Deep Network Models

CS6780 – Advanced Machine Learning
Spring 2019

Thorsten Joachims
Cornell University

Reading: Murphy 16.5
https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

Discriminative Training of ~~Linear~~ Rules

$$\min_{w,b} R(w) + C \frac{1}{n} \sum_{i=1}^{n} \Delta(\sout{w * x_i + b}, y_i)$$

$f(x_i|w)$

Regularizer

Regularization Parameter

Training Loss / Empirical Risk / Training error

- Soft-Margin SVM
  - $R(w) = \frac{1}{2} w * w$
  - $\Delta(\bar{y}, y_i) = \max(0, 1 - y_i \bar{y})$
- Perceptron
  - $R(w) = 0$
  - $\Delta(\bar{y}, y_i) =$
- Linear Regression
  - $R(w) = 0$
  - $\Delta(\bar{y}, y_i) = (y_i - \bar{y})^2$
- Ridge Regression
  - $R(w) = \frac{1}{2} w * w$
  - $\Delta(\bar{y}, y_i) = (y_i - \bar{y})^2$
- Lasso
  - $R(w) = \frac{1}{2} \sum |w_i|$
  - $\Delta(\bar{y}, y_i) = (y_i - \bar{y})^2$
- Regularized Logistic Regression / Conditional Random Field
  - $R(w) = \frac{1}{2} w * w$
  - $\Delta(\bar{y}, y_i) = \log(1 + e^{-y_i \bar{y}})$

Discriminative Training of Non-Linear Rules

$$\min_{w,b} R(w) + C\frac{1}{n}\sum_{i=1}^{n}\Delta(f(x_i|w), y_i)$$

Regularizer

Regularization Parameter

Training Loss / Empirical Risk / Training error

Options for $f(x_i|w)$:

- Kernelized linear functions $w \cdot \phi(x) + b$
  - Convex for L2 regularization → stochastic gradient descent
- Linear combinations of trees $\sum_j w_j\, DecTree_j(x)$
  - Special Boosting algorithms
- Deep Networks
  - Not convex, but stochastic gradient descent anyway

Naïve Two-layer Perceptron

Idea: $f(x|W)$ by stacking two layers perceptrons on top of each other.

- First layer: k perceptrons with
$$a = \begin{pmatrix} w_1 \cdot x + b_1 \\ \vdots \\ w_k \cdot x + b_k \end{pmatrix}$$
- Second layer: 1 perceptron with
$$f(x|w_0, w_1, \ldots, w_k) = (w_0 \cdot a + b)$$

→ Need nonlinearity $\sigma(w_i \cdot x + b_i)$

Two-layer Perceptron

Use nonlinear activation function σ:

- First layer: k perceptrons (aka hidden units) with

$$a = \begin{pmatrix} \sigma(w_1 \cdot x + b_1) \\ \vdots \\ \sigma(w_k \cdot x + b_k) \end{pmatrix}$$

- Final layer: 1 perceptron (aka output unit) with

$$f(x|w) = (w_0 \cdot a + b_0)$$

Choices for $\sigma(p)$

- Sigmoid: $\tanh(\mathrm{p})$
- Gaussian: $\exp(-\mathrm{p}^2)$
- ReLU: $\max(0, \mathrm{p})$
- SoftPlus: $\log(1 + e^p)$

Multi-layer Perceptron

Keep stacking layers with non-linear activation functions:

- First layer: k perceptrons with

$$a_0 = \begin{pmatrix} \sigma(w_{01} \cdot x + b_1) \\ \vdots \\ \sigma(w_{0k} \cdot x + b_k) \end{pmatrix} = \sigma(\mathrm{W}_0 \cdot x + b_0)$$

- d hidden layers: k perceptrons with

$$a_l = \begin{pmatrix} \sigma(w_{l1} \cdot a_{l-1} + b_{l1}) \\ \vdots \\ \sigma(w_{lk} \cdot x_{l-1} + b_{lk}) \end{pmatrix} = \sigma(W_l \cdot a_{l-1} + b_l)$$

- Final layer: 1 perceptron with

$$f(x|w) = (w_0 \cdot a_d + b_0)$$

Optimization Problem

Problem: Training optimization problem

$$\min_{W,B} R(W) + C \frac{1}{n} \sum_{i=1}^{n} \Delta(f(x_i|W,B), y_i)$$

is not convex! → local optima.

Algorithm:

- Stochastic Gradient Descent (SGD)
- Efficient via Backpropagation Algorithm

Gradient Descent

Optimization Problem:

$$\min_{W} R(W) + C\frac{1}{n}\sum_{i=1}^{n}\Delta(f(x_i|W), y_i)$$

Gradient Descent Algorithm

- REPEAT
  - Compute gradient ∇W

$$\nabla \mathrm{W} = \frac{\partial R(W)}{\partial W} + C\frac{1}{n}\sum_{i=1}^{n}\frac{\partial \Delta(f(x|W), y_i)}{\partial W}$$

  - Update weights $W = W - \alpha \nabla W$

Stochastic Gradient Descent

Idea:

- Computation of gradient is expensive (full pass)
- Replace gradient with cheaper approximation

Gradient Descent Algorithm

- REPEAT
  - Draw random subsample M of training examples
  - Approximate gradient ∇W

$$\nabla \mathrm{W} = \frac{\partial R(W)}{\partial W} + C \frac{1}{|M|} \sum_{i \in M} \frac{\partial \Delta(f(x|W), y_i)}{\partial W}$$

  - Update weights $W = W - \alpha \nabla W$

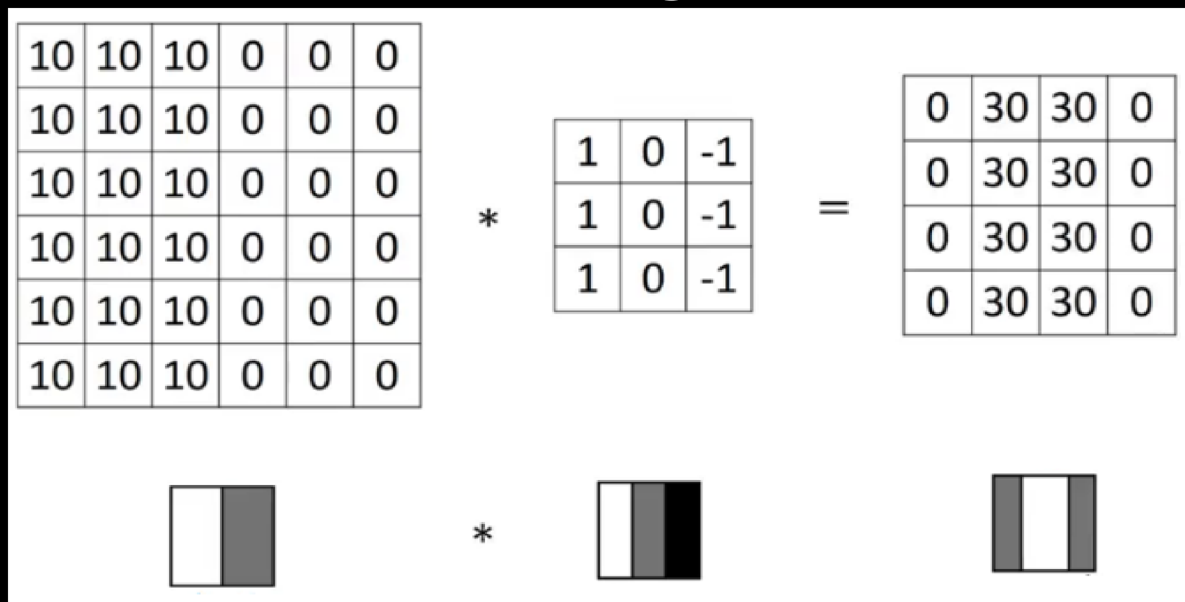
Optimization Issues and Tricks

Tricks:

- Normalize input features (e.g. standardize to zero mean and variance one)
- Batch normalization to normalize intermediate layers
- Use Momentum
- Reduce stepsize as training progresses
- Minibatches reduce variance of gradient

Convolutions

- Local filter that detects higher-order features

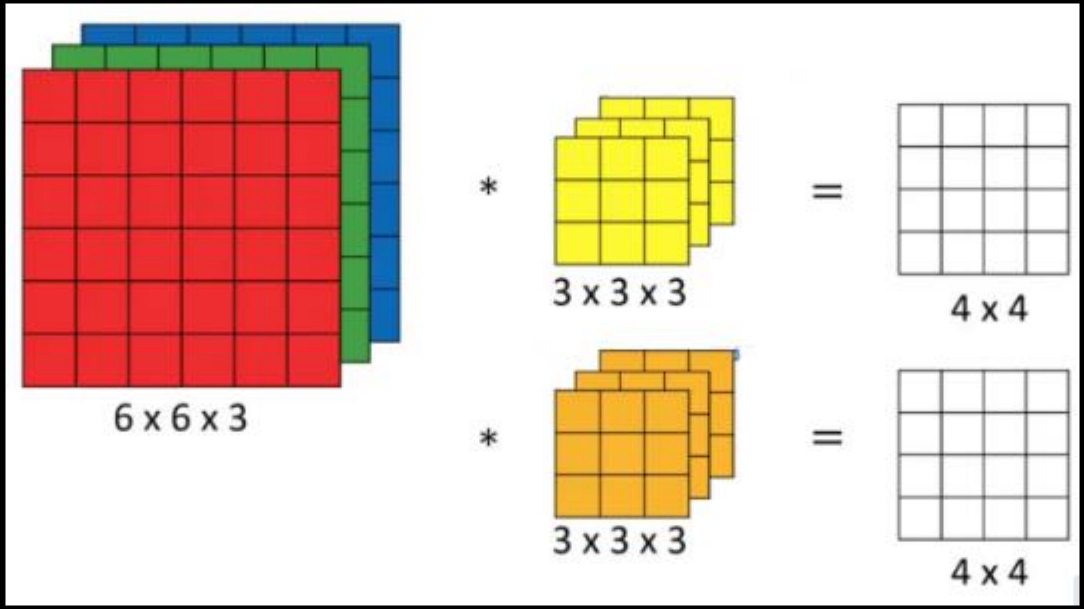


- Stride: Offset by which filter is moved
- Padding: Border to ensure size does not shrink

https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

Convolutions over Volumes

- Summing over multi-dimensional inputs

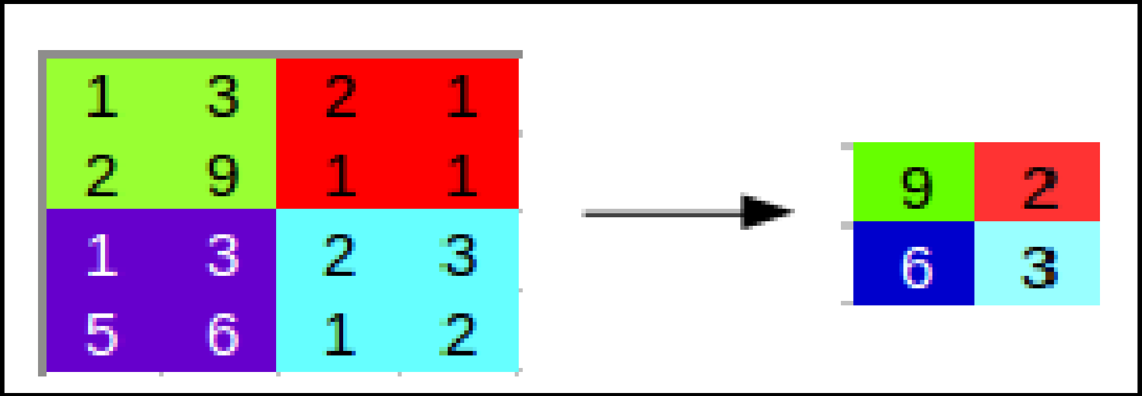


- Each filter creates one output dimension

https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

Pooling Layers

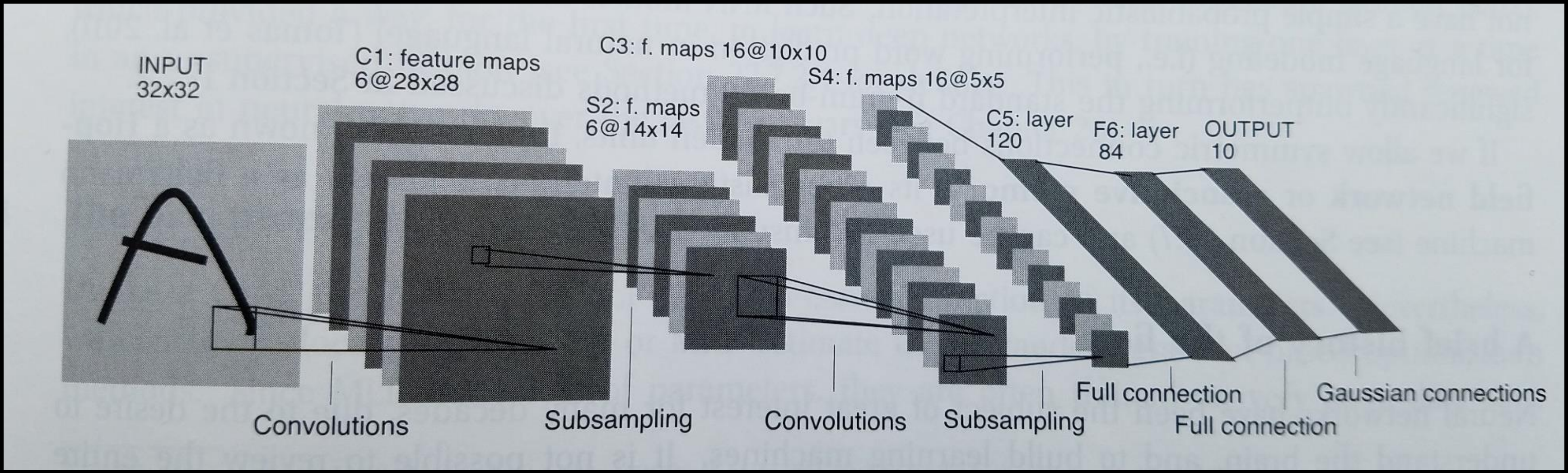
- Reduce input size



- Size of pooling area
- Stride
- Aggregation: max or average pooling

https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

LeNet5 for Vision



Murphy Figure 16.14

Other architectures

- AlexNet
- VGG
- ResNet
- DenseNet