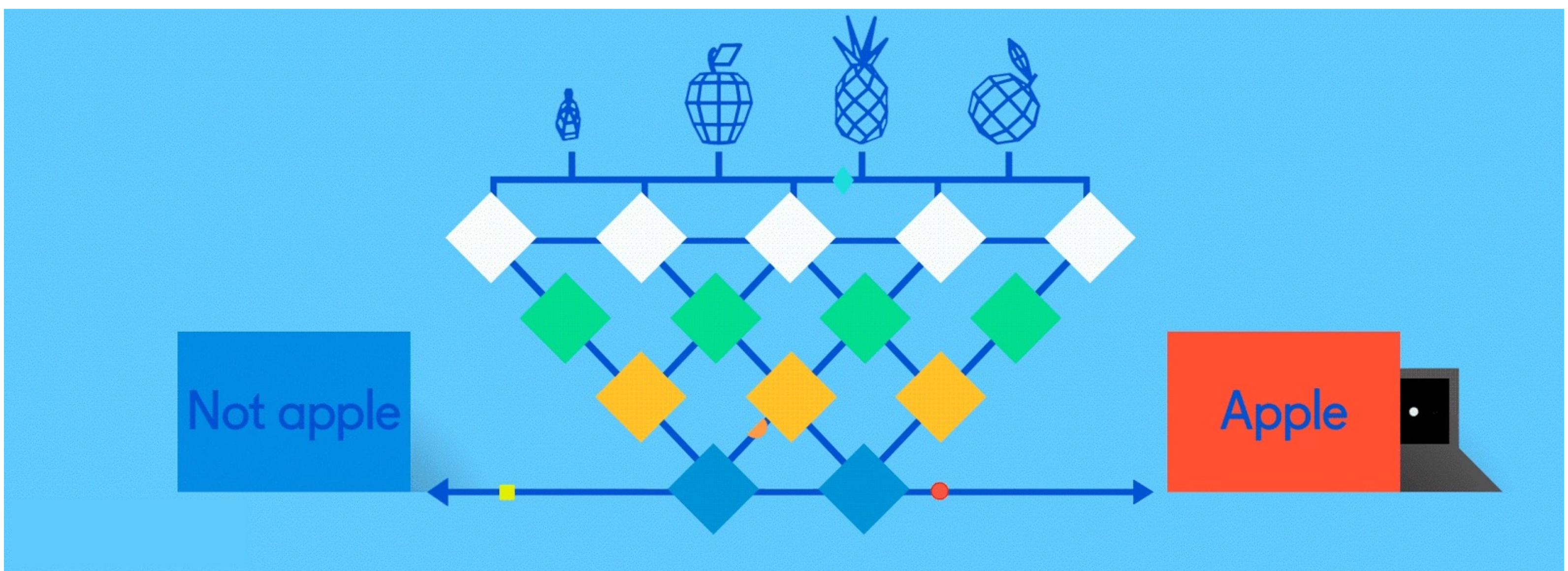
Neural Networks for Machine Learning

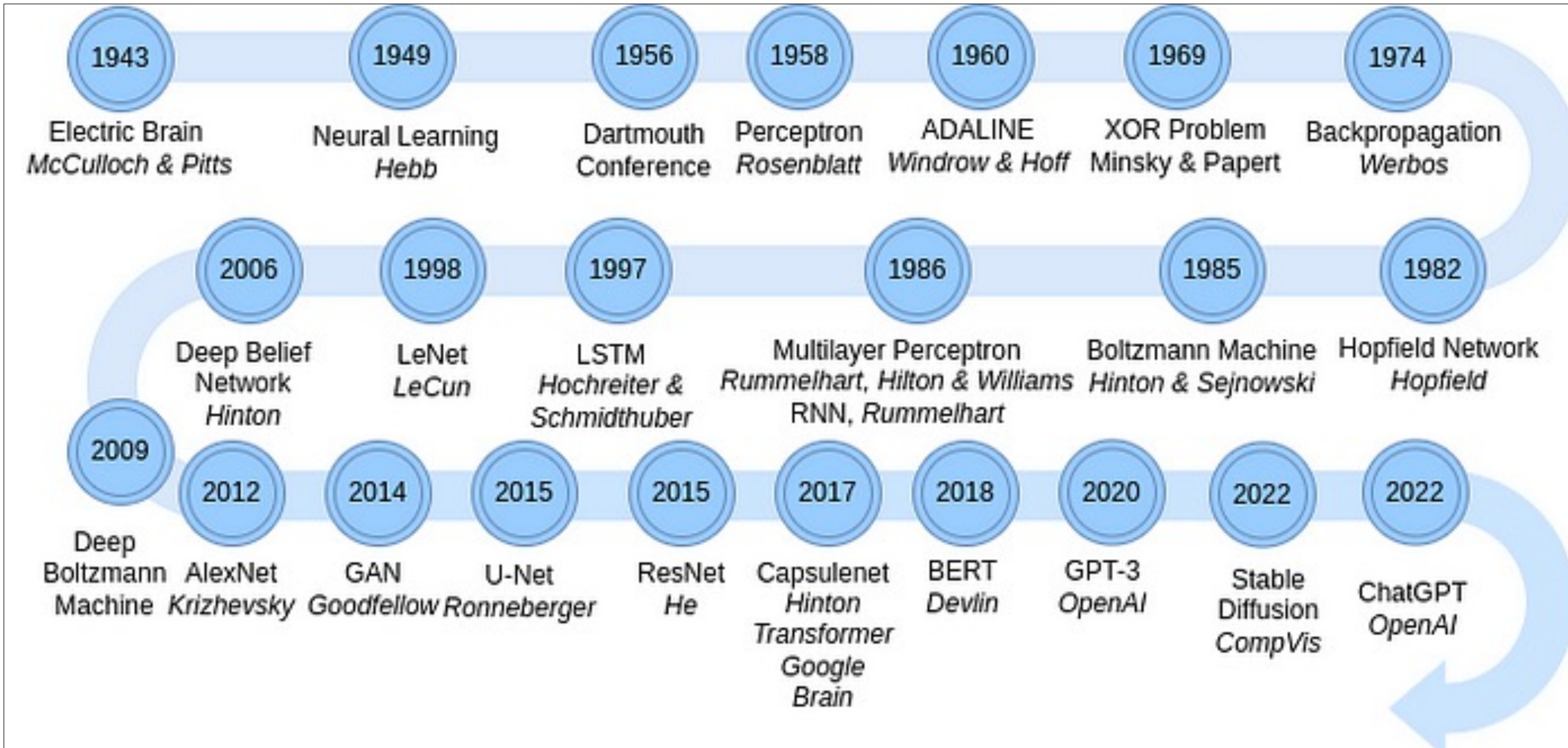
History and Concepts



Overview

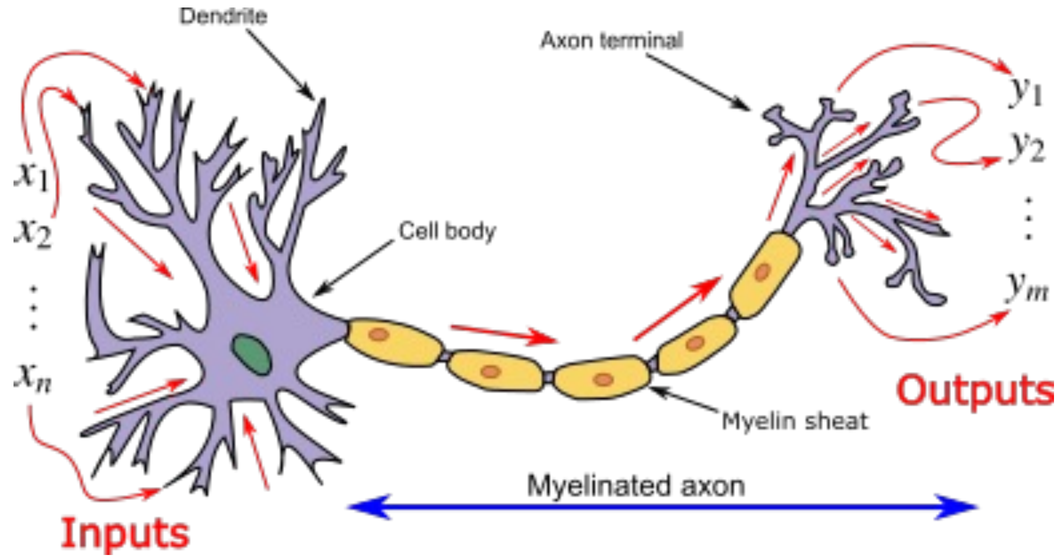
- The neural network computing model has a long history
- Evolved over 75 years to solve its inherent problems, becoming the dominant model for machine learning in the 2010s
- Neural network models often give better results than earlier ML models
- But they are expensive to train and apply
- The field is still evolving rapidly

Neural Network Timeline



Source: [Pumalin]

How do animal brains work?



Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals

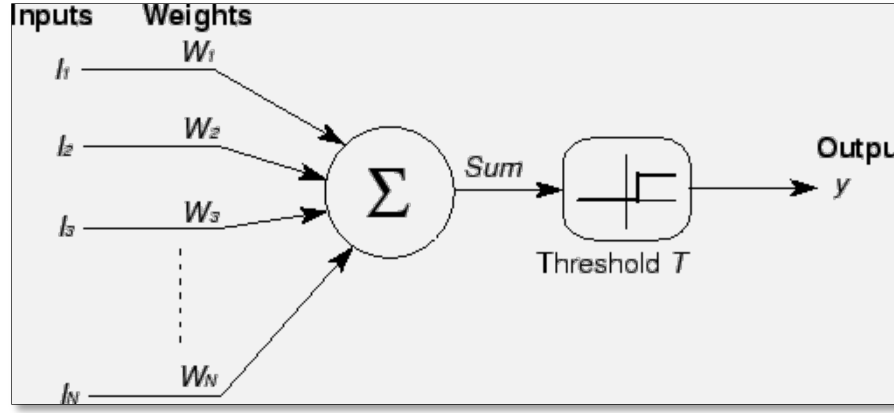
[Neurons](#) have body, axon and many dendrites

- In one of two states: firing and rest
- They fire if total incoming stimulus > threshold

Synapse: thin gap between axon of one neuron and dendrite of another

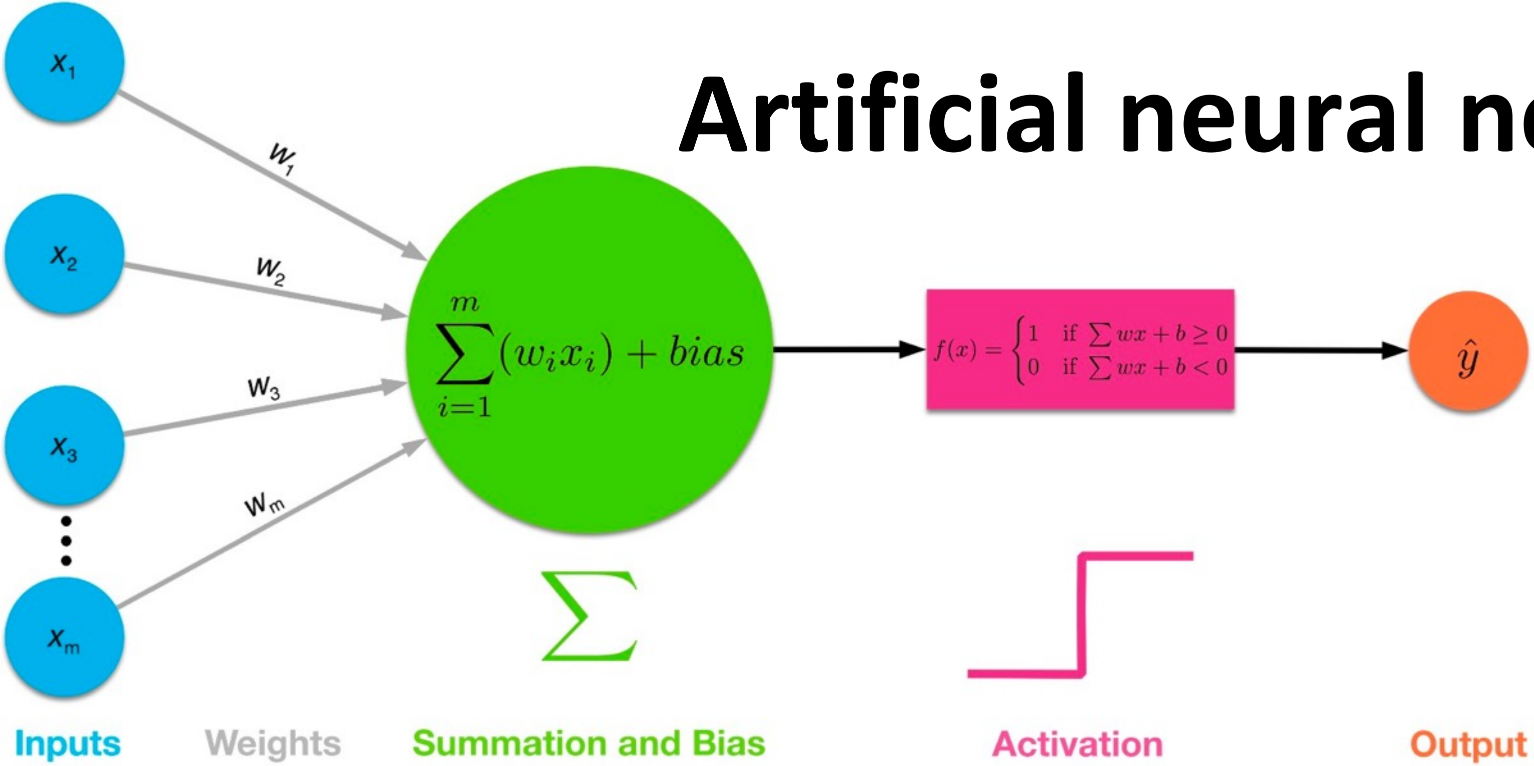
- Signal exchange

McCulloch & Pitts



- First mathematical model of biological neurons, **1943**
- All Boolean operations can be implemented by these neuron-like nodes
- Competitor to Von Neumann model for general purpose computing device
- Origin of automata theory

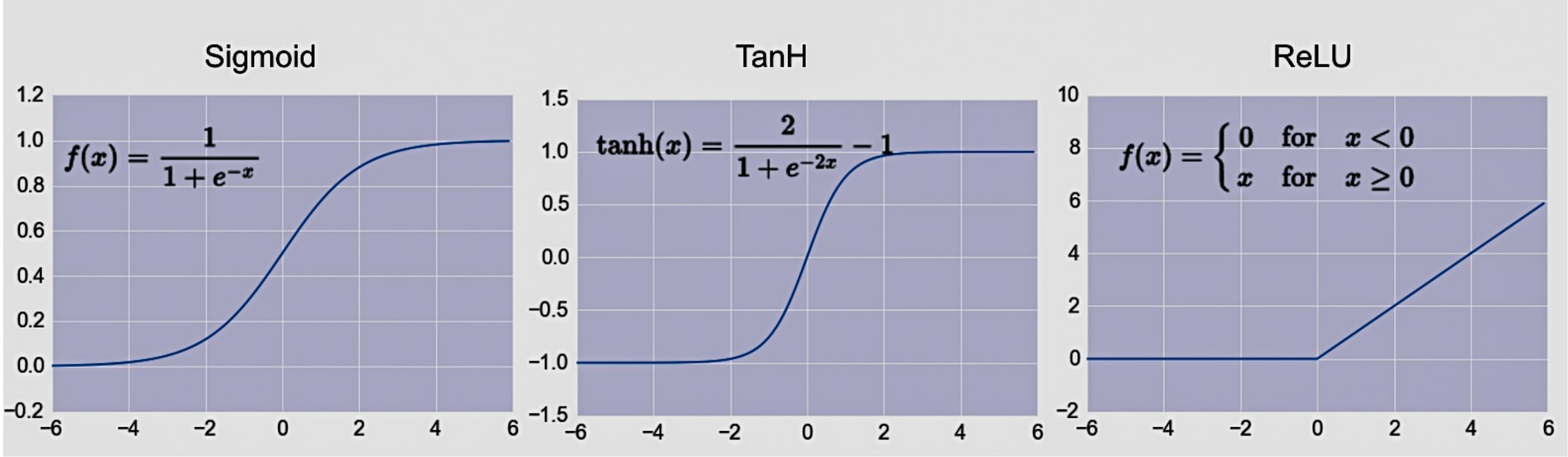
Artificial neural network



- Model still used today!
- Set of **nodes** with inputs and outputs
- Node performs computation via an **activation function**
- **Weighted connections** between nodes
- Connectivity gives network architecture
- NN computations depend on connections, weights, and activation function

Common Activation Functions

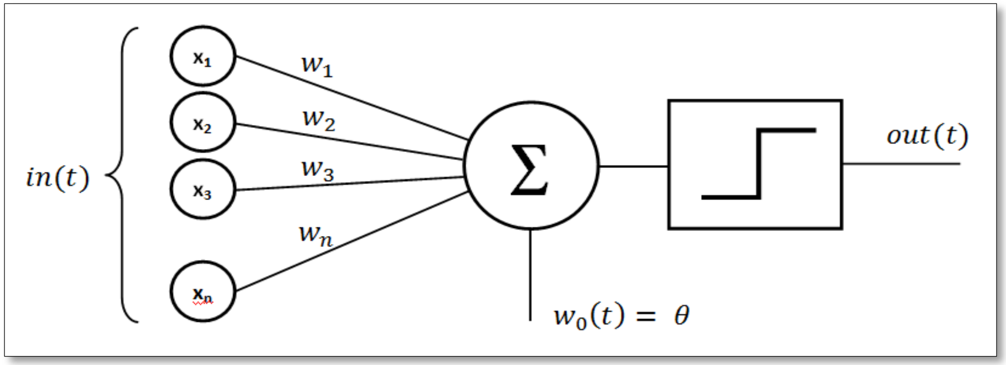
- Define the output of a node given an input
- Very simple functions!



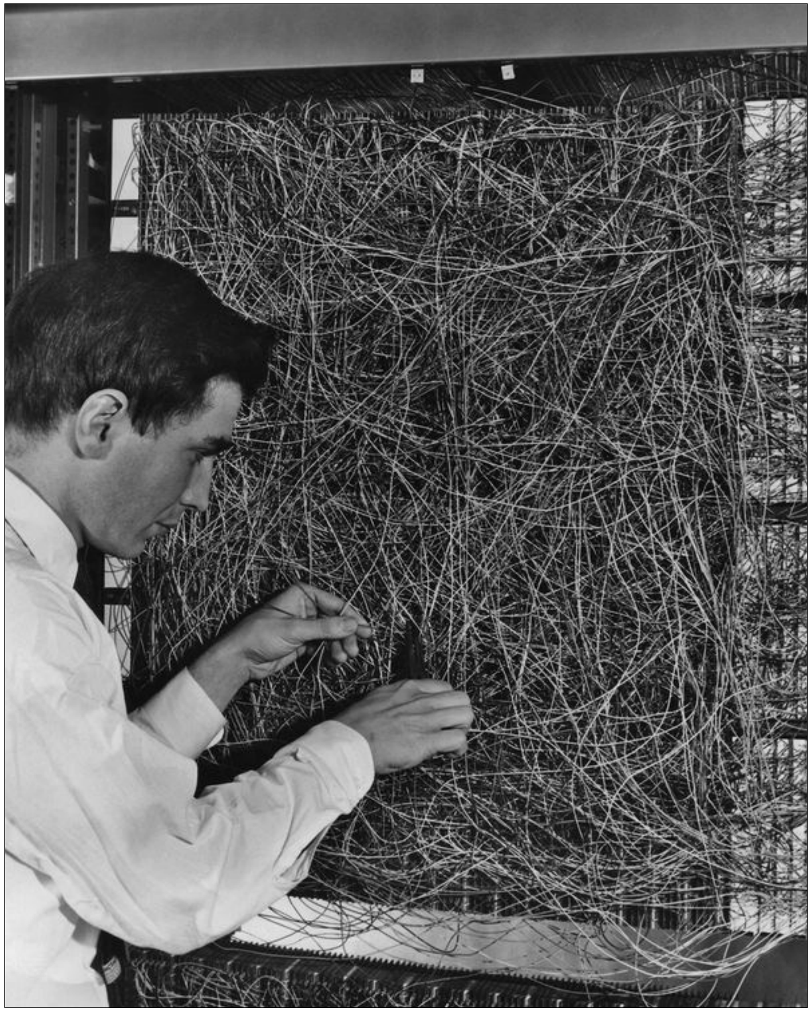
- Choice of activation function depends on problem and available computational power

Rosenblatt's perceptron (1958-60)

- Single layer network of nodes
- Real valued weights +/-
- Supervised learning using a simple learning rule

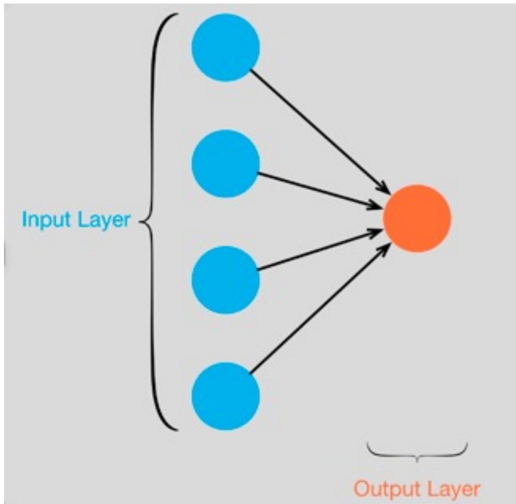


- Essentially a linear classifier
- Widrow & Hoff (1960-62) added better learning rule using gradient descent



Mark 1 perceptron computer, Cornell Aeronautical Lab, 1960

Single Layer Perceptron



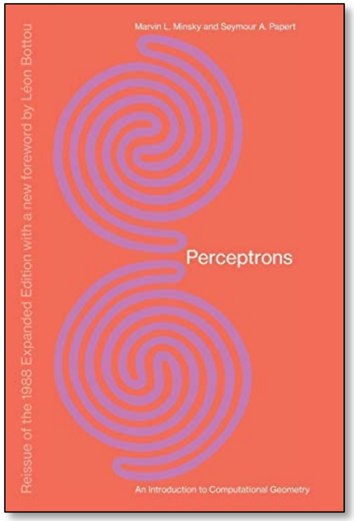
NEW NAVY DEVICE LEARNS BY DOING; Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

SPECIAL TO THE NEW YORK TIMES JULY 8, 1958

WASHINGTON, July 7 (UPI) -- The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

- See the full 1958 NYT article above here
- Rosenblatt: it can **learn** to compute functions by learning weights on inputs from examples

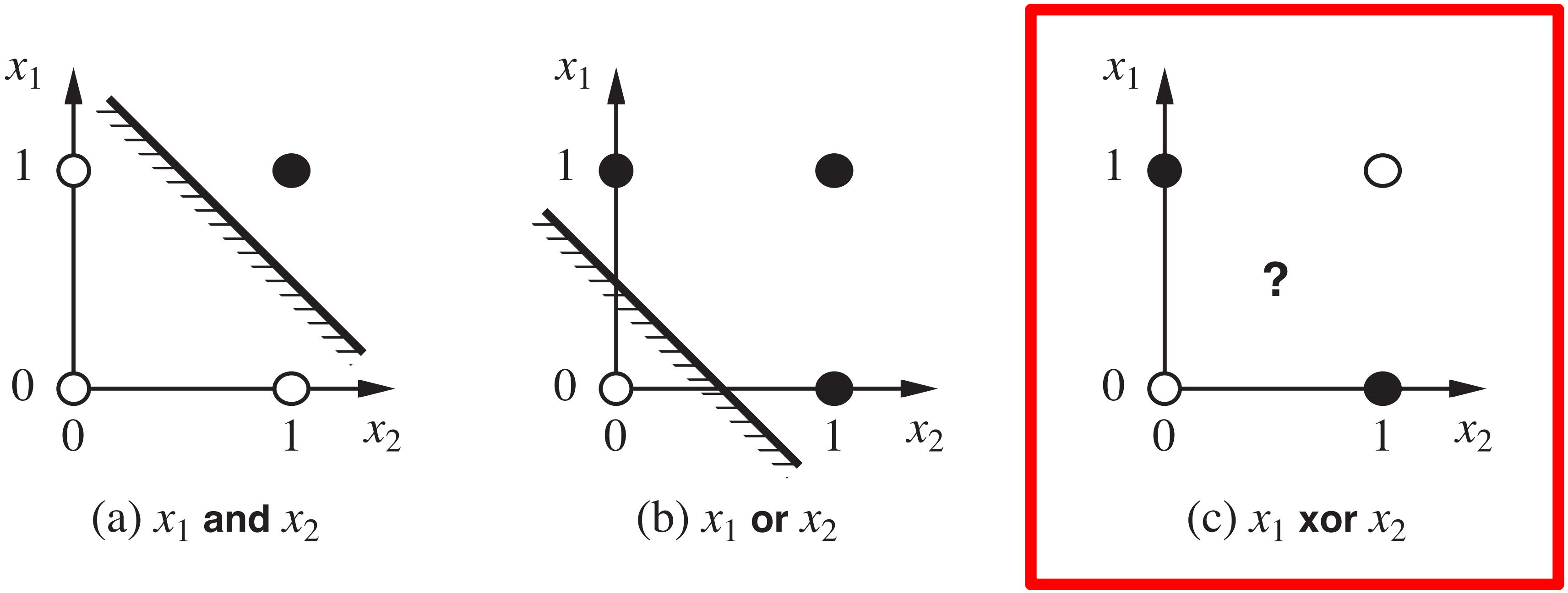
Setback in mid 60s – late 70s



- Perceptrons, Minsky and Papert, 1969
- Described serious problems with perceptron model
  - Single-layer perceptron cannot represent (learn) simple functions that are not linearly separable, such as XOR
  - Multi-layers of non-linear units may have greater power but there is no *learning rule* for such nets
  - Scaling problem: connection weights may grow infinitely
  - First two problems overcame by latter effort in 80s, but scaling problem persists
- Death of Rosenblatt (1964)
- AI focused on programming intelligent systems on traditional von Neuman computers

Not with a perceptron ☹

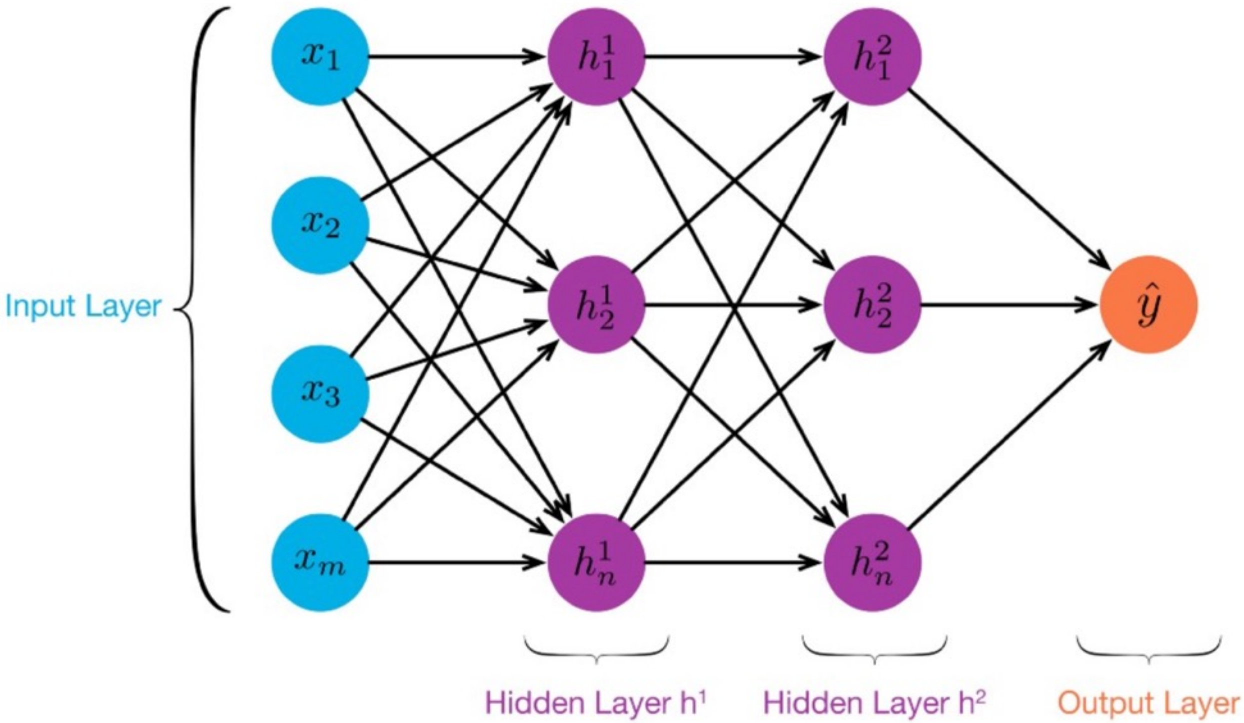
Consider Boolean operators (and, or, xor) with four possible inputs: 00 01 10 11



Training examples are **not linearly separable** for one case: *sum=1 iff x1 xor x2*

Renewed enthusiasm 1980s

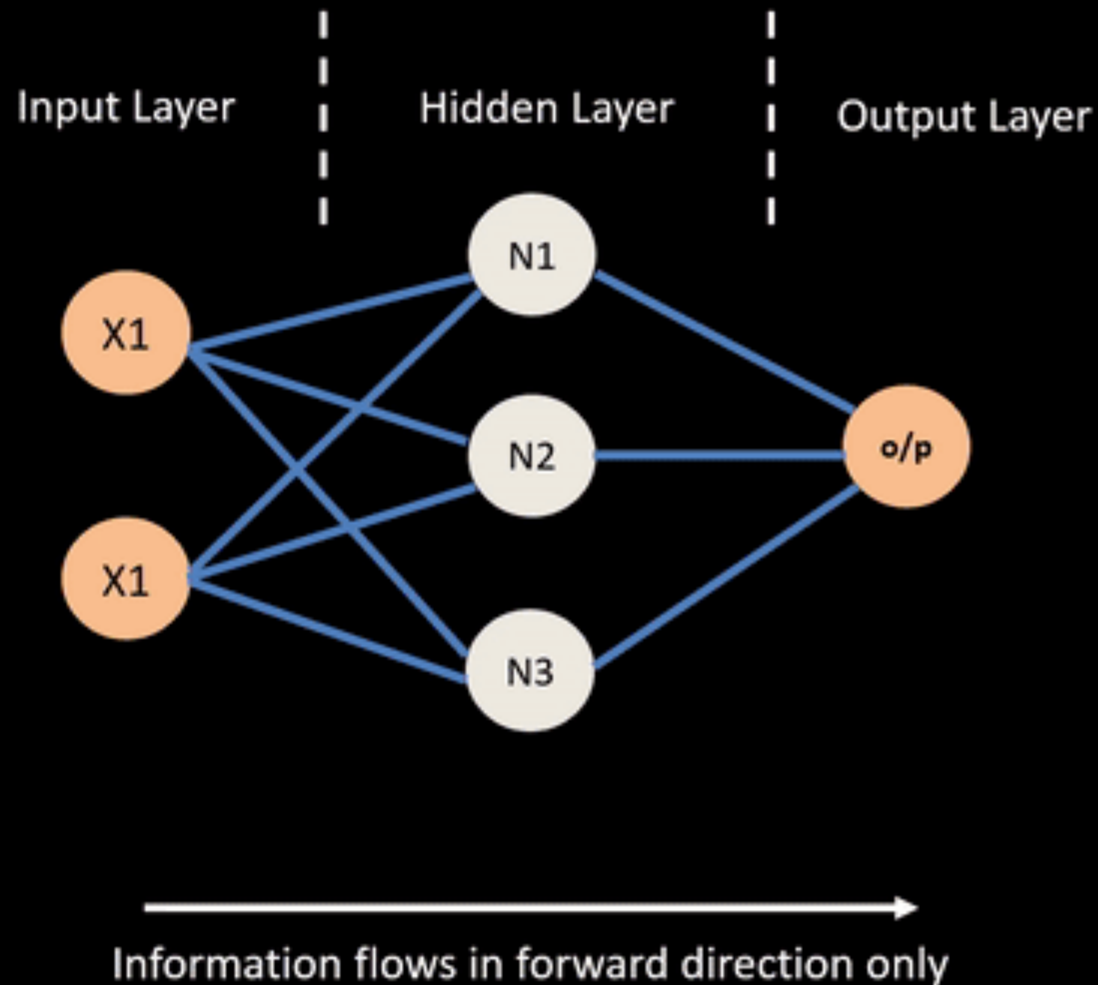
- Use **multi-layer perceptron**
- **Backpropagation** for multi-layer feed forward nets, with non-linear, differentiable node functions
  - Rumelhart, **Hinton**, Williams, Learning representations by back-propagating errors, Nature, 1986.
- Other ideas:
  - Thermodynamic models (Hopfield net, Boltzmann machine …), unsupervised learning, …
- Successful **applications** to character recognition, speech recognition, text-to-speech, etc.



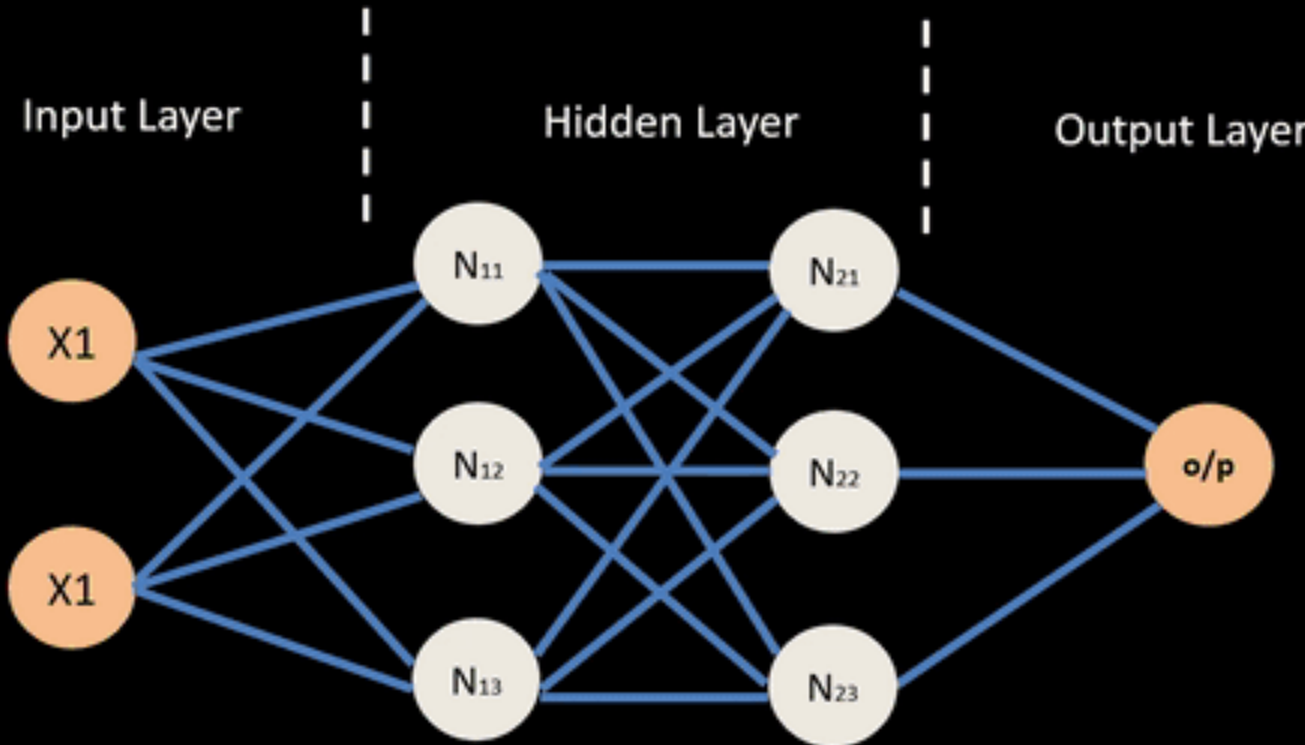
MLP: Multilayer Perceptron

- ≥ 1 "hidden layers" between inputs & output
- Can compute **non-linear functions** (why?)
- Training: adjust weights slightly to reduce error between output **y** and target value **t;** repeat
- Introduced in 1980s, still used today

Feed Forward Neural Network

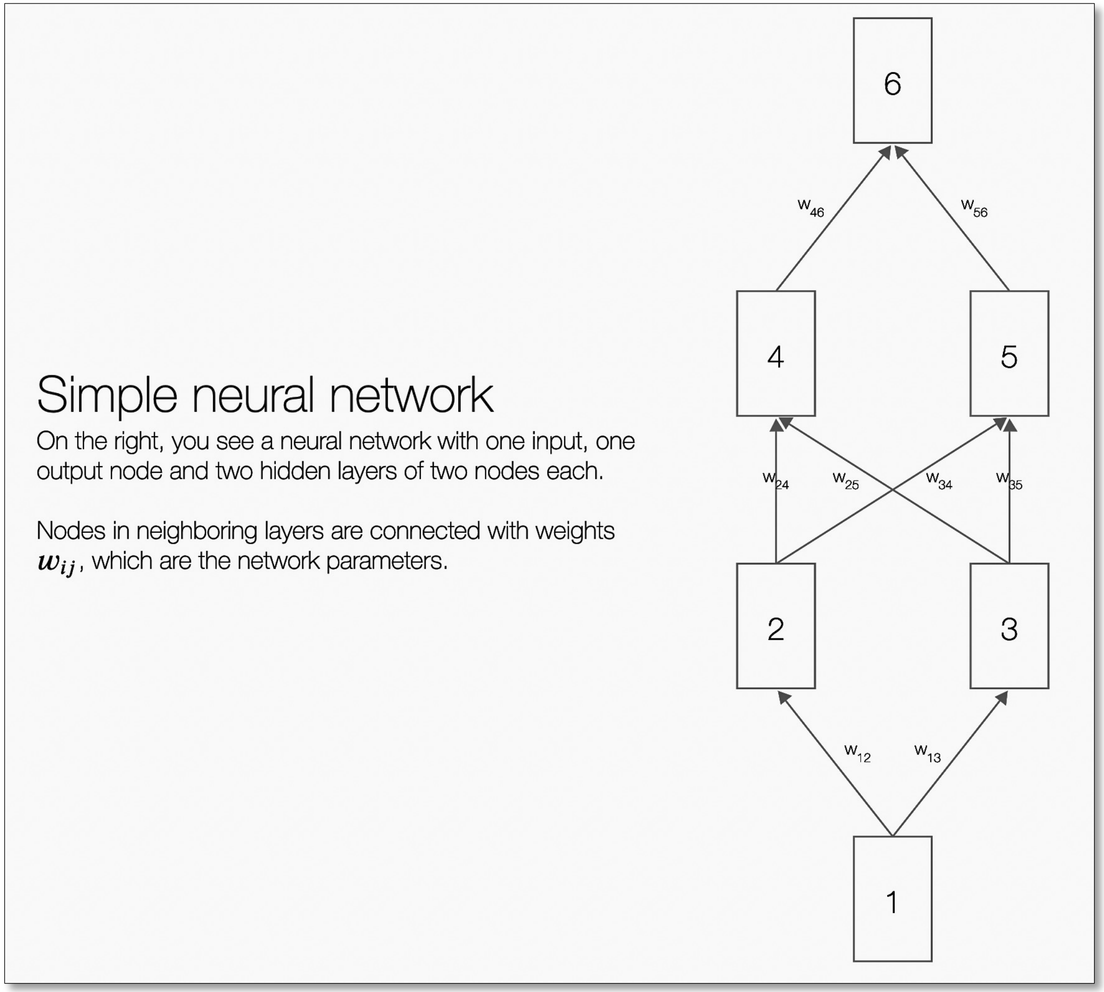


Neural Network – Backpropagation



Backpropagation Explained

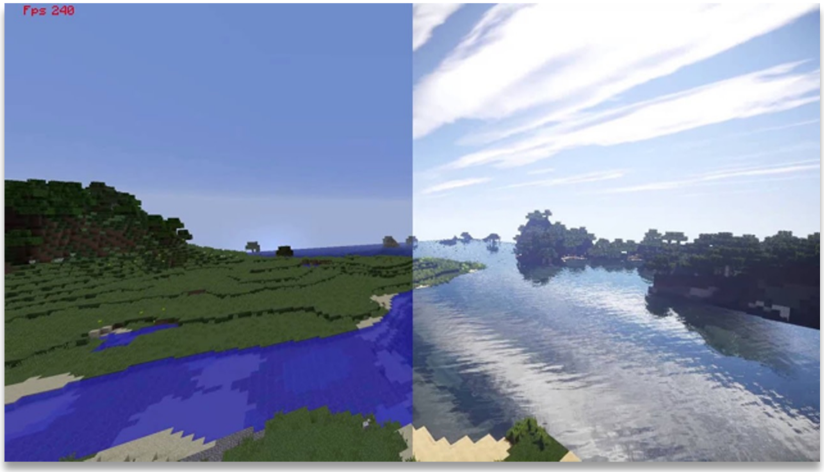
Click on image (or here) for a simple interactive demo in your browser of how backpropagation updates weights in a neural network to reduce errors when processing training data



But problems remained …

- It's often the case that solving a problem just reveals a new one that needs solving
- For a large MLPs, backpropagation takes forever to converge!
- Two issues:
  - Not enough compute power to train the model
  - Not enough labeled data to train the neural net
- SVMs may be better, since they converge to global optimum in O(n^2)

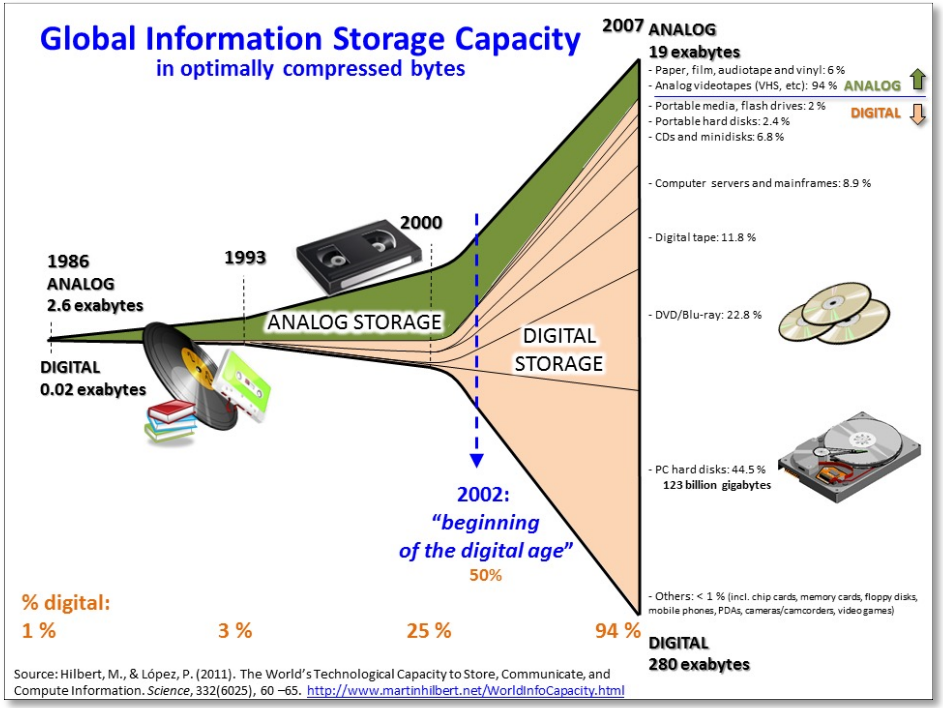
GPUs solve compute power problem



- GPUs (Graphical Processing Units) became popular in the 1990s to handle computing needed for better computer graphics
- GPUs are SIMD (single instruction, multiple data) processors
- Cheap, fast, and easy to program
- GPUs can do matrix multiplication and other matrix computations very fast

Need lots of data!

- 2000s introduced [big data]
- Cheaper storage
- Parallel processing (e.g., MapReduce, Hadoop, Spark, grid computing)
- Data sharing via the Web
  - Lots of images, many with captions
  - Lots of text, some with labels
- Crowdsourcing systems (e.g., [Mechanical Turk]) provided a way to get more human annotations

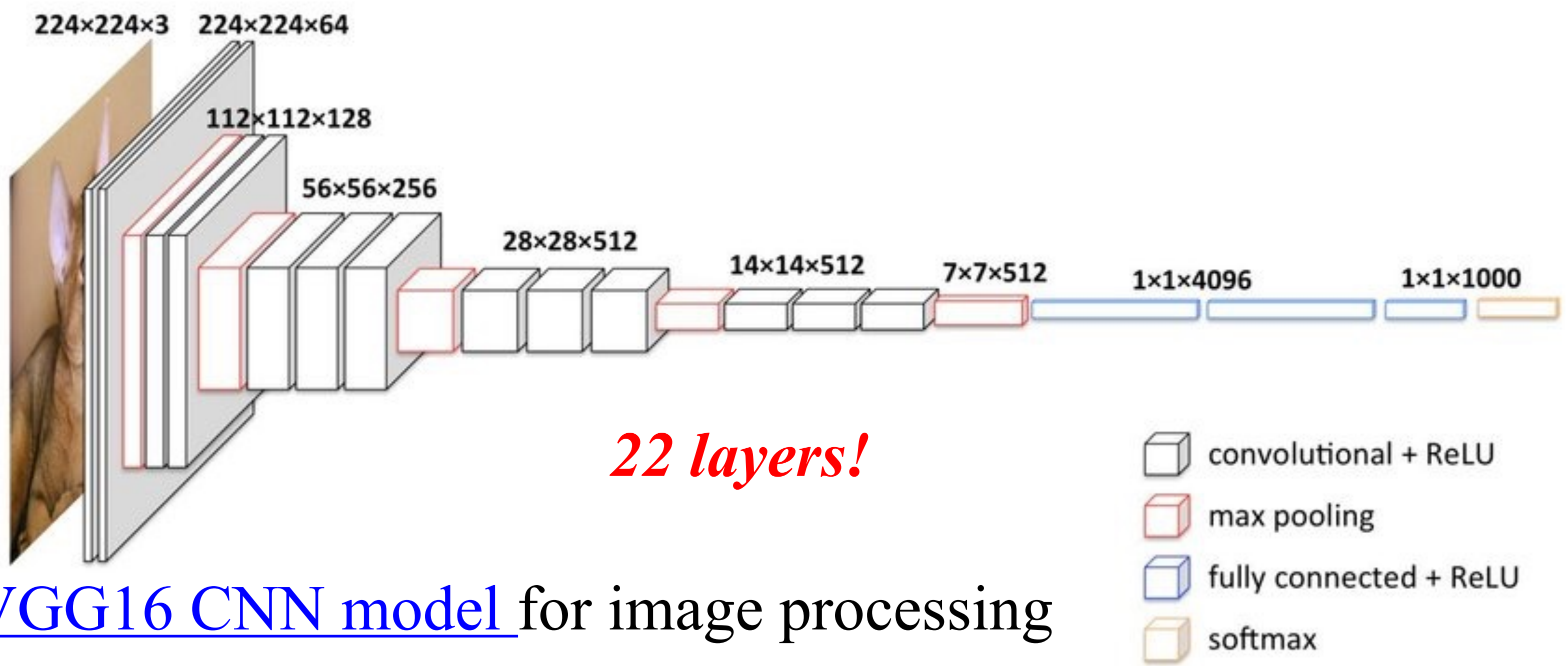


New problems are surfaced

- 2010s was a decade of domain applications
- These came with new problems, e.g.,
  - Images are too highly dimensioned!
  - Variable-length problems cause gradient problems
  - Training data is rarely labeled
  - Neural nets are uninterpretable
  - Training complex models required days or weeks
- This led to many new “deep learning” neural network models

Deep Learning

- Deep learning refers to models going beyond simple feed-forward multi-level perceptron
  - Though it was used in a ML context as early as 1986
- "deep" refers to the models having many layers (e.g., 10-20) that do different things



22 layers!

The VGG16 CNN model for image processing

Neural Network Architectures

Current focus on large networks with different "architectures" suited for different tasks

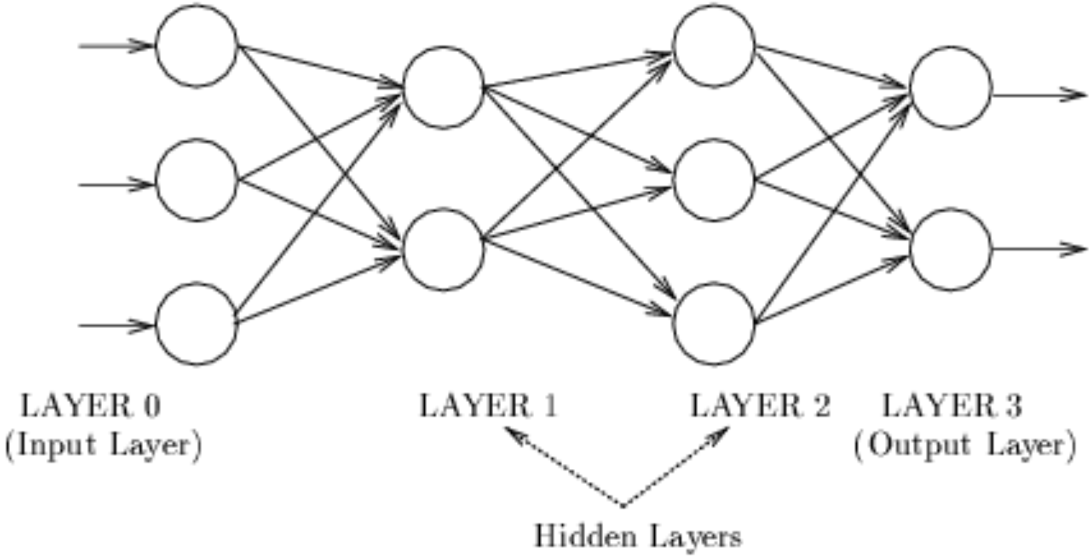
- Feedforward Neural Network
- CNN: **C**onvolutional **N**eural **N**etwork
- RNN: **R**ecurrent **N**eural **N**etwork
- LSTM: **L**ong Short **T**erm **M**emory
- GAN: **G**enerative **A**dversarial **N**etwork
- Transformers: generating output sequence from input sequence

Feedforward Neural Network

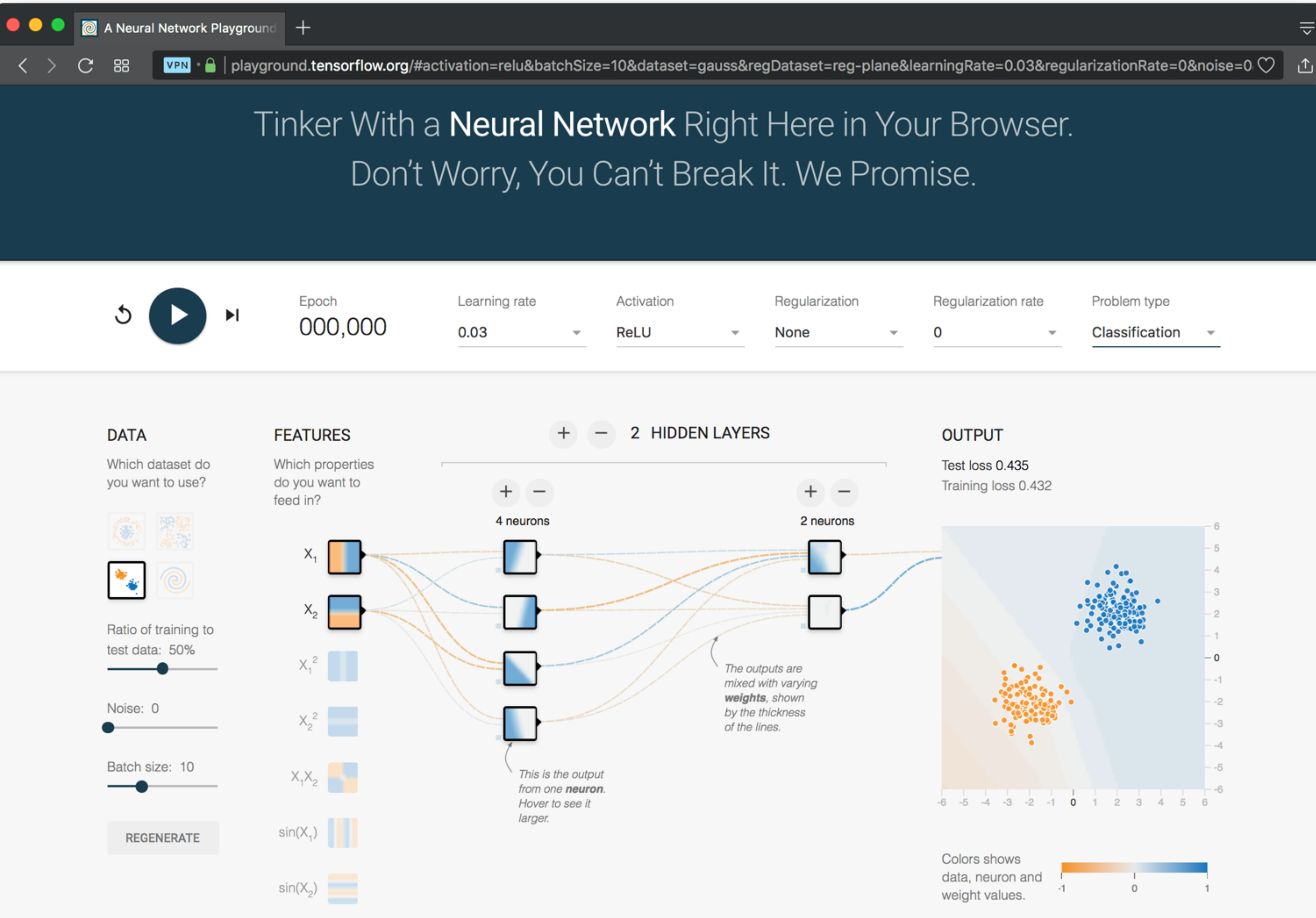
- Connections allowed from a node in layer *i* only to nodes in layer *i*+1

  i.e., no cycles or loops

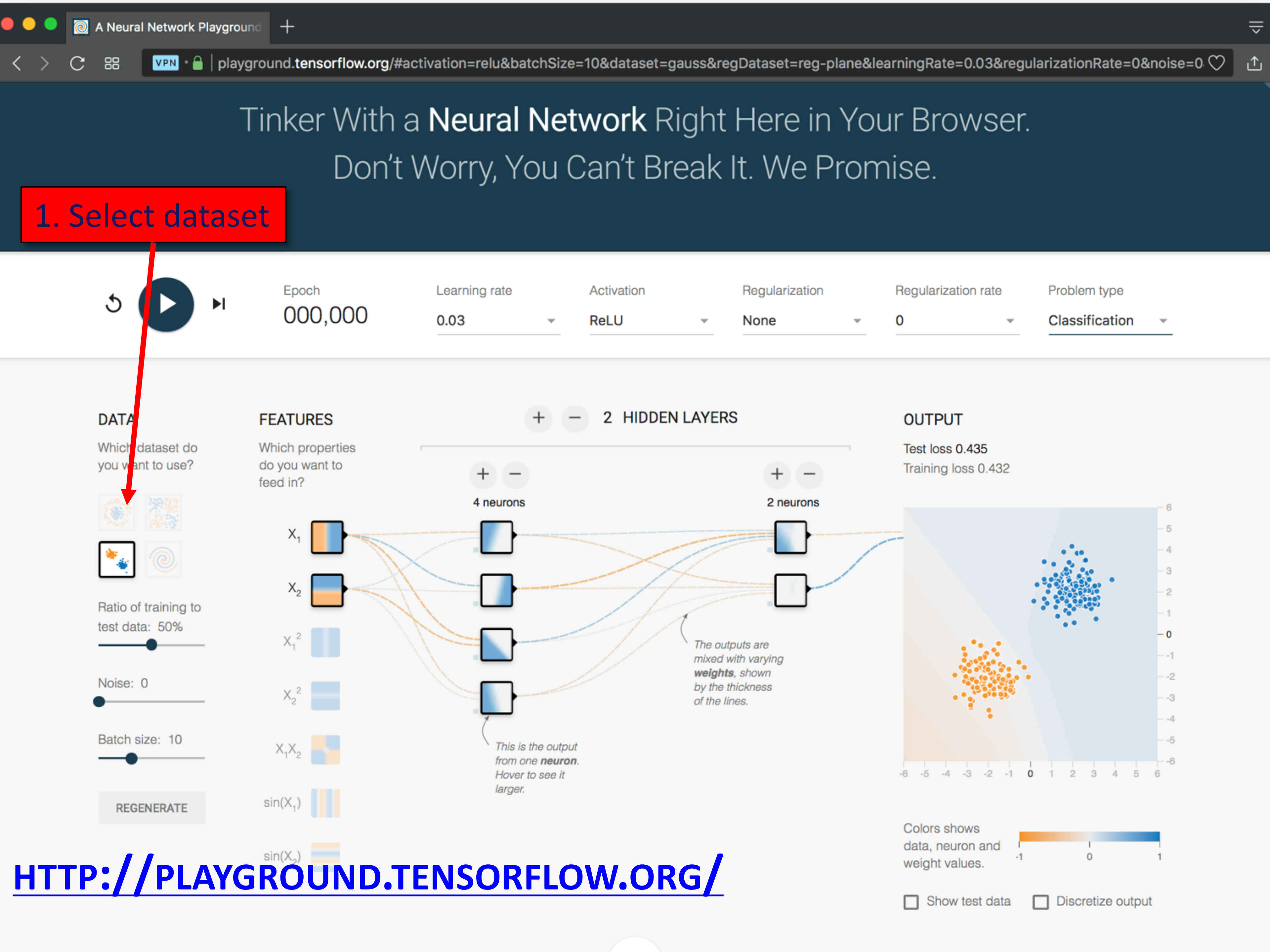
- Simple, widely used architecture, provides a good baseline



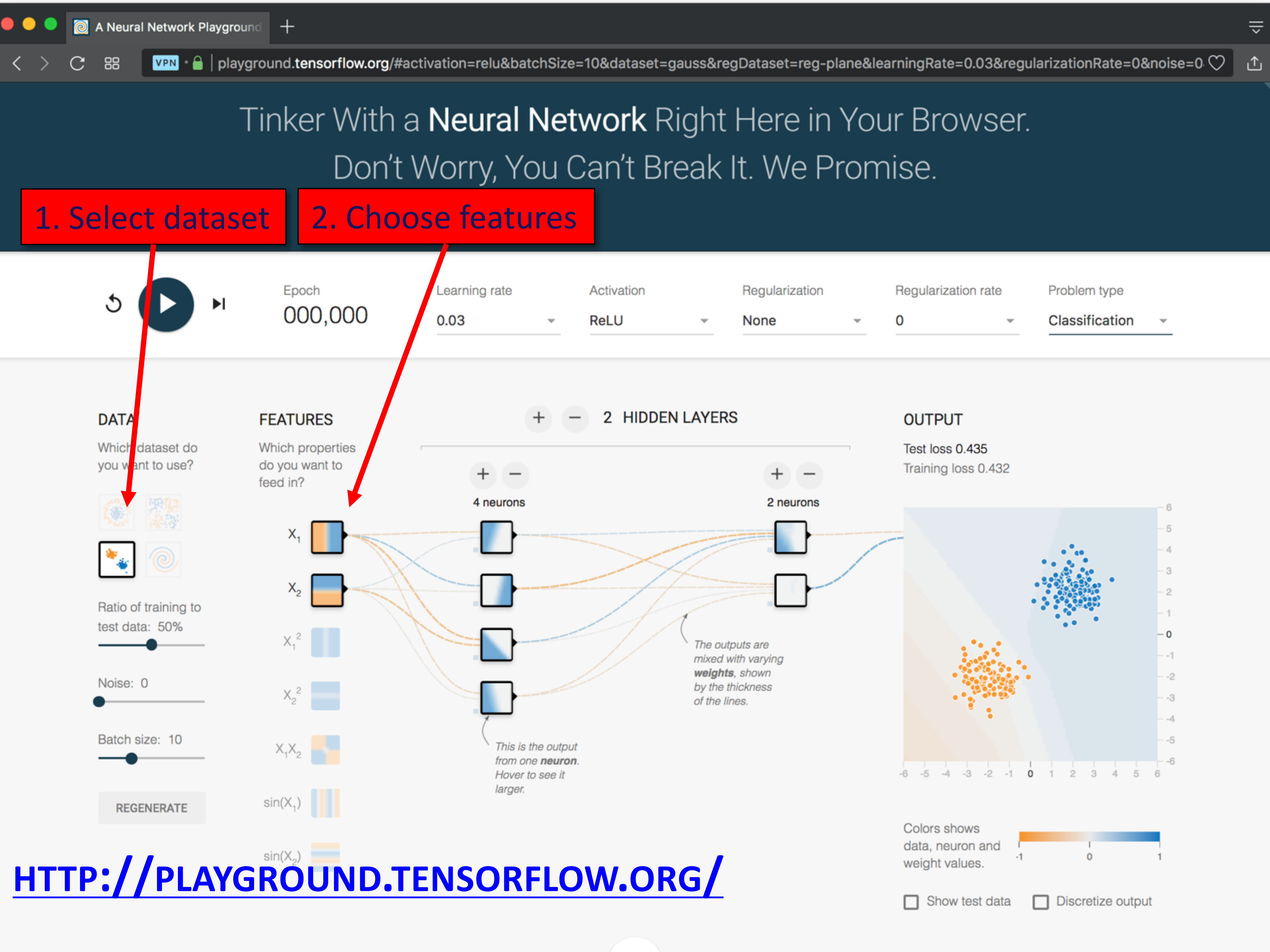
downstream nodes tend to successively abstract features from preceding layers



[HTTP://PLAYGROUND.TENSORFLOW.ORG/](http://playground.tensorflow.org/)

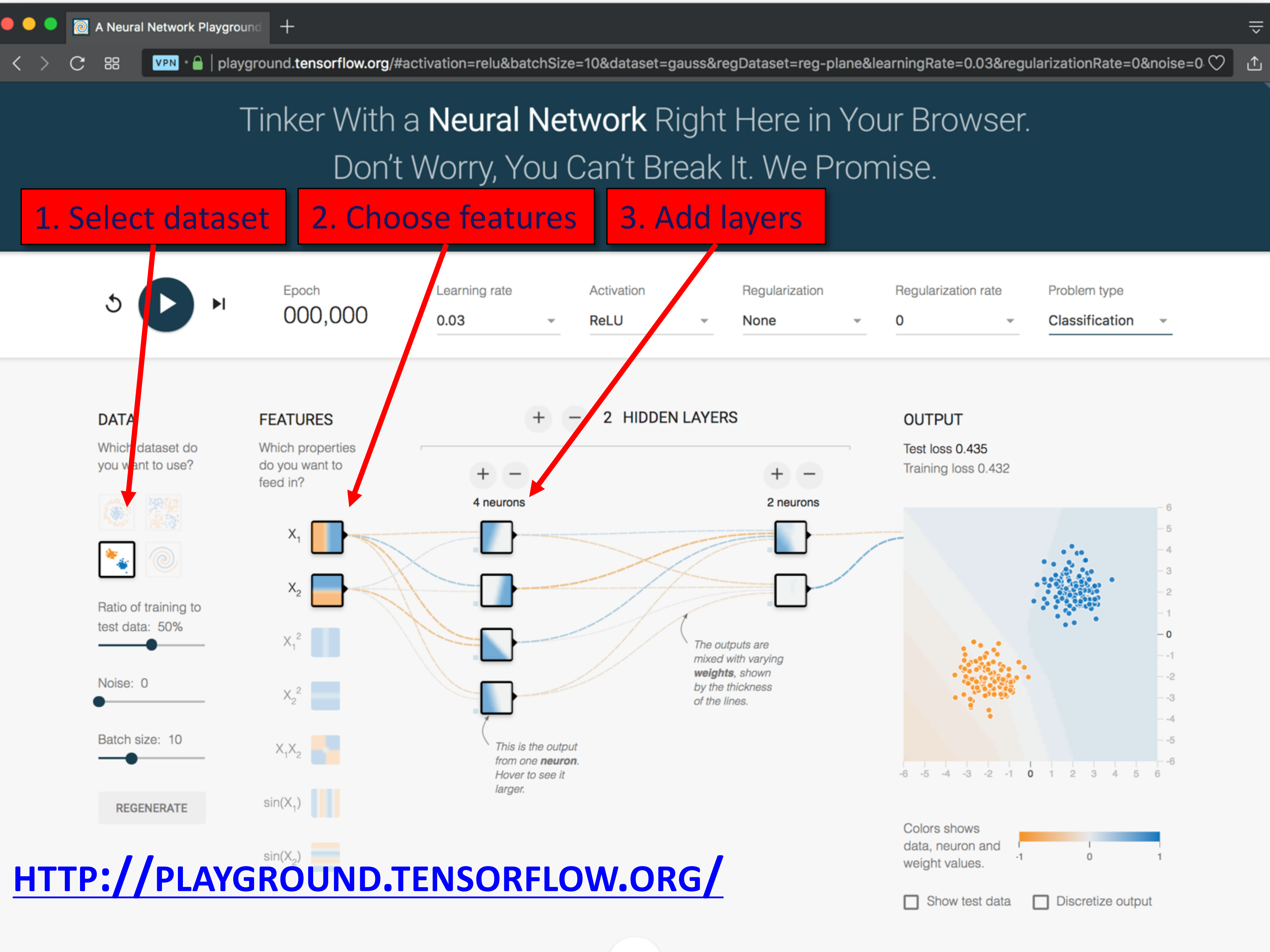


[HTTP://PLAYGROUND.TENSORFLOW.ORG/](http://playground.tensorflow.org/)

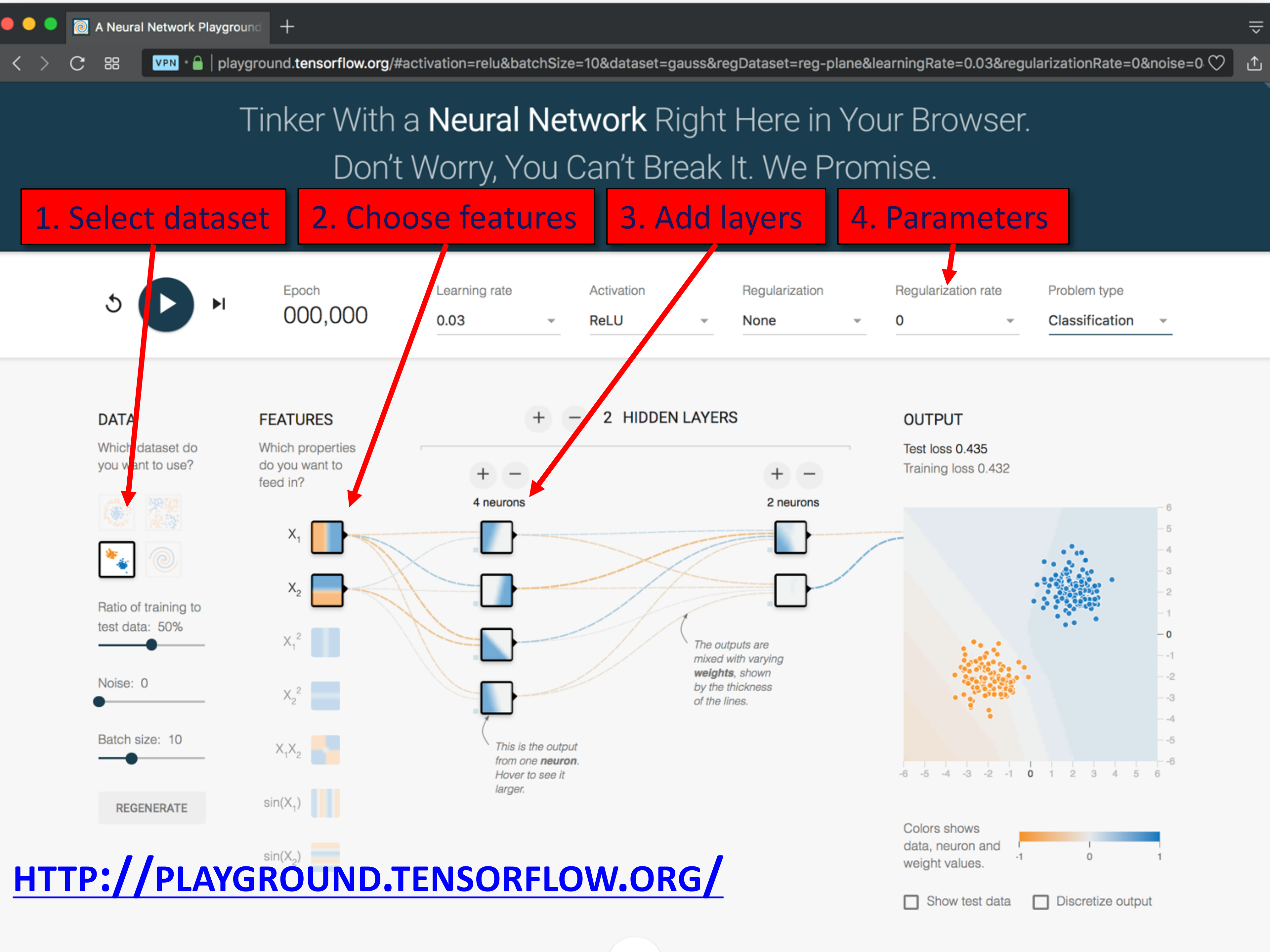


[HTTP://PLAYGROUND.TENSORFLOW.ORG/](http://playground.tensorflow.org/)

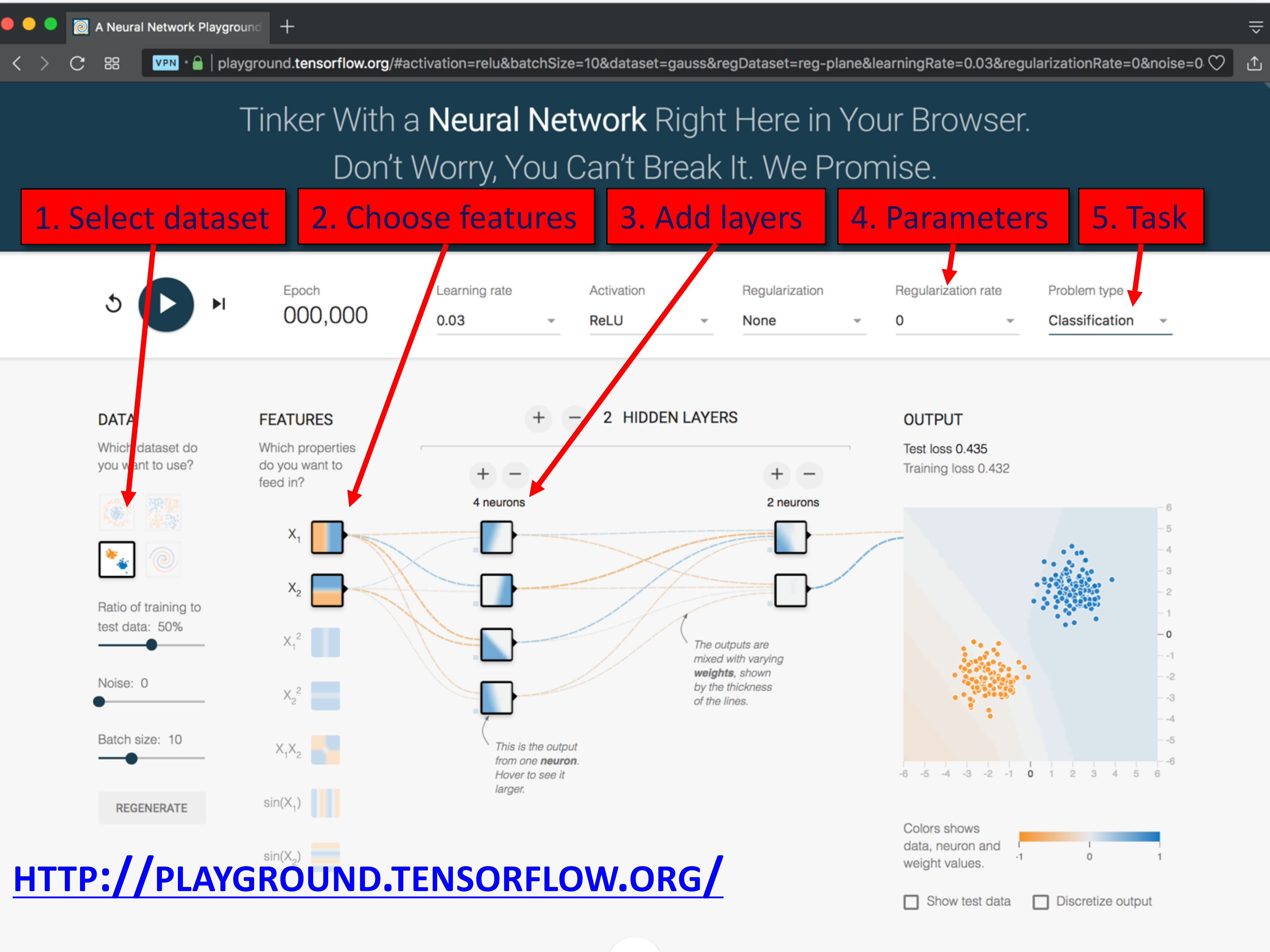
1. Select dataset
2. Choose features
3. Add layers



HTTP://PLAYGROUND.TENSORFLOW.ORG/

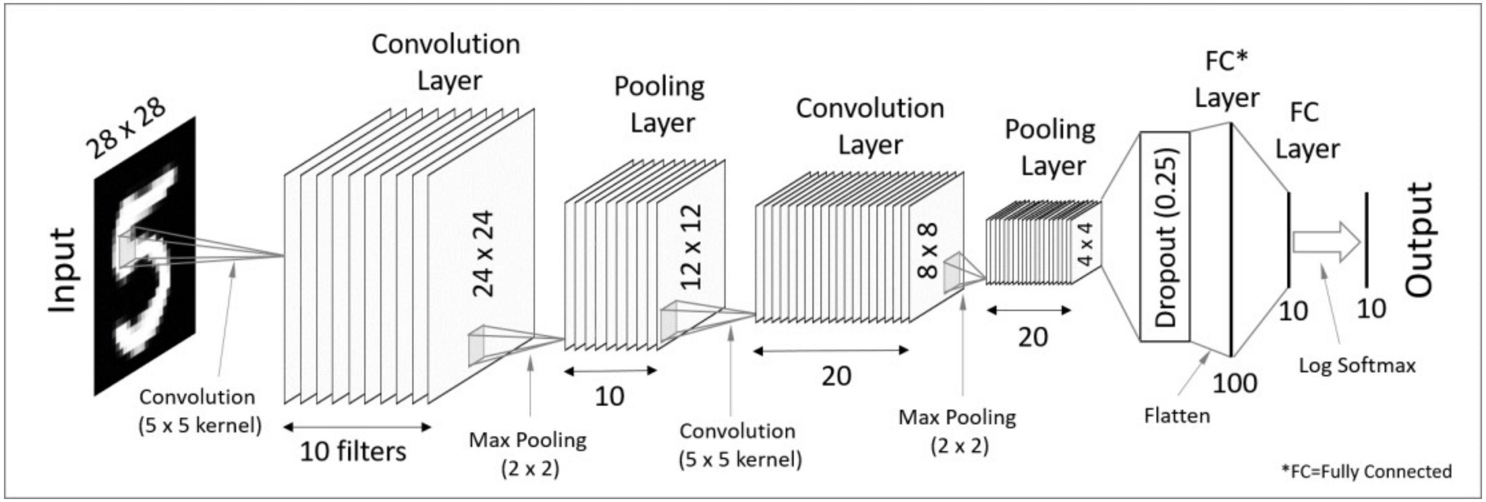


HTTP://PLAYGROUND.TENSORFLOW.ORG/



[HTTP://PLAYGROUND.TENSORFLOW.ORG/](http://playground.tensorflow.org/)

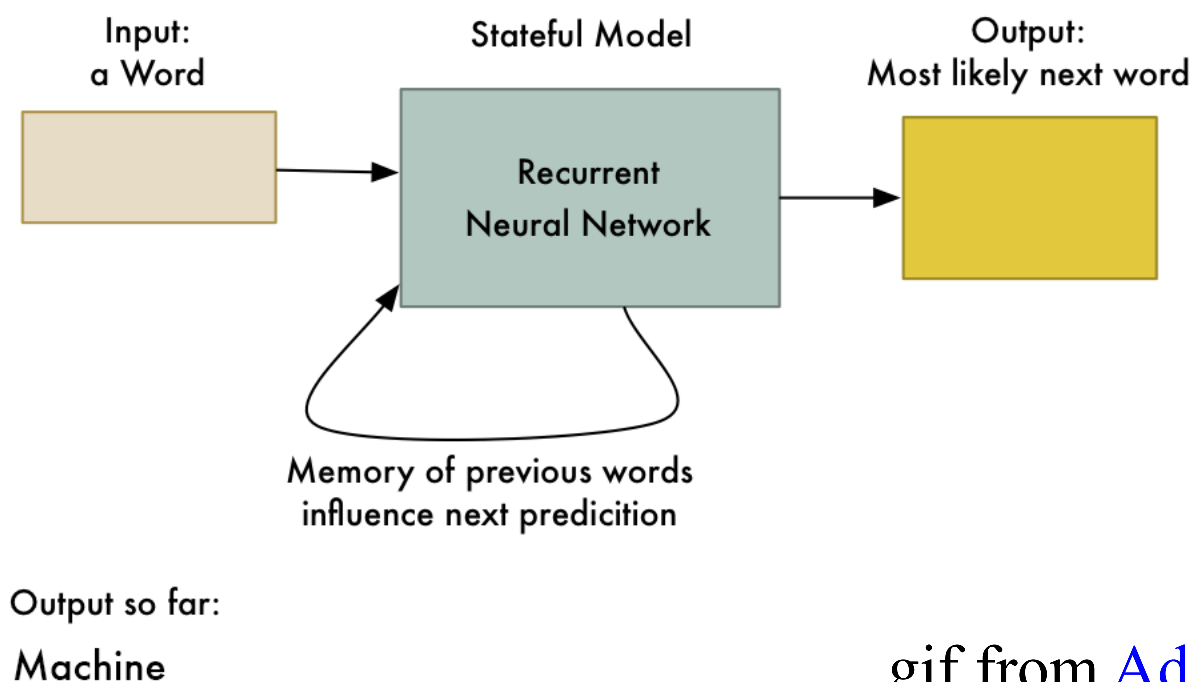
CNN: Convolutional Neural Network



- Good for 2D image processing: classification, object recognition, automobile lane tracking, etc.
- Successive convolution layers learn higher-level features
- Classic demo: learn to recognize hand-written digits from MNIST data with 70K examples

RNN: Recurrent Neural Networks

- Good for learning over sequences of data, e.g., a sentence of words
- LSTM (Long Short Term Memory) a popular architecture



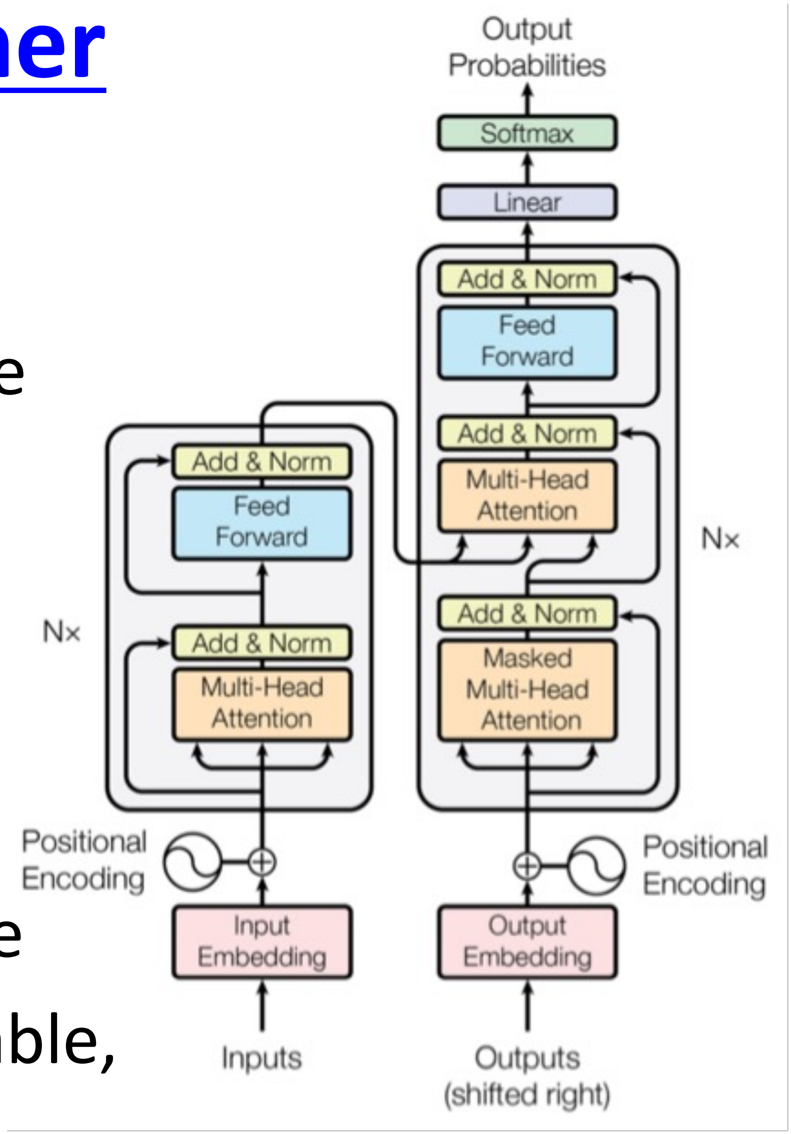
gif from [Adam Geitgey]

GAN: Generative Adversarial Network

- System of **two neural networks** competing against each other in a zero-sum game framework
- Provides a kind of **unsupervised learning** that improves the network
- Introduced by Ian Goodfellow et al. in 2014
- Can learn to draw samples from a model that is similar to data that we give them

Transformer

- Introduced in 2017
- Used primarily for natural language processing tasks
- NLP applications ”transform” an input text into an output text
  - E.g., translation, text summarization, question answering
- Uses encoder-decoder architecture
- Popular pre-trainted models available, e.g. BERT and GPT



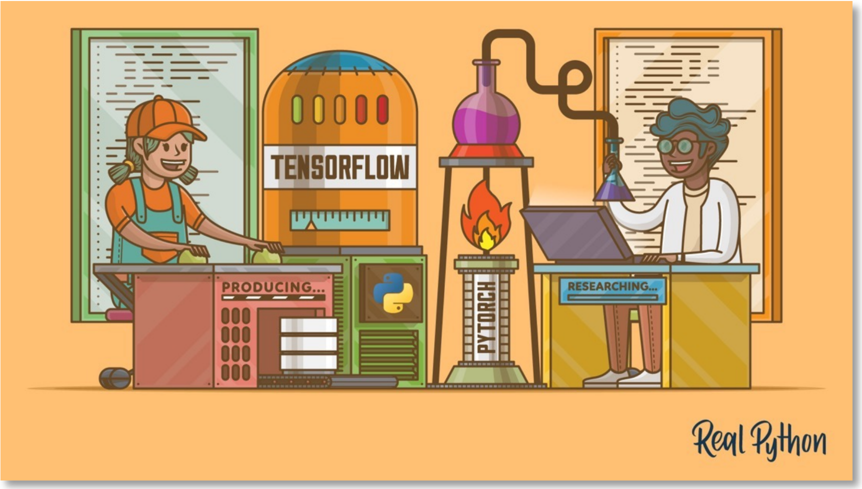
Deep Learning Frameworks (1)

- Popular open-source deep learning frame-works use Python at top-level; C++ in backend
  - TensorFlow (via Google)
  - Keras (Open Source, now TensorFlow's I/F)
  - PyTorch (via Facebook)
  - MxNet (Apache)
  - Caffe (Berkeley)
- TensorFlow and PyTorch now dominate; both make it easy to specify a complicated network

TensorFlow vs PyTorch

- TensorFlow is from Google, PyTorch from Apple
- Both make it each to define and use a neural network structure in Python
- TensorFlow used to dominate, but now PyTorch has become more popular

PyTorch vs TensorFlow for Your Python Deep Learning Project

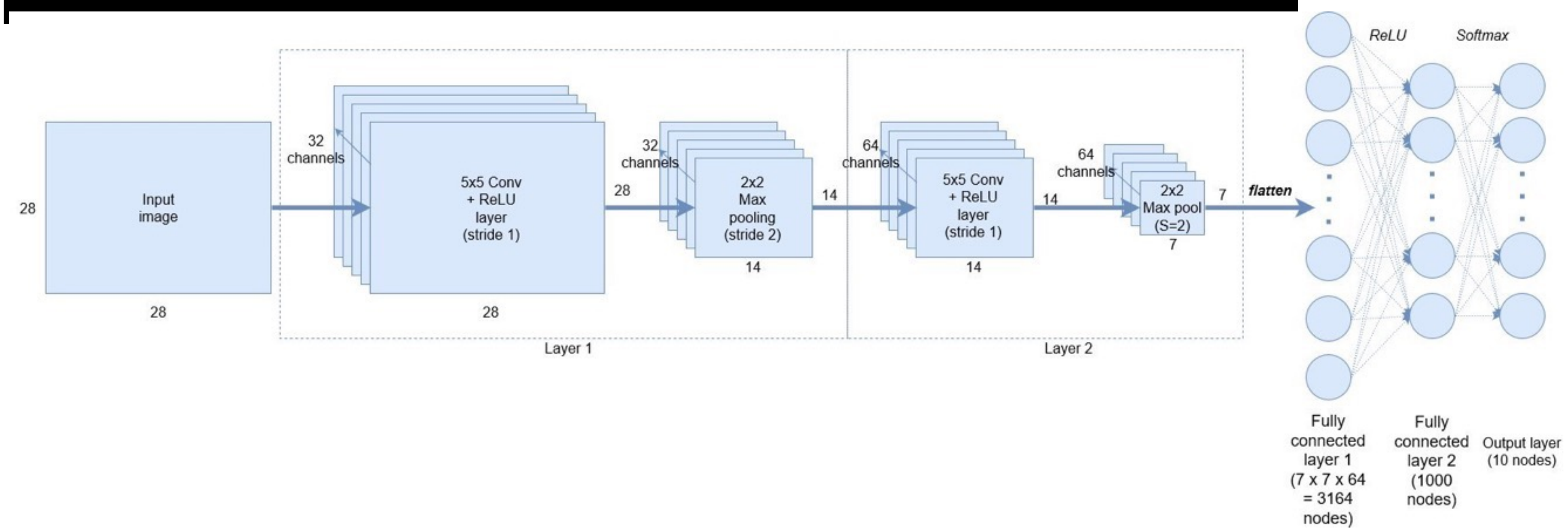


Keras

- “Deep learning for humans”
- A popular API works with TensorFlow provides good support at architecture level
- Keras (v2.4 +) only supports TensorFlow
- Supports CNNs and RNNs and common utility layers like dropout, batch normalization and pooling
- Coding neural networks used to be harder; Keras made it easier and more accessible
- Documentation: https://keras.io/

Keras: API works with TensorFlow

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

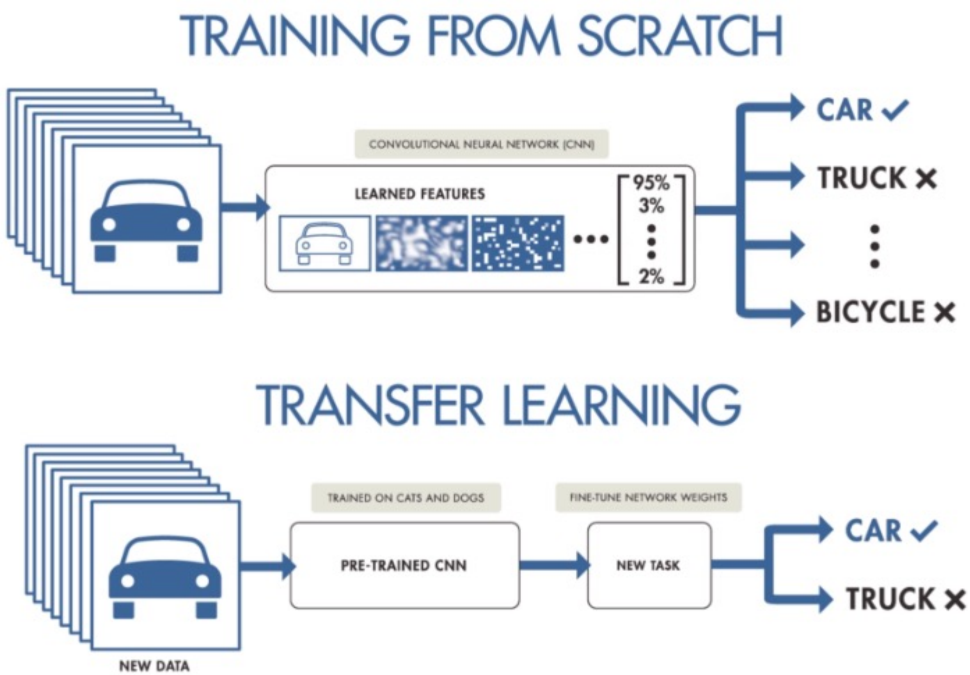


NNs Good at Transfer Learning

- Neural networks effective for [transfer learning]

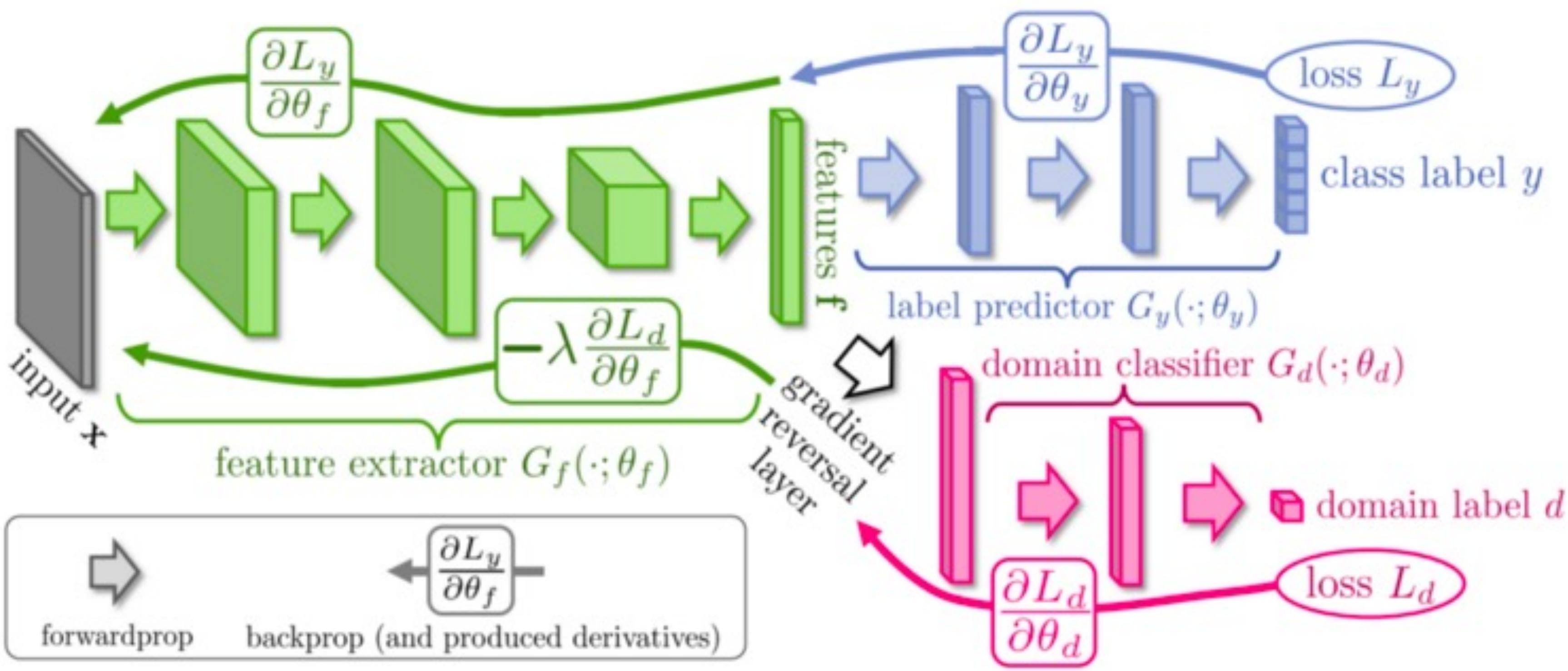
  Using parts of a model trained on a task as an initial model to train on a different task

- Particularly effective for image recognition



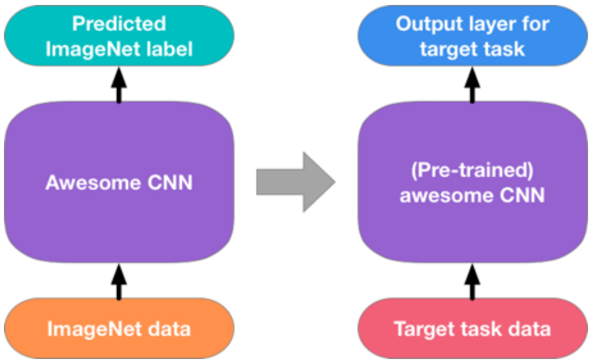
Good at Transfer Learning

- For images, the initial stages of a model learn high-level visual features (lines, edges) from pixels
- Final stages predict task-specific labels



source:http://ruder.io/transfer-learning/

Fine Tuning a NN Model



- Special kind of transfer learning
  - Start with a pre-trained model
  - Replace last output layer with a new one
  - One option: Fix all but last layer by marking as trainable:false
- Retraining on new task and data very fast
  - Only the weights for the last layer are adjusted
- Example
  - Start: NN to classify animal pix with 100s of categories
  - Finetune on new task: classify pix of 10 common pets

Conclusions

- Quick intro to neural networks & deep learning
- Learn more by
  - Take UMBC's CMSC 478 machine learning class
  - Try scikit-learn's neural network models
  - Explore Keras as : https://keras.io/
  - Explore Google's Machine Learning Crash Course
  - Work through examples
- and then try your own project idea