ML and Opt Lecture 19

- Transposed Convolutions
- Backprop for Convolutional Layers (sketch)
- Vanishing and Exploding Gradients
- Batch Normalization
- Google Inception architecture (v1); auxiliary losses
Transposed Convolutions

To "invert" a convolutional filter \( K \), we learn via BP, a filter \( \overline{K} \).

We need choose \( z > \overline{p'} \) so that the images from \( K \) when passed through \( \overline{K} \) have dimension \( d \).
Let \( o \) be the size of the image from the forward convolution

\[
o = \frac{d + 2p - k + 1}{s}
\]

Then \( d' \) be the size of the image from the transposed convolution when the input is size \( o \)

\[
d' = o + (o-1) \cdot z + 2p' - k + 1
\]

Want to choose \( z \) and \( p' \) so \( d' = d \)

Claim is:

\[
\begin{align*}
z &= s - 1 \\
p' &= k - p - 1
\end{align*}
\]

\[
d' = \left( \frac{d + 2p - k + 1}{s} \right) + \left( \frac{d + 2p - k}{s} \right) \cdot (s - 1) + 2(k - p - 1) - k + 1 \\
= 1 + d + 2p - k + ak - 2p - a - k + 1 \\
= d
\]
Efficient Convolutions & Backprop

Parallels b/w MLPs and ConvNets

MLP:

\[ a^{l+1} = w^{l+1} o^l + b^{l+1} \]
\[ o^{l+1} = \sigma(a^{l+1}) \]

Conv:

- CNNs:
  \[ A_1^{l+1} = o_1^l \ast K_{1,1}^{l+1} + \ldots + o_3^l \ast K_{1,3}^{l+1} + b^{l+1} \]
  \[ o_1^{l+1} = \sigma(A_1^{l+1}) \]

A scalar that is the same for each pixel
If layer $l$ has $n_l$ channels, then

$$A_{i}^{l+1} = \left( \sum_{j=1}^{n_l} O_{j}^{l} \ast K_{i,j}^{l+1} \right) + b_{i}^{l+1}$$

$$O_{i}^{l+1} = \sigma(A_{i}^{l+1})$$

The number of parameters for layer $l+1$ is

$$n_{l} n_{l+1} k_{1} k_{2} + n_{l+1} \quad \text{(assuming all the filters connecting layer $l$ to layer $l+1$ are $k_{1} \times k_{2}$)}$$
Efficient Computations

Sketch) Efficient Computations of Manas Sahni “Anatomy of a High-Speed Convolution”

idea: reduce convolution to the \texttt{im2col} operation and \texttt{GEMMs}

- \texttt{im2col} takes an image in $\mathbb{R}^{d_1 \times d_2}$ and maps to a matrix $\mathbb{R}^{k_1 k_2 \times d_1 d_2}$ (corresponding to zero-padding the input and convolving by a $k_1 \times k_2$ filter)
note that we want to compare \( O^i_j \cdot K^i_{i', j} \)
and this implies

\[
\text{vec} \left( O^i_j \cdot K^i_{i', j} \right) = \text{vec}(K^i_{i', j}) \cdot \text{im} \text{diag}(O^i_j)
\]
Issues with deep NNs (not just CNNs):
- overfitting (too much capacity for the amount of training data)
- vanishing & exploding gradients \(\Rightarrow\) slow learning
- hyperparameter selection, e.g.
  - kernel sizes?
  - # channels per layer?
  - # layers?
  - type of pooling & locations?
  - stride & dilation & padding?
  - learning rates? algorithm? minibatch size?
  - weight decay?
  -...
Vanishing & Exploding Gradients

Phenomenon that as $L \to \infty$

as $l \to 1$,

$\| \nabla_{\omega_l} f \|_2 \to \begin{cases} 0 & \text{vanishing gradients} \\ \infty & \text{exploding gradients} \end{cases}$

Why? Because of the chain rule.

E.g. for MLPs:

$\nabla_{\omega_l} f = \text{diag} \left( \sigma'(a^{l+1}) \right) (\omega^{l+1})^T \nabla_{\omega^{l+1}} f$

$= \left[ \prod_{i=2}^{L} \text{diag} \left( \sigma' \left( a^i \right) \right) (\omega^i)^T \right] \nabla_{\omega_L} f$
Two considerations:

1) How \( \text{diag}(\sigma'((q^{t+1}))) \) behaves

\[ \sigma' \ - \ logistic \ \text{sigmoid} \]

so if \( q \) is far from 0, then this looks like a zero matrix, so

\[ \| \nabla_{o_{ef}} \|_2 \ll \| \nabla_{o_{eftl}} \|_2 \]

2) If the norm of our weight matrix \( W^{t+1} \) is large, then \( \| \nabla_{o_{ef}} \|_2 \gg \| \nabla_{o_{eftl}} \|_2 \)

if it is small, then \( \| \nabla_{o_{ef}} \|_2 \ll \| \nabla_{o_{eftl}} \|_2 \)
Consequence:
naively training deep NN architectures either fails or gives poor performance
GoogleNet (22 layer) CNN Inception v1

Naïve Inception Block

Downside: lots of parameters

E.g. for the $3 \times 3$ conv features, we have

$$192 \times 128 \times 3 \times 3 + 128 = 221,312 \text{ parameters}$$
Issue: too many parameters
Soln: use 1x1 conv layers for dimensionality reduction

Check: # of 3x3 conv parameters (including the 96 1x1 conv) = 129,248 parameters
Original Inception (v1) architecture

CNN → Inception Block → 3x3 max pool
   x2

→ Inception Block → aux classification loss
   x5

→ Inception Block → 3x3 max pool

→ Inception Block → AugPool → Dropout
   x2

→ Linear → softmax → classification loss

Train this architecture to minimize the weighted sum of the three losses. Injects gradient information at intermediate layers to mitigate vanishing/exploding gradients.