ML and Optimization Lecture 21

- Convolutional Neural Networks continued

- issues of deep ANNs:
  - overfitting
  - vanishing & exploding gradients

remedies:
- dropout (2014)
- (2015) batch normalization
Multi-channel CNNs

MLP:

CNN:

\[ A_{l+1} = O_1^l \ast K_{13}^{l+1} + \\
O_2^l \ast K_{12}^{l+1} + \\
O_3^l \ast K_{13}^{l+1} + b_{l+1} \]

\[ O_{l+1}^1 = \sigma(A_{l+1}^1) \]
In general if layer $L$ has $n_L$ channels, then

$$A^{l+1}_i = \left( \sum_{j=1}^{n_L} O^{l}_j * K^{l+1}_{i,j} \right) + b^{l+1}_i$$

$$O^{l+1}_i = \sigma(A^{l+1}_i)$$

for $i=1, \ldots, n_{L+1}$

What about backprop?

- convolutions
Convolutions in practice (GEMM)

cf. Manas Sahni "Anatomy of a High-Speed Convolution"

reduce convolutions to the im2col operation and GEMMs

- im2col takes an image $\in \mathbb{R}^{d_1 \times d_2}$ and maps to a matrix in $\mathbb{R}^{k_1 k_2 \times d_1 d_2}$ (corresponding to zero-padding the input and convolving by a $k_1 \times k_2$ kernel)

[Diagram of im2col with zero-padding]
Then note that

$$\text{vec}(I \ast K) = \text{vec}(K) \circ \text{im2col}(I)$$
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
  - optimal kernel sizes
  - # of channels per layer
  - # of layers
  - pooling types & locations
  - stride sizes
  - learning rates (function of minibatch size)
  - weight decay amount
  - more...
Vanishing & Exploding Gradients

Phenomenon that

\[ \| \nabla_{\text{input}} f \|_2 \xrightarrow{l \to 1} \{ \begin{array}{ll} 0 & \text{vanishing gradients} \\ \infty & \text{exploding gradients} \end{array} \] 

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

\[
\nabla_{\text{output}} f = \text{diag}(\sigma'(z_{l+1})) (\omega^l_{l+1})^T \nabla_{\text{output}} f
\]

Two considerations:

1) The saturation of our activation function

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

If preactivations are far from origin, \( \sigma'(z_{l+1}) \) close to zero, so

\[ \| \nabla_{\text{output}} f \|_2 \ll \| \nabla_{\text{output}} f \|_2 \]
2) The norm of our weight matrix $W^{l+1}$

- if on average $\|W^{l+1}\|_2 \geq \|x\|_2$
  then we get $\|\nabla_{x} f\|_2 \geq \|\nabla_{x} f^{l+1}\|_2$

- if on average $\|W^{l+1}\|_2 \leq \|x\|_2$
  then we get $\|\nabla_{x} f\|_2 \leq \|\nabla_{x} f^{l+1}\|_2$

Consequence: naively training deep NN architectures either fails or gives poor performance.

Idea: the feature distributions at each layer vary wildly, so normalize them so they look like standard gaussians

[Diagram of distributions]
In convex ML we saw the advantage of normalizing our input data so the features are on the same scale. 

**Example:** training to predict credit default probability.

\[ x = [\text{salary}, \text{age}] \]

To get the features on the same scale we normalized them.

\[ x \xrightarrow{1} \left[ \text{salary} - \frac{\mu_{\text{salary}}}{\sigma_{\text{salary}}}, \text{age} - \frac{\mu_{\text{age}}}{\sigma_{\text{age}}} \right] \]
Idea of BN: add normalization layer after a regular layer to rescale and center the feature distributions.
Between layer $l$ and $l-1$ add a BN layer:

\[ a^l = \text{BN}_{\gamma^l, \beta^l} (\omega^l o^{l-1}) \]

\[ o^l = \sigma (a^l) \]

\[ \text{BN}_{\gamma^l, \beta^l} (o^{l-1}) = \gamma^l o^{l-1} \frac{o^{l-1} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} + \beta^l \]

where $\gamma^l \in \mathbb{R}^{n_{l-1}}$ is a scale vector and $\beta^l \in \mathbb{R}^{n_{l-1}}$ is a shift vector.

are useful to ensure we are in the nonlinear region of the activation function.
Forward & Backward pass during training is straightforward b/c $\mu_B$ and $\sigma^2_B$ are computed using minibatch means (model.train() in pytorch).

What about inference time (model.eval())

1) After fitting, run all our training data through and compute

$$\mu^e = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} (c_{0}^e)^i \in \mathbb{R}^{n_e-1}$$

$$\sigma^2_e = \frac{1}{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} \sum_{i=1}^{n_{\text{train}}} ((c_{0}^e)^i - \mu^e)^2 \in \mathbb{R}^{n_e-1}$$

and use these
2) or maintain moving average estimates of population mean & variance

\[ \mu^e < \beta_1 \mu^e + (1 - \beta_1) \mu^B \]

\[ \sigma^e < \beta_2 \sigma^e + (1 - \beta_2) \sigma^B \]

for each minibatch

and use \( \mu^e \) and \( \sigma^e \) at inference time
layer $l$ $\rightarrow$ BN $\rightarrow$ layer $l+1$

$w^l, b^l$

$w^{l+1}, b^{l+1}$

$\gamma^l, \beta^l$