[-1]*8*8 + M)
b3_init = np.zeros(M, dtype=np.float32)
W4_init = np.random.randn(M, K) / np.sqrt(M + K)
b4_init = np.zeros(K, dtype=np.float32)
```

Forward Pass

```
Z1 = convpool(X, W1, b1)
Z2 = convpool(Z1, W2, b2)

Z2_shape = Z2.get_shape().as_list()
Z2r = tf.reshape(Z2, [Z2_shape[0], np.prod(Z2_shape[1:])])
Z3 = tf.nn.relu( tf.matmul(Z2r, W3) + b3 )
Yish = tf.matmul(Z3, W4) + b4
```

More Invariance

- Translations - convolutions slide across the entire image
- From earlier: What about rotations?

Other:

- A red 9 should be the same as a blue 9 (color invariance)
- A small 9 should be the same as a big 9 (size invariance)
- More processing power + more data → deep learning popularity

More Invariance

- Translations - convolutions slide across the entire image
- From earlier: What about rotations?

Other:

- A red 9 should be the same as a blue 9 (color invariance)
- A small 9 should be the same as a big 9 (size invariance)
- More processing power + more data → deep learning popularity

More data

Make more data yourself!

Translate it. Rotate it. Change the size.

Change the color (if it should be invariant).

Very big networks, too many parameters

- GPU / Amazon Web Services
- Hyperparameter optimization
- Udemy.com/data-science-deep-learning-in-theano-tensorflow

Very big networks, too many parameters

- GPU / Amazon Web Services
- Hyperparameter optimization
- Udemy.com/data-science-deep-learning-in-theano-tensorflow

Learning

Linear regression: $y = w^T x + b$

From then until now:

1. Calculate likelihood P(data|model)
2. J = negative log-likelihood
3. $W \leftarrow W - \eta \nabla J$

Still hasn't changed! (Isn't it easy by now?)

Exercise

Try the CNN on MNIST data by yourself.

Completed • $500

Challenges in Representation Learning: Facial Expression Recognition Challenge

Fri 12 Apr 2013 – Fri 24 May 2013 (3 years ago)

Dashboard

Home
Data

Information
Description
Evaluation
Rules
Prizes
Timeline

Forum

Leaderboard
Public
Private

Visualization

Learn facial expressions from an image

One motivation for representation learning is that learning algorithms can design features better and faster than humans can. To this end, we hold this challenge that does not explicitly require that entries use representation learning. Rather, we introduce an entirely new dataset and invite competitors from all related communities to solve it. The dataset for this challenge is a facial expression classification dataset that we have assembled from the internet. Because this is a newly introduced dataset, this contest will see which methods are the easiest to get quickly working on new data.

Example baseline submissions are available as part of the pylearn2 python package available at https://github.com/lisa-lab/pylearn2

The baseline submissions for this contest are in

The data

- 48x48 grayscale images
- Faces are centered, approx same size
- 7 classes:
  - 0 = Angry
  - 1 = Disgust
  - 2 = Fear
  - 3 = Happy
  - 4 = Sad
  - 5 = Surprise
  - 6 = Neutral
- 3 column CSV: label, pixels (space-separated), train/test

2-class problem vs. 7-class problem

- When we switch to softmax (which we will in Deep Learning part 1), wi problem get easier or harder?
- 2 class:
  - Guess at random - expect 50% error
- 7 class:
  - Guess at random - expect 6/7 = 86% error
- K class: 1/K chance of being correct

Kaggle top score: ~70% correct

Structure of data

- Each image sample is 48x48 = 2304 dimensions
- No color
- If we had color, it would be 3x48x48 = 6912 dimensions
- With logistic regression and basic neural networks, we'll use a flat 2304 vector
  - Ignores spatial relationships, just considers each individual pixel intensity
  - [- row 1 -] (1-48)

    [- row 2 -] (49-96)

    ...
  - Becomes: [- row 1 - - row 2 - - row 3 - ...] (1-2304)
- With convolutional neural networks we keep the original image shape

Challenges

- Deep learning doesn't really give you plug-and-chug black boxes
- Correct learning rate?
  - Too high → NaN
  - Too low → Slow convergence
- Correct regularization
  - Too high → Just makes W very small, ignores actual pattern
  - Too low → Cost may still explode to NaN
- How many epochs?
  - Stop too early → Steep slope, not at minimum error
  - Stop too late → No effect on error, takes too long

Class Imbalance

- In facial expression recognition problem, we have 547 samples from class 1 and 4953 samples from class 0 (class 1 is the only class with substantially less samples)
- Why is this a problem?
- Medical testing:
  - Disease is present in 1% of population
  - Predict "no disease" every time → classifier is 99% accurate
  - Hasn't learned anything
- But we don't use accuracy to train, we use cross-entropy
  - Same problem

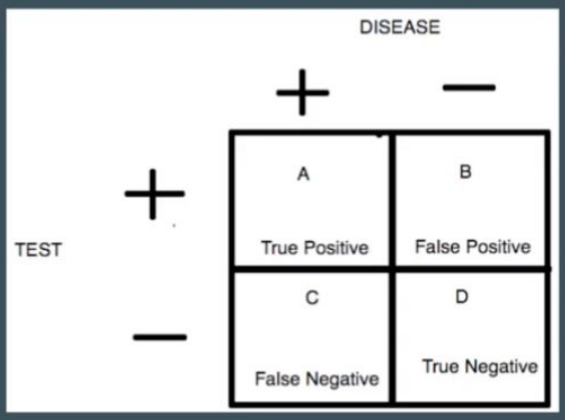
Solving class imbalance

- Suppose we have 1000 samples from class 1, 100 samples from class 2
- Method 1) Pick 100 samples from class 1, now we have 100 vs. 100
- Method 2) Repeat class 2 10 times, now we have 1000 vs. 1000
- Same *expected* error rate
- But method 2 is better (less variance, more data)
- Other options to expand class 2:
  - Add Gaussian noise
  - Add invariant transformations (shift left, right, rotate, etc.)

Other accuracy measures

- Measures can allow for class imbalance
- Medical field and information retrieval
- Assumes binary classification
- Basic idea:
- Maximize TP, TN
- Minimize FP, FN

Note: classification rate = (TP+TN)/(TP+TN+FP+FN)



Sensitivity and Specificity

- Used in medical field
- Sensitivity:

True positive rate = TPR = TP / (TP + FN)

- Specificity:

True negative rate = TNR = TN / (TN + FP)

Precision and Recall

- Used in information retrieval
- Precision

TP / (TP + FP)

- Recall

TP / (TP + FN)

- Note: recall = sensitivity

F1-score

- Combines precision and recall into a balanced measure

F1 = 2 * (precision * recall) / (precision + recall)

- Harmonic mean of precision and recall

ROC and AUC

- In logistic regression we use 0.5 as a threshold, makes sense because P(Y=1|X) > 0.5 → guess 1, P(Y=1|X) < 0.5 → guess 0
- But we can use any threshold
- Different thresholds → different TPR and FPR
- Receiver operating characteristic (ROC curve is a plot of these for every value of threshold between 0 and 1)
- Area under curve = AUC
  - 1 = perfect classifier, 0.5 = random guessing