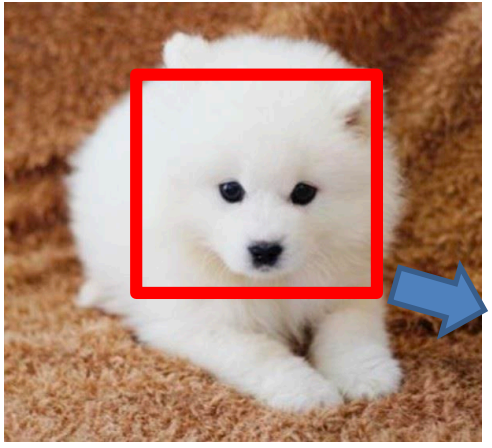
Lecture 4: Fundamentals of Deep Learning
(Linear Classification, Backpropagation)

Classification Example

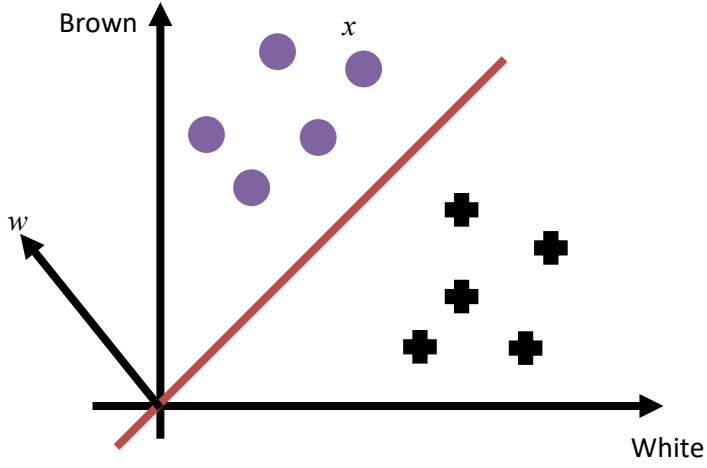
Distinguish between Red Panda and Dog

Brown 70%, white 50%



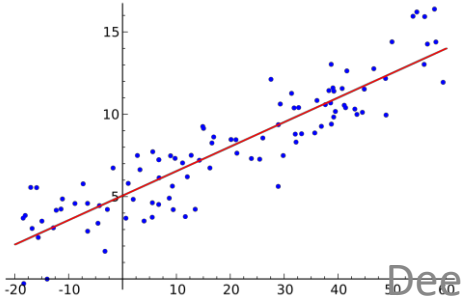
White 90%, Brown 5%

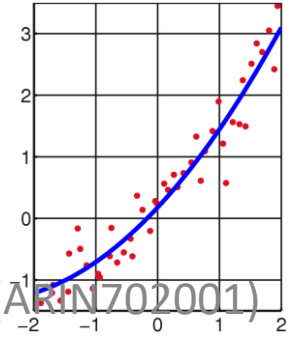
$w^T x > 0$ => Red Panda
$w^T x < 0$ => Dog



Least squares problem

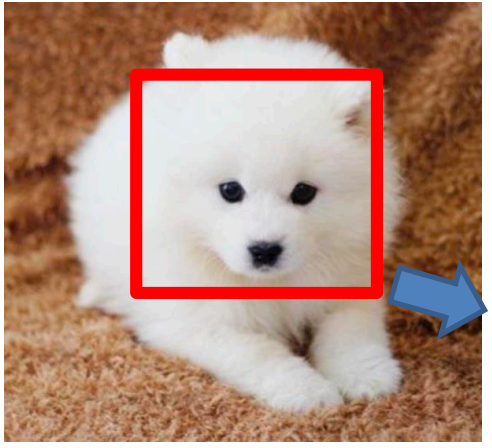
- $\hat{x}$ called *least squares approximate solution* of $Ax = b$
- $\hat{x}$ is sometimes called 'solution of $Ax = b$ in the least squares sense'
  - this is very confusing
  - never say this
  - do not associate with people who say this
- $\hat{x}$ need not (and usually does not) satisfy $A\hat{x} = b$
- but if $\hat{x}$ does satisfy $A\hat{x} = b$, then it solves least squares problem





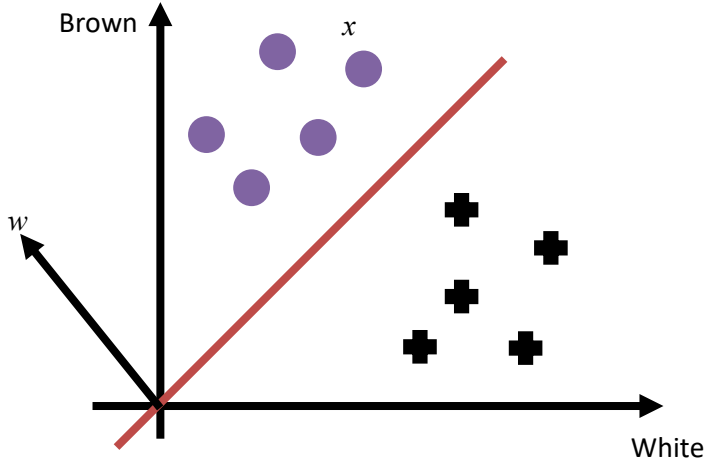
Distinguish between Red Panda and Dog

Brown 70%, white 50%

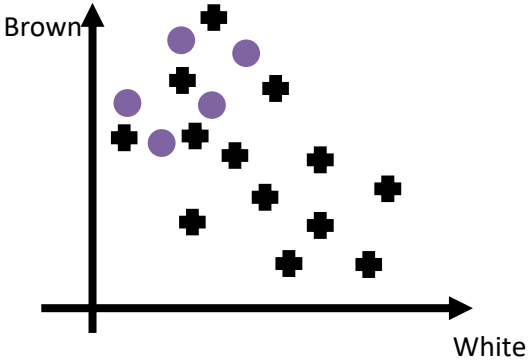


White 90%, Brown 5%

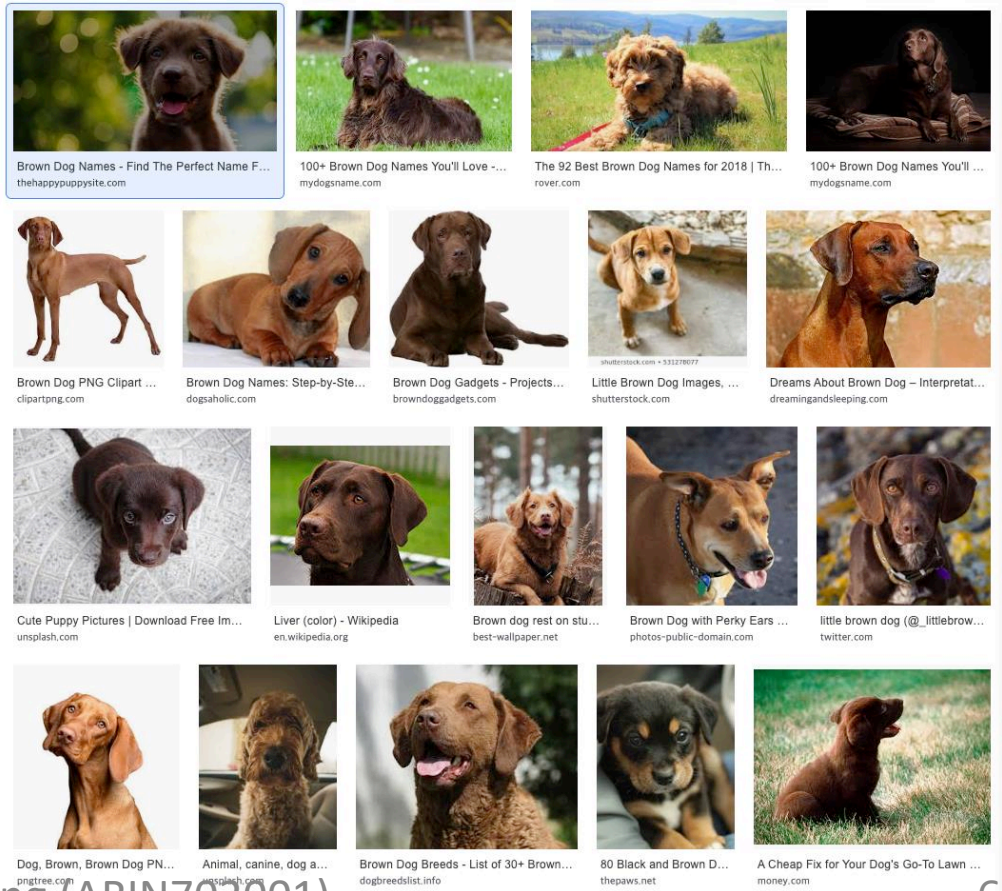
$w^T x > 0$ => Red Panda
$w^T x < 0$ => Dog



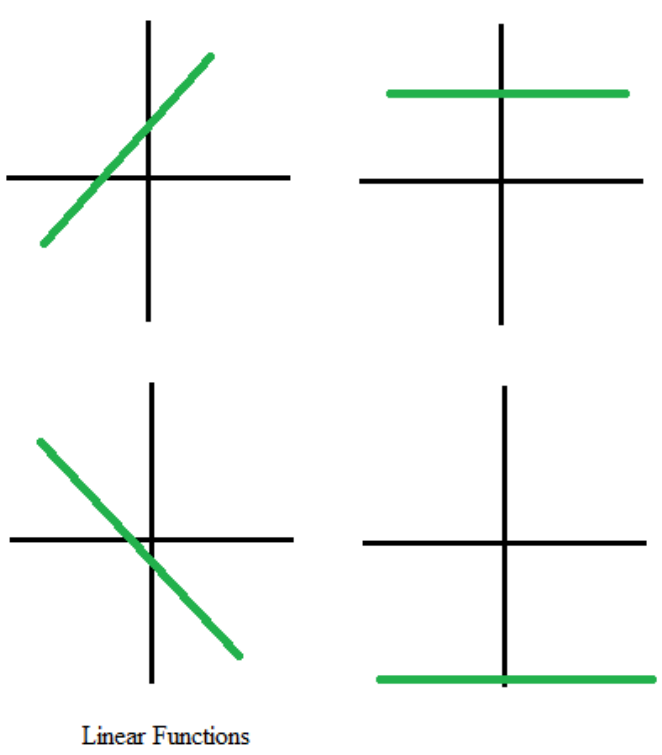
Hard Problem in the wild



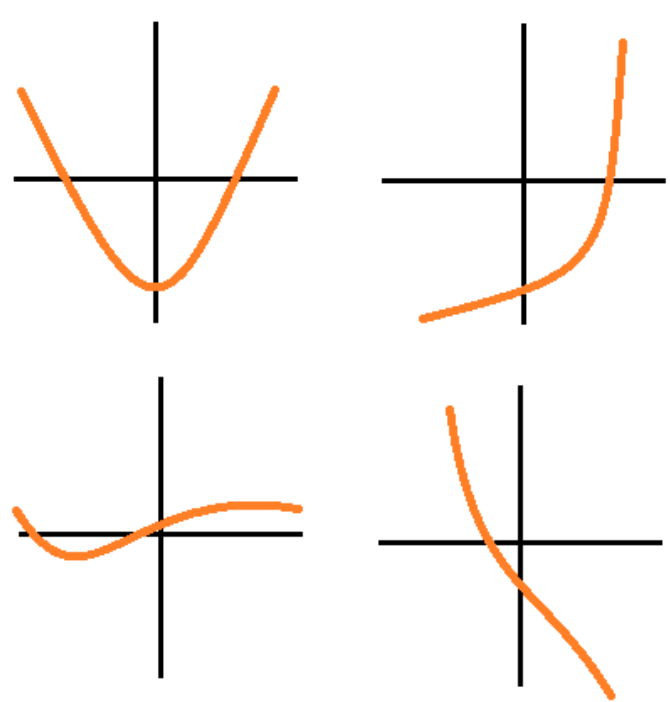
VS



Linear vs Non-linear Function



Linear Functions



Nonlinear Functions

Deep Learning

Most slides are from CS231n of Stanford University.

What is Deep Learning?

Compositional Models

Learned End-to-End

Hierarchy of Representations

- vision: pixel, motif, part, object
- text: character, word, clause, sentence
- speech: audio, band, phone, word

concrete → learning → abstract

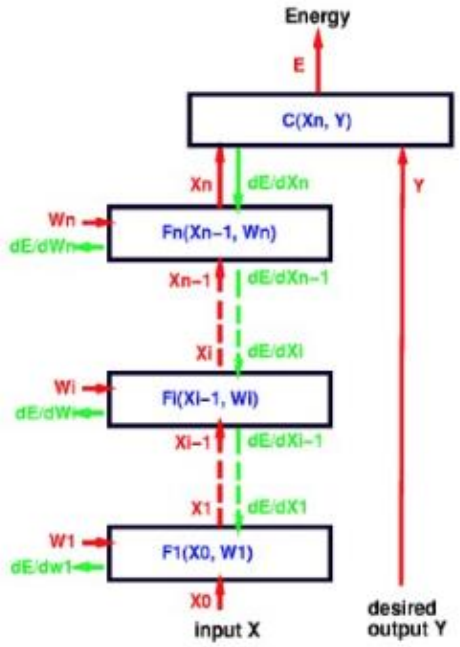


figure credit Yann LeCun, ICML '13 tutorial

What is Deep Learning?

Compositional Models
Learned End-to-End

Back-propagation jointly learns all of the model parameters to optimize the output for the task.

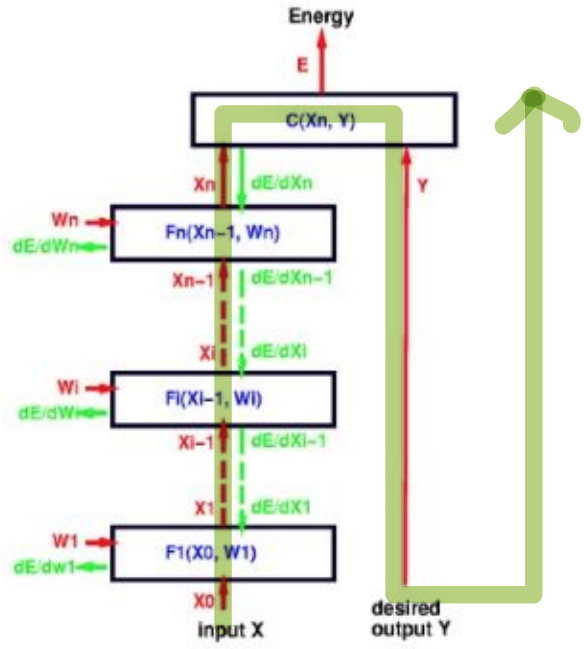
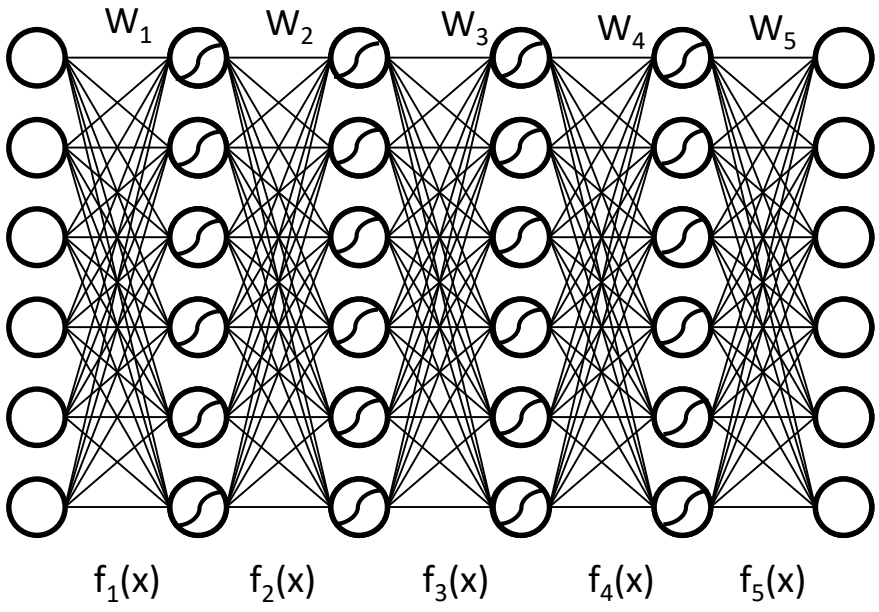


figure credit Yann LeCun, ICML '13 tutorial

Deep Neural Network

Too many learnable parameters
=> Requiring many data, Expensive computational cost.



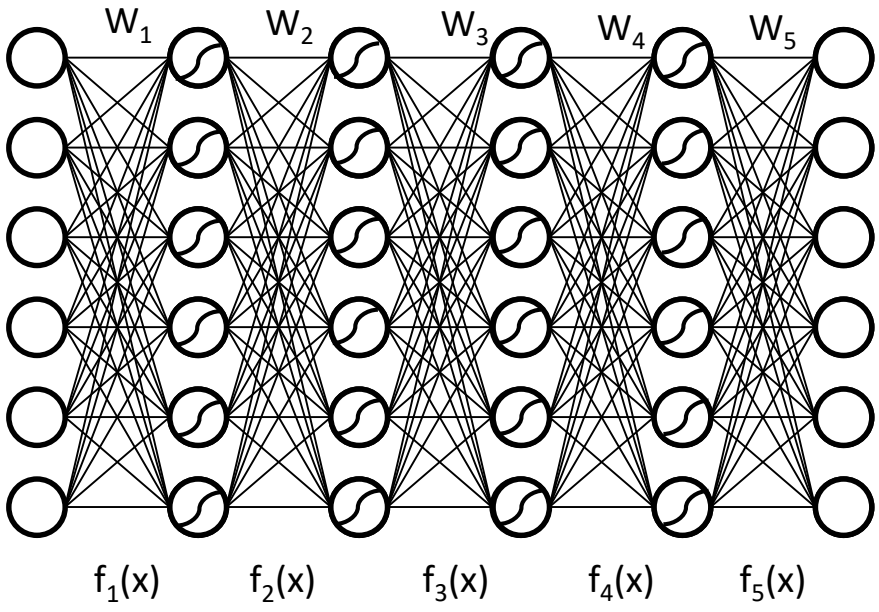
- Composition of linear functions?

$f_5(f_4(f_3(f_2(f_1(x)))))$

=>Non-linear functions are required.

Deep Neural Network

Too many learnable parameters
=> Requiring many data, Expensive computational cost.



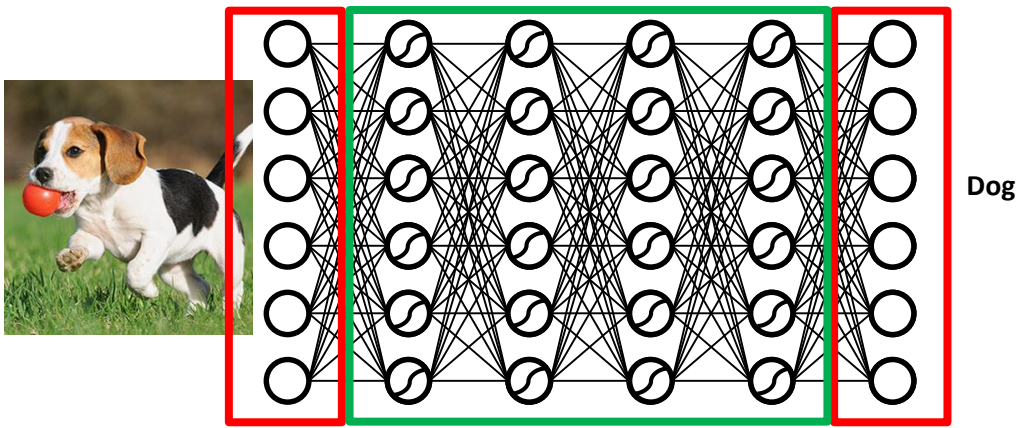
- Composition of linear functions?

$f_5(f_4(f_3(f_2(f_1(x)))))$

=>Non-linear functions are required.

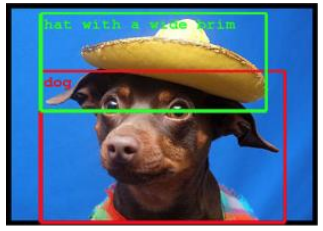
Image Classification

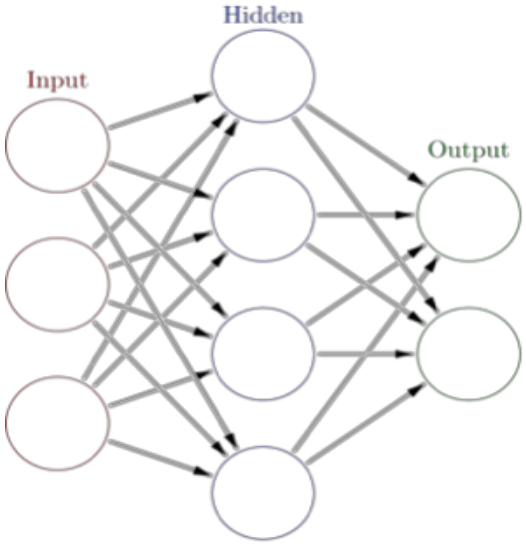
Assume given set of discrete labels (dog, cat, truck, plane…)

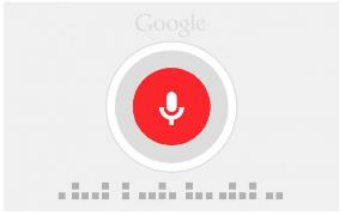


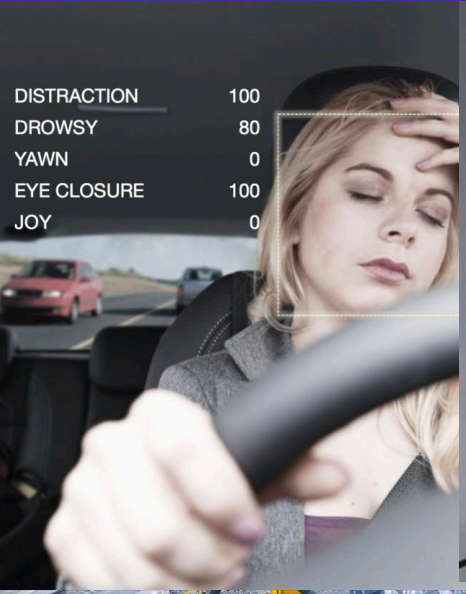
Why Deep Learning?

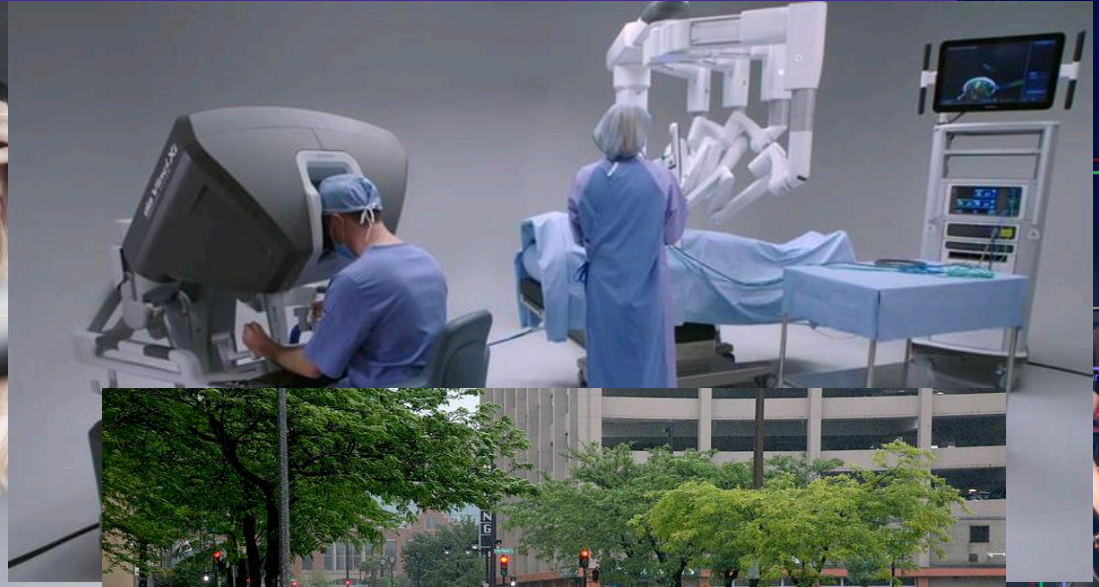
End-to-End Learning for Many Tasks

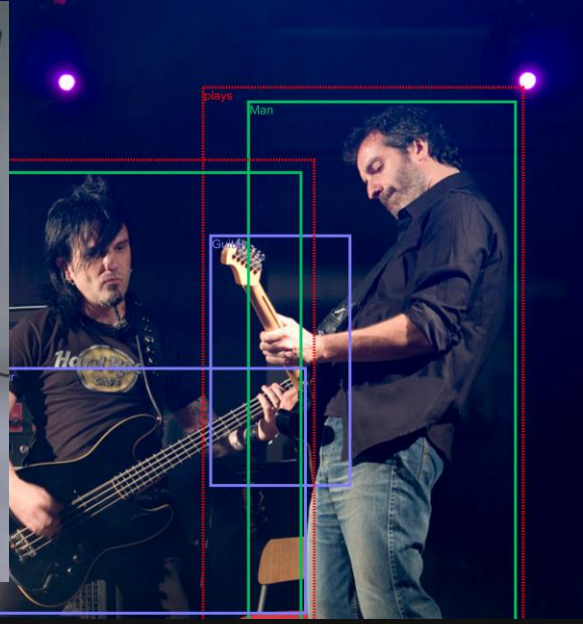


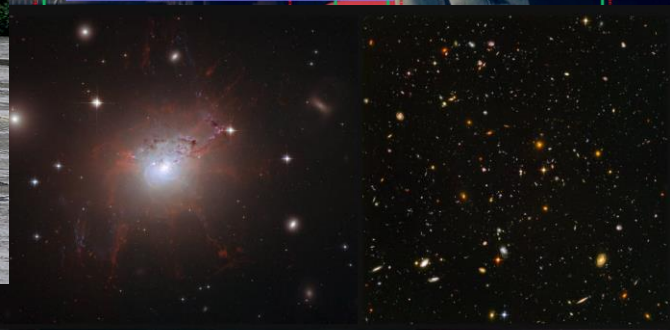





DISTRACTION 100
DROWSY 80
YAWN 0
EYE CLOSURE 100
JOY 0

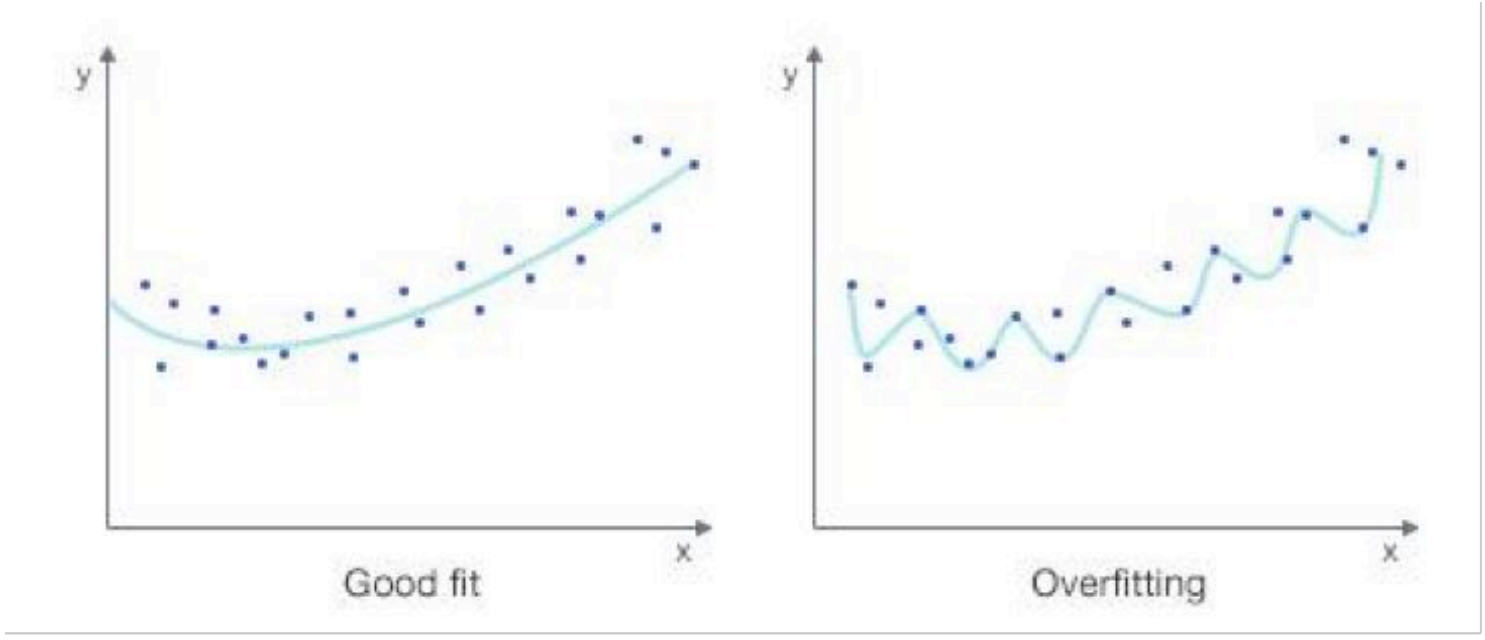



plays
Man

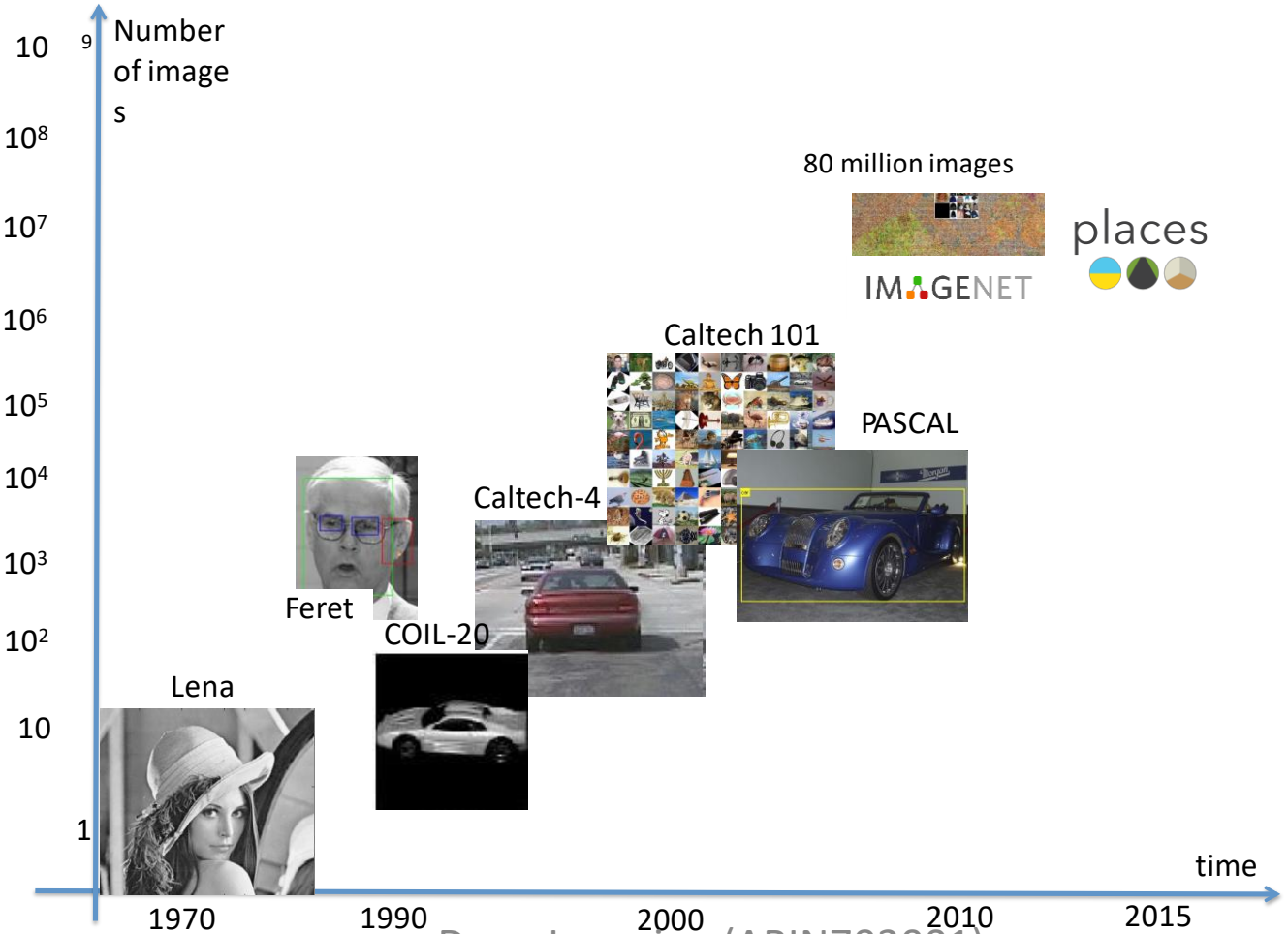


Overfitting problem

Insufficient data
Lots of parameters



Big data

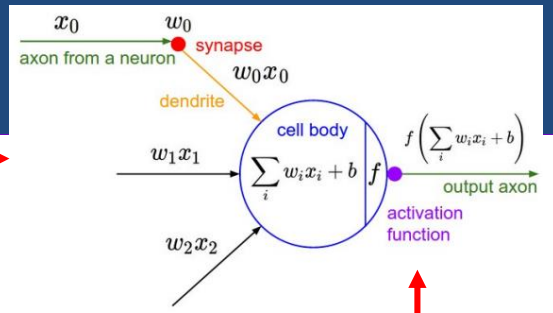

Number of images
10^9
10^8
10^7
10^6
10^5
10^4
10^3
10^2
10
1
80 million images
IMAGENET
places
Caltech 101
PASCAL
Caltech-4
Feret
COIL-20
Lena
time
1970
1990
2000
2010
2015

The time of big data

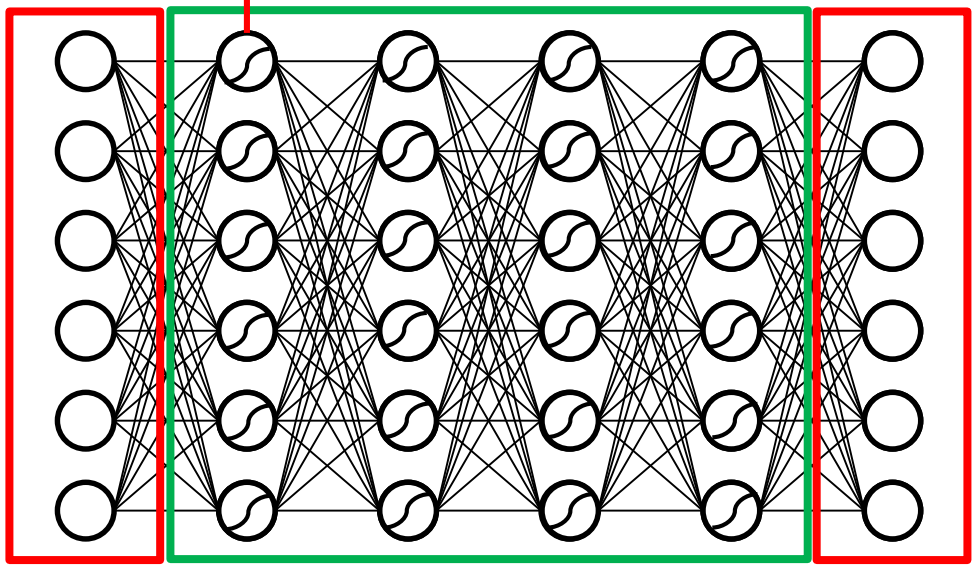
Parallel Processing

Let’s take a closer look.

Base of Deep Neural Networks

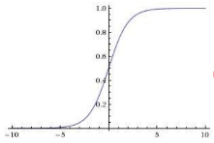


<Hidden Layers>

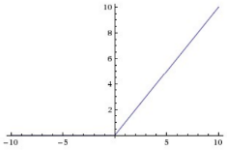


<Input Layer>

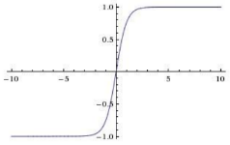
<Output Layer>



Sigmoid

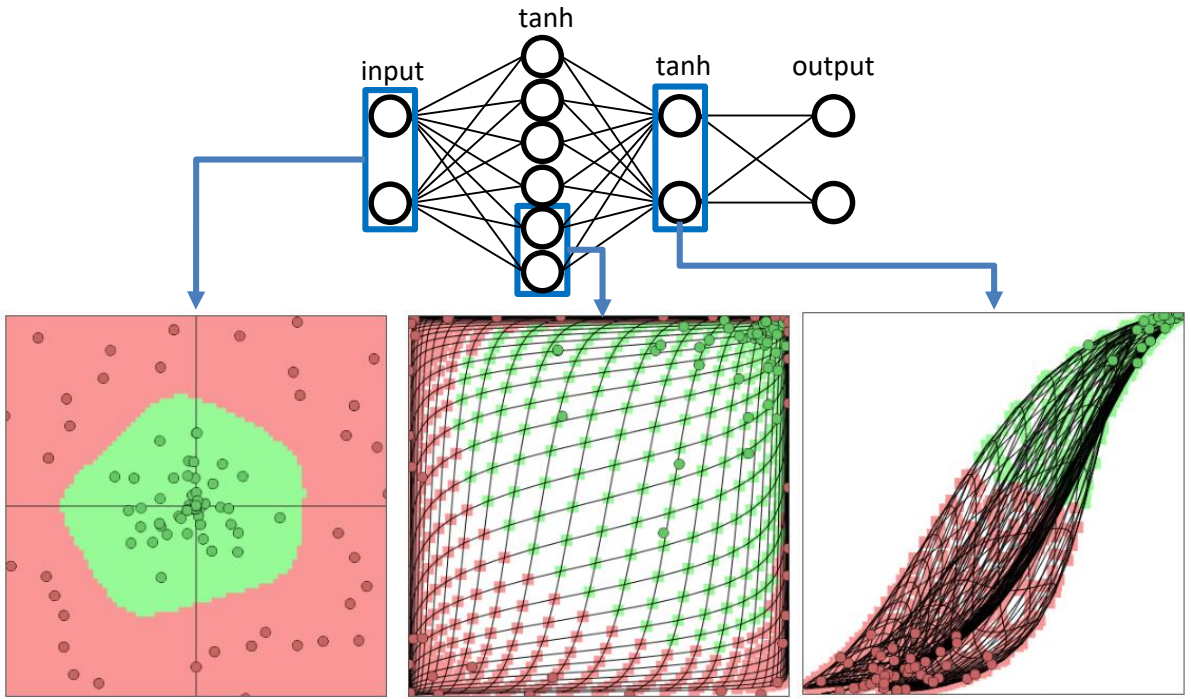


Rectified Linear Unit



Tangenth

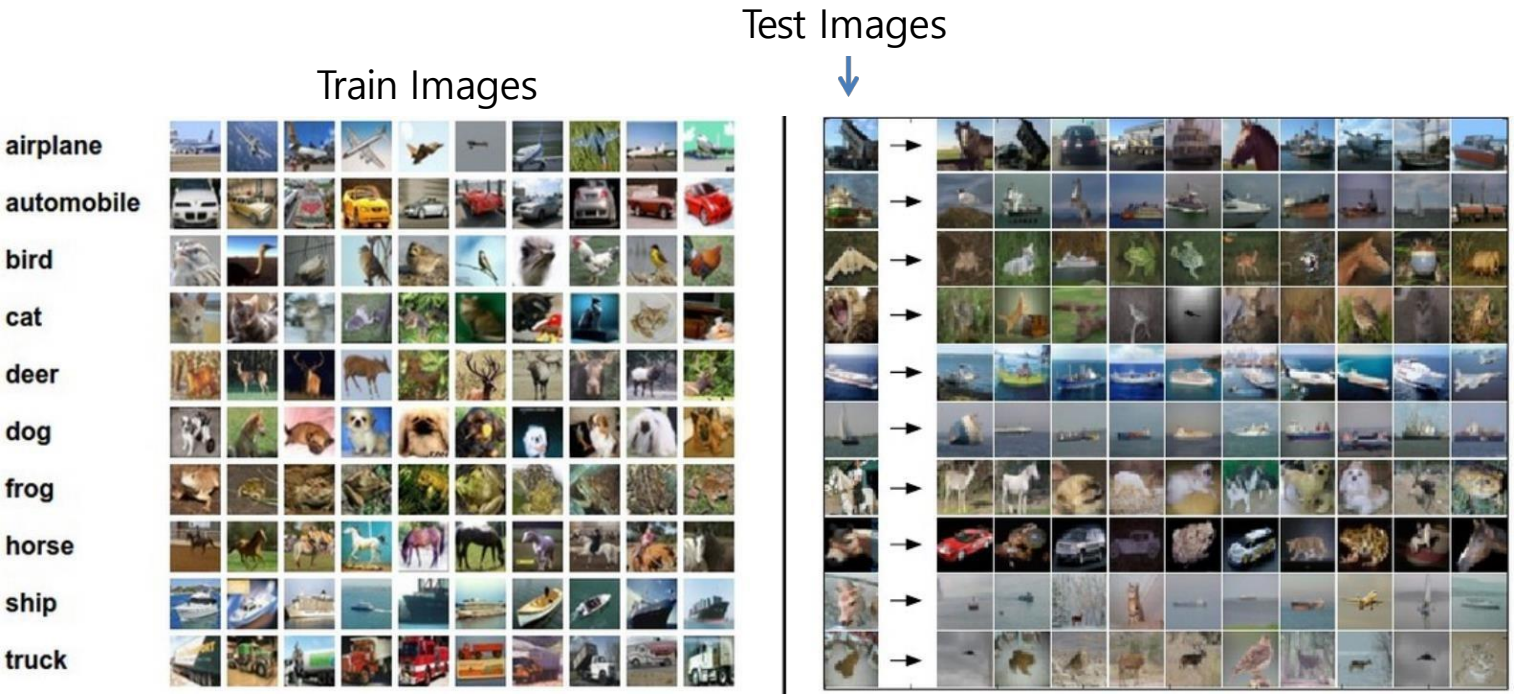
Effectiveness of Deep Neural Network



https://cs.stanford.edu/people/karpathy/convnetjs/

Go back to the linear classifier.

K-Nearest Neighbor

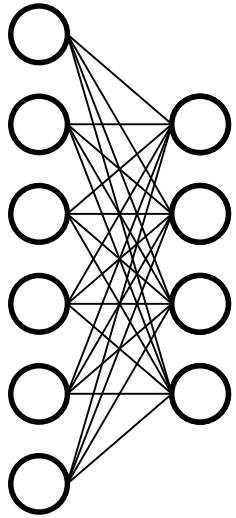


Left: Example images from the CIFAR-10 dataset. Right: first column shows a few test images and next to each we show the top 10 nearest neighbors in the training set according to pixel-wise difference.

Linear Classification

1. Define a score function

e.g.
dim(x) =
32x32x3

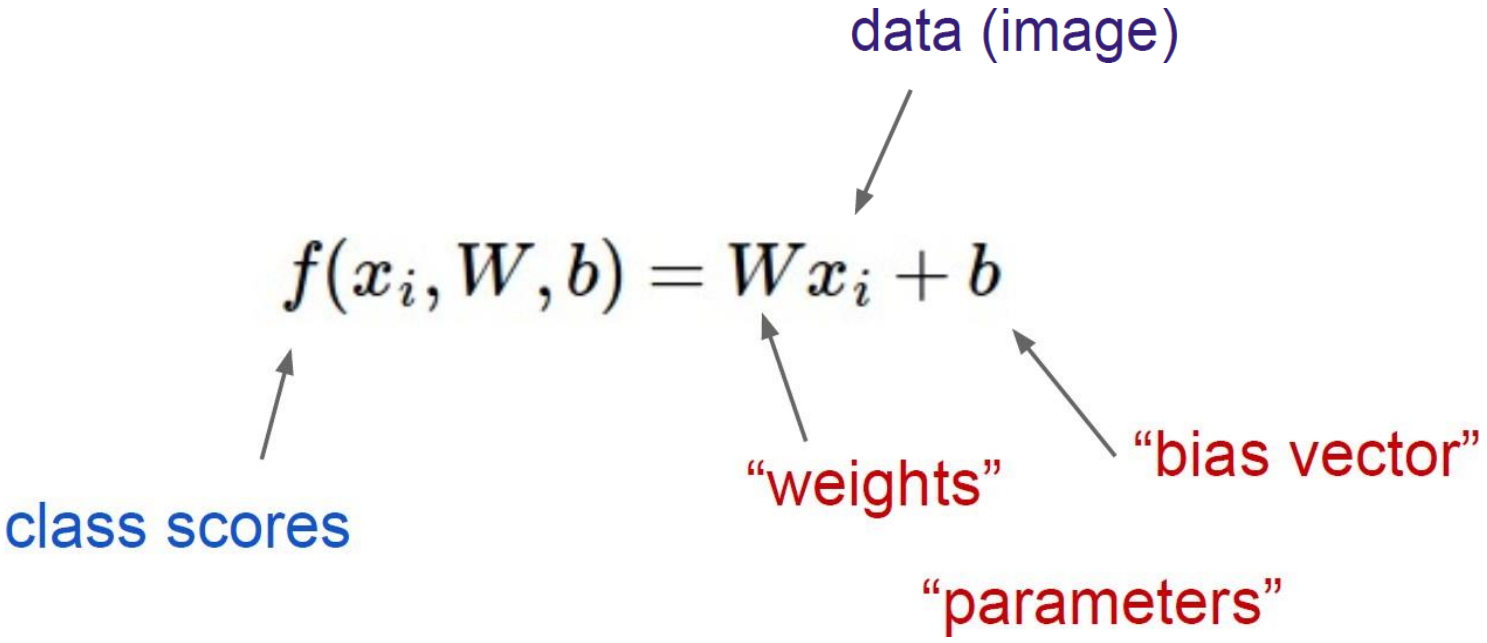


e.g.
dim(y) = 10
(cat, dog,
ship, plane)

class scores

Linear Classification

1. Define a score function



Linear Classification

1. Define a score function

- **Assume CIFAR-10 example**
- **32 x 32 x 3 images, 10 classes**

Array of **32x32x3** numbers
(3072 numbers total)

data (image)
[3072 x 1]

$$f(x_i, W, b) = Wx_i + b$$

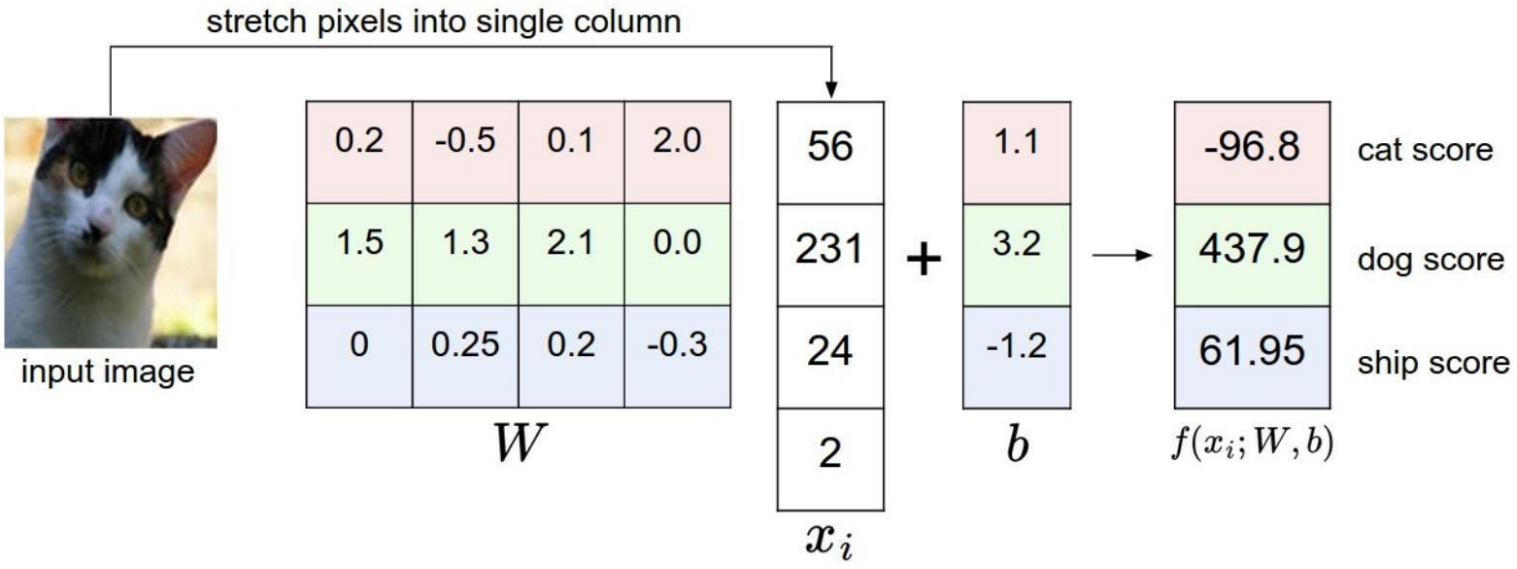
class scores
[10 x 1]

"weights"
[10 x 3072]

"bias vector"
[10 x 1]

Linear Classification

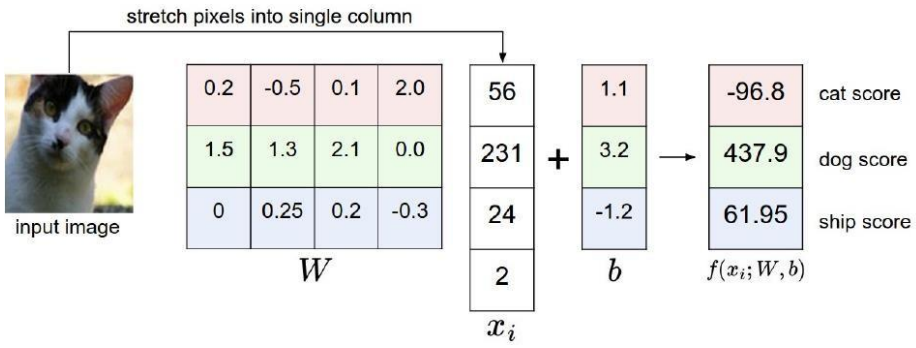
$$f(x_i, W, b) = Wx_i + b$$

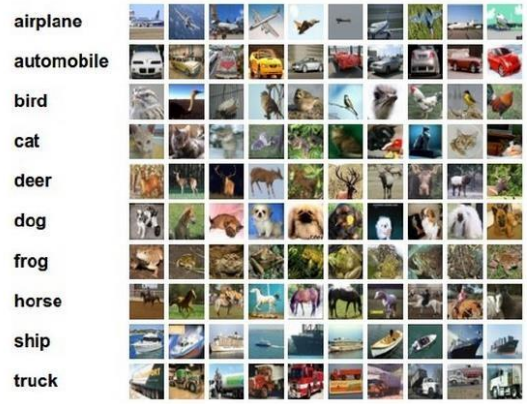


Interpreting a Linear Classifier

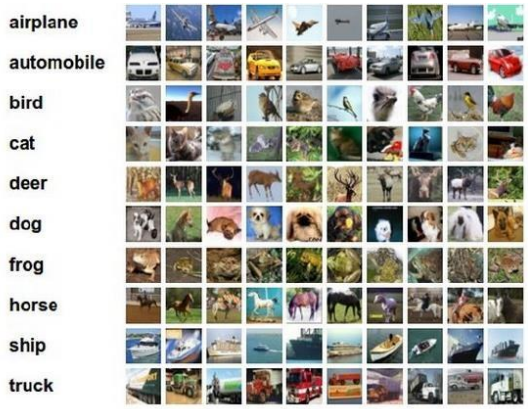
Question:
what can a linear classifier do?

$$f(x_i, W, b) = Wx_i + b$$





Interpreting a Linear Classifier

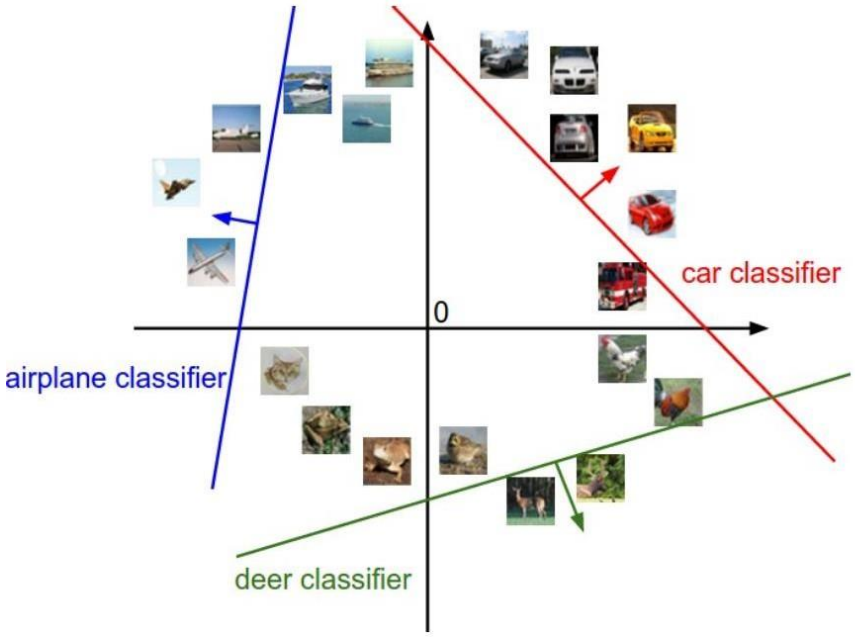


$$f(x_i, W, b) = Wx_i + b$$

Example training classifiers on CIFAR-10:

Skipping ahead a bit: Example learned weights at the end of learning for CIFAR-10. Note that, for example, the ship template contains a lot of blue pixels as expected. This template will therefore give a high score once it is matched against images of ships on the ocean with an inner product.

Interpreting a Linear Classifier

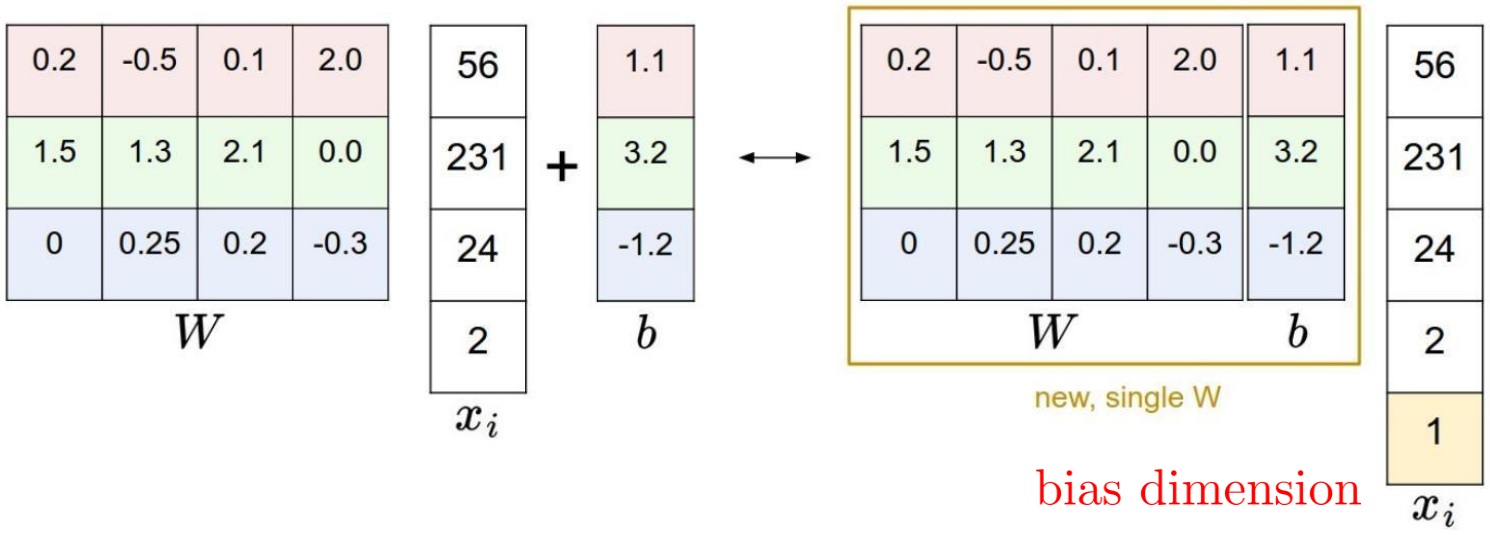


$$f(x_i, W, b) = Wx_i + b$$

Bias trick

$$f(x_i, W, b) = Wx_i + b \longrightarrow f(x_i, W) = Wx_i$$

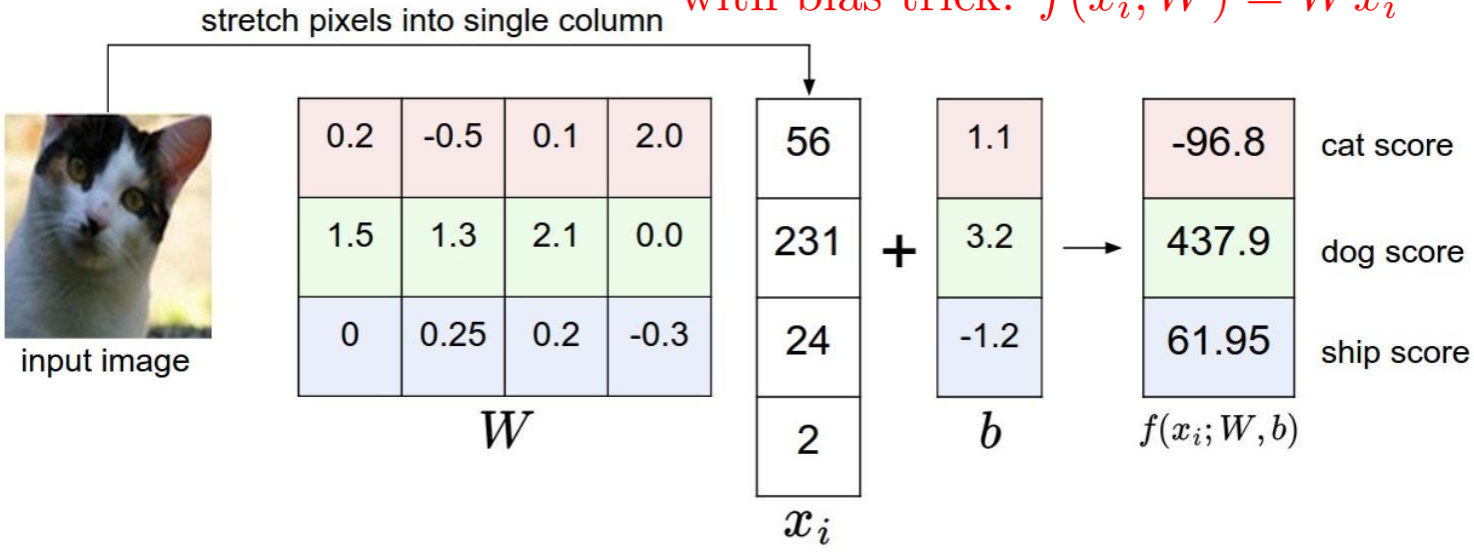
bias column



So far:

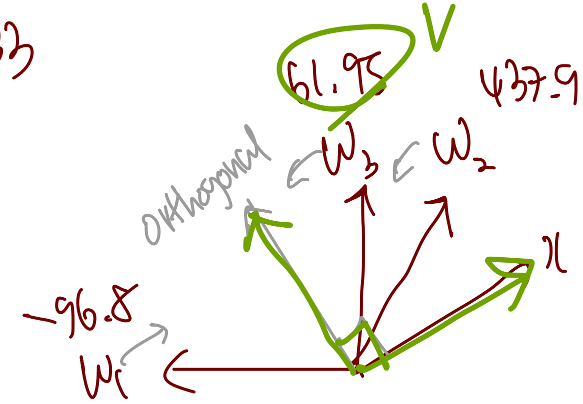
We defined a (linear) **score function**: $f(x_i, W, b) = Wx_i + b$

with bias trick: $f(x_i, W) = Wx_i$



How can we tell wheter W in $f(x_i, W) = Wx_i$ is good or bad ?

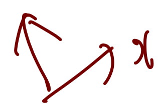
#33



★★★ (weight 찾아간다는 건, w_2 & w_3 x와 멀어지도록 ; dissimilarity ↑
w_1 , 가까워지도록)

< 물리적 공간에서의 Visualization >

; 내적이라는 것의 (공간상의) 물리적 의미 :



positive 90° 이내 : 내적 값 큼

$\theta = 90°$,

negative 90° 이상 작다

Softmax Classifier

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

score function is the same

Softmax classifier is a classfier with the following loss function:

a different loss function

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

softmax function

cross-entropy loss

divide by 2.71+0.14+1

$$[1, -2, 0]^T \rightarrow [e^1, e^{-2}, e^0]^T = [2.71, 0.14, 1]^T \rightarrow [0.7, 0.04, 0.26]^T$$

score vector f

now nonnegative

sum to one

probability for the class 0

Softmax Classifier

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

unnormalized probabilities

	unnormalized log probabilities	exp	unnormalized probabilities	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	

Softmax Classifier

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

unnormalized probabilities

Q: At initialization W is small so all $f(x, W) \approx 0$. What is the loss?

	unnormalized log probabilities	exp	unnormalized probabilities	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	

Cross-Entropy

Information theory view. The *cross-entropy* between a "true" distribution p and an estimated distribution q is defined as:

$$H(p, q) = -\sum_x p(x) \log q(x)$$

$$p = [0, \ldots 1, \ldots, 0]$$

$$\mathbf{q} = \text{softmax}(\mathbf{f}) = [\frac{e^{f_0}}{\sum_j e^{f_j}}, \ldots, \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}, \ldots]$$

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

L2 Regularization

L2 regularization: motivation

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda R(W)$$

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0] \qquad R(w_1) = ||w_1||_2^2 = 1$$

$$w_2 = [0.25, 0.25, 0.25, 0.25] \qquad R(w_2) = ||w_2||_2^2 = 4 \cdot \frac{1}{16} = \frac{1}{4}$$

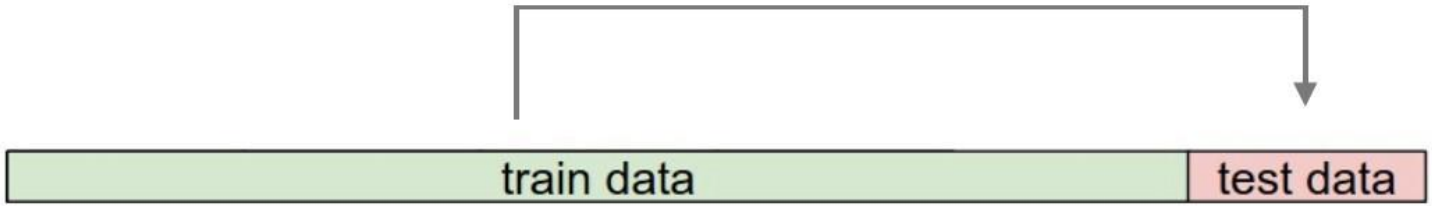
$$w_1^T x = w_2^T x = 1$$

same dot product (same score) : $f(x; w_1) = f(x; w_2)$,
but w_2 is preferred: weights in w_2 are smaller and more diffuse
$\Rightarrow$ the final classifier takes into account all input dimensions rather than a few input dimensions very strongly.
$\Rightarrow$ This leads to less overfitting.

Neural Networks for Visual Recognition, *Winter 2015*. (Stanford University)

Hyperparameter Tuning

Trying out what hyperparameters work best on test set:
Very bad idea. The test set is a proxy for the generalization performance

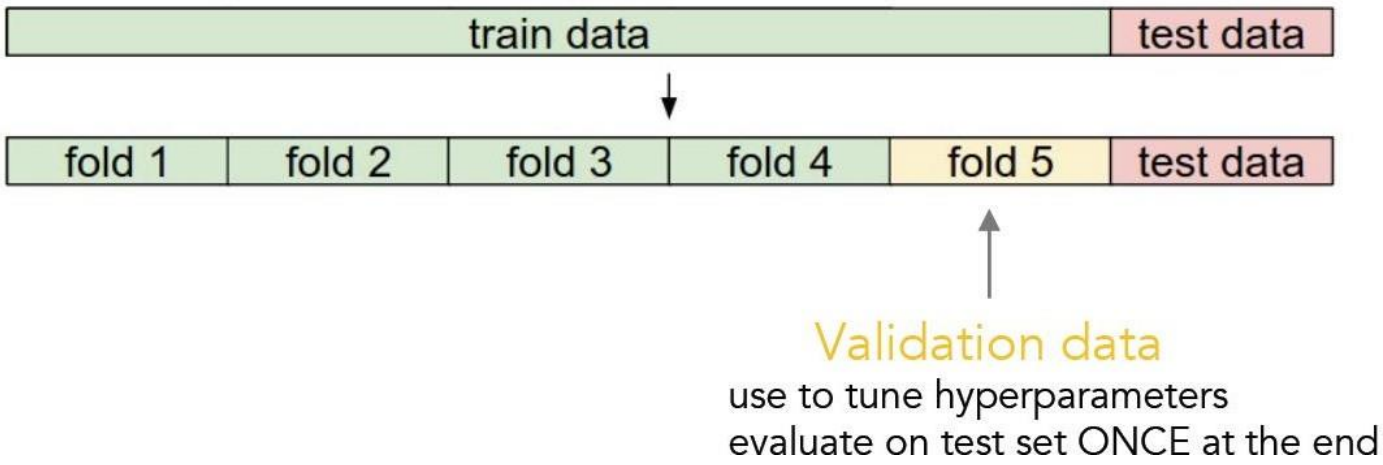


- If you tune your hyperparameters on the test set, you are effectively using the test set as the training set.
- If you only use the test set once at end, it remains a good proxy for measuring the generalization of your classifier.

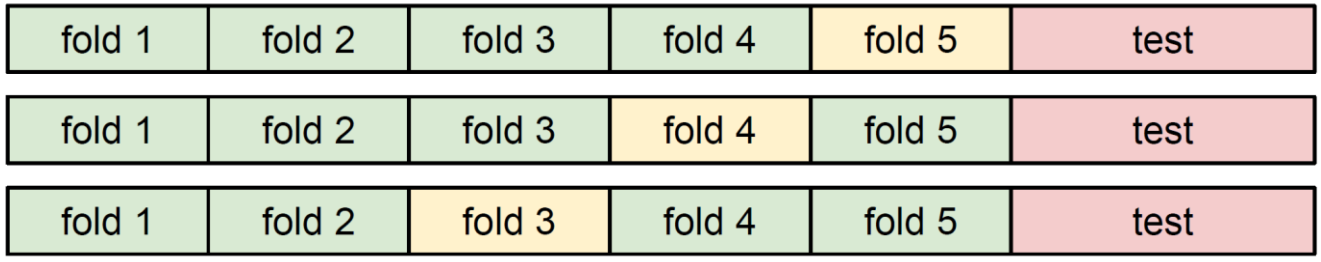
Using Validation Set

- Tune your hyperparameters using a validation set.

- If the size of training data (and therefore also the validation data) is small, use cross-validation.



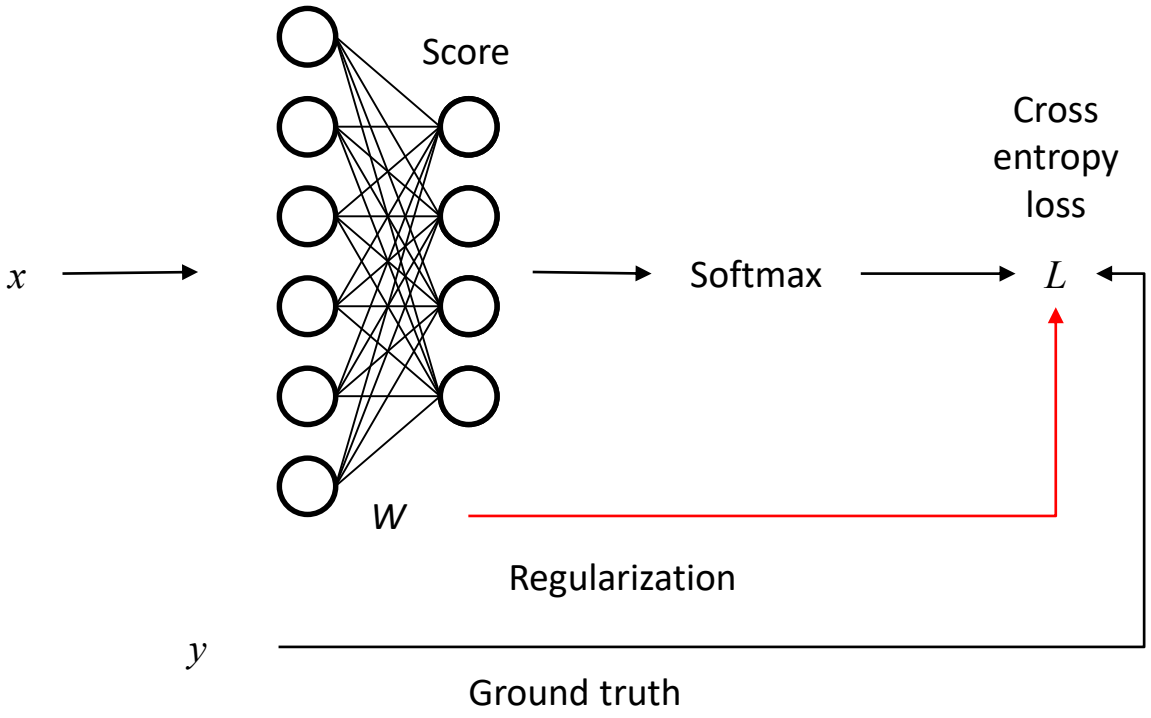
Cross-Validation



Cross-validation
cycle through the choice of which fold is the validation fold, average results.

Common data splits. A training and test set is given. The training set is split into folds (for example 5 folds here). The folds 1-4 become the training set. One fold (e.g. fold 5 here in yellow) is denoted as the Validation fold and is used to tune the hyperparameters. Cross-validation goes a step further iterates over the choice of which fold is the validation fold, separately from 1-5. This would be referred to as 5-fold cross-validation. In the very end once the model is trained and all the best hyperparameters were determined, the model is evaluated a single time on the test data (red).

Linear classification



Let’s compute weight parameters.

Optimization, Backpropagation

Optimization

- **Score function**

$$f(x_i, W, b) = Wx_i + b$$

- **Cross-entropy Loss + Softmax**

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

L_i : loss for ith data point x_i and true label y_i

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda R(W)$$

Optimization

- **Random search (**Very bad idea solution**)**
  - Simply try out many different random weights
  - Keep track of what works best

- **Random local search (**A better but still very bad idea solution**)**
  - Generate random perturbations δW
  - Update W if loss at the perturbed $W+\delta W$ is lower

- **Gradient Descent**
  - Use the gradient $\nabla f(x)$ vector which gives the direction of maximum descent at any point x

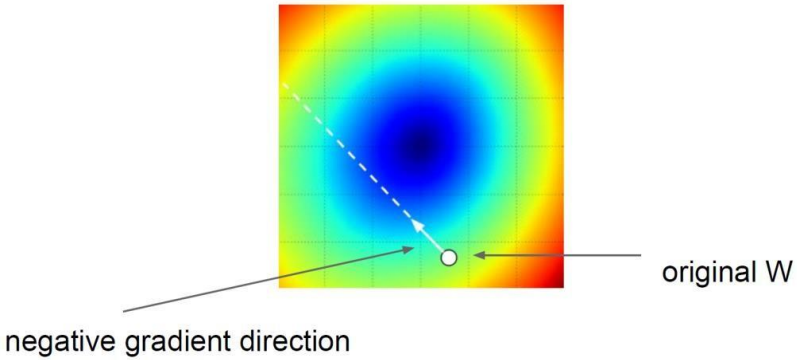
Optimization

- **Following the gradient**

In 1-dimension, the derivative of a function:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

In multiple dimension, the **gradient** is the vector of (partial derivatives).



Gradient Descent

1. **Weight random initialization.**
2. **Update weight using gradient.**

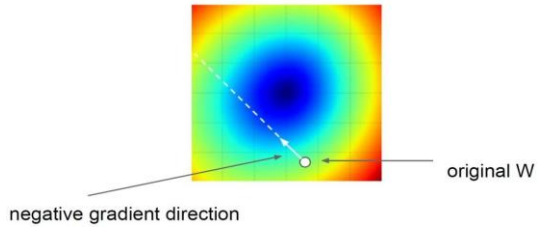
$$f(x_i, W, b) = Wx_i + b$$

class scores

"weights"

"bias vector"

$$w^{k+1} = w^k - \frac{\partial L}{\partial w}$$



Gradient Descent

- **Numerical gradient**

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

  - approximate, slow, easy to write

- **Analytic gradient**

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

  - exact, fast, error-prone

- In practice: Always use analytic gradient, but check implementation with numerical gradient. This is called a **gradient check**

Batch size of Gradient Descent

- **Batch Gradient Descent**
  - Use entire training set to compute the gradient
  - For large dataset, it is not very efficient.

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i$$

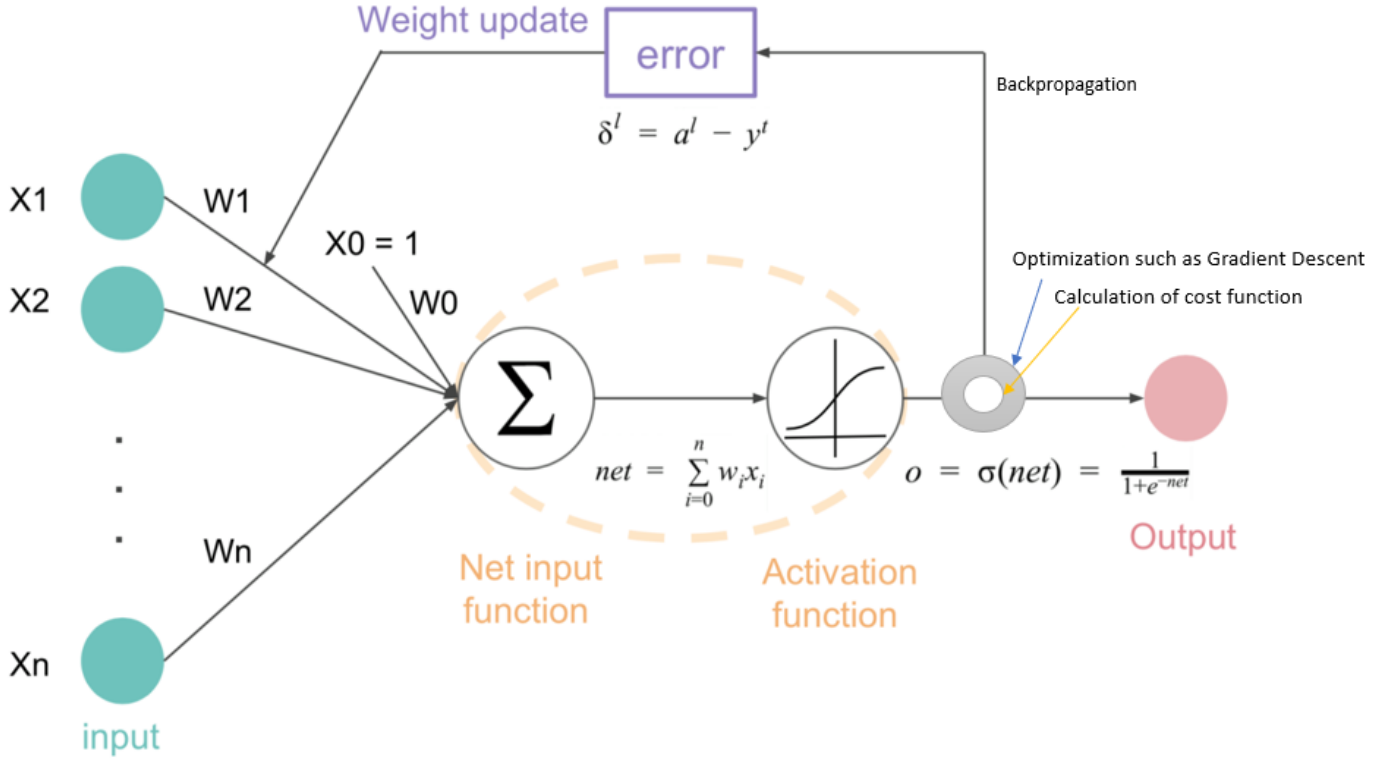
$$\frac{\partial L}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial L_i}{\partial w_j}$$

- **Mini-batch Gradient Descent**
  - only use a small portion of the training set to compute the gradient
  - Common mini-batch sizes are ~100 examples

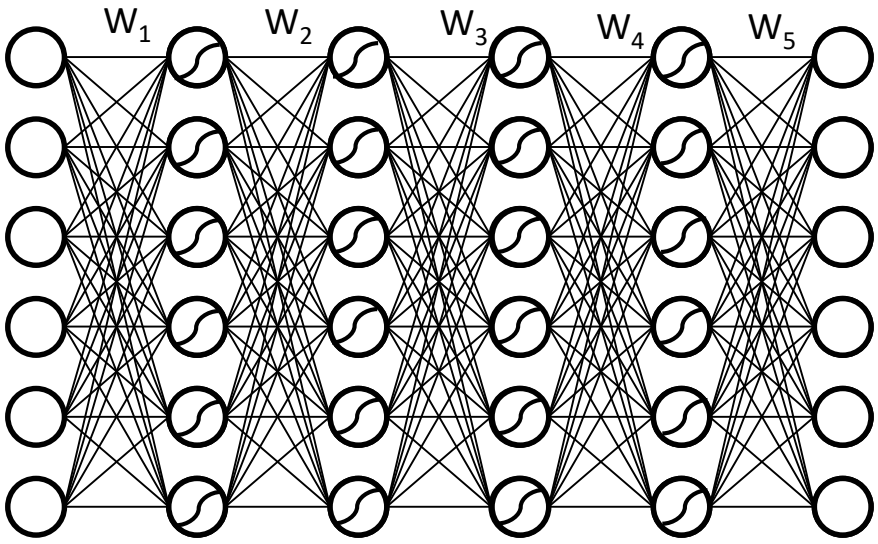
- **Stochastic Gradient Descent (SGD)**
  - use a single example at a time
  - also sometimes called **on-line** Gradient Descent

- **TIP**
  - Always use mini-batch gradient descent
  - People refer to mini-batch gradient descent as "doing SGD"
  - The mini-batch size is a hyper parameter, but it is not very common to cross-validate over it (usually based on practical concerns, e.g. space/time efficiency)

Backpropagation



Chain Rule



$$\frac{\partial\ \mathrm{L}(\mathrm{s}(\mathrm{f}_5(\mathrm{a}(\mathrm{f}_4(\mathrm{f}_3(\mathrm{f}_2(\mathrm{f}_1(\mathrm{x}))))))))}{\partial w_4} \Rightarrow \frac{\partial L}{\partial s}\frac{\partial s}{\partial f_5}\frac{\partial f_5}{\partial a}\frac{\partial a}{\partial f_4}\frac{\partial f_4}{\partial w_4}$$

=> Gradient

$$w_4(\tau+1) = w_4(\tau) - \eta \frac{\partial L}{\partial s}\frac{\partial s}{\partial f_5}\frac{\partial f_5}{\partial a}\frac{\partial a}{\partial f_4}\frac{\partial f_4}{\partial w_4}$$

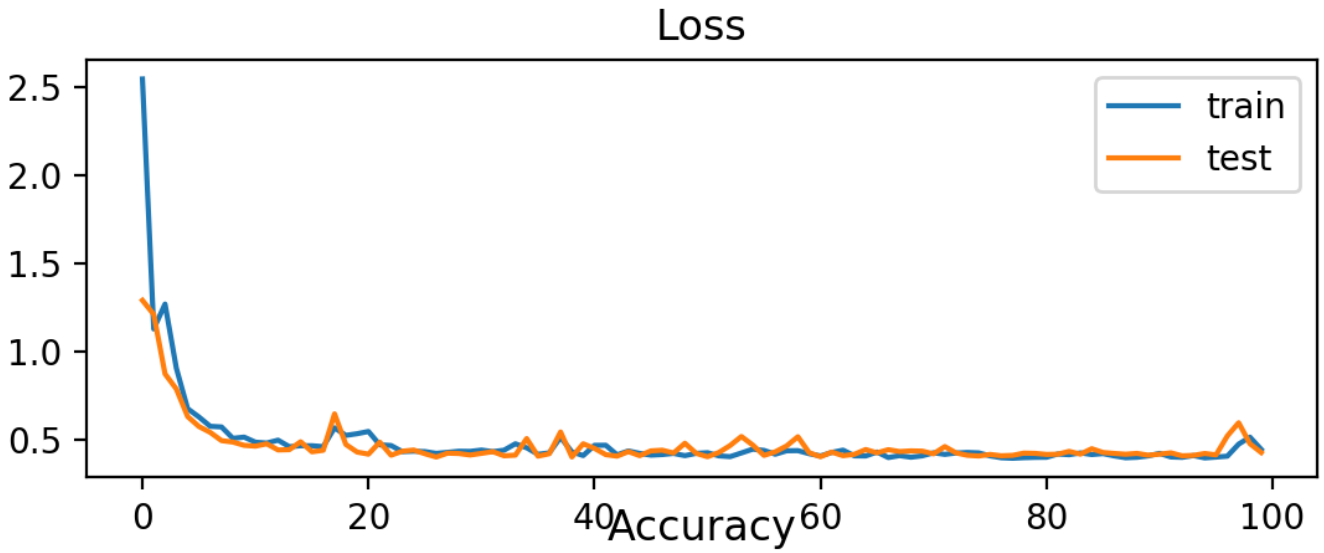
W_4 is a matrix

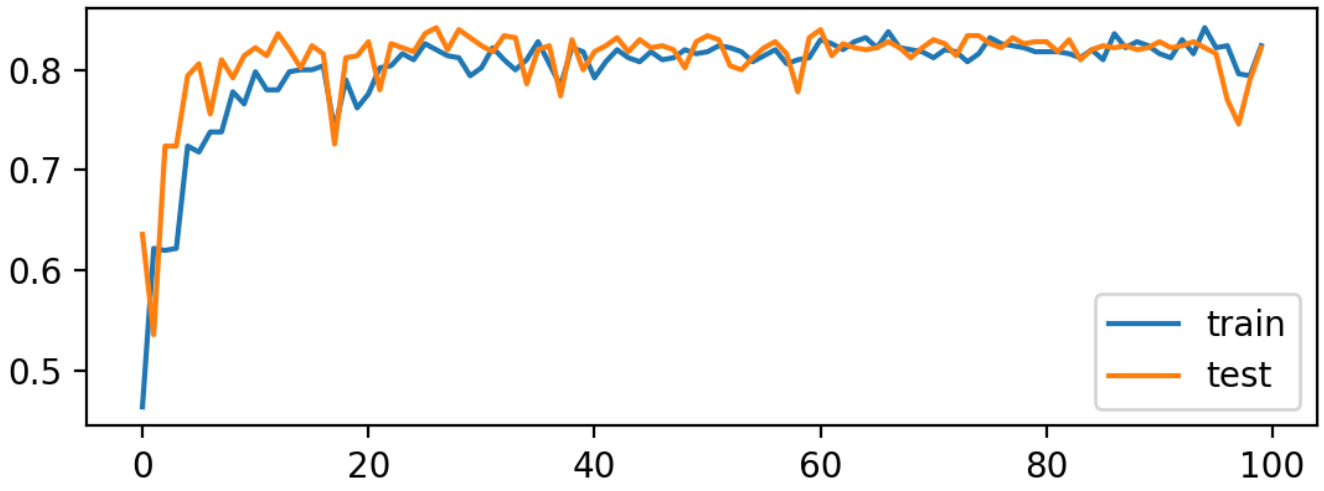
$$w_4^{(1,1)}(\tau+1) = w_4^{(1,1)}(\tau) - \eta \frac{\partial L}{\partial s}\frac{\partial s}{\partial f_5}\frac{\partial f_5}{\partial a}\frac{\partial a}{\partial f_4}\frac{\partial f_4}{\partial w_4^{(1,1)}}$$

$$w_4^{(1,2)}(\tau+1) = w_4^{(1,2)}(\tau) - \eta \frac{\partial L}{\partial s}\frac{\partial s}{\partial f_5}\frac{\partial f_5}{\partial a}\frac{\partial a}{\partial f_4}\frac{\partial f_4}{\partial w_4^{(1,2)}}$$

$$w_4^{(1,3)}(\tau+1) = w_4^{(1,3)}(\tau) - \eta \frac{\partial L}{\partial s}\frac{\partial s}{\partial f_5}\frac{\partial f_5}{\partial a}\frac{\partial a}{\partial f_4}\frac{\partial f_4}{\partial w_4^{(1,3)}}$$

.
.
.
.


Loss
2.5
2.0
1.5
1.0
0.5
train
test
0
20
40
60
80
100
epoch


Accuracy
0.8
0.7
0.6
0.5
train
test
0
20
40
60
80
100
epoch

Conclusion (Remind)

- **I am sure that you can answer following questions if you fully un derstood this material.**

1. What is Deep Neural Network? Linear Classifier?
2. What is Activation function?
3. What is Softmax?
4. What is Gradient descent?
5. What is Regularization?

Thank You!