Lecture 13: Neural Networks
CS486/686 Intro to Artificial Intelligence

2023-6-22

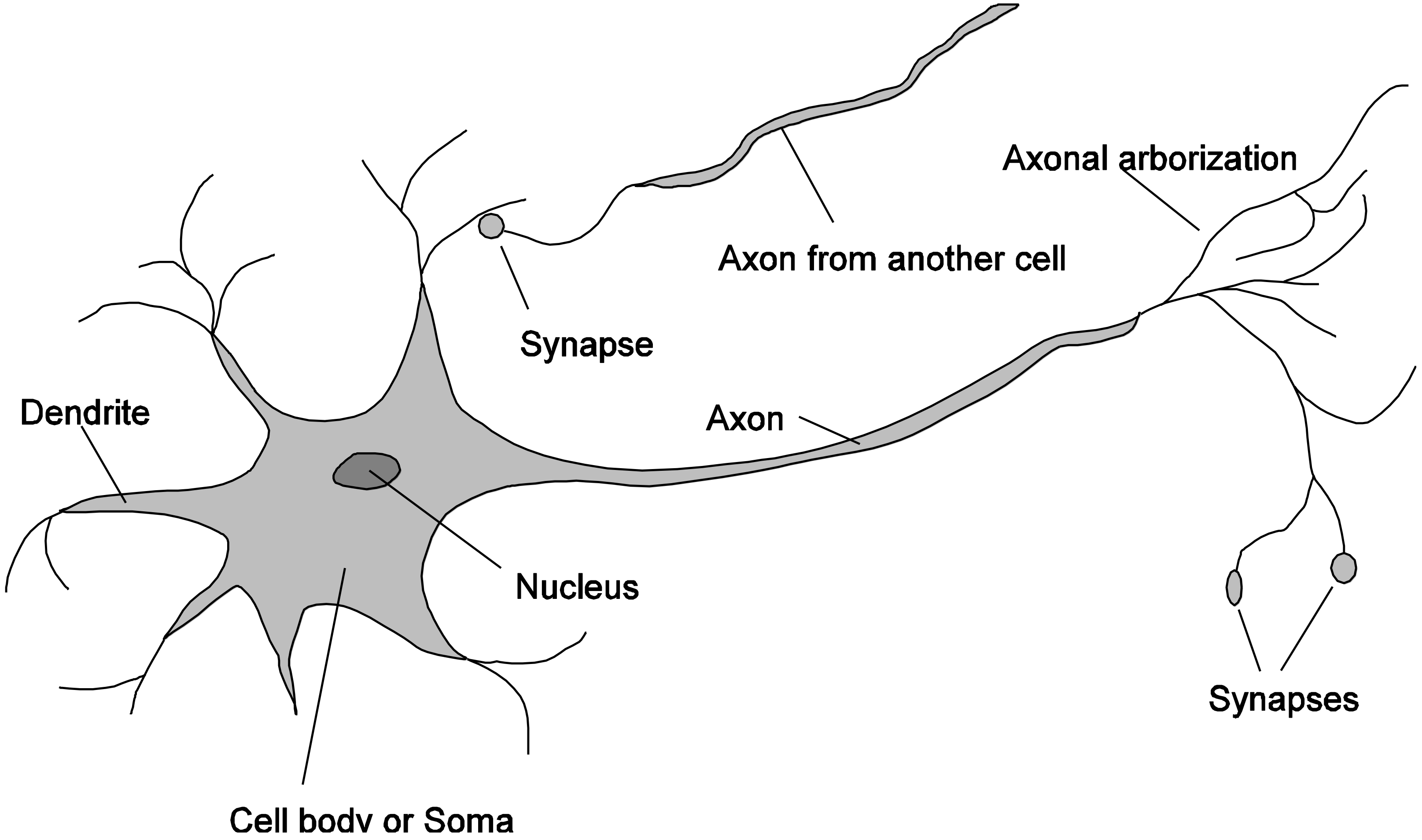
Pascal Poupart
David R. Cheriton School of Computer Science

UNIVERSITY OF WATERLOO

Outline

- Neural networks
  - Perceptron
  - Supervised learning algorithms for neural networks

Neuron



Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell bodv or Soma

Artificial Neural Networks

- Idea: **mimic the brain to do computation**

- Artificial neural network:
  - Nodes (a.k.a. units) correspond to neurons
  - Links correspond to synapses

- Computation:
  - Numerical signal transmitted between nodes corresponds to chemical signals between neurons
  - Nodes modifying numerical signal corresponds to neurons firing rate

ANN Unit

For each unit i:

- **Weights: *W***
  - Strength of the link from unit i to unit j
  - Input signals x_i weighted by W_{ji} and linearly combined:

$$a_j = \sum_i W_{ji} x_i + W_{j0} = \boldsymbol{W_j} \, \bar{\boldsymbol{x}}$$

- **Activation function: *h***
  - Numerical signal produced: $y_j = h(a_j)$

ANN Unit

- Picture

Activation Function

- Should be nonlinear
  - Otherwise, network is just a linear function
- Often chosen to mimic firing in neurons
  - Unit should be "active" (output near 1) when fed with the "right" inputs
  - Unit should be "inactive" (output near 0) when fed with the "wrong" inputs

Common Activation Functions

Threshold

Sigmoid

Logic Gates

- McCulloch and Pitts (1943)
  - Design ANNs to represent Boolean functions
- What should be the weights of the following units to code AND, OR, NOT ?

AND

OR

NOT

Network Structures

- **Feed-forward network**
  - Directed **acyclic** graph
  - No internal state
  - Simply computes outputs from inputs

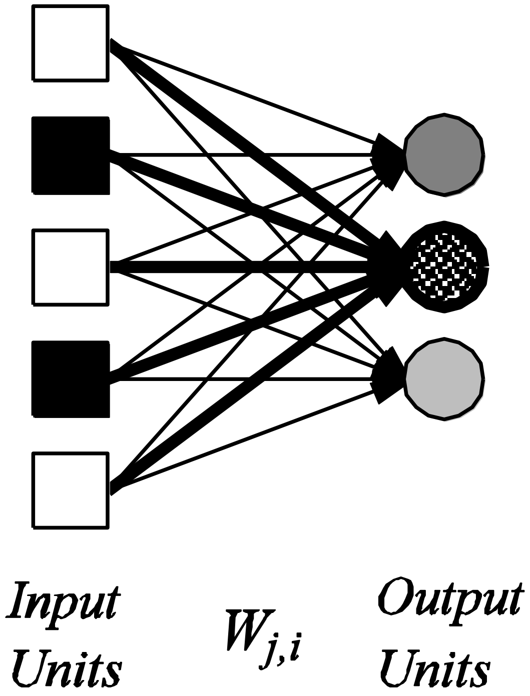
- **Recurrent network**
  - Directed **cyclic** graph
  - Dynamical system with internal states
  - Can memorize information

Feed-forward network

- Simple network with two inputs, one hidden layer of two units, one output unit

Perceptron

- Single layer feed-forward network



UNIVERSITY OF WATERLOO

Threshold Perceptron Hypothesis Space

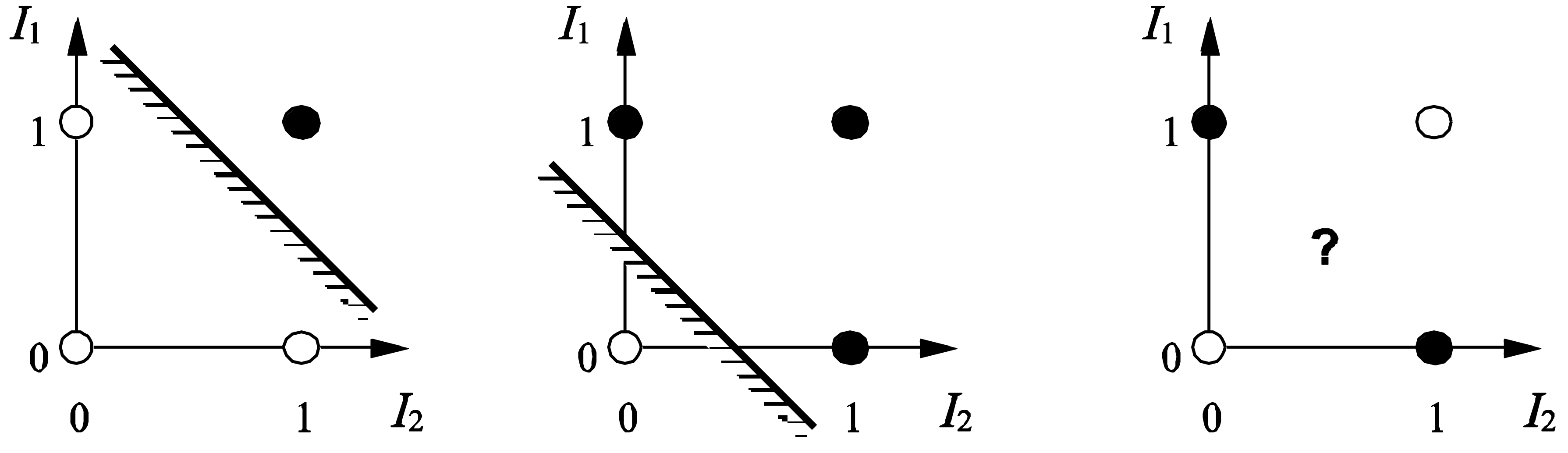
- Hypothesis space $h_{\boldsymbol{w}}$:
  - All binary classifications with parameters $\boldsymbol{w}$ s.t.

$$\boldsymbol{w}^T\overline{\boldsymbol{x}} > 0 \rightarrow +1$$
$$\boldsymbol{w}^T\overline{\boldsymbol{x}} < 0 \rightarrow -1$$

- Since $\boldsymbol{w}^T\overline{\boldsymbol{x}}$ is linear in $\boldsymbol{w}$, perceptron is called a **linear separator**

Linear Separability

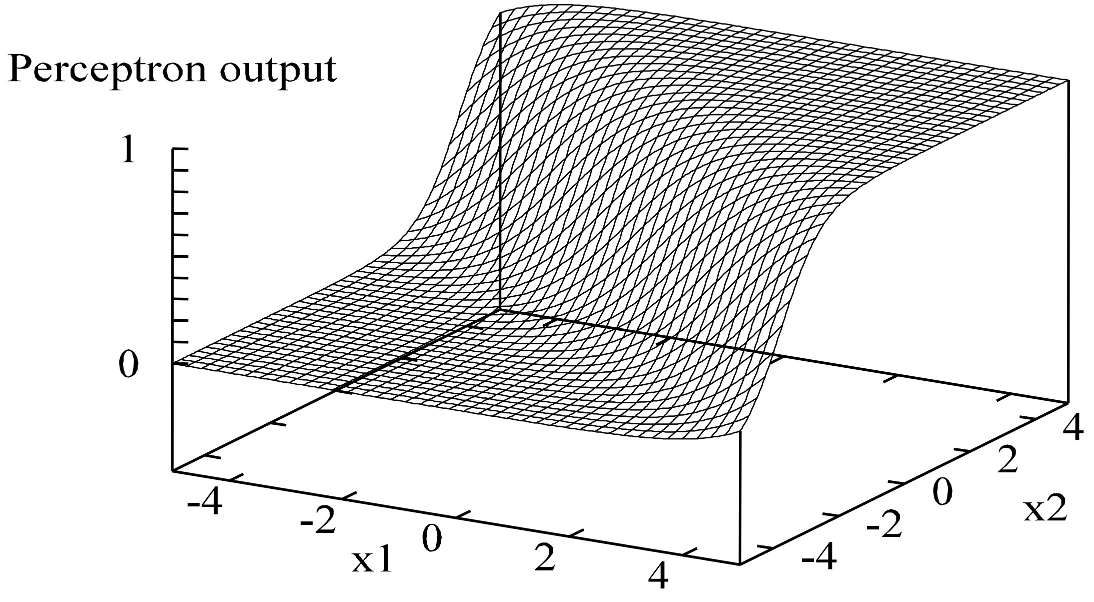
- Are all Boolean gates linearly separable?



(a) I_1 **and** I_2 (b) I_1 **or** I_2 (c) I_1 **xor** I_2

Sigmoid Perceptron

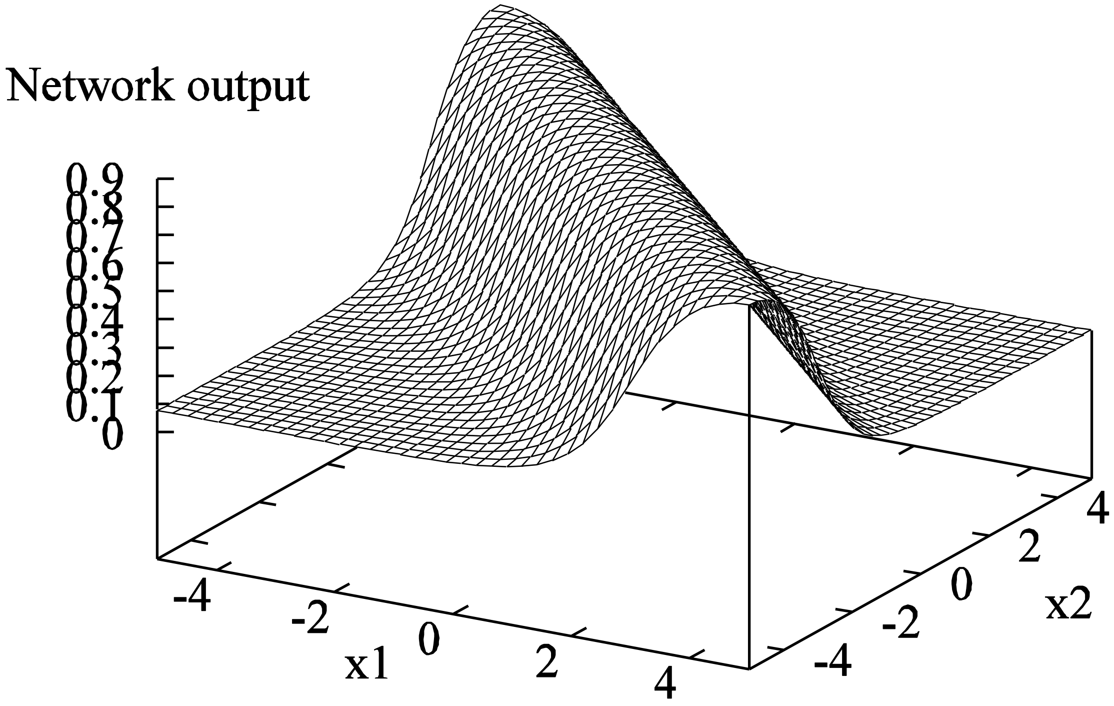
- Represent “soft” linear separators



UNIVERSITY OF WATERLOO

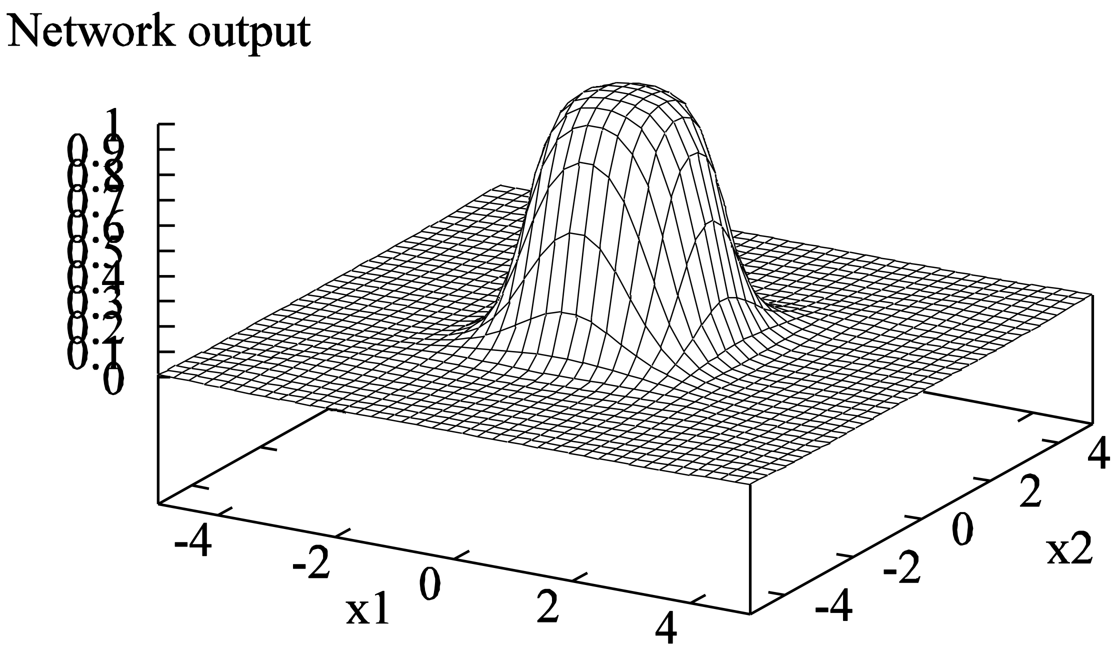
Multilayer Networks

- Adding two sigmoid units with parallel but opposite "cliffs" produces a ridge



Multilayer Networks

- Adding two intersecting ridges (and thresholding) produces a bump



Multilayer Networks

- By tiling bumps of various heights together, we can approximate any function.

- **Theorem:** Neural networks with at least one hidden layer of sufficiently many sigmoid units can approximate any function arbitrarily closely.

Common activation functions h

- Threshold: $h(a) = \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases}$
- Sigmoid: $h(a) = \sigma(a) = \frac{1}{1+e^{-a}}$
- Gaussian: $h(a) = e^{-\frac{1}{2}\left(\frac{a-\mu}{\sigma}\right)^2}$
- Tanh: $h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
- Identity: $h(a) = a$

Weight training

- Parameters: $< \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \ldots >$
- Objectives:
  - **Error minimization**
    - **Backpropagation (aka "backprop")**
  - Maximum likelihood
  - Maximum a posteriori
  - Bayesian learning

Least squared error

- Error function

$$E(\boldsymbol{W}) = \frac{1}{2}\sum_n E_n(\boldsymbol{W})^2 = \frac{1}{2}\sum_n \left|\left|f(\boldsymbol{x_n}, \boldsymbol{W}) - y_n\right|\right|_2^2$$

where $\boldsymbol{x_n}$ is the input of the n^{th} example

y_n is the label of the n^{th} example

$f(\boldsymbol{x_n}, \boldsymbol{W})$ is the output of the neural net

Sequential Gradient Descent

- For each example $(\boldsymbol{x}_n, y_n)$ adjust the weights as follows:

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E_n}{\partial w_{ji}}$$

- How can we compute the gradient efficiently given an arbitrary network structure?
- Answer: **backpropagation algorithm**
- Today: **automatic differentiation**

Backpropagation Algorithm

- Two phases:
  - Forward phase: compute output z_j of each unit j
  - Backward phase: compute delta δ_j at each unit j

Forward phase

- Propagate inputs forward to compute the output of each unit
- Output z_j at unit j:

$$z_j = h(a_j) \quad \text{where} \quad \mathrm{a}_j = \sum_i w_{ji} z_i$$

Backward phase

- Use chain rule to recursively compute gradient
  - For each weight w_{ji}: $\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$
  - Let $\delta_j \equiv \frac{\partial E_n}{\partial a_j}$ then

$$\delta_j = \begin{cases} h'(a_j)\left(z_j - y_j\right) & \text{base case: } j \text{ is an output unit} \\ h'(a_j)\sum_k w_{kj}\delta_k & \text{recursion: } j \text{ is a hidden unit} \end{cases}$$

  - Since $a_j = \sum_i w_{ji} z_i$ then $\frac{\partial a_j}{\partial w_{ji}} = z_i$

Simple Example

- Consider a network with two layers:
  - Hidden nodes: $h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
    - Tip: $tanh'(a) = 1 - (tanh(a))^2$
  - Output node: $h(a) = a$

- Objective: squared error

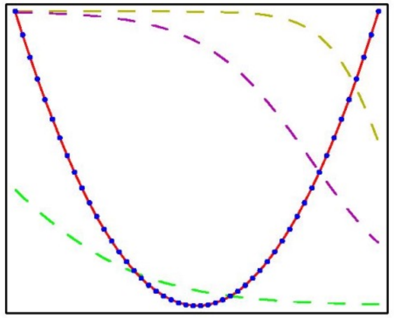
Simple Example

- Forward propagation:
  - Hidden units: $a_j =$ $z_j =$
  - Output units: $a_k =$ $z_k =$
- Backward propagation:
  - Output units: $\delta_k =$
  - Hidden units: $\delta_j =$
- Gradients:
  - Hidden layers: $\frac{\partial E_n}{\partial w_{ji}} =$
  - Output layer: $\frac{\partial E_n}{\partial w_{kj}} =$

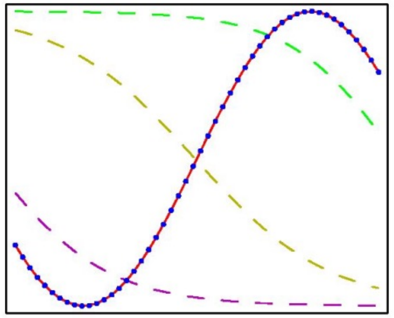
Non-linear regression examples

- Two-layer network:
  - 3 tanh hidden units and 1 identity output unit

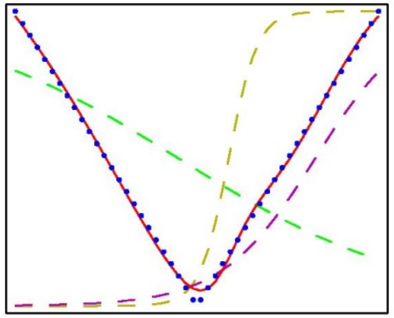
$y = x^2$



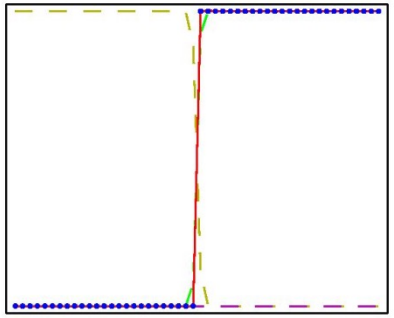
$y = \sin x$



$y = |x|$



$y = \int_{-\infty}^{x} \delta(t)dt$



Analysis

- Efficiency:
  - Fast gradient computation: linear in number of weights
- Convergence:
  - Slow convergence (linear rate)
  - May get trapped in local optima
- Prone to overfitting
  - Solutions: early stopping, regularization (add $||w||_2^2$ penalty term to objective), dropout