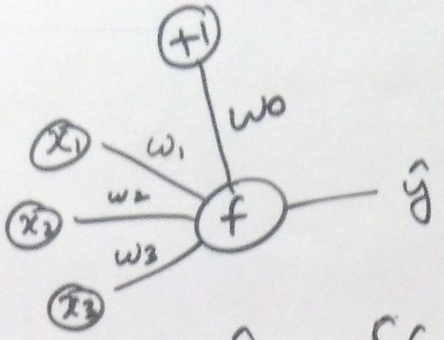
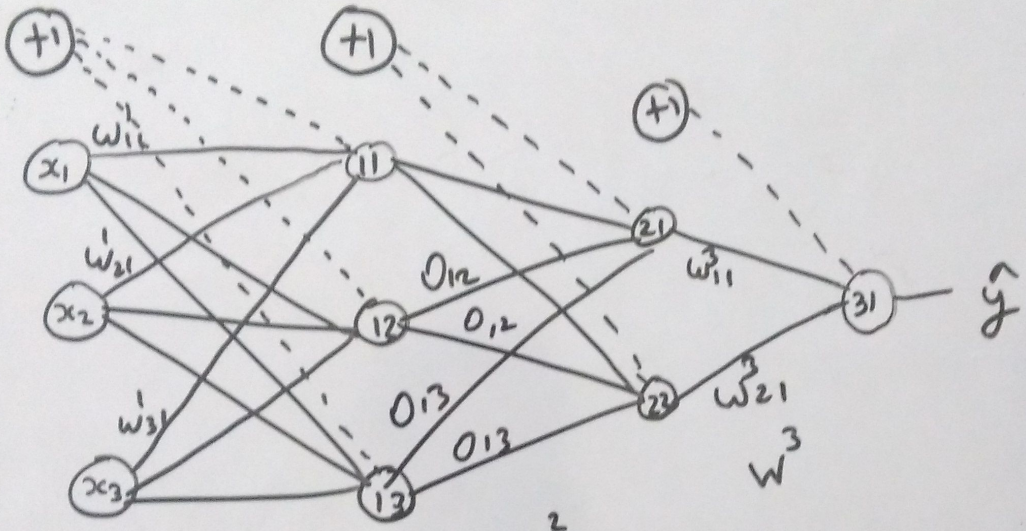
DL



Logistic regression unit
(Single neuron)

$$\hat{y} = f(w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3)$$



W^1 W^2

o/p layer1 layer2 o/p

$$W^1 = \begin{bmatrix} w_{01} & w_{02} & w_{03} \\ w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{matrix} \rightarrow \text{Bias} \\ \rightarrow x_1 \\ \rightarrow x_2 \\ \rightarrow x_3 \end{matrix}$$

$(3+1) \times 3$

$$W^2 = \begin{bmatrix} w_{01} & w_{02} \\ w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \begin{matrix} \rightarrow f_{10} \\ \rightarrow f_{11} \\ \rightarrow f_{12} \\ \rightarrow f_{13} \end{matrix}$$

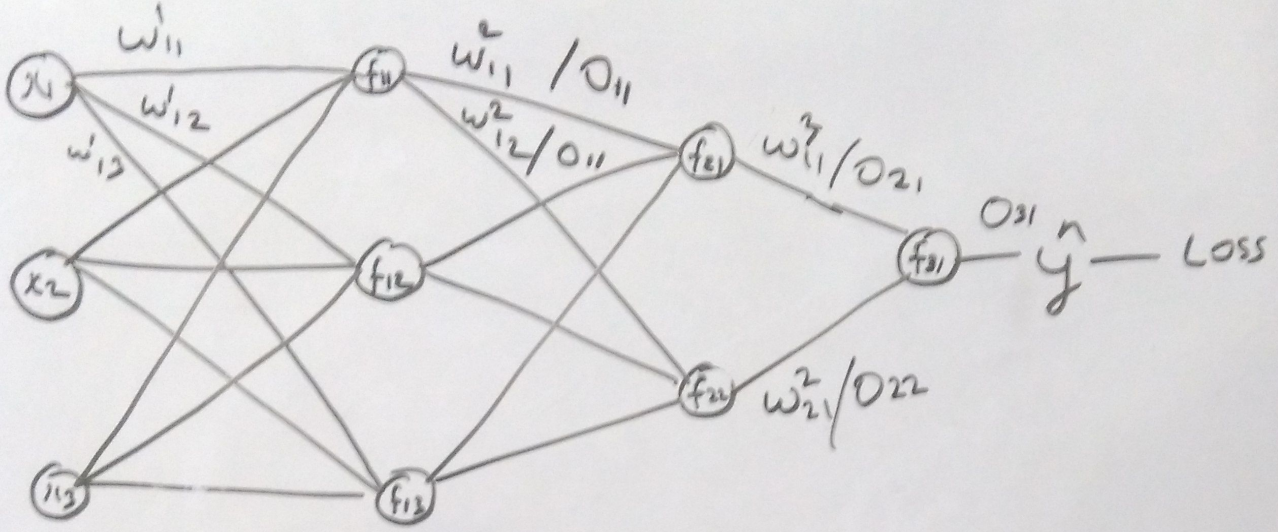
$\downarrow$ 21 $\quad$ $\downarrow$ $(1+3) \times 2$ 22

$$W^3 = \begin{bmatrix} w_{01} \\ w_{11} \\ w_{22} \end{bmatrix} \begin{matrix} \rightarrow f_{20} \\ \rightarrow f_{21} \\ \rightarrow f_{22} \end{matrix}$$

$(1+2) \times 1$

DL

Training of MLP

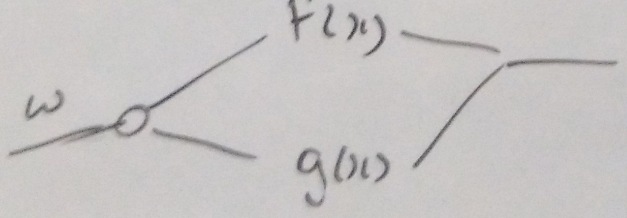


here $O_{31} = \hat{y}$

$$\frac{\partial L}{\partial w^3_{11}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w^3_{11}}$$

$$\frac{\partial L}{\partial w^2_{11}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{w^2_{11}}$$

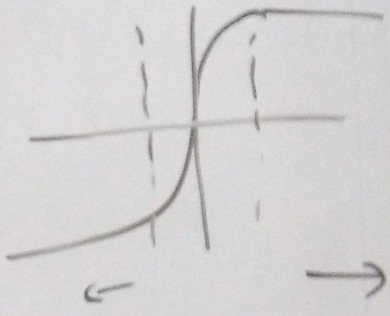
$$\frac{\partial L}{\partial w^1_{11}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{11}} \cdot \frac{\partial O_{21}}{\partial O_{11}} \cdot \frac{\partial O_{11}}{\partial w^1_{11}} + \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{22}} \cdot \frac{\partial O_{22}}{\partial O_{11}} \cdot \frac{\partial O_{11}}{\partial w^1_{11}}$$



DL

Training of MLP

Problem of Sigmoid in Backpropagation



Max slope of Sigmoid = 0.5

$$\frac{\partial \sigma}{\partial w} = 0.5 \times 0.1 \times 0.2 \times \cdots$$

- lead to very very small update to initial layer. It is known as vanishing gradient problem.

Exploding gradient problem:

Gradient become very-very large. hence lead to overshooting and failed to converge.

$$\frac{\partial L}{\partial w} = 10 \times 10 \times 10 \cdots$$

These occur due to chain rule in Backpropagation. and our activation functions structure.

Deep Learning

Activation Unit

① Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$; $\sigma'(x) = \sigma(x)\cdot(1-\sigma(x))$

$[0,1]$

② tanh : $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$; $f'(x) = 1 - f(x)^2$

③ Relu; $f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$; $f'(x) \begin{cases} 1 & ; x \geq 0 \\ 0 & ; x < 0 \end{cases}$

③ Leaky Relu: $f(x) = \begin{cases} x & ; x \geq 0 \\ 0.01x & ; x < 0 \end{cases}$;

④ SoftPlus : $f(x) = \log(1+e^{x})$; $f'(x) = \frac{1}{1+e^{-x}}$

Deep Learning

① Uniform initialization

$$w_{ij}^{k} \sim \text{Unif}\left(-\frac{1}{\sqrt{fan\_in}}, \frac{1}{\sqrt{fan\_in}}\right)$$

② Xavior / Glorot initialization (Sigmoid)

- Uniform $w_{ij}^{k} \sim \text{Unif}\left(-\frac{\sqrt{6}}{\sqrt{fan\_in + fan\_out}}, \frac{\sqrt{6}}{\sqrt{fan\_in + fan\_out}}\right)$

- Normal

$$w_{ij}^{k} \sim \text{Normal}(0, \alpha) \; ; \alpha = \frac{2}{fan\_in + fan\_out}$$

③ He initialization (Relu)

- Uniform $w_{ij}^{k} \sim \text{Unif}\left[-\sqrt{\frac{6}{fan\_in}}, \sqrt{\frac{6}{fan\_in}}\right]$

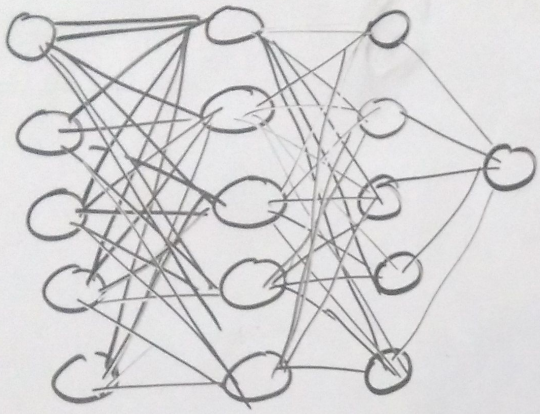
- Normal

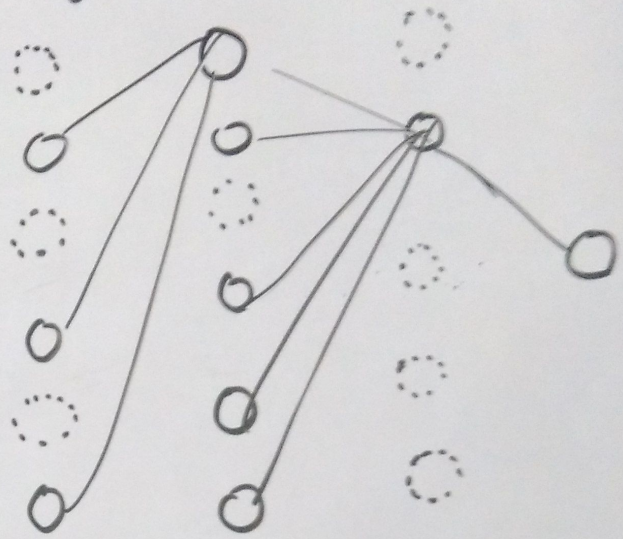
$$w_{ij}^{k} \sim N(0, \alpha) \; ; \alpha = \sqrt{\frac{2}{fan\_in}}$$

Deep Learning

Dropout

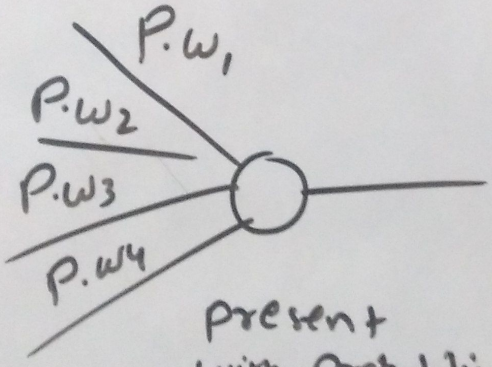
(Randomization for Regularization)





Dropout Probability P=0 =0 =0 =0. 0.5 0.2 0.8

Test Time



present with probability P in train time

Deep Learning

Batch Normalization

- Protect from Bad initialization
- Solve Problem of Internal Covariance shift.

B1:
B2: (x) → L_1 → L_2 → L_3 → L_4 → L_5 → (O/P)
B3:

for each Batch (Parameter to learnt):

$$\mu_B = \frac{1}{m}\sum_{i=1}^{B} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{B} (x_i - \mu_B)^2$$

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$\therefore \epsilon : 0.0001$ (Protect from div by zero)

$BN_{\gamma,\beta}(x_i)$: $\boxed{\tilde{x_i} = \gamma \cdot \hat{x_i} + \beta}$ Batch Normalised data.

Test time

μ_B : running avg of μ_B at train time.

σ_B^2 : Running avg. of σ_B^2 at train time

β, γ : learn by optimization.

Deep Learning - Optimizers

- Simple Model (SVM, Log. Reg.) : Convex loss
- Deep Learning Model : Non-convex losses / saddle points

SGD with Momentum:

idea: exponential weighted Average.

$$v_t = a_t + \gamma \cdot a_{t-1} + \gamma^2 \cdot a_{t-2} + \gamma^3 \cdot a_{t-3}$$

→ Reduced impaced. if $\gamma < 1$

$$v_1 = \eta \cdot g_t \quad \text{// initial}$$

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot g_t$$

$$\omega_t = \omega_{t-1} - v_t$$

here η = Learning rate

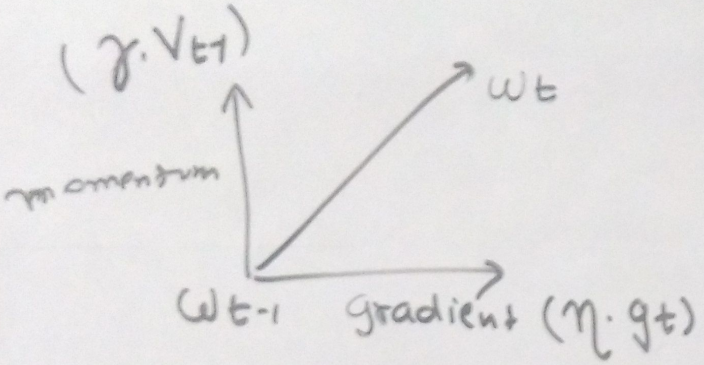
g_t = gradient update to make ω_t

$\gamma = 0.9$ // generally.

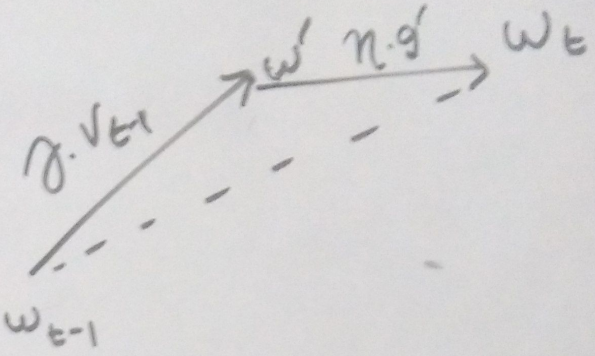
$$\omega_t = \omega_{t-1} - (\eta \cdot g_t + \gamma\eta \cdot g_{t-1} + \gamma^2\eta \cdot g_{t-2} \cdots)$$

Deep Learning - Optimizers

Nestrov Accelerated Gradient (NAG)



(SGD with Momentum)



(Nestrov)

- first go with Momentum then calculate gradient then get w_t.

$$w' = w_{t-1} - \gamma \cdot V_{t-1}$$

$$g' = \left(\frac{\partial L}{\partial w}\right)_{w_{t-1} - \gamma \cdot V_{t-1}}$$

$$w_t = w' - \eta \cdot g'$$

or

$$\boxed{w_t = w_{t-1} - (\gamma \cdot V_{t-1} + \eta \cdot g')}$$

Deep Learning - Optimizers

Adagrad (adaptive gradient)

each weight need to have its own adaptive learning rate.

$$\omega_t = \omega_{t-1} - \eta \cdot g_t$$

$$\eta = \frac{c}{\sqrt{\alpha_{t-1} + \epsilon}} \quad \therefore \text{here } c = 0.01$$

$$\alpha_{t-1} = \sum_{i=1}^{t-1} g_i^2$$

as iteration increases learning rate automatically decrease.

Adadelta (Similar RMSprop)

for long iteration η is very low hard to Converge

all same as Adagrad except α_{t-1}

$$\alpha_{t-1} = \sum_{i=1}^{t-1} g_i^2 \rightarrow \boxed{(1-\gamma) \cdot g_{t-1}^2 + \gamma \cdot \alpha_{t-2}}$$

here $\gamma = 0.95$ // generally.

Deep Learning - Optimizers

Adam (Adaptive Moment Estimation)

mean - 1st Moment

Variance - 2nd Moment

// $m_0 \leftarrow 0$

// $v_0 \leftarrow 0$

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 \cdot v_{t-2} + (1-\beta_2) g_t^2$$

// generally $\beta_1 = 0.9$; $\beta_2 = 0.99$

$$\hat{m}_t = \frac{m_t}{1-(\beta_1)^t} \quad ; \quad \hat{v}_t = \frac{v_t}{1-(\beta_2)^t}$$

$$W_t = W_{t-1} - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

Deep Learning - Otheos

Gradient Clipping

W = [2000 | | | |]

(mainly moniter initial layer for gradient vanishing and Exploding Problem.

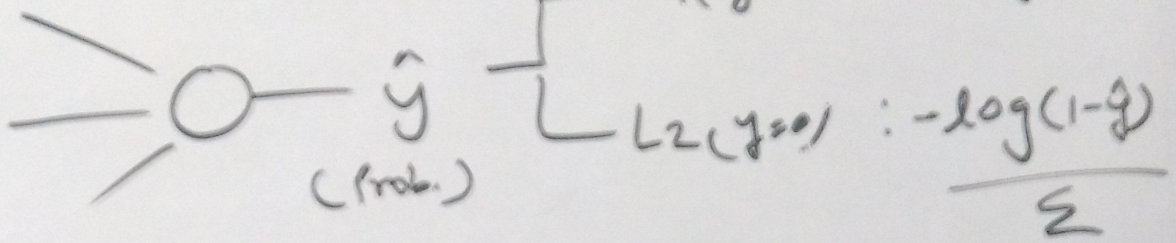
$$G_{new} = \frac{G}{\|G\|} \cdot 2$$

$\therefore \|G\| = \sqrt{G_1^2 + G_2^2}$

— It makes value b/w 0 and 1. so resolve problem of vanishing gradient.

Deep Learning - Otheros

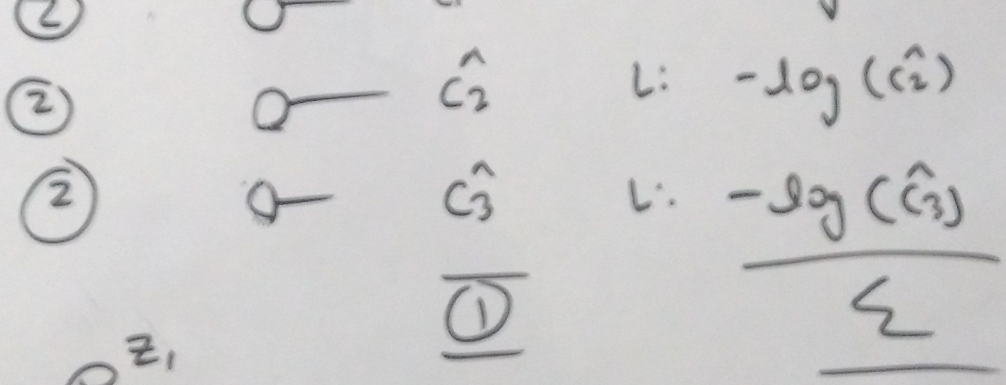
Logistic Regression [Sigmoid + log loss]



log loss:

$$\frac{1}{m}\sum[-y\log(\hat{y}) - (1-y)\log(1-\hat{y})]$$

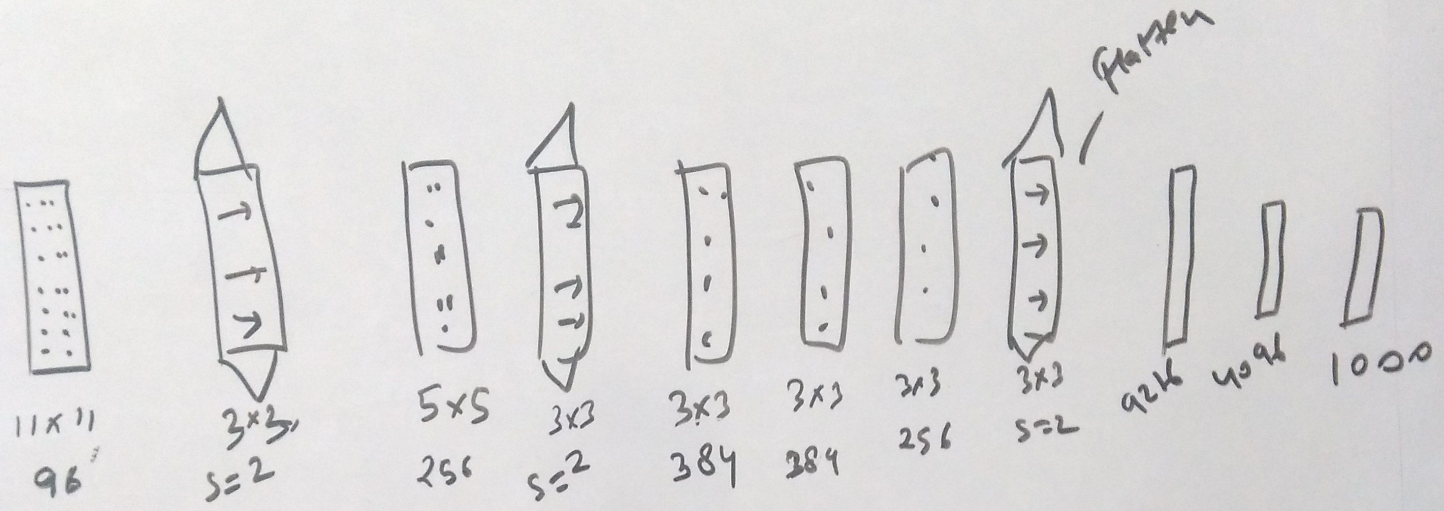
Multiclass



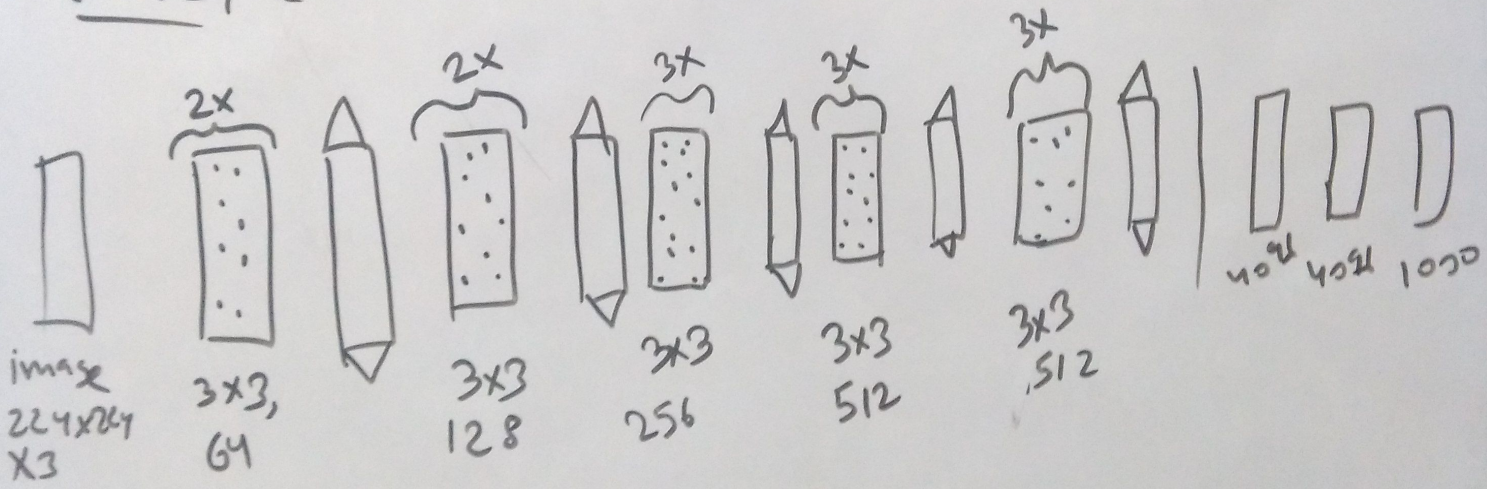
Softmax (Normalized Exponent) $= \frac{e^{z_i}}{\sum_{i=1}^{k} e^{z_i}}$ ——— use cross entropy loss

Deep Learning - Others

AlexNet (2012)

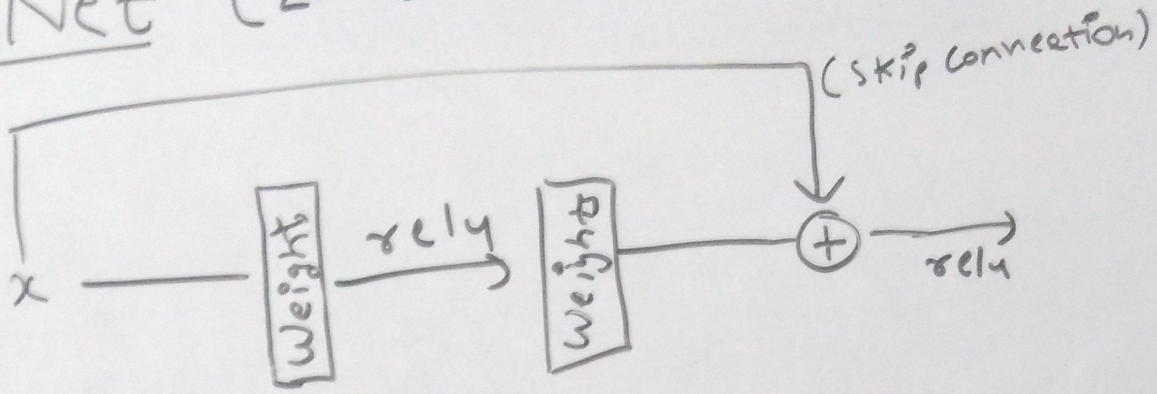


VGG (2014/2015)



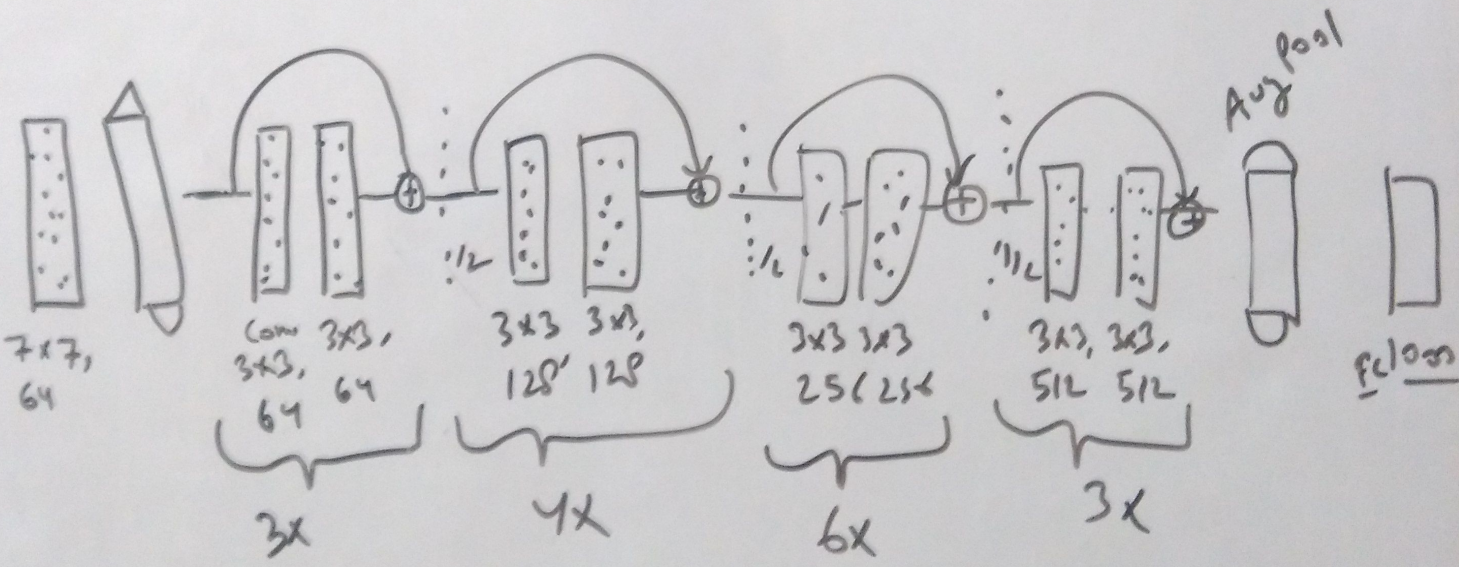
Deep Learning - Others

ResNet (2015)



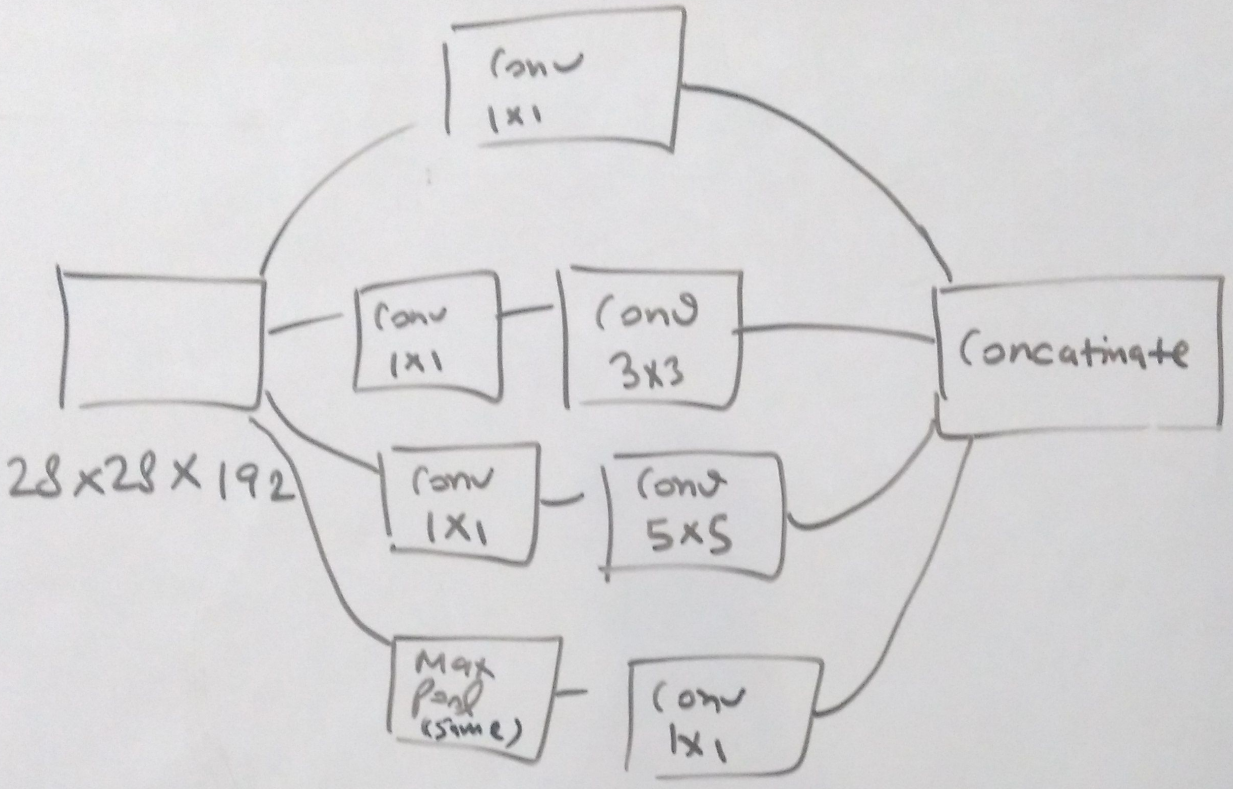
- Deeper Network lead to Reduce performance Solved this.

34 layer Residual



Deep Learning - Others

Inception (GoogLeNet) (2015)

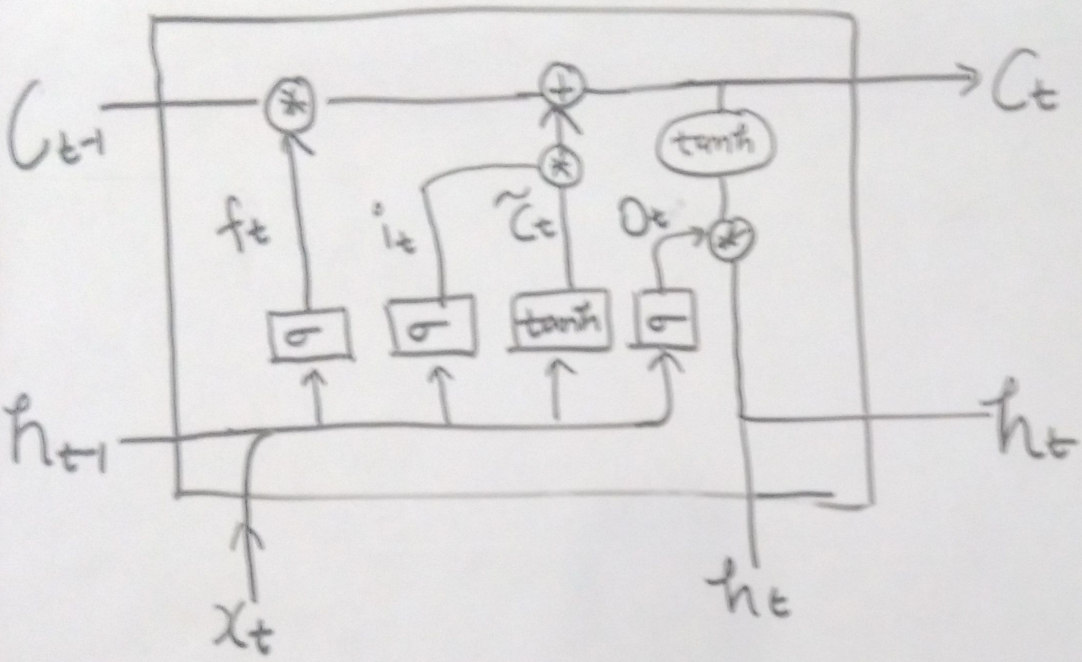


Inception V2

(googlenet + ResNet)

Deep Learning - Otheos

LSTM (Long short term memory)



$$f_t = \sigma(W_f[x_t, h_{t-1}] + b_f)$$

$$i_t = \sigma(W_i \cdot [x_t, h_{t-1}] + b_i)$$

$$\tilde{C}_t = \tanh(W_c \cdot [x_t, h_{t-1}] + b_c)$$

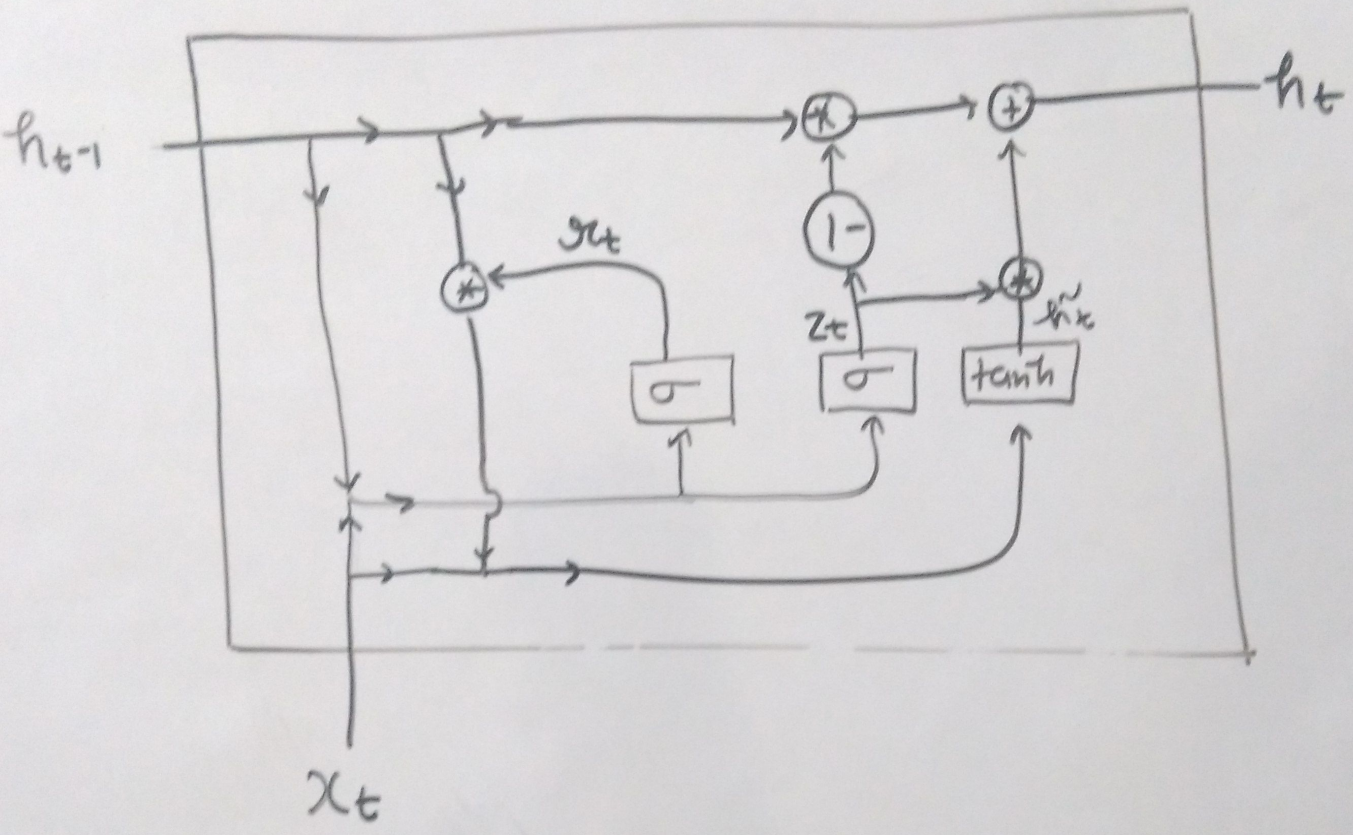
$$O_t = \sigma(W_o \cdot [x_t, h_{t-1}] + b_o)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$h_t = O_t * \tanh(C_t)$$

Deep Learning - Others

GRU (Gated Recurrent Unit)



$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$\tilde{h}_t = \tanh(W_{\tilde{h}} \cdot [r_t * h_{t-1}, x_t] + b_{\tilde{h}})$$

$$h_t = h_{t-1} * (1 - z_t) + \tilde{h}_t * z_t$$