ML and Optimization Lecture 19

- Normalization Approaches to Avoiding Vanishing & Exploding Gradients:
  - Batch Normalization
  - Layer Normalization
- Residual/Skip connections
Batch Normalization

Address activation saturation

- we have decreasing gradient size if \( a^l \) is far away from the "active" portion of the activation function.

\[
d^{l+1} = w^{l+1} o^l + b^{l+1}
\]

Depending on the distribution of the outputs from layer \( l \), each neuron decreases or contributes to the gradient in backprop.
In convex ML we saw the advantage of normalizing our input data so the features are on the same scale. Ex: training to predict credit default probability

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

\[ \begin{align*}
40K & \quad 80K \\
30 & \quad 40
\end{align*} \]

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

\[ x \rightarrow \left[ \frac{\text{salary} - \mu_{\text{salary}}}{\sigma_{\text{salary}}} \quad \frac{\text{age} - \mu_{\text{age}}}{\sigma_{\text{age}}} \right] \]
Idea of BN: add normalization layer after each preactivation to rescale and center the feature distributions.

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

\[
o^l = \sigma(\overline{a}^l)
\]

where

\[
\text{BN}_{\gamma^l, \beta^l}(a^l) = \sigma^l \left( \frac{a^l - \mu^l}{\sqrt{\sigma^l_a + \varepsilon}} \right) + \beta^l
\]
Here:

$\gamma^l \in \mathbb{R}^{n_l}$ is a scaling vector that controls the variance of the preactivations $\bar{a}^l$

$\beta^l \in \mathbb{R}^{n_l}$ is a shift vector that controls the means of the preactivations $\bar{a}^l$

$\varepsilon$ - some constant (small, $10^{-6}$) to avoid dividing by zero

$\gamma^l$ and $\beta^l$ are learned via backprop to ensure that each neuron's preactivation is in a useful portion of the activation function's domain.
The means \( \mu_e \) and variances \( \sigma_e^2 \) of the preactivations are computed during training using minibatches:

e.g. if our minibatch is of size \( m \)

\[
\mu_e = \frac{1}{m} \sum_{i=1}^{m} a^e(x_i)
\]

\[
\sigma_e^2 = \frac{1}{m} \sum_{i=1}^{m} (a^e(x_i) - \mu_e)^2
\]

Caveat: this only makes sense if \( m \gg 1 \).

- This means when we don't have enough memory to use a minibatch of size \( \gg 1 \), we can't use BN layers
During test time, we can use a running average of $\mu_e$ and $\sigma^2_e$:

$$\mu^e \leftarrow \beta_1 \mu^e + (1 - \beta_1) \mu^B$$

for each minibatch $B$

$$\sigma^2_e \leftarrow \beta_2 \sigma^2_e + (1 - \beta_2) \sigma^2_e, B$$

and use $\mu_e$ and $\sigma^2_e$ at test time.

- Batch normalization for CNNs is done on a per-channel basis: you compute the means and variances using all the preactivations in a channel, and shift and scale all those preactivations in a channel by the same amount. This preserves translational invariance of features.
Layer 2 ➔ Linear layer ➔ Batch Normalization ➔ Layer $l+1$
Layer Normalization

When Batch Normalization does not apply (e.g., not enough memory for large batches) or is expensive (e.g., with RNNs you need to store separate BN parameters for each possible sequence length), we can try layer normalization.

Treat all the neurons in a layer as i.i.d. samples, and use to estimate the means and variances used in normalization.

$$\bar{a}^l = \text{LayerNorm}_{\gamma, \beta}(a^l) = \frac{\gamma a^l - \mu}{\sigma} + \beta$$

Where $$\gamma \in \mathbb{R}^{n_{i}}$$ and $$\beta \in \mathbb{R}^{n_{i}}$$

And $$\mu = \frac{1}{n_{i}} \sum_{i=1}^{n_{i}} a^{l}(x)$$ and $$\sigma^2 = \frac{1}{n_{i}} \| a^{l}(x) - \mu \|_2^2$$
Skip/Residual Connections

Standard MLP

\[ f(\text{input}) \]

Input

vanishing or exploding gradients

Skip-connections (example)

\[ f(\text{input}) \]

Input

back prop longer indirect path for gradient information

Skip-connections are connections b/w layers \( l \) and layers higher than \( l + 1 \), e.g., DenseNet where all layers are connected.

Rationale for skip-connections:

1) purely pragmatic: helps w/ gradient propagation
2) inductive bias: higher layers have access to simple features (e.g., U-Nets)
A specific popular architecture in this class is the ResNet.

\[ o_{l+1} = \sigma (\omega^{l+1} o_l + b^{l+1}) + o^l \]

Idea: this architecture allows us to easily compute simple functions like \( f(x) = x \) as well as complex functions involving nonlinearities.