Lecture 11
(Artificial) Neural Networks

Song Chaoyang

Assistant Professor

Department of Mechanical and Energy Engineering

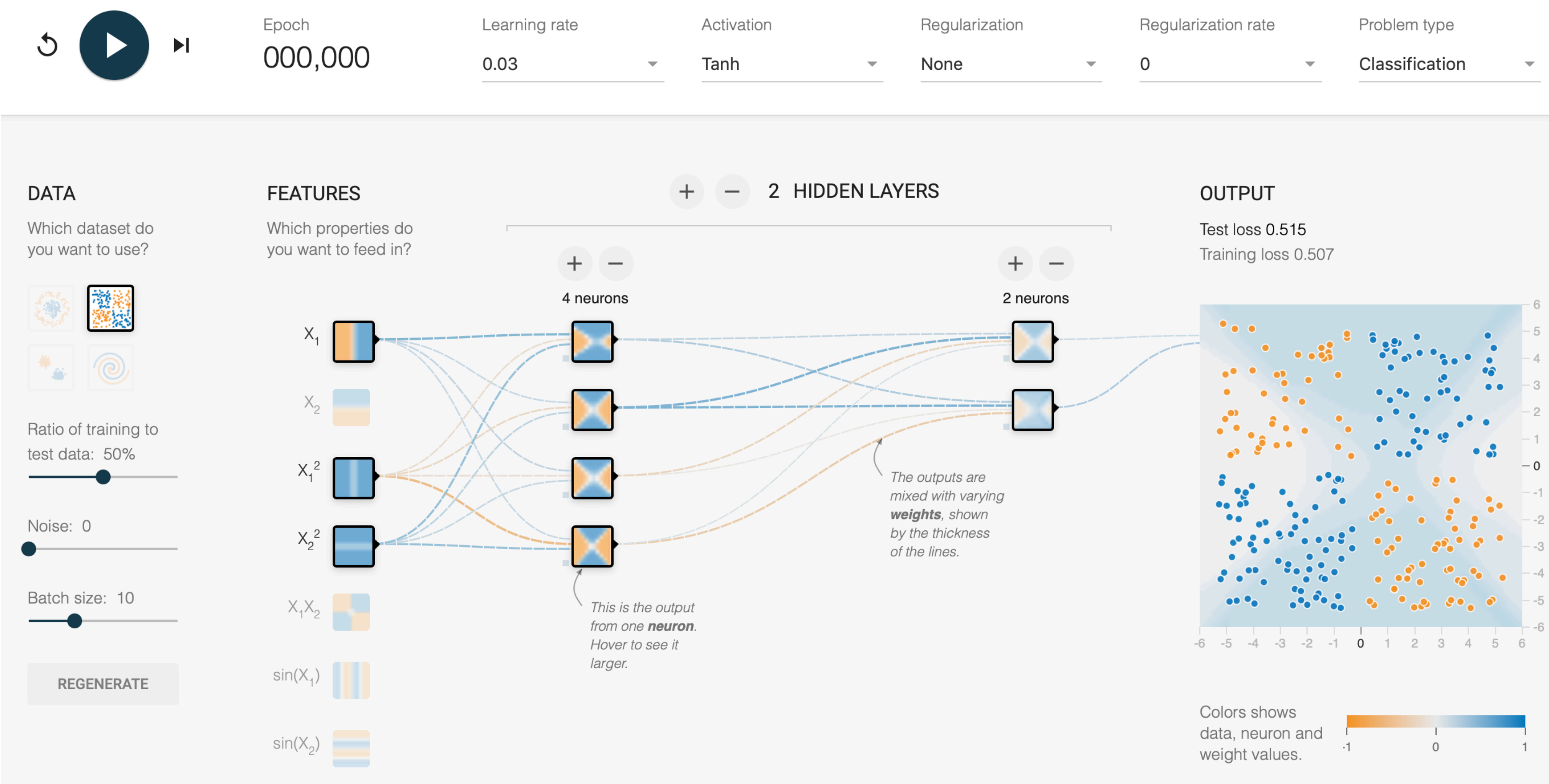
songcy@sustech.edu.cn

[Slides adapted from CS213n by Fei-Fei Li & CS230 by Andew Ng @ Stanford.]

SUSTech
Southern University
of Science and Technology

Tinker With a Neural Network

TensorBoard: Visualizing Learning

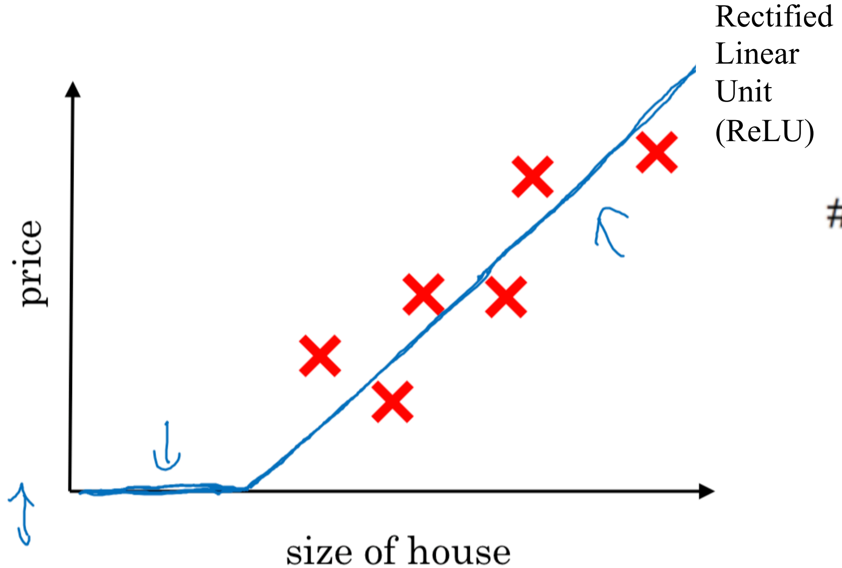


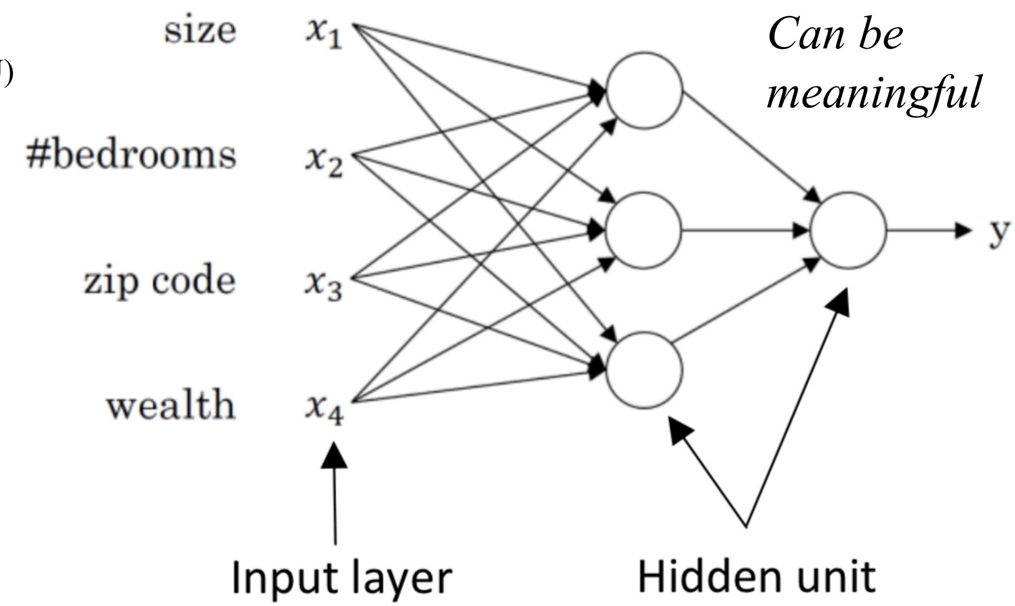
https://www.tensorflow.org/guide/summaries_and_tensorboard

What is a Neural Network?

It is a powerful learning algorithm inspired by how the brain works.

Housing Price Prediction





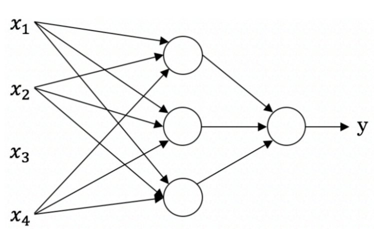
AncoraSIR.com

SUSTech
Southern University of Science and Technology

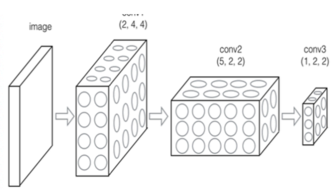
Supervised Learning with NNs

Examples, NNs & Data Types

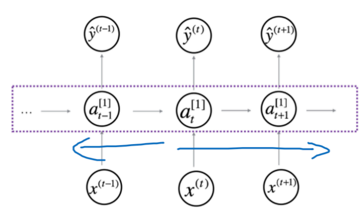
Input(x)	Output (y)	Application	
Home features	Price	Real Estate	Standard NN
Ad, user info	Click on ad? (0/1)	Online Advertising	
Image	Object (1,…,1000)	Photo tagging	CNN
Audio	Text transcript	Speech recognition	RNN
English	Chinese	Machine translation	
Image, Radar info	Position of other cars	Autonomous driving	Custom/ Hybrid



Standard NN



Convolutional NN



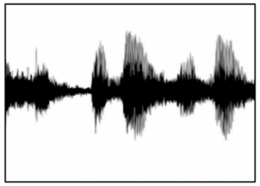
Recurrent NN

Structured Data

Size	#bedrooms	…	Price (1000$s)
2104	3		400
1600	3		330
2400	3		369
⋮	⋮		⋮
3000	4		540

User Age	Ad Id	…	Click
41	93242		1
80	93287		0
18	87312		1
⋮	⋮		⋮
27	71244		1

Unstructured Data



Audio

Image

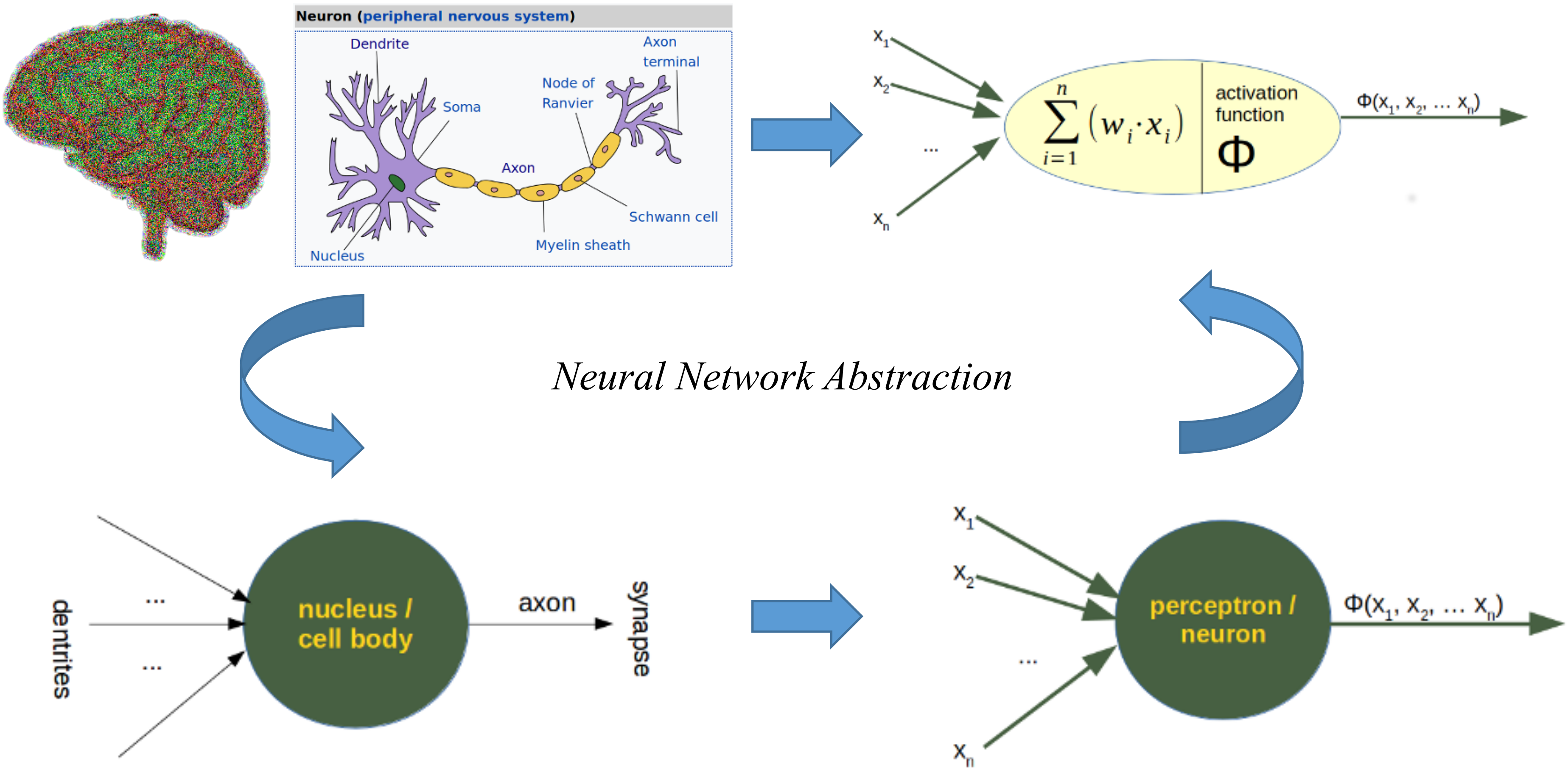
Four scores and seven years ago…

Text

AncoraSIR.com

What is a Neural Network?

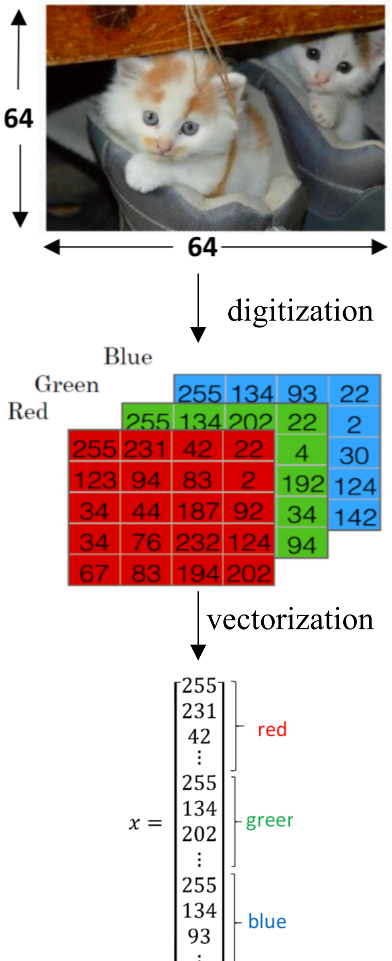
From biological inspiration to mathematical modeling



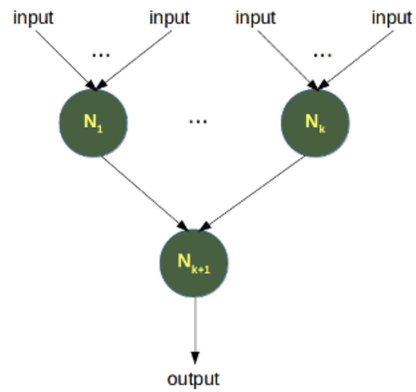
AncoraSIR.com

Example: Binary Classification

0 or 1 (*discrete value output*), it is a question …



AncoraSIR.com



A simple NN implementing AND

Input1	Input2	Output
0	0	0
0	1	0
1	0	0
1	1	1

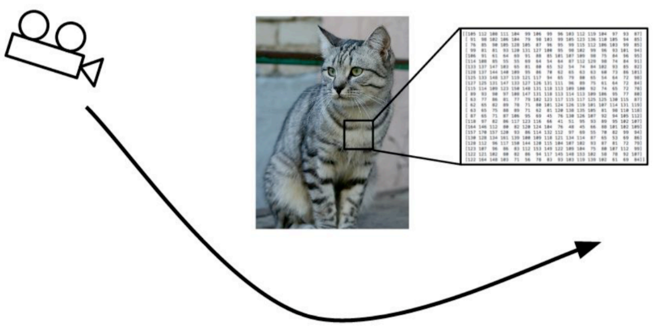
```
[0 0] 0
[0 1] 0
[1 0] 0
[1 1] 1
```

```python
#!/usr/bin/python3
# adapted from https://www.python-course.eu/neural_networks.php
# use for ME336 CoBot Learning @ SUSTech by Dr. Song Chaoyang

import numpy as np
class Perceptron:   # define the perceotron

    def __init__(self, input_length, weights = None):
        if weights == None:
            self.weights = np.ones(input_length) * 0.5
        else:
            self.weights = weights

    @staticmethod
    def unit_step_function(x):
        if x > 0.5:
            return 1
        return 0

    def __call__(self, in_data):
        weighted_input = self.weights * in_data
        weighted_sum = weighted_input.sum()
        return Perceptron.unit_step_function(weighted_sum)

p = Perceptron(2, np.array([0.5, 0.5])) # config the perceptron

for x in [np.array([0, 0]), np.array([0, 1]),   # input
          np.array([1, 0]), np.array([1, 1])]:
    y = p(np.array(x))                          # output
    print(x, y)
```

Challenges of Recognition

Image Classification: A core task in computer vision

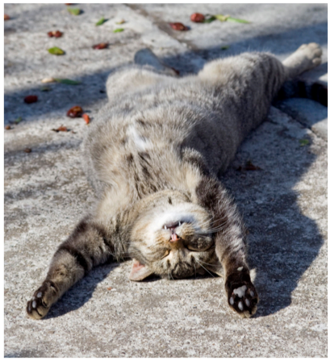
Viewpoint



All pixels change when the camera moves!

Illumination

Deformation

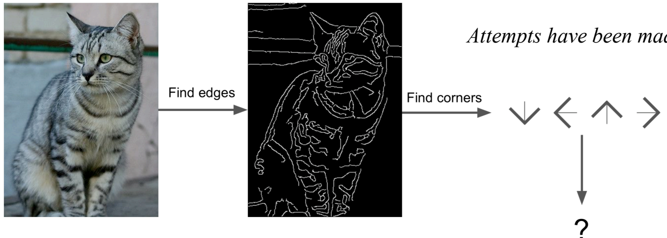


Occlusion

An image classifier

```
def classify_image(image):
  # Some magic here?
  return class_label
```

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.



Attempts have been made…

John Canny, "A Computational Approach to Edge Detection", IEEE TPAMI 1986

Clutter

Intraclass Variation

AncoraSIR.com

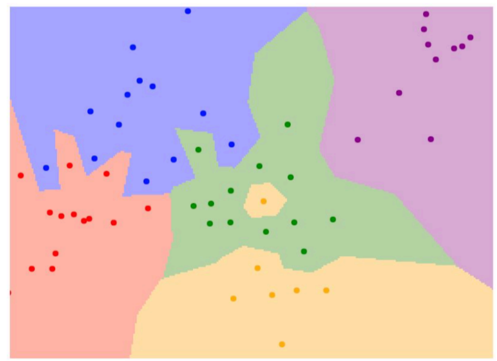
SUSTech Southern University of Science and Technology

Machine Learning: Data-driven approach

First classifier: Nearest Neighbor

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

1-NN classifier



```
def train(images, labels):
  # Machine learning!
  return model
```

→ Memorize all data and labels

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

→ Predict the label of the most similar training image

Distance Metric to compare images

L1 distance: $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

\-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

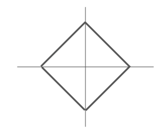
pixel-wise absolute value differences

46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

$\xrightarrow{\text{add}}$ 456

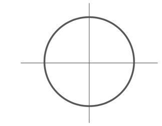
L1 (Manhattan) distance

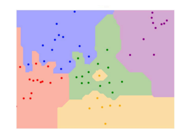
$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$



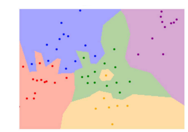
L2 (Euclidean) distance

$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$





K = 1



K = 1

Hyperparameters

- What is the best value of *k* to use?
- What is the best distance to use?

Choices about the algorithm that we ***set*** rather than ***learn***

- ***Very problem-dependent.***
- ***Must try them all out and see what works best.***

AncoraSIR.com

Southern University of Science and Technology

Parametric Approach: Linear Classifier

Image

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

$$\boxed{f(x,W)} = \boxed{W}\boxed{x} + \boxed{b}$$

3072x1

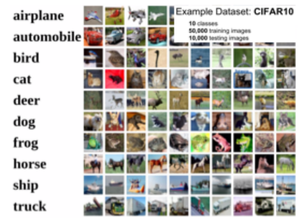
10x1 10x3072 10x1

$$\text{Image} \longrightarrow f(\mathbf{x},\mathbf{W}) \longrightarrow$$ **10** numbers giving class scores

$\uparrow$

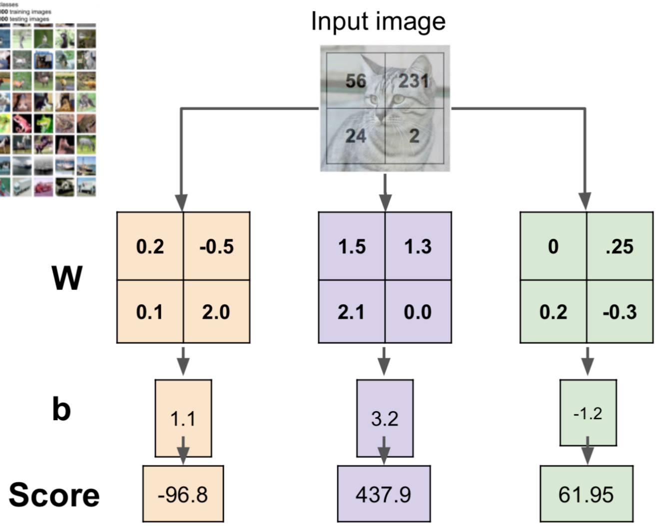
W

parameters or weights



A (linear) score function

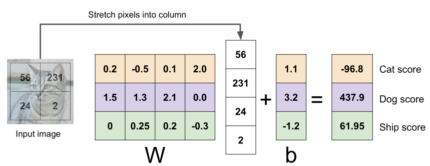
- *How can we tell whether this W is good or bad?*



Viewpoint

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Algebraic

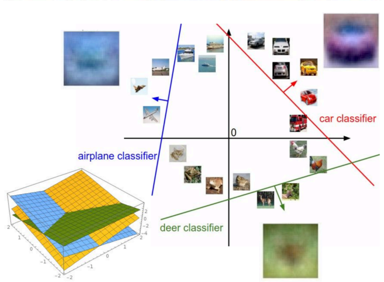


$$f(x, W) = Wx$$

Visual

One template per class

Geometric

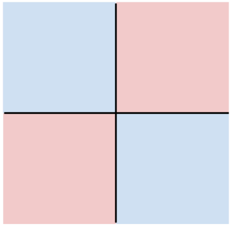


Hyperplanes cutting up space

Hard cases for a linear classifier

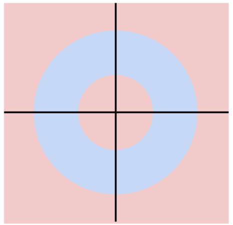
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



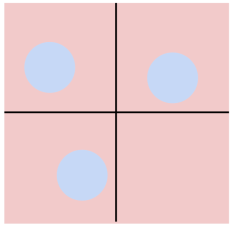
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

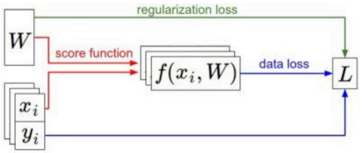
Class 2:
Everything else



AncoraSIR.com

Loss Function

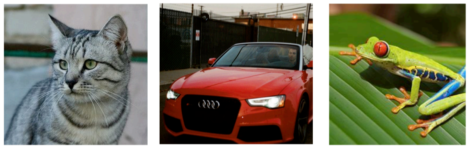
how good our current classifier is



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

Given a dataset of examples $\{(x_i, y_i)\}_{i=1}^{N}$

Where x_i is image and y_i is (integer) label



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	12.9

Loss over the dataset is a sum of loss over examples:

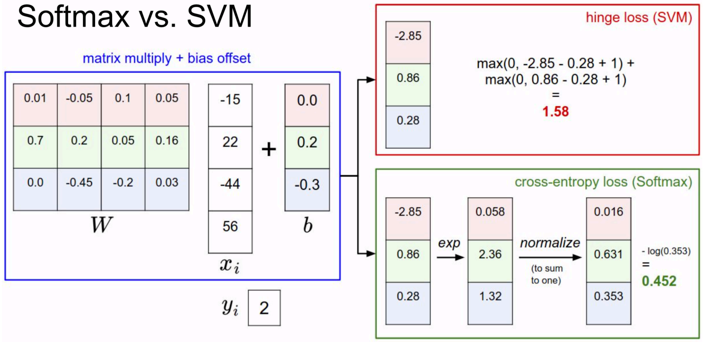
$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

We only care about the ***relative distance (property of the loss)****, not the* ***absolute difference (property of the data)***

e.g.
Multiclass
SVM loss

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Softmax vs. SVM



Regularization

λ = regularization strength (hyperparameter)

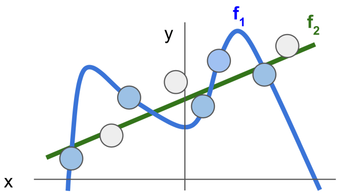
$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{} + \underbrace{\lambda R(W)}_{}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Simple examples
L2 regularization: $R(W) = \sum_k \sum_l W_{k,l}^2$
L1 regularization: $R(W) = \sum_k \sum_l |W_{k,l}|$
Elastic net (L1 + L2): $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

More complex:
Dropout
Batch normalization
Stochastic depth, fractional pooling, etc



Regularization pushes against fitting the data *too* well so we don't fit noise in the data

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

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

Probabilities must be >= 0

Probabilities must sum to 1

	Unnormalized log-probabilities / logits	$\xrightarrow{\text{exp}}$	unnormalized probabilities	$\xrightarrow{\text{normalize}}$	probabilities	compare	Correct
cat	**3.2**		**24.5**		**0.13**		**1.00**
car	5.1		164.0		0.87		0.00
frog	-1.7		0.18		0.00		0.00

Cross Entropy
$H(P, Q) = H(p) + D_{KL}(P \| Q)$

SUSTech
Southern University of Science and Technology

Optimization using Gradient Descent

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

In 1-dimension, the derivative of a function:

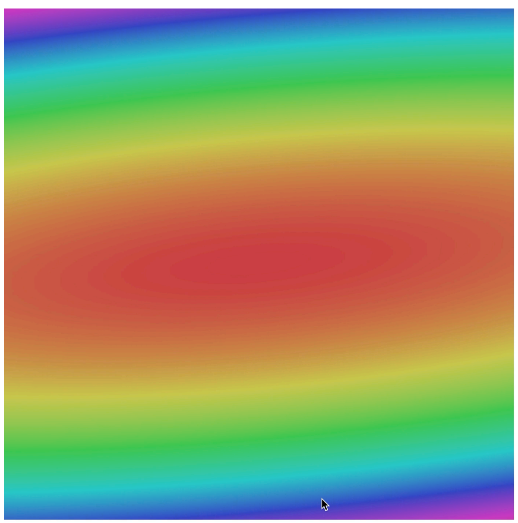
$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Numerical gradient: slow :(, approximate :(, easy to write :)
Analytic gradient: fast :), exact :), error-prone :(

In practice: Derive analytic gradient, check your implementation with numerical gradient

In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension

The slope in any direction is the **dot product** of the direction with the gradient
The direction of steepest descent is the **negative gradient**



```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

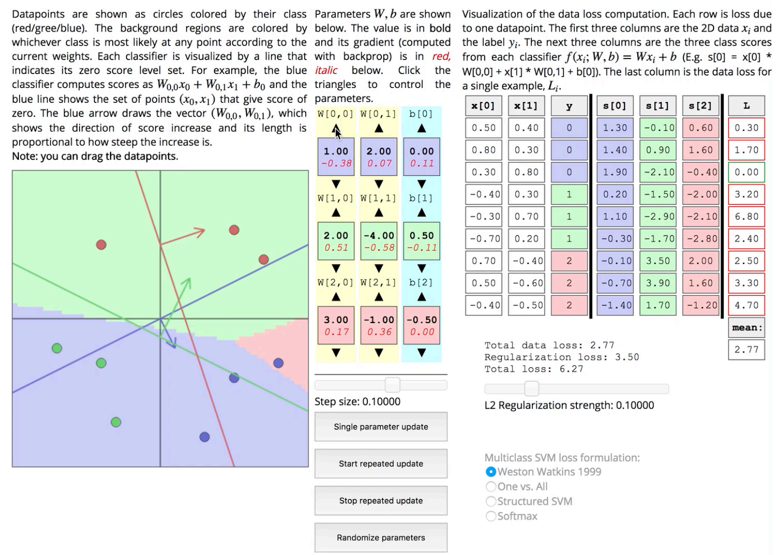
Stochastic Gradient Descent (SGD)

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

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

Full sum expensive when N is large!

Approximate sum using a **minibatch** of examples
32 / 64 / 128 common



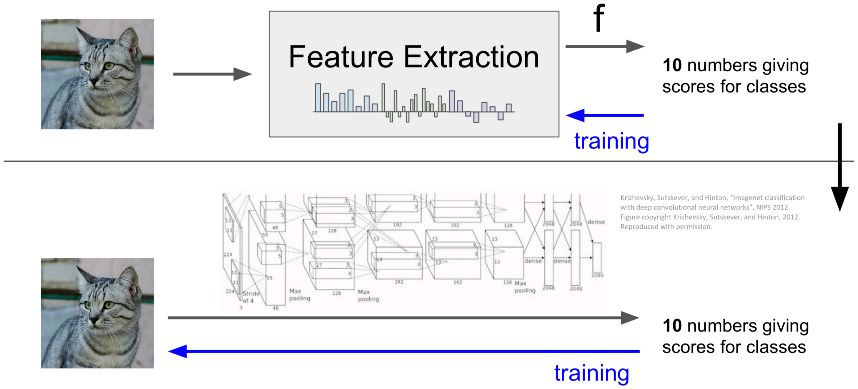
http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/

AncoraSIR.com

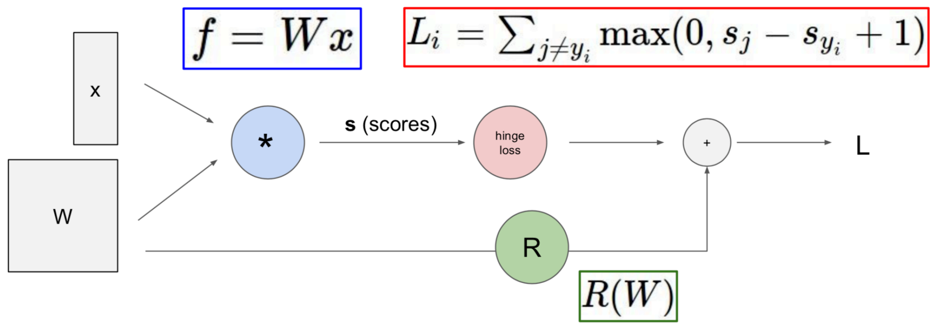
SUSTech
Southern University of Science and Technology

Convolutional Neural Network

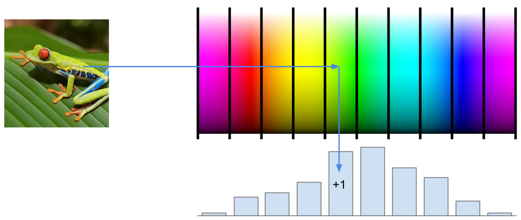
Image features vs ConvNets

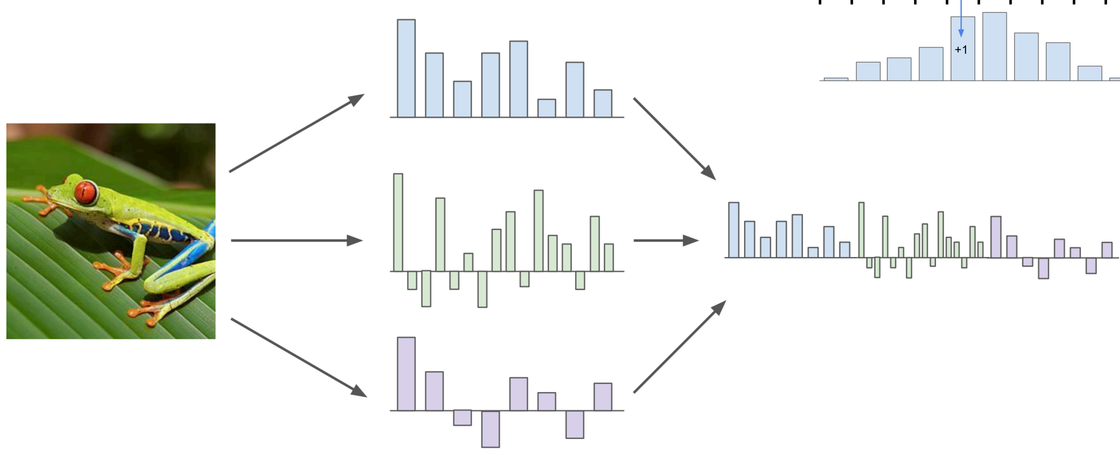


Computational Graph

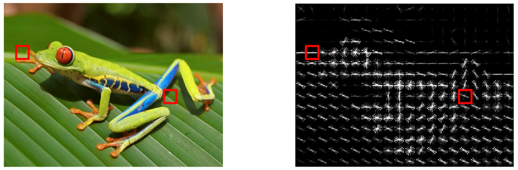


Example: Color Histogram





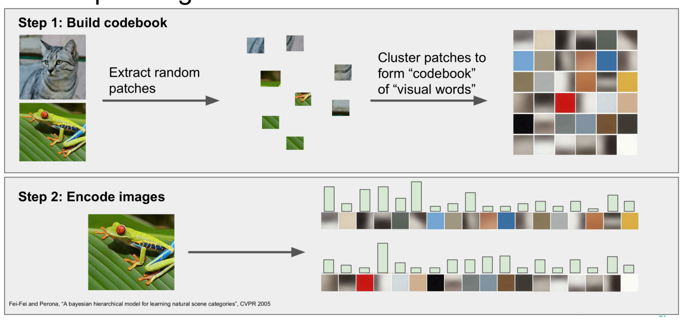
Example: Histogram of Oriented Gradients (HoG)



Divide image into 8x8 pixel regions
Within each region quantize edge direction into 9 bins

Example: 320x240 image gets divided into 40x30 bins; in each bin there are 9 numbers so feature vector has 30*40*9 = 10,800 numbers

Example: Bag of Words



Backpropagation

backward propagation of errors

Calculates the gradient of a loss function with respect to all the weights in the network.

Backpropagation: a simple example

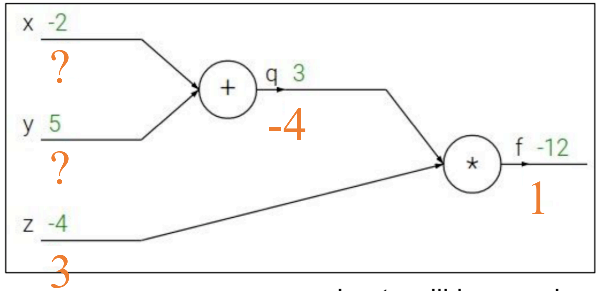
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

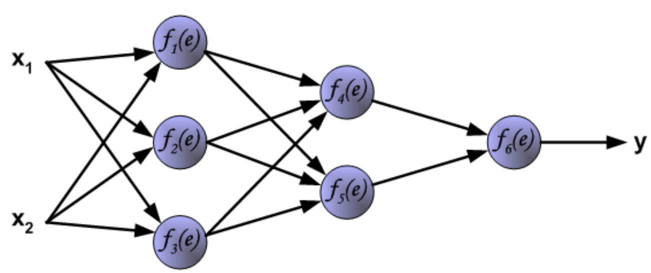
Upstream gradient   Local gradient

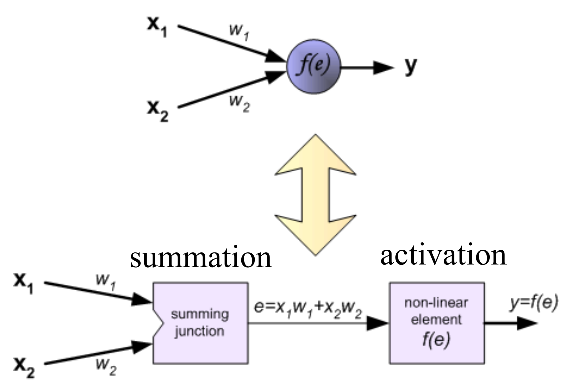
Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Upstream gradient   Local gradient

- neural nets will be very large: impractical to write down gradient formula by hand for all parameters
- **backpropagation** = recursive application of the chain rule along a computational graph to compute the gradients of all inputs/parameters/intermediates
- implementations maintain a graph structure, where the nodes implement the **forward**() / **backward**() API
- **forward**: compute result of an operation and save any intermediates needed for gradient computation in memory
- **backward**: apply the chain rule to compute the gradient of the loss function with respect to the inputs





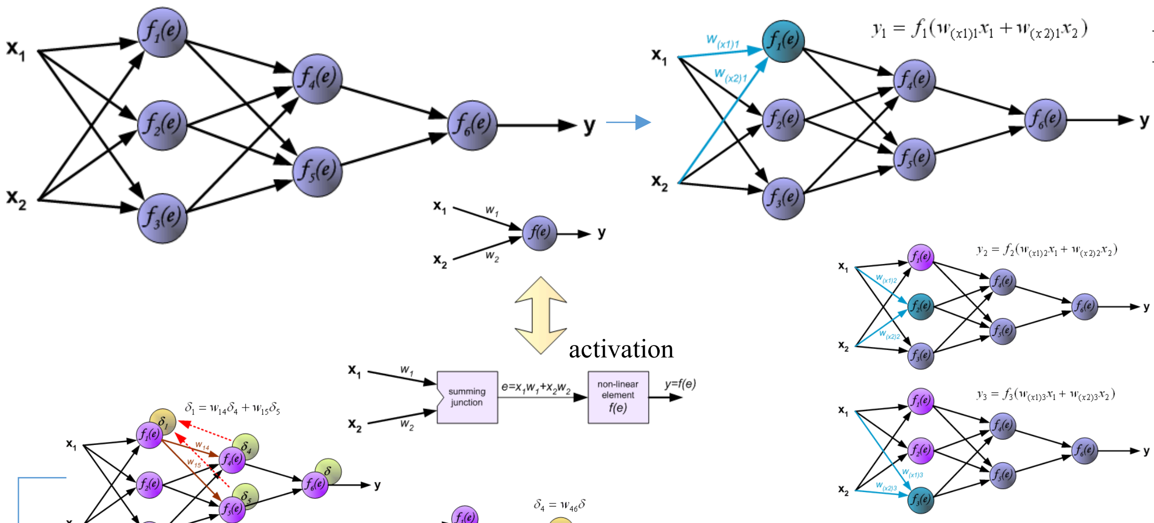
AncoraSIR.com

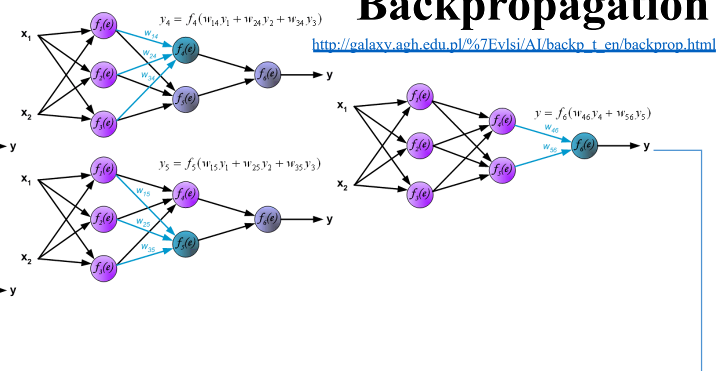
http://galaxy.agh.edu.pl/%7Evlsi/AI/backp_t_en/backprop.html

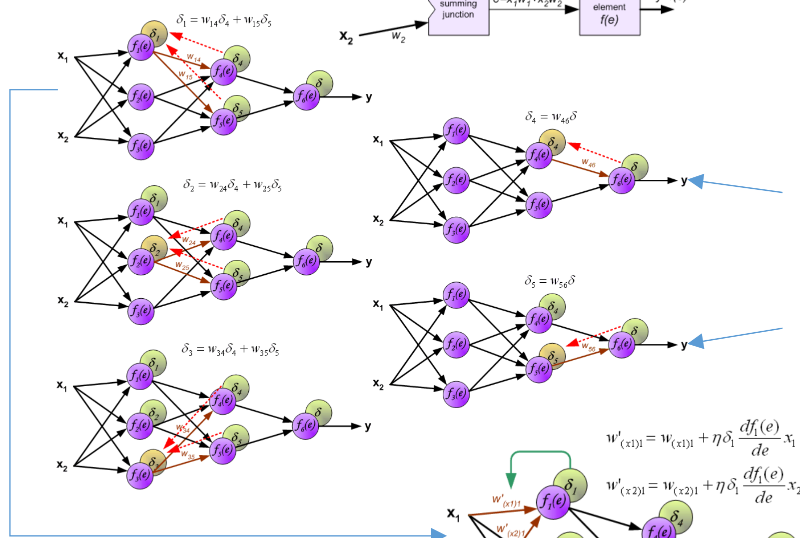
SUSTech
Southern University of Science and Technology

Principles of Training Multi-layer Neural Network Using Backpropagation

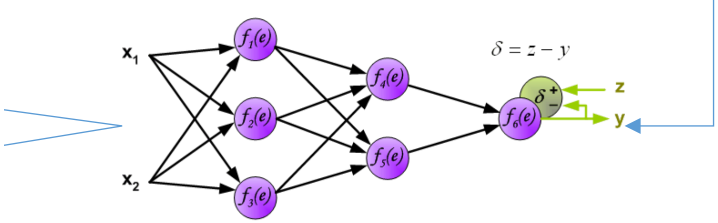
http://galaxy.agh.edu.pl/%7Evlsi/AI/backp_t_en/backprop.html





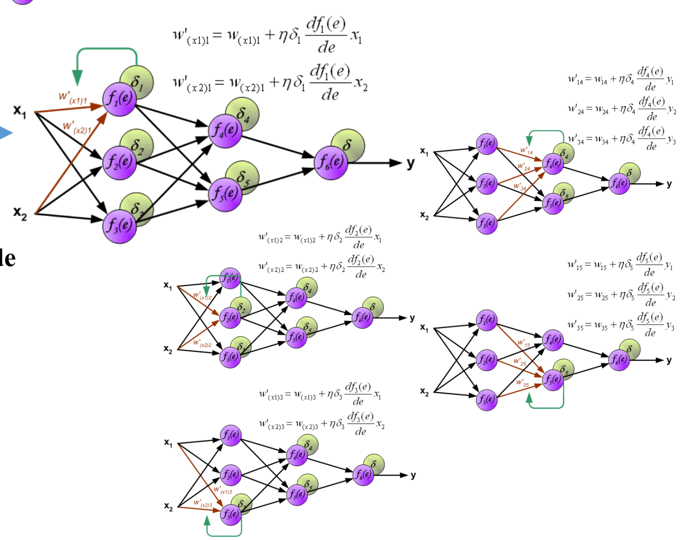


The idea is to propagate error signal *d* (computed in single teaching step) back to all neurons, which output signals were input for discussed neuron.



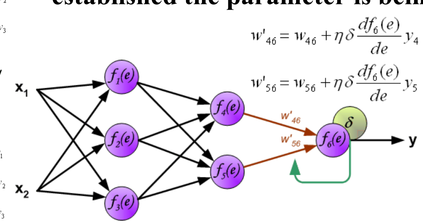
Coefficient η affects network teaching speed, to select this parameter:

- The first method is to start teaching process with large value of the parameter. While weights coefficients are being established the parameter is being decreased gradually.



When the error signal for each neuron is computed, the weights coefficients of each neuron input node may be modified.

In formulas on the right, *df(e)/de* represents derivative of neuron activation function (which weights are modified).



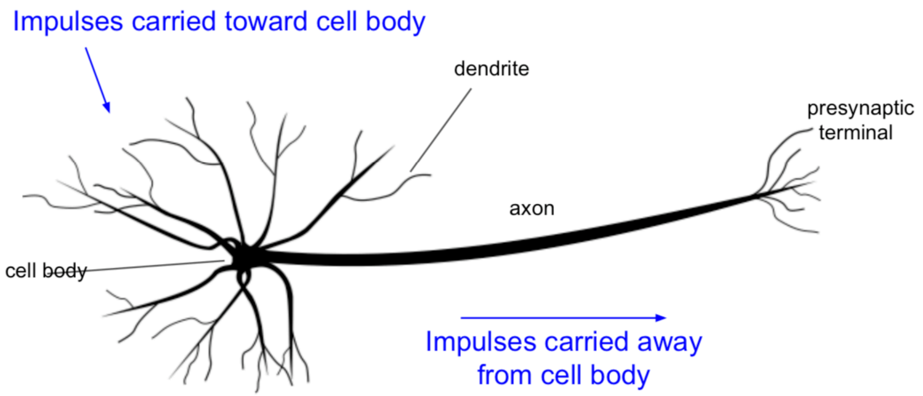
Finish one iteration of computation

- The second, more complicated, method starts teaching with small parameter value. During the teaching process the parameter is being increased when the teaching is advanced and then decreased again in the final stage. Starting teaching process with low parameter value enables to determine weights coefficients signs.

AncoraSIR.com

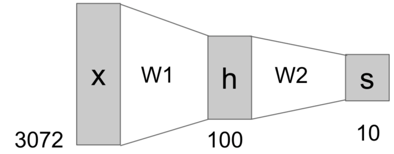
Neural Network

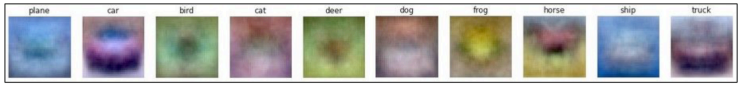
Biological Inspiration



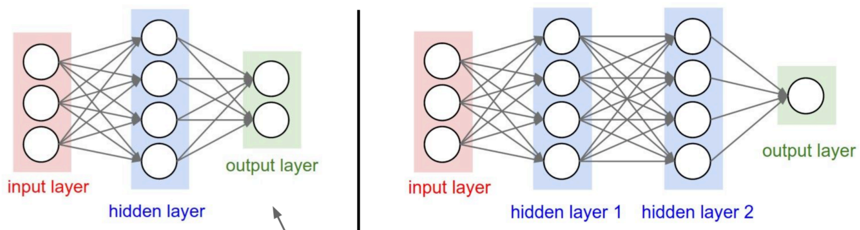
Linear score function: $f = Wx$

2-layer Neural Network $f = W_2 \max(0, W_1 x)$





Neural networks: Architectures



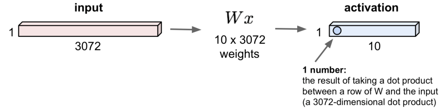
"2-layer Neural Net", or
"1-hidden-layer Neural Net"

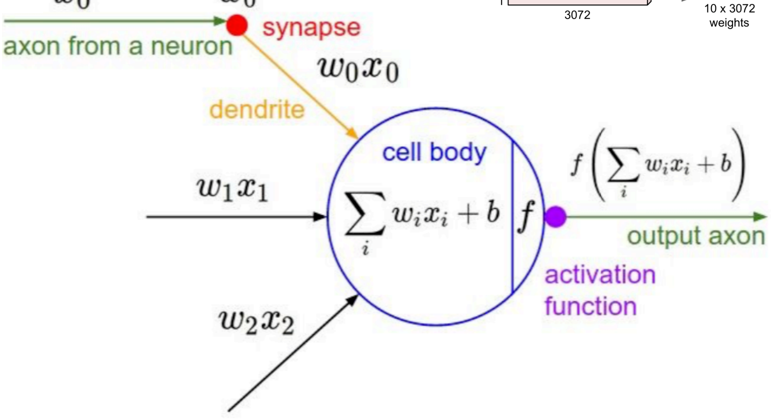
"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

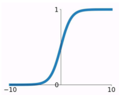




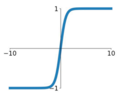

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate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

Activation functions

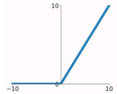
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



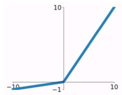
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

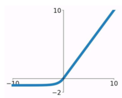


Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



AncoraSIR.com

SUSTech
Southern University of Science and Technology

Thank you!

Prof. Song Chaoyang

- Dr. Wan Fang (sophie.fwan@hotmail.com)

AncoraSIR.com