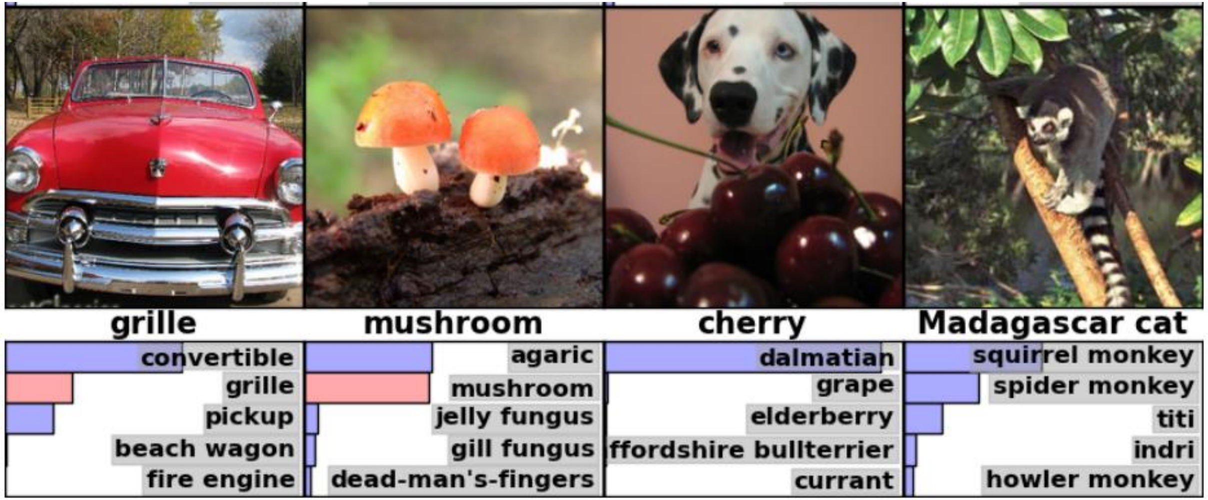
Machine Learning

Artificial Neural Networks


mite
container ship
motor scooter
leopard
mite
black widow
cockroach
tick
starfish
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
go-kart
moped
bumper car
golfcart
leopard
jaguar
cheetah
snow leopard
Egyptian cat


grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey

Overview:

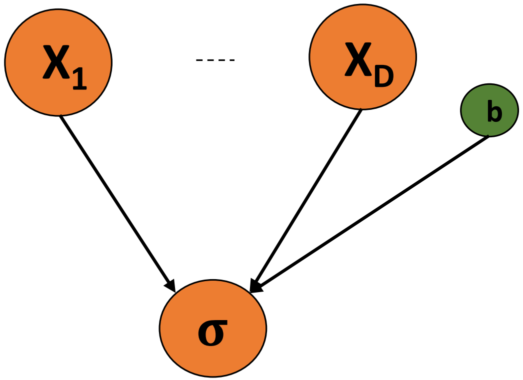
- Artificial Neuron
- Activation Function
- Neuron Capacity
- Single Layer ANN
- Multilayer ANN
- Universal Approximator
- Motivation behind ANN
- Backpropagation

- Hidden unit pre-activation:

$$z(x) = \sum_i w_i x_i + b = \mathrm{W}^{\mathrm{T}}\mathrm{X} + b$$

- Hidden unit activation:

$$f(x) = \sigma\left(z(x)\right) = \sigma\left(\sum_i w_i x_i + b\right)$$



W are weight matrix connects i^{th} hidden unit with i^{th} input unit

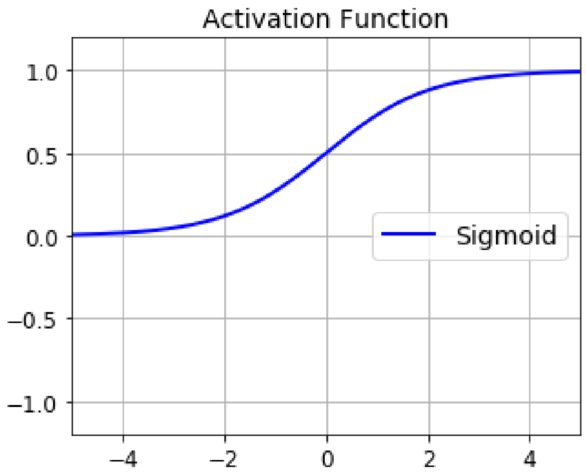
b are bias vectors

$\boldsymbol{\sigma}(\cdot)$ is the activation function

Sigmoid:

$$\sigma(z) = \mathrm{sigm}(z) = \frac{1}{1+\exp(-z)}$$

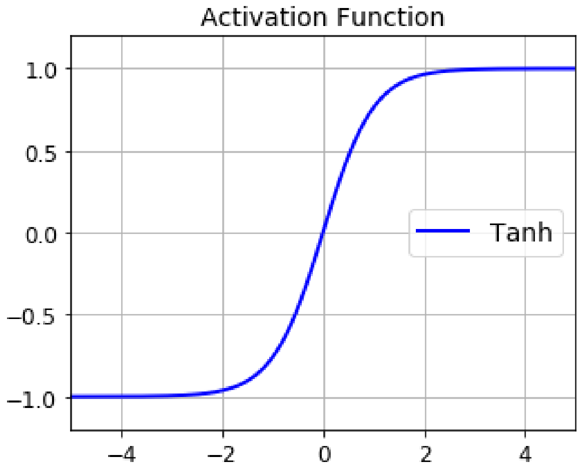
- Squashes the hidden unit's pre-activation to between 0 and 1.
- Always positive.
- Bounded.
- Strictly increasing.



Hyperbolic tangent ("tanh"):

$$\sigma(z) = \tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)} = \frac{\exp(2z)-1}{\exp(2z)+1}$$

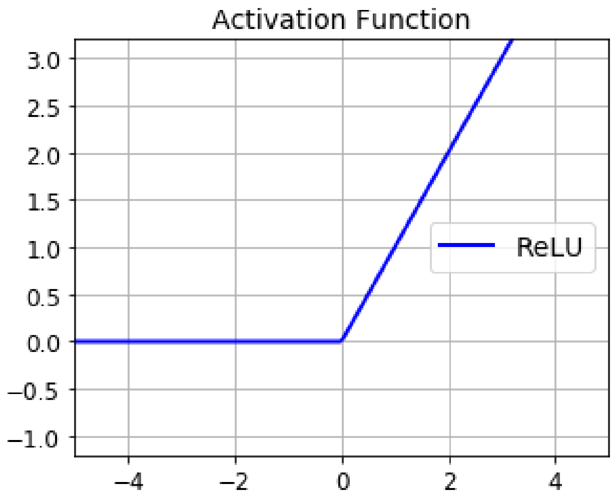
- Squashes the hidden unit's pre-activation to between -1 and 1.
- Can be positive or negative.
- Bounded.
- Strictly increasing.



Rectified Linear Unit (“ReLu”):

$\sigma(z) = \mathrm{ReLu}(z) = \max(0, z)$

- Bounded below by 0 (always non-negative).
- Not bounded above.
- Strictly increasing.



- Hidden layer pre-activation:

$$z(x)_i = b_i^{(1)} + W_{i,j}^{(1)} x_j$$

Similarly in Matrix form:

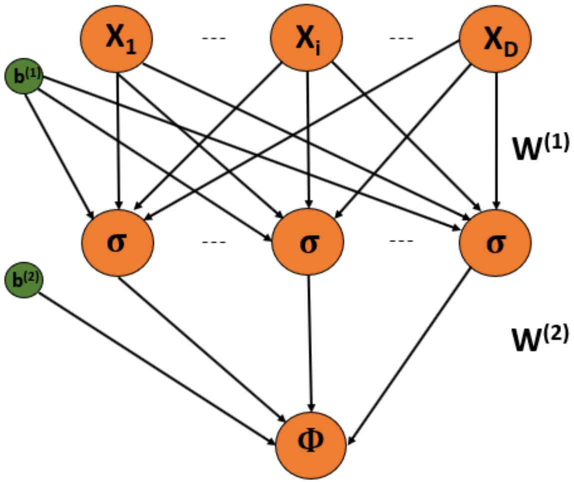
$$\mathrm{z(X)} = \mathrm{b}^{(1)} + \mathbf{W}^{(1)}\,\mathrm{X}$$

- Hidden layer activation:

$$\mathbf{h}(\mathbf{X}) = \boldsymbol{\sigma}(\mathbf{z}(\mathbf{X}))$$

- Output layer activation "Φ":

$$\mathrm{F(X)} = \Phi(\mathrm{b}^{(2)} + \mathbf{W}^{(2)^{\mathrm{T}}}\,\mathrm{h}^{(1)}\mathrm{X})$$



SoftMax Activation Function

- Multi-class classification:

  - requires multiple outputs i.e. 1 output per class.

  - need to estimate the conditional probability of output belonging to a particular class c, $p(y = c | \mathbf{x})$.

- Apply the SoftMax activation function at the output:

  $\Phi(z) = \mathrm{SoftMax}(z) = [\frac{e^{z_1}}{\sum_c e^{z_c}}, \dots, \frac{e^{z_1}}{\sum_c e^{z_c}}]^\mathrm{T}$

  - strictly positive

  - sums to one

- Predicted class is the one with highest estimated probability

Multilayer NN with *L* hidden layers

- **Hidden layer pre-activation for $k > 0$:**

  $\mathrm{h}^{(0)}(\mathrm{X}) = \mathrm{X}$,

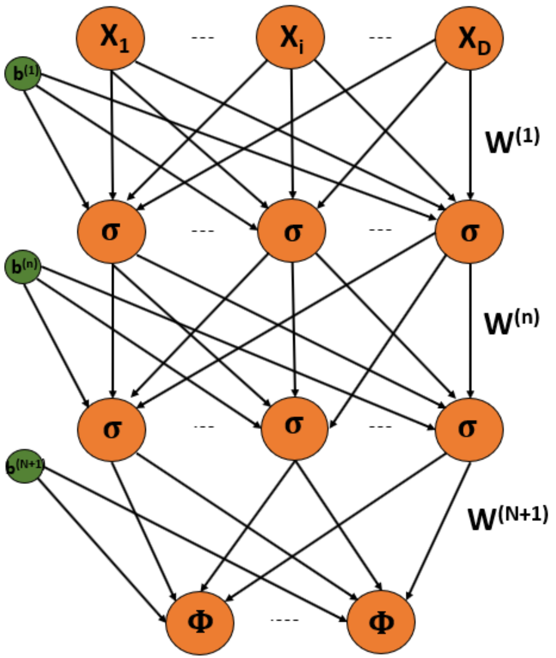
  $\mathrm{z}^{(k)}(\mathrm{X}) = \mathrm{b}^{(k)} + \mathbf{W}^{(k)}\, \mathrm{h}^{(k-1)}\mathrm{X}$

- **Hidden layer activation (*k* from 1 to *L*):**

  $\mathbf{h}^{(k)}(\mathrm{X}) = \boldsymbol{\sigma}(\mathbf{z}^{(k)}(\mathrm{X}))$

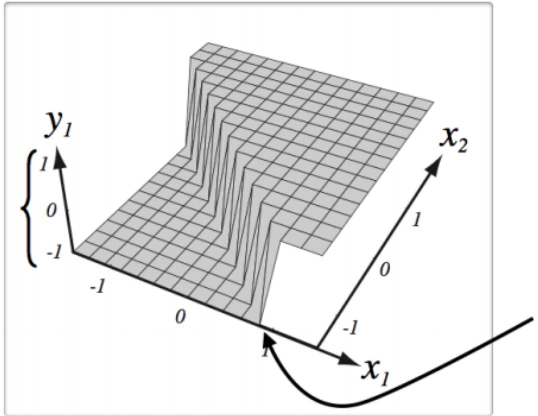
- **Output layer activation (*k* = *L+1*):**

  $\mathbf{F}(\mathbf{X}) = \mathbf{h}^{(L+1)}(\mathrm{X}) = \Phi(\mathrm{z}^{(L+1)}(\mathrm{X}))$



Capacity of Single Hidden Unit

- **Range of hidden unit determined by σ(.)**
- **Bias *b* changes the position of the riff.**

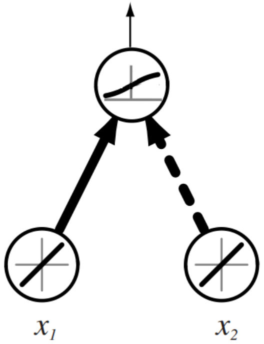


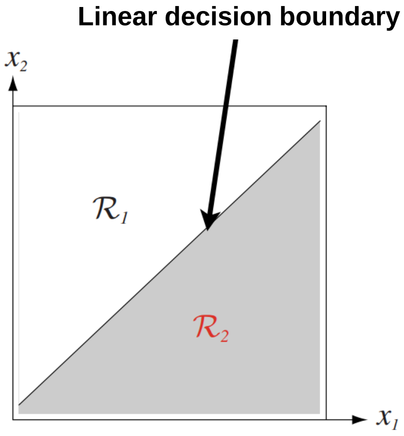
Bias only changes the position of the riff

Source: Pascal Vincent

Capacity of Single Hidden Unit

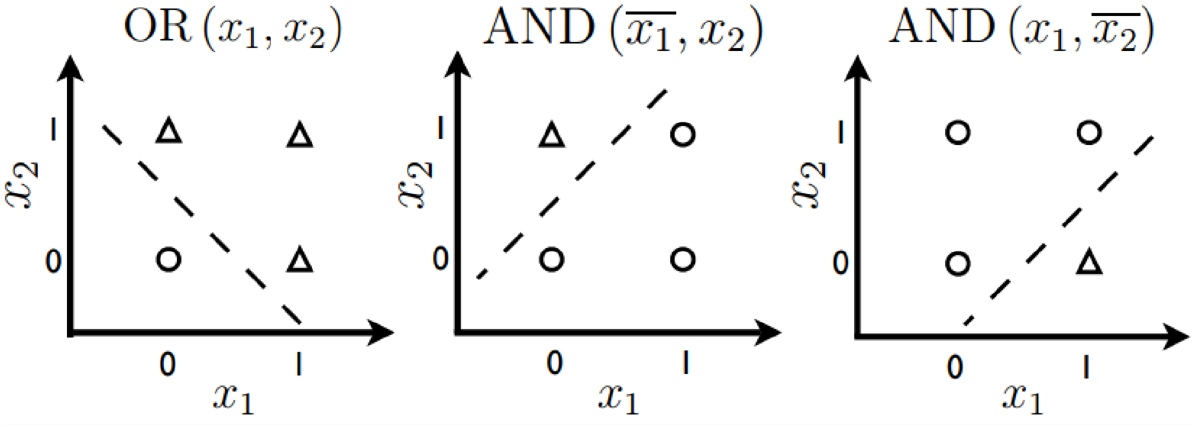
- Could do binary classification.
- With sigmoid, can interpret neuron as estimating $p(y = 1|\, \mathrm{x})$.
- Also known as logistic regression classifier, if greater than 0.5 predict class 1, otherwise predict class 0.





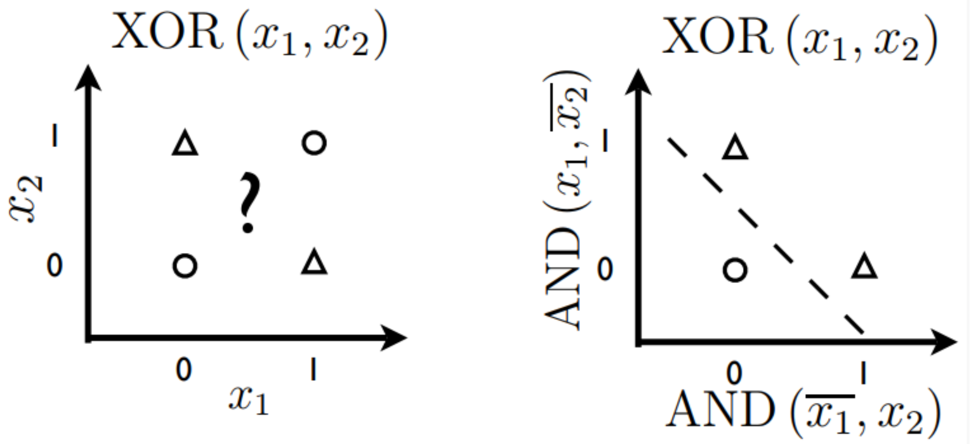
Capacity of Single Hidden Unit

- **Can solve linearly separable problems**



Capacity of Single Hidden Unit

- Cannot solve non-linearly separable problems
- Unless the input is transformed into a separable representation



- Hidden layer pre-activation:

$$z(x)_i = b_i^{(1)} + W_{i,j}^{(1)}\, x_j$$

Similarly in Matrix form:

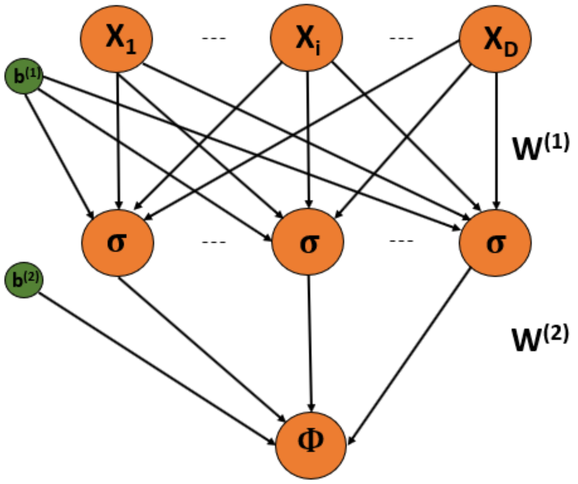
$$\mathrm{z}(\mathrm{X}) = \mathrm{b}^{(1)} + \mathbf{W}^{(1)}\, \mathrm{X}$$

- Hidden layer activation:

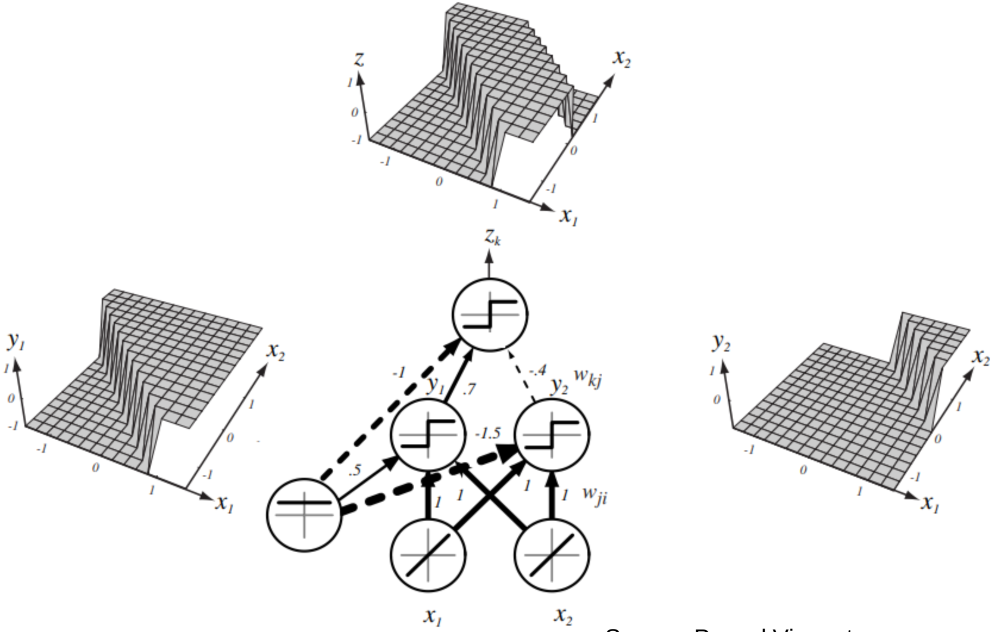
$$\mathbf{h}(\mathbf{X}) = \boldsymbol{\sigma}(\mathbf{z}(\mathbf{X}))$$

- Output layer activation “Φ”:

$$\mathrm{F}(\mathrm{X}) = \Phi(\mathrm{b}^{(2)} + \mathbf{W}^{(2)^{\mathrm{T}}}\, \mathrm{h}^{(1)}\mathrm{X})$$

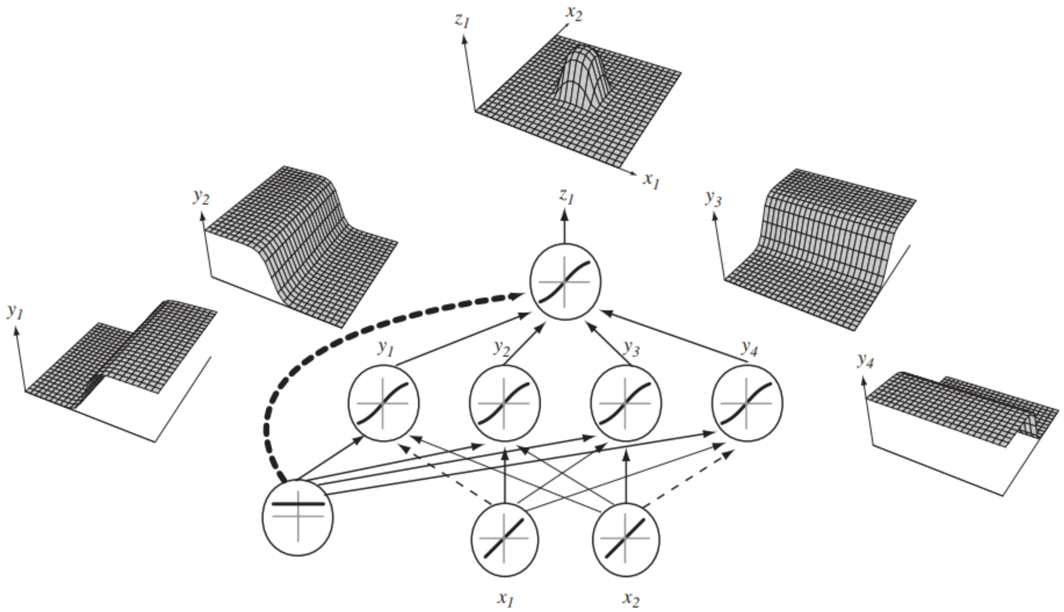


Capacity of Single Hidden Layer Neural Network



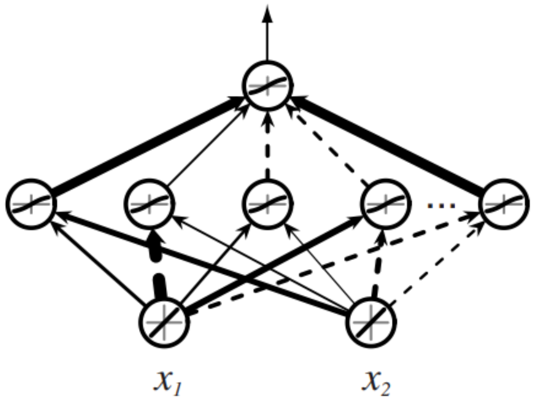
Source: Pascal Vincent

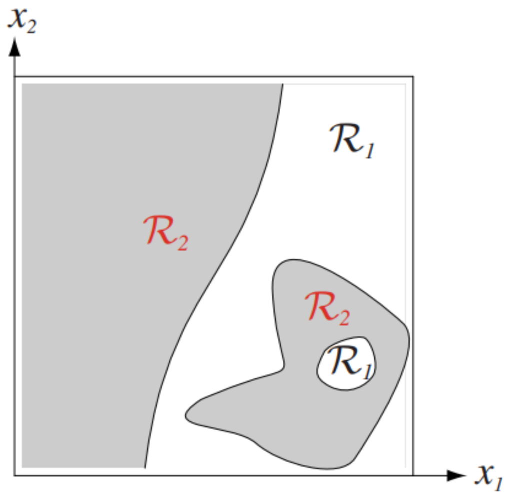
Capacity of Single Hidden Layer Neural Network



Source: Pascal Vincent

Capacity of Single Hidden Layer Neural Network





- **Universal Approximation Theorem (Hornik, 1991)**

  "Single layer feedforward network can approximate any continuous function arbitrarily well if and only if the network's activation function is continuous, non-constant, and bounded."

- Hornik's result applies for sigmoid, tanh and many other hidden layer activation functions.

- However, modern day defacto activation function is ReLu, and it does not satisfy Hornik's theorem as the $\mathrm{ReLu}(z) = \max(0, z)$ is unbounded from above.

Universal Approximator

- "Multilayer feedforward network can approximate any continuous function arbitrarily well if and only if the network's continuous activation function is not polynomial."

Definition

A set F of functions in $L^{\infty}_{loc}(R^n)$ is dense in $C(R^n)$ if for every function $g \in C(R^n)$ and for every compact set $K \subset R^n$, there exists a sequence of functions $f_j \in F$ such that

$$\lim_{f \to \infty} ||g - f_j||_{L^{\infty}(K)} = 0.$$

Theorem

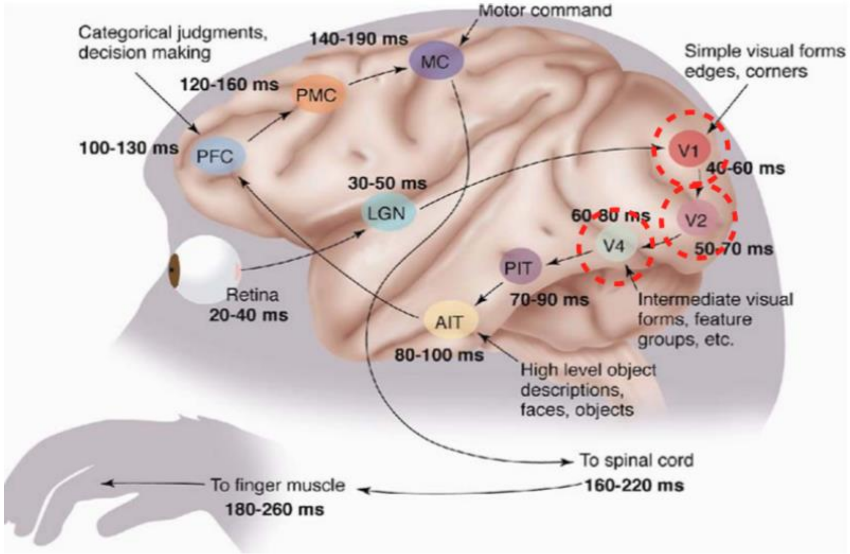
(Leshno et al., 1993) Let $\sigma \in M$, where M denotes the set of functions which are in $L^{\infty}_{loc}(\Omega)$.

$$\Sigma_n = span\{\sigma(w \cdot x + b) : w \in R^n, b \in R\}$$

Then Σ_n is dense in $C(R^n)$ if and only if σ is not an algebraic polynomial (a.e.).

Biological Inspiration

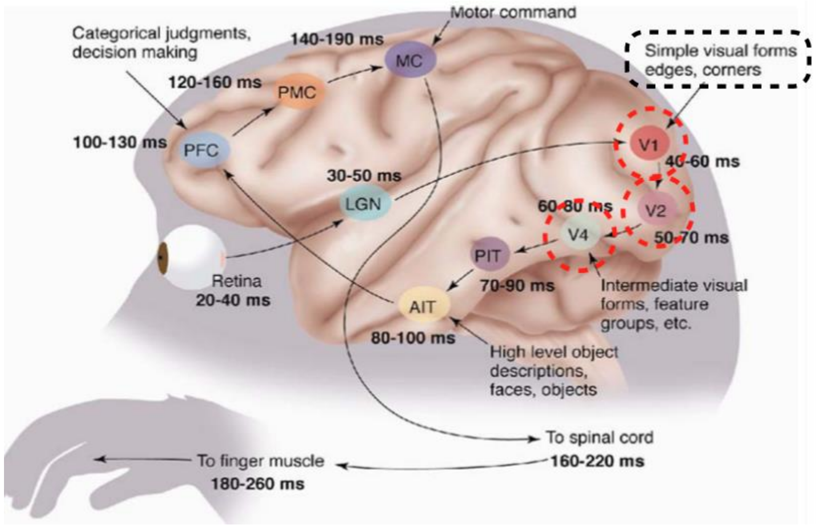
- **Parallel with the visual vortex**



Source: Simon Thorpe

Biological Inspiration

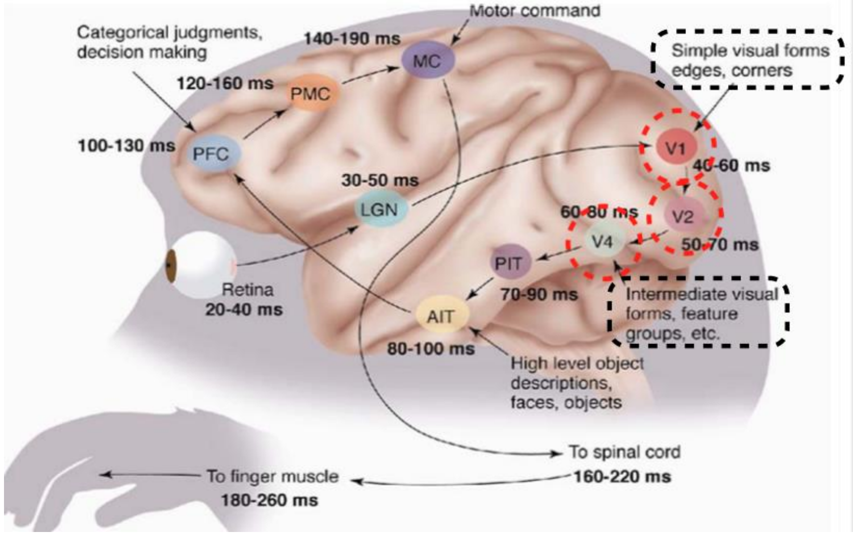
- Parallel with the visual vortex

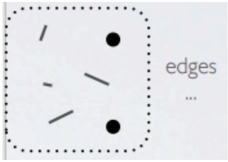


Source: Simon Thorpe

Biological Inspiration

- Parallel with the visual vortex

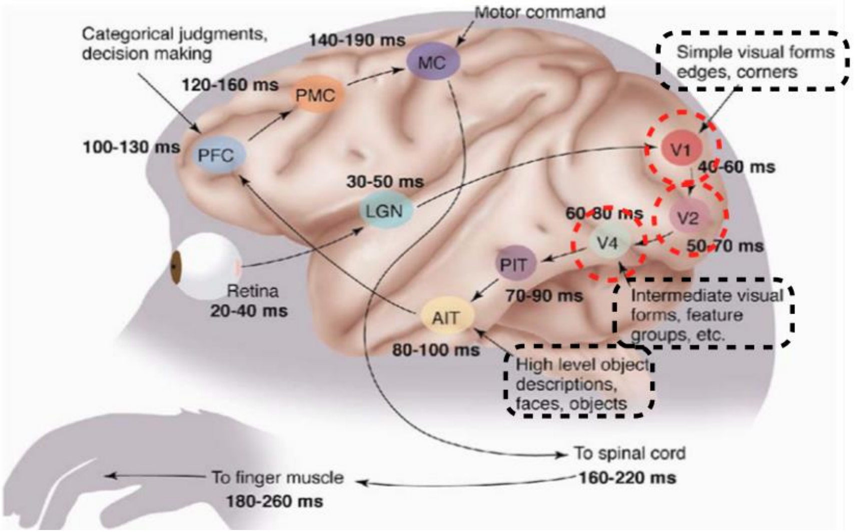


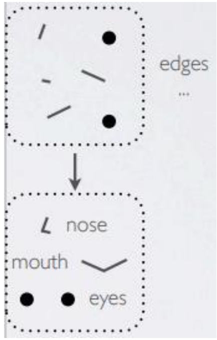


Source: Simon Thorpe

Biological Inspiration

- **Parallel with the visual vortex**
- **Edges**

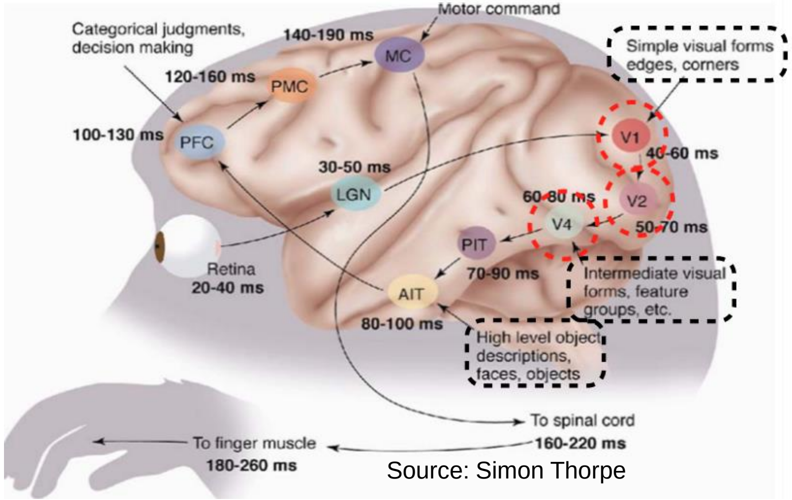




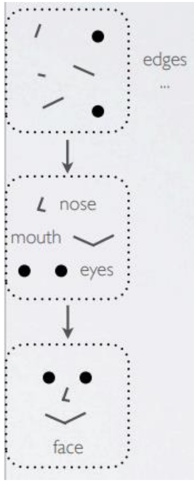
Source: Simon Thorpe

Biological Inspiration

- Parallel with the visual vortex
- Edges
- Higher level features such as nose, mouth, eyes
- Face



Source: Simon Thorpe



Motivation Behind ANN

- **Firing rates of different input neurons combine to influence the firing rate of other neurons:**

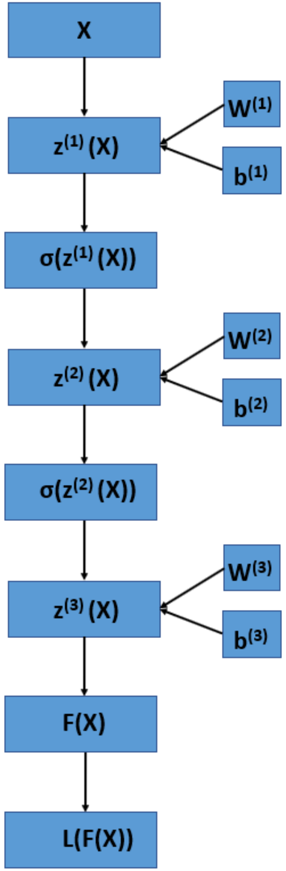
  - depending on the dendrite and axon, a neuron can either work to increase (excite) or decrease (inhibit) the firing rate of another neuron

- **This is what artificial neurons approximate:**

  - the activation corresponds to a ‘‘sort of’’ firing rate
  - the weights between neurons model whether neurons excite or inhibit each other
  - the activation function and bias model the threshold behavior of action potentials

Forward propagation

- Randomly Initialize $\boldsymbol{\Theta}$
- $\boldsymbol{\Theta}$ = $\{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}\ , \mathbf{W}^{(2)}, \mathbf{b}^{(2)}, \ldots, \mathbf{W}^{(L+1)}, \mathbf{b}^{(L+1)}\}$



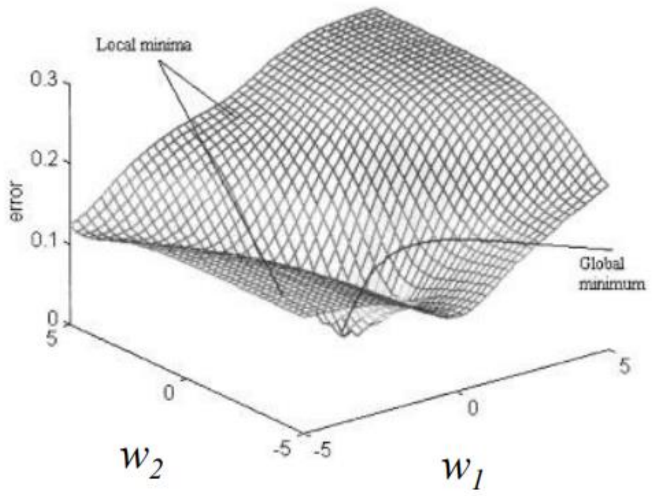
Objective Function

- Objective Function for Multi-class Classification

$$\underset{\boldsymbol{\theta}}{\mathbf{argmin}} \ \frac{\mathbf{1}}{\boldsymbol{m}} \sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{m}} \boldsymbol{l}(\mathbf{F}(\mathbf{x}^{(\boldsymbol{i})}; \boldsymbol{\theta}), \boldsymbol{y}^{(\boldsymbol{i})}) + \boldsymbol{\lambda}\, \boldsymbol{\Omega}(\boldsymbol{\theta})$$

- Loss Function: Negative log likelihood

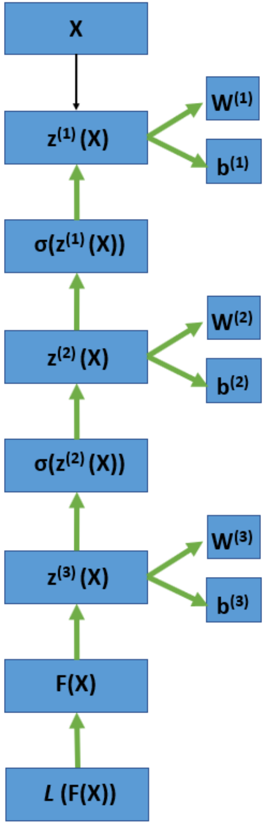
$$\boldsymbol{l}(\mathbf{F}(\mathbf{x}, \boldsymbol{\theta}), \boldsymbol{y}) \ = \ -\sum\nolimits_{\boldsymbol{c}} \mathbf{1}_{(\boldsymbol{y}=\boldsymbol{c})}\, \boldsymbol{log}\ \mathbf{F}(\mathbf{x})_{\mathbf{c}}$$



Cho & Chow, Neurocomputing 1999

Backpropagation

- Optimization Algorithm e.g. stochastic gradient descent
- Objective Function
- Gradient of Output Layer
- Gradient of Hidden Layer
- Gradient of Activation Function
- Gradient of Parameters



Optimization Algorithm

Stochastic (incremental) gradient descent

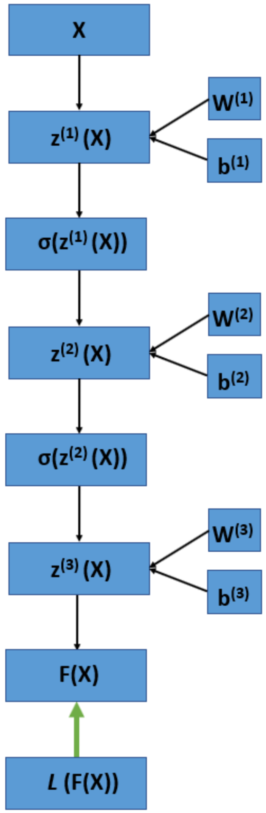
For each training example ($\mathrm{x}^{(i)}, y^{(i)}$)

For N iterations

- Compute $\Delta = - \nabla_{\boldsymbol{\theta}} \, \boldsymbol{l}(\mathbf{F}(\mathbf{x}^{(\boldsymbol{i})}; \boldsymbol{\theta}), \boldsymbol{y}^{(\boldsymbol{i})}) - \nabla_{\boldsymbol{\theta}} \boldsymbol{\lambda} \boldsymbol{\Omega}(\boldsymbol{\theta})$
- Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \, \Delta$

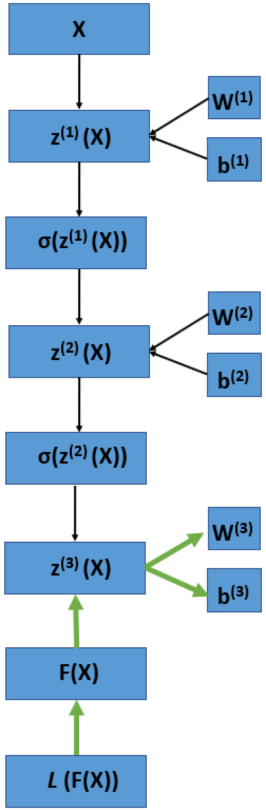
Loss Gradient at Output Layer

- $\nabla_{\mathbf{F}(\mathbf{x})} - \boldsymbol{log}\mathbf{F}(\mathbf{x})_{\boldsymbol{y}} = \frac{-\mathbf{e}(\boldsymbol{y})}{\mathbf{F}(\mathbf{x})_{\boldsymbol{y}}}$



Loss Gradient at Output Pre-activation

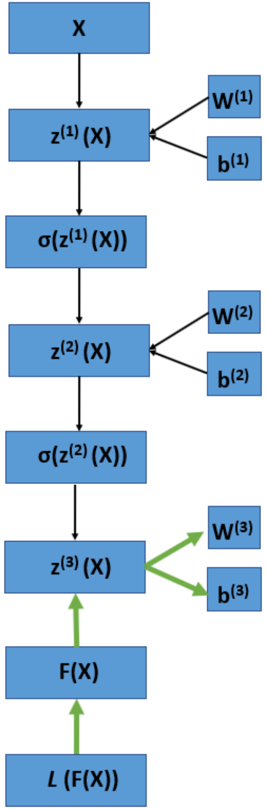
- $\nabla_{\mathbf{z}^{(L+1)}(\mathbf{x})} - \boldsymbol{log}\mathbf{F}(\mathbf{x})_{\boldsymbol{y}} = -(\mathbf{e}(\boldsymbol{y}) - \mathbf{F}(\mathbf{x}))$



Loss Gradient at Hidden Layer

- $\nabla_{z^{(k)}(x)} - log\mathbf{F}(\mathbf{x})_y$

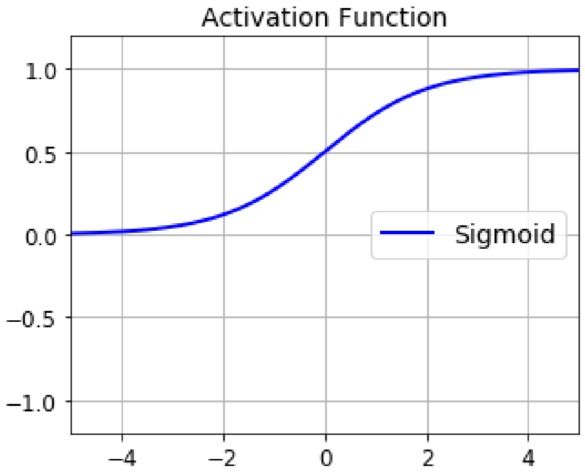
$$= \nabla_{\mathbf{h}^{(k)}(\mathbf{x})} - log\mathbf{F}(\mathbf{x})_y \odot [\ldots, \sigma'(\mathbf{z}^{(k)}(\mathbf{x})) \ ,\ldots]$$



Sigmoid Function Derivative:

- $\sigma(z) = \mathrm{sigm}(z) = \frac{1}{1+\exp(-z)}$

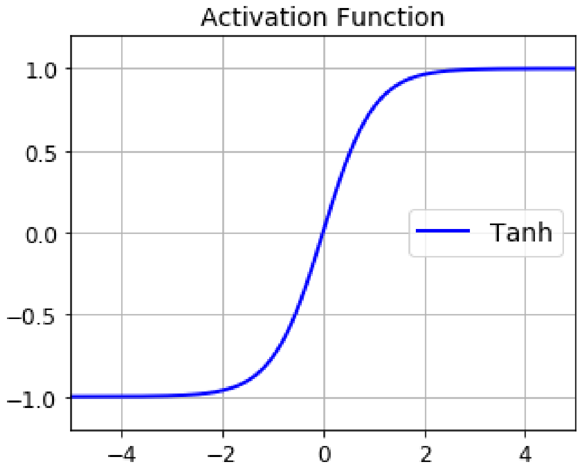
- $\sigma'(z) = \sigma(z)\,(1 - \sigma(z))$



Hyperbolic tangent ("tanh") Derivative:

- $\sigma(z) = \tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} = \frac{\exp(2z) - 1}{\exp(2z) + 1}$

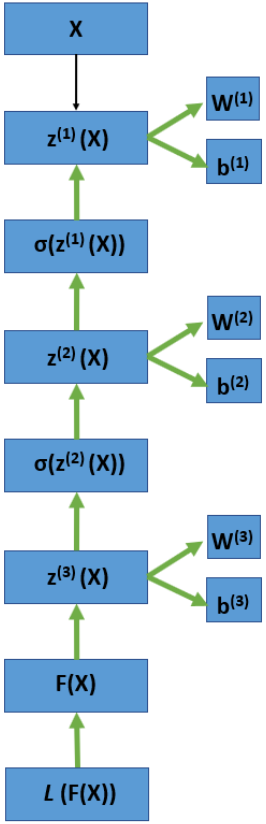
- $\sigma'(z) = 1 - \sigma(z)^2$



Gradient of Weights

- $\nabla_{\mathbf{w}^{(k)}} \; - \; \boldsymbol{log}\mathbf{F}(\mathbf{x})_{y}$

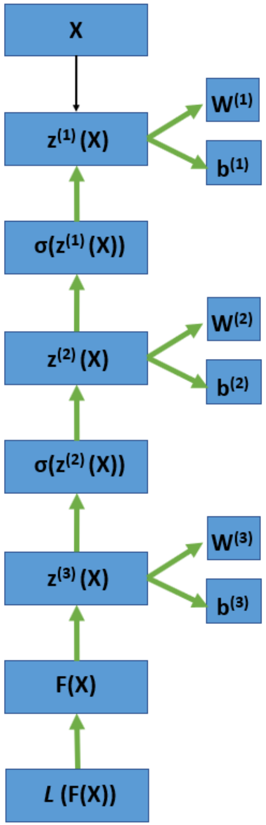
$$= (\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \boldsymbol{log}\mathbf{F}(\mathbf{x})_{y})\,\mathbf{h}^{(k-1)}(\mathbf{x})^{\mathsf{T}}$$



Gradient of Biases

- $\nabla_{\mathbf{b}^{(k)}} - \boldsymbol{log}\mathbf{F}(\mathbf{x})_y$

  $= (\nabla_{\mathbf{a}^{(k)}(\mathbf{x})} - \boldsymbol{log}\mathbf{F}(\mathbf{x})_y)$



Reference / Readings

Books

- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Aurélien Géron
- Pattern recognition and machine learning, Christopher M. Bishop
- The Elements of Statistical Learning, Hastie, Tibshirani, and Friedman

Blogs, code snippets, lecture notes, etc.

- https://github.com/stephencwelch/Neural-Networks-Demystified This tutorial uses numpy and python
- http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch This tutorial implements a classifier in python from scratch
- http://deeplearning.net/tutorial This set of tutorials uses the Theano package, which is pretty tricky to learn (but very powerful because it calculates gradients for you automatically, among other things)
- http://www.cs.stir.ac.uk/courses/ITNP4B/lectures/kms/1-Intro.pdf For some of the history and possible connections to neuroscience
- http://www.dmi.usherb.ca/~larocheh/index_en.html Modified from Hugo Larochelle