COMPUTER VISION: DEEP LEARNING LAB COURSE
DAY 4 – BAG OF TRICKS TO START CNNS

SEBASTIAN HOUBEN
EVALUATION
Schedule

Today

- Initialization - *how to start*
- Data Augmentation - *how to prepare*
- Plot Interpretation - *how to train*
- Regularization - *how to fit*
- *Inspection* – *how to understand*
Convolutional Neural Nets

- Stack repeating elements
  - Small convolutions
  - ReLU
  - MaxPool
  - Fully connected layers

Alex Krizhevsky et al.
Initialization

- Avoid that all weights receive the same gradient
  - Symmetry breaking
  - \( w_i \sim \mathcal{N}(0, 0.001) \)
- Avoid that gradient grows (or shrinks) over multiple layers
  - Vanishing gradient
  - Exploding gradient
- Xavier initialization

\[
\frac{\partial}{\partial w} L(x, y, w, u) = \sum_{i=1}^{n} 2 \left( \sigma \left( u\sigma \left(wx^{(i)}\right) \right) - y \right) \cdot \sigma' \left( u\sigma \left(wx^{(i)}\right) \right) \cdot u\sigma' \left( wx^{(i)} \right) \cdot x^{(i)}
\]
Xavier Initialization

\[ w_1x_1 + w_2x_2 + \cdots + w_dx_d \]

\[ \text{Var} [w_1x_1 + w_2x_2 + \cdots + w_dx_d] = d \cdot \text{Var} [w_i] \cdot \text{Var} [x_i] \]

\[ \text{Var} [w_i] = \frac{1}{d_{in}} \]

- no non-linearity
- independent \( x_i \)
- with ReLU on average half of the inputs are zero: \( \text{Var} [w_i] = \frac{2}{d_{in}} \)
Data Augmentation

- Generate more data (good for most machine learning techniques)
- But Deep Learning in particular
- Train the network on examples with more variance
- Avoid overfitting
Data Augmentation

\[ x^{(k)} := \frac{x^{(k)} - \text{mean}(x^{(1)}, \ldots, x^{(n)})}{\text{std}(x^{(1)}, \ldots, x^{(n)})} \]

- Recap: Zero-center and normalized range (or standard deviation)
- Transforms the recognition should be invariant to:
  - Random crop / shifting
  - Slight rotation
  - Color transform
  - Adding noise (regularization)
  - Horizontal flip
  - Scaling
  - Occlusion
  - Stretching / Shearing
Data Augmentation – Fancy PCA

- Color distribution over all training images (after normalization)
- PCA to find the main axis of variance of this distribution
- Sample from 3d unit Gaussian and transform sample point to color distribution
- Add sampled color vector to all pixels of an image

\[
\begin{pmatrix}
r_1^{(1)} & g_1^{(1)} & b_1^{(1)} \\
r_2^{(1)} & g_2^{(1)} & b_2^{(1)} \\
\vdots & \vdots & \vdots \\
r_d^{(1)} & g_d^{(1)} & b_d^{(1)} \\
r_1^{(2)} & g_1^{(2)} & b_1^{(2)} \\
\vdots & \vdots & \vdots \\
r_d^{(n)} & g_d^{(n)} & b_d^{(n)}
\end{pmatrix}
\]
Plot Interpretation

Is my training going well?

- No. Debug or reduce learning rate.
- Not bad. Reduce learning rate.
- Somewhat. Increase learning rate.
- Yes.
Plot Interpretation

- Is my training going well?

Unstable. Increase batch size.
Plot Interpretation

- Underfitting
- Increase model complexity
Plot Interpretation

- Overfitting
- Regularization

![Plot showing loss vs. epochs with arrows indicating training and validation errors.](image)
BAG OF TRICKS TO START CNNS

Regularization - Early Stopping

- Watch validation error
- Stop training when validation error increases
- Add validation set to training set
  - Restart and stop after same number of epochs
  - Continue until same error level is reached
- Popular, cheap and easy

![Graph showing training and validation error over epochs]
Regularization – “Weight control“

- Overfitting goes along with
  - Large absolute weights
  - Zero weight: „Ignore the feature“
- Only important features should be used
- Ignore as many features as possible
- Also: Large weights result in exploding gradient
- Renormalize weight vectors after update (max norm constraints)

\[
L^*(x, y, W, U, \lambda) = L(x, y, W, U, \lambda) + \lambda ||w||_2^2
\]

\[
L^*(x, y, W, U, \lambda) = L(x, y, W, U, \lambda) + \lambda ||w||_1^2
\]
Regularization – “Weight control“

$$L^*(x, y, W, U, \lambda) = L(x, y, W, U, \lambda) + \lambda\|w\|^2_2$$

$$L^*(x, y, W, U, \lambda) = L(x, y, W, U, \lambda) + \lambda\|w\|^2_1$$
Regularization – Dropout

- Neurons should use each input independently
  - Correlating inputs might be mixed together
  - Waste of resources
  - Should also work if only one of the inputs is present
  - Randomly select \(100p\%\) units and set their output to zero (during training and forward pass)
- But: Sum of input weights gets lower
  - Increase output by \(\frac{1}{p}\)
  - During test time activate all neurons (no dropout)
Regularization – Dropout

- Extension of Data Augmentation
  - Add noise to input
  - Add noise to intermediate activations
- Alternative interpretation
  - Train $2^n$ nets at the same time and average their result
  - tensorflow.dropout

Srivastava et al. (2014)
Learning Stability – Batch Normalization

- Good to have features of similar range
- Input / hidden distributions change during learning (covariate shift)
- At least: control mean and variance
Learning Stability – Batch Normalization

- Good to have features of similar range
- Input / hidden distributions change during learning (covariate shift)
- At least: control mean and variance
- For each mini-batch (because it's cheap)
- Also: estimated mean $\mu_B$ and variance $\sigma_B^2$ are a bit noisy (regularization similar to dropout)
- $\gamma, \beta$ are parameters (and may counteract the normalization) that must be trained

Algorithm 1: Batch Normalizing Transform, applied to activation $x$ over a mini-batch.

Ioffe and Szegedy (2015)
Learning Stability – Batch Normalization

- Also: estimated mean $\mu_B$ and variance $\sigma_B^2$ are a bit noisy (regularization similar to dropout)
- $\gamma, \beta$ are parameters (and may counteract the normalization) that must be trained
- Exponentially smooth mean and variance with each batch:
  - $\bar{\mu} \leftarrow \alpha \bar{\mu} + (1 - \alpha) \mu_B$
  - $\bar{\sigma}^2 \leftarrow \alpha \bar{\sigma}^2 + (1 - \alpha) \sigma_B^2$
- During test (only one example):
  - Normalize with $\bar{\mu}, \bar{\sigma}^2$
- Higher learning rates might be possible now!

\[\text{Algorithm 1: Batch Normalizing Transform, applied to activation } x \text{ over a mini-batch.}\]

Ioffe and Szegedy (2015)
Inspecting a Deep Neural Net

- Occlusion experiments
Inspecting a Deep Neural Net

- Occlusion experiments
- Inspect the first layer of convolutions
- Pick a neuron and look what sample it likes most
- Feed random noise into the network
- Look for Dead ReLUs

From: nImage: A Python Plugin to Visualize Networks from Deep Models, Dongli Wei et al.
QUESTIONS? EXERCISES.