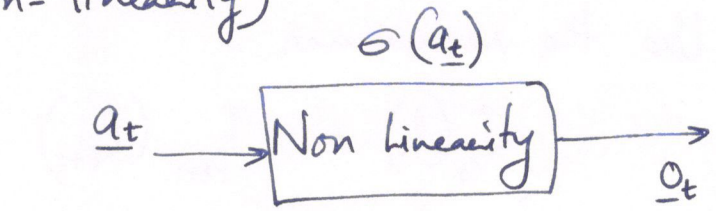
Sigmoid Function (Non-linearity)

$$o_{t_i} = \frac{1}{1+\exp(-a_{t_i})}$$



$$\sigma'(a_{t_i}) = \frac{1}{(1+\exp(-a_{t_i}))(1+\exp(a_{t_i}))}$$

i^{th} diagonal entry of $J_{a_t}(o_t)$

$J_{a_t}(o_t)$ is a diagonal matrix denoted as $\text{diag}(\sigma'(a_t))$

element wise nonlinearity

$$l_t(o_t) = l_{t+1}(o_t) = l_{t+1}(\sigma(W_t o_t))$$

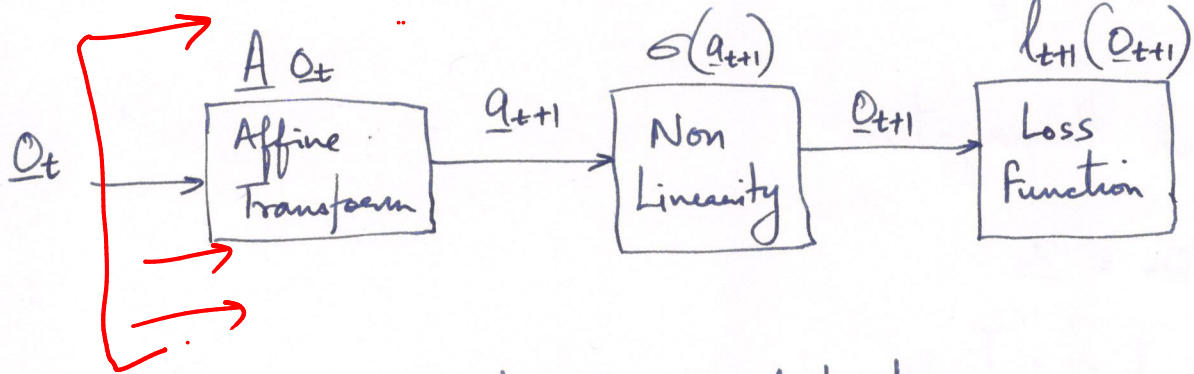
$$J_{o_t}(l_t) = J_{o_{t+1}}(l_{t+1})\; J_{a_{t+1}}(o_{t+1})\; J_{o_t}(a_{t+1})$$

Loss from further network

Non linearity σ

Affine

recursive



Computing $J_{o_T}(l_T)$ for the last layer

final layer

Final loss $l_T = \frac{1}{2}\|o_T - y\|^2$

y: ground truth target

$J_{o_T}(l_T) = o_T - y$ vector.

Use the recurrence

$$\gamma_t \; J_{a_t}(l_t) = \underbrace{J_{o_{t+1}}(l_{t+1})}_{\gamma_{t+1}} \underbrace{J_{a_{t+1}}(o_{t+1})}_{\sigma'} \underbrace{J_{o_t}(a_{t+1})}_{\text{affine } W}$$

to back propagate values for $J_{o_T}(l_T)$ starting from the last layer T and working towards the previous layers.

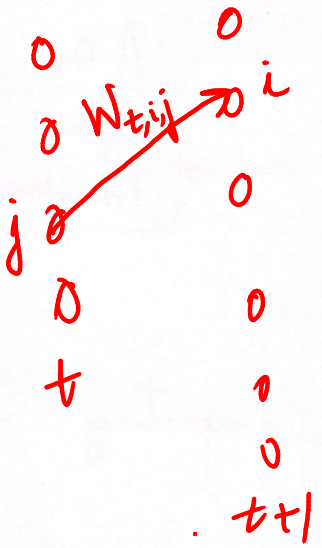
Backpropagation (Training examples (x, y), layered graph (V, E))

Initialize:

denote layers of the graph $V_0 \dots V_T$ where # nodes

$V_t = \{v_{t,1}, \dots v_{t,n_t}\}$ $\quad t \to n_t$

Define $W_{t,i,j}$ as the weight of the edge $(v_{t,j}, v_{t+1,i})$



Forward Propagation:

set $o_0 = x$ $\quad V_0 \quad o_0 \equiv x$ (initial)

for $t = 1, \dots T$ (final layer) (layers)

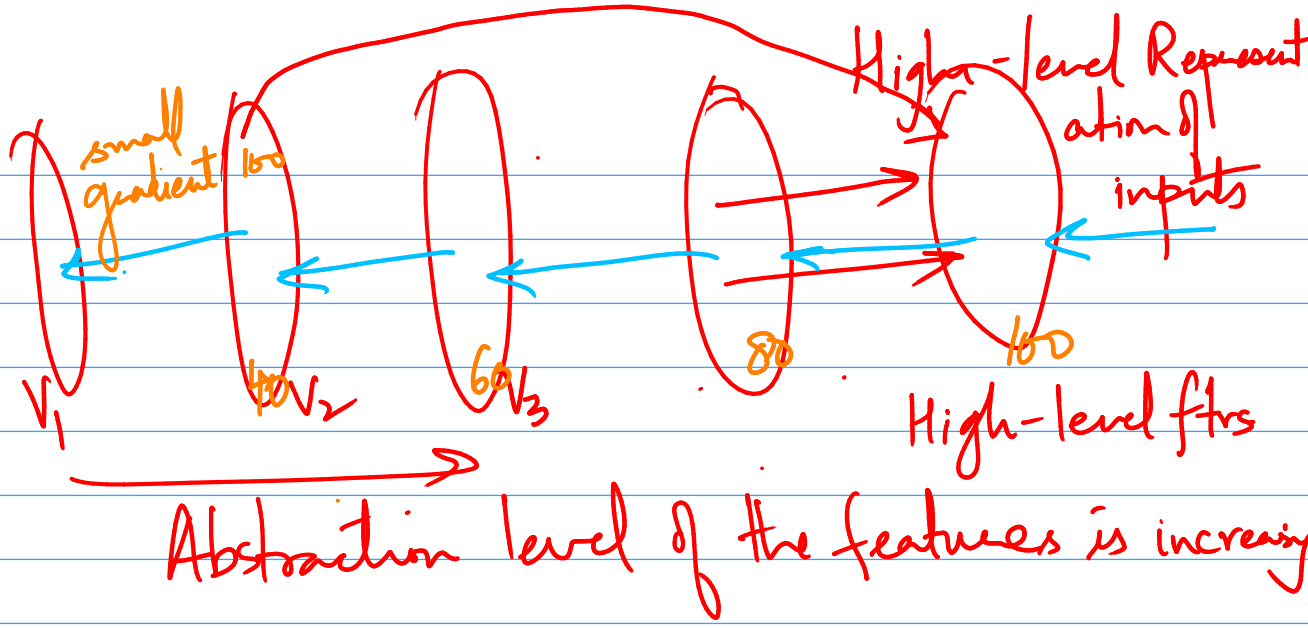
for $i = 1, \dots, k_t$ (nodes in a layer)

$$a_{t,i} = \sum_{j=1}^{k_{t-1}} o_{t-1,j} \, W_{t-1,i,j}$$

$a_{t,i}$: activation; $o_{t-1,j}$: prev; $W_{t-1,i,j}$: weights; sum: Affine

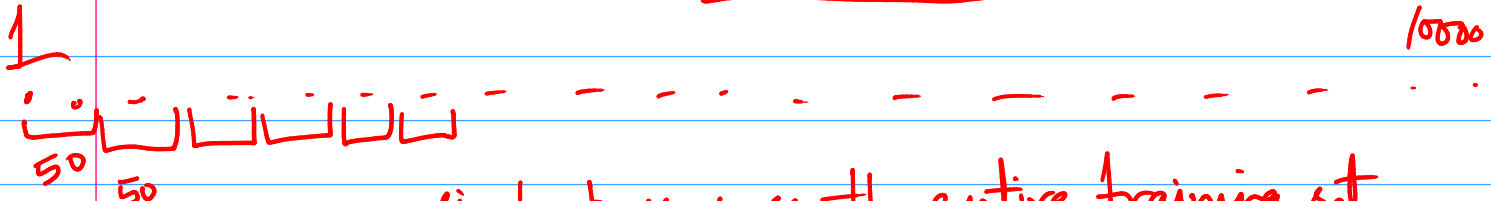
$$o_{t,i} = \sigma(a_{t,i})$$

σ: non-linearity



10,000

shuffled.

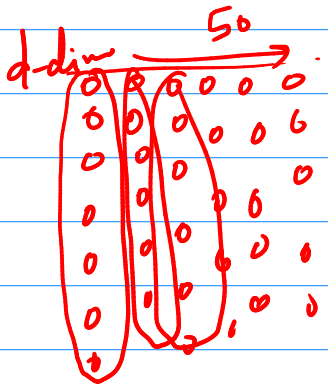


single pass over the entire training set (visited all the examples)

Epoch

↓

loss for the minibatch is used to compute the gradients.



Backward Propagation :

gradient of the final loss

set $\delta_T = (O_T - y)$ where $\delta_T \equiv J_{O_T}(l_T)$ ← scalar

δ_T, δ_{T-1}, δ_{T-2}

final layer ... initial layer

for $t = T-1, T-2, \ldots, 1$

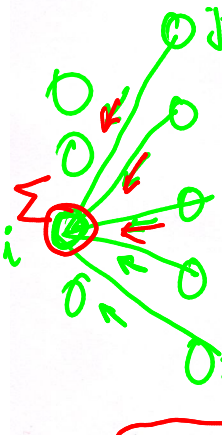
① for $i = 1, \ldots, n_t$

$$\delta_{t,i} = \sum_{j=1}^{n_{t+1}} W_{t,j,i}\, \delta_{t+1,j}\, \sigma'(a_{t+1,j})$$

$\delta_{t,i} = J_{O_t}(l_t)$; $J(a_t)$; $J(O_t)$; $J(O^*)$

Back propagation

incoming gradients from outgoing links.



② for each edge $(v_{t-1,j}, v_{t,i}) \in E$

set the partial derivative to $\delta_{t,i}\, \sigma'(a_{t,i})\, O_{t-1,j}$

$J(l_t)$ / w_{t-1}

used to update the weight using SGT

Loss Functions used for Neural Network Training

Regression Tasks

Use the squared error loss

$$MSE\_Loss(w) = \frac{1}{2} \mathop{\mathbb{E}}_{x,y \sim s} \| y - f(x;\theta) \|^2$$

target GT ; o/p of neural net

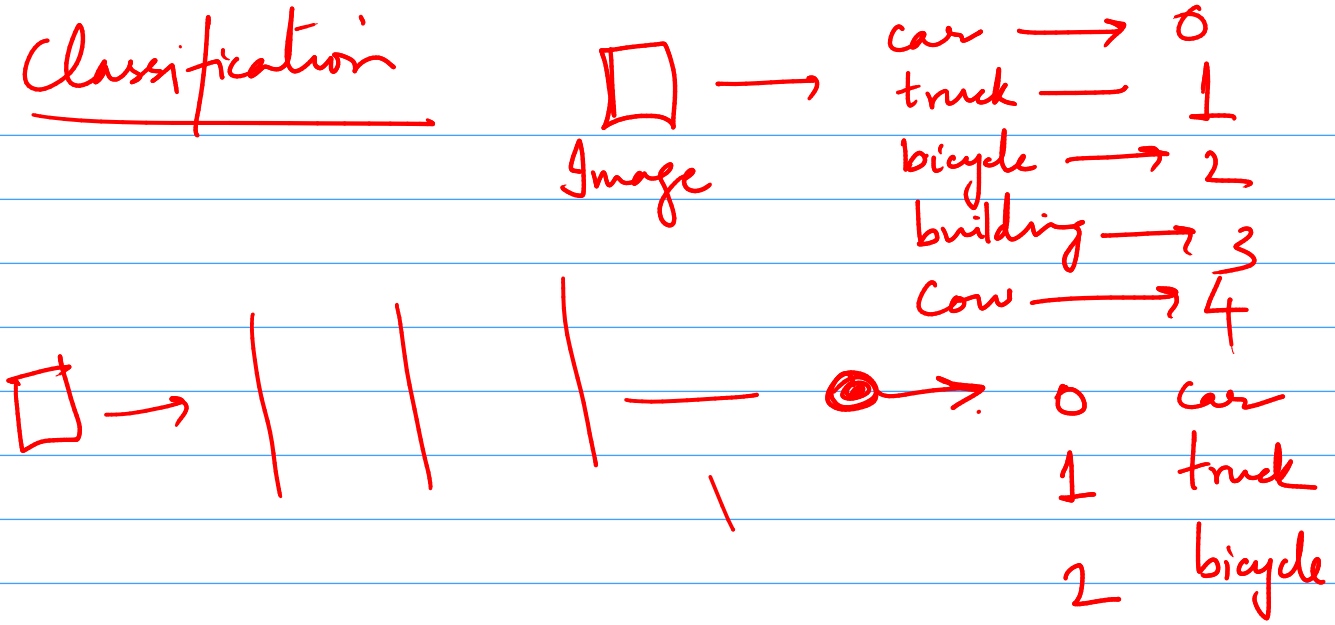
Classification tasks

Cross entropy between the training data distribution and model distribution

GT label

$$XentropyLoss = - \mathop{\mathbb{E}}_{x,y \sim s} \log \left(p_{model}(y \mid x) \right)$$

Classification



GT $y_{car} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ $y_{cow} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

1-hot encoding of target labels.

GT prob distribution of the target [ ] 1-hot vector

Network produces a prob distr over the target.

Cross entropy gives distance between two prob distributions

$$CE(P, \hat{P}) = \sum_i p_i \log \frac{1}{\hat{p}_i}$$