

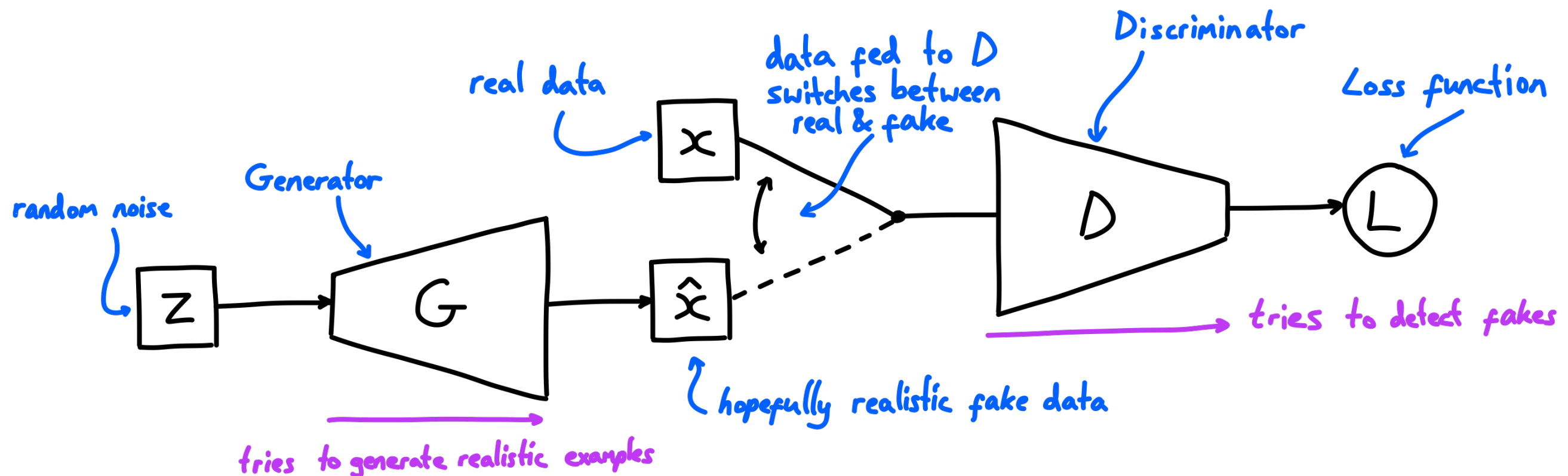
## What The hell is a GAN?

GANs are state of the art generative models, that can use unsupervised data to learn to create new content from the same distribution

This is done by battling two neural networks

One network, the generator, is trained to transform random noise into realistic training data

Another network, the discriminator, is trained to classify that data as either generated or real



## Training GANs

Joint loss function for D & G

$$\mathcal{L} = \min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p_z} \log [1 - D_{\theta_d}(G_{\theta_g}(z))] \right]$$

avg over real data  
 is real data real?  
 avg over randomly generated noise  
 fake data  
 is fake data real?  
 → G wants  $D(G) \rightarrow 1$   
 → D wants  $D(G) \rightarrow 0$

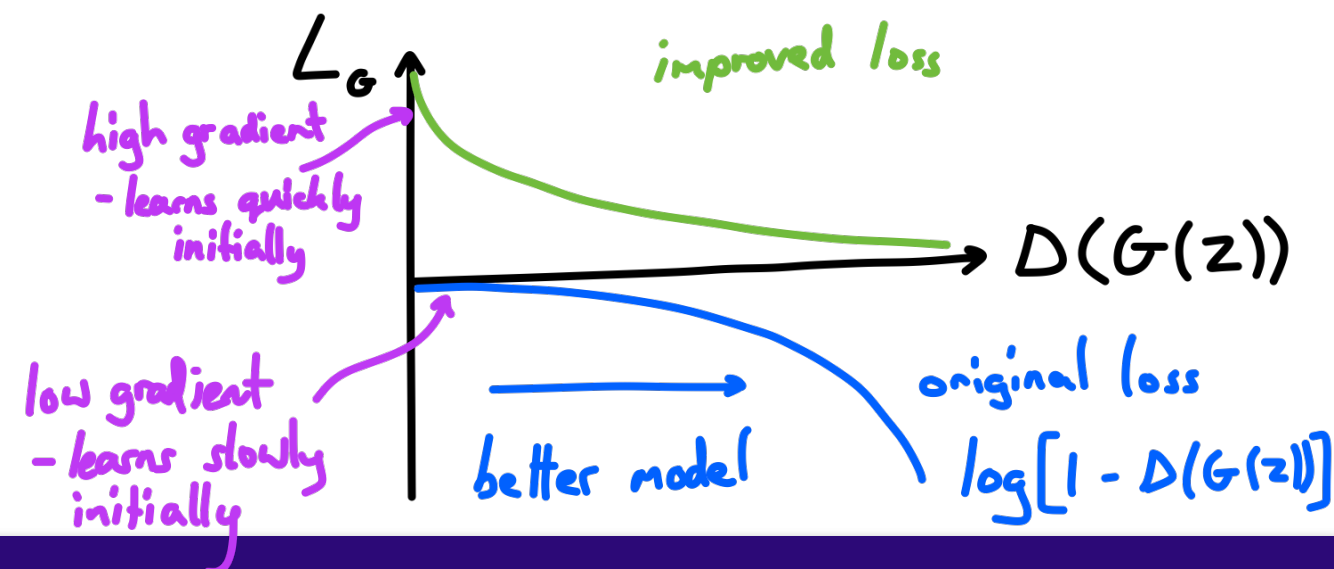
### Discriminator loss function

$$\mathcal{L}_D = - \left[ \mathbb{E}_{x \sim p_{\text{data}}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p_z} \log [1 - D_{\theta_d}(G_{\theta_g}(z))] \right]$$

### Generator loss function

~~$$\mathcal{L}_G = \mathbb{E}_{z \sim p_z} \log [1 - D_{\theta_d}(G_{\theta_g}(z))]$$~~

$$\mathcal{L}_G = - \mathbb{E}_{z \sim p_z} \log [D_{\theta_d}(G_{\theta_g}(z))]$$



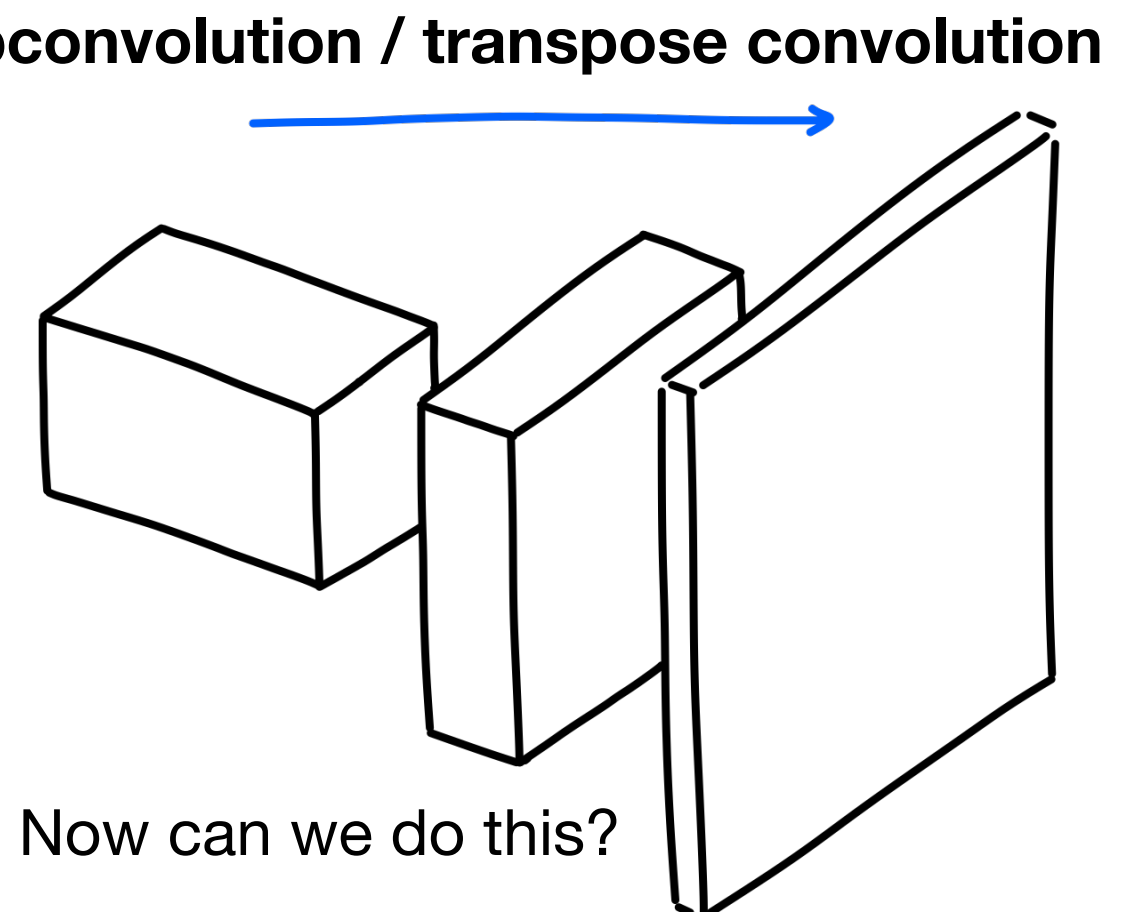
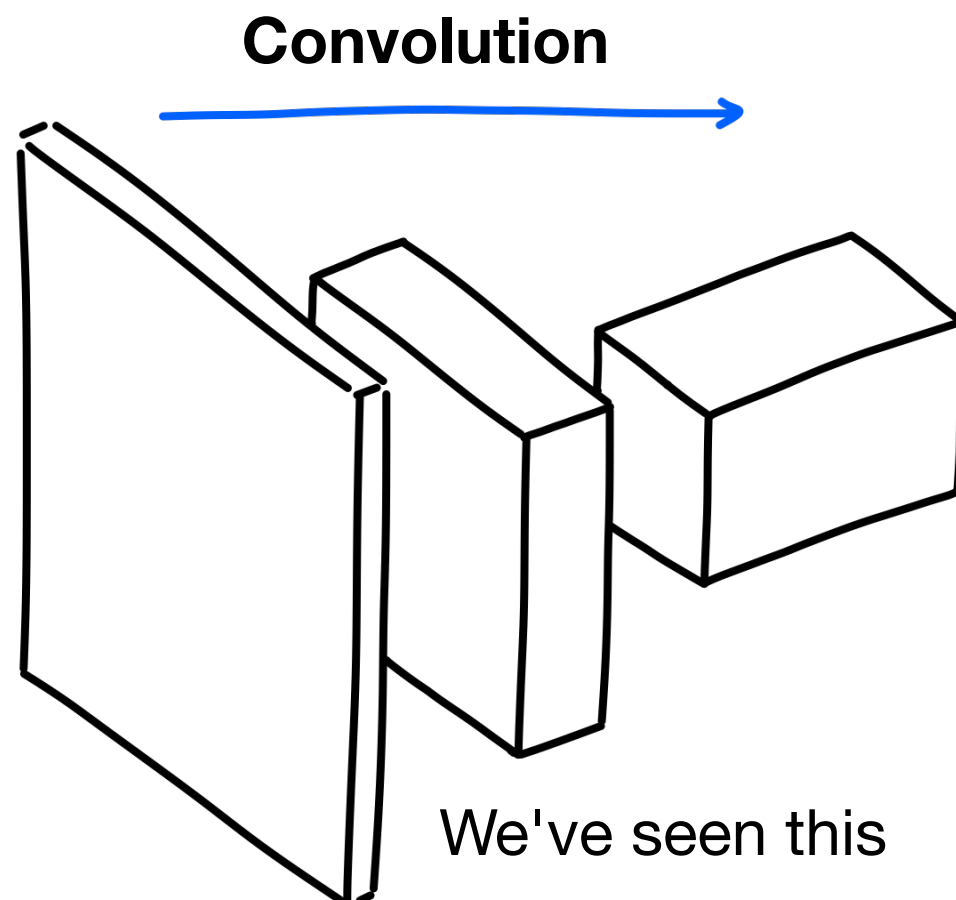
## Transpose convolutions

The random noise we will generate our fake data from is a low dimensional vector. Because we are generating images in our implementation, we will reshape this into a tensor, then upsample it in a learnable way.

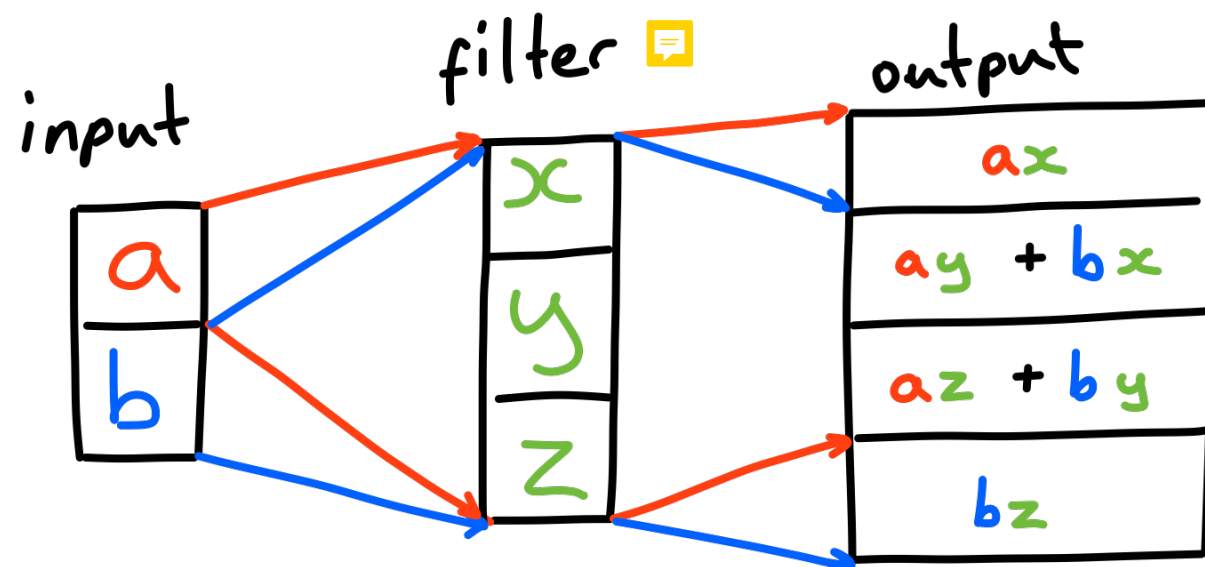
### Learnable upsampling retaining spacial information

We would like a way to upsample lower dimensional data, which is represented spatially

Like an inverse to convolution

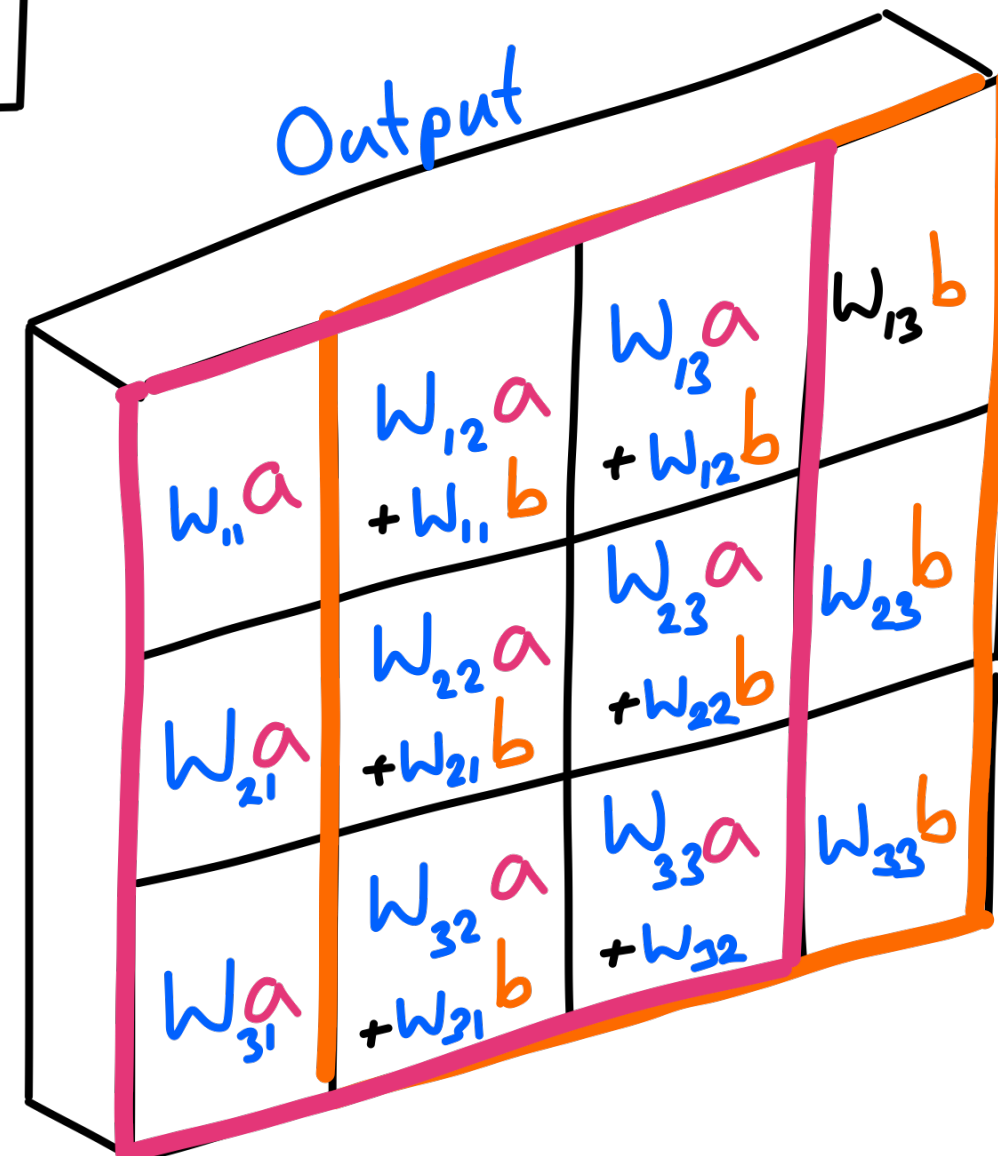
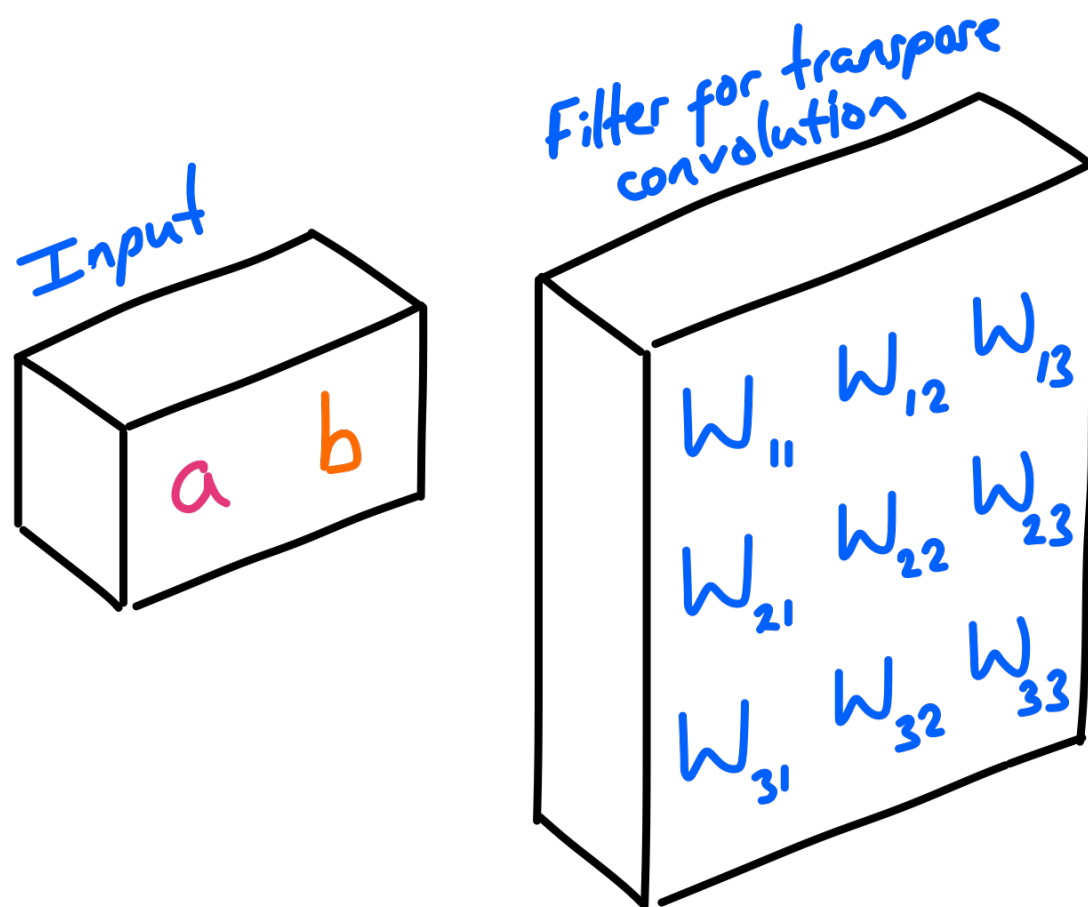


# Transpose convolutions



A convolution filter is weighted by each activation in the input

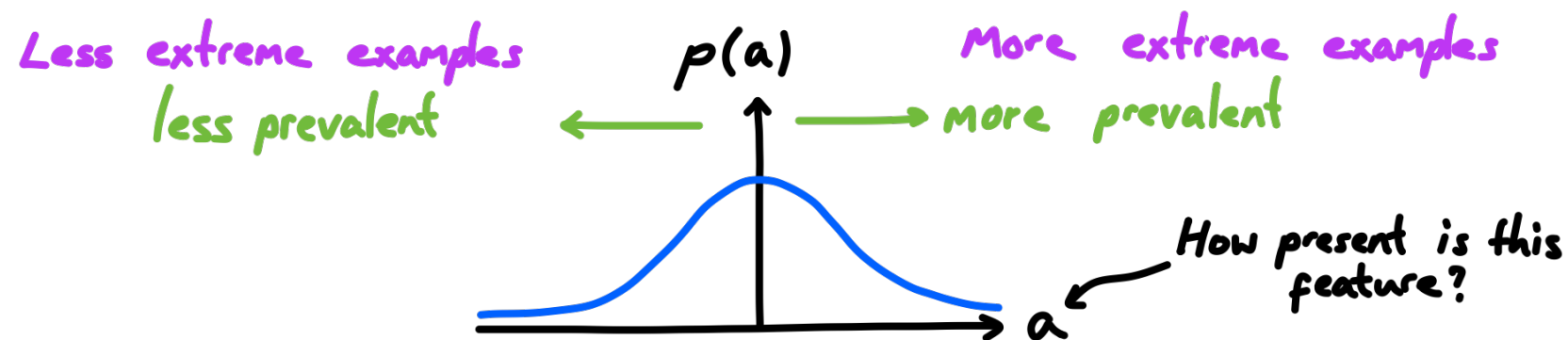
The stride gives the ratio of steps taken in output to steps taken in input




## Batch normalisation

Activations should represent the presence of some (perhaps complex feature) in the input data.

So it would be useful if they were distributed about some mean value.



Well if we want our activations to follow some distribution like this, then we can just make them by normalising them.

 Normalise activations:

$$a^{(k)'} = \frac{\text{original activations } a^{(k)} - \text{mean } E[a^{(k)}]}{\underbrace{\sqrt{\text{Var}[a^{(k)}]}}_{\text{standard deviation}}}$$

*normalised activations in layer k*

## Batch normalisation

However...

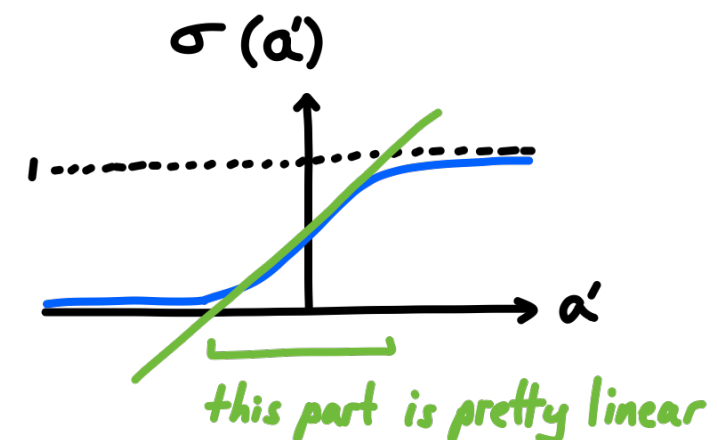
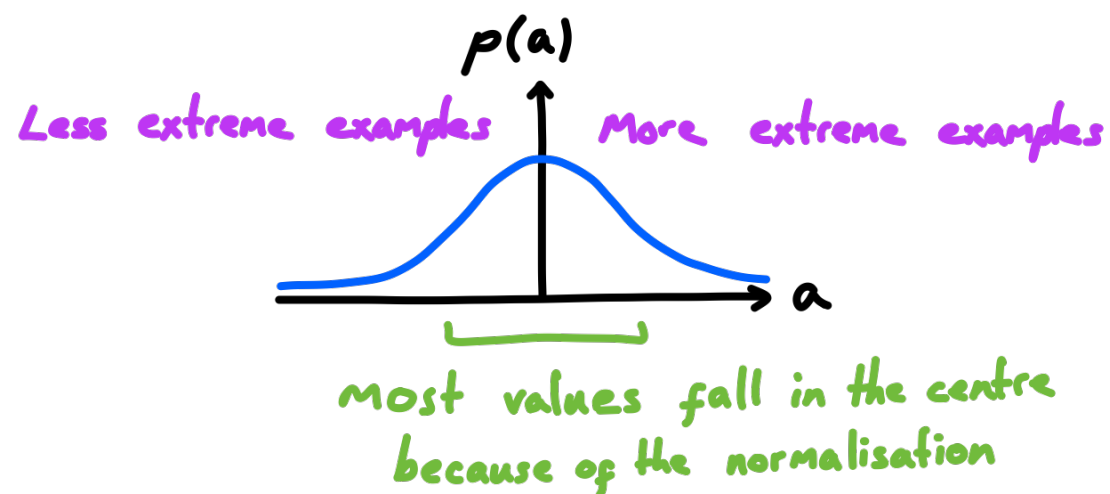
By normalising the activations, they are centred around zero, which has 2 effects:

### Problem 1

Some activation functions are almost linear around zero

The whole point of having them was to introduce non-linearity

The current normalisation we have done will hence limit the capacity of the model



## Batch normalisation

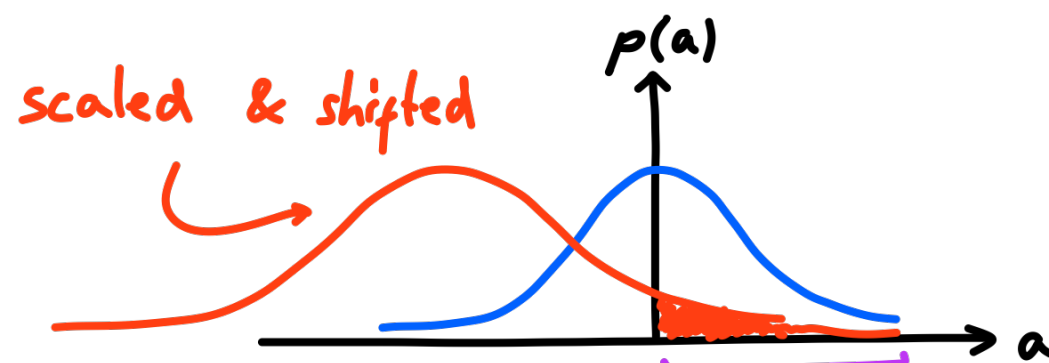
However...

By normalising the activations, they are centred around zero, which has 2 effects:

### Problem 2

Activations that might be important to later layers might require information that indicates not whether the input data had below/above mean presence of some feature, but whether that feature was above some threshold of extremity

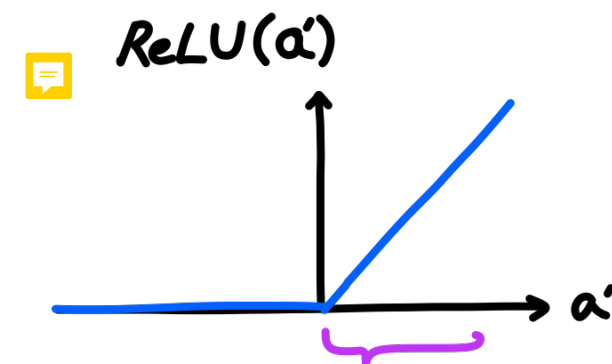
E.g we might only want to pass information through a particular node if we are DEFINITELY (say 95%) sure of something



95th percentile activations

→ extreme examples

e.g. I am definitely sure  
there's a person in this image



only the extreme examples pass information  
through this layer

→ only where  $a' > 0$



## Batch normalisation

So to counter these two problems...

...we introduce 2 more learnable parameters per activation

Computed per batch, over each activation, in each layer during training

Running train-time average is used during testing/deployment

↑  
train & test modes differ!

$$a^{(k)'} = \frac{a^{(k)} - E[a^{(k)}]}{\sqrt{\text{Var}[a^{(k)}]}} * \gamma^{(k)} + \beta^{(k)}$$

nice simple linear transformation  
↳ easily differentiable → learnable

shifts activations

flips & scales activations

