Feed-Forward Neural Network

Lecture 5

Readings: Murph 13-13.~~4.3~~ (1)

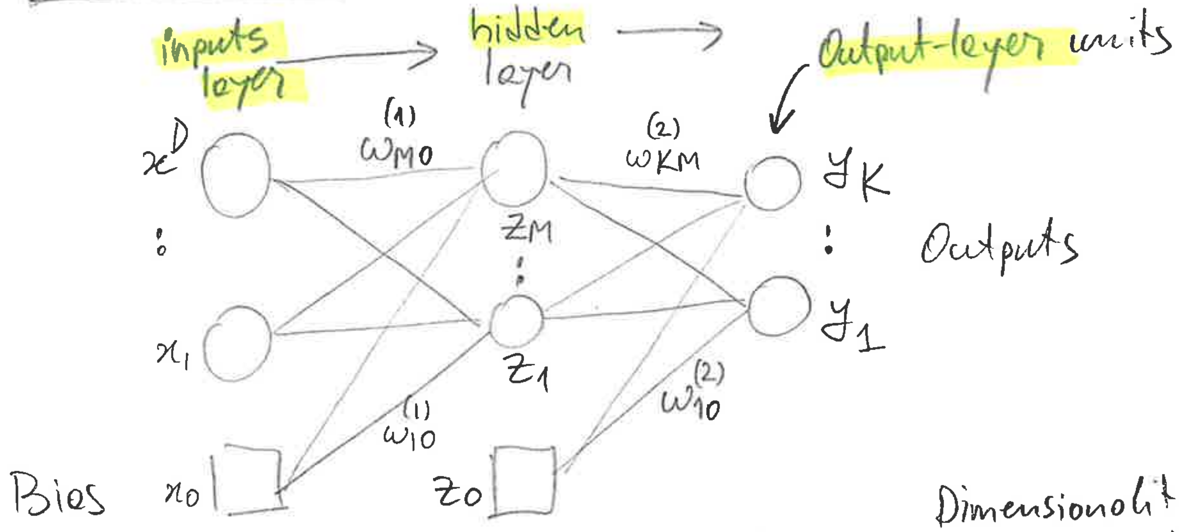

Fig. 1

Number units on hidden Layer

Dimensionality of input x

$$y_k(X,w) = \sigma\left(\sum_{m=0}^{M} w_{km}^{(2)} \cdot h\left(\sum_{d=0}^{D} w_{md}^{(1)} x_d\right)\right) \quad (1)$$

Activation

$h(a_m) = z_m^{(1)}$: output of hidden units on Layer 1

E.g.: Sigmoid function when Logistic Regression for Binary Classification

Denotes the non-linear activation function for a hidden unit.

△ A **Neural Network** is a non-linear function that transforms the input $\boxed{x}$ into an output $\boxed{y}$ that is controlled by the set of parameters $\boxed{w}$.

Training

– Need objective/loss/cost function.

– For Linear Regression, can use $\boxed{\text{least squares loss}}$:

$$\boxed{\mathcal{L}(w) = \frac{1}{N} \sum_{n=1}^{N} \left[y(x_n, w) - y_n \right]^2} \tag{2}$$

– For Binary classification (uses sigmoid output activation function) the $\boxed{\text{negated log-likelihood}}$ (or cross-entropy) is a typical loss function:

$$\boxed{\mathcal{L}(w) = -\sum_{n=1}^{N} \left[y_n \cdot \log(\hat{y}_n) + (1-y_n) \log(1-\hat{y}_n) \right]} \tag{3}$$

– For Multi-class classification problem (produced by softmax activ. function), we can use $\boxed{\text{negated-log-likelihood}}$ (cross-entropy) loss function:

$$\boxed{\mathcal{L}(w) = -\sum_{n=1}^{N} \sum_{k=1}^{K} y_{kn} \cdot \log\left[\frac{e^{a_k(x,w)}}{\sum_{j=1}^{K} e^{a_j(x,w)}} \right]} \tag{4}$$

Backpropagation = A procedure by which we pass errors backwards through a feed-forward NN in order to compute gradients for the weight parameters of the network.

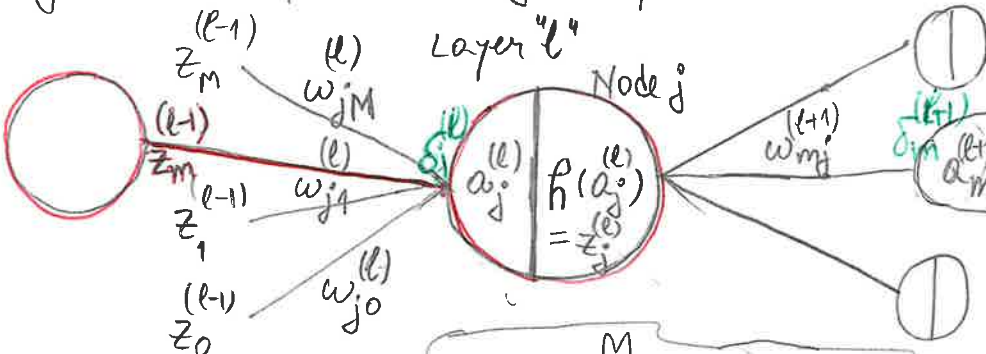

Fig. 2

Activation
$$a_j^{(l)} = \sum_{m=0}^{M} w_{jm}^{(l)} \cdot z_m^{(l-1)} \qquad (5)$$

— Output value: obtained by **activation function** $h(\cdot)$:
$$z_j^{(l)} = h(a_j^{(l)}) \qquad (6)$$

— Computing derivatives of the objective function with respect to weights; particularly w.r.t. individual weight $\boxed{w_{jm}^{(l)}}$ ($\boxed{m}$th weight for activation $\boxed{j}$ in layer $\boxed{l}$):

Use the chain rule:
$$\frac{\partial L}{\partial w_{jm}^{(l)}} = \underbrace{\frac{\partial L}{\partial a_j^{(l)}}}_{=\delta_j^{(l)}} \cdot \underbrace{\frac{\partial a_j^{(l)}}{\partial w_{jm}^{(l)}}}_{= z_m^{(l-1)}} \qquad (7)$$

$= \delta_j^{(l)}$ introduced notation for "errors"; $= z_m^{(l-1)}$ from eq. (5)

$$\boxed{\frac{\partial L}{\partial w_{jm}^{(l)}} = \delta_j^{(l)} \cdot z_m^{(l-1)}} \qquad (8)$$

← Derivative of Loss wrt an arbitrary weight in network can be calculated as $\boxed{\text{product}}$ of the error at the "output end of that weight" and value $z_m^{(l-1)}$ at "input end of the weight"

$$\frac{\partial E}{\partial w_{jm}} = \delta_j \cdot z_m$$

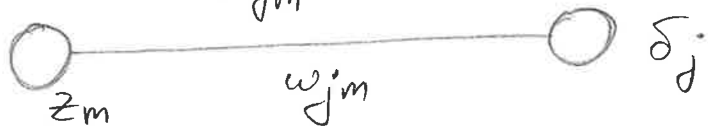

Fig. 3

- To compute derivatives, it suffices to:
  (1) compute values of δ_j for each node
  (2) also, save values z_m during forward pass through Network
  (will be multiplied w/ values of δ_j to get partials)

How to compute the errors?

- For an output-layer unit (indexed by $\boxed{k}$),
  assume regression problem (i.e., output activation function is linear)
  assume LSE loss function:

$$\delta_k^{(\ell)} = \frac{\partial L}{\partial a_k^{(\ell)}} = \frac{\partial L}{\partial \hat{y}_k} = \frac{d\left(\frac{1}{2}(\hat{y}_k - y_k)^2\right)}{d\hat{y}_k} = \hat{y}_k - y_k \tag{9}$$

$k = 1 \div K$

- For a hidden-layer unit (indexed by $\boxed{j}$), use chain-rule again:

$$\delta_j^{(\ell)} = \frac{\partial L}{\partial a_j^{(\ell)}} = \sum_{m=0}^{\boxed{M}} \boxed{\frac{\partial L}{\partial a_m^{(\ell+1)}}} \cdot \boxed{\frac{\partial a_m^{(\ell+1)}}{\partial a_j^{(\ell)}}} \tag{10}$$

← see Figure 2

$j = 0 \div M$

- all $\boxed{M}$ nodes that node $\boxed{j}$ sends connections to!

- Activation value of unit $\boxed{j}$ contributes only via its contribution to the activation value of those unit it is connected to! (i.e., $a_m^{(\ell+1)}$)

1^{st} term usually non-linear (dependence between Loss and activation value of a unit on next layer)

term #2 captures relationship between this activation $a_j^{(\ell)}$ and subsequent activation $a_m^{(\ell+1)}$ on next layer

Furthermore, we note:

term #1: $\dfrac{\partial L}{\partial a_m^{(l+1)}} \triangleq \delta_m^{(l+1)}$ by definition (11)

term #2: $\dfrac{\partial a_m^{(l+1)}}{\partial a_j^{(l)}} \overset{\text{by chain rule}}{=} \underbrace{\dfrac{\partial a_m^{(l+1)}}{\partial h(a_j^{(l)})}}_{= \frac{\partial a_m^{(l+1)}}{\partial z_j^{(l)}} = \boxed{w_{mj}^{(l+1)}}} \cdot \underbrace{\dfrac{\partial h(a_j^{(l)})}{\partial a_j^{(l)}}}_{= \frac{d h(a_j^{(l)})}{d a_j^{(l)}} \triangleq \boxed{h'(a_j^{(l)})}} = w_{mj}^{(l+1)} \cdot h'(a_j^{(l)})$ (12)

Substitute these 2 terms in equation (**) from previous page to get:

$$\boxed{\delta_j^{(l)} = h'(a_j^{(l)}) \cdot \sum_{m=0}^{M} w_{mj}^{(l+1)} \cdot \delta_m^{(l+1)}} \quad (13)$$

: this is a key insight for Backpropagation !!!

— Value of "error" can be computed by "passing back" (i.e. backpropagating) the "errors" for nodes further up in the network!

- We know values of errors δ for the final/output layer
  — by forward calculations
- We can recursively apply (13) to backpropagate errors from output units toward input.
- We can then use eq. (8) to estimate partial derivatives of Loss function L w.r.t. any weight!
  — those derivatives can be used by opt. methods like GD to train all weights !!!

Appendix A Loss Function - Network used for Logistic Regression

Log-Likelihood from Maximum Likelihood Estimation (MLE)

Assume probabilistic model, training pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, $y_i \in \{0, 1\}$

Let neural network w/ parameters θ produce a logit $z_i = z_\theta(x_i)$ and probability:

$$\boxed{p_i \equiv P_\theta(y_i = 1 \mid x_i) = \sigma(z_i) = \frac{1}{1+e^{-z_i}}}$$

Under a Bernoulli model:

$$P(y_i \mid x_i; \theta) = \underbrace{p_i^{y_i} \cdot (1-p_i)^{1-y_i}}_{\text{compact version}} = \underbrace{\begin{cases} p_i, & \text{if } y_i = 1 \\ 1-p_i, & \text{if } y_i = 0. \end{cases}}_{\text{expanded PMF of Bernoulli distribution.}}$$

Then, likelihood and log-likelihood (assuming of course iid):

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} p_i^{y_i} (1-p_i)^{1-y_i}$$

$$\ell(\theta) = \log(\mathcal{L}(\theta)) = \sum_{i=1}^{n} \left[y_i \cdot \log(p_i) + (1-y_i) \log(1-p_i) \right]$$

The average per-sample loss (divide by $\boxed{n}$) is the binary cross-entropy:

$$J(\theta) \equiv \boxed{BCE(y, p) = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log p_i + (1-y_i) \cdot \log(1-p_i) \right]}$$

popular notation

So, BCE is exactly the NLL of a Bernoulli model with $p = \sigma(z)$!

Minimizing BCE is equivalent to maximum likelihood for binary classification!