CS F425: Deep Learning

09 Delta Training Rule Activation Function

Dr. Kamlesh Tiwari
Assistant Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

Feb 09, 2023 ON-CAMPUS Campus @ BITS-Pilani [Jan-May 2023]

http://ktiwari.in/dl

Perceptron Training (delta rule)

When data is not linearly-separable; error fluctuates with *parameter training* updates. It is difficult to decide, when to stop?

- Delta rule converges to a best-fit approximation of the target
- Uses gradient descent
- Consider unthresholded perceptron, $o(\vec{x}) = \vec{w}.\vec{x}$
- Training error is defined as

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

- Gradient would specify direction of steepest increase $\nabla E(\vec{w}) = [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n}]$
- Weights can be learned as $w_i = w_i - \eta\frac{\partial E}{\partial w_i}$
- It can be seen that $\frac{\partial E}{\partial w_i} = \sum_{d\in D}(t_d - o_d)(-x_{id})$

Perceptron Training (delta rule)

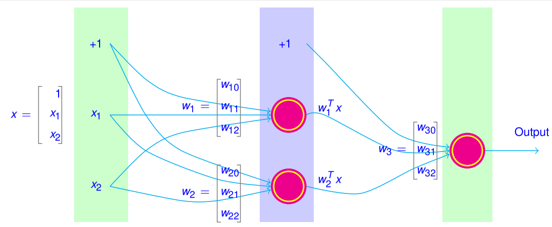
Algorithm 1: Gradient Descent (D, η)

Initialize w_i with random weights
repeat
  For each w_i, initialize $\triangle w_i = 0$
  for *each training example* $d \in D$ **do**
    Compute output o using model for d whose target is t
    For each w_i, update $\triangle w_i = \triangle w_i + \eta(t-o)x_i$
  For each w_i, set $w_i = w_i + \triangle w_i$
until *termination condition is met*;
return w

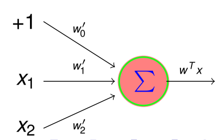
- A date item $d \in D$, is supposed to be multidimensional $d = (x_1, x_2, ..., x_n, t)$
- Algorithm converges toward the minimum error hypothesis.
- Linear programming can also be an approach

Linear Activation is Not Much Interesting

NN with perceptrons have limited capability, even with many layers



$$\begin{aligned} Output &= w_{30}\times 1 + w_{31}\times(w_1^T x) + w_{32}\times(w_2^T x)\\ &= w_{30}\times 1 + w_{31}\times[w_{10}\times 1 + w_{11}\times x_1 + w_{12}\times x_2]\\ &\quad + w_{32}\times[w_{20}\times 1 + w_{21}\times x_1 + w_{22}\times x_2]\\ &= (w_{30} + w_{31}w_{10} + w_{32}w_{20}) + (w_{31}w_{11} + w_{32}w_{21})\times x_1\\ &\quad + (w_{31}w_{12} + w_{32}w_{22})\times x_2\\ &= w_0' + w_1'\times x_1 + w_2'\times x_2 \end{aligned}$$



Expression of single perceptron

Perceptron Training (delta rule)

When data is not linearly-separable; error fluctuates with *parameter training* updates. It is difficult to decide, when to stop?

- Delta rule converges to a best-fit approximation of the target
- Uses gradient descent
- Consider unthresholded perceptron, $o(\vec{x}) = \vec{w}.\vec{x}$
- Training error is defined as

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

- Gradient would specify direction of steepest increase $\nabla E(\vec{w}) = [\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n}]$
- Weights can be learned as $w_i = w_i - \eta\frac{\partial E}{\partial w_i}$
- It can be seen that $\frac{\partial E}{\partial w_i} = \sum_{d\in D}(t_d - o_d)(-x_{id})$

Perceptron Training (delta rule)

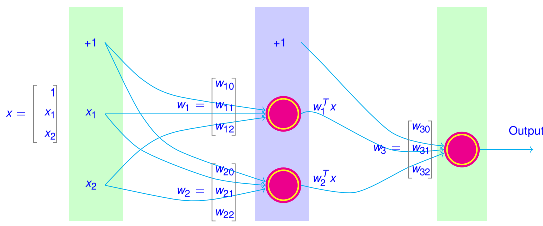
Algorithm 2: Gradient Descent (D, η)

Initialize w_i with random weights
repeat
  For each w_i, initialize $\triangle w_i = 0$
  for *each training example* $d \in D$ **do**
    Compute output o using model for d whose target is t
    For each w_i, update $\triangle w_i = \triangle w_i + \eta(t-o)x_i$
  For each w_i, set $w_i = w_i + \triangle w_i$
until *termination condition is met*;
return w

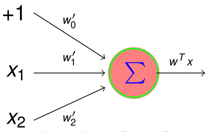
- A date item $d \in D$, is supposed to be multidimensional $d = (x_1, x_2, ..., x_n, t)$
- Algorithm converges toward the minimum error hypothesis.
- Linear programming can also be an approach

Limitation of Linear Activation

NN with perceptrons have limited capability, even with many layers



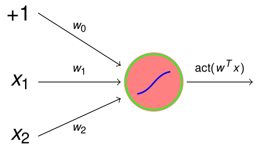
$$\begin{aligned}
Output &= w_{30} \times 1 + w_{31} \times (w_1^T x) + w_{32} \times (w_2^T x) \\
&= w_{30} \times 1 + w_{31} \times [w_{10} \times 1 + w_{11} \times x_1 + w_{12} \times x_2] \\
&\quad + w_{32} \times [w_{20} \times 1 + w_{21} \times x_1 + w_{22} \times x_2] \\
&= (w_{30} + w_{31}w_{10} + w_{32}w_{20}) + (w_{31}w_{11} + w_{32}w_{21}) \times x_1 \\
&\quad + (w_{31}w_{12} + w_{32}w_{22}) \times x_2 \\
&= w'_0 + w'_1 \times x_1 + w'_2 \times x_2
\end{aligned}$$

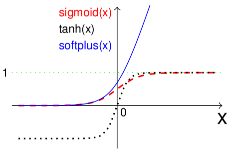


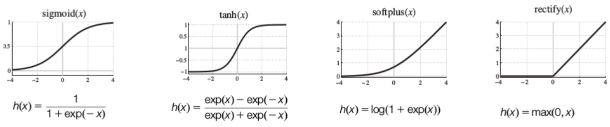
Expression of single perceptron

Neuron

Neuron uses nonlinear activation functions (*sigmoid*, *tanh*, *softplus*, *ReLU* leaky *ReLU etc.*) at the place of thresholding

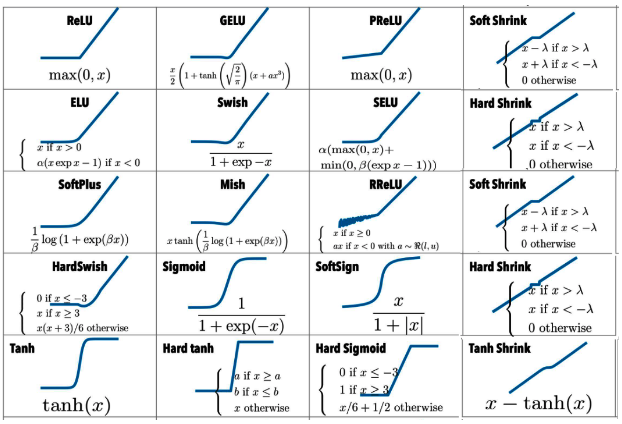






To avoid *dying reLU* problem, leaky-reLU is used *max*(0.1*x*, *x*)

More Activation Functions



Multilayer Networks and Backpropagation

- Single perceptron can only express linear decision surface
- We need units whose output is a nonlinear function of its inputs AND is also differentiable (Use Neuron not ~~Perceptron~~)

$$o(\vec{x}) = \sigma(\vec{w}.\vec{x})$$

where $\sigma(y) = \frac{1}{1+e^{-y}}$

- Backpropagation algorithm learns the weights for a fixed set of units and interconnections
- It employs gradient descent to minimize the error between the network output values and the target values for these outputs
- Let Error function is redefined as

$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2$$

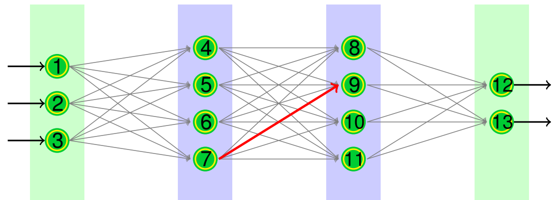
Backpropagation (for 2 layers)

Algorithm 3: Backpropagation(D, η, n_{in}, n_{out}, n_{hidden})

```
Create the feedforward network with n_in, n_out, n_hidden layers
Randomly initialize weights to small values ∈ [−0.05, +0.05]
repeat
    for each <x⃗, t⃗> ∈ D do
        o_u = get output from network ∀ unit u
        δ_k = o_k(1 − o_k)(t_k − o_k) for all output unit k
        δ_h = o_h(1 − o_h) Σ_{k∈outputs}(w_kh δ_k) for all hidden unit h
        w_ji = w_ji + △w_ji    where △w_ji = η δ_j x_ji
until converge;
```

- Recall error function is $E(\vec{w}) = \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}(t_{kd} - o_{kd})^2$
- For a single training example $E_d(\vec{w}) = \frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2$
- Weight w_{ji} is updated by adding $\triangle\, w_{ji} = -\eta\frac{\partial E_d}{\partial w_{ji}}$

Conventions Over The Network



x_{ji}	ith input to unit j (x_{94} is highlighted)
w_{ji}	weight associated with ith input to unit j
net_j	be $\sum_i w_{ji}x_{ji}$ the weighted sum of input for unit j
o_j	output computed by unit j. Consider it as $\sigma(net_j)$
t_j	target output for unit j
outputs	set of units in final layer ({12, 13} in our case)
Downstream(j)	units whose immediate input is the output of unit j

We are interested in $\frac{\partial E_d}{\partial w_{ji}}$ it is $\frac{\partial E_d}{\partial net_j} \times \frac{\partial net_j}{\partial w_{ji}}$ and therefore, $\frac{\partial E_d}{\partial net_j} \times x_{ji}$

Thank You!

Thank you very much for your attention!