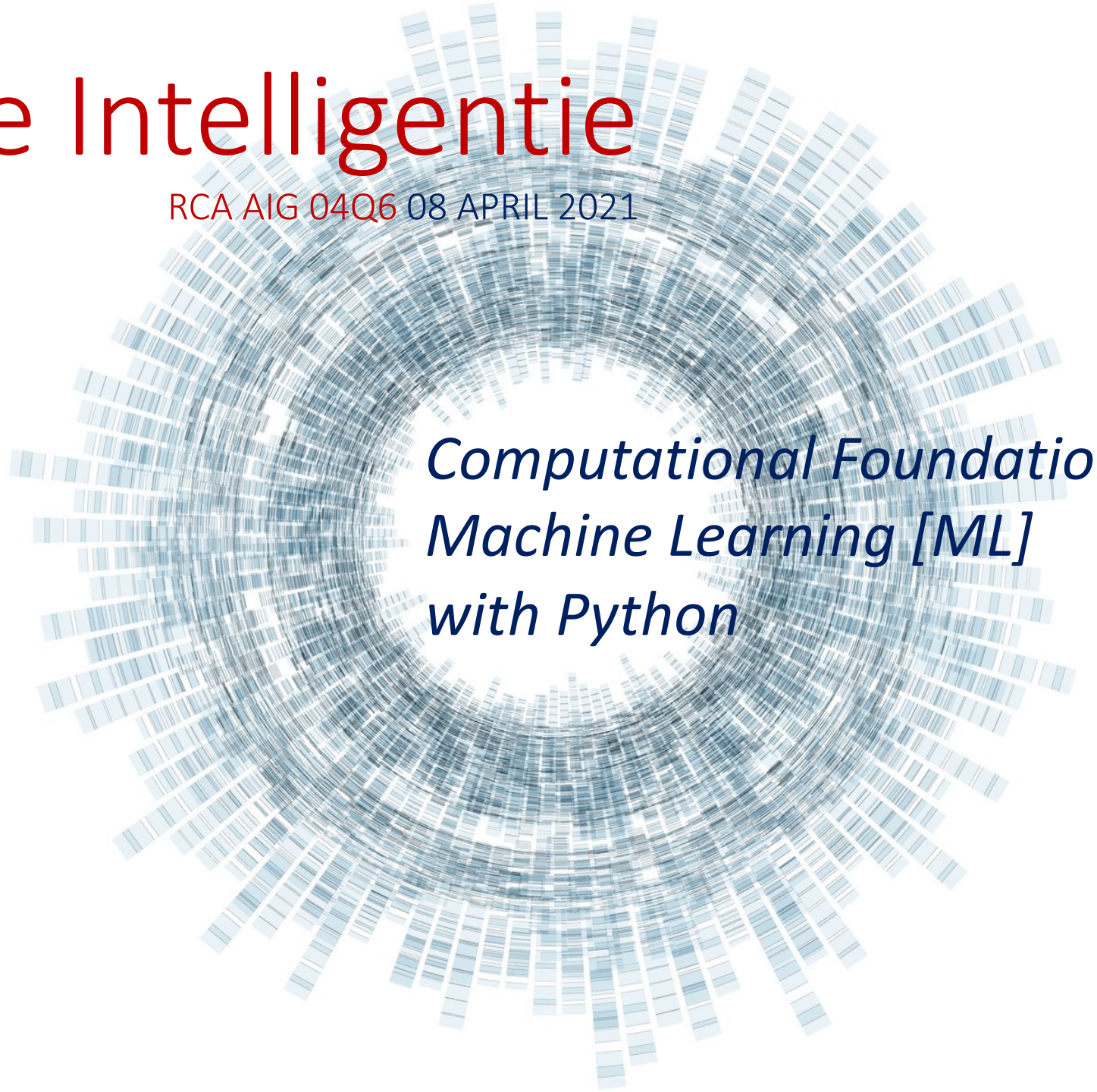
Artificiële Intelligentie

RCA AIG 04Q6 08 APRIL 2021

HOGESCHOOL ROTTERDAM



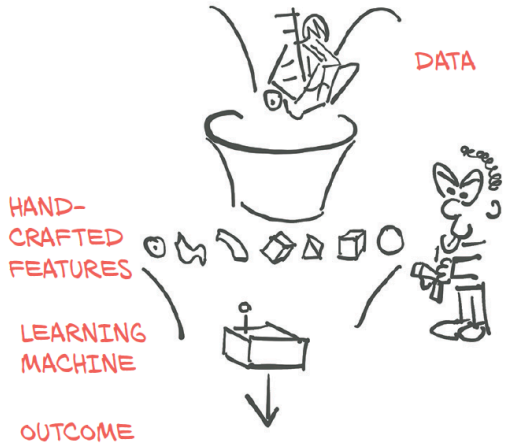
Computational Foundations of Machine Learning [ML] with Python

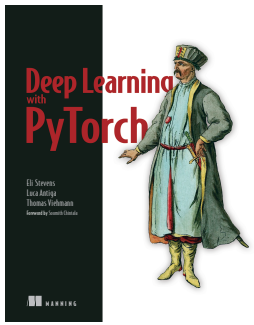
DataLab
Rob van der Willigen

MACHINE LEARNING [ML]

HOGESCHOOL ROTTERDAM

Interactive Visual experimentation with ML models & hyperparameters





CONTEXT

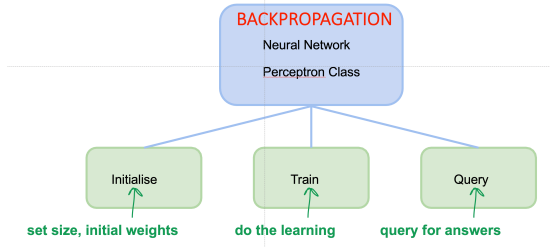
- Prerequisites: basics in linear algebra, probability, and analysis of algorithms.
- Workload: homework assignments
- GitHub: Start a ML repository at GitHub

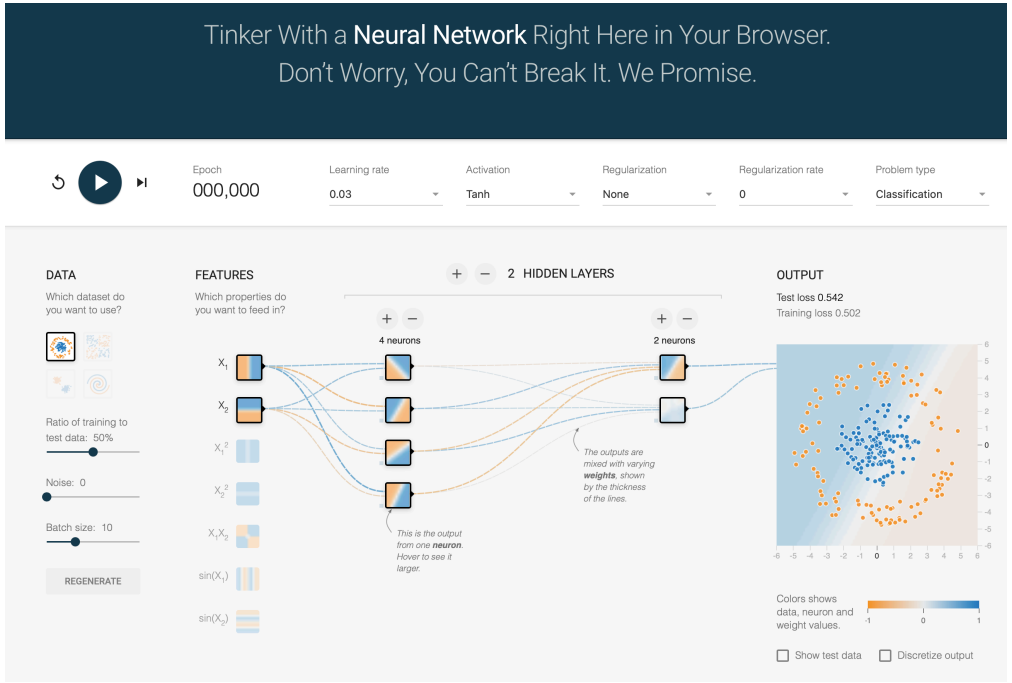
Lecture 05

- Basic definitions and concepts of Machine Learning (ML) PART03.
- How to get from ML concepts & Models to Python code.

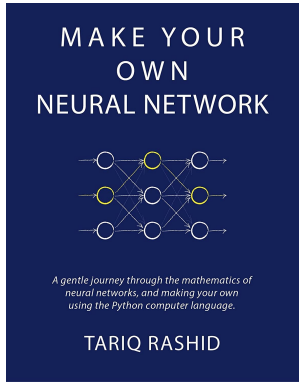
➔How to explain/design a Neural Network ←

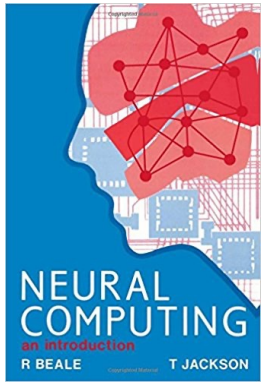
- TENSORFLOW PLAYGROUND

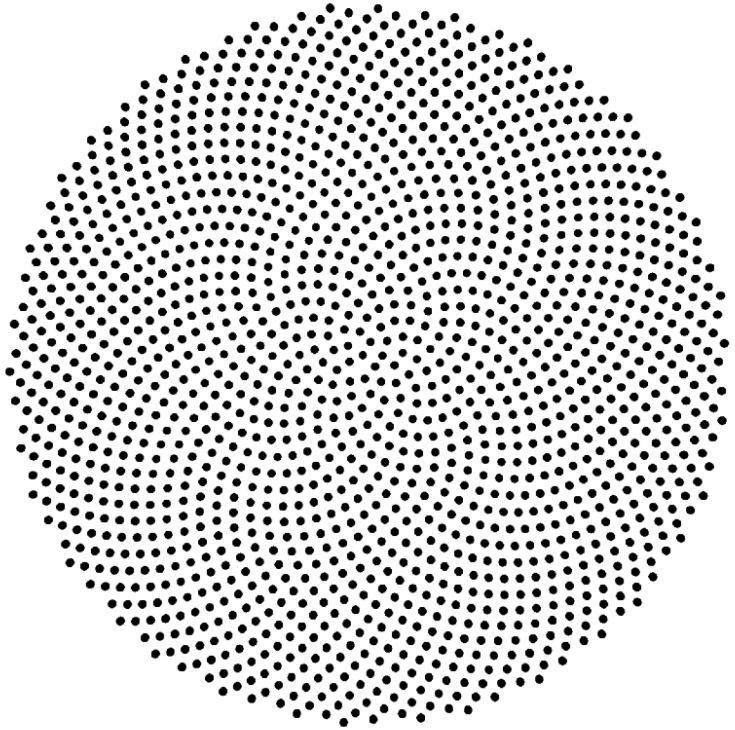




{01}
Fundamentals







HOGESCHOOL
ROTTERDAM

Understanding Neural Networks

https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/playground-exercises

https://poloclub.github.io/cnn-explainer/

Neural Networks
Need both a Cookbook & a Infrastructure

Neural networks aim to mimic the human brain
and one way to think about the brain is that
it works by accreting smaller abstractions into larger ones.

Complexity of thought, in this view,
is then measured by the range of smaller abstractions you can draw on, and the number of times you can combine lower-level abstractions into higher-level abstractions like the way we learn to distinguish dogs from birds

WIRED **Neural Networks Need a Cookbook. Here Are the Ingredients**

https://www.wired.com/story/neural-networks-need-a-cookbook-here-are-the-ingredients/

Neural Networks
Need both a Cookbook & a Infrastructure

"For a human, if you're learning how to recognize a dog you'd learn to recognize four legs, fluffy," said Maithra Raghu, a doctoral student in computer science at Cornell University and a member of Google Brain.

"Ideally we'd like our neural networks to do the same kinds of things."

WIRED **Neural Networks Need a Cookbook. Here Are the Ingredients**

https://www.wired.com/story/neural-networks-need-a-cookbook-here-are-the-ingredients/

Neural Networks
Need both a Cookbook & a Infrastructure

Abstraction comes naturally to the human brain.
Neural networks have to work for it.
As with the brain, neural networks are made of building blocks called "neurons" that are connected in various ways.

The neurons in a neural network are inspired by neurons in the brain but do not imitate them directly.

Each neuron might represent an attribute, or a combination of attributes, that the network considers at each level of abstraction.

WIRED **Neural Networks Need a Cookbook. Here Are the Ingredients**

https://www.wired.com/story/neural-networks-need-a-cookbook-here-are-the-ingredients/

Neural Networks
Need both a Cookbook & a Infrastructure

When joining these neurons together, engineers have many choices to make. They have to decide how many layers of neurons the network should have (or how "deep" it should be).

Consider, for example, a neural network with the task of recognizing objects in images.

The image enters the system at the first layer. At the next layer, the network might have neurons that simply detect edges in the image. The next layer combines lines to identify curves in the image.

Then the next layer combines curves into shapes and textures, and the final layer processes shapes and textures to reach a conclusion about what it's looking at: woolly mammoth!

WIRED **Neural Networks Need a Cookbook. Here Are the Ingredients**

https://www.wired.com/story/neural-networks-need-a-cookbook-here-are-the-ingredients/

Neural Networks
Need both a Cookbook & a Infrastructure

"The idea is that each layer combines several aspects of the previous layer. A circle is curves in many different places, a curve is lines in many different places," said David Rolnick, a mathematician at the University of Pennsylvania.

Engineers also have to decide the "width" of each layer, which corresponds to the number of different features the network is considering at each level of abstraction. In the case of image recognition, the width of the layers would be the number of types of lines, curves or shapes it considers at each level.

Beyond the depth and width of a network, there are also choices about how to connect neurons within layers and between layers, and how much weight to give each connection.

WIRED **Neural Networks Need a Cookbook. Here Are the Ingredients**

https://www.wired.com/story/neural-networks-need-a-cookbook-here-are-the-ingredients/

Neural Networks
Architecture Building blocks

Neural Networks typically consist of three layers:

•**Input layer:** This is the first layer of the neural network which passes the information to subsequent layers without performing any computational tasks.

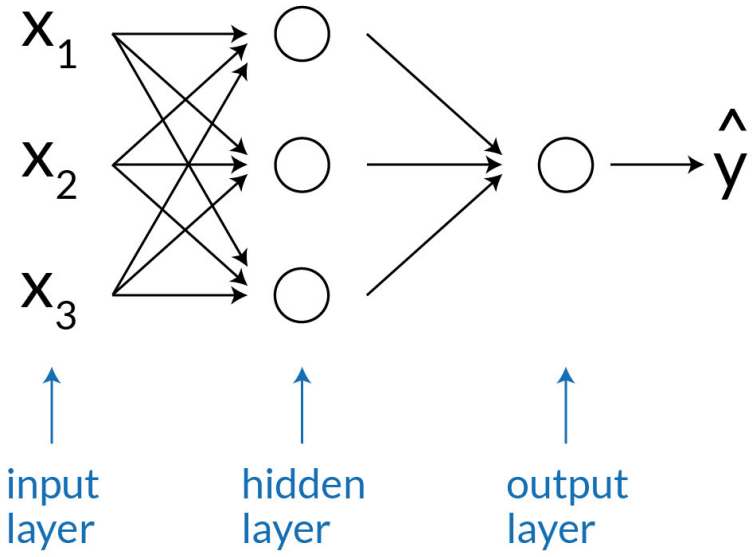
•**Hidden layer:** This consists of one or more layers and acts as the connection between the input and output layer. These layers perform all the computational work.

•**Output layer:** This layer is responsible for transferring the data to the outside world.
Each layer can have one or many nodes.

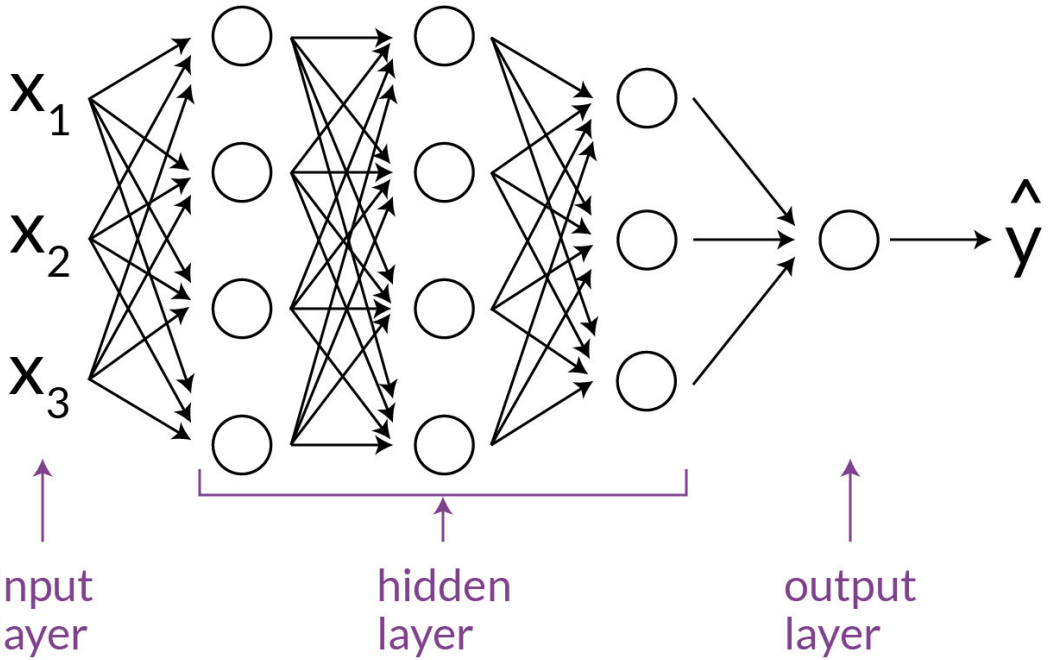
The neural network having hidden layers with one or more neurons is called a '**Deep Neural Network.**'
This type of model is used in various applications of machine learning such as machine translation, image processing, speech recognition, image recognition, etc.

Neural Networks
Need Infrastructure: Network Architecture

Shallow Neural Network



Deep Neural Network



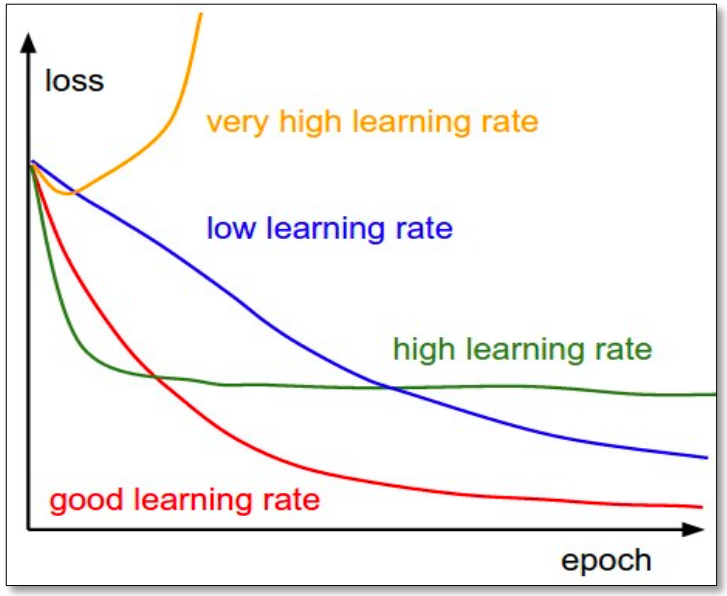
The neural network having multiple hidden layers with one or more neurons is called a '**Deep Neural Network.**' This type of model is used in various applications of machine learning such as machine translation, image processing, speech recognition, image recognition, etc.

Neural Network [hyper]Parameters

Epoch:
An epoch is one complete cycle when the entire dataset is passed forward and backward through the neural network once.

Learning rate:
Learning rate is a configurable hyperparameter (the properties/parameters that govern the whole Neural Network training process) used in the training of neural networks that has a small positive value. The learning rate ranges from 0.00001 to 10 in the Tensorflow playground.

It controls how quickly the model weights are adapted to the problem. Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epochs. If too high, we may miss some or most minima.
Result: erratic performance or never achieving a low loss.
If too low, learning will take longer than
necessary



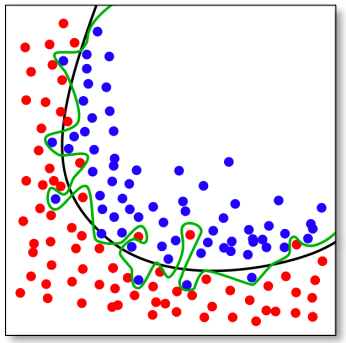
Activation function:
This is the function in an artificial neuron that delivers an output based on inputs.
ReLU, Tanh, Sigmoid, and Linear are available activation functions
in the Tensorflow playground.
Note: The output of each function is used as an input in the next function
in each epoch until we get the desired solution.

https://www.druva.com/blog/understanding-neural-networks-through-visualization/

Neural Network [hyper] Parameters

The ratio of the training and testing sets
Every given dataset is divided into categories: testing and training sets.
The training set is the set of data that is used to train the neural network to get the desired output.
The testing set is the set of data that is used to test the model prediction capability, on an unseen data set.
We usually set that ratio to 70/30 or 80/20.

Regularization type + rate:
This is a technique used for tuning the activation function by adding an additional penalty term in the error function.
The value of the regularization rate varies from 0 to 10 in the Tensorflow playground.).
Parameter to control overfitting, i.e. when the model does well on training data but poorly on new, unseen data
L2 regularization is the most common. Using dropout is another common way of controlling overfitting in neural networks. At each training stage, some hidden nodes temporarily removed (dropped out).



Features
Features are the properties of the node or neuron that we want to feed in during the artificial neural network generation. There are a number of properties to choose from in the Tensorflow Playground.

Noise
Noise is a distortion in data, that is unwanted by the perceiver of data. Adding noise to inputs is like telling the network to not change the output in a ball around your exact input.

https://www.druva.com/blog/understanding-neural-networks-through-visualization/

Neural Network Direct manipulation through the Tensorflow Playground

A student who wishes to learn about Neural Networks (NN) & Machine Learning (ML) needs to develop a mental model of not only NN architecture & ML algorithms,
but also how they affect each other in its training process.

The crux in learning about NN & ML, therefore, originates from
the iterative, dynamic, intricate interplay between NN architecture & ML algorithms.
Such complex interaction is challenging for novices to recognize, and sometimes even for experts to fully understand.

Typical NN architecture diagrams + hyper parameterization do not help students to develop the crucial mental models needed for a thorough understanding of ML-based AI.

GAN Lab: Understanding Complex Deep Generative Models using Interactive Visual Experimentation
https://doi.org/10.1109/TVCG.2018.2864500 - **https://poloclub.github.io/ganlab/**

Neural Network direct & iterative (experimental)manipulation through the Tensorflow Playground

In the following neural network (NN) manipulation experiments,
we will learn to use the **Tensorflow Playground**
as **CONCEPTUAL model** to explain, learn & build neural networks
that use **Machine Learning (ML)**
to solve REGESSION & CLASSIFICATION Tasks

https://cloud.google.com/blog/products/ai-machine-learning/understanding-neural-networks-with-tensorflow-playground

TENSORFLOW PLAYGROUND

Learning Problem (non-linear vs linear)
TENSORFLOW PLAYGROUND Problem type:

Problem type is broadly classified into two parts: **classification and regression**.

Classification problems are those problems that have **categorical output**,
e.g., an artificial neural network used to identify dog vs cat.
In these scenarios, the output is either of the two categories, i.e cat or dog.

In **regression** problems, the **output is a numerical value** ranging from starting and ending point (e.g., an artificial neural network used to identify dog vs cat, the output can also be shaped as a percentage match as dog or cat.
In this scenario, the output lies between 0–100).

TENSORFLOW PLAYGROUND

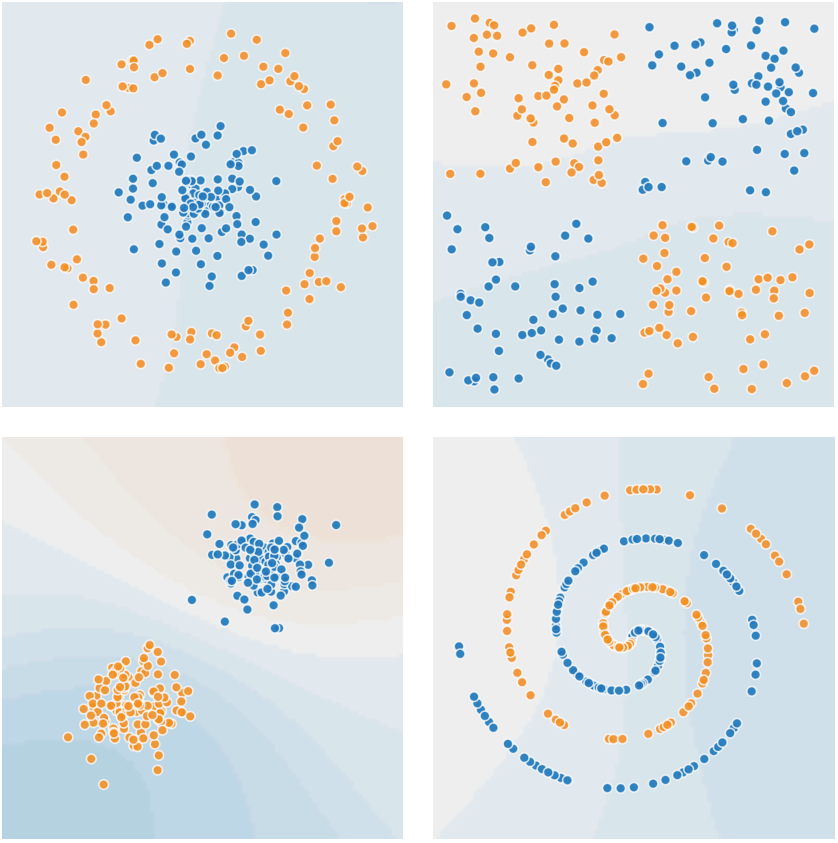
Supervised Learning Problem (non-linear vs linear)
TENSORFLOW PLAYGROUND Problem type:

Problem type is broadly classified into two parts: **classification and regression**.

The **TensorFlow Playground**
provides four different types
of Learning problems (tasks) datasets that functions as Teachers (Targets):

(1) Circles,
(2) Exclusive OR [XOR],
(3) Gaussian
(4) Spiral

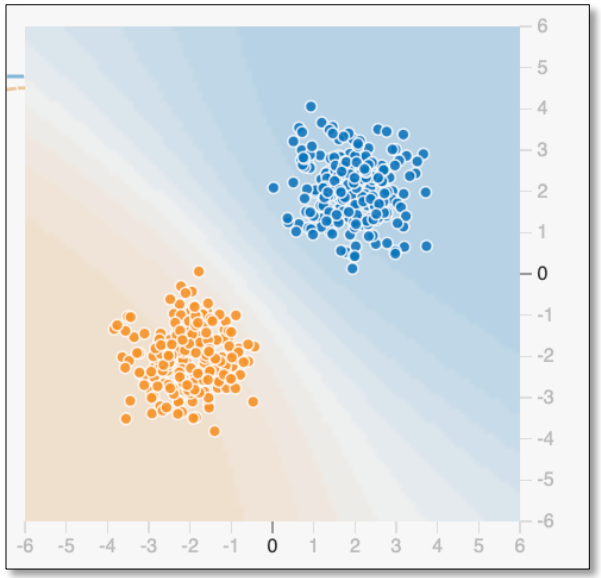
Through **trial-and-error**
we will determine which of the four learning tasks
can be **typified** as either **Classification or Regression**



Tensorflow Playground

Teacher [Target] Datasets + learning task features:

- Six datasets, each with 500 (x,y) points on a plane where x and y between -5 and +5
- Points have *labels* of positive (orange) or negative (blue)



- Two possible machine learning *tasks*:
  - Classification: Predict class of test points
  - Regression: find function to separate classes

- *Evaluation*: split dataset into training and test, e.g., 70% training, 30% test

Neural Network Tensorflow Playground hyperparameters

Input features:

X_1 Point's x value

X_2 Point's y value

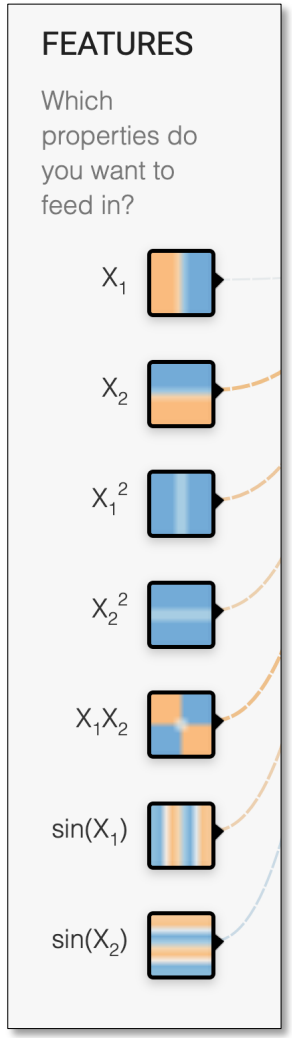
X_1^2 Point's x value squared

X_2^2 Point's y value squared

X_1X_2 Product of point's x & y values

$\sin(X_1)$ Sine of point's x value

$\sin(X_2)$ Sine of point's y value



Tensorflow Playground

Supervised Training:

- **Supervised** machine learning requires that a given neural network is trained adhereing to a learning Algor time, the tensorflowplayground uses **BackPropagation.**
- The training process is divided into a series of ***epochs***

  In each epoch, all of the training data is run through the system to adjust the NN parameters
- Process ends after a fixed # of epochs or when error rate flattens or starts increasing

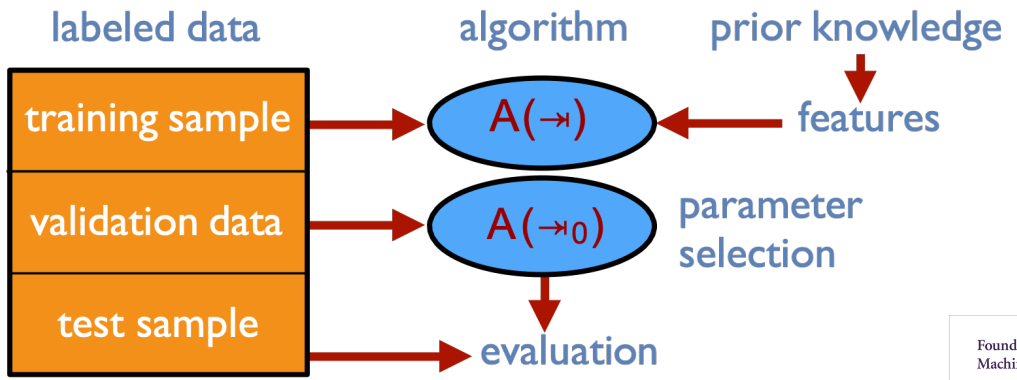
Neural Network direct & iterative (experimental)manipulation through the Tensorflow Playground

How do we find the best settings for a given learning problem/task?

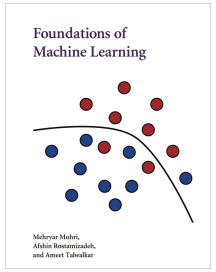
Solution:
Experiment with a range of different **hyperparameter settings** such as:

learning rate,
activation function,
amount op inputs,
number of hidden layers
Teacher (regression or classification)

via multiple runs



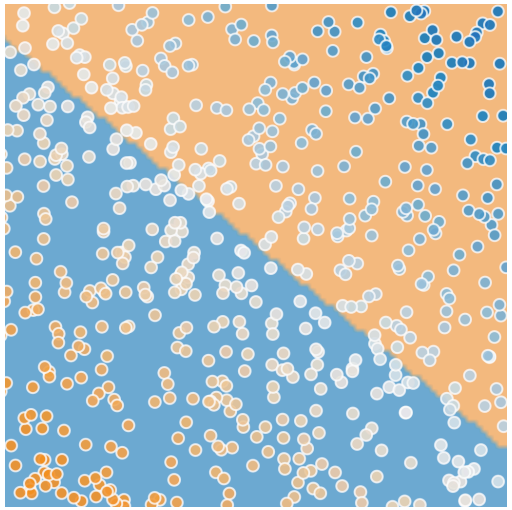
https://cloud.google.com/blog/products/ai-machine-learning/understanding-neural-networks-with-tensorflow-playground

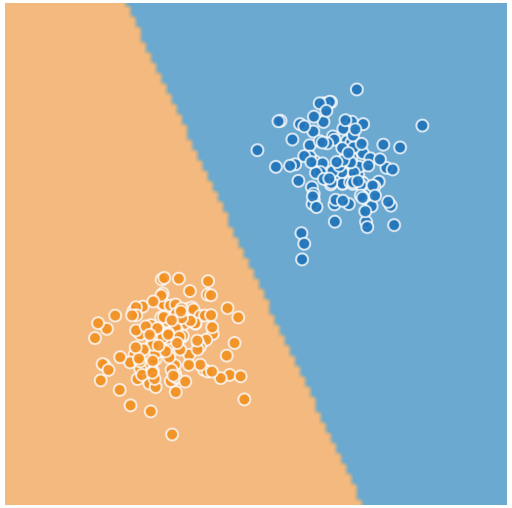


Neural Network
Experiment 01: Simulating a Perceptron

Learning Problem separate two type of data points

Let's train a neural network and see whether it can separate two groups of data points by means of a perceptron alone.



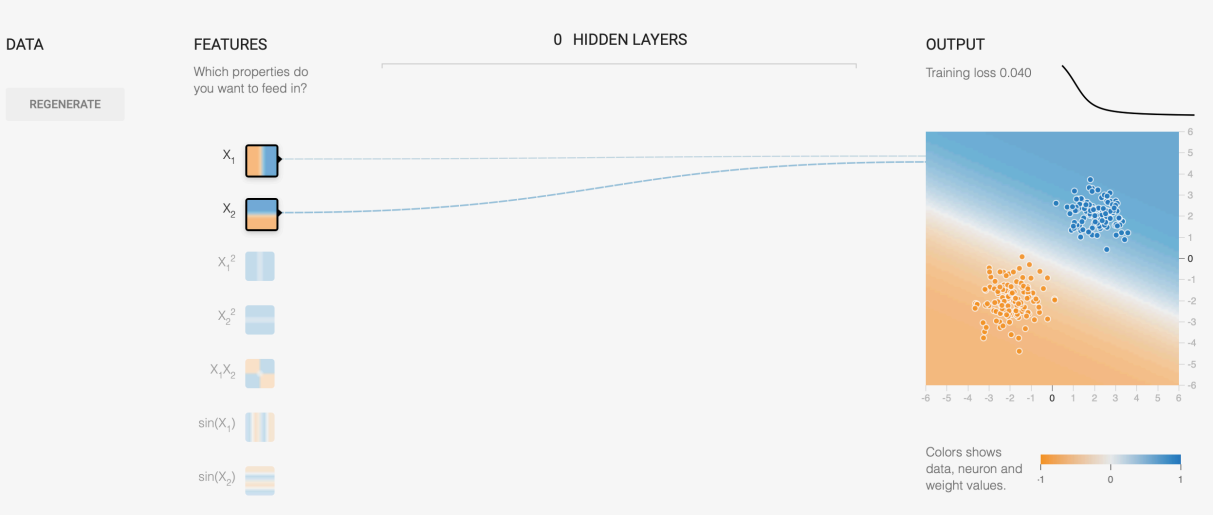


Neural Network
Experiment 01: Simulating a Perceptron

Learning Problem separate two type of data points

Let's train a neural network and see whether it can separate two groups of data points by means of a perceptron alone.

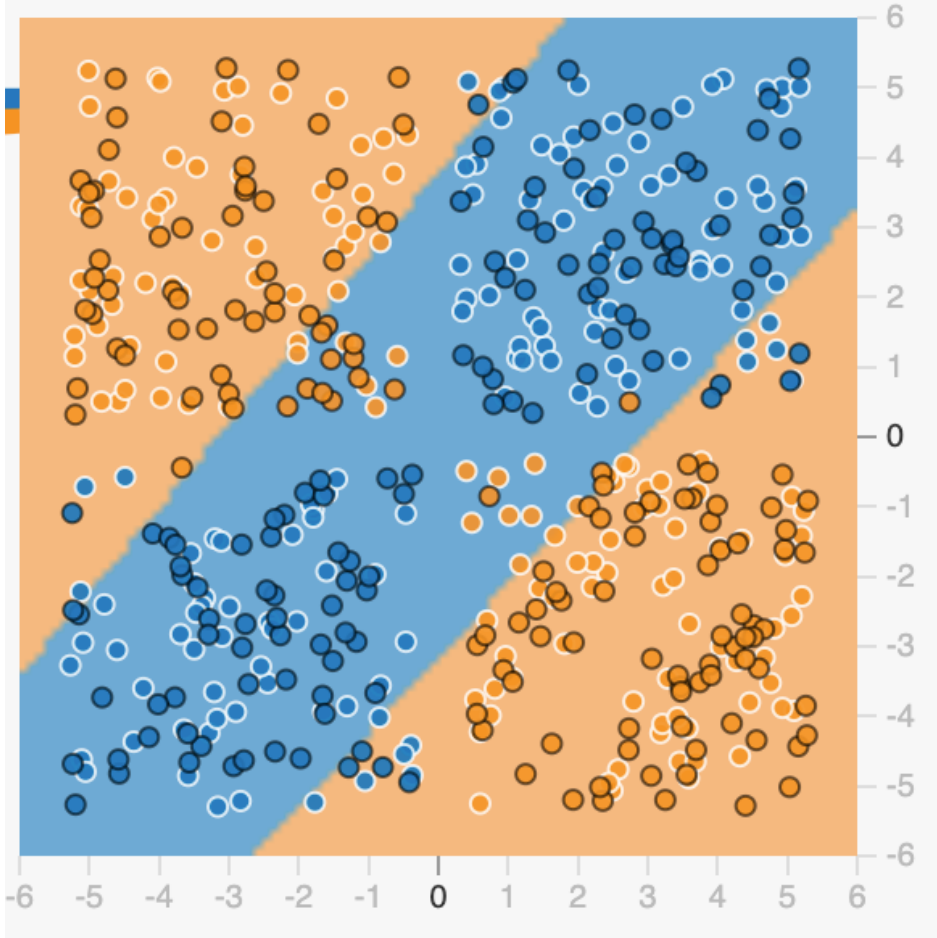
SOLUTION:



https://cloud.google.com/blog/products/ai-machine-learning/understanding-neural-networks-with-tensorflow-playground

Neural Network

Experiment 02: solve the XOR problem



Epoch
001,184

Learning rate: 0.03

Activation: Sigmoid

Regularization: L1

Regularization rate: 0

Problem type: Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 10

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

X_1

X_2

X_1^2

X_2^2

X_1X_2

$\sin(X_1)$

$\sin(X_2)$

2 HIDDEN LAYERS

2 neurons

2 neurons

*This is the output from one **neuron**. Hover to see it larger.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

OUTPUT

Test loss 0.157

Training loss 0.179

Colors shows data, neuron and weight values.

-1 0 1

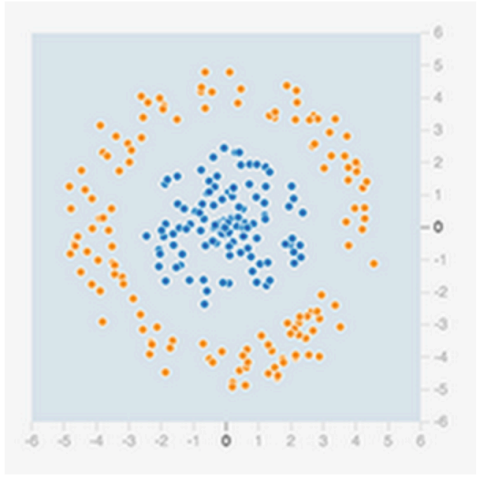
Show test data

Discretize output

Neural Network
Experiment 03: solve the CIRCLE problem

More neurons, more features to extract

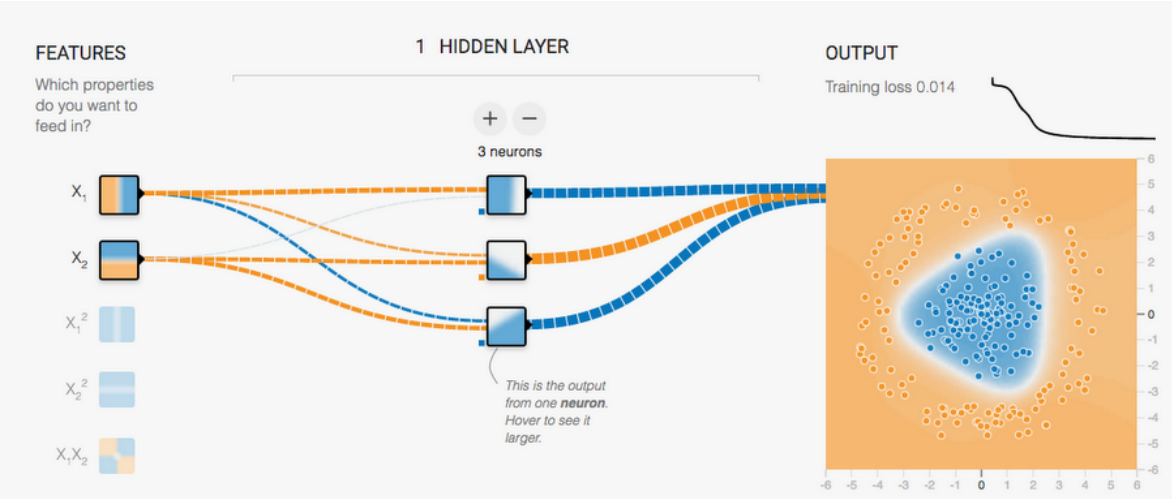
We've demonstrated how a single neuron can perform a simple classification, but you may be wondering how a simple neuron can be used to build a network that can recognize thousands of different images and compete with a professional Go player? There's a reason why neural networks can get much smarter than what we described above. Let's take a look at another example from TensorFlow Playground.



This dataset can not be classified by a single neuron, as the two groups of data points can't be divided by a single line. This is a so-called nonlinear classification problem. In the real world, there's no end to non-linear and complex datasets such as this one, and the question is how to capture these sorts of complex patterns?

https://cloud.google.com/blog/products/ai-machine-learning/understanding-neural-networks-with-tensorflow-playground

The solution is to add a hidden layer between the input values and output neuron. [Click here]() to try it out.



What's happening here? If you click each one of the neurons in the hidden layer, you see they're each doing a simple, single-line classification:

The first neuron checks if a data point is on the left or right
The second neuron checks if it's in the top right
The third one checks if it's in the bottom right

These three results are called features of the data. Outputs from these neurons indicate the strength of their corresponding features.

Finally, the neuron on the output layer uses these features to classify the data. If you draw a three dimensional space consisting of the feature values, the final neuron can simply divide this space with a flat plane.

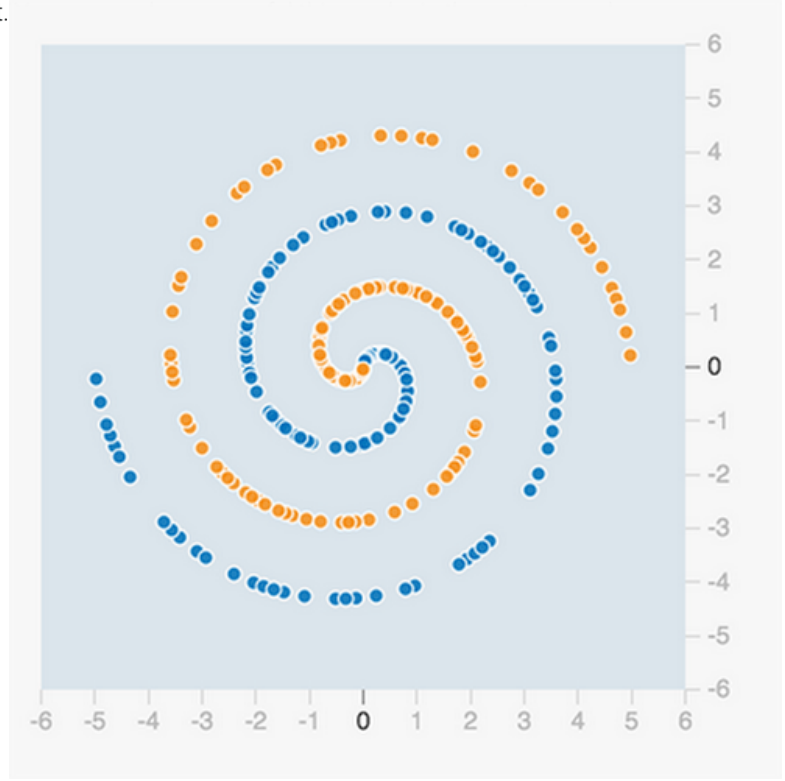
This is an example of a transformation of the original data into a feature space.

Neural Network

Experiment 04: solve the SPIRAL problem

We need to go deeper: building a hierarchy of abstractions

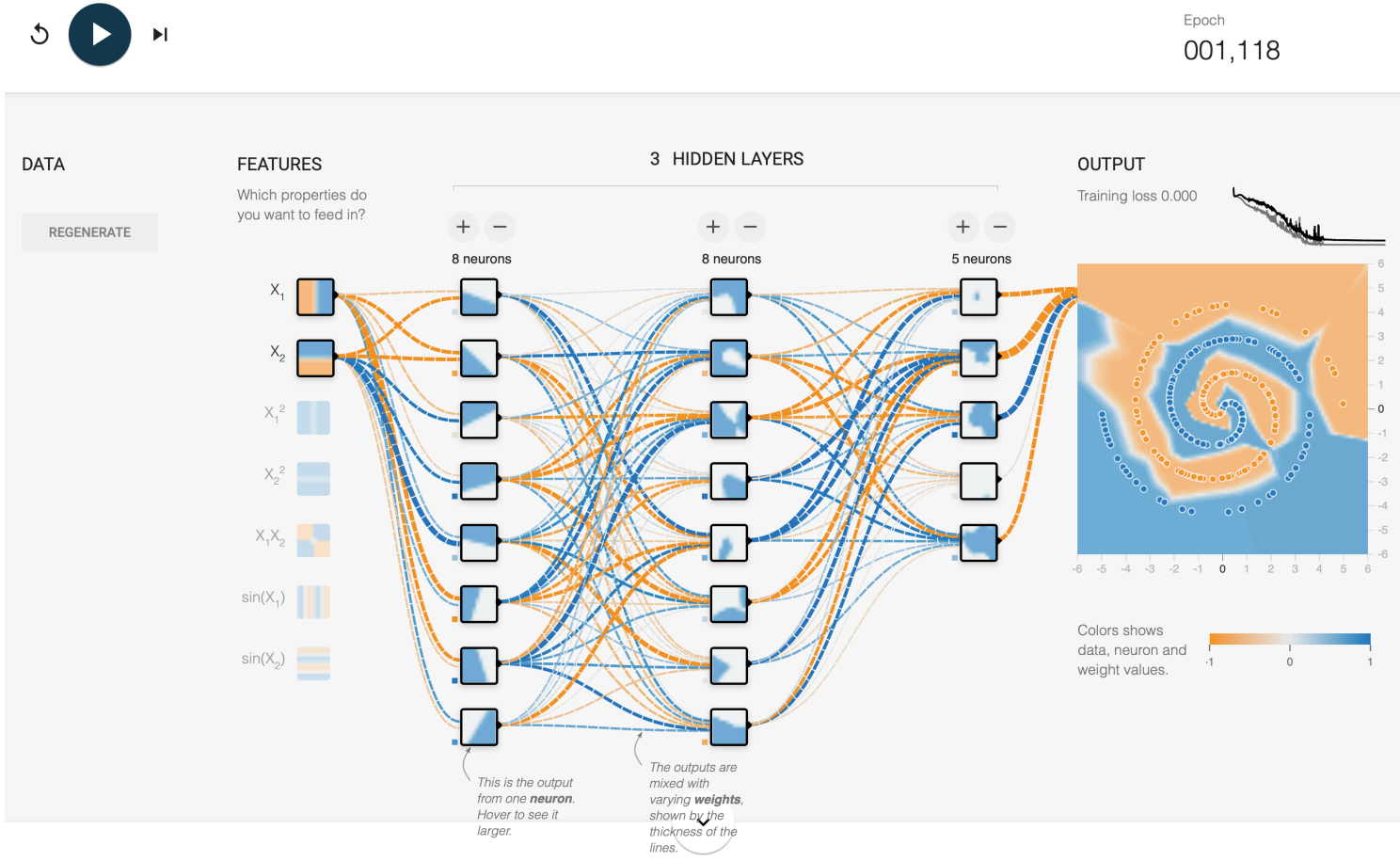
With more neurons in a single hidden layer, you can capture more features. And having more hidden layers means more complex constructs that you can extract from the dataset.



https://cloud.google.com/blog/products/ai-machine-learning/understanding-neural-networks-with-tensorflow-playground

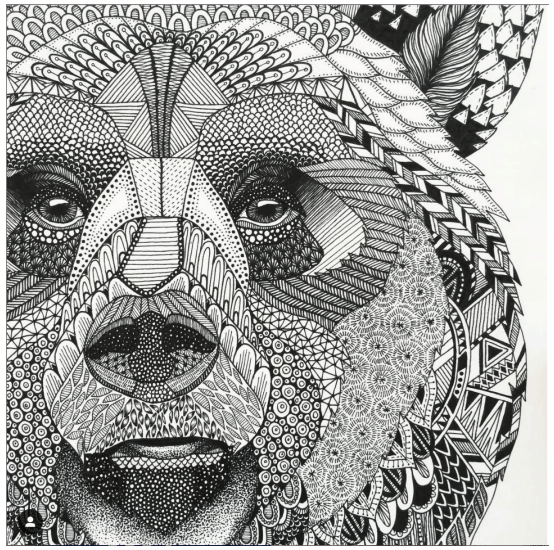
Neural Network
Experiment 04: solve the SPIRAL problem

[Solution Click here] to see it in action (it will take a couple of minutes to train).



https://cloud.google.com/blog/products/ai-machine-learning/understanding-neural-networks-with-tensorflow-playground

{04}
Do it Yourself



Do some online research.
Find examples of interactive NN + LM demonstration
Make a video wherein you demonstrate
(1) What learning problem (task) is solved (classification / regression)
(2) Explain the NN architecture
(3) Describe he hyper-parameterisation you used to solve the problem

+ Describe in your own words what kind of data-product you to build
See:
https://github.com/robvdw/RCA_AIG_042Q6_ARTIFICIAL_INTELLIGENCE

Direct-Manipulation Visualization of Deep Networks

Daniel Smilkov
Google, Inc.
5 Cambridge Center,
Cambridge MA, 02142
smilkov@google.com

Shan Carter
Google, Inc.
1600 Amphitheater Parkway,
Mountain View CA, 94043
shancarter@google.com

D. Sculley
Google, Inc.
5 Cambridge Center,
Cambridge MA, 02142
dsculley@google.com

Fernanda B. Viégas
Google, Inc.
5 Cambridge Center,
Cambridge MA, 02142
viegas@google.com

Martin Wattenberg
Google, Inc.
5 Cambridge Center,
Cambridge MA, 02142
wattenberg@google.com

ABSTRACT

The recent successes of deep learning have led to a wave of interest from non-experts. Gaining an understanding of this technology, however, is difficult. While the theory is important, it is also helpful for novices to develop an intuitive feel for the effect of different hyperparameters and structural variations. We describe TensorFlow Playground[1], an interactive, open sourced[2] visualization that allows users to experiment via direct manipulation rather than coding, enabling them to quickly build an intuition about neural nets.

1. INTRODUCTION

Deep learning systems are currently attracting a huge amount of interest, as they see continued success in practical applications. Students who want to understand this new technology encounter two primary challenges.

First, the theoretical foundations of the field are not always easy for a typical software engineer or computer science student, since they require a solid mathematical intuition. It's not trivial to translate the equations defining a deep network into a mental model of the underlying geometric transformations.

Even more challenging are aspects of deep learning where theory does not provide crisp, clean explanations. Critical choices experts make in building a real-world system–the number of units and layers, the activation function, regularization techniques, etc.–are currently guided by intuition and experience as much as theory. Acquiring this intuition is a lengthy process, since it typically requires coding and training many different working systems.

One possible shortcut is to use interactive visualization to help novices with mathematical and practical intuition. Recently, several impressive systems have appeared that do exactly this. Olah's elegant interactive online essays [5] let a viewer watch the training of a simple classifier, providing a multiple perspectives on how a network learns a transformation of space. Karpathy created a Javascript library [4] and provided a series of dynamic views of networks training, again in a browser. Others have found beautiful ways to visualize the features learned by image classification nets [10], [9].

Taking inspiration from the success of these examples, we created the TensorFlow Playground. As with the work of Olah and Karpathy, the Playground is an in-browser visualization of a running neural network. However, it is specifically designed for experimentation by direct manipulation, and also visualizes the derived "features" found by every unit in the network simultaneously. The system provides a variety of affordances for rapidly and incrementally changing hyperparameters and immediately seeing the effects of those changes, as well as for sharing experiments with others.

2. TENSORFLOW PLAYGROUND: VISUALIZATION

The structure of the Playground visualization is a standard network diagram. The visualization shows a network that is designed to solve either classification or regression problems based on two abstract real-valued features, x_1 and x_2, which vary between -1 and 1. Input units, representing these features and various mathematical combinations, are at the left. Units in hidden layers are shown as small boxes, with connections between units drawn as curves whose color and width indicate weight values. Finally, on the right, a visualization of the output of the network is shown: a square with a heatmap showing the output value of the single unit that makes up the final layer of the network. When the user presses the "play" button, the network begins to train.

There is a new twist in this visualization, however. Inside the box that represents each unit is a heatmap that maps the unit's response to all values of (x_1, x_2) in a square centered at the origin. As seen in Figure 1, this provides a quick geometric view of how the network builds complex features from simpler ones. For example, in the figure the input features are simply x_1 and x_2, which themselves are represented by the same type of heatmap. In the next layer, we see units that correspond to various linear combinations, leading to a final layer with more complicated non-linear classifiers. Moving the mouse over any of these units projects a larger version of the heatmap, on the final unit, where it can be overlaid with input and test data.

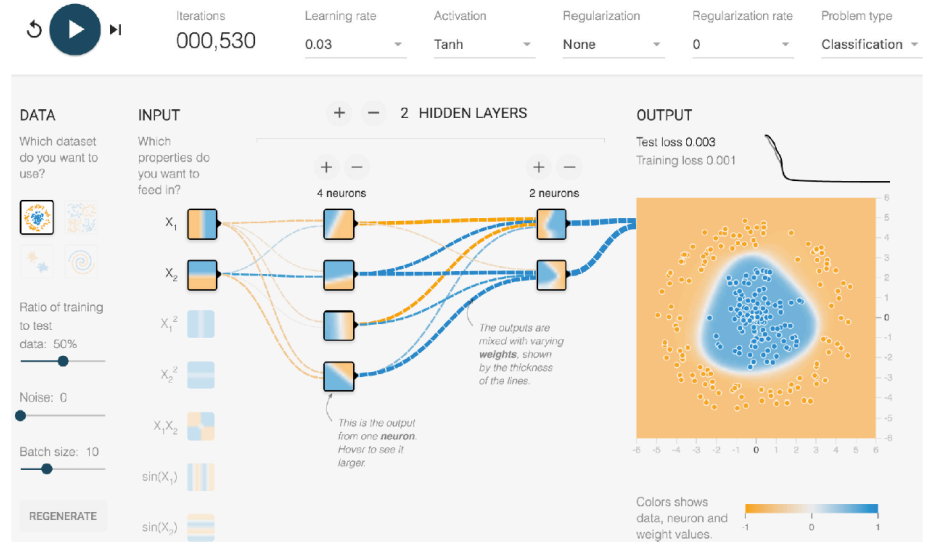


Figure 1: TensorFlow Playground. This network is, roughly speaking, classifying data based on distance to the origin. Curves show weight parameters, with thickness denoting absolute magnitude and color indicating sign. The feature heatmaps for each unit show how the classification function (large heatmap at right) is built from input features, then near-linear combinations of these features, and finally more complex features. At upper right is a graph showing loss over time. At left are possible features; x_1 and x_2 are highlighted, while other mathematical combinations are faded to indicate they should not be used by the network.

The activation heatmaps help users build a mental model of the mathematics underlying deep networks. For many configurations of the network, after training there is an obvious visual progression in complexity across the network. In these configurations, viewers can see how the first layer of units (modulo activation function, acting as linear classifiers) combine to recognize clearly nonlinear regions. The heatmaps also help viewers understand the different effects of various activation functions. For example, there is a clear visual difference in the effect of ReLU and tanh functions. Just as instructive, however, are suboptimal combinations of architecture and hyperparameters. Often when there are redundant units (Figure 3), it is easy to see that units in intermediate layers have actually learned the classifier perfectly well and that many other units have little effect on the final outcome. In cases where learning is simply unsuccessful, the viewer will often see weights going to zero, and that there is no natural progression of complexity in the activation heatmaps (Figure 4).

The visualization is implemented in JavaScript using d3.js[3]. It is worth noting that for the neural network computation, we are not using the TensorFlow library [1] since we needed the whole visualization to run in the browser. Instead, we wrote a small library[3] that meets the demands of this educational visualization.

3. AFFORDANCES FOR EDUCATION AND EXPERIMENTATION

The real strength of this visualization is its interactivity, which is especially helpful for gaining an intuition for the practical aspects of training a deep network. The Playground lets users make the following choices of network structure and hyperparameters:

- Problem type: regression or classification
- Training data: a choice of four synthetic data sets, from well-separated clusters to interleaved "swiss roll" spirals.
- Number of layers
- Number of units in each layer
- Activation function
- Learning rate

[1] http://playground.tensorflow.org
[2] https://github.com/tensorflow/playground
[3] https://github.com/tensorflow/playground/blob/master/nn.ts

CNN 101: Interactive Visual Learning for Convolutional Neural Networks

Zijie J. Wang
Georgia Institute of Technology
jayw@gatech.edu

Nilaksh Das
Georgia Institute of Technology
nilakshdas@gatech.edu

Robert Turko
Georgia Institute of Technology
rturko3@gatech.edu

Fred Hohman
Georgia Institute of Technology
fredhohman@gatech.edu

Omar Shaikh
Georgia Institute of Technology
oshaikh@gatech.edu

Minsuk Kahng
Oregon State University
minsuk.kahng@oregonstate.edu

Haekyu Park
Georgia Institute of Technology
haekyu@gatech.edu

Duen Horng (Polo) Chau
Georgia Institute of Technology
polo@gatech.edu

CHI '20 Extended Abstracts, April 25–30, 2020, Honolulu, HI, USA.

ACM ISBN 978-1-4503-6819-3/20/04.
http://dx.doi.org/10.1145/3334480.3382899

Abstract
The success of deep learning solving previously-thought hard problems has inspired many non-experts to learn and understand this exciting technology. However, it is often challenging for learners to take the first steps due to the complexity of deep learning models. We present our ongoing work, CNN 101, an interactive visualization system for explaining and teaching convolutional neural networks. Through tightly integrated interactive views, CNN 101 offers both overview and detailed descriptions of how a model works. Built using modern web technologies, CNN 101 runs locally in users' web browsers without requiring specialized hardware, broadening the public's education access to modern deep learning techniques.

Author Keywords
Interactive visualization; deep learning education; machine learning education.

CCS Concepts
•Human-centered computing → Visual analytics; •Computing methodologies → Machine learning;

Introduction
Deep learning has become a driving-force in our daily technologies. Its continued success and potential in various hard problems have attracted immense interest from non-

CNN EXPLAINER: Learning Convolutional Neural Networks with Interactive Visualization

Zijie J. Wang, Robert Turko, Omar Shaikh, Haekyu Park, Nilaksh Das,
Fred Hohman, Minsuk Kahng, and Duen Horng (Polo) Chau

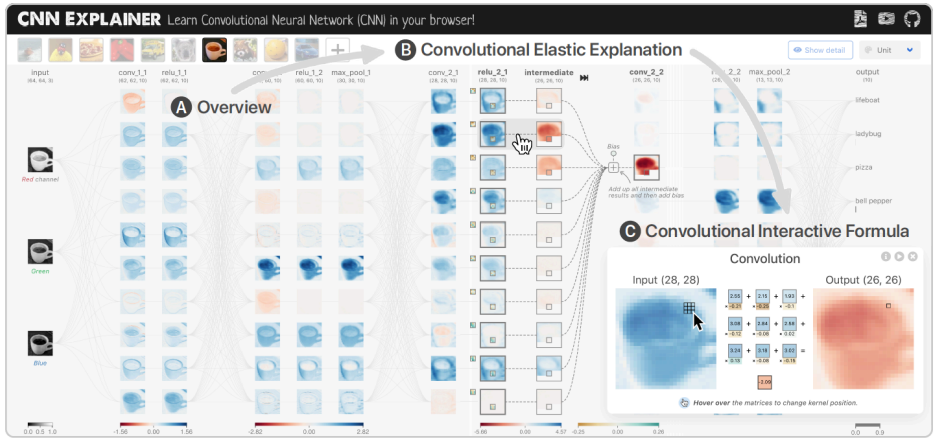


Fig. 1. With CNN EXPLAINER, learners can visually examine how *Convolutional Neural Networks* (CNNs) transform input images into classification predictions (e.g., predicting *espresso* for an image of a coffee cup), and interactively learn about their underlying mathematical operations. In this example, a learner uses CNN EXPLAINER to understand how convolutional layers work through three tightly integrated views, each explaining the convolutional process in increasing levels of detail. **(A)** The *Overview* visualizes a CNN architecture where each neuron is encoded as a square with a heatmap representing the neuron's output. **(B)** Clicking a neuron reveals how its activations are computed by the previous layer's neurons, displaying the often-overlooked intermediate computation through animations of sliding kernels. **(C)** *Convolutional Interactive Formula View* for inspecting underlying mathematics of the dot-product operation core to convolution. For clarity, some annotations are removed and views are re-positioned.

Abstract— Deep learning's great success motivates many practitioners and students to learn about this exciting technology. However, it is often challenging for beginners to take their first step due to the complexity of understanding and applying deep learning. We present CNN EXPLAINER, an interactive visualization tool designed for non-experts to learn and examine convolutional neural networks (CNNs), a foundational deep learning model architecture. Our tool addresses key challenges that novices face while learning about CNNs, which we identify from interviews with instructors and a survey with past students. CNN EXPLAINER tightly integrates a model overview that summarizes a CNN's structure, and on-demand, dynamic visual explanation views that help users understand the underlying components of CNNs. Through smooth transitions across levels of abstraction, our tool enables users to inspect the interplay between low-level mathematical operations and high-level model structures. A qualitative user study shows that CNN EXPLAINER helps users more easily understand the inner workings of CNNs, and is engaging and enjoyable to use. We also derive design lessons from our study. Developed using modern web technologies, CNN EXPLAINER runs locally in users' web browsers without the need for installation or specialized hardware, broadening the public's education access to modern deep learning techniques.

Index Terms—Deep learning, machine learning, convolutional neural networks, visual analytics

1 INTRODUCTION

Deep learning now enables many of our everyday technologies. Its continued success and potential application in diverse domains has attracted immense interest from students and practitioners who wish to learn and apply this technology. However, many beginners find it challenging to take the first step in studying and understanding deep learning concepts. For example, convolutional neural networks (CNNs), a foundational deep learning model architecture, is often one of the first and most widely used models that students learn. CNNs are often used in image classification, achieving state-of-the-art performance [33]. However, through interviews with deep learning instructors and a survey of past students, we found that even for this "introductory" model, it can be challenging for beginners to understand how inputs (e.g., image data) are transformed into class predictions. This steep learning curve stems from CNN's complexity, which typically leverages many computational layers to reach a final decision. Within a CNN, there are many types of network layers (e.g., fully-connected, convolutional, activation), each with a different structure and underlying mathematical operations. Thus, a student needs to develop a mental model of not only how each layer operates, but also how to choose different layers that work together to transform data. Therefore, a key challenge in learning about CNNs is the intricate interplay between *low-level mathematical operations* and *high-level integration* of such operations within the network.

Key challenges in designing learning tools for CNNs. There is a growing body of research that uses interactive visualization to explain the complex mechanisms of modern machine learning algorithms, such as TensorFlow Playground [50] and GAN Lab [29], which help students learn about dense neural networks and generative adversarial networks (GANs) respectively. Regarding CNNs, some existing visualization tools focus on demonstrating the high-level model structure and connections between layers (e.g., Harley's Node-Link Visualization [20]), while others focus on explaining the low-level mathematical operations (e.g., Karpathy's interactive CNN demo [30]). There is no visual learning tool that explains and connects CNN concepts from both levels of abstraction. This interplay between global model structure and local layer operations has been identified as one of the main obstacles to learning deep learning models, as discussed in [50] and corroborated from our interviews with instructors and student survey. CNN EXPLAINER aims to bridge this critical gap.

Contributions. In this work, we contribute:

- **CNN EXPLAINER, an interactive visualization tool designed for non-experts** to learn about both CNN's high-level model structure and low-level mathematical operations, addressing learners' key challenge in connecting unfamiliar layer mechanisms with complex model structures. Our tool advances over prior work [20, 30], overcoming unique design challenges identified from a literature review, instructor interviews and a survey with past students (Sect. 4).
- **Novel interactive system design** of CNN EXPLAINER (Fig. 1), which adapts familiar techniques such as *overview + detail* and *animation* to simultaneously summarize intricate model structure, while providing context for users to inspect detailed mathematical operations. CNN EXPLAINER's visualization techniques work together through fluid transitions between different abstraction levels (Fig. 2), helping users gain a more comprehensive understanding of complex concepts within CNNs (Sect. 6).
- **Design lessons distilled from user studies** on an interactive visualization tool for machine learning education. While visual and interactive approaches have been gaining popularity in explaining machine learning concepts to non-experts, little work has been done to evaluate such tools [28, 43]. We interviewed four instructors who have taught CNNs and conducted a survey with 19 students who have previously learned about CNNs to identify the needs and challenges for a deep learning educational tool (Sect. 4). In addition, we conducted an observational study with 16 students to evaluate the usability of CNN EXPLAINER, and investigated how our tool could help students better understand CNN concepts (Sect. 8). Based on these studies, we discuss the advantages and limitations of interactive visual educational tools for machine learning.
- **An open-source, web-based implementation** that broadens the public's education access to modern deep learning techniques without the need for advanced computational resources. Deploying deep learning models conventionally requires significant computing resources, e.g., servers with powerful hardware. In addition, even with a dedicated backend server, it is challenging to support a large number of concurrent users. Instead, CNN EXPLAINER is developed using modern web technologies, where all results are directly and efficiently computed in users' web browsers (Sect. 6.7). Therefore, anyone can access CNN EXPLAINER using their web browser without the need for installation or a specialized backend. Our code is open-sourced[1] and

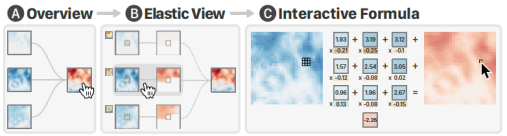


Fig. 2. In CNN EXPLAINER, tightly integrated views with different levels of abstractions work together to help users more easily learn about the intricate interplay between a CNN's high-level structure and low-level mathematical operations. **(A)** the *Overview* summarizes connections of all neurons; **(B)** the *Elastic View* animates the intermediate convolutional computation of the user-selected neuron in the *Overview*; and **(C)** *Interactive Formula* interactively demonstrates the detailed calculation on the selected input in the *Elastic View*.

CNN EXPLAINER is available at the following public demo link: https://poloclub.github.io/cnn-explainer.

Broadening impact of visualization for AI. In recent years, many visualization systems have been developed for deep learning, but very few are designed for non-experts [20, 29, 44, 50], as surveyed in [23]. CNN EXPLAINER joins visualization research that introduces beginners to modern machine learning concepts. Applying visualization techniques to explain the inner workings of complex models has great potential. We hope our work will inspire further research and development of visual learning tools that help democratize and lower the barrier to understanding and applying artificial intelligent technologies.

2 BACKGROUND FOR CONVOLUTIONAL NEURAL NETWORKS

This section provides a high-level overview of convolutional neural networks (CNNs) in the context of image classification, which will help ground our work throughout this paper.

Image classification has a long history in the machine learning research community. The objective of supervised image classification is to map an input image, X, to an output class, Y. For example, given a cat image, a sophisticated image classifier would output a class label of "cat". CNNs have demonstrated state-of-the-art performance on this task, in part because of their multiple layers of computation that aim to learn a better representation of image data.

CNNs are composed of several different layers (e.g., convolutional layers, downsampling layers, and activation layers)—each layer performs some predetermined function on its input data. Convolutional layers "extract features" to be used for image classification, with early convolutional layers in the network extracting low-level features (e.g., edges) and later layers extracting more-complex semantic features (e.g., car headlights). Through a process called backpropagation, a CNN learns kernel weights and biases from a collection of input images. These values also known as parameters, which summarize important features within the images, regardless of their location. These kernel weights slide across an input image performing an element-wise dot-product, yielding intermediate results that are later summed together with the learned bias value. Then, each neuron gets an output based on the input image. These outputs are also called activation maps. To decrease the number of parameters and help avoid overfitting, CNNs downsample inputs using another type of layer called pooling. Activation functions are used in a CNN to introduce non-linearity, which allows the model to learn more complex patterns in data. For example, a Rectified Linear Unit (ReLU) is defined as $\max(0, x)$, which outputs the positive part of its argument. These functions are also often used prior to the output layer to normalize classification scores, for example, the activation function called Softmax performs a normalization on unscaled scalar values, known as logits, to yield output class scores that sum to one. To summarize, compared to classic image classification models that can be over-parameterized and fail to take advantage of inherent properties in image data, CNNs create spatially-aware representations through multiple stacked layers of computation.

- *Zijie J. Wang, Robert Turko, Omar Shaikh, Haekyu Park, Nilaksh Das, Fred Hohman, and Duen Horng Chau are with Georgia Tech. E-mail: {jayw|rturko3|oshaikh|haekyu|nilakshdas|fredhohman|polo}@gatech.edu.*
- *Minsuk Kahng is with Oregon State University. E-mail: minsuk.kahng@oregonstate.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

[1] Code: https://github.com/poloclub/cnn-explainer

GAN Lab: Understanding Complex Deep Generative Models using Interactive Visual Experimentation

Minsuk Kahng, Nikhil Thorat, Duen Horng (Polo) Chau, Fernanda B. Viégas, and Martin Wattenberg

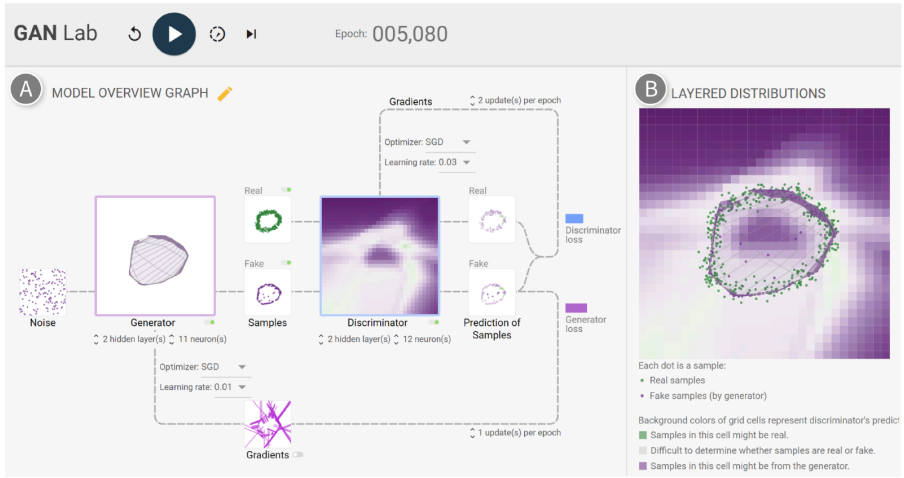


Fig. 1. With GAN Lab, users can interactively train Generative Adversarial Networks (GANs), and visually examine the model training process. In this example, a user has successfully used GAN Lab to train a GAN that generates 2D data points whose challenging distribution resembles a ring. **A.** The *model overview graph* summarizes a GAN model's structure as a graph, with nodes representing the generator and discriminator submodels, and the data that flow through the graph (e.g., fake samples produced by the generator). **B.** The *layered distributions* view helps users interpret the interplay between submodels through user-selected layers, such as the discriminator's classification heatmap, real samples, and fake samples produced by the generator.

Abstract—Recent success in deep learning has generated immense interest among practitioners and students, inspiring many to learn about this new technology. While visual and interactive approaches have been successfully developed to help people more easily learn deep learning, most existing tools focus on simpler models. In this work, we present GAN Lab, the first interactive visualization tool designed for non-experts to learn and experiment with *Generative Adversarial Networks (GANs)*, a popular class of complex deep learning models. With GAN Lab, users can interactively train generative models and visualize the dynamic training process's intermediate results. GAN Lab tightly integrates an *model overview graph* that summarizes GAN's structure, and a *layered distributions* view that helps users interpret the interplay between submodels. GAN Lab introduces new interactive experimentation features for learning complex deep learning models, such as *step-by-step* training at multiple levels of abstraction for understanding intricate training dynamics. Implemented using *TensorFlow.js*, GAN Lab is accessible to anyone via modern web browsers, without the need for installation or specialized hardware, overcoming a major practical challenge in deploying interactive tools for deep learning.

Index Terms—Deep learning, information visualization, visual analytics, generative adversarial networks, machine learning, interactive experimentation, explorable explanations

1 Introduction

Recent success in deep learning has generated a huge amount of interest from practitioners and students, inspiring many to learn about this technology. Visual and interactive approaches have successfully been used to describe concepts and underlying mechanisms in deep learning [17, 28, 36, 44]. For example, Karpathy's popular interactive demo [17] enables users to run convolutional neural nets and visualize neuron activations, inspiring researchers to develop more interactive tools for deep learning. Another notable example is Google's *TensorFlow Playground* [36], an interactive tool that visually represents a neural network model and allows users to interactively experiment with the model through direct manipulation; Google now uses it to educate their employees about deep learning [31].

The rise of GANs and their compelling uses. Most existing interactive tools, however, have been designed for simpler models. Meanwhile, modern deep learning models are becoming more complex. For example, *Generated Adversarial Networks (GANs)* [9], a class of deep learning models known for their remarkable ability to generate synthetic images that look like natural images, are difficult to train and for people to understand, even for experts. Since the first GAN publication by Goodfellow et al. [9] in 2014, GANs have become one of the most popular machine learning research topics [12, 18]. GANs have achieved state-of-the-art performance in a variety of previously difficult tasks, such as synthesizing super-resolution images based on low-resolution copies, and performing image-to-image translation (e.g., converting sketches to realistic images) [8].

Key challenges in designing learning tools for GANs. At the high level, a GAN internally combines two neural networks, called *generator* and *discriminator*, to play a game where the generator creates "fake" data and the discriminator guesses whether that data is real or fake (both types of data are mixed together). A perfect GAN is one that generates fake data that is virtually indistinguishable from real data. A user who wishes to learn about GANs needs to develop a mental model of not only what the two submodels do, but also how they affect each other in its training process. The crux in learning about GANs, therefore, originates from the iterative, dynamic, intricate interplay between these two submodels. Such complex interaction is challenging for novices to recognize, and sometimes even for experts to fully understand [32]. Typical architecture diagrams for GANs (e.g., Fig. 2, commonly shown in learning materials) do not effectively help people develop the crucial mental models needed for understanding GANs.

Contributions. In this work, we contribute:

- **GAN Lab, the first interactive tool designed for non-experts** to learn and experiment with GAN models, a popular class of complex deep learning models, that overcomes multiple unique challenges for developing interactive tools for GANs (Sect. 4).
- **Novel interactive visualization design** of GAN Lab (Fig. 1), which tightly integrates a *model overview graph* that summarizes GAN's structure (Fig. 1A) as a graph, selectively visualizing components that are crucial to the training process; and a *layered distributions* view (Fig. 1B) that helps users interpret the interplay between submodels through user-selected layers (Sect. 6). GAN Lab's visualization techniques work in tandem to help crystalize complex concepts in GANs. For example, GAN Lab visualizes the generator's data transformation, which turns input noise into fake samples, as a manifold (Fig. 1, big box with purple border). When the user hovers over it, GAN Lab animates the input-to-output transformation (Fig. 3) to visualize how the input 2D space is folded and twisted by the generator to create the desired ring-like data distribution, helping users more easily understand the complex behavior of the generator.
- **New interactive experimentation features** for learning complex deep learning models, such as *step-by-step* training at multiple levels of abstraction for understanding intricate training dynamics (Sect. 7). The user can also interact with the training process by directly manipulating GAN's hyperparameters.
- **A browser-based, open-sourced implementation** that helps *broaden public's education access to modern deep learning technologies* (Sect. 7.3). Training deep learning models conventionally requires significant computing resources. For example, deep learning frameworks, like TensorFlow [1], typically run on dedicated servers. They are not designed to support low-latency computation needed for real-time interactive tools, or large number of concurrent user sessions through the web. We overcome such practical challenges in deploying interactive visualization for deep learning by using *TensorFlow.js Core*,[1] an in-browser GPU-accelerated deep learning library recently developed by Google; the second author is a lead developer of TensorFlow.js Core. Anyone can access GAN Lab using their web browsers without the need for installation or specialized backend. GAN Lab runs locally on the user's web browser, allowing us to easily scale up deployment for our tool to the public, significantly broadening people's access to tools for learning about GANs. The source code is available in `https://github.com/poloclub/ganlab/`.
- Usage scenarios that demonstrate how GAN Lab can help beginners learn key concepts and training workflow in GANs, and assist practitioners to interactively attain optimal hyperparameters for reaching challenging equilibrium between submodels (Sect. 8).

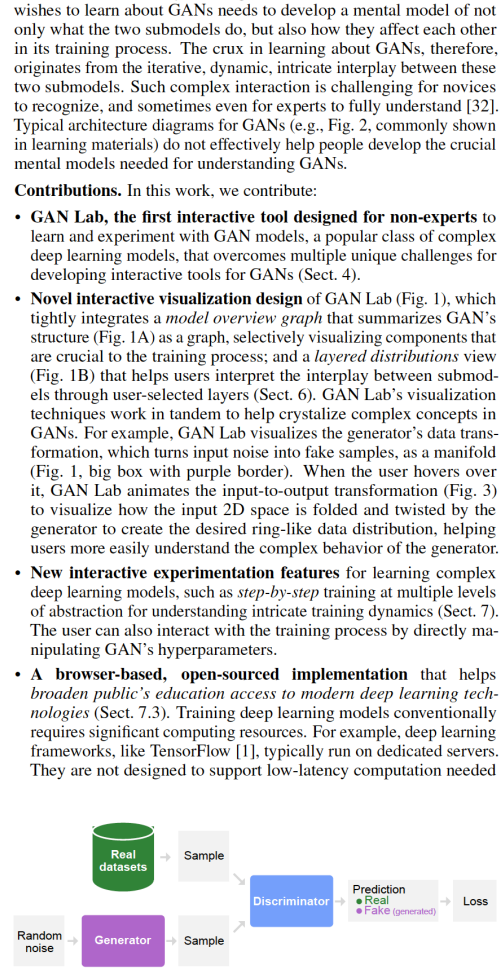


Fig. 2. A graphical schematic representation of a GAN's architecture commonly used.

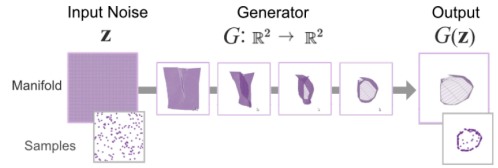


Fig. 3. In GAN Lab, the *generator*'s non-trivial data transformation is visualized as a manifold, which turns *input noise* (leftmost) into fake samples (rightmost). GAN Lab animates the input-to-output transformation to help users more easily understand this complex behavior.

VIS's central role in AI. We believe in-browser interactive tools developed by our VIS community, like GAN Lab, will play critical roles in promoting people's understanding of deep learning, and raising their awareness of this exciting new technology. To the best of our knowledge, our work is the first tool designed for non-experts to learn and experiment with complex GAN models, different from recent work in visualization for deep learning [16, 20, 21, 30, 37, 43] which primarily targets machine learning experts. Our work joins a growing body of research that aims to use interactive visualization to explain complex inner workings of modern machine learning techniques. Distill, a new interactive form of journal, is dedicated to achieving this exact goal [29]. We hope our work will help inspire even more research and development of visualization tools that help people better understanding artificial intelligence technologies.

2 Background: Generative Adversarial Networks

This section presents a brief introduction of Generated Adversarial Networks, which will help ground our discussion in this paper.

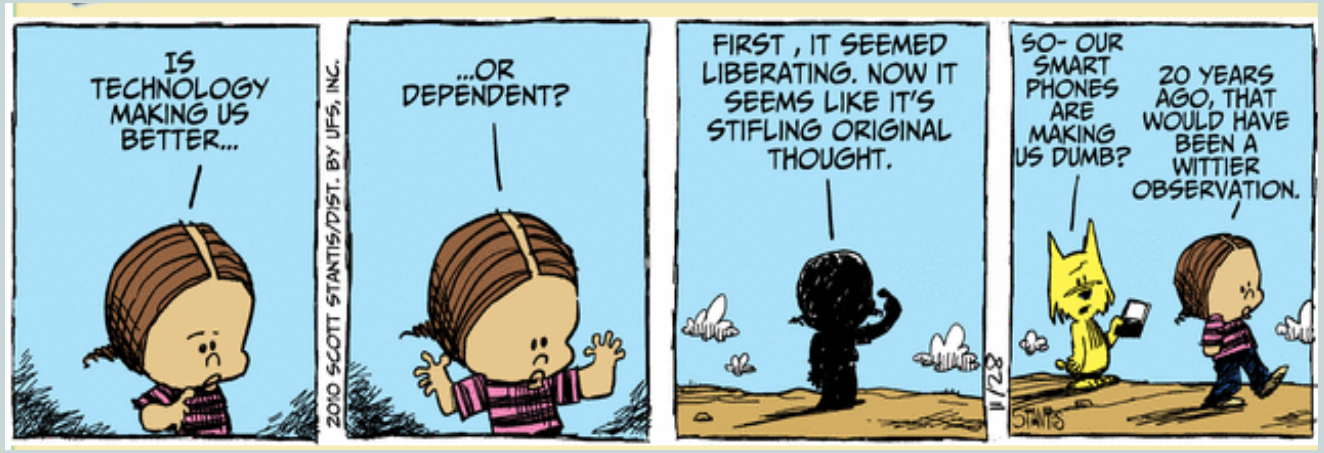
Generative Adversarial Networks (GANs) [9] are a new class of unsupervised generative deep learning models that model data distributions. It can be used for generating multi-dimensional data distributions (e.g., an image is a multi-dimensional data point, where each pixel is a dimension). The model takes *real samples* and random vectors (i.e., *random noise*) as inputs and transforms the random vectors into *fake samples* that mimic the real samples. Ideally, the distribution of the fake samples will be indistinguishable from the real samples. The architecture of GANs is composed of two neural networks, called *generator* and *discriminator*, and is often represented as an abstracted data-flow graph as in Fig. 2. The generator, G, takes a random noise vector, $\mathbf{z}$, as input and transforms it into a fake sample, $G(\mathbf{z})$ (i.e., a multi-dimensional vector); the discriminator, D, which is a binary classifier, takes either a real or fake sample, and determines whether it is real or fake ($D(\mathbf{x})$ represents the probability that $\mathbf{x}$ is real rather than fake).

A GAN model is iteratively trained through a game between the discriminator and generator. In GAN, two cost functions exist: the one for the discriminator measures the probability of assigning the

- *Minsuk Kahng and Duen Horng (Polo) Chau are with Georgia Institute of Technology. E-mail: {kahng | polo}@gatech.edu.*
- *Nikhil Thorat, Fernanda B. Viégas, and Martin Wattenberg are with Google Brain. E-mail: {nsthorat | viegas | wattenberg}@google.com.*

Manuscript received 31 Mar. 2018; accepted 1 Aug. 2018. Date of publication 16 Aug. 2018; date of current version 21 Oct. 2018. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TVCG.2018.2864500

[1] TensorFlow.js (`https://js.tensorflow.org`) was formerly deeplearn.js.



This lesson was developed by:

Robert Frans van der Willigen
CMD, Hogeschool Rotterdam
OKT 2020

Creative Commons License Types

	Can someone use it commercially?	Can someone create new versions of it?
Attribution	👍	👍
Share Alike	👍	Yup, AND they must license the new work under a Share Alike license.
No Derivatives	👍	👎
Non-Commercial	👎	Yup, AND the new work must be non-commercial, but it can be under any non-commercial license.
Non-Commercial Share Alike	👎	Yup, AND they must license the new work under a Non-Commercial Share Alike license.
Non-Commercial No Derivatives	👎	👎

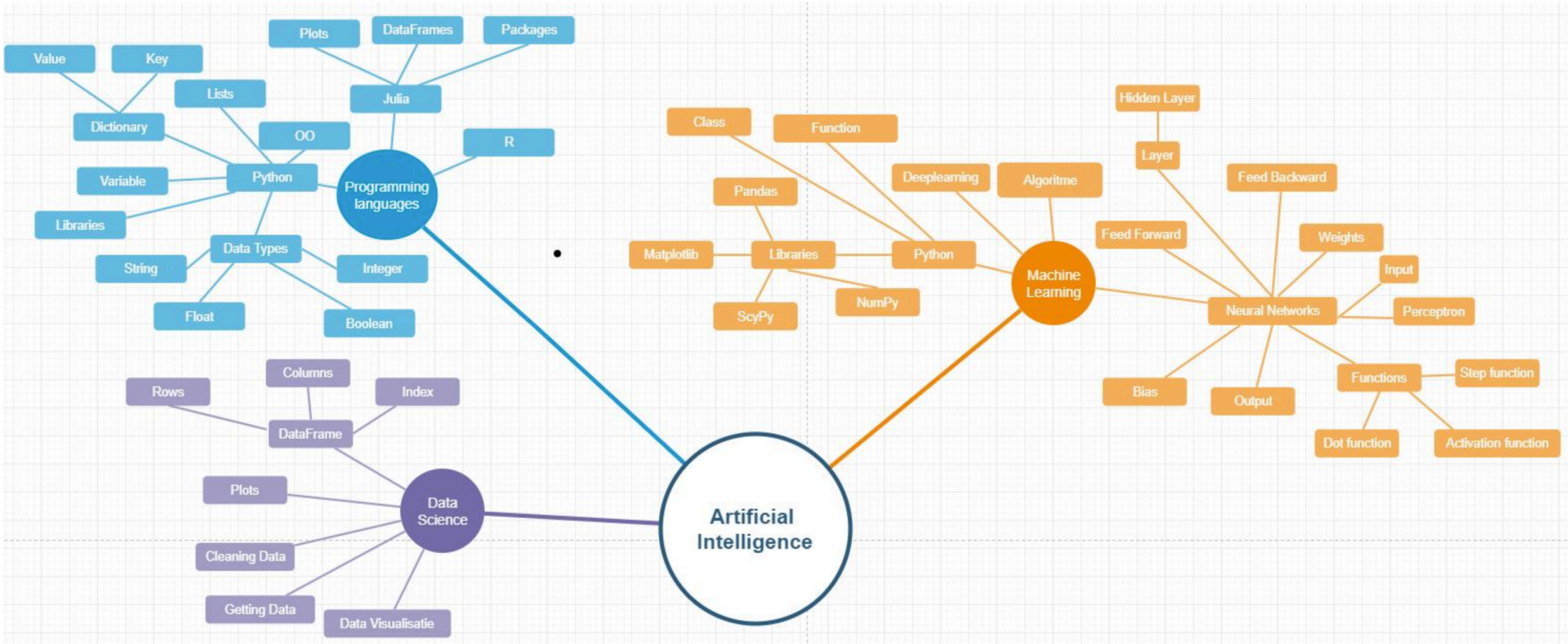
SOURCE
http://www.masternewmedia.org/how-to-publish-a-book-under-a-creative-commons-license/

http://empoweringthenatives.edublogs.org/2012/03/15/creative-commons-licenses/

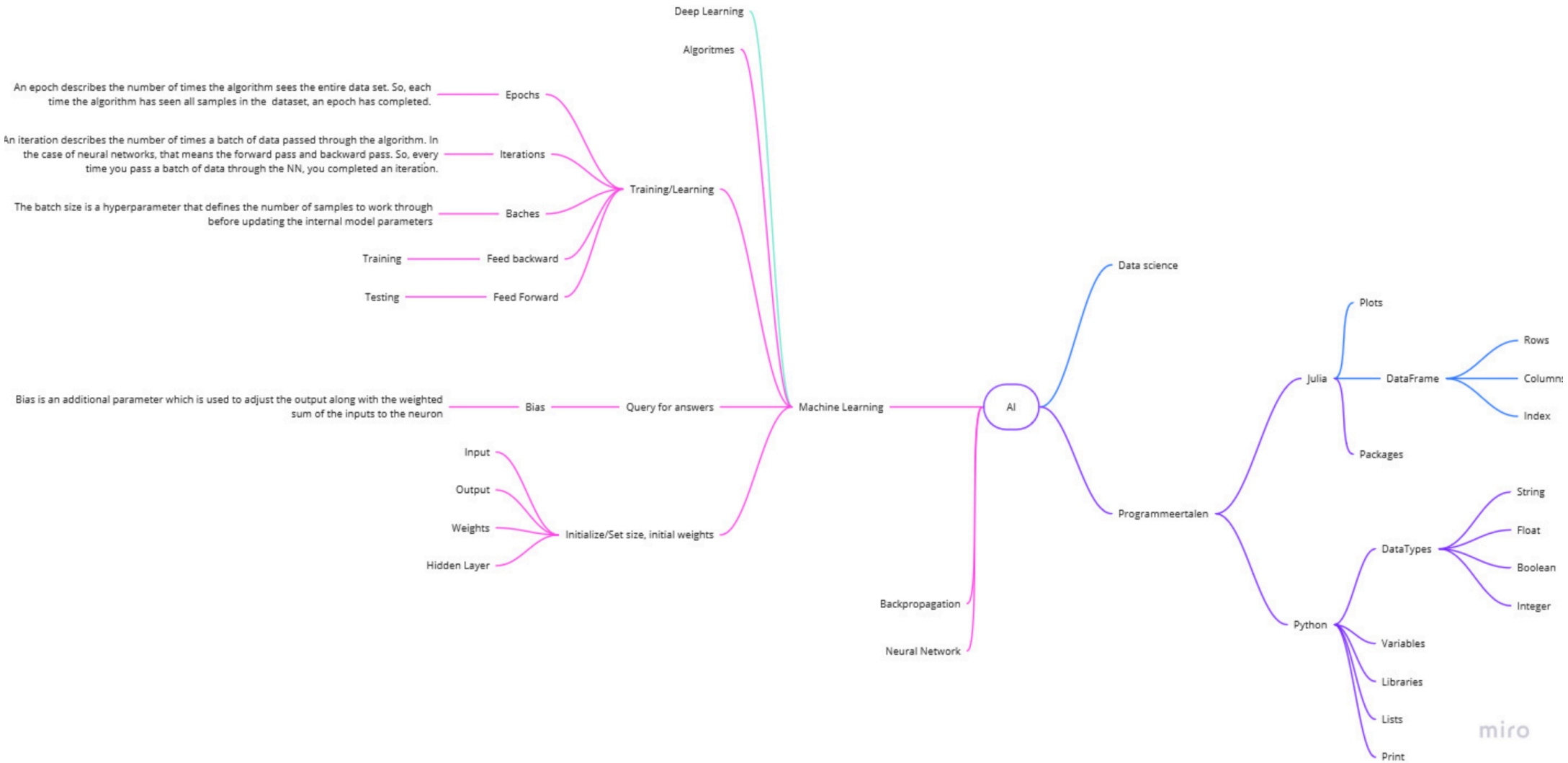
http://creativecommons.org/licenses/


the functional art
alberto cairo
TECHNOLOGY TRENDS TO WATCH
Christian Leborg
Visual Grammar
the analytics revolution
Bill Franks
MADE to STICK
generative art
matt pearson
EMPATHIC THINGS
Intieme technologie van wearables tot biohacking
INTERNET OF THINGS!
MAKERS
THE NEW INDUSTRIAL REVOLUTION
CHRIS ANDERSON
MARITIEME MAAKINDUSTRIE TERUG NAAR RDM
WebGL
TV AND VIDEO
An analysis of evolving consumer habits
THE DISRUPTION MACHINE
Learning to Program with MATLAB
BUILDING GUI TOOLS
Port Management Program
resonate
10 TRENDS FOR 2014
A giant cage
R Data Analysis
Design for Information
HACKATHO
OPEN HAVE
1 VOLLE DAG
7 SEPTEMBER
WERELDHAVENDAGEN 2013
the myths of innovation
scott berkun
O'REILLY
Cool Infograph
Effective Communication with Data Visualization and Design
The VISU
DATA VISUALIZATION, BIG DATA, AND THE QUEST FOR BETTER DECISIONS
PHIL SIMON
OUR WORK HERE IS DONE
VISIONS OF A ROBOT ECONOMY
INVENT TO LEARN
Making, Tinkering, and Engineering in the Classroom
MORE SECRETS with Big Data Analytics
Processing
A Programming Handbook for Visual Designers and Artists
THE WALL STREET JOURNAL.
Guide to Information Graphics
Dona M. Wong
Customer Journey Mapping
Wearable Technology Design
The Hidden Pitfall in your Design
Making Things See
Learn by Discovery
TEACHING that STICKS
Chip Heath & Dan Heath
Protecting Data and Privacy in the Cloud
SOGETI
DISCOVER METEOR
Making Things Talk
The Internet of Things
Business Adapts to a New Style of Computer
EVOLUTION OF AGILE

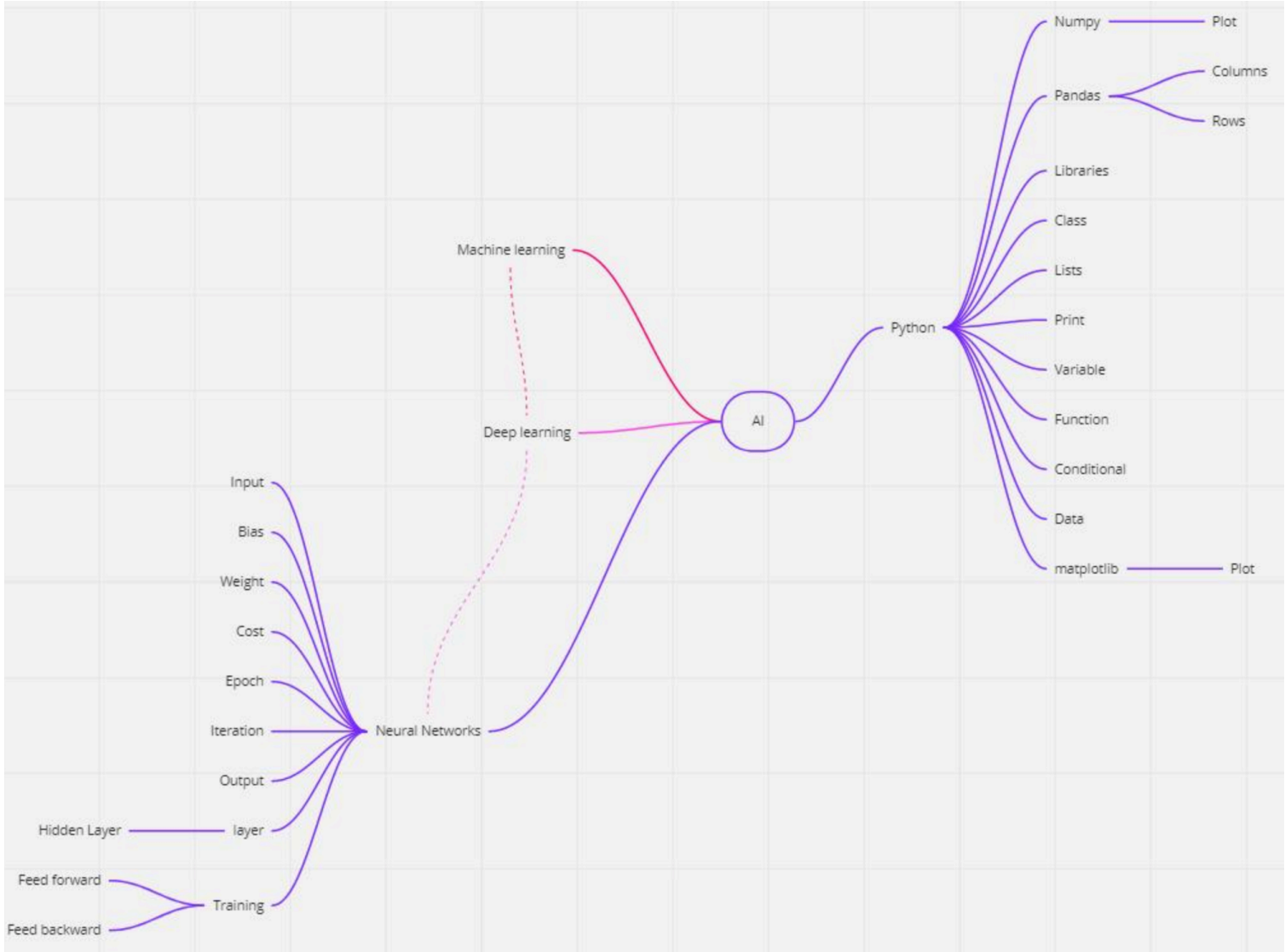
Raiklin Felicia (0786789)



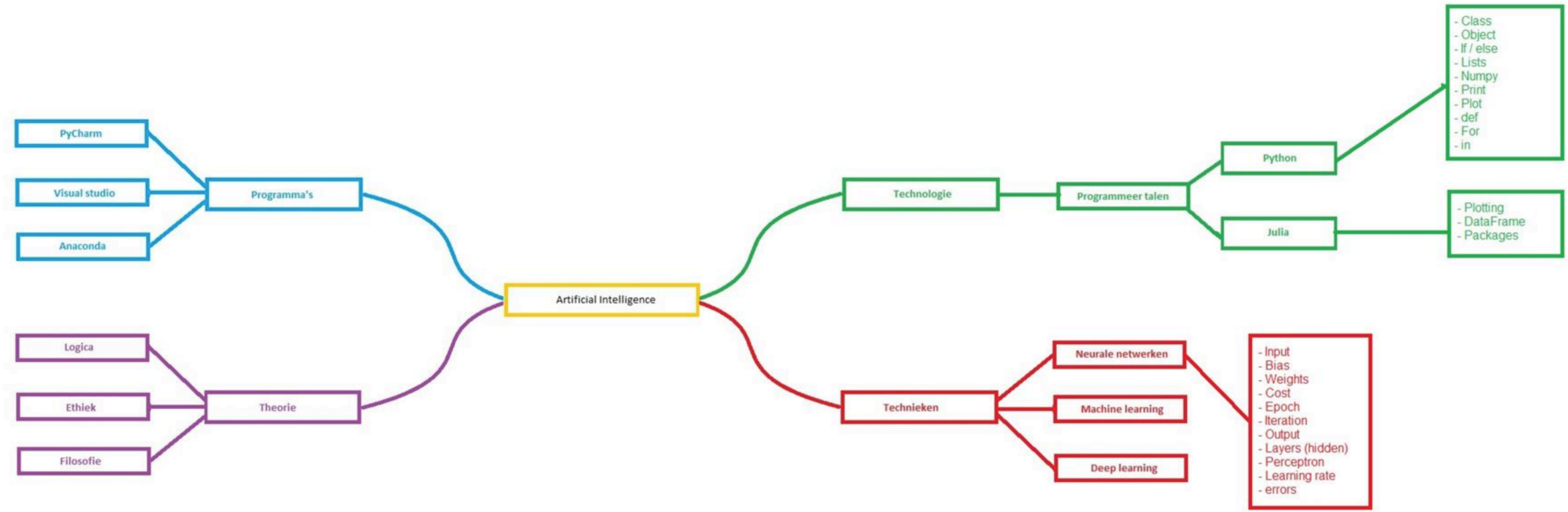
Jessy van Lier (0952765)



Yousef El Mokaddem (0950940)



Mark Vollebregt (0951040)



Martin Beverloo (0946822)