Midterm Review

Summary of Course Material

Basics of neural networks:

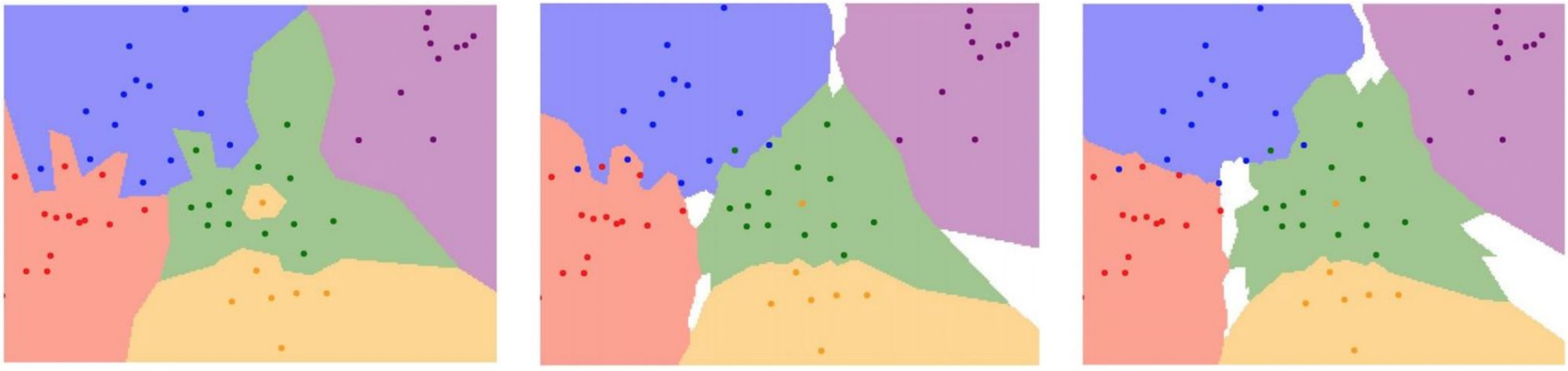
- Loss function & Regularization
- Optimization
- Activation Functions

How we build complex network models

- Convolutional Layers

KNN

- How does it work? Train? Test?
- What positive / negative effect would using larger / smaller k value have?
- Distance function? L1 vs. L2?



K = 1 K = 3 K = 5

Linear Classifier

Image

Array of **32x32x3** numbers (3072 numbers total)

$\longrightarrow$ f(**x**,**W**) $\longrightarrow$ **10** numbers giving class scores

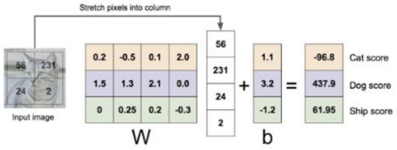
$\uparrow$

W
parameters or weights

$$f(x,W) = Wx + b$$

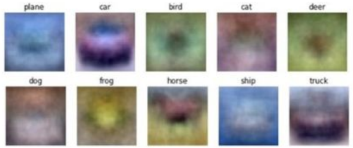
Algebraic Viewpoint

$f(x,W) = Wx$



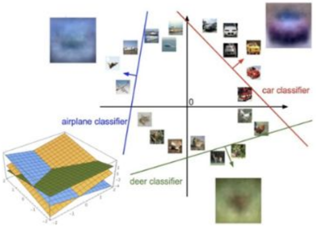
Visual Viewpoint

One template
per class



Geometric Viewpoint

Hyperplanes
cutting up space



Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and y_i is (integer) label

Loss over the dataset is a average of loss over examples:

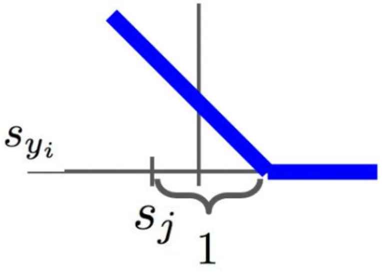
$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Multiclass SVM Loss

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$
$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

“Hinge loss”



cat	**3.2**
car	5.1
frog	-1.7
Losses:	2.9

$= \max(0, 5.1 - 3.2 + 1) + \max(0, -1.7 - 3.2 + 1)$

$= \max(0, 2.9) + \max(0, -3.9)$

$= 2.9 + 0$

$= 2.9$

Softmax Classifier

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i | X = x_i)$$

	Unnormalized log-probabilities / logits		unnormalized probabilities		probabilities
cat	**3.2**	exp	**24.5**	normalize	**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

$\rightarrow$ L_i = -log(0.13)
= 2.04

Loss Intuition:
- **Maximizing Likelihood**
- **Comparing Probabilities**

Regularization

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

E.g. Suppose that we found a W such that L = 0.
Is this W unique?

No! 2W also has L = 0!
How do we choose between W and 2W?
Regularization

Regularization

$$L(W) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{\text{Data loss}} + \underbrace{\lambda R(W)}_{\text{Regularization}}$$

Data loss: Model predictions should match training data

Regularization: Prevent the model from doing *too* well on training data

Why regularize?

- Express preferences over weights
- Make the model *simpler* to avoid overfitting

Optimization Motivation

- We have some dataset of (x,y)
- We have a **score function:**
- We have a **loss function**:

$$s = f(x; W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$

How do we find the best W?

Gradient Descent

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

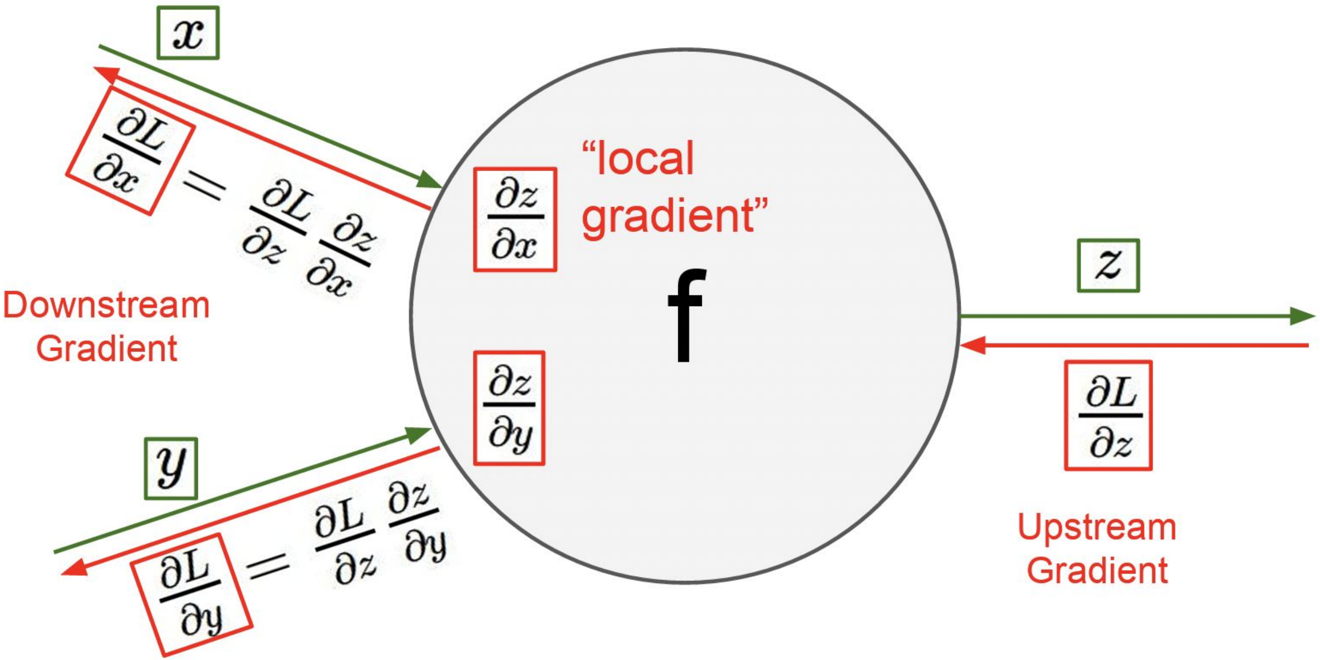
Full sum expensive when N is large!

Approximate sum using a **minibatch** of examples
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent

while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

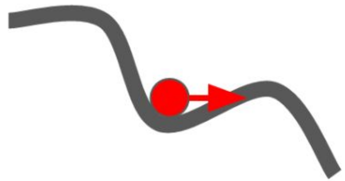
Optimization, Point 1: Calculating Gradients for Updates



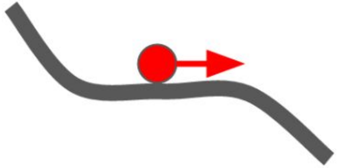
Computational Graphs + Backpropagation (Chain Rule)

Optimization, Point 2: Things to take care!

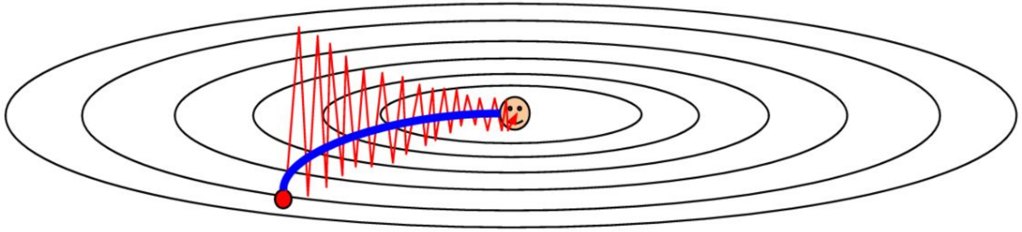
Local Minima

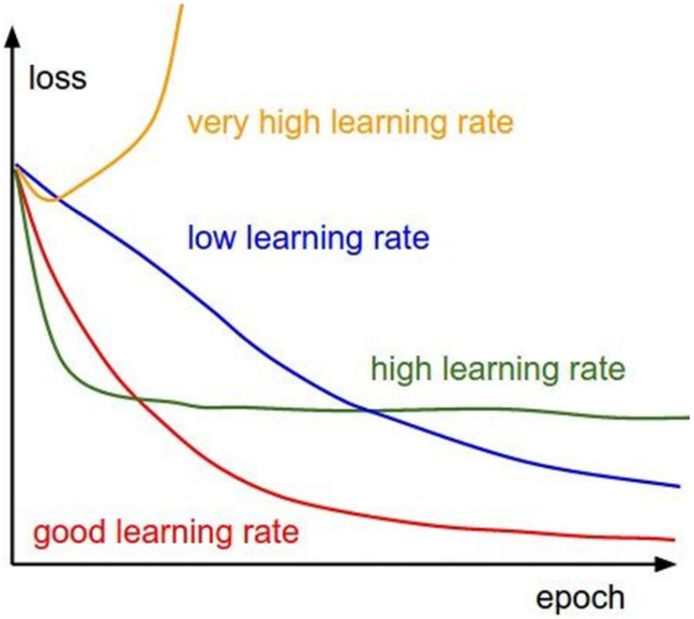


Saddle points



Poor Conditioning



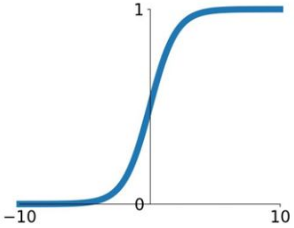


Other Algos: SGD+momentum, AdaGrad, RMSProp, Adam

Activation Functions

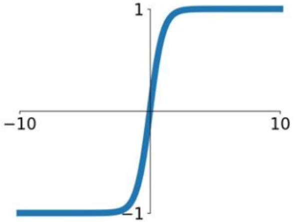
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



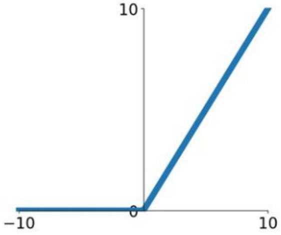
tanh

$\tanh(x)$



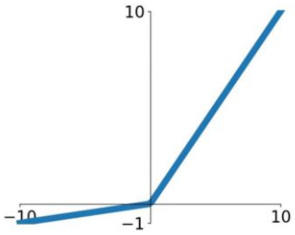
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

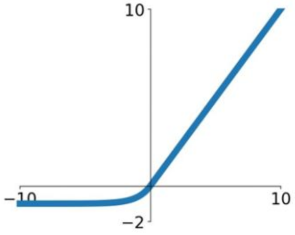


Maxout

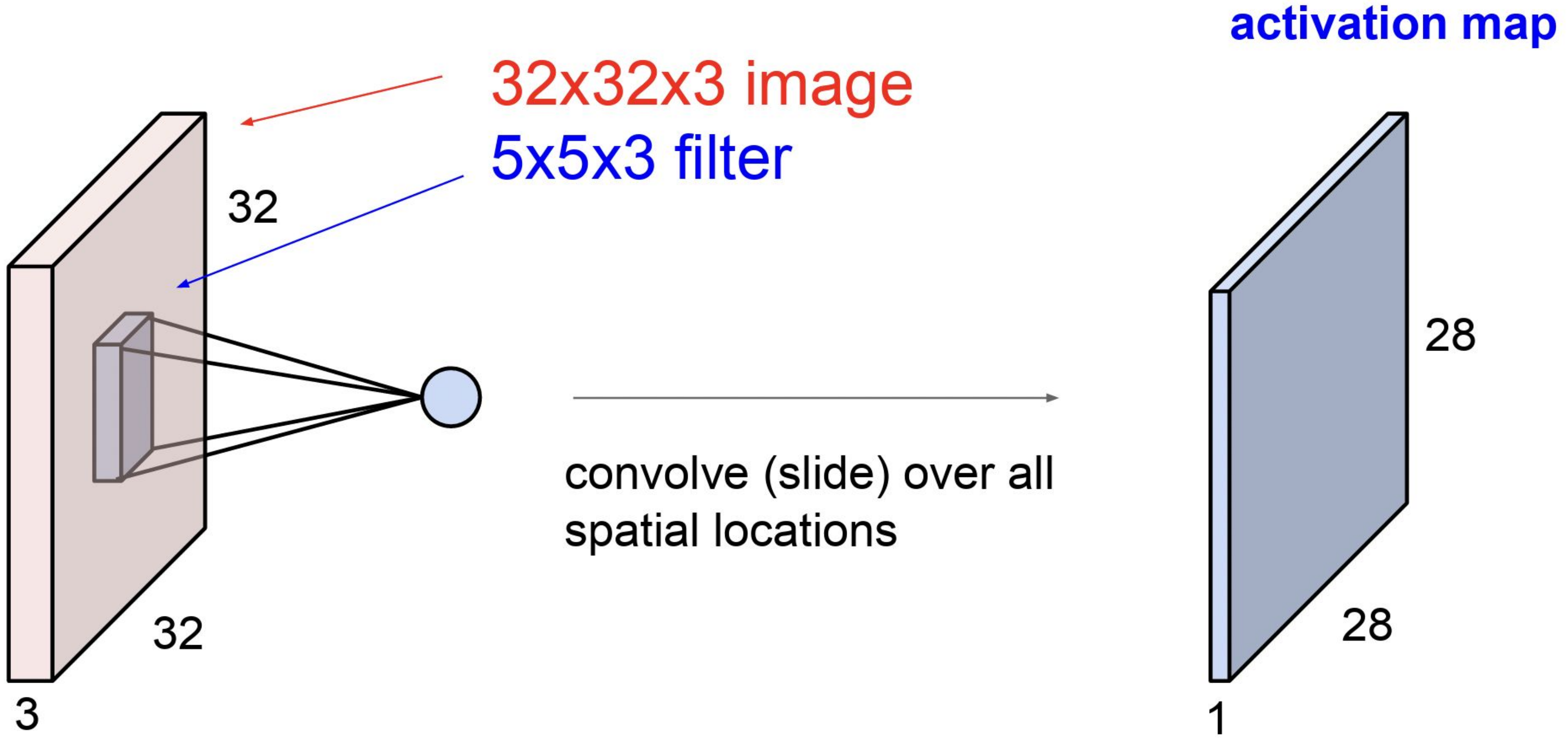
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

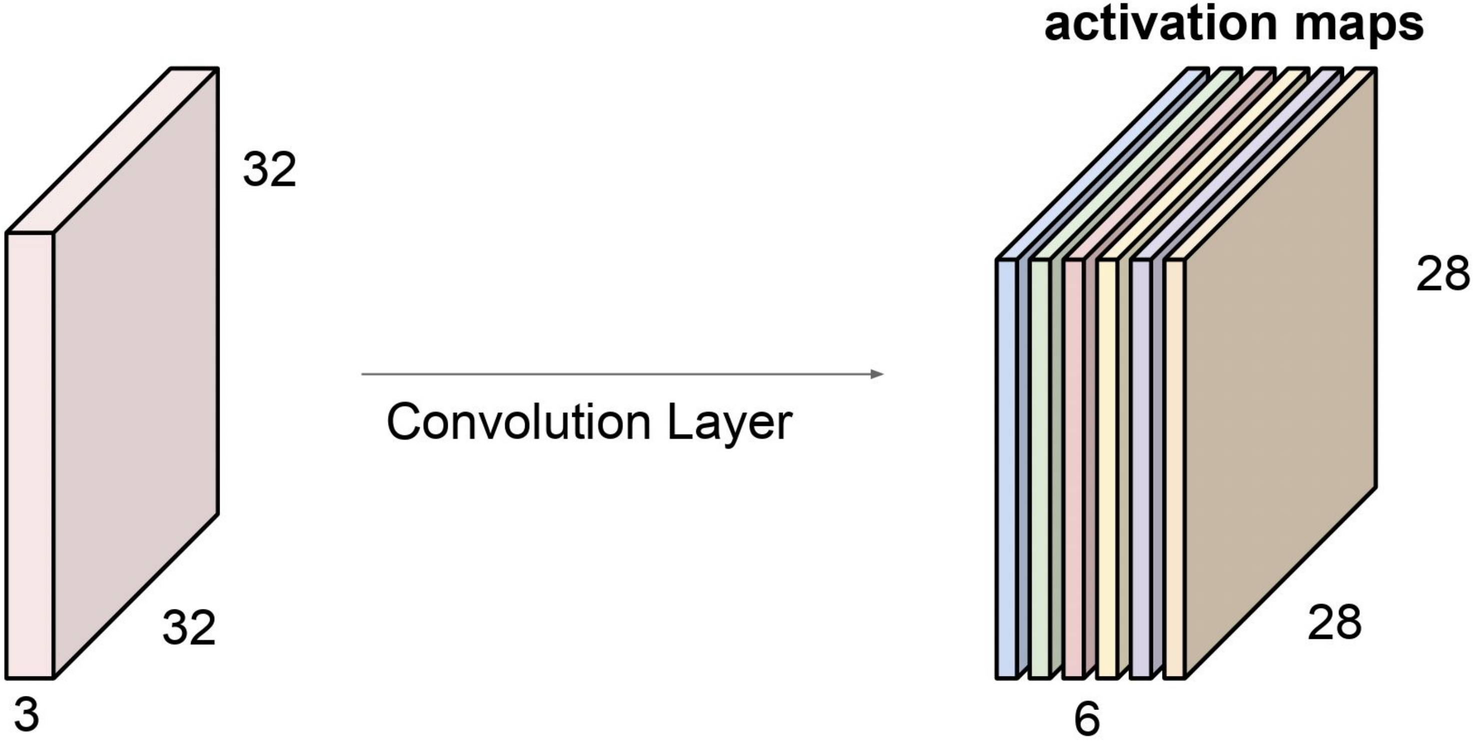
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Convolution Layer



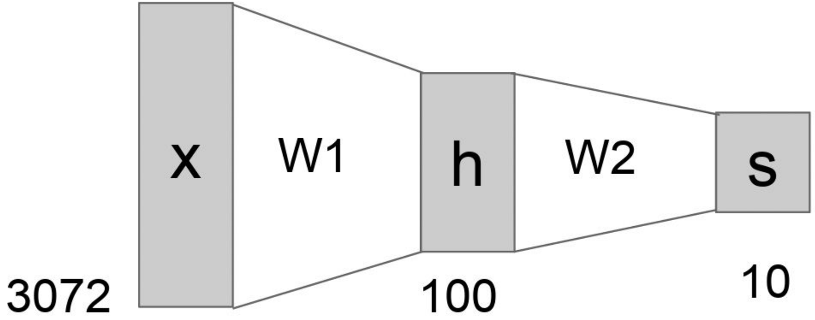
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Convolution Layer

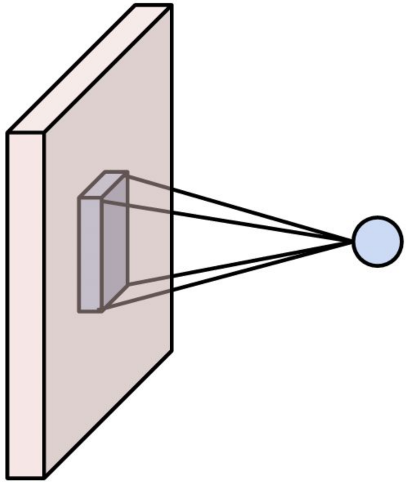
In contrast to fully connected layer,
Each term in output is dependent on spatially local 'subregions' of input



$$out_i = \sum_{j=1}^{H \times W \times C} w_{ij} \cdot in_j + b_i$$

Convolution Layer

In contrast to fully connected layer,
Each term in output is dependent on spatially local ‘subregions’ of input



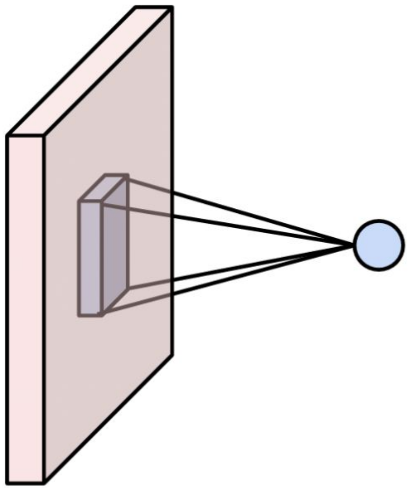
Question: connection between an FC layer and a convolutional layer?

$$out_i = \sum_{j=1}^{H \times W \times C} w_{ij} \cdot in_j + b_i$$

$$out_i = \sum_{j=1}^{HH \times WW \times C} w_{ij} \cdot in(patch_i)_j + b_i$$

Convolution Layer

In contrast to fully connected layer,
Each term in output is dependent on spatially local 'subregions' of input



Question: connection between an FC layer and a convolutional layer?

Answer: FC looks like convolution layer with filter size HxW

$$out_i = \sum_{j=1}^{H \times W \times C} w_{ij} \cdot in_j + b_i$$

$$out_i = \sum_{j=1}^{HH \times WW \times C} w_{ij} \cdot in(patch_i)_j + b_i$$

Convolution Layer

For kernel width $\mathbf{k}$ and stride $\mathbf{s}$,
Input width $\mathbf{w}_{in}$ and total padding $\mathbf{w}_{pad}$,
Output width $\mathbf{w}_{out}$ is

$$w_{out} = \frac{1}{s}(w_{in} + w_{pad} - k) + 1$$