Neural Networks and Backpropagation

Vibhav Gogate

THE UNIVERSITY OF TEXAS AT DALLAS
Erik Jonsson School of Engineering and Computer Science

Recap: Gradient Descent Rules for Linear Classifiers

Error driven: Sigmoid Approximation $o = \sigma(\sum_i w_i x_i)$, $y \in \{0, 1\}$

$$w_i = w_i + \alpha \sum_{k=1}^{d} \left(y^{(k)} - o(\mathbf{x}^{(k)}) \right) o(\mathbf{x}^{(k)})(1 - o(\mathbf{x}^{(k)}))x_i^{(k)}$$

Error driven: Tanh Approximation $t = \tanh(\sum_i w_i x_i)$, $y \in \{-1, 1\}$

$$w_i = w_i + \alpha \sum_{k=1}^{d} \left(y^{(k)} - t(\mathbf{x}^{(k)}) \right) (1 - [t(\mathbf{x}^{(k)})]^2)x_i^{(k)}$$

Error driven: Linear Approximation $l = \sum_i w_i x_i$, $y \in \{-1, 1\}$

$$w_i = w_i + \alpha \sum_{k=1}^{d} \left(y^{(k)} - l(\mathbf{x}^{(k)}) \right) x_i^{(k)}$$

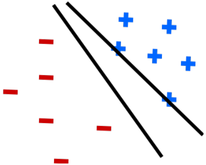
Probability driven: LR $o = \sigma(\sum_i w_i x_i)$, Y is binary.

$$w_i = w_i + \alpha \sum_{k=1}^{d} \left(y^{(k)} - o(\mathbf{x}^{(k)}) \right) x_i^{(k)}$$

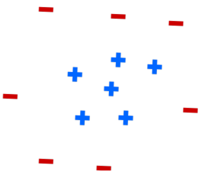
Linear Classifiers: Properties

- Good: Fast Optimization and optimality guarantee. In many cases, we in fact find the best possible "linear classifier" with respect to standard error measures.
- Good: When the number of features (dimensions) is larger than (or roughly the same as) the number of examples, they work amazingly well. Later on, we will see a formal proof for this using VC-dimensions.
- Bad: Limited expressive power. In practice, most datasets will need non-linear classifiers.

Linearly Separable



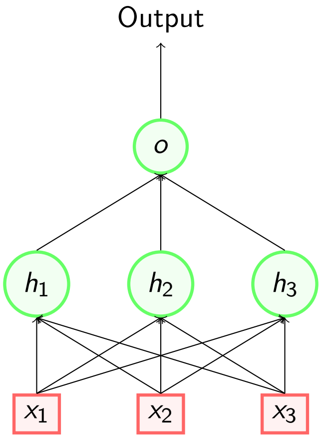
Not Linearly Separable



Neural Networks

Idea: Build a network of Linear Classifiers

- We will get a non-linear classifier if we construct a feed-forward network of linear classifiers.
  - Input layer: features in the data
  - Hidden layers: linear classifiers with output of nodes in the previous layer (including the input layer) as input
  - Output layer: Desired output node output of nodes in the hidden layer a level below as input
- Have to be careful because network of linear functions is a linear function.



What function does a Neural Network Represent?

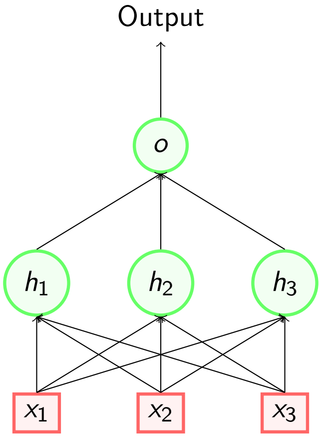
Some terminology: Linear unit outputs $\sum_i w_i a_i$, sigmoid unit outputs $\sigma(\sum_i w_i a_i)$, tanh unit outputs $\tanh(\sum_i w_i a_i)$ and threshold unit outputs $\text{sign}(\sum_i w_i a_i)$

- Assume that each hidden node and output node is a sigmoid unit
- Then the given neural network represents

$$o = \sigma\left(\sum_{i=1}^{n} w_{o,i} h_i\right)$$

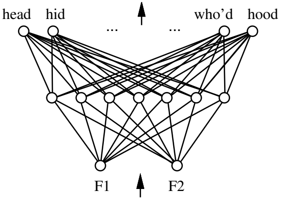
$$\text{where } h_i = \sigma\left(\sum_{j=1}^{3} w_{i,j} x_j\right)$$

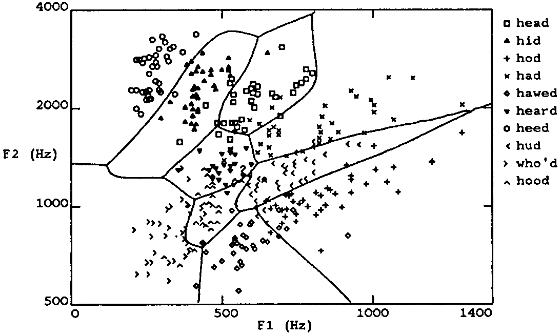
- Fun fact: Each edge is associated with a parameter.



Non-Linearity

Multi-layer perceptrons or neural networks having sigmoid hidden units represent a non-linear function.



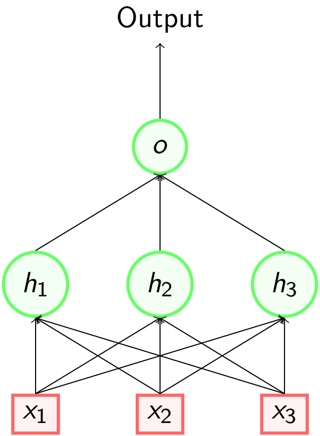


Learning = Rep. + Eval. measure + Optimization

Let us be error driven

$$E = \sum_{k=1}^{d} \left(y^{(k)} - \sigma \left(\sum_{i=0}^{n} w_{o,i} h_i^{(k)} \right) \right)^2 \text{ where } h_i^{(k)} = \sigma(\sum_{j=1}^{3} w_{i,j} x_j^{(k)})$$

- Optimization task: Find parameters $w_{o,1}$, $w_{o,2}$, $w_{o,3}$, $w_{1,1}$, $w_{1,2}$, $w_{1,3}$, $w_{2,1}$, $w_{2,2}$, $w_{2,3}$, $w_{3,1}$ $w_{3,2}$ and $w_{3,3}$ such that E is optimized.
- Algorithm: Take the gradient of E with respect to each parameter and run gradient descent.



Gradients for Stochastic Gradient Descent

Given data point (x_1, x_2, x_3, y)

$$
\begin{aligned}
\frac{\partial E}{\partial w_{o,i}} &= \frac{\partial}{\partial w_{o,i}}(y-o)^2 \\
&= 2(y-o)\left(-\frac{\partial}{\partial w_{o,i}} o\right) \\
&= -2(y-o)o(1-o)\left(\frac{\partial}{\partial w_{o,i}} \sum_{j=1}^{3} w_{o,j} h_j\right) \\
&= -2(y-o)o(1-o)h_i
\end{aligned}
$$

Gradients for Stochastic Gradient Descent

Given data point (x_1, x_2, x_3, y)

$$
\begin{aligned}
\frac{\partial E}{\partial w_{i,j}} &= \frac{\partial}{\partial w_{i,j}}(y-o)^2 \\
&= 2(y-o)\left(-\frac{\partial}{\partial w_{i,j}} o\right) \\
&= -2(y-o)o(1-o)\left(\frac{\partial}{\partial w_{i,j}} \sum_{a=1}^{3} w_{o,a} h_a\right) \\
&= -2(y-o)o(1-o)w_{o,i}\frac{\partial}{\partial w_{i,j}} h_i \\
&= -2(y-o)o(1-o)w_{o,i}h_i(1-h_i)\left(\frac{\partial}{\partial w_{i,j}} \sum_{a=1}^{3} w_{i,a} x_a\right) \\
&= -2(y-o)o(1-o)w_{o,i}h_i(1-h_i)x_j
\end{aligned}
$$

Dynamic Programming

Idea: Store intermediate results.

$$\frac{\partial E}{\partial w_{o,i}} = -(y - o)o(1 - o)h_i$$

$$\frac{\partial E}{\partial w_{i,j}} = -(y - o)o(1 - o)w_{o,i}h_i(1 - h_i)x_j$$

- For $w_{o,1}$, $w_{o,2}$ and $w_{o,3}$, the term $(y - o)o(1 - o)$ is the same. Let us call it δ_o. Then the gradient $\frac{\partial E}{\partial w_{o,i}}$ is $-\delta_o h_i$
- For $w_{1,1}$, $w_{1,2}$ and $w_{1,3}$ the term $(y - o)o(1 - o)w_{o,1}h_1(1 - h_1)$ is the same. Let $\delta_1 = \delta_o w_{o,1}h_1(1 - h_1)$. Then the gradient equals $-\delta_1 x_j$
- In general, $\frac{\partial E}{\partial w_{i,j}} = -\delta_i x_j$.

Dynamic Programming: Backpropagation

$$\frac{\partial E}{\partial w_{o,i}} = -\delta_o h_i \text{ where } \delta_o = (y - o)o(1 - o)$$

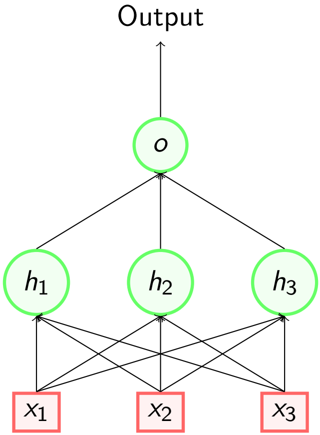
$$\frac{\partial E}{\partial w_{i,j}} = -\delta_i x_j \text{ where } \delta_i = \delta_o w_{o,i} h_i(1 - h_i)$$

Repeat Until Convergence
For each example $(\mathbf{x}, y)$ do

- Send $(\mathbf{x}, y)$ through the network and compute o and h_i's for all i
- For the output unit o, compute $\delta_o = (y - o)o(1 - o)$
- For each hidden unit h_i, compute $\delta_i = \delta_o h_i(1 - h_i)w_{o,i}$
- Update all weights $w_{o,i}$ using $w_{o,i} = w_{o,i} + \alpha\delta_o h_i$
- Update all weights $w_{i,j}$ using $w_{i,j} = w_{i,j} + \alpha\delta_i x_j$

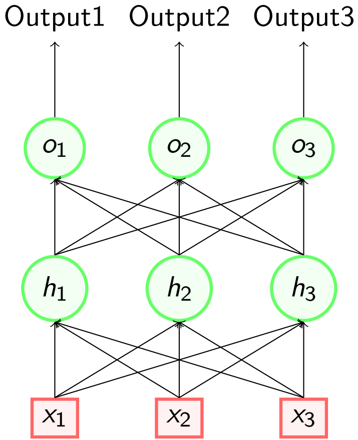
Understanding Backpropagation as Message Passing

- Forward pass: Send example to the hidden nodes and compute h_1, h_2 and h_2
- Forward pass: Send h_1, h_2 and h_3 to o and compute o
- Backward pass: Node o sends the message, δ_o to h_1, h_2 and h_3. Each h_i updates the weights using δ_o and the following equation $w_{o,i} = w_{o,i} + \alpha\delta_o h_i$.
- Backward pass: Each node h_i sends the message, δ_i to each x_j. Each x_j updates the weights using δ_i and the following equation $w_{i,j} = w_{i,j} + \alpha\delta_i x_j$.



Fun Exercise as Homework

Derive the backpropagation algorithm for the following network.
The only change: we have multiple output nodes.



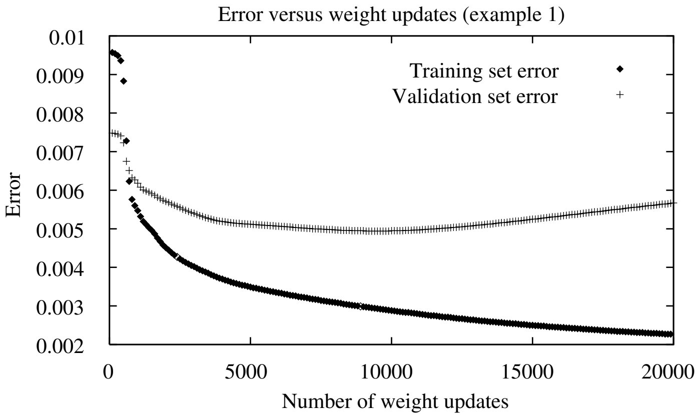
More on Backpropagation

- Gradient descent over entire *network* weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum
  - In practice, often works well (can run multiple times)
- Often include weight *momentum* α

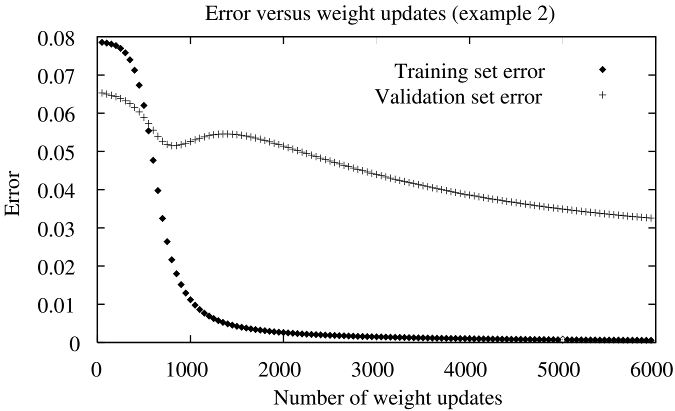
  $$\Delta w_{i,j}(n) = \alpha \delta_j x_{i,j} + \eta \Delta w_{i,j}(n-1)$$

  where $\Delta w_{i,j}(n)$ is the gradient of E w.r.t. $w_{i,j}$ at iteration n.
- Minimizes error over *training* examples
  - Will it generalize well to subsequent examples?
- Training can take thousands of iterations $\rightarrow$ slow!
- Using network after training is very fast

Overfitting in Neural Networks: #1



Overfitting in Neural Networks: #2



Overfitting Avoidance

- Penalize large weights:

$$\text{Error} + \text{ L2 Regularizer} : E + \lambda \sum_{i,j} w_{i,j}^2$$

- Early Stopping
- Tie together weights (Parameter sharing):
  - e.g., in phoneme recognition network

Representation Revisited

Expressive power of Neural networks

Boolean functions:

- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

Representing Boolean Functions: AND

Let $\{X_1, \ldots, X_n\}$ be the binary input attributes taking values from the set $\{0, 1\}$.

Question: Can you represent the following AND function using a Threshold unit.

$$f(X_1, \ldots, X_n) = \begin{cases} +1 & X_1 \wedge \ldots \wedge X_k \wedge \neg X_{k+1} \wedge \ldots \wedge \neg X_n \text{ is true} \\ -1 & \text{Otherwise} \end{cases}$$

Answer: Yes. $w_0 = -k + 0.5$; $w_1 = \ldots = w_k = 1$ and $w_{k+1} = \ldots = w_n = -1$. The output of this perceptron will be $+1$ if f is true and -1 otherwise.

If X_i's take values from the set $\{+1, -1\}$ instead of $\{0, 1\}$ then we can represent the AND function using a perceptron (with sign unit) having the following weights: $w_0 = -n + 0.5$; $w_1 = \ldots = w_k = 1$ and $w_{k+1} = \ldots = w_n = -1$.

Representing Boolean Functions: OR

Let $\{X_1, \ldots, X_n\}$ be the binary input attributes taking values from the set $\{0, 1\}$.

Question: Can you represent the following OR function using a Threshold unit.

$$g(X_1, \ldots, X_n) = \begin{cases} +1 & X_1 \vee \ldots \vee X_k \vee \neg X_{k+1} \vee \ldots \vee \neg X_n \text{ is true} \\ -1 & \text{Otherwise} \end{cases}$$

Answer: Yes. We can represent this using a perceptron (with sign unit) having the following weights: $w_0 = n - k - 0.5$; $w_1 = \ldots = w_k = 1$ and $w_{k+1} = \ldots = w_n = -1$. The output of this perceptron will be $+1$ if g is true and -1 otherwise.

If X_i's take values from the set $\{+1, -1\}$ instead of $\{0, 1\}$ then we can represent the OR function using a perceptron (with sign unit) having the following weights: $w_0 = n - 0.5$; $w_1 = \ldots = w_k = 1$ and $w_{k+1} = \ldots = w_n = -1$.

Representing Arbitrary Boolean Functions

- Any Boolean function can be written either in DNF or CNF. DNF is ORs of ANDs and CNF is ANDs of ORs.
- Since we can represent ORs and ANDs using a Threshold unit, we can represent any Boolean function using the following neural network construction procedure:
  - Convert the Boolean function of a CNF. Let m be the number of clauses in the CNF.
  - Construct a neural network with one output node and one hidden layer having m hidden nodes (one per clause).
  - Connect all hidden nodes to the output node
  - Connect each hidden node to all input variables involved in the corresponding clause
  - Set the weights according to the prescription given in the previous two slides.

  (We can also use a DNF instead of a CNF)