ML and Optimization Lecture 12

- Dropout regularization (model ensembling)
- Inception architecture
- Skip-connections & residual layers
  - Briefly mention ResNet
- Introduce RNNs (recurrent neural networks)
Dropout
- form of regularization introduced to prevent overfitting
  - led to a series of follow-ups: DropConnect, etc.

Intuition: to avoid "feature co-adaptation"
(where some neurons in one layer learn brittle, non-generalizing, combinations of features in the previous layer)
randomly drop some activations from the previous layer to zero, so neurons have to learn to compute robust features during training. These random dropouts change at each minibatch during training.
At each minibatch:
- randomly sample Bernoulli mask for each neuron to be considered dropping
- do forward & backward pass on the resulting subnetwork to update the model parameters
If we add a dropout layer b/w layers $l-1$ and $l$ of our NN, this is equivalent to changing our activation formulas to

$$a^l = \omega^l o^{l-1} + b^l$$

$$o^l = \sigma(a^l)$$

w/o dropout

$$a^l = \omega^l [\nu^{l-1}] o^{l-1} + b^l$$

$$o^l = \sigma(a^l)$$

dropout, where

$$\nu^{l-1} \in \{0, 1\}^{n_{l-1}}$$

and

$$(\nu^{l-1})_i \sim \text{Bern}(p)$$

where $p$ is a hyperparameter

and $\nu^{l-1}$ is resampled for each minibatch
How to use a network trained using dropout for inference?

Choices:
(1) could use dropout as before (random subnetwork) and get a random prediction

(2) (usually preferred) use the average activation in each layer:

$$E_{q_x} = E\left[ w^l (u^{l-1} \circ o^{l-1}) + b^l \right]$$

$$= w^l (E[u^{l-1}] \circ o^{l-1}) + b^l$$

$$= p(w^l \circ o^{l-1}) + b^l$$
An interpretation of Dropout as model ensembling

Very useful idea in machine learning: model averaging/ensembling

Fit models \( f_{\Theta_i}, \ldots, f_{\Theta_m} \) for the same task and take

\[
f(x) = \frac{1}{m} \sum_{i=1}^{m} f_{\Theta_i}(x)
\]

to be the model average. In practice \( f \) has lower error than the component models.

Problem is: training \( m \) models costs \( m \) times as much as training one
Dropout inexpensively trains the averaged model

\[ f(x) = \mathbb{E}_{p(\theta)} f_{\theta}(x) \]

where \( p(\theta) \) is the distribution over the weights of the architecture.

Dropout computes subgradient for \( f \) at each iteration of minibatch SGD:

if \( f_i \) is a sample from the dropout architecture distribution, then

\[ \mathbb{E} f_i = f(x) \]

and

\[ \mathbb{E} \nabla w f_i = \nabla_w \mathbb{E} f_i = \nabla_w f(x) \]

so \( \nabla w f_i \) is a stochastic subgrad for \( f(x) \).
Inception architecture  SOTA performance ILSVRC 2014 on
22 layer deep initially

Ideas: max pooling loses spatial details
CNNs as we've seen have issues w/ detecting
the same features at different scales
more channels you have, the more memory you use
in training & slower it is.

Trend before inception: increase # layers & # filters per layer,
use dropout to prevent overfitting
E.g. if we have \( n \) channels on layer \( L \) then there are \( n \times \sum_{k} k + n \) parameters connecting layer \( L \) to layer \( L+1 \).

Consequence
If increase # of output channels on each layer by \( c \), # parameters increase by factor of \( O(c^2) \).
GoogleNet (22 layer) Inception V1

- DepthConcat (256)
  - 1x1 conv (164)
  - 3x3 conv (28)
  - 5x5 conv (32)
  - 1x1 conv (32)
  - 13x3 max pool

28x28 x 192

"Naïve" Inception Block

Downside: lots of parameters

E.g., for 3x3 conv block, we have

192 x 128 x 3x3 + 128 = 221,312 parameters
Issue: increased \# params
Soln: use dimensionality reduction via \(1 \times 1\) convolutions

**Inception Block**

- Input: \(28 \times 28 \times 192\)
- \(1 \times 1\) conv (64)
- \(3 \times 3\) conv (128)
- \(1 \times 1\) conv (96)
- \(5 \times 5\) conv (32)
- \(1 \times 1\) conv (16)
- \(3 \times 3\) maxpool
- Depth Concat (256)
With the 1x1 conv for dim reduction, the # of parameters for the 3x3 conv features now becomes

\[(192 \times 96 \times 1 \times 1 + 96) + (96 \times 128 \times 3 \times 3 + 128)\]

for the 1x1 conv

for the 3x3 conv

= 129,248 parameters

significantly less
Original Inception (v1) architecture

CNN $\rightarrow$ Inception Block $\rightarrow$ 3x3 max pool $\times 2$ $\rightarrow$ aux classification loss

$\rightarrow$ Inception Block $\rightarrow$ 3x3 max pool $\times 5$ $\rightarrow$ aux classification loss

$\rightarrow$ Inception Block $\rightarrow$ AvgPool $\rightarrow$ Dropout $\rightarrow$ Linear $\times 2$

$\rightarrow$ softmax $\rightarrow$ classification loss

Train this architecture to minimize the weighted sum of the three losses.