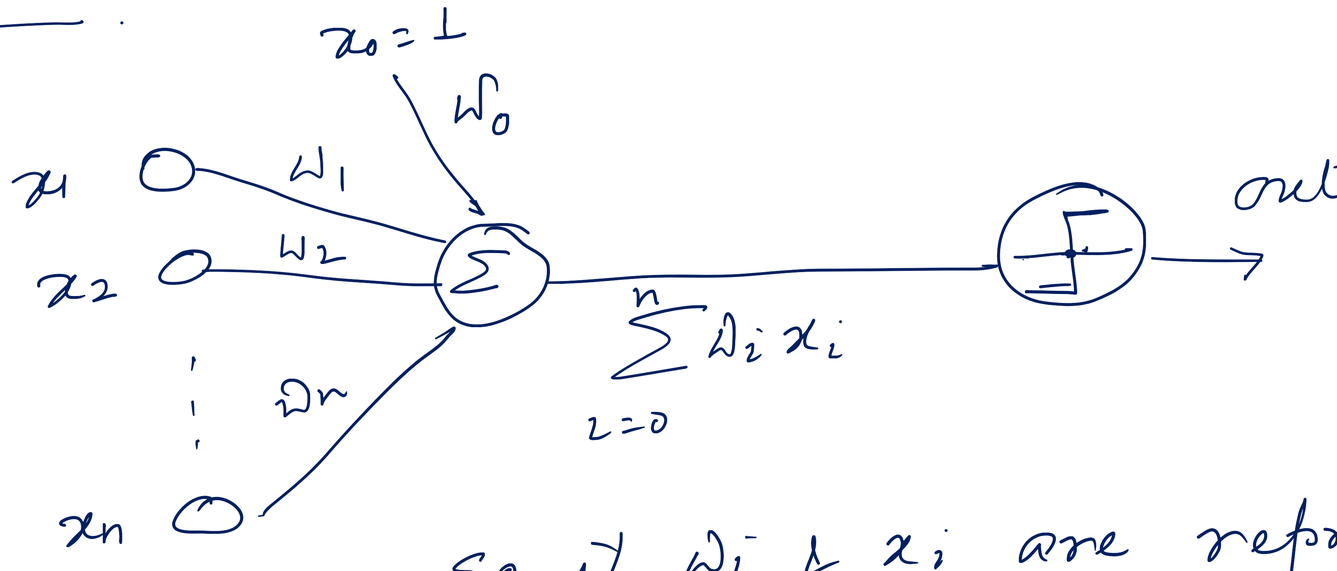
Node Embeddings

Neural Networks

Perceptron → Creating an automated logic function
AND, OR, NAND, NOR



So if w_i & x_i are represented as vectors then

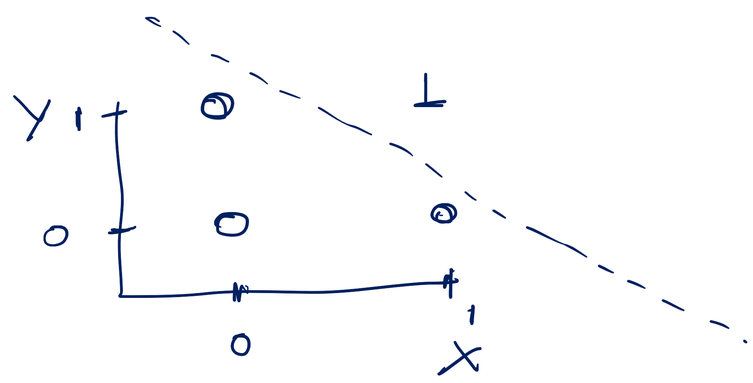
$$o(\vec{x}) = \text{Sgn}(\vec{w} \cdot \vec{x})$$

$$\text{Sgn}(y) = 1 \quad \text{if } y > 0$$
$$= 0 \quad \text{otherwise}$$

Learning the perceptron
- Involves learning $w_0, w_1, \ldots w_n$

Consider ~~AND~~

x	y	target (t) ~~output = o~~
0	0	0
0	1	0
1	0	0
1	1	1



How to the learn the weight such that for the given ∅ inputs, the outputs ~~result~~ is given as result.

Generally 2 ways — 1) ~~Delta rule~~ Perceptron training rule
2) ~~Stochas~~ Gradient Descent.

~~Delta rule~~ Perceptron training rule.

Start with some random initial weights $w_0, w_1, w_2 \ldots$

Weight w_i is revised as

$w_i = w_i + \Delta w_i$ where $\Delta w_i = \eta (t - o) x_i$

η = learning rate

t = target value

o = output value returned by perceptron

x_i = Input

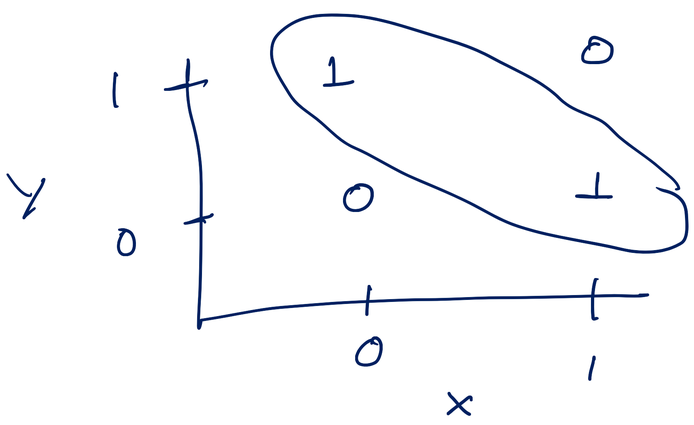
If $t = 0$, $o = 0$ so $\Delta w_i = 0$

If $t = 1$ $o = -1$ so $t - o = 2$ i.e w_i has to be increased.

So the linear perceptron with delta rule only works for those logics that have a linear separation between the classes.

So it doesnot work for function like XOR

x	y	t
0	0	0
0	1	1
1	0	1
1	1	0



For handling these cases use gradient descent.

- need to specify a measure for training error of a hypothesis or a loss function

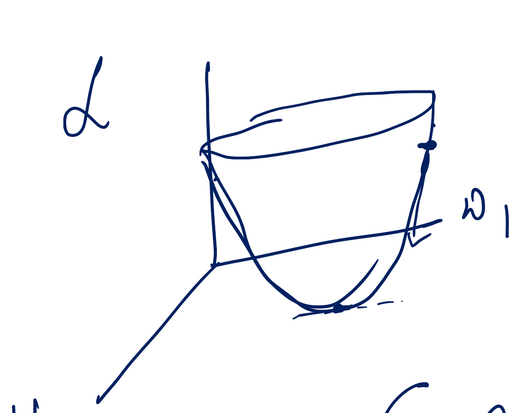
Gradient descent

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

D = set of training example.

o_d = output of perceptron for data instance d

t_d = Actual output.



Derive Gradient of $\mathcal{L}$, with respect to $w_0, w_1, w_2 \ldots$.

Start with some random $\vec{w}$

$\nabla \mathcal{L}(w)$

This would be a vector that gives the direction of the steepest ascent.

So $-\nabla \mathcal{L}(w)$ gives the direction of the steepest descent.

So the new update rule $w_i = w_i - \eta \nabla \mathcal{L}(w)$

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

$$o_d = \sum_j w_j x_j$$

$$\frac{\partial o_d}{\partial w_i} = x_i$$

$$\frac{\partial E}{\partial w_i} = \frac{1}{2}\sum_{d\in D} 2(t_d - o_d)(-x_{id})$$

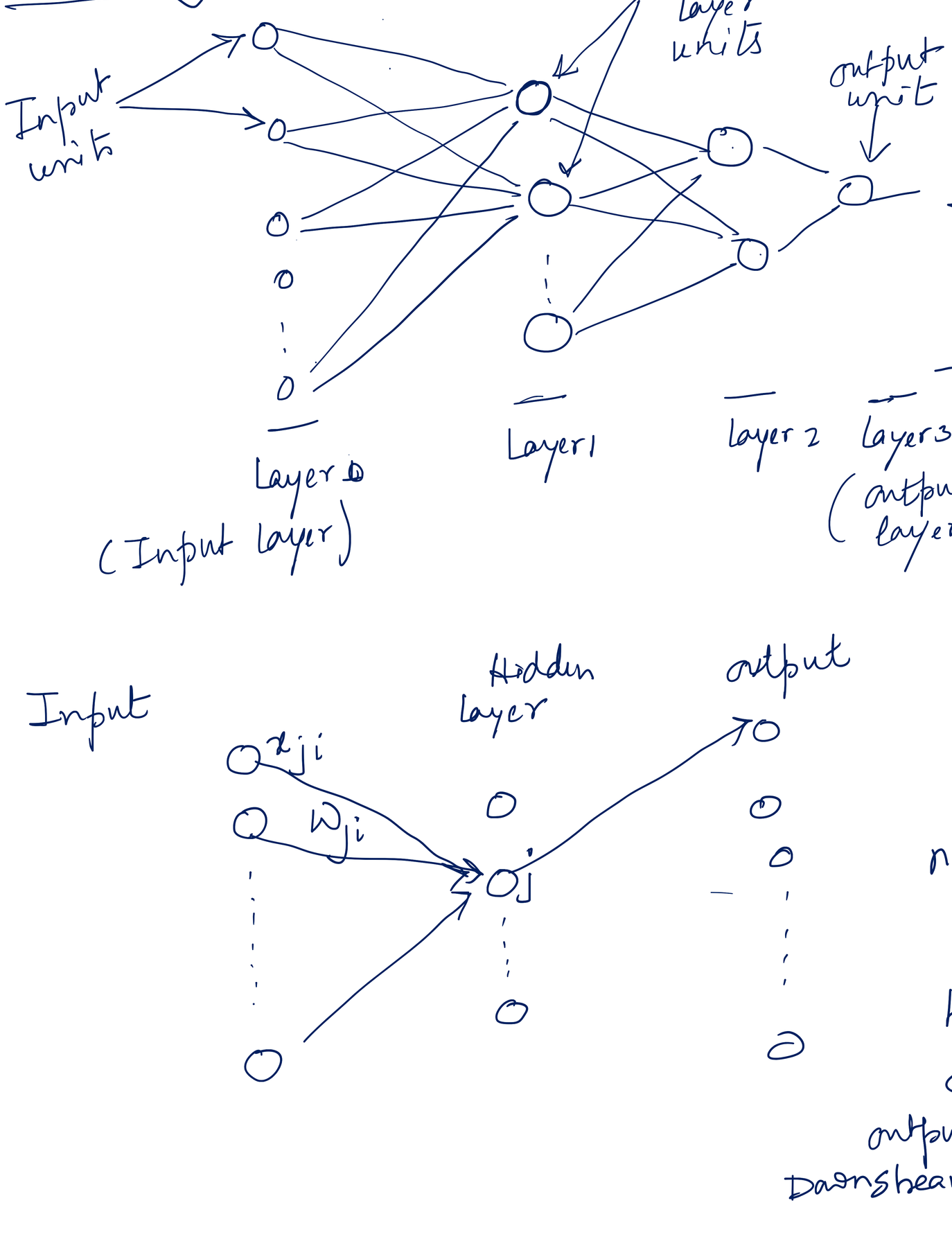
$$= -\sum_{d\in D}(t_d - o_d)(x_{id})$$

So update in $w_i = w_i - \sum_{d\in D}(t_d - o_d)(x_{id})$

But the sgn(x) function is non differentiable, so replace sgn with a differentiable function like Sigmoid or tan h. that looks similar to sgn.

Adding layers to the perceptron. → Enhance the capacity of the model.

- Multi layer perceptron



Advantage

- Arbitrary non linear class separation boundaries.
- Higher capacity.

Symbols

x_{ji} = The i^{th} input to the j^{th} unit

w_{ji} = The weight associated with the i^{th} input to unit j

$net_j = \sum_i w_{ji} x_{ji}$ (the weighted sum of inputs to unit j)

o_j = output of the j^{th} unit

t_j = target output for unit j

σ = sigmoid function

output = Set of units in the final layer of the network

Downstream(j) = Set of units whose immediate inputs include the output of unit j

~~At~~ Start with some arbitrary assignment of weight $w = [\quad]$

for each unit j in the downstream of the input layer do the following

a) $net_j = \sum_i w_{ji} x_i$

b) $o_j = \sigma(net_j) = \frac{1}{1 + e^{-net_j}}$

Repeat for further downstream nodes until an output is obtained from the output layer.

⟶ Feed forward function

↳ Calculates the output of the neural network

For updating the weights, use backpropagation.

i.e. the weights near to the output layer are updated first.

for updation $w_{ji} = w_{ji} - \eta \frac{\partial Ld}{\partial w_{ji}}$

w_{ji} can influence the rest of the network through net_j

$$\frac{\partial Ld}{\partial w_{ji}} = \frac{\partial Ld}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = x_i \frac{\partial Ld}{\partial net_j}$$

Consider 2 cases

a) where unit j is an output unit for the network

b) where unit j is an internal unit.

Case a). Training rule for output unit weights

→ w_{ji} can influence the rest of the network through net_j and net_j

can influence the " " " " o_j

$$\frac{\partial Ld}{\partial net_j} = \frac{\partial Ld}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

$$o_j = \sigma(net_j) = \frac{1}{1 + e^{-net_j}}$$

$$Ld = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

$$\frac{\partial Ld}{\partial o_j} = \frac{1}{2} \times 2 (t_j - o_j)(-1) = -(t_j - o_j)$$

$$\frac{\partial o_j}{\partial net_j} = \frac{e^{-net_j}}{(1 + e^{-net_j})^2} = \frac{e^{-net_j}}{(1 + e^{-net_j})} \cdot \left(1 \frac{1}{+ e^{-net_j}}\right) = o_j (1 - o_j)$$

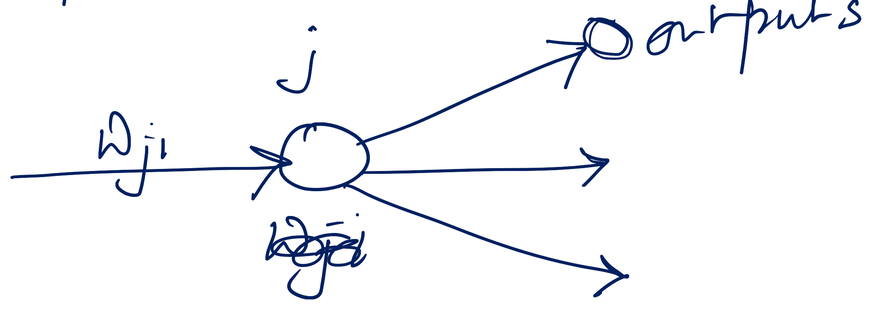
$$\frac{\partial Ld}{\partial net_j} = -(t_j - o_j) o_j (1 - o_j)$$

So $\Delta w_{ji} = -\eta \frac{\partial Ld}{\partial w_{ji}} = \eta (t_j - o_j) o_j (1 - o_j) x_i$

Case (b) Training rule for hidden unit weights

When unit j is a hidden layer unit

Then w_{ji} influences the output through neti of the hidden layer unit, that further influences ~~input~~ the output from the hidden layer



$$\frac{\partial L_d}{\partial net_j} = \sum_{K \in downstream(j)} \frac{\partial L_d}{\partial net_k} \cdot \frac{\partial net_k}{\partial net_j}$$

$$L_d = \frac{1}{2} \sum_{K \in outputs} (t_k - o_k)^2$$

$$\frac{\partial L_d}{\partial net_k} = \frac{\partial L_d}{\partial o_k} \cdot \frac{\partial o_k}{\partial net_k}$$

$$\frac{\partial o_k}{\partial net_k} = o_k(1-o_k)$$

$$\frac{\partial L_d}{\partial o_k} = -(t_k - o_k)$$

$$\frac{\partial L_d}{\partial net_k} = -(t_k - o_k) o_k (1-o_k)$$

$$\frac{\partial net_k}{\partial net_j} = \frac{\partial net_k}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial o_j}{\partial net_j} = o_j(1-o_j)$$

$$\frac{\partial net_k}{\partial o_j} = w_{kj}$$

$$\text{So } \frac{\partial net_k}{\partial net_j} = w_{kj}\, o_j(1-o_j)$$

$$\text{So } \frac{\partial L_d}{\partial net_j} = \sum_{K \in downstream(j)} -(t_k - o_k) o_k (1-o_k)\, w_{kj}\, o_j(1-o_j)$$

$$= -o_j(1-o_j) \sum_{K \in downstream(j)} \delta_k w_{kj} \quad = \delta_j$$

$$\text{where } \delta_k = (t_k - o_k) o_k (1-o_k)$$

$$\text{So } \Delta w_{ji} = -\eta \frac{\partial L_d}{\partial w_{ji}} = \eta\, x_i \delta_j$$

The neural network is used to generate the representation of words appearing in a vocabulary. (Word2Vec)

Continuous Bag of Word (CBoW) model.

Skip gram model.

In CBoW model.

I am a good boy.

Suppose good is missing

I am a ____ boy

missing word

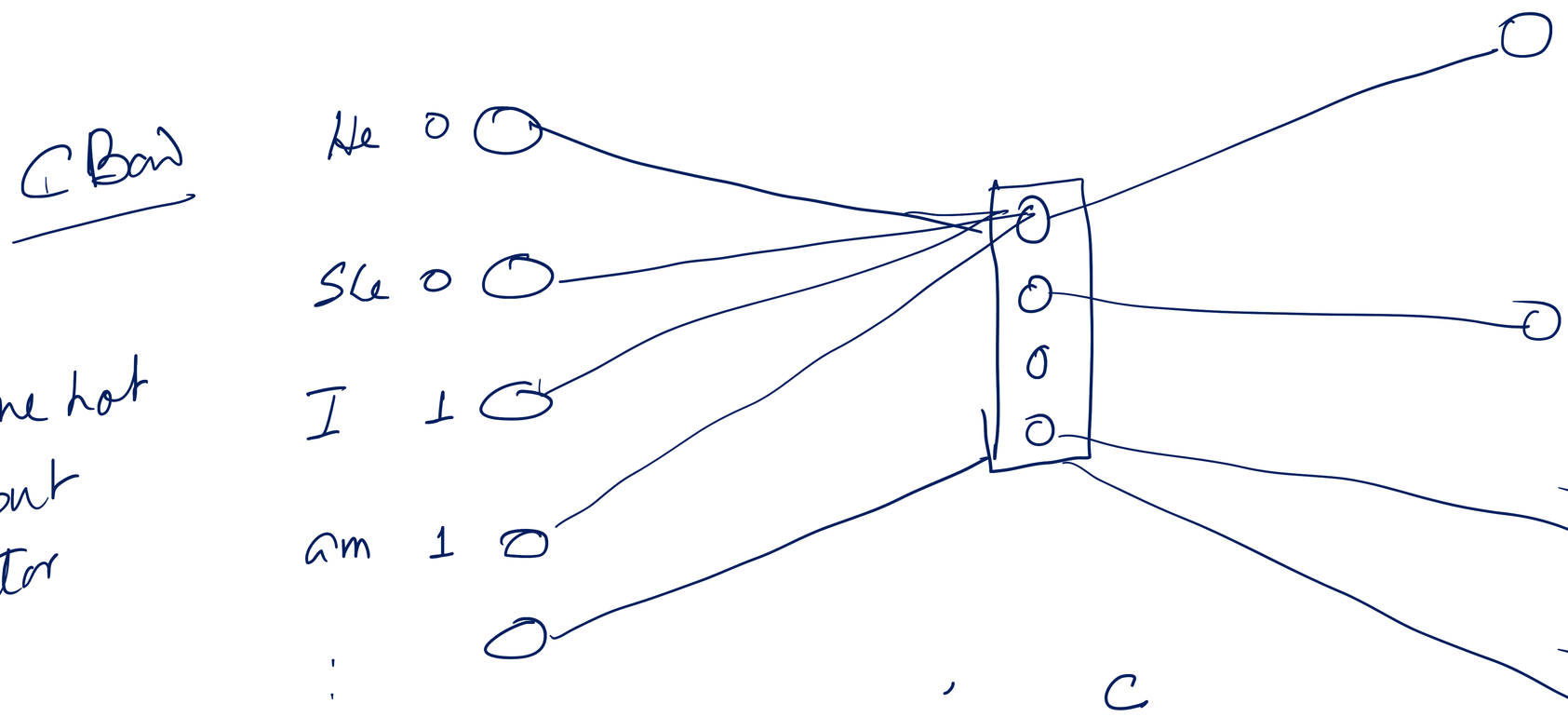
c_1 c_2 c_3 c_4

Given a set of context words, predict the missing word-w.

c

In skipgram model.

Given a word w, predict the context words.



Assuming one context word is given

Predict the word w adjacent to it.

$$h = W^T x = v_c$$

Let u be the ~~index~~ input vector to the output units

$$u = W'^T h$$

So the input to the i^{th} output unit

$$u_i = v'^T_w \cdot v_c$$

The output of the i^{th} output unit is generated using a softmax function

$$y_i = \text{softmax}(u_i) = \frac{e^{u_i}}{\sum_j e^{u_j}}$$

King queen
Emperor Empress
People
Cat
Dog

$$= \frac{e^{\underline{v}_w^T \underline{v}_c}}{\sum_{w'} e^{\underline{v}_{w'}^T \underline{v}_c}} = P(w \mid c)$$

$w_1\ w_3\ w_4 \ldots$

$P(w_1|c)\, P(w_2|c)\, P(w_3|c)$

To derive the parameters $\theta = \{\underline{v}_c, \underline{v}_w\}$, we need a loss function

- Suitable loss function is maximum likelihood.

The likelihood $L(\theta) = \prod_w P(w \mid c; \theta)$

$$= \prod_w \frac{e^{\underline{v}_w^T \underline{v}_c}}{\sum_{w'} e^{\underline{v}_{w'}^T \underline{v}_c}}$$

$$-\log L(\theta) = -l(\theta) = -\sum_{w^*} \log \left[\frac{e^{\underline{v}_w^T \underline{v}_c}}{\sum_{w'} e^{\underline{v}_{w'}^T \underline{v}_c}} \right]$$

$$= -\left[\sum_w \underline{v}_w^T \underline{v}_c - \sum_w \log \left(\sum_{w'} e^{\underline{v}_{w'}^T \underline{v}_c} \right) \right]$$

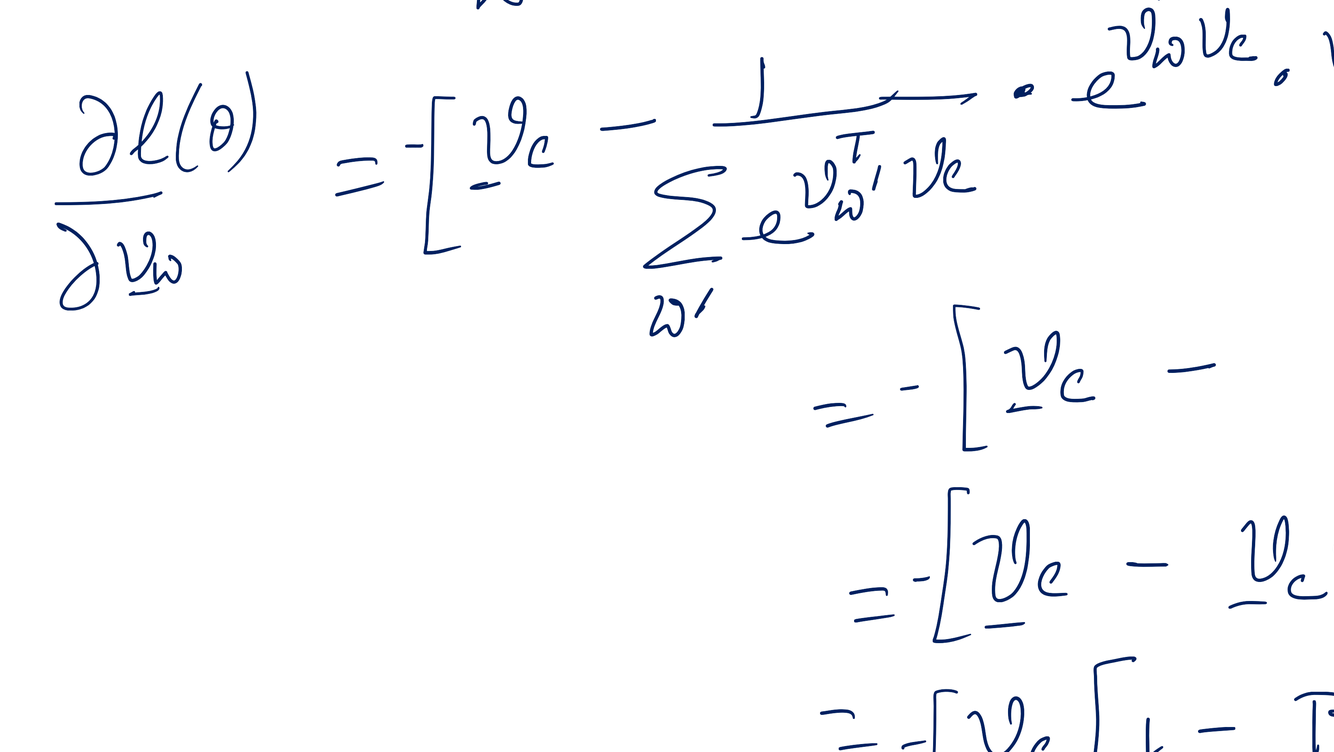
$$\frac{\partial l(\theta)}{\partial \underline{v}_w} = -\left[\underline{v}_c - \frac{1}{\sum_{w'} e^{\underline{v}_{w'}^T \underline{v}_c}} \cdot e^{\underline{v}_w \underline{v}_c} \cdot \underline{v}_c \right]$$

$$= -\left[\underline{v}_c - \underline{v}_c \cdot \left[\frac{e^{\underline{v}_w^T \underline{v}_c}}{\sum_{w'} e^{\underline{v}_{w'}^T \underline{v}_c}} \right] \right]$$

$$= -\left[\underline{v}_c - \underline{v}_c \cdot P(w \mid c) \right]$$

$$= -\left[\underline{v}_c \left[1 - P(w \mid c) \right] \right]$$

So update $\underline{v}_w$ as $\underline{v}_w^{new} = \underline{v}_w^{old} + \eta\, \underline{v}_c \left[1 - P(w/c) \right]$



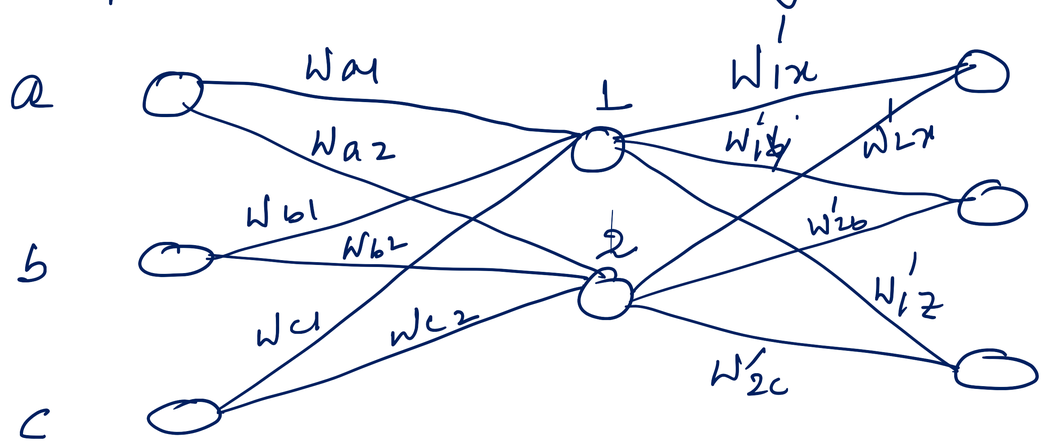
The bottleneck of this iterative method is that during every update we have to take the sum $\sum_{w'} e^{\underline{v}_{w'}^T \underline{v}_c}$ involving all the words in the vocabulary.

The number of words in the vocabulary may run up to million. Hence this method does not work practically. Soln Use a different approach

~~Negative Sampling~~ A working Example

→ Assume there are only 3 words in the vocabulary.
& Try to work on an example of CBoW model.

We are given a context word and we need to predict the missing word.



Suppose given a context vector for b

ie

input vector $\underline{x} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

Let W be the weights to the hidden layer units

$$W = \begin{bmatrix} W_{a1} & W_{a2} \\ W_{b1} & W_{b2} \\ W_{c1} & W_{c2} \end{bmatrix}$$

Then $\underline{h} = W^T x$

$$= \begin{bmatrix} W_{a1} & W_{b1} & W_{c1} \\ W_{a2} & W_{b2} & W_{c2} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} W_{b1} \\ W_{b2} \end{bmatrix} = \underline{v}_c = \underline{h}_b$$

$$W' = \begin{bmatrix} W'_{1x} & W'_{1y} & W'_{1z} \\ W'_{2x} & W'_{2y} & W'_{2z} \end{bmatrix}$$ Vector representation for the word z = $\underline{v}'_w$

input to the

So the output vector is $\underline{u} = W'^T \underline{h}$

$$= \begin{bmatrix} W'_{1x} & W'_{2x} \\ W'_{1y} & W'_{2y} \\ W'_{1z} & W'_{2z} \end{bmatrix} \begin{bmatrix} W_{b1} \\ W_{b2} \end{bmatrix} = \begin{bmatrix} W'_{1x} \cdot W_{b1} + W'_{2x} \cdot W_{b2} \\ W'_{1y} \cdot W_{b1} + W'_{2y} \cdot W_{b2} \\ W'_{1z} \cdot W_{b1} + W'_{2z} \cdot W_{b2} \end{bmatrix}$$

$$= \begin{bmatrix} v'^{(a)}_w \cdot v^{(b)}_c \\ v'^{(b)}_w \cdot v^{(b)}_c \\ v'^{(c)}_w \cdot v^{(b)}_c \end{bmatrix}$$

Convert to softmax score

$\underline{y} = \cancel{f(u)}$ softmax $(\underline{u})$

$$y_i = \text{softmax}(u_i) = \frac{e^{v'^T_c v_w}}{\sum_{w'} e^{v'^T_c v_{w'}}}$$

$y_a = P(w = a | c = b)$; $y_b = P(w = b | c = b)$; $y_c = P(w = c | c = b)$

Loss function to derive & Parameters $\theta = \{v_c, v_w\}$

Maximum likelihood

Consider the dataset

$D = (c_1, w_1)\ (c_1, w_2)\ (c_1, w_3) \ldots\ (c_2, w_3),\ (c_2, w_5) \ldots\ldots$

c_1 → Context $\{a, b, c\}$; w_1 → word $\{a, b, c\}$

— Attempt to learn the word distribution for each context word.

Suppose consider the context word c_1 → find the prob of appearance of each word adjacent to it.

Suppose in the actual data the following context & word pair appears

$(a, b)\ (a, c)\ ,\ (b, a)\quad (c, a)\ (a\ b)\ (a\ c)\ (b, a)\ (a, c)$

~~Given Context word a ; b & c appears as word.~~

Given Context word b ; the prob of a & c appearing as words.

$$= P(a|b) \cdot P(a|b)$$

$$\mathcal{L} = \frac{e^{v_c^{(b)T} v_w^{(a)}}}{\sum_{w'} e^{v_c^{(b)T} \cdot v_{w'}}} * \frac{e^{v_c^{(b)T} v_w^{(c)}}}{\sum_{w'} e^{v_c^{(b)T} v_{w'}}}$$

Max Log L and try to maximize Log L with respect to v_c & v_w

So to avoid calculating the denominator repeatedly, we use negative sampling.

Negative Sampling ← Distributed representation of words & phrases and their compositionality (Mikolov et. al)

for skipGram

Convert this problem to a Supervised problem.

Consider 2 datasets: D: set of all pairs (c, w) that are in text

D': set of all pairs (c, w) that are NOT in text

Let ~~Z = 1 if~~ Label $Z = 1$ for all pairs in D

& $Z = 0$ for all pairs in D'

Turned the problem to a classification problem.

$P(Z = 1 \mid (C, W))$ is the prob that (C, W) appears in D

$P(Z = 0 \mid (C, W))$ " " " " (C, W) doesn't appear in D.

$$\underbrace{P(Z \mid X)}_{\text{Posterior}} = \frac{\overbrace{P(X \mid Z)}^{\text{class conditional}} \overbrace{P(Z)}^{\text{Prior}}}{\underbrace{P(X)}_{\text{marginal}}}$$

So to directly find the posterior without calculating the class conditionals or priors, we can assume a function that returns the prob of Z given X.

They assume $P(Z \mid X)$ follows a logistic Regression function

i.e $$P(Z = 1 \mid (C, W)) = \frac{1}{1 + e^{-\underline{v}_C^T \underline{v}_W}}$$

$$P(Z = 0 \mid (C, W)) = 1 - \frac{1}{1 + e^{-\underline{v}_C^T \underline{v}_W}} = \frac{e^{\underline{v}_C^T \underline{v}_W}}{1 + e^{\underline{v}_C^T \underline{v}_W}} = \frac{1}{1 + e^{-\underline{v}_C^T \underline{v}_W}}$$

The $$P(Z \mid (C, W)) = \left(\frac{1}{1 + e^{\underline{v}_C^T \underline{v}_W}}\right)^Z \left(\frac{1}{1 + e^{-\underline{v}_C^T \underline{v}_W}}\right)^{1 - Z}$$

So $P(Z \mid (C, W))$ is also parameterized by $\theta = \{\underline{v}_C \; \underline{v}_W\}$

To derive these parameters we use maximum Likelihood.

Let $D = \{(C_1, W_1) \; (C_3, W_2) \; (C_4, W_4) \; \ldots\}$ ← Positive pairs

$D' = \{(C_2, W_1) \; (C_5, W_1) \; \ldots \; \ldots\}$ ← Negative pairs

$$L(\theta) = \prod_{(C, W) \in D \cup D'} P(Z \mid (C, W))$$

$$= \text{Log}\,L(\theta) = l = \sum_{(c,w)\in DUD'} \text{Log}\,P(z|(c,w)) = \sum_{(c,w)\in DUD'} \text{Log}\left[\left(\frac{1}{1+e^{v_c^T v_w}}\right)^{z}\left(\frac{1}{1+e^{-v_c^T v_w}}\right)^{1-z}\right]$$

(Log likelihood)

Minimize $-\text{Log}(L(\theta)) = -l$

bay

$$= -\sum_{(c,w)\in DUD'} z\,\text{log}\left(\frac{1}{1+e^{v_c^T v_w}}\right) + (1-z)\,\text{Log}\left(\frac{1}{1+e^{-v_c^T v_w}}\right)$$

$$= -\left[\sum_{(c,w)\in D} \text{log}\left(\frac{1}{1+e^{v_c^T v_w}}\right) + \sum_{(c,w)\in D'} \text{log}\left(\frac{1}{1+e^{-v_c^T v_w}}\right)\right]$$

The skipgram paper considers one positive pair and a small no. of negative pairs (around 5) for each gradient update

Here are (-)ve pairs identified.

— For each word they derive $P_n(w) = \frac{\#w}{\sum_{w'} \#w'}$

They sample 2 words randomly based on this probability.

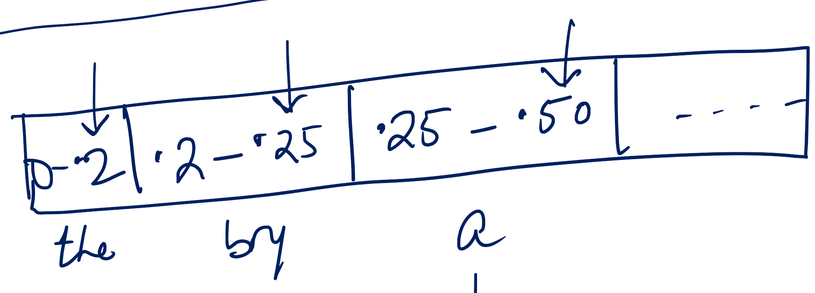
— And they assume that these 2 words will not appear as (word, context) pair in the corpus

Then they derive the gradient $\frac{\partial l}{\partial v_c}$, $\frac{\partial l}{\partial v_w}$

$P_1(\text{the}) = \frac{20}{100} = .2$

$P_1(\text{bay}) = \frac{5}{100} = .05$

$P_1(a) = \frac{25}{100} = .25$



Choose a random # between 0 & 1

Suppose tst → .3 (a)

2nd — .1 (the)

(a the) → ∈ D'

$$v_c^{new} = v_c^{old} - \eta \frac{\partial l}{\partial v_c}$$

$$v_w^{new} = v_w^{old} - \eta \frac{\partial l}{\partial v_w}$$

→ Set of running text documents.

→ From the set of running text documents & create (word & context) pairs. Let that be marked as D^{new}.

→ ~~Minimize the (-)ve log likelihood~~

→ Create the negative sample set D'

→ Create a batch of one positive and 5 negative samples.

→ Minimize the (-)ve log likelihood of generating the batch using the model.

Consider the text I am a very good boy.

$D :=$ (I, am) (am a) (am, I) (a, very) (a, am)
word content word cont word content

(very, good), (very, a), (good boy), (good very)

(boy, good)

For each ~~start~~ word w ~~w_0~~ initialize with a random embedding v_w

For each content c " " random " v_c

$$v_w^{new} = v_w^{old} - \eta \frac{\partial L}{\partial v_w}$$

$$v_c^{new} = v_c^{old} - \eta \frac{\partial L}{\partial v_c}$$

What should be ~~the same~~ D for a graph

Assignment

for a given u, the random walk should be biased towards nodes with degree similar to u.