Machine Learning for Intelligent Systems

Lecture 13: Deep Neural Networks

Reading: UML 20-20.3

Instructors: Nika Haghtalab (this time) and Thorsten Joachims

(Stochastic) Gradient Descent

Many learning problems can be written as the following optimization on the sample set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$.

$$\min_{\vec{w}} \mathcal{L}_S(\vec{w}) \quad \mathit{for} \quad \mathcal{L}_S(\vec{w}) = R(\vec{w}) + C\frac{1}{n}\sum_{i=1}^{n} L(\vec{w}\cdot\vec{x}_i, y_i)$$

Gradient Descent Update: $\vec{w}^{(t+1)} \leftarrow \vec{w}^{(t)} - \eta_t \nabla \mathcal{L}_S\left(\vec{w}^{(t)}\right)$

$$\vec{w}^{(t+1)} \leftarrow \vec{w}^{(t)} - \eta_t \nabla R(\vec{w}) - \frac{\eta_t \mathrm{C}}{n}\sum_{i=1}^{n} \nabla L\left(\vec{w}^{(t)}\cdot\vec{x}_i, y_i\right)$$

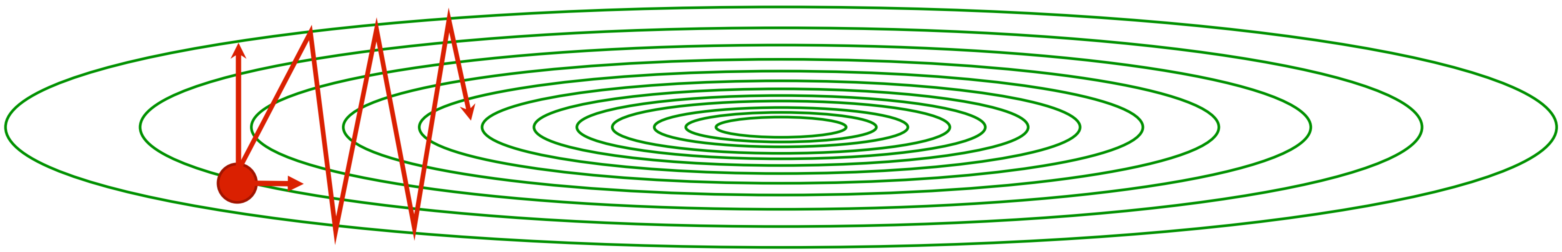
Stochastic Gradient Descent Update: Take a random $(\vec{x}_i, y_i) \sim S$

$$\vec{w}^{(t+1)} \leftarrow \vec{w}^{(t)} - \eta_t \nabla R(\vec{w}) - \eta_t C\, \nabla L\left(\vec{w}^{(t)}\cdot\vec{x}_i, y_i\right)$$

AdaGrad

Adaptive **Grad**ient**:**

- Adapt the learning rate for each parameter based on the previous gradients.



For all coordinates i:

Derivative $$g_{i,t} = \frac{\partial \mathcal{L}(\vec{w}^{(t)})}{\partial w_i}$$

Update $$w_i^{(t+1)} \leftarrow w_i^{(t)} - \eta \frac{g_{i,t}}{\sqrt{\underbrace{0.01}_{\text{Stabilizer}} + \underbrace{\sum_{\tau=1}^{t} (g_{i,\tau})^2}_{\text{Sum-square of derivatives}}}}$$

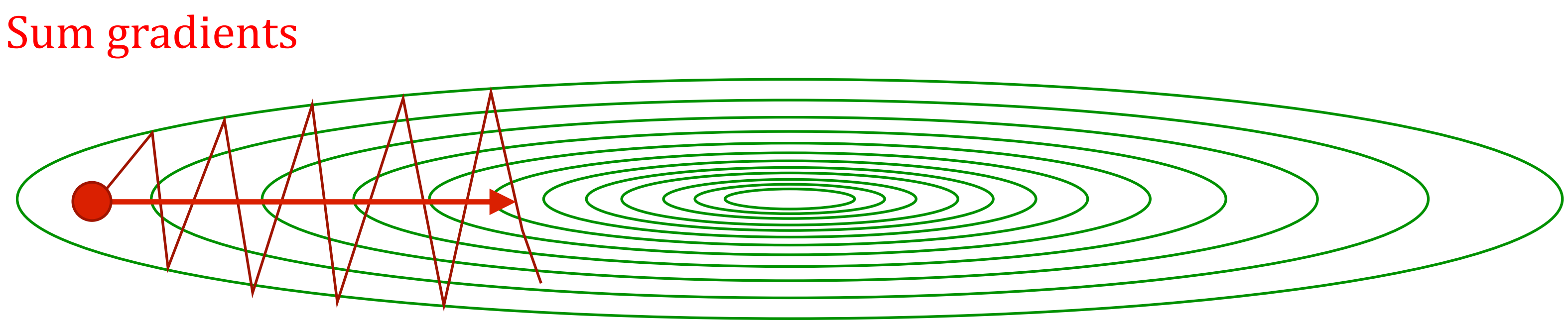
Stabilizer

Sum-square of derivatives

Momentum Method

Adaptive Gradient:

- Use previous gradients to encourage movement in important directions.

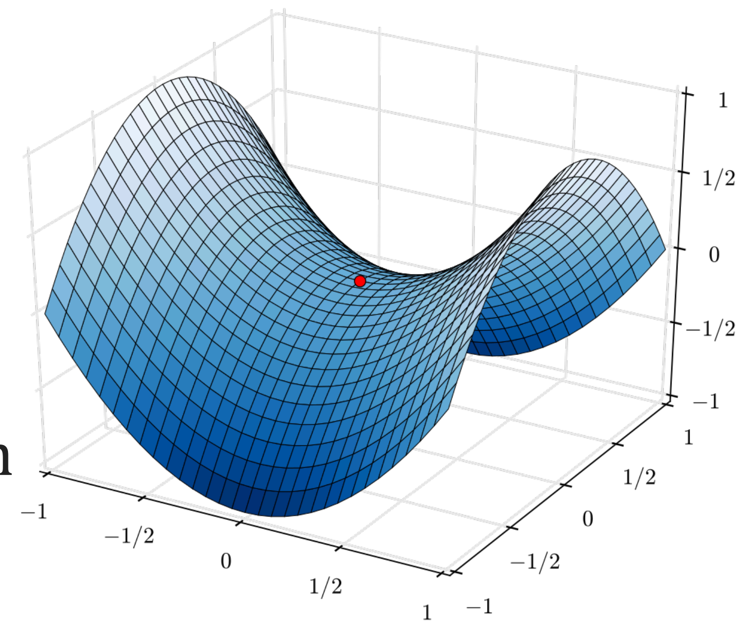


Exp-weighted Average Gradient $\vec{G}^{(t)} \leftarrow (1-\beta)\,\vec{G}^{(t-1)} + \beta \nabla \mathcal{L}(\vec{w}^{(t)})$

Update $\vec{w}^{(t+1)} \leftarrow \vec{w}^{(t)} - \eta \vec{G}^{(t)}$

SGD on Non-Convex

Non convex functions are challenging

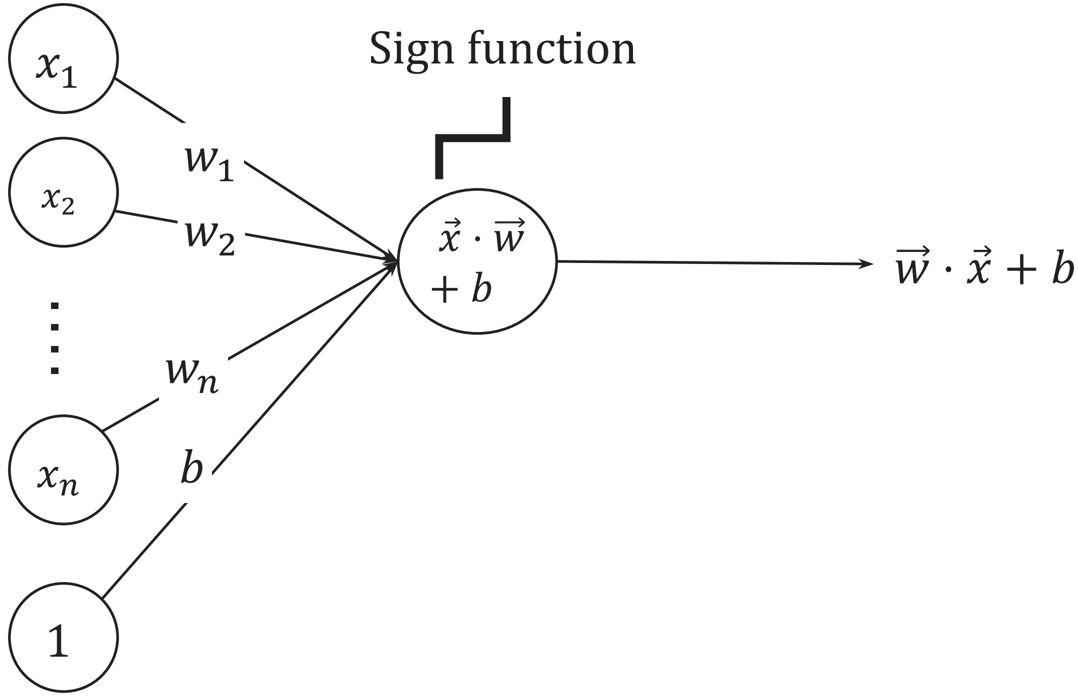


Under specific assumptions SGD provably converges to a local minimum

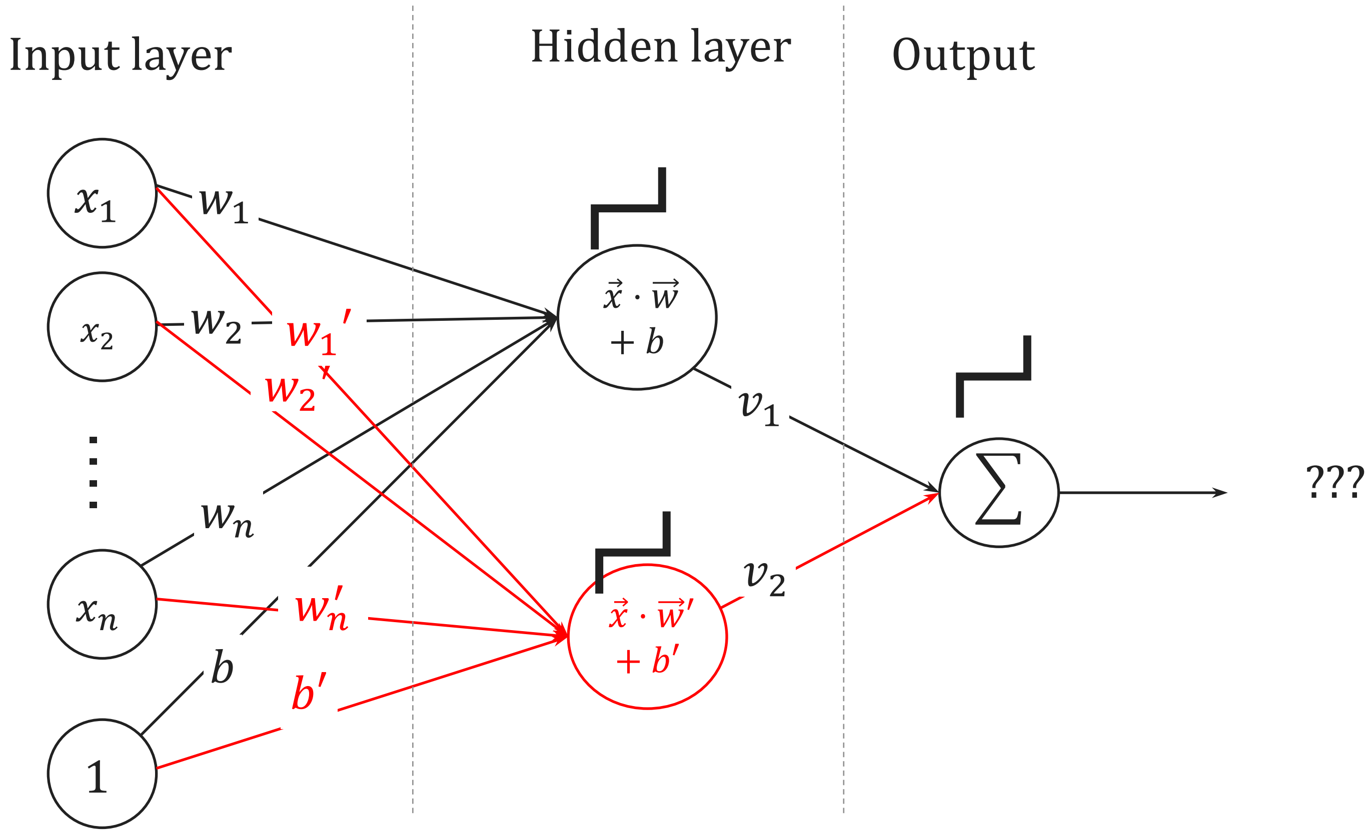
Neural networks are non-convex and SGD is used for training them.

Linear Model

We can represent a linear function as single layer neural network.



Naïve Hidden Layers



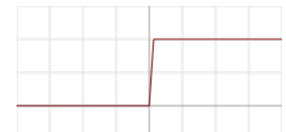
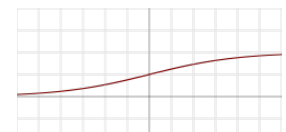
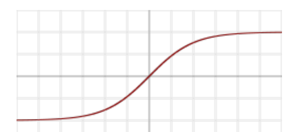
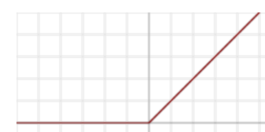
Linear function of linear functions, is linear.

$$v_1(\vec{x} \cdot \vec{w} + b) + v_2(\vec{x} \cdot \vec{w}' + b') = \vec{x} \cdot (v_1\vec{w} + v_2\vec{w}') + (v_1 b + v_2 b')$$

Beyond linearity: We need each layer to transform a linear function to something else.

Common Activation Functions

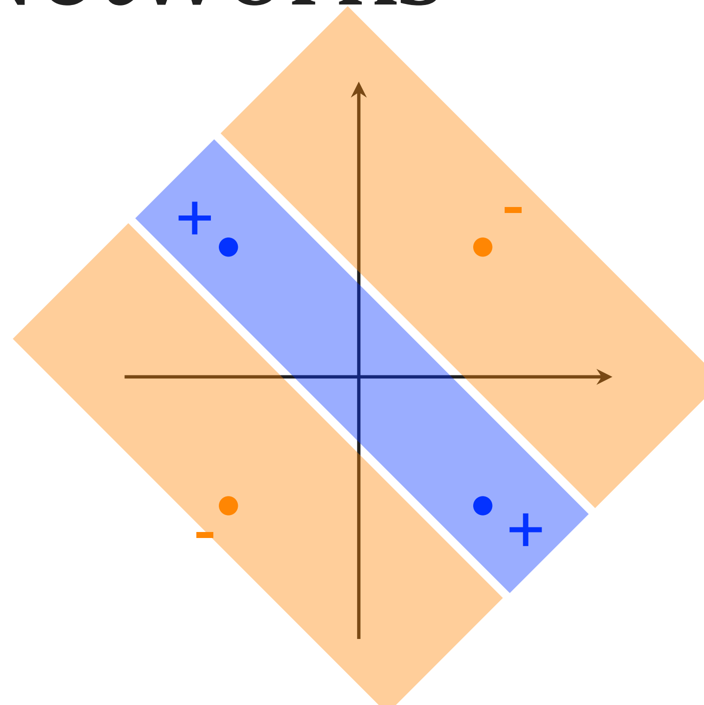
Use a non-linear activation function on nodes of a hidden layer.

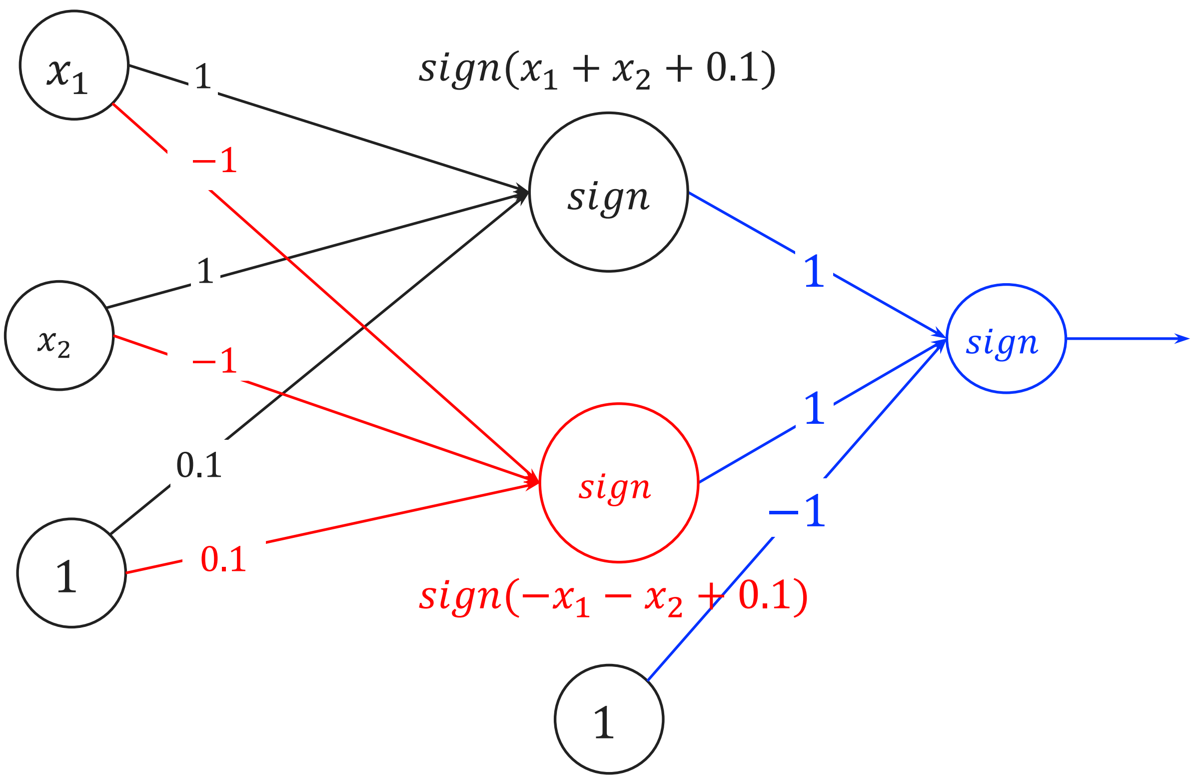
Name	Function	Gradient	Graph
Binary step	$\text{sign}(x)$	$\begin{cases} 0 & x \neq 0 \\ N/A & x = 0 \end{cases}$	
sigmoid	$\sigma(x) = \dfrac{1}{1 + \exp(-x)}$	$\sigma(x)(1 - \sigma(x))$	
Tanh	$tanh(x) = \dfrac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$	$\left(1 - tanh(x)\right)^2$	
Rectified Linear (ReLu)	$relu(x) = \max(x, 0)$	$\begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$	

Sometime, $\sigma(x)$ denotes the "generic" notion of activation function, not necessarily sigmoid.

Power of Neural Networks

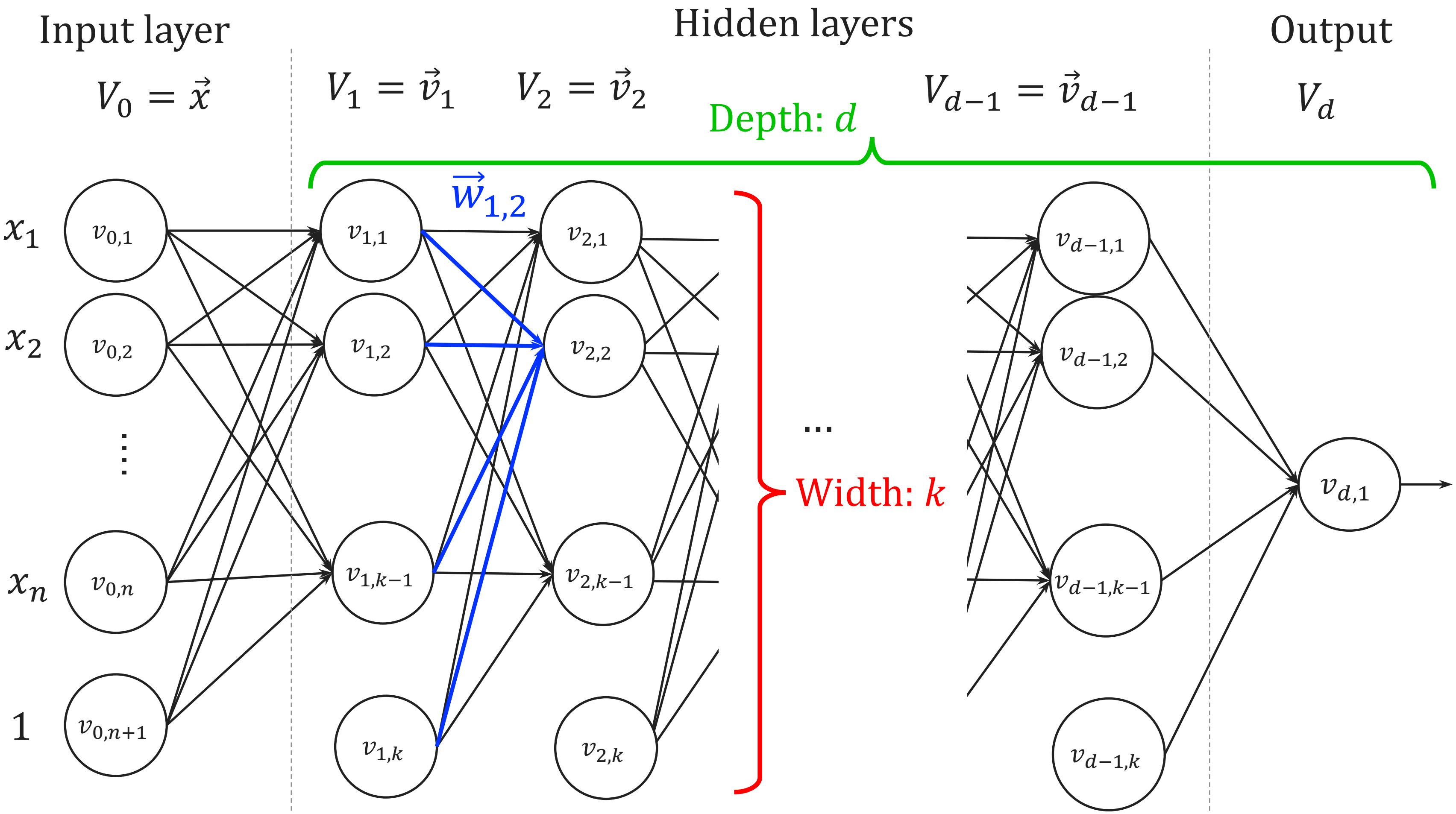
Represent XOR with 1 hidden layer.





$$sign(\ sign(x_1 + x_2 + 0.1) + sign(-x_1 - x_2 + 0.1) - 1)\ )$$

Multi Layer Neural Network



Vector of weights going from layer i to the j^{th} node of layer $i+1$: $\vec{w}_{i,j}$

Vector of values in layer $i+1$, $\vec{v}_{i+1} = \begin{pmatrix} \sigma(\vec{v}_i \cdot \vec{w}_{i,1}) \\ \vdots \\ \sigma(\vec{v}_i \cdot \vec{w}_{i,k}) \end{pmatrix}$

Output: $\sigma(v_d)$

Universal Approximators

If we allow a single hidden layer (depth 2 network) with very large width, we can approximate any continuous function on $\mathbb{R}^n$.

How large?

- For boolean functions, we need at least $\exp(n)$ width.
- Restricting ourselves to polynomial size networks
- → Can't approximate all functions
- → Reduce the chance of overfitting

} Bias-Variance Tradeoff

Other Types of Neural Networks

- Traditional multi-layer networks:

→Layers are fully connected

→Bad for overfitting

Other types

- Convolutional Neural Networks (CNNs)

→Some structured layers to learn features

→1-2 layers of fully connected network at the end

- Recurrent Neural Networks (RNNs)

→ Nodes can feed forward or backward.