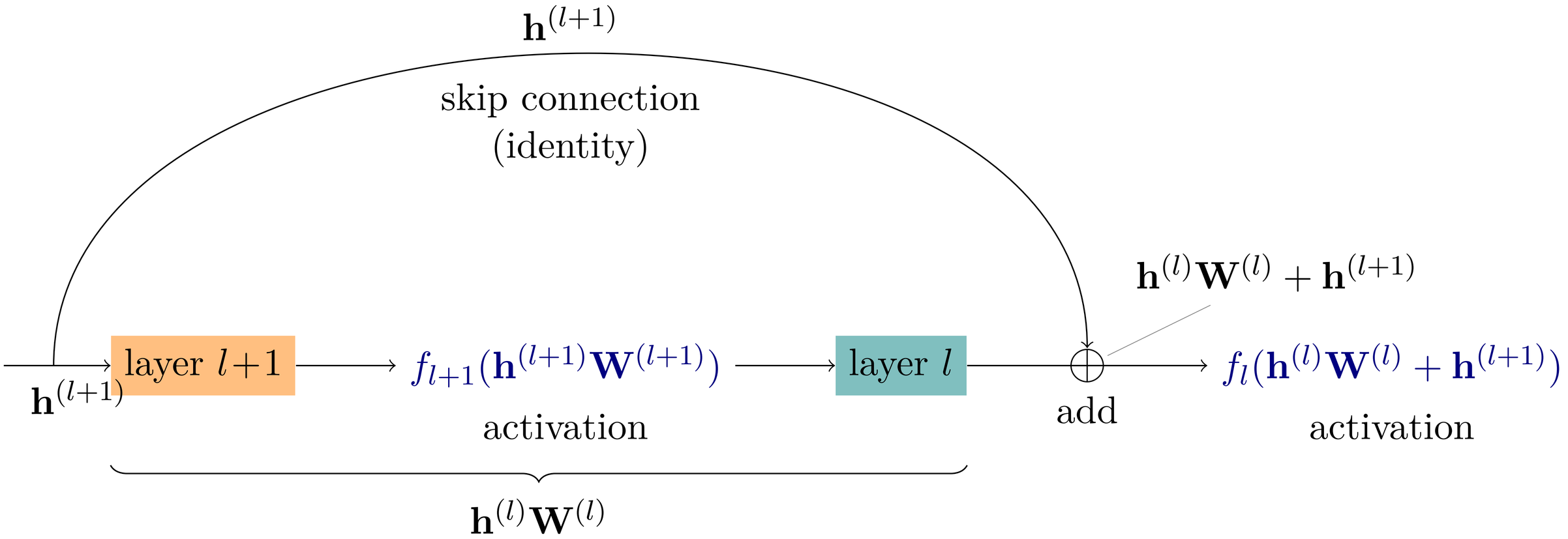
Chapter 10
Neural Network Architectures

CMPUT 467/567: Machine Learning II

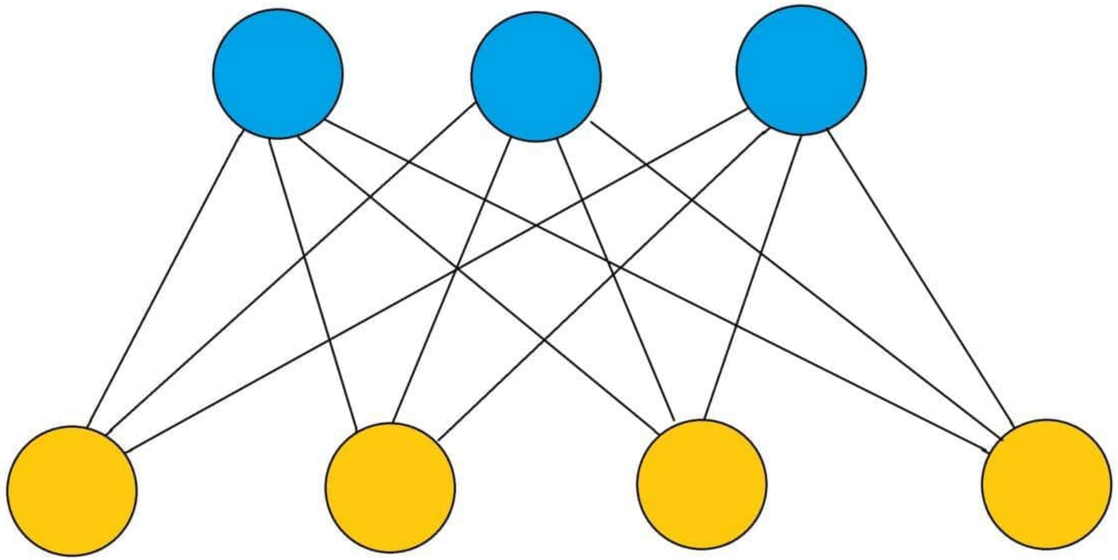
Winter 2026

Skip Connection

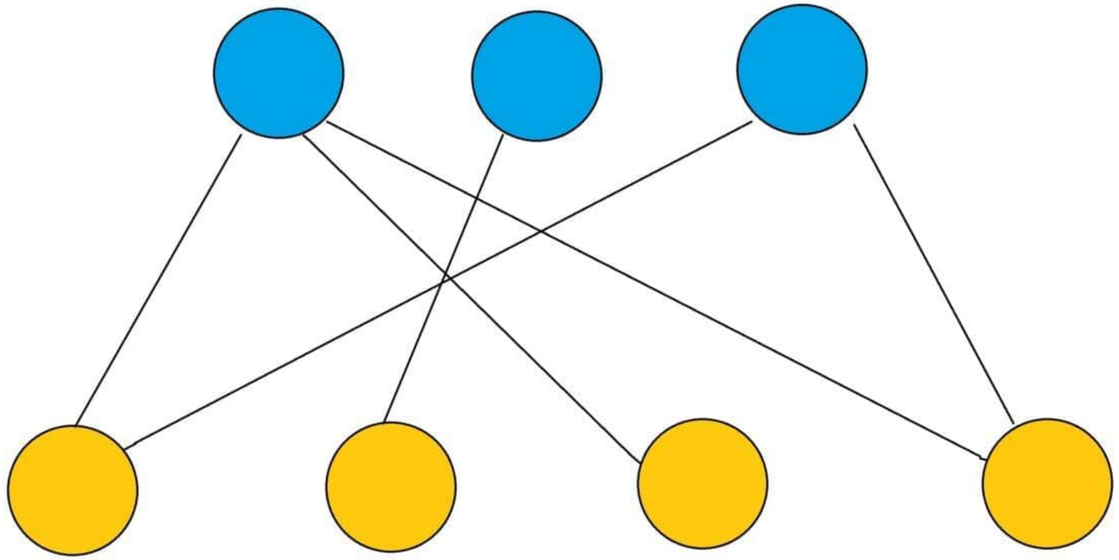


Sparse Neural Network

Densely Connected



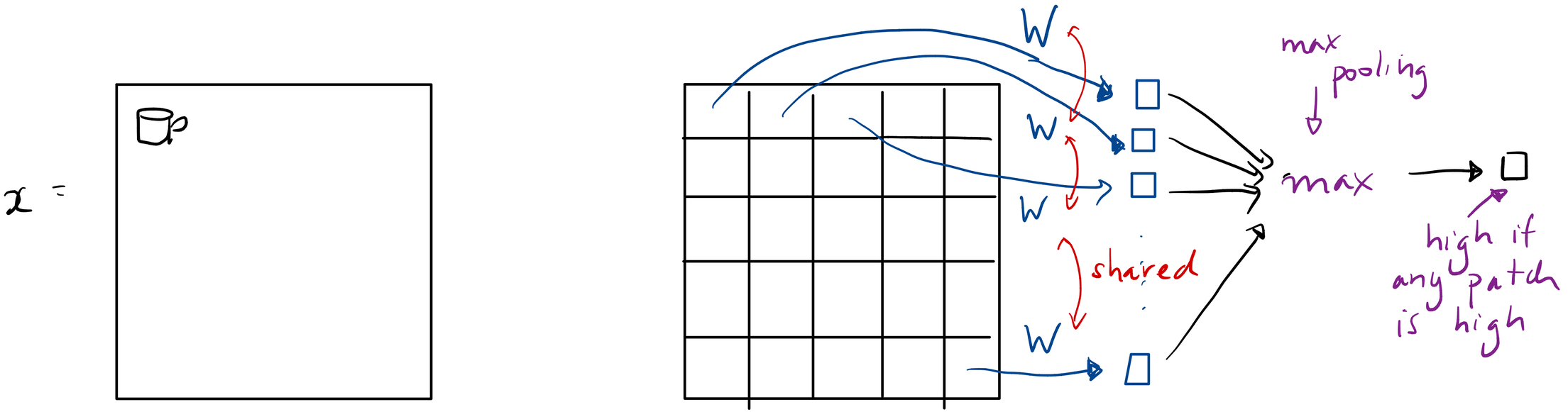
Sparsely Connected



Convolution

Input image

Process different patches in image

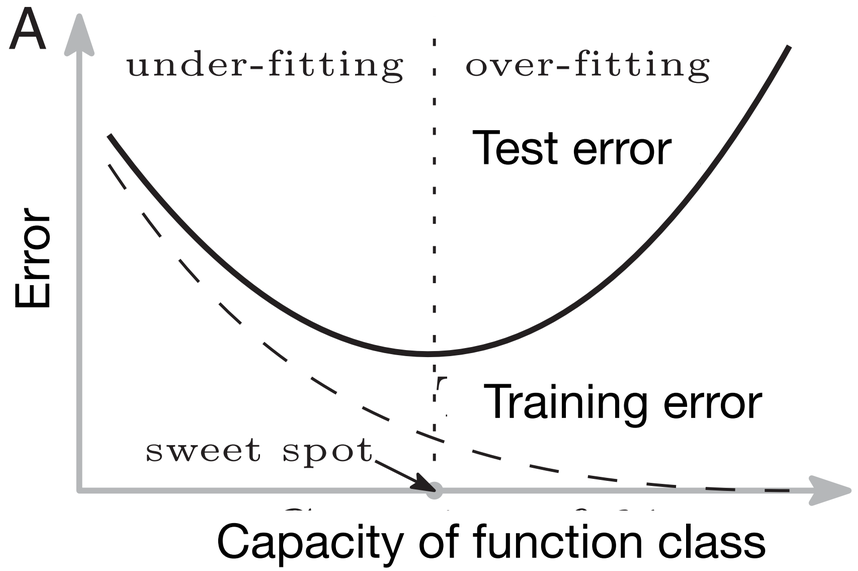


Nuance: often use ReLU right before max, so if all dot products are negative (not similar), then max returns 0 to indicate: not present

Overparameterized networks

- Recall p is the number of weights, n the number of samples
- When $p > n$, then we say the network is overparameterized
- When $p < n$, then we say the network is underparameterized (the typical setting we imagine)
- Why would we ever pick $p > n$?!

Typical Overfitting behavior

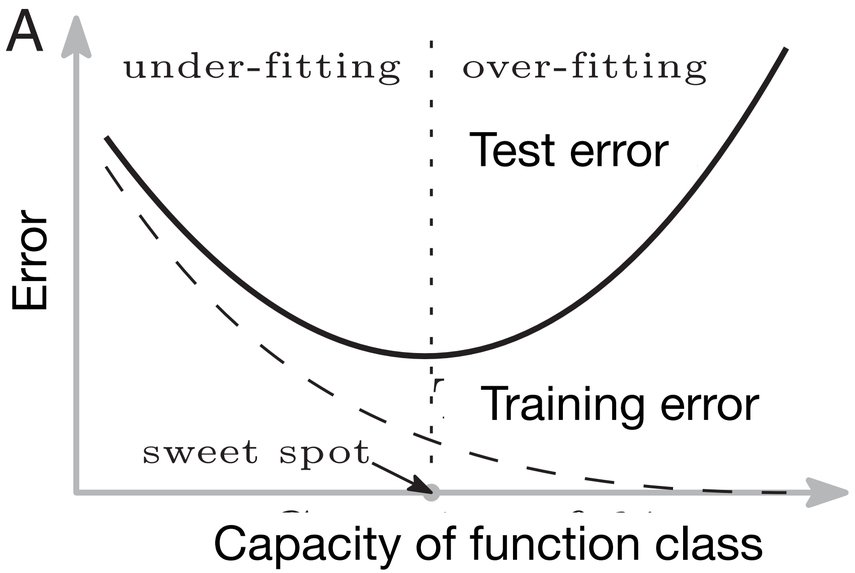


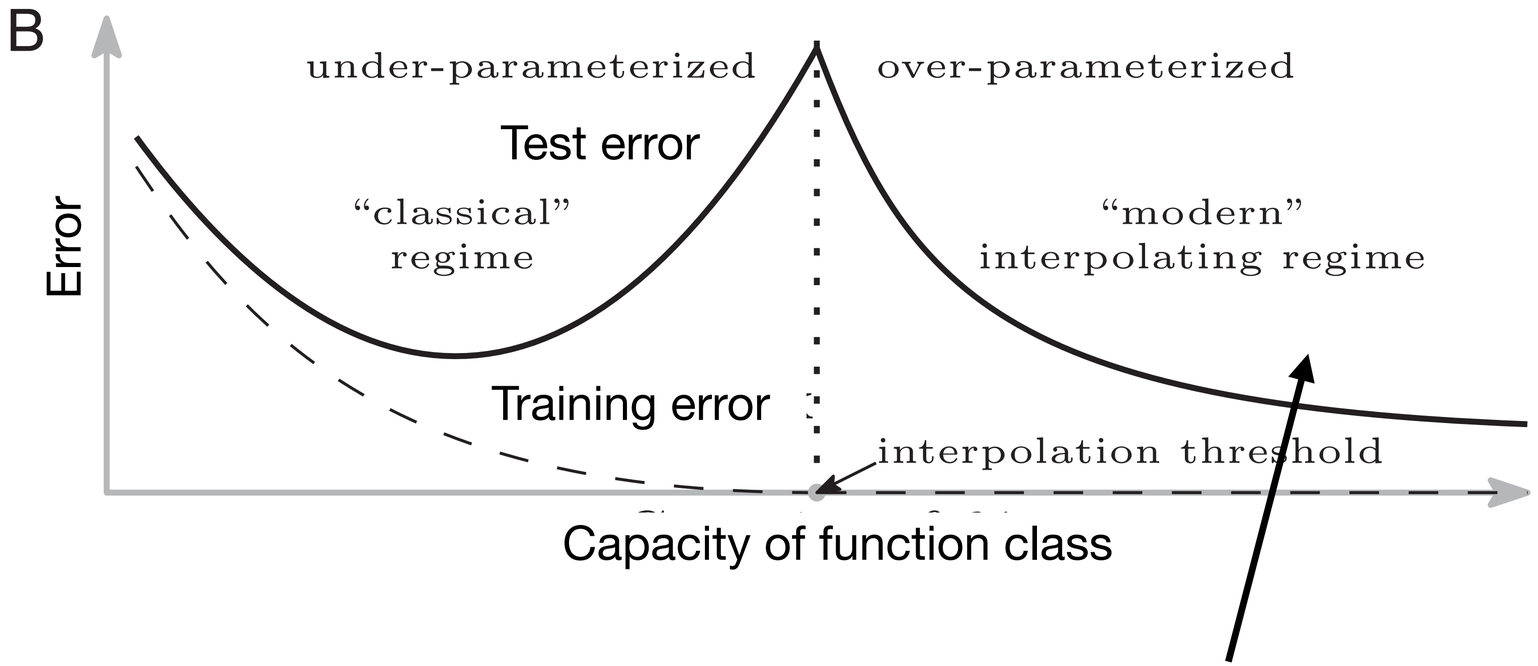
Training error decreases to zero

Testing error starts to increase as we overfit

Some interim (sweet spot) of function capacity (e.g., number of parameters) gives best test error

Double Descent





Classical regime

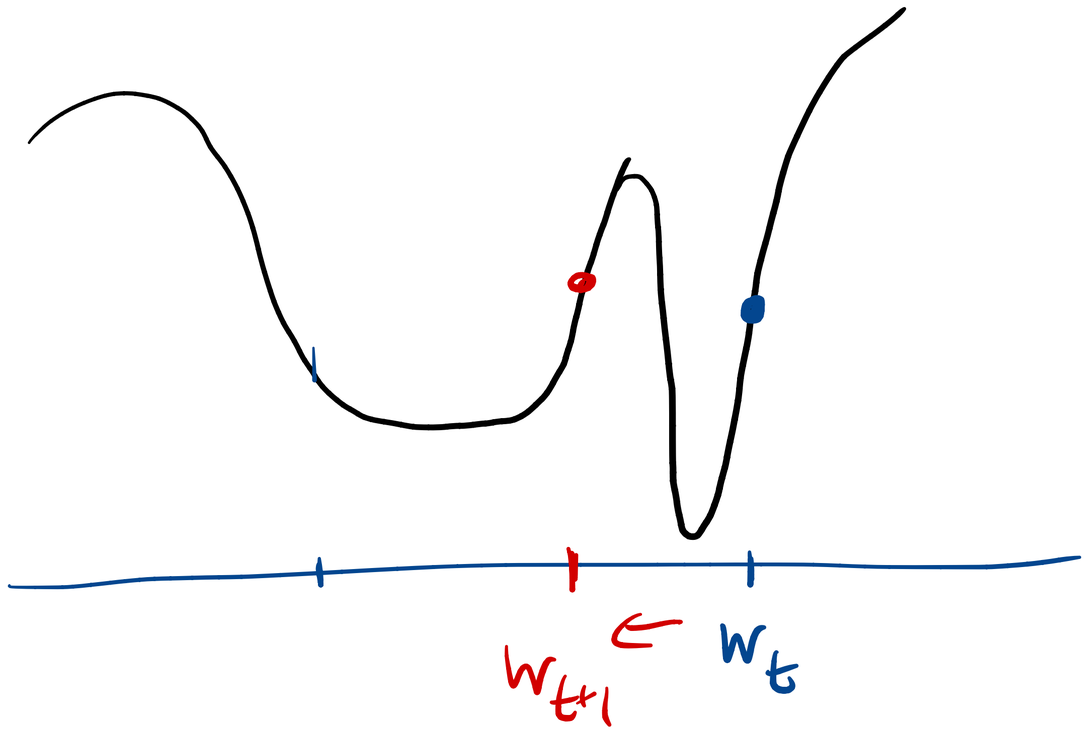
Assume optimizer picks simple solution amongst the space of solutions with zero training error

SGD picks simple solutions

- SGD implicitly regularizers to simpler solutions, because it prefers wider local minima rather than steep local minima

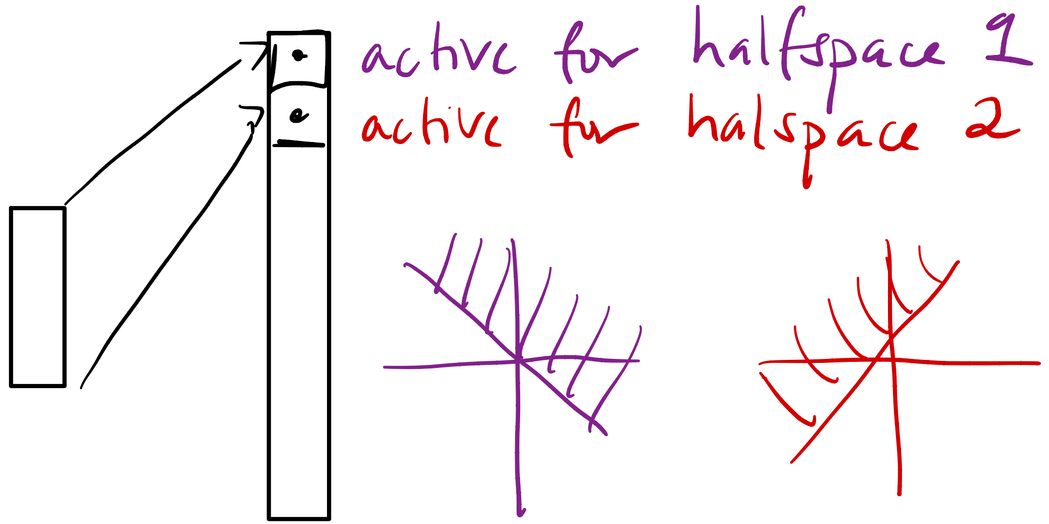
Sharp minima not stable under shifts to test (likely there due to noise, rather than signal)

Complexity lower in flat regions, since precision for weights is lower



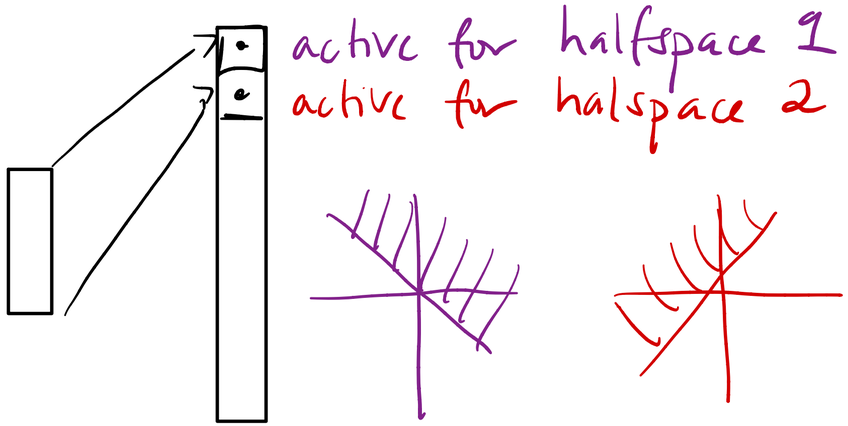
Depth allows us to And features

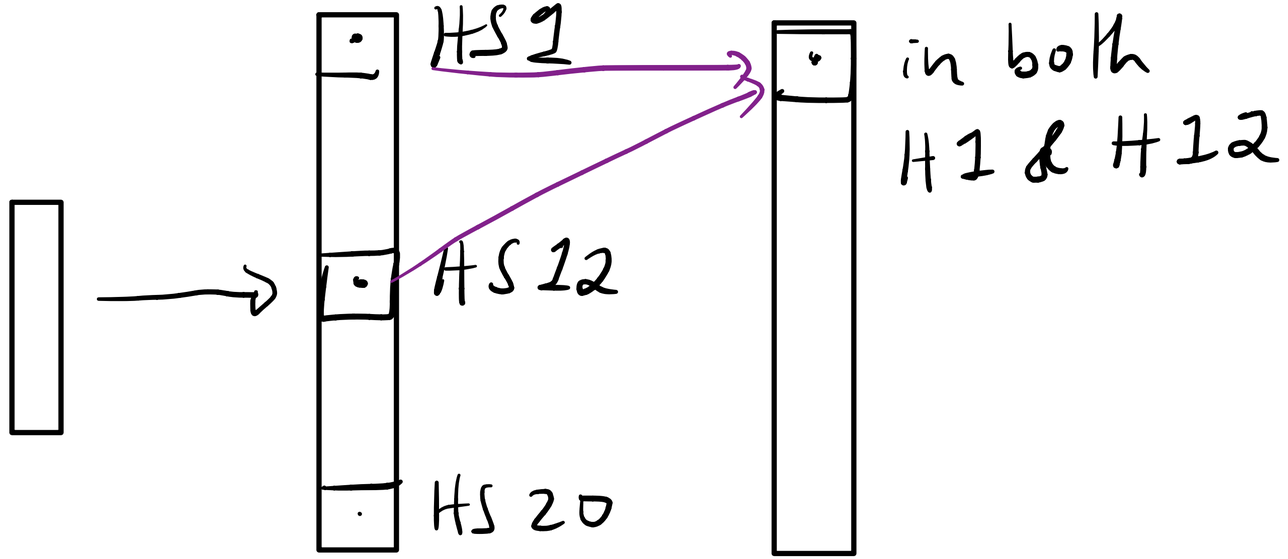
Recall ReLU creates halfspaces or partitions



Depth allows us to And features

Two layers





Overall choices

- Neural network is a directed acyclic graph, lot of choices we can make
- Can restrict ourselves with at least a few criteria
- Criteria 1: Sufficient complexity of the function class
- Criteria 2: Easy to optimize (e.g., differentiable, fast to compute)

Overall choices

- Criteria 1: Sufficient complexity of the function class
- Criteria 2: Easy to optimize (e.g., differentiable, fast to compute)
- Example: did not need to use $\sigma(\mathbf{h}\mathbf{W}_{:1})$ to get hidden node, could have used
  - RBF transformation (differentiable), $\exp(-\|\mathbf{h} - \mathbf{W}_{:1}\|_2^2)$
  - higher-order polynomial on $\mathbf{h}$: $w_1 h_1^2 + w_2 h_2^2 + w_3 h_1 h_2 + w_4 h_1 h_2 h_3 + \ldots$
  - Not clear either of these are a good idea, but can do backprop with them