Title: Introduction to Machine Learning

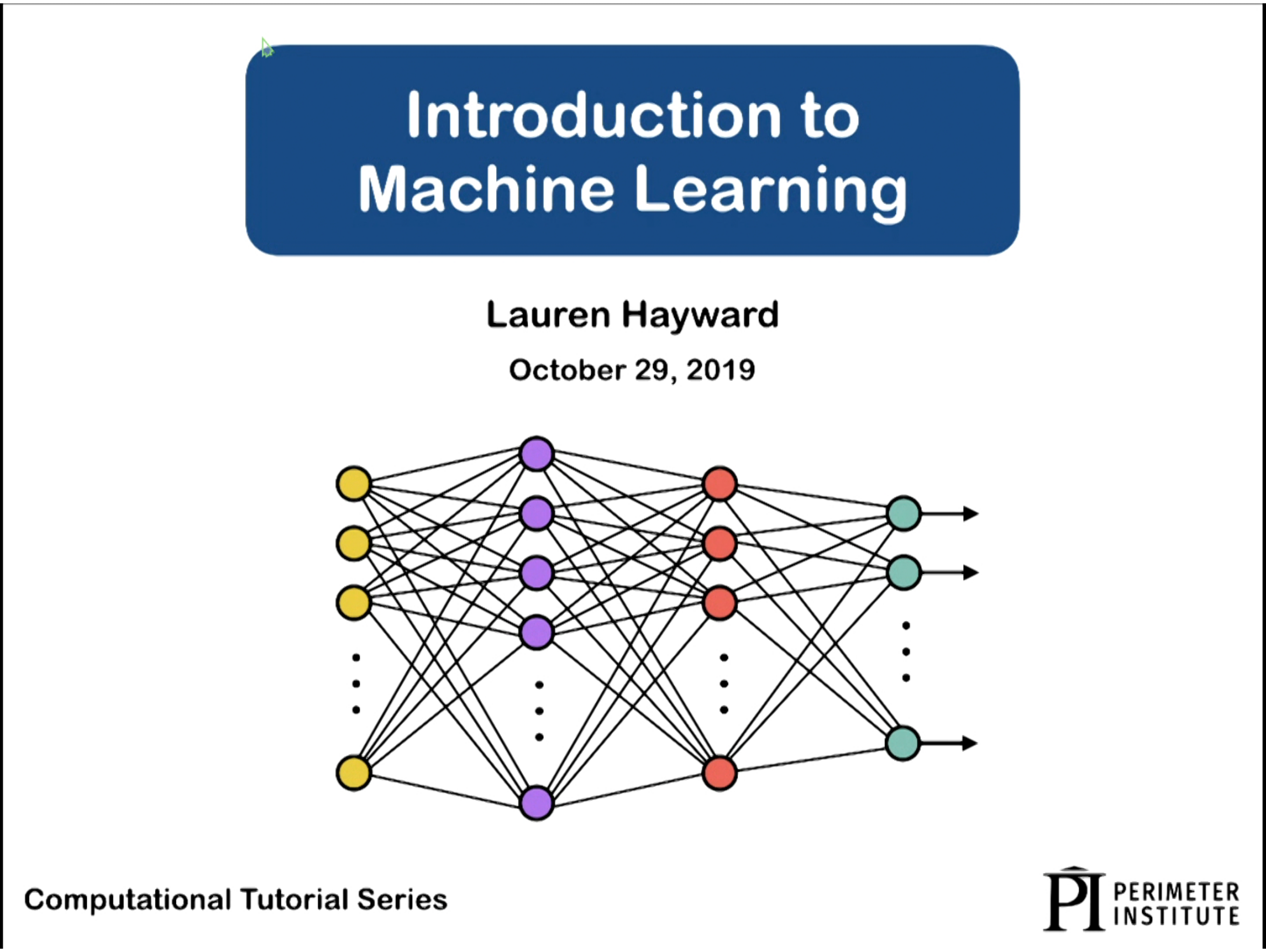
Speakers: Lauren Hayward
Date: October 29, 2019 - 1:00 PM

URL: http://pirsa.org/19100089

Abstract: Machine learning has led to recent advancements in image processing, language translation, finance, robotics, musical and visual arts, and medical diagnosis. In this session, we will explore how machine learning can be applied within fields of physics. We will introduce fundamental concepts in machine learning such a neural networks and supervised vs. unsupervised learning, and then proceed to learn to use tools from Python's TensorFlow library.

&nbsp;

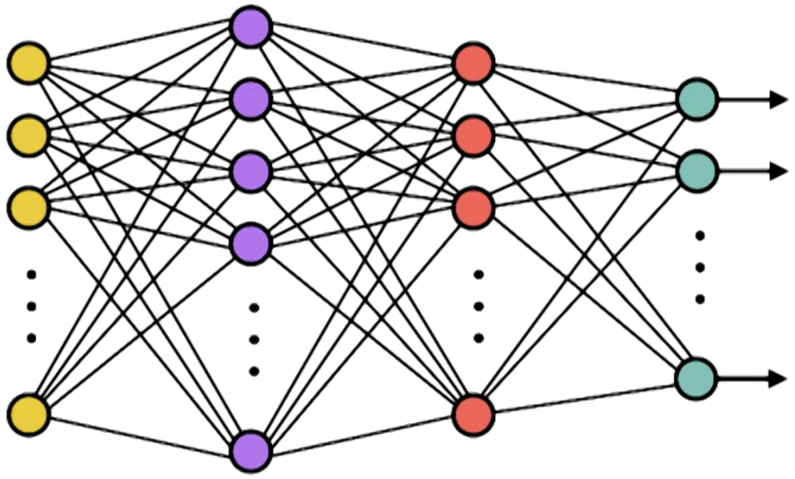
Bring your laptop. You can attend remotely via Zoom &lt;https://zoom.us/j/154009181&gt;.



# Introduction to Machine Learning

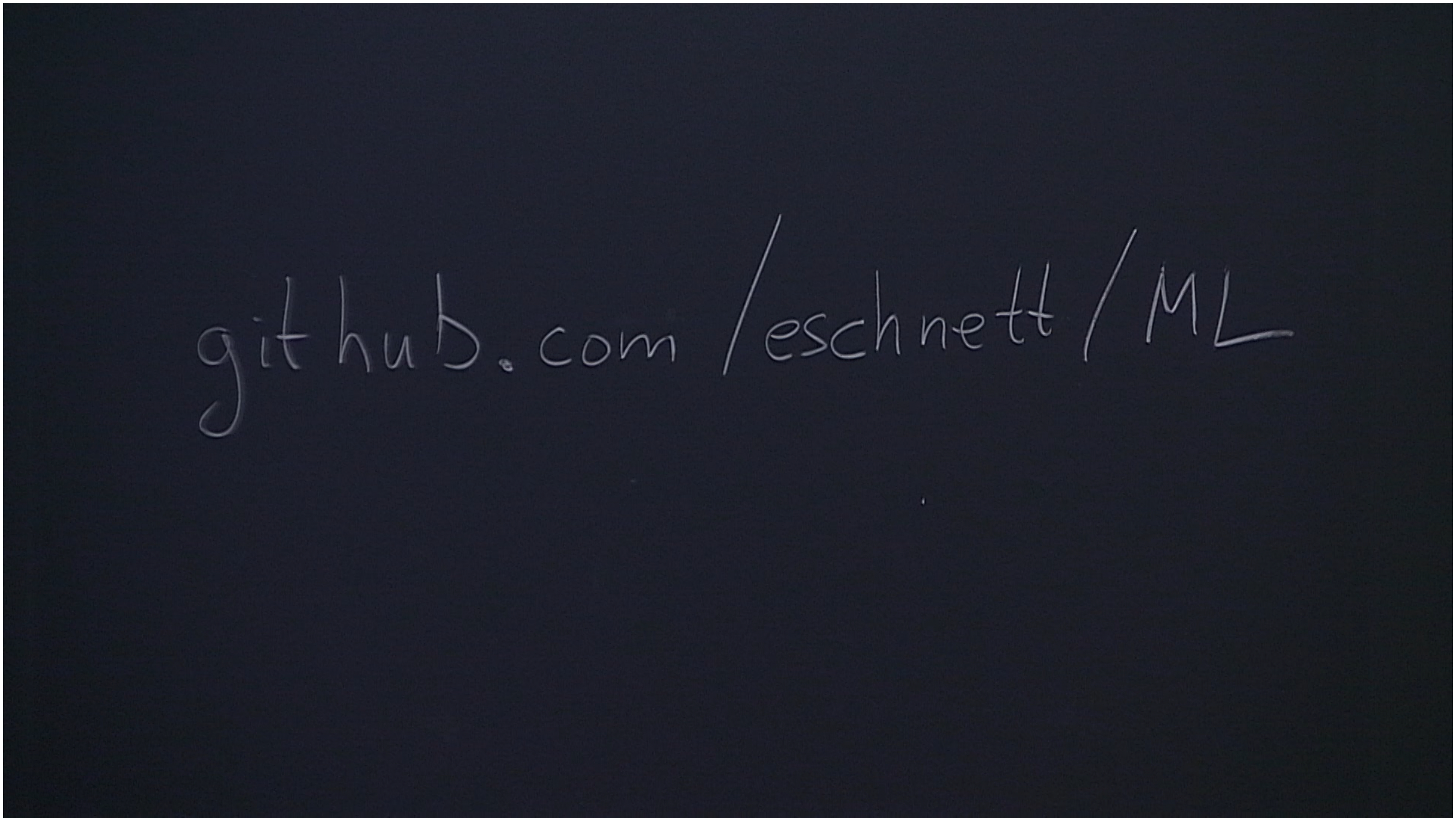
Lauren Hayward

October 29, 2019



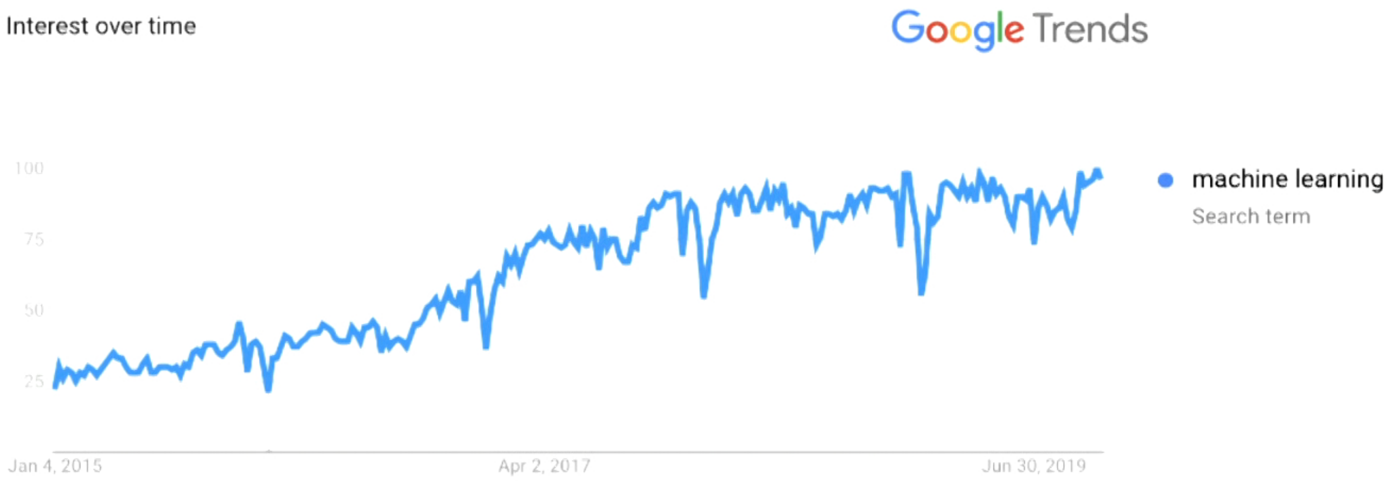
Computational Tutorial Series

PERIMETER INSTITUTE

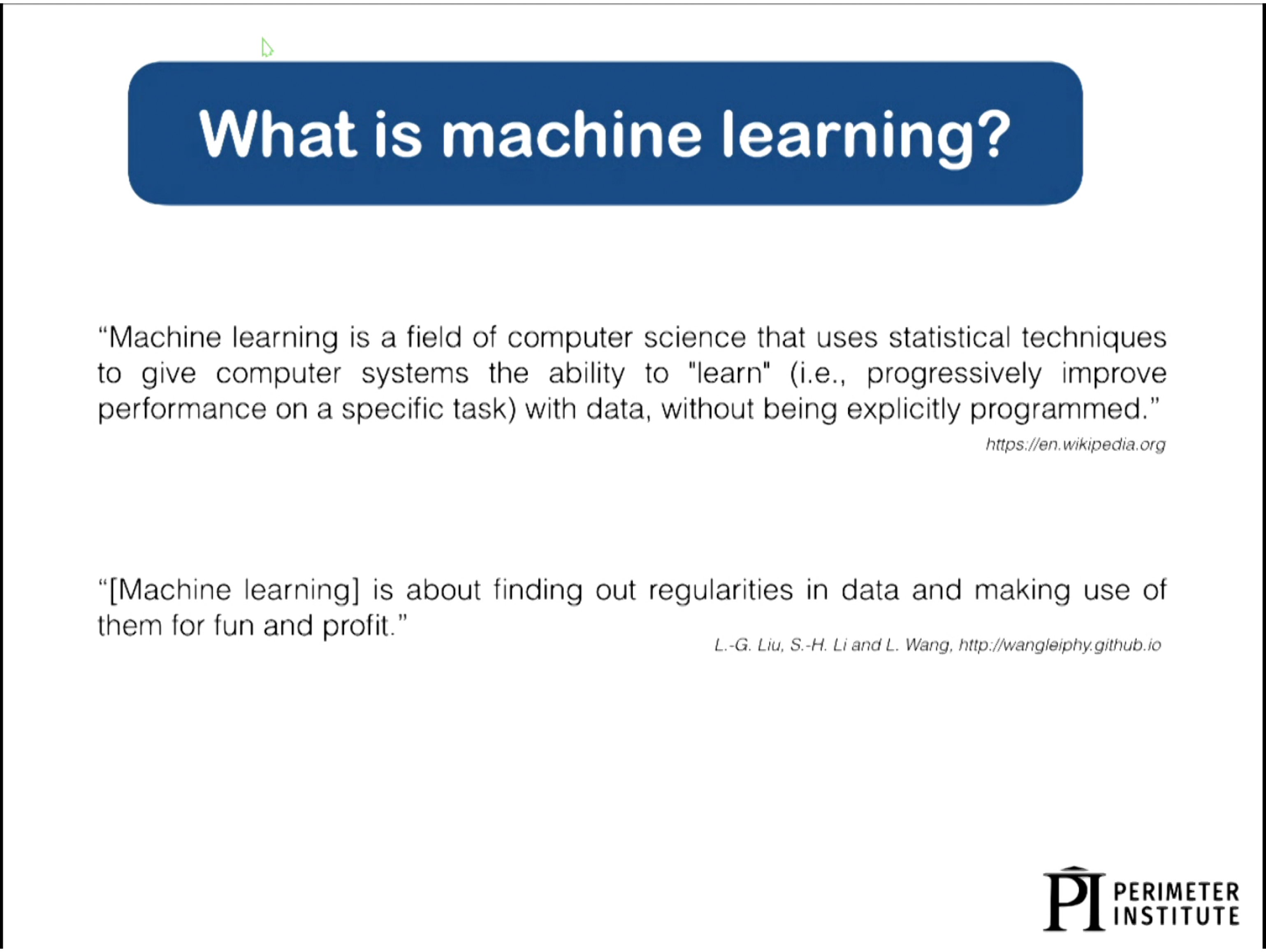


github.com/eschnett/ML

## Machine learning popularity



PERIMETER INSTITUTE



## What is machine learning?

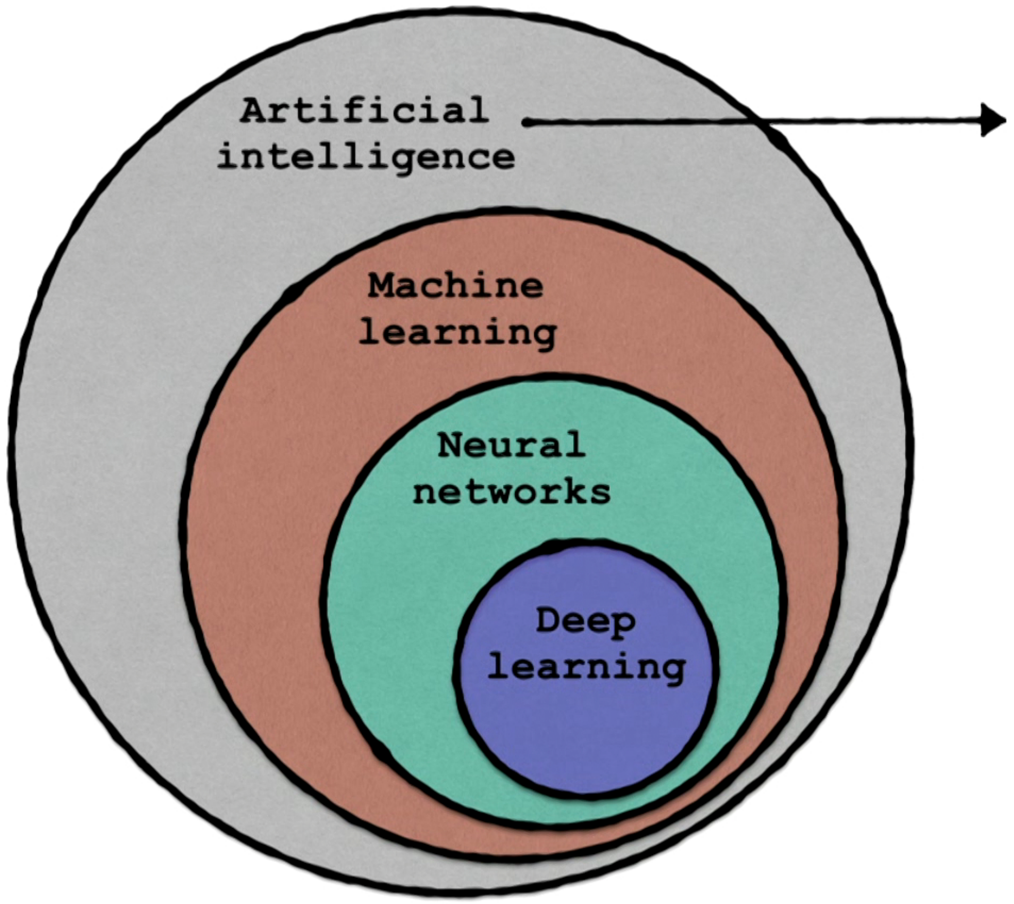
“Machine learning is a field of computer science that uses statistical techniques to give computer systems the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed.”

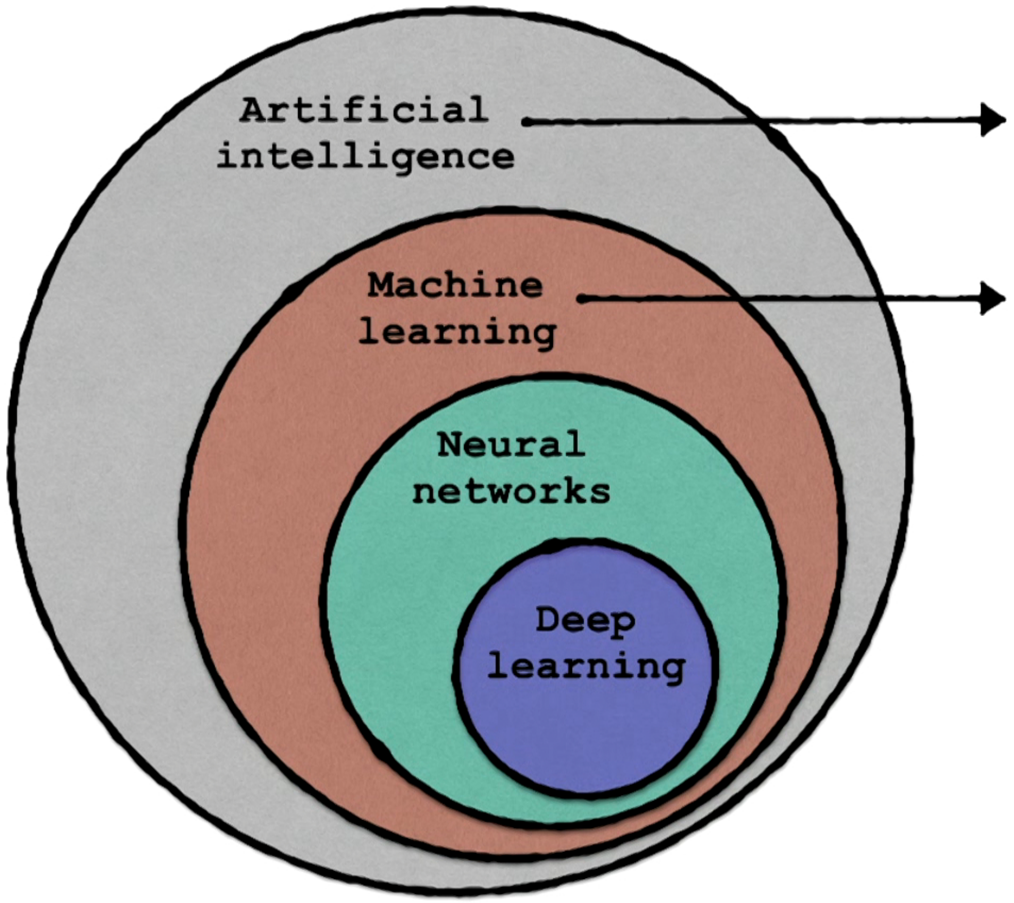
*https://en.wikipedia.org*

“[Machine learning] is about finding out regularities in data and making use of them for fun and profit.”

*L.-G. Liu, S.-H. Li and L. Wang, http://wangleiphy.github.io*

PERIMETER INSTITUTE

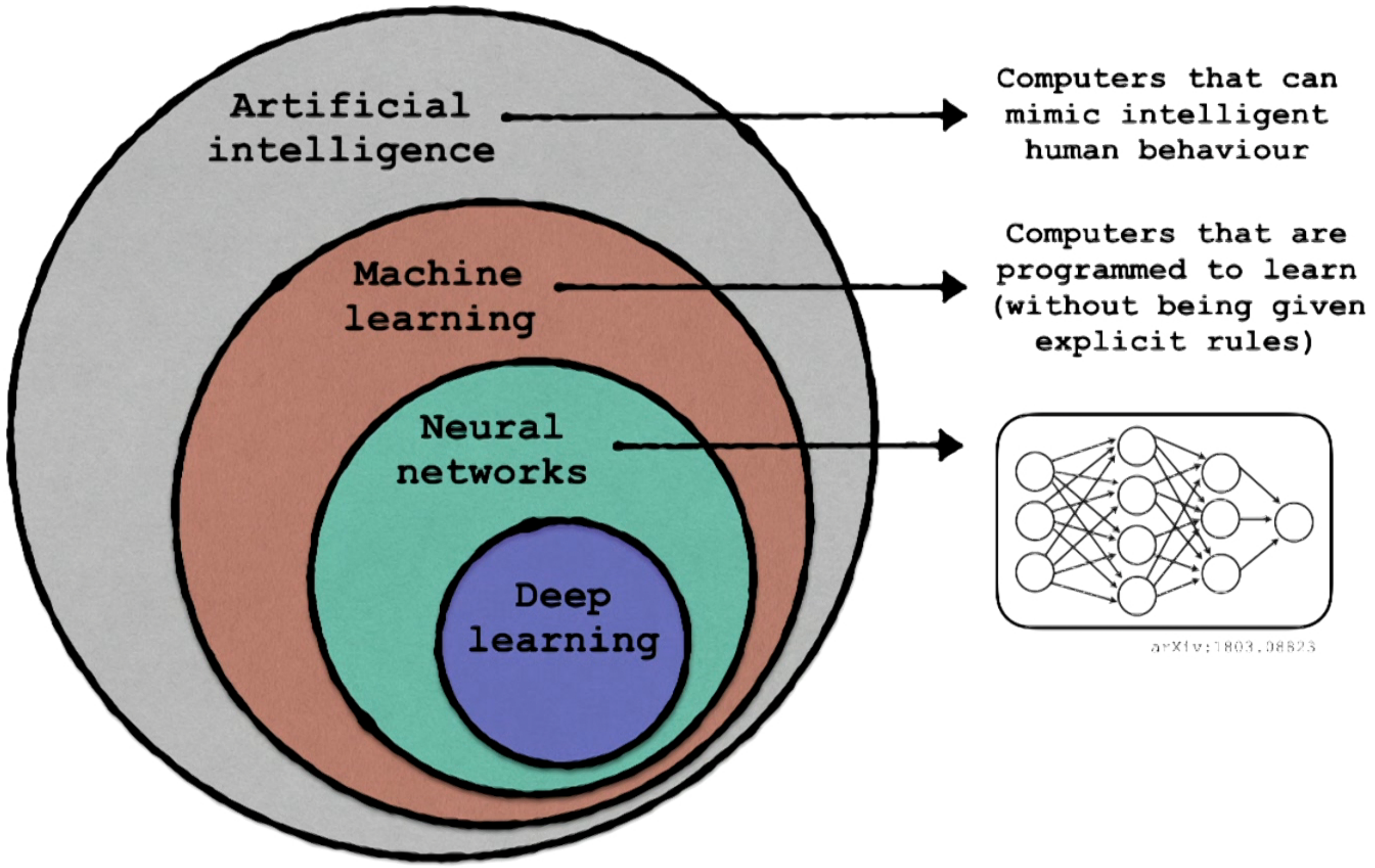

Artificial intelligence
Machine learning
Neural networks
Deep learning
Computers that can mimic intelligent human behaviour



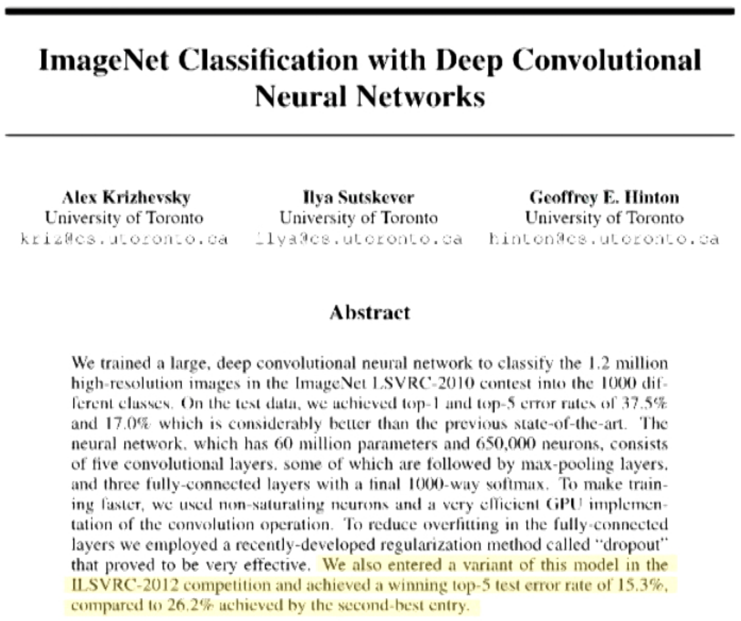
Computers that can mimic intelligent human behaviour

Computers that are programmed to learn (without being given explicit rules)

PERIMETER INSTITUTE


Artificial intelligence
Machine learning
Neural networks
Deep learning
Computers that can mimic intelligent human behaviour
Computers that are programmed to learn (without being given explicit rules)
arXiv:1803.08823

## Image classification



# ImageNet Classification with Deep Convolutional Neural Networks

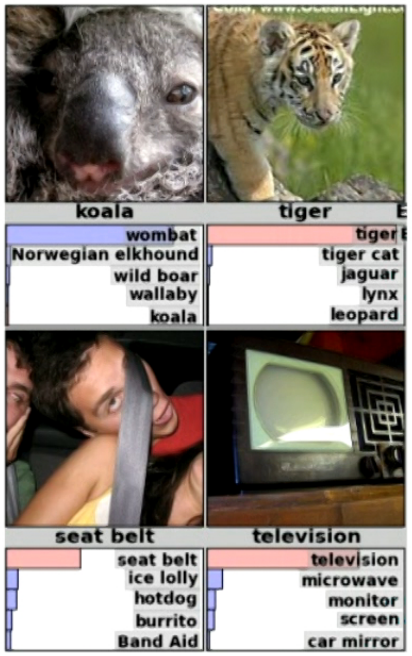
**Alex Krizhevsky**
University of Toronto
kriz@cs.utoronto.ca

**Ilya Sutskever**
University of Toronto
ilya@cs.utoronto.ca

**Geoffrey E. Hinton**
University of Toronto
hinton@cs.utoronto.ca

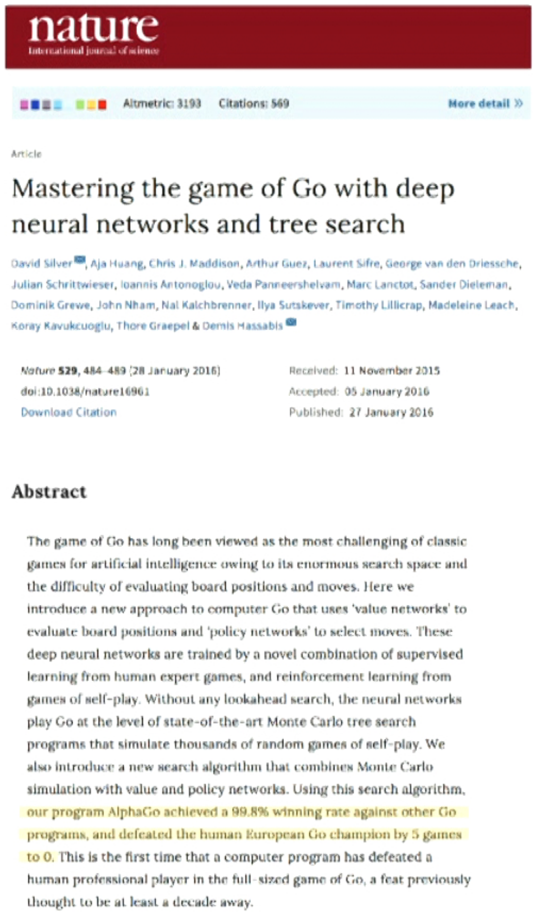
**Abstract**

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.



2012

# AlphaGo



nature
International journal of science

Altmetric: 3193 Citations: 569 More detail »

Article

## Mastering the game of Go with deep neural networks and tree search

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel & Demis Hassabis

*Nature* **529**, 484–489 (28 January 2016)
doi:10.1038/nature16961
Download Citation

Received: 11 November 2015
Accepted: 05 January 2016
Published: 27 January 2016

### Abstract

The game of Go has long been viewed as the most challenging of classic games for artificial intelligence owing to its enormous search space and the difficulty of evaluating board positions and moves. Here we introduce a new approach to computer Go that uses 'value networks' to evaluate board positions and 'policy networks' to select moves. These deep neural networks are trained by a novel combination of supervised learning from human expert games, and reinforcement learning from games of self-play. Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play. We also introduce a new search algorithm that combines Monte Carlo simulation with value and policy networks. Using this search algorithm, our program AlphaGo achieved a 99.8% winning rate against other Go programs, and defeated the human European Go champion by 5 games to 0. This is the first time that a computer program has defeated a human professional player in the full-sized game of Go, a feat previously thought to be at least a decade away.

## AlphaGo seals 4-1 victory over Go grandmaster Lee Sedol

**DeepMind's artificial intelligence astonishes fans to defeat human opponent and offers evidence computer software has mastered a major challenge**

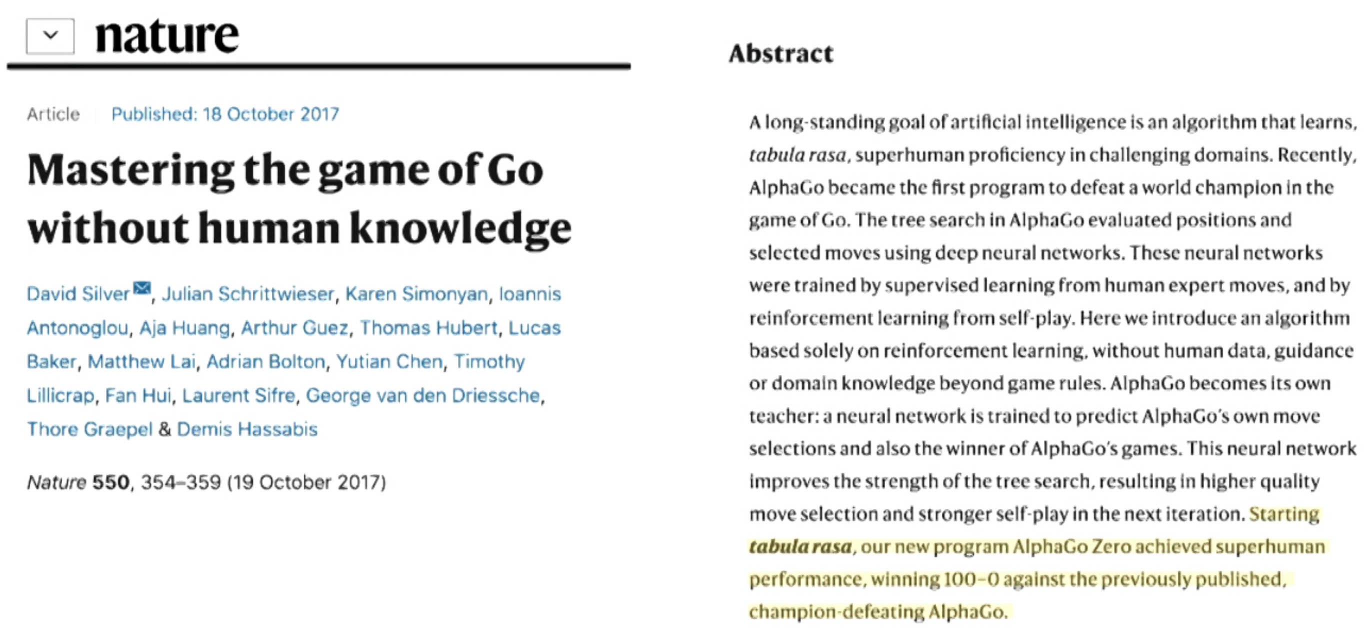
**Steven Borowiec**
Tue 15 Mar 2016 10.12 GMT

https://www.theguardian.com

2016

PERIMETER INSTITUTE

## AlphaGo Zero



**nature**

Article | Published: 18 October 2017

### Mastering the game of Go without human knowledge

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel & Demis Hassabis

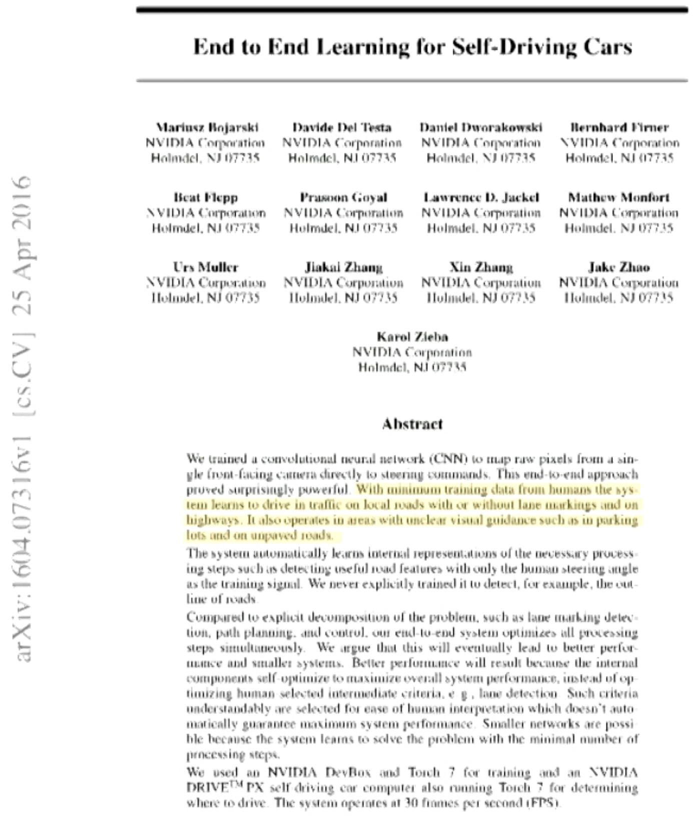
*Nature* **550**, 354–359 (19 October 2017)

**Abstract**

A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting ***tabula rasa***, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

2017

# Self-driving cars



## End to End Learning for Self-Driving Cars

Mariusz Bojarski, NVIDIA Corporation, Holmdel, NJ 07735
Davide Del Testa, NVIDIA Corporation, Holmdel, NJ 07735
Daniel Dworakowski, NVIDIA Corporation, Holmdel, NJ 07735
Bernhard Firner, NVIDIA Corporation, Holmdel, NJ 07735
Beat Flepp, NVIDIA Corporation, Holmdel, NJ 07735
Prasoon Goyal, NVIDIA Corporation, Holmdel, NJ 07735
Lawrence D. Jackel, NVIDIA Corporation, Holmdel, NJ 07735
Mathew Monfort, NVIDIA Corporation, Holmdel, NJ 07735
Urs Muller, NVIDIA Corporation, Holmdel, NJ 07735
Jiakai Zhang, NVIDIA Corporation, Holmdel, NJ 07735
Xin Zhang, NVIDIA Corporation, Holmdel, NJ 07735
Jake Zhao, NVIDIA Corporation, Holmdel, NJ 07735
Karol Zieba, NVIDIA Corporation, Holmdel, NJ 07735

arXiv:1604.07316v1 [cs.CV] 25 Apr 2016

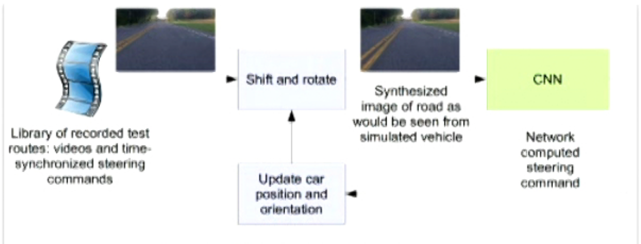
### Abstract

We trained a convolutional neural network (CNN) to map raw pixels from a single front-facing camera directly to steering commands. This end-to-end approach proved surprisingly powerful. With minimum training data from humans the system learns to drive in traffic on local roads with or without lane markings and on highways. It also operates in areas with unclear visual guidance such as in parking lots and on unpaved roads.

The system automatically learns internal representations of the necessary processing steps such as detecting useful road features with only the human steering angle as the training signal. We never explicitly trained it to detect, for example, the outline of roads.

Compared to explicit decomposition of the problem, such as lane marking detection, path planning, and control, our end-to-end system optimizes all processing steps simultaneously. We argue that this will eventually lead to better performance and smaller systems. Better performance will result because the internal components self-optimize to maximize overall system performance, instead of optimizing human-selected intermediate criteria, e. g., lane detection. Such criteria understandably are selected for ease of human interpretation which doesn't automatically guarantee maximum system performance. Smaller networks are possible because the system learns to solve the problem with the minimal number of processing steps.

We used an NVIDIA DevBox and Torch 7 for training and an NVIDIA DRIVE™ PX self-driving car computer also running Torch 7 for determining where to drive. The system operates at 30 frames per second (FPS).

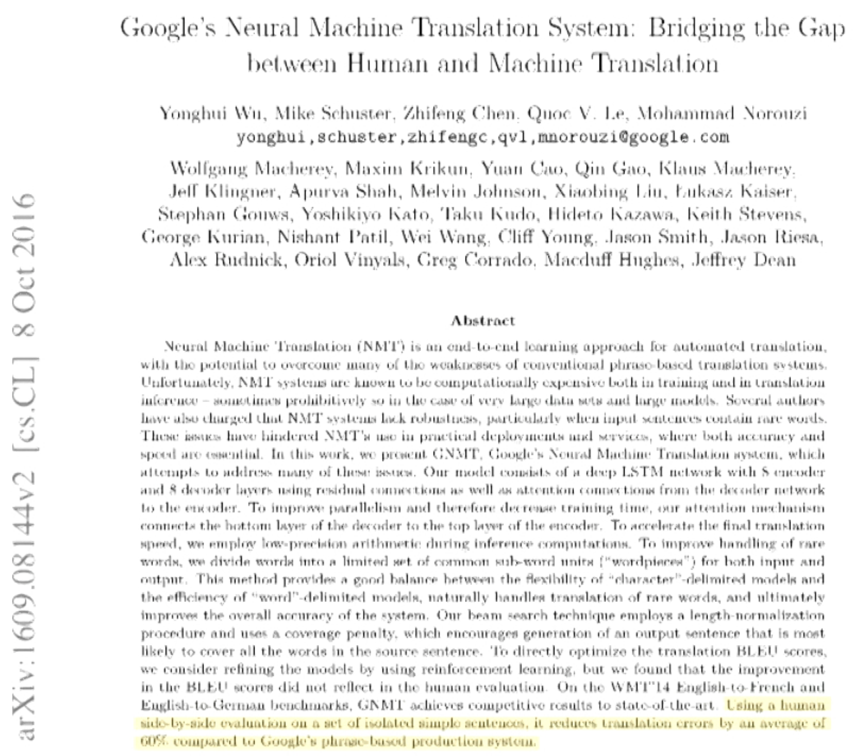


Library of recorded test routes: videos and time-synchronized steering commands
Shift and rotate
Update car position and orientation
Synthesized image of road as would be seen from simulated vehicle
CNN
Network computed steering command

Figure 5: Block-diagram of the drive simulator.

2016

# Language translation



Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi
yonghui,schuster,zhifengc,qvl,mnorouzi@google.com

Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean

arXiv:1609.08144v2 [cs.CL] 8 Oct 2016

**Abstract**

Neural Machine Translation (NMT) is an end-to-end learning approach for automated translation, with the potential to overcome many of the weaknesses of conventional phrase-based translation systems. Unfortunately, NMT systems are known to be computationally expensive both in training and in translation inference – sometimes prohibitively so in the case of very large data sets and large models. Several authors have also charged that NMT systems lack robustness, particularly when input sentences contain rare words. These issues have hindered NMT's use in practical deployments and services, where both accuracy and speed are essential. In this work, we present GNMT, Google's Neural Machine Translation system, which attempts to address many of these issues. Our model consists of a deep LSTM network with 8 encoder and 8 decoder layers using residual connections as well as attention connections from the decoder network to the encoder. To improve parallelism and therefore decrease training time, our attention mechanism connects the bottom layer of the decoder to the top layer of the encoder. To accelerate the final translation speed, we employ low-precision arithmetic during inference computations. To improve handling of rare words, we divide words into a limited set of common sub-word units ("wordpieces") for both input and output. This method provides a good balance between the flexibility of "character"-delimited models and the efficiency of "word"-delimited models, naturally handles translation of rare words, and ultimately improves the overall accuracy of the system. Our beam search technique employs a length-normalization procedure and uses a coverage penalty, which encourages generation of an output sentence that is most likely to cover all the words in the source sentence. To directly optimize the translation BLEU scores, we consider refining the models by using reinforcement learning, but we found that the improvement in the BLEU scores did not reflect in the human evaluation. On the WMT'14 English-to-French and English-to-German benchmarks, GNMT achieves competitive results to state-of-the-art. Using a human side-by-side evaluation on a set of isolated simple sentences, it reduces translation errors by an average of 60% compared to Google's phrase-based production system.

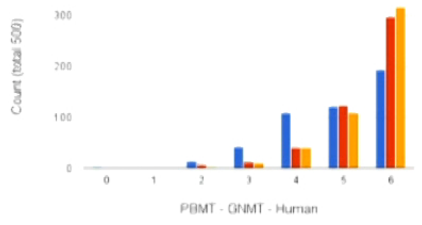
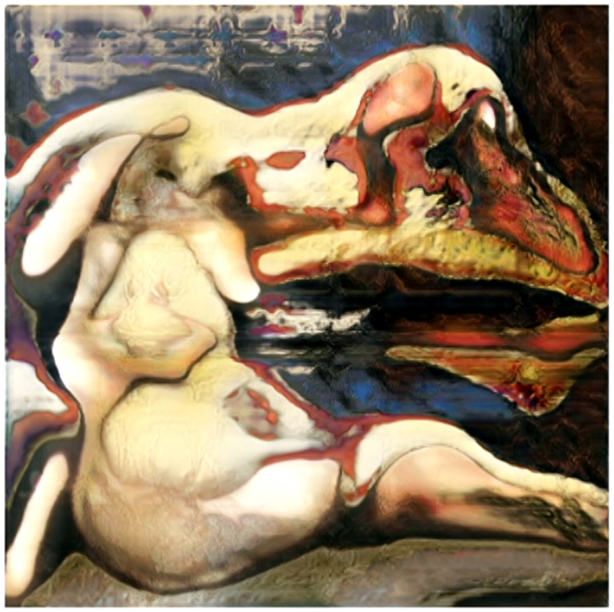


Figure 6: Histogram of side-by-side scores on 500 sampled sentences from Wikipedia and news websites for a typical language pair, here English → Spanish (PBMT blue, GNMT red, Human orange). It can be seen that there is a wide distribution in scores, even for the human translation when rated by other humans, which shows how ambiguous the task is. It is clear that GNMT is much more accurate than PBMT.

2016

## Generating art

Robbie Barrat
https://robbiebarrat.github.io



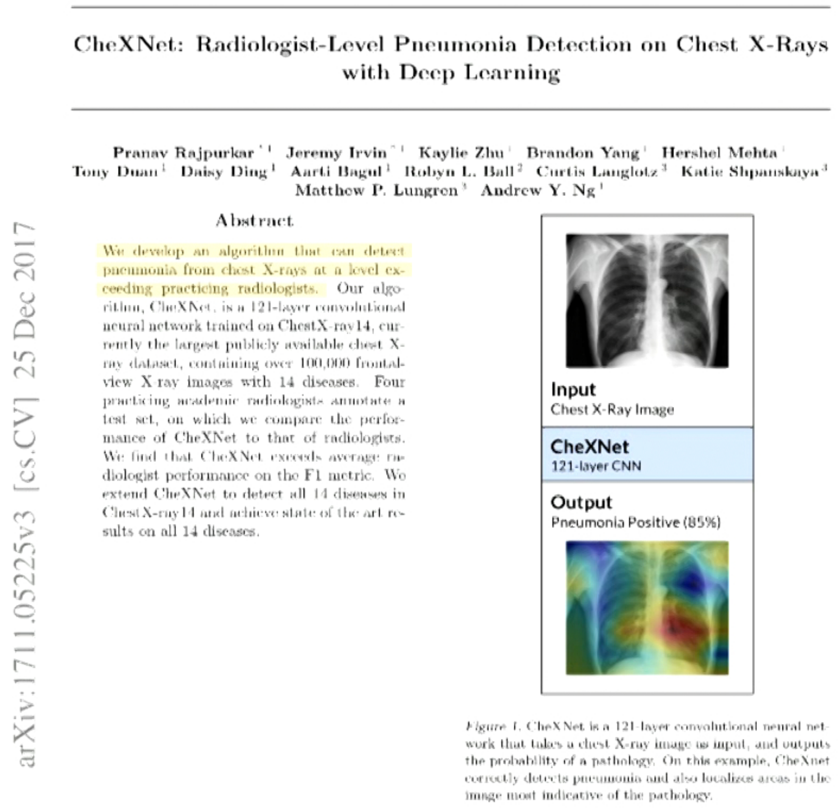
## Computer-generated people

https://thispersondoesnotexist.com

2018

PERIMETER INSTITUTE

# Medical diagnosis



CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning

Pranav Rajpurkar [*1] Jeremy Irvin [*1] Kaylie Zhu [1] Brandon Yang [1] Hershel Mehta [1]
Tony Duan [1] Daisy Ding [1] Aarti Bagul [1] Robyn L. Ball [2] Curtis Langlotz [3] Katie Shpanskaya [3]
Matthew P. Lungren [3] Andrew Y. Ng [1]

arXiv:1711.05225v3 [cs.CV] 25 Dec 2017

Abstract

We develop an algorithm that can detect pneumonia from chest X-rays at a level exceeding practicing radiologists. Our algorithm, CheXNet, is a 121-layer convolutional neural network trained on ChestX-ray14, currently the largest publicly available chest X-ray dataset, containing over 100,000 frontal-view X-ray images with 14 diseases. Four practicing academic radiologists annotate a test set, on which we compare the performance of CheXNet to that of radiologists. We find that CheXNet exceeds average radiologist performance on the F1 metric. We extend CheXNet to detect all 14 diseases in ChestX-ray14 and achieve state of the art results on all 14 diseases.

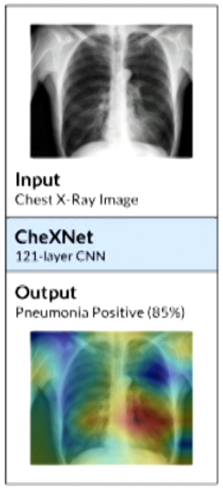
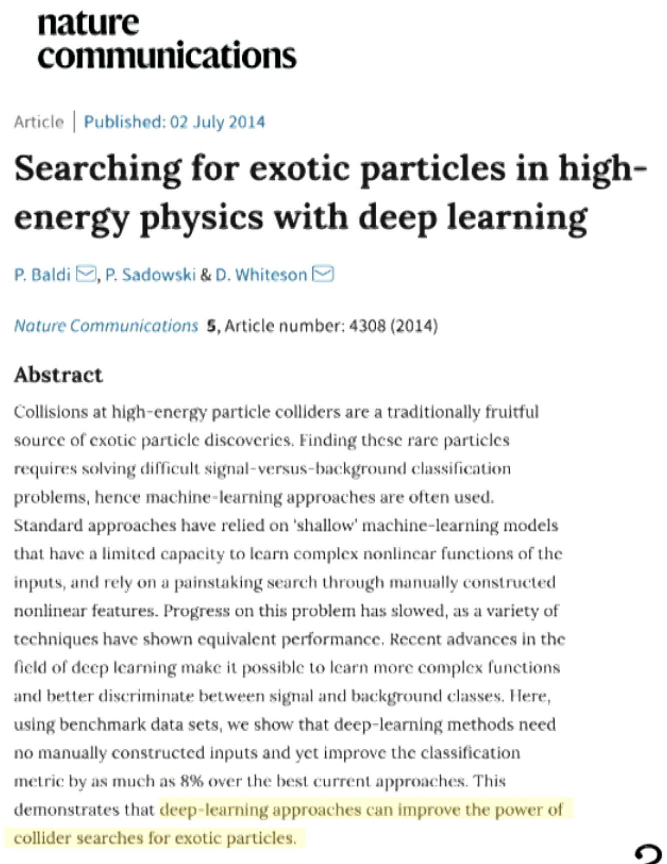


*Figure 1.* CheXNet is a 121-layer convolutional neural network that takes a chest X-ray image as input, and outputs the probability of a pathology. On this example, CheXnet correctly detects pneumonia and also localizes areas in the image most indicative of the pathology.

2017

## Machine Learning for Physics



**nature communications**

Article | Published: 02 July 2014

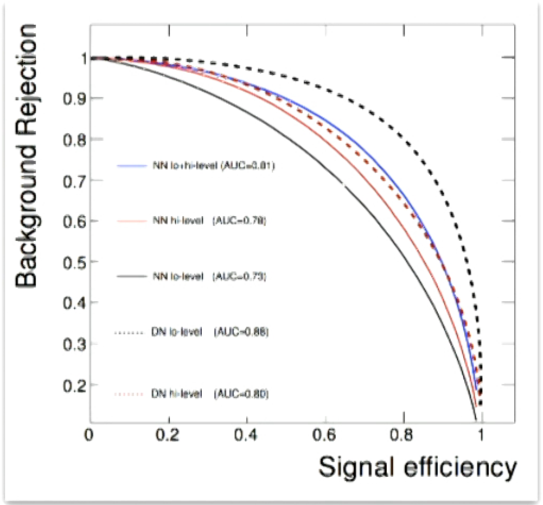
### Searching for exotic particles in high-energy physics with deep learning

P. Baldi, P. Sadowski & D. Whiteson

*Nature Communications* **5**, Article number: 4308 (2014)

**Abstract**

Collisions at high-energy particle colliders are a traditionally fruitful source of exotic particle discoveries. Finding these rare particles requires solving difficult signal-versus-background classification problems, hence machine-learning approaches are often used. Standard approaches have relied on 'shallow' machine-learning models that have a limited capacity to learn complex nonlinear functions of the inputs, and rely on a painstaking search through manually constructed nonlinear features. Progress on this problem has slowed, as a variety of techniques have shown equivalent performance. Recent advances in the field of deep learning make it possible to learn more complex functions and better discriminate between signal and background classes. Here, using benchmark data sets, we show that deep-learning methods need no manually constructed inputs and yet improve the classification metric by as much as 8% over the best current approaches. This demonstrates that deep-learning approaches can improve the power of collider searches for exotic particles.



arXiv:1806.11484

2014

PERIMETER INSTITUTE

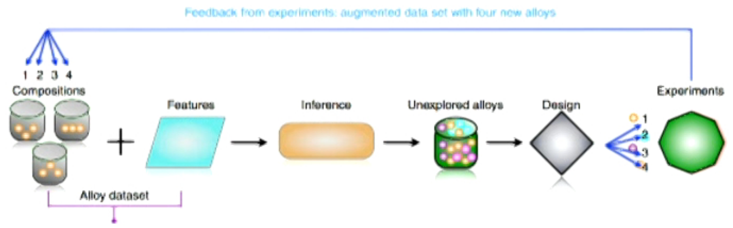
# Machine Learning for Physics

nature COMMUNICATIONS

## Accelerated search for materials with targeted properties by adaptive design

Dezhen Xue[1,2], Prasanna V. Balachandran[1], John Hogden[3], James Theiler[4], Deqing Xue[2] & Turab Lookman[1]

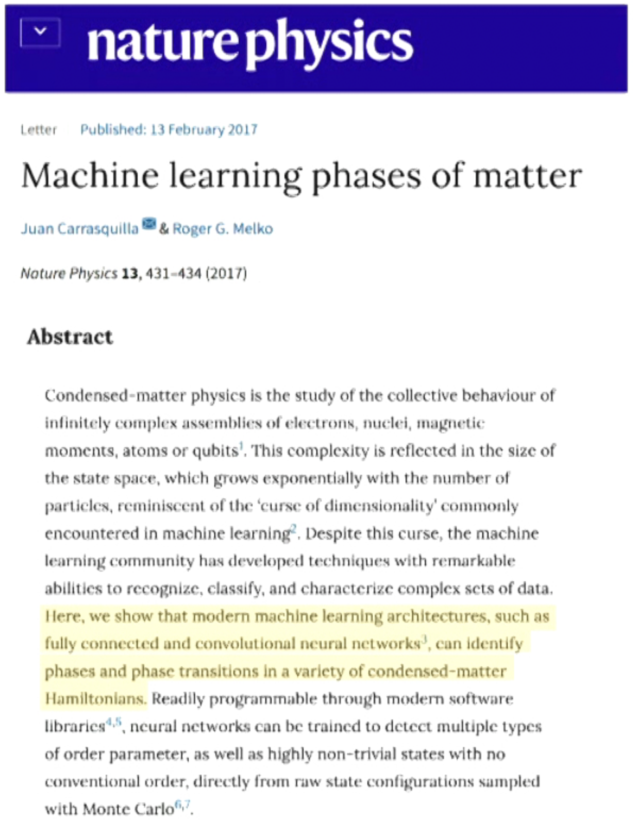
Finding new materials with targeted properties has traditionally been guided by intuition, and trial and error. With increasing chemical complexity, the combinatorial possibilities are too large for an Edisonian approach to be practical. Here we show how an adaptive design strategy, tightly coupled with experiments, can accelerate the discovery process by sequentially identifying the next experiments or calculations, to effectively navigate the complex search space. Our strategy uses inference and global optimization to balance the trade-off between exploitation and exploration of the search space. We demonstrate this by finding very low thermal hysteresis ($\Delta T$) NiTi-based shape memory alloys, with $Ti_{50.0}Ni_{46.7}Cu_{0.8}Fe_{2.3}Pd_{0.2}$ possessing the smallest $\Delta T$ (1.84 K). We synthesize and characterize 36 predicted compositions (9 feedback loops) from a potential space of ~800,000 compositions. Of these, 14 had smaller $\Delta T$ than any of the 22 in the original data set.



2016

PERIMETER INSTITUTE

# Machine Learning for Physics



nature physics

Letter Published: 13 February 2017

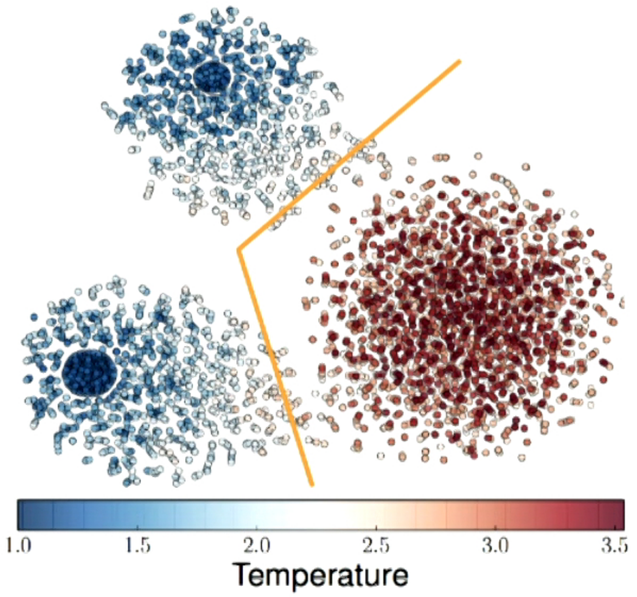
Machine learning phases of matter

Juan Carrasquilla & Roger G. Melko

*Nature Physics* **13**, 431–434 (2017)

**Abstract**

Condensed-matter physics is the study of the collective behaviour of infinitely complex assemblies of electrons, nuclei, magnetic moments, atoms or qubits[1]. This complexity is reflected in the size of the state space, which grows exponentially with the number of particles, reminiscent of the 'curse of dimensionality' commonly encountered in machine learning[2]. Despite this curse, the machine learning community has developed techniques with remarkable abilities to recognize, classify, and characterize complex sets of data. Here, we show that modern machine learning architectures, such as fully connected and convolutional neural networks[3], can identify phases and phase transitions in a variety of condensed-matter Hamiltonians. Readily programmable through modern software libraries[4,5], neural networks can be trained to detect multiple types of order parameter, as well as highly non-trivial states with no conventional order, directly from raw state configurations sampled with Monte Carlo[6,7].

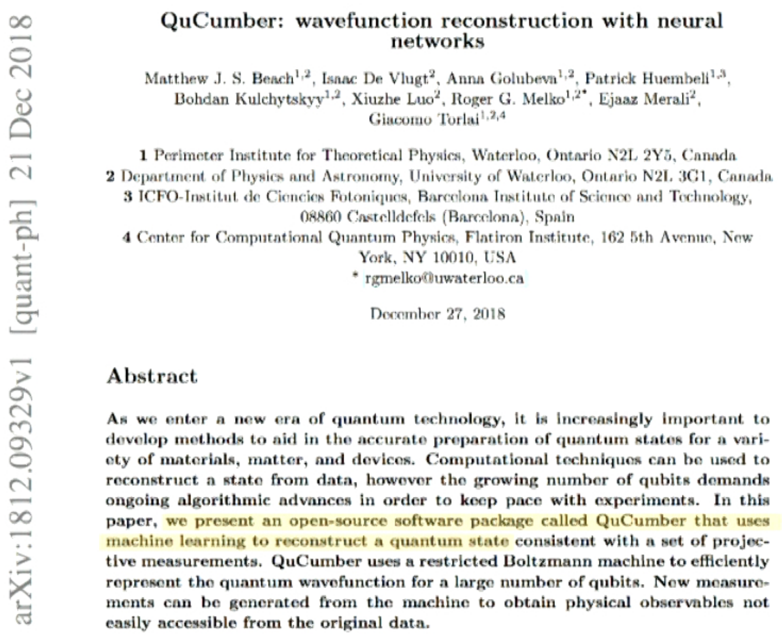


2016

PERIMETER INSTITUTE

# Machine Learning for Physics

QuCUMBER



arXiv:1812.09329v1 [quant-ph] 21 Dec 2018

QuCumber: wavefunction reconstruction with neural networks

Matthew J. S. Beach[1,2], Isaac De Vlugt[2], Anna Golubeva[1,2], Patrick Huembeli[1,3], Bohdan Kulchytskyy[1,2], Xiuzhe Luo[2], Roger G. Melko[1,2*], Ejaaz Merali[2], Giacomo Torlai[1,2,4]

**1** Perimeter Institute for Theoretical Physics, Waterloo, Ontario N2L 2Y5, Canada
**2** Department of Physics and Astronomy, University of Waterloo, Ontario N2L 3G1, Canada
**3** ICFO-Institut de Ciencies Fotoniques, Barcelona Institute of Science and Technology, 08860 Castelldefels (Barcelona), Spain
**4** Center for Computational Quantum Physics, Flatiron Institute, 162 5th Avenue, New York, NY 10010, USA
* rgmelko@uwaterloo.ca

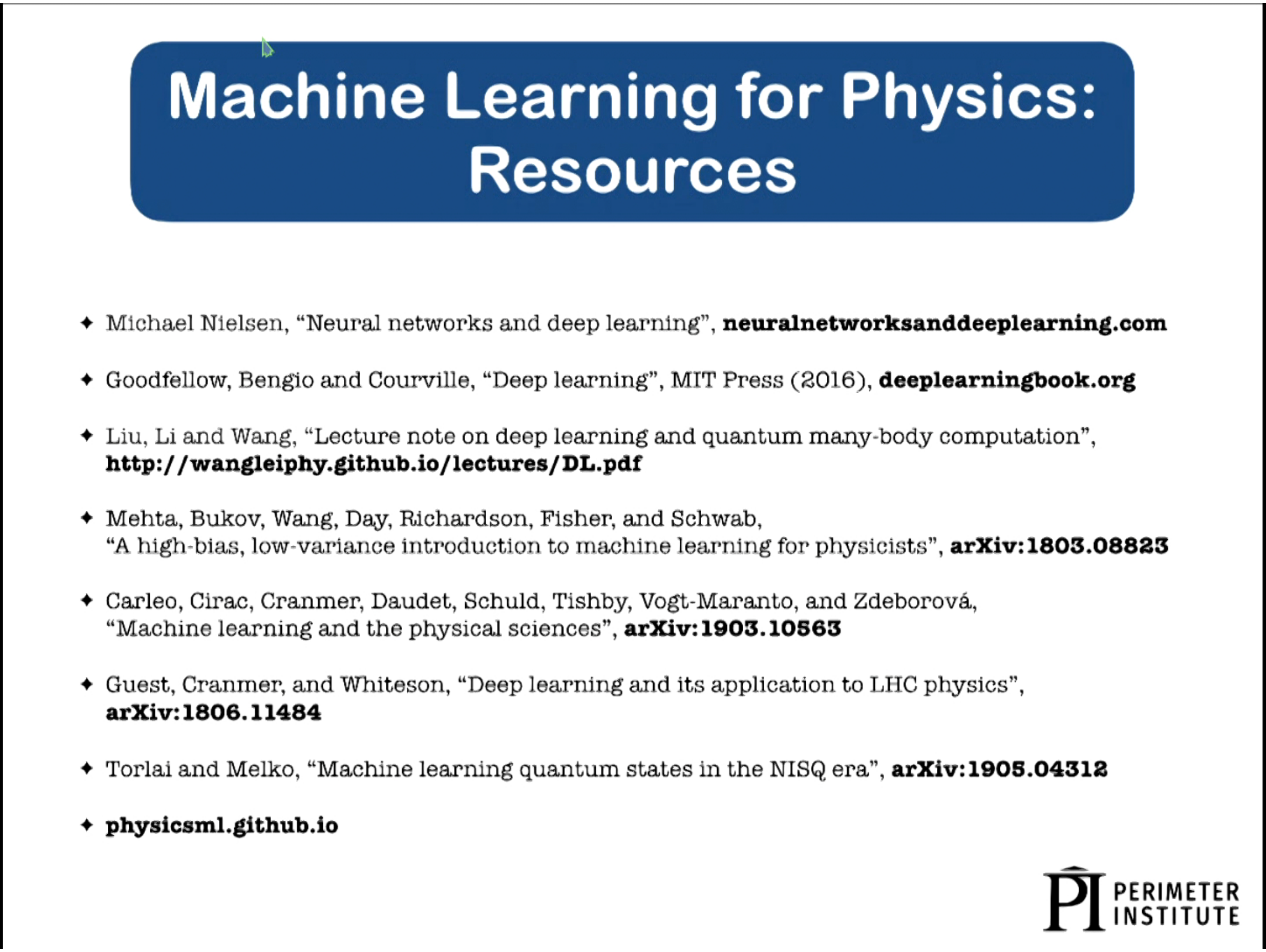
December 27, 2018

## Abstract

**As we enter a new era of quantum technology, it is increasingly important to develop methods to aid in the accurate preparation of quantum states for a variety of materials, matter, and devices. Computational techniques can be used to reconstruct a state from data, however the growing number of qubits demands ongoing algorithmic advances in order to keep pace with experiments. In this paper, we present an open-source software package called QuCumber that uses machine learning to reconstruct a quantum state consistent with a set of projective measurements. QuCumber uses a restricted Boltzmann machine to efficiently represent the quantum wavefunction for a large number of qubits. New measurements can be generated from the machine to obtain physical observables not easily accessible from the original data.**

2018

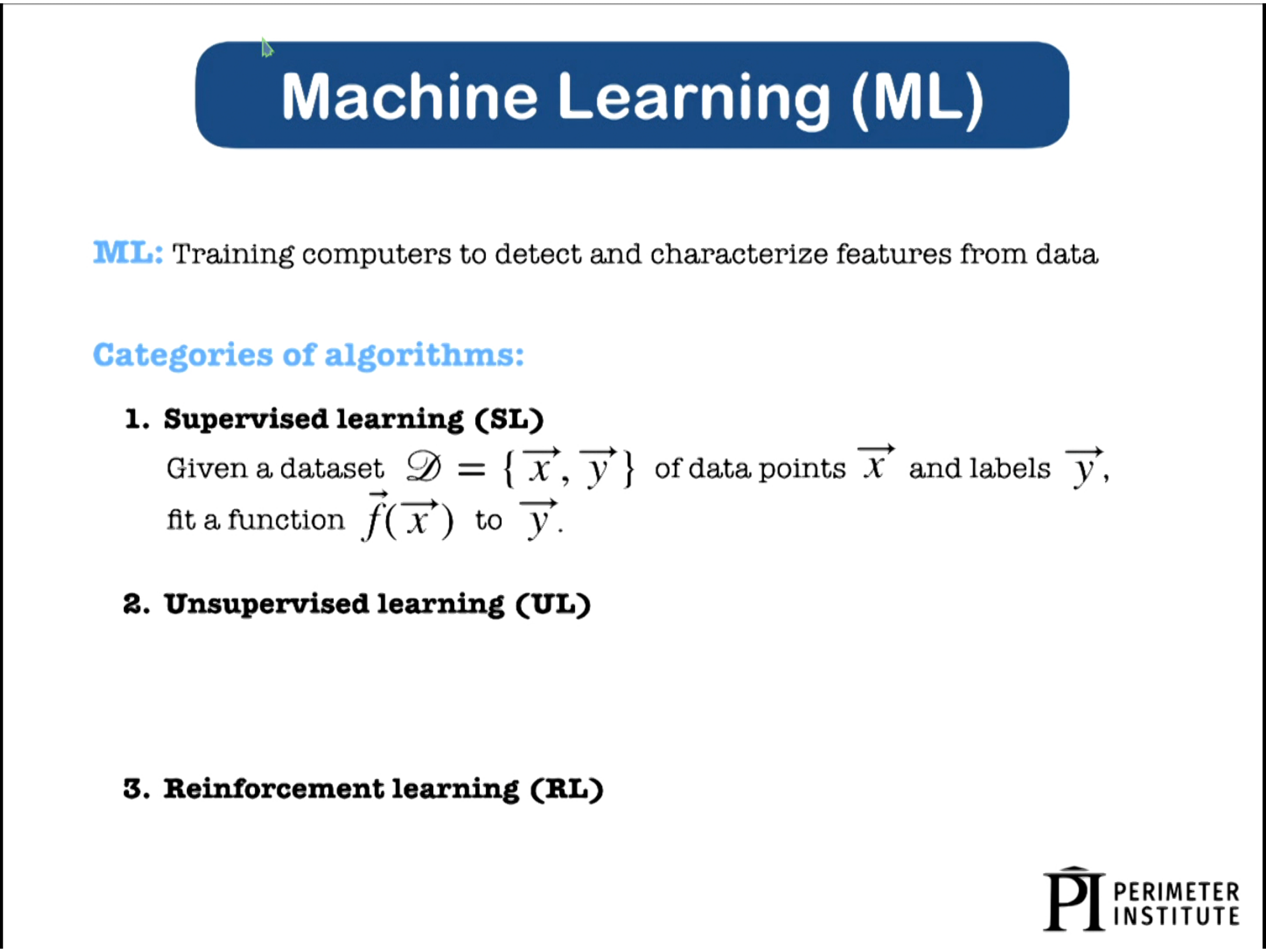
PERIMETER INSTITUTE



## Machine Learning for Physics: Resources

- ✦ Michael Nielsen, “Neural networks and deep learning”, **neuralnetworksanddeeplearning.com**
- ✦ Goodfellow, Bengio and Courville, “Deep learning”, MIT Press (2016), **deeplearningbook.org**
- ✦ Liu, Li and Wang, “Lecture note on deep learning and quantum many-body computation”, **http://wangleiphy.github.io/lectures/DL.pdf**
- ✦ Mehta, Bukov, Wang, Day, Richardson, Fisher, and Schwab, “A high-bias, low-variance introduction to machine learning for physicists”, **arXiv:1803.08823**
- ✦ Carleo, Cirac, Cranmer, Daudet, Schuld, Tishby, Vogt-Maranto, and Zdeborová, “Machine learning and the physical sciences”, **arXiv:1903.10563**
- ✦ Guest, Cranmer, and Whiteson, “Deep learning and its application to LHC physics”, **arXiv:1806.11484**
- ✦ Torlai and Melko, “Machine learning quantum states in the NISQ era”, **arXiv:1905.04312**
- ✦ **physicsml.github.io**

PERIMETER INSTITUTE



## Machine Learning (ML)

**ML:** Training computers to detect and characterize features from data

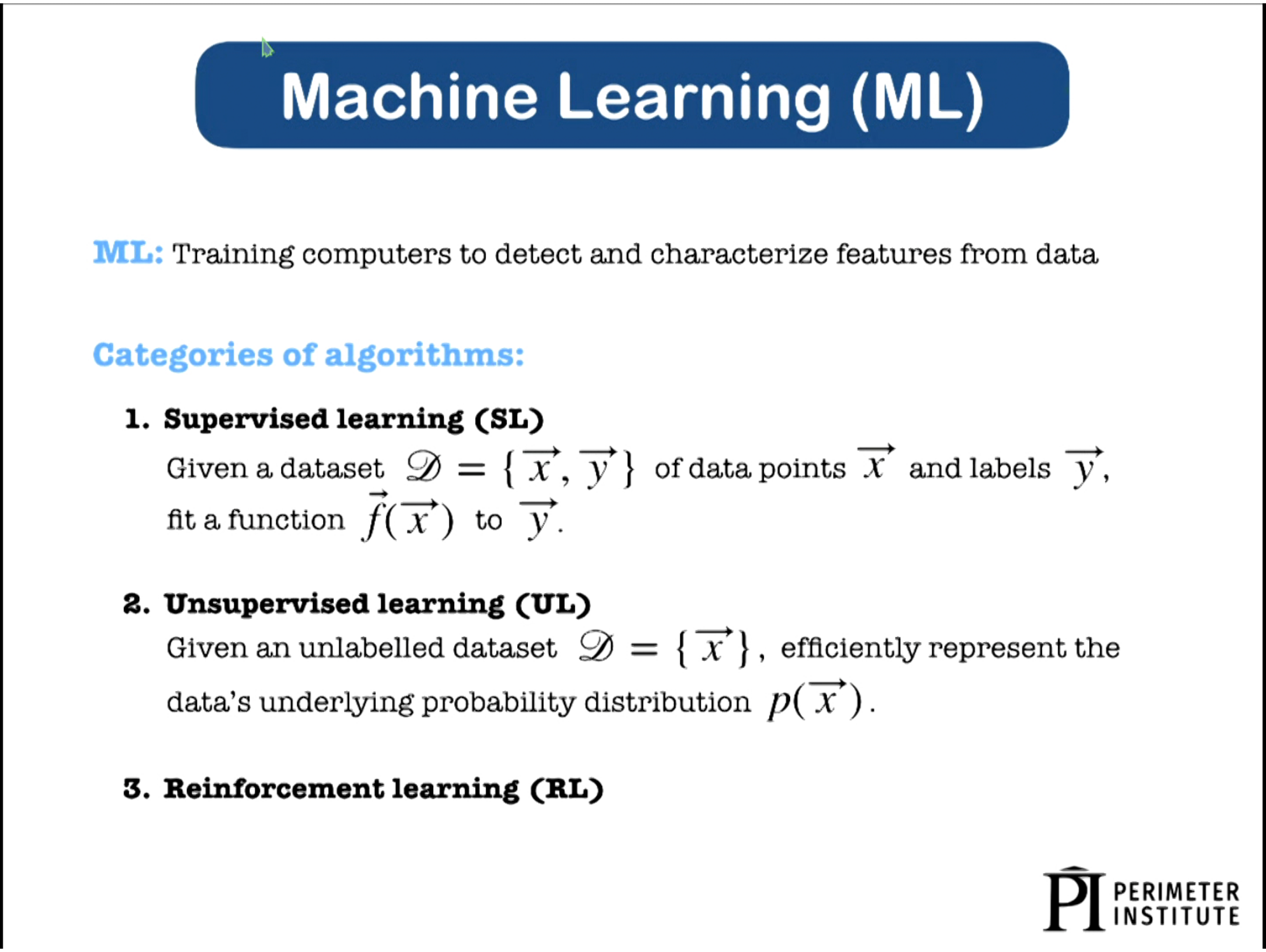
**Categories of algorithms:**

1. **Supervised learning (SL)**
   Given a dataset $\mathscr{D} = \{\vec{x}, \vec{y}\}$ of data points $\vec{x}$ and labels $\vec{y}$, fit a function $\vec{f}(\vec{x})$ to $\vec{y}$.

2. **Unsupervised learning (UL)**

3. **Reinforcement learning (RL)**

PERIMETER INSTITUTE



## Machine Learning (ML)

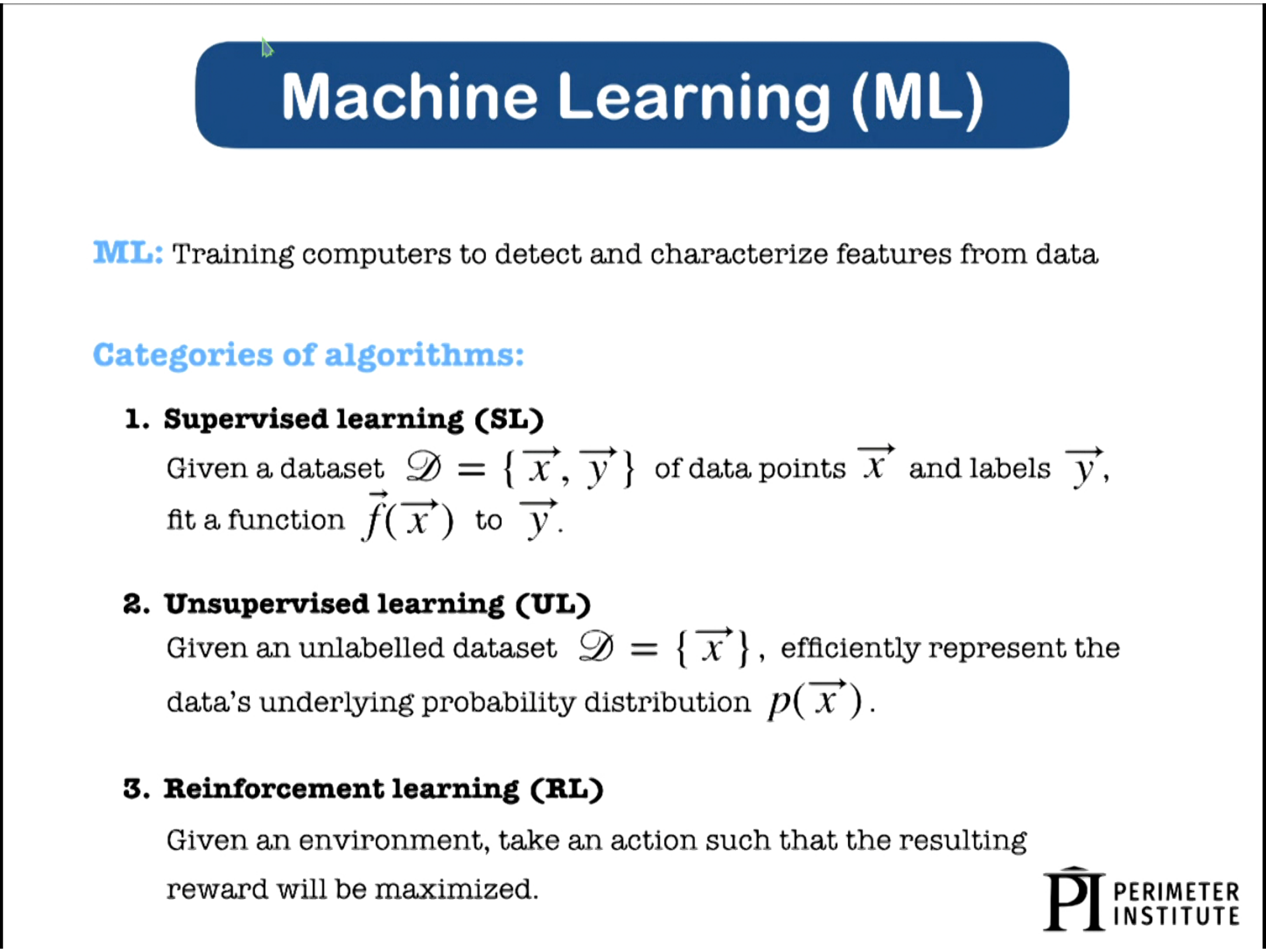
**ML:** Training computers to detect and characterize features from data

**Categories of algorithms:**

1. **Supervised learning (SL)**
   Given a dataset $\mathscr{D} = \{\vec{x}, \vec{y}\}$ of data points $\vec{x}$ and labels $\vec{y}$, fit a function $\vec{f}(\vec{x})$ to $\vec{y}$.

2. **Unsupervised learning (UL)**
   Given an unlabelled dataset $\mathscr{D} = \{\vec{x}\}$, efficiently represent the data's underlying probability distribution $p(\vec{x})$.

3. **Reinforcement learning (RL)**



## Machine Learning (ML)

**ML:** Training computers to detect and characterize features from data

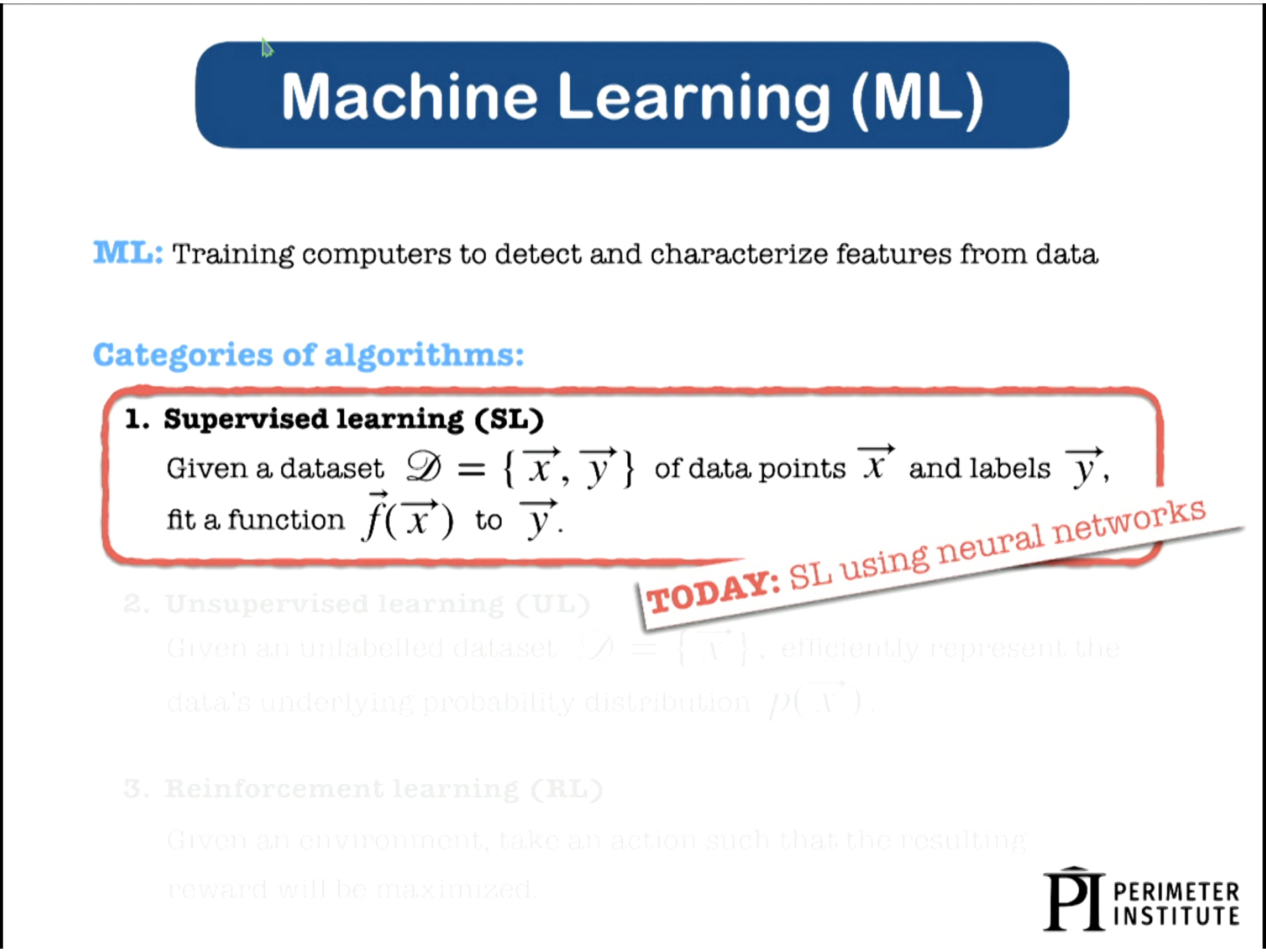
**Categories of algorithms:**

1. **Supervised learning (SL)**
   Given a dataset $\mathscr{D} = \{\vec{x}, \vec{y}\}$ of data points $\vec{x}$ and labels $\vec{y}$, fit a function $\vec{f}(\vec{x})$ to $\vec{y}$.

2. **Unsupervised learning (UL)**
   Given an unlabelled dataset $\mathscr{D} = \{\vec{x}\}$, efficiently represent the data's underlying probability distribution $p(\vec{x})$.

3. **Reinforcement learning (RL)**
   Given an environment, take an action such that the resulting reward will be maximized.

PERIMETER INSTITUTE



## Machine Learning (ML)

**ML:** Training computers to detect and characterize features from data

**Categories of algorithms:**

1. **Supervised learning (SL)**
   Given a dataset $\mathscr{D} = \{\vec{x}, \vec{y}\}$ of data points $\vec{x}$ and labels $\vec{y}$, fit a function $\vec{f}(\vec{x})$ to $\vec{y}$.

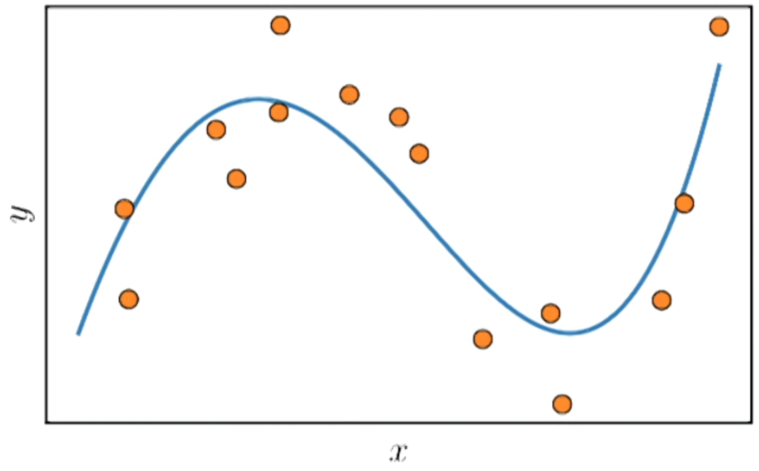
**TODAY:** SL using neural networks

2. **Unsupervised learning (UL)**
   Given an unlabelled dataset $\mathscr{D} = \{\vec{x}\}$, efficiently represent the data's underlying probability distribution $p(\vec{x})$.

3. **Reinforcement learning (RL)**
   Given an environment, take an action such that the resulting reward will be maximized.

PERIMETER INSTITUTE

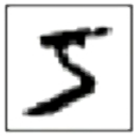
## Supervised Learning Example: 1D Regression

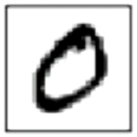


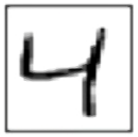
The data points $x$ and labels $y$ are both 1D coordinates. The goal is to find a function $f(x)$ (such as the blue curve) that describes the data.

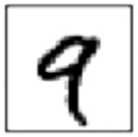
PERIMETER INSTITUTE

## Supervised Learning Example: Classifying handwritten digits

$\vec{x}_1 =$  $y_1 = 5$

$\vec{x}_2 =$  $y_2 = 0$

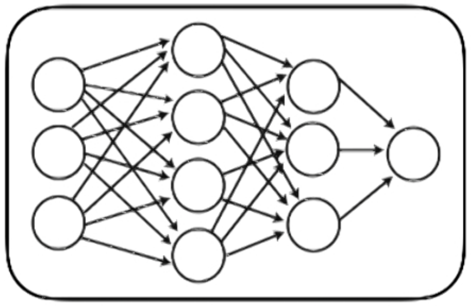
$\vec{x}_3 =$  $y_3 = 4$

$\vec{x}_4 =$  $y_4 = 9$

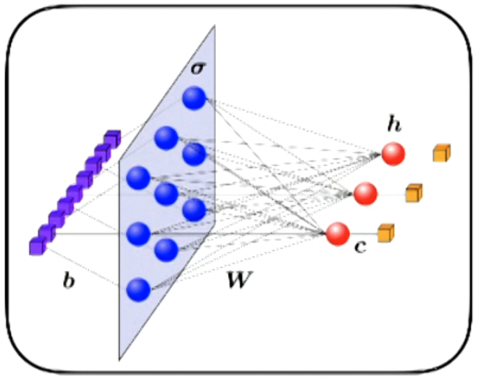
Data points taken from the MNIST database

PERIMETER INSTITUTE

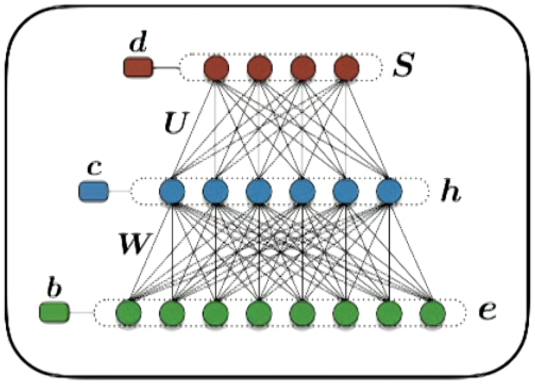
# Artificial neural networks



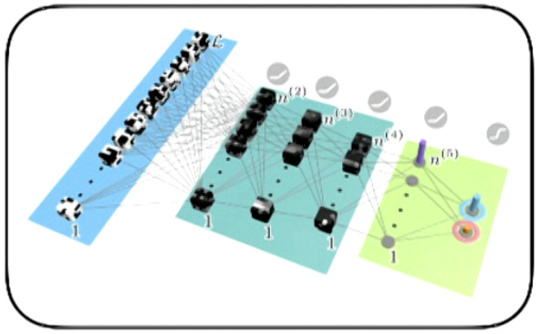
arXiv:1803.08823



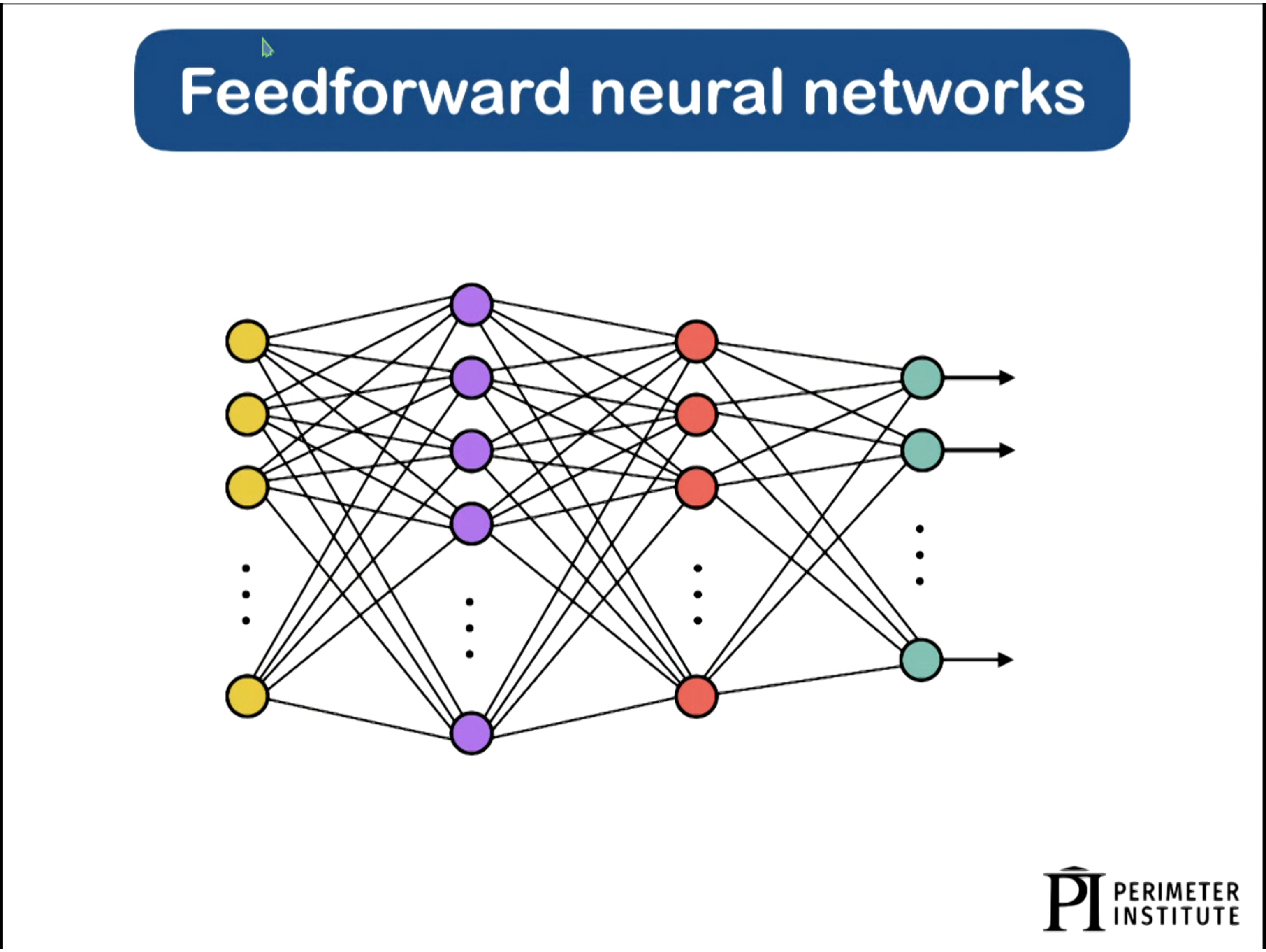
arXiv:1606.02718



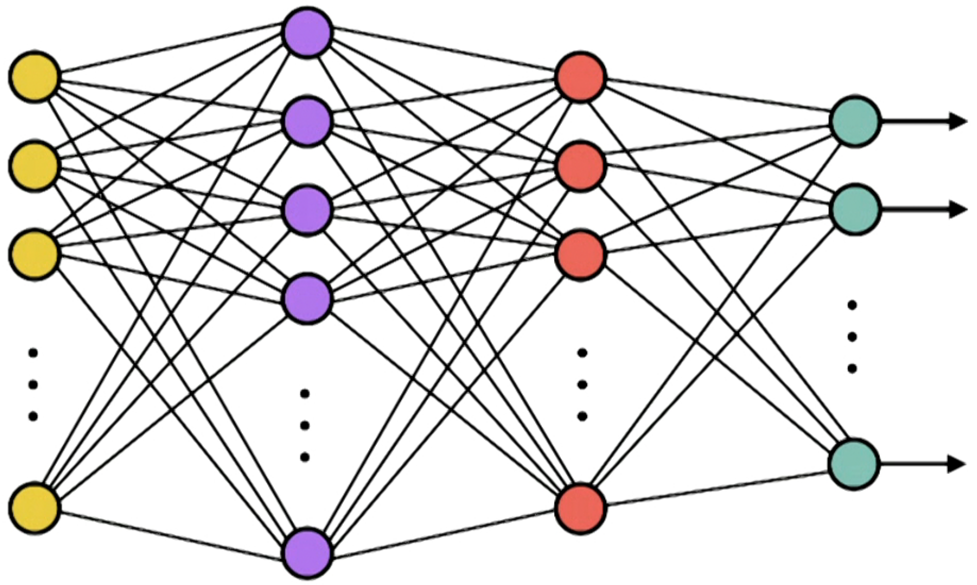
arXiv:1610.04238



arXiv:1609.02552

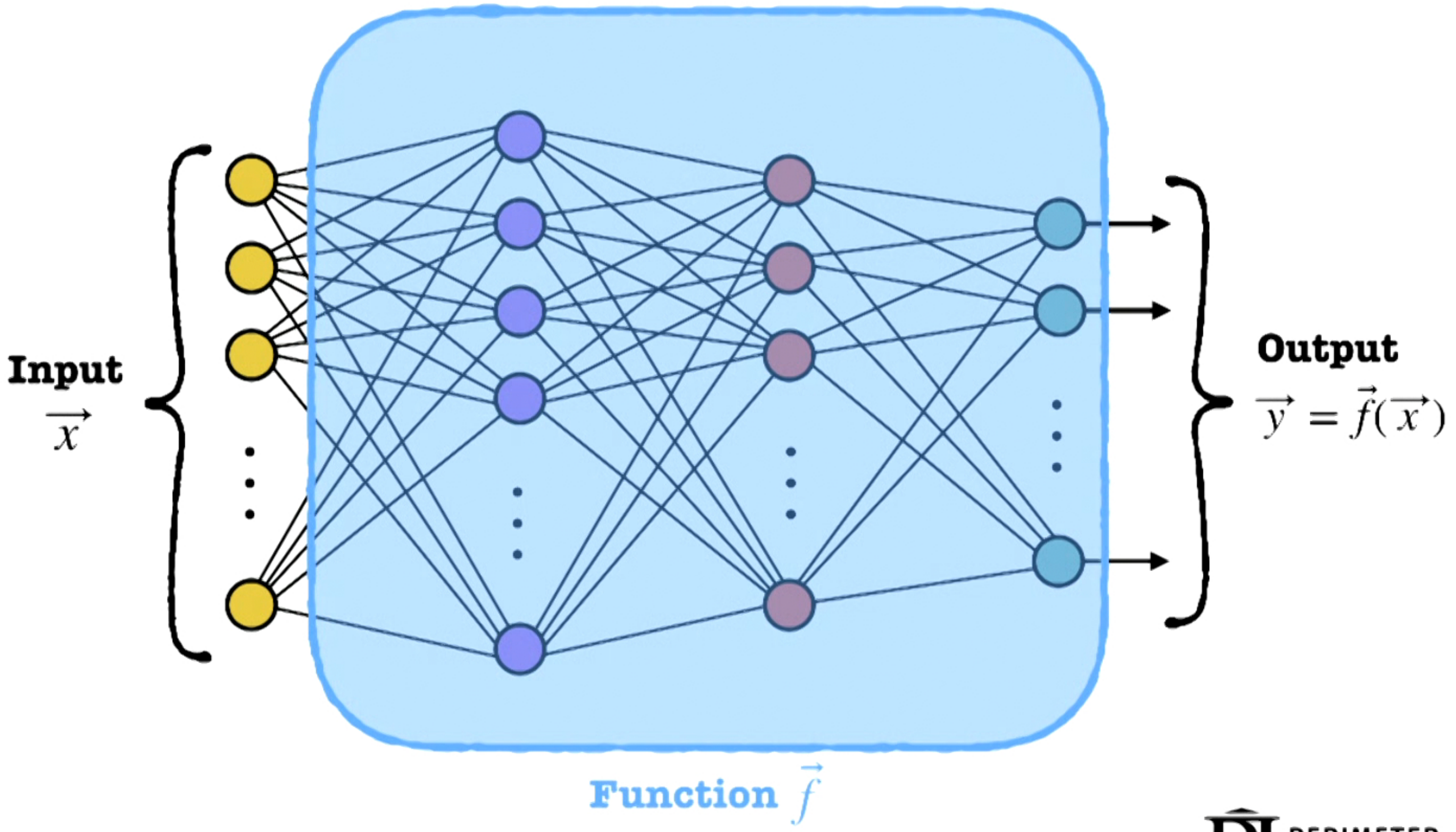


## Feedforward neural networks



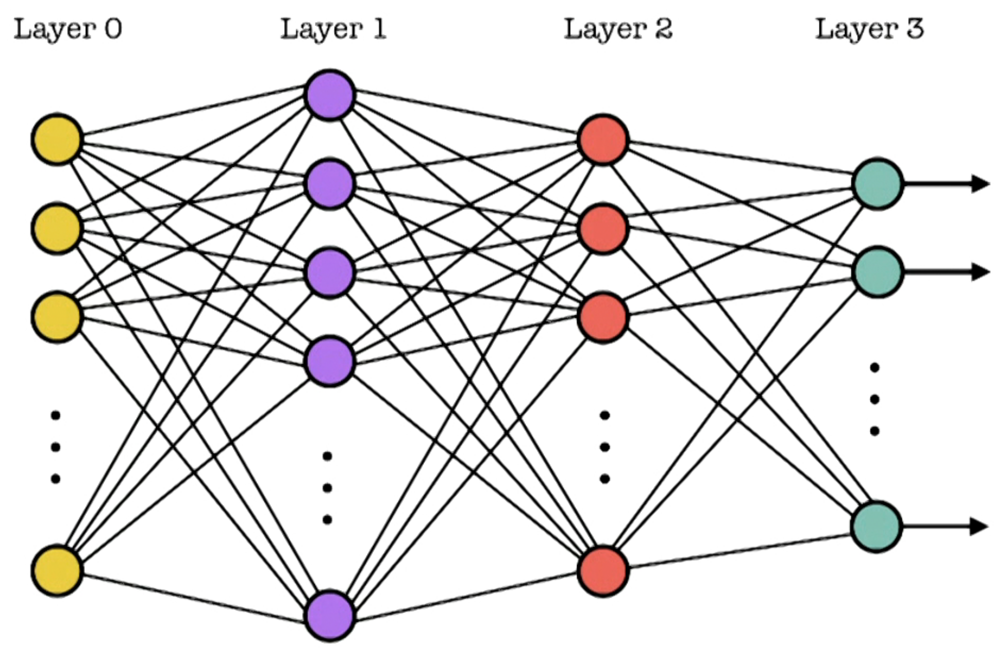
PERIMETER INSTITUTE

## Feedforward neural networks



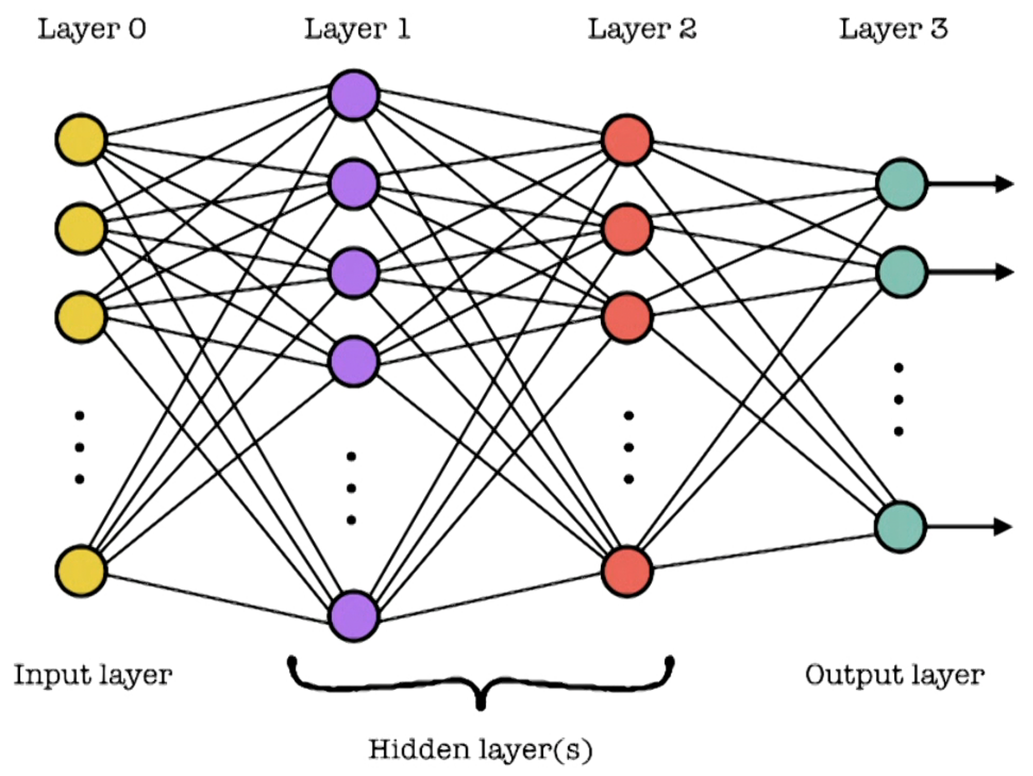
PERIMETER INSTITUTE

## Feedforward neural networks



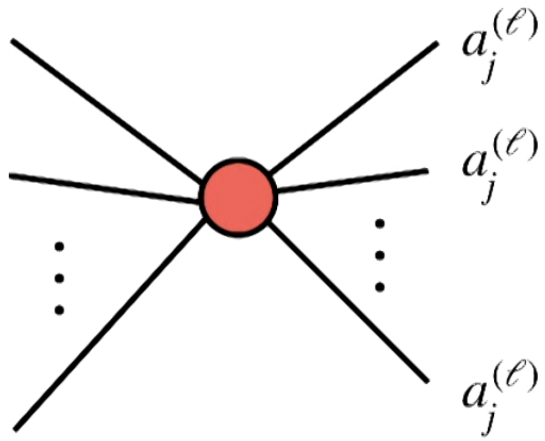
PERIMETER INSTITUTE

## Feedforward neural networks



## Neuron output

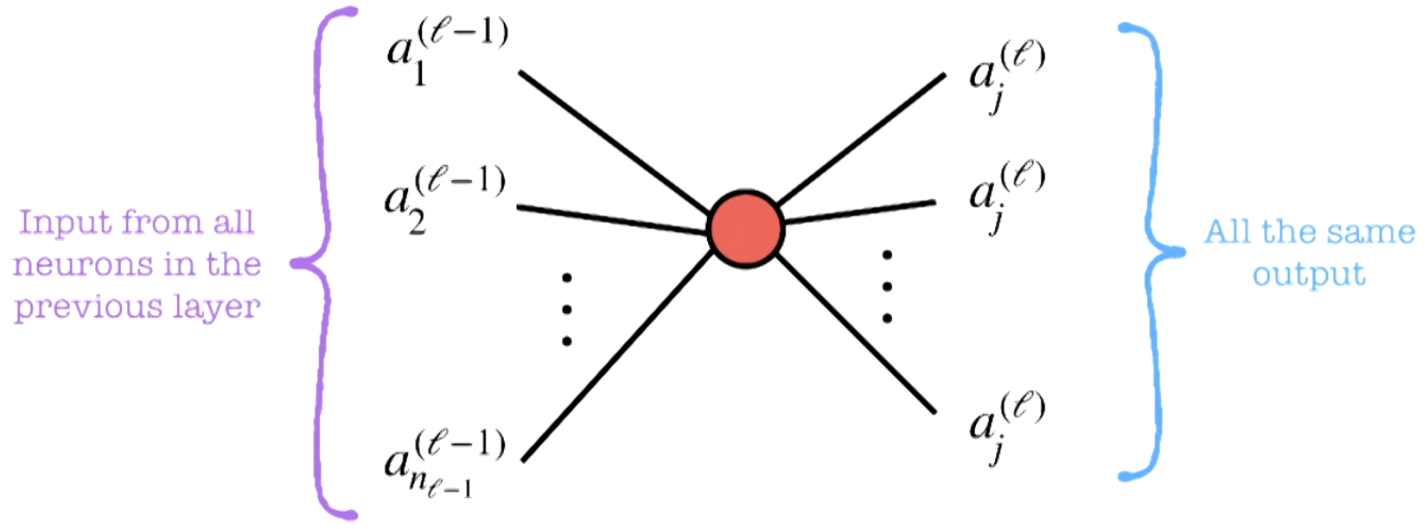
Let's zoom in on the $j^{\text{th}}$ neuron in layer $\ell > 0$

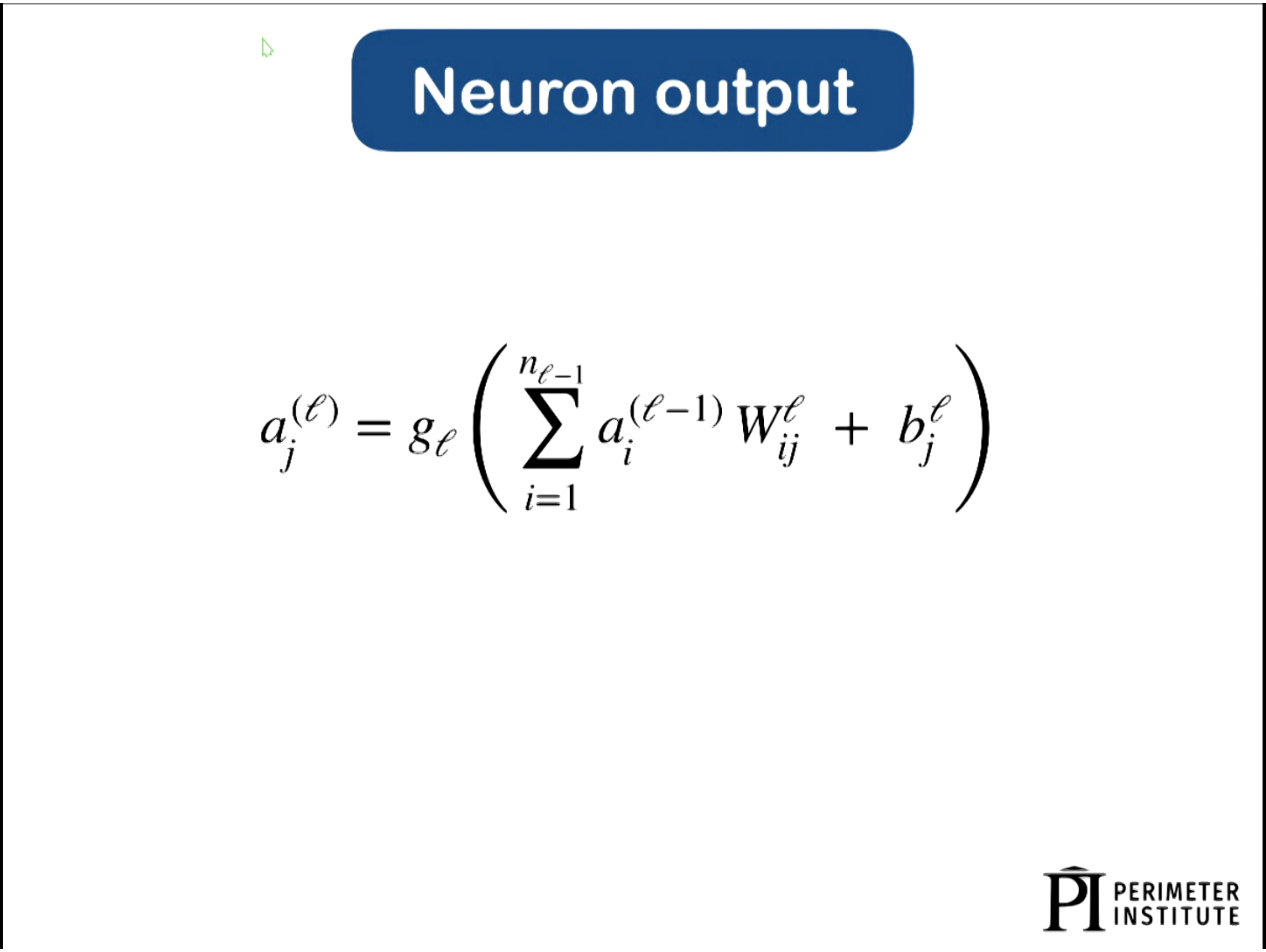


PERIMETER INSTITUTE

## Neuron output

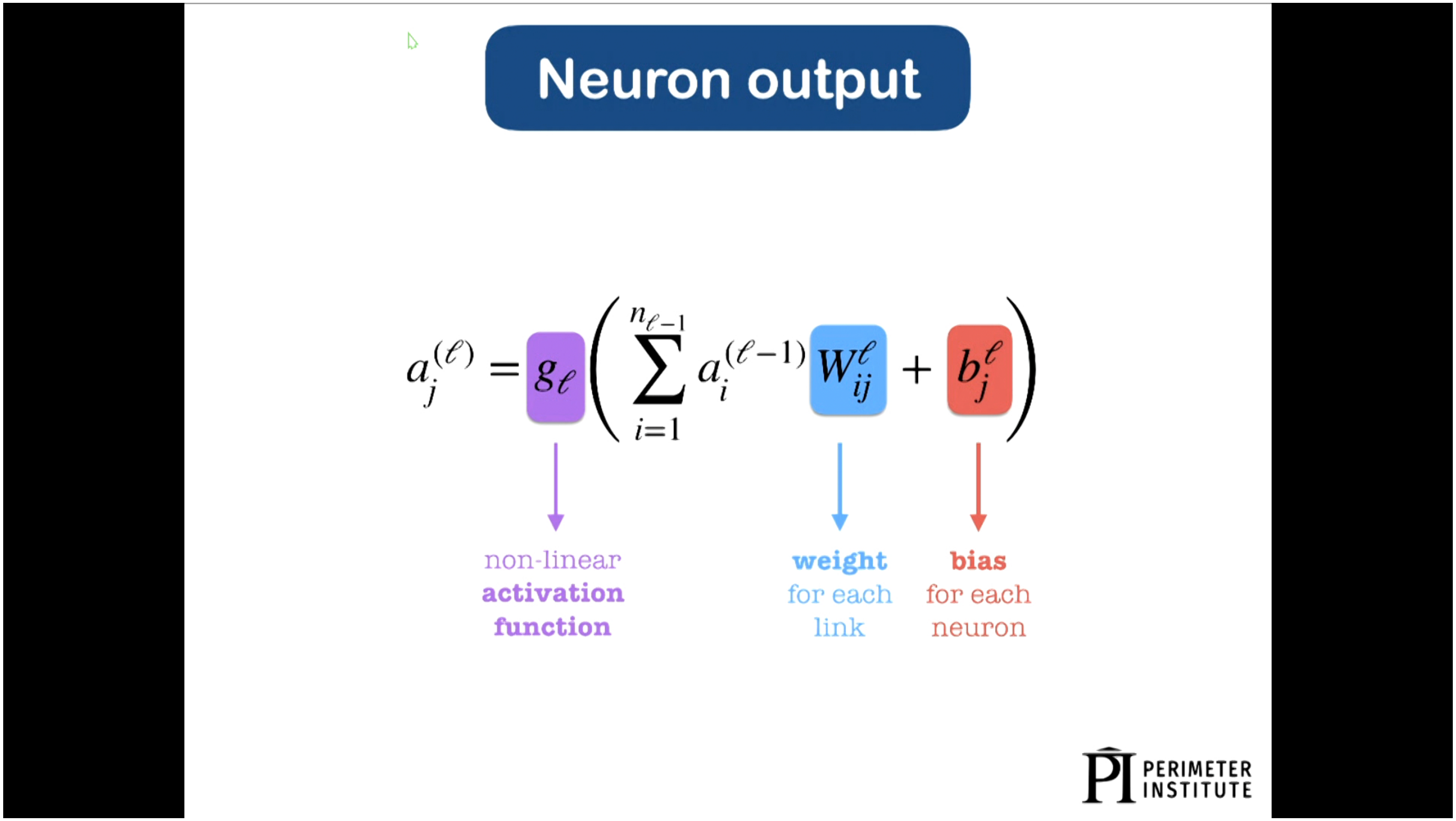
Let's zoom in on the $j^{\text{th}}$ neuron in layer $\ell > 0$





## Neuron output

$$a_j^{(\ell)} = g_\ell\left(\sum_{i=1}^{n_{\ell-1}} a_i^{(\ell-1)}\, W_{ij}^{\ell} \;+\; b_j^{\ell}\right)$$



## Neuron output

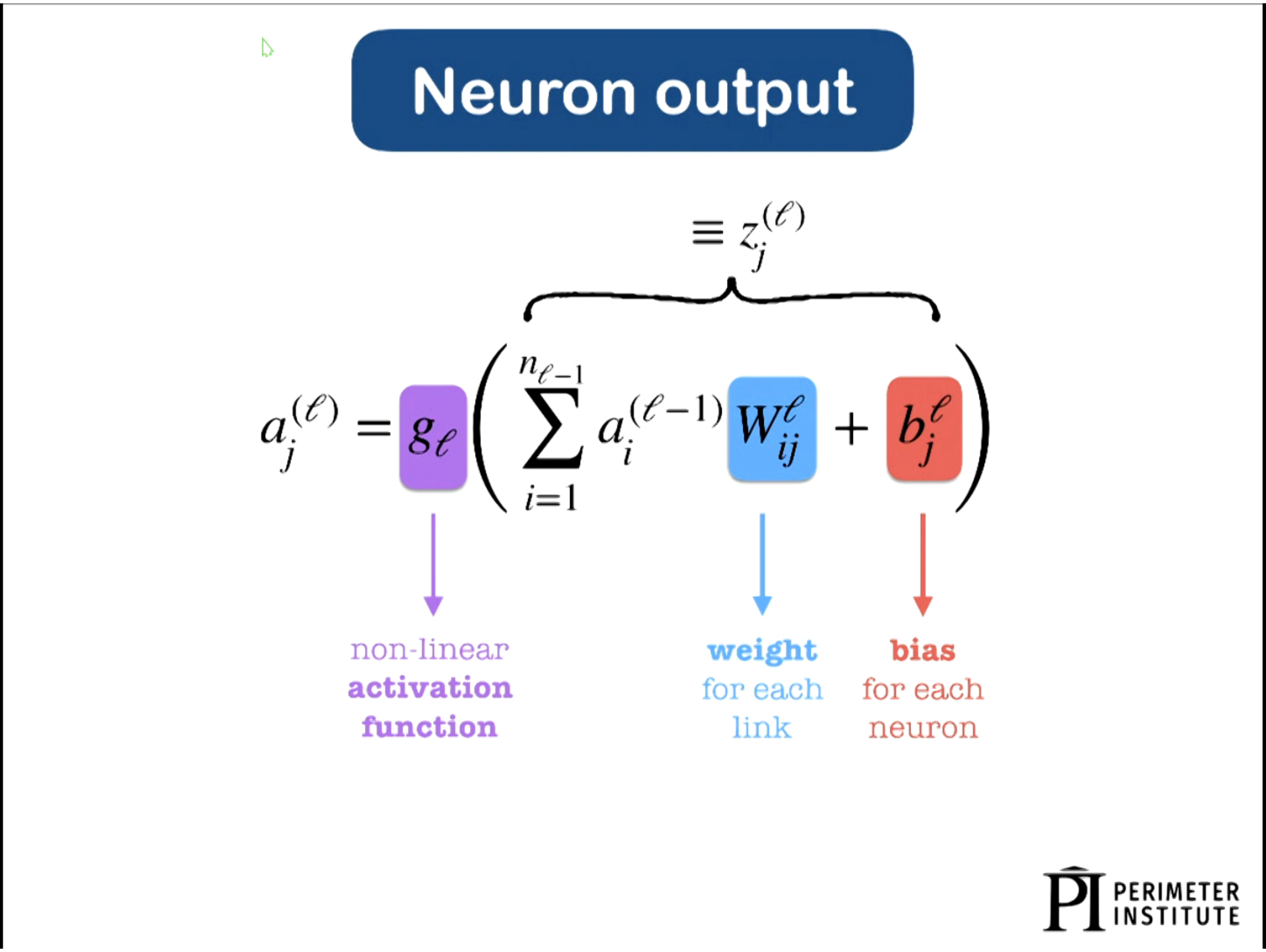
$$a_j^{(\ell)} = \boxed{g_\ell}\left(\sum_{i=1}^{n_{\ell-1}} a_i^{(\ell-1)} \boxed{W_{ij}^{\ell}} + \boxed{b_j^{\ell}}\right)$$

non-linear **activation function**

**weight** for each link

**bias** for each neuron

PERIMETER INSTITUTE



## Neuron output

$$a_j^{(\ell)} = \boxed{g_\ell}\left( \overbrace{\sum_{i=1}^{n_{\ell-1}} a_i^{(\ell-1)} \boxed{W_{ij}^{\ell}} + \boxed{b_j^{\ell}}}^{\equiv z_j^{(\ell)}} \right)$$

non-linear **activation function**

**weight** for each link

**bias** for each neuron

PERIMETER INSTITUTE

## Neuron output

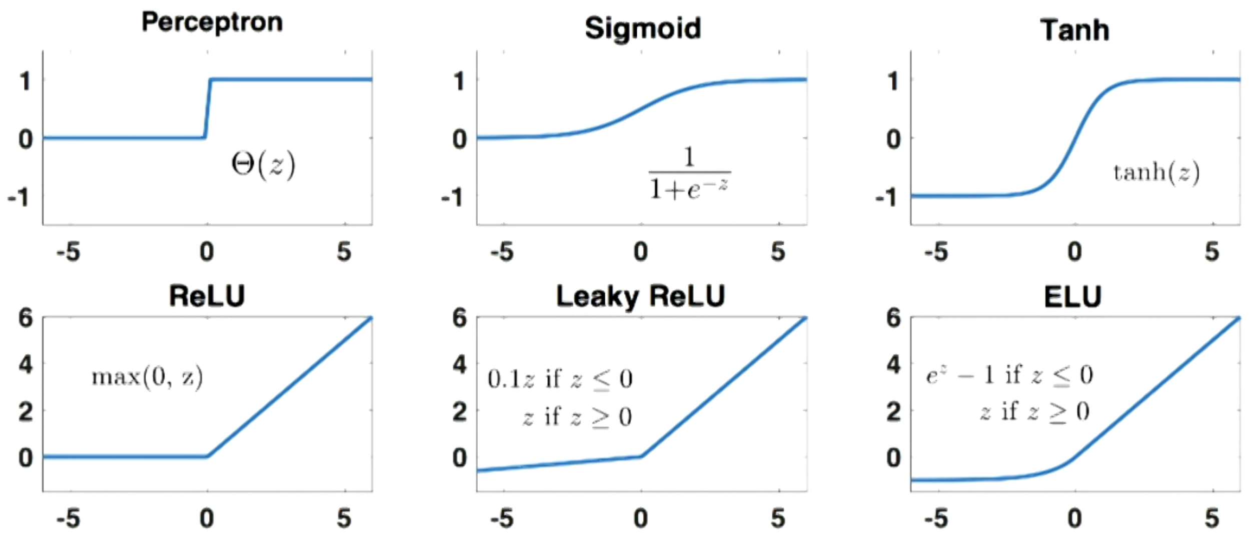
$$a_j^{(\ell)} = \underbrace{g_\ell}_{\substack{\text{non-linear}\\ \textbf{activation}\\ \textbf{function}}} \left( \overbrace{\sum_{i=1}^{n_{\ell-1}} a_i^{(\ell-1)} \underbrace{W_{ij}^{\ell}}_{\substack{\textbf{weight}\\ \text{for each}\\ \text{link}}} + \underbrace{b_j^{\ell}}_{\substack{\textbf{bias}\\ \text{for each}\\ \text{neuron}}}}^{\equiv z_j^{(\ell)}} \right)$$

The weights and biases are adjusted as the network **learns**.

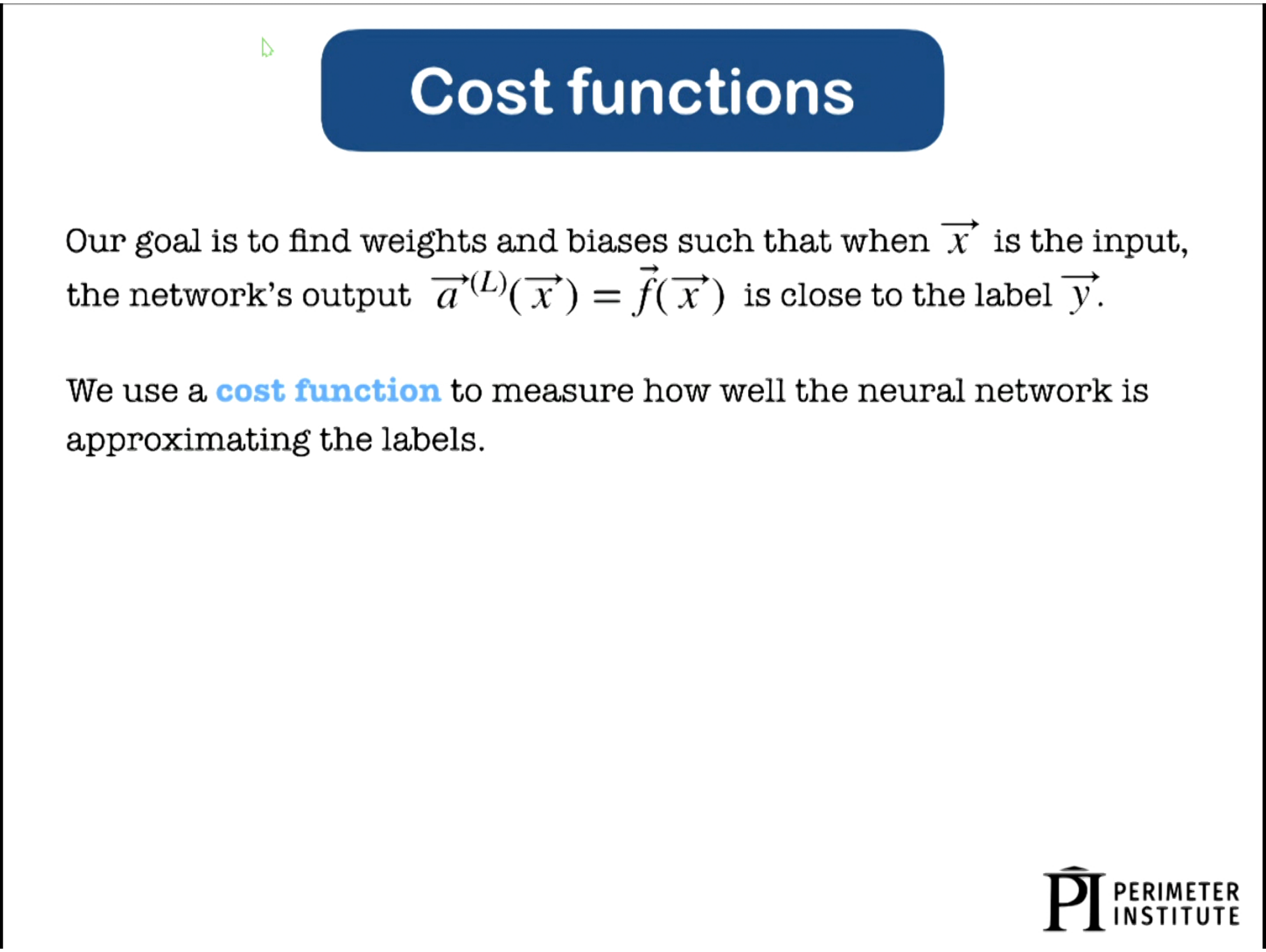
PI PERIMETER INSTITUTE

## Activation functions

We can choose various non-linear activation functions $g_\ell$, such as:



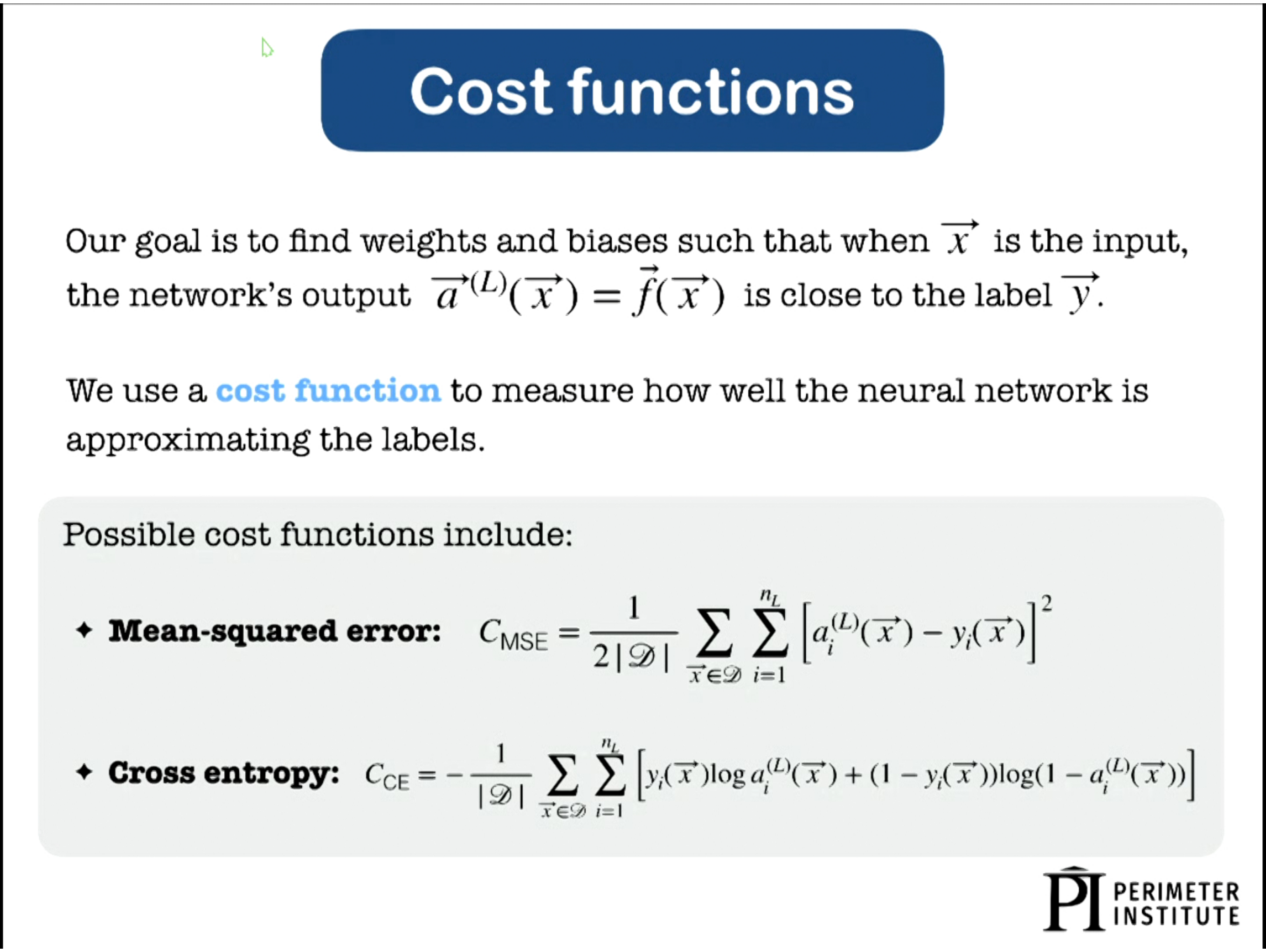
arXiv:1803.08823



## Cost functions

Our goal is to find weights and biases such that when $\overrightarrow{x}$ is the input, the network's output $\overrightarrow{a}^{(L)}(\overrightarrow{x}) = \vec{f}(\overrightarrow{x})$ is close to the label $\overrightarrow{y}$.

We use a **cost function** to measure how well the neural network is approximating the labels.



## Cost functions

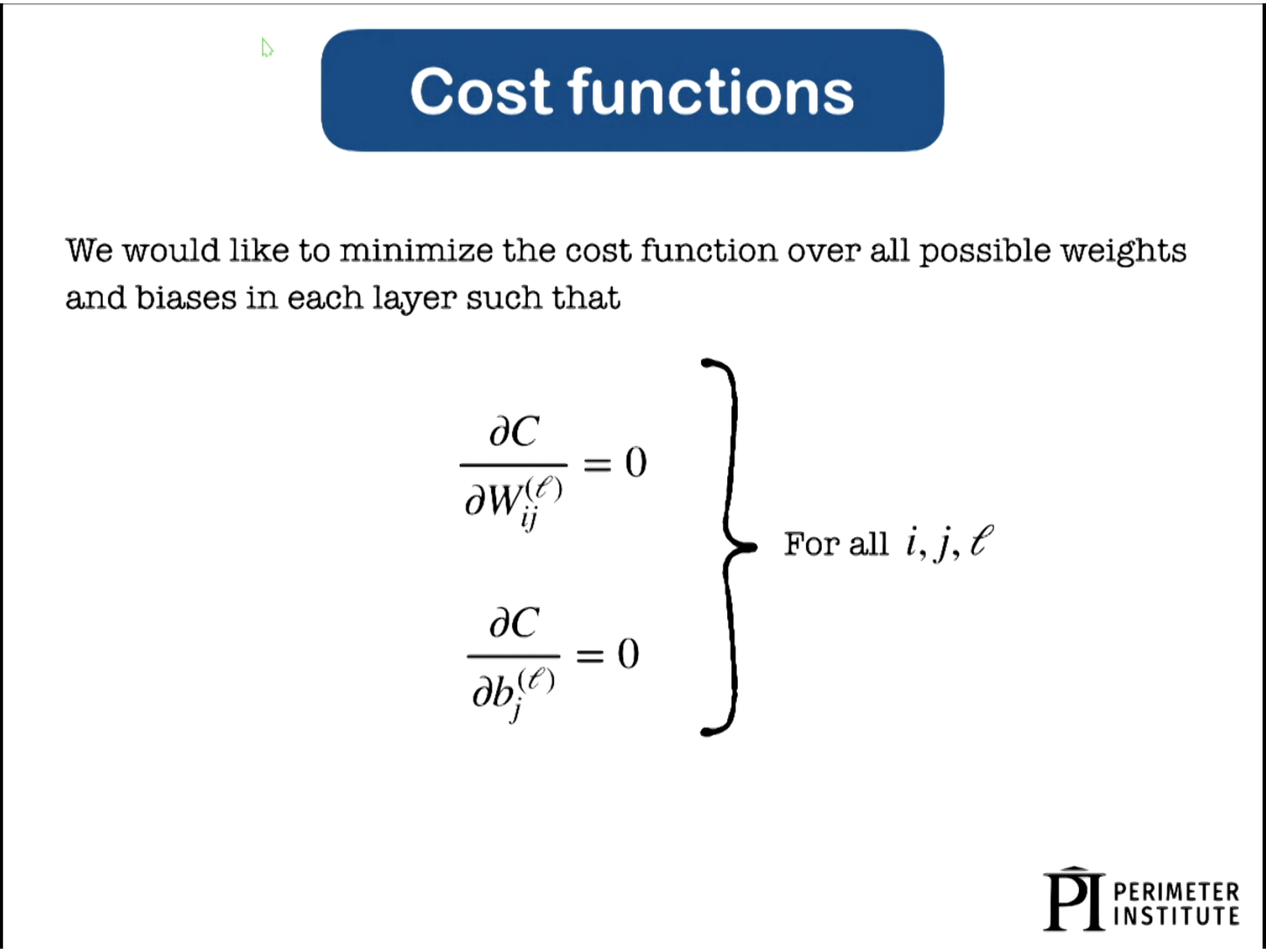
Our goal is to find weights and biases such that when $\vec{x}$ is the input, the network's output $\vec{a}^{(L)}(\vec{x}) = \vec{f}(\vec{x})$ is close to the label $\vec{y}$.

We use a **cost function** to measure how well the neural network is approximating the labels.

Possible cost functions include:

- **Mean-squared error:** $C_{\text{MSE}} = \frac{1}{2|\mathcal{D}|} \sum_{\vec{x} \in \mathcal{D}} \sum_{i=1}^{n_L} \left[ a_i^{(L)}(\vec{x}) - y_i(\vec{x}) \right]^2$
- **Cross entropy:** $C_{\text{CE}} = -\frac{1}{|\mathcal{D}|} \sum_{\vec{x} \in \mathcal{D}} \sum_{i=1}^{n_L} \left[ y_i(\vec{x}) \log a_i^{(L)}(\vec{x}) + (1 - y_i(\vec{x})) \log(1 - a_i^{(L)}(\vec{x})) \right]$

PERIMETER INSTITUTE

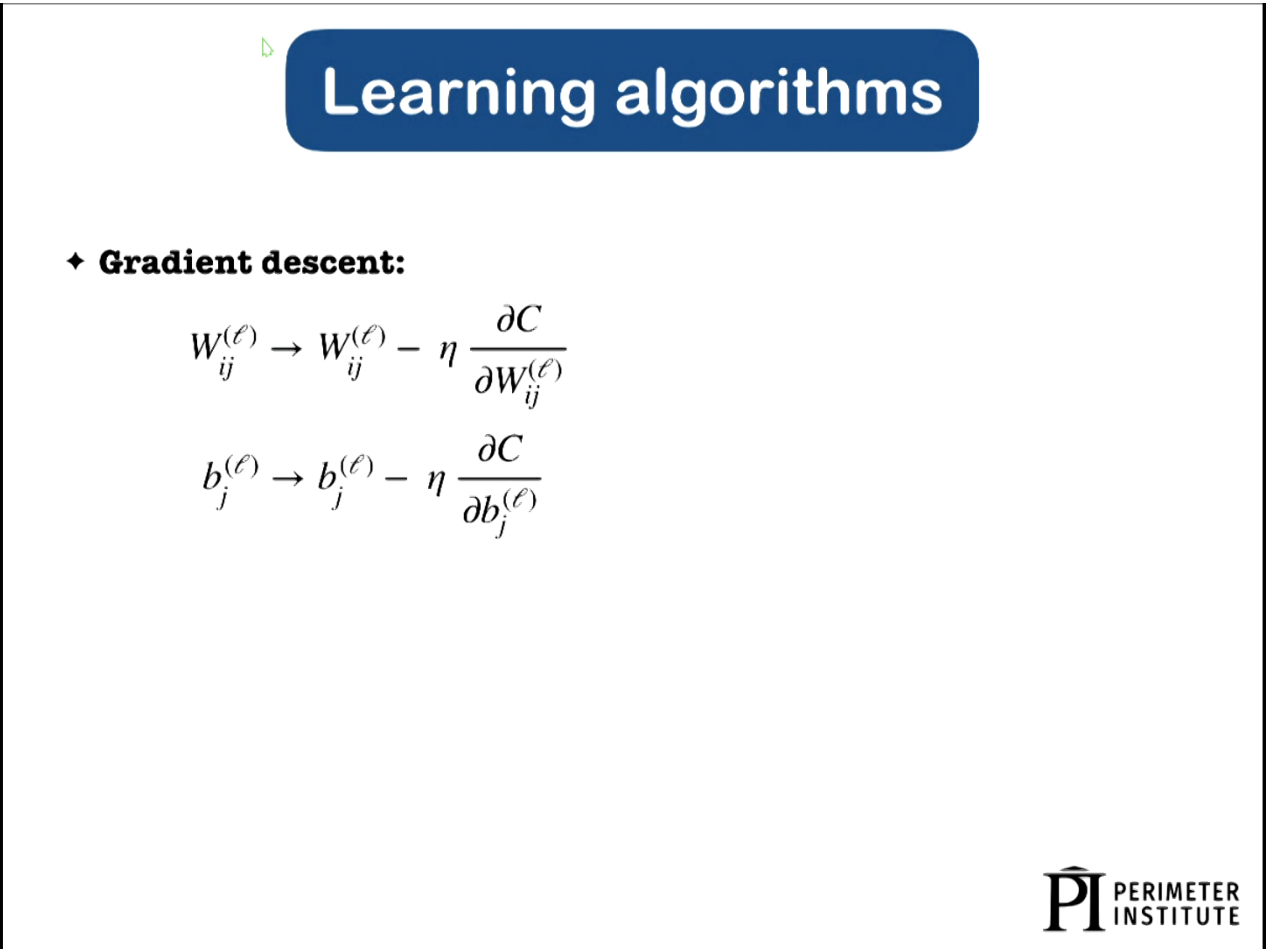


## Cost functions

We would like to minimize the cost function over all possible weights and biases in each layer such that

$$\left.\begin{aligned} \frac{\partial C}{\partial W_{ij}^{(\ell)}} &= 0 \\[2ex] \frac{\partial C}{\partial b_j^{(\ell)}} &= 0 \end{aligned}\right\} \text{For all } i, j, \ell$$

PERIMETER INSTITUTE



## Learning algorithms

- **Gradient descent:**

$$W_{ij}^{(\ell)} \to W_{ij}^{(\ell)} - \eta \frac{\partial C}{\partial W_{ij}^{(\ell)}}$$

$$b_j^{(\ell)} \to b_j^{(\ell)} - \eta \frac{\partial C}{\partial b_j^{(\ell)}}$$

PERIMETER INSTITUTE

## Learning algorithms

✦ **Gradient descent:**

$$W_{ij}^{(\ell)} \to W_{ij}^{(\ell)} - \eta \frac{\partial C}{\partial W_{ij}^{(\ell)}}$$

$$b_j^{(\ell)} \to b_j^{(\ell)} - \eta \frac{\partial C}{\partial b_j^{(\ell)}}$$

$\eta$ : **Learning rate**

PERIMETER INSTITUTE

## Learning algorithms

- **Gradient descent:**

$$W_{ij}^{(\ell)} \to W_{ij}^{(\ell)} - \eta \frac{\partial C}{\partial W_{ij}^{(\ell)}}$$

$$b_{j}^{(\ell)} \to b_{j}^{(\ell)} - \eta \frac{\partial C}{\partial b_{j}^{(\ell)}}$$

$\eta$ : **Learning rate**

- **Stochastic gradient descent**
- **RMSProp**
- **Adam optimizer**

$\vdots$

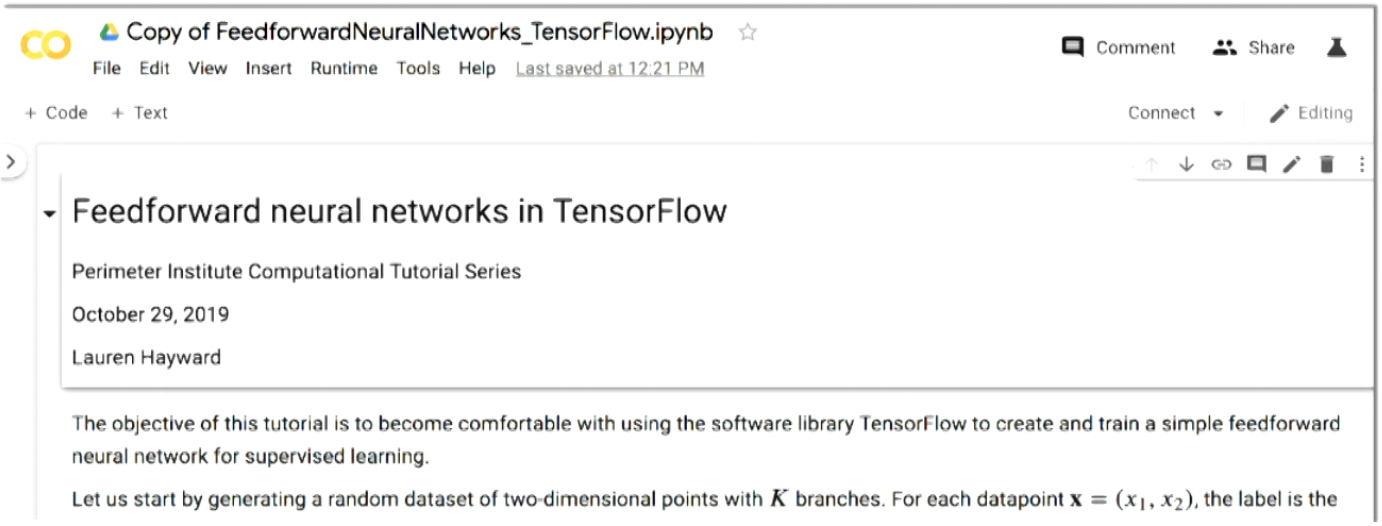
PERIMETER INSTITUTE

# Feedforward Neural Networks in TensorFlow

**Go to:**

https://drive.google.com/file/d/17CihZKb04U2TRwDXQwwmMDYODFXMfbCf/view?usp=sharing

- ✦ Open with Google Colaboratory
- ✦ Choose 'Open in Playground'
- ✦ Choose the option 'Copy to Drive'



Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File Edit View Insert Runtime Tools Help Last saved at 12:21 PM
Comment
Share
+ Code + Text
Connect
Editing

## Feedforward neural networks in TensorFlow

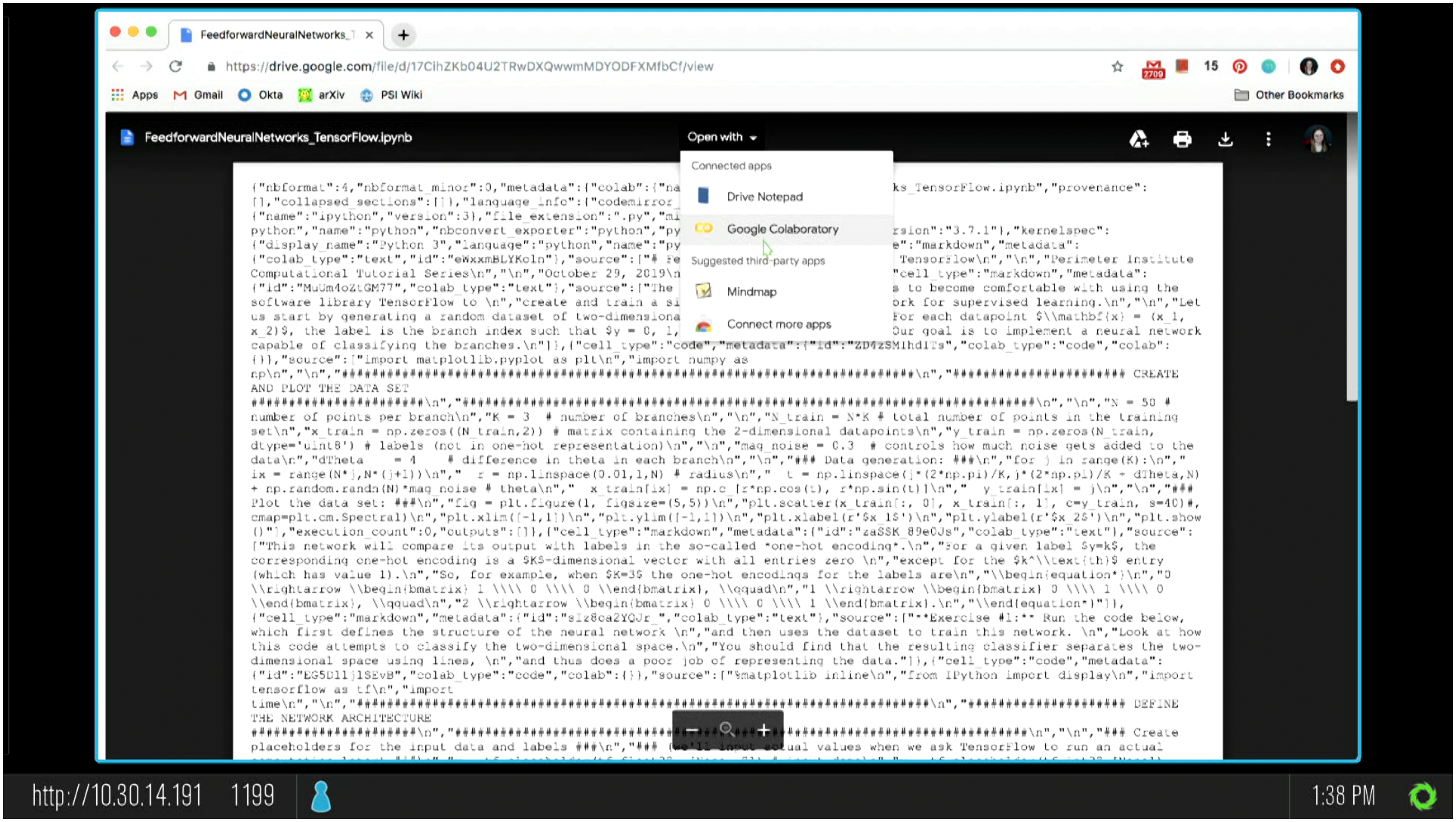
Perimeter Institute Computational Tutorial Series

October 29, 2019

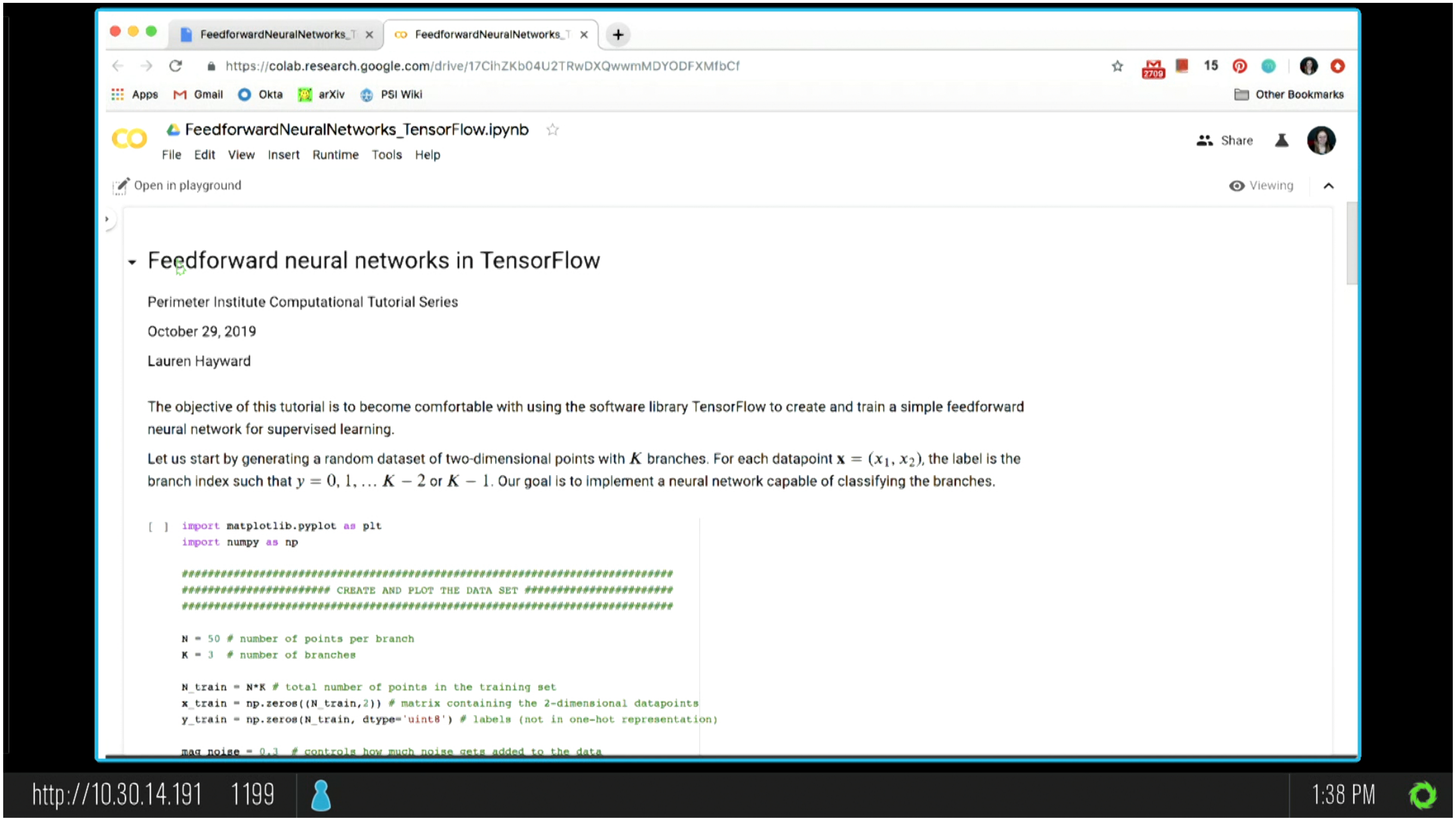
Lauren Hayward

The objective of this tutorial is to become comfortable with using the software library TensorFlow to create and train a simple feedforward neural network for supervised learning.

Let us start by generating a random dataset of two-dimensional points with $K$ branches. For each datapoint $\mathbf{x} = (x_1, x_2)$, the label is the


FeedforwardNeuralNetworks_T
https://drive.google.com/file/d/17CihZKb04U2TRwDXQwwmMDYODFXMfbCf/view
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
FeedforwardNeuralNetworks_TensorFlow.ipynb
Open with
Connected apps
Drive Notepad
Google Colaboratory
Suggested third-party apps
Mindmap
Connect more apps
http://10.30.14.191
1199
1:38 PM

```
{"nbformat":4,"nbformat_minor":0,"metadata":{"colab":{"na                              ks_TensorFlow.ipynb","provenance":
[],"collapsed_sections":[]},"language_info":{"codemirror_
{"name":"ipython","version":3},"file_extension":".py","mi
python","name":"python","nbconvert_exporter":"python","py                              rsion":"3.7.1"},"kernelspec":
{"display_name":"Python 3","language":"python","name":"py                              e":"markdown","metadata":
{"colab_type":"text","id":"eWxxmBLYKoln"},"source":["# Fe                               TensorFlow\n","\n","Perimeter Institute
Computational Tutorial Series\n","\n","October 29, 2019\n                              "cell_type":"markdown","metadata":
{"id":"MuUm4oZtGM77","colab_type":"text"},"source":["The                               s to become comfortable with using the
software library TensorFlow to \n","create and train a si                              ork for supervised learning.\n","\n","Let
us start by generating a random dataset of two-dimensiona                              For each datapoint $\\mathbf{x} = (x_1,
x_2)$, the label is the branch index such that $y = 0, 1,                              Our goal is to implement a neural network
capable of classifying the branches.\n"]},{"cell_type":"code","metadata":{"id":"ZD4zSMIhdITs","colab_type":"code","colab":
{}},"source":["import matplotlib.pyplot as plt\n","import numpy as
np\n","\n","######################################################\n","###################### CREATE
AND PLOT THE DATA SET
######################\n","#######################################################\n","\n","N = 50 #
number of points per branch\n","K = 3  # number of branches\n","\n","N_train = N*K # total number of points in the training
set\n","x_train = np.zeros((N_train,2)) # matrix containing the 2-dimensional datapoints\n","y_train = np.zeros(N_train,
dtype='uint8') # labels (not in one-hot representation)\n","\n","mag_noise = 0.3  # controls how much noise gets added to the
data\n","dTheta    = 4    # difference in theta in each branch\n","\n","### Data generation: ###\n","for j in range(K):\n","
ix = range(N*j,N*(j+1))\n","  r = np.linspace(0.01,1,N) # radius\n","  t = np.linspace(j*(2*np.pi)/K,j*(2*np.pi)/K + dTheta,N)
+ np.random.randn(N)*mag_noise # theta\n","  x_train[ix] = np.c_[r*np.cos(t), r*np.sin(t)]\n","  y_train[ix] = j\n","\n","###
Plot the data set: ###\n","fig = plt.figure(1, figsize=(5,5))\n","plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train, s=40)#,
cmap=plt.cm.Spectral)\n","plt.xlim([-1,1])\n","plt.ylim([-1,1])\n","plt.xlabel(r'$x_1$')\n","plt.ylabel(r'$x_2$')\n","plt.show
()"],"execution_count":0,"outputs":[]},{"cell_type":"markdown","metadata":{"id":"zaSSK_89e0Js","colab_type":"text"},"source":
["This network will compare its output with labels in the so-called *one-hot encoding*.\n","For a given label $y=k$, the
corresponding one-hot encoding is a $K$-dimensional vector with all entries zero \n","except for the $k^\\text{th}$ entry
(which has value 1).\n","So, for example, when $K=3$ the one-hot encodings for the labels are\n","\\begin{equation*}\n","0
\\rightarrow \\begin{bmatrix} 1 \\\\ 0 \\\\ 0 \\end{bmatrix}, \\qquad\n","1 \\rightarrow \\begin{bmatrix} 0 \\\\ 1 \\\\ 0
\\end{bmatrix}, \\qquad\n","2 \\rightarrow \\begin{bmatrix} 0 \\\\ 0 \\\\ 1 \\end{bmatrix}.\n","\\end{equation*}"]},
{"cell_type":"markdown","metadata":{"id":"sIz8ca2YQJr_","colab_type":"text"},"source":["**Exercise #1:** Run the code below,
which first defines the structure of the neural network \n","and then uses the dataset to train this network. \n","Look at how
this code attempts to classify the two-dimensional space.\n","You should find that the resulting classifier separates the two-
dimensional space using lines, \n","and thus does a poor job of representing the data."]},{"cell_type":"code","metadata":
{"id":"EG5Dl1j1SEvB","colab_type":"code","colab":{}},"source":["%matplotlib inline\n","from IPython import display\n","import
tensorflow as tf\n","import
time\n","\n","#######################################################\n","###################### DEFINE
THE NETWORK ARCHITECTURE
######################\n","#######################################################\n","\n","### Create
placeholders for the input data and labels ###\n","### (we'll input actual values when we ask TensorFlow to run an actual
```


FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
https://colab.research.google.com/drive/17CihZKb04U2TRwDXQwwmMDYODFXMfbCf
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
FeedforwardNeuralNetworks_TensorFlow.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
Share
Open in playground
Viewing
http://10.30.14.191
1199
1:38 PM

## Feedforward neural networks in TensorFlow

Perimeter Institute Computational Tutorial Series

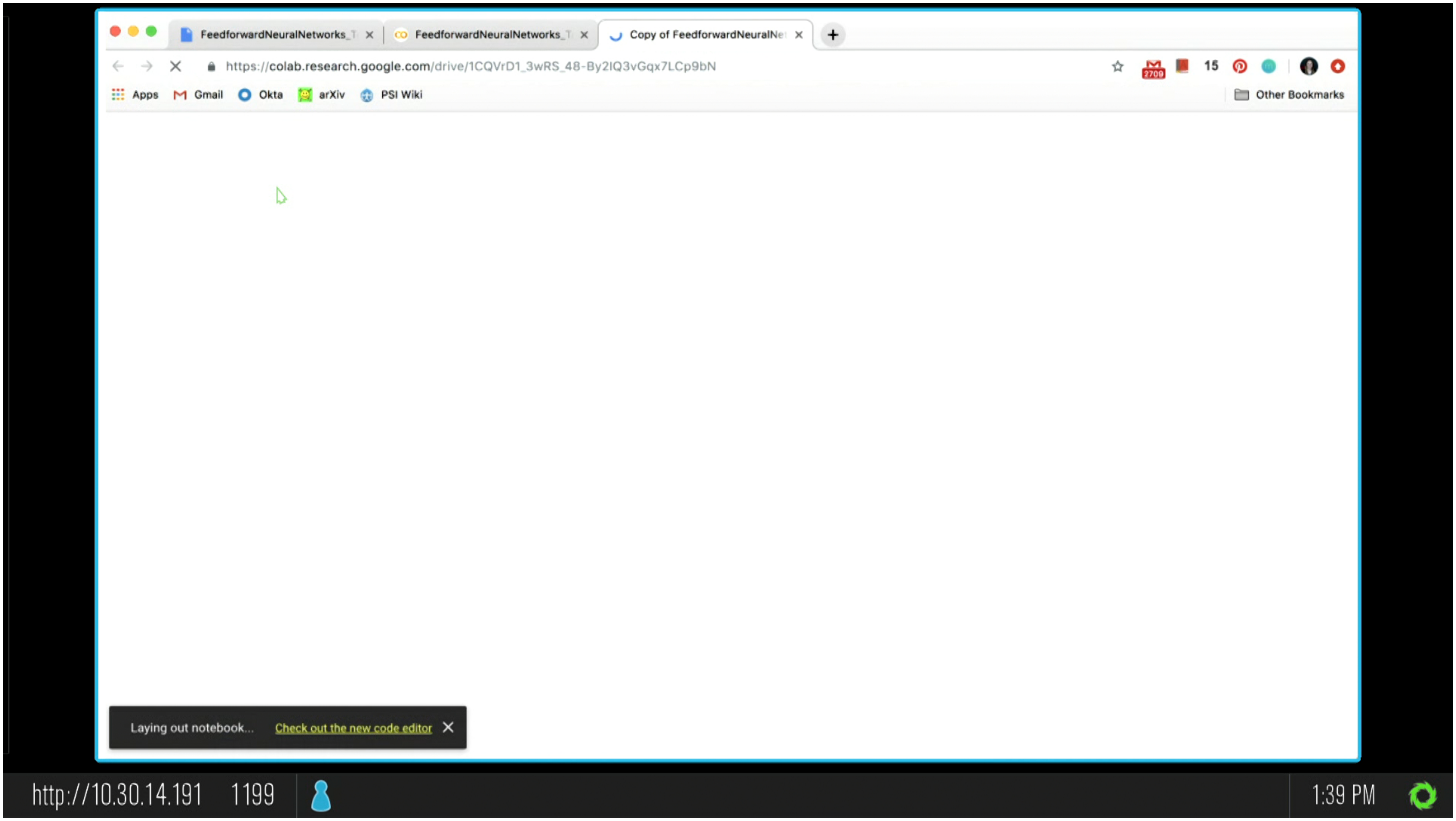
October 29, 2019

Lauren Hayward

The objective of this tutorial is to become comfortable with using the software library TensorFlow to create and train a simple feedforward neural network for supervised learning.

Let us start by generating a random dataset of two-dimensional points with $K$ branches. For each datapoint $\mathbf{x} = (x_1, x_2)$, the label is the branch index such that $y = 0, 1, \ldots K-2$ or $K-1$. Our goal is to implement a neural network capable of classifying the branches.

```
[ ] import matplotlib.pyplot as plt
    import numpy as np

    ##############################################################################
    ####################### CREATE AND PLOT THE DATA SET #######################
    ##############################################################################

    N = 50 # number of points per branch
    K = 3  # number of branches

    N_train = N*K # total number of points in the training set
    x_train = np.zeros((N_train,2)) # matrix containing the 2-dimensional datapoints
    y_train = np.zeros(N_train, dtype='uint8') # labels (not in one-hot representation)

    mag_noise = 0.3  # controls how much noise gets added to the data
```



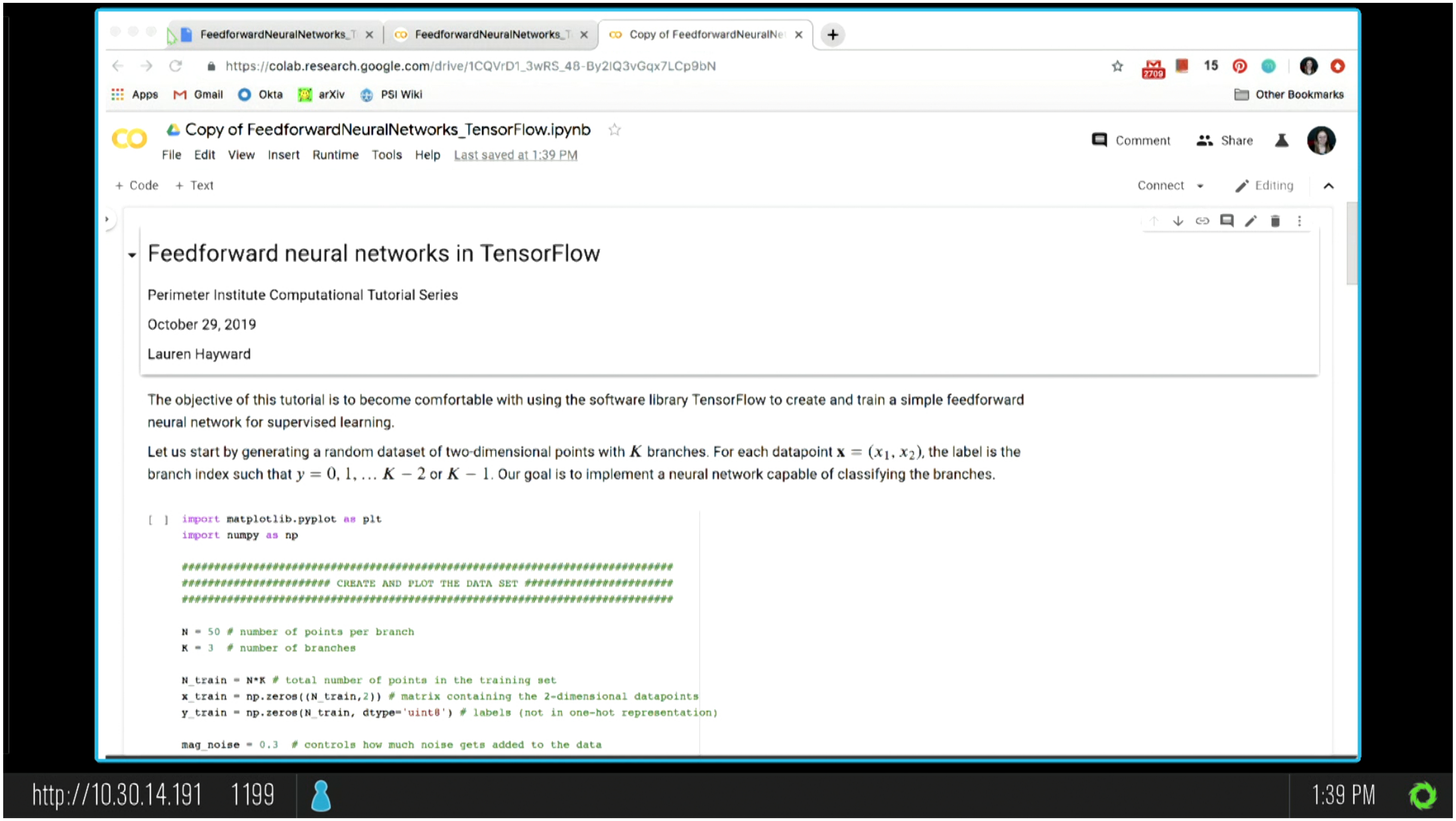
FeedforwardNeuralNetworks_T × FeedforwardNeuralNetworks_T × Copy of FeedforwardNeuralNe ×

https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2IQ3vGqx7LCp9bN

Apps Gmail Okta arXiv PSI Wiki Other Bookmarks

Laying out notebook... Check out the new code editor

http://10.30.14.191 1199 1:39 PM


FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2IQ3vGqx7LCp9bN
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
Last saved at 1:39 PM
Comment
Share
+ Code
+ Text
Connect
Editing
http://10.30.14.191
1199
1:39 PM

## Feedforward neural networks in TensorFlow

Perimeter Institute Computational Tutorial Series

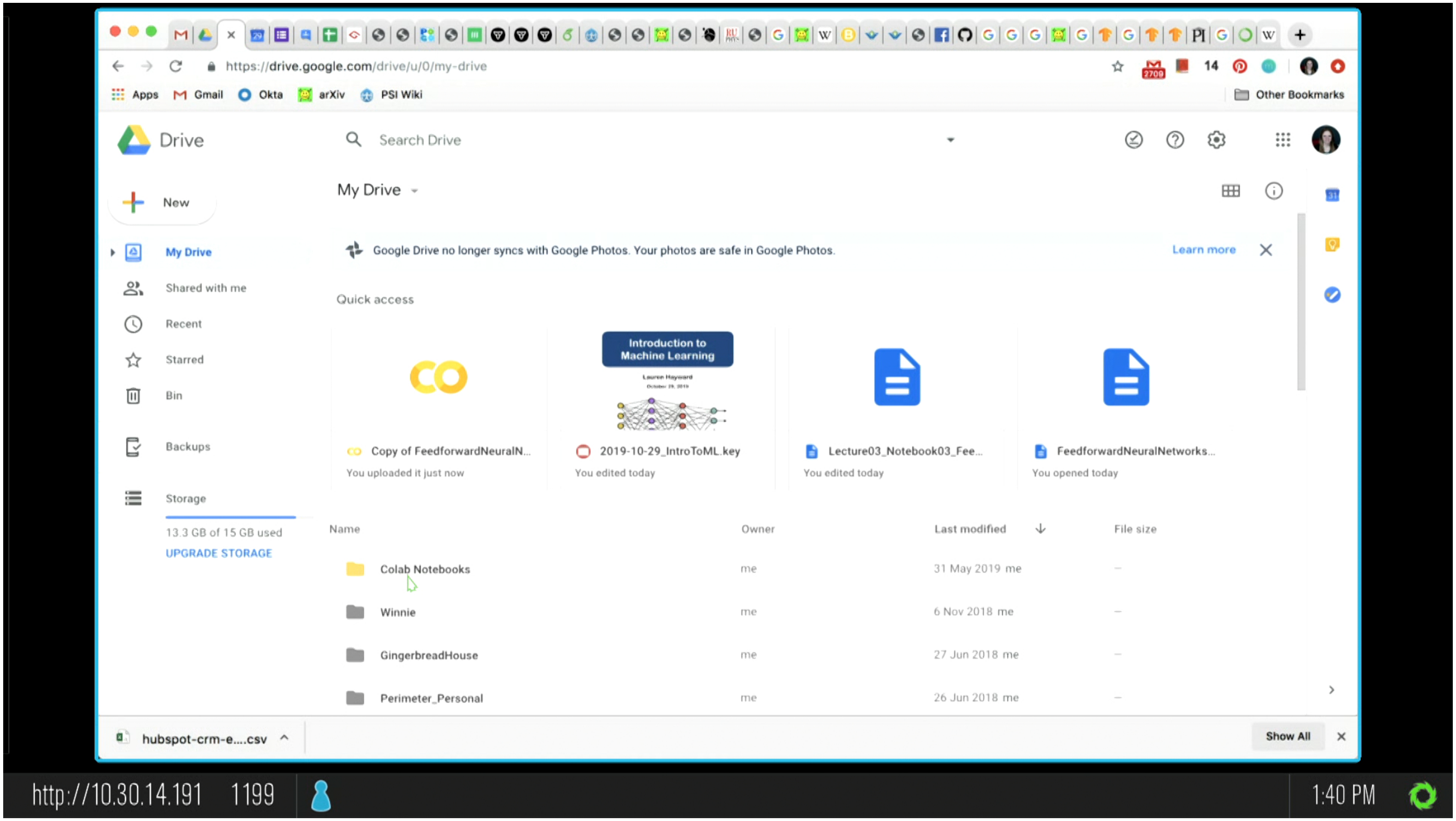
October 29, 2019

Lauren Hayward

The objective of this tutorial is to become comfortable with using the software library TensorFlow to create and train a simple feedforward neural network for supervised learning.

Let us start by generating a random dataset of two-dimensional points with $K$ branches. For each datapoint $\mathbf{x} = (x_1, x_2)$, the label is the branch index such that $y = 0, 1, \ldots K-2$ or $K-1$. Our goal is to implement a neural network capable of classifying the branches.

```
[ ] import matplotlib.pyplot as plt
    import numpy as np

    ############################################################################
    ####################### CREATE AND PLOT THE DATA SET #######################
    ############################################################################

    N = 50 # number of points per branch
    K = 3  # number of branches

    N_train = N*K # total number of points in the training set
    x_train = np.zeros((N_train,2)) # matrix containing the 2-dimensional datapoints
    y_train = np.zeros(N_train, dtype='uint8') # labels (not in one-hot representation)

    mag_noise = 0.3  # controls how much noise gets added to the data
```



https://drive.google.com/drive/u/0/my-drive

Apps Gmail Okta arXiv PSI Wiki Other Bookmarks

Drive Search Drive

New

My Drive
Shared with me
Recent
Starred
Bin
Backups
Storage
13.3 GB of 15 GB used
UPGRADE STORAGE

My Drive

Google Drive no longer syncs with Google Photos. Your photos are safe in Google Photos. Learn more

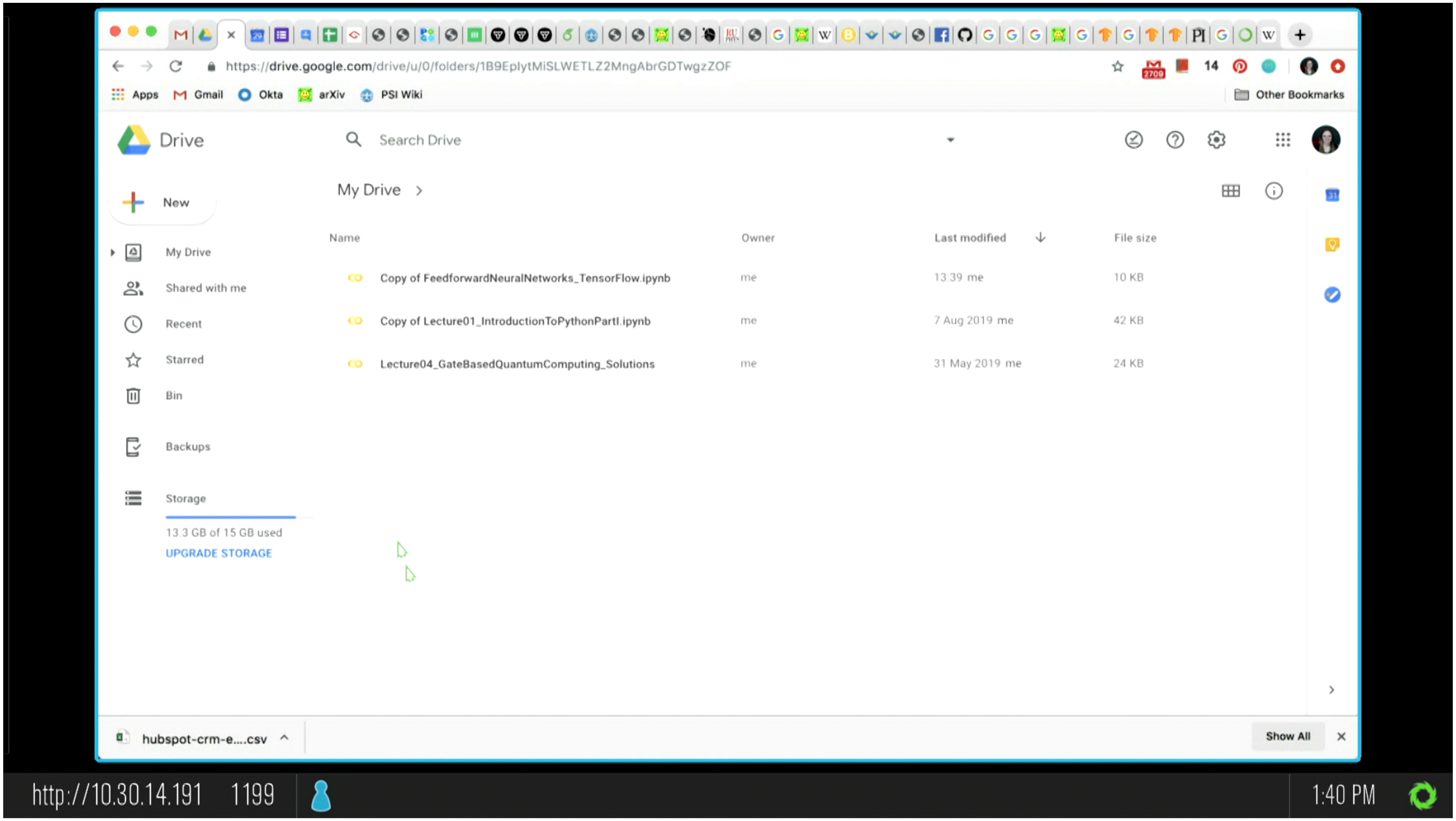
Quick access

Copy of FeedforwardNeuralN... You uploaded it just now
Introduction to Machine Learning — Lauren Hayward — 2019-10-29_IntroToML.key You edited today
Lecture03_Notebook03_Fee... You edited today
FeedforwardNeuralNetworks... You opened today

| Name | Owner | Last modified | File size |
|---|---|---|---|
| Colab Notebooks | me | 31 May 2019 me | – |
| Winnie | me | 6 Nov 2018 me | – |
| GingerbreadHouse | me | 27 Jun 2018 me | – |
| Perimeter_Personal | me | 26 Jun 2018 me | – |

hubspot-crm-e....csv Show All

http://10.30.14.191 1199 1:40 PM



https://drive.google.com/drive/u/0/folders/1B9EplytMiSLWETLZ2MngAbrGDTwgzZOF

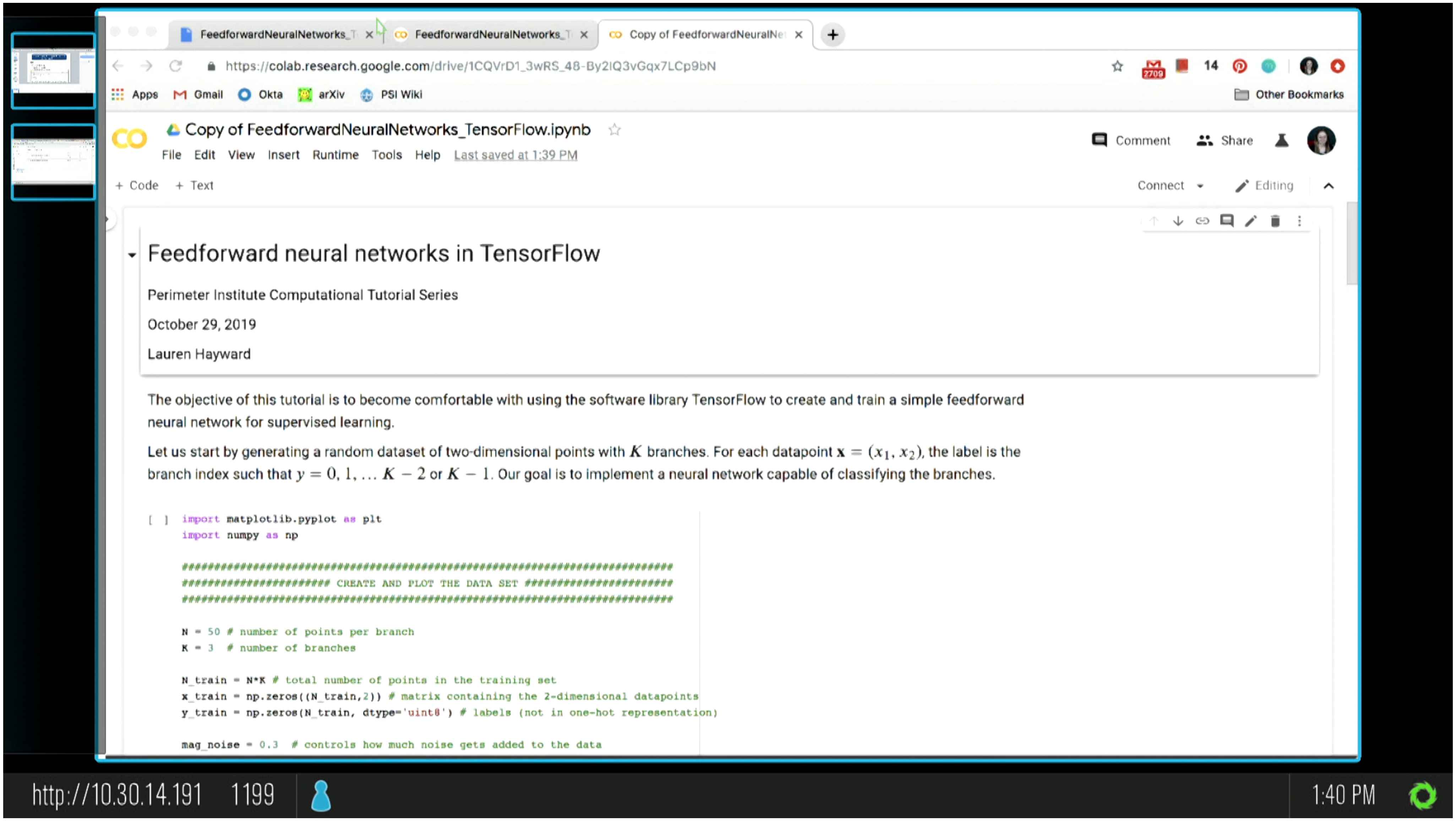
My Drive >

| Name | Owner | Last modified | File size |
|---|---|---|---|
| Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb | me | 13:39 me | 10 KB |
| Copy of Lecture01_IntroductionToPythonPartI.ipynb | me | 7 Aug 2019 me | 42 KB |
| Lecture04_GateBasedQuantumComputing_Solutions | me | 31 May 2019 me | 24 KB |

Storage
13.3 GB of 15 GB used
UPGRADE STORAGE

hubspot-crm-e....csv

http://10.30.14.191 1199 1:40 PM



FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2lQ3vGqx7LCp9bN
Apps Gmail Okta arXiv PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File Edit View Insert Runtime Tools Help Last saved at 1:39 PM
Comment Share
+ Code + Text
Connect Editing

# Feedforward neural networks in TensorFlow

Perimeter Institute Computational Tutorial Series

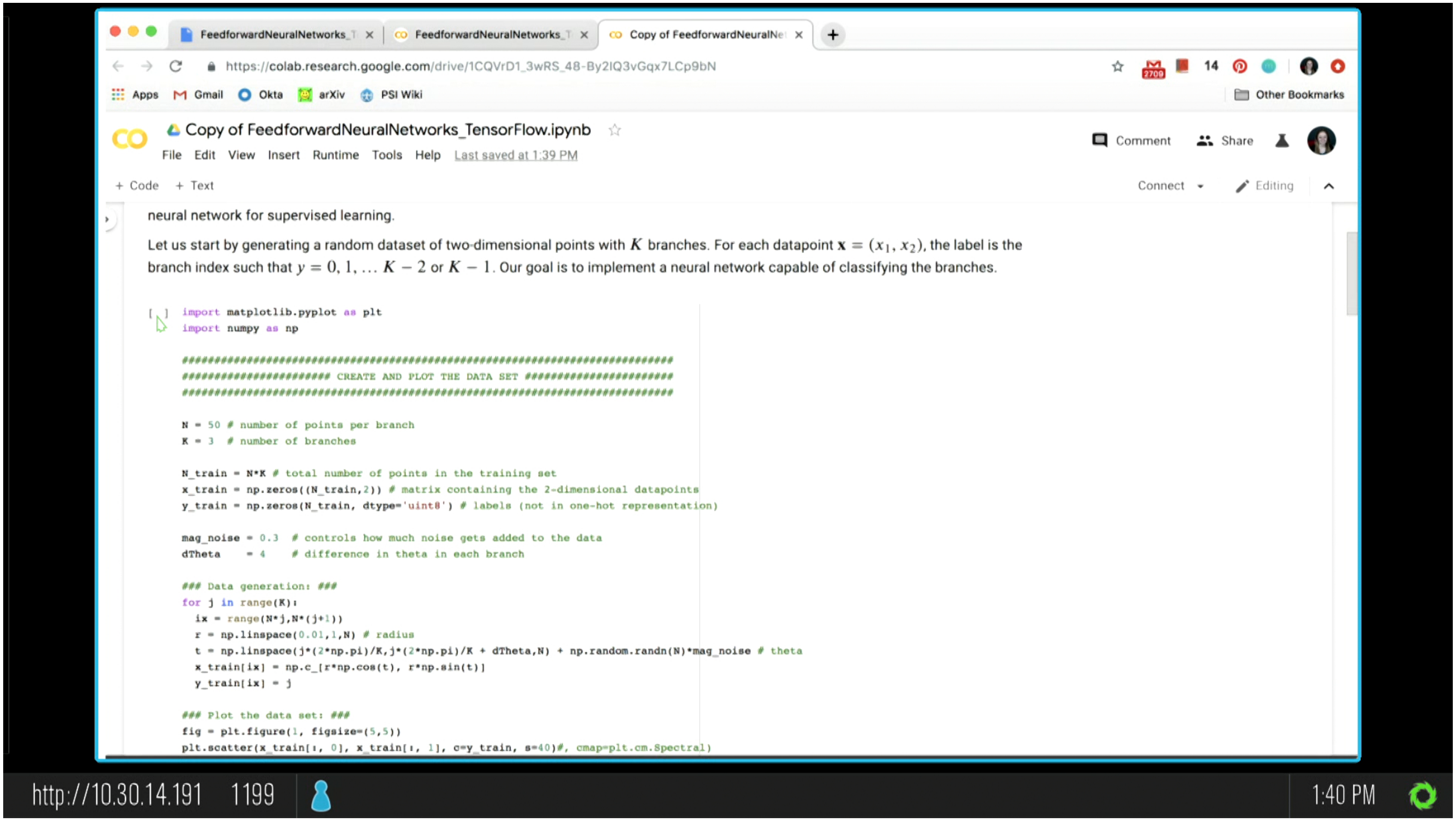
October 29, 2019

Lauren Hayward

The objective of this tutorial is to become comfortable with using the software library TensorFlow to create and train a simple feedforward neural network for supervised learning.

Let us start by generating a random dataset of two-dimensional points with $K$ branches. For each datapoint $\mathbf{x} = (x_1, x_2)$, the label is the branch index such that $y = 0, 1, \dots K-2$ or $K-1$. Our goal is to implement a neural network capable of classifying the branches.

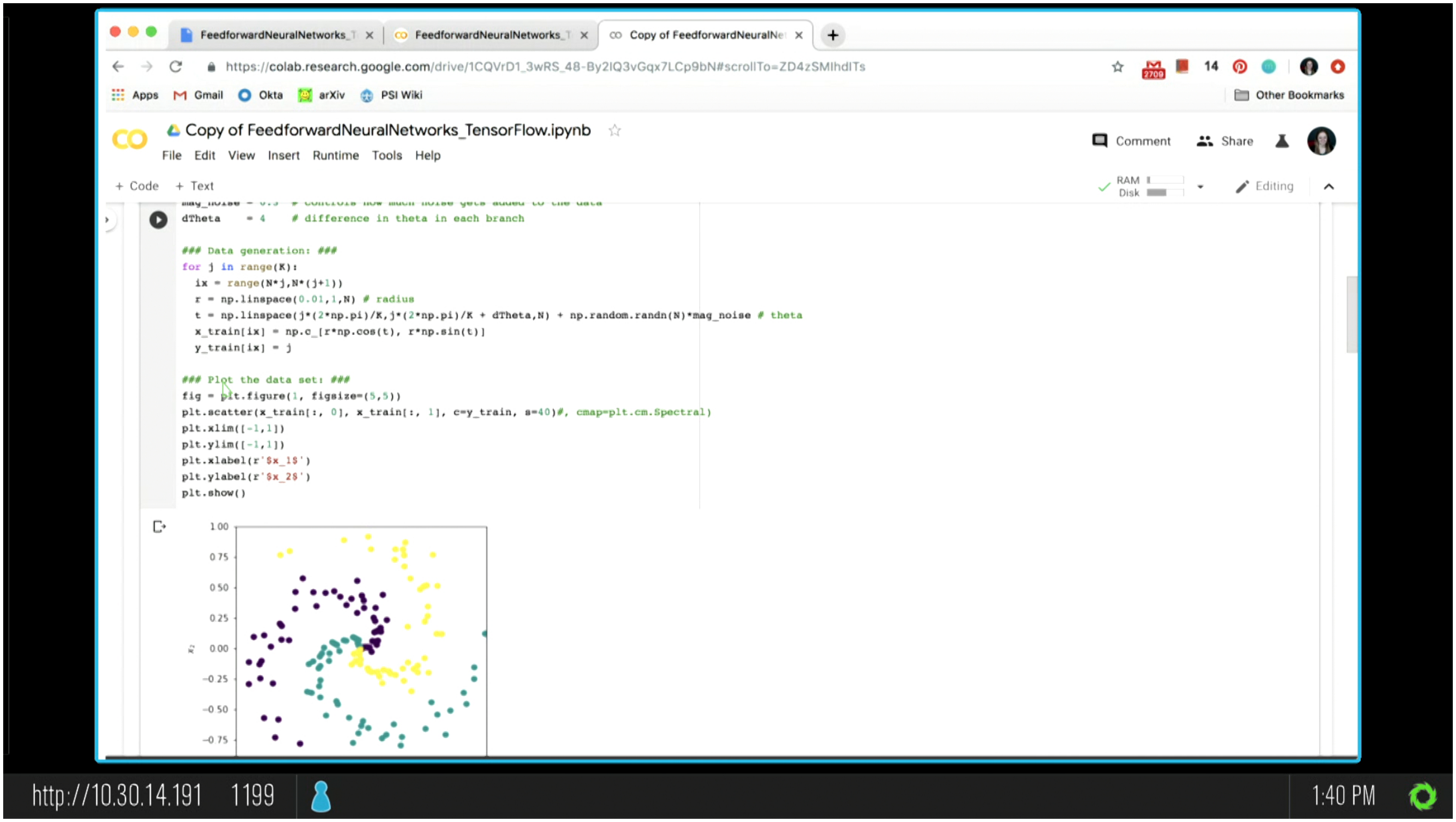
```
[ ]  import matplotlib.pyplot as plt
     import numpy as np

     ############################################################################
     ###################### CREATE AND PLOT THE DATA SET ######################
     ############################################################################

     N = 50 # number of points per branch
     K = 3  # number of branches

     N_train = N*K # total number of points in the training set
     x_train = np.zeros((N_train,2)) # matrix containing the 2-dimensional datapoints
     y_train = np.zeros(N_train, dtype='uint8') # labels (not in one-hot representation)

     mag_noise = 0.3  # controls how much noise gets added to the data
```


FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2lQ3vGqx7LCp9bN
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
Last saved at 1:39 PM
Comment
Share
+ Code
+ Text
Connect
Editing
http://10.30.14.191
1199
1:40 PM

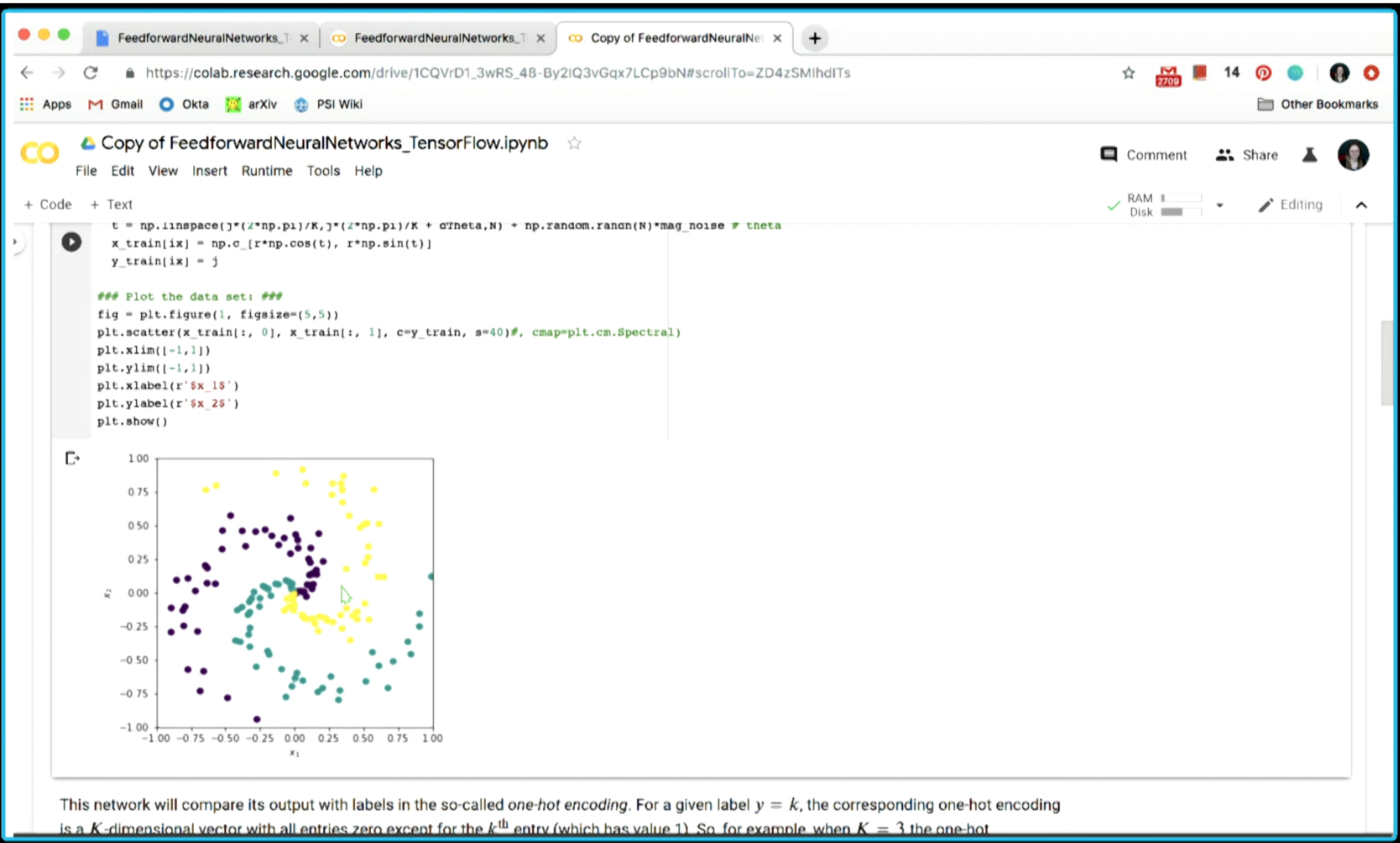
neural network for supervised learning.

Let us start by generating a random dataset of two-dimensional points with $K$ branches. For each datapoint $\mathbf{x} = (x_1, x_2)$, the label is the branch index such that $y = 0, 1, \dots K-2$ or $K-1$. Our goal is to implement a neural network capable of classifying the branches.

```
import matplotlib.pyplot as plt
import numpy as np

############################################################################
######################## CREATE AND PLOT THE DATA SET ######################
############################################################################

N = 50 # number of points per branch
K = 3  # number of branches

N_train = N*K # total number of points in the training set
x_train = np.zeros((N_train,2)) # matrix containing the 2-dimensional datapoints
y_train = np.zeros(N_train, dtype='uint8') # labels (not in one-hot representation)

mag_noise = 0.3  # controls how much noise gets added to the data
dTheta    = 4    # difference in theta in each branch

### Data generation: ###
for j in range(K):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.01,1,N) # radius
  t = np.linspace(j*(2*np.pi)/K,j*(2*np.pi)/K + dTheta,N) + np.random.randn(N)*mag_noise # theta
  x_train[ix] = np.c_[r*np.cos(t), r*np.sin(t)]
  y_train[ix] = j

### Plot the data set: ###
fig = plt.figure(1, figsize=(5,5))
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train, s=40)#, cmap=plt.cm.Spectral)
```

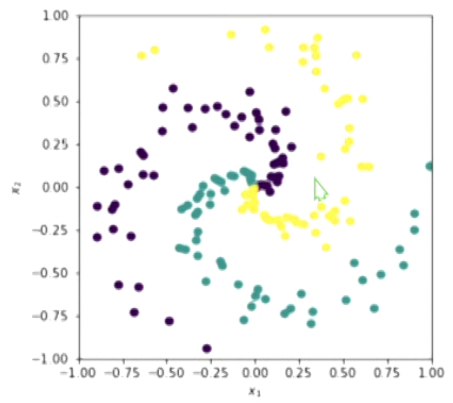

FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2IQ3vGqx7LCp9bN#scrollTo=ZD4zSMIhdITs
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
Comment
Share
+ Code
+ Text
RAM
Disk
Editing
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
$x_2$
http://10.30.14.191
1199
1:40 PM

```
mag_noise = 0.3  # controls how much noise gets added to the data
dTheta    = 4    # difference in theta in each branch

### Data generation: ###
for j in range(K):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.01,1,N) # radius
  t = np.linspace(j*(2*np.pi)/K,j*(2*np.pi)/K + dTheta,N) + np.random.randn(N)*mag_noise # theta
  x_train[ix] = np.c_[r*np.cos(t), r*np.sin(t)]
  y_train[ix] = j

### Plot the data set: ###
fig = plt.figure(1, figsize=(5,5))
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train, s=40)#, cmap=plt.cm.Spectral)
plt.xlim([-1,1])
plt.ylim([-1,1])
plt.xlabel(r'$x_1$')
plt.ylabel(r'$x_2$')
plt.show()
```



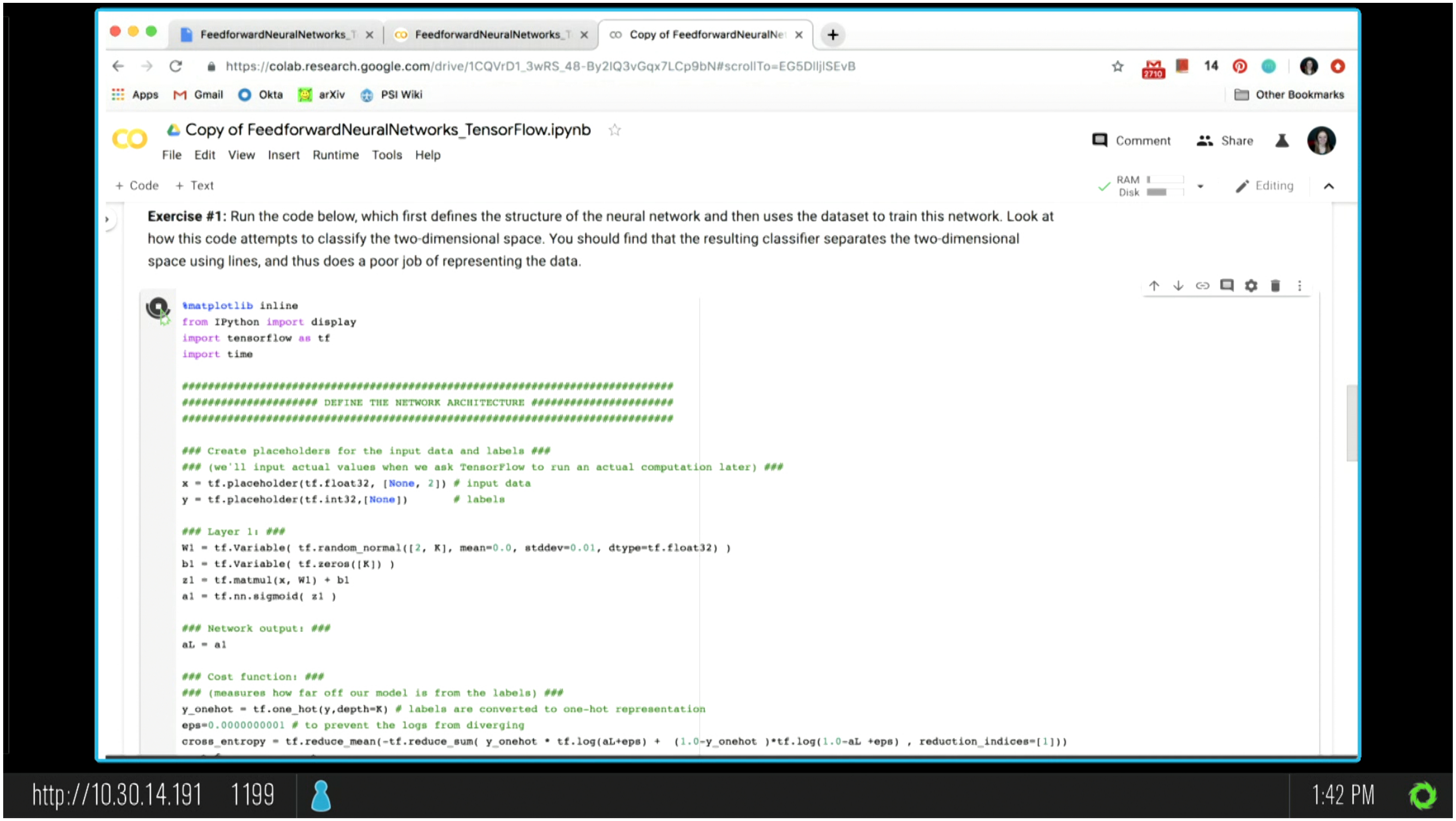
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb

```
  t = np.linspace(j*(2*np.pi)/K,j*(2*np.pi)/K + dTheta,N) + np.random.randn(N)*mag_noise # theta
  x_train[ix] = np.c_[r*np.cos(t), r*np.sin(t)]
  y_train[ix] = j

### Plot the data set: ###
fig = plt.figure(1, figsize=(5,5))
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train, s=40)#, cmap=plt.cm.Spectral)
plt.xlim([-1,1])
plt.ylim([-1,1])
plt.xlabel(r'$x_1$')
plt.ylabel(r'$x_2$')
plt.show()
```



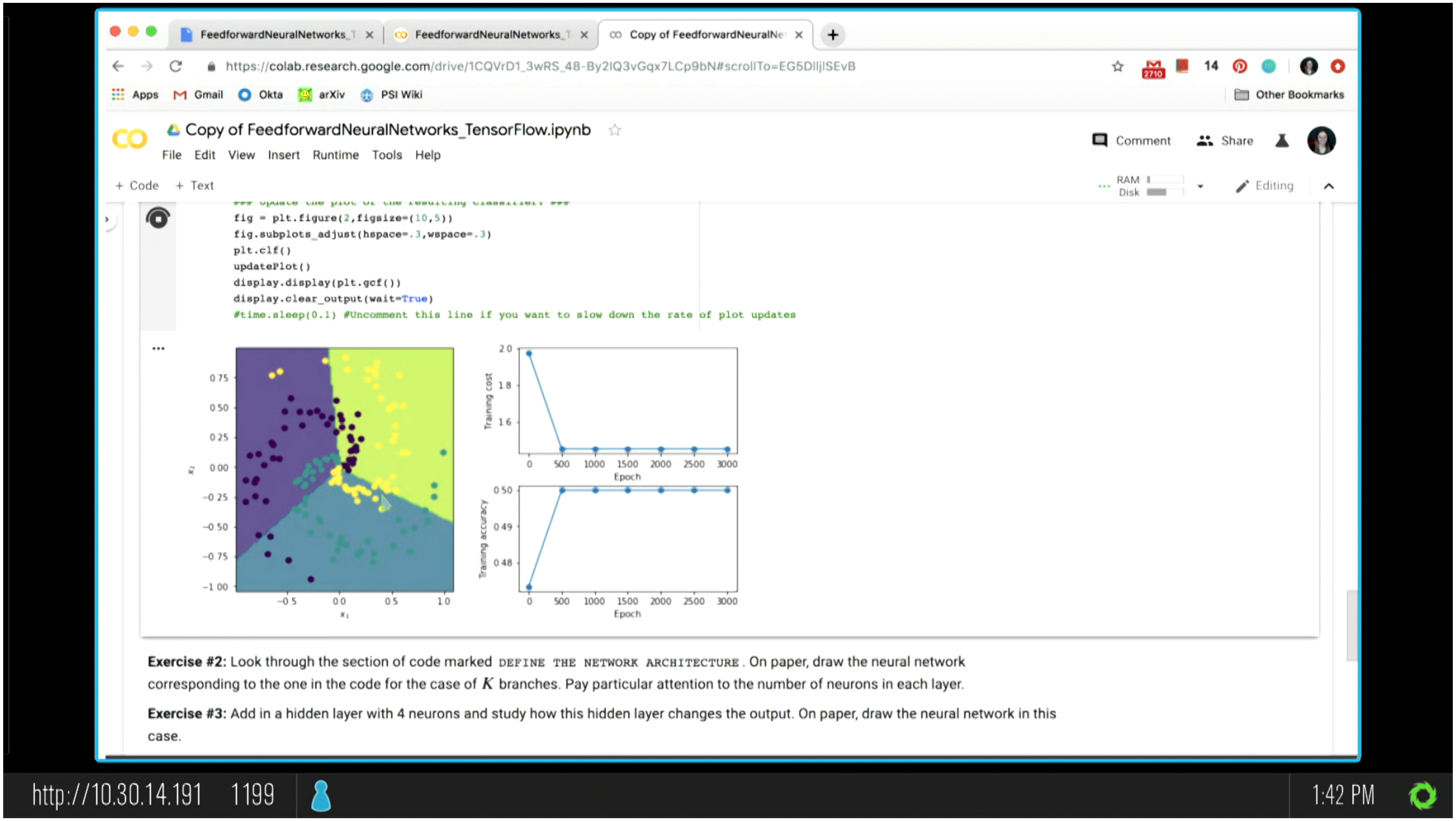
This network will compare its output with labels in the so-called *one-hot encoding*. For a given label $y = k$, the corresponding one-hot encoding is a $K$-dimensional vector with all entries zero except for the $k^{\text{th}}$ entry (which has value 1). So, for example, when $K = 3$ the one-hot

http://10.30.14.191 1199 1:40 PM


FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2lQ3vGqx7LCp9bN#scrollTo=EG5DlljlSEvB
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
Comment
Share
+ Code
+ Text
RAM
Disk
Editing
http://10.30.14.191
1199
1:42 PM

**Exercise #1:** Run the code below, which first defines the structure of the neural network and then uses the dataset to train this network. Look at how this code attempts to classify the two-dimensional space. You should find that the resulting classifier separates the two-dimensional space using lines, and thus does a poor job of representing the data.

```
%matplotlib inline
from IPython import display
import tensorflow as tf
import time

##############################################################################
#################### DEFINE THE NETWORK ARCHITECTURE ########################
##############################################################################

### Create placeholders for the input data and labels ###
### (we'll input actual values when we ask TensorFlow to run an actual computation later) ###
x = tf.placeholder(tf.float32, [None, 2]) # input data
y = tf.placeholder(tf.int32,[None])       # labels

### Layer 1: ###
W1 = tf.Variable( tf.random_normal([2, K], mean=0.0, stddev=0.01, dtype=tf.float32) )
b1 = tf.Variable( tf.zeros([K]) )
z1 = tf.matmul(x, W1) + b1
a1 = tf.nn.sigmoid( z1 )

### Network output: ###
aL = a1

### Cost function: ###
### (measures how far off our model is from the labels) ###
y_onehot = tf.one_hot(y,depth=K) # labels are converted to one-hot representation
eps=0.0000000001 # to prevent the logs from diverging
cross_entropy = tf.reduce_mean(-tf.reduce_sum( y_onehot * tf.log(aL+eps) +  (1.0-y_onehot )*tf.log(1.0-aL +eps) , reduction_indices=[1]))
```

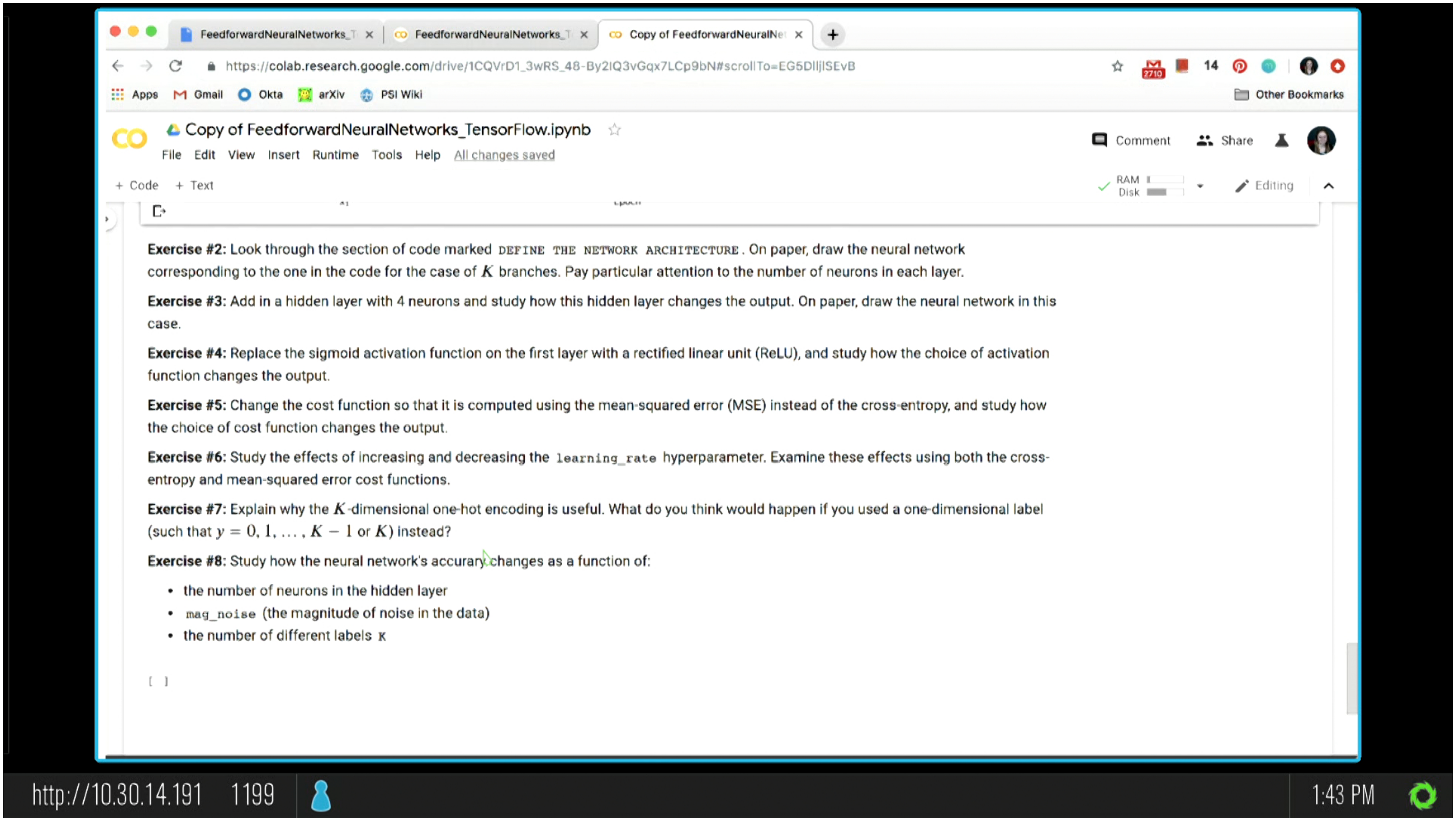


FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2IQ3vGqx7LCp9bN#scrollTo=EG5DlljISEvB
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
Comment
Share
+ Code
+ Text
RAM
Disk
Editing

```
fig = plt.figure(2,figsize=(10,5))
fig.subplots_adjust(hspace=.3,wspace=.3)
plt.clf()
updatePlot()
display.display(plt.gcf())
display.clear_output(wait=True)
#time.sleep(0.1) #Uncomment this line if you want to slow down the rate of plot updates
```

**Exercise #2:** Look through the section of code marked `DEFINE THE NETWORK ARCHITECTURE`. On paper, draw the neural network corresponding to the one in the code for the case of $K$ branches. Pay particular attention to the number of neurons in each layer.

**Exercise #3:** Add in a hidden layer with 4 neurons and study how this hidden layer changes the output. On paper, draw the neural network in this case.


FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2IQ3vGqx7LCp9bN#scrollTo=EG5DlljlSEvB
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
All changes saved
Comment
Share
+ Code
+ Text
RAM
Disk
Editing
http://10.30.14.191 1199
1:43 PM

**Exercise #2:** Look through the section of code marked `DEFINE THE NETWORK ARCHITECTURE`. On paper, draw the neural network corresponding to the one in the code for the case of $K$ branches. Pay particular attention to the number of neurons in each layer.

**Exercise #3:** Add in a hidden layer with 4 neurons and study how this hidden layer changes the output. On paper, draw the neural network in this case.

**Exercise #4:** Replace the sigmoid activation function on the first layer with a rectified linear unit (ReLU), and study how the choice of activation function changes the output.

**Exercise #5:** Change the cost function so that it is computed using the mean-squared error (MSE) instead of the cross-entropy, and study how the choice of cost function changes the output.

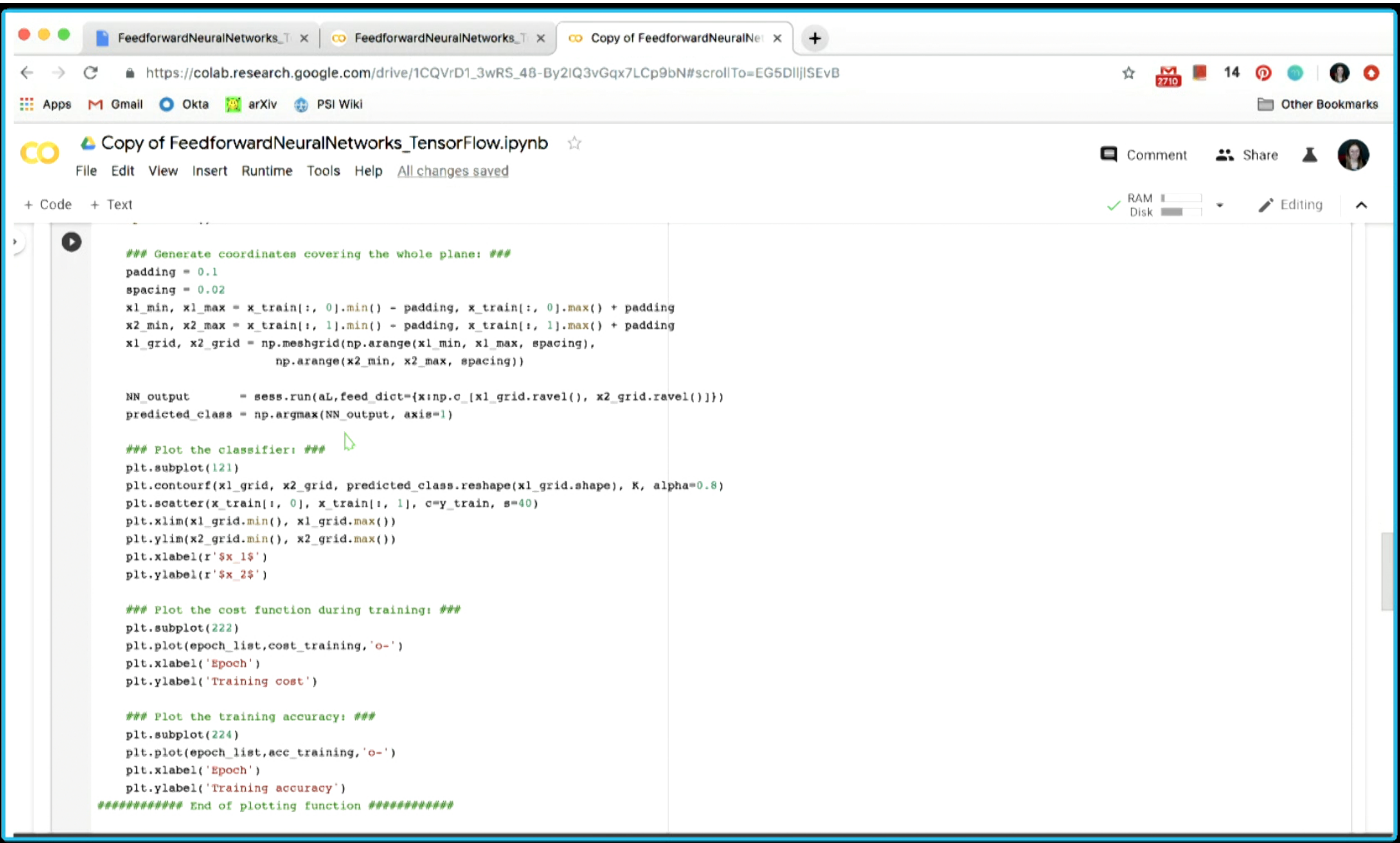
**Exercise #6:** Study the effects of increasing and decreasing the `learning_rate` hyperparameter. Examine these effects using both the cross-entropy and mean-squared error cost functions.

**Exercise #7:** Explain why the $K$-dimensional one-hot encoding is useful. What do you think would happen if you used a one-dimensional label (such that $y = 0, 1, \ldots, K-1$ or $K$) instead?

**Exercise #8:** Study how the neural network's accuracy changes as a function of:

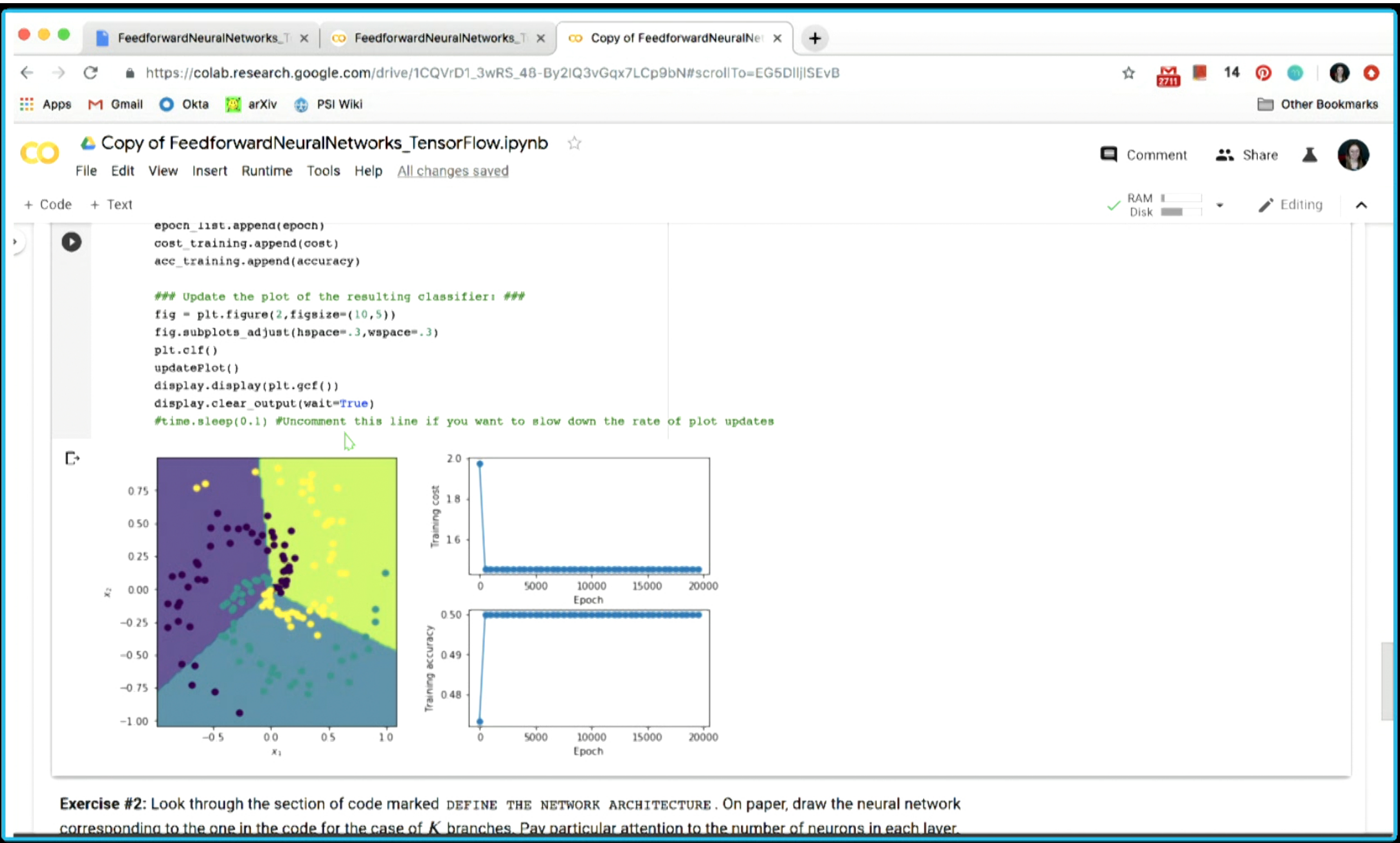
- the number of neurons in the hidden layer
- `mag_noise` (the magnitude of noise in the data)
- the number of different labels `K`

```
[ ]
```


FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2IQ3vGqx7LCp9bN#scrollTo=EG5DlljlSEvB
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
All changes saved
Comment
Share
+ Code
+ Text
RAM
Disk
Editing

```
    ### Generate coordinates covering the whole plane: ###
    padding = 0.1
    spacing = 0.02
    x1_min, x1_max = x_train[:, 0].min() - padding, x_train[:, 0].max() + padding
    x2_min, x2_max = x_train[:, 1].min() - padding, x_train[:, 1].max() + padding
    x1_grid, x2_grid = np.meshgrid(np.arange(x1_min, x1_max, spacing),
                                   np.arange(x2_min, x2_max, spacing))

    NN_output       = sess.run(aL,feed_dict={x:np.c_[x1_grid.ravel(), x2_grid.ravel()]})
    predicted_class = np.argmax(NN_output, axis=1)

    ### Plot the classifier: ###
    plt.subplot(121)
    plt.contourf(x1_grid, x2_grid, predicted_class.reshape(x1_grid.shape), K, alpha=0.8)
    plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train, s=40)
    plt.xlim(x1_grid.min(), x1_grid.max())
    plt.ylim(x2_grid.min(), x2_grid.max())
    plt.xlabel(r'$x_1$')
    plt.ylabel(r'$x_2$')

    ### Plot the cost function during training: ###
    plt.subplot(222)
    plt.plot(epoch_list,cost_training,'o-')
    plt.xlabel('Epoch')
    plt.ylabel('Training cost')

    ### Plot the training accuracy: ###
    plt.subplot(224)
    plt.plot(epoch_list,acc_training,'o-')
    plt.xlabel('Epoch')
    plt.ylabel('Training accuracy')
############ End of plotting function ############
```

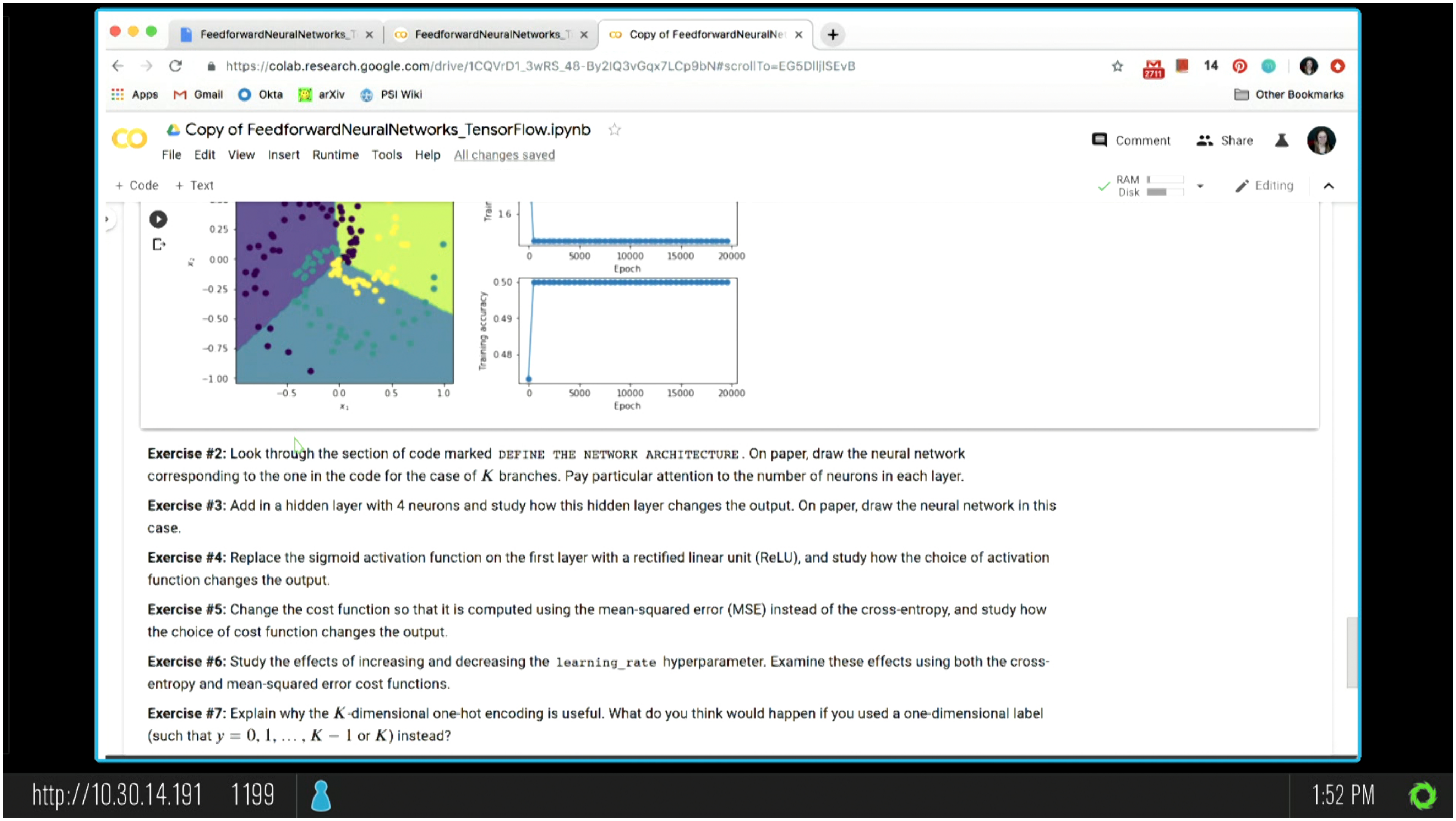
http://10.30.14.191 1199 1:44 PM



FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2lQ3vGqx7LCp9bN#scrollTo=EG5DlljlSEvB
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
All changes saved
Comment
Share
+ Code
+ Text
RAM
Disk
Editing

```
        epoch_list.append(epoch)
        cost_training.append(cost)
        acc_training.append(accuracy)

        ### Update the plot of the resulting classifier: ###
        fig = plt.figure(2,figsize=(10,5))
        fig.subplots_adjust(hspace=.3,wspace=.3)
        plt.clf()
        updatePlot()
        display.display(plt.gcf())
        display.clear_output(wait=True)
        #time.sleep(0.1) #Uncomment this line if you want to slow down the rate of plot updates
```

**Exercise #2:** Look through the section of code marked `DEFINE THE NETWORK ARCHITECTURE`. On paper, draw the neural network corresponding to the one in the code for the case of $K$ branches. Pay particular attention to the number of neurons in each layer.



**Exercise #2:** Look through the section of code marked `DEFINE THE NETWORK ARCHITECTURE`. On paper, draw the neural network corresponding to the one in the code for the case of $K$ branches. Pay particular attention to the number of neurons in each layer.

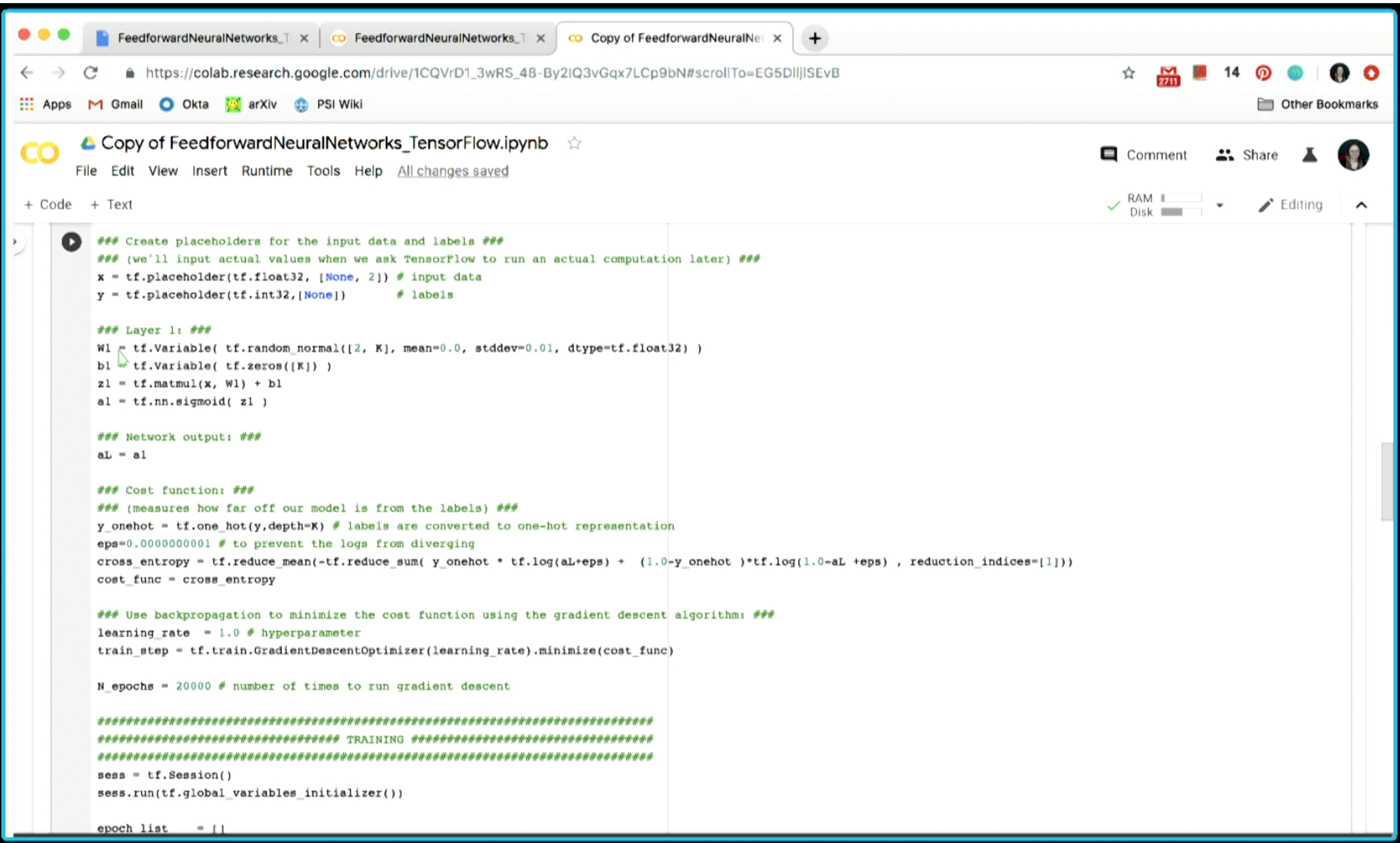
**Exercise #3:** Add in a hidden layer with 4 neurons and study how this hidden layer changes the output. On paper, draw the neural network in this case.

**Exercise #4:** Replace the sigmoid activation function on the first layer with a rectified linear unit (ReLU), and study how the choice of activation function changes the output.

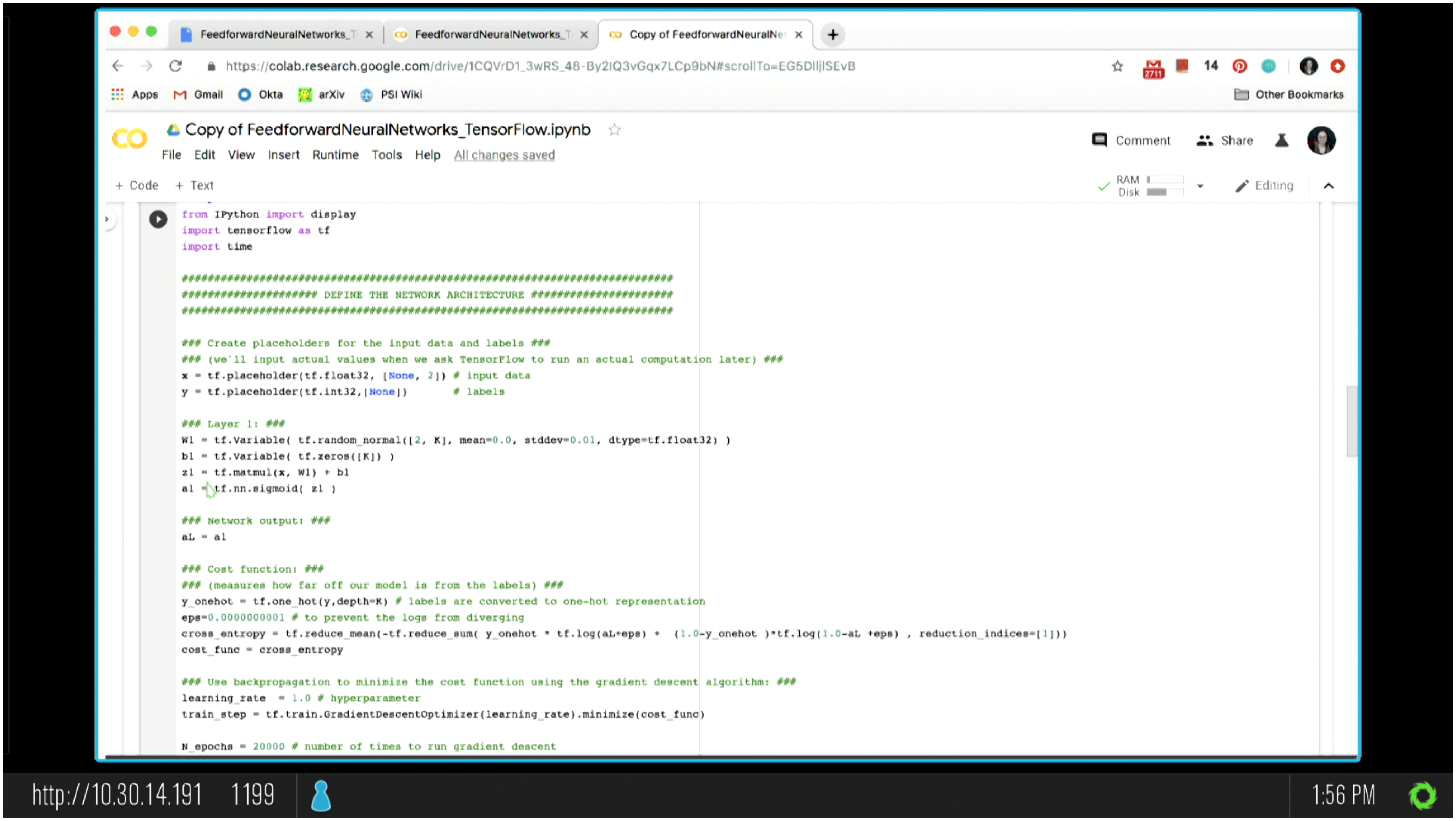
**Exercise #5:** Change the cost function so that it is computed using the mean-squared error (MSE) instead of the cross-entropy, and study how the choice of cost function changes the output.

**Exercise #6:** Study the effects of increasing and decreasing the `learning_rate` hyperparameter. Examine these effects using both the cross-entropy and mean-squared error cost functions.

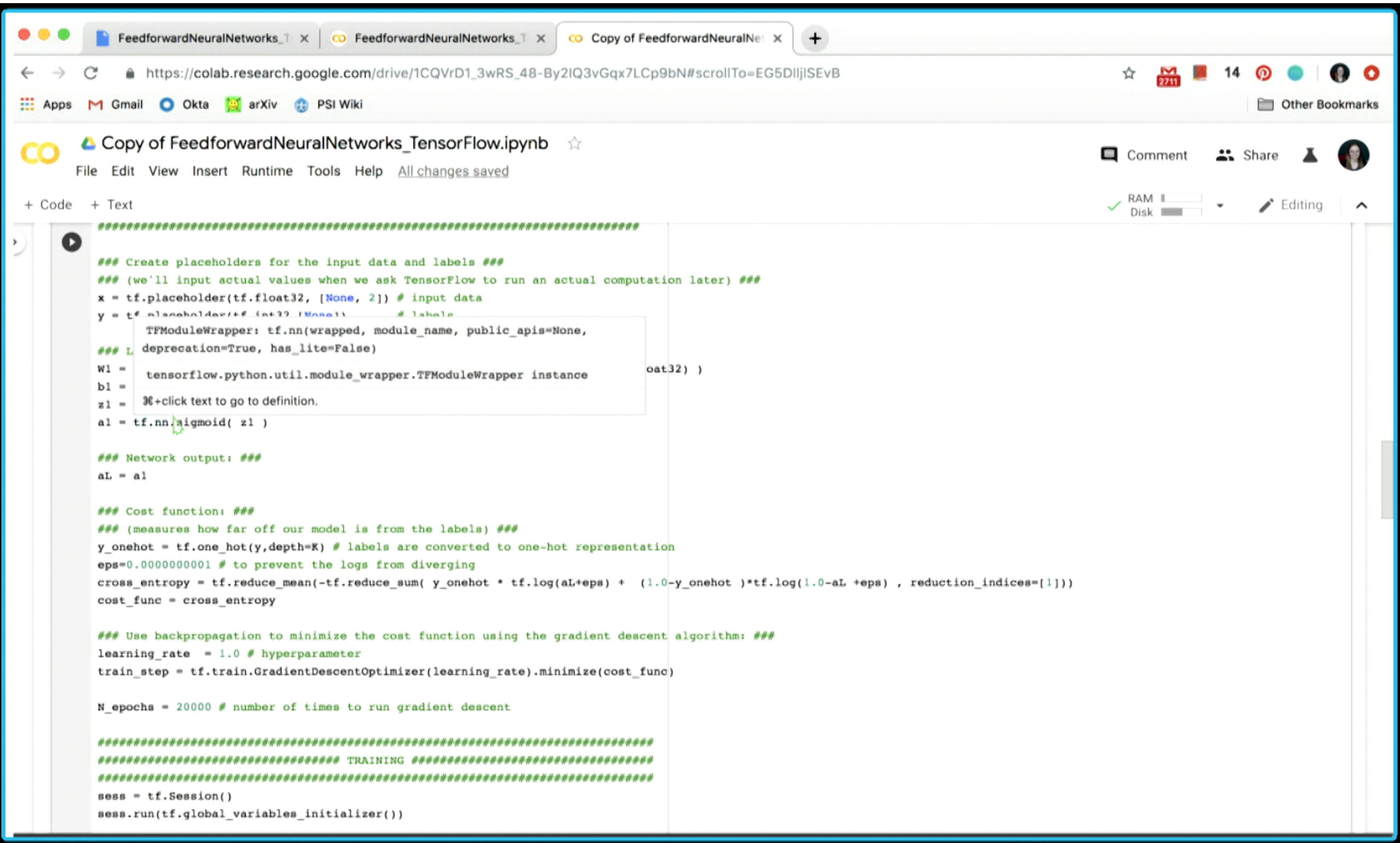
**Exercise #7:** Explain why the $K$-dimensional one-hot encoding is useful. What do you think would happen if you used a one-dimensional label (such that $y = 0, 1, \ldots, K - 1$ or $K$) instead?


FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2lQ3vGqx7LCp9bN#scrollTo=EG5DlljISEvB
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
All changes saved
Comment
Share
+ Code
+ Text
RAM
Disk
Editing

```
### Create placeholders for the input data and labels ###
### (we'll input actual values when we ask TensorFlow to run an actual computation later) ###
x = tf.placeholder(tf.float32, [None, 2]) # input data
y = tf.placeholder(tf.int32,[None])       # labels

### Layer 1: ###
W1 = tf.Variable( tf.random_normal([2, K], mean=0.0, stddev=0.01, dtype=tf.float32) )
b1 = tf.Variable( tf.zeros([K]) )
z1 = tf.matmul(x, W1) + b1
a1 = tf.nn.sigmoid( z1 )

### Network output: ###
aL = a1

### Cost function: ###
### (measures how far off our model is from the labels) ###
y_onehot = tf.one_hot(y,depth=K) # labels are converted to one-hot representation
eps=0.0000000001 # to prevent the logs from diverging
cross_entropy = tf.reduce_mean(-tf.reduce_sum( y_onehot * tf.log(aL+eps) +  (1.0-y_onehot )*tf.log(1.0-aL +eps) , reduction_indices=[1]))
cost_func = cross_entropy

### Use backpropagation to minimize the cost function using the gradient descent algorithm: ###
learning_rate  = 1.0 # hyperparameter
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_func)

N_epochs = 20000 # number of times to run gradient descent

##############################################################################
################################## TRAINING ##################################
##############################################################################
sess = tf.Session()
sess.run(tf.global_variables_initializer())

epoch_list    = []
```


FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2lQ3vGqx7LCp9bN#scrollTo=EG5DIljISEvB
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
All changes saved
Comment
Share
+ Code
+ Text
RAM
Disk
Editing
http://10.30.14.191
1199
1:56 PM

```
from IPython import display
import tensorflow as tf
import time

##############################################################################
##################### DEFINE THE NETWORK ARCHITECTURE #####################
##############################################################################

### Create placeholders for the input data and labels ###
### (we'll input actual values when we ask TensorFlow to run an actual computation later) ###
x = tf.placeholder(tf.float32, [None, 2]) # input data
y = tf.placeholder(tf.int32,[None])        # labels

### Layer 1: ###
W1 = tf.Variable( tf.random_normal([2, K], mean=0.0, stddev=0.01, dtype=tf.float32) )
b1 = tf.Variable( tf.zeros([K]) )
z1 = tf.matmul(x, W1) + b1
a1 = tf.nn.sigmoid( z1 )

### Network output: ###
aL = a1

### Cost function: ###
### (measures how far off our model is from the labels) ###
y_onehot = tf.one_hot(y,depth=K) # labels are converted to one-hot representation
eps=0.0000000001 # to prevent the logs from diverging
cross_entropy = tf.reduce_mean(-tf.reduce_sum( y_onehot * tf.log(aL+eps) +  (1.0-y_onehot )*tf.log(1.0-aL +eps) , reduction_indices=[1]))
cost_func = cross_entropy

### Use backpropagation to minimize the cost function using the gradient descent algorithm: ###
learning_rate  = 1.0 # hyperparameter
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_func)

N_epochs = 20000 # number of times to run gradient descent
```



Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb

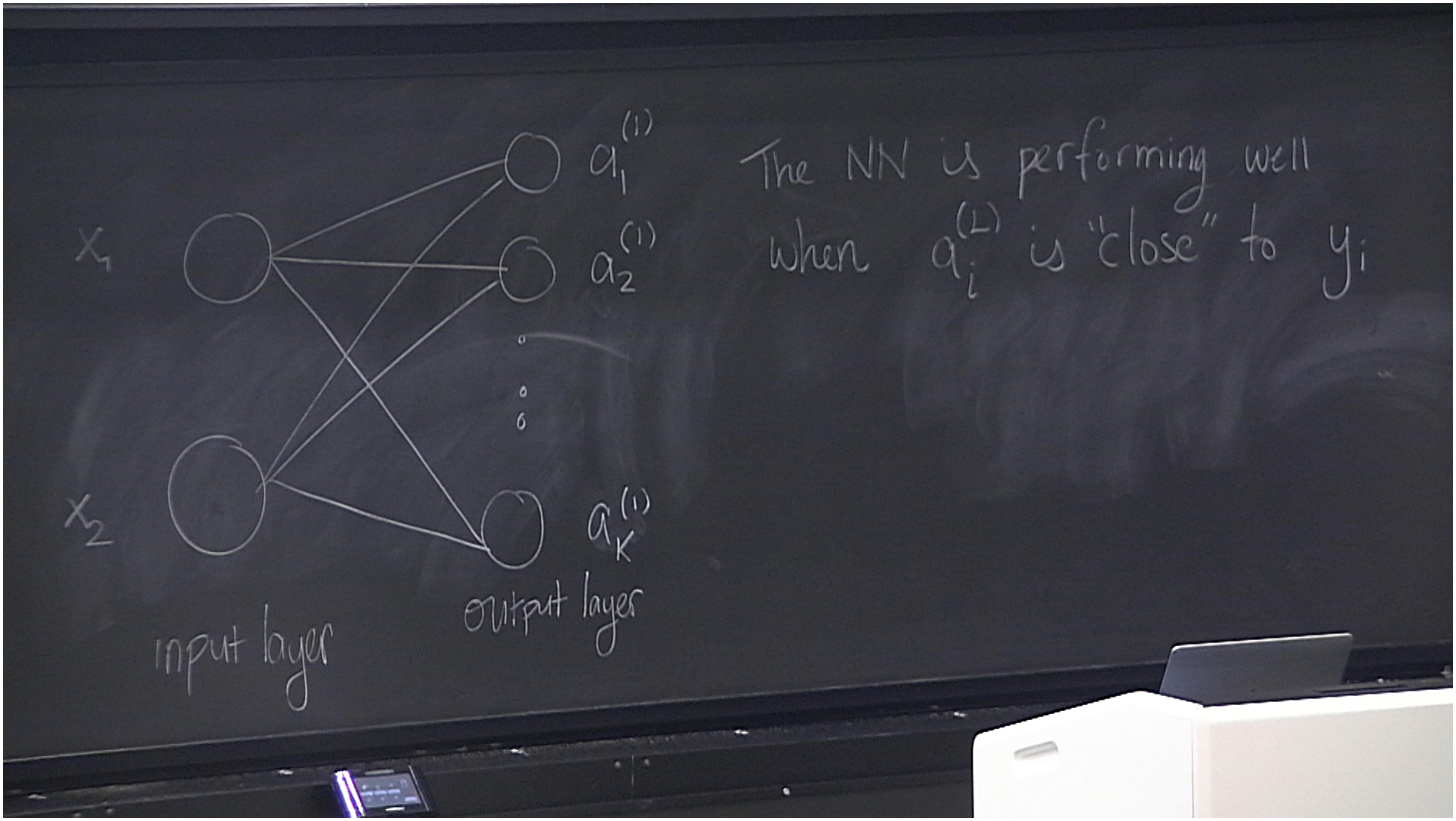
File Edit View Insert Runtime Tools Help All changes saved

```
################################################################################

### Create placeholders for the input data and labels ###
### (we'll input actual values when we ask TensorFlow to run an actual computation later) ###
x = tf.placeholder(tf.float32, [None, 2]) # input data
y = tf.placeholder(tf.int32 [None])         # labels

### L
W1 =                                                                          oat32) )
b1 =
z1 =
a1 = tf.nn.sigmoid( z1 )

### Network output: ###
aL = a1

### Cost function: ###
### (measures how far off our model is from the labels) ###
y_onehot = tf.one_hot(y,depth=K) # labels are converted to one-hot representation
eps=0.0000000001 # to prevent the logs from diverging
cross_entropy = tf.reduce_mean(-tf.reduce_sum( y_onehot * tf.log(aL+eps) +  (1.0-y_onehot )*tf.log(1.0-aL +eps) , reduction_indices=[1]))
cost_func = cross_entropy

### Use backpropagation to minimize the cost function using the gradient descent algorithm: ###
learning_rate  = 1.0 # hyperparameter
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_func)

N_epochs = 20000 # number of times to run gradient descent

################################################################################
################################## TRAINING ####################################
################################################################################
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

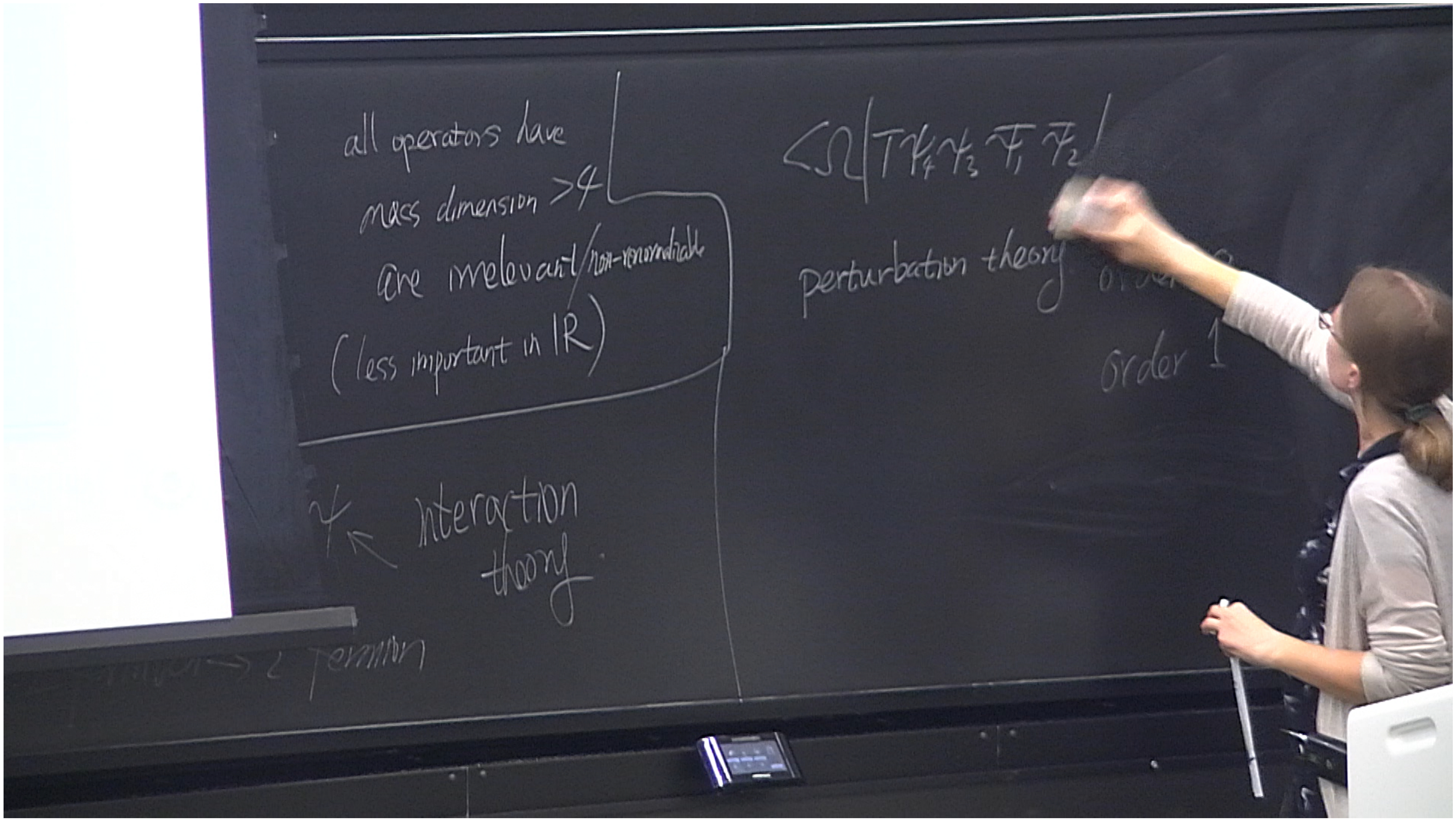
```
TFModuleWrapper: tf.nn(wrapped, module_name, public_apis=None,
deprecation=True, has_lite=False)

tensorflow.python.util.module_wrapper.TFModuleWrapper instance

⌘+click text to go to definition.
```

https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2IQ3vGqx7LCp9bN#scrollTo=EG5DlljlSEvB
+ Code
+ Text
Comment
Share
RAM
Disk
Editing

http://10.30.14.191 1199 1:58 PM



The NN is performing well when $a_i^{(L)}$ is "close" to $y_i$



all operators have mass dimension $>4$ are irrelevant/non-renormalizable (less important in IR)

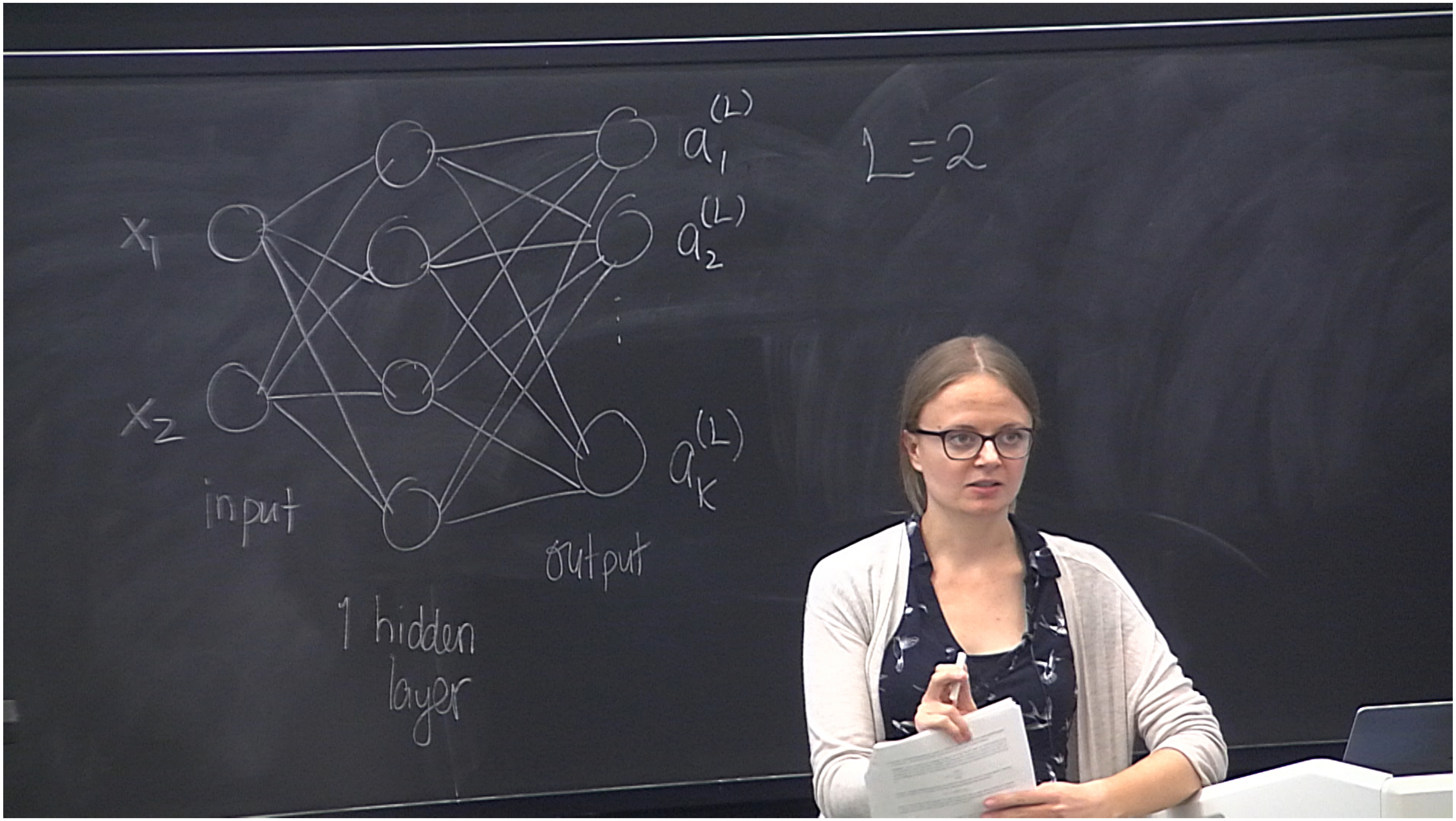
$\psi$ ← interaction theory.

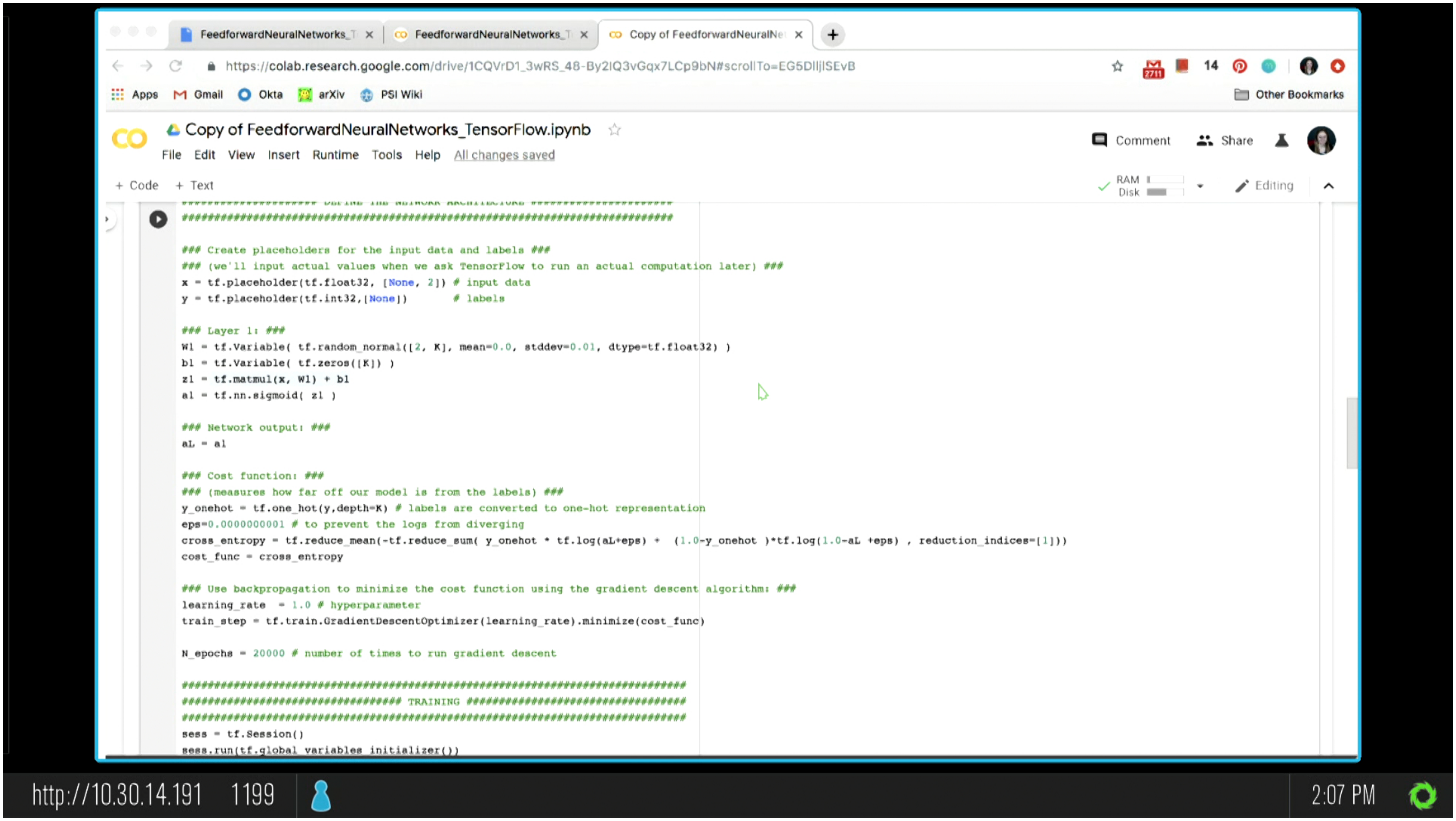
$\langle\Omega|T\psi_4\psi_3\bar{\psi}_1\bar{\psi}_2|$

perturbation theory

order 1

→ 2 fermion


$x_1$
$x_2$
input
1 hidden layer
output
$a_1^{(L)}$
$a_2^{(L)}$
$a_K^{(L)}$
$L=2$

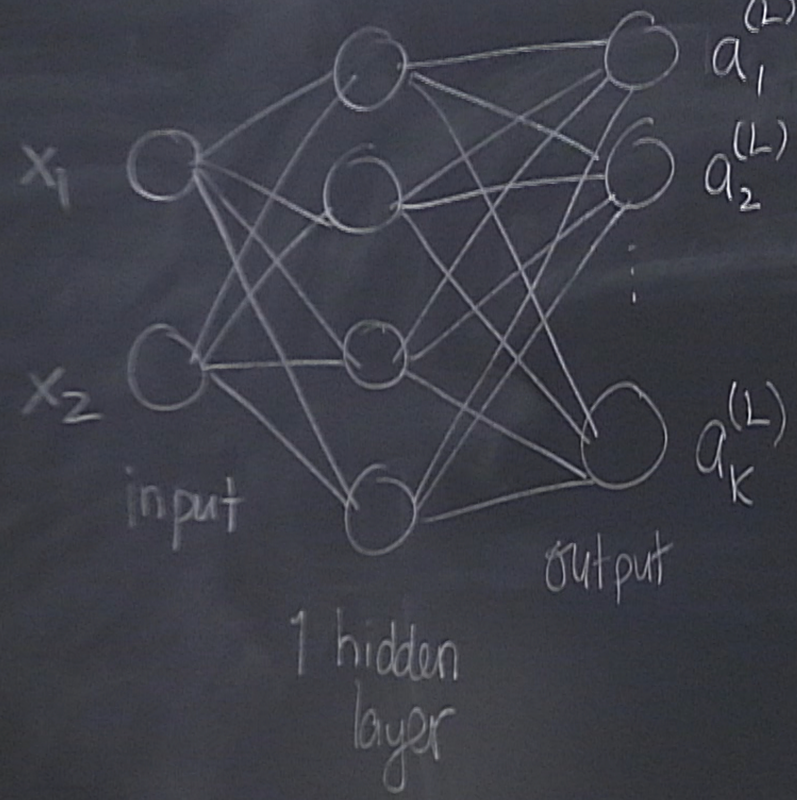


Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
###################### DEFINE THE NETWORK ARCHITECTURE ######################
############################################################################

### Create placeholders for the input data and labels ###
### (we'll input actual values when we ask TensorFlow to run an actual computation later) ###
x = tf.placeholder(tf.float32, [None, 2]) # input data
y = tf.placeholder(tf.int32,[None])       # labels

### Layer 1: ###
W1 = tf.Variable( tf.random_normal([2, K], mean=0.0, stddev=0.01, dtype=tf.float32) )
b1 = tf.Variable( tf.zeros([K]) )
z1 = tf.matmul(x, W1) + b1
a1 = tf.nn.sigmoid( z1 )

### Network output: ###
aL = a1

### Cost function: ###
### (measures how far off our model is from the labels) ###
y_onehot = tf.one_hot(y,depth=K) # labels are converted to one-hot representation
eps=0.0000000001 # to prevent the logs from diverging
cross_entropy = tf.reduce_mean(-tf.reduce_sum( y_onehot * tf.log(aL+eps) +  (1.0-y_onehot )*tf.log(1.0-aL +eps) , reduction_indices=[1]))
cost_func = cross_entropy

### Use backpropagation to minimize the cost function using the gradient descent algorithm: ###
learning_rate  = 1.0 # hyperparameter
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_func)

N_epochs = 20000 # number of times to run gradient descent

############################################################################
################################# TRAINING #################################
############################################################################
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

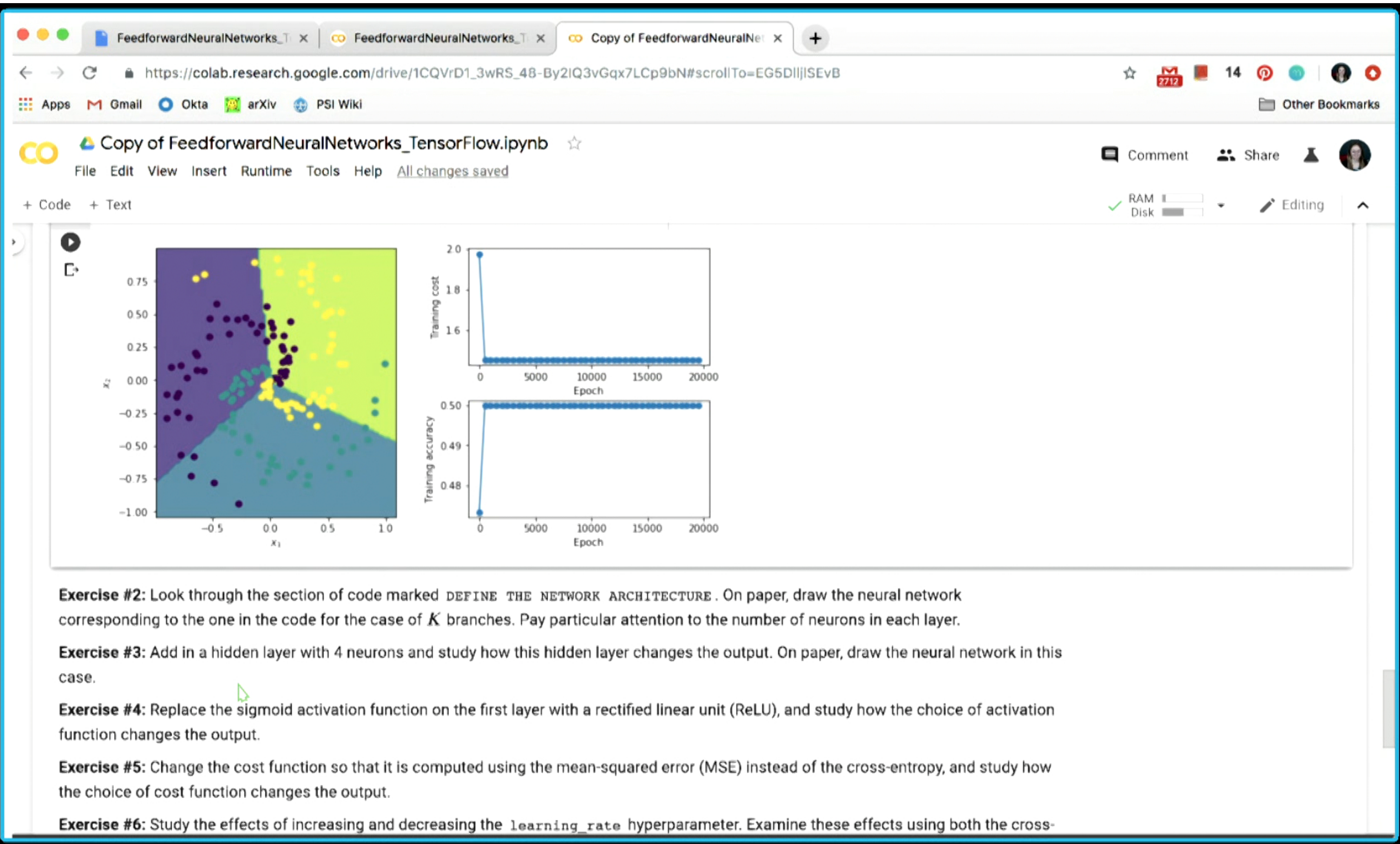
http://10.30.14.191 1199 2:07 PM



$L=2$

$$a_j^{(l)} = g_l\left(\sum_{i=1}^{n_{l-1}} a_i^{(l-1)} W_{ij} + b_j^{(l)}\right)$$

b) $\vec{x}$ : 2-dim. vector


Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File Edit View Insert Runtime Tools Help All changes saved
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2lQ3vGqx7LCp9bN#scrollTo=EG5DlljlSEvB
Training cost
Epoch
Training accuracy
$x_1$
$x_2$

**Exercise #2:** Look through the section of code marked `DEFINE THE NETWORK ARCHITECTURE`. On paper, draw the neural network corresponding to the one in the code for the case of $K$ branches. Pay particular attention to the number of neurons in each layer.

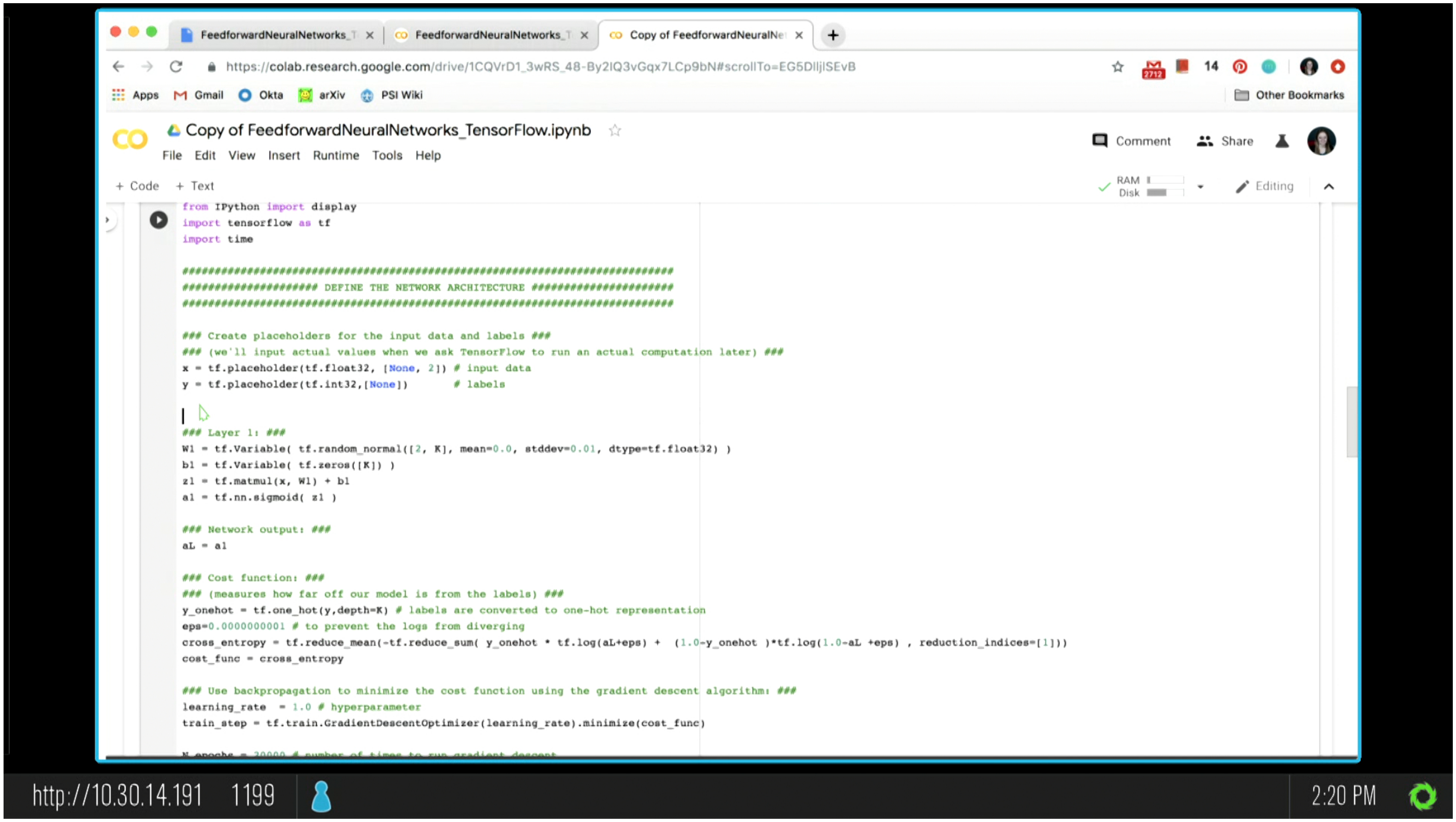
**Exercise #3:** Add in a hidden layer with 4 neurons and study how this hidden layer changes the output. On paper, draw the neural network in this case.

**Exercise #4:** Replace the sigmoid activation function on the first layer with a rectified linear unit (ReLU), and study how the choice of activation function changes the output.

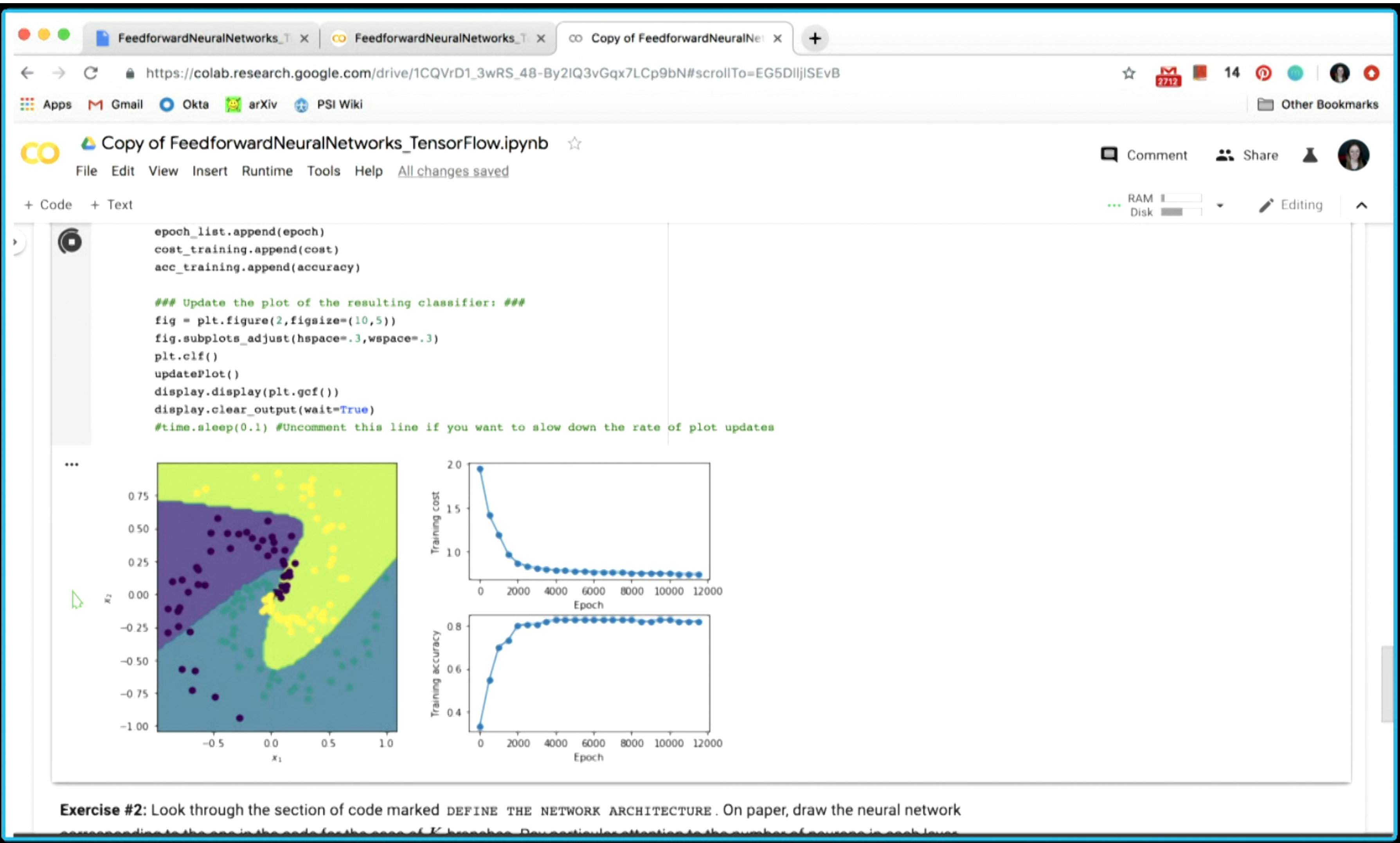
**Exercise #5:** Change the cost function so that it is computed using the mean-squared error (MSE) instead of the cross-entropy, and study how the choice of cost function changes the output.

**Exercise #6:** Study the effects of increasing and decreasing the `learning_rate` hyperparameter. Examine these effects using both the cross-

http://10.30.14.191 1199 2:16 PM


FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2IQ3vGqx7LCp9bN#scrollTo=EG5DlljISEvB
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
Comment
Share
+ Code
+ Text
RAM
Disk
Editing
http://10.30.14.191
1199
2:20 PM

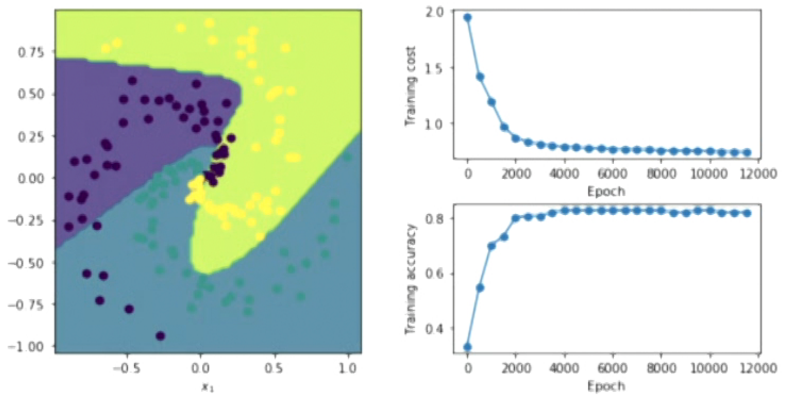
```
from IPython import display
import tensorflow as tf
import time

############################################################################
##################### DEFINE THE NETWORK ARCHITECTURE ######################
############################################################################

### Create placeholders for the input data and labels ###
### (we'll input actual values when we ask TensorFlow to run an actual computation later) ###
x = tf.placeholder(tf.float32, [None, 2]) # input data
y = tf.placeholder(tf.int32,[None])       # labels


### Layer 1: ###
W1 = tf.Variable( tf.random_normal([2, K], mean=0.0, stddev=0.01, dtype=tf.float32) )
b1 = tf.Variable( tf.zeros([K]) )
z1 = tf.matmul(x, W1) + b1
a1 = tf.nn.sigmoid( z1 )

### Network output: ###
aL = a1

### Cost function: ###
### (measures how far off our model is from the labels) ###
y_onehot = tf.one_hot(y,depth=K) # labels are converted to one-hot representation
eps=0.0000000001 # to prevent the logs from diverging
cross_entropy = tf.reduce_mean(-tf.reduce_sum( y_onehot * tf.log(aL+eps) +  (1.0-y_onehot )*tf.log(1.0-aL +eps) , reduction_indices=[1]))
cost_func = cross_entropy

### Use backpropagation to minimize the cost function using the gradient descent algorithm: ###
learning_rate  = 1.0 # hyperparameter
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_func)

N_epochs = 20000 # number of times to run gradient descent
```



https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2lQ3vGqx7LCp9bN#scrollTo=EG5DlljISEvB

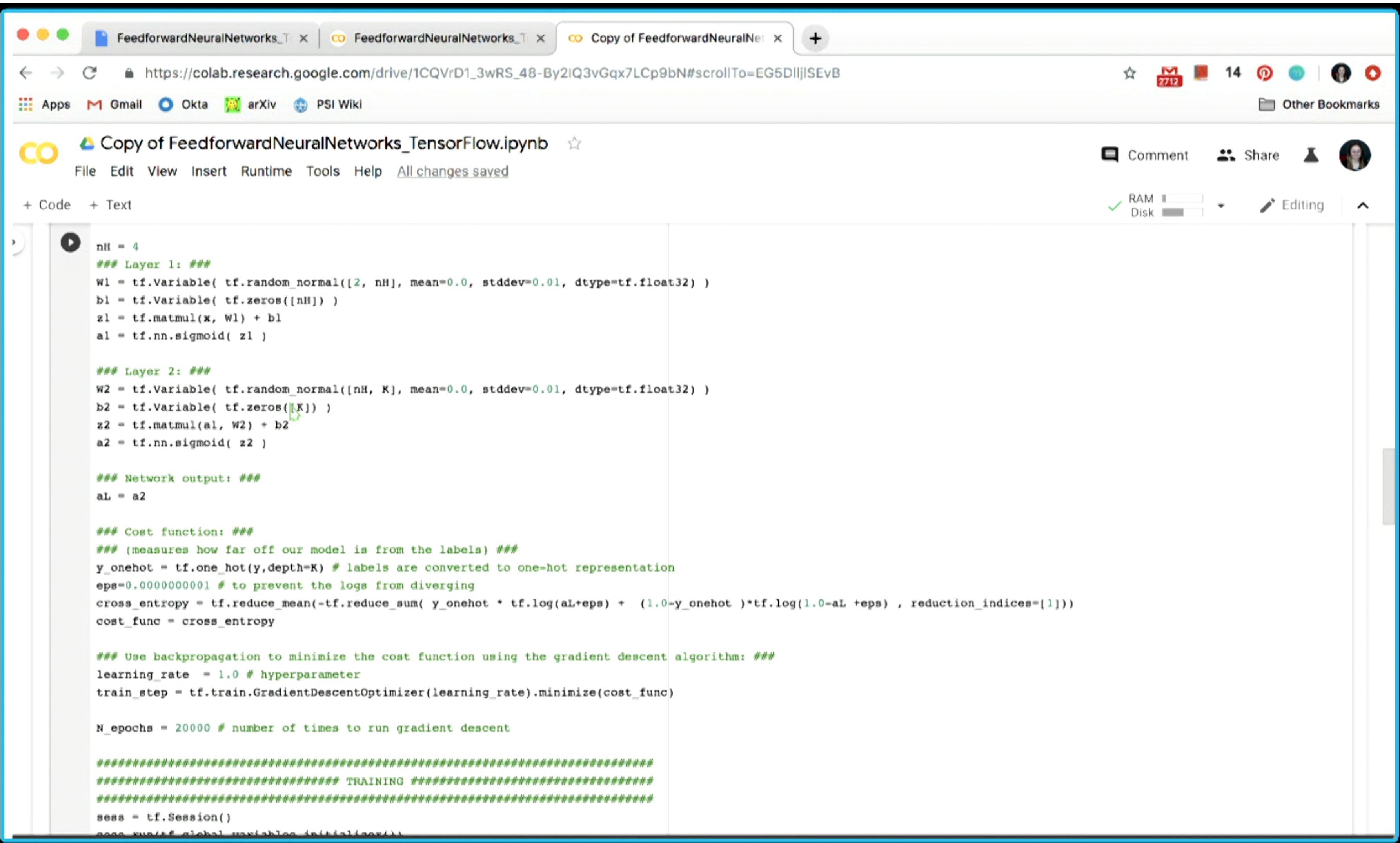
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb

```
    epoch_list.append(epoch)
    cost_training.append(cost)
    acc_training.append(accuracy)

    ### Update the plot of the resulting classifier: ###
    fig = plt.figure(2,figsize=(10,5))
    fig.subplots_adjust(hspace=.3,wspace=.3)
    plt.clf()
    updatePlot()
    display.display(plt.gcf())
    display.clear_output(wait=True)
    #time.sleep(0.1) #Uncomment this line if you want to slow down the rate of plot updates
```



**Exercise #2:** Look through the section of code marked `DEFINE THE NETWORK ARCHITECTURE`. On paper, draw the neural network

http://10.30.14.191 1199 2:23 PM


FeedforwardNeuralNetworks_T
FeedforwardNeuralNetworks_T
Copy of FeedforwardNeuralNe
https://colab.research.google.com/drive/1CQVrD1_3wRS_48-By2lQ3vGqx7LCp9bN#scrollTo=EG5DlljISEvB
Apps
Gmail
Okta
arXiv
PSI Wiki
Other Bookmarks
Copy of FeedforwardNeuralNetworks_TensorFlow.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
RAM
Disk
Editing

```
nH = 4
### Layer 1: ###
W1 = tf.Variable( tf.random_normal([2, nH], mean=0.0, stddev=0.01, dtype=tf.float32) )
b1 = tf.Variable( tf.zeros([nH]) )
z1 = tf.matmul(x, W1) + b1
a1 = tf.nn.sigmoid( z1 )

### Layer 2: ###
W2 = tf.Variable( tf.random_normal([nH, K], mean=0.0, stddev=0.01, dtype=tf.float32) )
b2 = tf.Variable( tf.zeros([K]) )
z2 = tf.matmul(a1, W2) + b2
a2 = tf.nn.sigmoid( z2 )

### Network output: ###
aL = a2

### Cost function: ###
### (measures how far off our model is from the labels) ###
y_onehot = tf.one_hot(y,depth=K) # labels are converted to one-hot representation
eps=0.0000000001 # to prevent the logs from diverging
cross_entropy = tf.reduce_mean(-tf.reduce_sum( y_onehot * tf.log(aL+eps) +  (1.0-y_onehot )*tf.log(1.0-aL +eps) , reduction_indices=[1]))
cost_func = cross_entropy

### Use backpropagation to minimize the cost function using the gradient descent algorithm: ###
learning_rate  = 1.0 # hyperparameter
train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_func)

N_epochs = 20000 # number of times to run gradient descent

##############################################################################
################################## TRAINING ##################################
##############################################################################
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

http://10.30.14.191 1199

2:24 PM