We can adapt the weights between V_0 and V_1 so that for every $i \in [k]$, the i^{th} neuron is $g_i(x)$ i

The neuron in the output layer implements a disjunction i.e. OR of the functions $g_i(x)$.

$-k+k-1 \Rightarrow -1$

AND

Bias $-k+1$

$$f(x) = \text{sign}\left(\underbrace{\sum_{i=1}^{k} g_i(x)}\ + \underbrace{k-1}_{\text{bias}}\right)$$

$-(k-2)+k-1 \Rightarrow 1$

sign(1)

Even if we try to model functions of the form $\underline{\{0,1\}^n} \rightarrow \underline{\{0,1\}}$ the size of the network will be exponential in n.

A neural network can approximate 1-Lipschitz function $f: [-1, +1]^n \longrightarrow [-1, 1]$ within a precision ϵ, but the size of the network will be exponential in n.

$[-1, +1]$

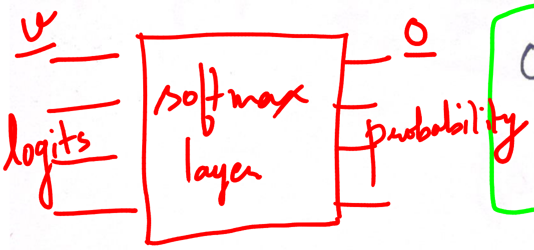
$f(u)$

$f(v)$

n_2 variables $[-1 \quad +1]$

$\|f(u) - f(v)\| \leq \|u - v\|^2$

Softmax converts k real valued predictions $v_1 \cdots v_k$ into output probabilities $o_1 \cdots o_k$ using the relation



$$\boxed{o_i = \frac{\exp(v_i)}{\sum_{j=1}^{k} \exp(v_j)}} \qquad \forall i \in \{1, \ldots k\}$$

Not an element-wise operation

The softmax is mostly paired with the cross-entropy loss.

If the target probability distribution over the k-classes is given by the vector $\boxed{y_1 \cdots y_k}$ GT then the cross-entropy loss is defined as

$$\underset{\text{loss.}}{L} = -\sum_{i=1}^{k} y_i \log(o_i)$$

$\underset{\text{output}}{o}$ $\underset{\text{GT}}{y}$

Ground truth $\underline{y}$ $\underline{O}$

$$\begin{array}{c} \text{Labels } 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \longrightarrow \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{bmatrix} \begin{array}{l} 0.05 \\ 0.7 \\ 0.1 \\ 0.1 \\ 0.05 \end{array} \quad \sum p_i = 1.$$

$$\text{Loss} = -\log p_{GT}$$

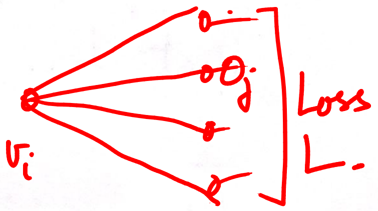
If $p_2 = 1$ $p_2 \to 0$

Loss $= 0$.

$\underbrace{\text{Loss} = \infty}$ if $p_2 = 0$

The partial derivative of the loss w.r.t. the logit value v_i

is $$\frac{\partial L}{\partial v_i} = \sum_{j=1}^{k} \left(\frac{\partial L}{\partial o_j}\right)\left(\frac{\partial o_j}{\partial v_i}\right) = \boxed{o_i - y_i}$$



Proof

$$o_i = \frac{\exp(v_i)}{\sum_j \exp(v_j)}$$ softmax

$$\frac{\partial o_j}{\partial v_i} = \underbrace{-\frac{\exp(v_i)\exp(v_j)}{\left(\sum \exp(v_j)\right)^2}}_{-o_i o_j} + \underbrace{\frac{\exp(v_i)}{\sum_j \exp(v_j)}}_{o_i}$$

the 2nd term is there if $i=j$

$$\frac{\partial o_j}{\partial v_i} = -o_i o_j + o_i \quad \text{if } i=j$$
$$= o_i(1-o_i) \quad \text{if } i=j$$

$$\frac{\partial o_j}{\partial v_i} = -o_i o_j \quad \text{if } i \neq j$$

Substituting these partial derivatives

$$-\frac{\partial L}{\partial v_i} = \sum_{j=1}^{k} \left(\frac{y_j}{o_j}\right) \underbrace{o_j(1-o_j)}_{\frac{\partial o_j}{\partial v_i}}$$

for $i=j$ $\frac{\partial o_j}{\partial v_i} = o_i(1-o_i)$

$i \neq j$ $\frac{\partial o_j}{\partial v_i} = -o_i o_j$ if $i \neq j$

$$= \sum_{\substack{j=1 \\ j\neq i}}^{k} \frac{y_j}{\cancel{o_j}}\left(-o_i \cancel{o_j}\right) + \underbrace{\frac{y_j}{o_j}\, o_j(1-o_j)}_{j=i}$$

$$= \sum_{\substack{j=1 \\ j\neq i}}^{k} -y_j o_i + \underbrace{y_j}_{j=i} - \underbrace{y_j o_j}_{j=i}$$

$$= \underbrace{y_j}_{j=i} - \underbrace{o_j}_{j=i} \underbrace{\sum_{j=1}^{k} y_j}_{=1} = \underbrace{y_j - o_j}_{j=i}$$

$$\therefore \boxed{\frac{\partial L}{\partial v_i} = o_j - y_j} \quad j=i$$

$= o_i - y_i$

$G_{V,E,\sigma}$

Recurrent Neural Network

speech, time-series
sentences, handwriting } text

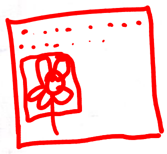
The feedforward dense connected neural network is unaware of the dependencies/orderings of the feature components of the input. Though the network can discover such orderings, it will require a very large training set.

For applications involving sequential input such as:

Language Modeling → predict the next word in a sentence
The sun rises . . .

Sentence Translation, sentence classification
sentiment

Image Captioning



— the input sequence length is not constant.

To exploit such local dependencies in sentences, images, speech, time series data, the network architecture should be aware of the ordering of the input.

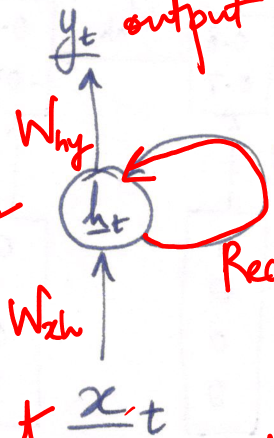
① aware of ordering
② variable size of input

A repeated recurrent neural network (RNN) uses a variable number of layers such that each layer takes input of a specific position (time stamp) in a sequence. The inputs are provided in a sequence to the "temporal layers". Each layer takes a multidimensional input and produces a multi-dimensional output.

Further, each layer uses the same set of parameters (repeated recurrent computation) single layer.

This is called as parameter sharing.

Every input is treated in the same manner.

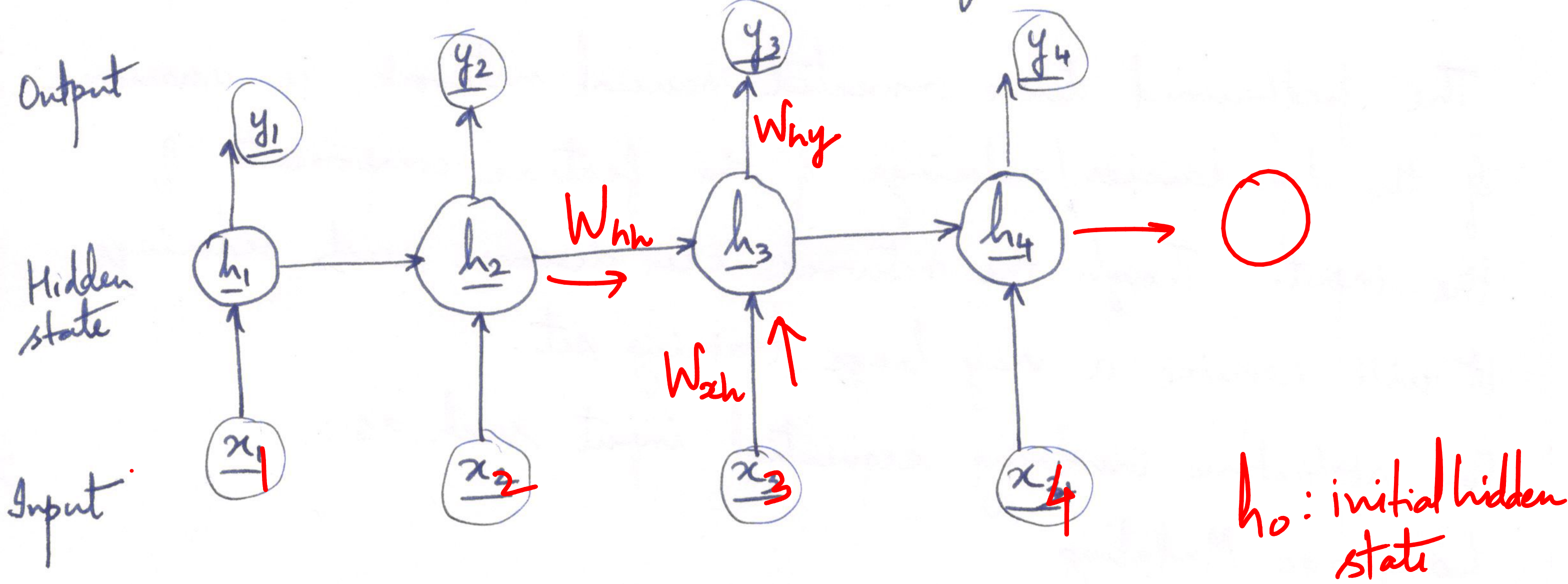


$h_t = f(h_{t-1}, x_t)$

h_t : hidden representation

$x_t \rightarrow h_t \rightarrow o_t$

The recurrent network can be unfolded in time



$$h_t = f(h_{t-1}, x_t)$$

$$y_t = g(h_t)$$

$$h_1 = f(h_0, x_1)$$

$$h_2 = f(\underbrace{f(h_0, x_1)}_{h_1}, x_2)$$

$$\vdots$$

Since $y_t = g(h_t)$

$$\boxed{y_t = F_t(x_1, x_2, \ldots x_t)}$$

$$\boxed{\begin{aligned} h_t &= \underbrace{\tanh}_{\text{nonlinearity}}\big(\underbrace{W_{xh} x_t + W_{hh} h_{t-1}}_{\text{Affine}}\big) \\ y_t &= W_{hy} h_t \end{aligned}}$$