Introduction to Machine Learning. CSCI-UA 9473, Zoom 2.

Augustin Cosse

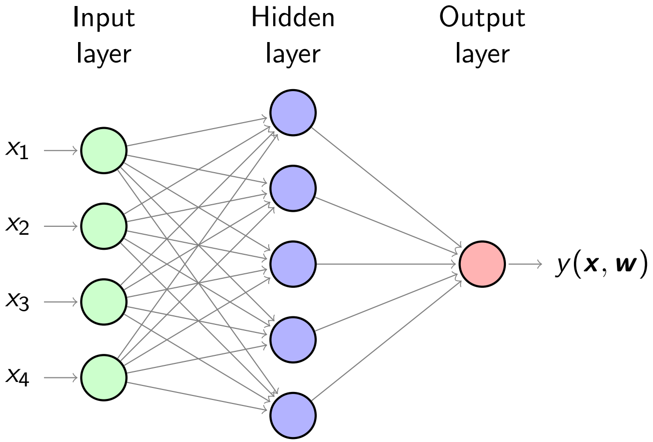
Ecole Normale Supérieure, DMA & NYU

2020

What have we seen so far?

- Linear regression
  - Solution through gradient descent
  - Solution through Normal equations
  - Regularization (Ridge, Lasso, Subset Selection)
- Linear classification
  - Separating hyperplane
  - Discriminative vs Generative classifiers
    - Logistic regression
    - Linear/Gaussian discriminant Analysis (GDA)
- Non parametric regression/classification
  - Kernel methods (use in gradient descent)
  - Support vector machines

Practical Neural Networks and Backprop



Practical Neural Networks and Backprop

- Consider the following (one hidden layer) neural network

$$y_k(\boldsymbol{x}) = \sigma\left(\sum_{j=0}^{D_2} w_{kj}^{(2)} \sigma\left(\sum_{i=1}^{D_1} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{j0}^{(2)}\right)$$

- For the moment we will consider a single binary output
- A common loss in classification with neural networks is the log loss or binary cross entropy loss

$$L(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} t_i \log y(\boldsymbol{x}; \boldsymbol{w}) + (1 - t_i)\log(1 - y(\boldsymbol{x}; \boldsymbol{w}))$$

Practical Neural Networks and Backprop

- Minimization of the binary cross entropy can be derived from the MLE.
- If we view the neural network as outputing a probability that for any point $\boldsymbol{x}_i$ the target will be one, $y(\boldsymbol{x};\boldsymbol{w}) = p$, the probability of observing a pair $\{\boldsymbol{x}_i, t_i\}$ is thus given by

$$\begin{aligned} p(\{\boldsymbol{x}_i, t_i\}) &= p^{t_i}(1-p)^{1-t_i} \\ &= (y(\boldsymbol{x}_i;\boldsymbol{w}))^{t_i}(1-y(\boldsymbol{x}_i;\boldsymbol{w}))^{1-t_i} \end{aligned}$$

- Now if we assume independent samples, we can write the probability of observing the whole dataset as

$$p(\{\boldsymbol{x}_i, t_i\}) = \prod_{i=1}^{N} (y(\boldsymbol{x}_i;\boldsymbol{w}))^{t_i}(1-y(\boldsymbol{x}_i;\boldsymbol{w}))^{1-t_i}$$

Practical Neural Networks and Backprop

$$p(\{\boldsymbol{x}_i, t_i\}) = \prod_{i=1}^{N} (y(\boldsymbol{x}_i; \boldsymbol{w}))^{t_i} (1 - y(\boldsymbol{x}_i; \boldsymbol{w}))^{1-t_i}$$

- Taking minus the log,

$$L(\boldsymbol{w}) = -\sum_{i=1}^{N} t_i \log(y(\boldsymbol{x}_i, \boldsymbol{w})) + (1 - t_i) \log(1 - y(\boldsymbol{x}_i; \boldsymbol{w}))$$

we recover the binary cross entropy loss.

Practical Neural Networks and Backprop

$$p(\{\boldsymbol{x}_i, t_i\}) = \prod_{i=1}^{N} (y(\boldsymbol{x}_i; \boldsymbol{w}))^{t_i} (1 - y(\boldsymbol{x}_i; \boldsymbol{w}))^{1-t_i}$$

- Taking minus the log,

$$L(\boldsymbol{w}) = -\sum_{i=1}^{N} t_i \log(y(\boldsymbol{x}_i, \boldsymbol{w})) + (1 - t_i) \log(y(\boldsymbol{x}_i; \boldsymbol{w}))$$

we recover the binary cross entropy loss.

Backpropagation

- To learn the weights, we will adopt an SGD approach.
- We will use the notation a to encode the activations (input to each neuron). E.g. for the inputs to the neurons of the first hidden layer, we have

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}$$

- We will use the notation z to denote the output of each neuron (output to the activation functions),

$$z_j = \sigma(a_j)$$

Backpropagation

- The derivative of the binary cross entropy with respect to y can be computed easily (for one sample),

$$\begin{aligned}\frac{dL(\boldsymbol{w})}{dy} &= \frac{-t_i}{y(\boldsymbol{x}_i;\boldsymbol{w})} + \frac{1-t_i}{1-y(\boldsymbol{x}_i;\boldsymbol{w})}\\ &= \frac{y(\boldsymbol{x}_i;\boldsymbol{w})-t_i}{y(\boldsymbol{x}_i;\boldsymbol{w})(1-y(\boldsymbol{x}_i;\boldsymbol{w}))}\end{aligned}$$

- A common choice for the activation function is the sigmoid (as in logistic regression). In this case we have

$$y_k(\boldsymbol{x}_i;\boldsymbol{w}) = \sigma(a_k) = \frac{1}{1+e^{-a_k}}$$

  In particular the derivative of y with respect to a is thus simply $\sigma'(a)$, which reads as

$$\begin{aligned}\sigma'(a) &= -(1+e^{-a})^{-2}\frac{d}{da}(1+e^{-a})\\ &= ae^{-a}(1+e^{-a})^{-2}\end{aligned}$$

Backpropagation

- In particular the derivative of y with respect to a is thus simply $\sigma'(a)$, which reads as

$$\begin{aligned}\sigma'(a) &= -(1+e^{-a})^{-2}\frac{d}{da}(1+e^{-a}) \\ &= ae^{-a}(1+e^{-a})^{-2} \\ &= \sigma(a)(1-\sigma(a))\end{aligned}$$

- At the output of the network, the (unique) activation can expand from the outputs to each of the neurons from the hidden layer as

$$a = \sum_j w_j^{(2)} z_j$$

- The z_j are the ouptputs to each of the neurons.

Backpropagation

- The derivative of a the weights $w_j^{(2)}$ is given by

$$\frac{\partial a}{\partial w_j^{(2)}} = z_j$$

- Now that we have $\frac{\partial L}{\partial y}$, $\frac{\partial y}{\partial a}$ and $\frac{\partial a}{\partial w_j^{(2)}}$, we can just use $\frac{\partial L}{\partial w_j^{(2)}} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial w_j^{(2)}}$

Backpropagation

- Combining the results above, we have

$$\frac{\partial L(\boldsymbol{w})}{\partial a_k} = \frac{\partial \ell}{\partial y_k}\frac{\partial y}{\partial a_k} = y_k(\boldsymbol{x}_i; \boldsymbol{w}) - t_{i,k} \equiv \delta_k$$

- On the other hand we now know $\frac{\partial a_k}{\partial w_{k,j}} = z_j$. We can thus write

$$\frac{\partial L(\boldsymbol{w})}{\partial w^{\ell}_{k,j}} = z^{\ell}_j \delta^{\ell}_k \tag{1}$$

- Then note that for any ℓ, we can backpropagate the information as

$$\frac{\partial L(\boldsymbol{w})}{\partial a^{\ell-1}_j} = \sum_k \frac{\partial L(\boldsymbol{w})}{\partial a^{\ell}_k}\frac{\partial a^{\ell}_k}{\partial a^{\ell-1}_j} \tag{2}$$

Backpropagation

- Then note that for any ℓ, we can backpropagate the information as

$$\frac{\partial L(\boldsymbol{w})}{\partial a_j^{\ell-1}} = \sum_k \frac{\partial L(\boldsymbol{w})}{\partial a_k^{\ell}} \frac{\partial a_k^{\ell}}{\partial a_j^{\ell-1}} \tag{3}$$

- Using the expressions that we derived earlier, we can thus write

$$\delta_j^{\ell-1} \equiv \frac{\partial L(\boldsymbol{w})}{\partial a_j^{\ell-1}} = \sum_k \delta_k^{\ell} \left(\sigma'(a_j^{\ell-1}) w_{k,j}^{\ell} \right) \tag{4}$$

where we use

$$\frac{\partial a_k^{\ell}}{\partial a_j^{\ell-1}} = \frac{\partial \sum_j w_{kj}^{\ell} \sigma(a_j^{\ell-1})}{\partial a_j^{\ell-1}} = w_{kj}^{\ell} \sigma'(a_j^{\ell-1}).$$

Backpropagation

- In short, the backpropagation algorithm can thus read as
    1. Apply an input sample $\boldsymbol{x}_i$ through the network and compute the activations a_j^ℓ for all hidden and output layers
    2. Evaluate all output δ_k^ℓ. Then backpropagate those δ_k^ℓ to find the $\delta_j^{\ell-1}$ for each hidden unit

$$\delta_j^{\ell-1} \equiv \frac{\partial L(\boldsymbol{w})}{\partial a_j^{\ell-1}} = \sum_k \delta_k^\ell \left(\sigma'(a_j^{\ell-1}) w_{k,j}^\ell \right)$$

    3. Get the final derivatives with respect to the weight by using

$$\frac{\partial L(\boldsymbol{w})}{\partial w_{ji}^\ell} = \frac{\partial L(\boldsymbol{w})}{\partial a_j^\ell} \frac{\partial a_j^\ell}{\partial w_{ij}^\ell} = \delta_j^\ell z_i$$

Backpropagation

- Often we will also consider an additional ℓ_2 penalty term. So that the total loss then reads as

$$L(\boldsymbol{w}) = -\frac{1}{N}\sum_{i=1}^{N} t_i \log(y(\boldsymbol{x}_i;\boldsymbol{w})) + (1-t_i)\log(1-y(\boldsymbol{x}_i;\boldsymbol{w})) + \frac{\lambda}{2N}\sum_{j=1}^{m} w_j^2$$

- Note that the contribution to the gradient arising from the regularization term can just be added to the backpropagation result.

- We typically choose to use a parameter norm penalty that penalizes only the weights of the affine transformation at each layer and leaves the biases unregularized. The biases typically require less data to fit accurately than the weights.