Scientific programming

Neural networks

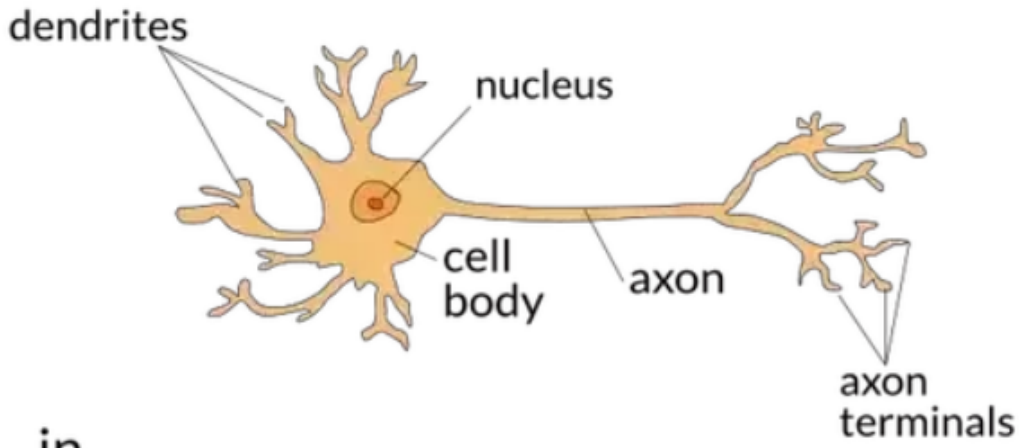
Janos Török

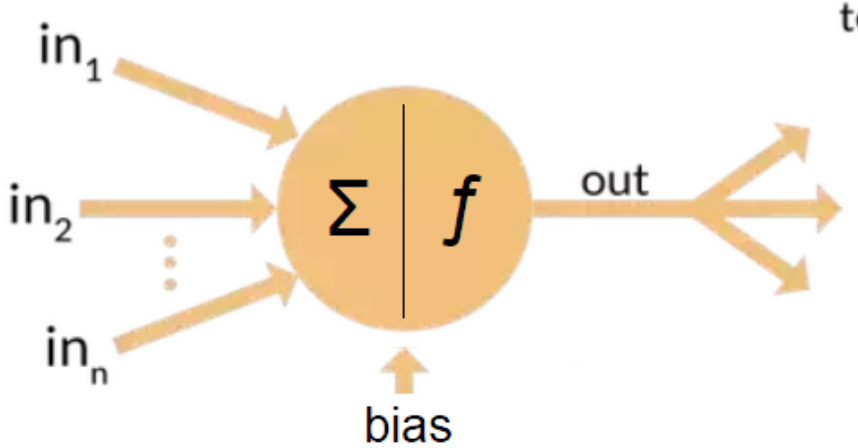
Department of Theoretical Physics

May 31, 2023

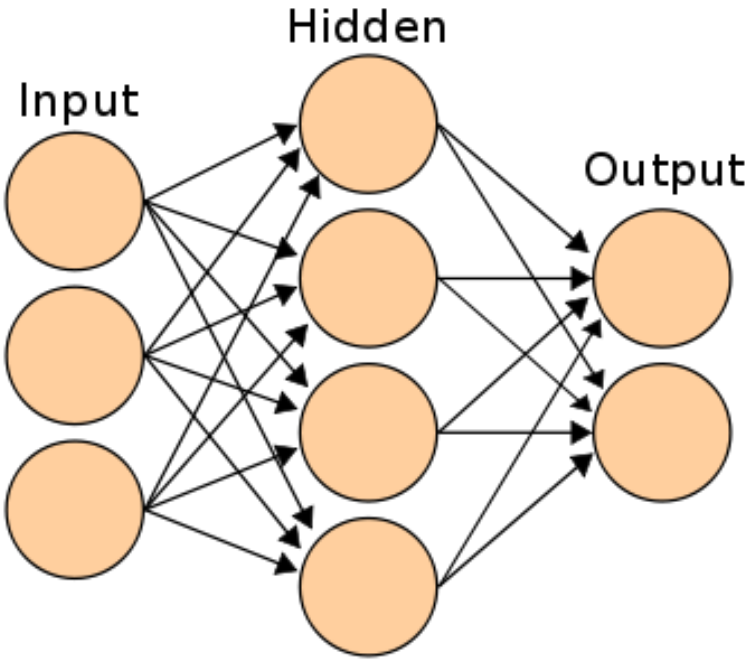
Artificial neuron

- Biological neuron:
  - Stimulus in dendrites
  - Fire (activate axon) when stimulus is large enough
- Artificial neuron:
  - Weighted sum of the input
  - Output is a non-linear function of the input



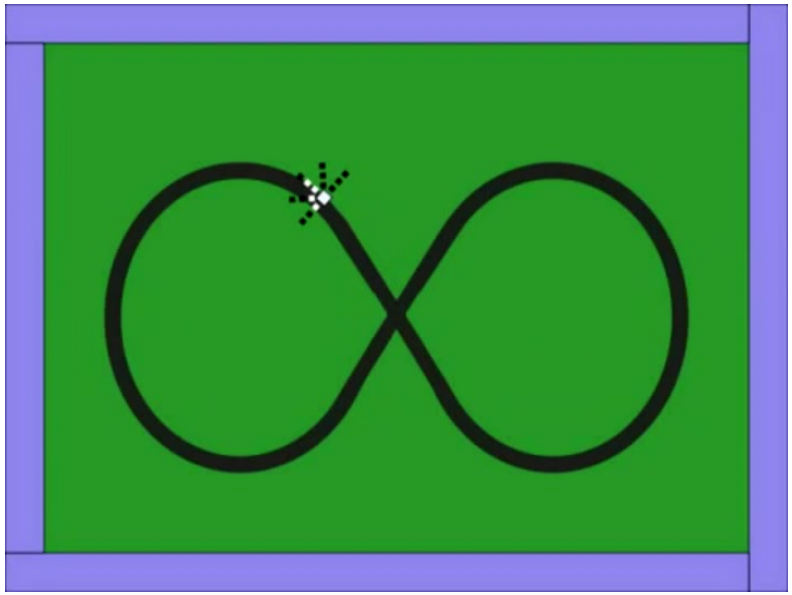


Neural networks



- Input pattern
- Output pattern
- Adaptive weights
- Approximating non-linear functions

- Pattern recognition
- Speech recognition
- Language models
- Pattern creation



Neural networks

- Input vector I
- Output vector $O(I)$
- Transition matrix W_{ij}^{l}, bias b_{i}^{l}, bias in layer l
- Learning using a cost function
- Test goodness

Neural networks: Learning

- Supervised learning
  - Data training
  - Error (continuous):

$$E = (T(I) - O(I))^2,$$

  - Cross entropy (discrete) $p \in \{y, 1-y\}$, $q \in \{\hat{y}, 1-\hat{y}\}$:

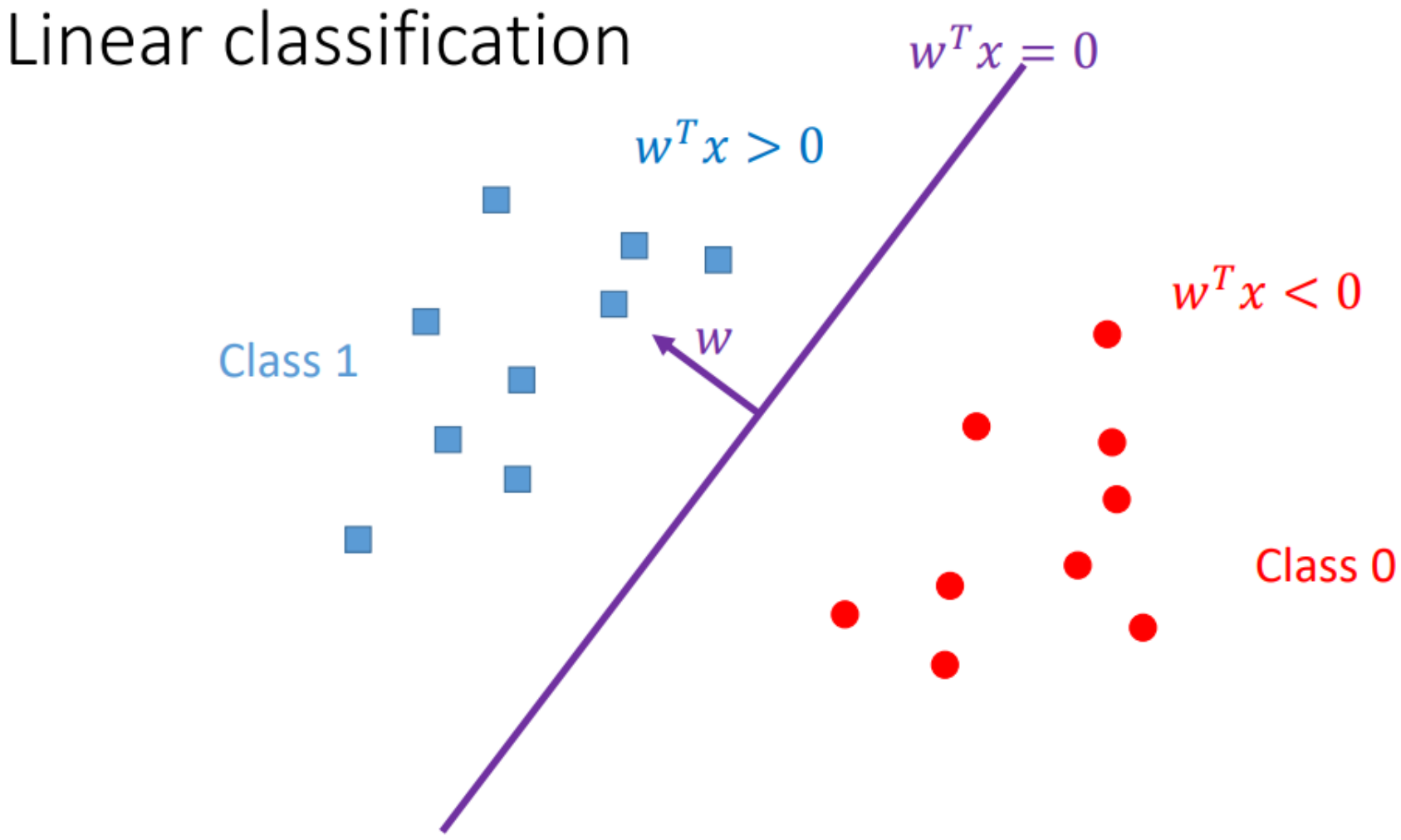
$$H(p,q) = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

    where $T(I)$ is the target vector for input I
  - Minimize E or H for available set of $\{I, I(O)\}$ pairs
  - Deep learning: many layers of neurons in the neural network
- Test goodness:
  - Use only part of $\{I, I(O)\}$ pairs for learning, the rest is for testing.

Neural networks: Learning

- Unsupervised learning: No fitness function
  - Reinforcement learning: e.g. Q-learning
    - Penalize wrong answers and reward good ones
  - Used for playing games
  - Further improve models

Deep learning: Classification, linear



Deep learning: Classification, linear

Attempt

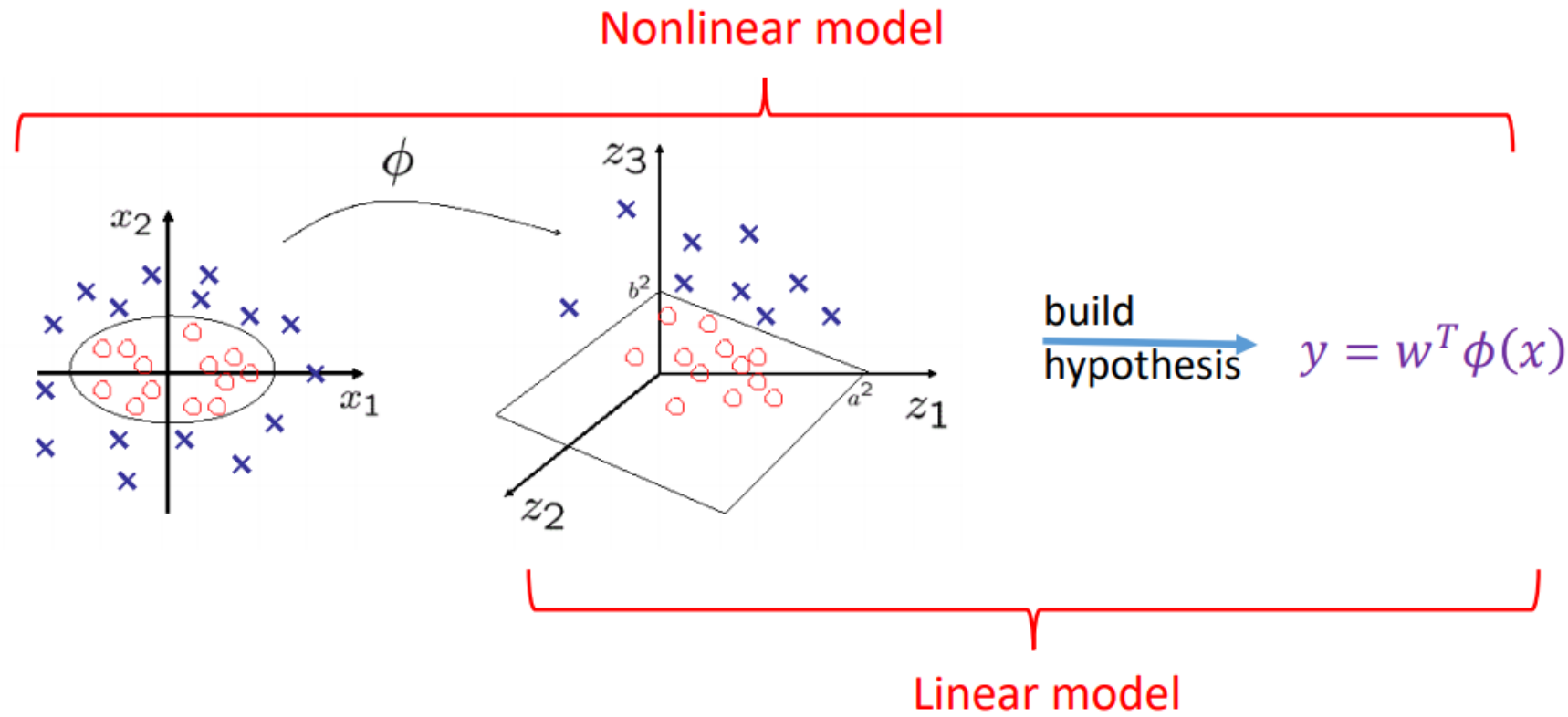
- Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
- Hypothesis $y = \text{sign}(f_w(x)) = \text{sign}(w^T x)$
  - $y = +1$ if $w^T x > 0$
  - $y = -1$ if $w^T x < 0$

- Let's assume that we can optimize to find w

Deep learning: Kernel model

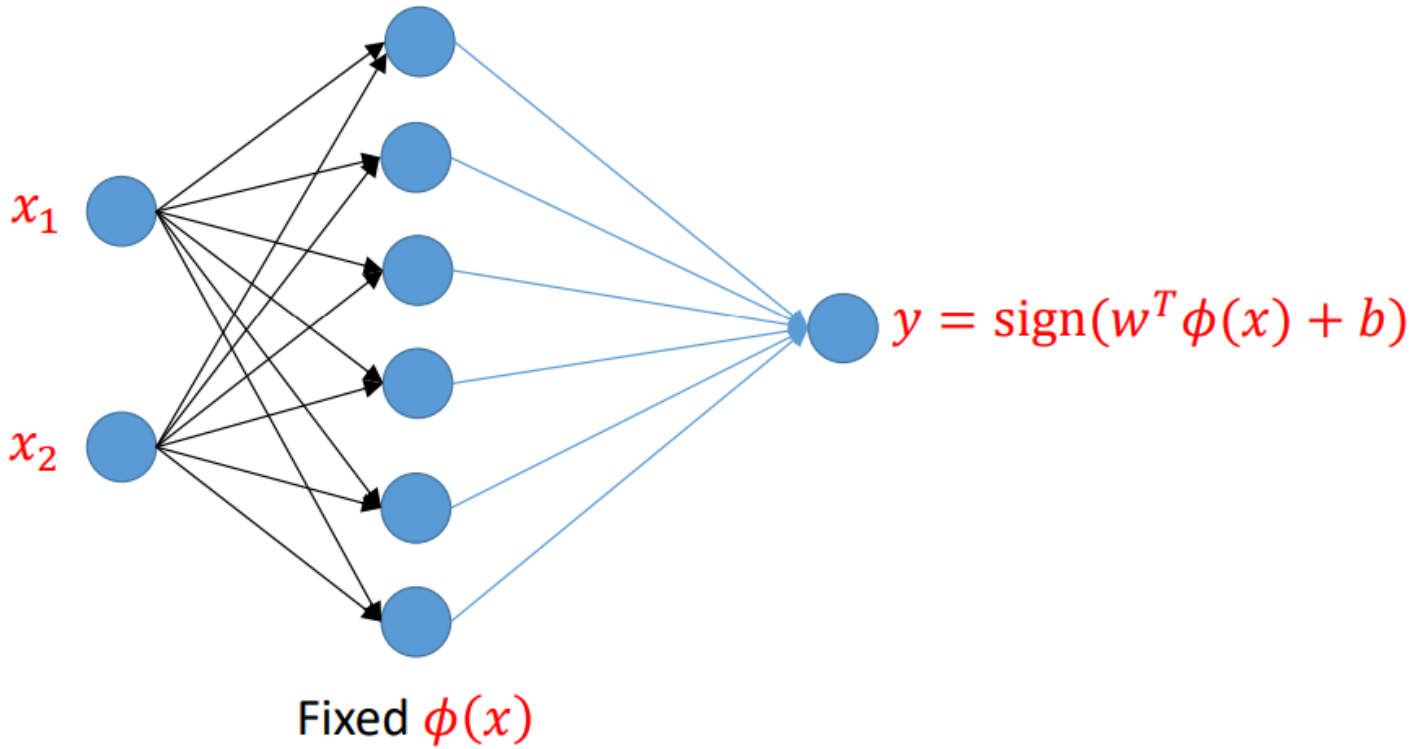
Make non-linear model linear

Features: part of the model



Deep learning: Feed forward

Example: Polynomial kernel SVM



Deep learning: Feed forward

Motivation: representation learning

- Why don't we also learn $\phi(x)$?

x

Learn $\phi(x)$

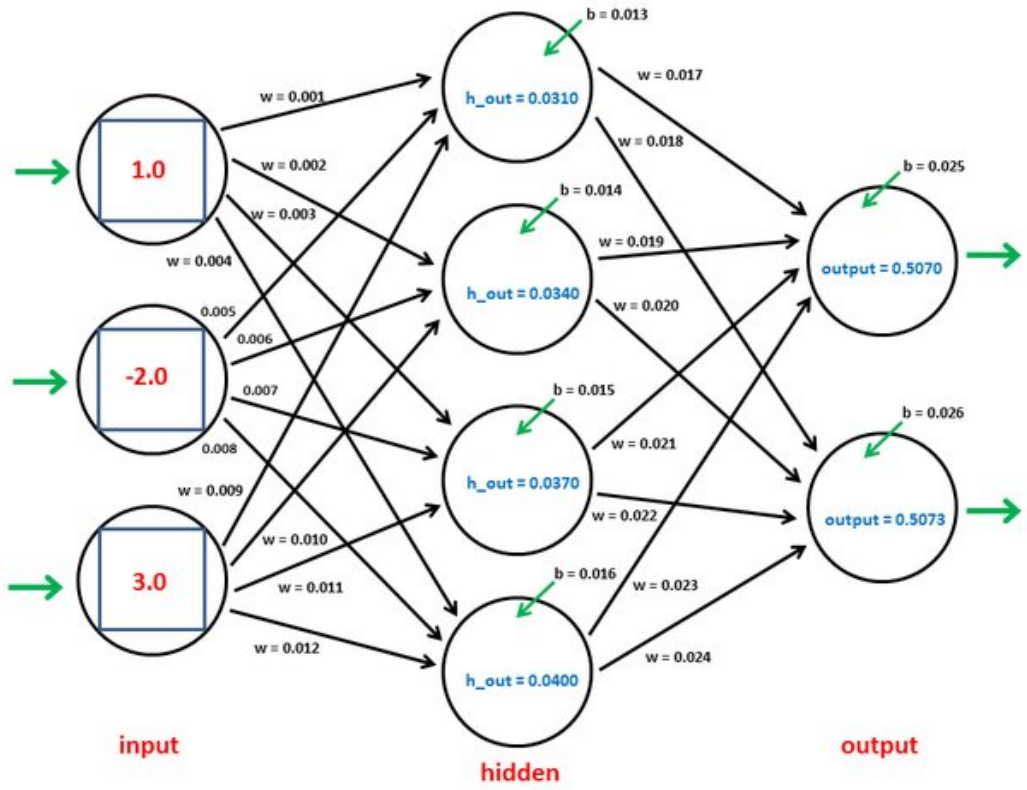
$\phi(x)$

Learn w

$y = w^T \phi(x)$

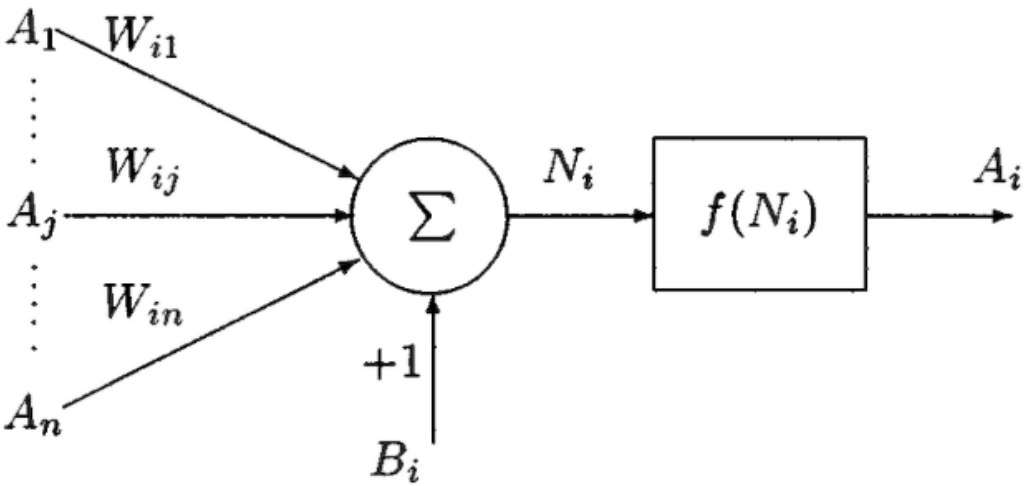
Fully connected neural networks

- Ideas from Piotr Skalski (practice), Pataki Bálint Ármin (lecture) and HMKCode (lecture)



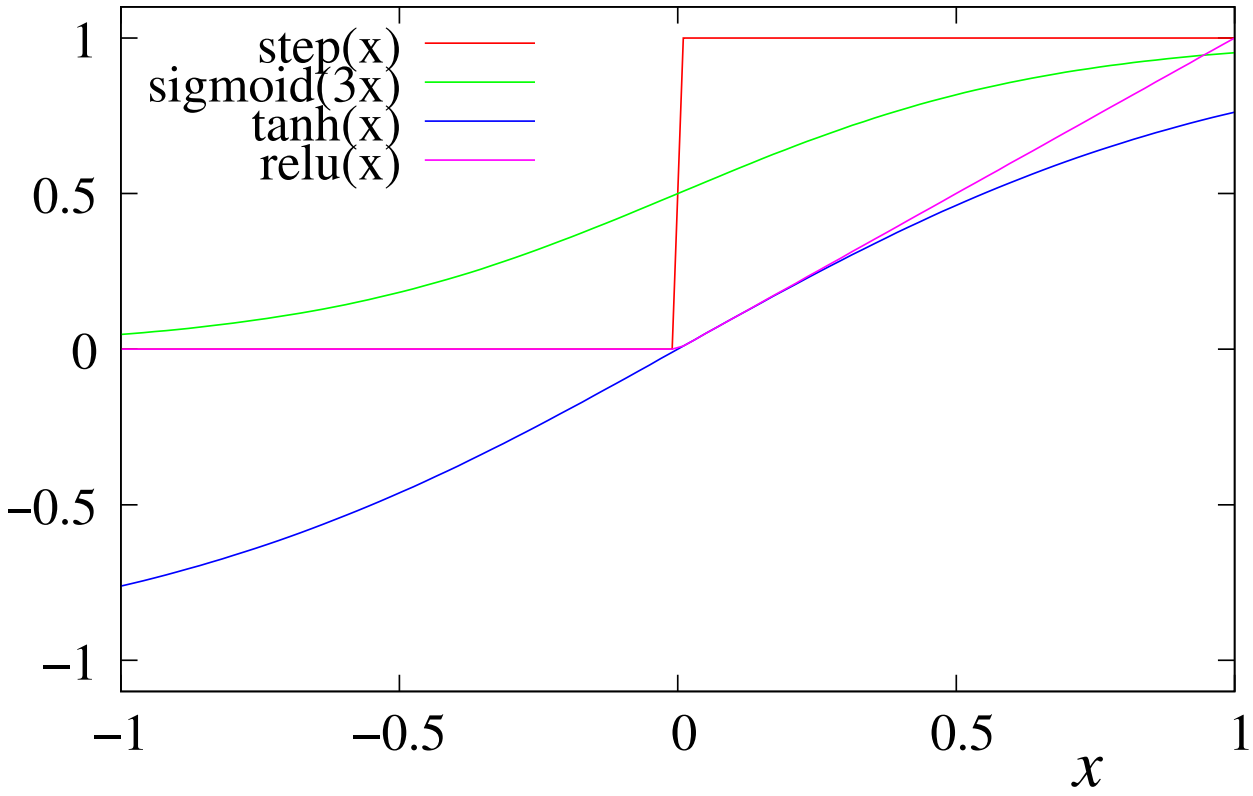
Fully connected neural networks

- Model:
  - Inputs (x_j) or for hidden layer l: A_j^{l-1}
  - Weight w_{ij}^l
  - Bias b_i^l
  - Weighted sum of input and bias: $z_i^l = \sum_j A_j^{l-1} w_{ij}^l + b_i^l$
  - Activation function (nonlinear) g: $A_i^l = g(z_i^l)$



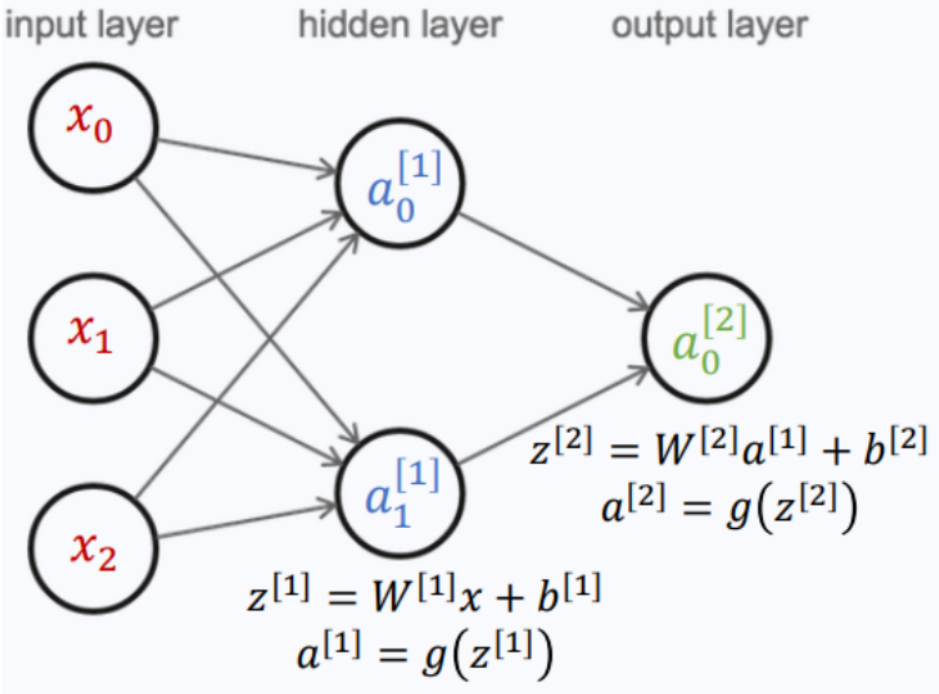
Yang et el, 2000.

Deep learning: Activation function



Feed forward

- Example



$$z_0^{[1]} = w_{0,0}^{[1]}x_0 + w_{0,1}^{[1]}x_1 + w_{0,2}^{[1]}x_2 + b_0^{[1]}$$
$$a_0^{[1]} = g\left(z_0^{[1]}\right)$$

$$z_1^{[1]} = w_{1,0}^{[1]}x_0 + w_{1,1}^{[1]}x_1 + w_{1,2}^{[1]}x_2 + b_0^{[1]}$$
$$a_1^{[1]} = g\left(z_1^{[1]}\right)$$

$$z_0^{[2]} = w_{0,0}^{[2]}a_0 + w_{0,1}^{[2]}a_1 + b_0^{[2]}$$
$$a_0^{[2]} = g\left(z_0^{[2]}\right)$$

- We have an output, how to change weights and biases to achieve the desired output?
- Error L

Backpropagation

- $$\Delta W = -\alpha \frac{\partial L}{\partial W}$$
- W is a large three dimansional matrix
- Chain rule!

Geoffrey Hinton

FOLLOW

Emeritus Prof. Comp Sci, U.Toronto & Engineering Fellow, Google
Verified email at cs.toronto.edu - Homepage

machine learning neural networks artificial intelligence cognitive science computer science

TITLE	CITED BY	YEAR
Learning internal representations by error-propagation DE Rumelhart, GE Hinton, RJ Williams Parallel Distributed Processing: Explorations in the Microstructure of …	63004 *	1986

Backpropagation

- Chain rule

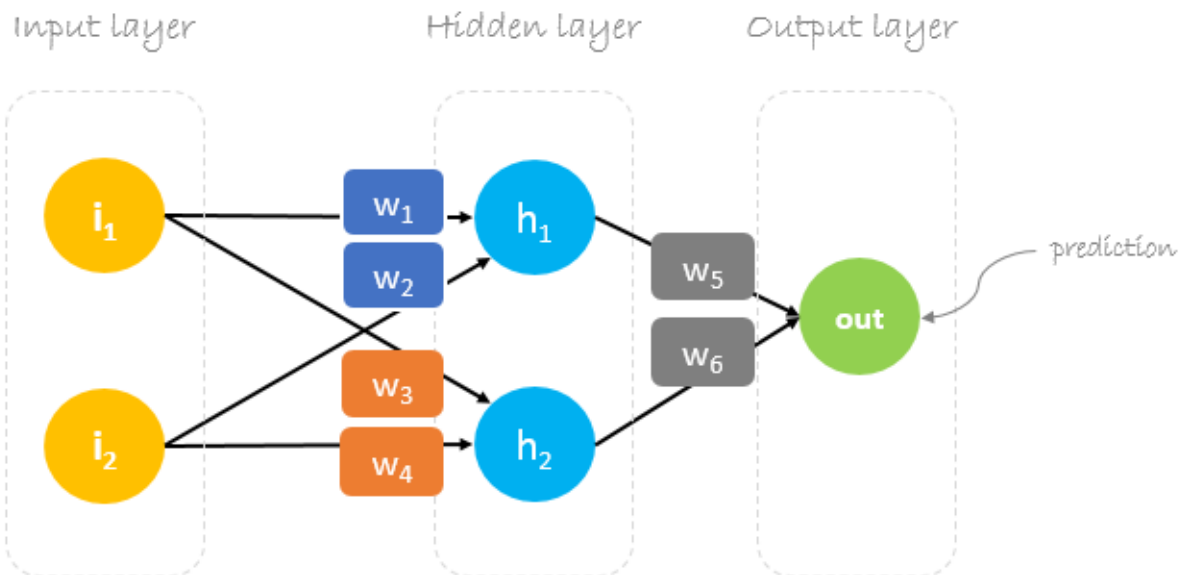
$$a_i = g(z_i) = g(w_{ij}a_j + b_i)$$

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial a_i}\frac{\partial a_i}{\partial z_i}\frac{\partial z_i}{\partial w_{ij}} = \frac{\partial L}{\partial a_i}g'(z_i)a_j$$

$$\frac{\partial L}{\partial a_i} = \sum_{l \in L}\frac{\partial L}{\partial a_l}\frac{\partial a_l}{\partial z_l}\frac{\partial z_l}{\partial a_i} = \sum_{l \in L}\frac{\partial L}{\partial a_l}g'(z_l)w_{li}$$

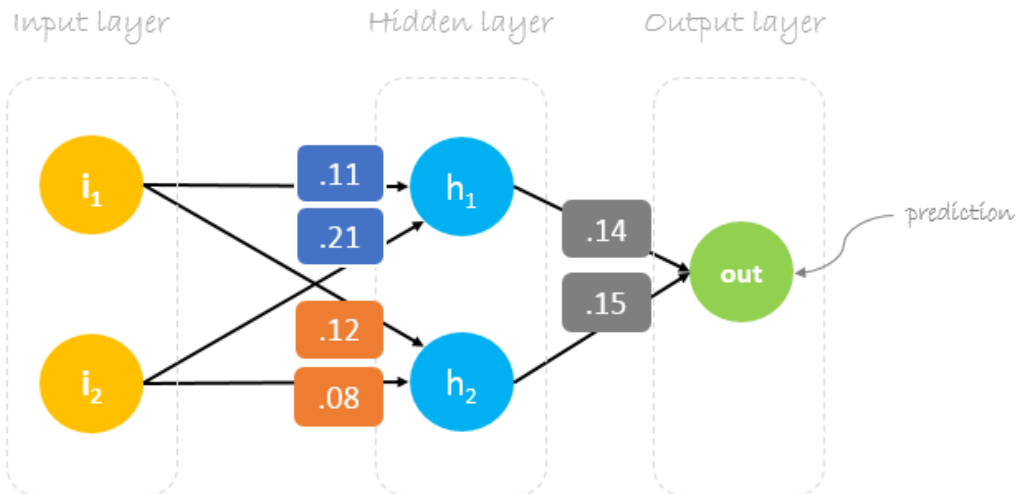
Backpropagation: Example

- From HMKCode
- Note that there is no activation function (it would just add one more step in the chain rule)



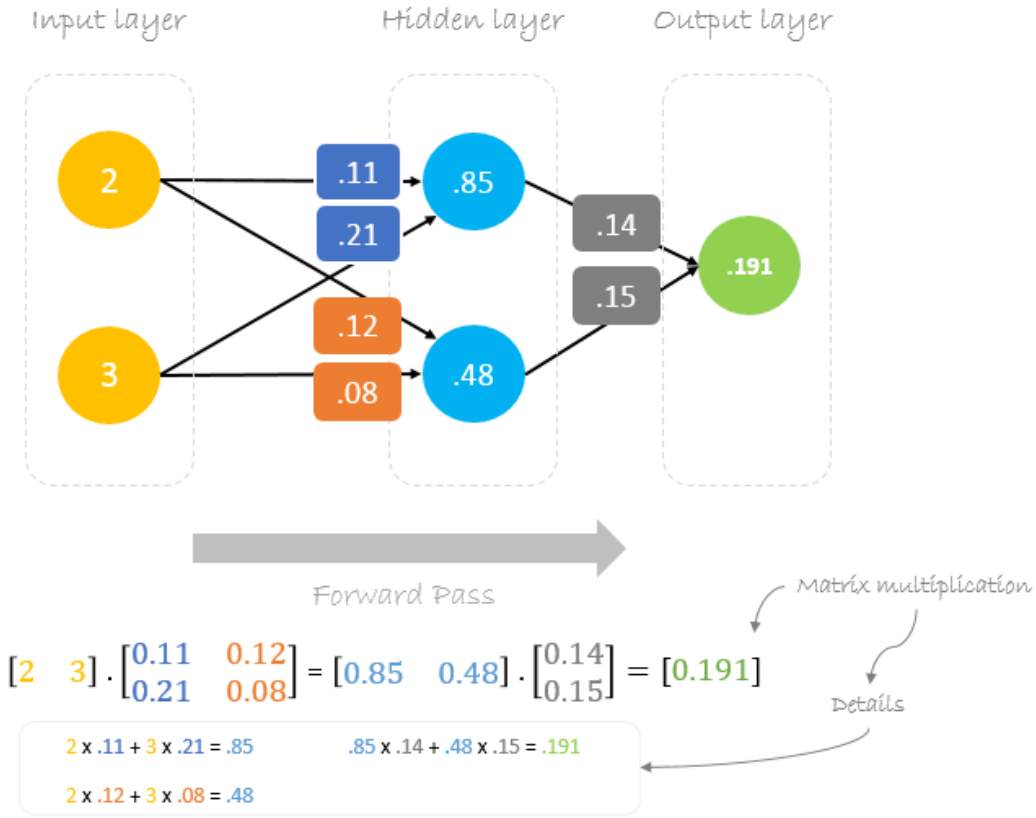
Backpropagation: Example

- Weights



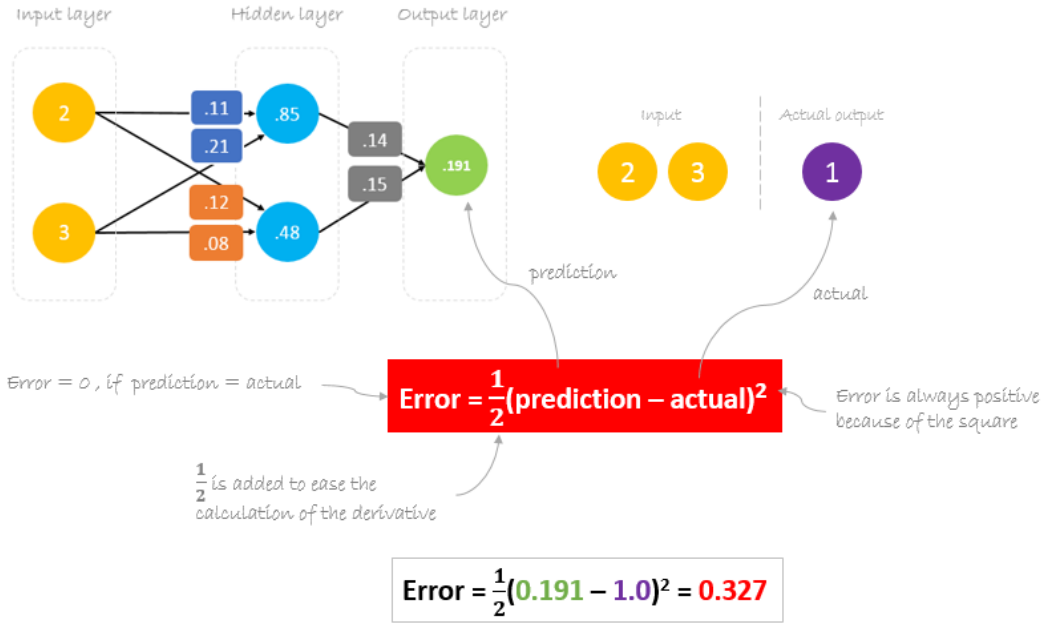
Backpropagation: Example

- Feedforward



Backpropagation: Example

- Error from the desired target



Backpropagation: Example

- Prediction function

$$\textbf{prediction} = \text{out}$$

$$\textbf{prediction} = (h_1)\, w_5 + (h_2)\, w_6$$

$$h_1 = i_1 w_1 + i_2 w_2$$
$$h_2 = i_1 w_3 + i_2 w_4$$

$$\textbf{prediction} = (i_1 w_1 + i_2 w_2)\, w_5 + (i_1 w_3 + i_2 w_4)\, w_6$$

to change ***prediction*** value,
we need to change ***weights***

Backpropagation: Example

- Gradient descent

Old weight

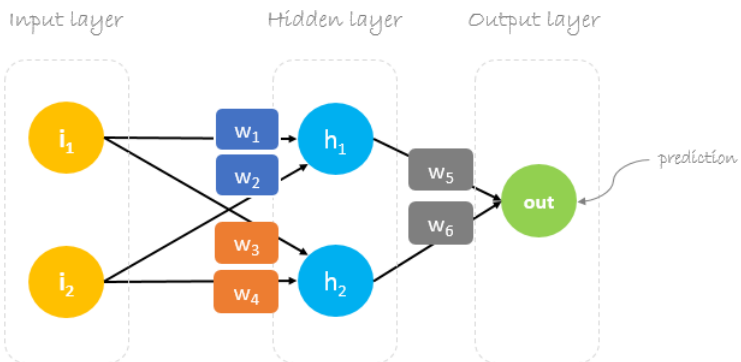
Derivative of Error with respect to weight

$$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

New weight

Learning rate

$$^{*}W_6 = W_6 - a\left(\frac{\partial Error}{\partial W_6}\right)$$



Backpropagation: Example

- Chain rule

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$ ← chain rule

Error = $\frac{1}{2}$(prediction – actual)²

prediction = $(i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6$

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial (i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(\mathbf{predictoin - actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (i_1 w_3 + i_2 w_4)$$ ← $h_2 = i_1 w_3 + i_2 w_4$

$$\frac{\partial Error}{\partial W_6} = (\mathbf{predictoin - actula}) * (h_2)$$

Δ = prediction – actual ← delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

Backpropagation: Example

- Chain rule

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$ ← chain rule

Error = $\frac{1}{2}$(prediction – actual)²

prediction = $(i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6$

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial (i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(\mathbf{predictoin - actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (i_1 w_3 + i_2 w_4)$$ ← $h_2 = i_1 w_3 + i_2 w_4$

$$\frac{\partial Error}{\partial W_6} = (\mathbf{predictoin - actula}) * (h_2)$$

Δ = prediction – actual ← delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

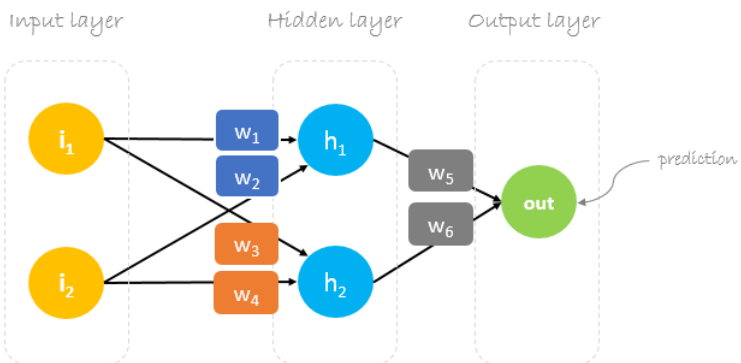
$$^*W_6 = W_6 - a\,\Delta h_2$$

Backpropagation: Example

- Chain rule

$$^{*}W_6 = W_6 - a\,\Delta h_2$$

$$^{*}W_5 = W_5 - a\,\Delta h_1$$



Backpropagation: Example

- Chain rule

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial h_1} * \frac{\partial h_1}{\partial W_1}$$ ← chain rule

$$\frac{\partial Error}{\partial W_1} = \frac{\partial \frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial (h_1)\, w_5 + (h_2)\, w_6}{\partial h_1} * \frac{\partial\, i_1\, w_1 + i_2\, w_2}{\partial w_1}$$

$$\frac{\partial Error}{\partial W_1} = 2 * \frac{1}{2}(\mathbf{predictoin - actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (w_5) * (i_1)$$

$$\frac{\partial Error}{\partial W_1} = (\mathbf{predictoin - actula}) * (w_5 i_1)$$

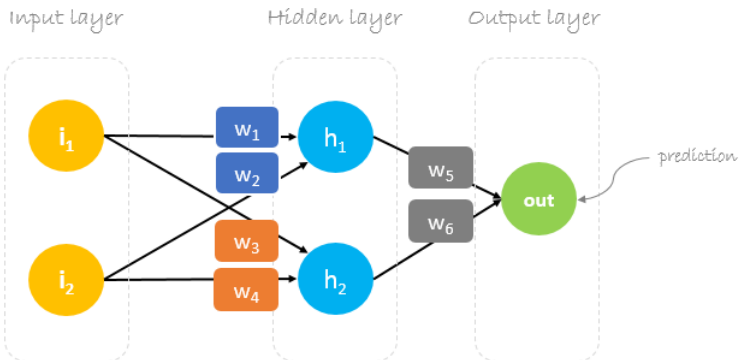
$$\frac{\partial Error}{\partial W_1} = \Delta\, w_5 i_1$$

Error = $\frac{1}{2}$(prediction − actual)²

prediction = $(h_1)\, w_5 + (h_2)\, w_6$

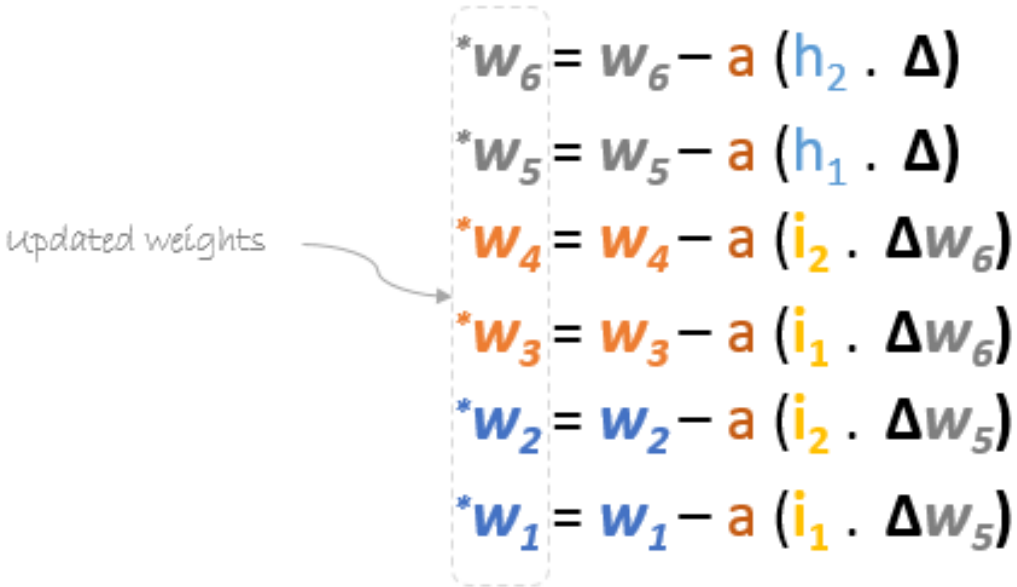
$h_1 = i_1 w_1 + i_2 w_2$

Δ = prediction − actual ← delta



Backpropagation: Example

- Summarized



Backpropagation: Example

- Summarized in matrix form

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} a h_1 \Delta \\ a h_2 \Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\,\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a\, i_1 \Delta w_5 & a\, i_1 \Delta w_6 \\ a\, i_2 \Delta w_5 & a\, i_2 \Delta w_6 \end{bmatrix}$$

Backpropagation: Multiple data points

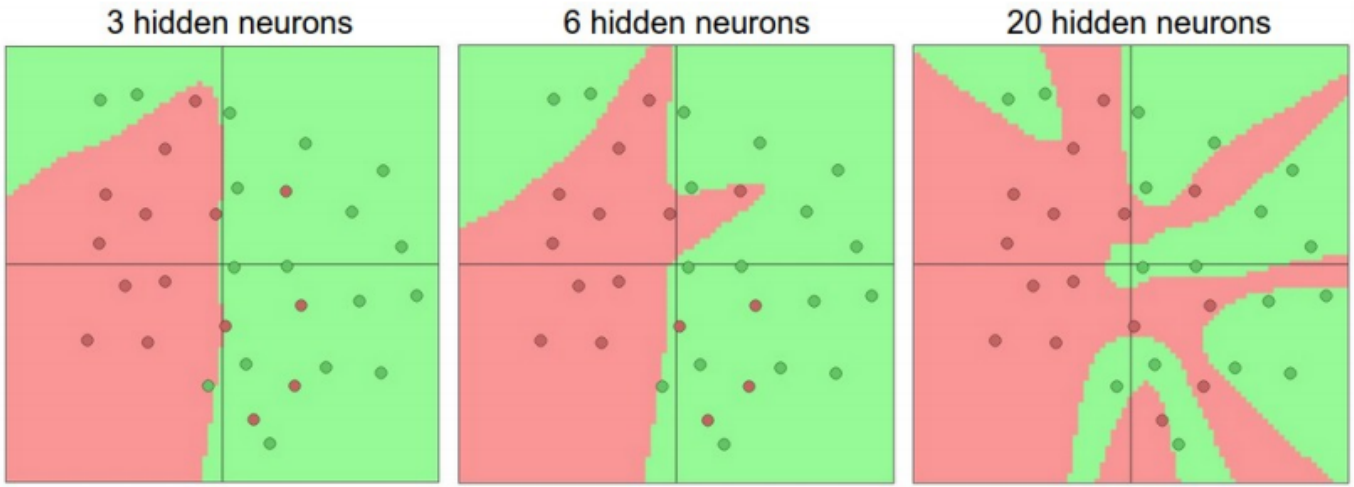
- Generally Δ is a vector, with the dimension of the number of training data points.
- The error can be the average of the error, so repeate the equations below for all training points and average the changes (the part after *a*)
- Fortunately numpy does not care about the number of dinemsions, so insted of the multiplication in the right matrices we can use dot product.

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} a h_1 \Delta \\ a h_2 \Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\,\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a\, i_1 \Delta w_5 & a\, i_1 \Delta w_6 \\ a\, i_2 \Delta w_5 & a\, i_2 \Delta w_6 \end{bmatrix}$$

How many layers?

- Neural network with at least one hidden layer is a universal approximator (can represent any function).

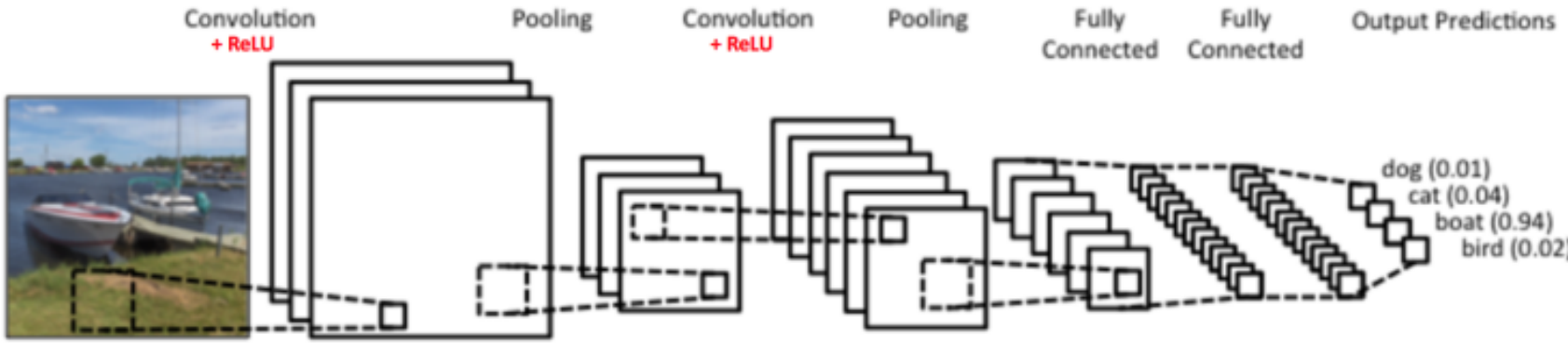


Do Deep Nets Really Need to be Deep? Jimmy Ba, Rich Caruana,

LeNet Architecture

Yann LeCun

- Convolution
- Non-linearity
- Pooling
- Classification



Author: ujjwalkarn

Convolution operator

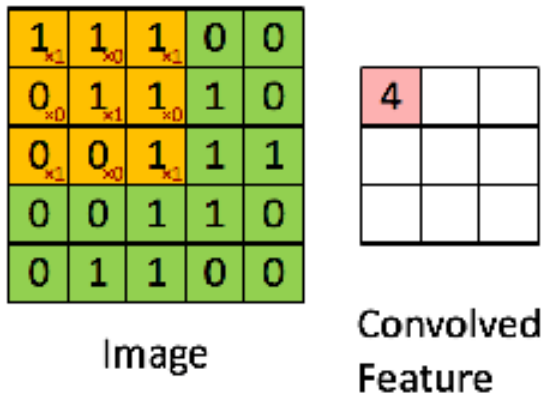
- 2d matrix
- Example:
  - Original image:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

  - Convolution matrix:

1	0	1
0	1	0
1	0	1

  - Result:



Convolution operator

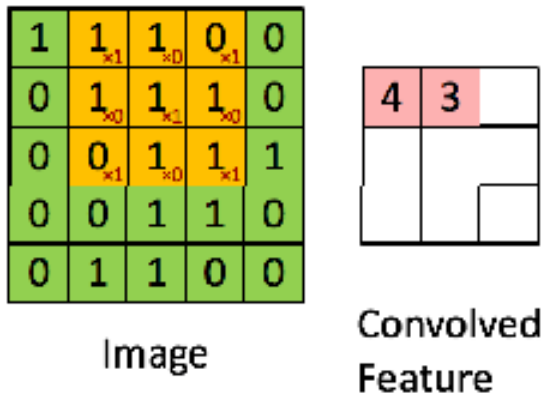
- 2d matrix
- Example:
  - Original image:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

  - Convolution matrix:

1	0	1
0	1	0
1	0	1

  - Result:



Convolution operator

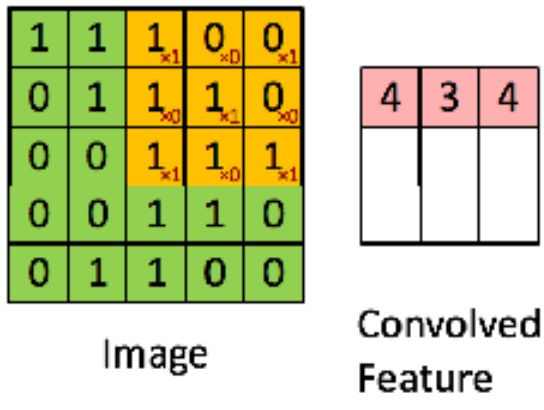
- 2d matrix
- Example:
  - Original image:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

  - Convolution matrix:

1	0	1
0	1	0
1	0	1

  - Result:



Convolution operator

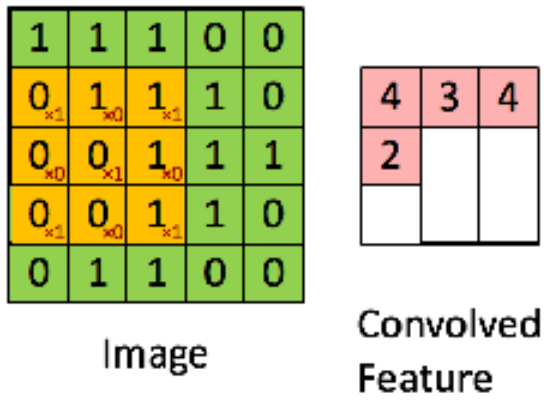
- 2d matrix
- Example:
  - Original image:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

  - Convolution matrix:

1	0	1
0	1	0
1	0	1

  - Result:



Convolution operator

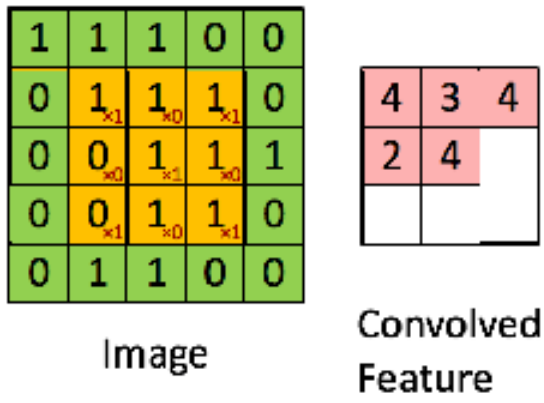
- 2d matrix
- Example:
  - Original image:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

  - Convolution matrix:

1	0	1
0	1	0
1	0	1

  - Result:



Convolution operator

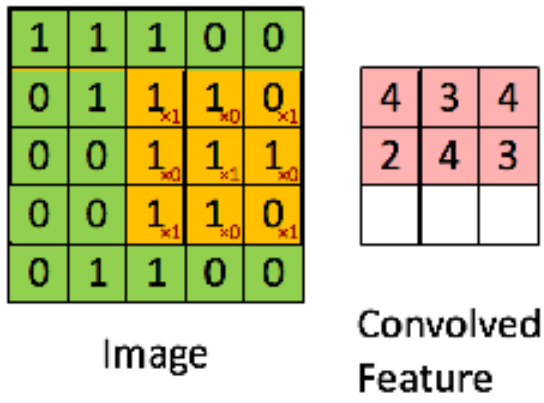
- 2d matrix
- Example:
  - Original image:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

  - Convolution matrix:

1	0	1
0	1	0
1	0	1

  - Result:



Convolution operator

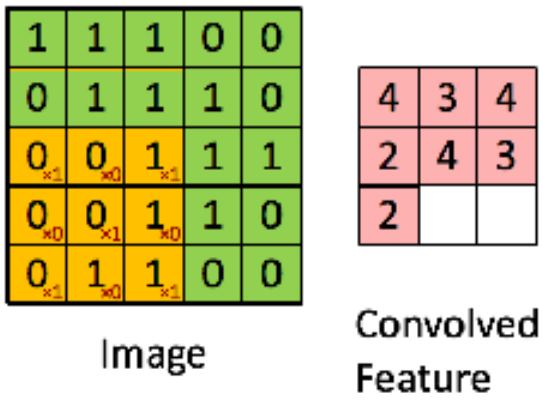
- 2d matrix
- Example:
  - Original image:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

  - Convolution matrix:

1	0	1
0	1	0
1	0	1

  - Result:



Convolution operator

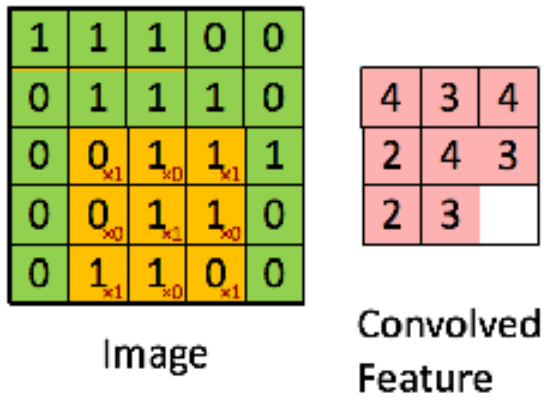
- 2d matrix
- Example:
  - Original image:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

  - Convolution matrix:

1	0	1
0	1	0
1	0	1

  - Result:



Convolution operator

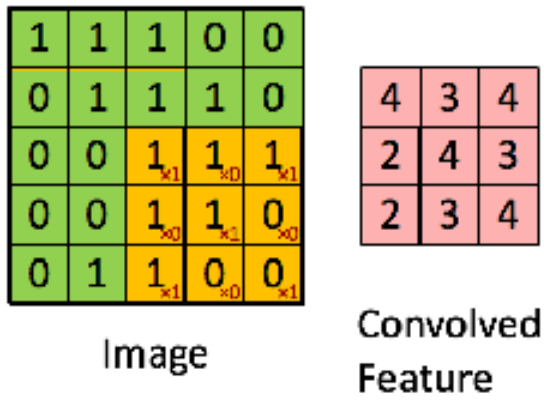
- 2d matrix
- Example:
  - Original image:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

  - Convolution matrix:

1	0	1
0	1	0
1	0	1

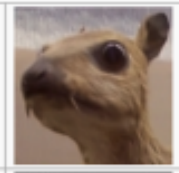
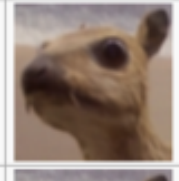
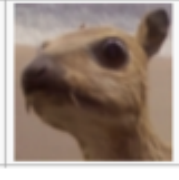
  - Result:



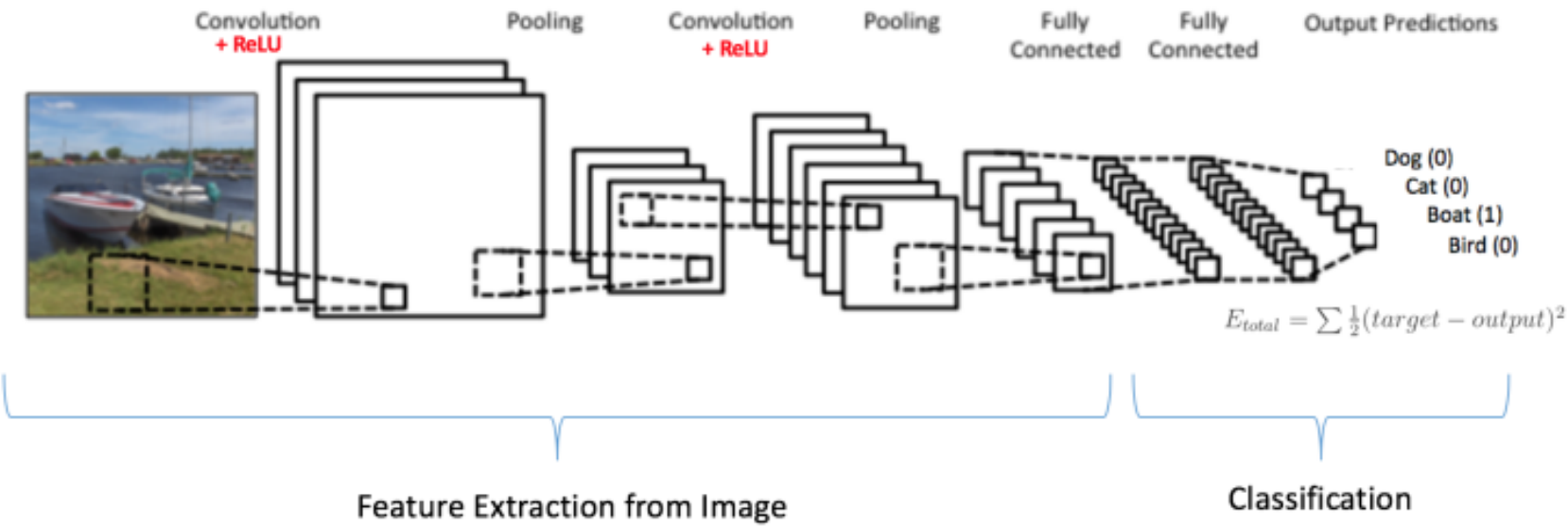
Convolution operator

- The convolution operator is called *filter or kernel*
- The result of the convolution is *feature map*

Convolution operator: examples

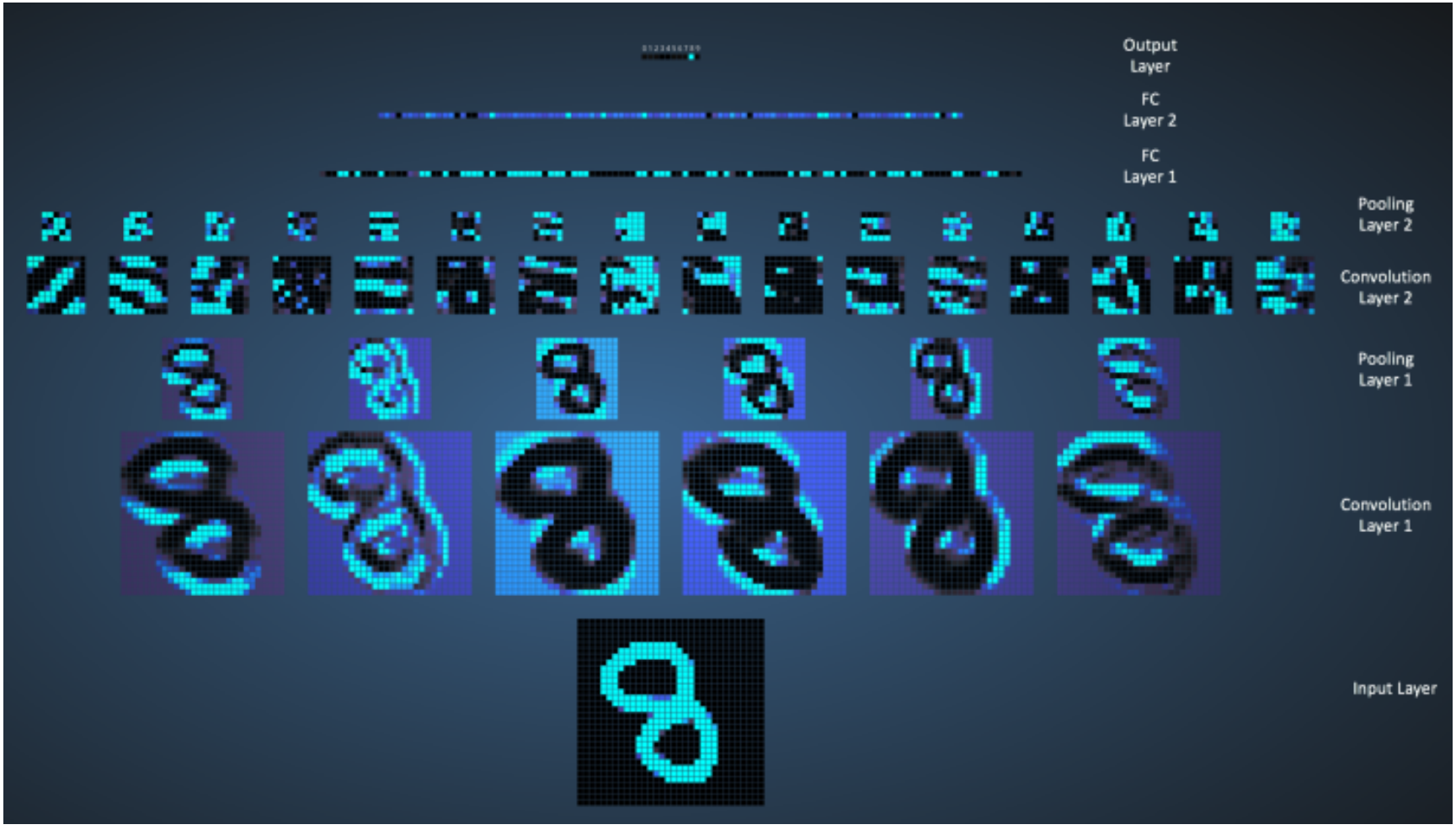
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Full convolution neural network



Examples of learned features

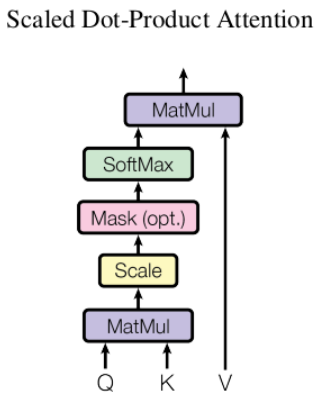
- MNIST example
  `https://adamharley.com/nn_vis/cnn/2d.html`

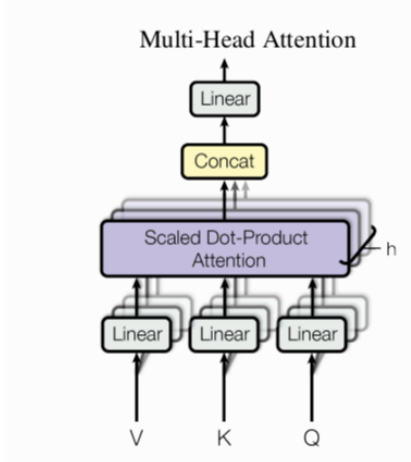


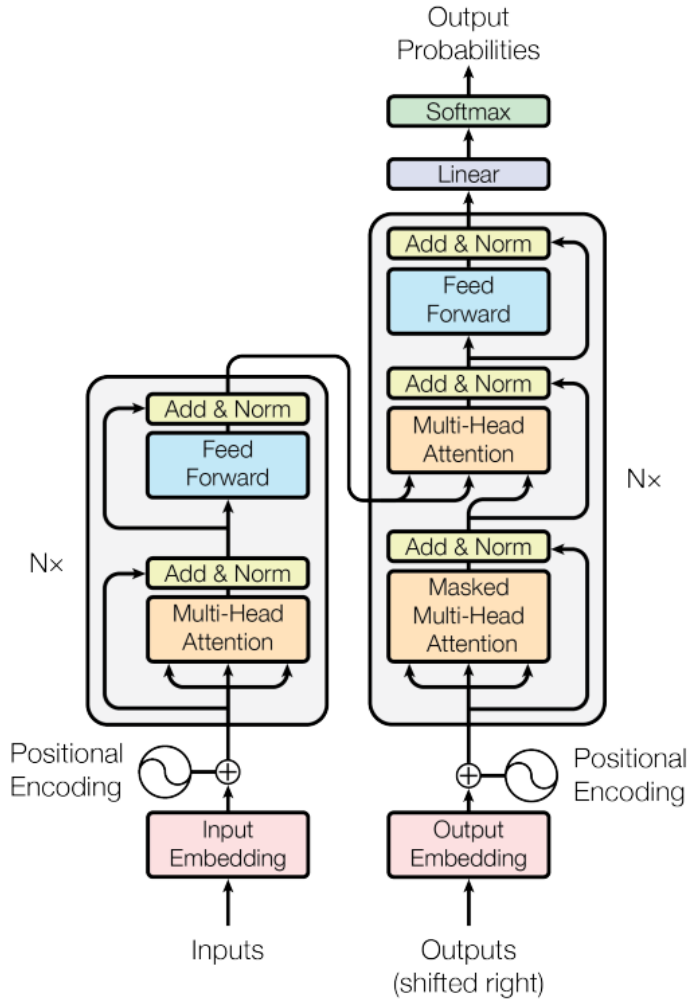
Convolutional layers

- Number of parameters (200x200 RGB image):
  - Fully connected layer to a layer of 300 nodes:
    - Weights: $(200 \cdot 200 \cdot 3) \cdot 300$
    - Biases: 300
    - Total: $36000300 \simeq 3.6 \cdot 10^7$
  - Convolutional layer
    - Weights per filter $w \cdot w \cdot 3$, where w is the width of the filters
    - One bias
    - Number of weights per filter $w^2 + 1$
    - For 300 filter (usually people use only a few dozens)
    - Total: $300 \cdot 10 = 3000$

ChatGPT structure







Attention is all you need

A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - proceedings.neurips.cc

… to attend to **all** positions in the decoder up to and including that position. **We need** to prevent … **We** implement this inside of scaled dot-product **attention** by masking out (setting to $-\infty$) …

☆ Save 99 Cite Cited by 76320 Related articles All 46 versions »