Fundamentals of Deep Learning With Applications

Jon Krohn – chief data scientist at untapt

Slides at jonkrohn.com/talks

Lots of Jupyter notebooks there that can learn from

Outline:

Case study in vision building blocks

Overview of neuro theory

Contemporary applications – CNNs, LSTMs, deep reinforcement learning

# Vision

Trilobite eye acts as simple light/ movement detector. Involved in Cambrian explosion

Hubel and Wiesel 1959 study on vision in cats. Single-cell recording. V1 processing is the simplest processing

Found no activation in V1 when showed complex images. But when showed edge of a slide that were taking offscreen, got V1 activation! Processing edges, rather than complex images

Were able to discover from there that specific neurons respond to edges at diff orientations, diff angles. Amt of firing in a particular neuron peaks at 45 deg, is 0 at 90 deg, follows normal distribution at intermediate angles

Edge detectors feed in corner detectors and so on, which feed into increasingly complex feature detectors

Humans have fusiform face area, specialized in processing faces. Relay starting w/ edges in V1 up into more complex shapes

Throughout most of 20th cen, most machine vision involved explicit coding of features in the image. Need to code what kind of characteristics a face, or a house, tend to have

Viola and Jones 2001 – simple algo w/ alternating black and white boxes that ended up being v good face detector.

Fukushima 1980 Neurocognitron – early attempt at neural net machine vision

## Yann LeCun’s MNIST for digit recognition.

First applied for zip codes on mail.

Input layer -> lowlevel feature maps like edge detectors -> feature detectors for more complex shapes -> output layer

# Post 2012 nnets

An AI winter that followed this – ended in 2012 when Jeff Hinton’s ILSVRC beat everything else on ImageNet competition

Every year since than a deep learning net won ImageNet

In 2015, NNs outperformed humans on ImageNet

AlexNet – is Hinton’s team’s algo. Similar structure to LeCun’s MNIST from 1990s!

Edge/ blob at 1st layer, texture at 3rd, object parts at 5th, object classes at output layer

More processing power and larger training sets were key.

Since 2012, deep learning in many apps. Face recog facebook, autoreplies on Gmail, Siri voice recog

Sunspring – film written by LSTM.

# Hardware and Libraries

W/ small amt of data, local machine; few hundred to thousand $ of GPUs to build own server; AWS or Google Cloud platform instances (GCloud now has GPU instances! Amazon has them too)

Most popular libs for deep learning are Caffe (C++/ Python), Torch (Lua, now Python too), Theano (Python), TensorFlow (Python)

All of these work on pretrained models; Caffe and Torch are best for work on big already-trained models, using pre-trained models and teaching them to recognize a new specific thing

Can v easily train a new top layer to take advantage of pretrained mode

TensorFlow is best by far for distributing across many GPUs

RNNs are best in Theano or TensorFlow

Keras and TFLearn are higher-level APIs good for getting into some of these

Jon has a lot of getting-started material on his site

# Theory

Bio neurons are binary – can be either on or off, fire at same output regardless of input (preeetty much)

W/ perceptrons, small input changes can have dramatic output changes b/c of threshold. Simple threshold is v coarse, makes it hard to learn at the output layer.

So, sigmoid units!

Also tanh neurons – vary from -1 to 1 instead of 0 to 1

Since 2010, ReLus have been more common than sigmoid units

There are also “leaky” ReLus, which go negative like tanh units do

ReLus capture less info, but have some advantages

V hard to learn when stuck at top or bottom of the s in sigmoid or tanh – big input changes have small output effects

Much easier to calculate changes on ReLu than on sigmoid, so easier to have lots of ReLus than lots of sigmoids, and here quantity overcomes quality of info each unit gives

# MNIST

MNIST – input is a 28x28 pixel image.

Fully connected neural net; all 784 input units feed into all 15 neurons in the first hidden layer

So 784\*15 connections b/w input and 1st hidden layer

10 output units, for digits 0 thru 9

# TensorFlow Playground

Great learning environment! Runs in browser.

Default example w/ a single output neuron works like linreg, can only divide out straight lines. Want to divide out a core from a ring around it.

2 and then 3 output layers lets you get a polygon output – 3 classifies pretty well!

A swirling shape gets more complicated; adding another hidden layer makes it get more doable

2 8n hidden layers to 4n hidden layer to 4n output layer does it well

Edges -> corners -> 3rd and 4th layers have more complex representations

Training strengthens some connections but not others.

# How many units do you need?

# of output units you need depends on output you need. Binary output needs 2, digit recog needs 10. Image recog usually does 1k outputs

Picking number of hidden layers? How many neurons in a layer? Play around! Can peruse the lit for what works for similar problems to yours, but it’s art. Need to compromise w/ the processing power you have, of course.

Deep network is a net w/ 3 or more hidden layers

# Example w/ MNIST dataset:

Links in slides to example of a simple deep net in TFLearn

Split data into test and training, naturally

Input layer -> fully connected layer (= “dense layer”), here 64 in the layer, tanh activation fct, a regularization method, and a parameter on the regularization that enables you to avoid overfitting in way that won’t go into

This feeds into *dropout layer* to avoid overfitting. This dropout layer cuts out 20% of neurons in each training pass. (50% is more standard). This is really just a regularization of 1st dense layer

Feeds into 2nd dense layer, which itself has a dropout

Output: softmax layer to squish it into probability

Input w/ 784 units, 2x64 dense layers, 10 output units

Then training fct – stochastic grad descent

Here, consider correct if answer is in top 3 guesses

Then specify network in line 13 – ties all the above info together

Then a couple lines on training

This v simple network has 99% in top 3 guesses

# Useful concepts

Synaptic pruning – human brains work like random initialization! Tons of connections, that get cut as repeated stimuli teach us to recognize particular inputs

Stochastic grad descent

Backprop

Michael Nielsen (?) has great online textbook for deep learning

Overfitting and avoiding it:

Have a reasonable number of parameters; too many won’t generalize well

NNets have thousands or millions of parameters! Easy to overfit

L1/ L2 regularization helps, dropout helps

“Artificial data set expansion” is the thing of perturbing training data – it helps!

Summary of how to improve your NNet – link in slides, long to go thru

NNets are proven “universal” – can solve any continuous fct

Unstable gradient – a problem in deep nnets. The more layers you add, the harder it is to backprop error thru all layers

“Vanishing gradient” – harder and harder to learn at early layers the deeper your net is

OR, if parameterize differently, can get an “exploding gradient” where learns way too fast. This is much rarer problem than vanishing grad

# State of the art on ImageNet

Deeper and deeper nets *are* cutting edge, though! Imagenet winners are getting deeper! 2014 winner had 152 layers!

ResNet had super good recent results – look it up

AlexNet and VGGNet are similar convents for image recog. Alternating convolutional and pooling layers

Pooling layers make things similar. E.g. take 4 adjacent neurons, takes max of that and makes it one value. Reduces amt of data.

Tend to have 2 fully connected layers at the end before the output, don’t they?

# CNNs

Convolutional layer scans a “patch” across the image that makes it invariant to position of feature you’re looking for

Keras – higher-level API to call Theano or TensorFlow. Really good middle ground b/w simplicity of TFLearn and complexity of using those frameworks directly

# Some apps

“2.5 dimensional” CT scans

Computer-aided detection for medical imaging

“Model Zoo” of pretrained nets is very useful! Great for just adding a top layer

Video processing

# There’s a recommended book for deep learning in medical imaging!

LSTMs – a kind of RNN that’s great for language processing

e.g. Sunspring script

Vector Space Embeddings – NLP, v good. See Word2Vec. Basically understanding of language!

One app Jon’s company does is learn hiring manager preferences, w/ vector space embeddings, to filter resumes they want to see. Is done with LSTMs.

Also provide feedback to candidates applying to jobs re what’s optimal for their application

Effort to “gamify” the experience and get people to know what works

Mutlistage Bayesian regression models, then ensemble it w/ a deepnet, gets great model fits

Deep-orange.untapt.com – chance of getting interview based on bullet points you enter

“Quick, Draw!” uses convnets and LSTM!

OpenAI and Google deepmind have good stuff for deep reinforcement models

Unsupervised learning for clustering – Vectorspace embeddings are great example! Unlabeled corpus, go to word meanings. 50 or 100 dim space often used, in which more related words cluster.

A more general approach to working w/ unlabeled data is autoencoders.

GANs are great!