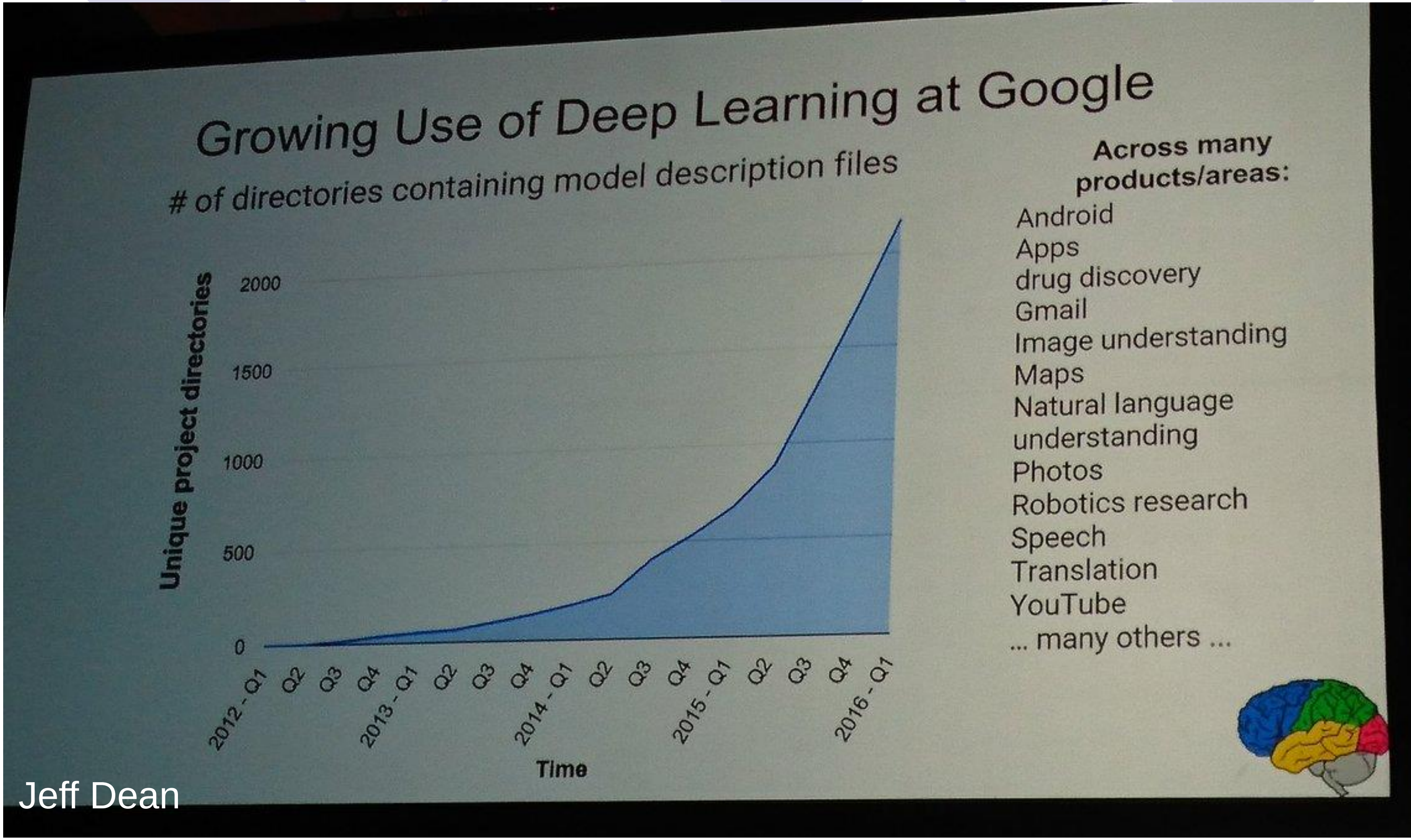
Computer Vision

01

Acknowledgement:
I wish to thank Li Ming for their lectures, notes and blogs.

Lecture 1: Introduction

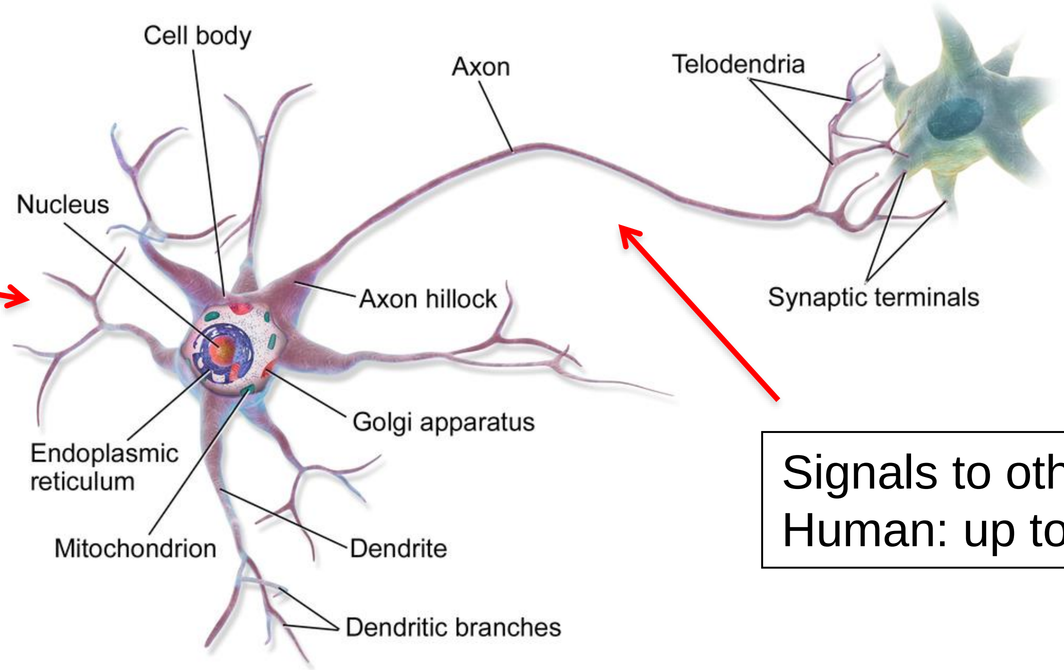


Many predictions, by 2025 – 2030, 1-2 billion people will lose their jobs to AI

Neurons in nature

- Human has ~100 billion neurons/nerve cells (& many more supporting cells)
- Each neuron has 3 parts: cell body, dendrites, axon connected up to ~10,000 other neurons. Passing signals to each other via 1000 trillion synaptic connections, approximately 1 trillion bit per second processor.
- Human memory capacity 1~1000 terabytes.

Signals from other neurons
There are synaptic weights &they adapt



Signals to other neurons.
Human: up to 1 meter long

What is our natural system good at?

- Vision
- Hearing (very adaptive)
- Speech recognition / speaking
- Driving
- Playing games
- Natural language understanding

- "Not good at": multiply 2 numbers, memorize a phone number.

Why not other types of learning?

- Linear regression?
  - Why is it linear?
- Bayesian?
  - What is the prior?
- SVM?
  - What are the features?
- Decision tree?
  - What are the nodes/variables?
- PAC learning?
  - What is the function to be learnt?
- KNN?
  - Cluster on what features?

These methods do not suit well with very complex models.

Ups and Downs of AI

- In the 1956 Dartmouth meeting, it has already mentioned neuron networks
- How did learning go deep. Easy hype target as AI borders science and science fiction.
  - Perceptron popularized by F. Rosenblatt, 1957 (Principles of Neurodynamics 1961).
    - *Times*: .. A revolution ..
    - *New Yorker* …
    - A science magazine title "Human brains replaced?"
    - False claims: "After 5 years all of us will have smart robots in our homes …"
    - It turns out that Rosenblatt's experiments of distinguishing tanks from trucks were because of lightings.
    - 1969, Minsky and Papert proved Perceptron, being a linear separator, is not very powerful. For example, can't do exclusive-or. But this was misconstrued as NNs being too week.
    - 1980s, multi-layer perceptron
    - 1986 Backpropagation, hard to train > 3 layers.
    - 1989: 1 hidden layer can do all, why deep?
    - 2006 RBM initialization (breakthrough) re-kindled fire.
    - < 2009: Game industry has pushed the growth of GPU's
    - 2011: Speech recognition (Waterloo professor Li Deng invited Hinton to Microsoft)
    - 2012: won ILSVRC image competition (with ImageNet training data)
  - 1980's expert system
  - Japan's 5^{th} generation computers (thinking machines)

Perceptron Architecture

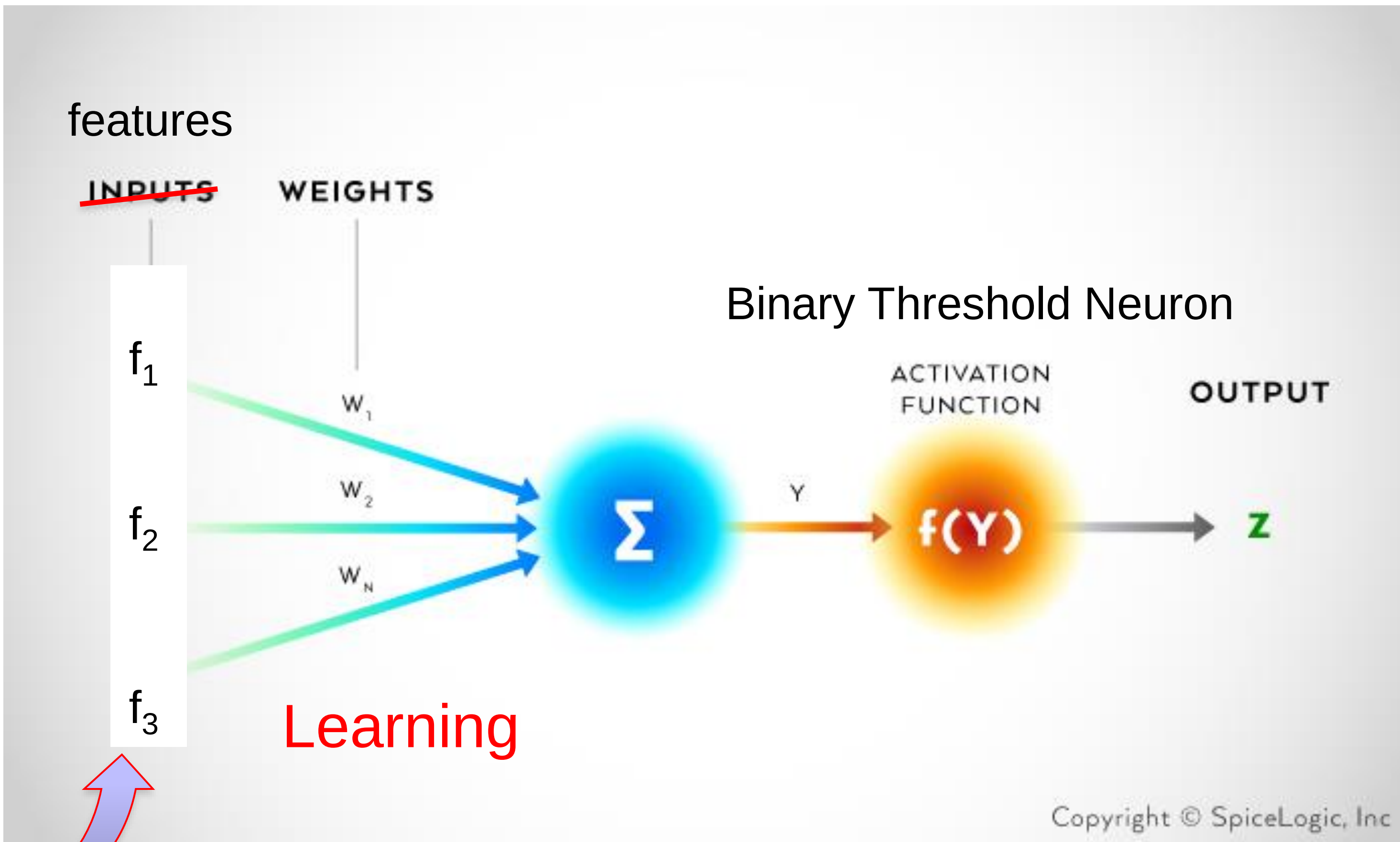
features

Inputs

x_1 f_1

x_2 f_2

x_3 f_3



Binary Threshold Neuron

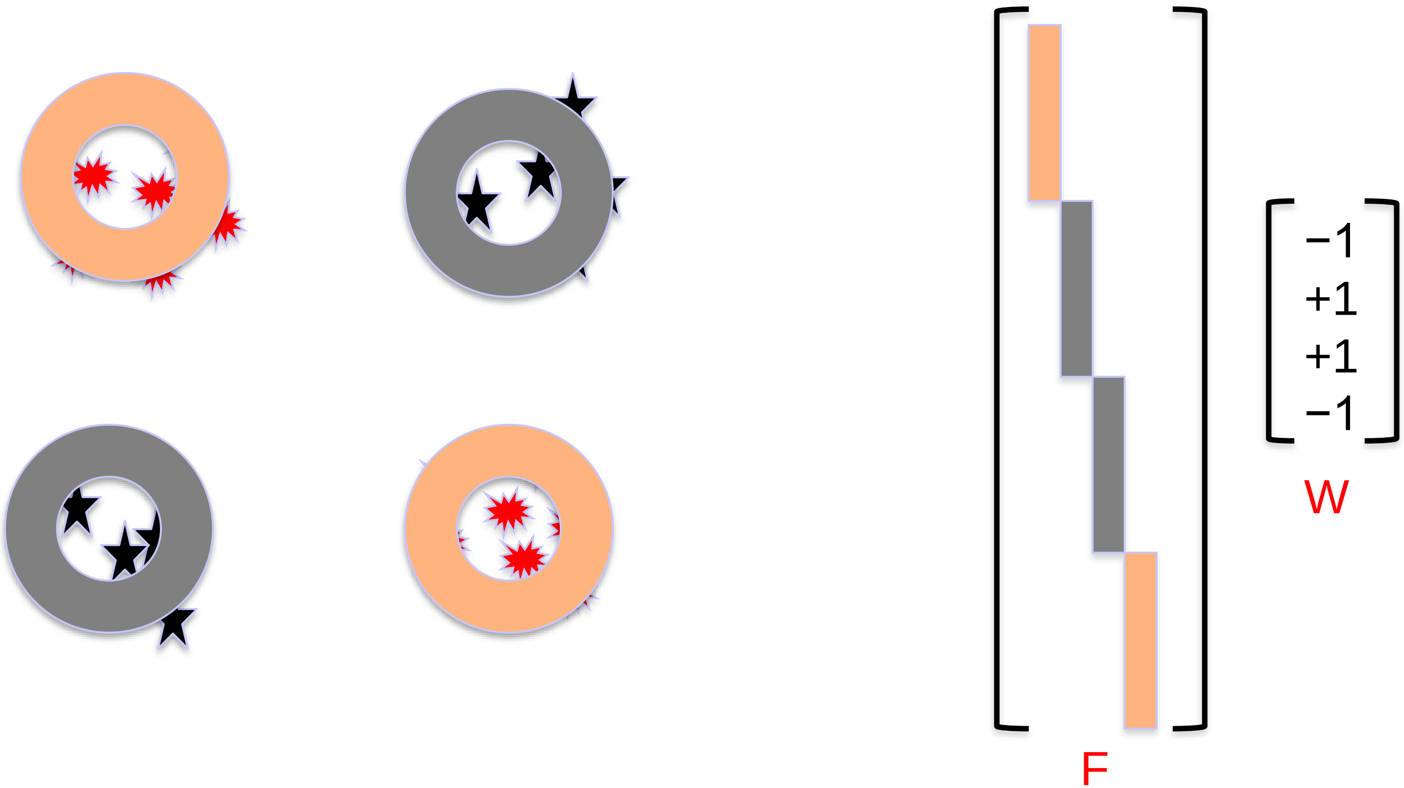
Learning

By hand!

As long as you pick right features, this can learn almost anything.

This actually gives a powerful machine learning paradigm:

- Pick right features by clustering
- Linearly separate the features.
- This is essentially what Rosenblatt initially claimed for perceptron. Chomsky & Papert actually attacked a different target.



Binary threshold neuron

- McCulloch-Pitts (1943)

There are two ways of describing the binary threshold neuron:
1. Threshold = 0
2. Threshold ≠ 0

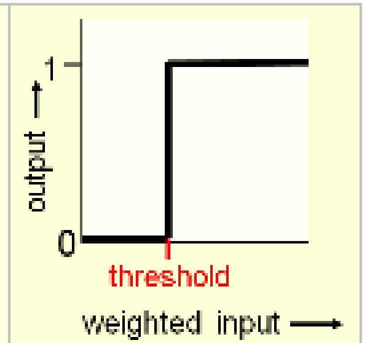
$$z = \sum_i x_i w_i$$

$$y = \begin{cases} 1 \text{ if } z \geq \theta \\ 0 \text{ otherwise} \end{cases}$$

$$\boxed{\theta = -b}$$

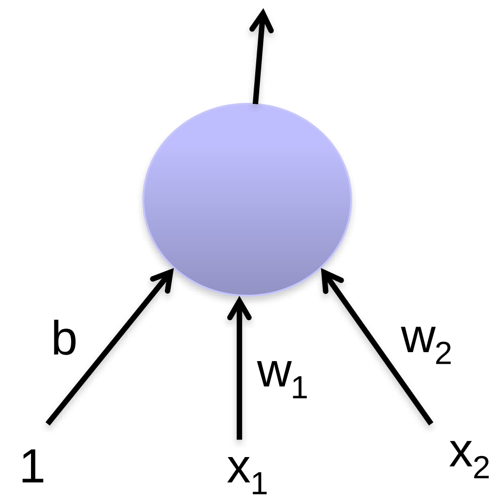
$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ otherwise} \end{cases}$$



Avoiding learning biases separately

- By a trick of adding 1 to input.
- We now can learn a bias as if it were a weight.
- Hence we get rid of the threshold.



A converging perceptron learning alg.

- If the output unit is correct, leave its weights unchanged.
- If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
- If the output unit incorrectly outputs a 1, subtract the input vector from the weight vector.

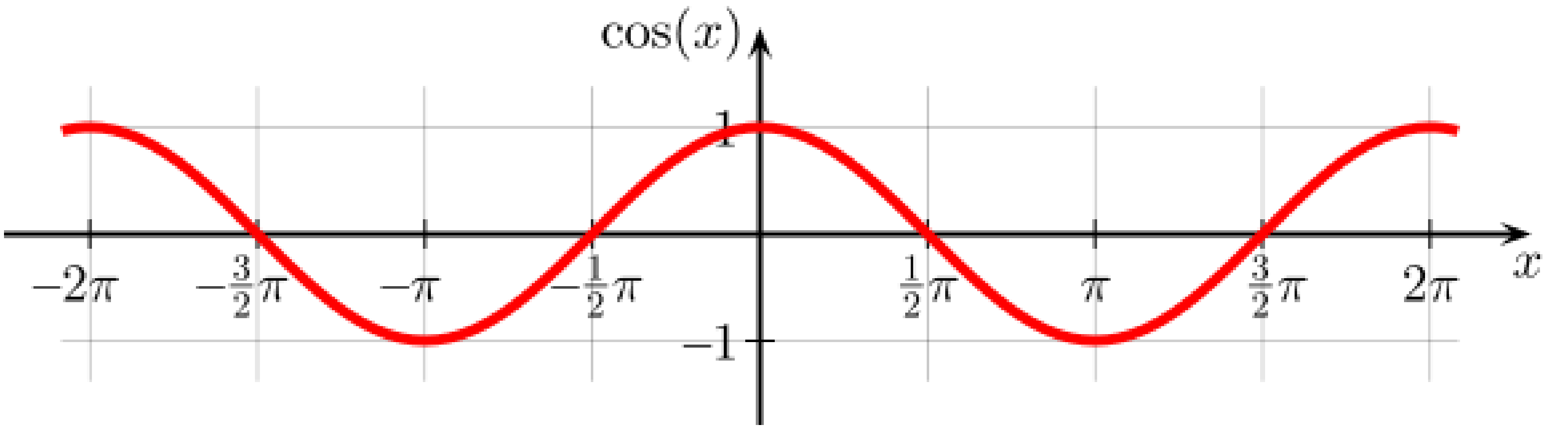
This is guaranteed to find a set of weights that is correct for all training cases if such “solution” exists.

Weight space

- The dimension k is number of the weights $w=(w_1, \dots, w_k)$.
- A point in the space represents a weight vector $(w_1, \dots, w_k)$ as its coordinates .
- Each training case is represented as a hyper-plane through the origin (assuming we move the threshold to the bias weight)
  - The weights must lie on one side of this hyper-plane to get answer correct.

Remember dot product facts:

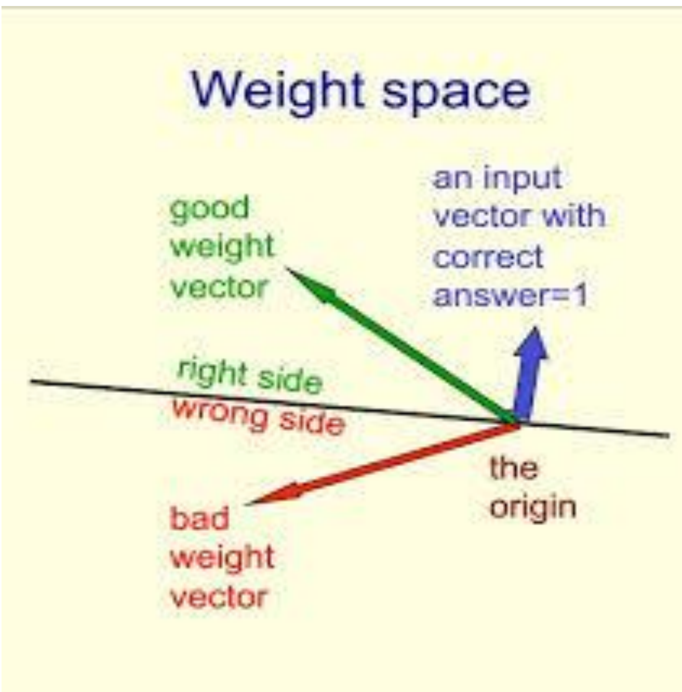
$a \bullet b = ||a||\,||b||\cos(\theta_{ab})$
$= a_1b_1+a_2b_2+ \ldots +a_nb_n$

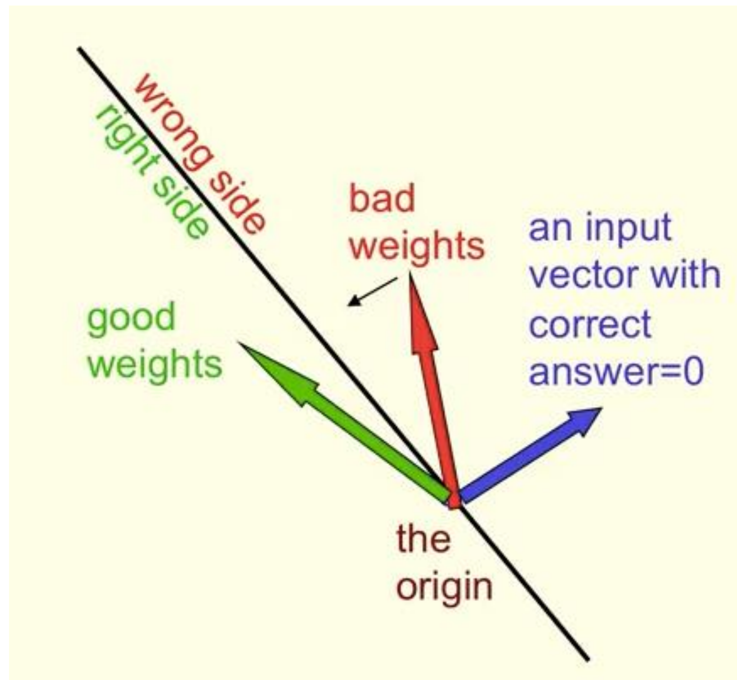


Thus, $a \bullet b \geq 0$, if $-\pi/2 \leq \theta_{ab} \leq \pi/2$
$a \bullet b \leq 0$, if $-\pi \leq \theta_{ab} \leq -\pi/2$ or $\pi/2 \leq \theta_{ab} \leq \pi$

Weight space

A point in the space represents a weight vector
Training case is a hyper-plane through the origin, assuming threshold represented by bias.



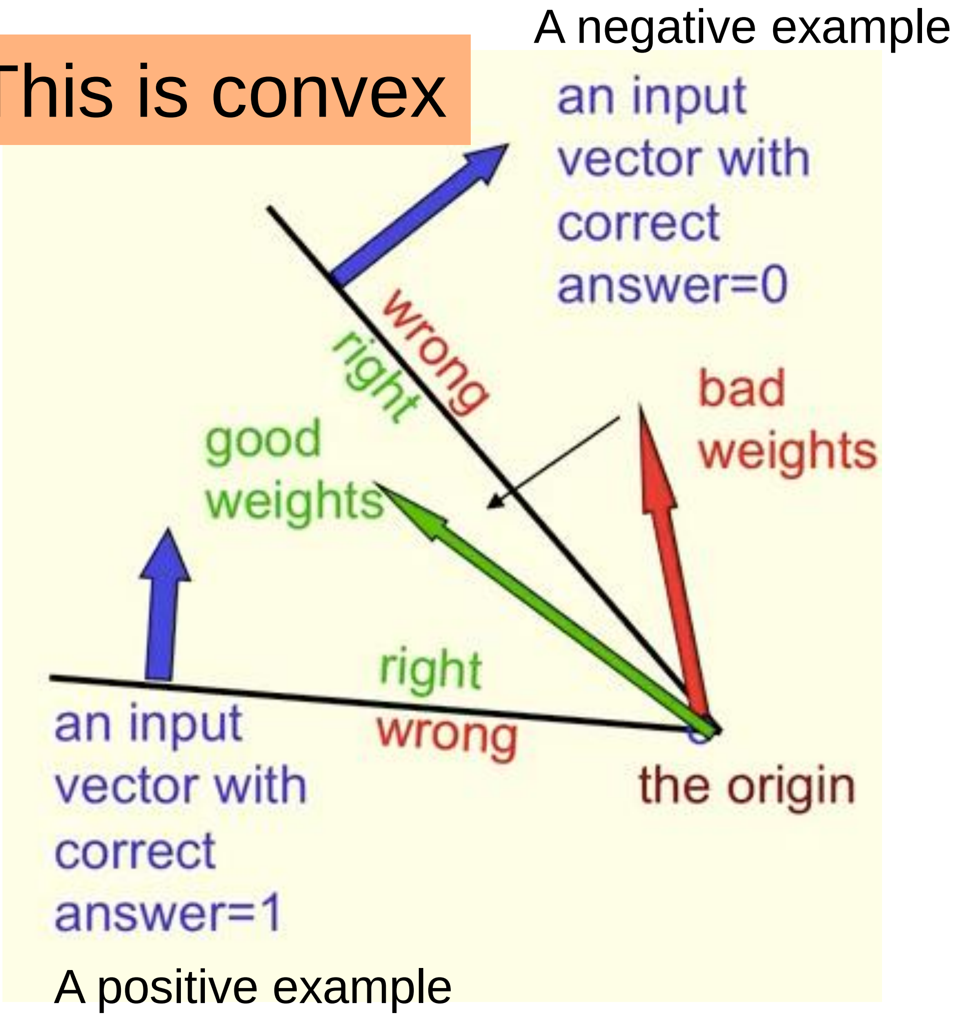


The cone of feasible solutions

To get all training cases right, we need to find a point on the “right side” of all planes (representing training cases).

The solution region, if exists, is a cone and is convex.

This is convex



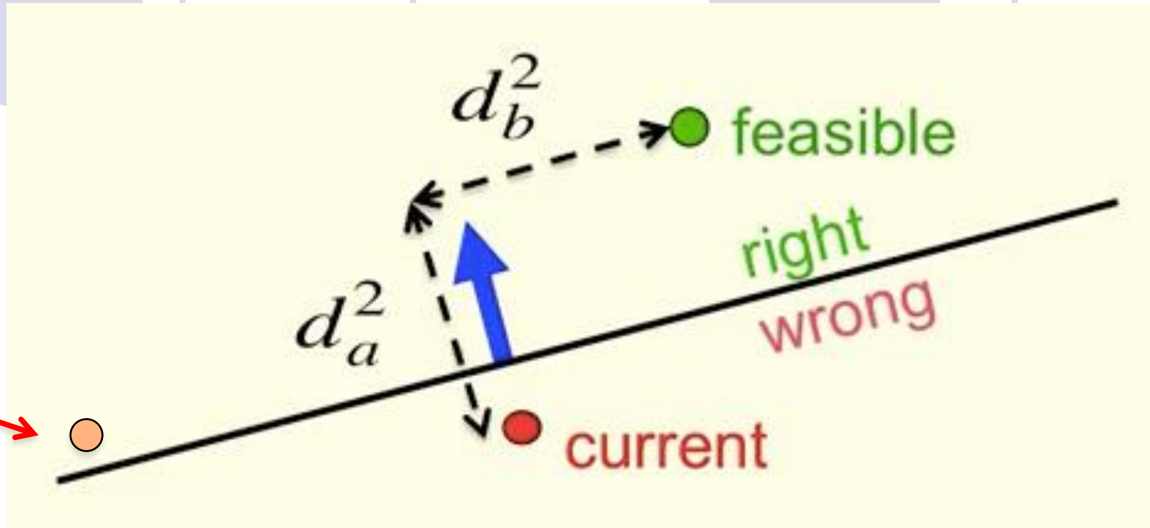
A converging perceptron learning alg.

- If the output unit is correct, leave its weights unchanged.
- If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
- If the output unit incorrectly outputs a 1, subtract the input vector from the weight vector.

This is guaranteed to find a set of weights that is correct for all training cases if such solution exists.

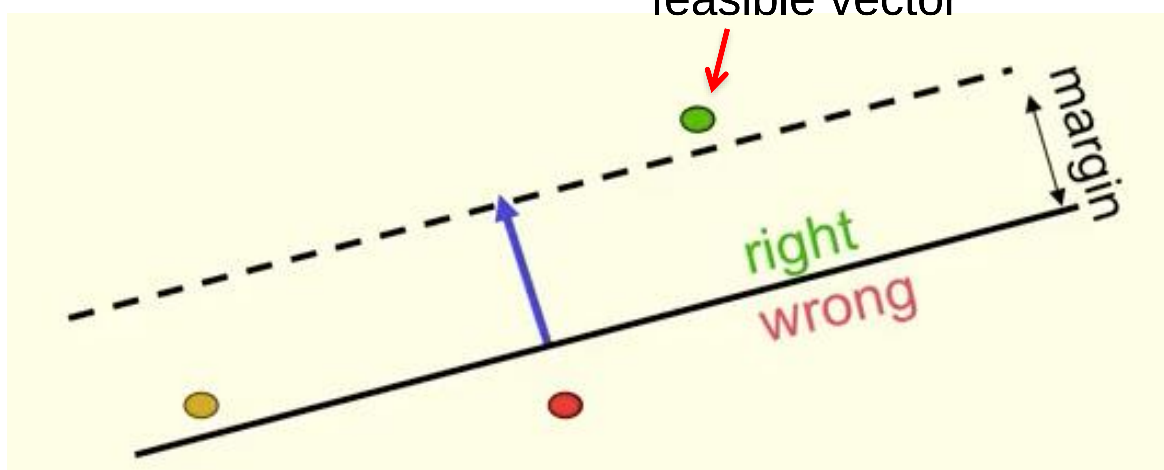
Proof of convergence by picture

But what about this point?
We might move farther.



Proof: If there is a generously feasible vector, then each step we move closer to the feasible region. After finitely many steps, the weight vector is in the feasible region.

Generously feasible vector



Note: this is assuming generously feasible vector exists.

The limitations of Perceptrons

- If we are allowed to choose features by hand, then we can do anything. But this is not learning.
- If we do not hand-pick features, then Minsky and Papert showed that perceptrons cannot do much. We will look at these proofs.

XOR cannot be learnt by a perceptron

- We prove that binary threshold output unit cannot do exclusive-or:

  Positive examples: (1,1) → 1; (0,0) →1

  Negative examples: (1,0)→0; (0,1) → 0
- The 4 input-output pairs give 4 inequalities, T being threshold:

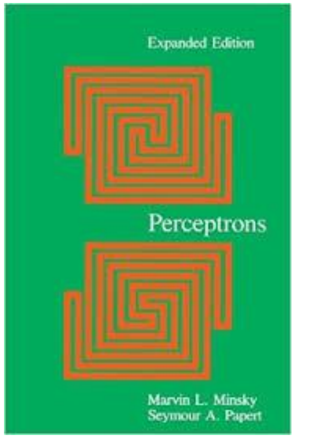
  $w_1 + w_2 \geq T, \quad 0 \geq T \quad \Rightarrow w_1 + w_2 \geq 2T$

  $w_1 < T, \quad w_2 < T \quad \Rightarrow w_1 + w_2 < 2T$

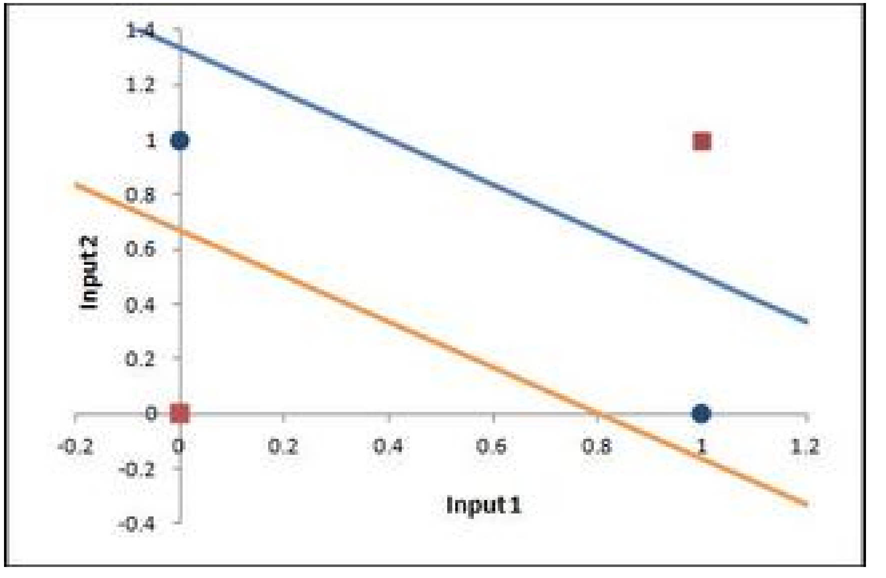
  Contradiction. QED

Geometric view

Chapter 0
of this book



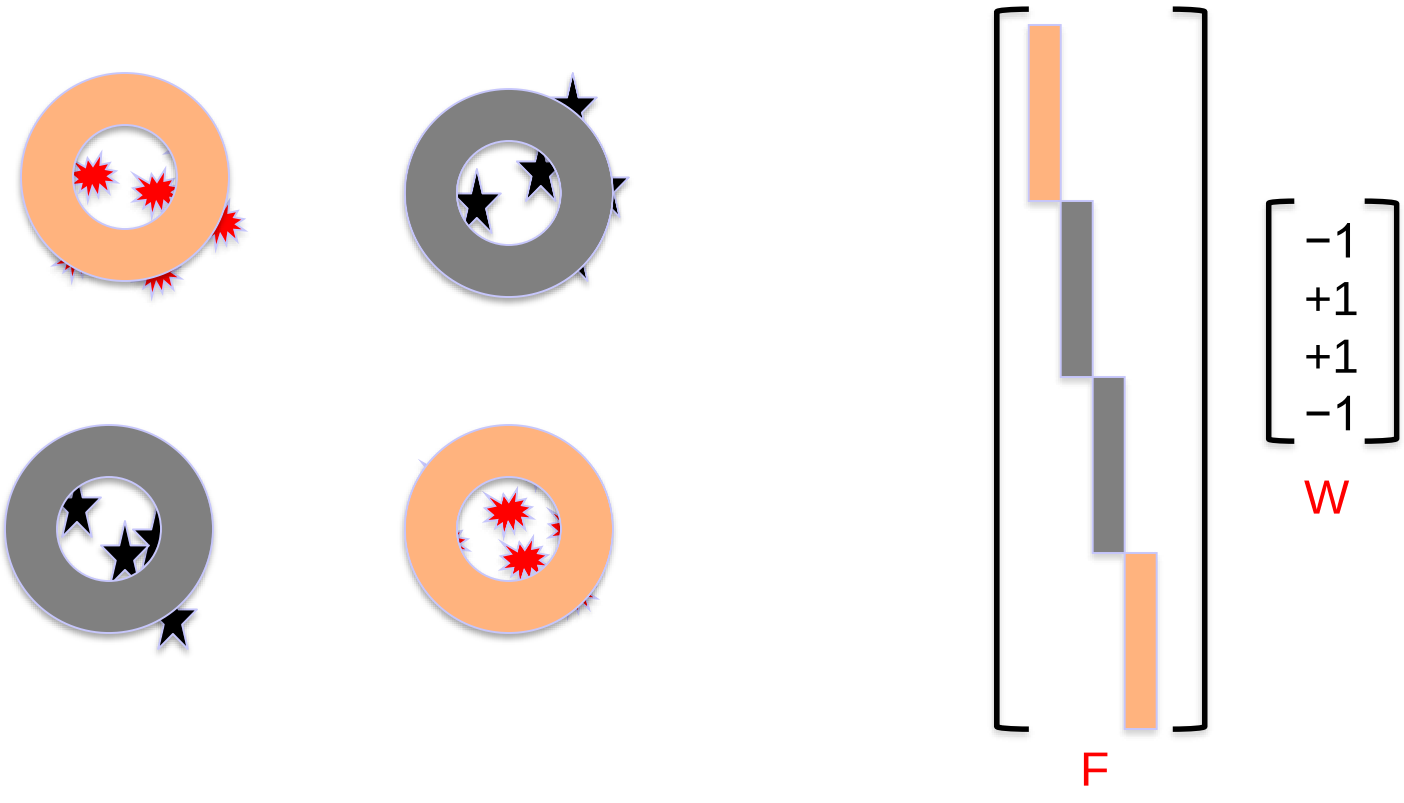
- Data-space view
  - Each input is point
  - A weight vector defines a hyperplane
  - The weight plane is perpendicular to the weight vector and misses the origin by a distance equal to the threshold



Blue dots and red dots are not linearly separable.

But this can be easily solved:

- Just pick right features (clusters)
- Then linearly separate the features, solves all.
- This is essentially what Rosenblatt initially claimed for perceptron. Chomsky & Papert actually attacked a different target.



Group Invariance Theorem (Minsky-Papert):

Perceptron cannot distinguish following two patterns under translation.

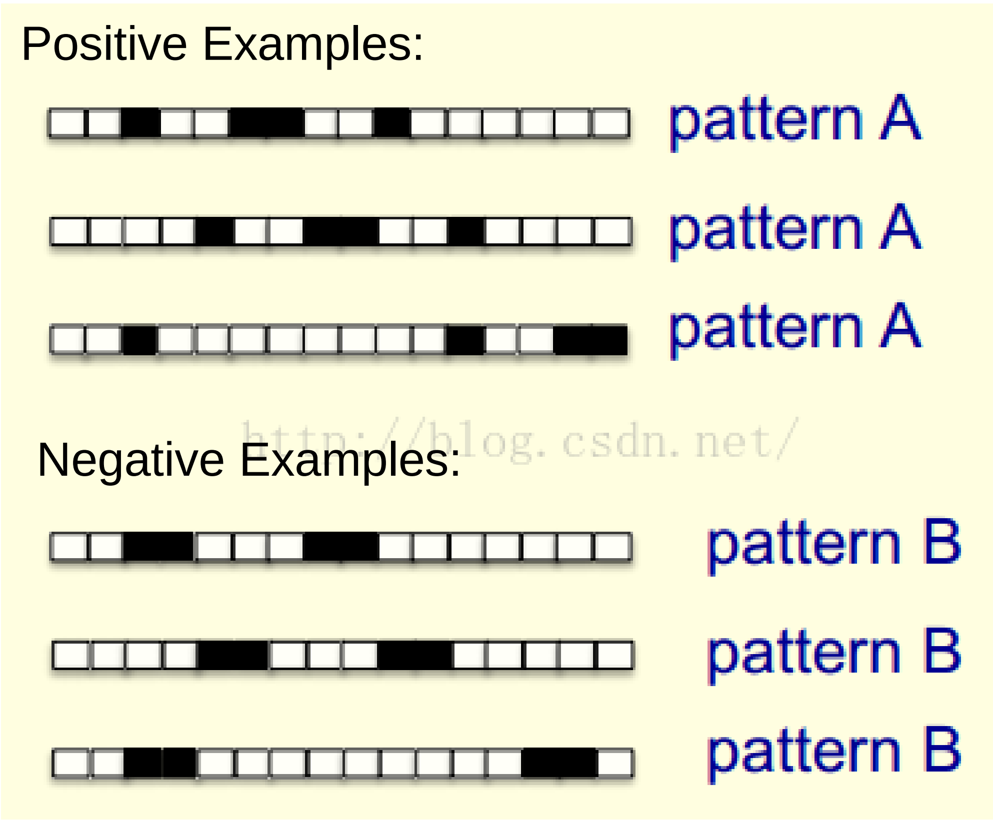
Proof.
Each pixel is activated by 4 different translations of both Pattern A and B.

Hence the total input received by the decision unit over all these patterns is four times the sum of all weights for both patterns A and B.

No threshold can always accept A & reject B. QED.

In general Perceptrons cannot do groups. Image translation forms a group. This was sometimes mis-interpreted as NN's are no good.

Hidden units can learn such features.
But deeper NN are hard to train.



Translation with wrap-around of two patterns

Basic Neurons

- To model neurons, we must idealize them:
  - Idealization removes complicated details that are not essential for understanding the main principles.
  - It allows us to apply mathematics and to make analogies to other familiar systems
  - Once we understand the basic principles, its easy to add complexity to make the model more faithful.

Linear neurons

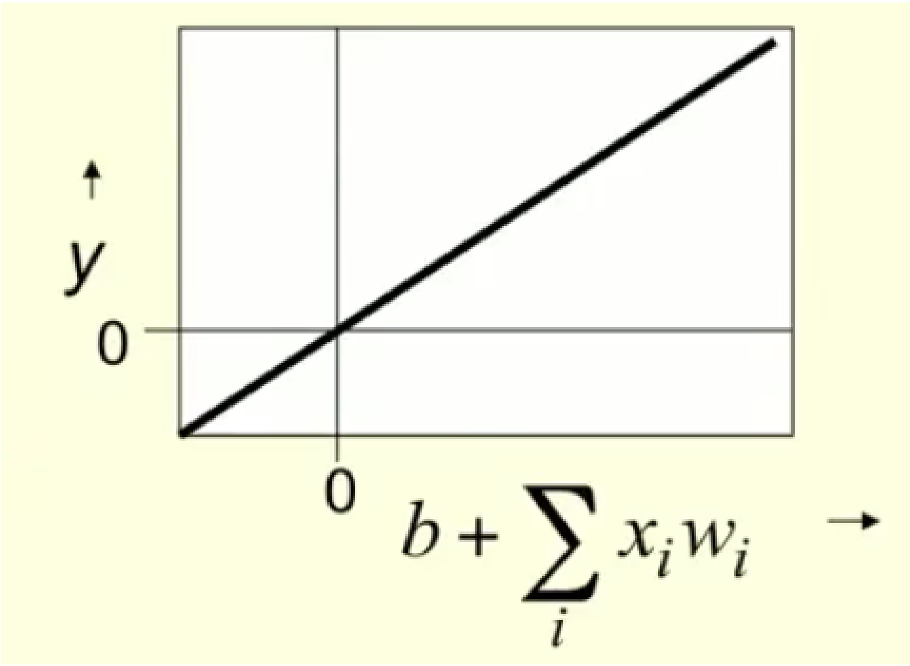
- These are the basic building parts for all other neuron networks.

bias　　i-th input

$$y = b + \Sigma_i x_i w_i$$

output

Weight on i-th input



Binary threshold neuron

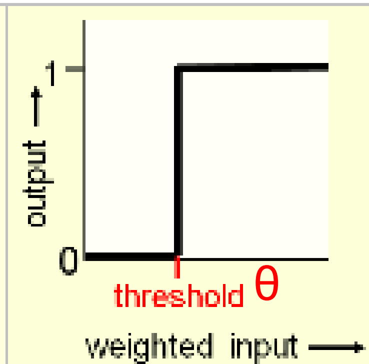
- McCulloch-Pitts (1943)
  - First compute a weighted sum of inputs
  - Then send out a fixed size spike of activity if the weighted sum exceeds a threshold.
  - McCulloch and Pitts thought that each spike is like the truth value of a proposition and each neuron combines truth values to compute the truth value of another proposition.
  - This has influenced Von Neumann.

There are two equivalent ways to describe a binary threshold neuron

$$z = \sum_i x_i w_i \qquad y = \begin{cases} 1 \text{ if } z \geq \theta \\ 0 \text{ otherwise} \end{cases}$$

$$\boxed{\theta = -b}$$

$$z = b + \sum_i x_i w_i \qquad y = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

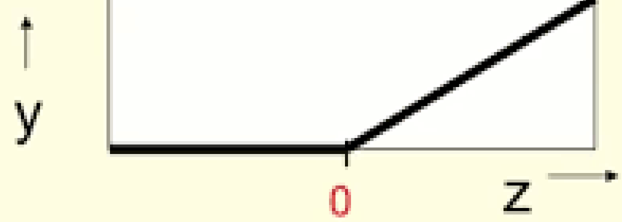


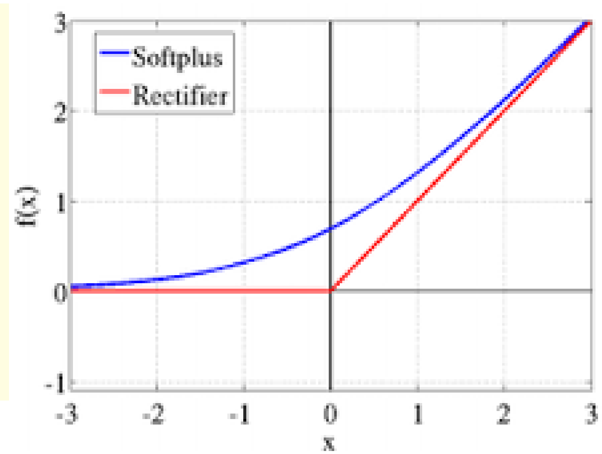
Rectified Linear Unit (ReLU)

- They compute a linear weighted sum of their inputs.
- The output is a non-linear function of the total input.
- This is the most popularly used neuron.

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$





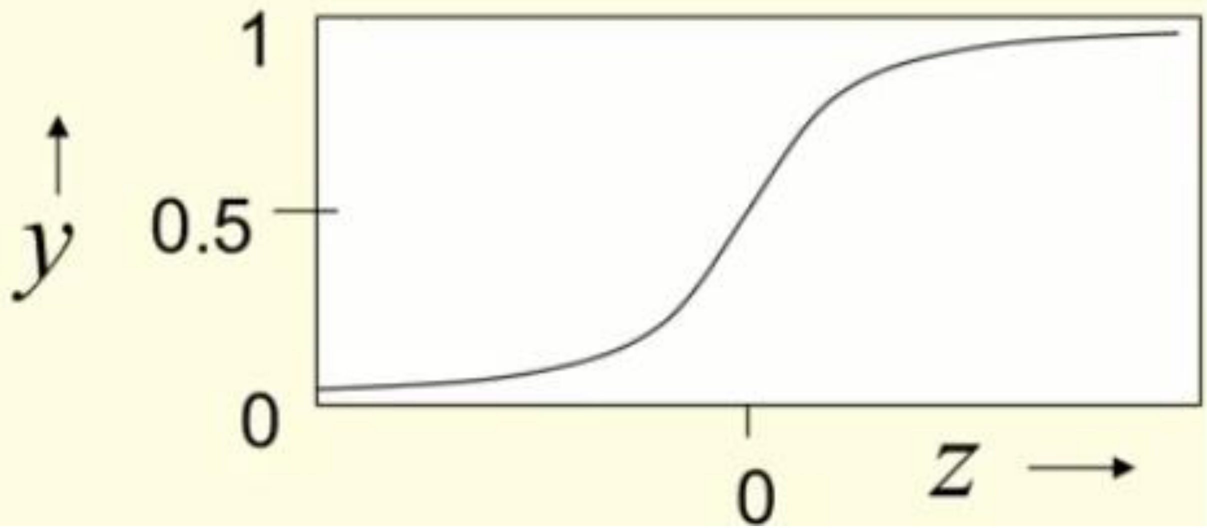
Or written as: f(x) = max {0,x}

A smooth approximation of the ReLU is "softplus" function

$f(x) = \ln(1+e^x)$

Sigmoid neurons

$$z = b + \sum_i x_i w_i \qquad y = \frac{1}{1 + e^{-z}}$$



Typically they use the logistic function

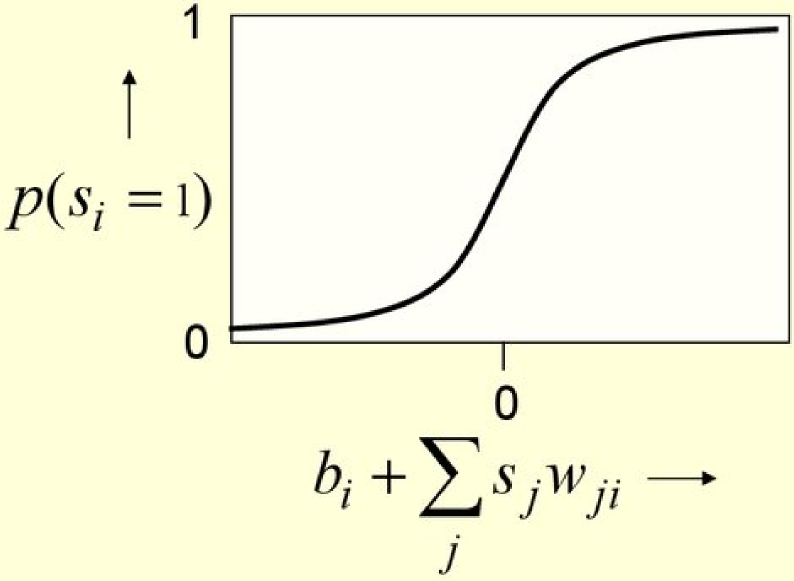
They have nice derivatives which makes learning easy.

But they cause vanishing gradients during backpropogation.

Stochastic binary neurons

(Bernoulli variables)

- These have a state of 1 or 0.
- The probability of turning on is determined by the weighted input from other units (plus a bias)



$$p(s_i = 1) \quad = \quad \frac{1}{1 + \exp(-b_i - \sum s_j w_{ji})}$$

Softmax function
(Normalized exponential function)

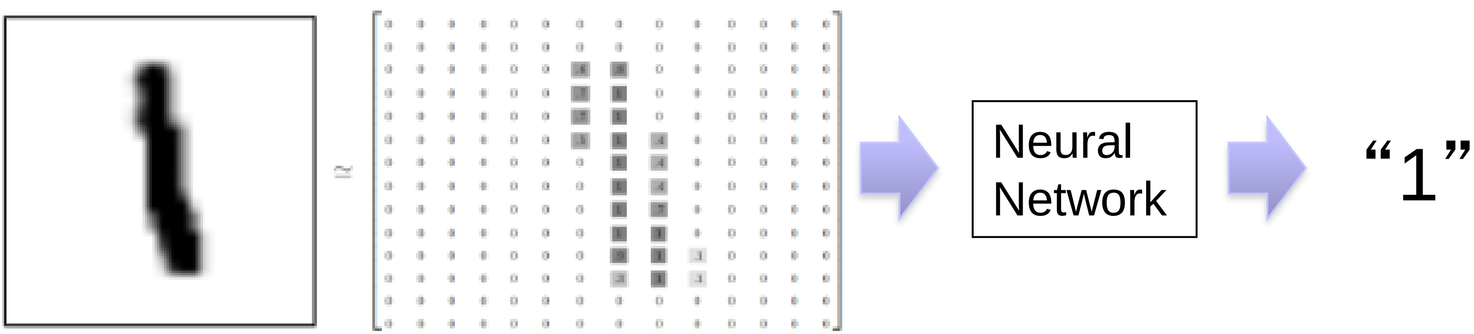
$$\boxed{\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}}$$

If we take an input of [1,2,3,4,1,2,3], the softmax of that is [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175].
The softmax function highlights the largest values and suppress other values.

Comparing to “max” function, softmax is differentiable.

Lecture 3. Fully Connected NN & Hello World of Deep Learning

0–9 handwritten digit recognition:



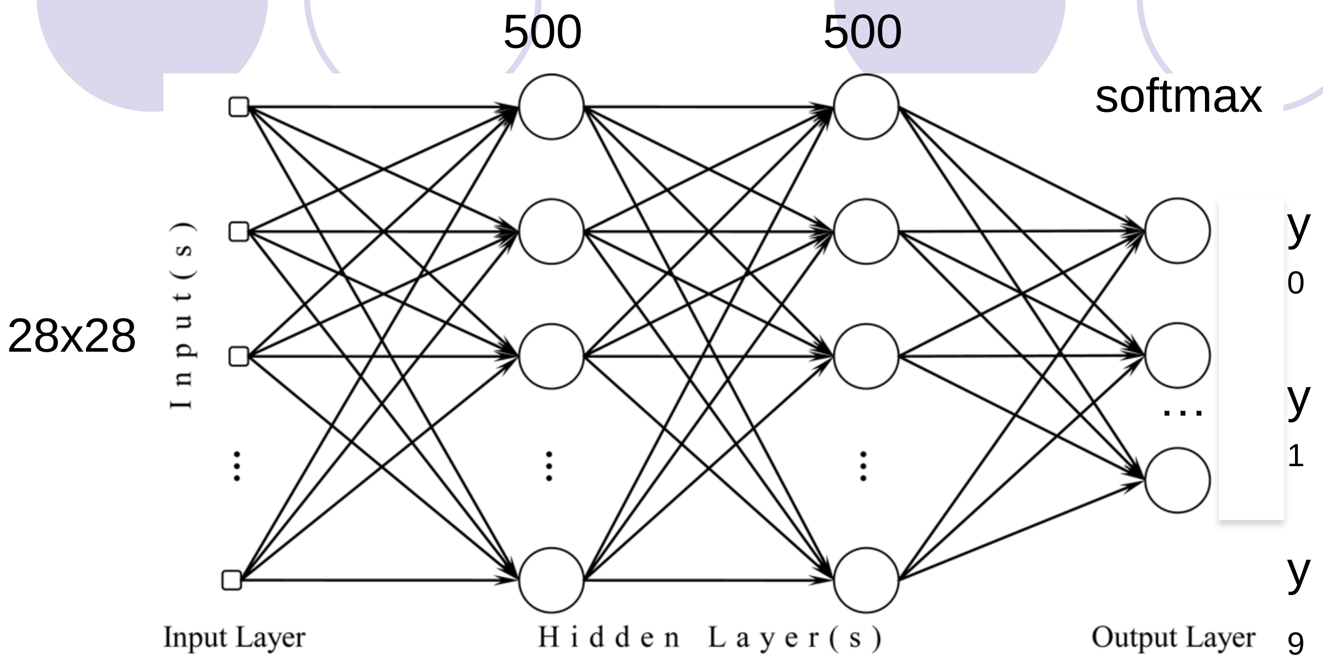
28 x 28

MNIST Data maintained by Yann LeCun: http://yann.lecun.com/exdb/mnist/
Keras provides data sets loading function at http://keras.io/datasets

Keras & Tensorflow

- Interface of Tensorflow and Theano.
- Francois Chollet, author of Keras is at Google, Keras will become Tensorflow API.
- Documentation: http://keras.io.
- Examples: https://github.com/fchollet/keras/tree/master/examples
- Simple course on Tensorflow: https://docs.google.com/presentation/d/1zkmVGobdPfQgsjIw6gUqJsjB8wvv9uBdT7ZHdaCjZ7Q/edit#slide=id.p

Implementing in Keras



Fully connected NN

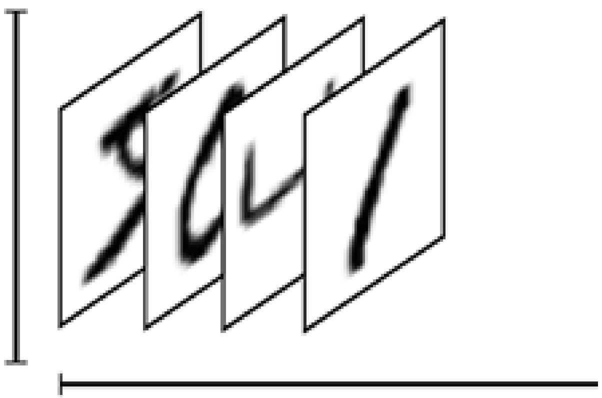
```
model = sequential()   # layers are sequentially added
model.add( Dense(input_dim=28*28, output_dim=500))
model.add(Activation( 'sigmoid' ))  #: softplus, softsign,relu,tanh, hard_sigmoid
model.add(Dense( output_dim = 500))
model.add (Activation( 'sigmoid' ))
Model.add(Dense(output_dim=10))
Model.add(Activation('softmax'))
model.compile(loss= 'categorical_crossentropy' , optimizer= 'adam' , metrics=[ 'accuracy' ])
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

Training

model.fit(x_train, y_train, batch_size=100, nb_epoch=20)

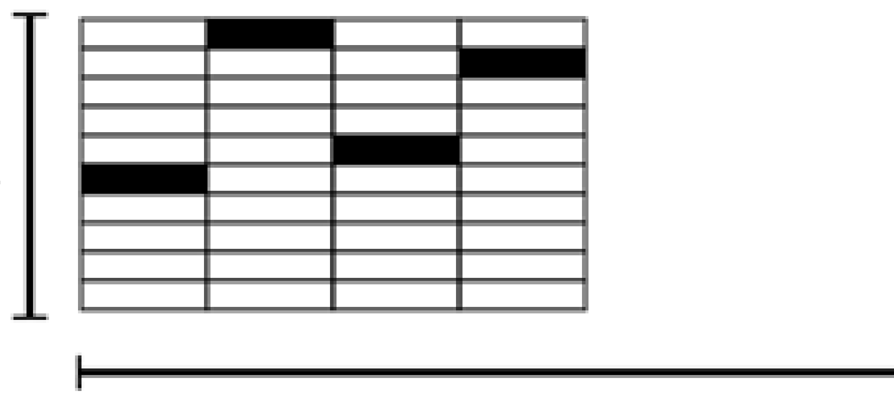
numpy array

28 x 28
=784



Number of training examples

10

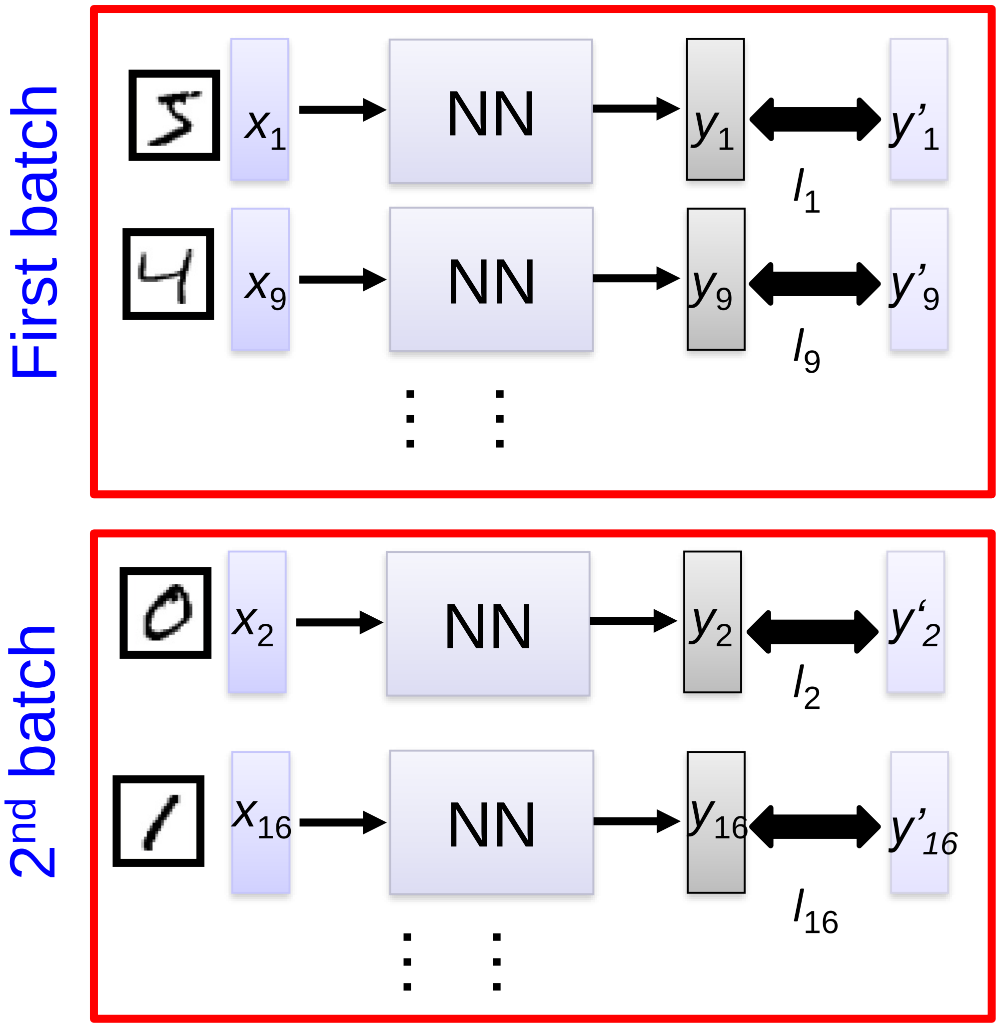


Number of training examples

We do not really minimize total loss!

model.fit(x_train, y_train, batch_size=100, nb_epoch=20)

Batch: parallel processing



- Randomly initialize network parameters
- Pick the 1st batch

  $L' = l_1 + l_9 + \ldots$

  Update parameters
- Pick the 2nd batch

  $L'' = l_2 + l_{16} + \ldots$

  Update parameters

  ⋮
- Until all batches have been picked

one epoch

Repeat the above process

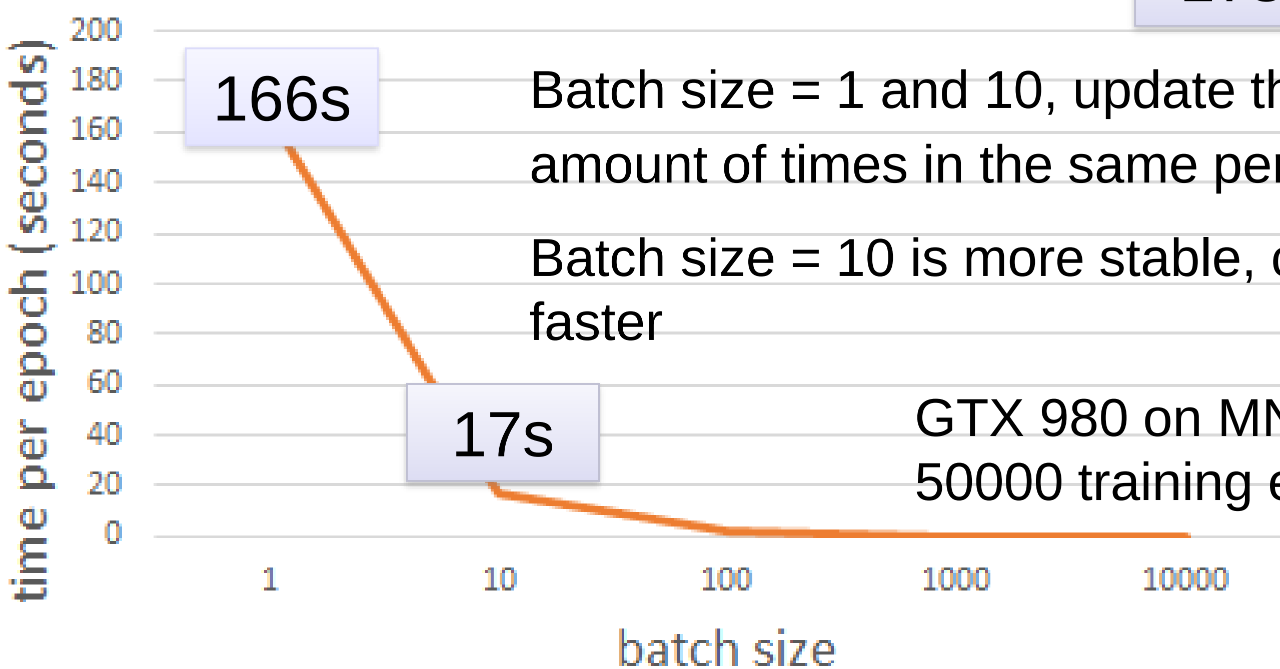
Speed

Very large batch size can yield worse performance

- Smaller batch size means more updates in one epoch
  - E.g. 50000 examples
  - batch size = 1, 50000 updates in one epoch
  - batch size = 10, 5000 updates in one epoch

166s 1 epoch

17s 10 epochs



Batch size = 1 and 10, update the same amount of times in the same period.

Batch size = 10 is more stable, converge faster

GTX 980 on MNIST with 50000 training examples

Speed - Matrix Operation

- Why is batching faster?

One at a time:

Batch by GPU, 2 at a time, 1/2 time cost

Matrix operation