Quantitative Methods for Cognitive Scientists

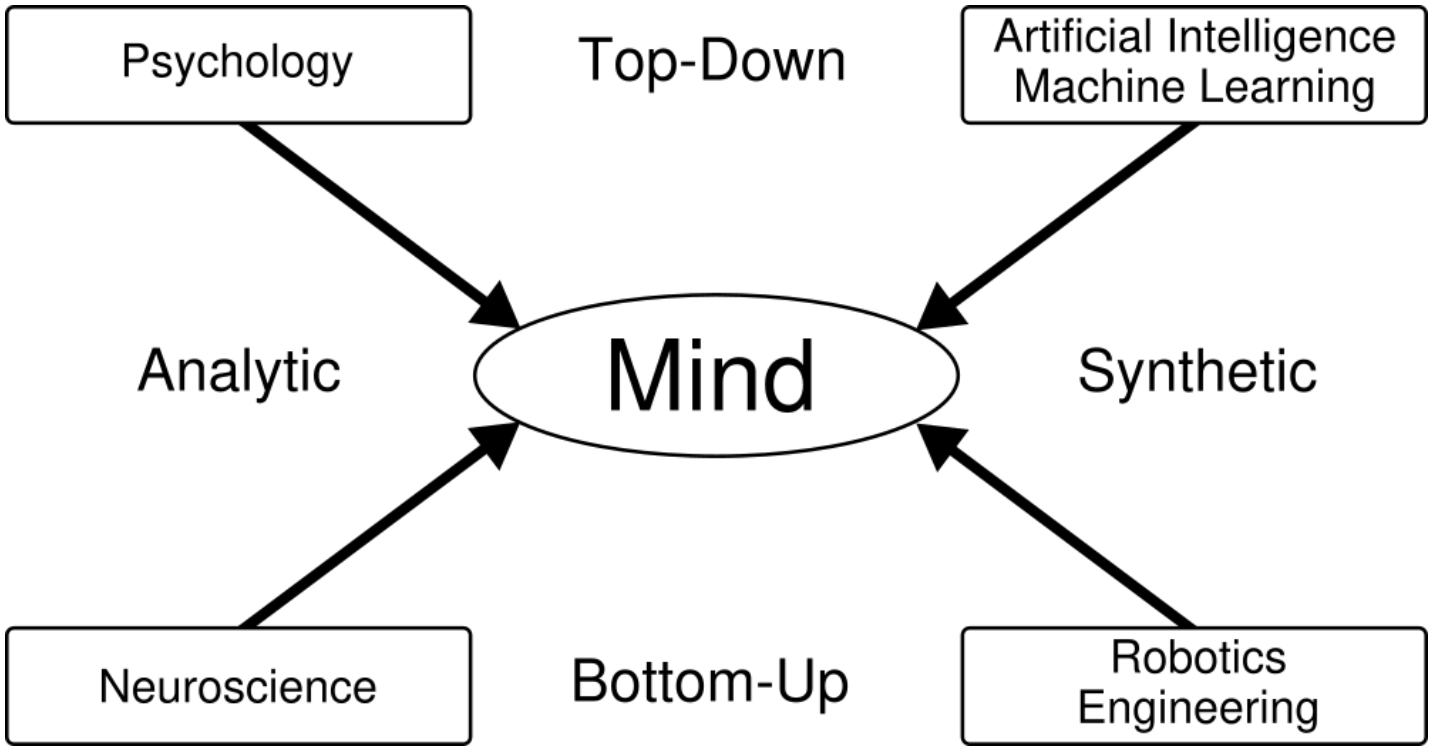
Machine Learning and Neural Networks

Emre Neftci

Department of Cognitive Sciences, UC Irvine,

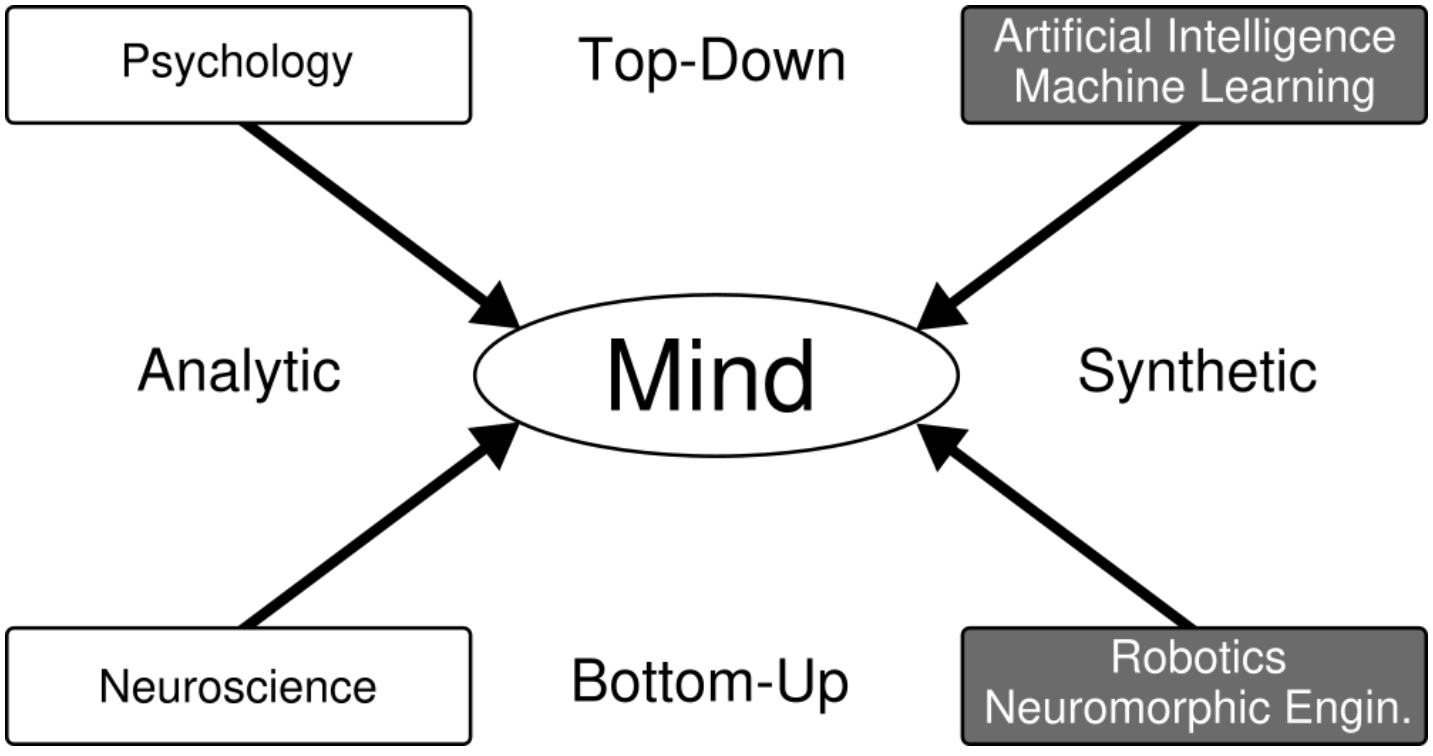
June 4, 2019

Approaches to the Mind



Franklin, *Artificial Minds*, 1995

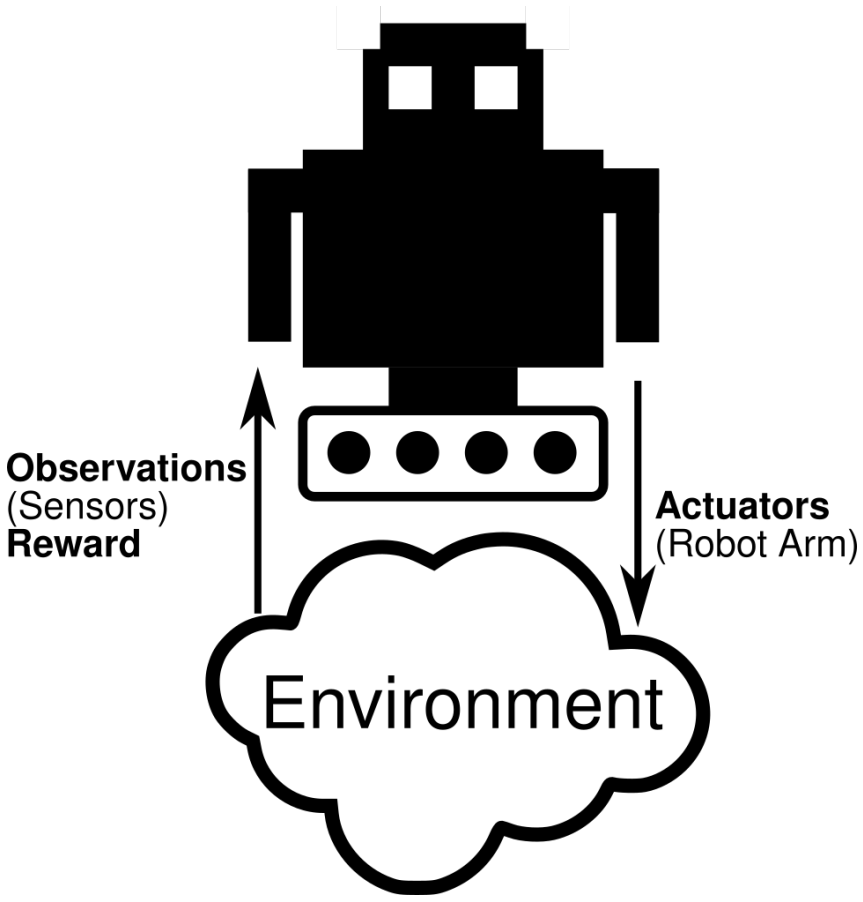
Approaches to the Mind



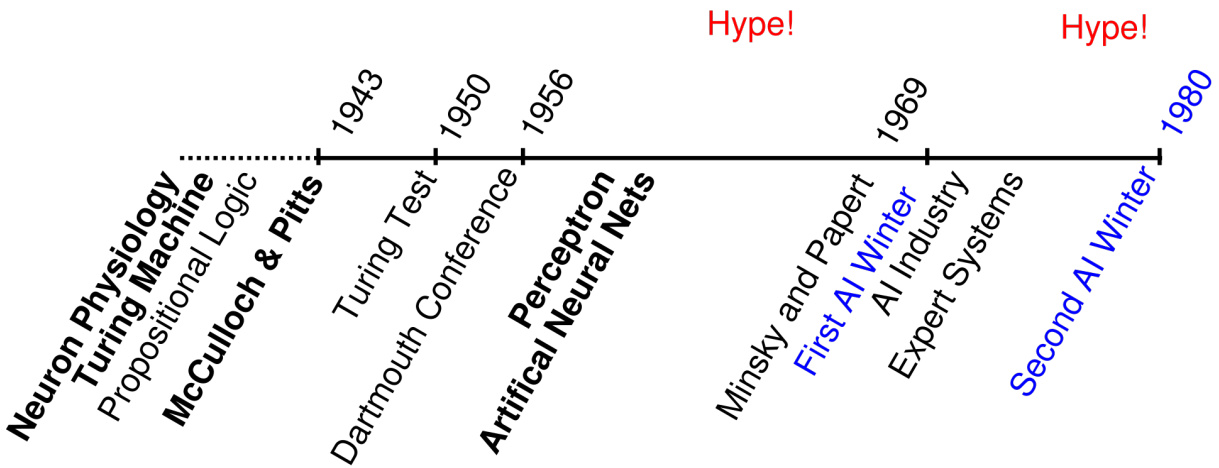
Franklin, *Artificial Minds*, 1995

Research in cognitive science to understand and build cognitive systems

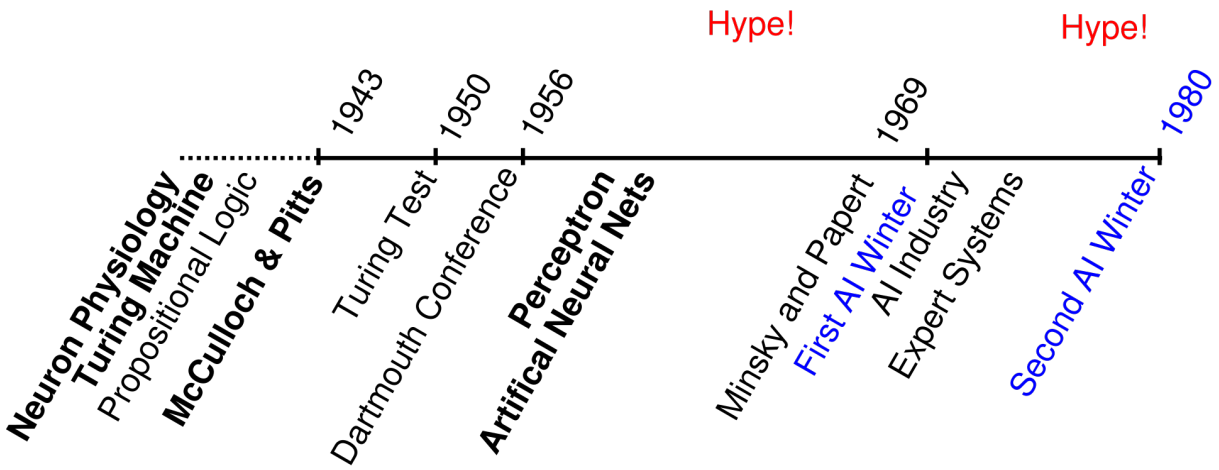
Artificial Intelligence & Cognitive Agents



History of Artificial Intelligence and Neural Networks



History of Artificial Intelligence and Neural Networks



Early AI shortcomings:

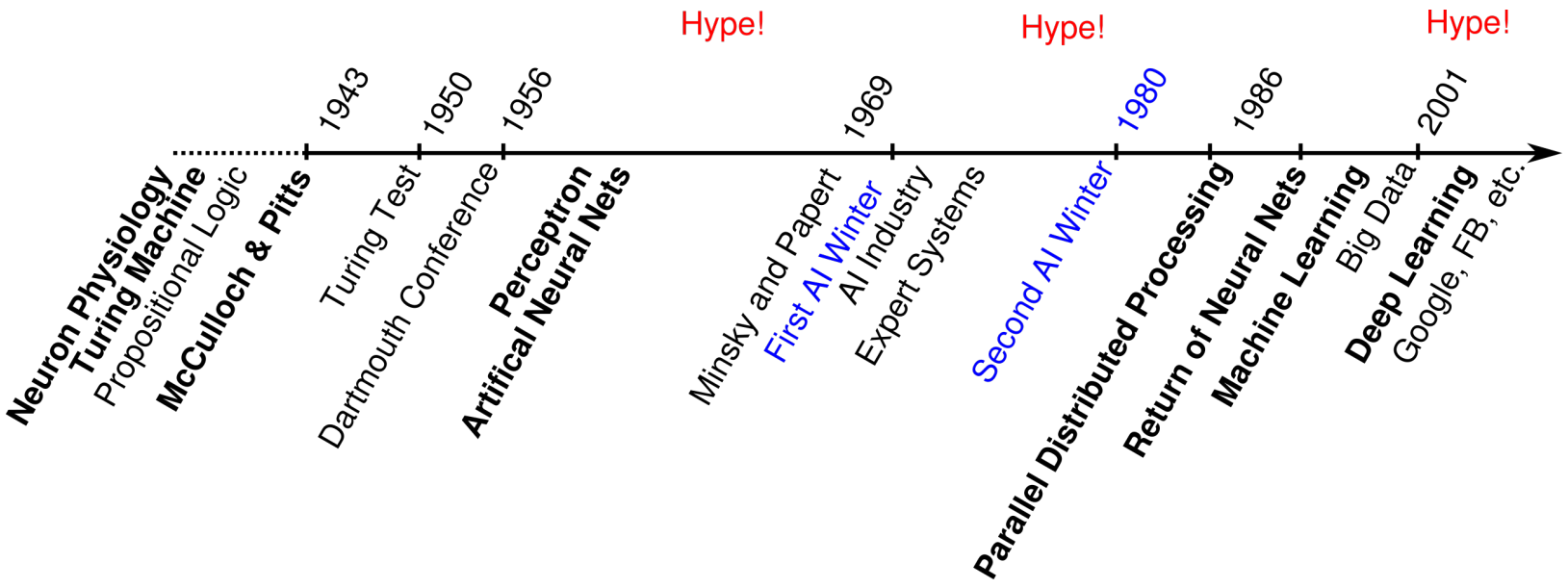
- Symbol based processing lacks domain-specific knowledge
- Combinatorial explosion: solutions to small problems did not scale to exponentially large problems.
- Solving a problem in principle is very different than solving it practically

AI’s moonshot

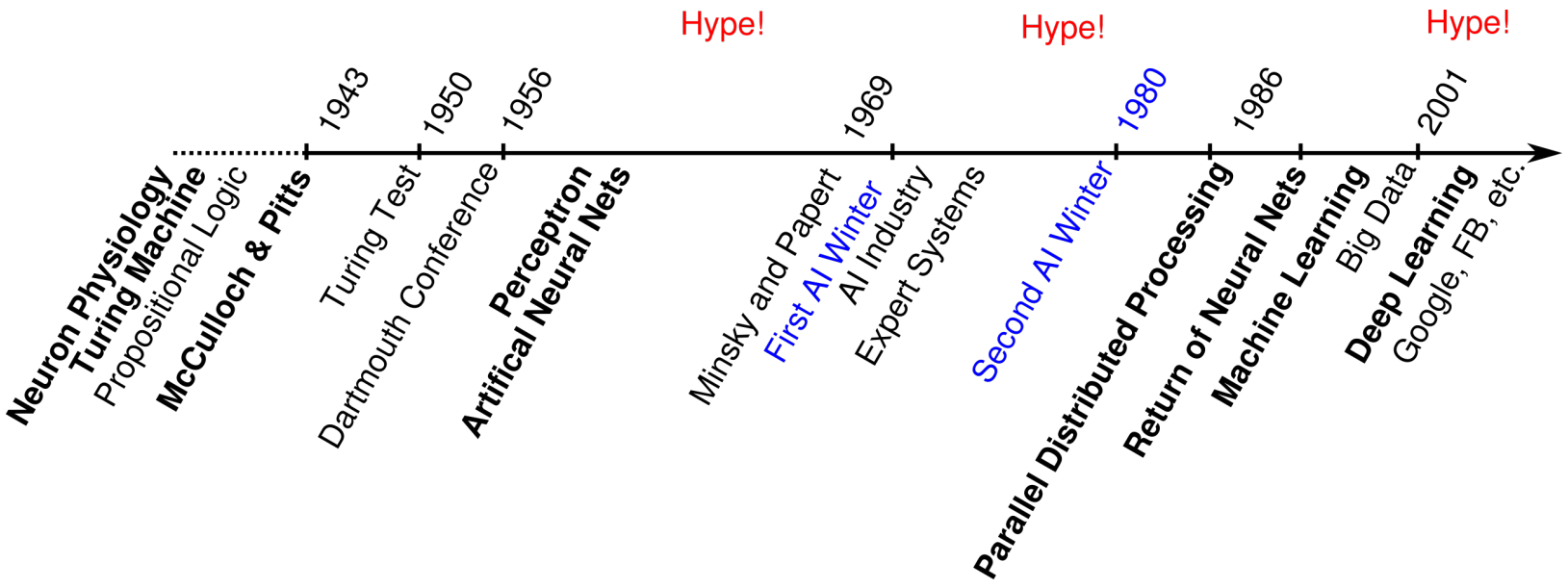
AI's moonshot

There is a need to systematically learn domain-specific knowledge to solve practical AI tasks

Modern Artificial Intelligence and Machine Learning



Modern Artificial Intelligence and Machine Learning



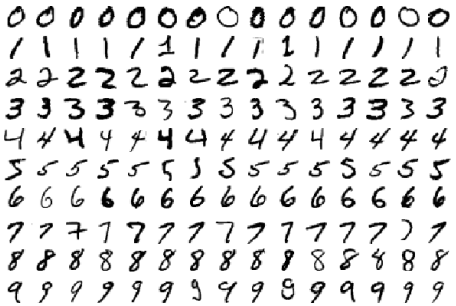
Good old online backpropagation for plain multilayer perceptrons yields a very low 0.35% error rate on the MNIST handwritten digits benchmark. All we need to achieve this best result so far are many hidden layers, many neurons per layer, numerous deformed training images to avoid overfitting, and graphics cards to greatly speed up learning.

Cireşan, Meier, Gambardella, and Schmidhuber, *Neural computation*, 2010

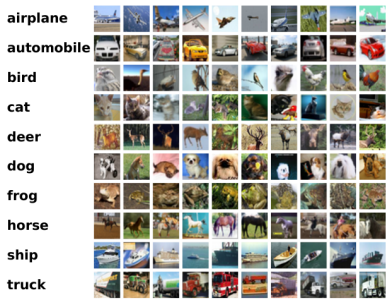
A lot of progress in machine learning can be attributed to better hardware and more data

Machine Learning

A machine learning algorithm is a program that can learn to solve a given using **data**.
Example Datasets:



MNIST



CIFAR-10

ImageNet

DARPA Neovision2 Tower benchmark

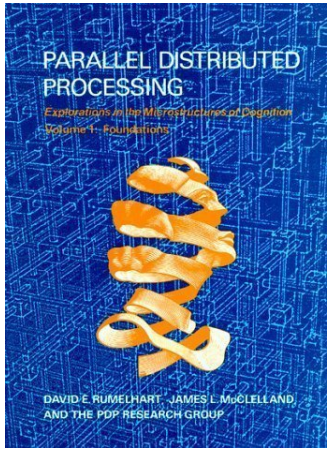
Provided enough data, machine learning usually can outperform experts' knowledge

Machine learning models for supervised learning:

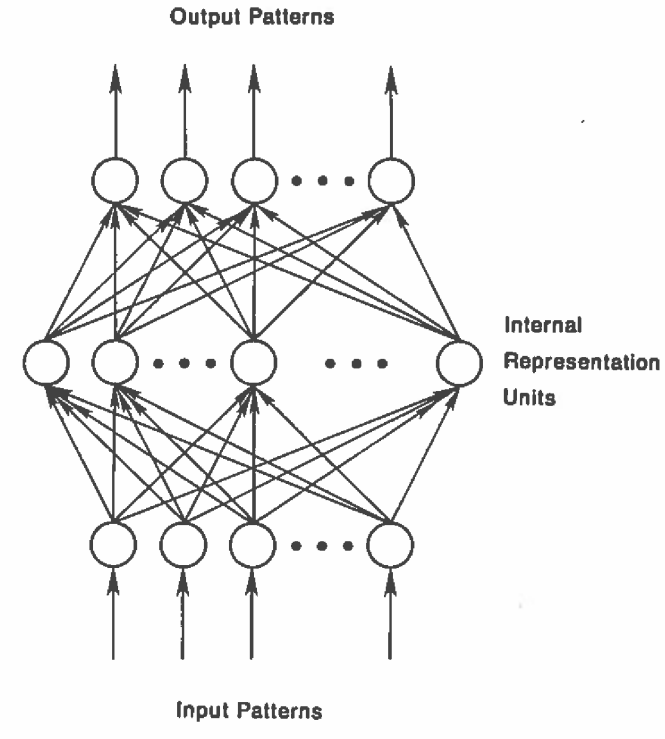
- Discriminant function
- Discriminative model
- Generative model

Connectionism and Neural Networks

"A set of approaches that models artificial intelligence using networks of simple (neuron-like) units."



Rumelhart, McClelland, Group, et al.,, 1988



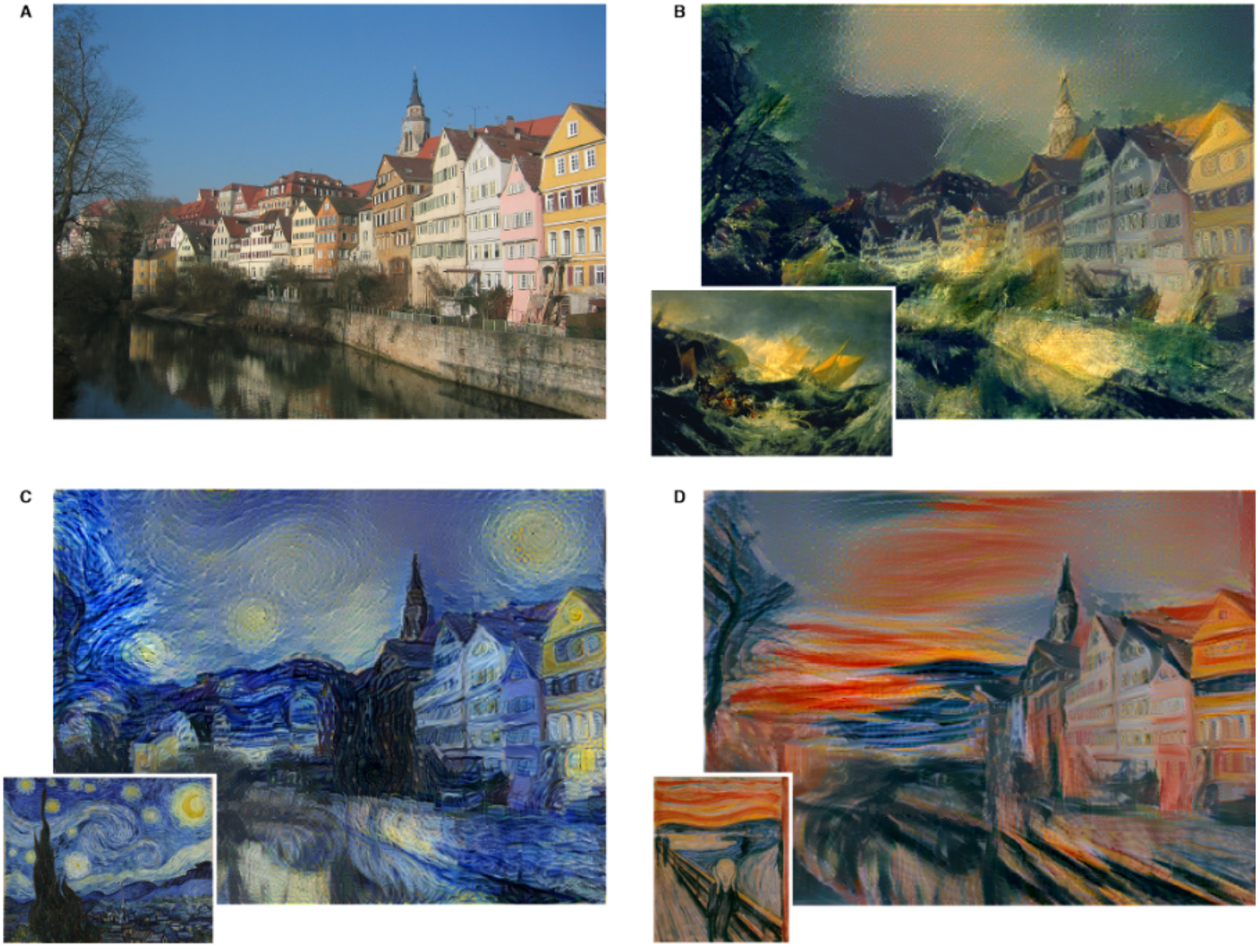
Neural networks can be viewed as machine learning algorithms that run on networks of neuron-like units

Example Applications
Classification

- ImageNet: 1M images, 1000 classes of images, 469x387 pixels
- Google's Inception v3 achieves 22.55% error, top-1 6.44 % top-5 error

Example Applications

Deep neural networks applied to style transfer



Example Applications

Deep neural networks applied to style transfer

London, England

Chicago, USA

Melbourne, Australia

https://deepempathy.mit.edu/

Example Applications

Pose Estimation

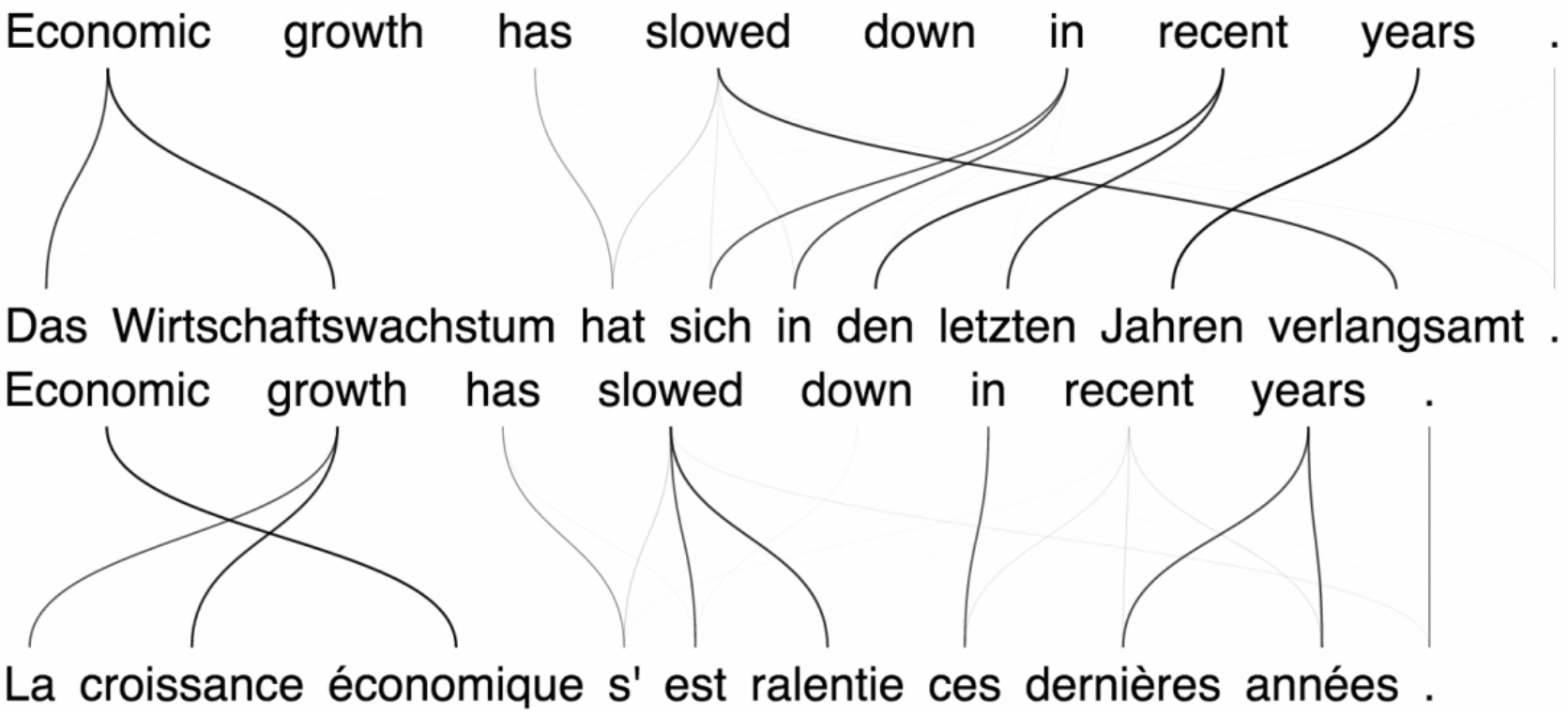
Example Applications

Image Generation

Generates images never seen before

Example Applications

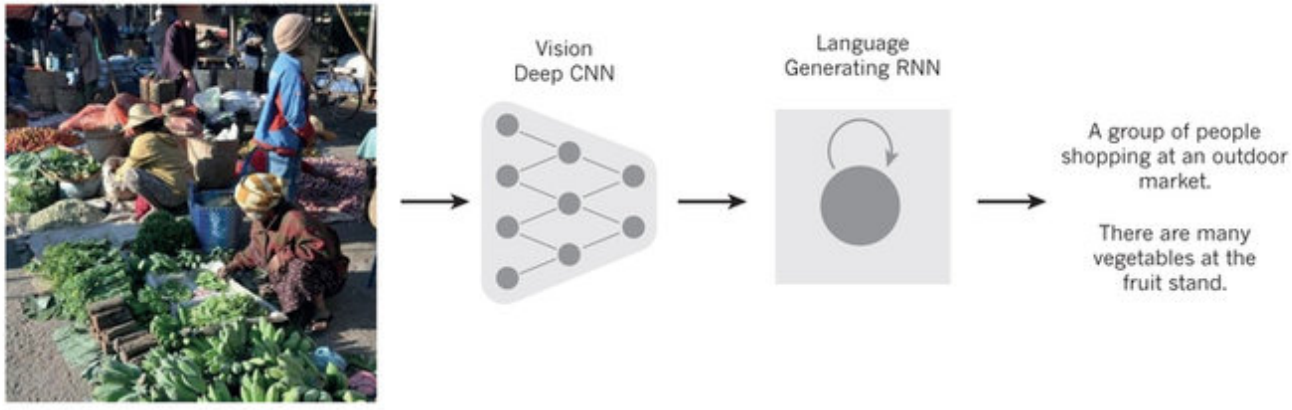
Translation



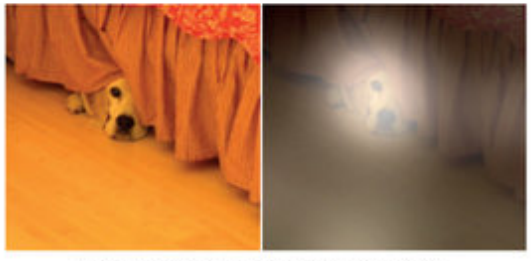
State-of-the-art uses neuroscience-inspired “Attention” mechanisms for natural language processing

Example Applications

Captioning

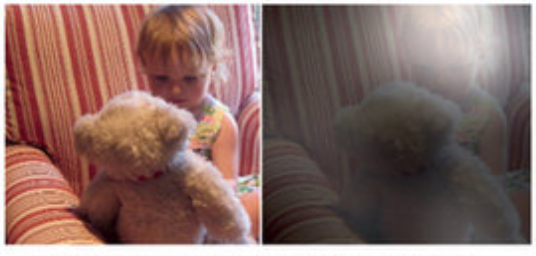


A woman is throwing a **frisbee** in a park.



A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background



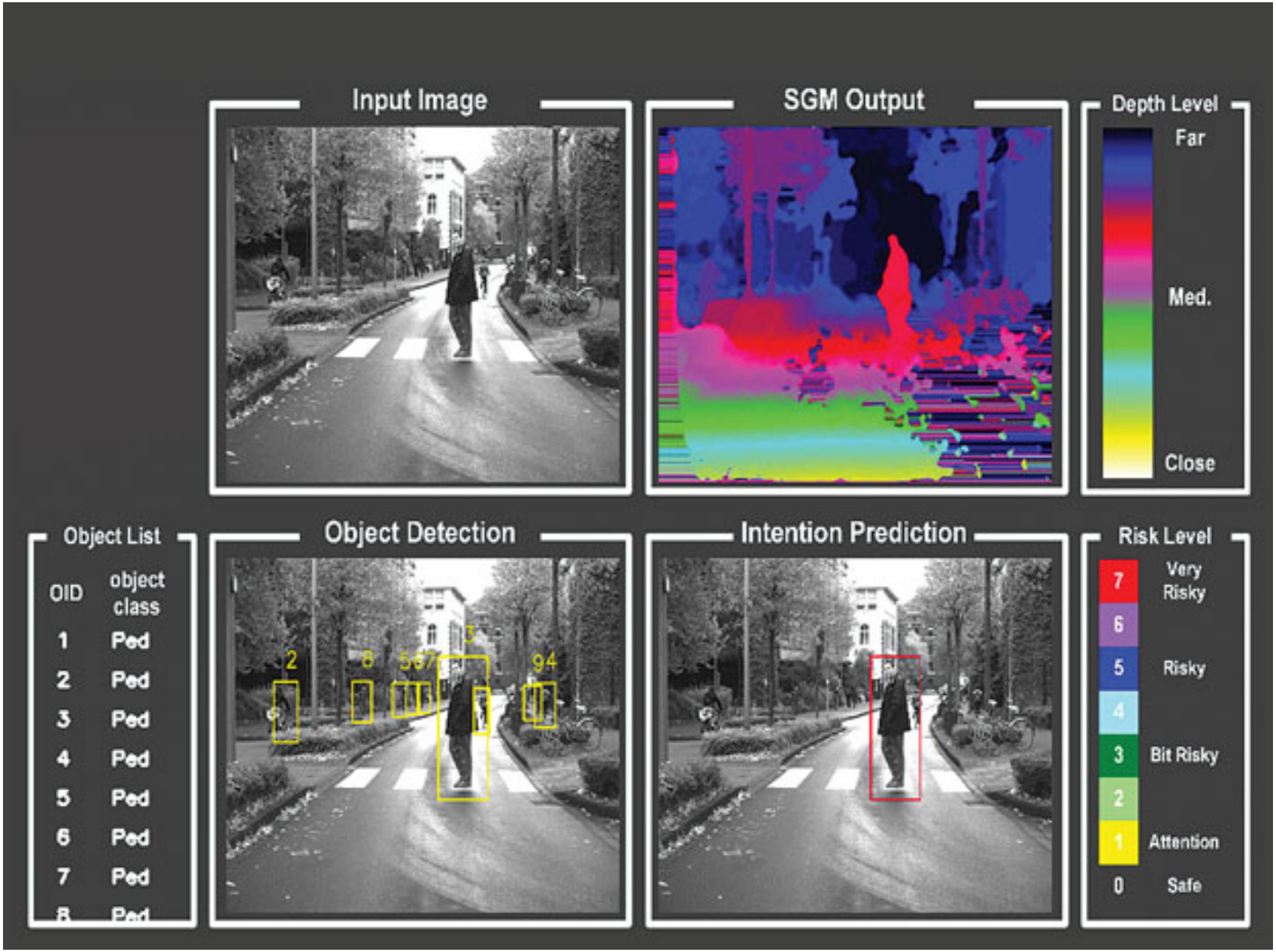
A little **girl** sitting on a bed with a teddy bear.

A group of **people** sitting on a boat in the water.

A giraffe standing in a forest with **trees** in the background.

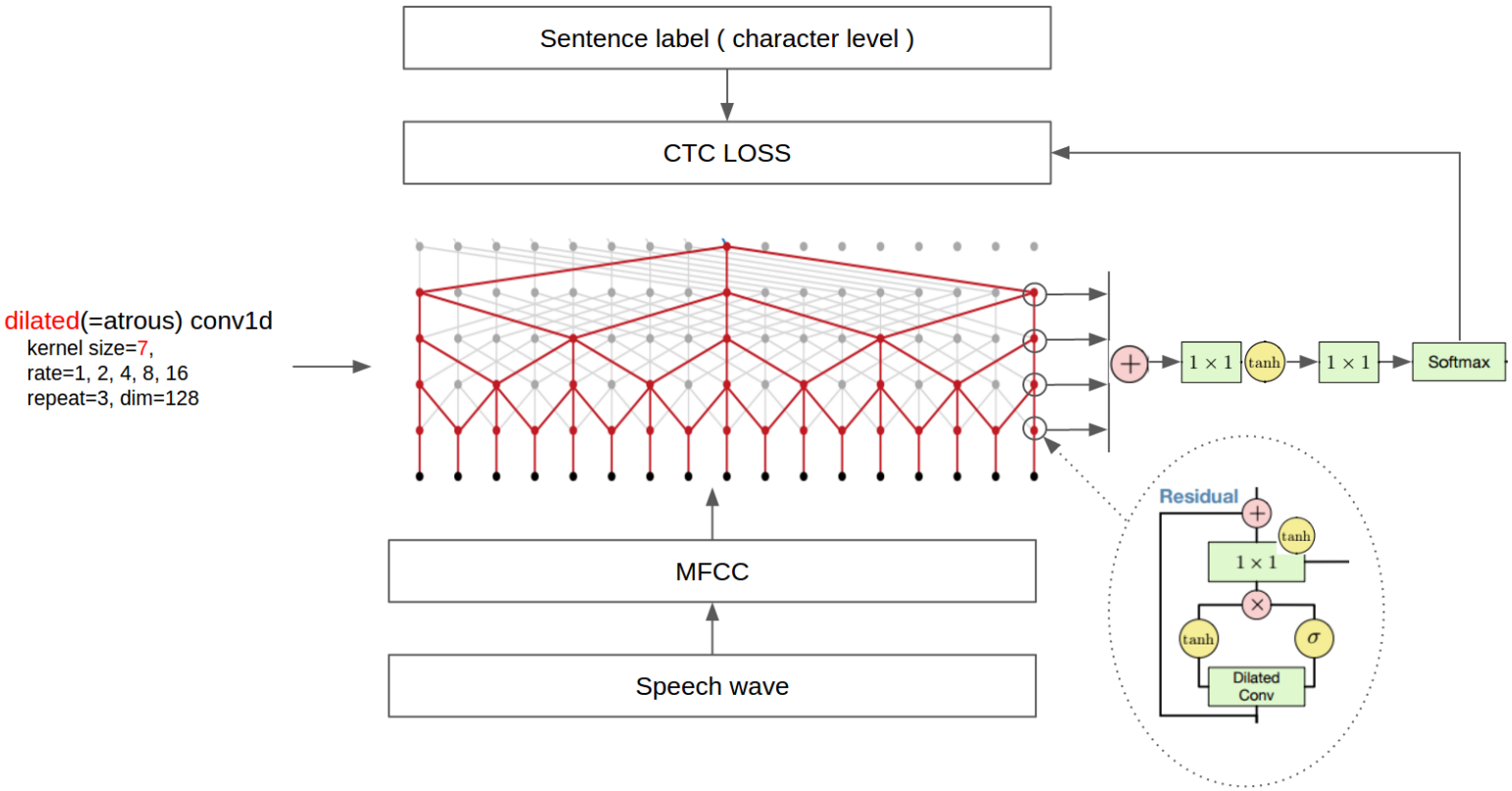
Example Applications

Autonomous Vehicles



Example Applications

Wavenets



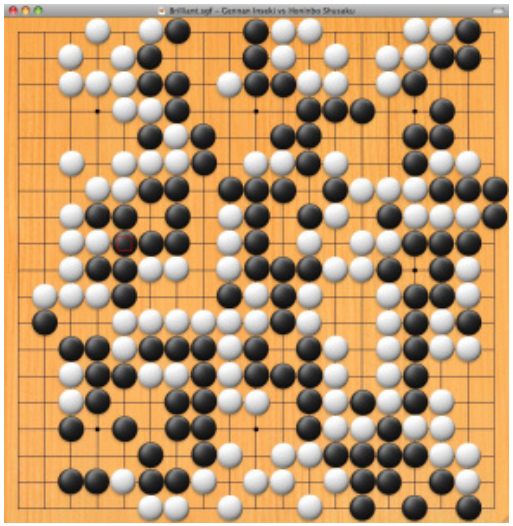
https://deepmind.com/blog/wavenet-generative-model-raw-audio/

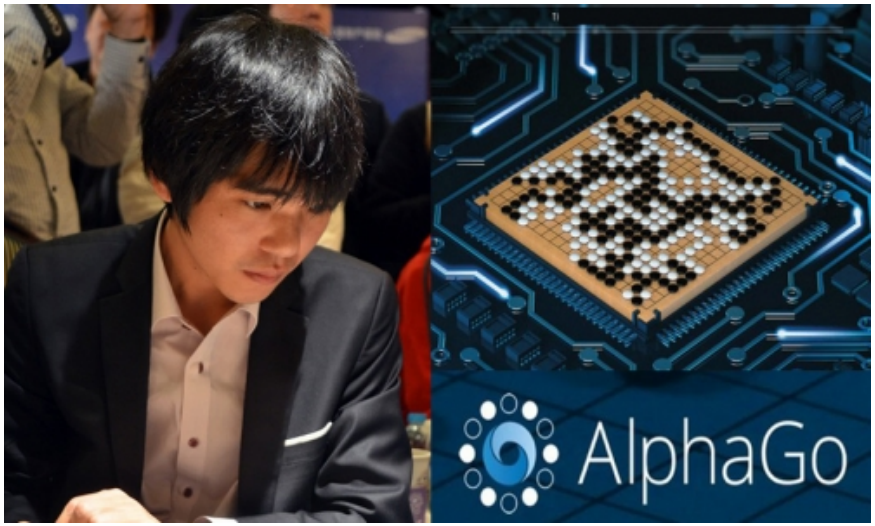
Example Applications

Game Playing

Deepmind uses Deep Reinforcement Learning to play Games, e.g. Go and Atari

Go





- The game Go: 19×19 grid, 10^{360} possible states. Number of atoms in the universe: 10^{82}
- Mar. 2016: Google Deepmind's AlphaGo beats World champion.

  "Deep neural networks trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play."

Silver et al., *Nature*, 2016

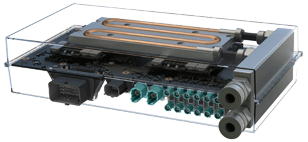
Machine Learning Accelerators

- Alphago training: 200 000 W (without cooling)
- Go chamption Lee Sedol: 20W
- Need for more efficient hardware driving a huge computing industry

TPU (Google)

Movidius (Intel)

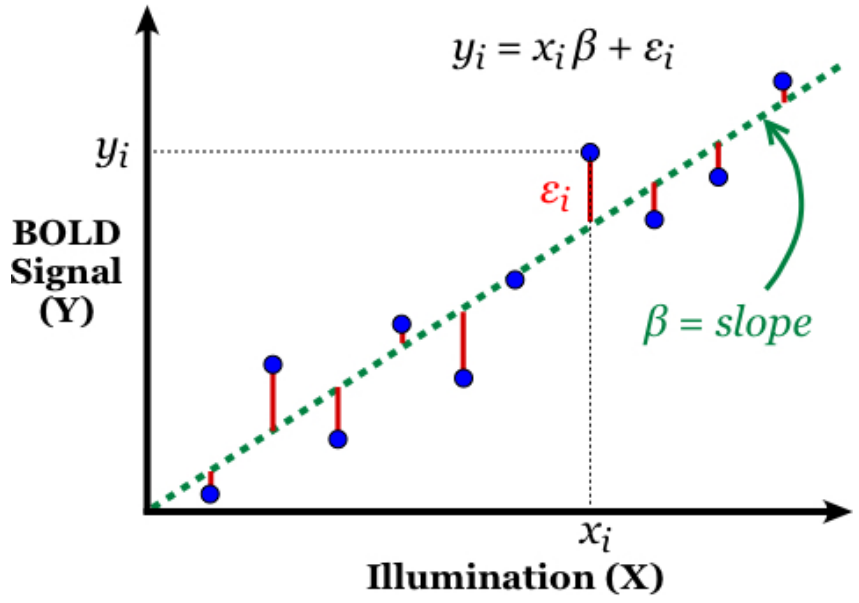
Microsoft Brainwave



NVIDIA

A (gentle) Mathematical Introduction to Machine Learning & Neural Networks

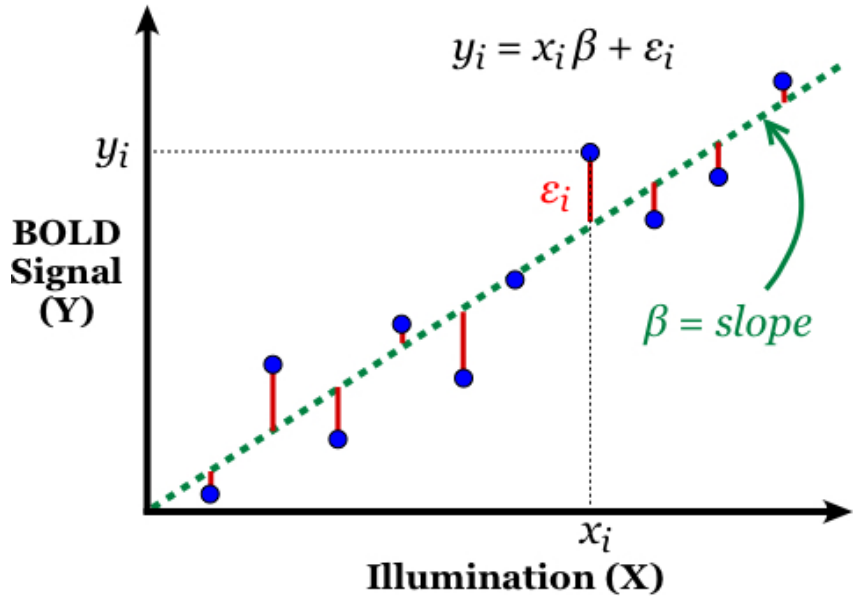
Example Model: BOLD signal in fMRI



- Data dimension: 1, Model dimension: 1 (Original model we studied, but β renamed to W)

$$y_n = \alpha + Wx_n \tag{1}$$

Example Model: BOLD signal in fMRI



- Data dimension: 1, Model dimension: 1 (Original model we studied, but β renamed to W)

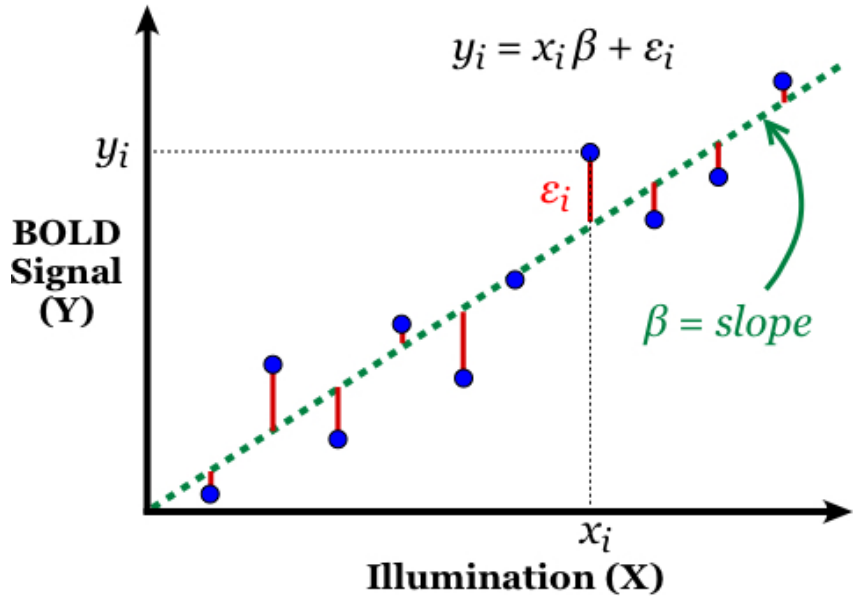
$$y_n = \alpha + Wx_n \tag{1}$$

- Data dimension: M, Model dimension: 1

$$y_n = \alpha + W_1x_{1,n} + W_2x_{2,n} + \ldots + W_Mx_{M,n}$$

$$y_n = \alpha + \sum_{j=1}^{M} W_jx_{j,n} \tag{2}$$

Example Model: BOLD signal in fMRI (continued)

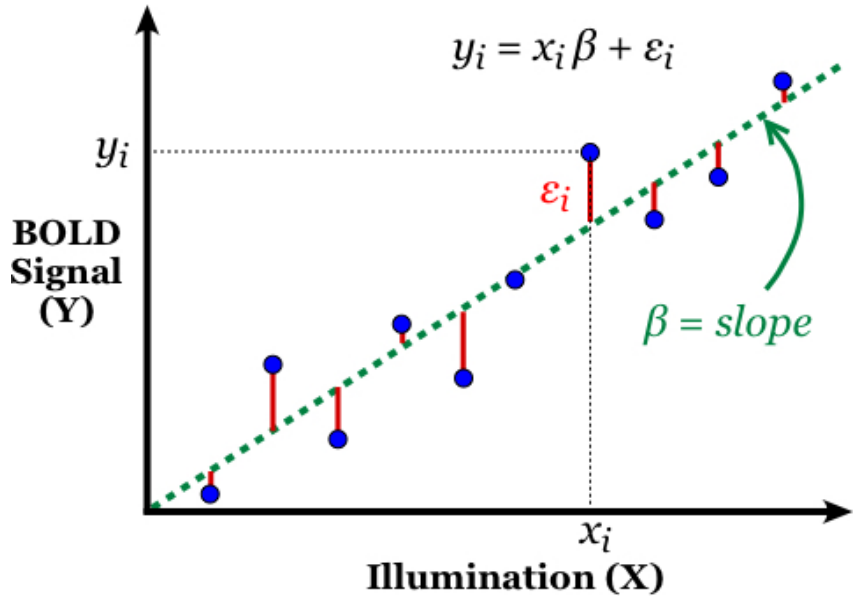


- Data dimension: M, Model dimension: 1

$$y_n = \alpha + \beta_1 x_{1,n} + W_2 x_{2,n} + \ldots + W_M x_{M,n}$$

$$y_n = \alpha + \sum_{j=1}^{M} W_j x_{j,n} \qquad (3)$$

Example Model: BOLD signal in fMRI (continued)



- Data dimension: M, Model dimension: 1

$$y_n = \alpha + \beta_1 x_{1,n} + W_2 x_{2,n} + \ldots + W_M x_{M,n}$$

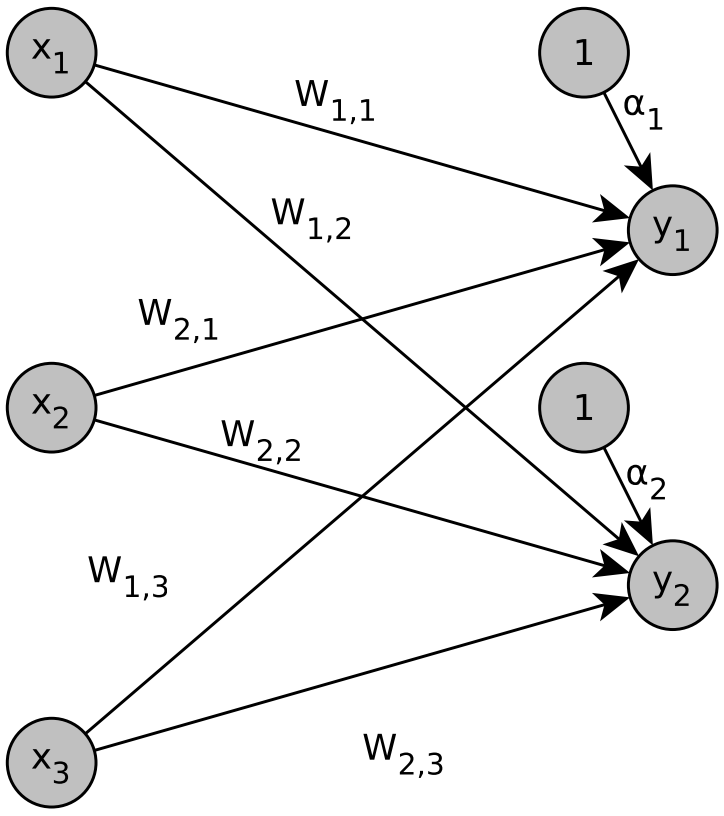
$$y_n = \alpha + \sum_{j=1}^{M} W_j x_{j,n} \tag{3}$$

- Data dimension: M, Model dimension: N

$$y_{i,n} = \alpha_i + \sum_{j=1}^{M} W_{i,j} x_{j,n} \tag{4}$$

Neural Networks for Linear Regression

This model can be viewed as a “neural network” with M inputs and N outputs (Here $M = 3$, $N = 2$):



$$y_{i,n} = \alpha_i + \sum_{j=1}^{M} W_{i,j} x_{j,n} \tag{5}$$

Vectors

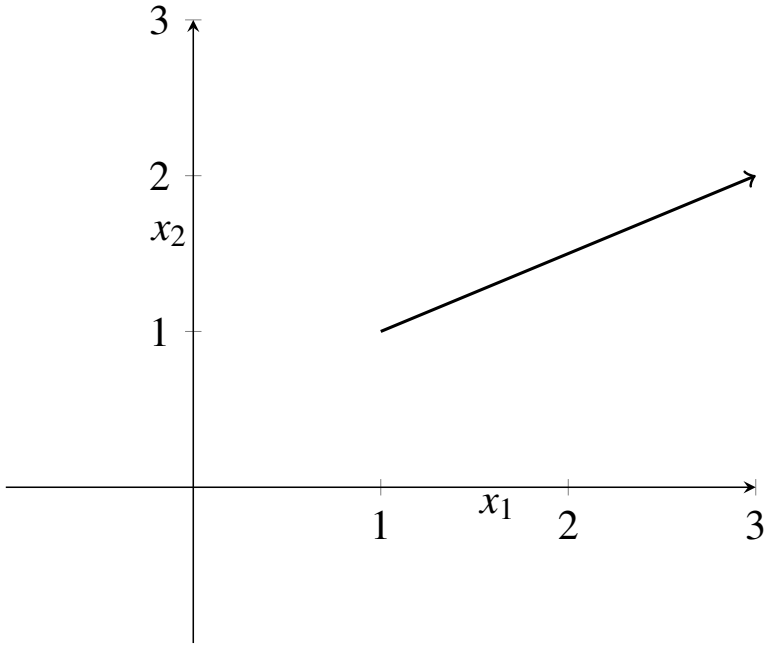
- Mathematically, $\mathbf{x}_n$ and $\mathbf{y}_n$ are vectors and $\mathbf{W}$ is a matrix

Vectors

- Mathematically, $\mathbf{x}_n$ and $\mathbf{y}_n$ are vectors and $\mathbf{W}$ is a matrix
- A **vector** is a combination of numbers representing a magnitude and a direction.They are defined by an origin and an endpoint.

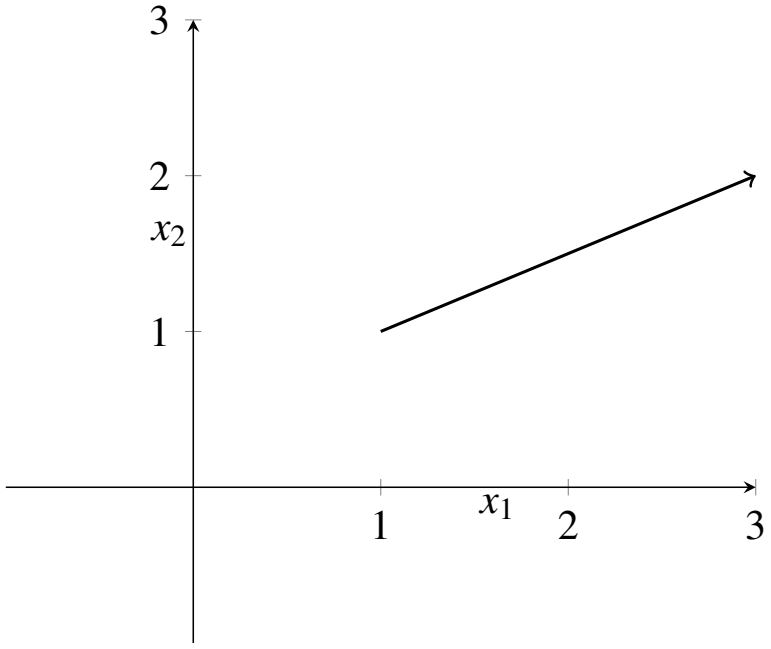
Vectors

- Mathematically, $\mathbf{x}_n$ and $\mathbf{y}_n$ are vectors and $\mathbf{W}$ is a matrix
- A **vector** is a combination of numbers representing a magnitude and a direction.They are defined by an origin and an endpoint.
- Example vector originating at coordinate $(x_1,x_2) = (1, 1)$ and ending at $(x_1,x_2) = (3, 2)$



Vectors

- Mathematically, $\mathbf{x}_n$ and $\mathbf{y}_n$ are vectors and $\mathbf{W}$ is a matrix
- A **vector** is a combination of numbers representing a magnitude and a direction.They are defined by an origin and an endpoint.
- Example vector originating at coordinate $(x_1,x_2) = (1,1)$ and ending at $(x_1,x_2) = (3,2)$



- When a vector is defined by $\mathbf{x} = (x_1, x_2, \cdots, x_N)$, it is implied that the origin is at $\mathbf{0} = (\underbrace{0, \cdots, 0}_{Nzeros})$.

Matrix Vector Multiplication

- Vectors of similar dimension can be added and subtracted from each other. In N dimensions:

$$\mathbf{y} - \mathbf{x} = (y_1 - x_1, y_2 - x_2, ..., y_N - x_N,)$$

Matrix Vector Multiplication

- Vectors of similar dimension can be added and subtracted from each other. In N dimensions:

$$\mathbf{y} - \mathbf{x} = (y_1 - x_1, y_2 - x_2, ..., y_N - x_N,)$$

- Adding a vector to another is equivalent to "concatenating" them. So if $\mathbf{x} = (2, 1)$ and $\mathbf{y} = (1, 2)$, then $\mathbf{z} = \mathbf{y} + \mathbf{x} = (3, 3)$.

Matrix Vector Multiplication

- Vectors of similar dimension can be added and subtracted from each other. In N dimensions:

$$\mathbf{y} - \mathbf{x} = (y_1 - x_1, y_2 - x_2, ..., y_N - x_N,)$$

- Adding a vector to another is equivalent to "concatenating" them. So if $\mathbf{x} = (2, 1)$ and $\mathbf{y} = (1, 2)$, then $\mathbf{z} = \mathbf{y} + \mathbf{x} = (3, 3)$.
- A **scalar** multiplication with a vector is (r is a real number):

$$r\mathbf{x} = (rx_1, rx_2)$$

Matrix Vector Multiplication

- Vectors of similar dimension can be added and subtracted from each other. In N dimensions:

$$\mathbf{y} - \mathbf{x} = (y_1 - x_1, y_2 - x_2, ..., y_N - x_N,)$$

- Adding a vector to another is equivalent to "concatenating" them. So if $\mathbf{x} = (2, 1)$ and $\mathbf{y} = (1, 2)$, then $\mathbf{z} = \mathbf{y} + \mathbf{x} = (3, 3)$.
- A **scalar** multiplication with a vector is (r is a real number):

$$r\mathbf{x} = (rx_1, rx_2)$$

- The length of a vector (called the **norm**) is the distance between the origin and the end point:

$$||\mathbf{x}|| = \sqrt{x_1^2 + ... + x_N^2} = \sqrt{\sum_{i=1}^{N} x_i^2}$$

Dot products

- The **dot product** of two vectors is defined as:

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{N} x_i y_i$$

Dot products

- The **dot product** of two vectors is defined as:

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{N} x_i y_i$$

- The dot product of the vector itself is the square of its norm:

$$\mathbf{x} \cdot \mathbf{x} = \sum_{i=1}^{N} x_i x_i = ||\mathbf{x}||^2$$

Dot products

- The **dot product** of two vectors is defined as:

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{N} x_i y_i$$

- The dot product of the vector itself is the square of its norm:

$$\mathbf{x} \cdot \mathbf{x} = \sum_{i=1}^{N} x_i x_i = ||\mathbf{x}||^2$$

- The dot product quantifies the "projection" of one vector to another, such that:

$$\mathbf{x} \cdot \mathbf{y} = ||\mathbf{x}||||\mathbf{y}|| \cos(\theta)$$

where θ is the angle between the two vectors.

Matrices

- A **Matrix** is a collection of vectors. The following defines a $N \times M$ matrix. N is the number of rows, and M is the number of columns.

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1M} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2M} \\ \dots & \dots & \dots & \dots & \dots \\ A_{N1} & A_{N2} & A_{N3} & \dots & A_{NM} \end{bmatrix}$$

Matrices

- A **Matrix** is a collection of vectors. The following defines a $N \times M$ matrix. N is the number of rows, and M is the number of columns.

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1M} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2M} \\ \dots & \dots & \dots & \dots & \dots \\ A_{N1} & A_{N2} & A_{N3} & \dots & A_{NM} \end{bmatrix}$$

- A scale is a special case with $N = 1, M = 1$.
- A vector is a special case with $N = 1, M > 1$ (row vector) or $N > 1, M = 1$ (column vector).

Matrices

- A **Matrix** is a collection of vectors. The following defines a $N \times M$ matrix. N is the number of rows, and M is the number of columns.

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1M} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2M} \\ \dots & \dots & \dots & \dots & \dots \\ A_{N1} & A_{N2} & A_{N3} & \dots & A_{NM} \end{bmatrix}$$

- A scale is a special case with $N = 1, M = 1$.
- A vector is a special case with $N = 1, M > 1$ (row vector) or $N > 1, M = 1$ (column vector).
- The generalization of matrices to higher dimensions are called tensors

Matrices

- A **Matrix** is a collection of vectors. The following defines a $N \times M$ matrix. N is the number of rows, and M is the number of columns.

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1M} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2M} \\ \dots & \dots & \dots & \dots & \dots \\ A_{N1} & A_{N2} & A_{N3} & \dots & A_{NM} \end{bmatrix}$$

- A scale is a special case with $N = 1, M = 1$.
- A vector is a special case with $N = 1, M > 1$ (row vector) or $N > 1, M = 1$ (column vector).
- The generalization of matrices to higher dimensions are called tensors
- Matrices are often (but not always) denoted by uppercase letters. The element of a matrix is A_{ij}

Matrices

- A **Matrix** is a collection of vectors. The following defines a $N \times M$ matrix. N is the number of rows, and M is the number of columns.

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1M} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2M} \\ \dots & \dots & \dots & \dots & \dots \\ A_{N1} & A_{N2} & A_{N3} & \dots & A_{NM} \end{bmatrix}$$

- A scale is a special case with $N = 1, M = 1$.
- A vector is a special case with $N = 1, M > 1$ (row vector) or $N > 1, M = 1$ (column vector).
- The generalization of matrices to higher dimensions are called tensors
- Matrices are often (but not always) denoted by uppercase letters. The element of a matrix is A_{ij}
- In Python, the `np.array` function builds vectors and matrices

Basic Operations with Matrices

- **Addition/Subtraction** of matrices: The matrix $C = A + B$ has elements $C_{ij} = A_{ij} + B_{ij}$. The sum of any $N \times M$ matrices is a matrix of size $N \times M$.

  Matrix addition is made element-wise

  .

- **Multiplication** of matrices is a generalization of the dot product. To multiply matrices, the dimensionalities much match as follows

$$\underbrace{C}_{M\times P} = \underbrace{A}_{M\times N} \cdot \underbrace{B}_{N\times P}$$

  The number of columns of A must match the number of rows of B

  .

- A matrix can be multiplied by a vector:

$$\underbrace{\mathbf{c}}_{N\times 1} = \underbrace{A}_{N\times M} \cdot \underbrace{\mathbf{b}}_{M\times 1}$$

The result is a vector.

$$\mathbf{c} = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1M} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2M} \\ \dots & \dots & \dots & \dots & \dots \\ A_{N1} & A_{N2} & A_{N3} & \dots & A_{NM} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_M \end{bmatrix}$$

Matrix Vector Multiplication

- Each component of $\mathbf{c}$ is equal to a dot product between rows of $\mathbf{A}$ and columns of $\mathbf{B}$ (here we have only one column). For the first component of the vector $\mathbf{c}$ is c_1:

$$c_1 = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1M} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_M \end{bmatrix}$$

$$= A_{11}b_1 + A_{12}b_2 + \cdots + A_{1M}b_M$$

Matrix Vector Multiplication

- Each component of $\mathbf{c}$ is equal to a dot product between rows of $\mathbf{A}$ and columns of $\mathbf{B}$ (here we have only one column). For the first component of the vector $\mathbf{c}$ is c_1:

$$c_1 = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1M} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_M \end{bmatrix}$$

$$= A_{11}b_1 + A_{12}b_2 + \cdots + A_{1M}b_M$$

- For all components of $\mathbf{c}$:

$$c_i = A_{i1}b_1 + A_{i2}b_2 + \cdots + A_{iM}b_M$$

Matrix Vector Multiplication

- Each component of $\mathbf{c}$ is equal to a dot product between rows of $\mathbf{A}$ and columns of $\mathbf{B}$ (here we have only one column). For the first component of the vector $\mathbf{c}$ is c_1:

$$c_1 = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1M} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_M \end{bmatrix}$$

$$= A_{11}b_1 + A_{12}b_2 + \cdots + A_{1M}b_M$$

- For all components of $\mathbf{c}$:

$$c_i = A_{i1}b_1 + A_{i2}b_2 + \cdots + A_{iM}b_M$$

- Written compactly

$$c_i = \sum_{j=1}^{M} A_{ij}b_j$$

Matrix Multiplication

- The multiplication of two matrices $\underbrace{C}_{N\times P} = \underbrace{A}_{N\times M} \cdot \underbrace{B}_{M\times P}$ is defined in a similar manner:

$$C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} + \cdots + A_{iM}B_{Mj}$$

Matrix Multiplication

- The multiplication of two matrices $\underbrace{C}_{N\times P} = \underbrace{A}_{N\times M} \cdot \underbrace{B}_{M\times P}$ is defined in a similar manner:

$$C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} + \cdots + A_{iM}B_{Mj}$$

- Written compactly

$$c_{ij} = \sum_{k=1}^{M} A_{ik}B_{kj}$$

Matrix Multiplication Example

Multiply the following two matrices

$$A = \begin{bmatrix} 1 & 3 & 5 \\ -2 & 4 & -6 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 2 \end{bmatrix}$$

Calculate $C = AB$

Properties of matrices

- $A + B = B + A$, A, B are $M \times N$
- $(A + B) + C = A + (B + C)$, A, B, C are $M \times N$
- $A(B + C) = AB + AC$
- $\alpha(A + B) = \alpha A + \alpha B$, α is a scalar

Data Matrix

D observations of M variables can be represented as a $M \times D$ matrix:

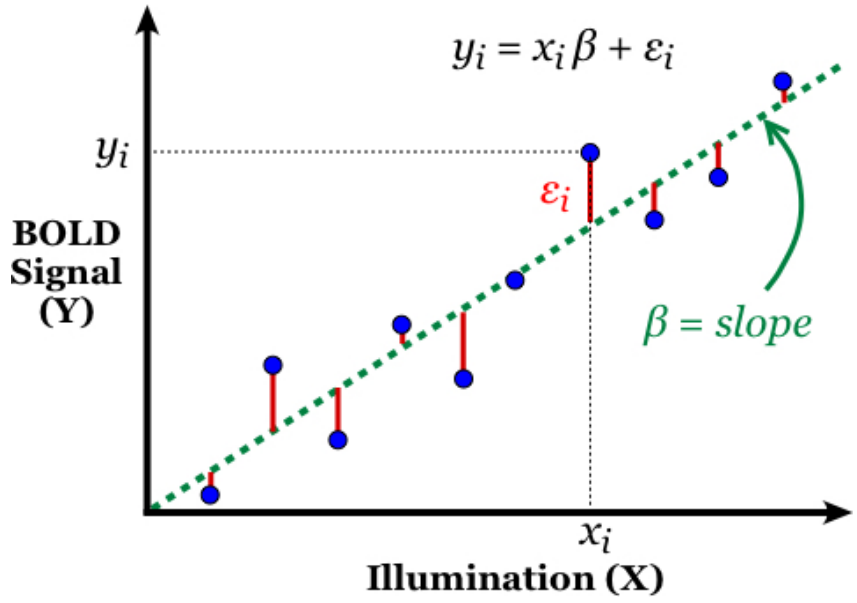
$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1D} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2D} \\ \dots & \dots & \dots & \dots & \dots \\ x_{M1} & x_{M2} & x_{M3} & \dots & x_{MD} \end{bmatrix}$$

X can be written as D measurements of P variables/dimensions (*i.e.* P column vectors):

$$X = \begin{bmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_p \end{bmatrix}$$

where $\mathbf{x}_i$ is the $D \times 1$ vector of D observations of the i^{th} dimension.

Back to our Linear Model



For a sample n ($n = 1, \ldots D$):

$$\mathbf{y}_n = \mathbf{W}\mathbf{x}_n \tag{6}$$

$\mathbf{y}_n$ is a N vector, $\mathbf{W}$ is a $N \times M$ matrix and $\mathbf{x}_n$ is a M vector

The goal of linear regression is to find W such that the following mean-square error (MSE) cost function is minimized:

$$C_{\text{MSE}} = \frac{1}{2D} \sum_{n \in \text{training set}} (\mathbf{y}(\mathbf{x}_n) - \hat{\mathbf{y}}_n)^2. \tag{7}$$

$$C_{\text{MSE}} = \frac{1}{2} \sum_{n=1}^{D} ||\mathbf{y}_n - \hat{\mathbf{y}}_n||^2$$

$\hat{\mathbf{y}}_n$ are the targets.
(MSE is similar to RMSE, except that there is no $\sqrt{}$. For optimization, MSE is identical to RMSE)

We already know how to solve this using basic parameter estimation.

Intermediate Summary

- Machine learning and neural networks can be viewed as models for understanding cognition

Intermediate Summary

- Machine learning and neural networks can be viewed as models for understanding cognition
- The linear model seen earlier can be cast as a neural network $\mathbf{y}_n = \mathbf{W}\mathbf{x}_n$

Intermediate Summary

- Machine learning and neural networks can be viewed as models for understanding cognition
- The linear model seen earlier can be cast as a neural network $\mathbf{y}_n = \mathbf{W}\mathbf{x}_n$
- The operations in the linear model correspond to matrix-vector multiplication

Intermediate Summary

- Machine learning and neural networks can be viewed as models for understanding cognition
- The linear model seen earlier can be cast as a neural network $\mathbf{y}_n = \mathbf{W}\mathbf{x}_n$
- The operations in the linear model correspond to matrix-vector multiplication
- Vectors are a combination of numbers representing a magnitude and a direction

Intermediate Summary

- Machine learning and neural networks can be viewed as models for understanding cognition
- The linear model seen earlier can be cast as a neural network $\mathbf{y}_n = \mathbf{W}\mathbf{x}_n$
- The operations in the linear model correspond to matrix-vector multiplication
- Vectors are a combination of numbers representing a magnitude and a direction
- Matrices are “stacked vectors”

Intermediate Summary

- Machine learning and neural networks can be viewed as models for understanding cognition
- The linear model seen earlier can be cast as a neural network $\mathbf{y}_n = \mathbf{W}\mathbf{x}_n$
- The operations in the linear model correspond to matrix-vector multiplication
- Vectors are a combination of numbers representing a magnitude and a direction
- Matrices are “stacked vectors”
- Vectors can be multiplied using the dot product $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{N} x_i y_i$

Intermediate Summary

- Machine learning and neural networks can be viewed as models for understanding cognition
- The linear model seen earlier can be cast as a neural network $\mathbf{y}_n = \mathbf{W}\mathbf{x}_n$
- The operations in the linear model correspond to matrix-vector multiplication
- Vectors are a combination of numbers representing a magnitude and a direction
- Matrices are “stacked vectors”
- Vectors can be multiplied using the dot product $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{N} x_i y_i$
- Matrix-vector or Matrix-Matrix multiplications are generalizations of the dot product

Intermediate Summary

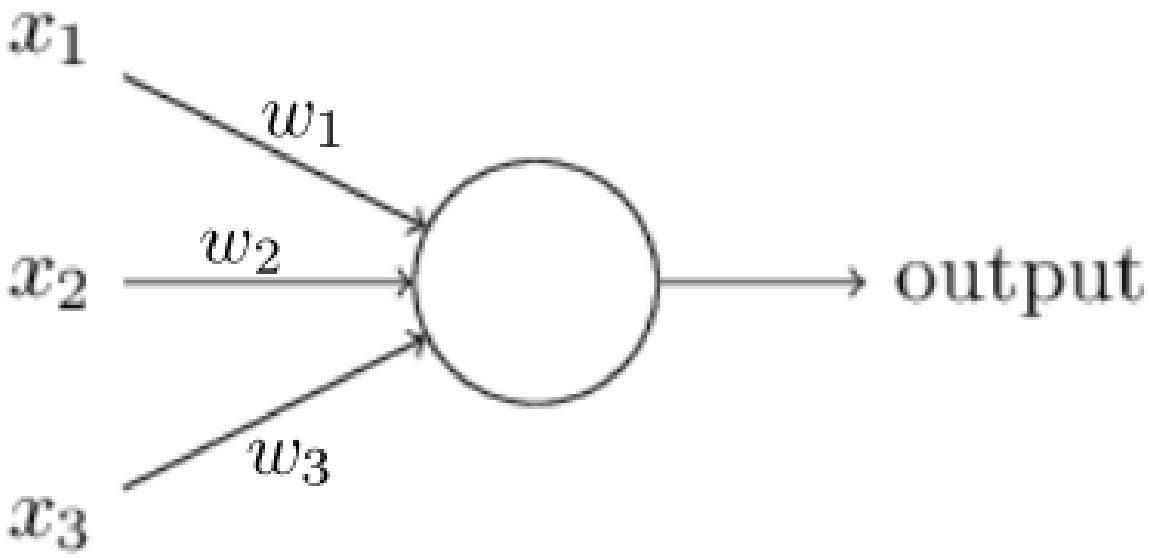
- Machine learning and neural networks can be viewed as models for understanding cognition
- The linear model seen earlier can be cast as a neural network $\mathbf{y}_n = \mathbf{W}\mathbf{x}_n$
- The operations in the linear model correspond to matrix-vector multiplication
- Vectors are a combination of numbers representing a magnitude and a direction
- Matrices are “stacked vectors”
- Vectors can be multiplied using the dot product $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{N} x_i y_i$
- Matrix-vector or Matrix-Matrix multiplications are generalizations of the dot product
- The norm of a vector is its length

Intermediate Summary

- Machine learning and neural networks can be viewed as models for understanding cognition
- The linear model seen earlier can be cast as a neural network $\mathbf{y}_n = \mathbf{W}\mathbf{x}_n$
- The operations in the linear model correspond to matrix-vector multiplication
- Vectors are a combination of numbers representing a magnitude and a direction
- Matrices are “stacked vectors”
- Vectors can be multiplied using the dot product $\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{N} x_i y_i$
- Matrix-vector or Matrix-Matrix multiplications are generalizations of the dot product
- The norm of a vector is its length
- The Root Mean Square Error is the norm of the error vector defined by $\mathbf{y}_n - \hat{\mathbf{y}}_n$

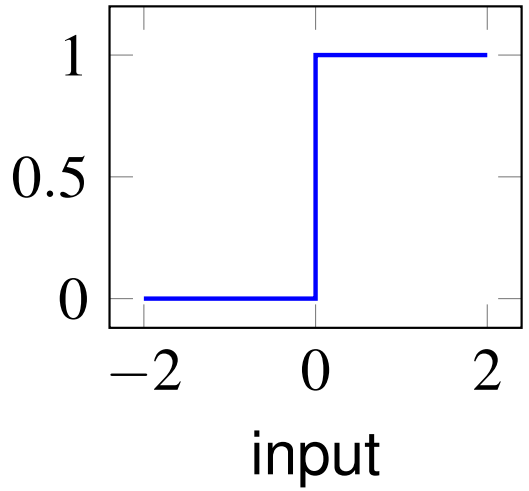
Neural Networks for Classification

The Perceptron



- Inputs $\mathbf{x} = (x_1, x_2, x_3)$ with weights $\mathbf{w} = (w_1, w_2, w_3)$, and bias b

$$\text{output} = \begin{cases} -1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$



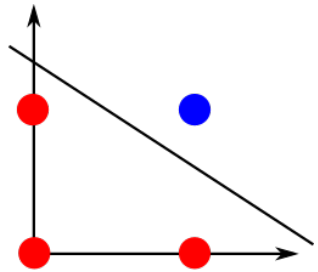
A Perceptron implementing the AND

Example: two inputs x_1 and x_2 that can take values -1 or 1. Find their weights w_1 and w_2 such that the output is 1 only when $x_1 = 1$ and $x_2 = 1$.

Linear separability

A perceptron is equivalent to a decision boundary.

- A straight line can separate blue vs. red = Linearly Separable



The Perceptron Learning Rule

The parameters of a Perceptron can be learning automatically.

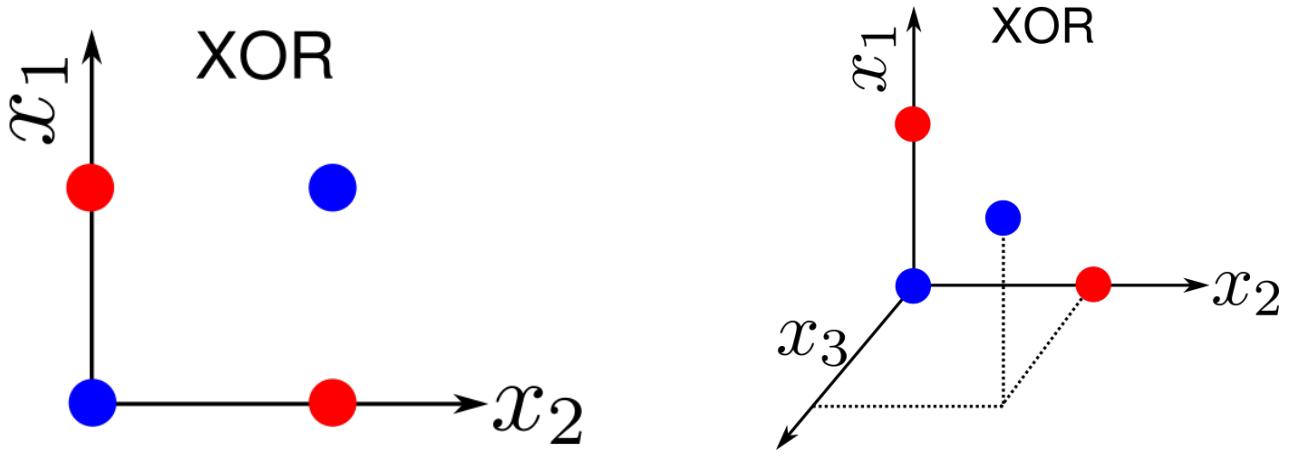
Error = Number of Misclassified Samples

To minimize error, repeat for every data sample:

$$\text{new } w_i = w_i + \eta(\text{target} - \text{output})x_i \quad \text{for every i,}$$
$$\text{new } b = b + \eta(\text{target} - \text{output}),$$

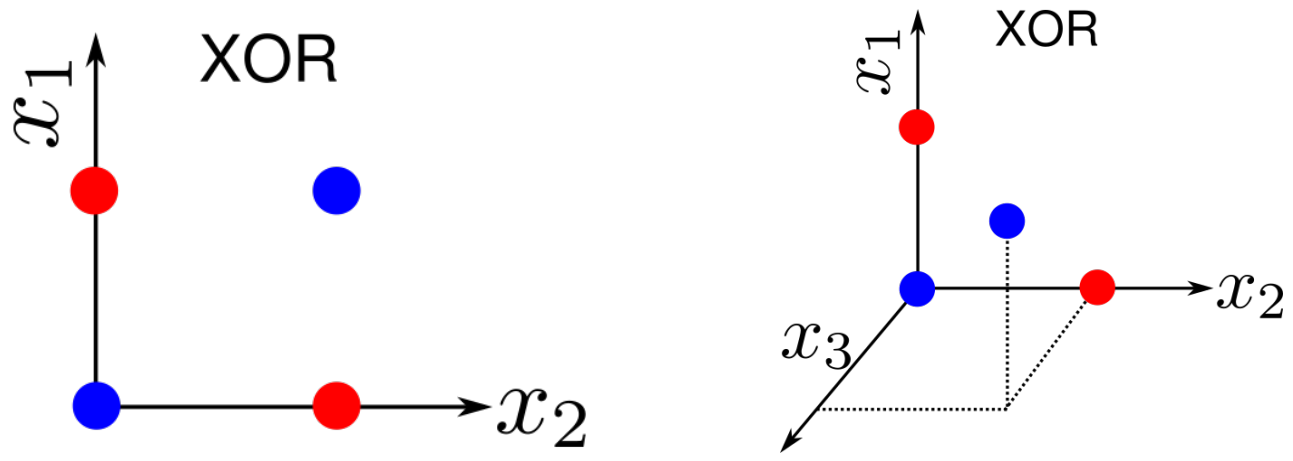
where η is a "learning rate".

Non linearly separable problems can be solved with an intermediate perceptron



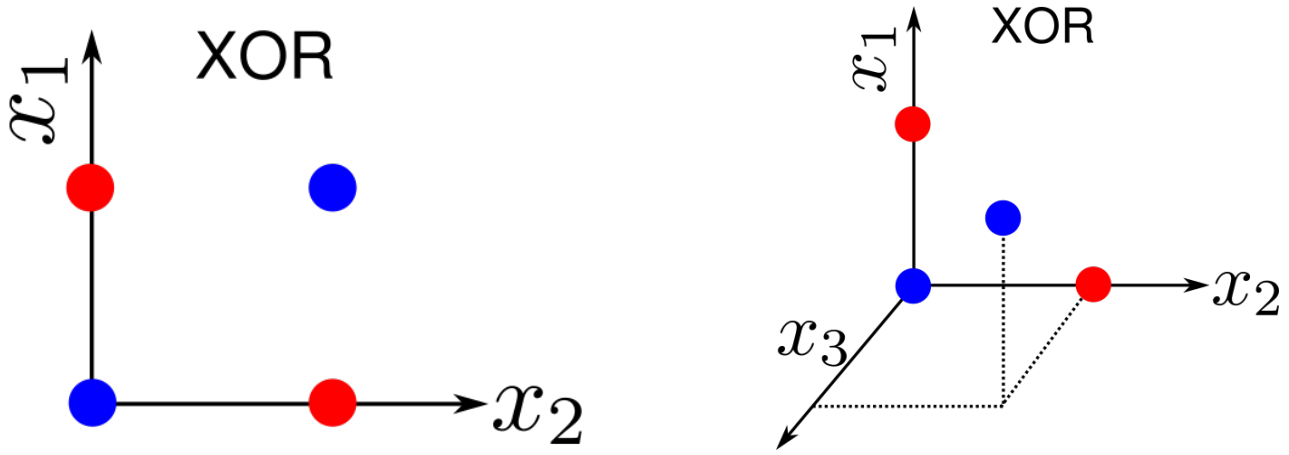
- We need an intermediate unit that is on only when x_1 and x_2 are both on.

Non linearly separable problems can be solved with an intermediate perceptron

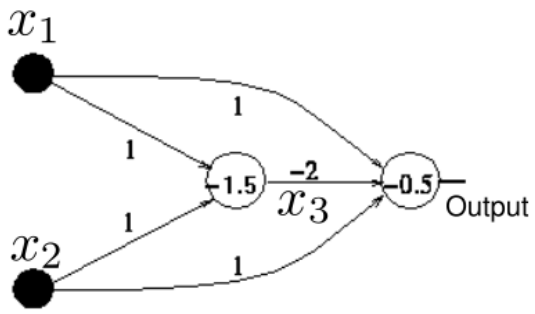


- We need an intermediate unit that is on only when x_1 and x_2 are both on.
- XOR gate with two perceptrons

Non linearly separable problems can be solved with an intermediate perceptron



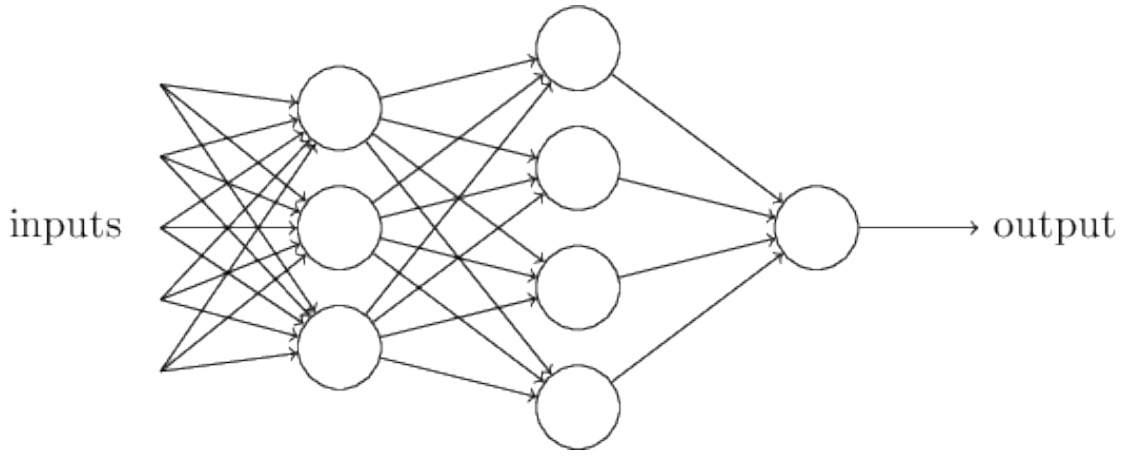
- We need an intermediate unit that is on only when x_1 and x_2 are both on.
- XOR gate with two perceptrons



The strategy of extending networks with intermediate units is

Network of Perceptrons

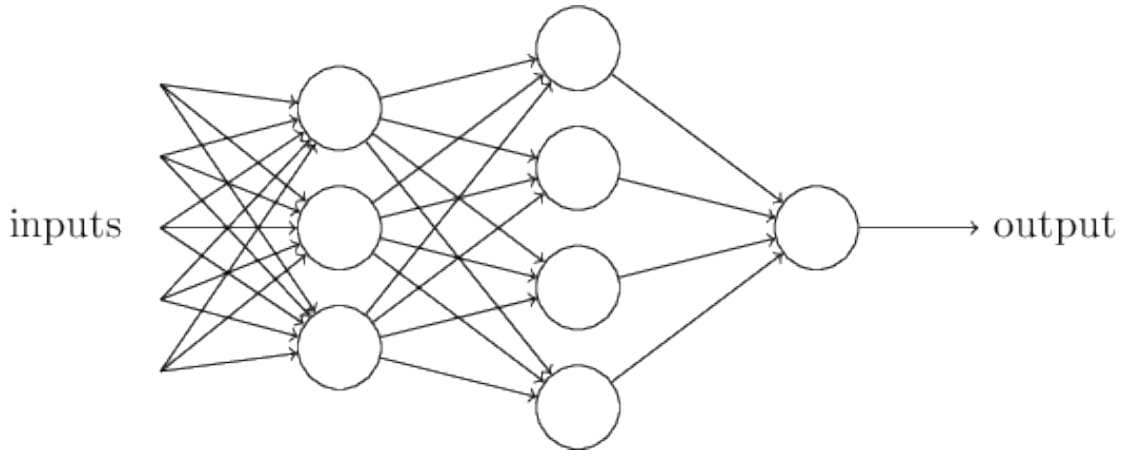
Systematically building intermediate layers.



- Each layer makes a decision based on previous inputs
- Subsequent layers make decisions based on more and more abstract information

Network of Perceptrons

Systematically building intermediate layers.

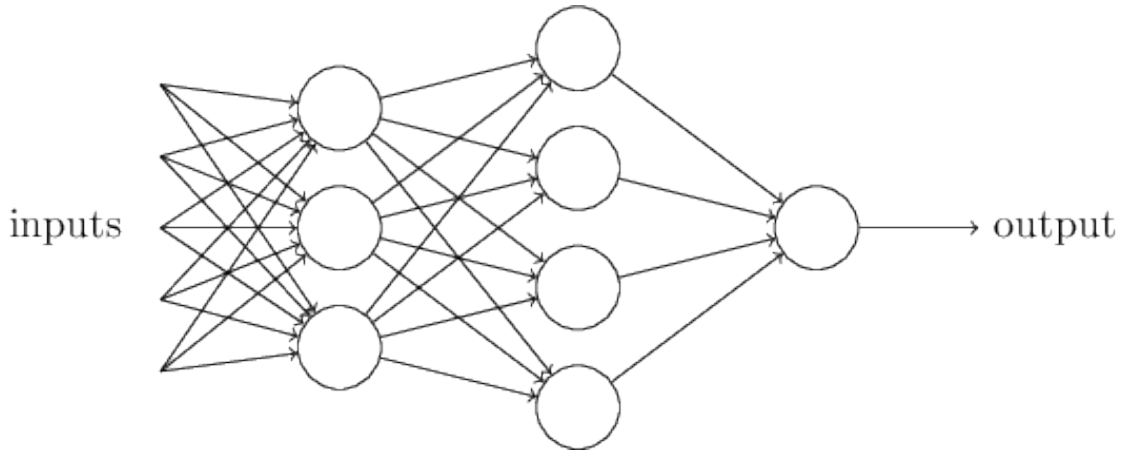


- Each layer makes a decision based on previous inputs
- Subsequent layers make decisions based on more and more abstract information

Weights and biases can be learning using gradient-based optimization.

Network of Perceptrons

Systematically building intermediate layers.



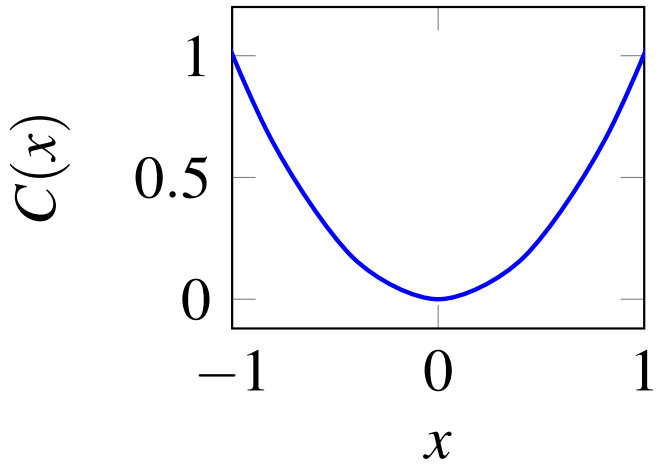
- Each layer makes a decision based on previous inputs
- Subsequent layers make decisions based on more and more abstract information

Weights and biases can be learning using gradient-based optimization.

Gradient-descent in multilayer neural networks is the Gradient backpropagation algorithm.

Minimizing Arbitrary Functions by Gradient Descent

Example: Find x that minimizes $C(x) = x^2$



Incremental change in Δx:

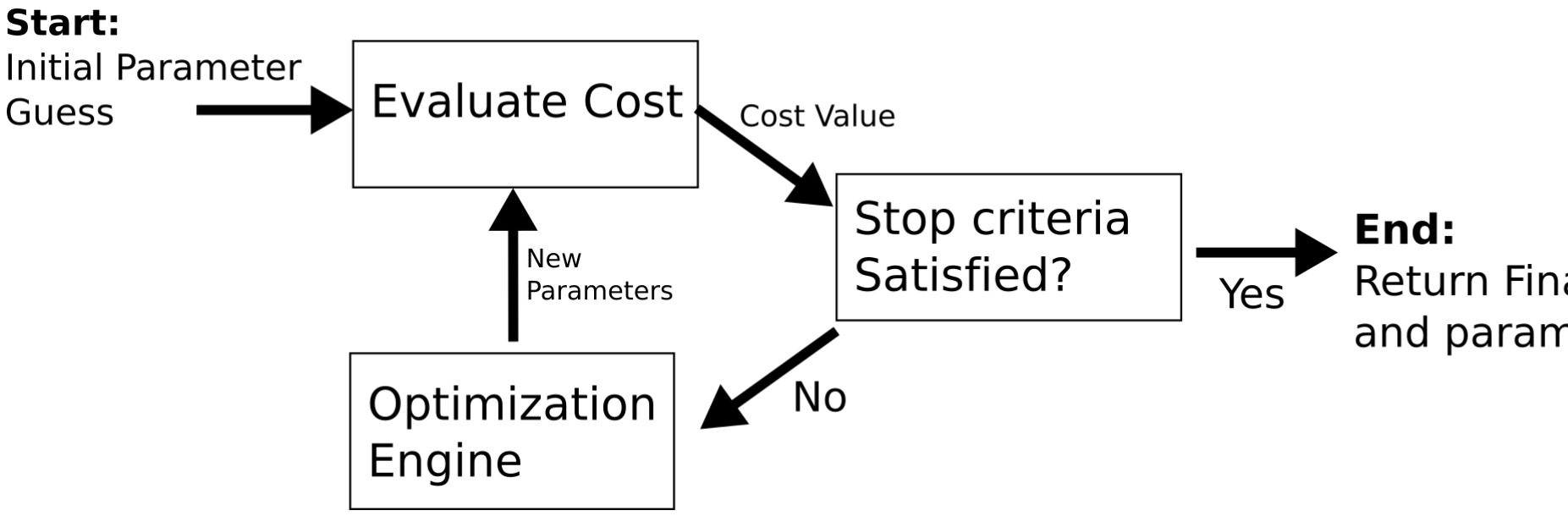
$$\Delta C = \underbrace{\frac{\partial C}{\partial x}}_{\text{=Slope of } C(x)} \Delta x \tag{8}$$

With $\Delta x = -\eta \frac{\partial C}{\partial x}$, $\Delta C = -\eta \left(\frac{\partial C}{\partial x}\right)^2$

Gradient Descent for finding the optimal x

$$\text{new } x = \text{old } x - \eta \frac{\partial C}{\partial x} \tag{9}$$

Training Neural Networks

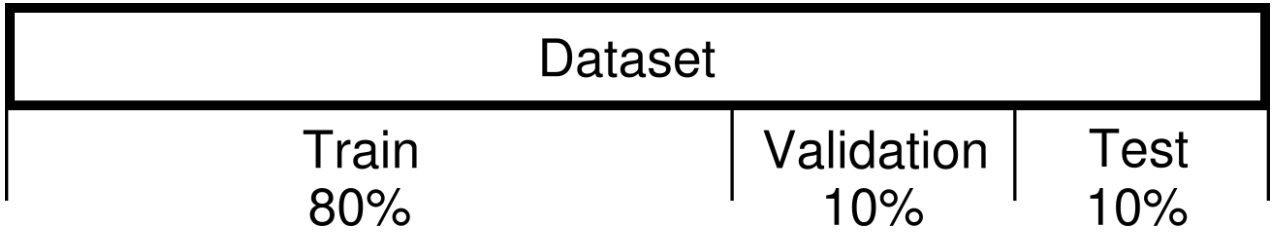


1. Define a neural network model, e.g.

$$\mathbf{y}_n = \sigma(\mathbf{W} \cdot \mathbf{x}_n)$$

2. Define a cost function, e.g.
   $C_{\text{MSE}}(w, b) = \frac{1}{2} \sum_n \sum_k (y_{kn} - \hat{y}_{kn})^2$
3. Apply a gradient descent iteratively update until convergence

Training, Testing and Validation Datasets



- Training set: Apply learning rule to sampling in dataset
- Testing set: Set that is never used during training to test the classifier
- Validation set: Set for monitoring overfitting and testing algorithm with different learning parameters

Demo

http://playground.tensorflow.org/

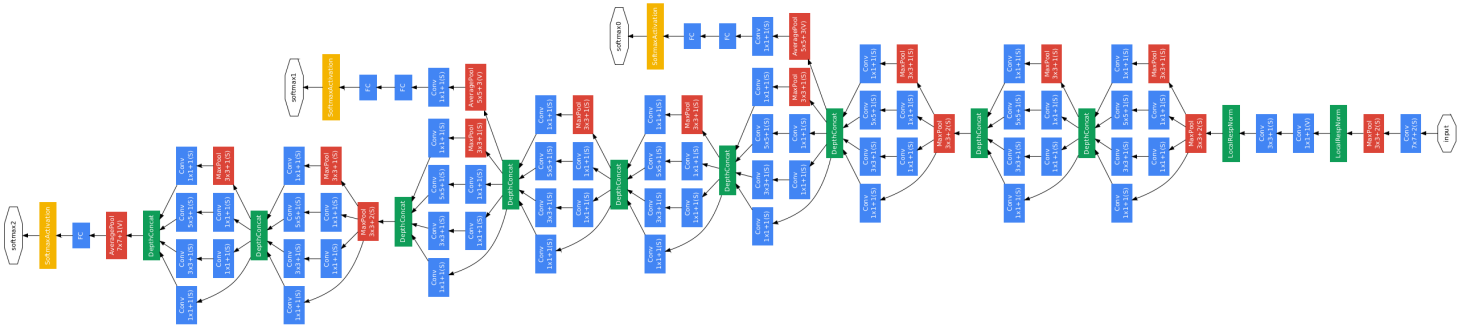
Deep neural networks

Before:

> *"How many hidden layers and how many units per layer do we need? The answer is at most two"*

Hertz, Krogh, and Palmer,, 1991

Now:



Szegedy et al., *arXiv preprint arXiv:1409.4842*, 2014

What happened between 1990s and now?

Good old online backpropagation for plain multilayer perceptrons yields a very low 0.35% error rate on the MNIST handwritten digits benchmark. All we need to achieve this best result so far are many hidden layers, many neurons per layer, numerous deformed training images to avoid overfitting, and graphics cards to greatly speed up learning.

Cireşan, Meier, Gambardella, and Schmidhuber, *Neural computation*, 2010

Better hardware, bigger data and tricks!

Convolutional Neural Networks: Neural Networks for Vision

Challenges in computer vision

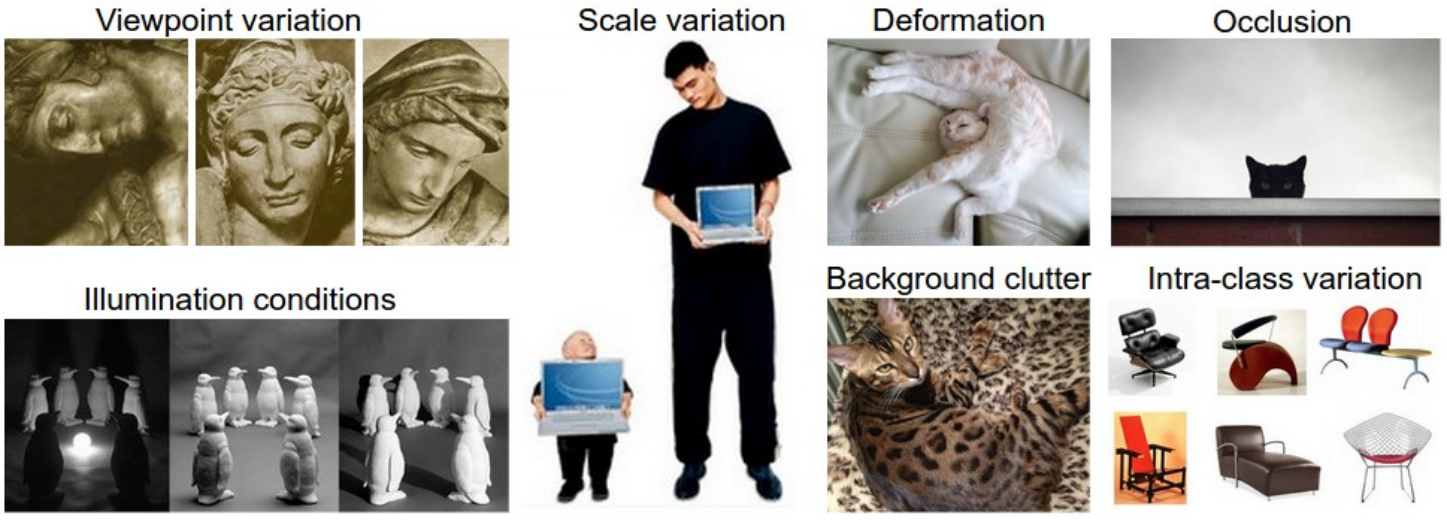
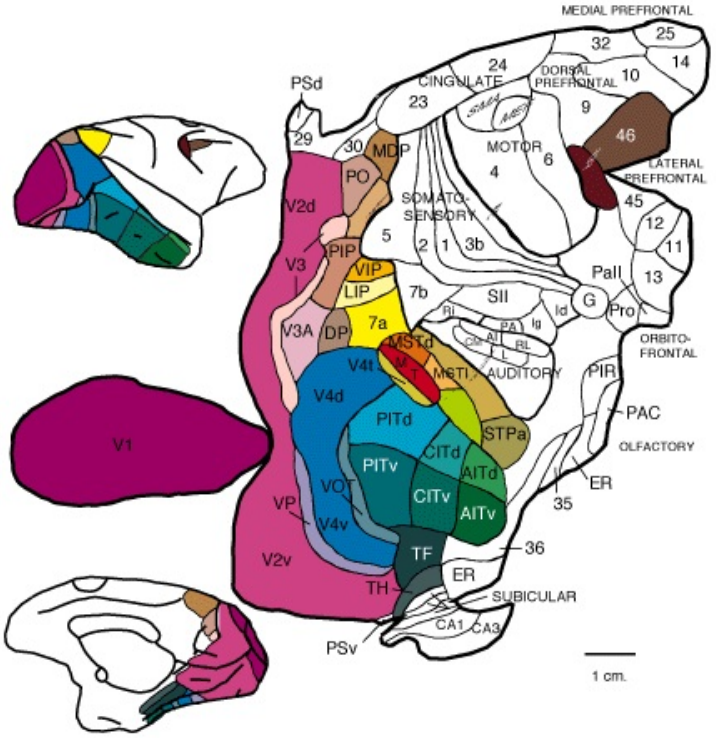
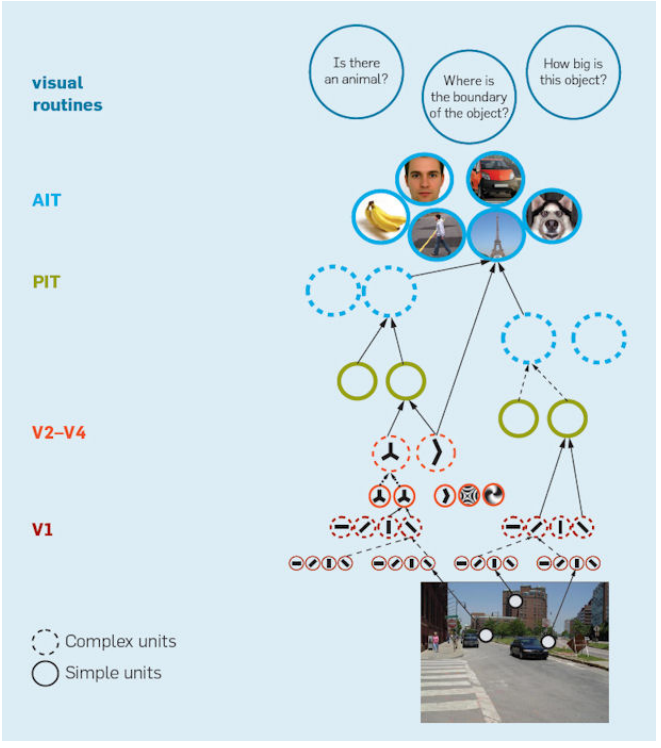


Image from Stanford CS231n Convolutional Neural Networks for Visual Recognition Class

Hierarchical Organization of the Visual Pathway

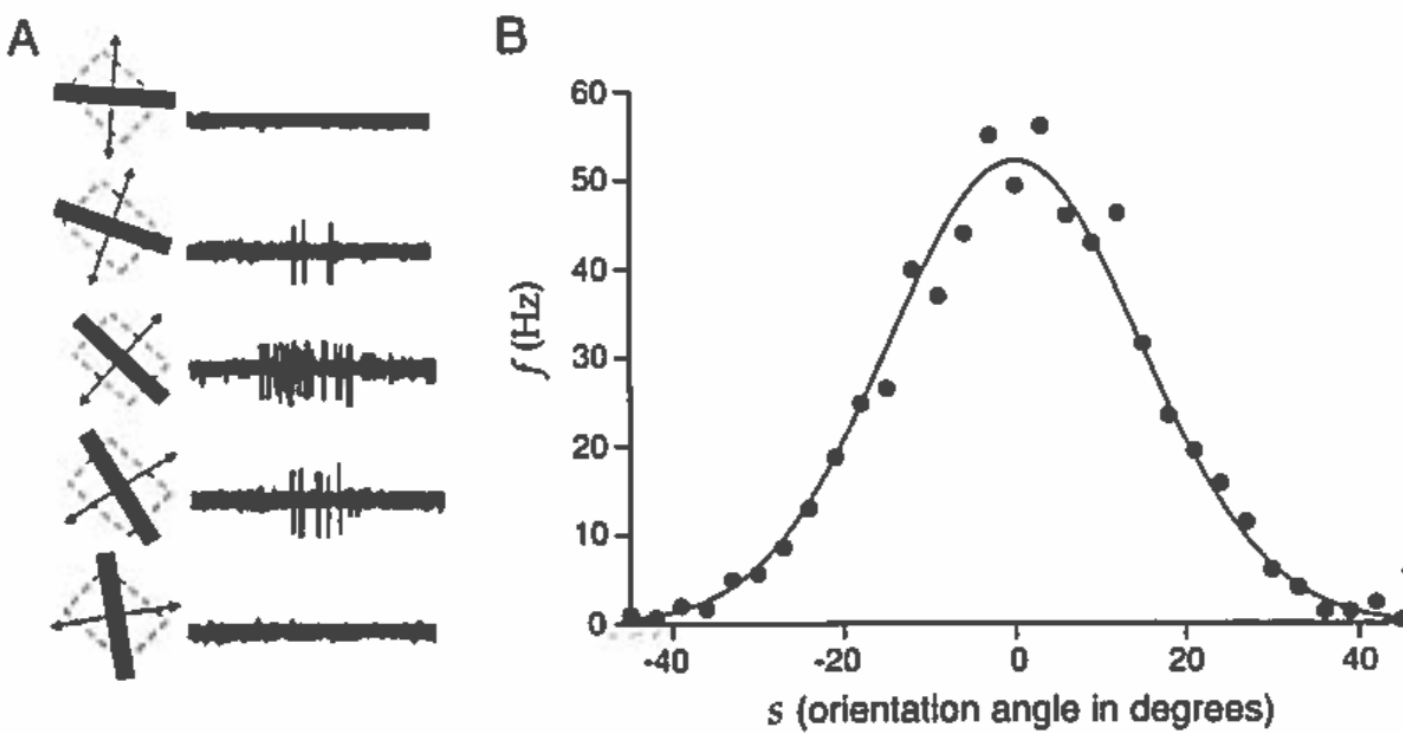




Felleman and Van Essen, 1991 (left), Cerebral Cortex 1:1-47. Serre and Poggio, 2007 (right)

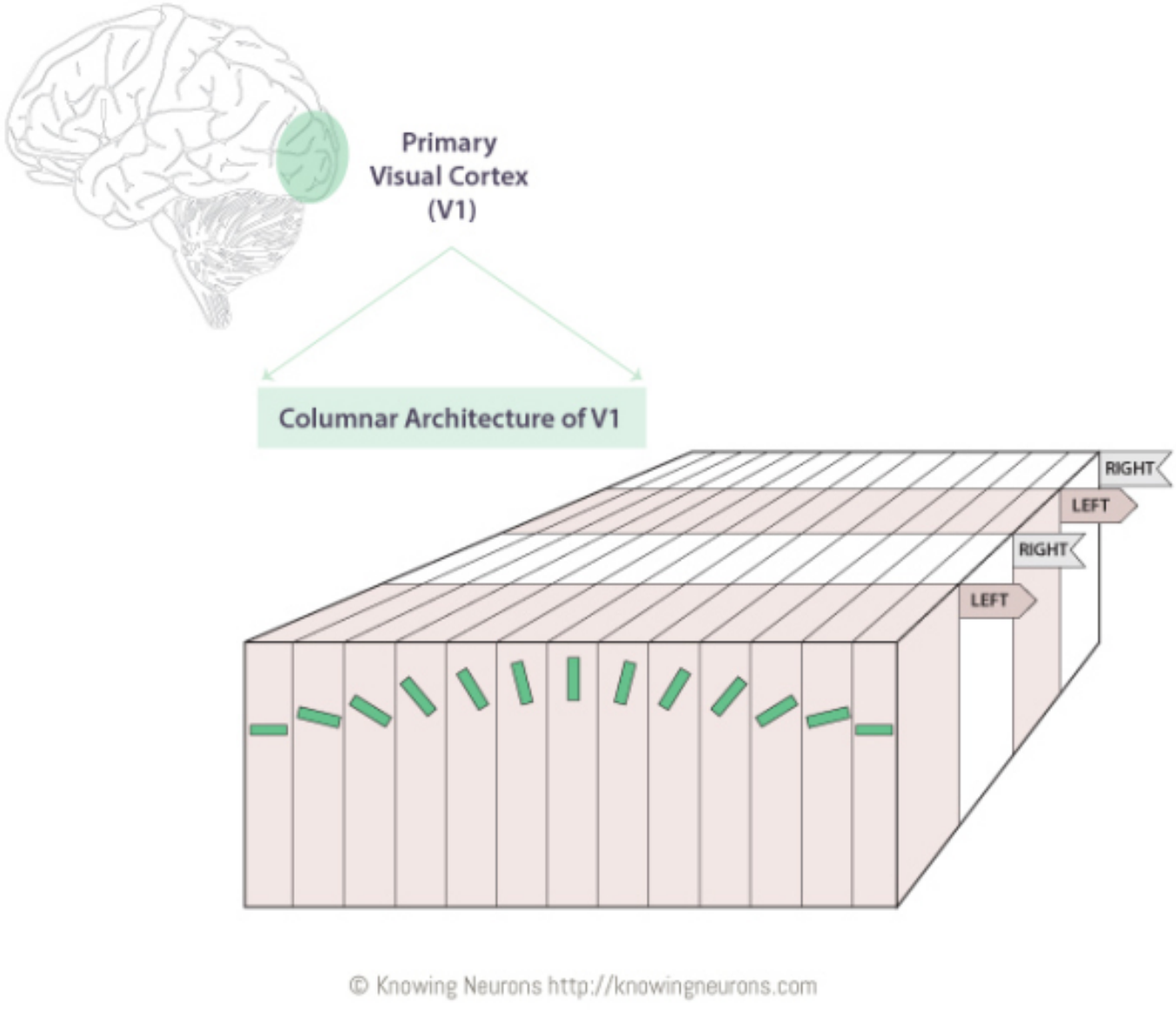
Neurons higher in the hierarchy represent more abstract features

Tuning Curves

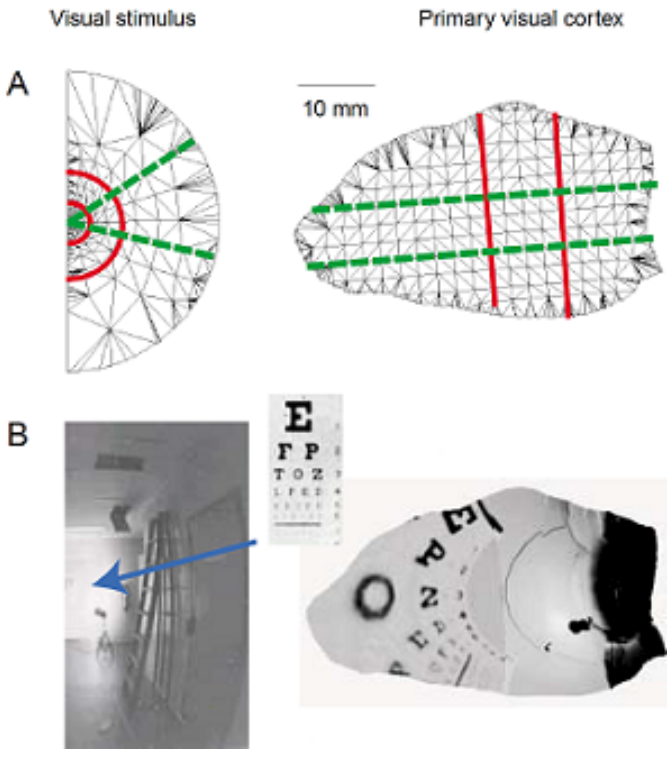


Hubel & Wiesel, 1968

Receptive Fields



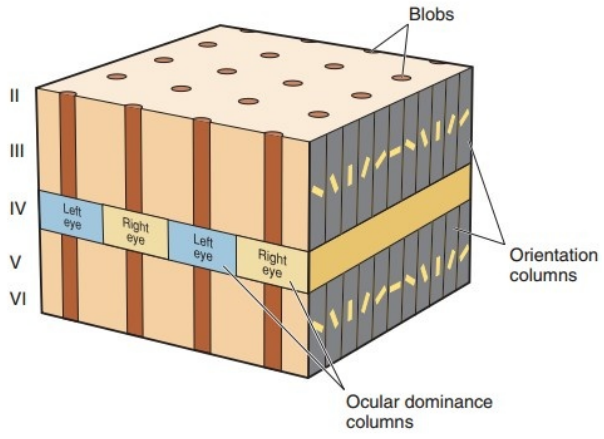
Retinotopic Map

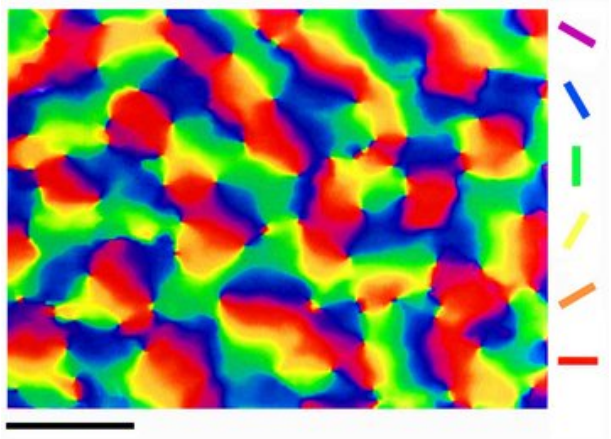


Matteo Carandini (2012), Scholarpedia, 7(7):12105

Nearby points in visual field project to nearby neurons in V1

Columnar Organization of V1

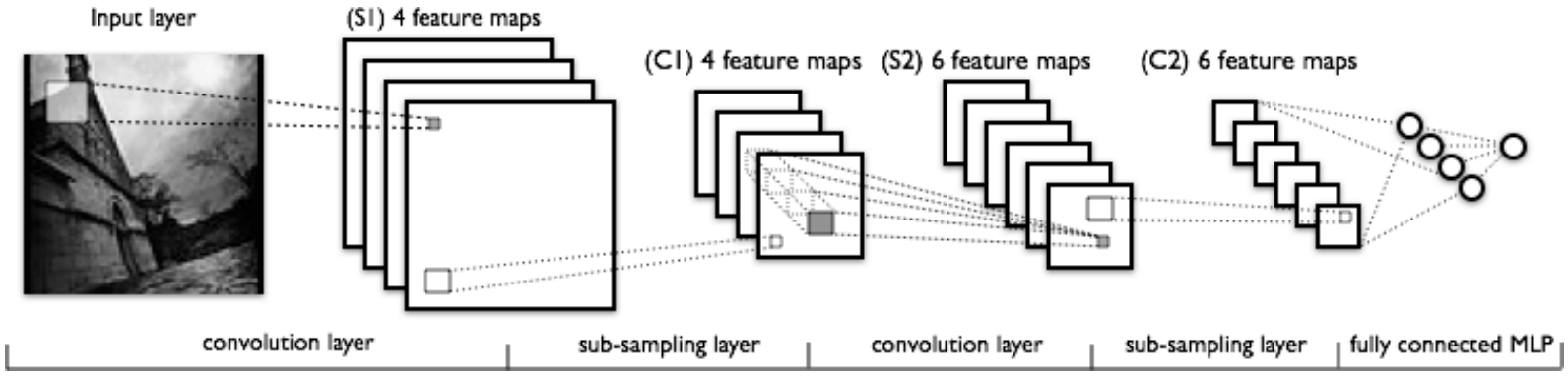




Right: Blasdel & Salama (1986)

The receptive fields are tiled to cover the entire visual field

Convolutional Neural Networks

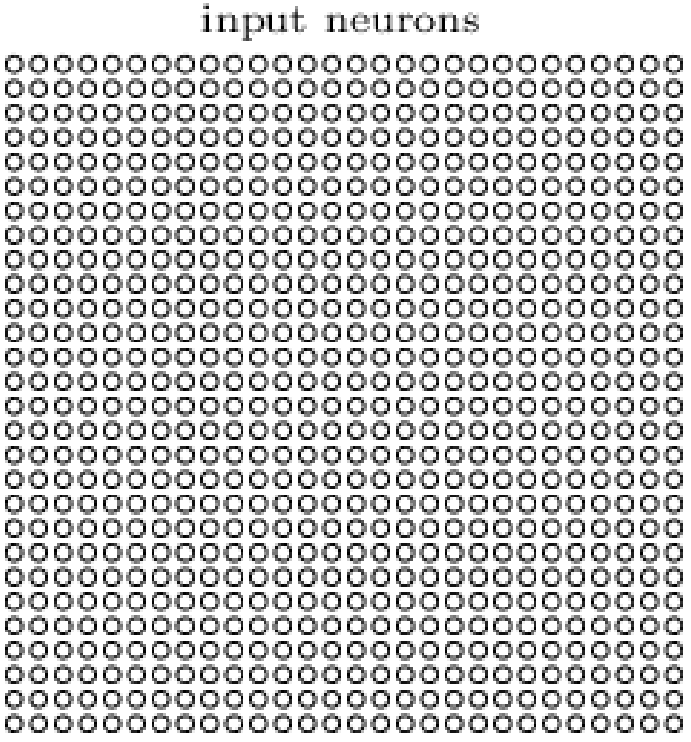


The visual cortex and neural networks solve the same task: use **retinotopy**, **local receptive fields** and **hierarchy** to constrain fully connected neural networks.

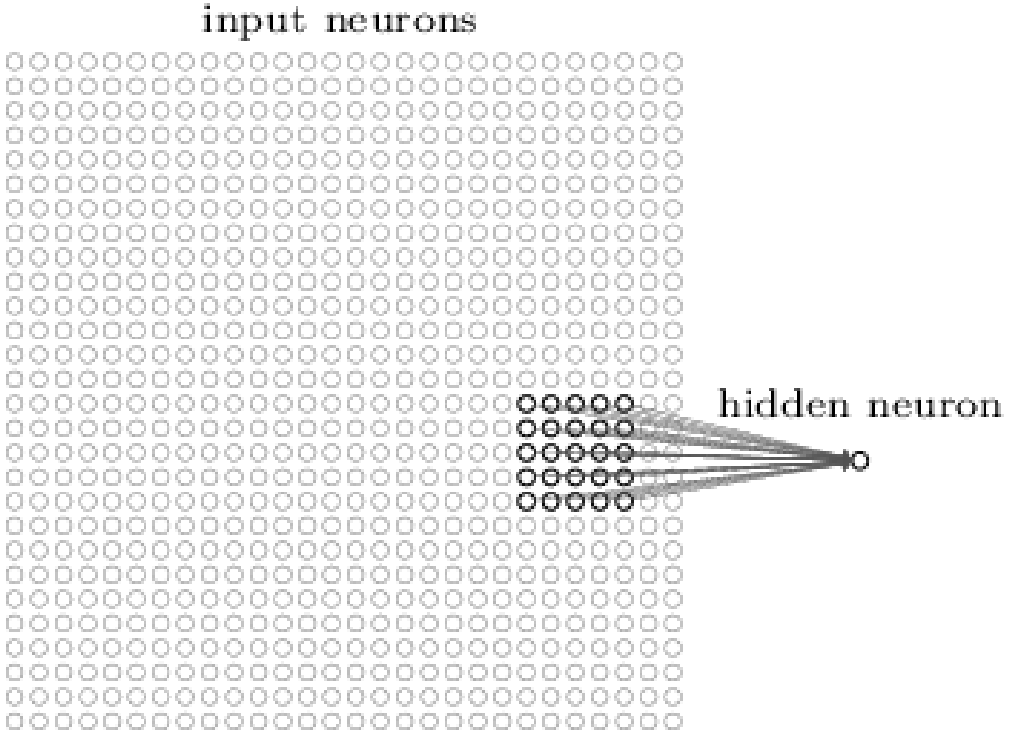
Two new type of layers:

- Convolutions
- Sub-sampling layers (pooling)

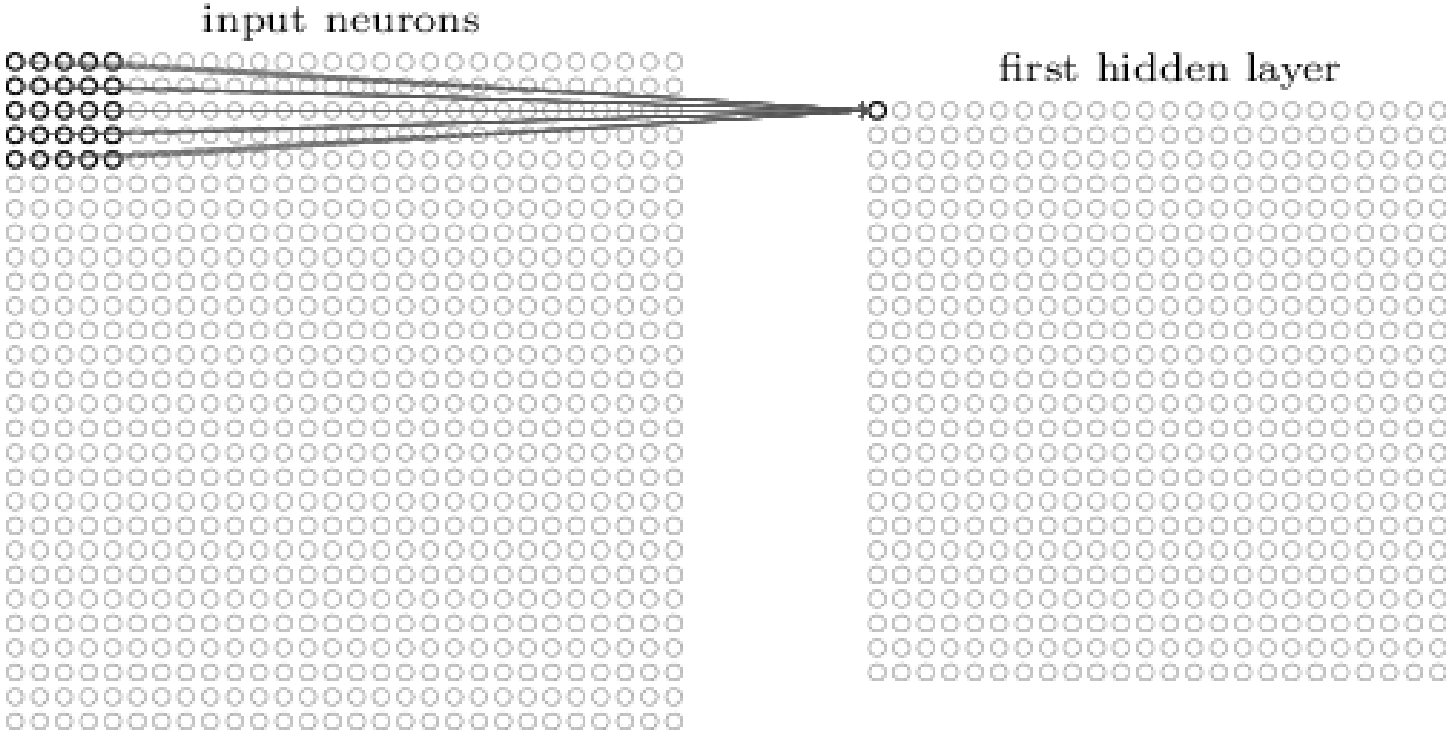
Convolution Layer



Convolution Layer

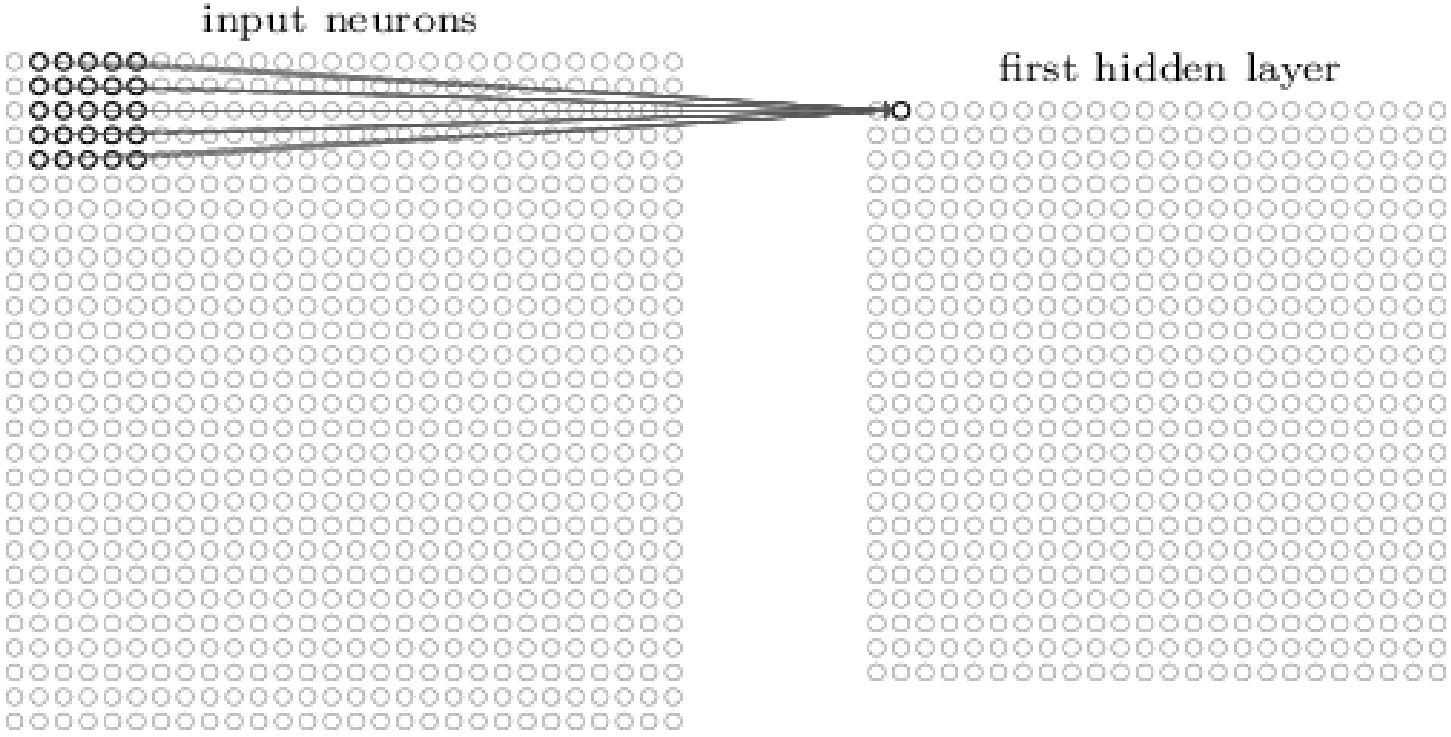


Convolution Layer



$$\text{output hidden unit (0,0)} = \sigma\left(b_0 + \sum_{l=0}^{5}\sum_{m=0}^{5} w_{lm} input_{0+l,0+m}\right).$$

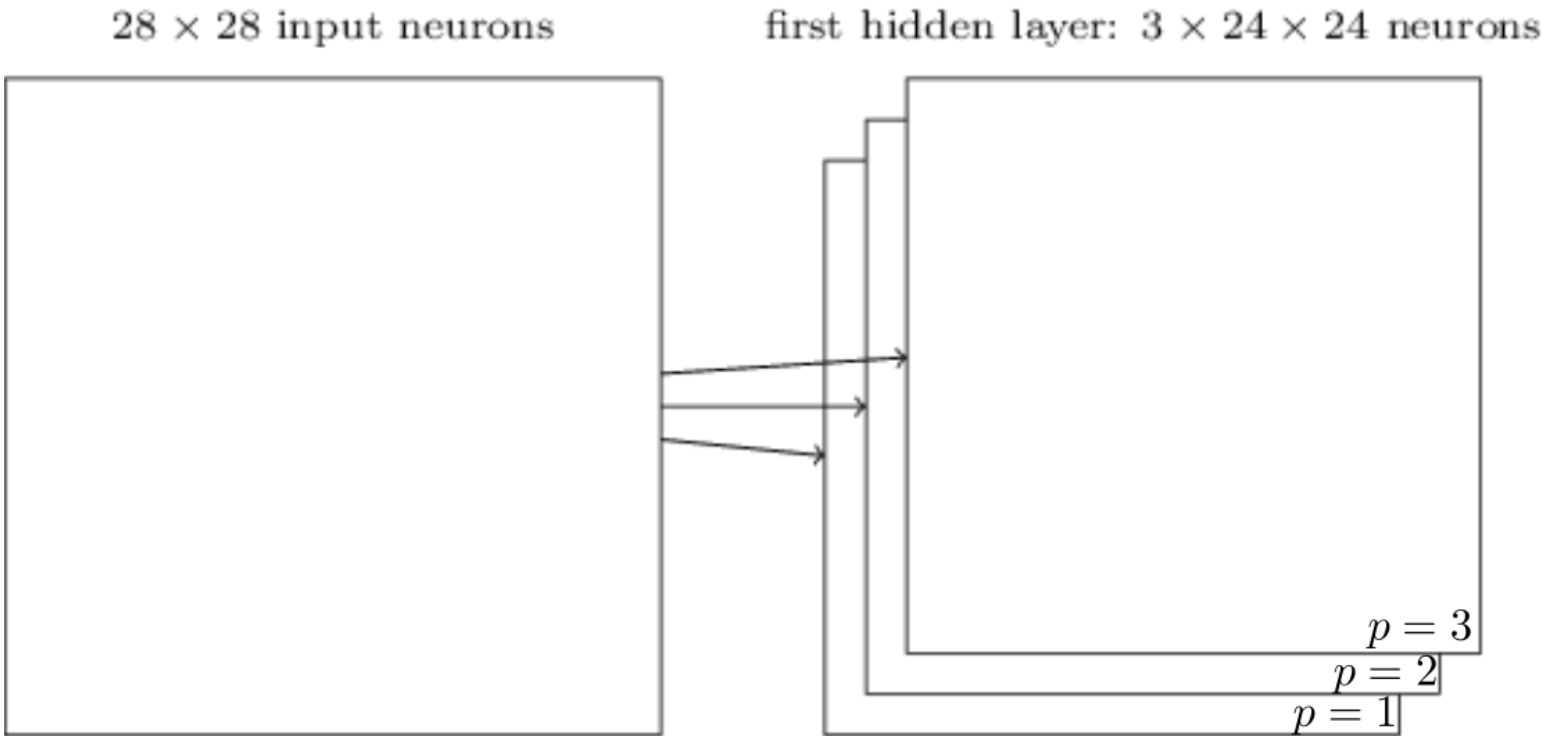
Convolution Layer

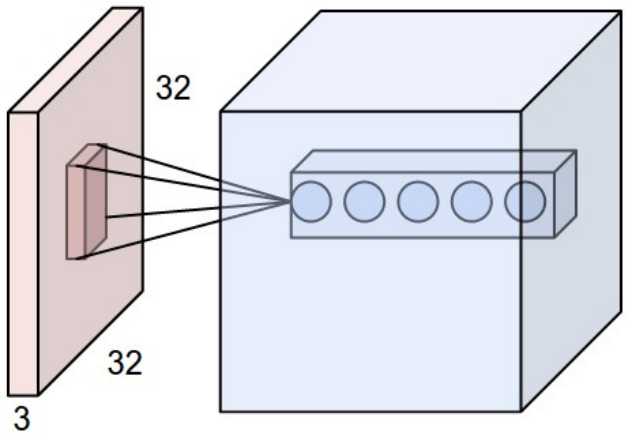


$$\text{output hidden unit (1,0)} = \sigma\left(b_1 + \sum_{l=0}^{5}\sum_{m=0}^{5} w_{lm} input_{1+l,1+m}\right).$$

Multiple features

Convolution Layer



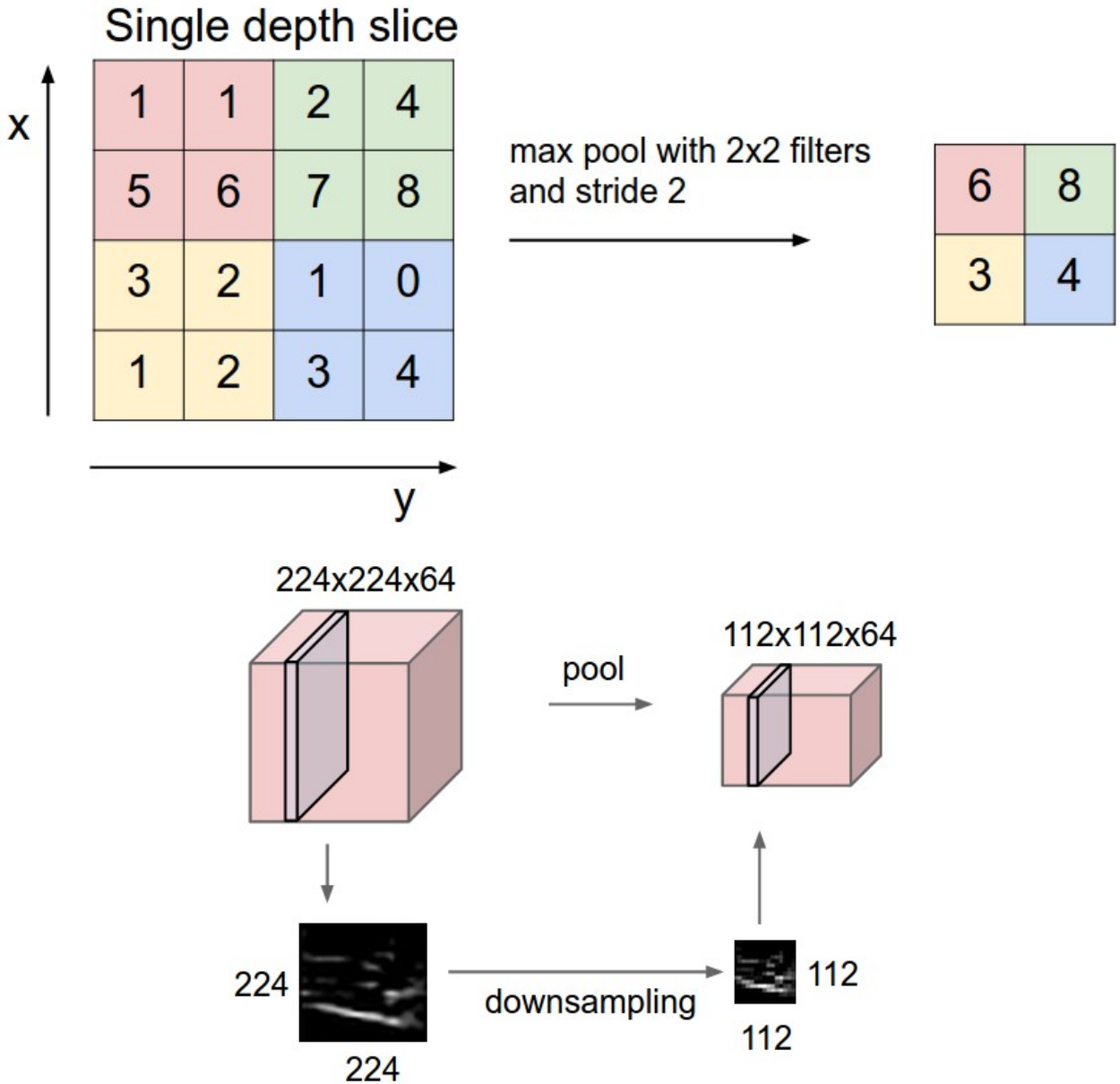


Stanford CS231n class slides

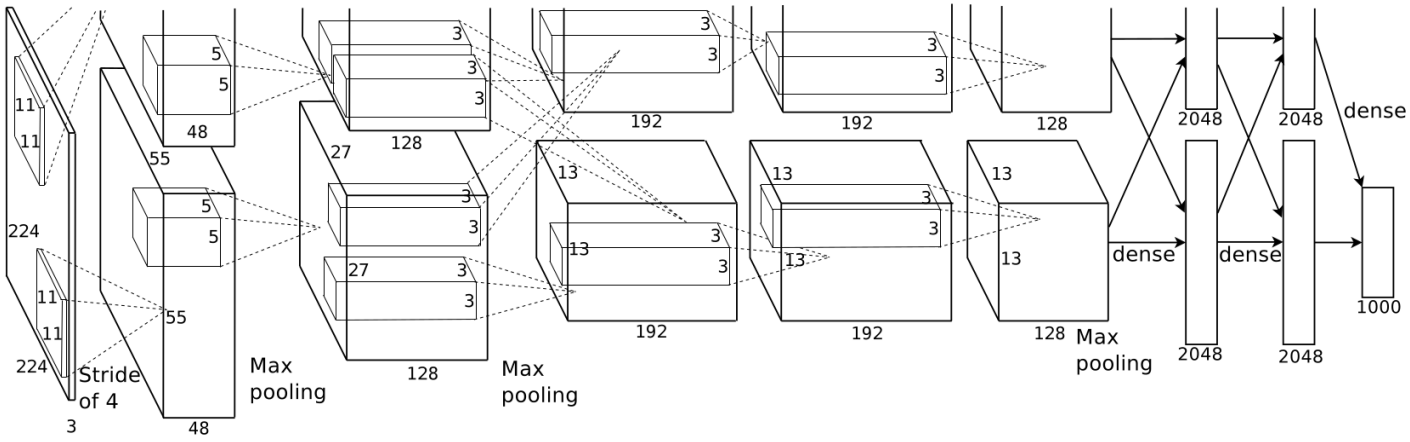
Convolutional layer parameters:

- Depth: Number of filters we would like to us
- Stride: The number of pixels by which we slide the filter
- Padding: Coping with boundaries by adding zeros around the input

Example: max pooling

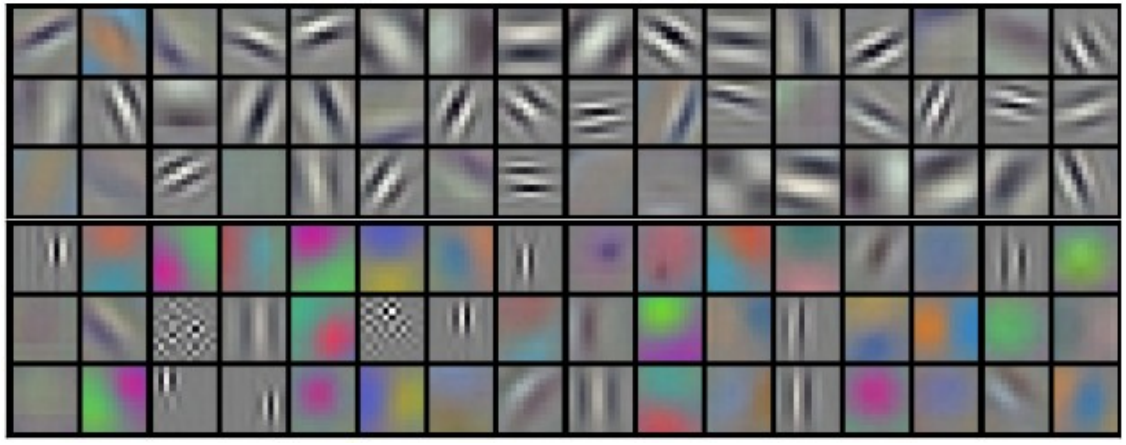


Krizhevsky et al. architecture

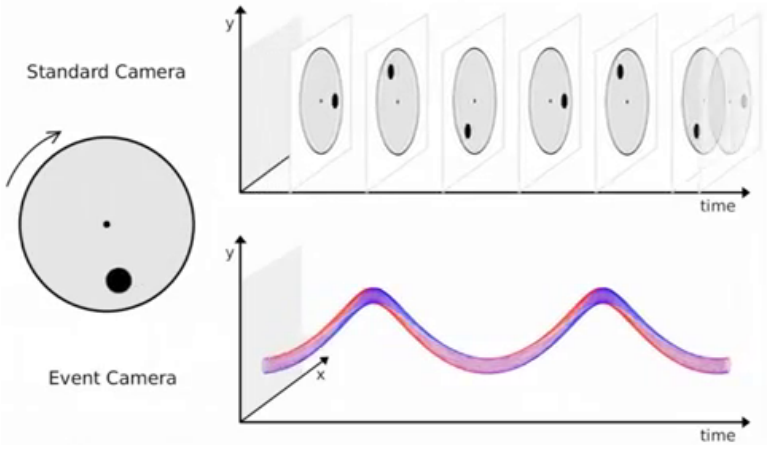


Krizhevsky, Sutskever, and Hinton, *Advances in neural information processing systems*, 2012

Features obtained at the first convolutional layer



Neural Network Hardware: Silicon Retina



Lichtsteiner_etal08, Lichtsteiner_etal08Lichtsteiner_etal08, Lichtsteiner_etal08

- Spike-Events on intensity change
- High temporal resolution (<1μs) and Dynamic Range

Neural Network Hardware: Edge TPU