AlexNet Notes

Sean Wu

September 30, 2019

Contents

1 Intro

1.1 Specs

- 60 M params
- 650 K neurons
- 5 CONV and 3 FC layers
- Final 1000-way softmax layer
- 5-6 days of training on two GTX 580 3GB GPUs

1.2 Dataset: ImageNet

- ImageNet has $>$ 15 M labelled high res images w/ 22 K categories
- AlexNet trained on ILSVRC subset (1000 images in 1000 categories)
  - 1.2 M training images
  - 50 K validation images
  - 150 K testing images
- 2 main KPIs

1. Top-1 error: error rate for most probable label according to model
2. Top-5 error: fraction of test images for which correct label is not in model's top 5 predictions

- Since ImageNet has variable-res images, images were down-sampled to 256×256
  - AlexNet req constant input dimensionality
  - Images rescaled s.t. shorter side was $L = 256$, then cropped out central 256×256 image from resulting image

2 Architecture

2.1 ReLU Nonlinearity

Standard model of neuron output $f(x) = tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$ (saturating nonlinearities)

ReLU $f(x) = max(0, x)$ (non-saturating nonlinearity)

- ReLU units train much faster than *tanh* units

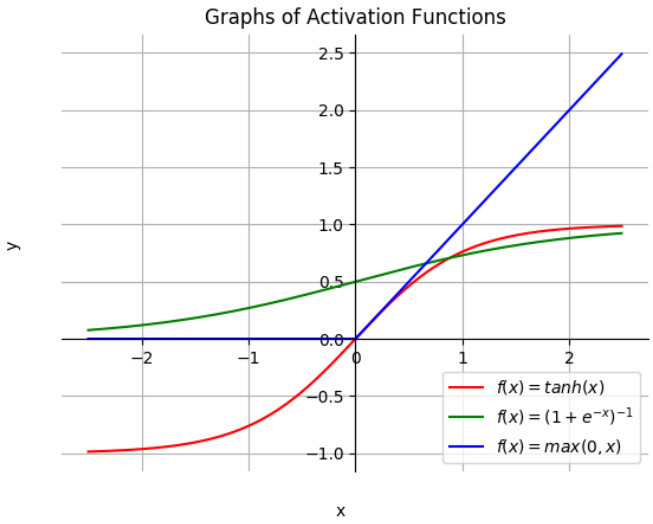

Figure 1: Comparison of ReLU vs other activation functions

2.2 Multiple GPU Training

- Current GPUs good at cross-GPU parallelization
  - Can read/write to another's memory directly w/out going thru host machine memory
- AlexNet splits kernels (or neurons) 50/50 across 2 GPUs
- AlexNet GPUs only communicate on certain layers which reduces error rates and training time

2.3 Local Response Normalization (Brightness Normalization)

- ReLUs good bec do not req input normalization to prevent saturation
- If at least some training examples produce input to a ReLU, learning will happen in that neuron
- Adding a local normalization scheme helps generalization

Response-normalized activity $b^i_{x,y}$ is given by:

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=max(0,1-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta} \tag{1}$$

where $a^i_{x,y}$ is activity of a neuron computed by applying kernel i at position (x, y) and then applying ReLU nonlinearity and N is total number of kernels in layer. The sum runs over n "adjacent" kernel maps at same spatial position,

- Constants k, n, α, and β are hyperparameters determined using validation set

2.4 Overlapping Pooling

- Pooling layers in CNNs summarize outputs of groups of neurons in same kernel map
- Pool layer like grid of pooling units spaced s units apart, each summarizing a $z \times z$ neighborhood centered at location of unit

1. if $s = z$: traditional local pooling
2. if $s < z$: overlapping pooling

- AlexNet uses overlapping pooling w/ $s = 2$ and $z = 3$
  - Slightly reduced error and harder to overfit

2.5 Layers Architecture

- INPUT $\longrightarrow$ CONV $\longrightarrow$ 3 FC $\longrightarrow$ 1000-way softmax
- Maximizes multinomial logistic regression objective
  - Equivalent to maximizing average across training cases of log-probability of correct label under prediction distribution
- Kernels of 2nd, 4th, 5th CONV layers only connected to kernel maps in prev layer on same GPU
  - Kernels in 3rd GPU fully connected to all kernel maps in 2nd layer
- Response-normalization follow 1st and 2nd CONV layers
  - 3rd, 4th, 5th layers connected w/ no pooling or normalization layers
- Max pooling follow both response-normalization and 5th CONV
- ReLU non-linearity applied to output of all CONV and FC

2.5.1 Diagrams of AlexNet Layers

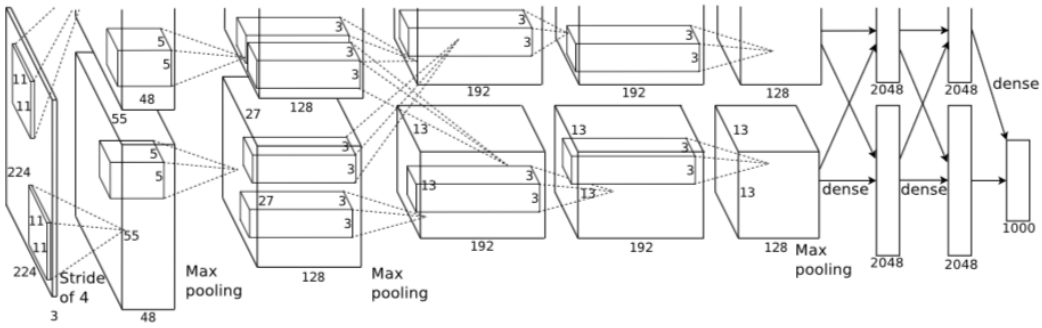

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Figure 2: Diagram showing AlexNet layers split by GPU

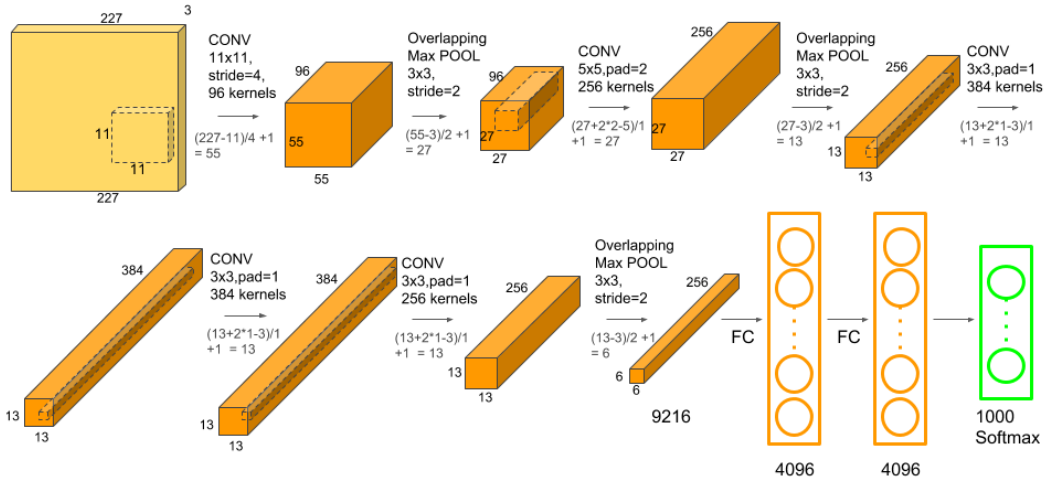

Figure 3: Diagram showing AlexNet layers with filter sizes and strides

3 Reducing Overfitting

3.1 Data Augmentation

- Artificially enlarged dataset w/2 forms of label-preserving transformations
- Transformed images generated on CPU while GPU training on previous set of training images

– No need to store transformed images
– Effectively computationally free

3.2 Data Augmentation Methods

1. Image translations and reflections

- Extract random 224×224 patches (sometimes horizontal reflected) from 256×256 images and train on extracted patches
- Increases training set size by factor of 2048, but resulting traning examples are very inter-dependent
- At test time, network makes prediction by selecting 10 patches (4 corners + 1 centre + 5 reflections) and averaging predictions on patches using softmax

2. Altering RGB channel intensities

- Perform PCA (principal component analysis) on set of RGB pixel values
- Add multiples of principal components w/ magnitudes proportional to corresponding eigenvalues times a random var from Gaussian w/ mean 0 and standard dev 0.1
- Thus for each RGB image pixel $I_{xy} = \left[I_{xy}^R, I_{xy}^G, I_{xy}^B\right]^T$, add:

$$\left[\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}\right] \left[\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3\right]^T \tag{2}$$

  where $\mathbf{p_i}$ and λ_i are the ith eigenvector and eigenvalue of the 3×3 covariance matrix of RGB pixel values and α_i is the random variable

- Transformation works because object identity is invariant to changes in intensity and color of illumination

3.3 Dropout

- Can reduce test errors by combining predictions of different models, but very time expensive

Dropout zeroing output of each hidden neuron w/ probability 0.5

- Very efficient model combination method, but doubles # of iterations req to converge

- Dropped out neurons do not contribute to forward pass or backpropagation
- For every new input, neural net samples a different architecture, but all architectures share weights
- Reduces complex co-adaptations of neurons since a neuron cannot rely on presence of particular neurons
  - Neurons force to learn more robust features that are useful in conjunction w/ other random subsets of neurons
- At test time, use all neurons but multiply outputs by 0.5 (approximation of taking geometric mean of predictive distributions produced by exponentially many dropout networks)
- Use dropout in first 2 FC

4 Details of Learning

- AlexNet trained using gradient descent w/ batch size of 128, momentum of 0.9, and weight decay of 0.0005
  - Small weight decay reduces training error
- Weights initialized from zero-mean Gaussian distribution w/ standard deviation of 0.01
- Update rule for weight w was:

$$v_{i+1} \coloneqq 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i} \tag{3}$$

$$w_{i+1} \coloneqq w_i + v_{i+1} \tag{4}$$

  where i is the iteration index, v is the momentum, ϵ is the learning rate, and $\left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$ is the average over the ith batch D_i of the derivative of the objective wrt w, evaluated at w_i

- Neuron biases in 2nd, 4th, and 5th layers & FC layers initialized w/ constant 1
  - This initialization speeds up early learning stages by providing ReLUs w/ positive input
- Neuron biases in other layers initialized w/ constant 0
- Used equal learning rate for all layers, and adjusted manually through training
- Adjustment heuristic: divide learning rate by 10 when validation error stopped improving
- Learning rate initialized at 0.01 and reduced 3 times before termination