MSCI 546: Advanced Machine Learning

Sirisha Rambhatla

University of Waterloo

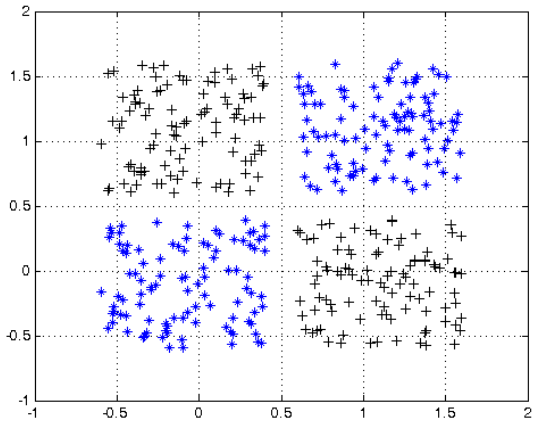
Lecture

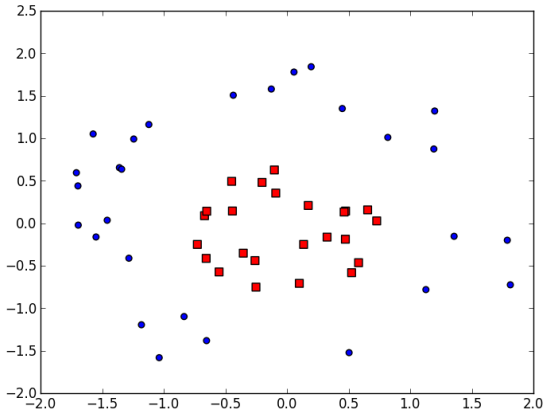
Feed-Forward Neural Networks

Outline

Outline

Linear models are not always adequate





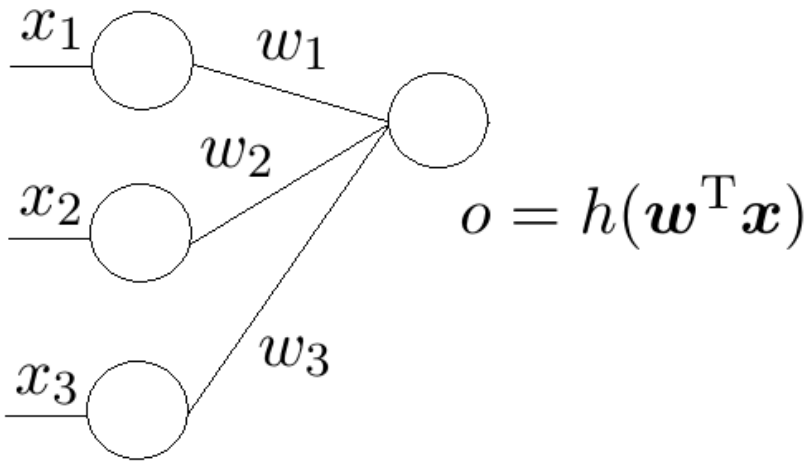
We can use a nonlinear mapping:

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^{\mathsf{D}} \rightarrow \boldsymbol{z} \in \mathbb{R}^{\mathsf{M}}$$

But what kind of nonlinear mapping ϕ should be used? Can we actually learn this nonlinear mapping?

THE most popular nonlinear models nowadays: **neural nets**

Linear model as a one-layer neural net



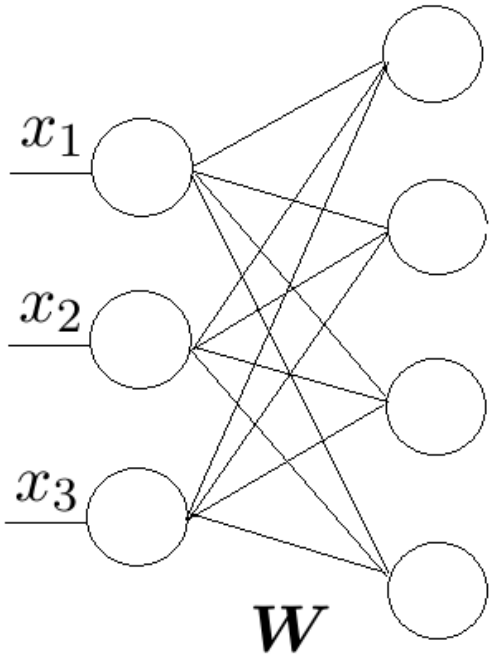
$h(a) = a$ for linear model

To create non-linearity, can use

- Rectified Linear Unit (**ReLU**): $h(a) = \max\{0, a\}$
- sigmoid function: $h(a) = \frac{1}{1+e^{-a}}$
- TanH: $h(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$
- many more

Outline

More output nodes

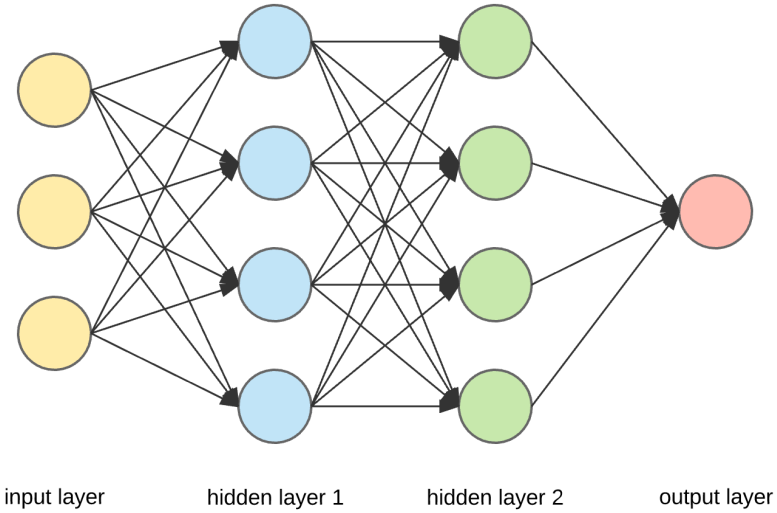


$$\boldsymbol{o} = \boldsymbol{h}(\boldsymbol{W}\boldsymbol{x})$$

$\boldsymbol{W} \in \mathbb{R}^{4\times 3}$, $\boldsymbol{h} : \mathbb{R}^4 \to \mathbb{R}^4$ so $\boldsymbol{h}(\boldsymbol{a}) = (h_1(a_1), h_2(a_2), h_3(a_3), h_4(a_4))$

Can think of this as a nonlinear basis: $\boldsymbol{\Phi}(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{W}\boldsymbol{x})$

More layers



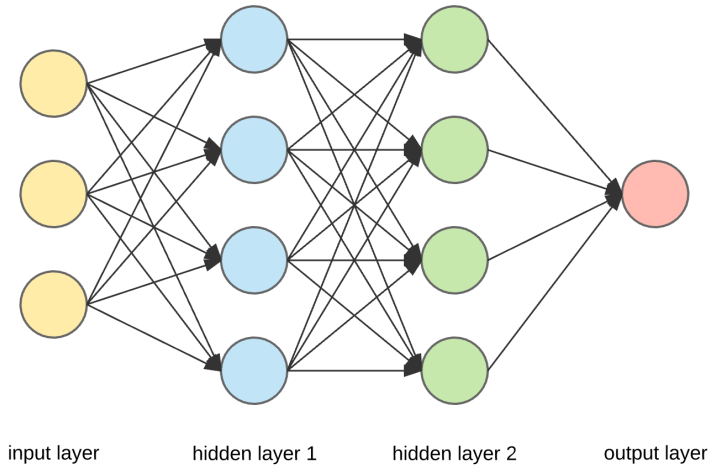
Becomes a network:

- each node is called a **neuron**
- $\boldsymbol{h}$ is called the **activation function**
  - can use $h(a) = 1$ for one neuron in each layer to *incorporate bias term*
  - output neuron can use $h(a) = a$
- #layers refers to #hidden_layers (plus 1 or 2 for input/output layers)
- **deep** neural nets can have many layers and *millions* of parameters
- this is a **feedforward, fully connected** neural net, there are many variants

Math formulation

An L-layer neural net can be written as

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{h}_{\mathsf{L}}\left(\boldsymbol{W}_L \boldsymbol{h}_{\mathsf{L}-1}\left(\boldsymbol{W}_{L-1} \cdots \boldsymbol{h}_1\left(\boldsymbol{W}_1 \boldsymbol{x}\right)\right)\right)$$



To ease notation, for a given input $\boldsymbol{x}$, define recursively

$$\boldsymbol{o}_0 = \boldsymbol{x}, \qquad \boldsymbol{a}_\ell = \boldsymbol{W}_\ell \boldsymbol{o}_{\ell-1}, \qquad \boldsymbol{o}_\ell = \boldsymbol{h}_\ell(\boldsymbol{a}_\ell) \qquad (\ell = 1, \ldots, \mathsf{L})$$

where

- $\boldsymbol{W}_\ell \in \mathbb{R}^{\mathsf{D}_\ell \times \mathsf{D}_{\ell-1}}$ is the weights between layer $\ell - 1$ and ℓ
- $\mathsf{D}_0 = \mathsf{D}, \mathsf{D}_1, \ldots, \mathsf{D}_{\mathsf{L}}$ are numbers of neurons at each layer
- $\boldsymbol{a}_\ell \in \mathbb{R}^{\mathsf{D}_\ell}$ is input to layer ℓ
- $\boldsymbol{o}_\ell \in \mathbb{R}^{\mathsf{D}_\ell}$ is output to layer ℓ
- $\boldsymbol{h}_\ell : \mathbb{R}^{\mathsf{D}_\ell} \to \mathbb{R}^{\mathsf{D}_\ell}$ is activation functions at layer ℓ

Outline

Learning the model

No matter how complicated the model is, our goal is the same: minimize

$$\mathcal{E}(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_\mathsf{L}) = \frac{1}{N} \sum_{n=1}^{\mathsf{N}} \mathcal{E}_n(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_\mathsf{L})$$

where

$$\mathcal{E}_n(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_\mathsf{L}) = \begin{cases} \|\boldsymbol{f}(\boldsymbol{x}_n) - \boldsymbol{y}_n\|_2^2 & \text{for regression} \\ \ln\left(1 + \sum_{k \neq y_n} e^{f(\boldsymbol{x}_n)_k - f(\boldsymbol{x}_n)_{y_n}}\right) & \text{for classification} \end{cases}$$

Learning the model

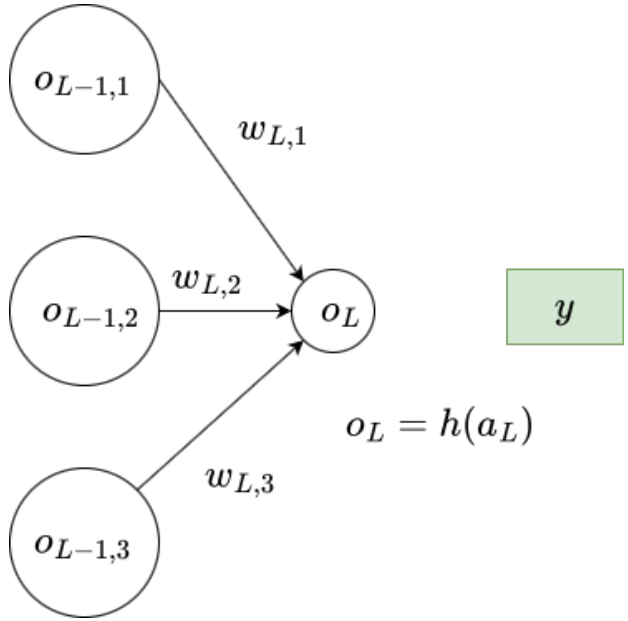
Let's take the regression task:

$$\mathcal{E}(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_{\mathsf{L}}) = \frac{1}{N} \sum_{n=1}^{\mathsf{N}} \|\boldsymbol{f}(\boldsymbol{x}_n) - \boldsymbol{y}_n\|_2^2$$

where the goal is to learn the function $\boldsymbol{f}(\cdot)$

Learning the model: How to optimize such a complicated function?

The function $\boldsymbol{f}(\cdot)$ corresponds to the output of the neural network o_L. Due to the non-linear activations $\boldsymbol{f}(\cdot)$ is in general *nonconvex*, and we use **SGD**!



Note that the output is our regression task is a scalar, therefore o_L is a scalar, but in general the output can be a vector. There is also a bias term, but we ignore it for brevity.

Analyzing the last layer for gradient descent

$$\boldsymbol{o}_0 = \boldsymbol{x}, \qquad \boldsymbol{a}_\ell = \boldsymbol{W}_\ell \boldsymbol{o}_{\ell-1}, \qquad \boldsymbol{o}_\ell = \boldsymbol{h}_\ell(\boldsymbol{a}_\ell) \qquad (\ell = 1, \ldots, \mathsf{L})$$

For the last layer, for square loss-based regression case we are considering, we can evaluate the gradient w.r.t. $w_{\mathsf{L},1}$ as follows

$$\frac{\partial \mathcal{E}_n}{\partial w_{\mathsf{L},1}} = \frac{\partial \mathcal{E}_n}{\partial o_{\mathsf{L}}} \frac{\partial o_{\mathsf{L}}}{\partial a_{\mathsf{L}}} \frac{\partial a_{\mathsf{L}}}{\partial w_{\mathsf{L},1}} \qquad \textbf{(Chain Rule)}$$

$$\frac{\partial \mathcal{E}_n}{\partial w_{\mathsf{L},1}} = \frac{\partial (o_{\mathsf{L}} - y_n)^2}{\partial o_{\mathsf{L}}} \frac{\partial o_{\mathsf{L}}}{\partial a_{\mathsf{L}}} \frac{\partial a_{\mathsf{L}}}{\partial w_{\mathsf{L},1}} = 2(o_{\mathsf{L}} - y_n) h'_{\mathsf{L}}(a_{\mathsf{L}}) o_{\mathsf{L}-1,1}$$

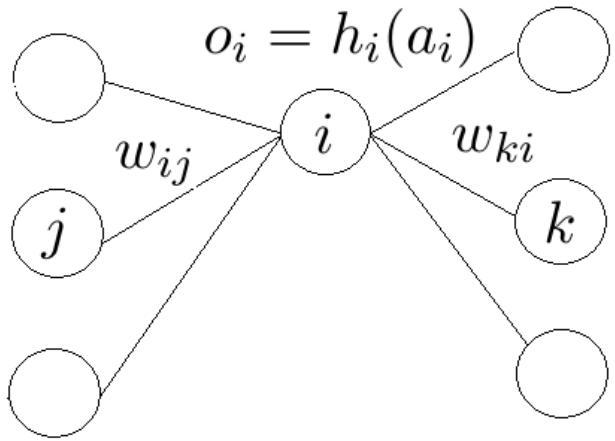
We will follow a similar procedure for remaining weights $w_{2,\mathsf{L}}$ and $w_{3,\mathsf{L}}$, which gives us gradient w.r.t. all weights.

Notice how the weights in the last layer here constitute a vector. Is this always the case? **No!** If the output layer is a vector instead of a scalar, we will have a matrix!

Computing the gradients for inner layers

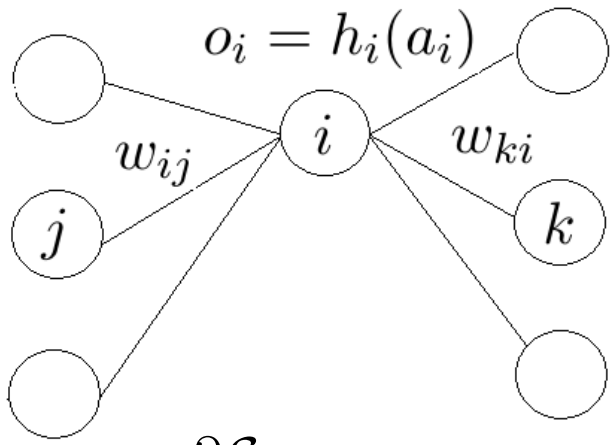
We essentially do a similar application of chain rule

- We move from output layers to inner layers, and use the evaluated gradients as we move inward



Computing the derivative

$$\boldsymbol{o}_0 = \boldsymbol{x}, \qquad \boldsymbol{a}_\ell = \boldsymbol{W}_\ell \boldsymbol{o}_{\ell-1}, \qquad \boldsymbol{o}_\ell = \boldsymbol{h}_\ell(\boldsymbol{a}_\ell) \qquad (\ell = 1, \ldots, \mathsf{L})$$



Drop the subscript ℓ for layer for simplicity.

Find the **derivative of $\mathcal{E}_n$ w.r.t. to w_{ij}**

$$\frac{\partial \mathcal{E}_n}{\partial w_{ij}} = \overbrace{\frac{\partial \mathcal{E}_n}{\partial o_i}\frac{\partial o_i}{\partial a_i}}^{\frac{\partial \mathcal{E}_n}{\partial a_i}} \frac{\partial a_i}{\partial w_{ij}} = \frac{\partial \mathcal{E}_n}{\partial o_i}\frac{\partial o_i}{\partial a_i}\frac{\partial (w_{ij} o_j)}{\partial w_{ij}} = \overbrace{\frac{\partial \mathcal{E}_n}{\partial o_i} h_i'(a_i)}^{\frac{\partial \mathcal{E}_n}{\partial a_i}} o_j$$

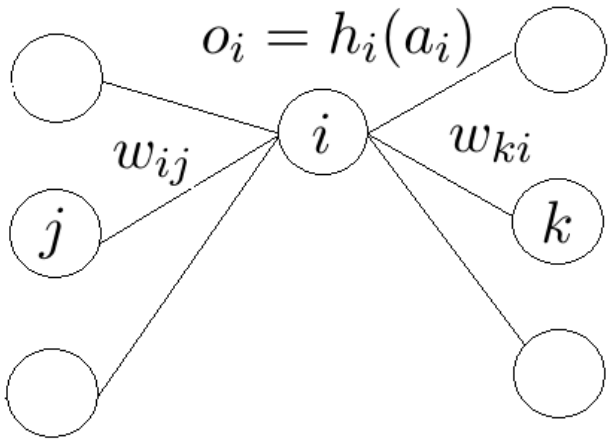
$$\frac{\partial \mathcal{E}_n}{\partial o_i} = \left(\sum_k \frac{\partial \mathcal{E}_n}{\partial a_k}\frac{\partial a_k}{\partial o_i} \right) = \left(\sum_k \frac{\partial \mathcal{E}_n}{\partial a_k} w_{ki} \right)$$

Computing the derivative

Adding the subscript for layer:

$$\frac{\partial \mathcal{E}_n}{\partial w_{\ell,ij}} = \frac{\partial \mathcal{E}_n}{\partial a_{\ell,i}} o_{\ell-1,j}$$

$$\frac{\partial \mathcal{E}_n}{\partial a_{\ell,i}} = \left(\sum_k \frac{\partial \mathcal{E}_n}{\partial a_{\ell+1,k}} w_{\ell+1,ki} \right) h'_{\ell,i}(a_{\ell,i})$$



For the last layer, for square loss

$$\frac{\partial \mathcal{E}_n}{\partial a_{\mathsf{L},i}} = \frac{\partial (h_{\mathsf{L},i}(a_{\mathsf{L},i}) - y_{n,i})^2}{\partial a_{\mathsf{L},i}} = 2(h_{\mathsf{L},i}(a_{\mathsf{L},i}) - y_{n,i}) h'_{\mathsf{L},i}(a_{\mathsf{L},i})$$

Computing the derivative

Using **matrix notation** greatly simplifies presentation and implementation:

$$\frac{\partial \mathcal{E}_n}{\partial \boldsymbol{W}_\ell} = \frac{\partial \mathcal{E}_n}{\partial \boldsymbol{a}_\ell} \boldsymbol{o}_{\ell-1}^\top$$

$$\frac{\partial \mathcal{E}_n}{\partial \boldsymbol{a}_\ell} = \begin{cases} \left(\boldsymbol{W}_{\ell+1}^\top \frac{\partial \mathcal{E}_n}{\partial \boldsymbol{a}_{\ell+1}} \right) \circ \boldsymbol{h}'_\ell(\boldsymbol{a}_\ell) & \text{if } \ell < \mathsf{L} \\ 2(\boldsymbol{h}_\mathsf{L}(\boldsymbol{a}_\mathsf{L}) - \boldsymbol{y}_n) \circ \boldsymbol{h}'_\mathsf{L}(\boldsymbol{a}_\mathsf{L}) & \text{else} \end{cases}$$

where $\boldsymbol{v}_1 \circ \boldsymbol{v}_2 = (v_{11}v_{21}, \cdots, v_{1\mathsf{D}}v_{2\mathsf{D}})$ is the element-wise product (a.k.a. Hadamard product)

Putting everything into SGD

The **backpropagation** algorithm (**Backprop**)

Initialize $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_{\mathsf{L}}$. Repeat:

1. randomly pick one data point $n \in [\mathsf{N}]$
2. **forward propagation**: for each layer $\ell = 1, \ldots, \mathsf{L}$
   - compute $\boldsymbol{a}_\ell = \boldsymbol{W}_\ell \boldsymbol{o}_{\ell-1}$ and $\boldsymbol{o}_\ell = \boldsymbol{h}_\ell(\boldsymbol{a}_\ell)$ $\qquad (\boldsymbol{o}_0 = \boldsymbol{x}_n)$
3. **backward propagation**: for each $\ell = L, \ldots, 1$
   - compute

$$\frac{\partial \mathcal{E}_n}{\partial \boldsymbol{a}_\ell} = \begin{cases} \left(\boldsymbol{W}_{\ell+1}^\top \frac{\partial \mathcal{E}_n}{\partial \boldsymbol{a}_{\ell+1}}\right) \circ \boldsymbol{h}_\ell'(\boldsymbol{a}_\ell) & \text{if } \ell < \mathsf{L} \\ 2(\boldsymbol{h}_{\mathsf{L}}(\boldsymbol{a}_{\mathsf{L}}) - \boldsymbol{y}_n) \circ \boldsymbol{h}_{\mathsf{L}}'(\boldsymbol{a}_{\mathsf{L}}) & \text{else} \end{cases}$$

   - update weights

$$\boldsymbol{W}_\ell \leftarrow \boldsymbol{W}_\ell - \eta \frac{\partial \mathcal{E}_n}{\partial \boldsymbol{W}_\ell} = \boldsymbol{W}_\ell - \eta \frac{\partial \mathcal{E}_n}{\partial \boldsymbol{a}_\ell} \boldsymbol{o}_{\ell-1}^\top$$

More tricks to optimize neural nets

Many variants based on backprop

- SGD with **minibatch**: randomly sample a batch of examples to form a stochastic gradient
- SGD with **momentum**
- ...

References

- Relevant sections of Chapter 13 (I built my notes from various sources for simplicity)