DEEP LEARNING

Lecture based on:
Deep learning Lecture by Elad Hoffer and
Oxford Deep Learning Course by Nando de Freitas
Agenda

• Background

• Building a Neural Network

• Training and Optimization

• Convolutional Neural Networks
  • Pooling Layers

• Recurrent Neural Networks

• Vision Tasks

• Generative Models
BACKGROUND
Background

• “Traditional machine learning” is about:
  • Feature engineering
  • Creating models and representations
  • Optimizing and predicting
Feature Learning

- Can we create a model to learn and extract hierarchical feature representations from data?
What is Deep Learning?

- **Deep learning** (deep structured learning or hierarchical learning) is a set of algorithms that attempt to model *high-level abstractions* in data by using model architectures composed of *multiple non-linear transformations*.

Usually a deep learning model is simply a: Neural networks with *more than 1 hidden layers*. 
A Brief History of Neural Networks

- **Electronics Brain**
  - 1943
  - S. McCulloch - W. Pitts
  - Adjustable Weights
  - Weights are not Learned

- **Perceptron**
  - 1957
  - F. Rosenblatt
  - B. Widrow - M. Hoff
  - Learnable Weights and Threshold

- **ADALINE**
  - 1960
  - M. Minsky - S. Papert
  - XOR Problem

- **XOR Problem**
  - 1969
  - Dark Age ("AI Winter")

- **Multi-layered Perceptron (Backpropagation)**
  - 1986
  - D. Rumelhart - G. Hinton - R. Williams
  - Solution to nonlinearly separable problems
  - Big computation, local optima and overfitting

- **SVM**
  - 1995
  - V. Vapnik - C. Cortes
  - Limitations of learning prior knowledge
  - Kernel function: Human Intervention

- **Deep Neural Network (Pretraining)**
  - 2006
  - G. Hinton - S. Ruslan
  - Hierarchical feature Learning

- **2010**
Motivation slides…

a) Original images

b) Generated images
Antoine-Joseph Sax was born on 6 November 1814, in Dinant, in what is now Belgium, to Charles-Joseph Sax and his wife. While his given name was Antoine-Joseph, he was referred to as Adolphe from childhood.[2][3] His father and mother were instrument designers themselves, who made several changes to the design of the French horn. Adolphe began to make his own instruments at an early age, entering two of his flutes and a clarinet into a competition at the age of 15. He subsequently studied performance on those two instruments as well as voice at the Royal Conservatory of Brussels.[2][5]

Sax faced many near-death experiences. Over the course of his childhood, he:

fell from a height of three floors, hit his head on a stone and could barely stand afterwards,
at the age of three, drank a bowl full of vitriolized water and later swallowed a pin, burnt himself seriously in a gunpowder explosion, fell onto a hot cast iron frying pan, burning his side, survived poisoning and suffocation in his own bedroom where varnished items were kept during the night, was hit on the head by a cobblestone, and fell into a river and barely survived.

His mother once said that "he's a child condemned to misfortune, he won't live." His neighbors called him "little Sax, the ghost".[4][5]
Object Detection
Play Go
BUILDING A NEURAL NETWORK
Perceptron

- Recall the basic perceptron:

\[
\text{out} = \begin{cases} 
1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\
-1 \text{ otherwise}
\end{cases}
\]

- Training using SGD
- Is it really SGD…?
Training a perceptron

Algorithm 1 Perceptron Rule

1: $\vec{w} \leftarrow \vec{w}_0$  
2: $ErrorFound \leftarrow True$  
3: while $ErrorFound$ do  
4:    $ErrorFound \leftarrow False$  
5:    for $i = 1, \ldots, N$ do  
6:        $\hat{y}_i \leftarrow sgn (\vec{x}_i \cdot \vec{w}_i)$  
7:        if $\hat{y}_i \neq y_i$ then  
8:            $ErrorFound \leftarrow True$  
9:            $\vec{w} \leftarrow \vec{w} + \eta y_i \vec{x}_i$  
10:        end if  
11:    end for  
12: end while  
13: return $\vec{w}$
Adaptive Linear Neuron (Adaline)

- Same principle, different error function

\[ E(x) = \frac{1}{2} \sum_{i \in D} (y^i - w^T x^i)^2 \]

- Can we use SGD?
  - Yes!
Reminder - Modular approach (Lego)

We code layers not networks!

Layer specification
Each layer needs to provide three functions:

1. What is the layer output given its input (Forward)
2. Derivative with respect to the input
3. Derivative with respect to parameters
NN layer view

\[
\text{Loss } Z_{L+1} = E \\
\delta^{L+1} = \frac{\partial E}{\partial E} = 1 \\
\text{Loss (Layer L)} \\
\text{Prediction } Z^L \\
\delta^L \\
\text{Layer } L - 1 \\
\text{Layer } k \\
\delta^k \\
\text{Layer } k - 1 \\
\text{Layer 1} \\
\text{Input } x = Z^1 \\
\delta^1 = \frac{\partial E}{\partial Z^1}
\]
Single layer

Output of the layer $Z^{k+1}$

$\delta^{k+1}$

Derivative of the loss with respect to layer output

$\frac{\partial E}{\partial W^k}$ Gradient for the layer

Input $x$

$Z^k$

$\delta^k$

Derivative of the loss with respect to layer input
Backpropagation

• Forward pass

\[ Z^{k+1} = f(Z^k) \]

• Backward pass

\[ \delta^L = \frac{\partial E}{\partial Z^L} \]

• Applying chain rule for a single layer

\[ \frac{\partial E}{\partial Z^k} = \frac{\partial E}{\partial Z^{k+1}} \frac{\partial Z^{k+1}}{\partial Z^k} = \delta^{k+1} \frac{\partial Z^{k+1}}{\partial Z^k} \]

• Gradient with respect to layer parameters if it has parameters

\[ \frac{\partial E}{\partial W^k} = \frac{\partial E}{\partial Z^{k+1}} \frac{\partial Z^{k+1}}{\partial W^k} = \delta^{k+1} \frac{\partial Z^{k+1}}{\partial W^k} \]
Adaline – Layer View

\[ \sum w_i x_i \]

\[ \text{out} = \begin{cases} 
1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\
-1 & \text{otherwise}
\end{cases} \]

\[ E(x) = \frac{1}{2} \sum_{i \in D} (y^i - w^T x^i)^2 \]
Adaline derivatives

- Error layer:

\[ E(z) = \frac{1}{2} \sum_{i \in D} (y^i - z^i)^2 = \frac{1}{2} ||Y - Z||^2 \]

\[ \frac{\partial E}{\partial z} = - \sum_{i \in D} (y^i - z^i) \nabla z^i = -(Y - Z) \nabla Z \]

- Linear layer:

\[ z = f(x) = w^T X \]

\[ \frac{\partial f}{\partial w} = \sum_{i \in D} x^i = X \]

Together:

\[ \frac{\partial E}{\partial w} = \frac{\partial E}{\partial z} \frac{\partial z}{\partial w} = - \sum_{i \in D} (y^i - w^T x^i) x^i = -(Y - w^T) X \]
Stochastic Gradient Descent

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration $k$

Require: Learning rate $\epsilon_k$.
Require: Initial parameter $\theta$

while stopping criterion not met do

Sample a minibatch of $m$ examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.
Compute gradient estimate: $\hat{g} \leftarrow +\frac{1}{m} \nabla_{\theta} \sum_{i} L(f(x^{(i)}; \theta), y^{(i)})$

Apply update: $\theta \leftarrow \theta - \epsilon \hat{g}$

end while
The Perceptron – a variation

- We can replace the step function with a sigmoid:

\[
\text{out} = \begin{cases} 
1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\
-1 \text{ otherwise}
\end{cases}
\]

\[
E(x) = \sum_{i \in D_{\text{miss}}} w^T x^i y^i
\]
The Perceptron – a variation

Sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

Linear layer

Activation layer

Cost function

$$E(y_i) = - \sum_{i=1}^{n} (y_i \log \pi_{i0} + (1 - y_i) \log \pi_{i1})$$
For convenience we’ll use the following notations:

\[
P(y_i | x_i, w) = \begin{cases} 
\pi_{i1} &\equiv \sigma(w^T x) = \frac{1}{1 - e^{-w^T x}}, & \text{if } y_i = 1 \\
\pi_{i0} &\equiv 1 - \sigma(w^T x) = 1 - \frac{1}{1 - e^{-w^T x}}, & \text{if } y_i = 0 
\end{cases}
\]
Logistic Regression

• Expanding the expression
  \[ P(y|x, \theta) = Ber(y|\sigma(w^T x)) \]

• we get:

  \[
  NLL(\theta) = \sum_{i=1}^{n} \log \pi_{i1}^{y_i} \pi_{i0}^{1-y_i}
  \]

  \[ = - \left( \sum_{i=1}^{n} y_i \log \pi_{i1} + (1 - y_i) \log \pi_{i0} \right) \]

• The expression we get is known as **cross-entropy**
The Perceptron – another variation

- Let’s create an output for each class

\[ \pi_i = \frac{e^{x_iw_0}}{e^{x_iw_0} + e^{x_iw_1}} \]

Cost function

\[ E(y_i) = -\sum_{i=1}^{n} \mathbb{1}_0 \log p_{i0} + \mathbb{1}_1 \log p_{i1} \]

- Why use this “architecture”? 
Perceptron – yet another variation

- Remove sigmoid and replace error function with hinge loss (and add $L_2$ regularization):

$$E(y_i) = C \sum_{i=1}^{n} \max(0, 1 - (wx_j + b)y_i) + \|w\|^2$$

- And we get SVM!
MULTI-LAYERS PERCEPTRON
Multi-Layer Perceptron

- Using the “building blocks”, we can stack multiple layers to form a simple network
  - Networks are fed with inputs and trained to output the corresponding targets according to a loss function.
  - Hidden layers, are said to represent the data, by applying non-linear transformations.
Multi-layer intuition

- The problem below is separable, but not linearly
Multi-layer intuition

- The problem below is separable, but not linearly
Multi-layer intuition – cont.

• The multi layers can be thought of transforming the new data to a new representation
Multi-layer intuition – cont.

- The multi layers can be thought of transforming the new data to a new representation
PCA – another point of view

- Let’s re-examine PCA in a “neural network view”
- Recall that PCA minimized the “Reconstruction Error"

\[ E(y_i) = \sum_{i=1}^{n} ||X - Y||^2 \]
“Deep PCA”

• Let’s re-examine PCA in a “neural network view”
• Recall that PCA minimized the “Reconstruction Error"

\[ E(y_i) = \sum_{i=1}^{n} \|X - Y\|^2 \]

By adding non-linearities between layers we can learn an efficient non-linear encoding (these are called “deep autoencoders”)

Linear layer  →  “code”  →  C
Why are non-linearities important?

- Why bother with non-linear units?
  \[ Z^3 = W_2^T z^2 = W_2^T W_1^T X = \tilde{W}^T X \]

The two linear layers “collapsed” to a single linear layer.
Common non-linearities

- **Sigmoid** - \( \sigma(x) = \frac{1}{1-e^{-x}} \)
- **Hyperbolic tangent** - \( \tanh(x) = \frac{1-e^{-x}}{1+e^x} \)
- **Rectified Linear Unit** - \( \text{ReLU}(x) = \max(0, x) \)
MNIST Visualization

- [http://scs.ryerson.ca/~aharley/vis/fc/](http://scs.ryerson.ca/~aharley/vis/fc/)
OPTIMIZATION IN NEURAL NETWORKS
Stochastic Gradient Descent

- Training a neural network with a large number of parameters requires a simple optimization technique.
  - Usually, a variant of *Stochastic gradient descent (SGD)* is used, using a noisy subset estimation of the gradient (mini-batch).
  - It requires $O(n)$ number of computations and memory use, where $n$ is the number of parameters.
Efficient optimization of NNs

• Optimizing neural networks can be very challenging, as we’re dealing with a highly non-convex problem, residing in a high dimensional space. Problems can be observed even in a very simple non-convex error landscape:
Efficient optimization of NNs – cont.

• Some obvious difficulties:
  • Strong dependency on initial weights
  • Possibly poor error estimation leading to noisy gradients
  • Exploring a (very) high-dimensional space using only local information

• Updating the network is done using backpropagation (chain-rule)
CONVOLUTIONAL NEURAL NETWORKS
ConvNets

A convolutional network (ConvNet/CNN), is composed of multiple layers of convolutions, pooling and non-linearities.
Background

- Convolutional networks are rooted in ideas from “NeoCognitron” (Fukoshima 80’) and inspired from Hubel’s and Wiesel’s work on visual primary cortex (59’).

- Their first modern form appeared in LeCun’s 98’ work to recognize handwritten digits (LeNet).

- They reached their current status as visual recognition de-facto standard, after beating traditional computer-vision techniques in 2012 ImageNet competition by huge margin (AlexNet).
Spatial Convolution

- Natural images have the property that many patches share statistical properties, and can often be described with a fairly small set of the same features.

- This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations.

- This is a widely used property in computer-vision and image-processing related tasks.
Linear layer for images

- Let take an example of a small image: 200x200x3 that is fully connected (linear connection) to 1000 units.
- This layer has 200x200x3x1000=120M parameters!
- Furthermore, it isn’t shift invariant!
Linear layer for images

- Every “neuron” in the output is connected to every pixel in the input
Improvement #1

- Every “neuron” is connected only to a local neighbored of neurons (pixels) of the input
Improvement #2

- The weights connecting each output neuron to the inputs neurons are the same weights (shared)
Spatial Convolution

- For example, having learned features over small (say 5\times5) patches sampled from the image, we can then convolve them with the larger image, thus obtaining a different feature activation value at each location in the image.
Spatial Convolution – cont.

• We perform spatial convolutions on volumes as well:
  • In the example below, we apply 3 kernels of size $3\times3\times2$
  • The output is a tensor of size $3\times3\times3$

Reduced dimension due to kernel size  #kernels
2D convolution
Convention and terminology

• *2D* grids that represent image space are called *feature maps*. Each value in a feature map represent information about a specific location in the input image.

• The parameters learned for each convolution layer are also ordered as a *2D map* - usually called “*kernels*” or “*filters*”

• Convolutions can have *strides* - moving the filters with a fixed step and *padded* - adding zero values to avoid spatial size decrease

• *Bias* values are added per-map (number of bias elements is the number of output feature maps)
Forward Function

It is most natural to think of this as correlation between two cubes:

- Our input is $X \in \mathbb{R}^{N \times W \times H}$ where $N$ is the number of feature maps, each of spatial size $W \times H$.
- We wish to correlate them with $W \in \mathbb{R}^{M \times N \times K \times K}$ - $M$ different filters of size $N \times K \times K$
- Output is $Y \in \mathbb{R}^{M \times (W-K+1) \times (H-K+1)}$ (no stride, no padding)

$$Y_{m,i,j} = \sum_{n=0}^{N-1} \sum_{k=0}^{K-1} \sum_{l=0}^{K-1} w_{m,k,l} \cdot x_{n,(i+k),(j+l)} + b_m$$

It can also be expressed as sum of $M$ 2d-convolutions

$$Y_m = \sum_{n=0}^{N-1} W_m * X_n + b_n$$
POOLING LAYERS
Pooling

• To describe a large image, one natural approach is to aggregate statistics of features at various locations. For example, one could compute the mean (or max) value of a particular feature over a region of the image.

Max pooling
With 2x2 filters
Stride 2
Pooling – cont.

Pooling operations serve two main purposes:

- **Dimensionality reduction** - pooling is usually done with stride bigger than 1, effectively reducing the spatial size of the maps and allowing easier processing.

- **Local invariance to translation/transformation** - by pooling a spatial area to single point, we allow small invariance to location and deformations. This also reduces the ability to overfit on specific images (generalizes from patches).
Pooling functions

Most effective pooling methods are

- Max Pooling - taking the maximum element from the pooled area.
- Average Pooling - taking the average of the pooled area.
ConvNets

A convolutional network (ConvNet/CNN), is composed of multiple layers of convolutions, pooling and non-linearities.
MNIST Visualization

- [http://scs.ryerson.ca/~aharley/vis/conv/](http://scs.ryerson.ca/~aharley/vis/conv/)
What do convolutional networks learn?

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
AlexNet

The first (recent) work that popularized Convolutional Networks in Computer Vision was the “AlexNet”, developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton.

- AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and significantly outperformed the second runner-up (top 5 error of 16% vs 26% error).
- The Network was deeper, bigger, than previous networks - $60M$ parameters, 8 trainable layers.
GoogLeNet

The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al. from Google called GoogLeNet. It featured some new ideas:

• Inception Module - concatenation of several convolution sizes
• $1 \times 1$ convolutions + Average Pooling instead of Fully Connected layers (both introduced by Lin in “Network-in-network” paper from 2013).
• This dramatically reduced the number of parameters in the network (5M, compared to AlexNet with 60M).
• Google later improved this network by introducing batch-normalization + different inception configuration (Inception v2-v4)
GoogLeNet – cont.
VGG Network

The runner-up in ILSVRC 2014 was the network from Karen Simonyan and Andrew Zisserman that became known as the VGGNet. Its main contribution was in showing that the depth of the network is a critical component for good performance.

- Their best network contains 16 trainable layers.

- Features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from the beginning to the end.

- Expensive to evaluate and uses a lot more memory and parameters (140M).
Residual networks

Residual Network developed by Kaiming He et al. was the winner of ILSVRC 2015. It features a very deep architecture (152 layers) with special skip connections and heavy use of batch normalization. The architecture is also missing fully connected layers at the end of the network.
Trends in designing CNNs

Some noticeable trends in recent CNN architectures:

- Smaller convolution kernels - $3 \times 3$ is very dominant.

- Deeper - using smaller kernels is rectified by stacking more layers - effectively increasing the receptive field of the network.

- Less pooling - going deeper means we do not want to reduce spatial size too quickly. Modern network have small number (if any) of pooling layers.
RECURRENT NEURAL NETWORKS
Recurrent Neural Networks

- Recurrent neural networks (RNNs) are networks where connections between units form a directed cycle.

\[ h_t = \phi(W_i x_t + W_r h_{t-1} + b) \]

- \( x_t \in \mathbb{R}^N \) - input at time \( t \)
- \( h_t \in \mathbb{R}^M \) - state at time \( t \)
- \( W_i \in \mathbb{R}^{M \times N}, W_r \in \mathbb{R}^{M \times M}, b \in \mathbb{R}^M \)

- \( \phi \) is usually a bounded non-linearity (tanh, sigmoid)
RNN Properties

- RNN hidden states work as memory on previous inputs and states.

- Temporal dependency and internal memory allow processing of sequences of inputs

- Able to address wide range of time-dependencies

- Able to learn and infer sequences of varying length
Long- short- term memory

• Most successful recurrent model used today: Long Short-Term Memory (LSTM) architecture (Hochreiter and Schmidhuber, ‘97)
VISION TASKS & USAGES
Transfer learning using deep networks

We’ve seen how, given enough data, we can train a convolutional network to learn complicated visual tasks.

• In practice, few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size.

• Instead, it is common to use a network that was pre-trained on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories),
Transfer learning

• This turned out to work pretty well in practice:
When transferring our model to the new task, our two main approaches for using the trained CNN are

- **Generic feature extractor** - Take the pretrained network, remove the last fully-connected layer, then treat the rest of the ConvNet as a fixed feature extractor for the new dataset. You can then train a linear classifier (e.g. Linear SVM or logistic regression) on those features for the new dataset.

- **Fine-tuning** - Replace the classifier with one for the new task, then fine-tune the weights of the pretrained network by continuing the backpropagation. It is possible to fine-tune all the layers of the ConvNet, or it’s possible to keep some of the layers fixed.
Multi-layer intuition – reminder

- The multi layers can be thought of transforming the new data to a new representation.

![Non-linear separator in original space](image1.png)

![Linear separator in transformed space](image2.png)
Segmentation using CNNs

• Image segmentation (*DeepMask* Pinheiro 15’)
Caption Generation

• Another usage is transferring visual knowledge into language domain - such as generating captions or question answering. This is done by augmenting a convolutional network with a recurrent one.
Caption Generation
Style transfer
Style transfer
GENERATIVE MODELS
Generative Models

One new and exciting framework for unsupervised, generative models is the *Adversarial network* (Goodfellow 14’).
It consists of two networks trained simultaneously:

• A *generative model* $G$ that tries to capture the data distribution and generate new samples.

• A *discriminative model* $D$ that estimates the probability that a sample came from the training data rather than $G$.

The training procedure for $G$ is to maximize the probability of $D$ making a mistake - it is “cheating” by using the gradients with respect to $D$. 
Generative Adversarial Network

- The Loss function is defined as such:
  \[
  \min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_{z}(z)} [\log(1 - D(G(z)))]
  \]
Generative adversarial networks

• “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks” (Alec Radford, Luke Metz, Soumith Chintala 15’)

---

88
Generative adversarial networks
Generative adversarial networks

- Many GAN papers over the past year - with impressive improvement over time.
Cycle-GAN (Zhu et al. 17’)
Deep dreaming

Another interesting usage is to try and maximize the $L_2$ norm of intermediate layer activations, by optimizing an input image. This can help visualize the role each layer play:
Deep Learning – solution to all our problems?

**What data science methods are used at work?**

Logistic regression is the most commonly reported data science method used at work for all industries except Military and Security, where Neural Networks are used slightly more frequently.

![Bar chart showing the percentage of companies using different data science methods. Logistic Regression is used by 63.5%, Decision Trees by 49.9%, Random Forests by 46.3%, Neural Networks by 37.6%, Bayesian Techniques by 30.6%, Ensemble Methods by 28.5%, SVMs by 26.7%, Gradient Boosted Machines by 23.9%, CNNs by 18.9%, RNNs by 12.3%, Other by 8.3%, Evolutionary Approaches by 5.5%, HMMs by 5.4%, Markov Logic Networks by 4.9%, and GANs by 2.8%.

7,301 responses
Deep Learning – solution to all our problems?

- Neural network are unable to identify low-dimensional signals (i.e. encode the label of the image in one or more pixels)

- Neural network are unable to identify global signals (i.e. encode the label as the mean value of the image)

*Hoffer, Fine & Soudry, On the Blindspots of Convolutional Networks, 2018*