ML6 – Neural Nets

Machine Learning – Tools and applications for policy – Lecture 7

Iman van Lelyveld – Michiel Nijhuis

DNB Data Science Hub

ML6 – Neural Nets

1. How to go from Perceptron to Neural Nets and beyond?
   - Multilayer Perceptrons (MLP), ADALine, backpropagation
2. What are common Neural Net architectures?
   - Convolutional NN, Recurrent NN, Long Short Term Memory (LSTM)
3. Other considerations
   - Fairness, adverserial attacks
4. Practical tips for implementation

Outline

- ML6 – Neural Nets
  - The original Perceptron models
  - From step activation to linear activation
  - A review of Perceptrons and Adaline
  - Multi-layer feedforward neural network
  - Other types of neural networks
  - Other considerations
    - Fairness
    - Manipulation
  - Practical advice for NNs

Knowledge clips

- Neural Networks Demystified (Stephen Welch) (link)

  Sets the stage for how to think about neural networks. The remaining videos in this series explain the details of programming a NN. This goes a further than this course but are a great lead into the topic.

- But what is a Neural Network? (3Blue1Brown) (link)

  Visualizing a NN is a great way to approach this topic. Watch the video until 13:25. The linear algebra afterwards is not key for the intuition.

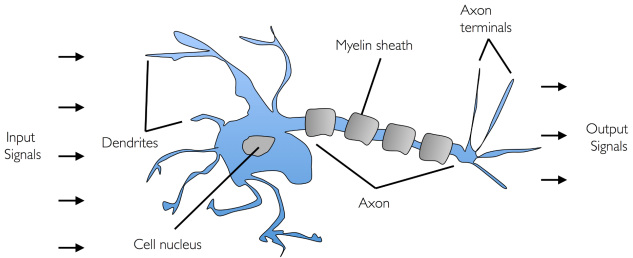
- Recurrent Neural Networks (Ava Soleimany) (link)

  This advanced video explains RNNs. Focus on the following sections:

  2:39 – Sequence modeling
  9:57 – Recurrent neural networks
  14:04 – RNN intuition
  37:36 – RNN applications

- To discuss in class: visualising neural nets at www.cybercontrols.org (link)

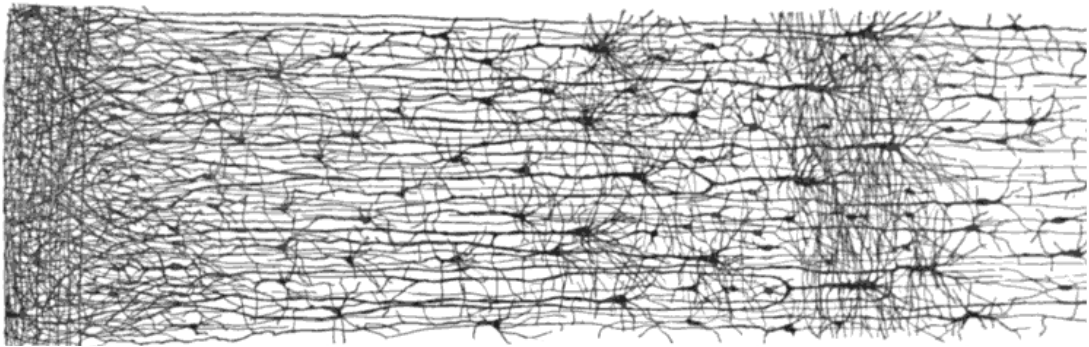
How we perceive: the Logic Gate



- Simple logic gate with binary outputs
- Signals arrive at dendrites
- If signal exceeds threshold, generate output, and pass to axon

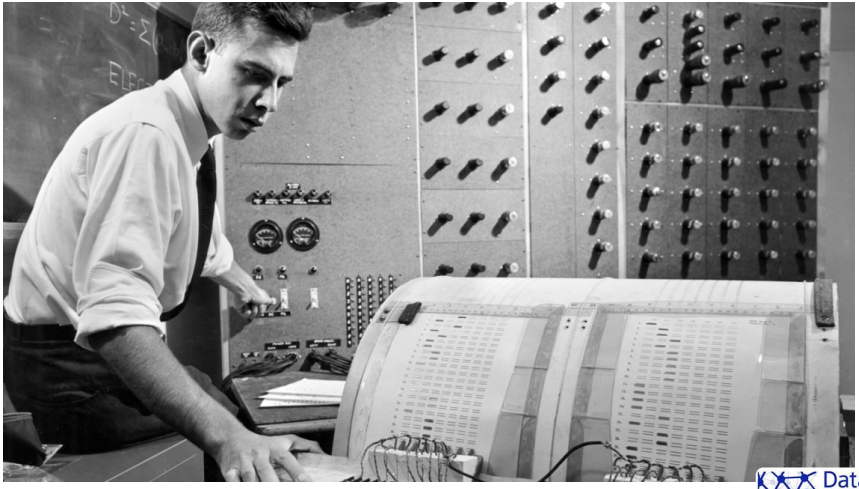
Neurons connect to form a web

Cortex of infant (45 days). “Golgi” stain shows the dendrites and axons of a random subset of neurons



Source: Santiago Ramon y Cajal - Comparative study of the sensory areas of the human cortex, 1899 Wik

Frank Rosenblatt: a AI founding father



The original: the Rosenblatt Perceptron

- Binary classification task:
  - Positive class (1) vs. negative class (-1)
- Takes as input a dot product of inputs x and weights w

$$\mathbf{w} = \begin{bmatrix} w^{(1)} \\ w^{(2)} \\ \vdots \\ w^{(m)} \end{bmatrix}, \mathbf{x} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(m)} \end{bmatrix}$$

- Net input: $z = w_1 x_1 + \cdots + w_m x_m$
- Define activation function $\phi(z)$

- $\phi(z)$ is the activation
- 2 well known (step) functions

Heaviside step function

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise .} \end{cases}$$

Sign step Function

$$\phi(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0. \end{cases}$$

Step functions shown are simplified

- To start with the step functions are defined with $z \leq \theta$
- Bring the threshold θ to the left side of the equation and define a weight-zero as $w_0 = -\theta$ and $x_0 = 1$, so that we write $\mathbf{z}$ in a more compact form

$$z = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \mathbf{w}^{\mathbf{T}} \mathbf{x}$$

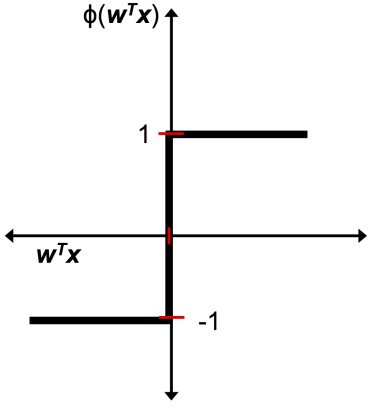
and

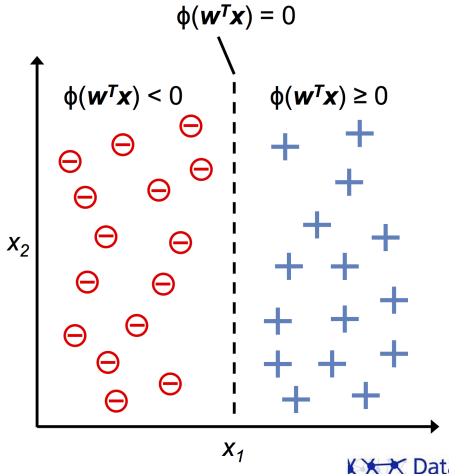
$$\phi(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ -1 & \text{otherwise .} \end{cases}$$

becomes

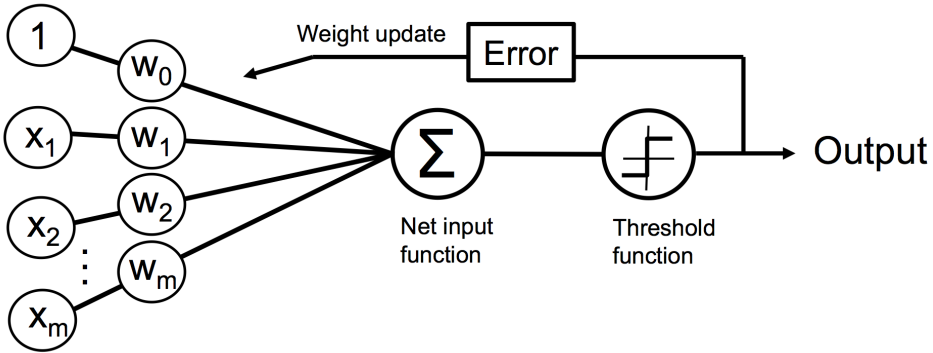
$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise .} \end{cases}$$

Input squashed into a binary output





Linear separability



iPython notebook on github

Rosenblatt perceptron algorithm

1. Initialize the weights to 0 or small random numbers.
2. For each training sample $\mathbf{x}^{(i)}$, perform the following steps:
   2.1 Compute the output value $\hat{y}^{(i)}$
   2.2 Update the weights

Weight update rule:

$$w_j := w_j + \Delta w_j$$

Perceptron learning rule:

$$\Delta w_j = \eta \left(y^{(i)} - \hat{y}^{(i)} \right) x_j^{(i)}$$

Where η is the (constant) learning rate between 0 and 1, $y^{(i)}$ is the true class label of the ith training sample, and $\hat{y}^{(i)}$ is the predicted class label.

Update rule examples

If prediction correct then weights unchanged:

$$\Delta w_j = \eta\Big(-1 - -1\Big)x_j^{(i)} = 0$$

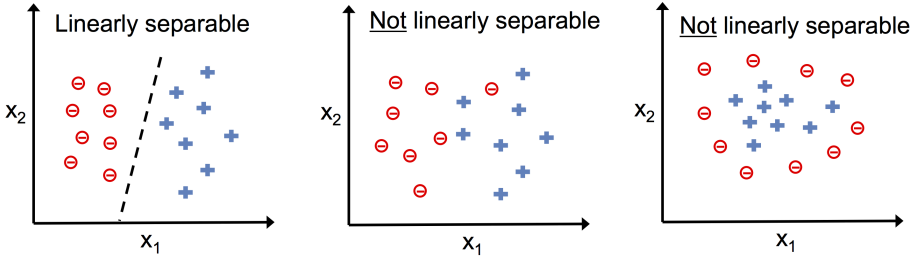
$$\Delta w_j = \eta\Big(1 - 1\Big)x_j^{(i)} = 0$$

If wrong, then weights pushed towards the positive or negative class:

$$\Delta w_j = \eta\Big(1 - -1\Big)x_j^{(i)} = \eta(2)x_j^{(i)}$$

$$\Delta w_j = \eta\Big(-1 - 1\Big)x_j^{(i)} = \eta(-2)x_j^{(i)}$$

Linear separability



Perceptron convergence

Convergence guaranteed if

- The two classes are linearly separable
- Learning rate is sufficiently small

If classes cannot be separated:

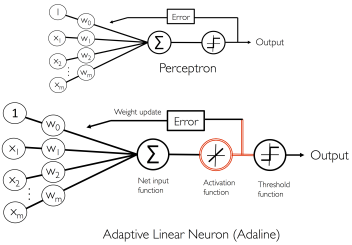
- Set a maximum number of passes over the training dataset (epochs)
- Set a threshold for the number of tolerated misclassifications
- Otherwise, it will never stop updating weights (i.e., never converge)

Outline

ML6 – Neural Nets
- The original Perceptron models
- From step activation to linear activation
- A review of Perceptrons and Adaline
- Multi-layer feedforward neural network
- Other types of neural networks
- Other considerations
  - Fairness
  - Manipulation
- Practical advice for NNs

ADAptive LInear NEuron (Adaline)

- Weights updated based on a linear activation function
  - Remember that – in contrast – the Perceptron used a unit step function
- A quantizer is then used to predict class label



Adaline vs Perceptron

- Perceptron
  - Update all weights, then recompute $\hat{y}$
  - Weight update done after seeing each sample

$$\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)}$$

- Adaline
  - Weight update done after entire training set has been seen and update all weights as follows:

$$\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}, \quad \text{where } \Delta\mathbf{w} = -\eta\nabla J(\mathbf{w})$$

  - I.e. compute the gradient based on all samples in the training set (this is known as batch gradient descent). Computationally expensive: mini-batch and stochastic gradient descent reduce the workload but result in a jittery path to the optimum

Partial derivative for each weight w_j in the weight vector $\mathbf{w}$:

$$\frac{\partial}{\partial w_j} J(\mathbf{w}) = \sum_i \left(y^{(i)} - a^{(i)}\right) x_j^{(i)}$$

Here $y^{(i)}$ is the target class label of a particular sample $x^{(i)}$, and $a^{(i)}$ is the activation of the neuron, which is a linear function in the case of Adaline: Remember that we defined the activation function $\phi(\cdot)$ as follows:

$$\phi(z) = z = a$$

Here, the net input z is a linear combination of the weights that are connecting the input to the output layer:

$$z = \sum_j w_j x_j = \mathbf{w}^T \mathbf{x}$$

Many, many activation functions

Sigmoid logistic $y = \frac{1}{1+e^{-z}}$

Tanh hyperbolic tangent $y = tanh(z)$

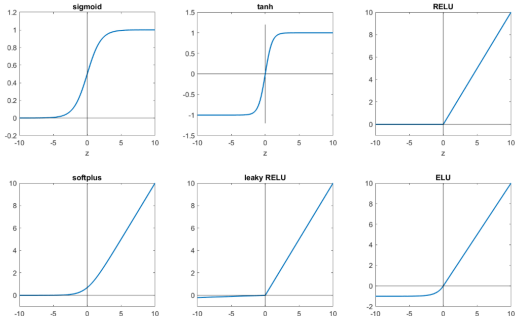
ReLU Rectified Linear Unit $y = max(z, 0)$

Softplus $y = log(1 + e^z)$

Leaky Regularization $y = max(z, \alpha z), 0 < \alpha < 1$

α typically a small number e.g. 0.01

Elu $y = \begin{cases} z & z > 0 \\ \alpha(e^z - 1) & z \leq 0 \end{cases}$



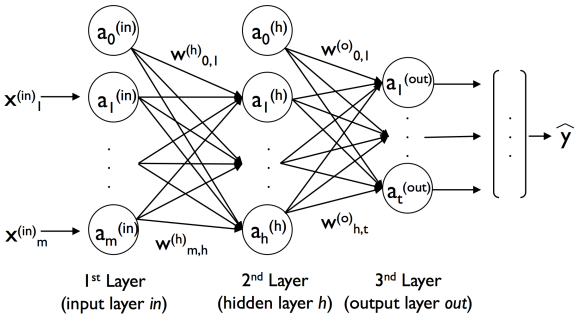
Many more functions can be found here

Experiment with different functions on Colab

Outline

ML6 – Neural Nets
- The original Perceptron models
- From step activation to linear activation
- A review of Perceptrons and Adaline
- Multi-layer feedforward neural network
- Other types of neural networks
- Other considerations
  - Fairness
  - Manipulation
- Practical advice for NNs

Multi-layer feedforward neural network



Key questions:

1. How to decide on the weights?
2. How to assess the performance?
   - Speed
   - Accuracy
3. What design to pick?
   - How many nodes? Layers? Activation functions?

DataScience Hub

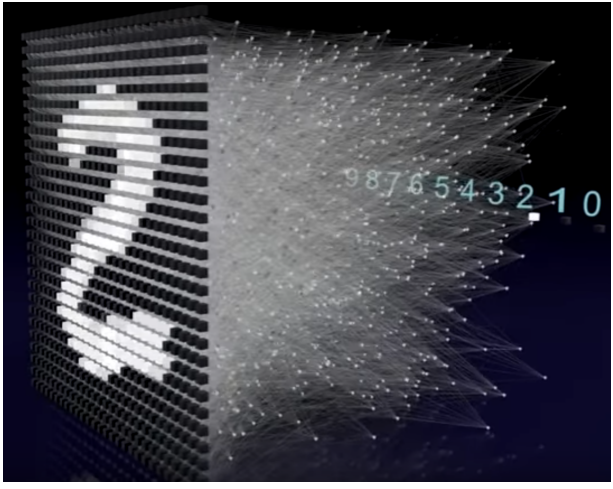
Forward and backward propagation

1. Input has m dimensions for which we use mini-batch (for speed). Each pass is called and epoch
2. Forward pass of the input through the hidden layers
   - See Welch for proofs.
3. Calculate the error that we will want to minimize
   - Network RSS = $\frac{1}{n}\Sigma_{i=1}^{n} RSS_i$
4. Using the chain rule, calculate the contribution of each connection to the error. The contributions of each instance in the epoch are then passed back through the network until you reach the input layer
   - See Welch for proofs.
5. Given the error for outcomes in an epoch, figure out the contribution to the error of each weight and then adjust the weights to improve the model

Backpropagation

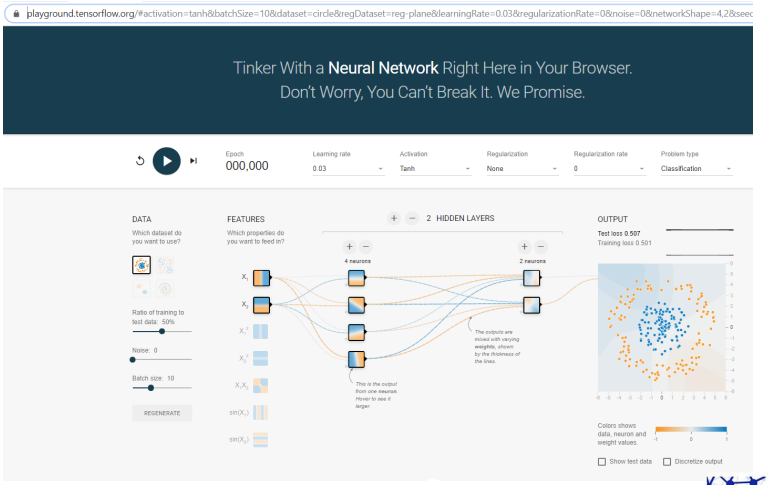
- Why would this work? It is a numerical optimization to achieve gradient descent
- Given the large number of weights in Neural Nets, an analytical solution is often not feasible
- Furthermore, with granular input data we get computational challenges
  - Cell phones make pictures with resolution 4032 x 2268 pixels ($\leq$ 9 million weights)
  - With for example 'just' 1000 neurons in the second layer this means $>$ 9 billion weights to adjust (4032*2268*1000)
  - To tackle this $\rightarrow$ dimension reduction $\equiv$ Convolutional NN

Vizualising neural nets



Source: www.cybercontrols.org

Experiment with training a Neural Net



Source: playground.tensorflow.org

DataScience Hub

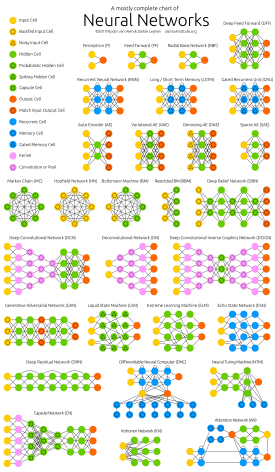
What packages to use to implement your models?

Source: Irina Popova's blog on Top 10 Python packages

DataScience Hub

Outline

Neural Network Zoo



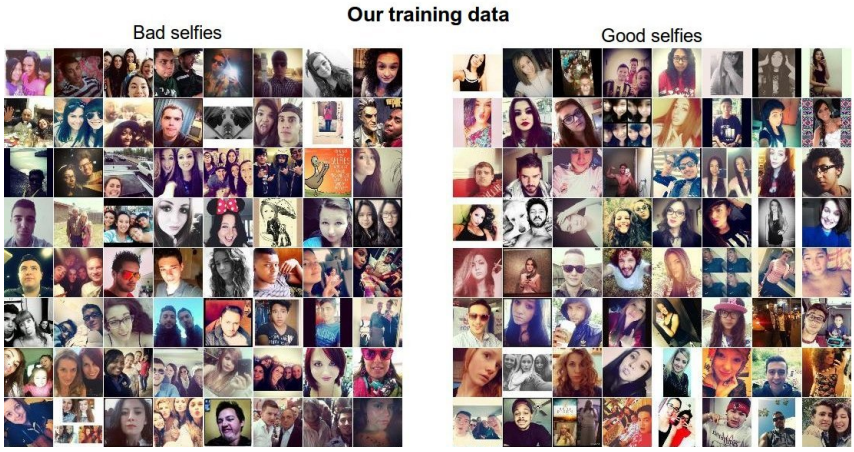
Other network architectures

1. Convolutional Neural Networks (CNN)
2. Recurrent Neural Networks (RNN)
3. Long Short-term memory (LSTM)
4. Reinforcement learning
5. and many, many more ....

1. Convolutional Neural Network (CNN)

- Often used for classifying images but also useful for time series
- Extract data features by passing multiple filters over overlapping segments of the data
  - Regularized versions of multilayer perceptrons
- Aims to reduce inputs and thus reduce the number of weights $\rightarrow$ fight the curse of dimensionality

CNN – what is a good selfie?



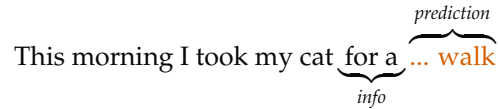
Source: Andrey Karpathy What a Deep Neural Network thinks about your #selfi

CNN – is image recognition of use in finance?

- Identify economically meaningful objects and predict demand/supply faster or better
- Sam Walton, founder of Walmart, in the 1950s used airplanes to fly over and count cars on parking lots to assess real estate investments.
- Satellite firms offer data on every asset class
  - Orbital Insights tracks car counts in parking lots for trading equities; Genscape tracks oil tankers; RezaTec provides data on wheat, corn, coffee and timber; Skywatch for oil. Maritime data on ships is provided by Windward (note that 90% of trade is transported by sea). DroneDeploy conducts aerial surveillance with drones providing data for agriculture, mining and construction industries

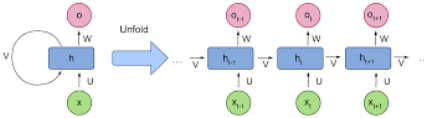
Ordered sequences

How can we predict the next item in an ordered sequence?



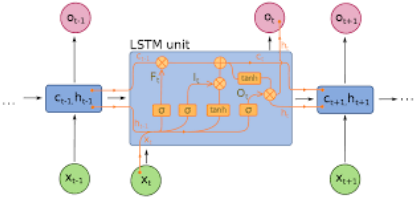
- Ordered sequences are everywhere: language, music, and ... stock prices
- This is where Recurrent Neural Networks come into play

- Remembers the internal state as it goes through "time" $\rightarrow$ can incorporate the information encoded in the ordered sequence
- Suffers from dying gradients problem but, in addition, also from exploding gradients as well



3. Long Short-Term Memory (LSTM)

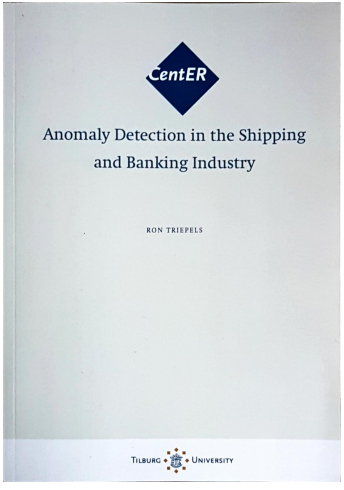
- RNN variant
- Includes feedback loops between elements. This can also simulate memory, by passing the previous signals through the same nodes. Allows to "forget" irrelevant information (e.g. stale info)
- LSTM neural networks are suitable for time series analysis because they can more effectively recognize patterns and regimes across different time periods
- Fixes exploding gradients problem by introducing gates that manage information flow
- See Ava Soleimany's lecture or Rian Dolphin's Medium article



Applications of LSTM

- Facebook: In 2017, 4.5 billion daily automatic language translations
- Google: Speech recognition in Android phones, translations on translate.google.com
- Apple: Quicktype function on iPhone
- Amazon: Alexa in speech recognition
- Microsoft: has reached error rate in speech recognition well above human transcribers
- Activity recognition/video description (CNN LSTMs)
  - Are you comfortable with a police state?
- Robotics, Self-driving cars

LSTMs in Finance



Triepels (2019)

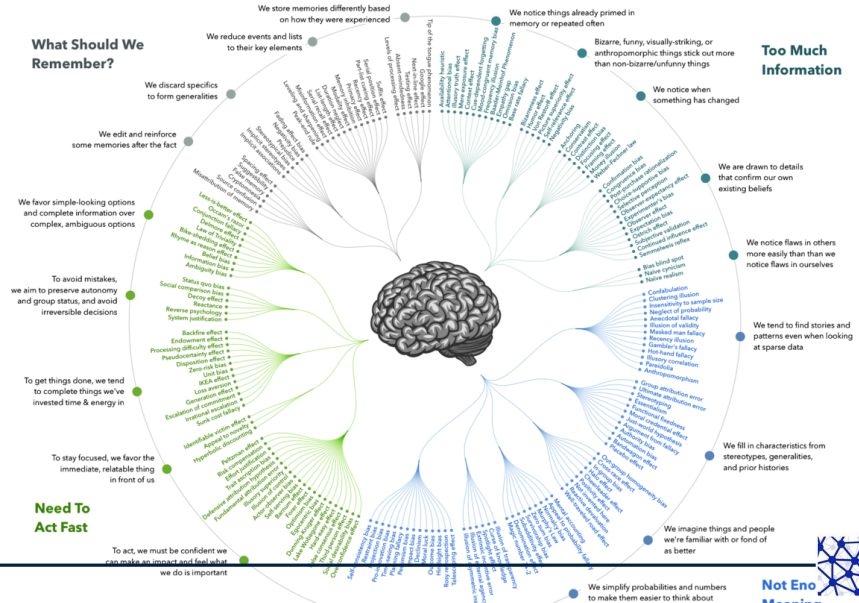
DataScience Hub

4. Reinforcement learning

- Goal is to choose a course of successive actions in order to maximize the final (or cumulative) reward
- For instance, one may look for a set of trading rules that maximizes profits after 100 trades
- Unlike supervised learning (which is typically a one-step process), the model doesn't know the correct action at each step
- At the core of reinforcement learning are two challenges that the algorithm needs to solve:
  1. Explore vs. Exploit dilemma – should the algorithm explore new alternative actions that may maximize the final reward, or stick to the established one that maximizes the immediate reward?
  2. Assigning credit problem – given that we know the final reward only at the last step, it is not straightforward to assess which step was critical for the final outcome

Outline

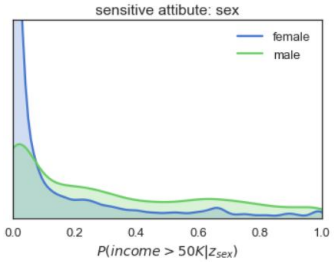
COGNITIVE BIAS CODEX



DataScience Hub

Are classifiers biased or unfair?

- Let's train a classifier to predict 'high income'
- Features are age, education level and occupation but **not** gender or race
- The resulting model will nevertheless be 'unfair': women / non-whites are less likely to be classified as high income
- This is a widespread phenomenon



Source: Stijn Tonk

Solutions

1. Add more data that does not contain the bias
   - Do you see a problem here?
2. Train a model to be immune to the sensitive variable → Generative Adversarial Networks

How to define fairness?

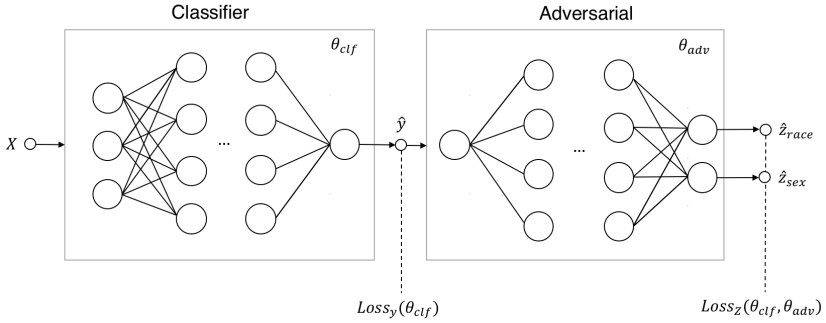
A classifier that makes a binary class prediction $\hat{y} \in [0, 1]$ given a sensitive binary attribute $g \in [0, 1]$ satisfies the p%-rule if the following inequality holds:

$$min\left(\frac{P(\hat{y}=1|g=1)}{P(\hat{y}=1|g=0)}, \frac{P(\hat{y}=1|g=0)}{P(\hat{y}=1|g=1)}\right) \leq \frac{p}{100} \text{ or, for our example}$$

$$min\left(\frac{P(\hat{y}=High|woman)}{P(\hat{y}=High|man)}, \frac{P(\hat{y}=High|man)}{P(\hat{y}=High|woman)}\right) \leq \frac{p}{100}$$

- The rule states that the ratio between the probability of a positive outcome given the sensitive attribute being true and the same probability given the sensitive attribute being false is no less than p:100
- So, when a classifier is completely fair it will satisfy a 100%-rule. In contrast, when it is completely unfair it satisfies a 0%-rule.

Could a Generative Adversarial Network (GAN) work here?



- The classifier tries to predict the class label (income) *and* make sure the adverserial gets it wrong. The adverserial's only goal is to guess gender and race.

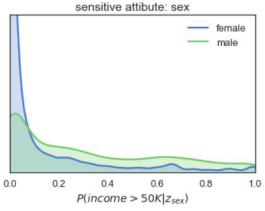
In terms of loss functions this means:

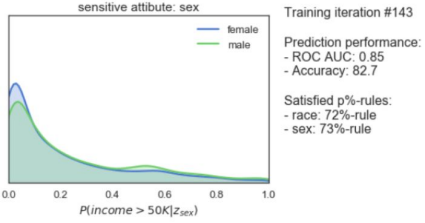
Classifier objective twofold: make the best possible income level predictions whilst ensuring that race or sex cannot be derived from them.

$\min_{\theta_{clf}} \left[Loss_y(\theta_{clf}) - \lambda Loss_Z(\theta_{clf}, \theta_{adv}) \right]$

Adversarial does not care about the classifier, only concerned with its own prediction losses on the sensitive features:

$\min_{\theta_{adv}} \left[Loss_Z(\theta_{clf}, \theta_{adv}) \right]$



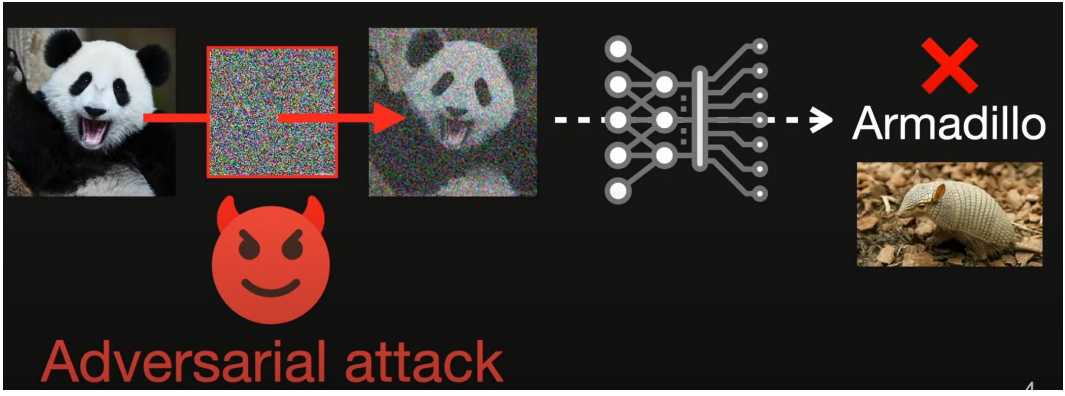


After sacrificing only 7% of prediction performance, we end up with a classifier that makes fair predictions when it comes to race and sex.

DataScience Hub

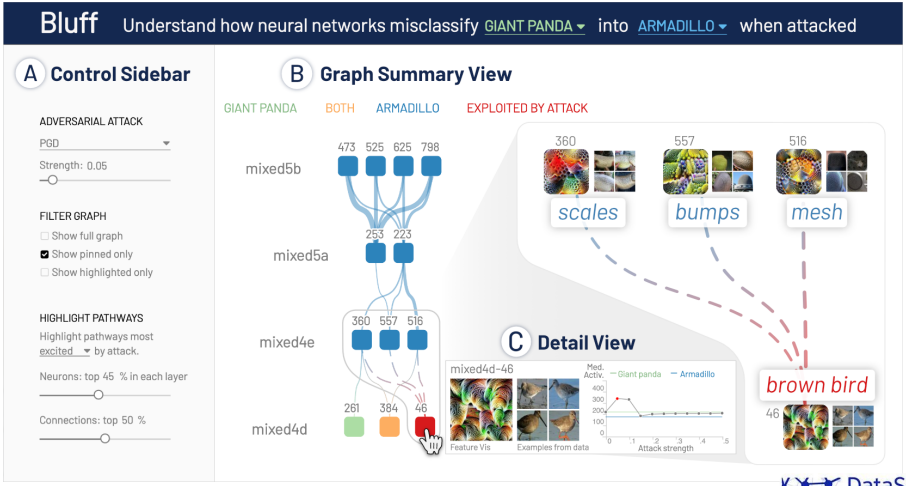
Outline

An attack can derail a Deep Neural Network



Source: Das *et al*, 2020, GitHub repo

An attack can derail a Deep Neural Network



Source: github.com/poloclub/bluff

DataScience Hub

Outline

ML6 – Neural Nets
- The original Perceptron models
- From step activation to linear activation
- A review of Perceptrons and Adaline
- Multi-layer feedforward neural network
- Other types of neural networks
- Other considerations
  - Fairness
  - Manipulation
- Practical advice for NNs

Rules of thumbs for designing neural networks

- Input layer
  - Always 1 input layer with number of neurons $\equiv$ the number of features in the training data (plus perhaps one more neuron for a bias term)
- Hidden layers
  - If data is linearly separable, we don't need any
  - Not many. 1 (perhaps 2) hidden layers are sufficient. Additional ones give little performance improvements
  - How many neurons in a hidden layer? Between the number of features and the number of neurons in the output layer. Often, the average of the two is used
- Output layers
  - If 1 continuous value is predicted (e.g., price) then 1.
  - If NN is a classifier, then one or the number of classes.
- Training deep Neural Nets is not simple. See Karpathy's excellent blog on Github

Summary

In this lecture we covered:

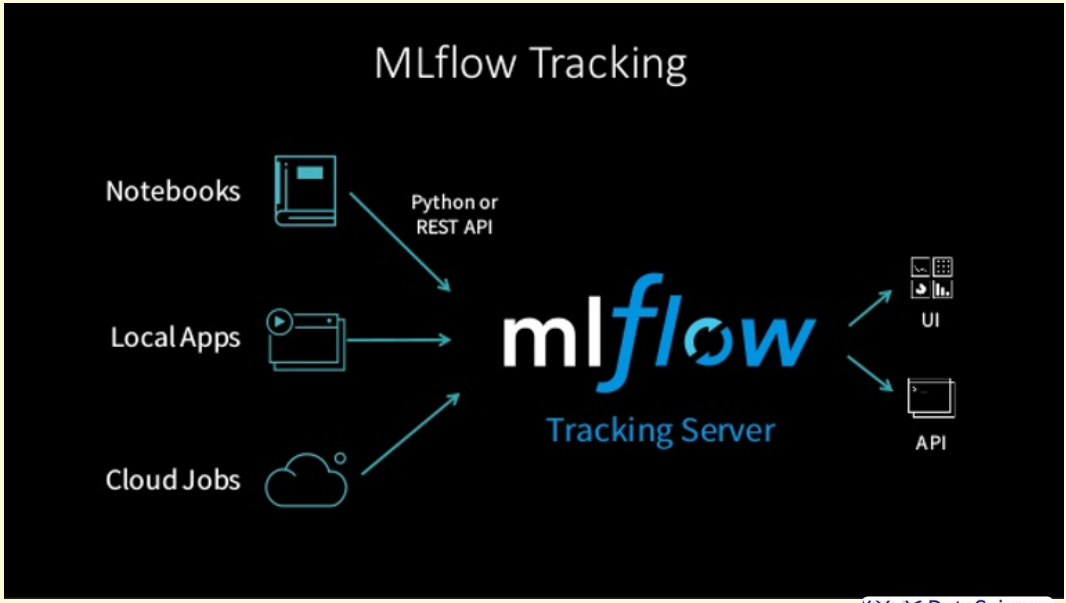
1. reviewed the Perceptron model
2. combined different learners into a neural net
3. experimented with different neural network architectures
4. discussed how ML methods could be helpful in finance

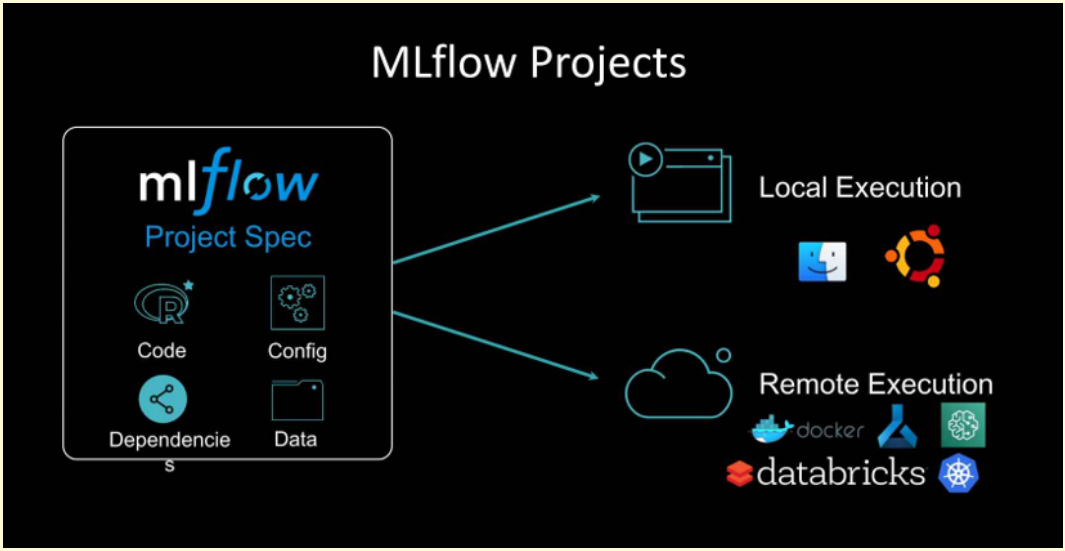
Further reading

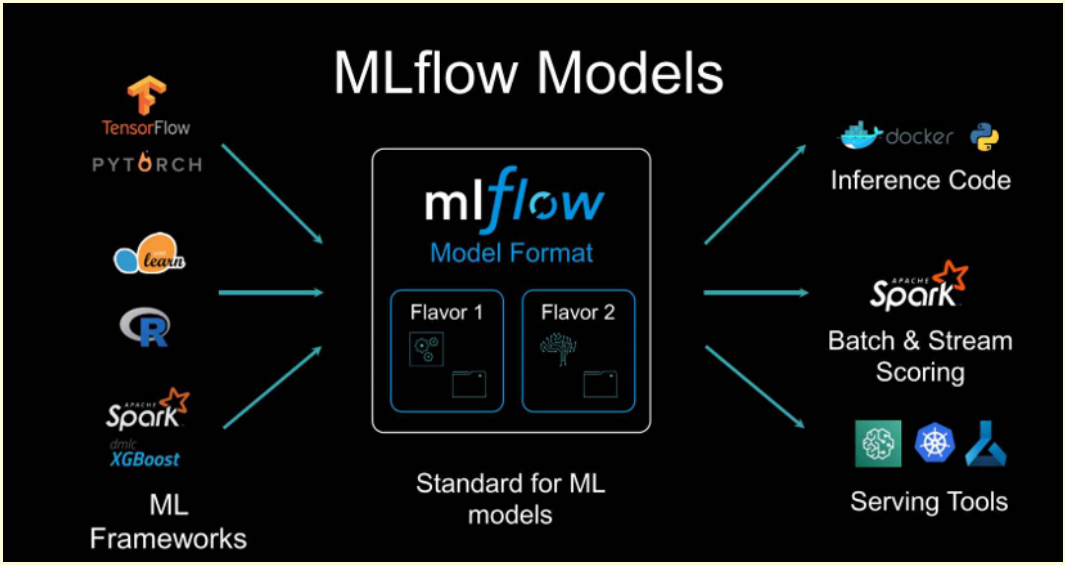
- Distributed TensorFlow using Horovod https://towardsdatascience.com/distributed-tensorflow-using-horovod-6d572f8790c4
- Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis https://arxiv.org/pdf/1802.09941.pdf
- Prace best practice guide for Deep Learning http://www.prace-ri.eu/IMG/pdf/Best-Practice-Guide-Deep-Learning.pdf
- Technologies behind Distributed Deep Learning https://preferredresearch.jp/2018/07/10/technologies-behind-distributed-deeplearning-allreduce/
- Great tool to draw neural nets

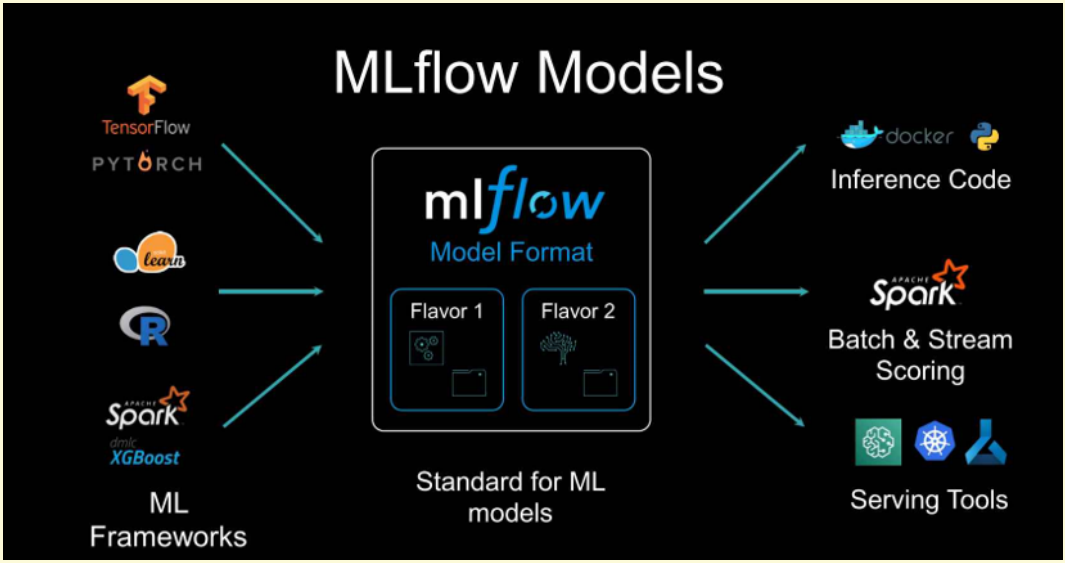
How can MLFlow help you

- Open source platform for managing end-to-end machine learning lifecycle developed by Databricks
- Four primary functions:
  1. Tracking experiments to record and compare parameters and results
  2. Packaging ML code in a reusable, reproducible form
  3. Managing and deploying models
  4. Model registry
- Has both a Python and an R API (also Java)
- A managed Mlflow version is also running in Azure




MLflow Projects
mlflow
Project Spec
Code
Config
Dependencies
Data
Local Execution
Remote Execution
docker
databricks





References I

Triepels, R. (2019). Anomaly Detection in the Shipping and Banking Industry [Doctoral dissertation].