CSDS 440: Machine Learning

Soumya Ray (he/him, sray@case.edu)

Olin 516

Office hours T, Th 11:15-11:45 or by appointment

Announcements

- Test 2 next week 10/17
  - Topics include everything up to and including today's lecture
  - Bring a calculator (phone app is ok)

How to scale an ANN?

Suppose we create an ANN with LOTS of layers.

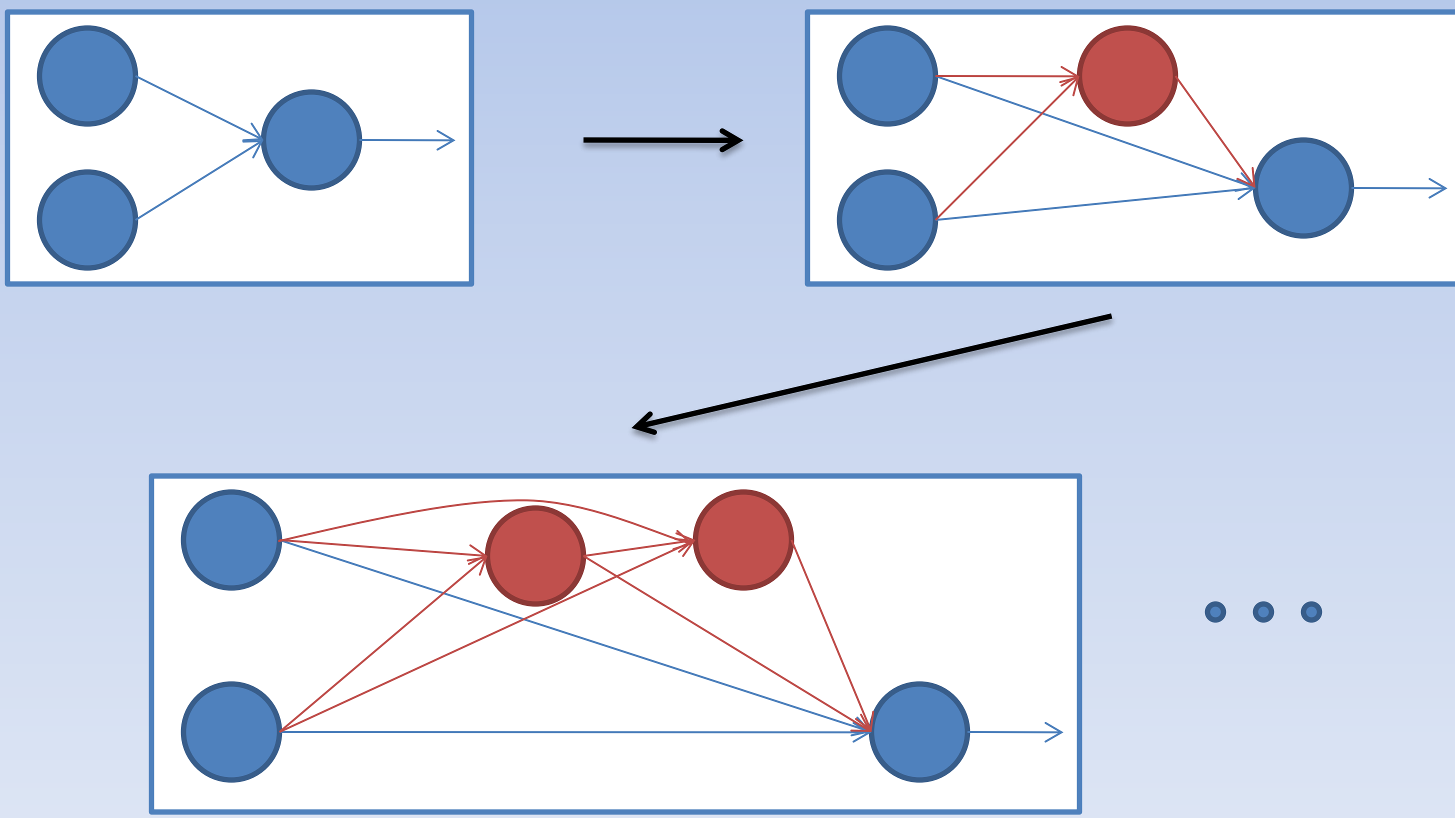
1. Why might we want to do that?
2. What will these layers *do*?
3. How can learning scale?
4. How to deal with vanishing gradients?
5. How to deal with overfitting?

Why might we need many layers? 1

- In theory, *two* layers are enough!
- But in practice, this would mean those layers would
  - Need to have a huge number of nodes
  - Have no structure to exploit
- Empirically, networks with more layers perform better

Why might we need many layers? 2

- Cascade Correlation (ask for paper)



Why might we need many layers? 3

- One way to interpret hidden units in ANNs is as "constructors" of a high-dimensional nonlinear (w.r.t. original attributes) space in which classification is possible with a perceptron

- Each layer then is an *abstraction*---a feature constructor that builds on previous features

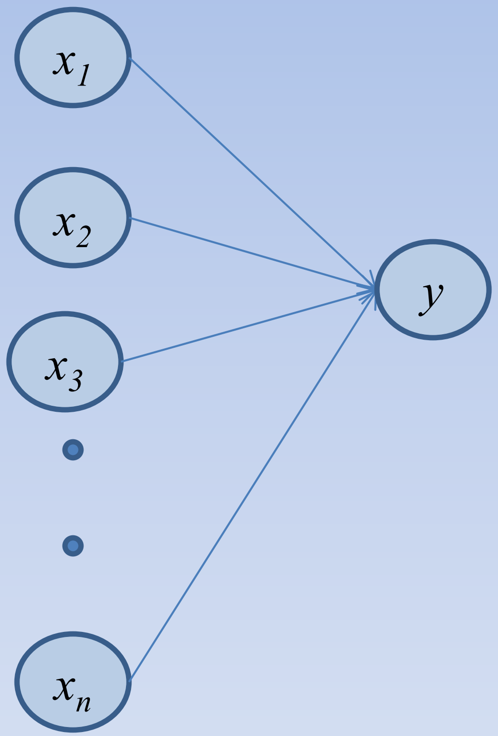
Interpretation of Hidden Units

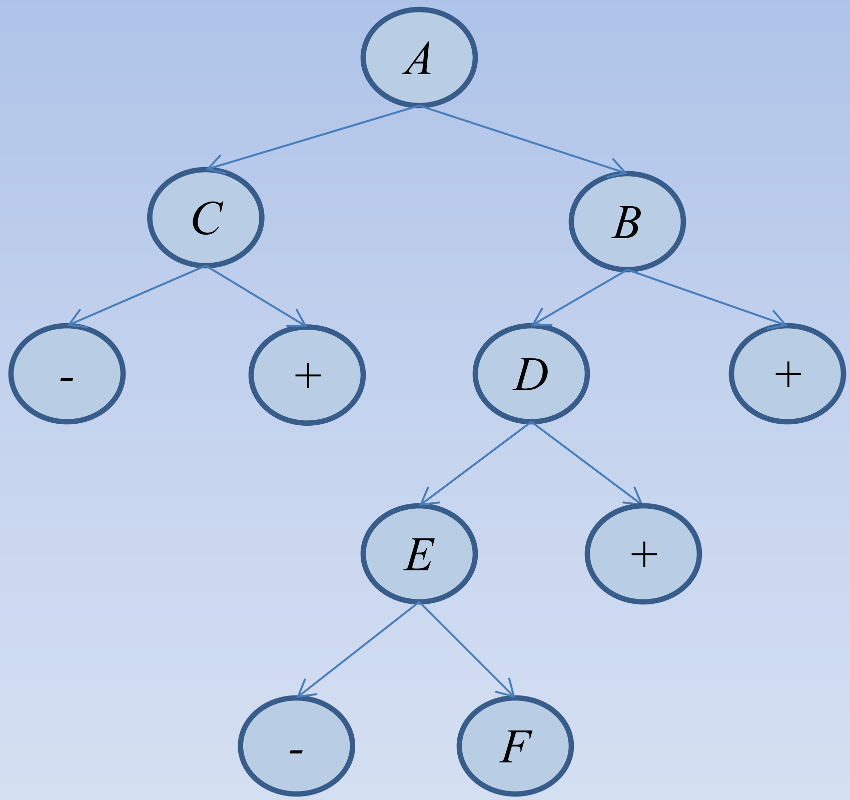
- Central idea in “Deep Learning”
  - Key idea: Try to learn good representations at different levels of abstraction
  - Larochelle, H., Erhan, D., Courville, A., Bergstra, J., Bengio, Y. An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation. International Conference on Machine Learning, 2007.

What should all these layers do? 1

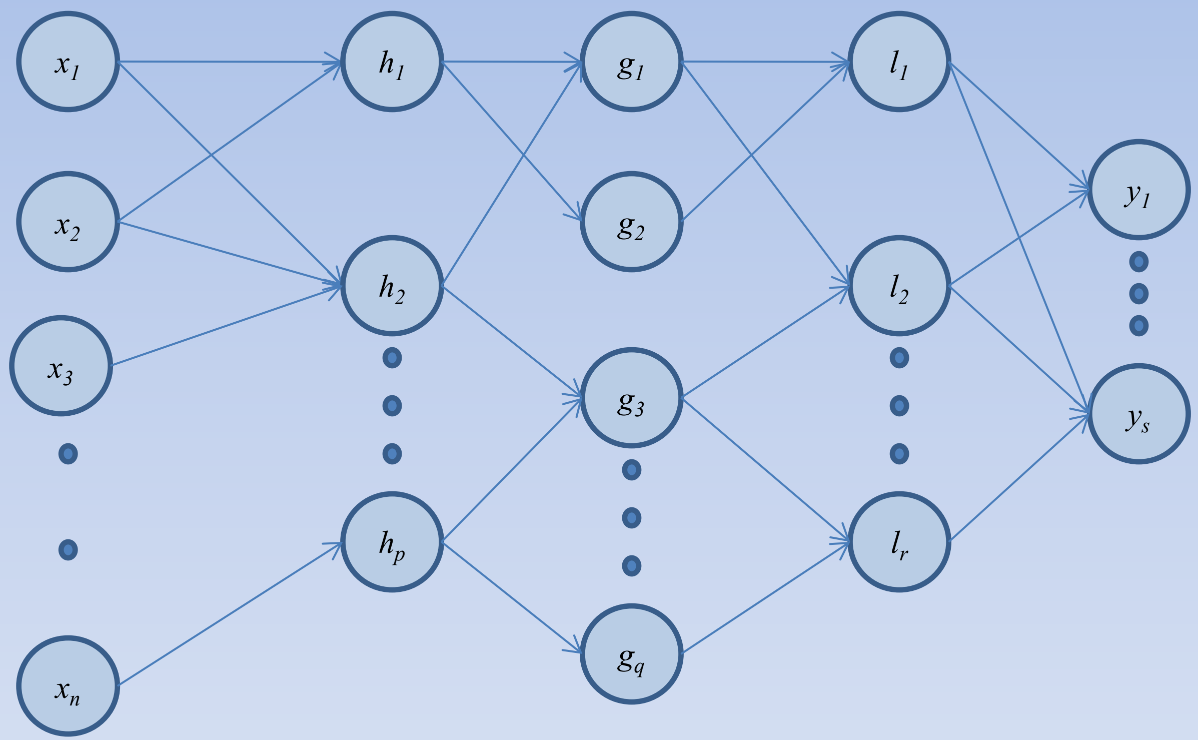
1. Allow *long computational paths*
   a) Key Idea: allow network to compute complex functions over input

Computational paths





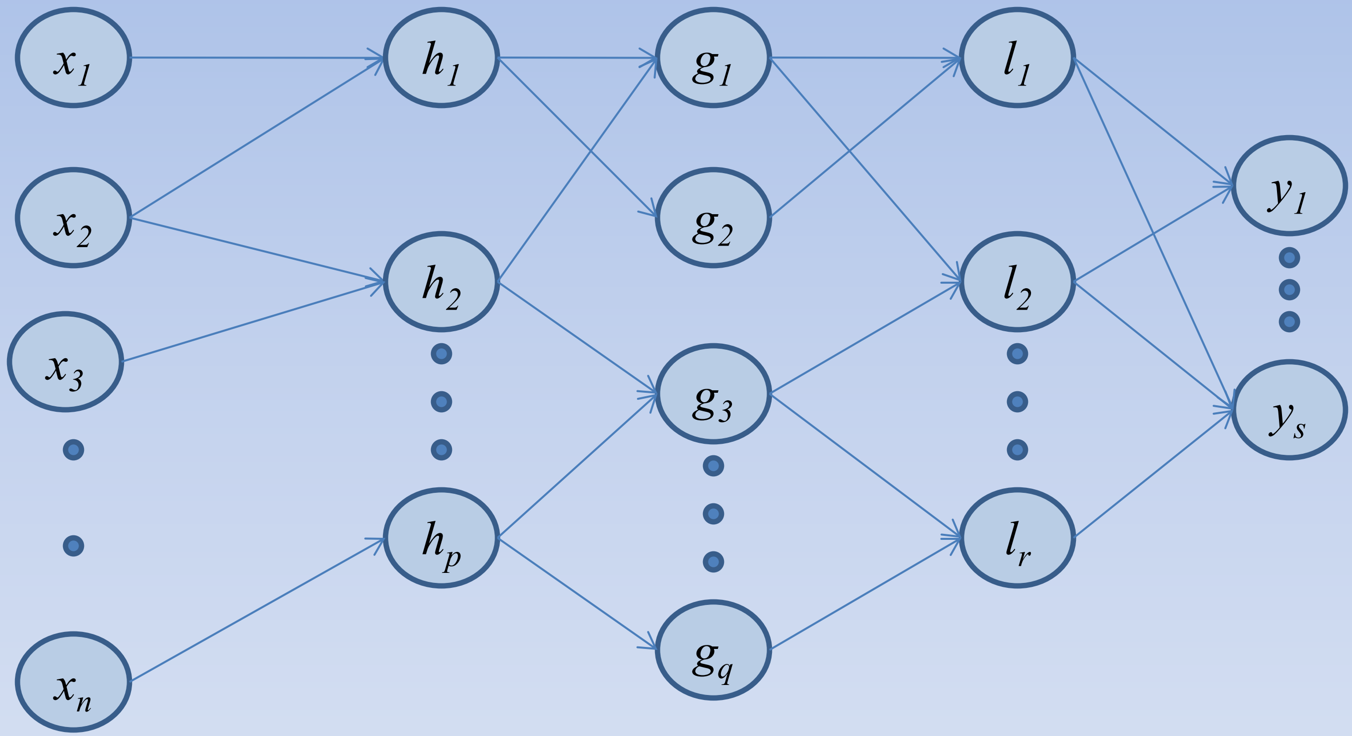
Computational paths



What should all these layers do? 2

2. *Aggregate information* across many different parts of the input
   a) Key idea: allow different parts of the input to interact with each other

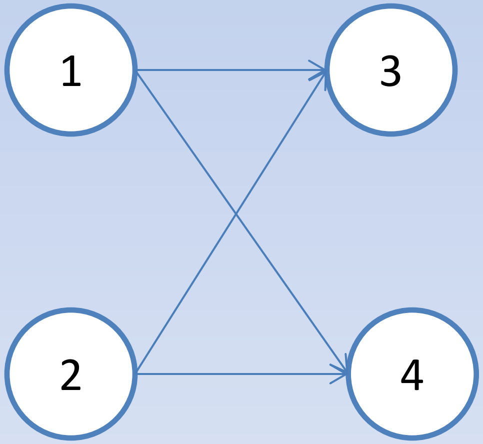
The “Receptive Field” analogy



As we go deeper, a node is aggregating information across more of the input.

Scaling the computation

- A network is a *computation graph*
- We can view each *layer* as a matrix/vector operation on the previous layer



$$\begin{pmatrix} w_{13} & w_{23} \\ w_{14} & w_{24} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \mathbf{W}^l \mathbf{z}^l$$

$$\mathbf{z}^{l+1} = h(\mathbf{W}^l \mathbf{z}^l)$$

$$\mathbf{z}^{l+2} = h(\mathbf{W}^{l+1} \mathbf{z}^{l+1})$$

Backprop as matrix computation

- Since the forward computation is layer-wise, the gradients can be expressed using vectors and matrices too

$$\hat{y} = h(\mathbf{wz})$$

$$L(\mathbf{w}) = \frac{1}{2}\left(y - \hat{y}\right)^2$$

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \mathbf{w}} = (\hat{y} - y)\frac{\partial h}{\partial (\mathbf{wz})}\frac{\partial (\mathbf{wz})}{\partial \mathbf{w}}$$

$$= (\hat{y} - y)\hat{y}(1 - \hat{y})\mathbf{z}$$

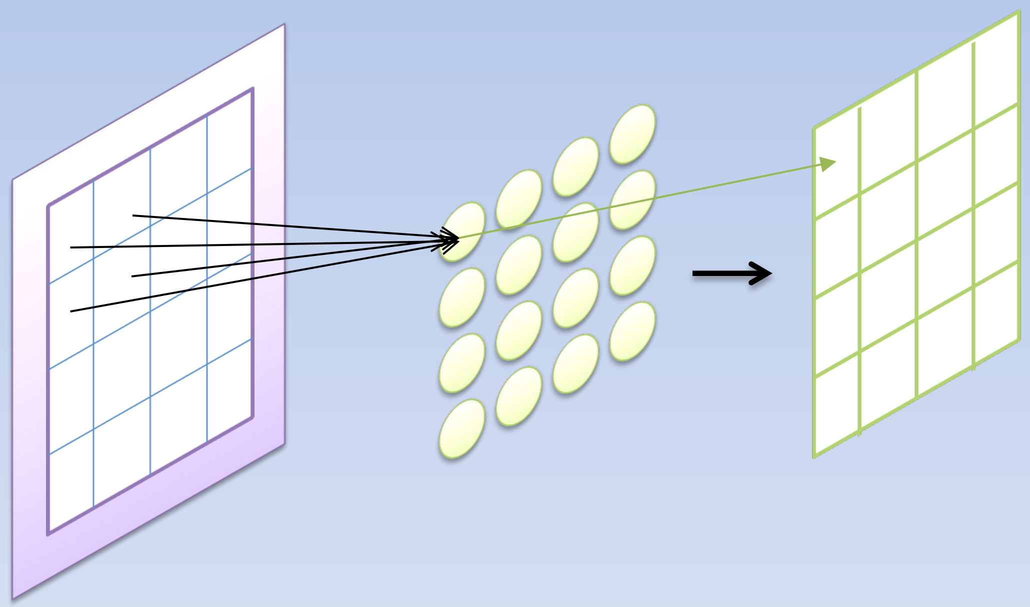
A scaling problem

- As the network grows, the number of parameters can scale quadratically with layer size
- Suppose the input is a complex object like a 256x256 image and each pixel is an input node
  - If there are an equal number of hidden units, there would be $(256)^4$=4e9 weights *per layer*

Locality and Invariance

- How to build an architecture that scales?
  - Let each hidden unit only look at *a local part* of the input **(Locality)**
  - Let different hidden units compute *the same feature* for different local regions **(Invariance)**

Example

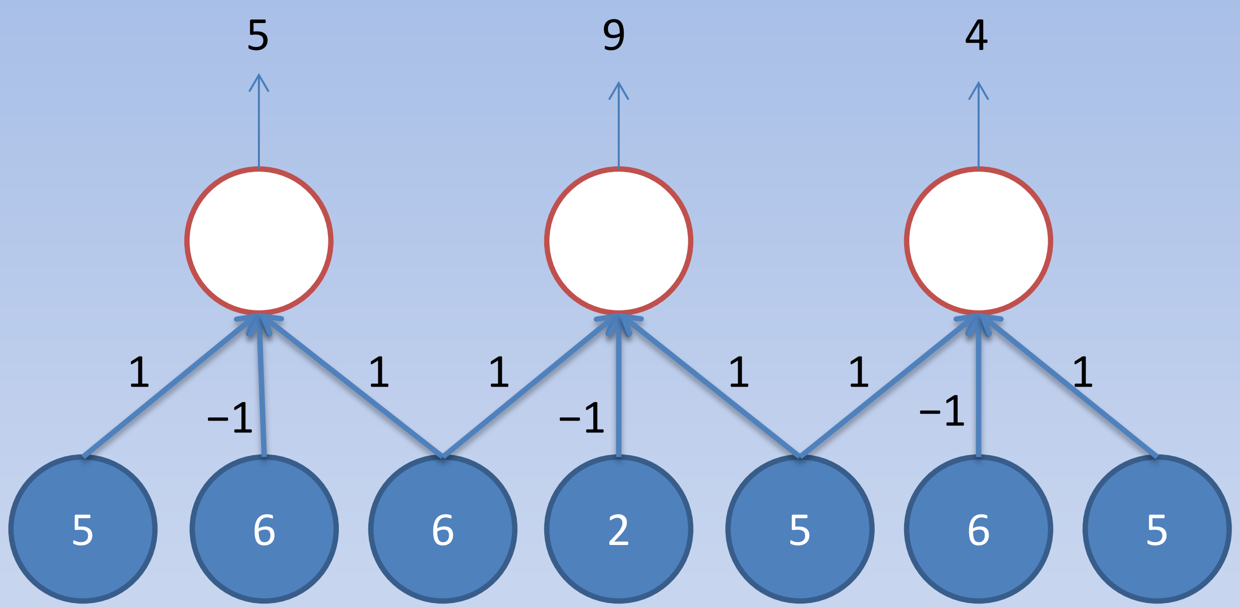


Convolutional Neural Networks

- Introduce a **kernel** k : a set of weights *replicated* across multiple local regions
  - Generally multiple such kernels will be used
  - Each kernel computes one local feature

- The operation of applying the kernel to the input is called **convolution**

Convolution



k=[1,-1,1], **size** l=3, **stride** s=2

Convolution

$$\mathbf{z} = \mathbf{x} * \mathbf{k}$$

$$z_i = \sum_{j=1}^{l} k_j x_{i+j-(l+1)/2}$$

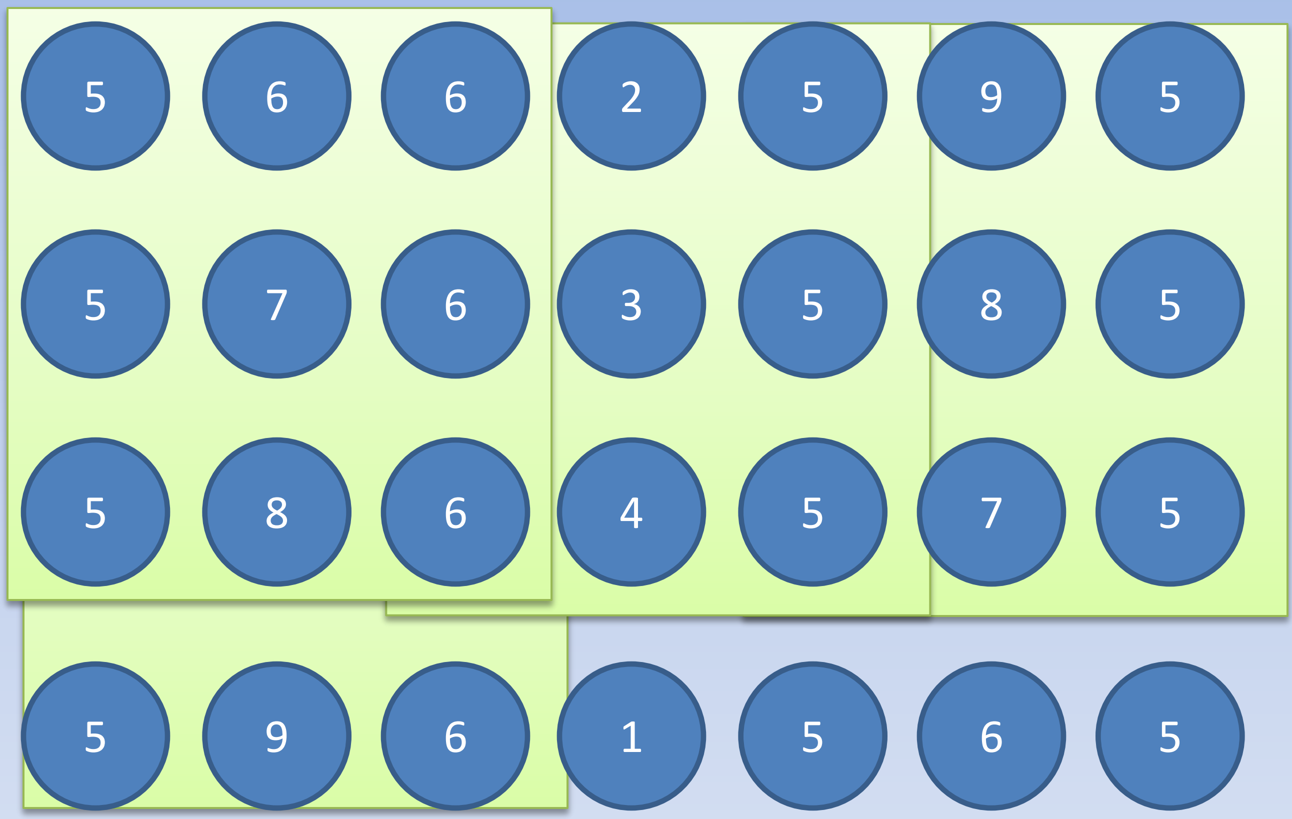
- Convolution is a linear operation
  - Can also be represented as a matrix operation
- The output $\mathbf{z}$ will have roughly n/s entries

Convolutional NNs

- A kernel detects a specific feature, but what kernels to use?

- In a CNN, the kernels (detectors/filters) *themselves can be learned*
  - Parameterize as a set of weights, and learn via backpropagation

2D Convolution



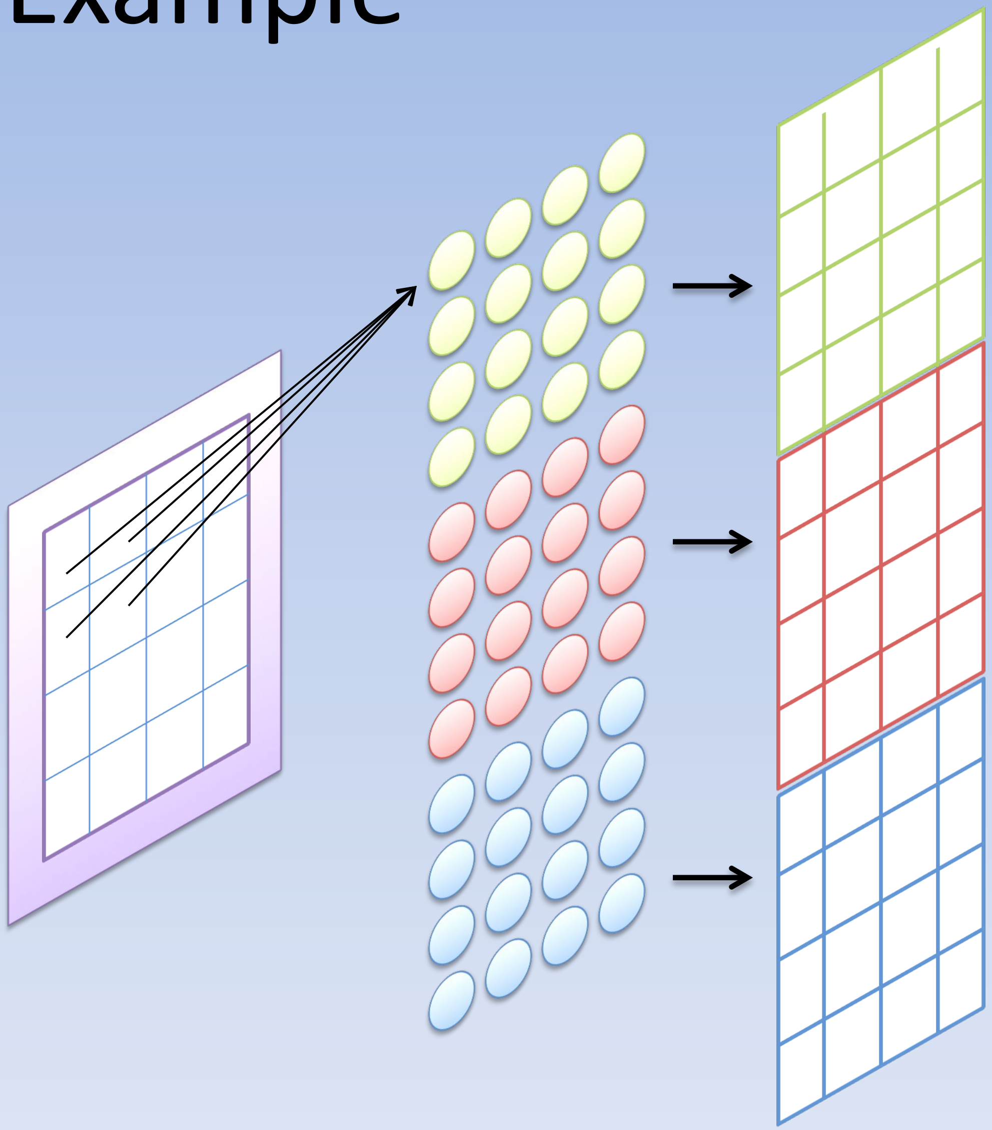
k=[w_1 w_2 w_3;w_4 w_5 w_6;w_7 w_8 w_9]; stride=(2,1)

Tensors

- Each convolution kernel creates one “feature detector” that is looking for a specific property in a local patch

- Typically, will have many such kernels, each looking for a different feature

Example

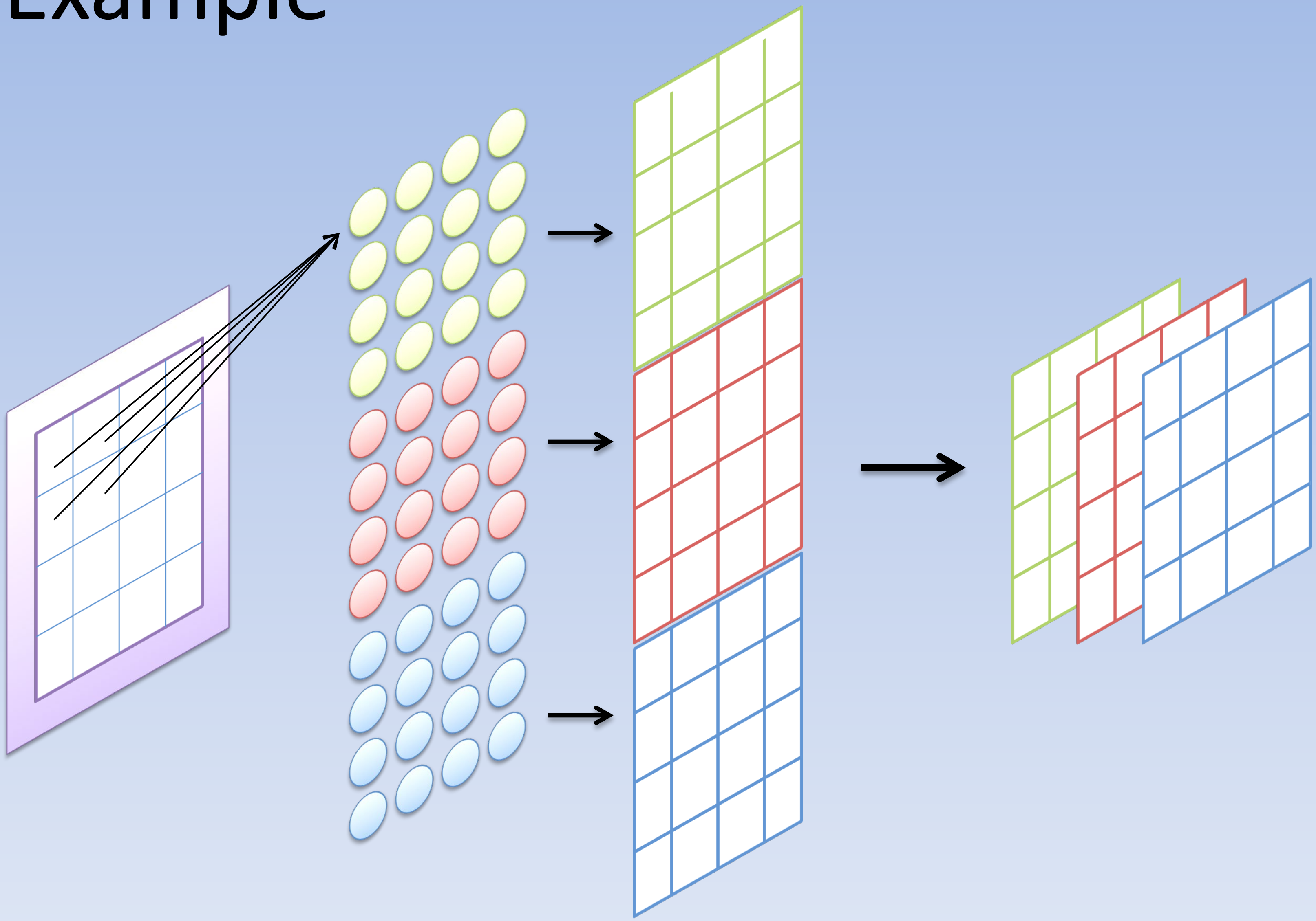


Tensors

- In order to maintain locality and invariance, instead of concatenating these kernels, we stack them along a new dimension

- If the input has two dimensions or more (e.g. images, video) then this results in a multidimensional matrix at each layer
  - These are tensors

Example



Example

- Suppose we have a batch of 32x32 images
  - Each input has dimensions (32, 32)
- Suppose we apply 10 2x2 kernels to each image with stride 1x1 (with padding)
- The output will be a tensor with dimensions (10, 32, 32)