Data Science Training

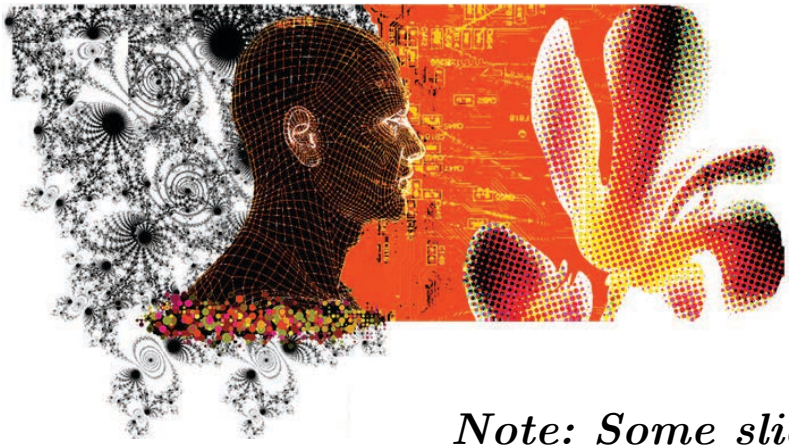
November 2017

Classification with Neural Networks

Xavier Bresson

Data Science and AI Research Centre

NTU, Singapore



http://data-science-optum17.tk

Note: Some slides are from Li, Karpathy, Johnson's course on Deep Learning.

Outline

- **The Classification Problem**
- Nearest Neighbor Classifier
- Linear Classifier
- Loss Function
- Softmax Classifier
- Neural Network Classifier
- Brain Analogy
- Conclusion

Classification Problem

Q: What it the classification problem?

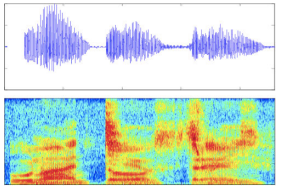
- Classification is a **core problem** in many applications:

  (1) Computer Vision: Image classification

  Image → Class (original deep learning [Hinton-et.al'12])

  (2) Speech: Sound recognition

  Sound → Class (deep learning [Dahl-et.al'12])



  (3) Text document: Text categorization

  Text → Class (Wikipedia analysis)

  (4) Neuroscience: Brain functionality

  Activation pattern → Class (vision, hearing, body control)

- **Pipeline** of classification models:

  Raw data → Feature extraction → Classifier function

Image Classification

- We will consider the **image classification** problem in Computer Vision as a **generic** classification problem (generalization will be discussed later).

- **Problem:**

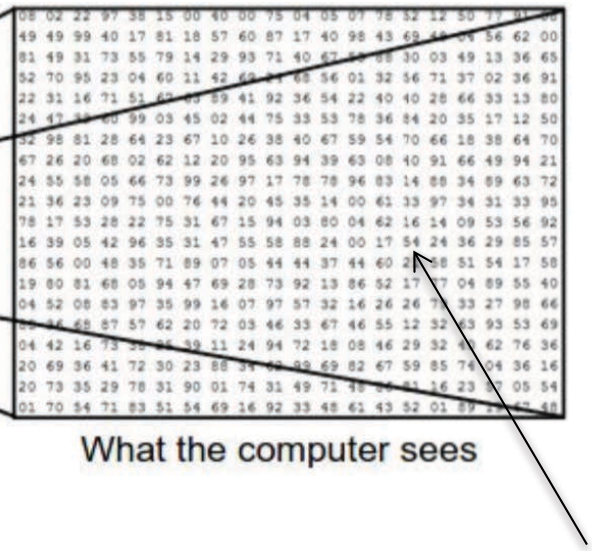
(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

Image Classification → cat

Main Challenge

- **Bridge the semantic gap** between *raw data* (N-D array of numbers) and *cognitive/human understanding*.

What humans see [cat]



What the computer sees

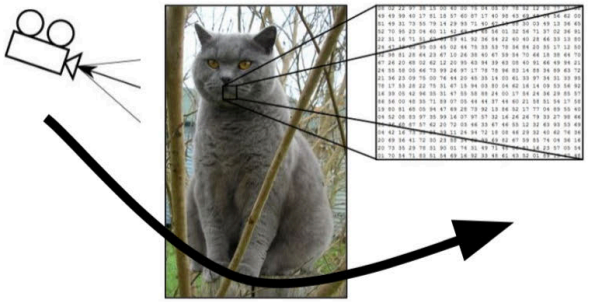
Images are represented as 3D arrays of numbers, with integers between [0, 255].

Matrices of e.g. size 300x100x3.

Semantic Information is Invariant to Many Deformations

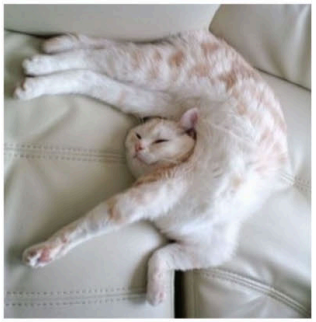
- **Deformations** in Computer Vision:

(1) Spatial variations
(translation, rotation, Scaling, shearing)



(2) Illumination changes

(3) Object deformation



(4) Occlusion

(5) Background clutter

(6) Intra-class variation

How to Solve the Classification Problem?

- **Highly challenging problem** (≠ sorting problem):

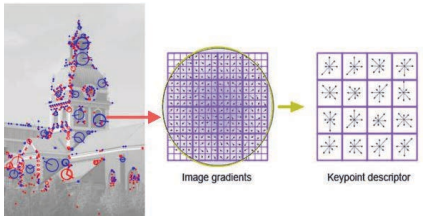
  In Computer Vision (CV), early works from 1950's (history later), only recently (2012) algorithms have achieved **super-human performances**.

- **Before 2012:**

  Many works exist, mostly based on **two separate steps**:

  (1) Handcrafting best possible filters/features (e.g. SIFT features)

  (2) Linear SVM classification on extracted features



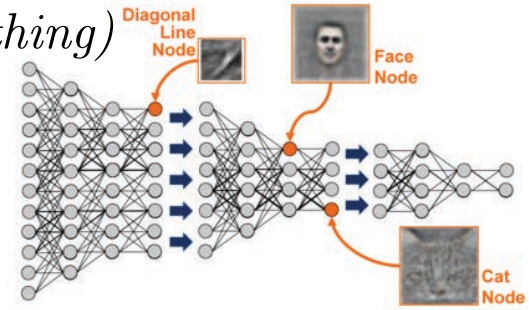
- **After 2012:**

  Deep learning revolution: Learn simultaneously

  (1) Filters/features from raw data (do not handcraft anything)

  (2) Linear SVM classification on extracted features

  → State-of-the-art in CV, speech recognition, etc



Classification is a Data-Driven Approach

- **Generic approach:**
  (1) Collect a training dataset of images and labels (training set).
  (2) Train an image classifier.
  (3) Evaluate classifier on test images (test set).

```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

def predict(model, test_images):
  # predict test_labels using the model...
  return test_labels
```

- **Note:** Collecting data is easy (big data era) but *labeling is time consuming.*

Outline

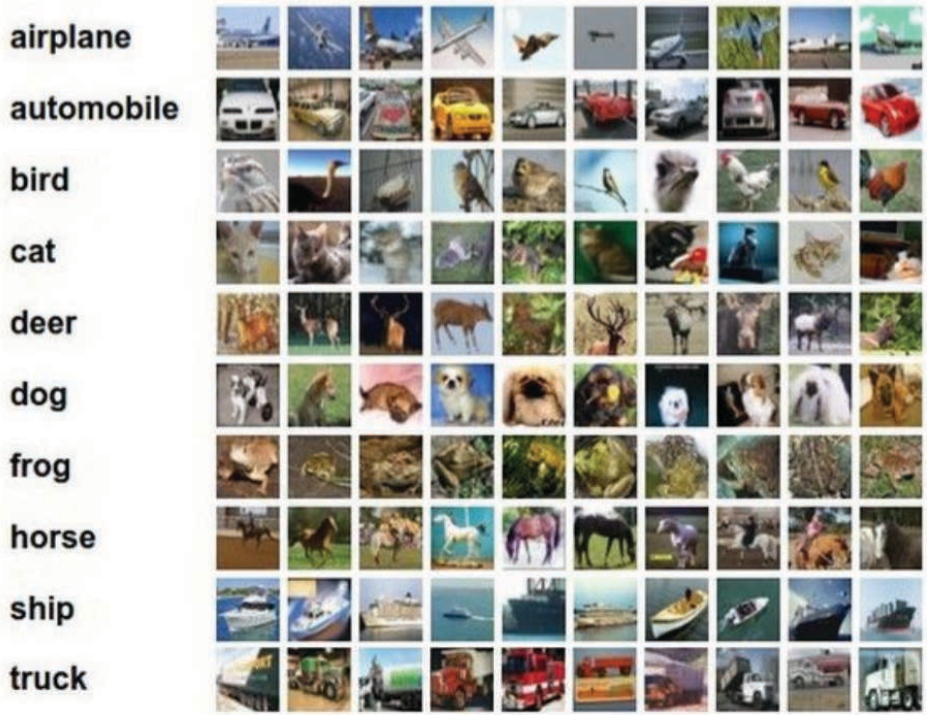
- The Classification Problem
- **Nearest Neighbor Classifier**
- Linear Classifier
- Loss Function
- Softmax Classifier
- Neural Network Classifier
- Brain Analogy
- Conclusion

Nearest Neighbor Classifier

Q: What it the nearest neighbor classifier?

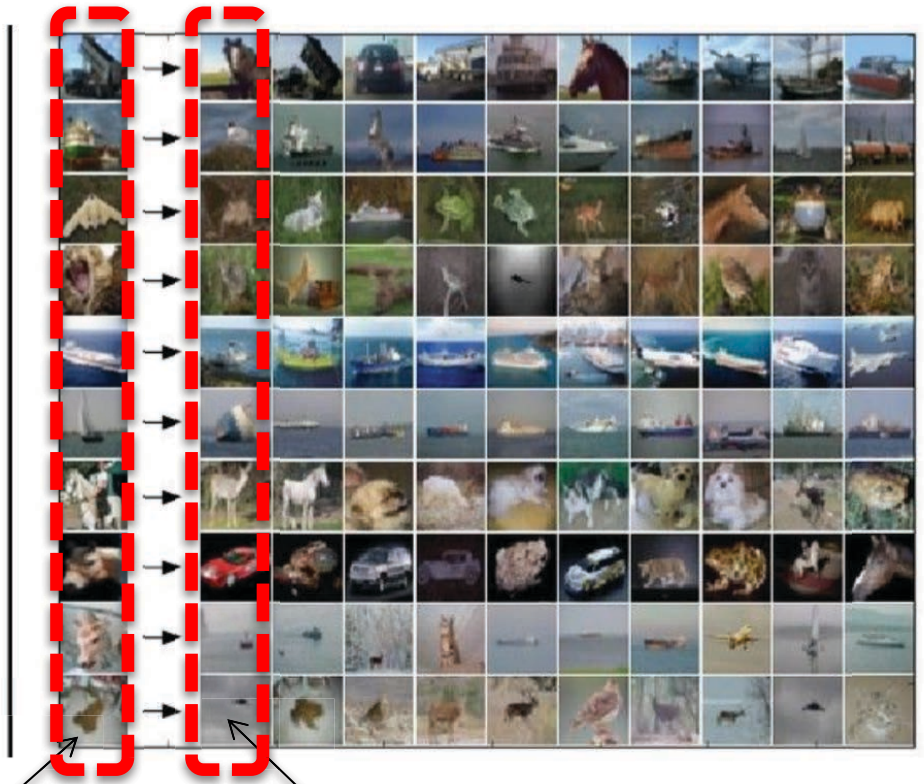
- Naive Classifier:

Example dataset: **CIFAR-10**
10 labels
50,000 training images
10,000 test images.



Training set

For every test image (first column), examples of nearest neighbors in rows



Test data

Nearest data in training set

How to find Nearest Neighbor?

- **Distance metrics:** L2, L1, cosine, Kullback–Leibler, your favorite..

this example →

$$d_{\ell_1}(I_1, I_2) = \sum_{ij} |I_{1ij} - I_{2ij}|$$

- **Python code:**

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

Nearest Neighbor classifier

remember the training data

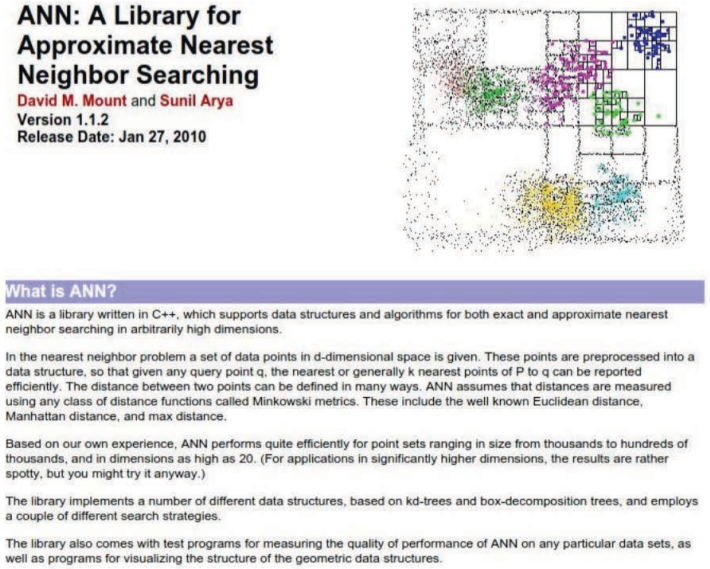
for every test image:
- find nearest train image with L1 distance
- predict the label of nearest training image

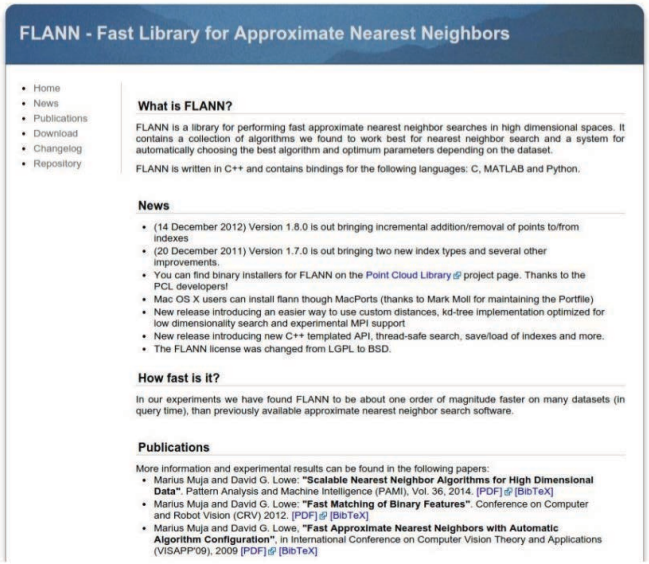
Test Time

- **Q: What is the test time?** And how the classification speed depends on the size n of training data? A: **O(n), linearly** ☹. This is a (major) limitation. *Fast test time is preferred in practice.*

  Note: Neural Networks have **fast** test time, but *expensive* training time.

- **Partial solution:** Use approximate nearest neighbor techniques, which finds approximate nearest neighbors quickly.



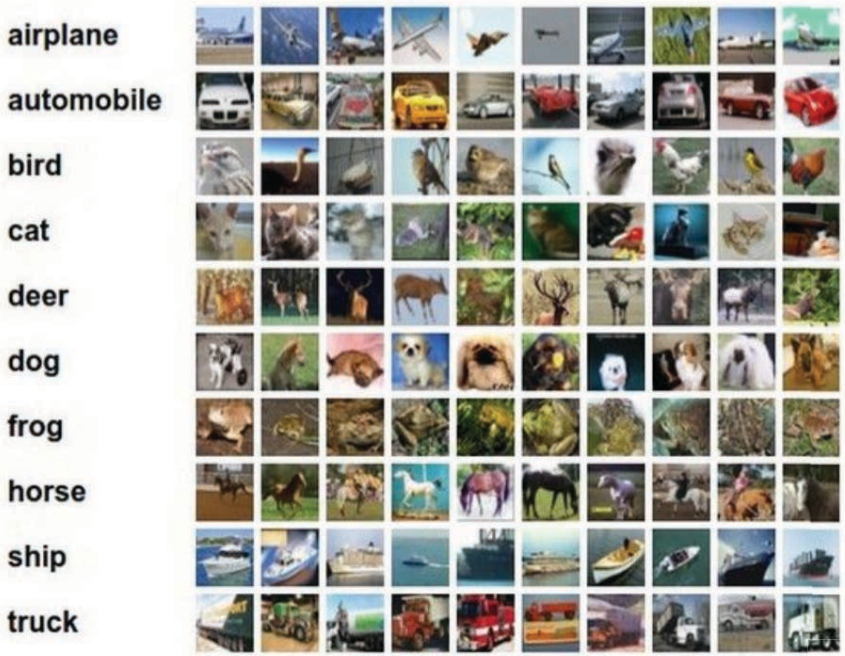


k-Nearest Neighbor Classifier

- **Limitation:** Nearest Neighbor is **sensitive** to outliers/noise.

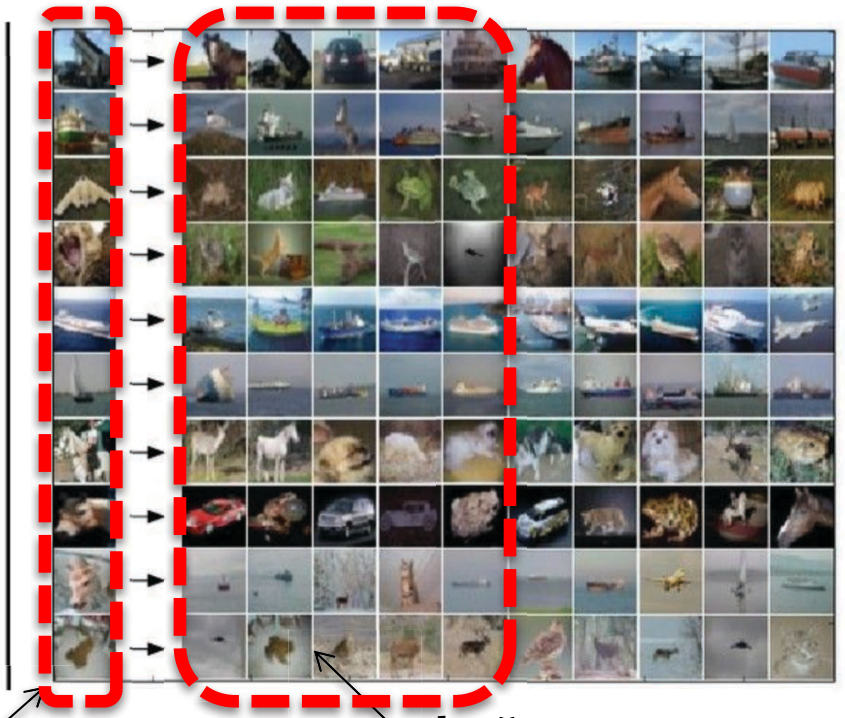
  Solution: Find the **k** nearest images, and pick the label with **majority voting** (regularization process).

Example dataset: **CIFAR-10**
10 labels
50,000 training images
10,000 test images.



Training set

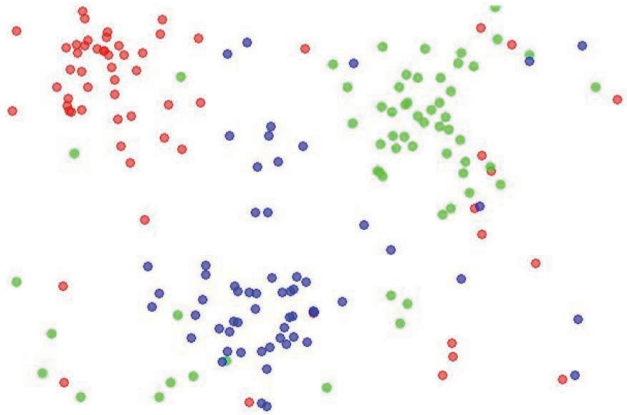
For every test image (first column), examples of nearest neighbors in rows



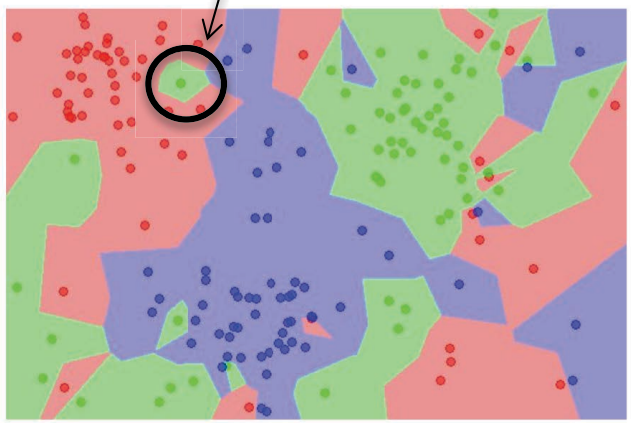
Test data

k=5
k-Nearest data in training set

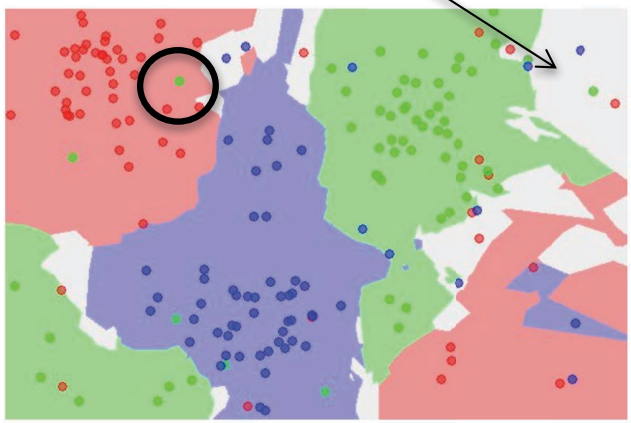
Illustration



Data



NN/1-NN classifier



5-NN classifier

Q: What it the accuracy of **1**-NN on **training** data? 100%

Q: What it the accuracy of **5**-NN on **training** data?

Q: What it the accuracy of **5**-NN on **test** data?

Hyperparameters

Q: What is the difference between parameter and hyperparameter?

- There exist **two types** of parameters:
  (1) **Parameters**: Variables that can be estimated by **optimization** ☺.
  (2) **Hyperparameters**: Variables that can be estimated by **cross-validation** (cannot be estimated by optimization) ☹.

- Examples of hyperparameters: *distance metric, k value.*

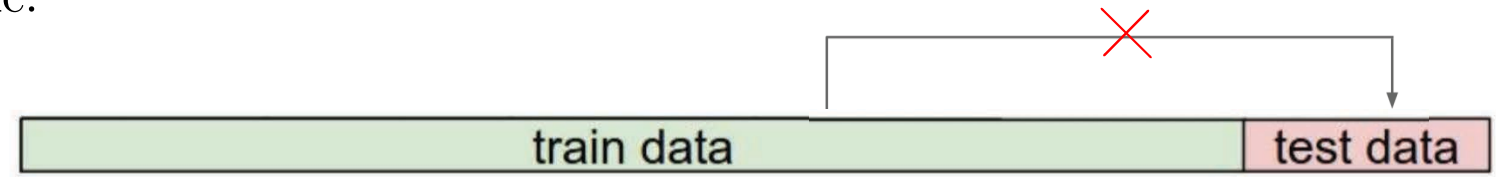
L2, L1, cosine, Kullback–Leibler? k=1,2,5,10,15?

Q: What is cross-validation?

- **Cross-validation:**

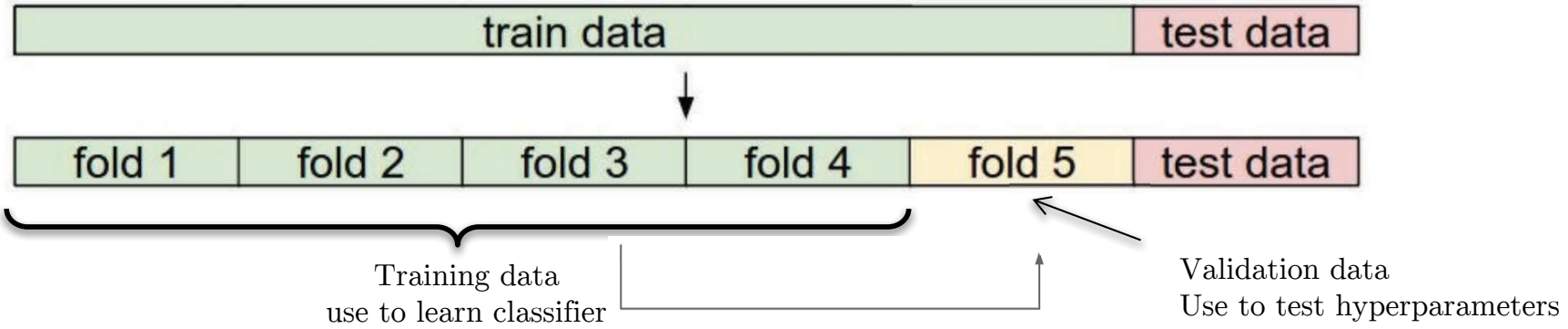
  Q: Try out what hyperparameters work best on test set? **Bad idea.**

  Test set used for the generalization performance! Use it only after training is done.

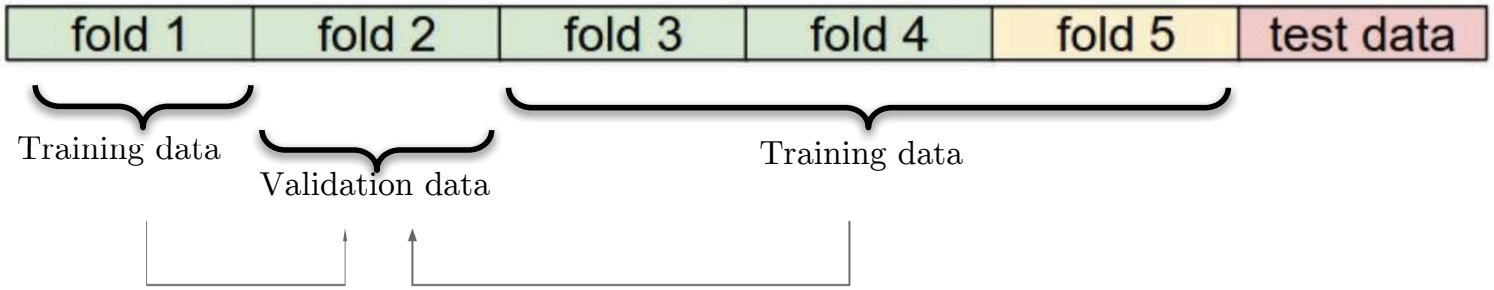


Cross-Validation

- **Split training data into training set and validation:**

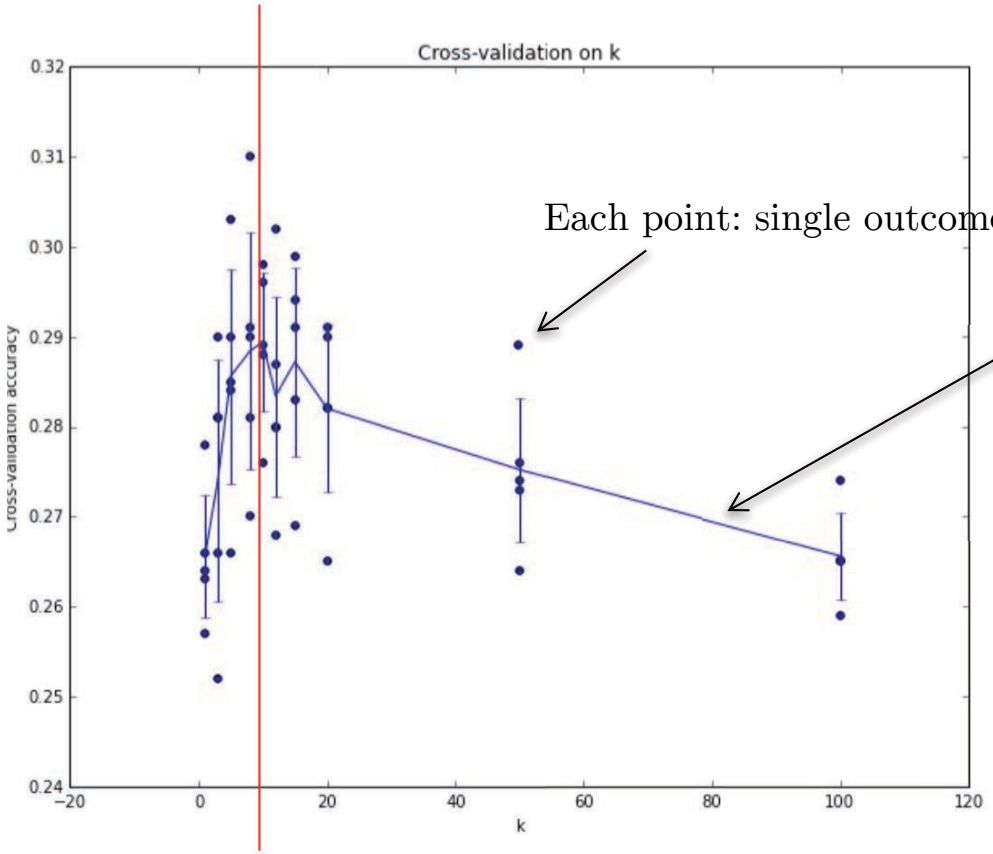


- **Cross-validate:** Cycle through the 5 folds, and record results:



Cross-Validation Result

- Example of **5-fold** cross-validation for finding the value of k:



$\Rightarrow$ Value $\mathbf{k = 7}$ works best for this data.

Demo: K-Nearest Neighbor

- Run **code01.ipynb**

```
# Visualize some examples from the dataset.
# We show a few examples of training images from each class.
classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse'
num_classes = len(classes)
samples_per_class = 7
for y, cls in enumerate(classes):
    idxs = np.flatnonzero(y_train == y)
    idxs = np.random.choice(idxs, samples_per_class, replace=False)
    for i, idx in enumerate(idxs):
        plt_idx = i * num_classes + y + 1
        plt.subplot(samples_per_class, num_classes, plt_idx)
        plt.imshow(X_train[idx].astype('uint8'))
        plt.axis('off')
        if i == 0:
            plt.title(cls)
plt.show()
```

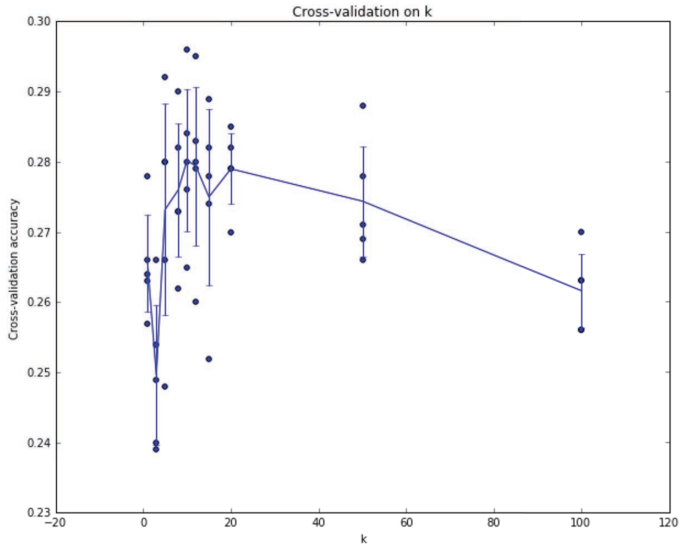
Cross-validation

We have implemented the k-Nearest Neighbor classifier but we set the value k = 5 arbitrarily. cross-validation.

```
# Computational time: 2-3 min

num_folds = 5
k_choices = [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]

X_train_folds = []
y_train_folds = []
```



k-Nearest Neighbor Performances

- Best accuracy (for k=7) is **29%** on validation sets, and **27%** on test set.

- **Conclusion:**
  (1) **Never use k-NN** (at least for image classification).
  (2) **Not robust to perturbations** (spatial variations, illumination changes, object deformation, occlusion, background clutter, intra-class variation)
  (3) **Bad test time**

Outline

- The Classification Problem
- Nearest Neighbor Classifier
- **Linear Classifier**
- Loss Function
- Softmax Classifier
- Neural Network Classifier
- Brain Analogy
- Conclusion

Linear Classifier

- **Image classification:**

Array 32x32x3

Image classification task →

Class of Images: **CAT**

- **Linear classifier:**

3D array 32x32x3

vectorize →

x
1D array
3072x1

input

Parameters/ Weights

Offset/ Bias

$$f(x, W, b) = Wx + b$$

10 numbers indicating class scores (highest is the choice)

f
1D array
10x1

Linear classifier/ Score function:

$$f = Wx + b$$

10x1 10x3072 3072x1 10x1

Interpreting Linear Classifier

Q: **What does a linear classifier do?**

A: Template matching technique: It scores the image (data) by matching it with 10 templates.

$$W \quad = \quad \begin{bmatrix} W_{1,\cdot} \\ \vdots \\ W_{10,\cdot} \end{bmatrix} \quad \longrightarrow \quad Wx \quad = \quad \begin{bmatrix} \langle W_{1,\cdot}, x \rangle \\ \vdots \\ \langle W_{10,\cdot}, x \rangle \end{bmatrix}$$

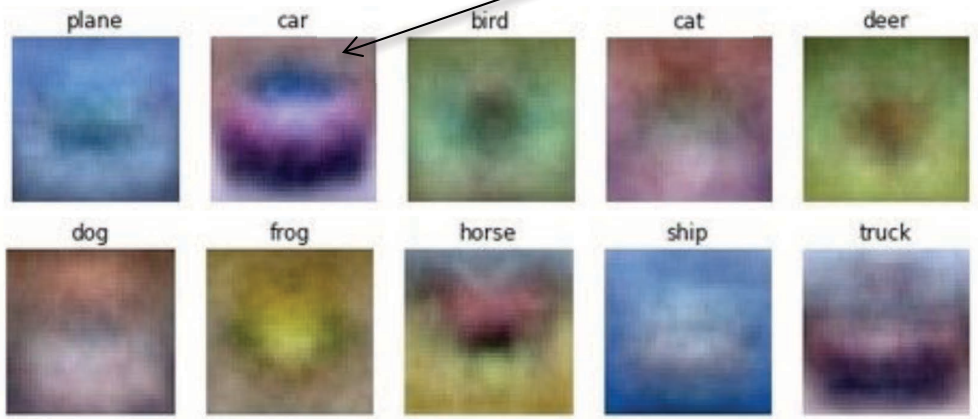
10x3072 — 10x1

Template/filters matching

Highest score decides the class

$W_{i,\cdot} =$  un-vectorize → 

1x3072 — 32x32x3



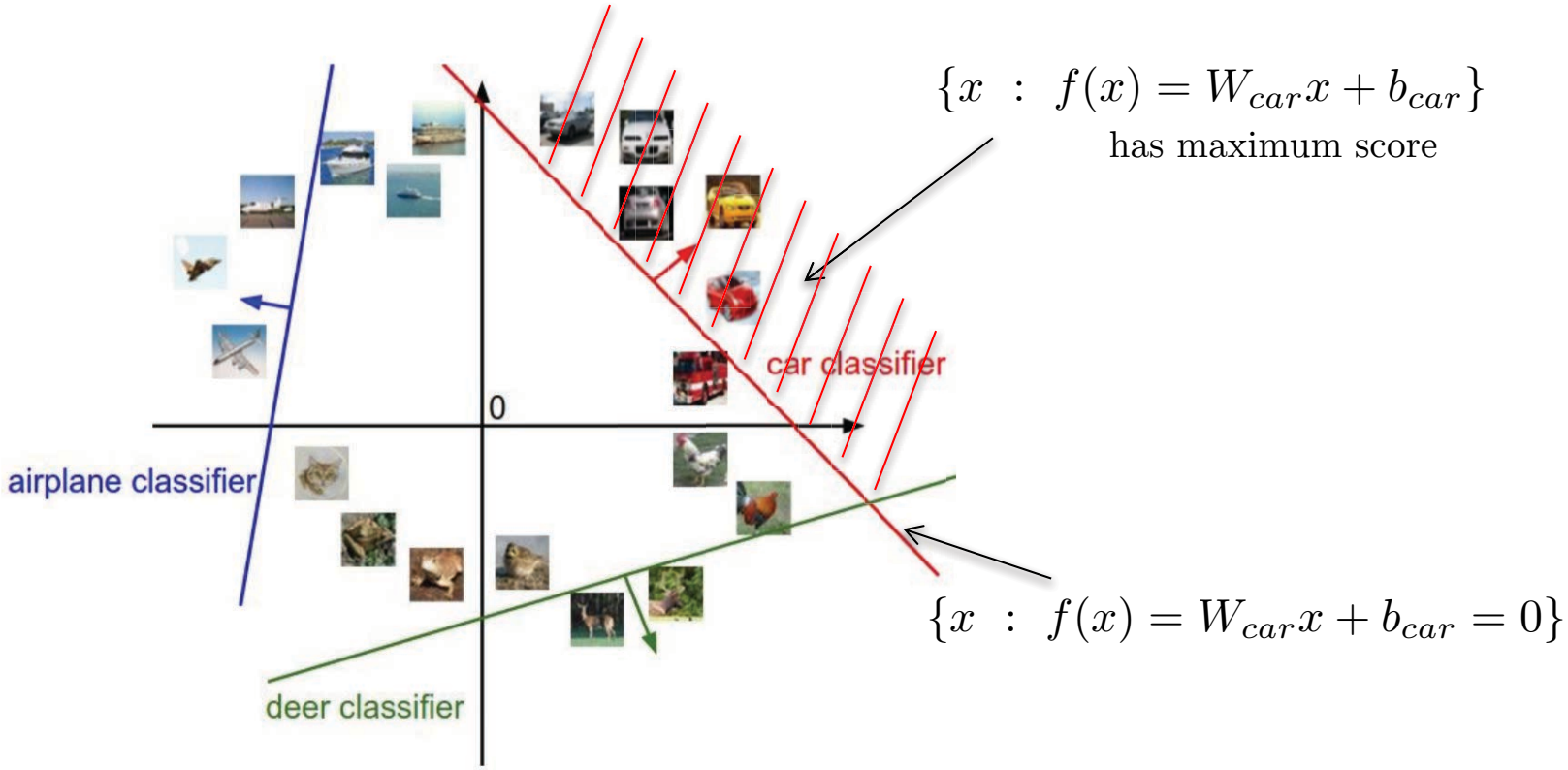
Trained weights W* (later discussed)

airplane	-3.45	-0.51	3.42
automobile	-8.87	**6.04**	4.64
bird	0.09	5.31	2.65
cat	**2.9**	-4.22	5.1
deer	4.48	-4.19	2.64
dog	8.02	3.58	5.55
frog	3.78	4.49	**-4.34**
horse	1.06	-4.37	-1.5
ship	-0.36	-2.09	-4.79
truck	-0.72	-2.93	6.14

Interpreting Linear Classifier

Q: **What does a linear classifier do?**

A: Linear mapping: It maps the high-dim image (data) R^{3072} to a low-dim linear space R^{10}.



Ouline

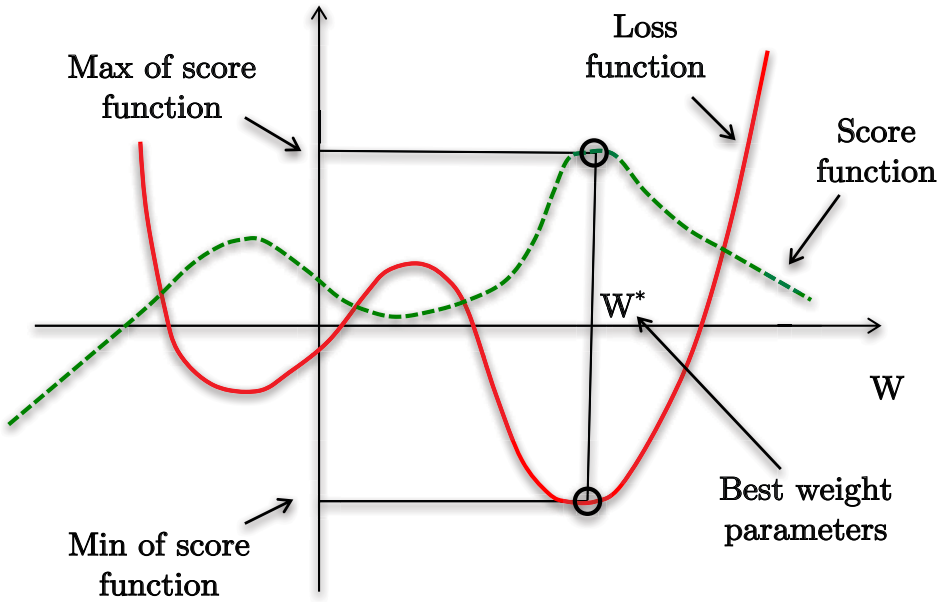
- The Classification Problem
- Nearest Neighbor Classifier
- Linear Classifier
- **Loss Function**
- Softmax Classifier
- Neural Network Classifier
- Brain Analogy
- Conclusion

How to Compute Weights?

- **(1) Define a loss function(/objective/energy):** *A loss function L quantifies how well the parameters (weights, offset) are chosen to get the highest possible score across all training data.*

  Exs of loss functions: SVM, L2, Logistic, Huber, etc.

- **(2) Optimization:** *The process of changing the parameters (weights, offset) to minimize the loss function in order to get the highest possible score across all training data.*



SVM Loss Function

- **Multiclass SVM loss:** Given (1) training data (x_i, y_i) and (2) score function $s_i = f(x_i, W)$,

the **multiclass SVM loss function** is:

$$L_i = \sum_{j \neq i} \max(0, s_j - s_i + 1)$$

comes form the margin

SVM loss measures how well the weights s are chosen to get the highest possible score. Then, L_i is **0** when s_i is **well classified**, that is when it has the highest score for its own class y_i, and L_i is **large** when s_i is **misclassified**.

Example when s_i is *well* classified:

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	

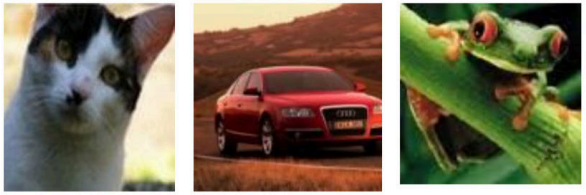
$L_i = \sum_{j \neq i} \max(0, s_j - s_i + 1)$

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

SVM Loss Function

Example when s_i is *mis*classified:

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

$L_i = \sum_{j \neq i} \max(0, s_j - s_i + 1)$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

- Total SVM loss: $L = \frac{1}{n} \sum_{i=1}^{n} L_i$

Q: What is the min value of L? A: 0.

Q: What is the max value of L? A: $+\infty$.

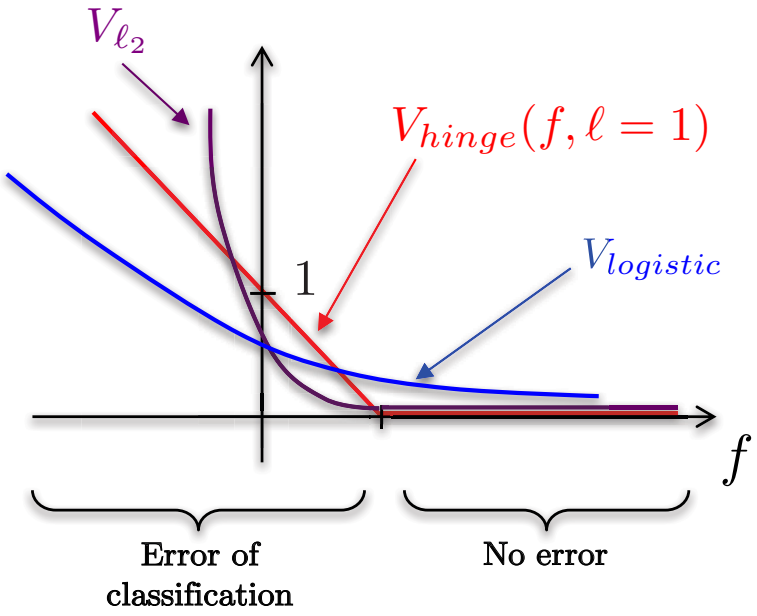
Loss Functions

- **Q: Will we get the same classification for this loss function?**

  A: Probably not.

$$L_i = \sum_{j \neq i} \max(0, s_j - s_i + 1)^2$$

- There are multiple available **loss functions:**
  - *(1) Hinge/SVM loss*
  - *(2) L2 loss*
  - *(3) Logistic regression loss (later discussed)*
  - *(4) Huber loss*



Non-Uniqueness of Solutions

- **Optimization problem:**

$$\min_W \frac{1}{n}\sum_{i=1}^{n} L_i(W) \quad (1)$$

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{j\neq i}\max(0, s_j - s_i + 1)$$

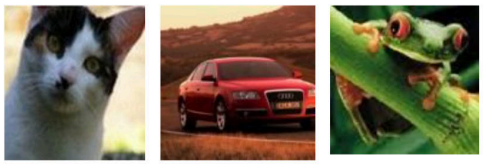
$$\frac{1}{n}\sum_{i=1}^{n}\sum_{j\neq i}\max(0, Wx_j - Wx_i + 1)$$

$\Rightarrow$ This opt problem is **ill-posed**.

Call W^* the solution of (1) then αW^*, $\alpha > 1$ is also solution of (1).

Example:

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	

$L_i = \sum_{j\neq i} \max(0, s_j - s_i + 1)$

Before:

= max(0, 1.3 - 4.9 + 1)
+max(0, 2.0 - 4.9 + 1)
= max(0, -2.6) + max(0, -1.9)
= 0 + 0
= 0

With W twice as large:

= max(0, 2.6 - 9.8 + 1)
+max(0, 4.0 - 9.8 + 1)
= max(0, -6.2) + max(0, -4.8)
= 0 + 0
= 0

- **Q: How to fix this issue?** Regularization.

Regularization

- New loss function: $\min_W \frac{1}{n}\sum_{i=1}^{n} L_i(W) + \lambda \|W\|_F^2$

Equivalent to maximize margins between training data

- *Other math interpretation:* Strongly **convex** term $\Rightarrow$ make the solution **unique!**

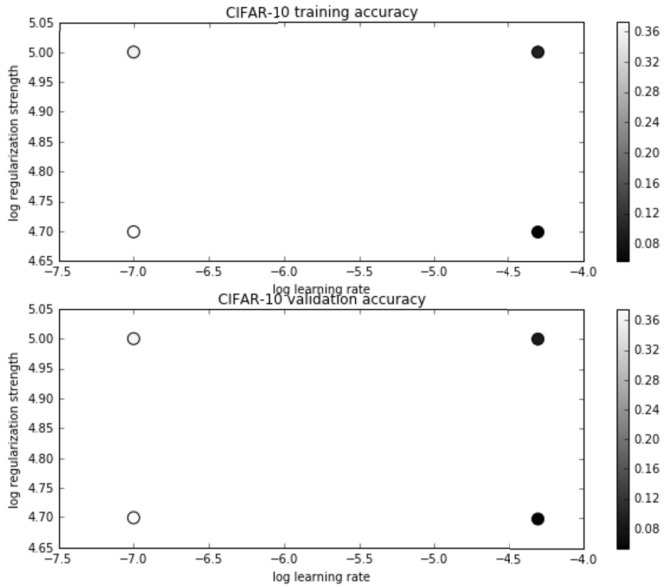
- **Regularization terms:**
  - *(1) L2 regularization:* smooth and differentiable.
  - *(2) L1 regularization:* non-smooth and non-differentiable, but promotes *sparsity (a few non-zero elements).*
  - *(3) Elastic net regularization:* mixture of L1 and L2.
  - *(4) Dropout for Neural Nets* (later discussed).

Demo: Linear Classifier and SVM Loss

- Run **code02.ipynb**

```
# Write the LinearSVM.predict function and evaluate the performance on both the
# training and validation set
y_train_pred = svm.predict(X_train)
print 'training accuracy: %f' % (np.mean(y_train == y_train_pred), )
y_val_pred = svm.predict(X_val)
print 'validation accuracy: %f' % (np.mean(y_val == y_val_pred), )
```

```
training accuracy: 0.370061
validation accuracy: 0.379000
```




```
# Visualize the cross-validation results
import math
x_scatter = [math.log10(x[0]) for x in results]
y_scatter = [math.log10(x[1]) for x in results]
```

```
# Evaluate the best svm on test set
y_test_pred = best_svm.predict(X_test)
test_accuracy = np.mean(y_test == y_test_pred)
print 'linear SVM on raw pixels final test set accuracy: %f' % test_accuracy
```

```
linear SVM on raw pixels final test set accuracy: 0.375000
```

Limitations of Linear Classifier

- **Conclusion:**

  (1) **Never use linear classifier directly on raw data** (at least for image classification).

  (2) **Not robust to perturbations** (spatial variations, illumination changes, object deformation, occlusion, background clutter, intra-class variation)

  (3) **Excellent test time**

Outline

- The Classification Problem
- Nearest Neighbor Classifier
- Linear Classifier
- Loss Function
- **Softmax Classifier**
- Neural Network Classifier
- Brain Analogy
- Conclusion

Softmax Classifier

- *Softmax classifier* = **Multinomial logistic regression**

- **Motivation (from statistics):** Maximize the log likelihood of the **score probabilities** of the classes:

$$s_i = f(x_i, W)$$

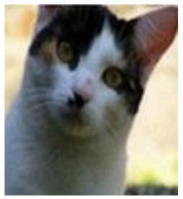
Scores = unnormalized log probabilities of the classes

$$P(Y = y_i | X = x_i) = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

Softmax function

$$L_i = -\log P(Y = y_i | X = x_i)$$

$$L_i = -\log \left(\frac{e^{s_i}}{\sum_j e^{s_j}} \right)$$



unnormalized probabilities

	unnormalized log probabilities	exp		normalize	probabilities	
cat	3.2	→	24.5	→	0.13	→ L_i = -log(0.13) = 0.89
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	

Demo: Softmax Classifier

- Run **code03.ipynb**

```
# Print out results.
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print 'lr %e reg %e train accuracy: %f val accuracy: %f' % (
                lr, reg, train_accuracy, val_accuracy)

print 'best validation accuracy achieved during cross-validation: %f' % best_val
```

```
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.333265 val accuracy: 0.344000
lr 1.000000e-07 reg 1.000000e+08 train accuracy: 0.100265 val accuracy: 0.087000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.327714 val accuracy: 0.334000
lr 5.000000e-07 reg 1.000000e+08 train accuracy: 0.100265 val accuracy: 0.087000
best validation accuracy achieved during cross-validation: 0.344000
```

```
# evaluate on test set
# Evaluate the best softmax on test set
y_test_pred = best_softmax.predict(X_test)
test_accuracy = np.mean(y_test == y_test_pred)
print 'softmax on raw pixels final test set accuracy: %f' % (test_accuracy, )
```

```
softmax on raw pixels final test set accuracy: 0.342000
```

Outline

- The Classification Problem
- Nearest Neighbor Classifier
- Linear Classifier
- Loss Function
- Softmax Classifier
- **Neural Network Classifier**
- Brain Analogy
- Conclusion

Neural Network Classifier

- **Image classification:**

Array 32x32x3

Image classification task →

Class of Images: **CAT**

- **Linear classifier:**

3D array
32x32x3

vectorize →

x
1D array
3072x1

W →

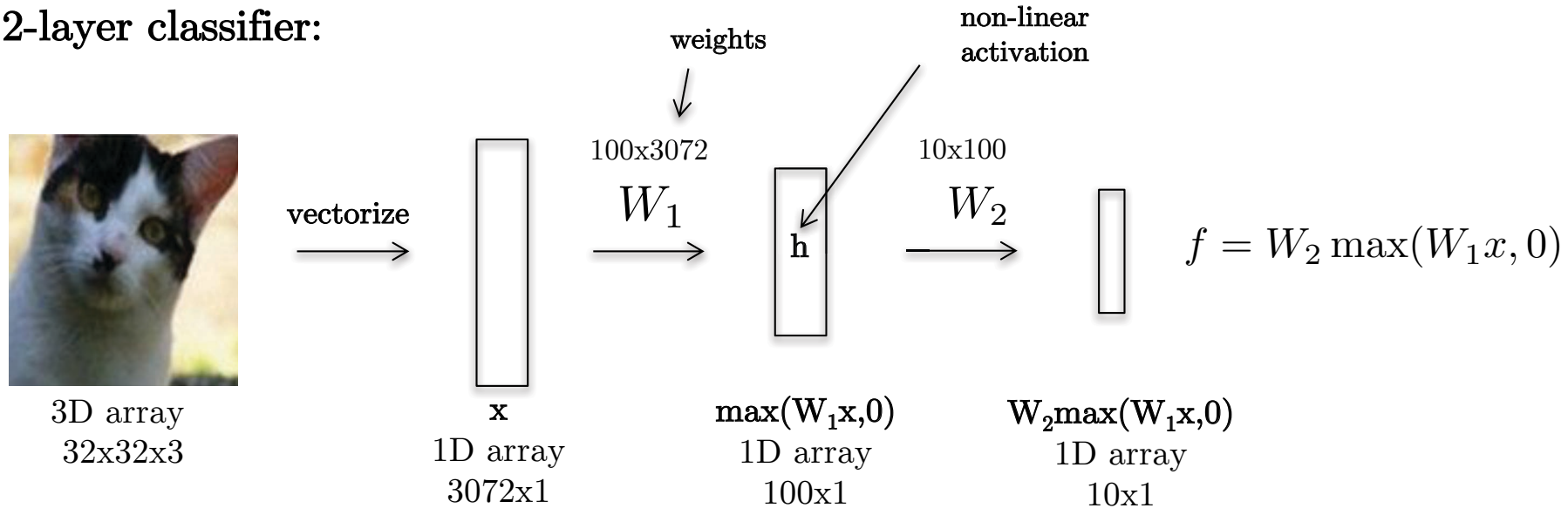
s
1D array
10x1

$$f = Wx$$

10x1 10x3072 3072x1

Neural Network Classifier

- **2-layer classifier:**



- **3-layer classifier:** $f = W_3 \max(W_2 \max(W_1 x, 0), 0)$

- **Conclusion:** *Neural Networks (NN) are simply series of linear classification and non-linear activations (max).*

Code for 2-Layer NN Classifier

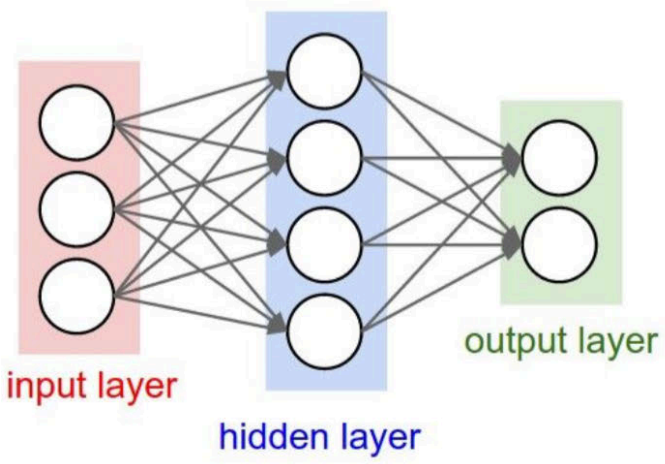
- **Full implementation** of training a 2-layer Neural Network needs **11 lines**:

```
01.  X = np.array([ [0,0,1],[0,1,1],[1,0,1],[1,1,1] ])
02.  y = np.array([[0,1,1,0]]).T
03.  syn0 = 2*np.random.random((3,4)) - 1
04.  syn1 = 2*np.random.random((4,1)) - 1
05.  for j in xrange(60000):
06.      l1 = 1/(1+np.exp(-(np.dot(X,syn0))))
07.      l2 = 1/(1+np.exp(-(np.dot(l1,syn1))))
08.      l2_delta = (y - l2)*(l2*(1-l2))
09.      l1_delta = l2_delta.dot(syn1.T) * (l1 * (1-l1))
10.      syn1 += l1.T.dot(l2_delta)
11.      syn0 += X.T.dot(l1_delta)
```

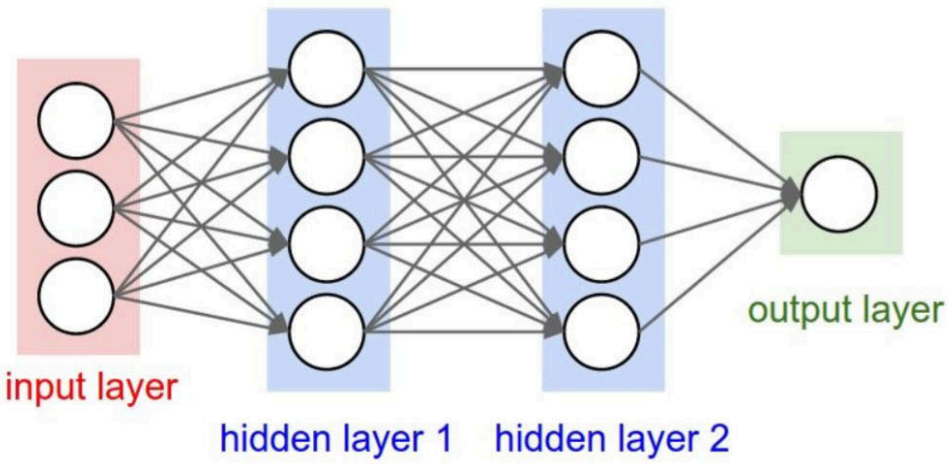
http://iamtrask.github.io/2015/07/12/basic-python-network

Neural Network Architecture

- **Fully connected (FC) layers:** Each neuron is connected to all neurons in the next layer.



2-layer Neural Net
or 1-hidden-layer Neural Net



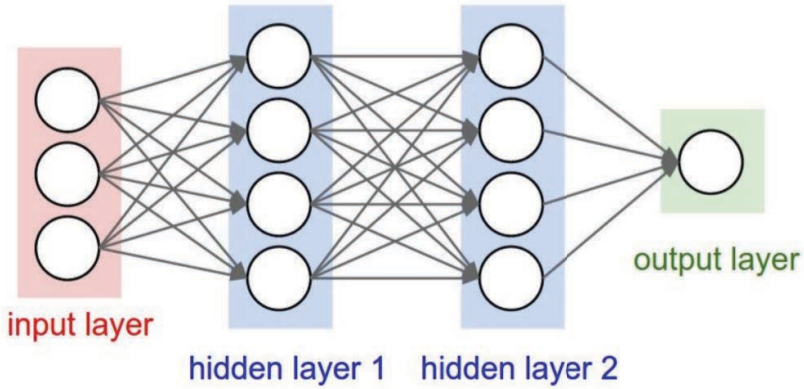
3-layer Neural Net,
or 2-hidden-layer Neural Net

- **The need for more structure:** FC networks are very *generic* but also *highly computationally expensive* to learn (huge number of parameters). They cannot be deep.

  However, using **special structures of data** (like *local stationarity* in convolutional neural networks, and *recurrence* in recurrent neural networks) allow to construct **deep networks** that can be learned (later discussed).

Test Time

- Once training is done, it is **fast to classify new data** (simple linear algebra operations):




```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

- This operation is called **forward pass** (later discussed).

Demo: Neural Network Classifier

- Run **code04.ipynb**

Cross-Validation

```
# Computational time: 10 counts ~ 2 min
max_count = 100 #
#max_count = 10 # EARLY STOPPING !

best_net = None # store the best model into this
```

```
2016-09-04 10:37:23.382828 val_acc: 0.402       lr: 0.000238896021592   reg: 5.89861458769e-05  0/100
2016-09-04 10:37:29.397552 val_acc: 0.467       lr: 0.000322293163585   reg: 0.00272677936957   1/100
2016-09-04 10:37:36.164308 val_acc: 0.489       lr: 0.000617194237683   reg: 6.02707161361e-05  2/100
2016-09-04 10:37:43.081210 val_acc: 0.506       lr: 0.00019194669295    reg: 0.0491183062678    3/100
2016-09-04 10:37:50.047093 val_acc: 0.499       lr: 0.000323878163807   reg: 0.0181277939526    4/100
2016-09-04 10:37:57.322166 val_acc: 0.514       lr: 0.000738230215214   reg: 0.00189137238623   5/100
2016-09-04 10:38:04.559132 val_acc: 0.516       lr: 0.000792758850874   reg: 0.413112908571     6/100
2016-09-04 10:38:11.623970 val_acc: 0.54        lr: 0.000121476846565   reg: 1.22079423363      7/100
2016-09-04 10:38:18.526379 val_acc: 0.55        lr: 0.000140736089124   reg: 0.000406162567027  8/100
2016-09-04 10:38:25.340778 val_acc: 0.516       lr: 0.000827607042798   reg: 0.000392744255504  9/100
```

Run on the test set

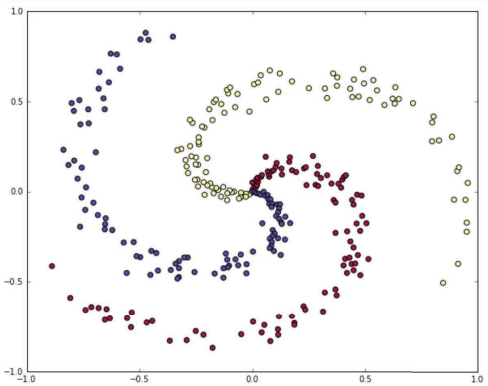
When you are done experimenting, you should evaluate your final trained network on the test set;

```
test_acc = (best_net.predict(X_test) == y_test).mean()
print 'Test accuracy: ', test_acc
```

```
Test accuracy:  0.536
```

Demo: Linear vs. Neural Network Classifiers

- Run **code05.ipynb**

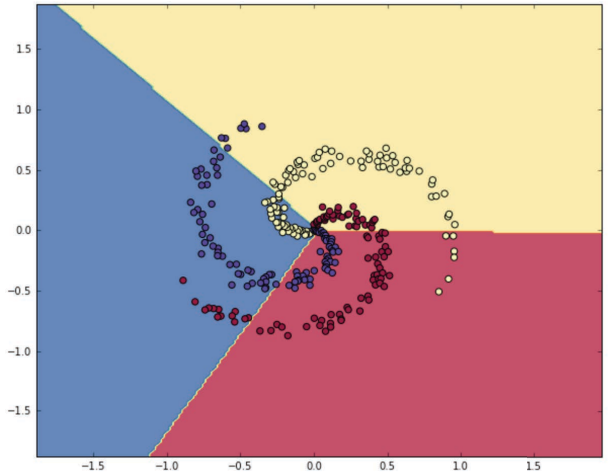


Linear Classifier

```
# Train a Linear Classifier
```

```
# evaluate training set accuracy
scores = np.dot(X, W) + b
predicted_class = np.argmax(scores, axis=1)
print 'training accuracy: %.2f' % (np.mean(predicted_class == y))
```

```
training accuracy: 0.49
```

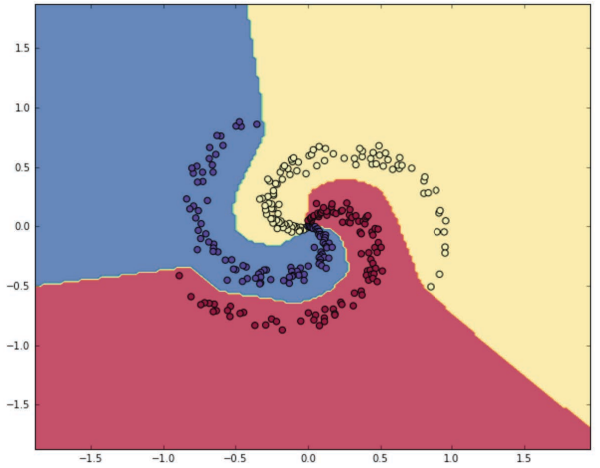


Neural Network Classifier

```
# Train a Neural Network Classifier
```

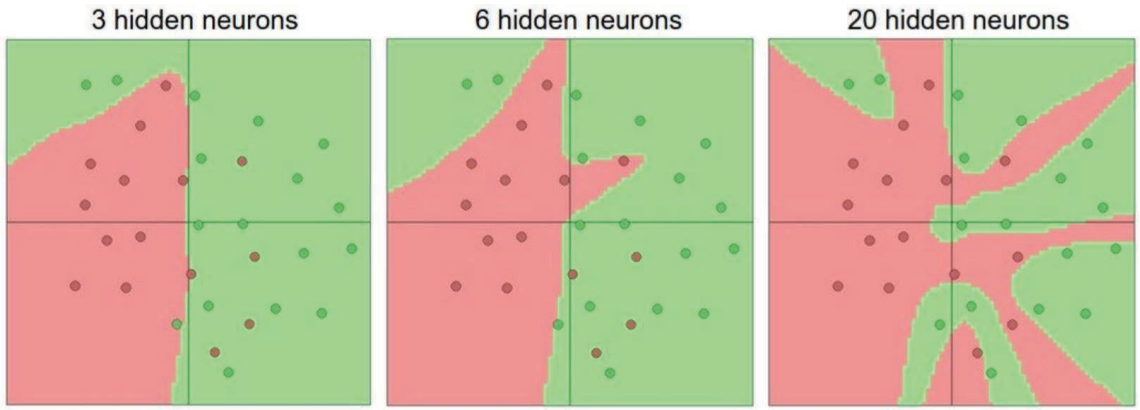
```
# evaluate training set accuracy
hidden_layer = np.maximum(0, np.dot(X, W) + b)
scores = np.dot(hidden_layer, W2) + b2
predicted_class = np.argmax(scores, axis=1)
print 'training accuracy: %.2f' % (np.mean(predicted_class == y))
```

```
training accuracy: 0.98
```

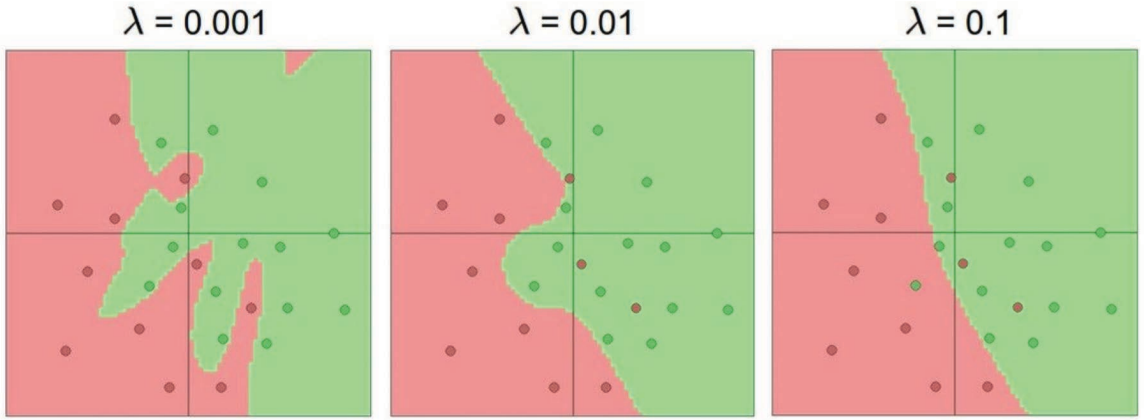


Online Demo

- ConvNetJS: http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html



more neurons = more capacity

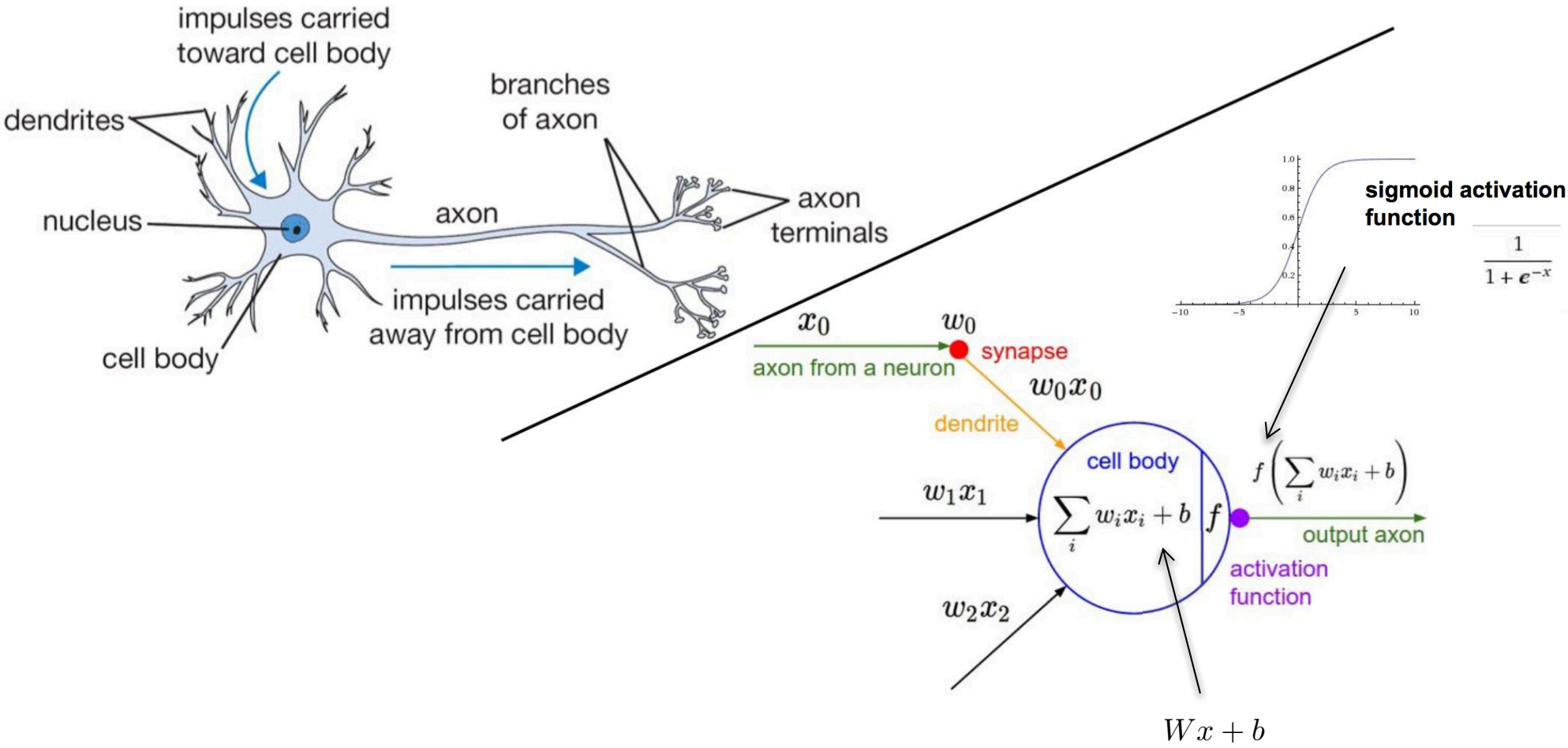


Regularization handles outliers

Outline

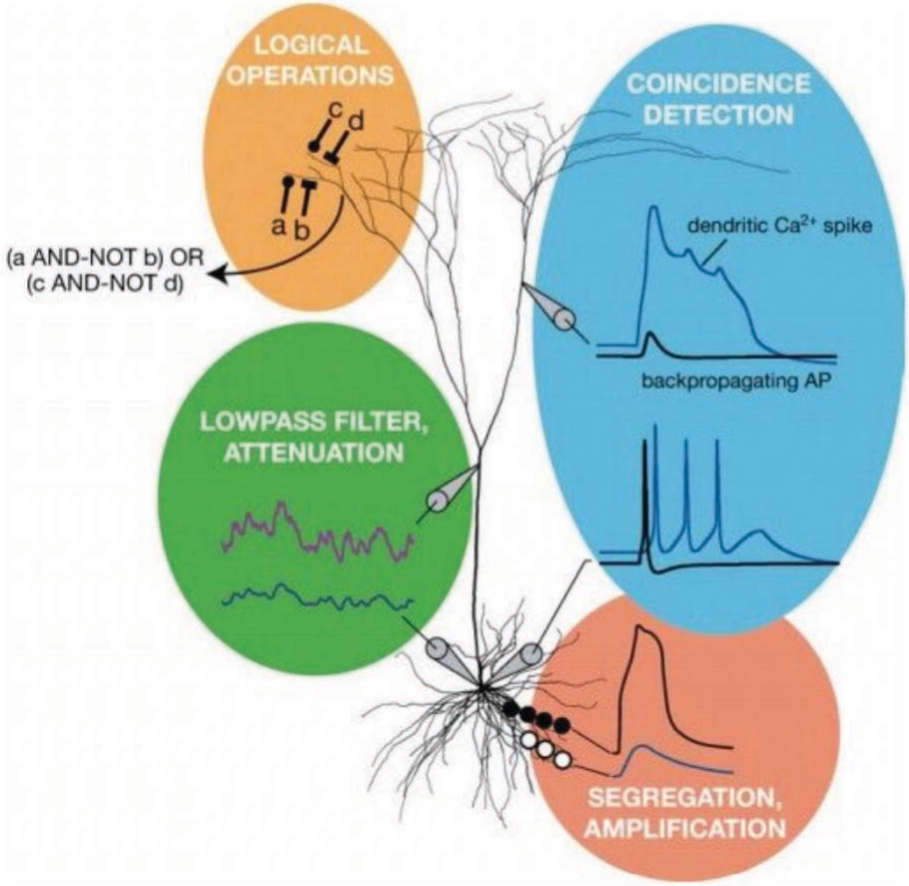
- The Classification Problem
- Nearest Neighbor Classifier
- Linear Classifier
- Loss Function
- Softmax Classifier
- Neural Network Classifier
- **Brain Analogy**
- Conclusion

Brain Analogy



Very Limited Analogy

- **Biological Neurons:**
  - Many different types
  - Dendrites can perform complex non-linear computations
  - Synapses are not a single weight but a complex non-linear dynamical system



[Dendritic Computation. London and Hausser]

Outline

- The Classification Problem
- Nearest Neighbor Classifier
- Linear Classifier
- Loss Function
- Softmax Classifier
- Neural Network Classifier
- Brain Analogy
- **Conclusion**

Summary

- **Image/data classification:** Given a training set, design a classifier and predict labels for test set.

- **k-Nearest Neighbor classifier:**

  Predict labels from k nearest images in the training set.

  (Almost) never used as bad accuracy, and bad test time.

- **Linear/softmax classifier:**

  Predict labels with a linear function.

  Has been used for a long time (kernel techniques) but overcome by deep learning.

  Score function: $f = Wx + b$

  SVM loss function: $L_i = \sum_{j \neq i} \max(0, s_j - s_i + 1)$

  Softmax loss function: $L_i = -\log\left(\frac{e^{s_i}}{\sum_j e^{s_j}}\right)$

Summary

- **FC Neural Networks (NNs):**

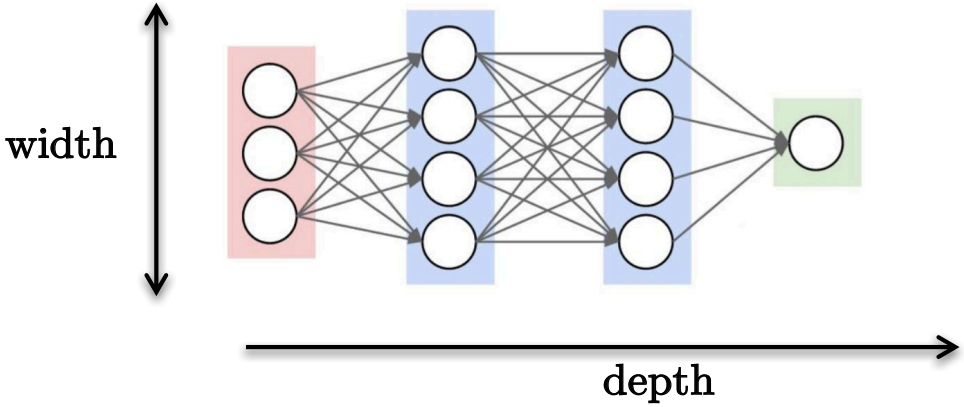
  Neurons arranged as fully connected layers.

  Series of linear functions and non-linear activations.

  Fast test time (matrix multiplications)

  Performances: bigger = better, but expensive training time (thanks GPUs)

  Bigger = (layer) width and depth (deep)



Q: How to train Neural Networks?

Questions?