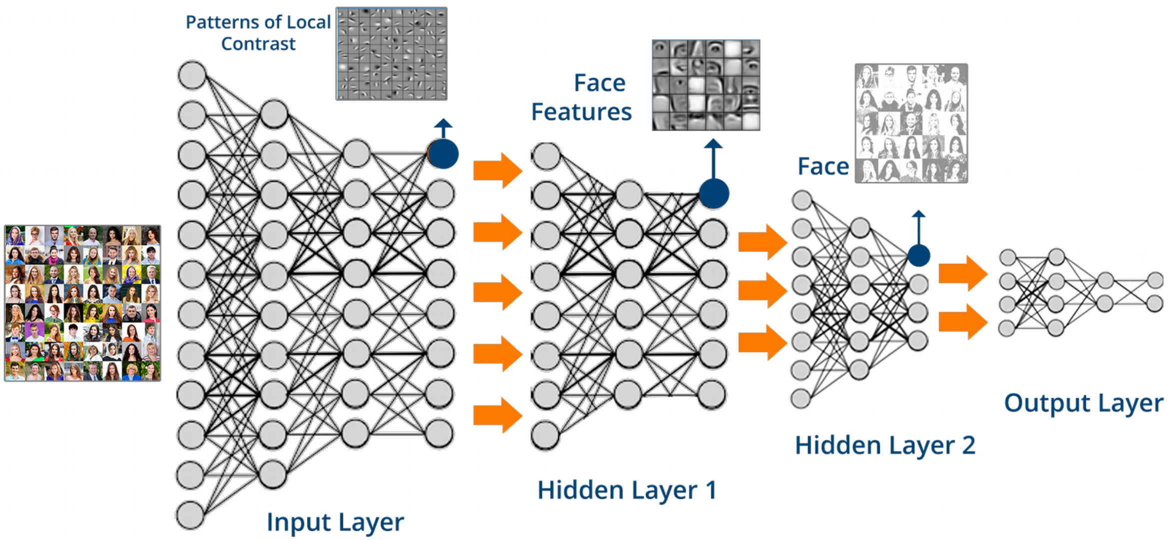
Deep Learning Overview

Robert Platt
Northeastern University



Problem we want to solve

<u>Scenario</u>:

- A pattern exists
- We don't know what it is, but we have a bunch of examples of the pattern

<u>Machine Learning problem</u>: find a rule for making predictions from the data

<u>Examples</u>:

Image classification: Given a dataset of images and corresponding class labels, learn a rule for predicting the image class given the image

Spam detection: Given a dataset of email text and corresponding spam/ham labels, learn a rule for predicting whether spam/ham given the text of a new email

Problem we want to solve

Scenario:

- A pattern exists
- We don't know what it is, but we have a bunch of examples of the pattern

Machine Learning problem: find a rule for making predictions from the data

Classification vs regression:

– if a labels are discrete, then we have a *classification* problem

– if the labels are real-valued, then we have a *regression* problem

Problem we want to solve

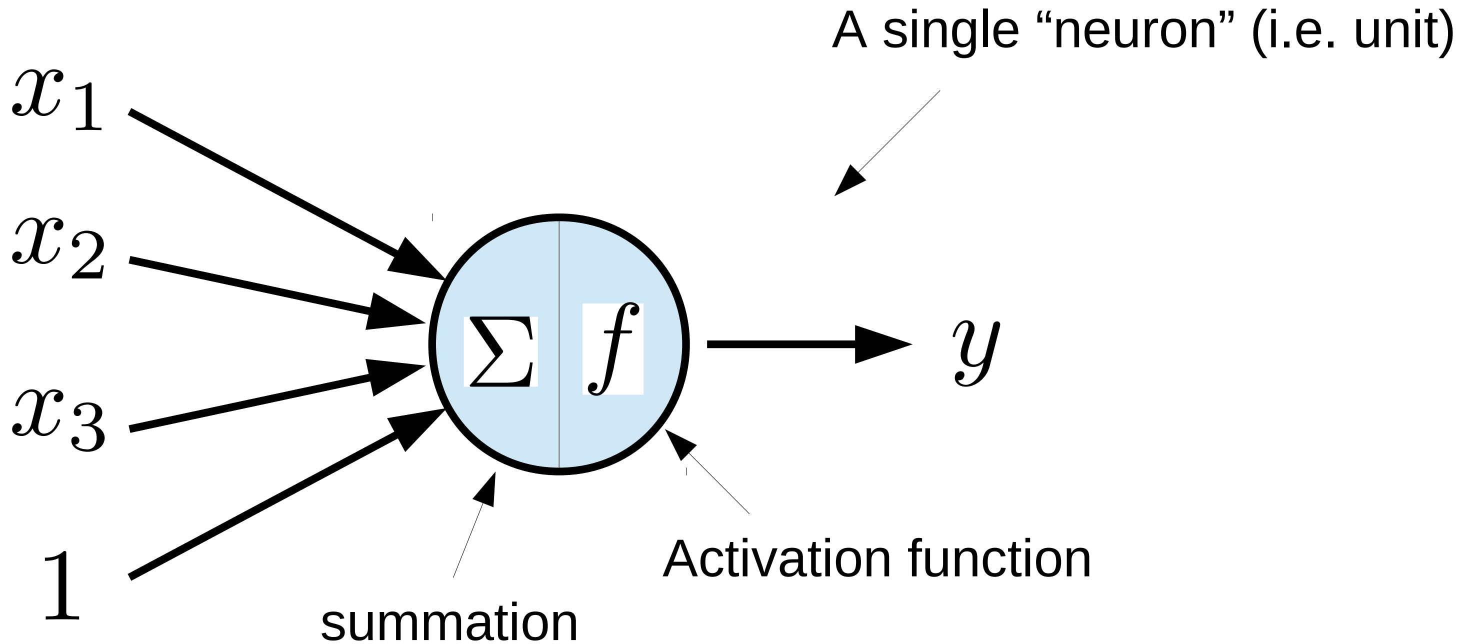
Input: x

Label: y

Data: $\mathbb{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$

Given $\mathbb{D}$, find a rule for predicting x given y

A single unit neural network

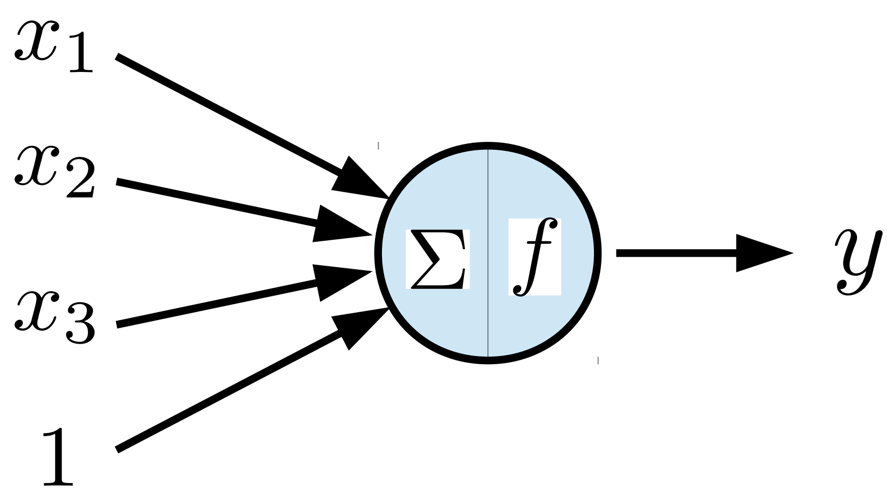


$$y = f(w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + b)$$

$$= f(w^T x + b)$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \qquad w = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix}$$

A single unit neural network

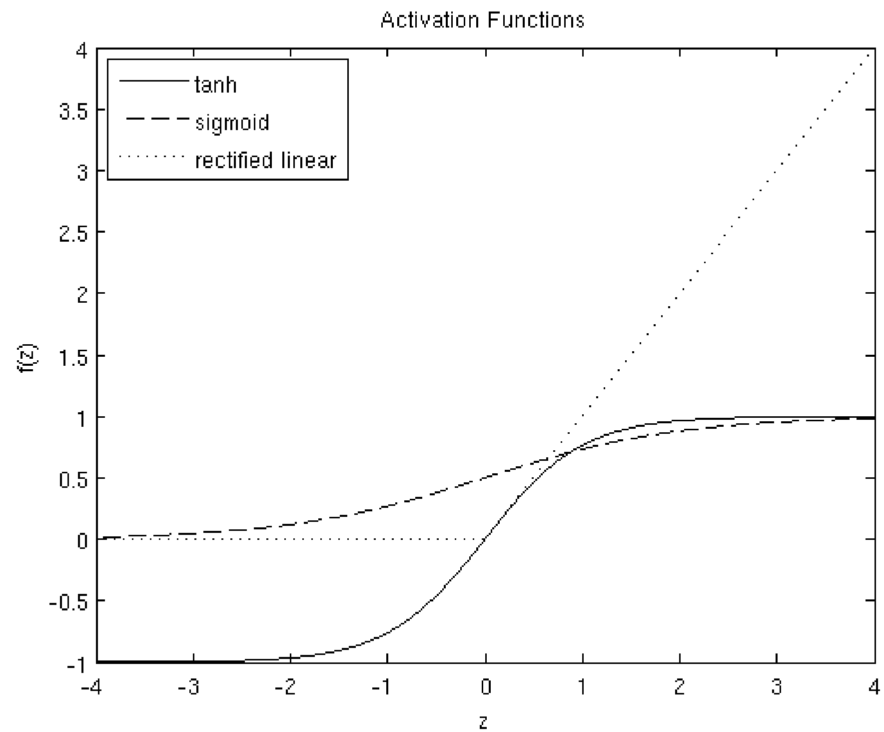


Different activation functions:

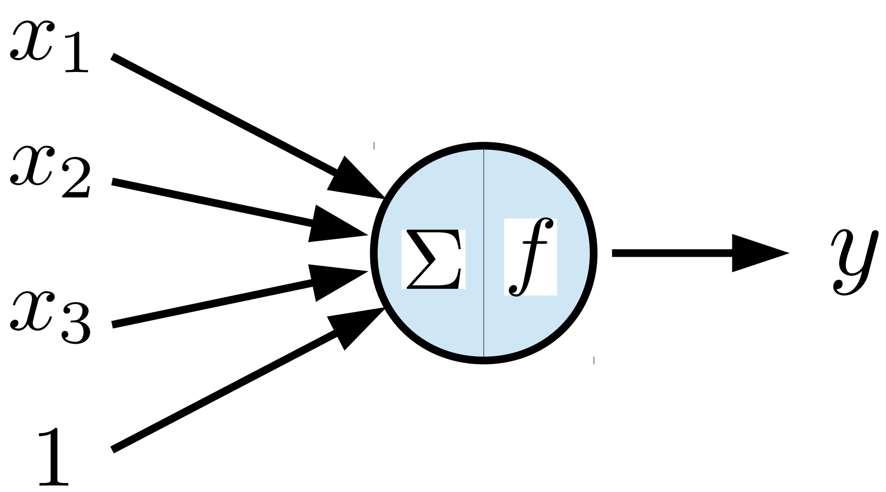
– sigmoid $$f(z) = \frac{1}{1+e^{-z}}$$

– tanh $$f(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

– rectified linear unit (ReLU) $$f(z) = \max(0, z)$$



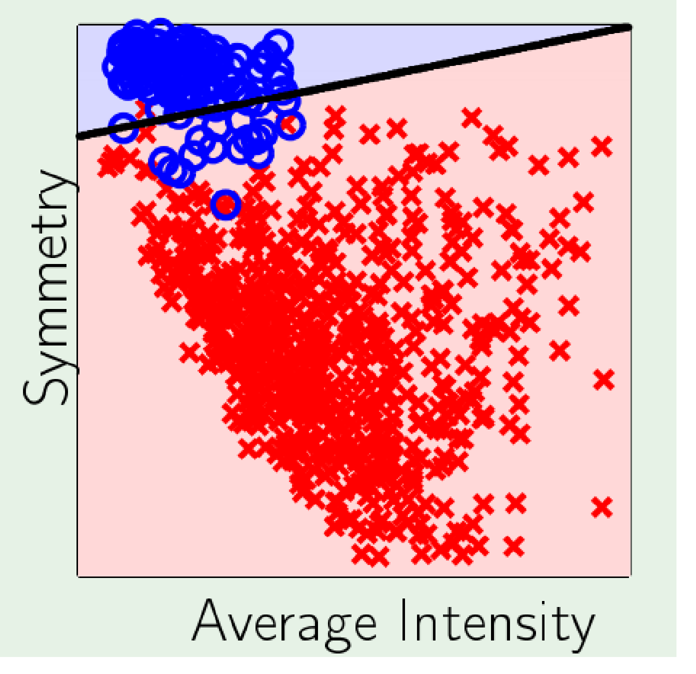
A single unit neural network



One-layer neural network has a simple interpretation: linear classification.

$$y = f(w^T x + b)$$

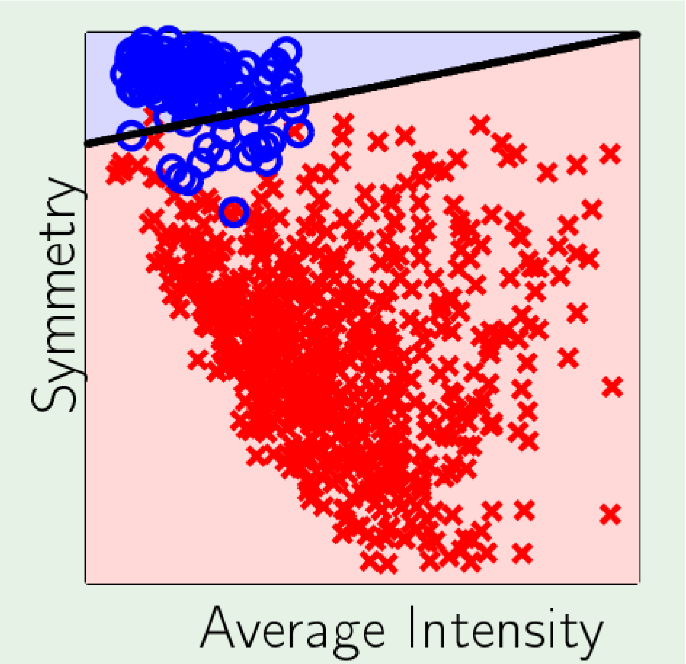
X_1 == symmetry
X_2 == avg intensity
Y == class label (binary)



Think-pair-share

$$y = f(w^T x + b)$$

X_1 == symmetry
X_2 == avg intensity
Y == class label (binary)



What do *w* and *b* correspond to in this picture?

Training

Given a dataset: $D = \{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\}$

Define loss function: $L(x^i, y^i; w, b) = \frac{1}{2}(y^i - f(w^T x^i + b))^2$

Training

Given a dataset: $D = \{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\}$

Define loss function: $L(x^i, y^i; w, b) = \frac{1}{2}(y^i - f(w^T x^i + b))^2$

Loss function tells us how well the network classified x^i

Training

Given a dataset: $D = \{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\}$

Define loss function: $L(x^i, y^i; w, b) = \frac{1}{2}(y^i - f(w^T x^i + b))^2$

Loss function tells us how well the network classified x^i

Method of training: adjust *w, b* so as to minimize the net loss over the dataset

i.e.: adjust *w, b* so as to minimize: $\sum_{(x^i, y^i) \in D} L(x^i, y^i; w, b)$

The closer to zero, the better the classification

Training

Method of training: adjust *w, b* so as to minimize the net loss over the dataset

i.e.: adjust *w, b* so as to minimize: $$\sum_{(x^i, y^i) \in D} L(x^i, y^i; w, b)$$

How?

Training

Method of training: adjust *w, b* so as to minimize the net loss over the dataset

i.e.: adjust *w, b* so as to minimize: $$\sum_{(x^i,y^i)\in D} L(x^i, y^i; w, b)$$

How?

Gradient Descent

Gradient descent

Suppose someone gives you an unknown function F(x)
– you want to find a minimum for F
– but, you do not have an analytical description of F(x)

Use gradient descent!
– all you need is the ability to evaluate F(x) and its gradient at any point x

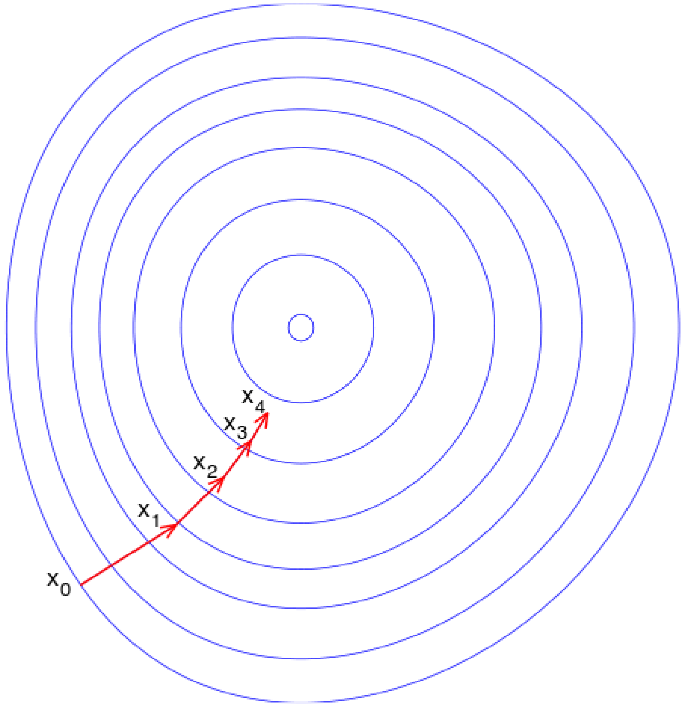
1. pick x_0 at random

2. $x_1 = x_0 - \alpha \nabla_x F(x_0)$

3. $x_2 = x_1 - \alpha \nabla_x F(x_1)$

4. $x_3 = x_2 - \alpha \nabla_x F(x_2)$

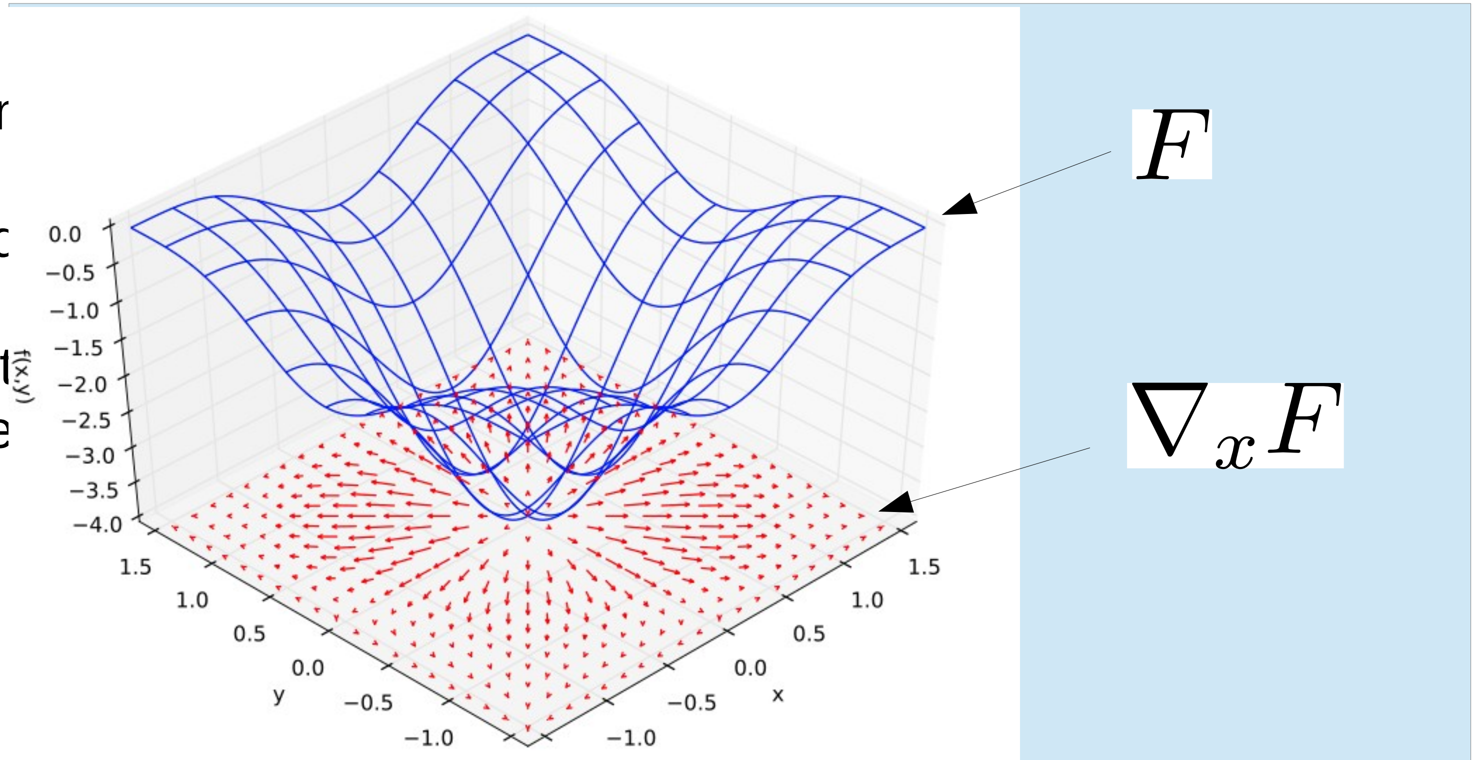
5. ...



Gradient descent

Suppose sor
– you want
– but, you c

Use gradient
– all you ne



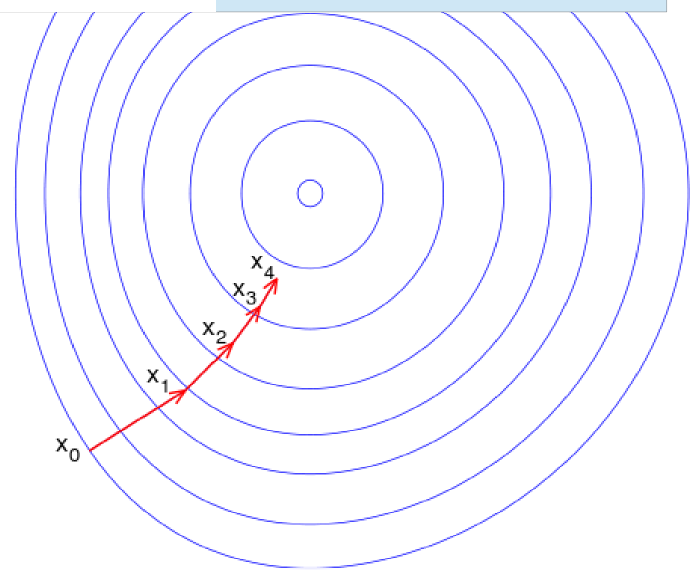
1. pick x_0 at random

2. $x_1 = x_0 - \alpha \nabla_x F(x_0)$

3. $x_2 = x_1 - \alpha \nabla_x F(x_1)$

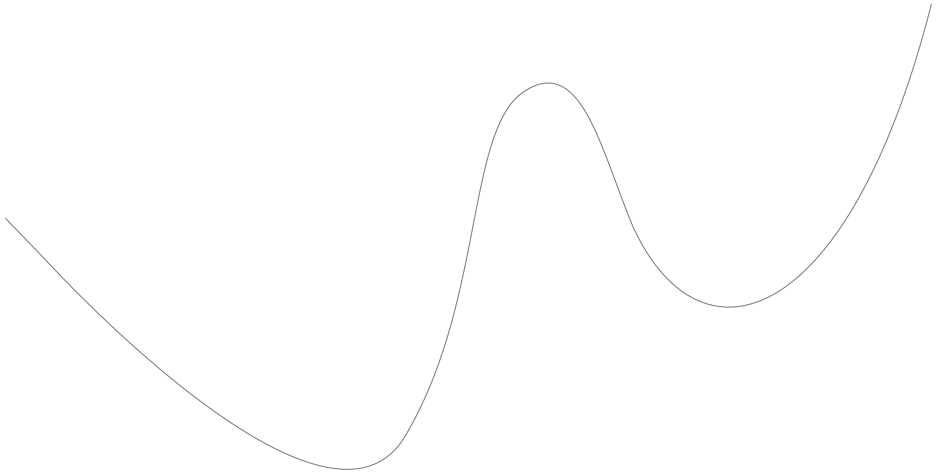
4. $x_3 = x_2 - \alpha \nabla_x F(x_2)$

5. ...

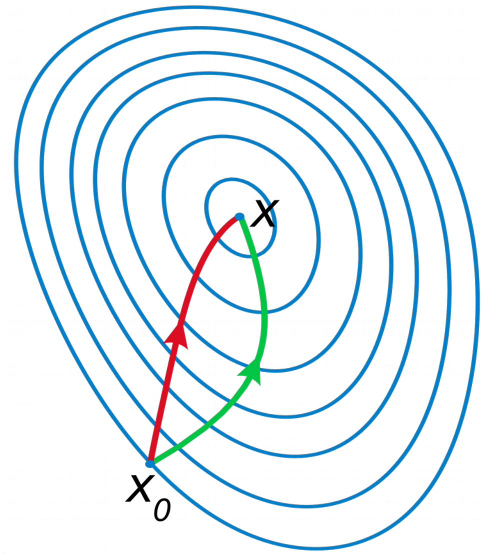


Think-pair-share

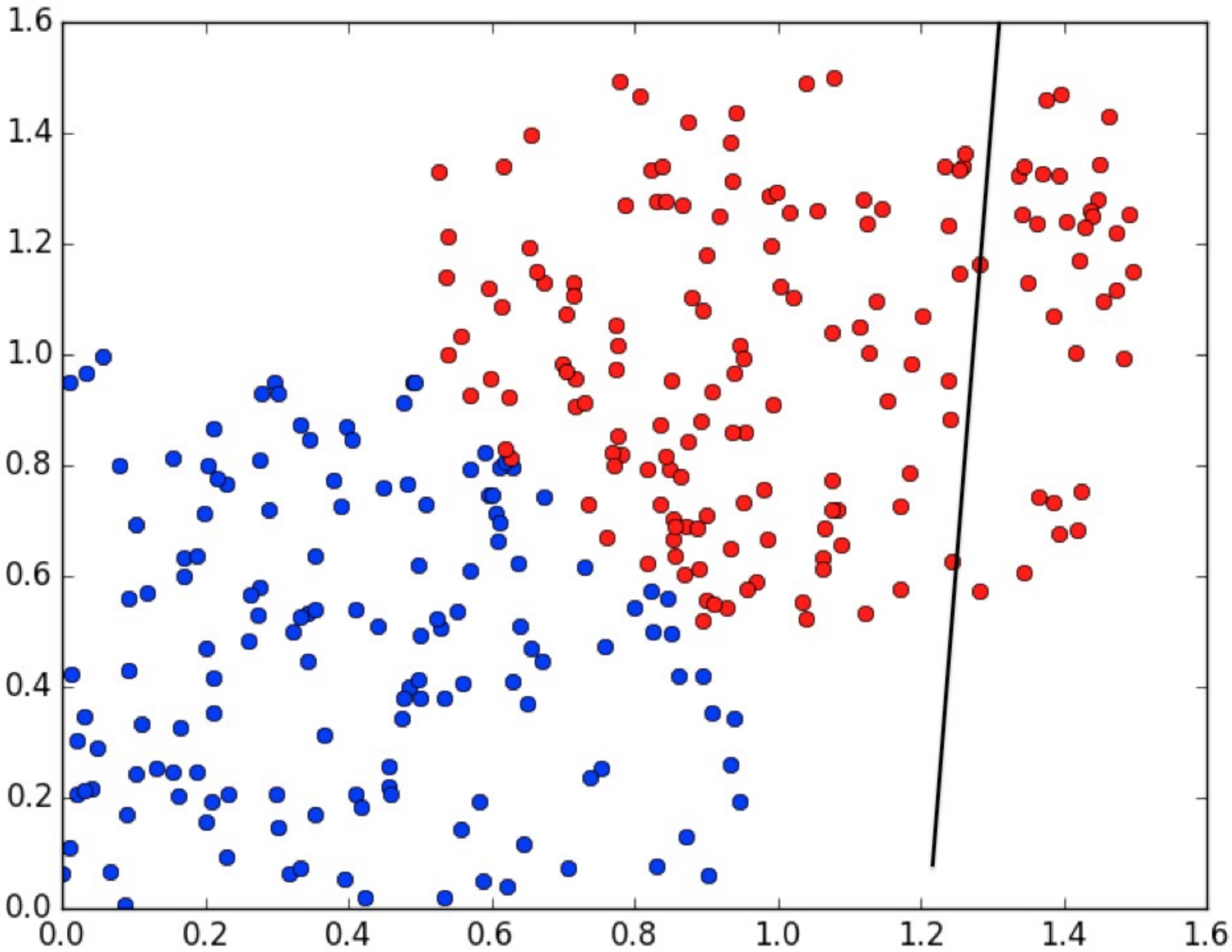
1. Label all the points where gradient descent could converge to:



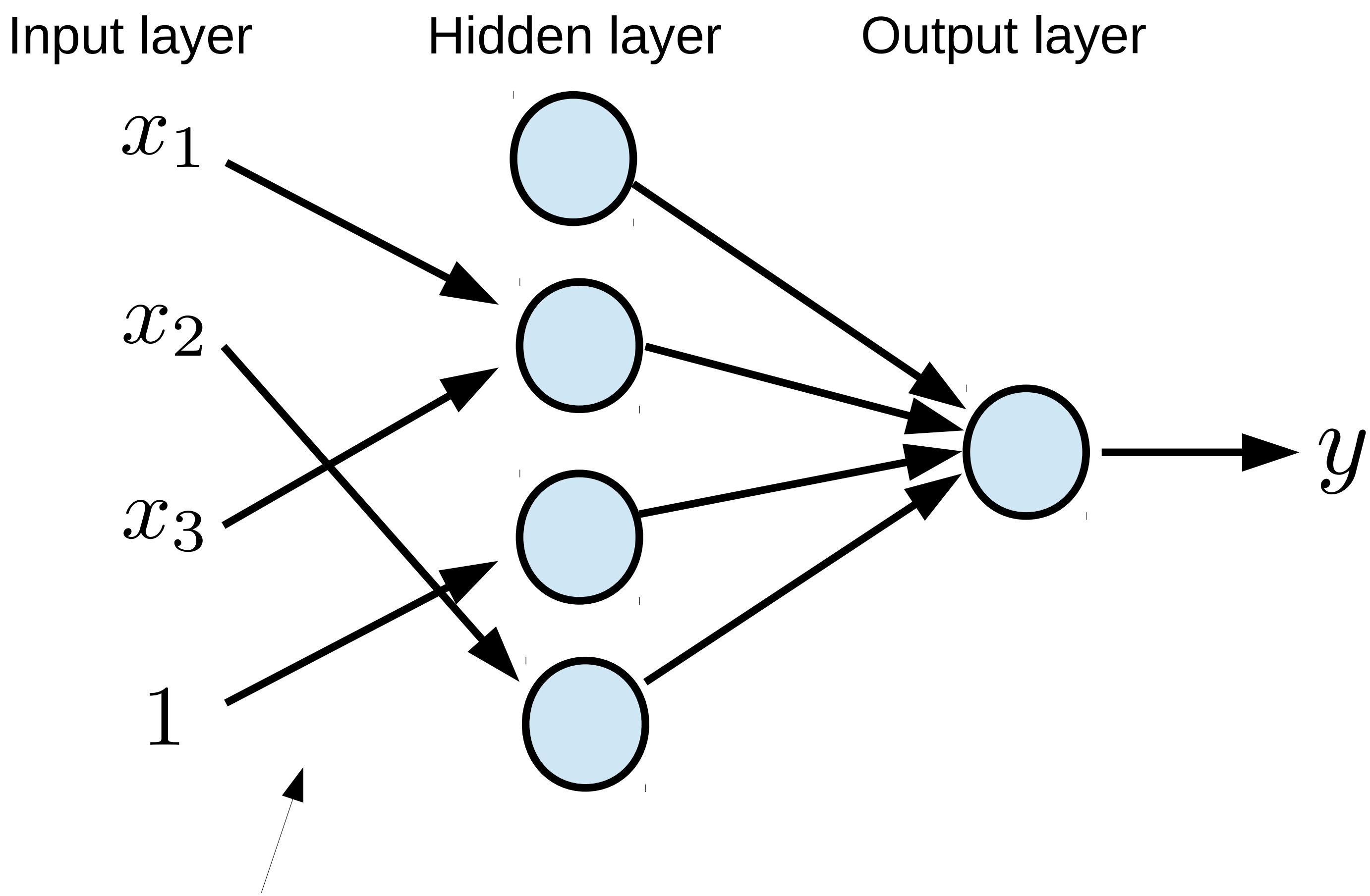
2. Which path does gradient descent take?



Training a one-unit neural network

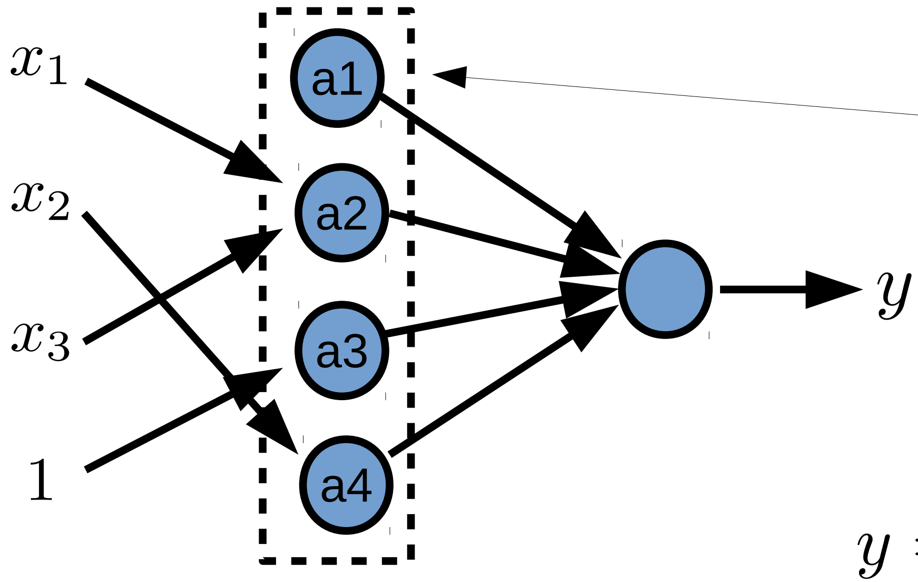


Going deeper: a one layer network



Each hidden node is
connected to every input

Going deeper: a one layer network



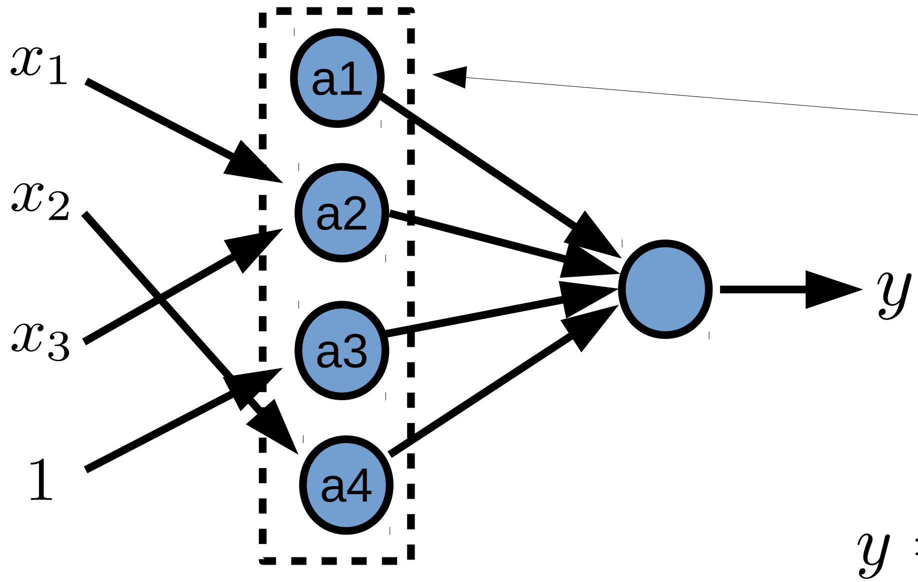
$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

Vector of hidden layer activations

$$y = f({w^{(2)}}^T a + b)$$

Single activation: $a_i = f({w_i^{(1)}}^T x + b_i^{(1)})$

Going deeper: a one layer network



$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

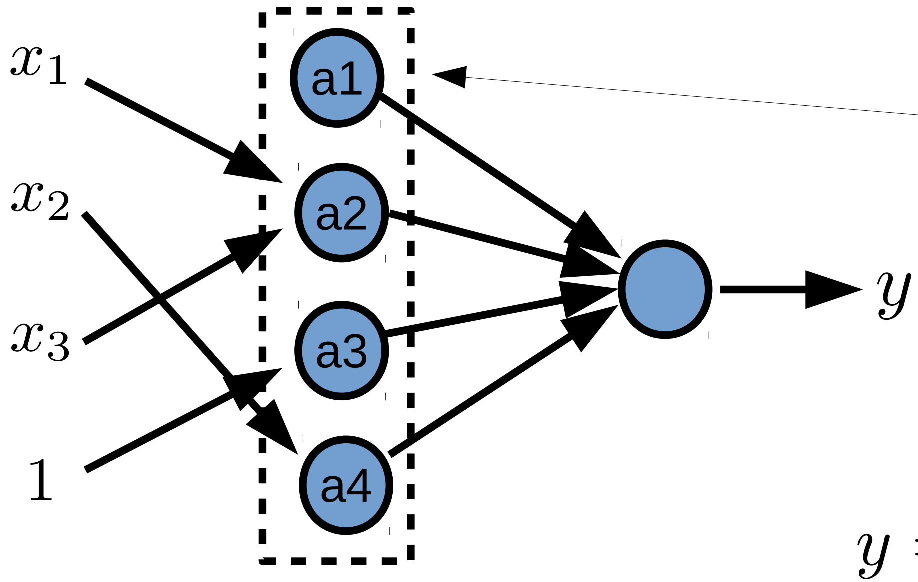
Vector of hidden layer activations

$$y = f({w^{(2)}}^T a + b)$$

Single activation: $a_i = f({w_i^{(1)}}^T x + b_i^{(1)})$

This is called “forward propogation”

Think-pair-share



$$a = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

Vector of hidden layer activations

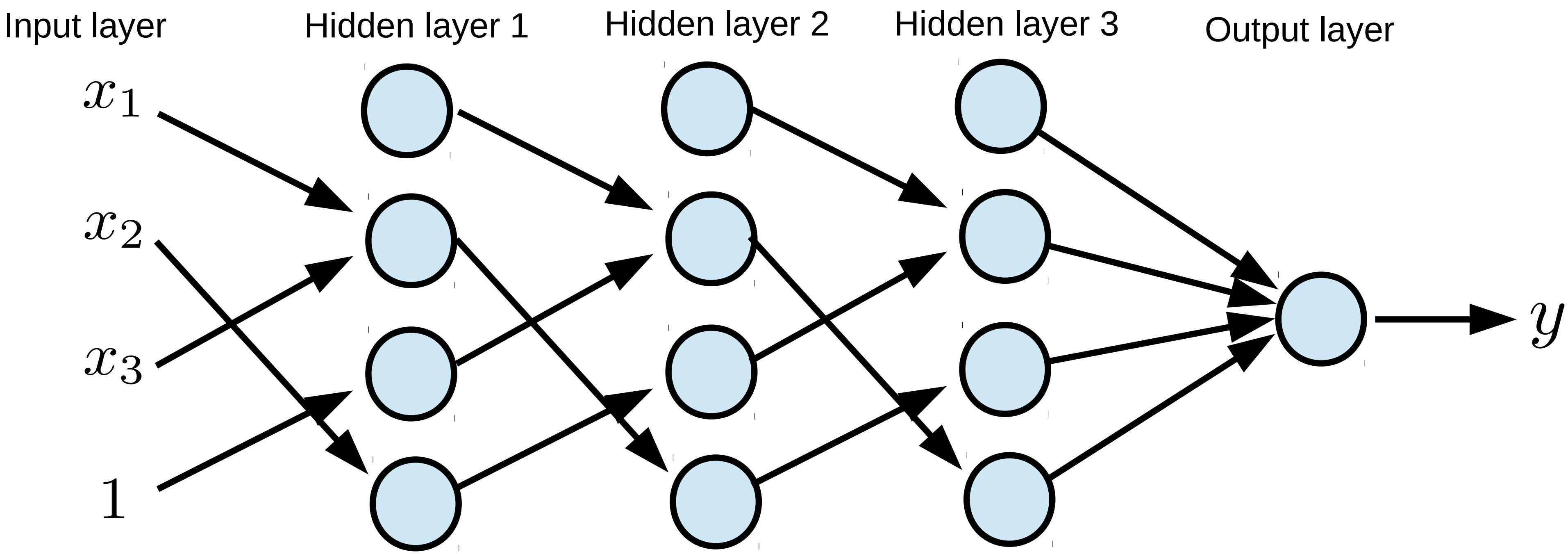
$$y = f({w^{(2)}}^T a + b)$$

Single activation: $a_i = f({w_i^{(1)}}^T x + b_i^{(1)})$

<u>Write an expression for *y* in terms of *x*, *f*, and the weights</u>

<u>Hint:</u> use matrix multiplication

Can create networks of arbitrary depth...



– Forward propagation works the same for any depth network.

– Whereas a single output node corresponds to linear classification, adding hidden nodes makes classification non-linear

How do we train multi-layer networks?

Almost the same as in the single-node case...

Do gradient descent on dataset:

1. repeat

2. $$w \leftarrow w - \alpha \frac{1}{n} \sum_{(x^i, y^i) \in D} \nabla_w L(x^i, y^i; w, b)$$

3. $$b \leftarrow b - \alpha \frac{1}{n} \sum_{(x^i, y^i) \in D} \nabla_b L(x^i, y^i; w, b)$$

4. until converged

Now, we're doing gradient descent on all weights/biases in the network
– not just a single layer

– this is called *backpropagation*

Stochastic gradient descent: mini-batches

A batch is typically between 32 and 128 samples

1. repeat

2. randomly sample a mini-batch: $B \subset D$

3. $$w \leftarrow w - \alpha \frac{1}{n} \sum_{(x^i, y^i) \in D} \nabla_w L(x^i, y^i; w, b)$$

4. $$b \leftarrow b - \alpha \frac{1}{n} \sum_{(x^i, y^i) \in D} \nabla_b L(x^i, y^i; w, b)$$

5. until converged

Training in mini-batches helps b/c:
– don’t have to load the entire dataset into memory
– training is still relatively stable
– random sampling of batches helps avoid local minima

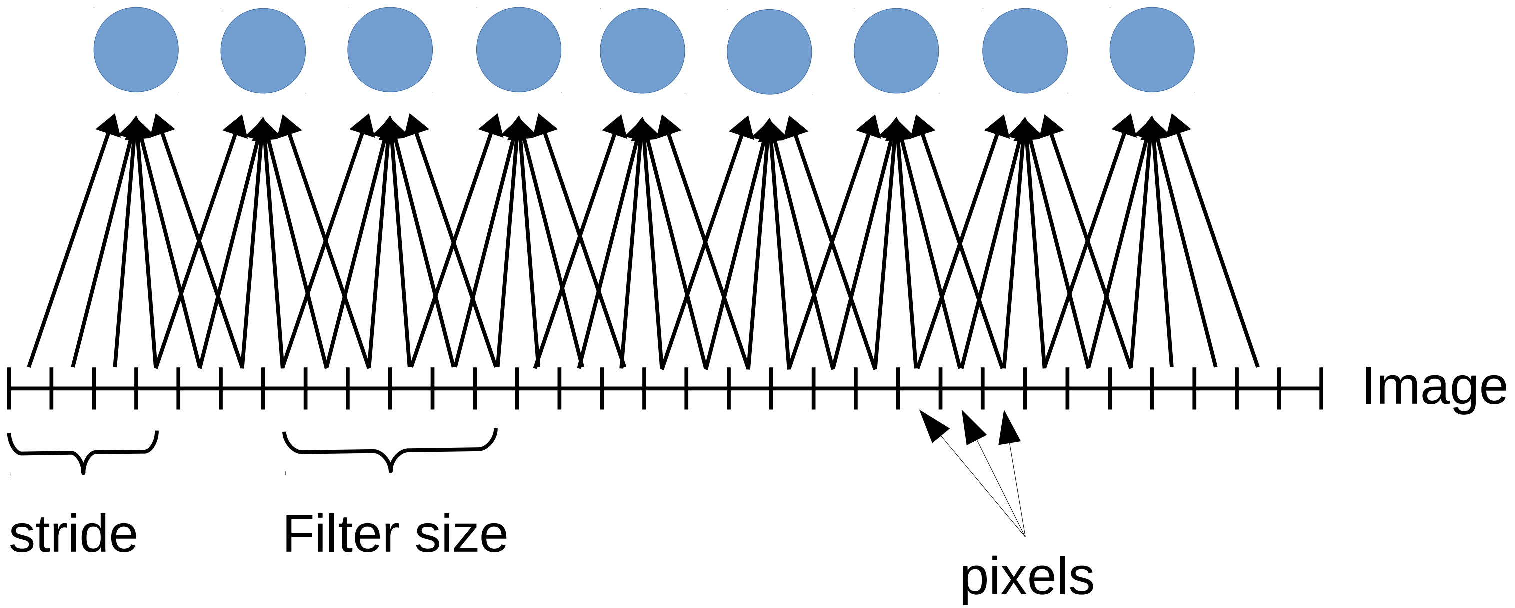
Convolutional layers

Deep multi-layer perceptron networks
– general purpose
– involve huge numbers of weights

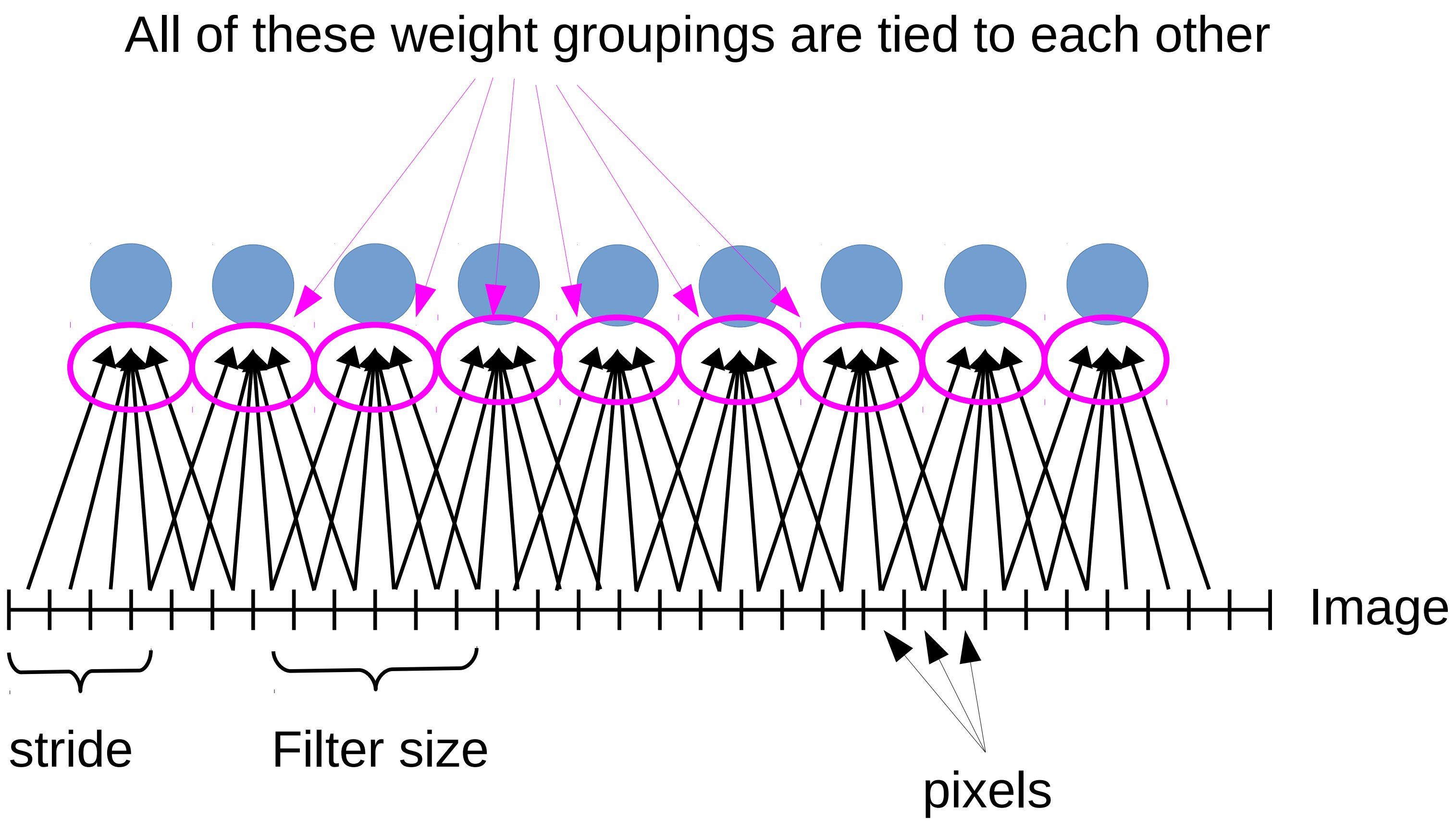
We want:
– special purpose network for image and NLP data
– fewer parameters
– fewer local minima

Answer: convolutional layers!

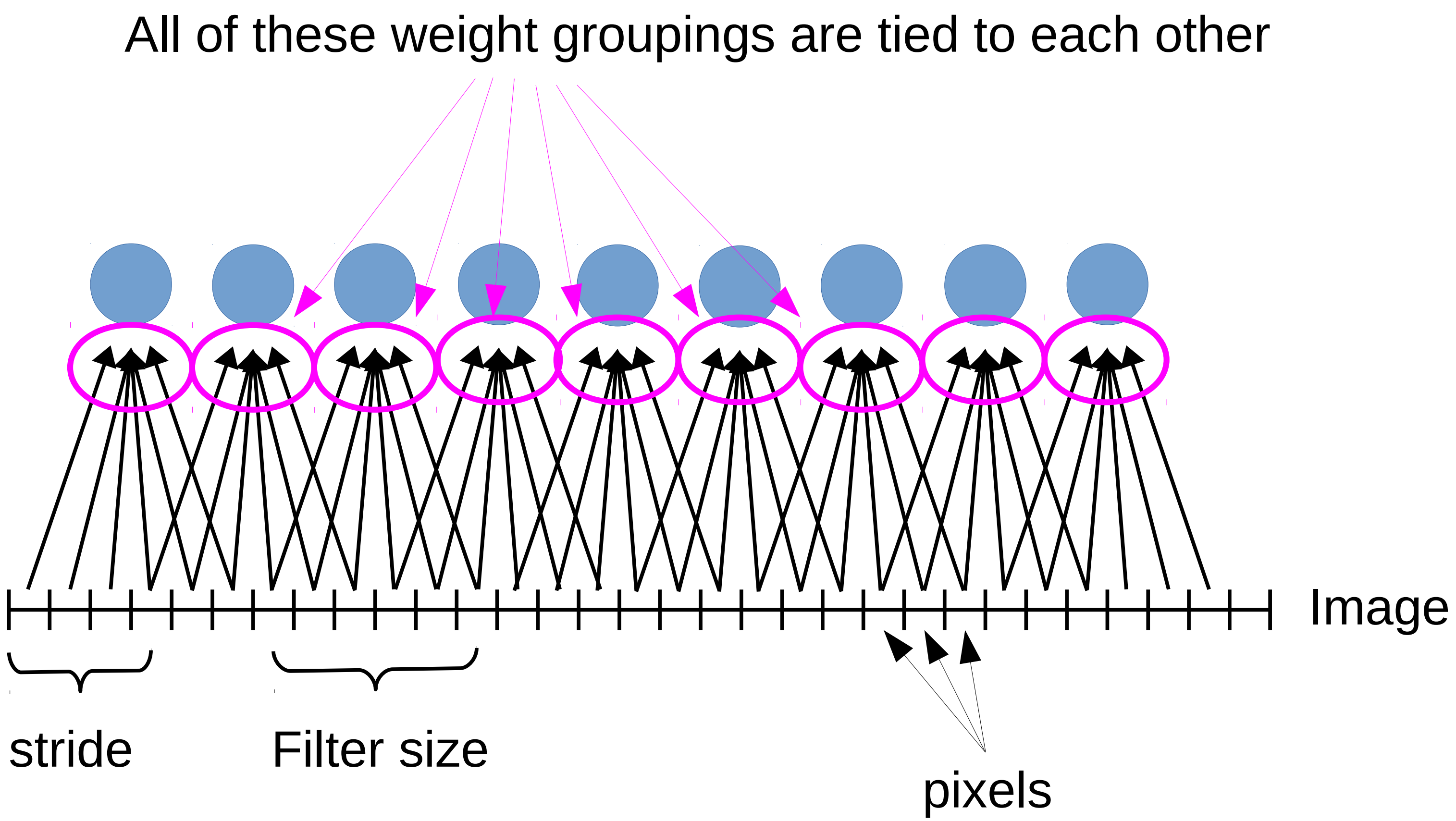
Convolutional layers



Convolutional layers



Convolutional layers



Because of the way weights are tied together

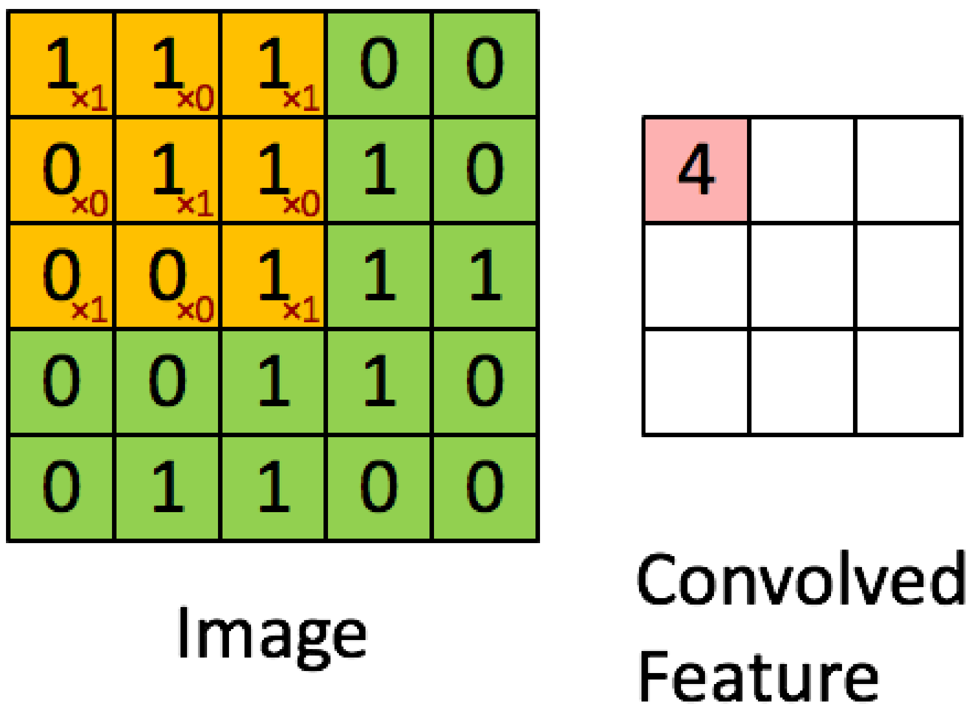
– reduces number of parameters (dramatically)

– encodes a prior on structure of data

In practice, convolutional layers are essential to computer vision...

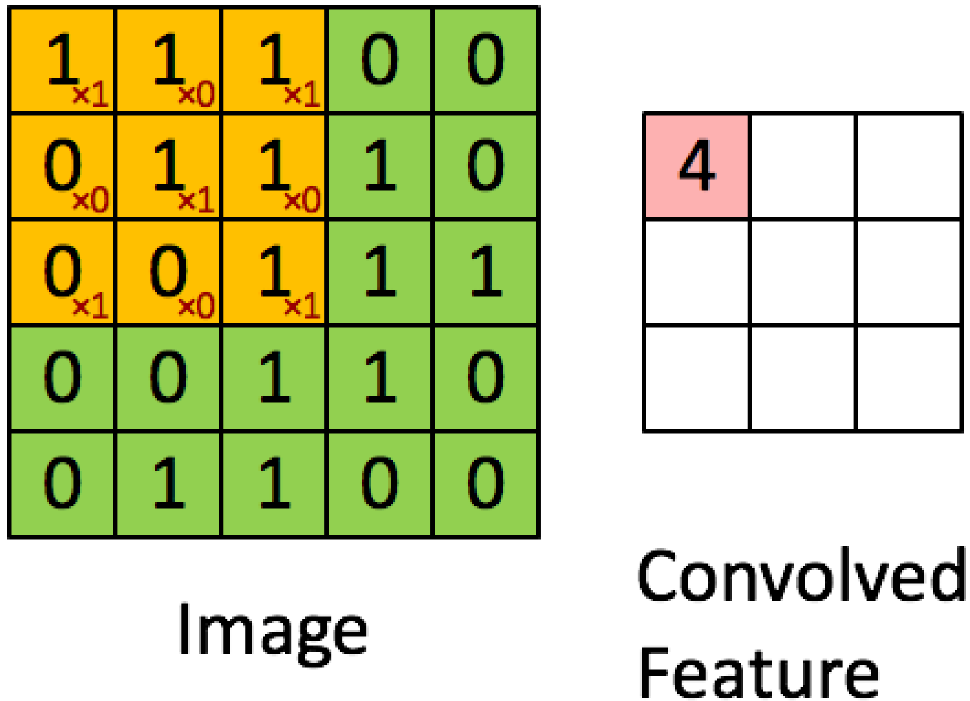
Convolutional layers

Two dimensional example:



Why do you think they call this “convolution”?

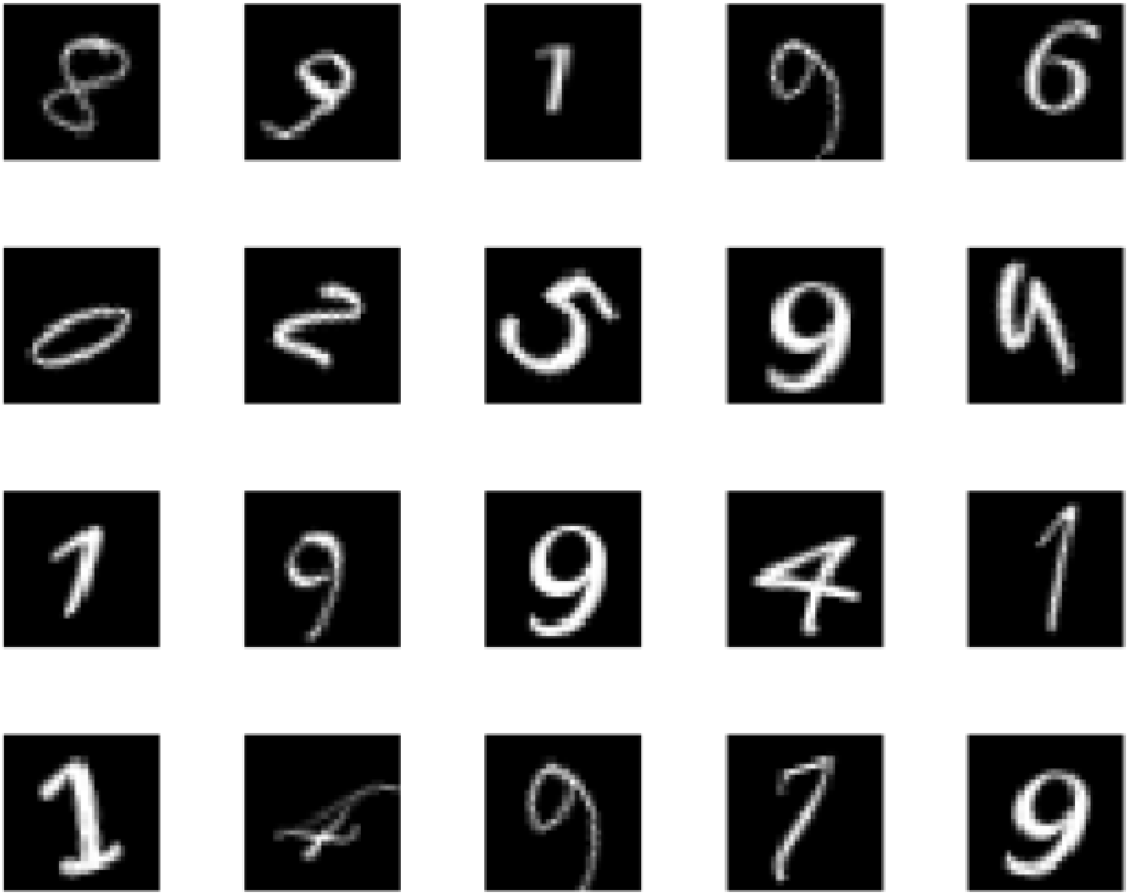
Think-pair-share



What would the convolved feature map be for <u>this</u> kernel?

$$\begin{matrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{matrix}$$

Example: MNIST digit classification with LeNet

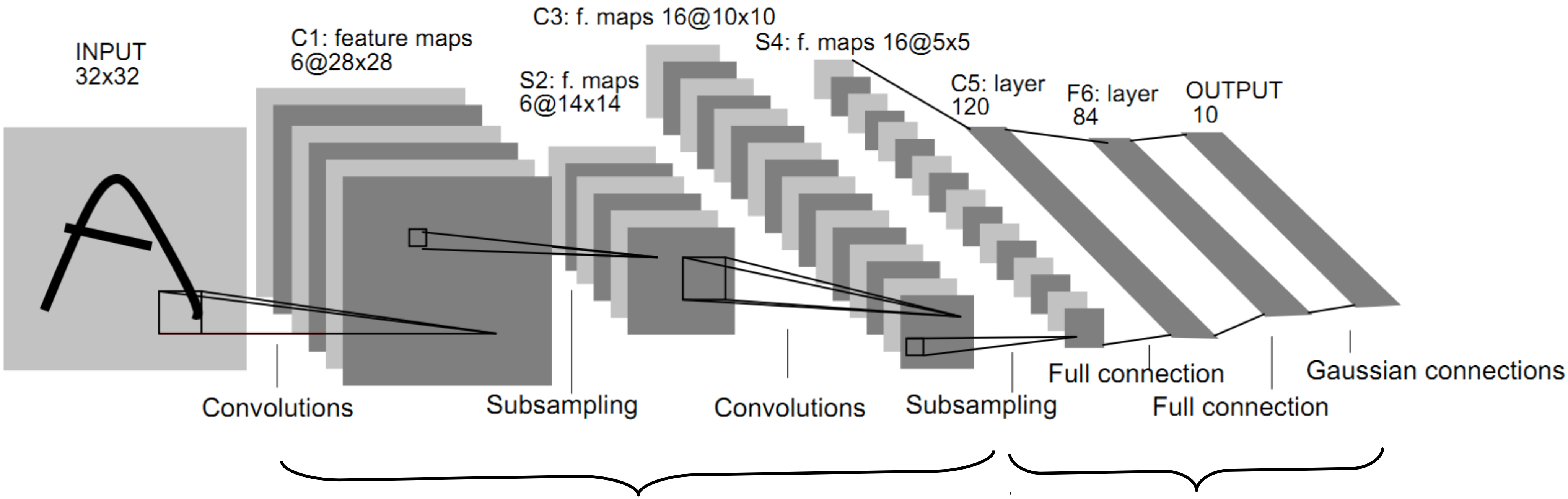


MNIST dataset: images of 10,000 handwritten digits

Objective: classify each image as the corresponding digit

Example: MNIST digit classification with LeNet

LeNet:



two convolutional layers
– conv, relu, pooling

two fully connected layers
– relu
– last layer has logistic
activation function

Example: MNIST digit classification with LeNet

Load dataset, create train/test splits

```
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos',...
    'nndatasets','DigitDataset');
digitData = imageDatastore(digitDatasetPath,...
    'IncludeSubfolders',true,'LabelSource','foldernames');

trainNumFiles = 750;
[trainDigitData,valDigitData] = splitEachLabel(digitData,trainNumFiles,'randomize');
```

Example: MNIST digit classification with LeNet

<u>Define the neural network structure:</u>

```
layers = [
    imageInputLayer([28 28 1])
    convolution2dLayer(3,16,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,32,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,64,'Padding',1)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(50)
    reluLayer
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

Input

Conv1

Conv2

FC1

FC2

```
options = trainingOptions('sgdm',...
'MaxEpochs',3, ...
'ValidationData',valDigitData,...
'ValidationFrequency',30,...
'Verbose',true,...
'ExecutionEnvironment','gpu',...
'Plots','training-progress');
```

Example: MNIST digit classification with LeNet

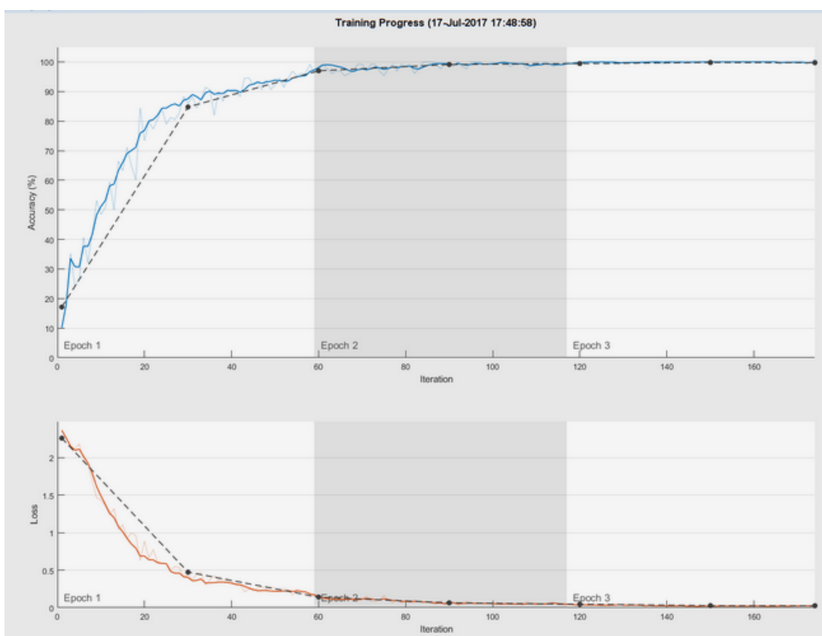
<u>Train network, classify test set, measure accuracy</u>

– notice we test on a different set (a holdout set) than we trained on

```
net = trainNetwork(trainDigitData,layers,options);

predictedLabels = classify(net,valDigitData);
valLabels = valDigitData.Labels;

accuracy = sum(predictedLabels == valLabels)/numel(valLabels);
```



Using the GPU makes
a huge differece...

Deep learning packages

You don’t need to use Matlab (obviously)

Tensorflow is probably the most popular platform

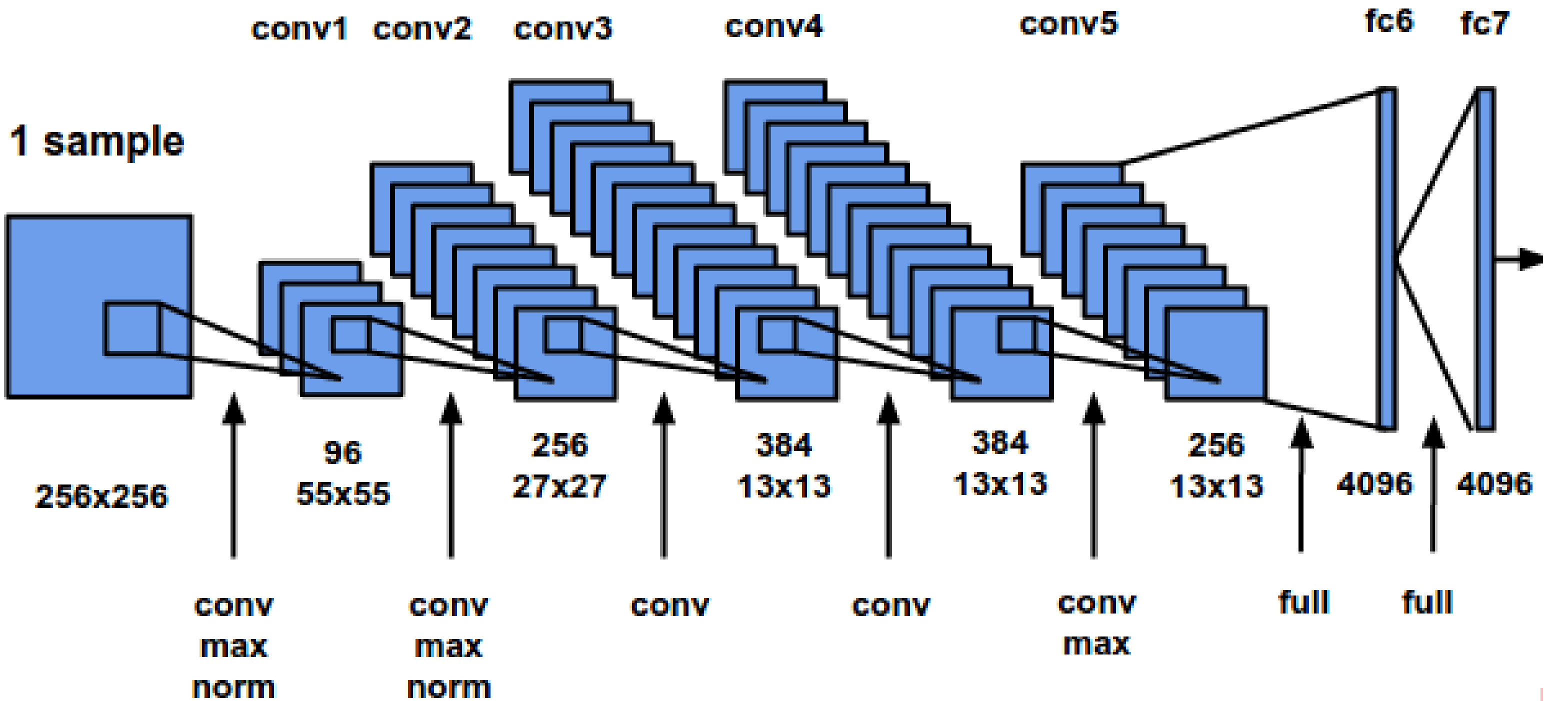
Caffe and Theano are also big

Another example: image classification w/ AlexNet

ImageNet dataset: millions of images of objects

Objective: classify each image as the corresponding object (1k categories in ILSVRC)

Another example: image classification w/ AlexNet



AlexNet has 8 layers: five conv followed by three fully connected

Another example: image classification w/ AlexNet

```
 1   'data'     Image Input                   227x227x3 images with 'zerocenter' normalization
 2   'conv1'    Convolution                   96 11x11x3 convolutions with stride [4  4] and padding [0  0]
 3   'relu1'    ReLU                          ReLU
 4   'norm1'    Cross Channel Normalization   cross channel normalization with 5 channels per element
 5   'pool1'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0]
 6   'conv2'    Convolution                   256 5x5x48 convolutions with stride [1  1] and padding [2  2]
 7   'relu2'    ReLU                          ReLU
 8   'norm2'    Cross Channel Normalization   cross channel normalization with 5 channels per element
 9   'pool2'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0]
10   'conv3'    Convolution                   384 3x3x256 convolutions with stride [1  1] and padding [1  1]
11   'relu3'    ReLU                          ReLU
12   'conv4'    Convolution                   384 3x3x192 convolutions with stride [1  1] and padding [1  1]
13   'relu4'    ReLU                          ReLU
14   'conv5'    Convolution                   256 3x3x192 convolutions with stride [1  1] and padding [1  1]
15   'relu5'    ReLU                          ReLU
16   'pool5'    Max Pooling                   3x3 max pooling with stride [2  2] and padding [0  0]
17   'fc6'      Fully Connected               4096 fully connected layer
18   'relu6'    ReLU                          ReLU
19   'drop6'    Dropout                       50% dropout
20   'fc7'      Fully Connected               4096 fully connected layer
21   'relu7'    ReLU                          ReLU
22   'drop7'    Dropout                       50% dropout
23   'fc8'      Fully Connected               1000 fully connected layer
24   'prob'     Softmax                       softmax
25   'output'   Classification Output         crossentropyex with 'tench', 'goldfish', and 998 other classes
```

AlexNet has 8 layers: five conv followed by three fully connected

Another example: image classification w/ AlexNet

AlexNet won the 2012 ILSVRC challenge
– sparked the deep learning craze