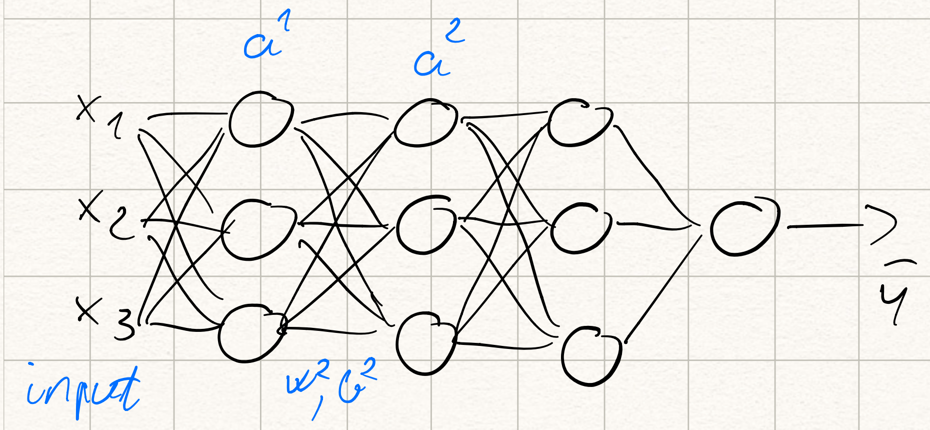
Normalizing inputs to speed up learning



Normalizing input

$$\mu = \frac{1}{m} \sum_i x^i$$

$$x = x - \mu$$

$$\sigma^2 = \frac{1}{m} \sum_i x^{i^2}$$

$$x = x / \sigma^2$$

Can we normalize $a^1, a^2 \dots$ to train $w^2, b^2, w^3, b^3 \dots$ faster

=> <u>Batch norm</u> : normalize $z^1, z^2 \dots$

Implementing batch norm

=> given some intermediate values in NN $(z^1 \dots z^m)$

1) $\mu = \frac{1}{m} \sum_i z^i$

2) $\sigma^2 = \frac{1}{m} \sum_i (z^i - \mu)^2$

3) $z^i_{norm} = \dfrac{z^i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$ — for numeric stability

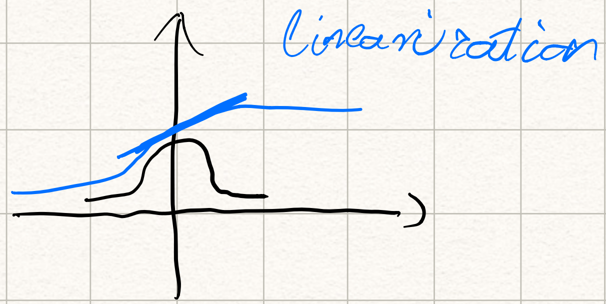
4) $\underbrace{\tilde{z}^i = \gamma\, z^i_{norm} + \beta}_{\text{learnable parameters}}$

if we don't want to the hidden units between mean zero and variance we want different distribution

use $\tilde{z}^i$ for later computation instead of z^i

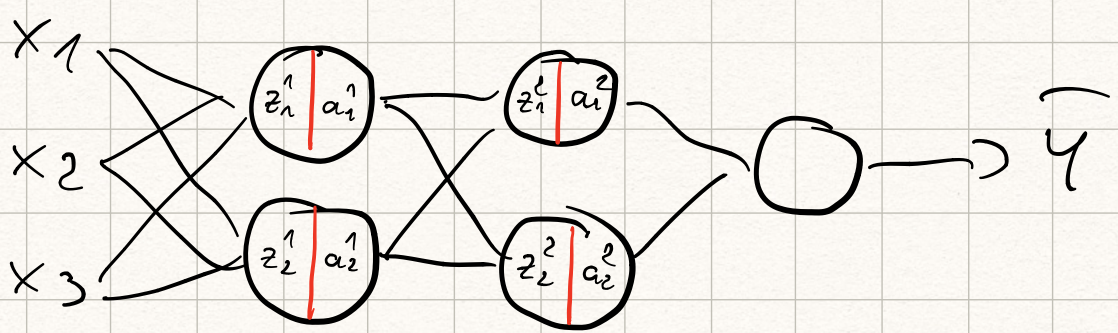
using 3) i.e z^i norm

$\Rightarrow$



$\Rightarrow$ so we can use 4) to make sure that $\tilde{z}^i$ is the range what we want

Fitting batch norm into a NN



$$X \xrightarrow{w^1,\, b^1} z^1 \xrightarrow[\text{Batch norm (BN)}]{\beta^1,\, \gamma^1} \tilde{z}^1 \longrightarrow a^1 = g^1(\tilde{z}^1) \xrightarrow{w^2,\, b^2} z^2 \xrightarrow[\text{BN}]{\beta^2,\, \gamma^2} \tilde{z}^2 \dots$$

Parameters:

$$\left.\begin{matrix} w^1, b^1, w^2, b^2 \dots w^l, b^l \\ \beta^1, \gamma^1, \beta^2, \gamma^2 \dots \beta^l, \gamma^l \end{matrix}\right\}$$

using gradient descent, RMSprop, Adam ...
to optimize the parameters

$$d\beta^l \Rightarrow \beta^l = \beta^l - \eta\, d\beta^l$$

In practice working with mini batches

$$x^{\{1\}} \xrightarrow{w^1, b^1} z^1 \xrightarrow{\beta^1, \gamma^1} \tilde{z}^1 \rightarrow a^1 \dots$$

$$x^{\{2\}} \rightarrow z^1 \xrightarrow[BN]{\beta^1, \gamma^1} \tilde{z}^1 \rightarrow a^1 \dots$$

$$z^l = w^l a^{l-1} + \cancel{b^l}$$

batch norm zero mean eliminate the effect of b^l

$$z^l = w^l a^{l-1}$$

$$z^l_{norm} \Rightarrow \tilde{z}^l = \gamma z^l_{norm} + \boxed{\beta^l}$$

instead of b^l

Implement gradient descent with batch norm

for $t = 1 \dots$ num of minibatch

compute forward prop on $x^{\{t\}}$

In each hidden layer use BN to compute $\tilde{z}^l$

use back prop to compute:

$$dw^l, \cancel{db^l}, d\beta^l, d\gamma^l$$

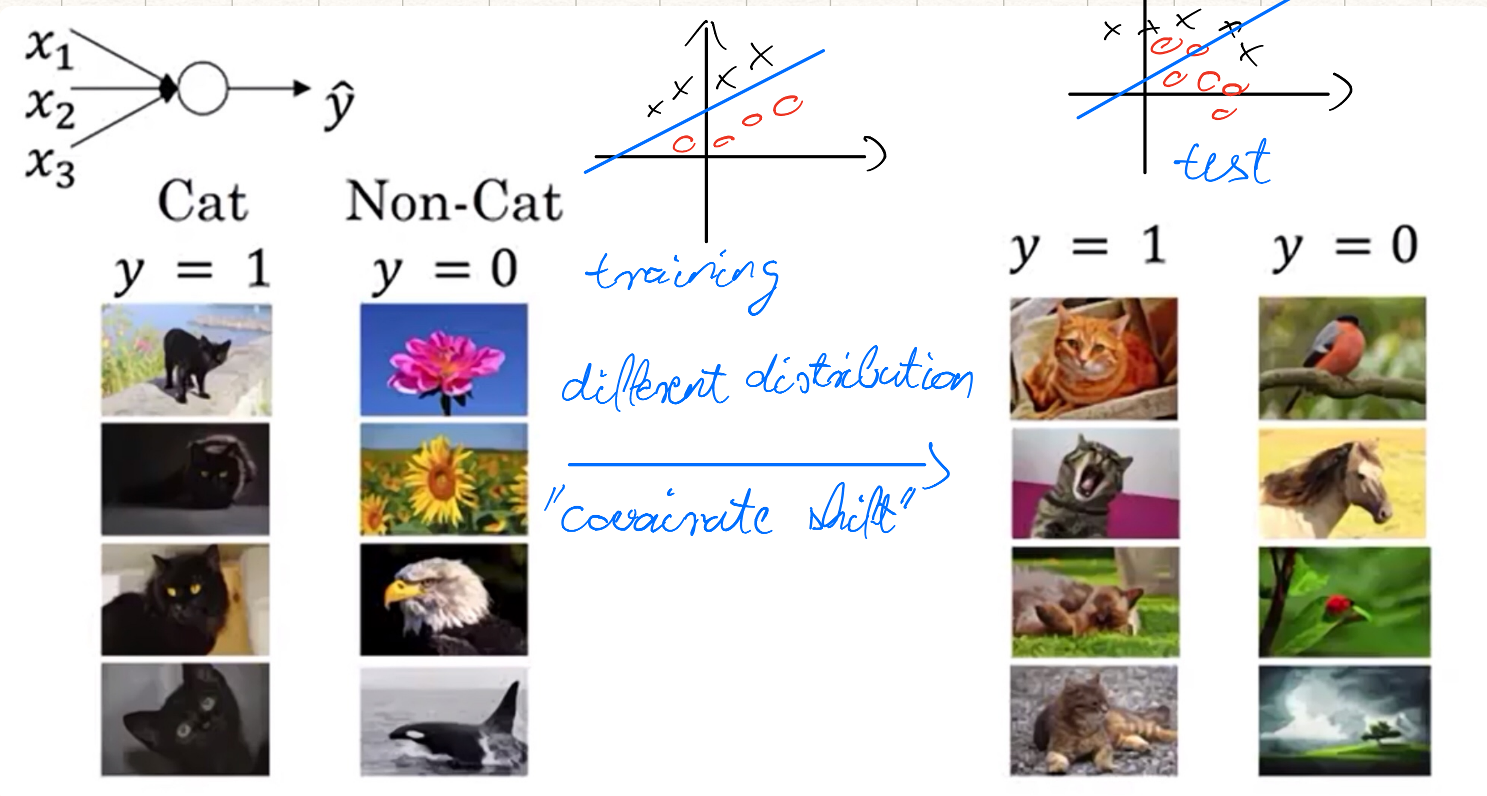
update:

$$\left.\begin{aligned} w^l &= w^l - \alpha\, dw^l \\ \beta^l &= \beta^l - \alpha\, d\beta^l \\ \gamma^l &= \gamma^l - \alpha\, d\gamma^l \end{aligned}\right\}$$

RMSprop, Adam, gradient descent

Normalize the values of the hidden units between zero mean and variance one speed up the learning.

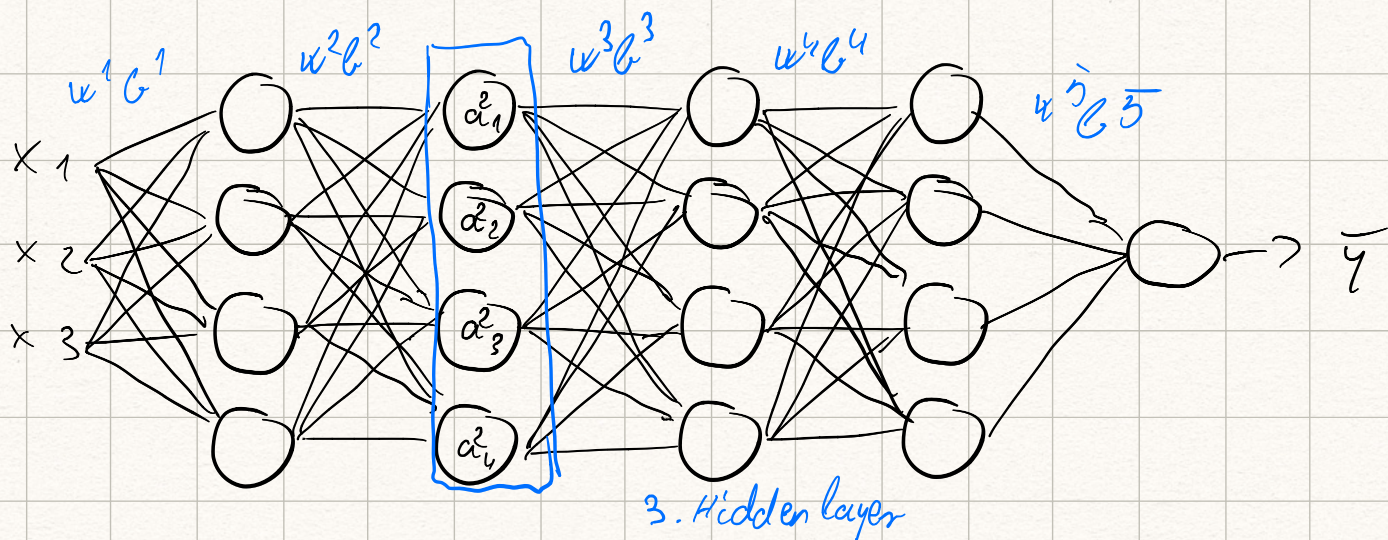
Learning on shifting input distribution



training only black cats

color cats on test

⇓ same effect on deep nets



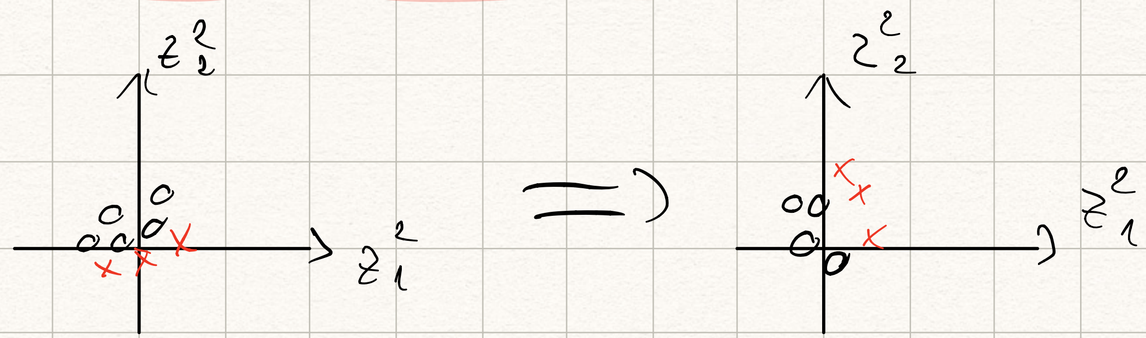
3. Hidden layer job is to map a^2 to $\bar{y}$ =>
learn w^3, b^3

=> If w^1, b^1, w^2, b^2 change => a^2 also change

=> from the perspective of the 3. hidden unit w^1, b^1, w^2, b^2 change all the time

=> suffering of the problem "covariate shift"

=> **Batchnorm reduce the distribution**



Because of batch norm mean and variance is the same no matter that the values change

- limit the effect of the previous layer's values changes to the later layers
- easier to learn to the later layers

Batch norm as regularization

Each mini batch is scaled by mean / variance computed on just the given mini batch

- ⇒ this adds some noise to the values z, similar to Dropout, it adds some noise to each hidden layer's activations
  - ⇒ a slight regularization effect

Batch size : 64 , 128 ... 512 $\longrightarrow$ reduce regularization effect

Batch norm at test time

Predicting a single example ⇒ no mini-batch ⇒

- using exponentially weighted average to compute a mean and a variance across the mini batches during the training

use this mean and variance at Test time to compute:

$$z^{i}_{norm} = \frac{z^{i} - \mu}{\sqrt{\sigma^2 + \varepsilon}} , \quad \tilde{z}^{i} = \gamma z^{i}_{norm} + \beta$$