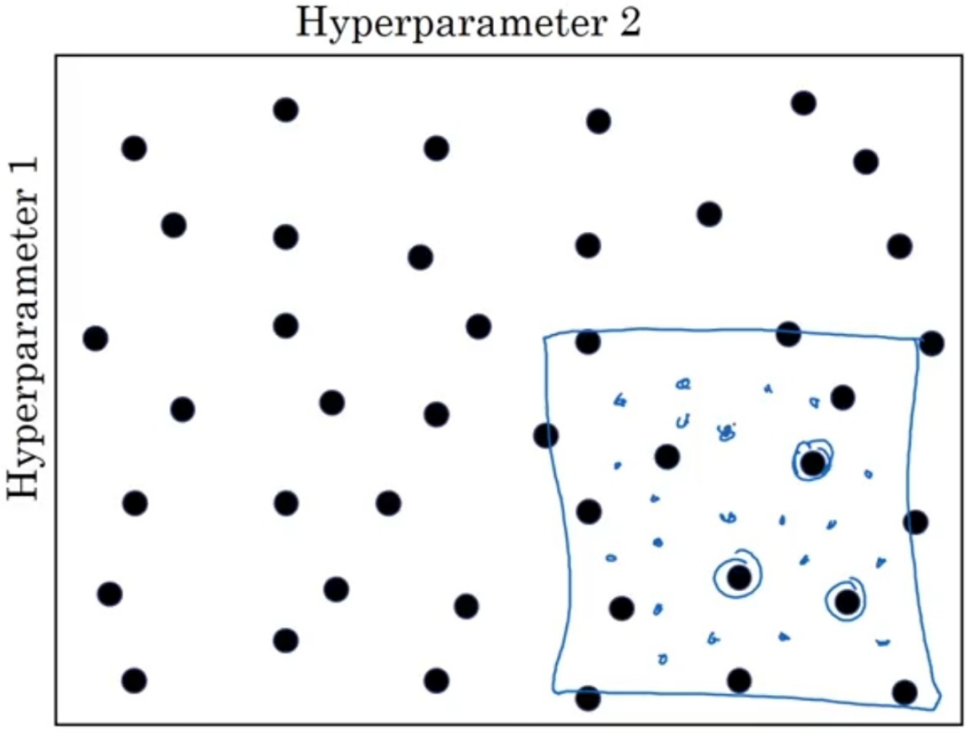
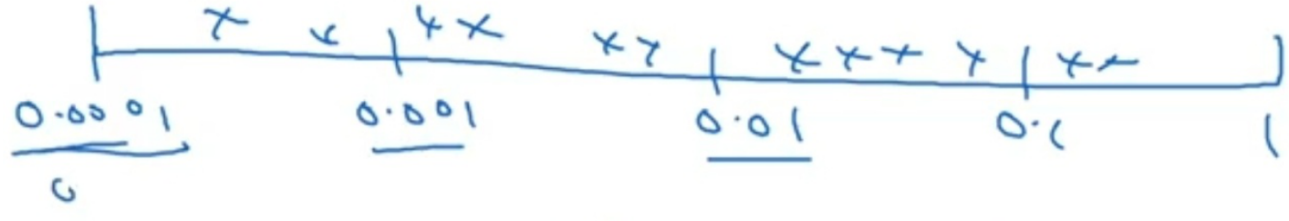
Don't just use a grid! Create random combinations, and then, look at the regions. Repeat until convergence.

Coarse to fine



Choose the right scale! Logarithmic is best.



$r = -4 * np.random.rand()$ ← $r \in [-4, 0]$

$\alpha = 10^r$ ← $10^{-4} \dots 10^0$

BATCH NORMALIZATION

Normalize neurons' values, before the activation function.

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

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

$$z^{(i)}_{\text{norm}} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{\text{norm}} + \beta$$

New z for $a(z)$

It makes $\mu = 0$ and $\sigma = 1$; No point on using b!

But, WHY does this work?

In a neural network, for each layer, its data changes every iterations, and with it the distribution.

Consequently, each layer suffers from "covariate shift".

Normalizing, we reduce the range of values, affecting to the distribution the minimum.

Batch Norm as regularization

- Each mini-batch is scaled by the mean/variance computed on just that mini-batch.
- This adds some noise to the values $z^{[l]}$ within that minibatch. So similar to dropout, it adds some noise to each hidden layer's activations.
- This has a slight regularization effect.

Batch norm processes your data one mini batch at a time, but the test time you might need to process the examples one at a time.