# 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



Pic credit: <u>NVIDIA blog</u>



# Success of Deep Learning



IM GENET







CONSERVATION SONGBIRDS A LA CARTE legal harvest of million of Mediterranean bird Patt 412

SAFEGUARD TRANSPAREST Don't let openness backfire on individuals nat 60

RESEARCH ETHECS





O NATURE COM NATURE POPULAR SCIENCE 28.5anuary 2008 E WHEN GENES GOT 'SELFISH' Dankins's calling card forty years on MEME 



## Deep Learning in Vision



Pic credit: Kaiming He



ImageNet Classification top-5 error (%)

#### Deep Learning in Vision



Pic credit: Kaiming He



# 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 <u>Ilya Sutskever</u>

#### Universal function approximator

#### Generalization ability

#### Hierarchical representation

Back propagation

TensorFlow



# 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]



Slide credit: A. G. Schwing Pic credit: Stanford CS231n





## Activation functions / Nonlinearity

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



Slide credit: Raquel Urtasun

# Neural Network (Multi-Layer Perception)



Slide credit: Raquel Urtasun Pic credit: Stanford CS231n

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)



• One node is controlled by two parameters w, b







Slide credit: A. G. Schwing

$$y = f(w1 * x + b)$$

#### where the activation function is sigmoid f(x) = 1 / (1 + exp(-x))

- 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

#### meters w, b a pair of nodes



 $w_1 = -100, b_1 = 40, w_3 = 100, b_2 = 60, w_2 = 1, w_4 = 1$ 

- 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



corresponds to one hidden layer

Slide credit: A. G. Schwing

#### meters w, b a pair of nodes curate approximation

- 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

- 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: Raguel Urtasun

#### meters w, b a pair of nodes ccurate approximation **dden layer** is a universal function

# 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



$$egin{array}{rll} h_j({f x}) &=& f(v_{j0} + \sum_{i=1}^D x_i v_{ji}) \ o_k({f x}) &=& g(w_{k0} + \sum_{j=1}^J h_j({f x}) w_{kj}) \end{array}$$

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)
```

Efficient implementation via matrix operations. 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 b3: 1-d vector W3: 1x4 matrix



# 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



Slide credit: A. G. Schwing

#### e output g individual contributions





## Loss function

Classification

- Cross-entropy:  $sum_i(-y_i^*log(f(x_i)))$
- Hinge loss:  $max(0, 1-y_i*f(x_i))$

#### Regression

- L1:  $sum_i(|y_i-f(x_i)|)$
- L2:  $sum_i((y_i-f(x_i))^2)$

Pair-wise similarity

- Contrastive loss:  $E = \frac{1}{2N} \sum_{n=1}^{N} (y) d^{2} + (1)$  Triplet loss:  $\sum_{i=1}^{N} \left[ \|f(x_{i}^{a}) f(x_{i}^{p})\|_{2}^{2} \|f(x_{i}^{a})\|_{2}^{2} \|f(x_{i}^{$

$$egin{split} (1-y) \max\left(margin-d,0
ight)^2 \ (x_i^a) &- f(x_i^a) \|_2^2 + lpha \end{bmatrix}_+ \end{split}$$



## How do we update $w_{ki}$ to minimize the loss?

Output layer



Input layer



Input layer

#### Slide credit: Raquel Urtasun

22

#### Use gradient descent!



#### Pic credit: Sebastian Raschka



Update rule:

 $w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}}$ 

23

## Compute gradient: chain rule



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

Slide credit: Raquel Urtasun

$$w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}}$$
$$\frac{\partial E}{\partial w_{ki}} = \sum_{n=1}^{N} \frac{\partial E}{\partial o_k^{(n)}} \frac{\partial o_k^{(n)}}{\partial z_k^{(n)}} \frac{\partial z_k^{(n)}}{\partial w_{ki}}$$

• L2 loss 
$$\frac{\partial E}{\partial o_k^{(n)}} = o_k^{(n)} - t_k^{(n)} := \delta_k^o$$
  
• g(z) = 1/(1+exp(-z)) 
$$\frac{\partial o_k^{(n)}}{\partial z_k^{(n)}} = o_k^{(n)}(1-e^{n)})$$





## Multi-layer NN case



Slide credit: Raquel Urtasun

If a node has multiple outputs, we have to sum over all gradients from these paths back to that node.

$$\frac{\partial E}{\partial h_j^{(n)}} = \sum_k \frac{\partial E}{\partial o_k^{(n)}} \frac{\partial o_k^{(n)}}{\partial z_k^{(n)}} \frac{\partial z_k^{(n)}}{\partial h_j^{(n)}} = \sum_k \frac{\delta_k^{z,(n)}}{w_{kj}} = \sum_k \delta_k^{z,(n)} w_{kj} := \delta_j^{h,(n)}$$





# 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



# Optimizing neural networks

$$w_{ki} \leftarrow w_{ki} - \eta \frac{\partial E}{\partial w_{ki}}$$

- these error derivatives:
  - Optimization issues
    - how often to update the weights
    - how much to update the weights
  - Ways to reduce overfitting

Slide credit: Geoffrey Hinton Pic credit: Sebastian Raschka



 The back-propagation algorithm is an efficient way of computing the error derivative dE/dw for every weight on a single training case. However, we still need to make other decisions about how to use



### 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.



#### **Small Batch Large Batch**







#### Learning rate

 $w_{ki} \leftarrow$ 

- Don't start too big, and not too small.



Large learning rate: Overshooting. Slide credit: Geoffrey Hinton Pic credit: <u>Sebastian Raschka</u>

Small learning rate: Many iterations until convergence and trapping in local minima.

$$-w_{ki}-\eta \frac{\partial E}{\partial w_{ki}}$$

• 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.

> *# Momentum update* **x += v** # integrate position



Pic credit: Stanford CS231n











## Different optimizers



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

Pic credit: Stanford CS231n



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.



## 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 (loffe & Szegedy):** normalize the activations through a network to take on a unit gaussian distribution Slide credit: Stanford CS231n

33

#### Prevent overfitting

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



Pic credit: Stanford CS231n

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



 $\lambda = 0.1$ 

The effects of regularization strength.



## Prevent overfitting

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



(a) Standard Neural Net



(b) After applying dropout.

Pic credit: Stanford CS231n

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

**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.

Pic credit: Markus, ECCV14

• We want our representation to be translation-invariant.



## Convolutional layer (local connectivity + weight sharing)



## fully connected layer

Pic credit: Stanford CS231n & Geoffrey Hinton



local connectivity

spatial weight-sharing





## Convolution operation on 2D data





4

Image

Convolved Feature

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







## An example ConvNet architecture



Pic credit: Stanford CS231n

41

## Revolution of depth

AlexNet, 8 layers (ILSVRC 2012)



Pic credit: Kaiming He

42

## 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
<b></b>
3x3 conv, 256, pool/2
♥
fc, 4096
♥
fc, 4096
♥
fc, 1000

### VGG, 19 layers (ILSVRC 2014)

Pic credit: Kaiming He



3x3 conv, 64, pool/2
¥
3x3 conv, 128
¥
3x3 conv, 128, pool/2
¥
3x3 conv, 256
¥
3x3 conv, 256
¥
3x3 conv, 256
<b>—</b>
3x3 conv, 256, pool/2
¥
3x3 conv, 512
<b></b>
3x3 conv, 512
<b>—</b>
3x3 conv, 512
¥
3x3 conv, 512, pool/2
¥
3x3 conv, 512
¥
3x3 conv, 512
<b>—</b> •
3x3 conv, 512
<b>•</b>
3x3 conv, 512, pool/2
<b>— •</b>
fc, 4096
<b></b>
fc, 4096
<b>*</b>
fc, 1000

3x3 conv, 64

### GoogleNet, 22 layers (ILSVRC 2014)





## Revolution of depth

AlexNet, 8 layers (ILSVRC 2012)

(ILSVRC 2014)



Pic credit: Kaiming He



44

# 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

45



Pic credit: <u>http://www.cnblogs.com/cvision/p/CNN.html</u> & Kaiming He





# HOG by Convolutional Layers

## **Steps of computing HOG:**

- Computing image gradients
- Binning gradients into 18 directions
- Computing cell histograms
- Normalizing cell histograms

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

Slide credit: Kaiming He

## **Convolutional perspective:**

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

[Mahendran & Vedaldi, CVPR2015]



## Feature maps = features and their locations





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

one feature map of conv<sub>5</sub> (#55 in 256 channels of a model trained on ImageNet)

Slide credit: Kaiming He

ImageNet images with strongest responses of this channel





## Feature maps = features and their locations







one feature map of conv<sub>5</sub> (#66 in 256 channels of a model trained on ImageNet)

Intuition of *this* response: There is a " $\lambda$ -shaped" object (likely an underarm) at this position. What Where

Slide credit: Kaiming He

ImageNet images with strongest responses of this channel



## Receptive field

- Receptive field of the first layer is the filter size  $\bullet$
- 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 lacksquarepixel and an image pixel is not unique
- How to map a feature map pixel to the center lacksquareof the receptive field:
  - For each layer, pad  $\lfloor F/2 \rfloor$  pixels for a filter size F ٠ (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



MD Zeiler, et al. Visualizing and Understanding Convolutional Networks, ECCV2014

51

## Hierarchical feature maps



MD Zeiler, et al. Visualizing and Understanding Convolutional Networks, ECCV2014

52



### Spatial Pyramid Pooling / Roi-Pooling

- fix the number of bins instead of filter sizes
- adaptively-sized bins

Pic credit: Kaiming He



SPP-net & Fast R-CNN (the same forward pipeline)

- Complexity: ~600 × 1000 × 1
- ~160x faster than R-CNN

53



Jonathan Long, et al. Fully Convolutional Networks for Semantic Segmentation, CVPR2015



54



Saining Xie, et al. Holistically-Nested Edge Detection, ICCV2015



55



David Eigen, et al. Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture, ICCV2015

56

# 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

57

# The unreasonable easiness of deep learning

- Modify the network architecture (usually from a pretrained model) (the forward pass specifically, backward pass is handled automatically by autodifferentiation 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



58



## "The only stupid question is the one you never asked" -Rich Sutton

59