Problem Solving with AI Techniques
Artificial Neural Networks

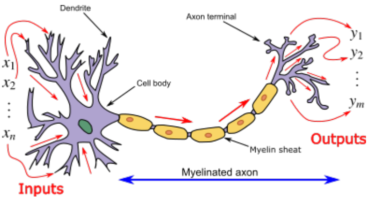
Paul Weng

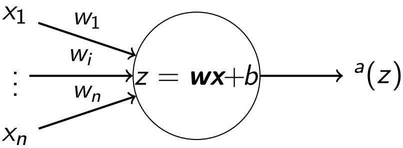
UM-SJTU Joint Institute

VE593, Fall 2018

JOINT INSTITUTE
交大密西根学院

Artificial Neuron



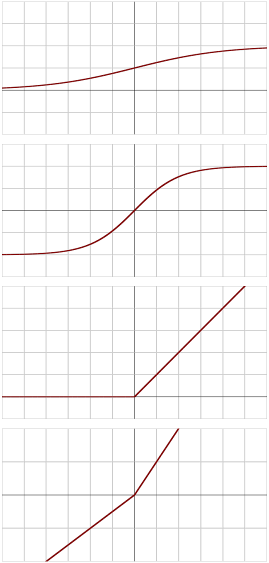


a

- Perceptron: $f(x) = \text{sign}(x)$
- Logistic regression: $f(x) = \frac{1}{1+e^{-x}}$
- Linear regression: $f(x) = x$
- f called activation function, usually it is (sub)differentiable
- Why non-linear activation function?

Examples of Activation Functions

- logistic or sigmoid function
  - $f(x) = \frac{1}{1+e^{-x}}$
  - $f'(x) = f(x)(1 - f(x))$
- tanh
  - $f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
  - $f'(x) = 1 - f(x)^2$
- reLU (Rectified Linear Unit)
  - $f(x) = \max(0, x)$
  - $f'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$
- leaky ReLU
  - $f(x) = \max(-\beta x, x)$ with small $\beta > 0$
  - $f'(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -\beta & \text{otherwise} \end{cases}$



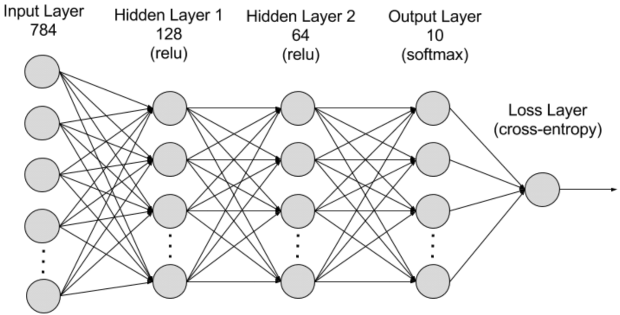
from wikipedia

Artificial Neural Networks (ANN)

- ANN is defined as a directed weighted graph:
  - three types of nodes: input nodes (nodes without predecessors), output nodes (nodes without successors) and hidden nodes
  - an activation function is associated to each non-input node
  - a weight is associated to each edge
  - Computation:
    - Input nodes send a fixed value to their successors
    - Non-input nodes compute weighted sum of their inputs and send results transformed by their activation function to their successors
- Feedforward vs recurrent ANNs
- ANN with fixed architecture defines a class of functions parametrized by the ANN's weights
- ANN is a composition of functions
- Multi-Layer Perceptron: graph is organized in layers

Multi-Layer Perceptron (MLP)

- Multi-Layer Perceptron = ANN whose graph is organized in layers + any non-input node connected to all nodes of its previous layer



from Amazon

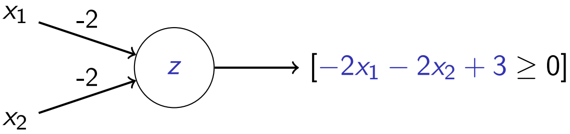
Empirical Argument

- State-of-the-art performance in some domains:
  - Computer vision
  - Speech processing
  - Natural language processing

- Possible thanks to:
  - Very large dataset
  - Powerful hardware
  - Efficient techniques for training ANNs (and especially deep learning)

CS Theoretical Argument

- Perceptrons can implement a NAND gate:



- Any logic function can be performed using only NAND gates

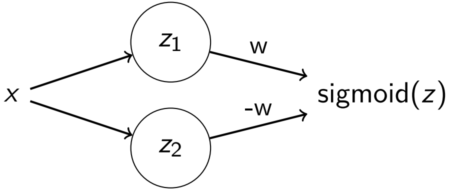
- Therefore ANN can perform any computation

Mathematical Theoretical Argument

- ANN is a universal function approximator:
  $\forall f : \mathbb{R}^D \rightarrow \mathbb{R}, \forall \varepsilon > 0$, $\exists$ an MLP with one hidden layer $g : \mathbb{R}^D \rightarrow \mathbb{R}$,

$$\forall \boldsymbol{x} \in \mathbb{R}^D, |f(\boldsymbol{x}) - g(\boldsymbol{x})| \leq \varepsilon$$

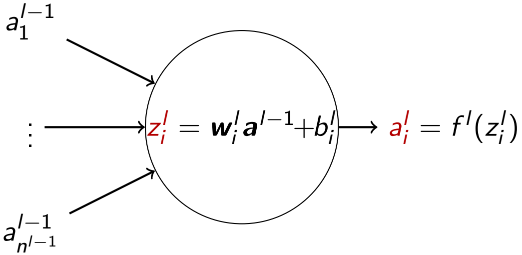
- Illustration: ANN can approximate a step function



where $z_i = w_i x + b_i = w_i(x + b_i/w_i)$ with w_i large

Notations for MLP

- n^l=# of nodes at layer $l = 1, \dots, L$
- $\boldsymbol{a}^l$=vector of outputs of the l-th layer of an MLP. Therefore, $\boldsymbol{a}^1$=$\boldsymbol{x}$
- w^l_{ij}=weight of connection between node i of layer l and node j of layer $l-1$
- Focusing on the i-th node of layer $l = 2, \dots, L$ of an MLP



- In matrix notations, $\boldsymbol{a}^l = f^l(\boldsymbol{z}^l) = f^l(\boldsymbol{w}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l) = g^l(\boldsymbol{a}^{l-1})$

Forward Pass: Illustration

- Therefore $\boldsymbol{a}^L = g^L(g^{L-1}(\dots g^2(\boldsymbol{a}^1)))$
- Example on blackboard

Forward Pass: Algorithm

- Assuming

$$\boldsymbol{W} = (\boldsymbol{w}^2, \dots \boldsymbol{w}^L)$$
$$\boldsymbol{B} = (\boldsymbol{b}^2, \dots \boldsymbol{b}^L)$$
$$\boldsymbol{F} = (\boldsymbol{f}^2, \dots \boldsymbol{f}^L)$$

where $\boldsymbol{w}^l \in \mathbb{R}^{n_l \times n_{l-1}}$, $\boldsymbol{b}^l \in \mathbb{R}^{n_l}$, and $f^l : \mathbb{R} \to \mathbb{R}$

Forward$\big(\boldsymbol{x},\ \boldsymbol{W}, \boldsymbol{B}, \boldsymbol{F}\big)$

$\boldsymbol{a}^0 \leftarrow \boldsymbol{x}$

for $l = 2$ *to* L **do**

  $\boldsymbol{z}^l \leftarrow \boldsymbol{w}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$

  $\boldsymbol{a}^l \leftarrow f^l(\boldsymbol{z}^l)$

return $\boldsymbol{a}^L$

General Problem

- Recall ERM (possibly with regularization)

$$R_{\mathcal{D}}(\boldsymbol{W}, \boldsymbol{B}) - \rho(\boldsymbol{W}, \boldsymbol{B}) = \frac{1}{N} \sum_{n=1}^{N} \ell(g_{\boldsymbol{W},\boldsymbol{B}}(\boldsymbol{x}^n), y^n) - \rho(\boldsymbol{W}, \boldsymbol{B})$$

where $g_{\boldsymbol{W},\boldsymbol{B}}$ is the function computed by ANN with parameters $\boldsymbol{W}, \boldsymbol{B}$

General Problem

- Recall ERM (possibly with regularization)

$$R_{\mathcal{D}}(\boldsymbol{W}, \boldsymbol{B}) - \rho(\boldsymbol{W}, \boldsymbol{B}) = \frac{1}{N}\sum_{n=1}^{N} \ell(g_{\boldsymbol{W},\boldsymbol{B}}(\boldsymbol{x}^n), y^n) - \rho(\boldsymbol{W}, \boldsymbol{B})$$

  where $g_{\boldsymbol{W},\boldsymbol{B}}$ is the function computed by ANN with parameters $\boldsymbol{W}, \boldsymbol{B}$
- Method: (Mini-batch stochastic) gradient descent

General Problem

- Recall ERM (possibly with regularization)

$$R_{\mathcal{D}}(\boldsymbol{W}, \boldsymbol{B}) - \rho(\boldsymbol{W}, \boldsymbol{B}) = \frac{1}{N}\sum_{n=1}^{N} \ell(g_{\boldsymbol{W},\boldsymbol{B}}(\boldsymbol{x}^n), y^n) - \rho(\boldsymbol{W}, \boldsymbol{B})$$

  where $g_{\boldsymbol{W},\boldsymbol{B}}$ is the function computed by ANN with parameters $\boldsymbol{W}, \boldsymbol{B}$
- Method: (Mini-batch stochastic) gradient descent
- Issues:
  - Non-convex optimization problem
  - How to compute the gradient?

Backpropagation: Introduction

- ANN is a composition of functions $g_{\boldsymbol{W},\boldsymbol{B}} = g^L \circ g^{L-1} \circ \ldots \circ g^2$
- Gradient can be computed by the chain rule
- Closed-form equation of gradient for an ANN may be complex, but
- Its value at a fixed $(\boldsymbol{W}, \boldsymbol{B})$ can be computed recursively!
- This recursive computation is called backpropagation

Backpropagation: Last Layer L

- For node j of layer L,

$$\frac{\partial \ell(\boldsymbol{a}^L, \boldsymbol{y})}{\partial w_{jk}^L} = \frac{\partial \ell(f^L(\boldsymbol{z}^L), \boldsymbol{y})}{\partial w_{jk}^L}$$
$$= \frac{\partial z_j^L}{\partial w_{jk}^L} f^{L'}(z_j^L) \frac{\partial \ell(\boldsymbol{a}^L, \boldsymbol{y})}{\partial a_j}$$
$$= a_k^{L-1} f^{L'}(z_j^L) \frac{\partial \ell(\boldsymbol{a}^L, \boldsymbol{y})}{\partial a_j}$$

$$\frac{\partial R}{\partial w_{jk}^L} = a_k^{L-1} f^{L'}(z_j^L) \frac{\partial R}{\partial a_j}$$

$$\frac{\partial R}{\partial b_j^L} = f^{L'}(z_j^L) \frac{\partial R}{\partial a_j}$$

- Let $\boldsymbol{\delta}^L = f^{L'}(\boldsymbol{z}^L) \otimes \nabla_{\boldsymbol{a}} R = \frac{\partial R}{\partial \boldsymbol{z}^L}$

Backpropagation: Recurrence

- At node j of layer l,

$$\delta_j^l = \frac{\partial R}{\partial z_j^l} = \sum_k \frac{\partial R}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}$$

Backpropagation: Recurrence

- At node j of layer l,

$$\delta_j^l = \frac{\partial R}{\partial z_j^l} = \sum_k \frac{\partial R}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}$$

- By definition, $z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} f^l(z_j^l) + b_k^{l+1}$

Backpropagation: Recurrence

- At node j of layer l,

$$\delta_j^l = \frac{\partial R}{\partial z_j^l} = \sum_k \frac{\partial R}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}$$

- By definition, $z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} f^l(z_j^l) + b_k^{l+1}$
- By differentiating, $\dfrac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} f^{l\prime}(z_j^l)$

Backpropagation: Recurrence

- At node j of layer l,

$$\delta_j^l = \frac{\partial R}{\partial z_j^l} = \sum_k \frac{\partial R}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}$$

- By definition, $z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} f^l(z_j^l) + b_k^{l+1}$
- By differentiating, $\dfrac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} f^{l\prime}(z_j^l)$
- Therefore,

$$\delta_j^l = \sum_k w_{kj}^{l+1} f^{l\prime}(z_j^l) \delta_k^{l+1} = \boldsymbol{w}_{\cdot j}^{l+1\intercal} \boldsymbol{\delta}^{l+1} f^{l\prime}(z_j^l)$$

$$\boldsymbol{\delta}^l = \boldsymbol{w}^{l+1\intercal} \boldsymbol{\delta}^{l+1} \otimes f^{l\prime}(\boldsymbol{z}^l)$$

Backpropagation: Layer l

- At node j of layer l, $\frac{\partial R}{\partial w_{jk}^l} = \frac{\partial R}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l}$

Backpropagation: Layer l

- At node j of layer l, $\dfrac{\partial R}{\partial w^l_{jk}} = \dfrac{\partial R}{\partial z^l_j}\dfrac{\partial z^l_j}{\partial w^l_{jk}}$
- By definition, $z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$

Backpropagation: Layer l

- At node j of layer l, $\dfrac{\partial R}{\partial w_{jk}^l} = \dfrac{\partial R}{\partial z_j^l} \dfrac{\partial z_j^l}{\partial w_{jk}^l}$
- By definition, $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$
- By differentiating, $\dfrac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1}$

Backpropagation: Layer l

- At node j of layer l, $\dfrac{\partial R}{\partial w^l_{jk}} = \dfrac{\partial R}{\partial z^l_j} \dfrac{\partial z^l_j}{\partial w^l_{jk}}$
- By definition, $z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$
- By differentiating, $\dfrac{\partial z^l_j}{\partial w^l_{jk}} = a^{l-1}_k$
- Therefore, $\dfrac{\partial R}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$

Backpropagation: Layer l

- At node j of layer l, $\dfrac{\partial R}{\partial w_{jk}^l} = \dfrac{\partial R}{\partial z_j^l} \dfrac{\partial z_j^l}{\partial w_{jk}^l}$
- By definition, $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$
- By differentiating, $\dfrac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1}$
- Therefore, $\dfrac{\partial R}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$
- Similarly, $\dfrac{\partial R}{\partial b_j^l} = \delta_j^l$

Backpropagation: Summary

The gradient of the ANN at current parameter can be computed with:

- $\delta_j^L = \frac{\partial R}{\partial a_j^L} f^{l\prime}(z_j^L)$
- $\boldsymbol{\delta}^l = \left(\boldsymbol{w}^{l+1\intercal} \boldsymbol{\delta}^{l+1}\right) \otimes f^{l\prime}(\boldsymbol{z}^l)$
- $\frac{\partial R}{\partial b_j^l} = \delta_j^l$
- $\frac{\partial R}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$

Backpropagation: Algorithm

Backpropagation$\left(\boldsymbol{x},\ \boldsymbol{W}, \boldsymbol{B}, \boldsymbol{F}\right)$

$\boldsymbol{a}^1 \leftarrow \boldsymbol{x}$

for $l = 2$ *to* L **do**

  $\boldsymbol{z}^l \leftarrow \boldsymbol{w}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$

  $\boldsymbol{a}^l \leftarrow f^l(\boldsymbol{z}^l)$

$\boldsymbol{\delta}^L \leftarrow \nabla_a R \otimes f^{L\prime}(\boldsymbol{z}^L)$

for $l = L - 1$ *to* 2 **do**

  $\boldsymbol{\delta}^l \leftarrow \boldsymbol{w}^{l+1\intercal} \boldsymbol{\delta}^{l+1} \otimes f^{l\prime}(\boldsymbol{z}^l)$

return $\frac{\partial R}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial R}{\partial b^l_j} = \delta^l_j$

How to Choose the Number of Layers/Neurons?

Two schools:

- Start by training small network
- Grow it by increasing # nodes in a layer or adding layers
- Stop when performance on test set doesn't improve anymore

or

- Start with as large network as you can
- Train with regularized ERM

Regularization

- Recall

$$\min_{\boldsymbol{W},\boldsymbol{B}} R_{\mathcal{D}}(\boldsymbol{W},\boldsymbol{B}) - \frac{\lambda}{2N}\rho(\boldsymbol{W},\boldsymbol{B})$$

Regularization

- Recall

$$\min_{\boldsymbol{W},\boldsymbol{B}} R_{\mathcal{D}}(\boldsymbol{W}, \boldsymbol{B}) - \frac{\lambda}{2N}\rho(\boldsymbol{W}, \boldsymbol{B})$$

- L2 regularization: $\rho(\boldsymbol{W}, \boldsymbol{B}) = ||W||_2^2 = \sum_{l,i,j}(w_{ij}^l)^2$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha\nabla_{\boldsymbol{W}} R_{\mathcal{D}} - \alpha\frac{\lambda}{N}\boldsymbol{W} = \left(1 - \frac{\alpha\lambda}{N}\right)\boldsymbol{W} - \alpha\nabla_{\boldsymbol{W}} R_{\mathcal{D}}$$

Regularization

- Recall

$$\min_{\boldsymbol{W},\boldsymbol{B}} R_{\mathcal{D}}(\boldsymbol{W},\boldsymbol{B}) - \frac{\lambda}{2N}\rho(\boldsymbol{W},\boldsymbol{B})$$

- L2 regularization: $\rho(\boldsymbol{W},\boldsymbol{B}) = ||W||_2^2 = \sum_{l,i,j}(w_{ij}^l)^2$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha\nabla_{\boldsymbol{W}} R_{\mathcal{D}} - \alpha\frac{\lambda}{N}\boldsymbol{W} = \left(1 - \frac{\alpha\lambda}{N}\right)\boldsymbol{W} - \alpha\nabla_{\boldsymbol{W}} R_{\mathcal{D}}$$

- L1 regularization: $\rho(\boldsymbol{W},\boldsymbol{B}) = ||W||_1 = \sum_{l,i,j}|w_{ij}^l|$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha\frac{\lambda}{N}\mathrm{sign}(\boldsymbol{W}) - \alpha\nabla_{\boldsymbol{W}} R_{\mathcal{D}}$$

Regularization

- Recall

$$\min_{\boldsymbol{W},\boldsymbol{B}} R_{\mathcal{D}}(\boldsymbol{W}, \boldsymbol{B}) - \frac{\lambda}{2N}\rho(\boldsymbol{W}, \boldsymbol{B})$$

- L2 regularization: $\rho(\boldsymbol{W}, \boldsymbol{B}) = ||W||_2^2 = \sum_{l,i,j}(w_{ij}^l)^2$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha\nabla_{\boldsymbol{W}} R_{\mathcal{D}} - \alpha\frac{\lambda}{N}\boldsymbol{W} = \big(1 - \frac{\alpha\lambda}{N}\big)\boldsymbol{W} - \alpha\nabla_{\boldsymbol{W}} R_{\mathcal{D}}$$

- L1 regularization: $\rho(\boldsymbol{W}, \boldsymbol{B}) = ||W||_1 = \sum_{l,i,j}|w_{ij}^l|$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha\frac{\lambda}{N}\text{sign}(\boldsymbol{W}) - \alpha\nabla_{\boldsymbol{W}} R_{\mathcal{D}}$$

- Intuition: Less sensitive to noise

Regularization

- Recall

$$\min_{\boldsymbol{W},\boldsymbol{B}} R_{\mathcal{D}}(\boldsymbol{W}, \boldsymbol{B}) - \frac{\lambda}{2N}\rho(\boldsymbol{W}, \boldsymbol{B})$$

- L2 regularization: $\rho(\boldsymbol{W}, \boldsymbol{B}) = ||W||_2^2 = \sum_{l,i,j}(w_{ij}^l)^2$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha\nabla_{\boldsymbol{W}} R_{\mathcal{D}} - \alpha\frac{\lambda}{N}\boldsymbol{W} = \left(1 - \frac{\alpha\lambda}{N}\right)\boldsymbol{W} - \alpha\nabla_{\boldsymbol{W}} R_{\mathcal{D}}$$

- L1 regularization: $\rho(\boldsymbol{W}, \boldsymbol{B}) = ||W||_1 = \sum_{l,i,j}|w_{ij}^l|$

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha\frac{\lambda}{N}\text{sign}(\boldsymbol{W}) - \alpha\nabla_{\boldsymbol{W}} R_{\mathcal{D}}$$

- Intuition: Less sensitive to noise
- Bias $\boldsymbol{B}$ generally not regularized. Why?

Dropout

- Principle:
  - Training: at each gradient step, remove temporally half of the nodes of the ANN at random and use the remaining for training
  - Inference: divide weights from hidden nodes by 2
- Intuition: more robust learning

Data Augmentation Techniques (Not Only for ANNs)

- More data $=>$ better trained model, but data costly to collect
- Idea: artificially expand dataset at hand

- Input noise
  - generate new sample $(\boldsymbol{x}+\varepsilon, \boldsymbol{y})$ where $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}\}$ and ε some (Gaussian) random noise

- Problem (roughly) invariant for some transform $f : \mathcal{X} to \mathcal{X}$
  - generate new sample $(f(\boldsymbol{x}), \boldsymbol{y})$ where $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}\}$
  - for images, $f=$ (small) translation, rotation, zoom in/out...