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

Convolution

pad $P$

convolve $w/\text{stride}$

filter of size $k$

Transposed Convolution

pad $P'$

convolve $w/\text{stride}$

filter of size $k$

To “invert” a convolutional filter $K$, we learn via BP a filter $\overline{K}$. We need choose $z > P'$ so that the images from $K$ when passed through $\overline{K}$ have dimension $d$. 

$z$
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:

$$z = s-1$$

$$p' = k - p - 1$$

$$d' = \left( \frac{d + 2p - k + 1}{s} \right) + \left( \frac{d + 2p - k}{s} \right) \cdot (s-1) + 2(k-p-1) - k + 1$$

$$= 1 + d + 2p - k + ak - 2p - a - k + 1$$

$$= d$$
Efficient Convolutions & Backprop

Parallels between MLPs and CNNs:

Multilayer Perceptron (MLP):
\[ a^{l+1} = \omega^{l+1} o^l + b^{l+1} \]
\[ o^{l+1} = \sigma(a^{l+1}) \]

Convolutional Neural Networks (CNNs):

\[ A_1^{l+1} = o_1^l * k_{1,1}^{l+1} + \ldots + o_2^l * k_{1,2}^{l+1} + o_3^l * k_{1,3}^{l+1} + b_1^{l+1} \]
\[ o_1^{l+1} = \sigma(A_1^{l+1}) \]
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

\[
n_l n_{l+1} K_1 K_2 + n_{l+1} \] (assuming all the filters connecting layer \( l \) to layer \( l+1 \) are \( K_1 \times K_2 \))
(Sketch) Efficient Computations

of Manas Sahni “Anatomy of a High-Speed Convolution”

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

- \texttt{im2col} takes an image in $\mathbb{R}^{d_1 \times d_2}$ and maps to a matrix $\mathbb{R}^{k_1 \times k_2 \times d_1 \times d_2}$ (corresponding to zero-padding the input and convolving by a $k_1 \times k_2$ filter)
note that we want to compare $O_{i,j}^l \ast K_{i,j}^{l+1}$
and this implies

$$\text{vec} \left( O_{i,j}^l \ast K_{i,j}^{l+1} \right) = \text{vec}(K_{i,j}^{l+1}) \ast \text{im2col}(O_{i,j}^l)$$
Issues with deep NNs (not just CNNs):
- overfitting (too much capacity for the amount of training data)
- vanishing & exploding gradients ⇒ slow learning!

- hyperparameter selection, e.g.
  - kernel sizes?
  - # channels per layer?
  - # layers?
  - type of pooling & locations?
  - stride & dilatation & padding?
  - learning rates? algorithm? minibatch size?
  - weight decay?
  - ....
Vanishing & Exploding Gradients

Phenomenon that as $L \to \infty$
as $l \to 1$: $||\nabla_{w^l} f||_2 \to \{0 \text{ vanishing gradients}, \infty \text{ exploding gradients}\}$

Why? Because of the chain rule.
E.g. for MLPs:

$$\nabla_{o^l} f = \text{diag}(\sigma'(a^{l+1})) (w^{l+1})^T \nabla_{o^{l+1}} f$$

$$= \left( \prod_{i=l+1}^{L} \text{diag}(\sigma'(a^i)) (w^i)^T \right) \nabla_{o^l} f$$
Two considerations:

1) How $\text{diag}(\sigma'(a^T x))$ behaves

- $\sigma'$ - logistic sigmoid

so if $a^T$ far from 0, then this looks like a zero matrix, so

$$\|\nabla_o e f\|_2 \ll \|\nabla_o e + f\|_2$$

2) If the norm of our weight matrix $W^{l+1}$ is large, then $\|\nabla_o e f\|_2 \gg \|\nabla_o e + f\|_2$

if it is small, then $\|\nabla_o e f\|_2 \ll \|\nabla_o e + f\|_2$
Consequence:
naiively training deep NN architectures
either fails or gives poor performance
GoogleNet (22 layer) CNN Inception v1

Naive Inception Block

Downside: lots of parameters

E.g. for the 3x3 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 → 3x3 max pool
        x5
        → Inception Block → AvgPool → 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.