CSDS 440: Machine Learning

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

Olin 516

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

Zoom Link

Announcements

- Quiz 2 Thursday
  - Same format as Q1
  - Topics: Everything up to and including Lecture 8 (Calculus)
  - Bring a scientific calculator (or phone app)

Recap

- Having very few layers is problematic because they will need many n_____ and not have s_______.
- A further motivation is from C____ c_____. This creates a deep architecture by successively learning p____, each modeling the r____ of the previous.
- One way to interpret hidden units is as f______ c_________ for an h___ d____ space where classification can be done with a p_____.
- To generate good features, we should allow l_____ c_____ p_____.
- We should also a______ i______ across different parts of the input.
- Neural Networks can be viewed as c___ g___. Each layer of a network performs a m____ operation on the input v____.
- Fully connected layers can be problematic because the number of weights g_____ q_____.
- We should let each hidden unit only look at a l_____ part of the input.
- If the weights connecting a region to hidden units are the same across hidden units, this is called i_____.
- In a c_____ neural network, we introduce a k_____. This is a set of weights that are r_____ across multiple l_____ regions.

Today

- Artificial Neural Networks (Ch 4, Mitchell)
- Probabilistic Machine Learning

How to scale an ANN?

Suppose we create an ANN with LOTS of layers.

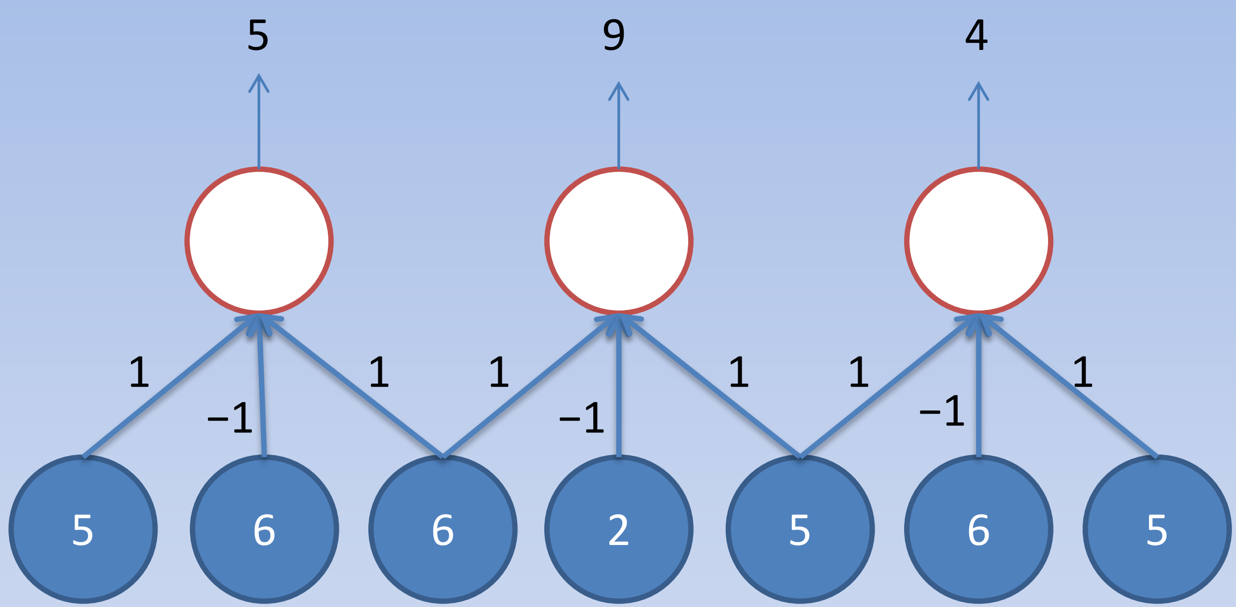
1. ~~Why might we want to do that?~~
2. What will these layers *do*?
3. How can learning scale?
4. How to deal with vanishing gradients?
5. How to deal with overfitting?

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

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

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

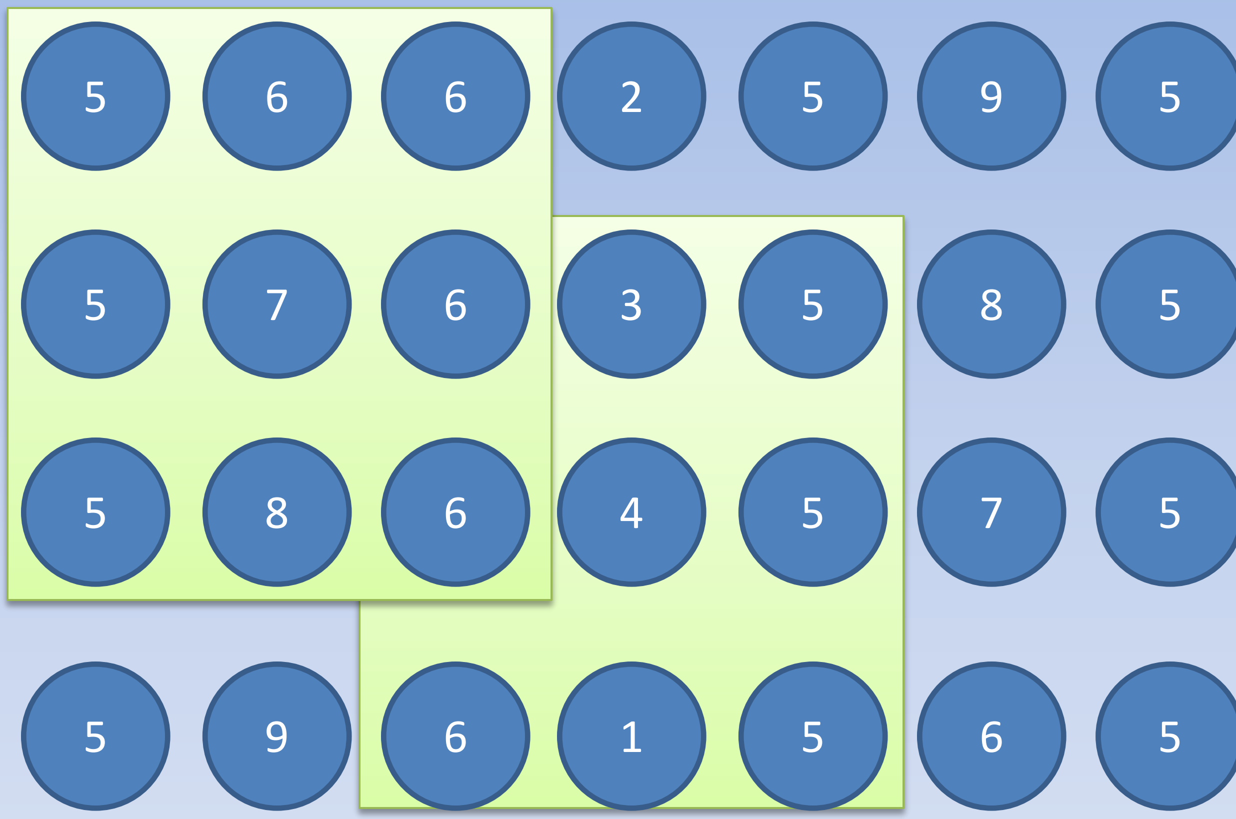
- Convolution is a linear operation
  - Can also be represented as a matrix operation
- The output $\mathbf{z}$ will have roughly n/s entries

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



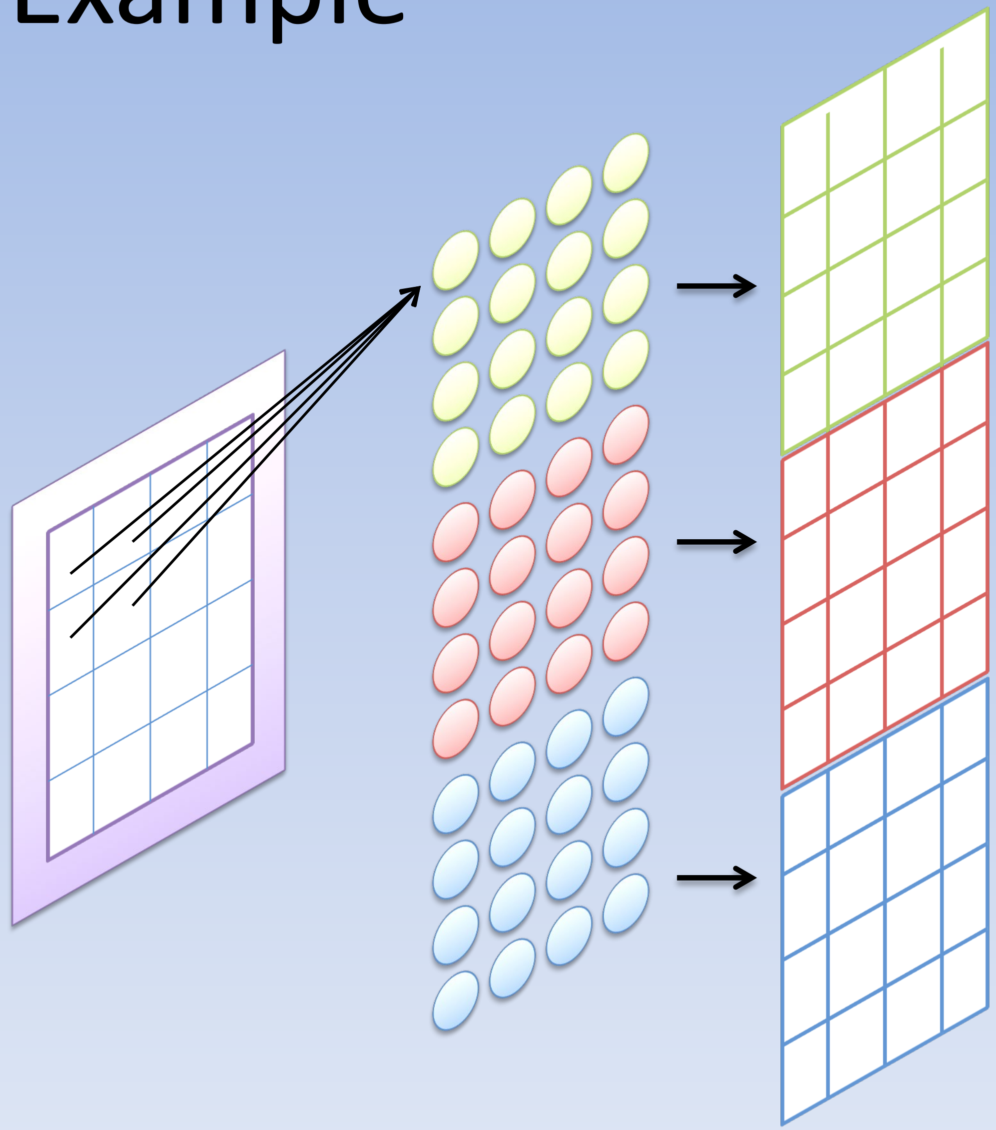
k=[$w_1\ w_2\ w_3;w_4\ w_5\ w_6;w_7\ w_8\ w_9$]; stride=(2,1)

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

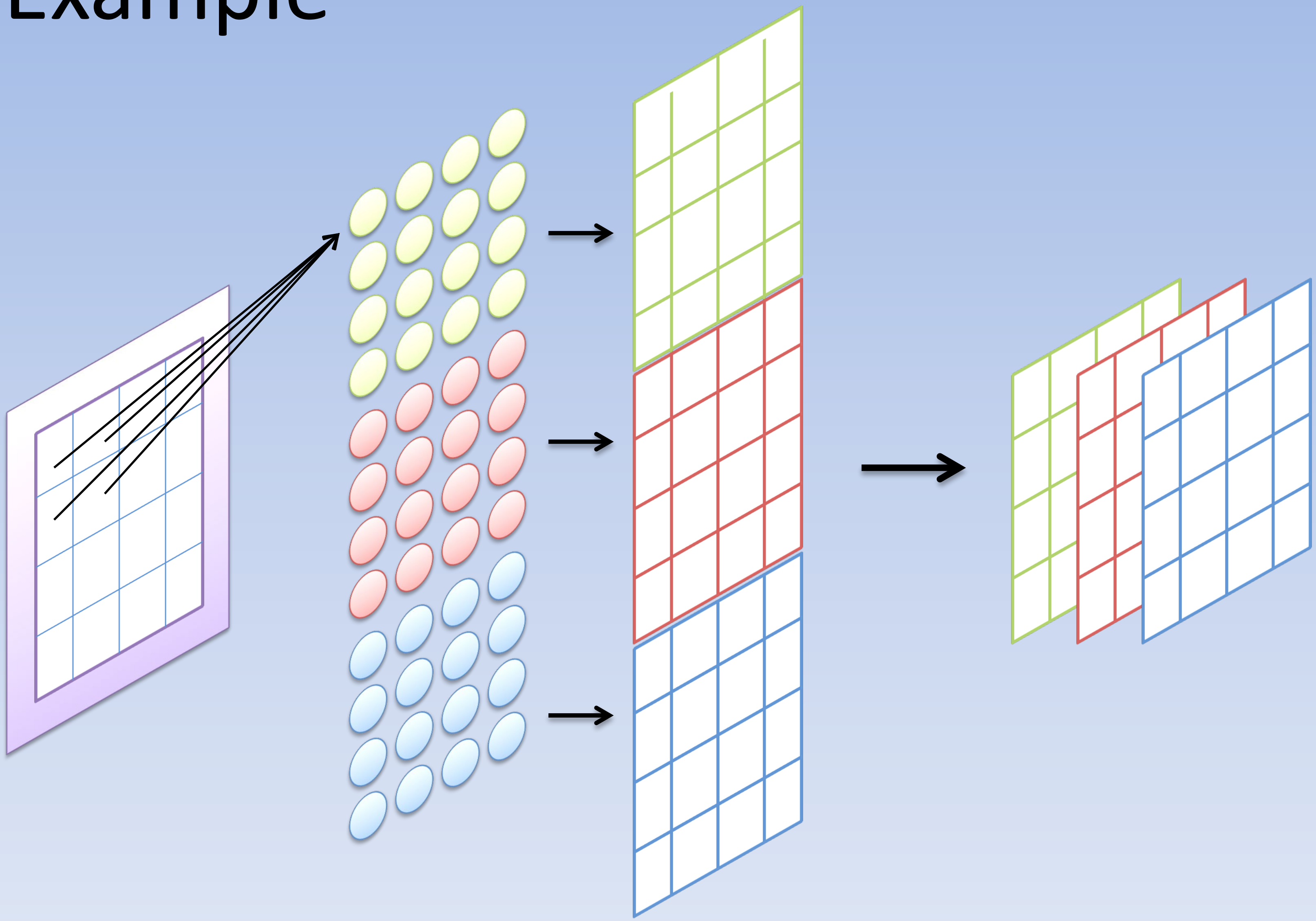


Tensors

- 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 create "tensors"

Example



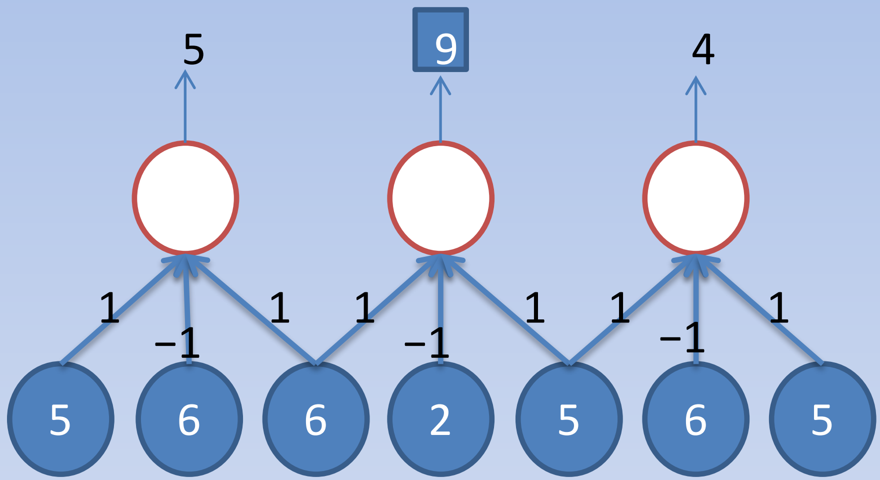
Example

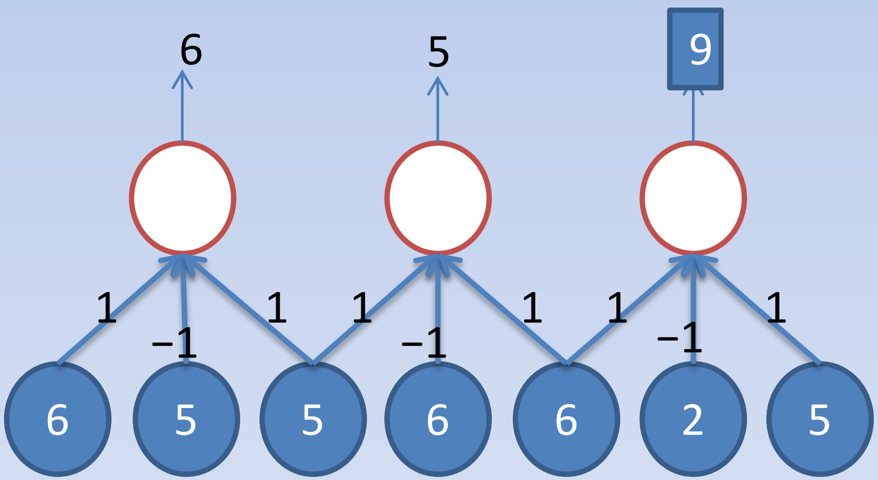
- Suppose we have a batch of 32x32 images
  - Each input has dimensions (32, 32)
- Suppose we apply 10 4x4 kernels to each image with stride 1x1
- The output will be a tensor with dimensions (10, 32, 32)

Pooling Layers

- A *pooling* layer aggregates information from an adjacent layer
- Average pooling: $k=(1/l,\ 1/l, \ldots 1/l)$
- Max pooling: computes the maximum value of l inputs
  - For each feature detector, identifies whether that feature was found somewhere in the previous layer
- Downsamples input by factor of l

Max Pooling





Vanishing Gradients 1

- A key problem in ANNs is *vanishing gradients*
- To prevent vanishing gradients, we can use the "Rectified Linear Unit" (ReLU) activation function:

$$h(x) = \max(0, x)$$

Vanishing Gradients 2

- Each layer in an ANN *learns a completely new representation* from the previous layer
  – Can cause catastrophic failure due to one "bad" layer

- Instead, each layer can *add on to* the learned representation of the previous layer
  – Allows building much deeper structures robustly

Residual Networks

- Perturbing the representation is done through adding a “residual” function to each layer
- Replace

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

- With

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

Residual function (learned from data; could be identity)

How do we prevent overfitting?

- ANNs are very prone to overfitting
  - Structure can be very complex, lots of parameters
  - Decision surface can be very nonlinear

Controlling Overfitting

- One strategy: add a "weight decay" term

$$L_{OC}(\mathbf{w}) = L(\mathbf{w}) + \boxed{\gamma \sum_i \sum_j w_{ji}^2}$$

Weight Decay
Term
"Complexity
Penalty"

Tradeoff Hyperparameter

- This will prevent weights from growing too large