Today:

**CNNs**

- working w/ multi-channel images
- backprop for CNNs
- issues w/ backprops: vanishing & exploding gradients
- remedies:
  - pretraining
  - (2014) dropout
  - (2015) batch normalization
  - modify architecture of the network

- common CNN architectures:
  - VGG (repeating structures)
  - GoogLeNet (inception architecture)
  - ResNets (residual blocks)
Multi-channel inputs to convolutional layers?

Example: Consider a 3-channel input with 32-channel output (a Conv2D layer with 32 filters) and each filter is $3 \times 3 \Rightarrow$ you have $3 \times 3 \times 3 \times 32 \times 3 + 32 = 8960$ parameters for this layer.
Say we have $n_l$ channels on convolutional layer $l$ and for each channel we have kernels $K_j^l, c$ where $j = 1, \ldots, n_{l-1}$ and $c = 1, \ldots, n_l$ and we write $O_j^l$ as the $j$th output channel on layer $l$ and $A_j^l$ as the preactivation of the $j$th channel on layer $l$.

$$A_c^l = \sum_{t=1}^{n_{l-1}} O_t^{l-1} \ast K_j^l, c + b_c^l$$ bias for the $j$th channel on layer $l$.

$$O_c^l = \sigma(A_c^l)$$ for each output channel $c$ on layer $l$. 

Backprop for CNNs

Now we want to learn \( \frac{\partial f}{\partial x_{t,c}} \rightarrow \frac{\partial f}{\partial b_{t,c}} \)

Difficulty:
Vanishing & Exploding Gradients

Phenomenon that arises in deep neural networks: L-2 norm of the gradient tends to zero or infinity. 

Why? Because of the chain rule:

\[ \nabla_{\omega} f = \left[ \omega^{d+1} \right]^T \left( \nabla_{\omega^{d+1}} f \circ \sigma'(a^{d+1}) \right) \]

and

\[ \nabla_{w} f = \text{diag}(\sigma'(a^{d})) \left( \nabla_{\omega} f \circ \sigma'(a^{d}) \right) \]

\[ \nabla_{b} f = \nabla_{\omega} f \circ \sigma'(a^{d}) \]
Two considerations:

1) The saturation of our activation function, given by $\sigma'(a_{d+1})$ and $\sigma'(a_{d})$

2) The norm of our weight matrix $\|W^{d+1}\|_2$

- if, on average, $\|W^{d+1}\|_2 \geq \|x\|_2$
  then we will get exploding gradients

- if, on average, $\|W^{d+1}\|_2 \leq \|x\|_2$
  then we will get vanishing gradients
Vanishing & Exploding gradients are difficult to handle w/ first order optimization algorithms.
- The scales of the gradients can vary widely b/w layers

- Next class: remedies for this problem
Basic idea is to note that we can represent our images however we want (and the operations on them).

Given an input image \( I \) we note \((d_1 - k_1 + 1) \times (d_2 - k_2 + 1)\).

\[ d_1 \rightarrow d_2 \rightarrow \text{"unfold"} \rightarrow (d_1 - k_1 + 1) \times (d_2 - k_2 + 1) \]

Reach into the valid regions.
\[ \text{vec}(I \star R) \in \mathbb{R}^{(d_1-k_1+1) \times (d_2-k_2+1)} \]

\[ \text{vec}(I \star R) \in \mathbb{R}^{(d_1-k_1+1) \times (d_2-k_2+1)} \]

Question: What is the relationship between \( \text{vec}(I \star K) \) and \( \text{unfold}(I) \)?
\[ \text{vec}(I \ast K) \]
\[ R_{(a_1-k_1+1) \times (d_2-k_2+1)} \]
\[ = \frac{\text{vec}(K)}{R_{k_1 \times k_2}} \]

**Consequence:**

\[ \text{vec}(I \ast K) = \text{vec}(K) \cdot \text{unfold}(I) \]

\[ \Rightarrow I \ast K = \text{unvec} \left( \frac{\text{vec}(K) \cdot \text{unfold}(I)}{\text{this is a vector, and we know how to backprop w.r.t. param vectors}} \right) \]