Introduction to ConvNets

CSC2541, 2017 Winter
Bin Yang
16 Jan. 2017

slides adopted from Raquel Urtasun, Geoffrey Hinton, A. G. Schwing, Kaiming He, Stanford CS231n and many others
Big Picture

ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.

MACHINE LEARNING
Machine learning begins to flourish.

DEEP LEARNING
Deep learning breakthroughs drive AI boom.

Pic credit: NVIDIA blog
Success of Deep Learning
Deep Learning in Vision

ImageNet Classification top-5 error (%)

- ILSVRC'15 ResNet: 3.57
- ILSVRC'14 GoogleNet: 6.7
- ILSVRC'14 VGG: 7.3
- ILSVRC'13: 11.7 (8 layers)
- ILSVRC'12 AlexNet: 16.4 (8 layers)
- ILSVRC'11: 25.8 (shallow)
- ILSVRC'10: 28.2

Total layers: 152

Pic credit: Kaiming He
Deep Learning in Vision

PASCAL VOC 2007 **Object Detection** mAP (%)
What is Deep Learning?

The goal of supervised deep learning is to solve almost any problem of the form “map $x$ to $y$”. $x$ can include images, speech, or text, and $y$ can include categories or even sentences. Mapping images to categories, speech to text, text to categories, go boards to good moves, and the like, is extremely useful, and cannot be done as well with other methods.

An attractive feature of deep learning is that it is largely domain independent: many of the insights learned in one domain apply in other domains.

Under the hood, the model builds up layers of abstraction. These abstractions get the job done, but it’s really hard to understand how exactly they do it. The model learns by gradually changing the synaptic strengths of the neural network using the incredibly simple yet mysteriously effective backpropagation algorithm. As a result, we can build massively sophisticated systems using very few lines of code (since we only code the model and the learning algorithm, but not the end result).

Quote from Ilya Sutskever
Introduction to ConvNets

• Some Deep Learning figures

• Neural Networks
  • Architecture
  • Forward pass (inference)
  • Backward pass (learning)
  • Optimization

• Convolutional Neural Networks
  • Architecture
  • Feature maps

• TensorFlow demo
Introduction to ConvNets

• Some Deep Learning figures

• Neural Networks
  • Architecture
  • Forward pass (inference)
  • Backward pass (learning)
  • Optimization

• Convolutional Neural Networks
  • Architecture
  • Feature maps

• TensorFlow demo
What are neural networks?

...**Neural networks** (NNs) are computational models inspired by biological neural networks [...] and are used to estimate or approximate functions... [Wikipedia]
Activation functions / Nonlinearity

- **Sigmoid**: \( f(x) = \frac{1}{1+e^{-x}} \)
- **Tanh**: \( f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \)
- **ReLU (Rectified Linear Unit)**: \( f(x) = \max(0, x) \)

[Graphs of Sigmoid, Tanh, and ReLU functions]

Slide credit: Raquel Urtasun
Neural Network (Multi-Layer Perception)

The network approximates the function:
\[ y = f(x; w) \]
which can be de-composed as:
\[ h = g(w_1 * x + b_1) \]
\[ y = g(w_2 * h + b_2) \]

Naming convention: a 2-layer neural network
• 1 layer of hidden units
• 1 output layer
(we do not count the inputs as a layer)
Representational power

• One node is controlled by two parameters \( w, b \)

\[ y = f(w_1 \times x + b) \]

where the activation function is sigmoid

\[ f(x) = \frac{1}{1+\exp(-x)} \]

Slide credit: A. G. Schwing
Representational power

- One node is controlled by two parameters $w, b$
- We can get a bump function given a pair of nodes

Slide credit: A. G. Schwing
Representational power

- One node is controlled by two parameters $w, b$
- We can get a bump function given a pair of nodes
- Given more bumps, we get more accurate approximation

Slide credit: A. G. Schwing
Representational power

- One node is controlled by two parameters $w, b$
- We can get a bump function given a pair of nodes
- Given more bumps, we get more accurate approximation
- Neural network with **at least one hidden layer** is a universal function approximator

Proof in: Approximation by Superpositions of Sigmoidal Function, Cybenko

Pic credit: Stanford CS231n
Slide credit: Raquel Urtasun
Representational power

- One node is controlled by two parameters $w$, $b$
- We can get a bump function given a pair of nodes
- Given more bumps, we get more accurate approximation
- Neural network with at least one hidden layer is a universal function approximator

Proof in: Approximation by Superpositions of Sigmoidal Function, Cybenko

- The capacity of the network increases with more hidden units and more hidden layers

Slide credit: Raquel Urtasun
Introduction to ConvNets

- Some Deep Learning figures
- Neural Networks
  - Architecture
  - Forward pass (inference)
  - Backward pass (learning)
  - Optimization
- Convolutional Neural Networks
  - Architecture
  - Feature maps
- TensorFlow demo
Forward pass

Efficient implementation via matrix operations.

\[ h_j(x) = f(v_{j0} + \sum_{i=1}^{D} x_i v_{ji}) \]

\[ o_k(x) = g(w_{k0} + \sum_{j=1}^{J} h_j(x) w_{kj}) \]

x: 3-d vector                y: 1-d vector
h1: 4-d vector              h2: 4-d vector
W1: 4x3 matrix             b1: 4-d vector
W2: 4x4 matrix             b2: 4-d vector
W3: 1x4 matrix             b3: 1-d vector

Pic credit: Stanford CS231n

# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
Introduction to ConvNets

- Some Deep Learning figures
- Neural Networks
  - Architecture
  - Forward pass (inference)
  - Backward pass (learning)
  - Optimization
- Convolutional Neural Networks
  - Architecture
  - Feature maps
- TensorFlow demo
Back-propagation algorithm

An intuitive explanation:

• Compute approximation error at the output
• Propagate error back by computing individual contributions of parameters to error
Loss function

Classification
• Cross-entropy: \( \text{sum}_i(-y_i \times \log(f(x_i))) \)
• Hinge loss: \( \max(0, 1 - y_i \times f(x_i)) \)

Regression
• L1: \( \text{sum}_i(|y_i - f(x_i)|) \)
• L2: \( \text{sum}_i((y_i - f(x_i))^2) \)

Pair-wise similarity
• Contrastive loss: \( E = \frac{1}{2N} \sum_{n=1}^{N} (y) d^2 + (1 - y) \max(margin - d, 0)^2 \)
• Triplet loss: \( \sum_i \left[ \|f(x^a_i) - f(x^p_i)\|_2^2 - \|f(x^a_i) - f(x^n_i)\|_2^2 + \alpha \right]^+ \)
How do we update $w_{ki}$ to minimize the loss?
Use gradient descent!

Update rule:

\[ w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}} \]
Compute gradient: chain rule

- L2 loss
- \( g(z) = 1/(1+\exp(-z)) \)

\[
\frac{\partial E}{\partial w_{ki}} = \sum_{n=1}^{N} \frac{\partial E}{\partial o_k^{(n)}} \frac{\partial o_k^{(n)}}{\partial z_k^{(n)}} \frac{\partial z_k^{(n)}}{\partial w_{ki}} = \sum_{n=1}^{N} (o_k^{(n)} - t_k^{(n)}) o_k^{(n)} (1 - o_k^{(n)}) x_i^{(n)}
\]

\[
\frac{\partial E}{\partial o_k^{(n)}} = o_k^{(n)} - t_k^{(n)} := \delta_k^o
\]

\[
\frac{\partial o_k^{(n)}}{\partial z_k^{(n)}} = o_k^{(n)} (1 - o_k^{(n)})
\]

\[
w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}}
\]
If a node has multiple outputs, we have to sum over all gradients from these paths back to that node.

\[
\frac{\partial E}{\partial h_j^{(n)}} = \sum_k \frac{\partial E}{\partial o_k^{(n)}} \frac{\partial o_k^{(n)}}{\partial z_k^{(n)}} \frac{\partial z_k^{(n)}}{\partial h_j^{(n)}} = \sum_k \delta_k^{z, (n)} w_{kj} := \delta_j^{h, (n)}
\]
Introduction to ConvNets

- Some Deep Learning figures
- Neural Networks
  - Architecture
  - Forward pass (inference)
  - Backward pass (learning)
  - Optimization (bag of tricks)
- Convolutional Neural Networks
  - Architecture
  - Feature maps
- TensorFlow demo
The back-propagation algorithm is an efficient way of computing the error derivative $\frac{dE}{dw}$ for every weight on a single training case. However, we still need to make other decisions about how to use these error derivatives:

- Optimization issues
  - how often to update the weights
  - how much to update the weights
- Ways to reduce overfitting
Batch size

How often to update the weights:

- **Online**: after each training case
- **Full batch**: after a full sweep through the training data
- **Mini-batch**: after a small sample of training cases

- Theoretically, we should do **full batch** update, but the computation is expensive.
- When the dataset is highly redundant, we can get a good estimate of the gradient by computing only a subset of samples. The extreme version of this is ‘**online**’.
- **Mini-batch** is a good trade-off. The computation for many cases simultaneously can be implemented efficiently using matrix-matrix multiplies on GPUs.
- **Mini-batches** need to be balanced for classes.

Slide credit: Geoffrey Hinton
Learning rate

- Don’t start too big, and not too small.
- Start as big as you can without diverging, then when getting to a plateau start reducing the learning rate. Be careful not to reduce the learning rate too early.
Momentum

**Intuition:** imagine a ball falling down along the hill of loss surface. Giving the ball velocity would make it more likely to get out of local minima.

```python
# Momentum update
v = mu * v - learning_rate * dx  # integrate velocity
x += v  # integrate position
```

Pic credit: Stanford CS231n
Different optimizers

Different convergence speed. Notice the over-shooting of momentum based methods.

A visualization of saddle point. SGD has a very hard time breaking symmetry and gets stuck on top. RMSprop will see very low gradients in the saddle direction.

Pic credit: Stanford CS231n
Data preprocessing

Normalization

PCA/whitening

Pic credit: Stanford CS231n
Weight initialization

Why we shouldn’t use all 0 initialization: if two neurons are initialized with the same weights, they will give the same output, get the same gradient and update, and therefore they will always be the same.

Random initialization from Gaussian: symmetry breaking. However, the distribution of the outputs from a randomly initialized neuron has a variance that grows with the number of inputs.

Random initialization from Gaussian/sqrt(n): where n is the number of the neuron’s inputs.

Best practice: ReLU units with Gaussian*sqrt(2/n) (He et al.)

Batch normalization (Ioffe & Szegedy): normalize the activations through a network to take on a unit gaussian distribution

Slide credit: Stanford CS231n
Prevent overfitting

1. Get more data!
2. Use L2 regularization on weights

\[ E(\vec{w}) = \frac{1}{2} \sum_{n=0}^{N-1} (t_n - y(x_n, \vec{w}))^2 + \frac{\lambda}{2} ||\vec{w}||^2 \]

The effects of regularization strength.

Pic credit: Stanford CS231n
Prevent overfitting

1. Get more data!
2. Use L2 regularization on weights
3. Dropout (Srivastava et al.)

Training time: keep a neuron active with probability p
Testing time: keep all neurons active but scale their activations by p
Introduction to ConvNets

- Some Deep Learning figures
- Neural Networks
  - Architecture
  - Forward pass (inference)
  - Backward pass (learning)
  - Optimization (bag of tricks)
- Convolutional Neural Networks
  - Architecture
  - Feature maps
- TensorFlow demo
Motivation

- Dimension of image data is usually large.
- We want our representation to be translation-invariant.

Pic credit: Markus, ECCV14
Convolutional layer (local connectivity + weight sharing)

- Fully connected layer
- Local connectivity
- Spatial weight-sharing

Pic credit: Stanford CS231n & Geoffrey Hinton
Convolution operation on 2D data

**Param:** filter size, stride

Pic credit: Stanford CS231n & UFLDL & A. G. Schwing
Pooling layer

- **Types:**
  - Max-pooling
  - Average-pooling
- **Advantages:**
  - Reduce representation dimensionality
  - Robustness against tiny shifts

**Param:** pool size, stride

Pic credit: Stanford CS231n
An example ConvNet architecture
Revolution of depth

AlexNet, 8 layers (ILSVRC 2012)

- 11x11 conv, 96, /4, pool/2
- 5x5 conv, 256, pool/2
- 3x3 conv, 384
- 3x3 conv, 384
- 3x3 conv, 256, pool/2
- fc, 4096
- fc, 4096
- fc, 1000

Pic credit: Kaiming He
Revolution of depth

AlexNet, 8 layers (ILSVRC 2012)

VGG, 19 layers (ILSVRC 2014)

GoogleNet, 22 layers (ILSVRC 2014)

Pic credit: Kaiming He
Revolution of depth

AlexNet, 8 layers (ILSVRC 2012)

VGG, 19 layers (ILSVRC 2014)

ResNet, 152 layers (ILSVRC 2015)

ImageNet Classification top-5 error (%)
Introduction to ConvNets

- Some Deep Learning figures
- Neural Networks
  - Architecture
  - Forward pass (inference)
  - Backward pass (learning)
  - Optimization (bag of tricks)
- Convolutional Neural Networks
  - Architecture
  - Feature maps
- TensorFlow demo
Convolutional Feature Maps

Convolutional:
sliding-window operations

Feature:
encoding “what”
(and implicitly encoding “where”)

Map:
explicitly encoding “where”

Pic credit: http://www.cnblogs.com/cvision/p/CNN.html & Kaiming He
HOG by Convolutional Layers

Steps of computing HOG:
- Computing image gradients
- Binning gradients into 18 directions
- Computing cell histograms
- Normalizing cell histograms

Convolutional perspective:
- Horizontal/vertical edge filters
- Directional filters + gating (non-linearity)
- Sum/average pooling
- Local response normalization (LRN)

HOG, dense SIFT, and many other “hand-engineered” features are convolutional feature maps.

[Mahendran & Vedaldi, CVPR2015]
Feature maps = features and their locations

ImageNet images with **strongest** responses of this channel

Intuition of *this* response:
There is a "circle-shaped" object (likely a tire) *at this position.*

---

**What**

**Where**

---

Slide credit: Kaiming He
Feature maps = features and their locations

ImageNet images with **strongest** responses of this channel

Intuition of *this* response:
There is a "λ-shaped" object (likely an underarm) at this position.

What

Where
Receptive field

- Receptive field of the first layer is the filter size
- Receptive field (w.r.t. input image) of a deeper layer depends on all previous layers' filter sizes and strides
- **Correspondence** between a feature map pixel and an image pixel is not unique
- How to map a feature map pixel to **the center of the receptive field**:

  - For each layer, pad \( \lfloor F/2 \rfloor \) pixels for a filter size \( F \)
    (e.g., pad 1 pixel for a filter size of 3)
  - On each feature map, the response at \((0, 0)\) has a receptive field centered at \((0, 0)\) on the image
  - On each feature map, the response at \((x, y)\) has a receptive field centered at \((Sx, Sy)\) on the image (stride \(S\))

Slide credit: Kaiming He
Hierarchical feature maps

Hierarchical feature maps

Applications by exploiting conv feature maps

Spatial Pyramid Pooling / Roi-Pooling
- fix the number of bins instead of filter sizes
- adaptively-sized bins

SPP-net & Fast R-CNN (the same forward pipeline)
- Complexity: $\sim 600 \times 1000 \times 1$
- $\sim 160x$ faster than R-CNN

Pic credit: Kaiming He
Applications by exploiting conv feature maps

Applications by exploiting conv feature maps
Applications by exploiting conv feature maps

Introduction to ConvNets

• Some Deep Learning figures

• Neural Networks
  • Architecture
  • Forward pass (inference)
  • Backward pass (learning)
  • Optimization (bag of tricks)

• Convolutional Neural Networks
  • Architecture
  • Feature maps

• TensorFlow demo
The unreasonable easiness of deep learning

• Modify the network architecture (usually from a pre-trained model) (the forward pass specifically, backward pass is handled automatically by auto-differentiation in most python based libraries)

• Define an objective function

• Pick a proper optimizer to train your network

• Feed your data properly to the net

• Show demo here

Slide credit: David Duvenaud
Codes adopted from Tensorflow tutorials
Q&A

“The only stupid question is the one you never asked” - Rich Sutton