Machine Learning for Intelligent Systems

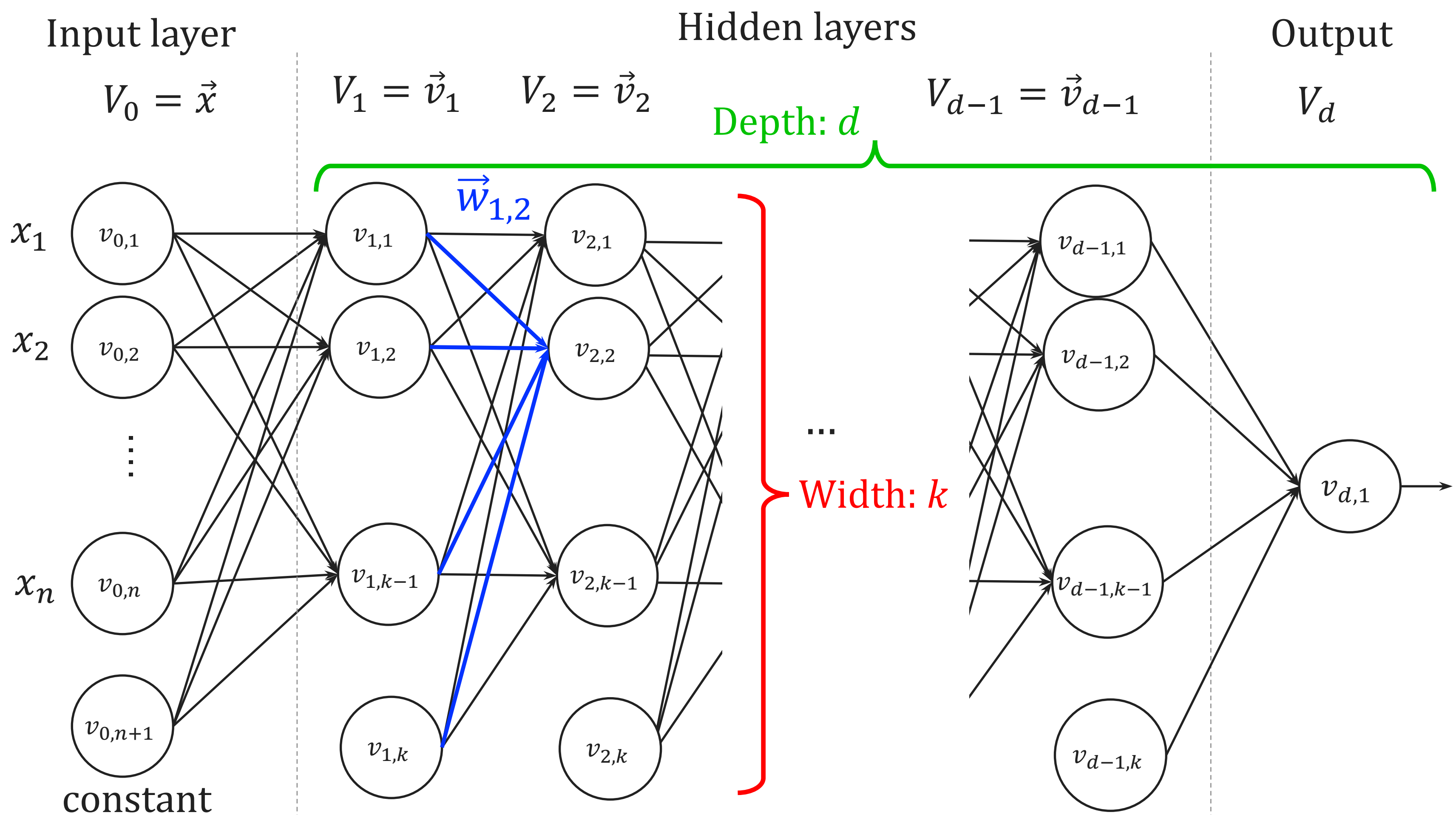
Lecture 14: Backpropagation in Networks

Reading: UML 20.6

The slides are altered from class to use column matrices throughout.

Instructors: Nika Haghtalab (this time) and Thorsten Joachims

Multi Layer Neural Network



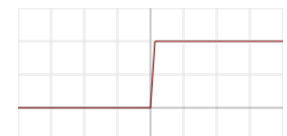
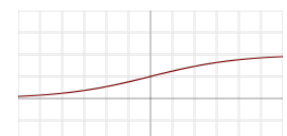
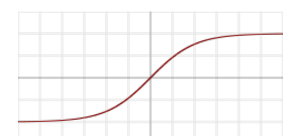
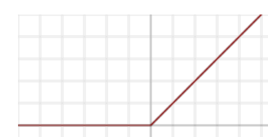
Vector of weights going from layer i to the j^{th} node of layer $i+1$: $\vec{w}_{i,j}$

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

Output: $v_{d,1}$

Common Activation Functions

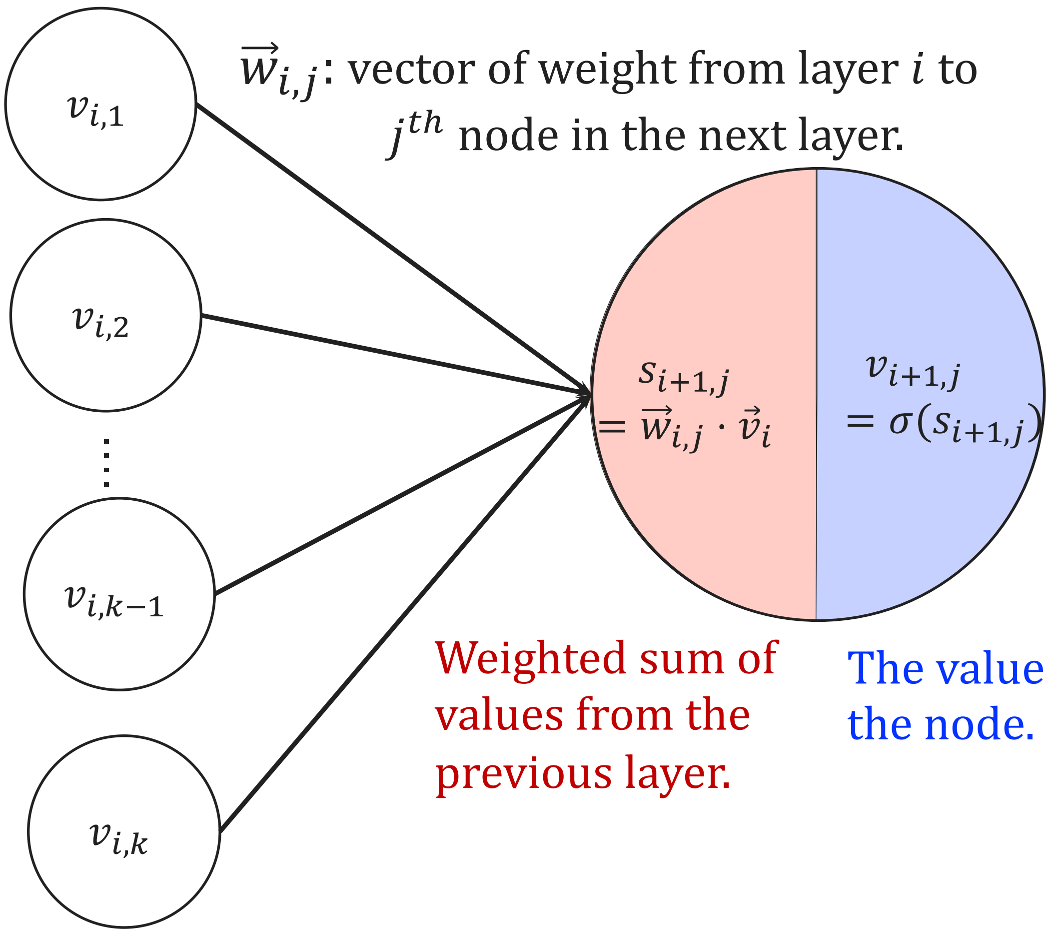
Use a non-linear activation function on nodes of a hidden layer.

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	$\tanh(x) = \dfrac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$	$1 - \tanh^2(x)$	
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.

Closer look at the nodes

$\vec{w}_{i,j}$: vector of weight from layer i to j^{th} node in the next layer.



Weighted sum of values from the previous layer.

The value of the node.

Vectorized form:

$$\vec{v}_{i+1} = \begin{pmatrix} \sigma(\vec{v}_i \cdot \vec{w}_{i,1}) \\ \vdots \\ \sigma(\vec{v}_i \cdot \vec{w}_{i,k-1}) \\ c \end{pmatrix}$$

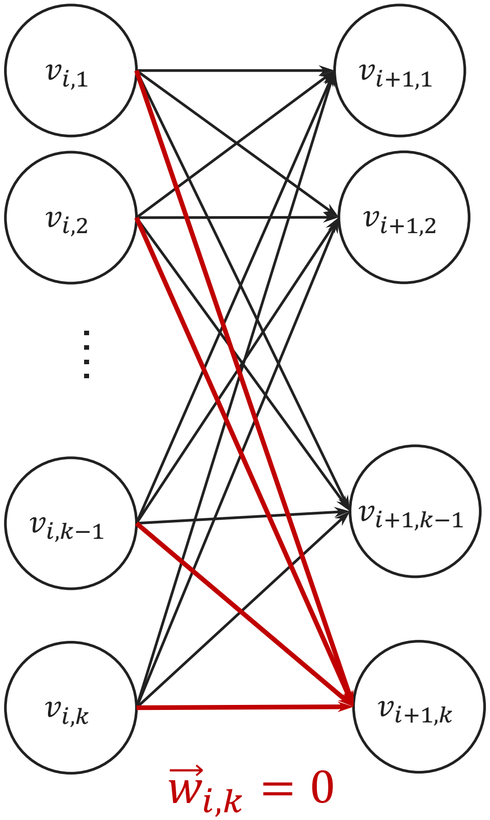
Bias term, has no incoming weights, so it's a fixed constant.

Simpler form

Vectorized form:

$$\vec{v}_{i+1} = \begin{pmatrix} \sigma(\vec{v}_i \cdot \vec{w}_{i,1}) \\ \vdots \\ \sigma(\vec{v}_i \cdot \vec{w}_{i,k-1}) \\ 1/2 \end{pmatrix}$$

Bias term, has no incoming weights.



Presentation Trick:

→ When σ is sigmoid, $\sigma(0) = 1/2$.

→ We can assume that $\vec{w}_{i,k} = 0$

→ The network is fully connected.

$$\rightarrow \vec{v}_{i+1} = \sigma(W_i^{\mathrm{T}}\, \vec{v}_i)$$

Prediction with Neural Networks

How do we use Neural Networks for predictions?

$$\sigma\left(W_{d-1}^{T} \ldots\ \sigma\left(W_2^{\mathrm{T}}\ \sigma\left(W_1^{\mathrm{T}} \sigma(W_0^{\mathrm{T}}\ \vec{v}_0)\right)\right) \ldots .\right)$$

Forward Pass (Prediction)

Input: Neural Network with weight matrices $W_0, W_1, \ldots, W_{d-1}$
and instance $(\vec{x}, y)$

$\vec{v}_0 = \vec{x}$

For $\ell = 1, \ldots, d$

- $\vec{s}_\ell = \mathrm{W}_{\ell-1}^{\mathrm{T}}\ \vec{v}_{\ell-1}$
- $\vec{v}_\ell = \sigma(\vec{s}_\ell)$

End For

Output $\vec{v}_d$

Learning the Weights

How can we learn weight matrices $W_0, W_1, \dots, W_{d-1}$ given an sample set $S = \{(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m)\}$.

Need to write the optimization:

$$\min_{W_0,\dots,W_{d-1}} \frac{1}{m} \sum_{i=1\dots m} L(\text{forward pass}_{W_0,\dots,W_{d-1}}(\vec{x}_i), y_i)$$

For convenience, we don't use a regularizer in this lecture.

$L(y', y_i)$: Evaluates how good is y' as a prediction for y_i. We use

$$L(y', y_i) = \frac{1}{2}(y' - y_i)^2$$

SGD for Neural Networks

How can we learn weight matrices $W_0, W_1, \dots, W_{d-1}$ given an sample set $S = \{(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m)\}$.

Stochastic Gradient Descent updates rule:

$$\min_{W_0,\dots,W_{d-1}} \frac{1}{m} \sum_{i=1\dots m} L(\text{forward pass}_{W_0,\dots,W_{d-1}}(x), y_i)$$

Take a random $(\vec{x}, y)$ from the samples, let ∂L denote the partial derivate for this sample with respect to the current weights

For all $i = 0, \dots, d-1, \qquad W_i \leftarrow W_i - \eta \frac{\partial L}{\partial W_i}$

Derivative of the network

We need to compute the derivates $\frac{\partial L}{\partial W_i}$

Chain rule in derivates:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g(x)} \cdot \frac{\partial g(x)}{\partial h(x)} \cdot \frac{\partial h(x)}{\partial x}$$

(Element wise) Application of chain rule:

$$\frac{\partial L}{\partial w_{i,j}(z)} = \frac{\partial L}{\partial v_{i+1,j}} \cdot \frac{\partial v_{i+1,j}}{\partial s_{i+1,j}} \cdot \frac{\partial s_{i+1,j}}{\partial w_{i,j}(z)}$$

$$= \frac{\partial L}{\partial v_{i+1,j}} \sigma'\left(s_{i+1,j}\right) \quad v_{i,z}$$

Derivative wrt Node Values

We need to compute the derivates $\frac{\partial L}{\partial v_{i,j}}$

For v_d, it's just the derivate of the loss with respect to the actual label y. For square loss:

$$\frac{\partial L}{\partial v_d} = v_d - y$$

Application of chain rule:

$$\frac{\partial L}{\partial v_{d-1,1}} = \frac{\partial L}{\partial v_d} \cdot \frac{\partial v_d}{\partial s_d} \cdot \frac{\partial s_d}{\partial v_{d-1,1}}$$

$$= (v_d - y) . \sigma'(s_d) . w_{d-1,1}(1)$$

Derivative wrt Node Values

More generally, recursively

$$\frac{\partial L}{\partial \vec{v}_i} = \frac{\partial L}{\partial \vec{v}_{i+1}} \cdot \frac{\partial \vec{v}_{i+1}}{\partial \vec{s}_{i+1}} \cdot \frac{\partial \vec{s}_{i+1}}{\partial \vec{v}_i}$$

$$= W_i \left(\sigma'(\vec{s}_{i+1}) \odot \frac{\partial L}{\partial \vec{v}_{i+1}} \right)$$

Element-wise matrix multiplication: $\begin{pmatrix} a \\ \vdots \\ b \end{pmatrix} \odot \begin{pmatrix} c \\ \vdots \\ d \end{pmatrix} = \begin{pmatrix} ac \\ \vdots \\ bd \end{pmatrix}$

All together

We can compute all the derivates $\frac{\partial L}{\partial \vec{v}_i}$ before each round of SGD.

$$\frac{\partial L}{\partial v_d} = v_d - y$$

$$\frac{\partial L}{\partial \vec{v}_i} = W_i \left(\frac{\partial L}{\partial \vec{v}_{i+1}} \odot \sigma'(\vec{s}_{i+1}) \right)$$

When we start, we now have the derivates with respect to weights.

$$\frac{\partial L}{\partial W_i} = \vec{v}_i \left(\frac{\partial L}{\partial \vec{v}_{i+1}} \odot \sigma'(\vec{s}_{i+1}) \right)^T$$

Back Propagation

Each Round of SGD:
Forward Pass then Back Propagation

Forward Pass (Prediction)

$\vec{v}_0 = (\vec{x}, 1)$

For $\ell = 1, \dots, d$

- $\vec{s}_\ell = \mathrm{W}_{\ell-1}^{\mathrm{T}} \, \vec{v}_{\ell-1}$
- $\vec{v}_\ell = \sigma(\vec{s}_\ell)$

End For

Output $\vec{s}_0, \vec{v}_0, \dots, \vec{s}_d, \vec{v}_d$

Backward Pass

$\vec{\delta}_d = \frac{\partial L}{\partial v_d} \sigma'(s_d)$

For $\ell = d - 1 \ \dots, 0$

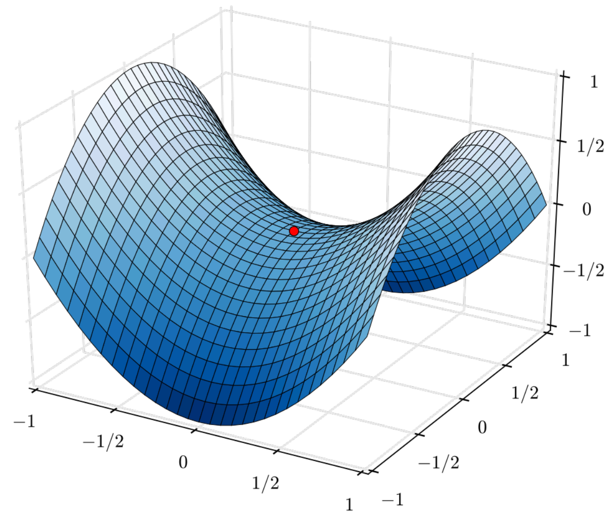
- $\mathrm{W}_\ell = \mathrm{W}_\ell - \eta \, \vec{v}_\ell \, \vec{\delta}_{\ell+1}^{\mathrm{T}}$
- $\vec{\delta}_\ell = (W_\ell \vec{\delta}_{\ell+1}) \odot \sigma'(\vec{s}_\ell)$

End For

Output $\mathrm{W}_0, \mathrm{W}_1, \dots \mathrm{W}_{\mathrm{d}-1}$

SGD on Non-Convex

The loss in neural networks is non-convex.



Many of the theoretical guarantees of SGD only hold for convex losses.

1. Initialization.

→Don't start $W_0, W_1, \ldots, W_{d-1} = 0$

→ Randomized your starting weights.

2. Run separate SGDs and take the best.