CAP5415
Computer Vision

Yogesh S Rawat

yogesh@ucf.edu

HEC-241
Training Neural Networks- II

Lecture 7
Network Parameters - recap

32x32x3 image

3x3x3 filter

Convolve over all spatial locations

Activation map (feature map)
Visualizing CNN

Source: http://cs231n.github.io
Learning phases - recap

**Training**
- Images
- Image Features
- Training
- Trained classifier

**Testing**
- Image (not in training set)
- Image Features
- Apply classifier
- Prediction

Slide credit: D. Hoiem and L. Lazebnik
Loss Function - recap

• Way to define how good the network is performing
  • In terms of prediction

• Network training (Optimization)
  • Find the best network parameters to minimize the loss

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) \]
Train CNN with Gradient Descent

- $x^i, y^i = n$ training examples
- $f(x) = \text{feed forward network}$
- $L(x, y; \theta) = \text{some loss function}$

*Loss function* measures how ‘good’ our network is at classifying the training examples wrt. the parameters of the model (the perceptron weights).
Train CNN with Gradient Descent

Loss function
(Evaluate CNN on training data)

Model parameters
(network weights)
Loss Functions

• Cross entropy

\[-\frac{1}{N} \sum_{i=1}^{N} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))\]

• Mean squared error (MSE)

\[\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2\]
Differentiability

• Loss function
• Activation function
• Convolution
• Pooling
• ...
Backpropagation – Chain Rule

- Chain rule \( \frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial x} \)

**Forwardpass**

- \( f(x, y) \)

**Backwardpass**

- \( df \)
- \( \frac{dl}{dx} = \frac{dl}{dz} \frac{dz}{dx} \)
- \( \frac{dl}{dy} = \frac{dl}{dz} \frac{dz}{dy} \)
Backpropagation – Chain Rule

- Add gate:
  - gradient distributor
- Max gate:
  - gradient router
- Mul gate:
  - gradient switcher

\[
\frac{\partial E}{\partial w_{ij}} = a_i \delta_j \\
\frac{\partial E}{\partial w_{jk}} = a_j \delta_k
\]
Optimization demo

• http://www.emergentmind.com/neural-network

• Thank you Matt Mazur
Stochastic Gradient Descent

• Dataset can be too large
  • Can not apply gradient descent wrt. all data points.

• Randomly sample a data point
  • Perform gradient descent per sample and iterate.
  • Picking a subset of points: “mini-batch”

Randomly initialize starting $W$ and pick learning rate $\gamma$

While not at minimum:
  • Shuffle training set
  • For each data point $i=1...n$ (maybe as mini-batch)
    • Gradient descent
Stochastic Gradient Descent

- Loss will not always decrease (locally)
  - As training data point is random.

- Still converges over time.
Gradient descent oscillations
Gradient descent oscillations

Slow to converge to the (local) optimum
Momentum

• Adjust the gradient by a weighted sum of the previous amount plus the current amount.

• Without momentum: \[ \theta_{t+1} = \theta_t - \gamma \frac{\partial L}{\partial \theta} \]

• With momentum (new \(\alpha\) parameter):

\[
\theta_{t+1} = \theta_t - \gamma \left( \alpha \left[ \frac{\partial L}{\partial \theta} \right]_{t-1} + \left[ \frac{\partial L}{\partial \theta} \right]_t \right)
\]
Lowering the learning rate

-Takes longer to get to the optimum
Learning rate

\[ a_1 \quad a_2 \quad a_3 \quad a_{\text{stop}} \]

- \text{very high learning rate}
- \text{low learning rate}
- \text{high learning rate}
- \text{good learning rate}
Problem of fitting

• Too many parameters = overfitting
• Not enough parameters = underfitting
• More data = less chance to overfit

• How do we know what is required?
Regularization

• Attempt to guide solution to *not overfit*
• But still give freedom with many parameters
Data fitting problem

[Nielson]
Which is better?

9th order polynomial

1st order polynomial

[Neilson]
Data fitting problem
Data fitting problem

- Underfitting
- Overfitting
Data fitting problem

- Early stopping
- Regularization
- Dropout
- ...

Fighting Overfitting

Data fitting problem
Early stopping
Regularization

• Attempt to guide solution to *not overfit*
• But still give freedom with many parameters

• Idea:
  *Penalize the use of parameters to prefer small weights.*
Regularization

• Add a cost to having high weights

\[
L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)
\]

• In common use,
  - L1 Norm – \( R(W) = \sum_i \sum_j |W_{ij}| \)
  - L2 Norm – \( R(W) = \sum_i \sum_j W_{ij}^2 \)
  - Elastic net – \( R(W) = \sum_i \sum_j \beta W_{ij}^2 + |W_{ij}| \)
Regularization: Dropout

Our networks typically start with random weights. Every time we train, slightly different outcome.

- Why random weights?
- If weights are all equal, response across filters will be equivalent.
  - Network doesn’t train.
Regularization

Our networks typically start with random weights.
Every time we train = slightly different outcome.

• Why not train 5 different networks with random starts and vote on their outcome?
  • Works fine!
  • Helps generalization because error due to overfitting is averaged; reduces variance.
Dropout

• Stochastically switch neurons off
  • Each neuron is set to 0 with probability $p$
  • Hidden units cannot co-adapt to each other
  • Units are useful independently

• Hyperparameter
  • $P$ is usually set to 0.5
Training steps

• Define network
• Loss function
• Initialize network parameters
• Get training data
  • Prepare batches
• Feedforward one batch
  • Compute loss
  • Backpropagate gradients
  • Update network parameters
  • Repeat
AlexNet - Training

- Parameters:
  - First use of ReLU
  - Dropout 0.5
  - Batch size 128
  - Optimizer SGD
  - Momentum 0.9
  - Learning rate 1e-2
  - Decay – lr reduced by 10 manually when val accuracy plateaus
  - L2 weight decay 5e-4
CNN Variants
Residual Networks

- Deep networks performs worse
  - As we add more layers

- Problem
  - Vanishing gradients

- It models
  - $H(x) = F(x) + x$

- Skip connections
  - Help in backpropagation

*He et. al. Deep Residual Learning for Image Recognition, 2015*
GoogleNet - Inception

• ResNet is about going deeper
• Inception is about going wider
• Focused on computational efficiency
• The network learns
  • Which features are useful

*Going deeper with convolutions, Szegedy et al. (2014)*
*Inception-v4, Szegedy et al. (2016)*
DenseNet

• Densely Connected Convolutional Network
  • Similar to ResNet
  • Concat features instead of summation
  • A layer is passed all previous maps
  • Sequence of dense blocks

Questions?

Sources for this lecture include materials from works by Abhijit Mahalanobis, James Tompkin, and Fei Fei Li