Deep Learning Software II

SP18 18739

Agenda

- Logistics
- Finish On CNN Architectures
  - Batch Normalization
- CNN Training
- CPU/GPU
- Tensorflow

Logistics

- HW2 will be out by the end of today
  - Due March 20th before class. Start Early!
  - You should have gotten an email about PSC
  - If you haven't sent me the Xsede username yet, please send me a private post
  - Cite your references
- OH cancelled today

Batch Normalization [paper]

- Distribution of each layer's inputs changes during training, as the parameters of the previous layers change
- Solution:
    - Normalizing for each batch
        - Output should have mean 0 and variance 1
    - Learn a scale parameter and shift parameter on the normalized value
        - Output should have mean alpha, variance beta
- In CNN, Batch Normalization Layer are typically placed before Non-linear Activation. (Topic of Debate)
- Regularize & Save time

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

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

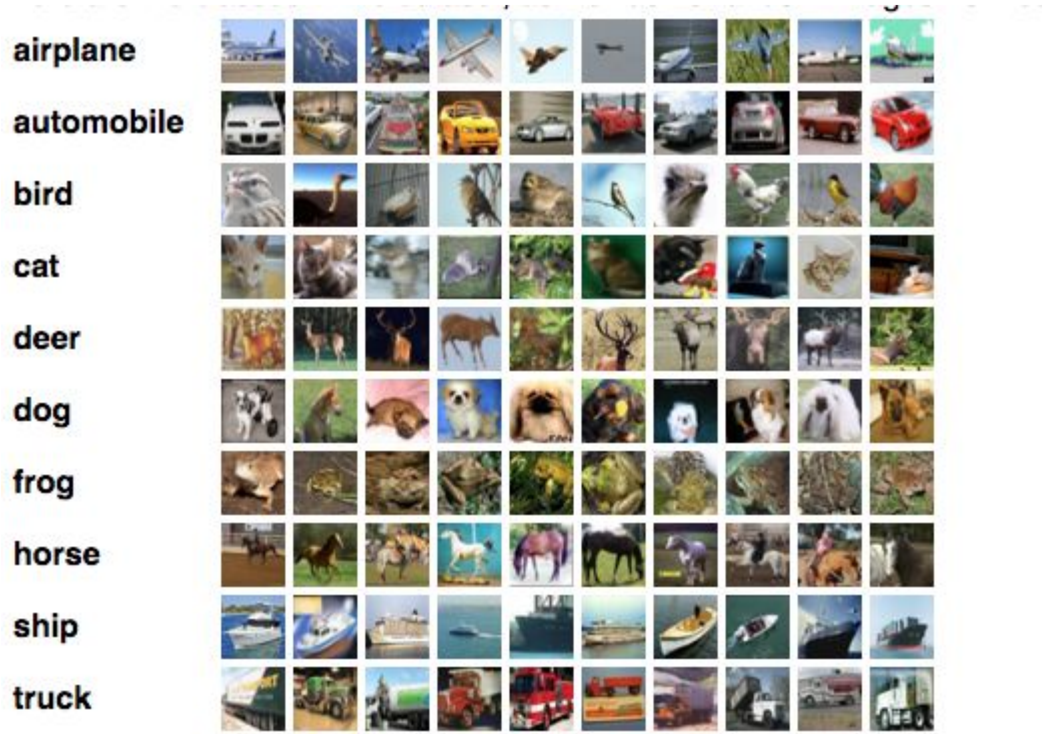
Image Classification Datasets

- MNIST
- CIFAR -10
  - 60000 32x32 colour images in 10 classes, with 6000 images per class
- CIFAR -100
  - 100 classes containing 600 images each
- ImageNet
  - Over 14 Million images, 20000 classes,most 256x256

Image Classification Datasets

- MNIST
- $\boxed{\text{CIFAR -10}}$
  - 60000 32x32 colour images in 10 classes, with 6000 images per class
- CIFAR -100
  - 100 classes containing 600 images each
- ImageNet
  - Over 14 Million images, 20000 classes,most 256x256

CIFAR-10


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

CNN Training

- Specify filters:
  - Number of filters, Filter size, Strides/zero padding
- Network architecture

– (conv-relu-pool)$\times$N $\to$ (affine)$\times$M $\to$ (softmax or SVM)

– (conv-relu-conv-relu-pool)$\times$N $\to$ (affine)$\times$M $\to$ (softmax or SVM)

– (batchnorm-relu-conv)$\times$N $\to$ (affine)$\times$M $\to$ (softmax or SVM)

CNN Training

- Regularization
  - Dropout
  - Batch Normalization
  - L2 weight regularization
- Optimizers
  - Adam/Rmsprop
- Activations
  - ReLus
  - Leaky Relus….

CNN Training: Parameter Tuning

- Lots of knobs: Architecture, layer params, optimizer params ….
  - Empirical evidence
- If the parameters are working well, you should see improvement within a few hundred iteration
- Coarse-to-fine approach
  - Start by large range + a few training iterations to find the combinations
  - Search more finely + more iterations

CNN Training

- Train, Validate and Test!
  - Test Data can only be seen once!
  - Don’t let your test result steer your training
- Time / Accuracy Trade-off

CPU vs. GPU

	# Cores	Clock Speed	Memory	Price
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.4 GHz	Shared with system	$339
CPU (Intel Core i7-6950X)	10 (20 threads with hyperthreading)	3.5 GHz	Shared with system	$1723
GPU (NVIDIA Titan Xp)	3840	1.6 GHz	12 GB GDDR5X	$1200
GPU (NVIDIA GTX 1070)	1920	1.68 GHz	8 GB GDDR5	$399

Spot the CPU!

Spot the GPU!

CPU vs. GPU

- **CPU**: Fewer cores, but each core is much faster and much more capable; great at sequential tasks
- **GPU**: More cores, but each core is much slower and “dumber”; great for parallel tasks

Tensorflow

```
# clear old variables
tf.reset_default_graph()

# setup input (e.g. the data that changes every batch)
# The first dim is None, and gets sets automatically based on batch size fed in
X = tf.placeholder(tf.float32, [None, 32, 32, 3])
y = tf.placeholder(tf.int64, [None])
is_training = tf.placeholder(tf.bool)

def model1(X,y):
    # define our weights (e.g. init_two_layer_convnet)

    # setup variables
    Wconv1 = tf.get_variable("Wconv1", shape=[7, 7, 3, 32])
    bconv1 = tf.get_variable("bconv1", shape=[32])
    W1 = tf.get_variable("W1", shape=[5408, 10])
    b1 = tf.get_variable("b1", shape=[10])

    # define our graph (e.g. two_layer_convnet)
    a1 = tf.nn.conv2d(X, Wconv1, strides=[1,2,2,1], padding='VALID') + bconv1
    h1 = tf.nn.relu(a1)
    h1_flat = tf.reshape(h1,[-1,5408])
    y_out = tf.matmul(h1_flat,W1) + b1
    return y_out

y_out = model1(X,y)

# define our loss
total_loss = tf.losses.hinge_loss(tf.one_hot(y,10),logits=y_out)
mean_loss = tf.reduce_mean(total_loss)

# define our optimizer
optimizer = tf.train.AdamOptimizer(5e-4) # select optimizer and set learning rate
train_step = optimizer.minimize(mean_loss)
```