Deep Learning:

- Dominating success of machine learning
- Extremely successful adaptation of neural networks

- Applications
  - image classification
  - video prediction
  - speech recognition

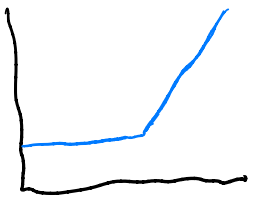
Deep Learning: deep networks

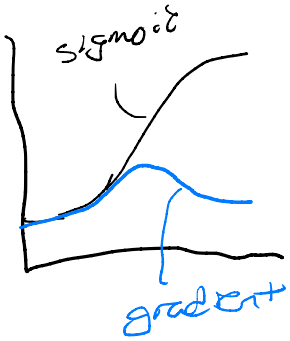
- NN with large number of hidden layers
- learn "features" of increasing levels of abstraction
  - each layer learns "features" that next layer finds useful
  - overall, network learns layers that final classification/output layer finds useful

  ↳ further layers go to more abstract representations

- more layers but less in each ends up with more powerful, generalizable representation
  ↳ typically end up with fewer parameters, better performance
  ↳ better to many compositions of simpler functions

- Concerns / Issues:
  - vanishing gradients (backpropagation through multiple layers)





    ReLU (encourages sparsity)
  - long training time

    use GPU parallelism
    ↳ great task for GPUs

    turn days/weeks into minutes/hours
  - lots to learn → need lots of data to train well

    ↳ appropriate for big datasets

    Example: 1000 images ✗ likely too small

    100,000 images ✓ better

  - overfitting

    dropout

    ensure no unit relies too heavily on output of another

    at each step, randomly set some outputs to 0 (or do for inputs), update weights, take step

Concerns (cont.)

- big/messy/difficult to code
  - think in terms of tensors / tensor manipulation
  - use software, don't build from scratch

Software

PyTorch - developed by Facebook research
- newer, but increasingly popular

TensorFlow - developed at Google
- often use Keras on top

Keras - high-level NN API
- makes it easy to build models / add layers
- use on top of other frameworks

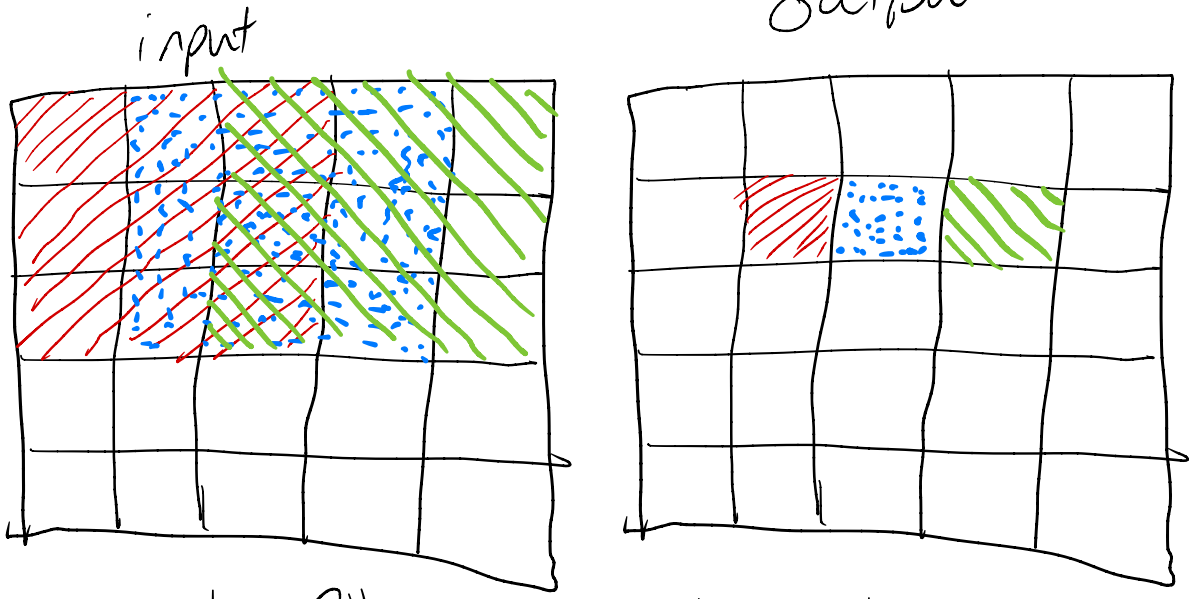
CNN: Convolutional Neural Networ

Idea:

- capitalize on spatial data in DNN
- do so through special layers

Convolutional Layer

- apply convolution



convolution = apply filter to small portion (weighted sum) to compute one output value

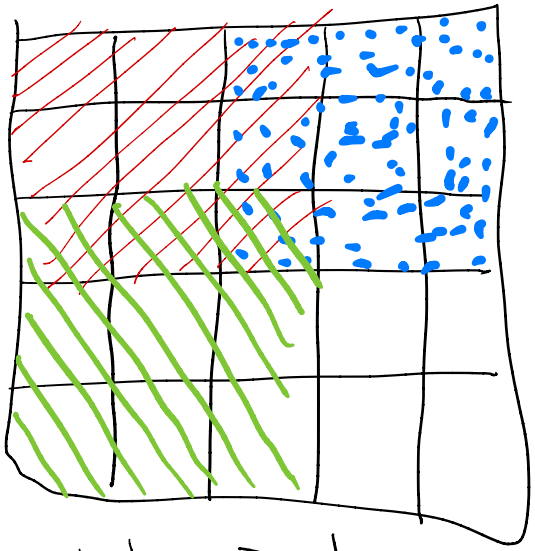
Convolutional Layer (cont.)

filter is like a window

slide filter across data

stride = how much you slide by

↳ 1 in above image



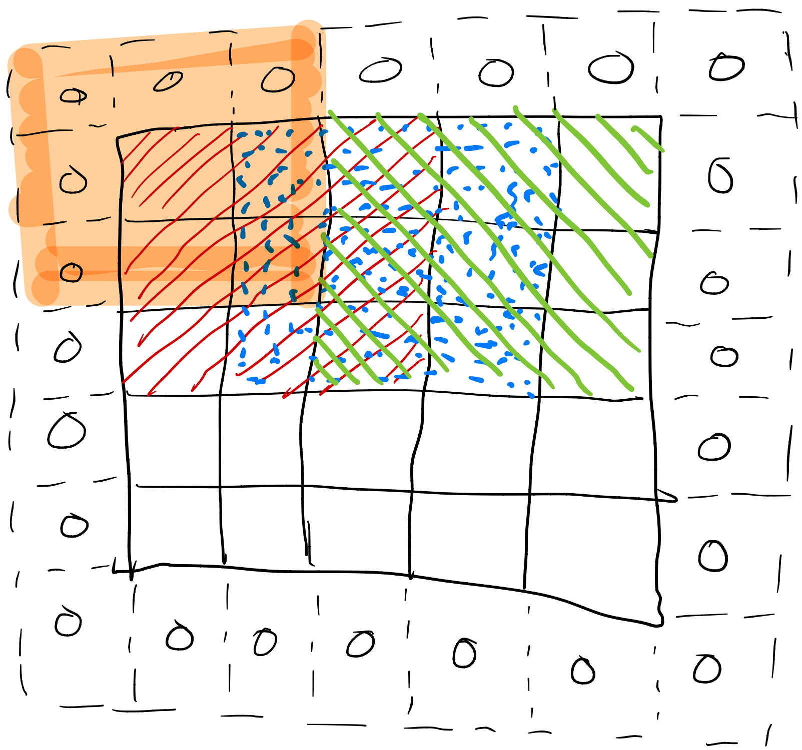
↳ slide > 1 → output from convolutional layer is smaller

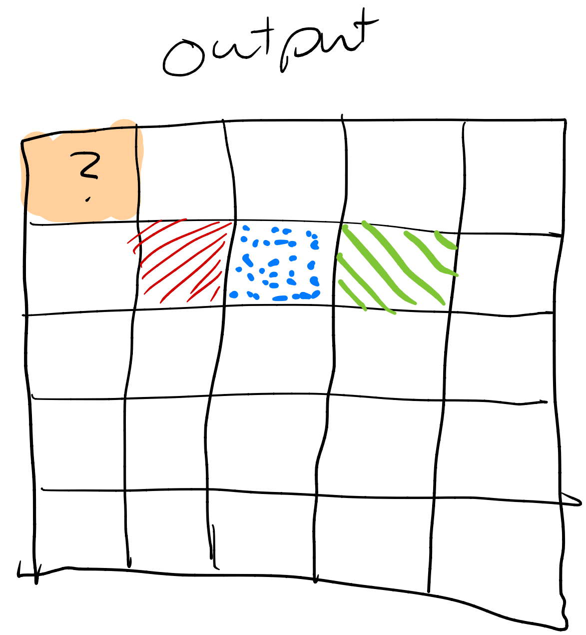
typically use odd sized window (so there's a center)

Convolutional Layer (cont.)

Padding — what about edge rows/corners

assume some "ghost" value outside (often 0)





- not dense layers
  - units connected to subset of previous layer
    - the window

- learns filters that activate for specific features (like edges)
- provides translation invariance

Pooling Layer

- another way (other than stride) to decrease size
- windowed pooling
  - ↳ go from big to smaller by taking max or average of all in window
  - ↳ like downsampling with rule for which one to take rather than just skipping

  Example: 2x2 pooling filter with stride of 2 downsamples by 2 in both dimensions

- why use?
  - decrease size = reduce parameters
    - ↳ reduce training time
    - ↳ prevent overfitting

Architecture

Conv → Pooling → ReLU → Conv → Pooling → ReLU → ...

↳ don't necessarily need pooling every time
if convolution has stride > 1

- start bigger at input
  get smaller as it goes towards output
  not necessarily getting smaller in all dimensions
  (might get shorter and fatter)

bigger ⟶ smaller

- eventually pass through softmax (or choice of output activation function) at end

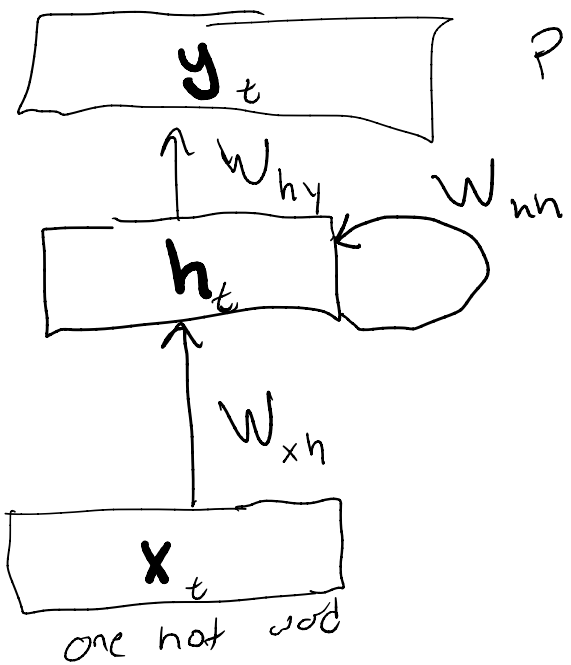
Training CNNs:

- same process (backpropagation)
- common to use
  a) regularization
  b) adaptive learning
     Adam, RMSProp, Adagrad
     typically specify as optimizer

Recurrent Neural Network

- designed for sequential data
  Ex: text sequences, speech, weather, stocks
  ↳ in sequences, successive values/words dependent on one another
- idea: implement recurrence relation
  ↳ use a feedback loop
  ↳ provides "memory"
  ↳ no longer just "feed-forward"
- computation can be unfolded in time

Example: predicting next word in text



Unfolded