Back Propagation in Netz

Nick Ewing

0.1 Descripton

The following definitions and algorithm characterize the implementation of `netz.core`.

0.2 Definitions

The following variables are used:

m: The number of training examples.

n: The number of input features for an example.

k: The number of outputs for an example.

$\{(x^i, y^i), \ldots, (x^m, y^m)\}$: The training set, composed of m input-output example pairs where $x^{(i)}$ is an n dimensional vector containing the i^{th} input and $y^{(i)}$ a k dimensional vector containing the i^{th} expected output.

X: An $m \times n$ matrix where the i^{th} row contains $(x^{(i)})^T$.

Y: An $m \times k$ matrix where the i^{th} row contains $(y^{(i)})^T$.

L: The number of layers in the network.

s_l: The number of neurons in layer l. $s_1 = n$ and $s_L = k$.

λ: The regularization constant.

α: The learning rate constant.

γ: The learning momentum constant.

$\Theta^{(l)}_{ij}$: The synapse weight between neuron i of layer l and neuron j of layer $l+1$. $\Theta^{(l)}$ is thus a $s_{(j+1)} \times s_j + 1$ matrix.

$A^{(l)}_{ij}$: The change in synapse weight between neuron i of layer l and neuron j of layer $l+1$ of the last epoch. $A^{(l)}$ is thus a $s_{(j+1)} \times s_j + 1$ matrix.

$a^{(l)}$: A vector of length $s_l + 1$ when $l < L$ and s_l when $l = L$, containing the activation values for neurons in layer l where $a^{(l)}_0$ is the bias neuron when $l < L$.

$\delta^{(l)}$: A vector of length $s_l + 1$ when $l < L$ and s_l when $l = L$, containing the back propagated error values associated with neurons in layer l.

$\Delta^{(l)}_{ij}$: The sum change from all examples in synapse weight between neuron i of layer l and neuron j of layer $l+1$. $\Delta^{(l)}$ is thus a $s_{(j+1)} \times s_j + 1$ matrix.

$D^{(l)}_{ij}$: The regularized mean change from all examples in synapse weight between neuron i of layer l and neuron j of layer $l+1$. $D^{(l)}$ is thus a $s_{(j+1)} \times s_j + 1$ matrix.

$.*$: Element-wise matrix multiplication operator.

The sigmoid activation function:

$$g(z) = \frac{1}{1 - e^{-z}} \tag{1}$$

0.3 Algorithm

$\Theta^{(j)} \leftarrow$ RandomValuedMatrix$(s_{(j+1)},\, s_j + 1)$ **for** $j = 1 \ldots L-1$
$A^{(j)} \leftarrow$ ZeroValuedMatrix$(s_{(j+1)},\, s_j + 1)$ **for** $j = 1 \ldots L-1$
repeat
  for $i \leftarrow 1 \ldots m$ **do**
    {Forward propagate input activations through network}
    $a^{(1)} \leftarrow x^{(i)}$
    $a^{(l)} \leftarrow g(\Theta^{(l-1)} a^{(l-1)})$ **for** $l = 2 \ldots L$
    {Find error of output layer from training outputs}
    $\delta^{(L)} \leftarrow a^{(L)} - y^{(i)}$
    {Back propagate error through network}
    $\delta^{(L)} \leftarrow (\Theta^{(l)})^T \delta^{(l+1)} .*\ a^{(l)} .* (1 - a^{(l)})$ **for** $l = L-1 \ldots 2$
    {Multiply errors by activations}
    $\Delta^{(l)} \leftarrow \Delta^{(l)} + \delta^{(l+1)}\ (a^{(l)})^T$ **for** $l \leftarrow 1 \ldots L-1$
  end for
  {Divide errors by the number of training examples and add in regularization}
  $D_{ij}^{(l)} \leftarrow \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta^{(l)}\mathbf{if} j \neq 0$
  $D_{ij}^{(l)} \leftarrow \frac{1}{m}\Delta_{ij}^{(l)}\mathbf{if} j = 0$
  {Adjust weights}
  for $j \leftarrow 1 \ldots L-1$ **do**
    $A^{(j)} \leftarrow \alpha D^{(j)} + \gamma A^{(j)}$
    $\Theta^{(j)} \leftarrow \Theta^{(j)} - A^{(j)}$
  end for
  $epoch \leftarrow epoch + 1$
  $MSE \leftarrow$ CalculateMeanSquareError$(\Theta,\, X,\, Y)$
until MSE $<$ desired-error **or** epoch $>$ max-epoch