Foundation of Deep Learning

Anthony Faustine

Datascientist
(CeADER)

Friday 26^{th} June, 2020

Learning goal

- Understand the basic building block of deep learning model.
- Learn how to train deep learning models.
- Learn different techniques used in practise to train deep learning models.
- Understand different modern deep learning architectures and their application.
- Explore opportunities and research direction in deep learning.

Outline

What is Deep Learning

Deep Learning a subclass of machine learning algorithms that learn underlying features in data using multiple processing layers with multiple levels of abstarction.

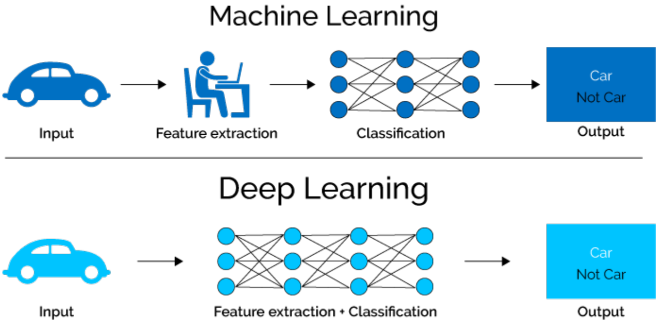


Figure 1: ML vs Deep learning: *credit:*

Deep Learning Success

Automatic Colorization

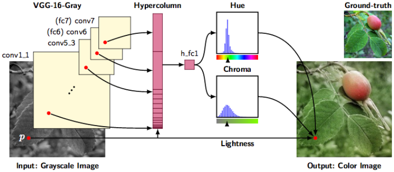


Figure 2: Automatic colorization

Object Classification and Detection

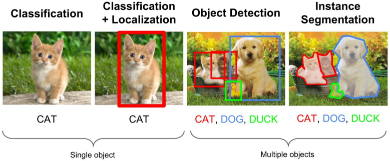


Figure 3: Object recognition

Image Captioning

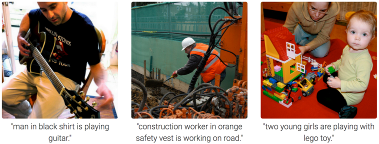
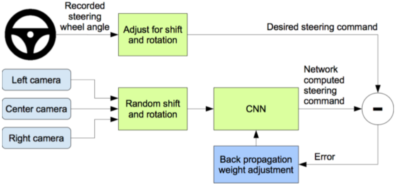


Image Style Transfer

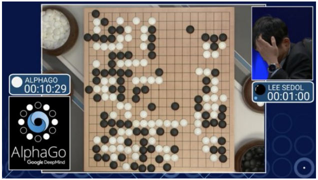
Deep Learning Success

Self driving car

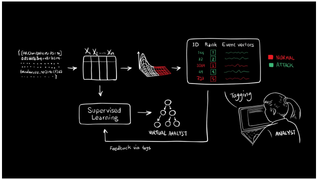


Drones

Game

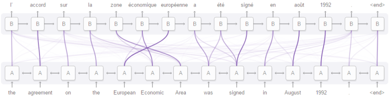


Cyber attack prediction

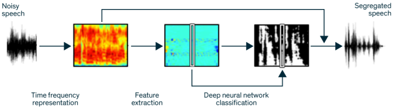


Deep Learning Success

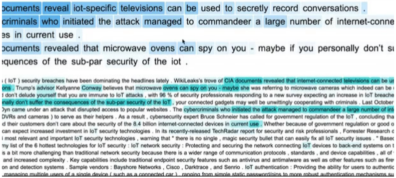
Machine translation



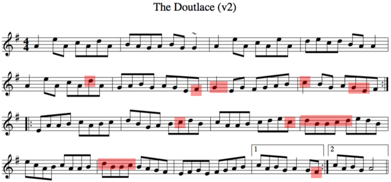
Speach Processing



Automatic Text Generation

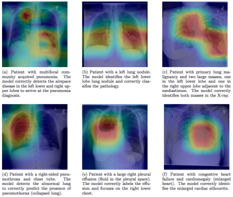


Music composition



Deep Learning Success

Pneumonia Detection on Chest X-Rays



(a) Patient with multifocal community acquired pneumonia. The model correctly detects the airspace disease in the left lower and right upper lobes to arrive at the pneumonia diagnosis.

(b) Patient with a left lung nodule. The model identifies the left lower lobe lung nodule and correctly classifies the pathology.

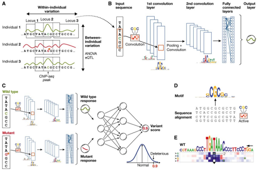
(c) Patient with primary lung malignancy and two large masses, one in the left lower lobe and one in the right upper lobe adjacent to the mediastinum. The model correctly identifies both masses in the X-ray.

(d) Patient with a right-sided pneumothroax and chest tube. The model detects the abnormal lung to correctly predict the presence of pneumothorax (collapsed lung).

(e) Patient with a large right pleural effusion (fluid in the pleural space). The model correctly labels the effusion and focuses on the right lower chest.

(f) Patient with congestive heart failure and cardiomegaly (enlarged heart). The model correctly identifies the enlarged cardiac silhouette.

Computational biology



Pedict heart disease risk from eye scans

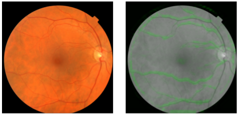
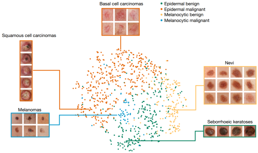


Image of retina

Blood pressure predictions focus on blood vessels

Diagnosis of Skin Cancer



More stories

Why Deep Learning and why now?

Why deep learning: Hand-Engineered Features vs. Learned features.

Traditional ML

- Use engined feature to extract useful patterns from data.
- Complex and difficult since different data sets require different feature engineering approach

Deep learning

- Automatically discover and extract useful pattern from data.
- Allows learning complex features e.g speach and complex networks.

Why Deep Learning and why now?

Why Now?

Big data availability

- Large datasets
- Easier collection and storage

Increase in computaional power

- Modern GPU architecture.

Improved techniques

- Five decades of research in machine learning.

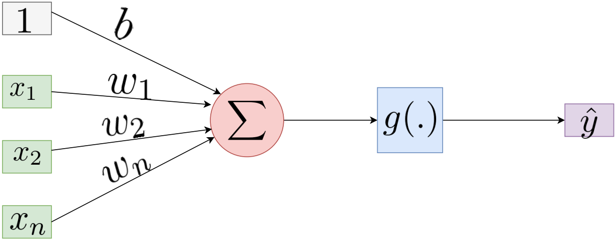
Open source tools and models

- Tensorflow.
- Pytorch
- Keras

Outline

The Perceptron

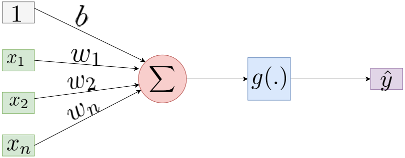
A perceptron is a simple model of a neuron.



The output: $\hat{y} = f(x) = g(z(x))$ where

- x, y input, output.
- w, b weight and bias parameter θ
- activation function: $g(.)$
- pre-activation: $z(x) = \sum_{i=1}^{n} w_i x_i + b$

Perceptron



$$\hat{y} = g(z(x))$$

$$\hat{y} = g(b + \sum_{i=1}^{n} w_i x_i))$$

$$\hat{y} = g(\mathbf{b} + \mathbf{wx}))$$

The Perceptron: Activation Function

Why Activation Functions?

- Activation functions add non-linearity properties to neuro network function.
- Most real-world problems + data are non-linear.
- Activation function need to be differentiable.

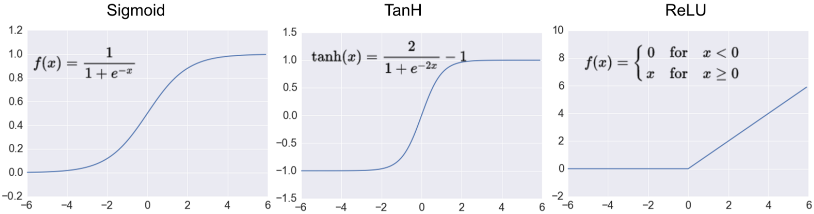
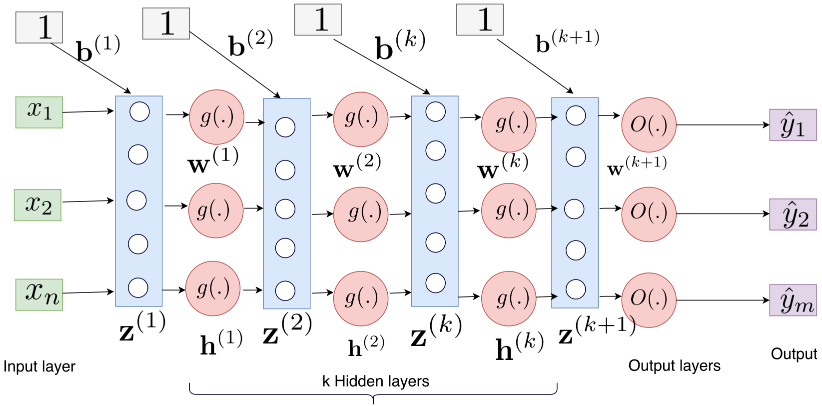


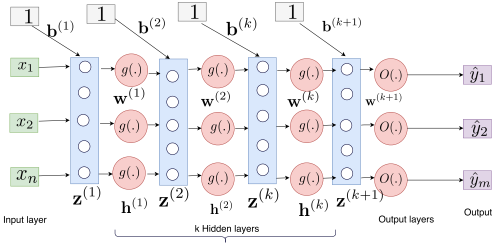
Figure 4: Activation function `credit:kdnuggets.com`

Multilayer Perceptrons (MLP)

We can connect lots perceptron units together into a directed acyclic graph.

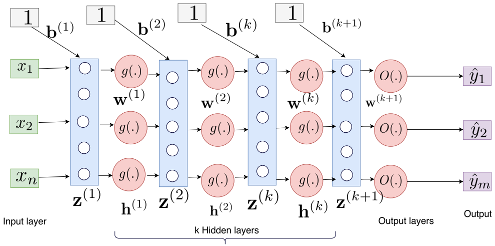


Multilayer Perceptrons (MLP)



- Consists of L multiple layers $(l_1, l_2 \dots l_L)$ of pecepron, interconnected in a feed-forward way.
- The first layer l_1 is called the input layer $\Rightarrow$ just pass the information to the next layer.
- The last layer is the ouput layer $\Rightarrow$ maps to the desired output format.
- The intermediate k layers are hidden layers $\Rightarrow$ perform computations and transfer the weights from the input layer.

Multilayer Perceptrons (MLP)



- Input:

$$\mathbf{x} = \{x_1, x_2, \ldots x_d\} \in \mathbb{R}^{(d \times N)}$$

- Pre-activation:

$$\mathbf{z^{(1)}}(\mathbf{x}) = \mathbf{b^{(1)}} + \mathbf{w^{(1)}}(\mathbf{x})$$

where $z(x)_i = \sum_j w_{i,j}^{(1)} x_j + b_i^{(1)}$

Hidden layer 1

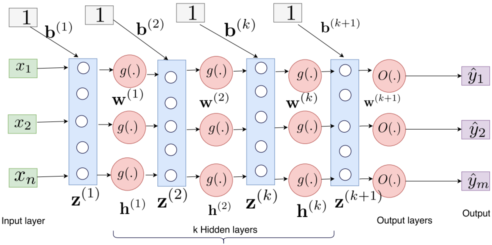
- Activation

$$\mathbf{h^{(1)}}(\mathbf{x}) = g(\mathbf{z^{(1)}}(\mathbf{x})) \\ = g(\mathbf{b^{(1)}} + \mathbf{w^{(1)}}(\mathbf{x}))$$

- Pre-activation

$$\mathbf{z^{(2)}}(\mathbf{x}) = \mathbf{b^{(2)}} + \mathbf{w^{(2)}}\mathbf{h^{(1)}}(\mathbf{x})$$

Multilayer Perceptrons (MLP)



Hidden layer 2

- Activation

$$\begin{aligned}\mathbf{h}^{(\mathbf{2})}(\mathbf{x}) &= g(\mathbf{z}^{(\mathbf{2})}(\mathbf{x})) \\ &= g(\mathbf{b}^{(\mathbf{2})} + \mathbf{w}^{(\mathbf{2})}\mathbf{h}^{(\mathbf{1})}(\mathbf{x}))\end{aligned}$$

- Pre-activation

$$\mathbf{z}^{(\mathbf{3})}(\mathbf{x}) = \mathbf{b}^{(\mathbf{3})} + \mathbf{w}^{(\mathbf{3})}\mathbf{h}^{(\mathbf{2})}(\mathbf{x})$$

Hidden layer k

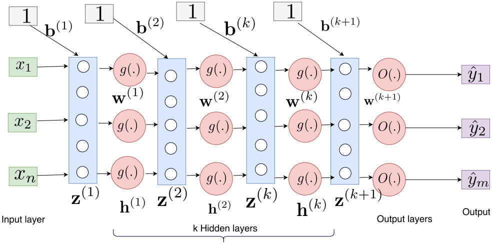
- Activation

$$\begin{aligned}\mathbf{h}^{(\mathbf{k})}(\mathbf{x}) &= g(\mathbf{z}^{(\mathbf{k})}(\mathbf{x})) \\ &= g(\mathbf{b}^{(\mathbf{k})} + \mathbf{w}^{(\mathbf{k})}\mathbf{h}^{(\mathbf{k-1})}(\mathbf{x}))\end{aligned}$$

- Pre-activation

$$\mathbf{z}^{(\mathbf{k+1})}(\mathbf{x}) = \mathbf{b}^{(\mathbf{k+1})} + \mathbf{w}^{(\mathbf{k+1})}\mathbf{h}^{(\mathbf{k})}(\mathbf{x})$$

Multilayer Perceptrons (MLP)



Output layer

- Activation

$$\mathbf{h}^{(\mathbf{k+1})}(\mathbf{x}) = O(\mathbf{z}^{(\mathbf{k+1})}(\mathbf{x}))$$
$$= O(\mathbf{b}^{(\mathbf{k+1})} + \mathbf{w}^{(\mathbf{k+1})}\mathbf{h}^{(\mathbf{k})}(\mathbf{x}))$$
$$= \hat{\mathbf{y}}$$

where $O(.)$ is output activation function

Output activation function

- Binary classification: $y \in \{0, 1\} \Rightarrow sigmoid$
- Multiclass classfication:$y \in \{0, K-1\} \Rightarrow softmax$
- Regression: $y \in \mathbb{R}^n \Rightarrow$ identity sometime RELU.

Demo Playground

MLP: Pytorch

import torch model = torch.nn.Sequential(torch.nn.Linear(2, 16), torch.nn.ReLU(), torch.nn.Linear(16, 64), torch.nn.ReLU(), torch.nn.Linear(64, 1024), torch.nn.ReLU(), torch.nn.Linear(1024, 1), torch.nn.Sigmoid())

MLP: Pytorch

import torch from torch.nn import functional as F
class MLP(torch.nn.Module): def

$_{i}nit_{(}{}^{self):super(MLP,self).}{}_{i}nit_{(}{}^{)self.fc1=torch.nn.Linear(2,16)self.fc2=torch.nn.Linear(16,64)s}$

def forward(self, x): x = F.relu(self.fc1(x)) x =
F.relu(self.fc2(x)) x = F.relu(self.fc3(x)) out =
F.sigmoid(self.out(x))
return x
model = MLP()

Outline

Training Deep neural networks

To train DNN we need:

1. Define loss function:

$$\mathcal{L}(f(\mathbf{x}^{(i)} : \theta), \mathbf{y}^{(i)})$$

2. A procedure to compute gradient $\frac{\partial J_\theta}{\partial \theta}$
3. Solve optimisation problem.

Training Deep neural networks: Define loss function

The type of Loss function is determined by the output layer of MLP.

Binary classification

Output

- Predict $y \in \{0, 1\}$
- Use sigmoid $\sigma(.)$ activation function.

$$p(y = 1|x) = \frac{1}{1 + e^{-x}}$$

Loss

- Binary cross entropy.

$$\mathcal{L}(\hat{y}, y) = y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

- pythontorch.nn.BCELoss()

Training Deep neural networks: Define loss function

Mutli class classification

Output

- Predict $y \in \{1, k\}$
- Use softmax $\sigma(.)$ activation function.

$$p(y = i|x) = \frac{\exp(x_i)}{\sum_j^k}$$

Loss

- Cross entropy.

$$\mathcal{L}(\hat{y}, y) = \sum_{i=1}^{k} y_i \log \hat{y}_i$$

- pythontorch.nn.CrossEntropyLoss()

Training Deep neural networks: Define loss function

Regression

Output

- Predict $y \in \mathbb{R}^n$
- Use identity activation function and sometime ReLU activation.

Loss

- Squared error loss.

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(y_i - \hat{y}_i)^2$$

- pythontorch.nn.MSELoss()

Training Deep neural networks: Compute Gradients

Backpropagation: a procedure that is used to compute gradients of a loss function.

- It is based on the application of the chain rule and computationally proceeds 'backwards'.

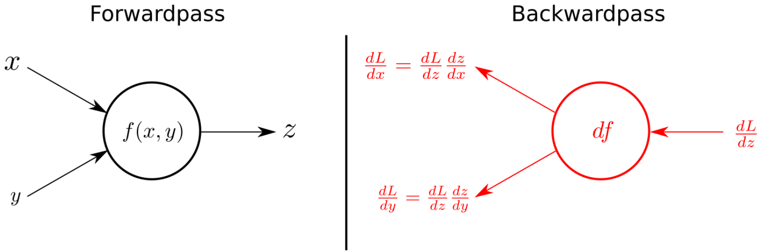
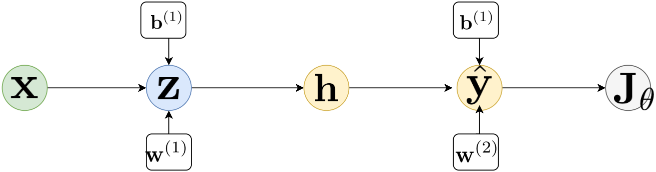


Figure 5: Back propagation: `credit: Flair of Machine Learnin`

Training Deep neural networks: Backpropagation

Consider a following single hidden layer MLP.



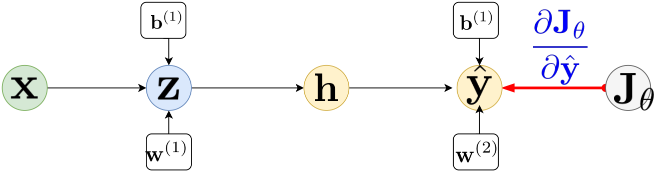
Forward path

$$\mathbf{z} = \mathbf{w^1 x} + \mathbf{b^1}$$
$$\mathbf{h} = g(\mathbf{z})$$
$$\hat{\mathbf{y}} = \mathbf{w^2 h} + \mathbf{b^2}$$
$$\mathbf{J}_\theta = \frac{1}{2}||\mathbf{y} - \hat{\mathbf{y}}||^2$$

We need to find: $\frac{\partial \mathbf{J}_\theta}{\partial \mathbf{w^{(1)}}}$, $\frac{\partial \mathbf{J}_\theta}{\partial \mathbf{b^{(1)}}}$, $\frac{\partial \mathbf{J}_\theta}{\partial \mathbf{w^{(2)}}}$ and $\frac{\partial \mathbf{J}_\theta}{\partial \mathbf{b^{(2)}}}$

Training Deep neural networks: Backpropagation

Back ward path

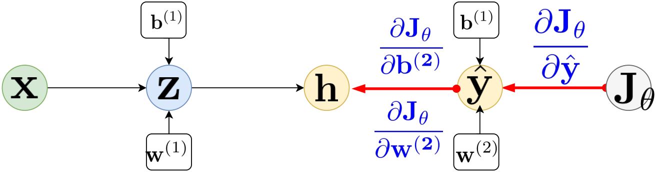


$$\mathbf{J}_\theta = \frac{1}{2}||\mathbf{y} - \hat{\mathbf{y}}||^2$$

$$\frac{\partial \mathbf{J}_\theta}{\partial \hat{\mathbf{y}}} = ||\mathbf{y} - \hat{\mathbf{y}}||$$

Training Deep neural networks: Backpropagation

Back ward path



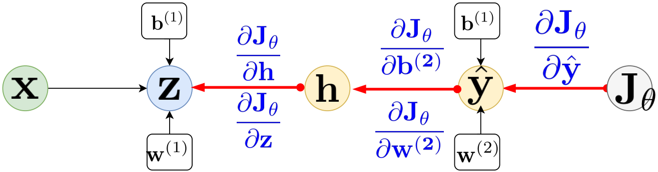
$$\hat{\mathbf{y}} = \mathbf{w^2 h} + \mathbf{b^2}$$

$$\frac{\partial \mathbf{J}_\theta}{\partial \mathbf{w^{(2)}}} = \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{w^{(2)}}} \cdot \frac{\partial \mathbf{J}_\theta}{\partial \hat{\mathbf{y}}} = \mathbf{h^T} \cdot ||\mathbf{y} - \hat{\mathbf{y}}||$$

$$\frac{\partial \mathbf{J}_\theta}{\partial \mathbf{b^{(2)}}} = \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{b^{(2)}}} \cdot \frac{\partial \mathbf{J}_\theta}{\partial \hat{\mathbf{y}}} = ||\mathbf{y} - \hat{\mathbf{y}}||$$

Training Deep neural networks: Backpropagation

Back ward path



$$\hat{\mathbf{y}} = \mathbf{w}^{\mathbf{2}}\mathbf{h} + \mathbf{b}^{\mathbf{2}}$$

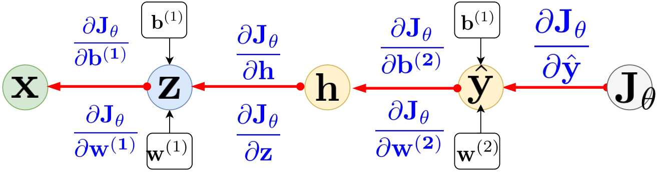
$$\mathbf{h} = g(\mathbf{z})$$

$$\frac{\partial \mathbf{J}_\theta}{\partial \mathbf{h}} = \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{J}_\theta}{\partial \hat{\mathbf{y}}} = \mathbf{w}^{(\mathbf{2})\mathbf{T}} \cdot ||\mathbf{y} - \hat{\mathbf{y}}||$$

$$\frac{\partial \mathbf{J}_\theta}{\partial \mathbf{z}} = \frac{\partial \mathbf{h}}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{J}_\theta}{\partial \mathbf{h}} = g^{'}((z)) \cdot \frac{\partial \mathbf{J}_\theta}{\partial \mathbf{h}}$$

Training Deep neural networks: Backpropagation

Back ward path



$$\mathbf{z} = \mathbf{w}^{\mathbf{1}}\mathbf{h} + \mathbf{b}^{\mathbf{1}}$$

$$\frac{\partial \mathbf{J}_\theta}{\partial \mathbf{w}^{(1)}} = \frac{\partial \mathbf{z}}{\partial \mathbf{w}^{(1)}} \cdot \frac{\partial \mathbf{J}_\theta}{\partial \mathbf{z}} = \mathbf{x}^{\mathbf{T}} \cdot \frac{\partial \mathbf{J}_\theta}{\partial \mathbf{z}}$$

$$\frac{\partial \mathbf{J}_\theta}{\partial \mathbf{b}^{(1)}} = \frac{\partial \mathbf{z}}{\partial \mathbf{b}^{(1)}} \cdot \frac{\partial \mathbf{J}_\theta}{\partial \mathbf{z}} = \frac{\partial \mathbf{J}_\theta}{\partial \mathbf{z}}$$

Training Neural Networks: Solving optimisation problem

Objective: Find parameters θ : $\mathbf{w}$ and $\mathbf{b}$ that minimize the cost function:

$$\arg\max_{\theta} \frac{1}{N} \sum_{i} \mathcal{L}(f(\mathbf{x}^{(i)} : \theta), \mathbf{y}^{(i)})$$

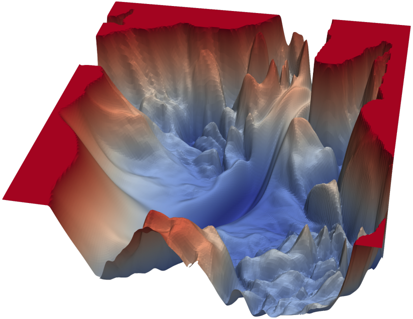


Figure 6: Visualizing the loss landscape of neural nets: *credit: Hao Li*

Training Neural Networks: Gradient Descent

Gradient Descent

1. Initilize parameter θ,
2. Loop until converge
   1. Compute gradient:
      $$\frac{\partial J_\theta}{\partial \theta}$$
   2. Update parameters:
      $$\theta^{t+1} = \theta^t - \alpha \frac{\partial J_\theta}{\partial \theta}$$
3. Retrn parameter θ

Limitation: Take time to compute

Training Neural Networks: Stochastic Gradient Descent (SGD)

SGD consists of updating the model parameters θ after every sample.

SGD

Initialize θ randomly.

For each training example:

- Compute gradients: $\frac{\partial J_{i\theta}}{\partial \theta}$
- Update parameters θ with update rule:

$$\theta^{(t+1)} := \theta^{(t)} - \alpha \frac{\partial J_{i\theta}}{\partial \theta}$$

Stop when reaching criterion

Easy to compute $\frac{\partial J_{i\theta}}{\partial \theta}$ but very noise.

Training Neural Networks: Mini-batch SGD training

Make update based on a min-batch B of example instead of single example i

Mini-batch SGD

1. Initialize θ randomly.
2. For each mini-batch B:
   - Compute gradients: $\frac{\partial J_\theta}{\partial \theta} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_{k(\theta)}}{\partial \theta}$
   - Update parameters θ with update rule: $\theta^{(t+1)} := \theta^{(t)} - \alpha \frac{\partial J_{i\theta}}{\partial \theta}$
3. Stop when reaching criterion

Fast to compute $\frac{\partial J_\theta}{\partial \theta} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_{k(\theta)}}{\partial \theta}$ and much better estimate of the true gradient.

Standard procedure for training deep learning.

Training Neural Networks: Gradient Descent Issues

Setting the learning rate α

- Small learning rate: Converges slowly and gets stuck in false local minima.
- Large learning rate: Overshoot became unstable and diverge.
- Stable learning rate: Converges smoothly and avoid local minima.

How to deal with this ?

1. Try lots of different learning rates and see what works for you.
   - Jeremy propose a technique to find stable learning rate
2. Use an adaptive learning rate that adapts to the landscape of your loss function.

Training Neural Networks: Adaptive Learning rates algorithm

1. Momentum
2. Adagrad
3. Adam
4. RMSProp

pytorch optimer algorithms

Outline

Deep learning in Practice: Regularization

Regularization: Technique to help deep learning network perform better on unsee data.

- Constraints optimization problem to discourage complex model.

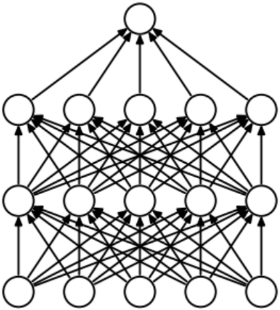
$$\arg\max_{\theta} \frac{1}{N} \sum_{i} \mathcal{L}(f(\mathbf{x}^{(i)} : \theta), \mathbf{y}^{(i)}) + \lambda\Omega(\theta)$$

- Improve generalization of deep learning model.

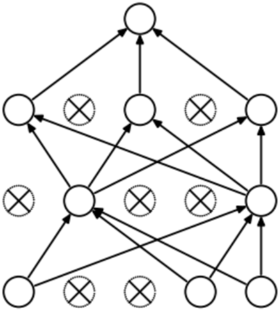
Regularization 1: Dropout

Dropout: Randomly remove hidden unit from a layer during training step and put them back during test.

- Each hidden unit is set to 0 with probability p.
- Force network to not rely on any hidden node $\Rightarrow$ prevent neural net from ovefitting (improve performance).
- Any dropout probability can be used but 0.5 usually works well.



(a) Standard Neural Net



(b) After applying dropout.

Regularization 1: Dropout

Dropout: in pytorch is implemented as pythontorch.nn.Dropout

If we have a network: model = torch.nn.Sequential(torch.nn.Linear(1,100), torch.nn.ReLU(), torch.nn.Linear(100,50), torch.nn.ReLU(), torch.nn.Linear(50,2)) We can simply add dropout layers: model = torch.nn.Sequential(torch.nn.Linear(1,100), torch.nn.ReLU(), torch.nn.Dropout() torch.nn.Linear(100,50), torch.nn.ReLU(), torch.nn.Dropout() torch.nn.Linear(50,2))

Note: A model using dropout has to be set in train or eval model.

Regularization 1: Dropout

Dropout: in pytorch is implemented as pythontorch.nn.Dropout

If we have a network: model = torch.nn.Sequential(torch.nn.Linear(1,100), torch.nn.ReLU(), torch.nn.Linear(100,50), torch.nn.ReLU(), torch.nn.Linear(50,2)) We can simply add dropout layers: model = torch.nn.Sequential(torch.nn.Linear(1,100), torch.nn.ReLU(), torch.nn.Dropout() torch.nn.Linear(100,50), torch.nn.ReLU(), torch.nn.Dropout() torch.nn.Linear(50,2))

Note: A model using dropout has to be set in train or eval model.

Regularization 2: Early Stopping

Early Stopping: Stop training before the model overfit.

- Monitor the deep learning training process from overfiting.
- Stop training when validation error increases.

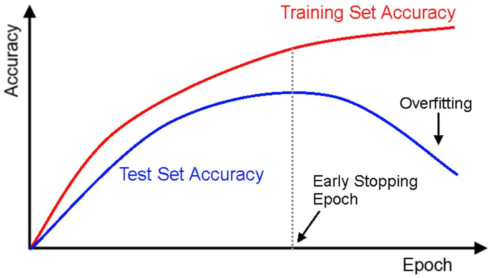


Figure 7: Early stopping: `credit: Deeplearning4j.com`

Deep learning in Practice: Batch Normalization

Batch normalisation: A technique for improving the performance and stability of deep neural networks.

Training deep neural network is complicated

- The input of each layer changes as the parameter of the previous layer change.
- This slow down the training $\Rightarrow$ require low learning rate and careful parameter initilization.
- Make hard to train models with saturation non-linearity.
- This phenomena is called Covariate shift

To address covariate shift $\Rightarrow$ normalise the inputs of each layer for each mini-batch (Batch normalization)

- To have a mean output activation of zero and standard deviation of one.

Deep learning in Practice: Batch Normalization

Batch normalisation: A technique for improving the performance and stability of deep neural networks.

Training deep neural network is complicated

- The input of each layer changes as the parameter of the previous layer change.
- This slow down the training $\Rightarrow$ require low learning rate and careful parameter initilization.
- Make hard to train models with saturation non-linearity.
- This phenomena is called Covariate shift

To address covariate shift $\Rightarrow$ normalise the inputs of each layer for each mini-batch (Batch normalization)

- To have a mean output activation of zero and standard deviation of one.

Deep learning in Practice: Batch Normalization

If $x_1, x_2, \ldots x_B$ are the sample in the batch with mean $\hat{\mu}_b$ and variance $\hat{\sigma}_b^2$.

- During training batch normalization shift and rescale each component of the input according to batch statistics to produce output y_b:

$$y_b = \gamma \odot \frac{x_b - \hat{\mu}_b}{\sqrt{\hat{\sigma}_b^2 + \epsilon}} + \beta$$

  where

  - $\odot$ is the Hadamard component-wise product.
  - The parameter γ and β are the desired moments which are either fixed or optimized during training.
- As for dropout the model behave differently during training and test.

Deep learning in Practice: Batch Normalization

Batch Normalization: in pytorch is implemented as pythontorch.nn.BatchNorm1d

If we have a network: model = torch.nn.Sequential(torch.nn.Linear(1,100), torch.nn.ReLU(), torch.nn.Linear(100,50), torch.nn.ReLU(), torch.nn.Linear(50,2)) We can simply add batch normalization layers: model = torch.nn.Sequential(torch.nn.Linear(1,100), torch.nn.ReLU(), torch.nn.BatchNorm1d(100) torch.nn.Linear(100,50), torch.nn.ReLU(), torch.nn.BatchNorm1d(50) torch.nn.Linear(50,2)) Note: A model using batch has to be set in train or eval model.

Deep learning in Practice: Batch Normalization

Batch Normalization: in pytorch is implemented as pythontorch.nn.BatchNorm1d

If we have a network: model = torch.nn.Sequential(torch.nn.Linear(1,100), torch.nn.ReLU(), torch.nn.Linear(100,50), torch.nn.ReLU(), torch.nn.Linear(50,2)) We can simply add batch normalization layers: model = torch.nn.Sequential(torch.nn.Linear(1,100), torch.nn.ReLU(), torch.nn.BatchNorm1d(100) torch.nn.Linear(100,50), torch.nn.ReLU(), torch.nn.BatchNorm1d(50) torch.nn.Linear(50,2)) Note: A model using batch has to be set in train or eval model.

Deep learning in Practice: Batch Normalization

When Applying Batch Normalization

- Carefully shuffle your sample.
- Learning rate can be greater.
- Dropout is not necessary.
- L^2 regularization influence should be reduced.

Deep learning in Practice: Weight Initilization

Before training the neural network you have to initialize its parameters.

Set all the initial weights to zero

- Every neuron in the network will computes the same output $\Rightarrow$ same gradients.
- Not recommended

Deep learning in Practice: Weight Initilization

Random Initilization

- Initilize your network to behave like zero-mean standard gausian function.

$$w_i \sim \mathcal{N}\left(\mu = 0, \sigma = \sqrt{\frac{1}{n}}\right)$$

$$b_i = 0$$

where n is the number of inputs.

Deep learning in Practice: Weight Initilization

Random Initilization: Xavier initilization

- Initilize your network to behave like zero-mean standard gausian function such that

$$w_i \sim \mathcal{N}\left(\mu = 0, \sigma = \sqrt{\frac{1}{n_{in} + n_{out}}}\right)$$
$$b_i = 0$$

where n_{in}, n_{out} are the number of units in the previous layer and the next layer respectively. where n is the number of inputs.

Deep learning in Practice: Weight Initilization

Random Initilization: Kaiming

- Random initilization that take into account ReLU activation function.

$$w_i \sim \mathcal{N}\left(\mu = 0, \sigma = \sqrt{\frac{2}{n}}\right)$$

$$b_i = 0$$

  - Recommended in practise.

Deep learning in Practice: Pytorch Parameter Initilization

Consider the previous model: model = torch.nn.Sequential(
torch.nn.Linear(1,100),
torch.nn.ReLU(),
torch.nn.BatchNorm1d(100)
torch.nn.Linear(100,50),
torch.nn.ReLU(),
torch.nn.BatchNorm1d(50)
torch.nn.Linear(50,2))

To apply weight initilization to nn.linear module.

def weights$_{i}nit(m):$
$ifisinstance(m, nn.Linear): size = m.weight.size()n_out = size[0]n_in = size[1]variance = np.sqrt(2.0/(n_in + n_out))m.weight.data.normal_(0.0, variance$
model.apply(weights$_{i}nit)$

Outline

Deep learning Architecture: Convolutional Neural Network

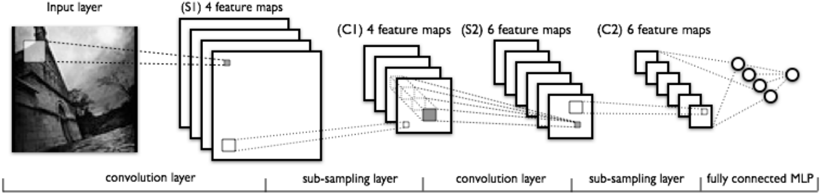
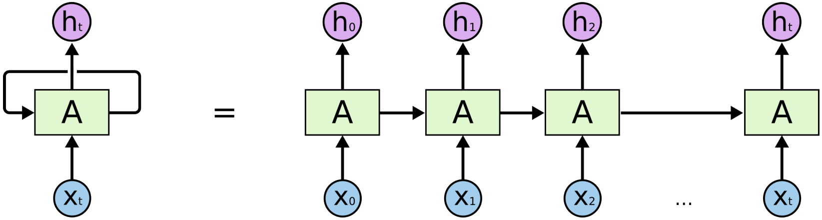


Figure 8: CNN [credit:deeplearning.net]

- Enhances the capabilities of MLP by inserting convolution layers.
- Composed of many "filters", which convolve, or slide across the data, and produce an activation at every slide position
- Suitable for spatial data, object recognition and image analysis.

Deep learning Architecture: Recurrent Neural Networks (RNN)

RNN are neural networks with loops in them, allowing information to persist.



- Can model a long time dimension and arbitrary sequence of events and inputs.
- Suitable for sequenced data analysis: time-series, sentiment analysis, NLP, language translation, speech recognition etc.
- Common type: LSTM and GRUs.

Deep learning Architecture: Auto-enceoder

Autoenceoder:A neural network where the input is the same as the output.

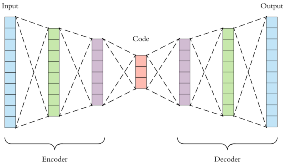


Figure 9: credit:Arden Dertat

- They compress the input into a lower-dimensional code and then reconstruct the output from this representation.
- It is an unsupervised ML algorithm similar to PCA.
- Several types exist: Denoising autoencoder, Sparse autoencoder.

Deep learning Architecture: Auto-enceoder

Autoencoder consists of components: encoder, code and decoder.

- The encoder compresses the input and produces the code,
- The decoder then reconstructs the input only using this code.

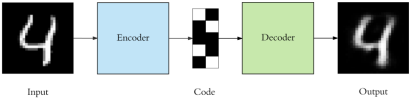


Figure 10: credit:Arden Dertat

Deep learning Architecture: Deep Generative models

Idea:learn to understand data through generation $\rightarrow$ replicate the data distribution that you give it.

- Can be used to generate Musics, Speach, Langauge, Image, Handwriting, Language
- Suitable for unsupervised learning as they need lesser labelled data to train.

Two types:

1. Autoregressive models: Deep NADE, PixelRNN, PixelCNN, WaveNet, ByteNet
2. Latent variable models: VAE, GAN.

Outline

Limitation

- Very data hungry (eg. often millions of examples)
- Computationally intensive to train and deploy (tractably requires GPUs)
- Poor at representing uncertainty (how do you know what the model knows?)
- Uninterpretable black boxes, difficult to trust
- Difficult to optimize: non-convex, choice of architecture, learning parameters
- Often require expert knowledge to design, fine tune architectures

Research Direction

- Transfer learning.
- Unsepervised machine learning.
- Computational efficiency.
- Add more reasoning (uncertatinity) abilities $\Rightarrow$ Bayesian Deep learning
- Many applications which are under-explored especially in developing countries.

Python Deep learning libraries

Tensorflow

Pytorch

Edward

Keras

Theano

MXNET

Pyro

Lab 3: Introduction to Deep learning

Part 1: Feed-forward Neural Network (MLP):

Objective: Build MLP classifier to recognize handwritten digits using the MNIST dataset.

Part 2: Weight Initilization:

Objective: Experiments with different initilization techniques (zero, xavier, kaiming)

Part 3: Regularization:

Objective: Experiments with different regulaization techniques (early stopping, dropout)

References I

- Deep learning for Artificial Intelligence master course: TelecomBCN Bercelona(winter 2017)
- 6.S191 Introduction to Deep Learning: MIT 2018.
- Deep learning Specilization by Andrew Ng: Coursera
- Deep Learning by Russ Salakhutdinov: MLSS 2017
- Introductucion to Deep learning: CMU 2018
- Cs231n: Convolution Neural Network for Visual Recognition: Stanford 2018
- Deep learning in Pytorch, Francois Fleurent: EPFL 2018
- Advanced Machine Learning Specialization: Coursera