Neural Networks

Navy FCU Day 3

Plan of the Day

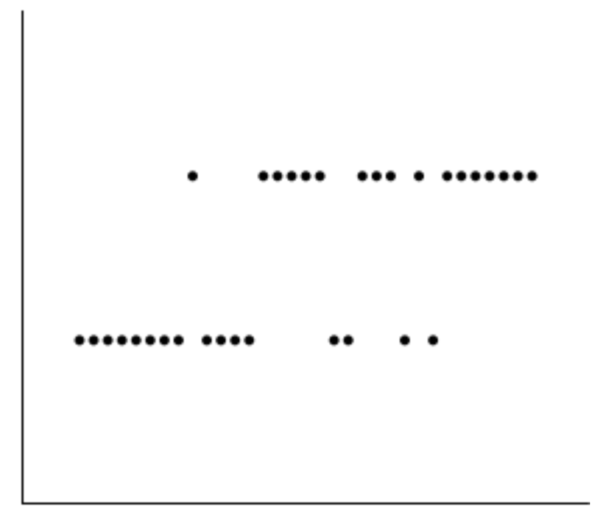
- Logistic Regression
- Artificial Neural Network
  - Feed Forward
  - Back Propagation
- Softmax Regression vs. ANNs
- A Brief History of ANNs
- Deep Learning Taxonomy
- Distributed Computing Approaches
- Business Cases for Deep Learning

Logistic Regression

Aka logit-regression, maximum entropy (MaxEnt), or log-linear classifier

Binary Output Variable

Given a binary classification target we can see that it is very difficult to establish a relationship between X and Y.

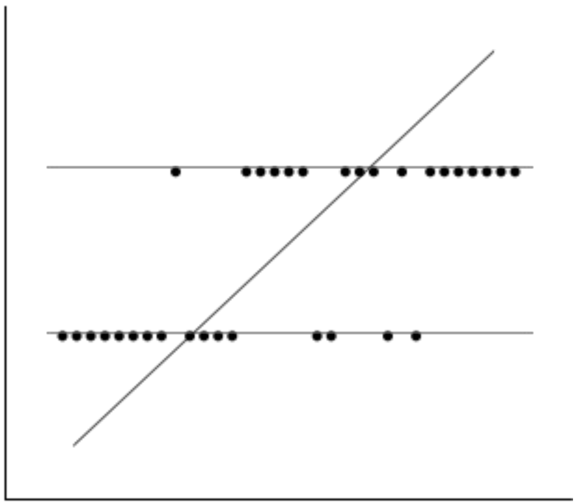


Binary Output Variable

Given a binary classification target we can see that it is very difficult to establish a relationship between X and Y.

Fitting a linear relationship on top of this model severely oversimplifies the relationship.

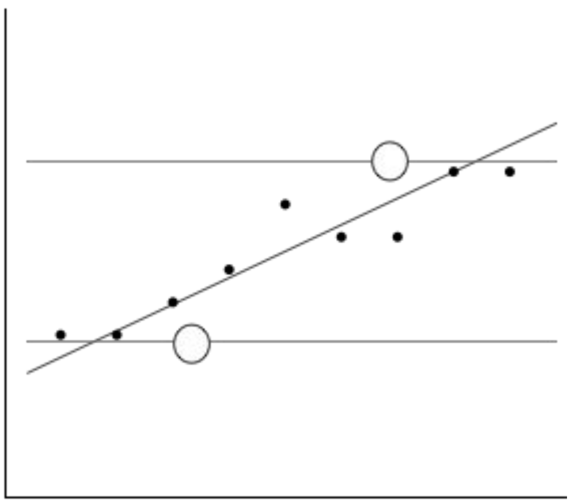
Provides unobservable predictions for small and large values of X.



Binomial Transformation

The mean of binomial variable is a proportion, so we can re-encode our data as conditional probabilities and fit our model to it: e.g. learn:

`p(y | x)`



However the linear model does not predict the **maximum likelihood estimates** (shown by the circles).

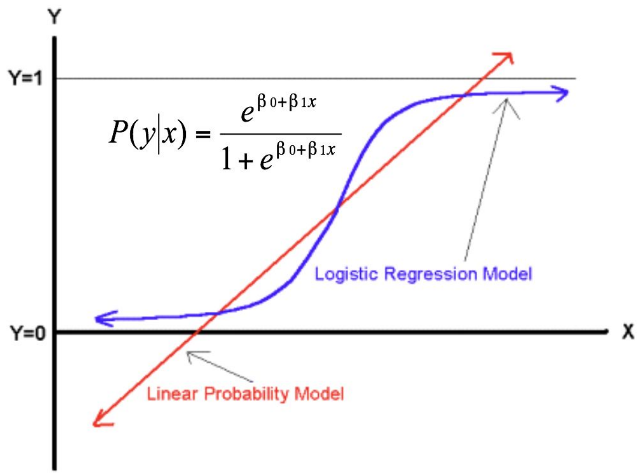
The regression is sigmoidal.

Logistic Regression

Recall we can model non-linear models via a transformation of our dataset, X.

Two possible transformations that result in sigmoidal functions:

- **Probit**: imposes cumulative normal function on the dataset X. Unfortunately, they are difficult to work with (requires integral solver).
- **Logit:** gives nearly identical values to probit, but can be modeled more easily with a linear equation.



Logit: Logarithm of Odds (Log Odds → Logit?)

Odds: range of 0 to ∞: if odds > 1 then the event is more likely to occur than not occur. If odds < 1 vent is less likely to occur than not occur.

$$odds = \frac{p}{1-p}$$

The log odds transformation creates a variable with a range from -∞ to +∞ and solves the problem with fitting the linear model to probabilities.

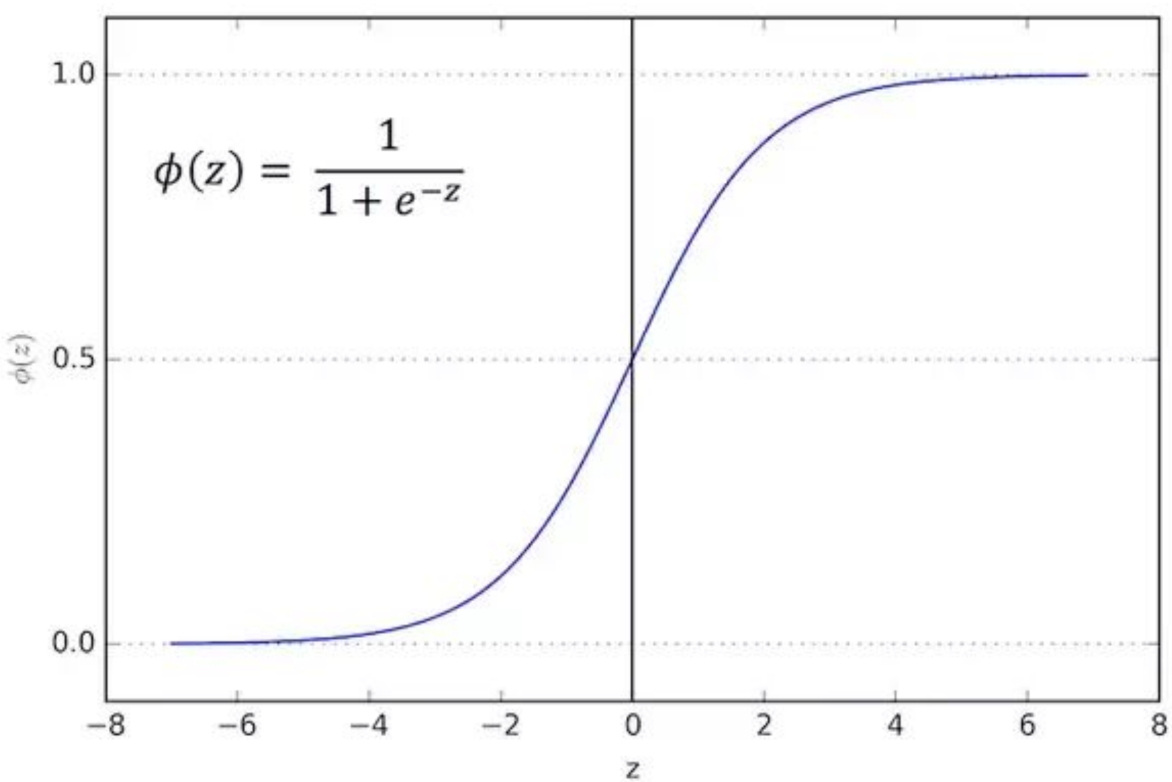
$$\ln(odds) = \ln\left(\frac{p}{1-p}\right) = \ln(p) - \ln(1-p)$$

The linear model can now capture all logits outside the range 0 to 1 and is not underfit.

In addition, if you take the exponent of the logit, you have the odds for the two groups, so the predicted logit can be transformed back to probability.

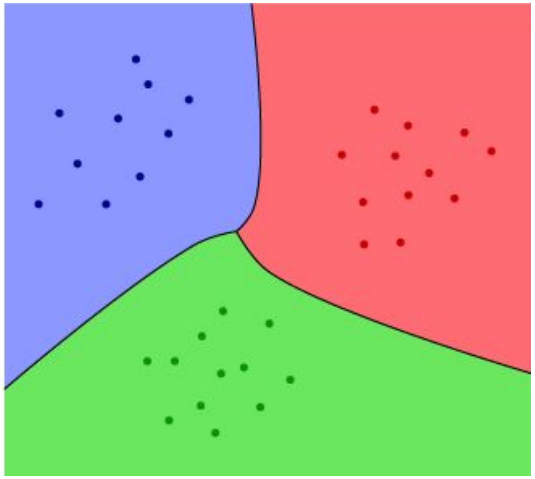
Logistic Regression

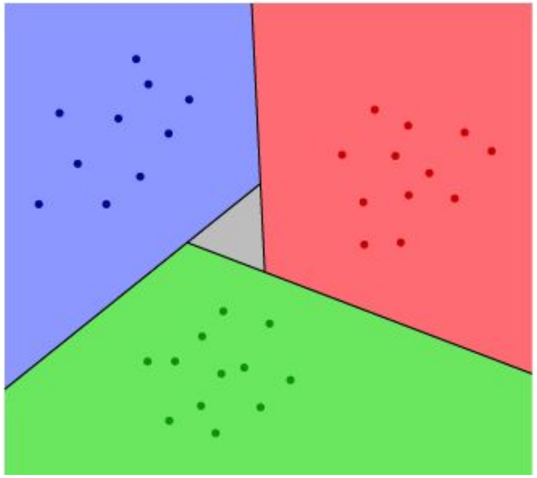
- Sensitive to multicollinearity - use regularization to handle.
- Sensitive to outliers, predictors are standardized then treated with z-score outlier removal.
- R^2 values can be artificially high or low, generally tested with F1 score.



Binary vs. Multi-Class Logistic Regression

- Multiclass Logistic Regression can be computed with a multinomial logistic regression that solves the set of all binary logits. The probabilities are computed by taking the **softmax** of the odds.
- One vs. Rest strategy computes a logistic regression for A vs. not A, B vs. not B and selects the most likely estimate.





Logistic Regression in Scikit-Learn

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split as tts


X = data.features
y = data.target

X_train, y_train, X_test, y_test = tts(X, y, test_size=0.2)

penalty = l2 # norm to use for penalization
fit_int = True # whether to add bias/intercept to decision function
slvr = liblinear # algorithm to use for optimization

model = LogisticRegression(
    penalty=penalty, fit_intercept=fit_int, solver=slvr
)

model.fit(X_train, y_train)
model.predict(X_test)
```

Artificial Neural Networks

What is an Artificial Neural Network?

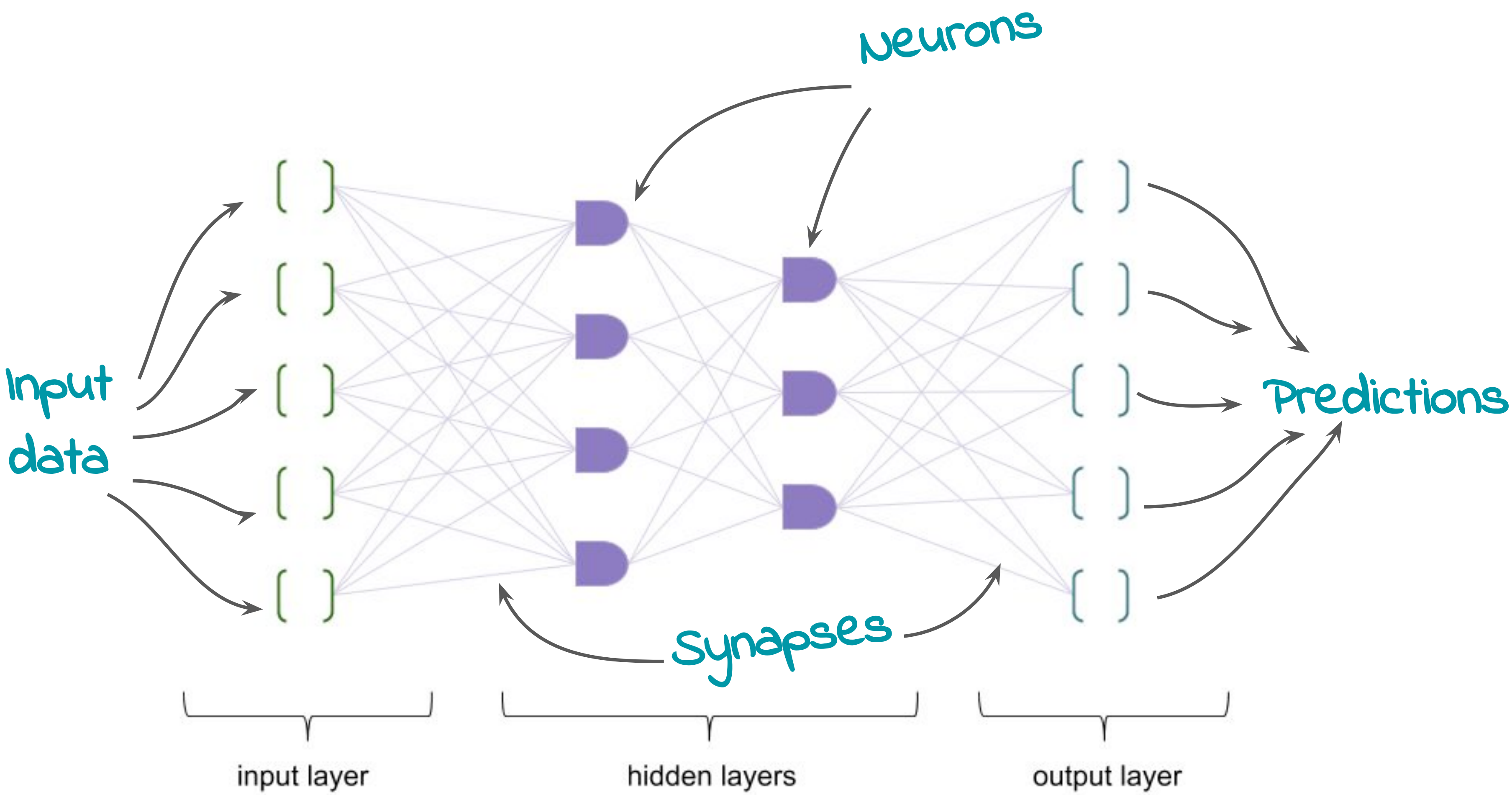
- Arbitrary set of inputs → arbitrary set of outputs
- Hidden layer of weight functions (neurons) that map a lot of other functions (a series of simple weighted functions)

Denoted by # neurons per layer

Components:

- Input Layer
- Hidden Layer
- Neurons
- Output Layer
- Training Algorithm

ANN Architecture

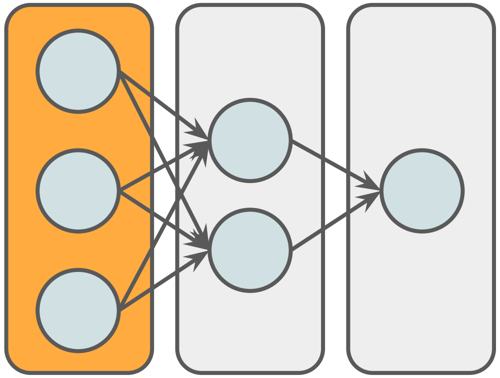


Input Layer

No neurons, just a conduit to the hidden layer.

Inputs can be:

- standard: value is between 0 and 1
- symmetric: values between -1 and 1



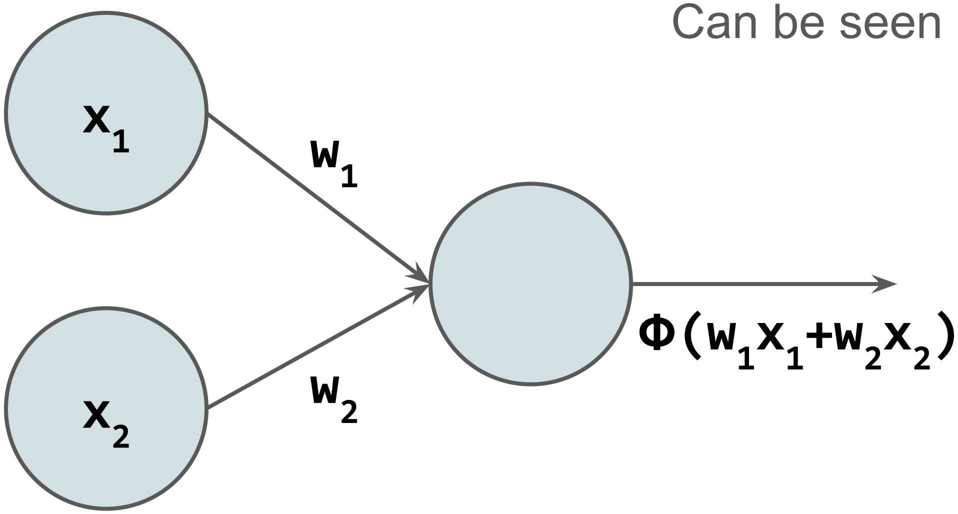
Can also be seen as a vector of inputs values.

One-hot encoding for language models.

Hidden Layer: Synapses and Neurons

Neurons are weighted linear combinations wrapped in an activation function, **Φ**.

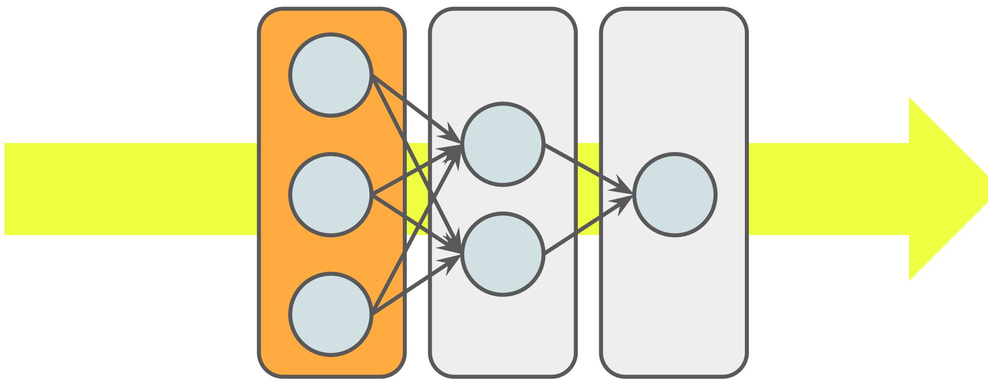
Can be seen as aggregating previous inputs.



Activation functions normalize data, and must be differentiable for training

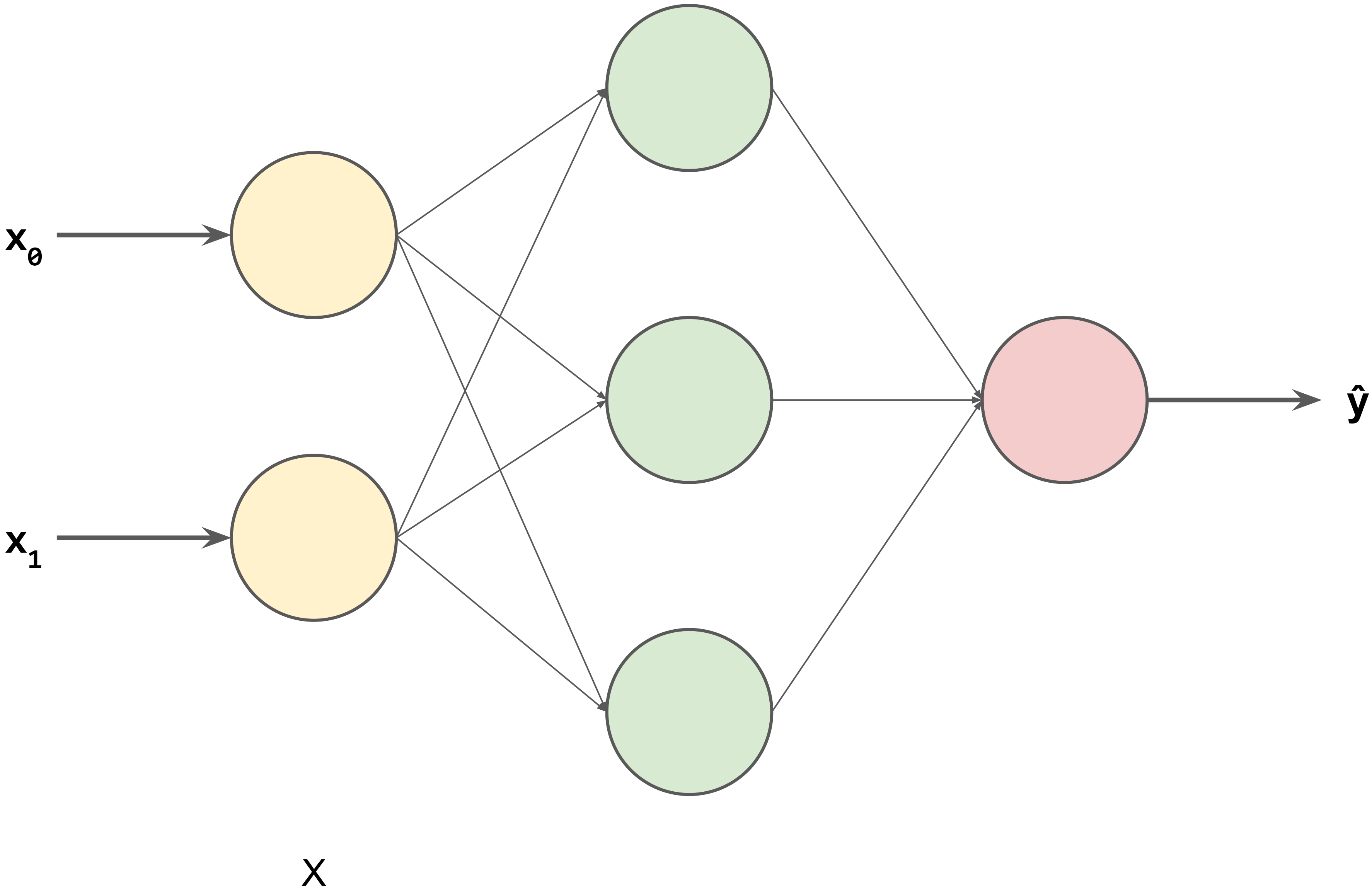
Feedforward

In a feedforward network, signals travel from the input to the output layer in a single direction.

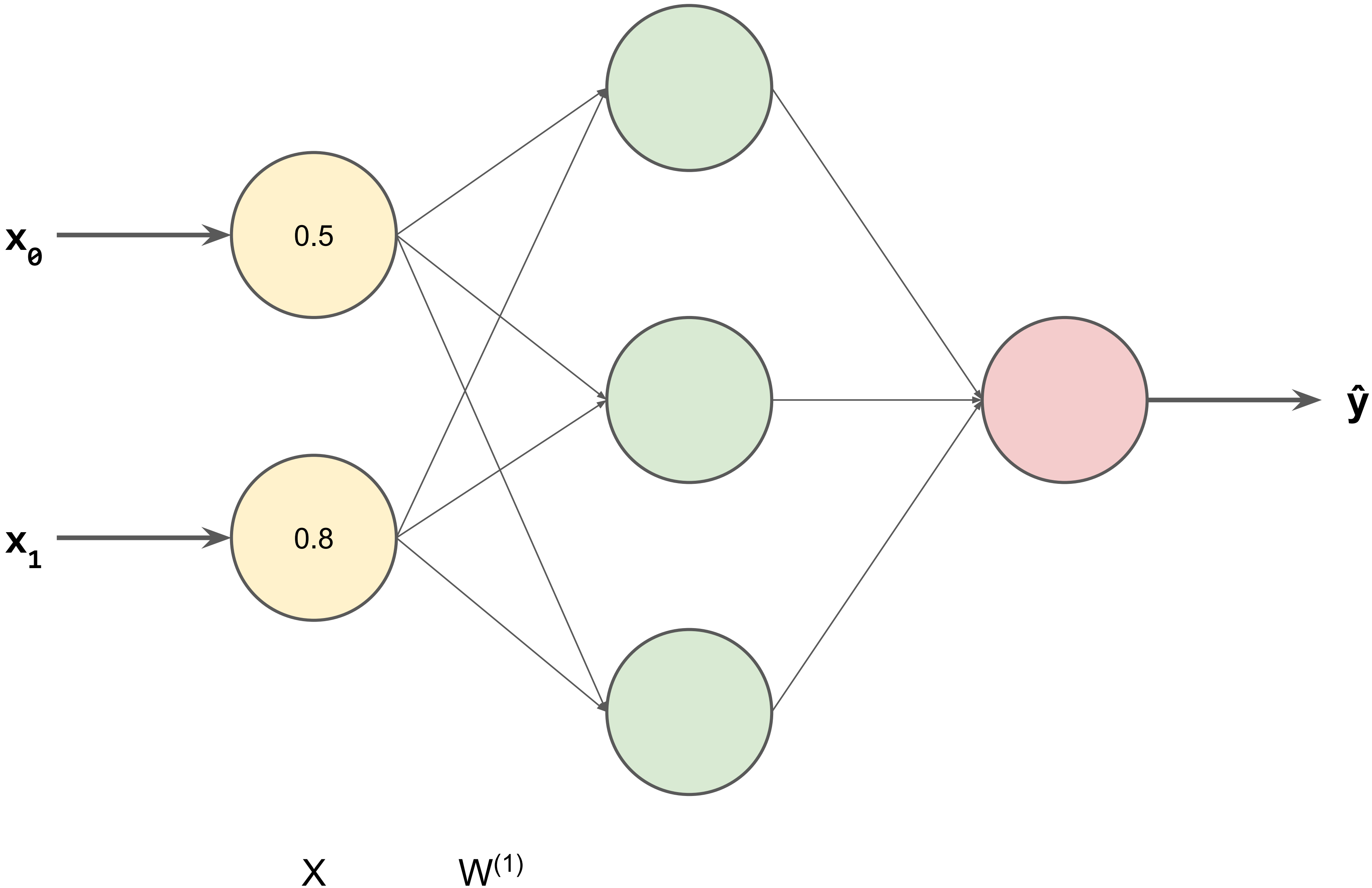


(In more complex architectures like recurrent and recursive networks, signal buffering can combine or recur between the nodes within a layer.)

Forward Propagation

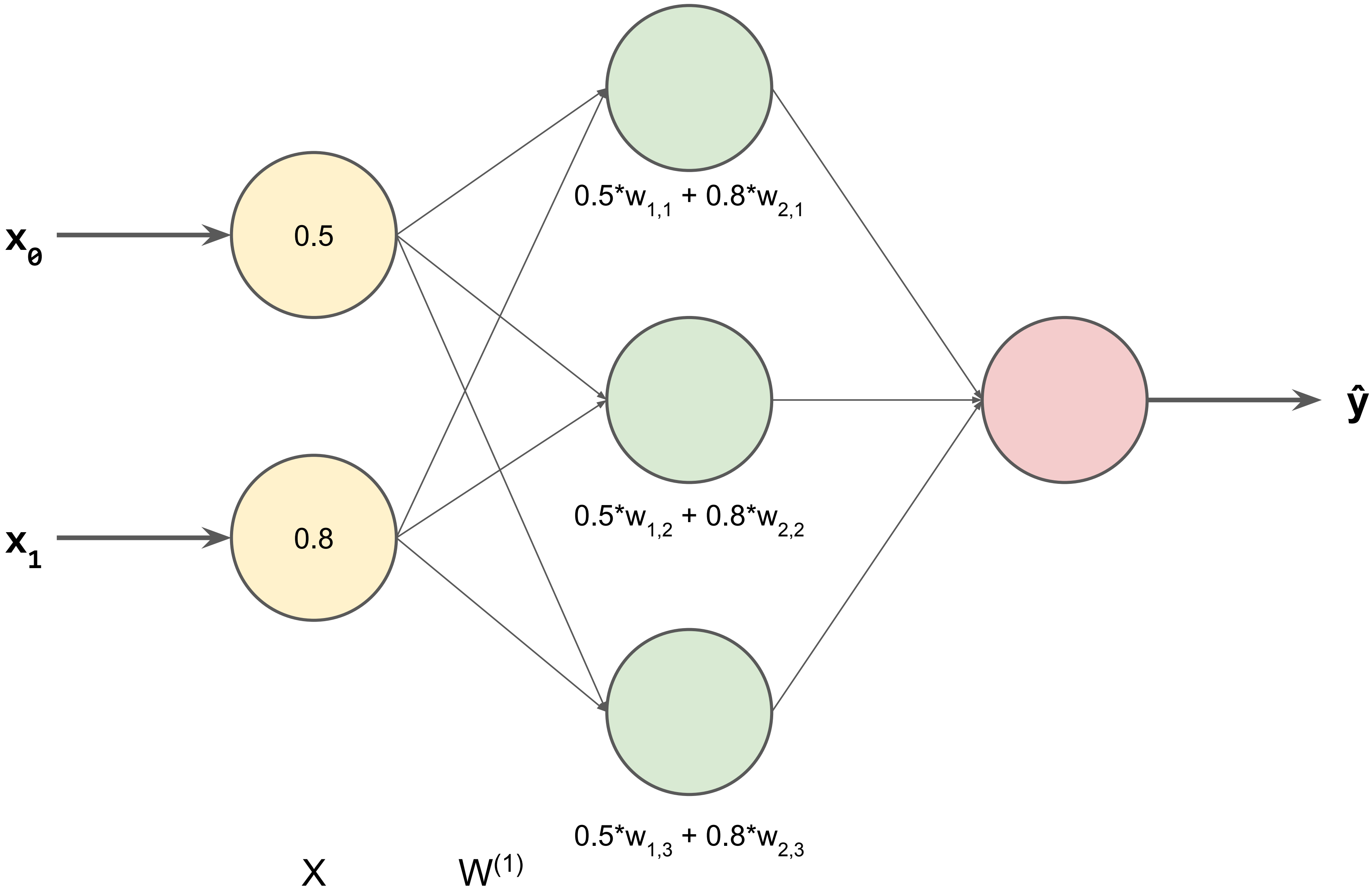


Forward Propagation



$Z^{(2)} = XW^{(1)}$

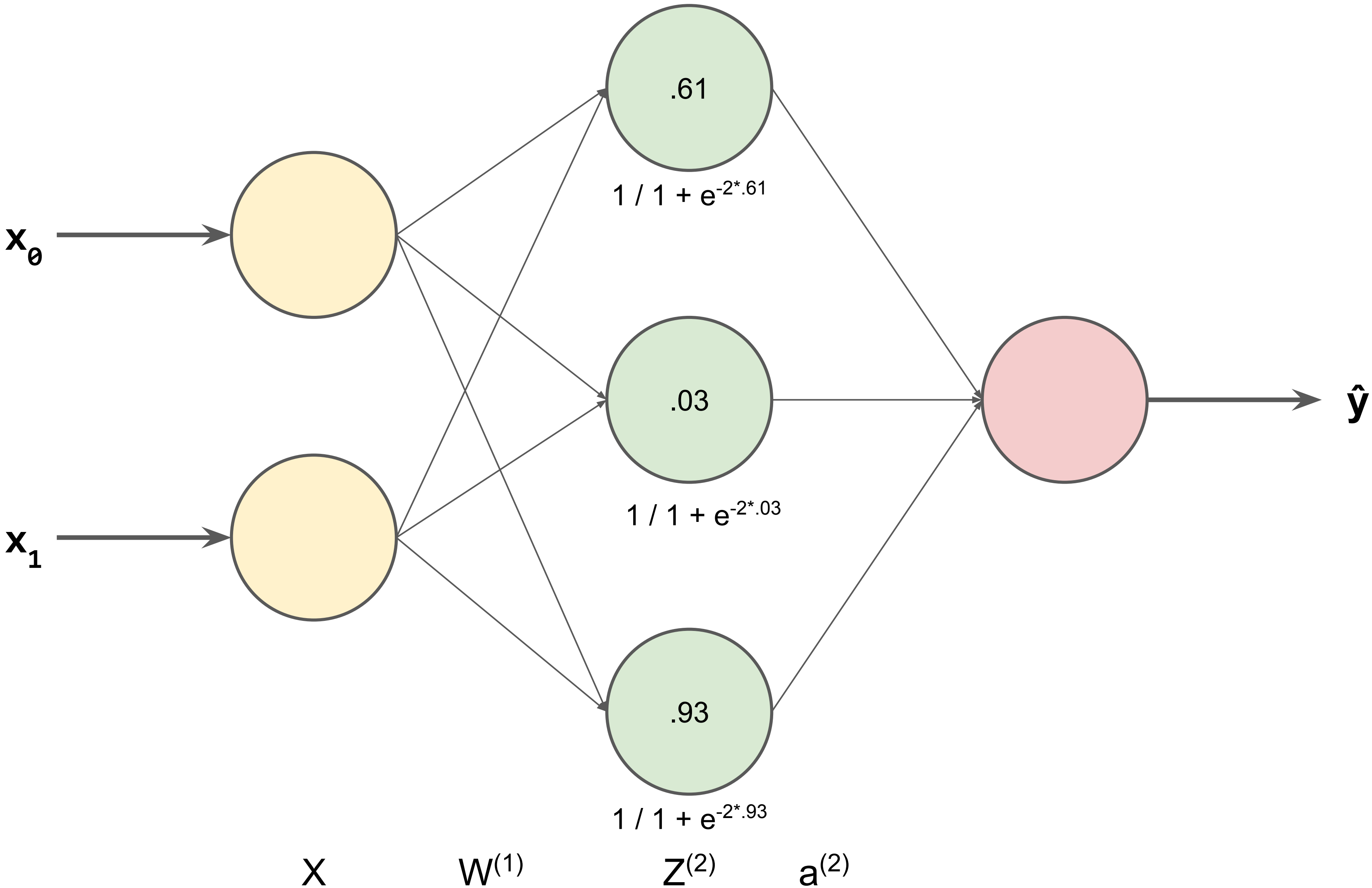
Forward Propagation



Forward Propagation

$$Z^{(2)} = XW^{(1)}$$

$$a^{(2)} = \Phi(Z^{(2)})$$

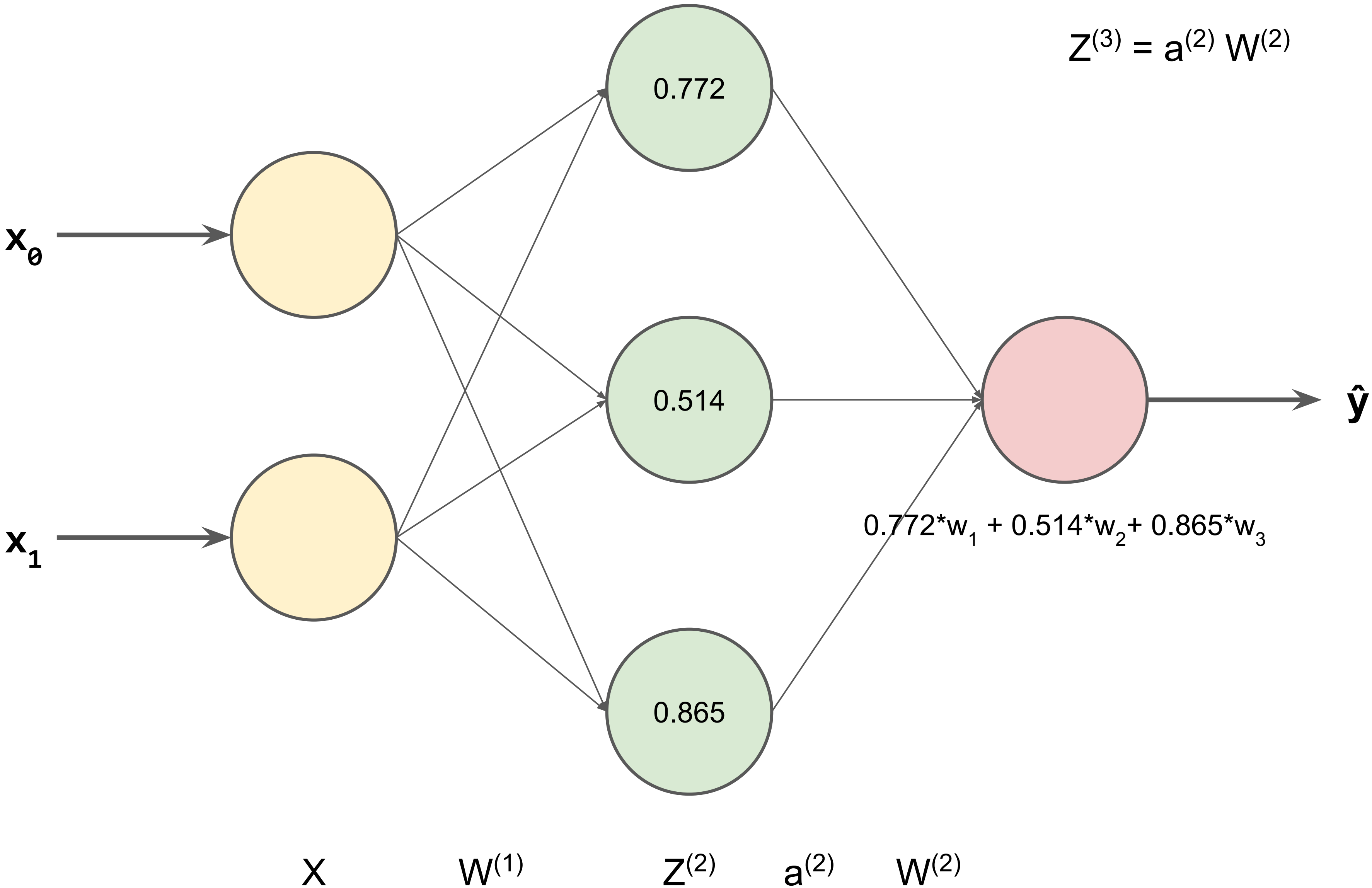


Forward Propagation

$$Z^{(2)} = XW^{(1)}$$

$$a^{(2)} = \Phi(Z^{(2)})$$

$$Z^{(3)} = a^{(2)} W^{(2)}$$



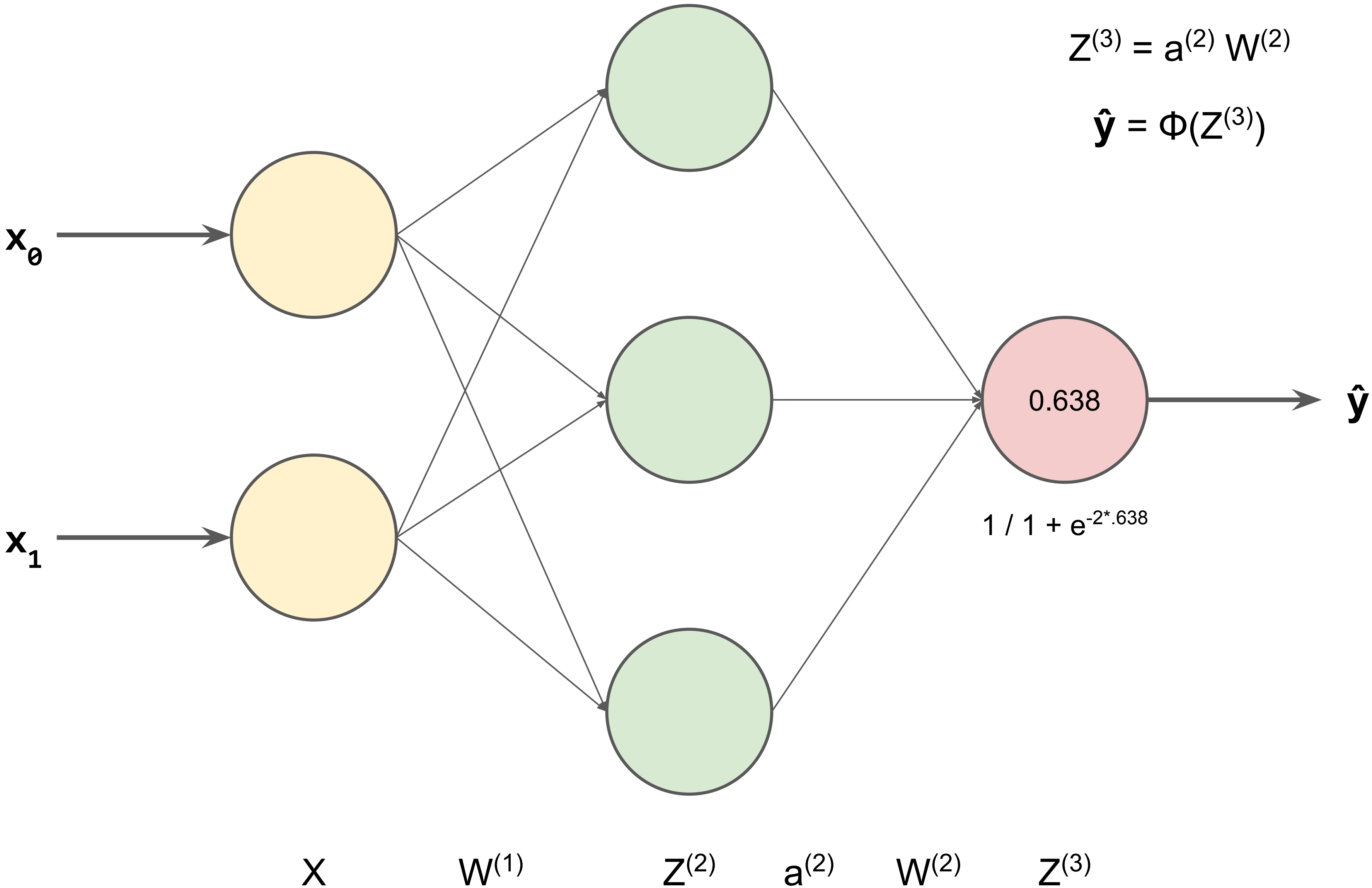
Forward Propagation

$$Z^{(2)} = XW^{(1)}$$

$$a^{(2)} = \Phi(Z^{(2)})$$

$$Z^{(3)} = a^{(2)}\,W^{(2)}$$

$$\hat{\mathbf{y}} = \Phi(Z^{(3)})$$



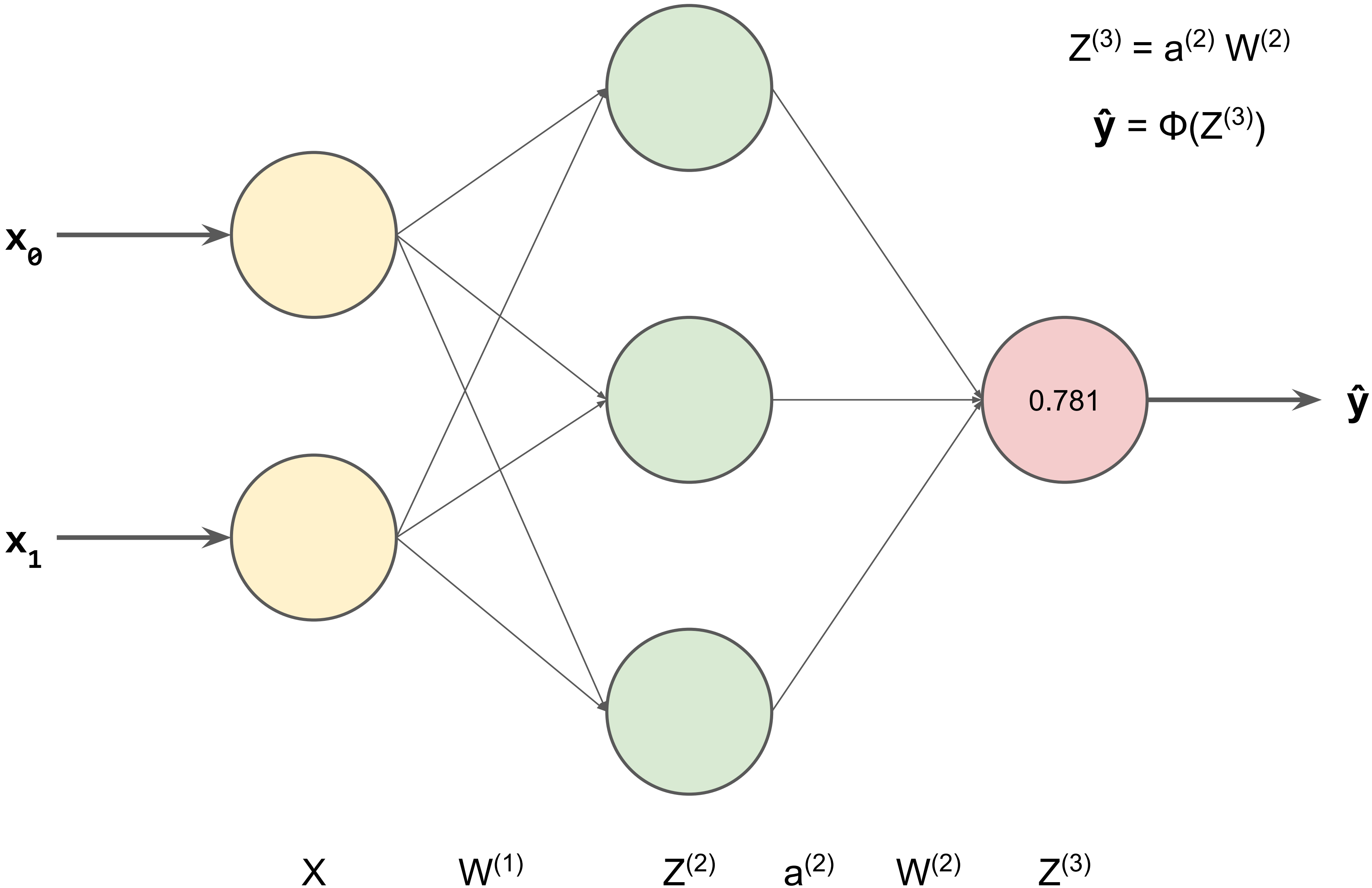
Forward Propagation

$$Z^{(2)} = XW^{(1)}$$

$$a^{(2)} = \Phi(Z^{(2)})$$

$$Z^{(3)} = a^{(2)} W^{(2)}$$

$$\hat{\mathbf{y}} = \Phi(Z^{(3)})$$



Activation Functions

Weighted Sum $y = \sum_{i=0}^{n} w_i x_i$

Gaussian $\phi(y) = \frac{1}{e^{y2}}$

Sigmoid $\phi(y) = \frac{1}{1+e^{-2y}}$

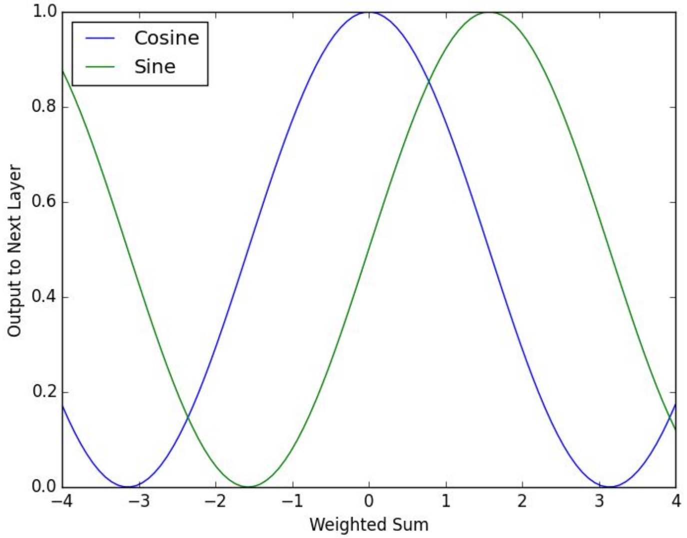
Elliott $\phi(y) = \frac{0.5y}{1+|y|} + 0.5$

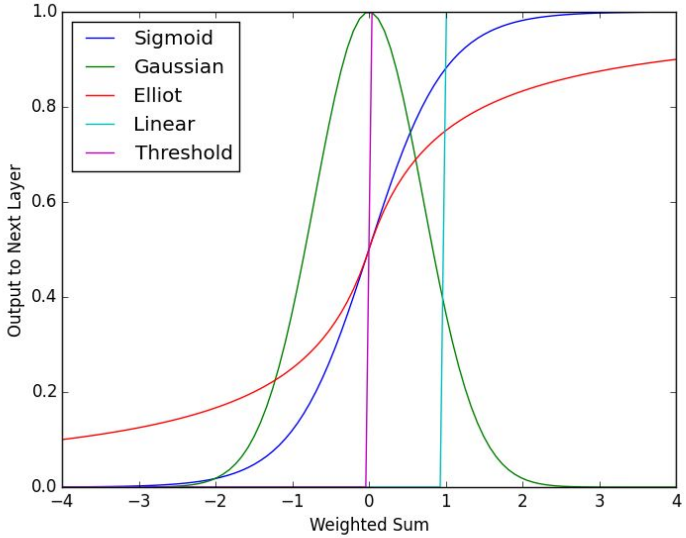
Cosine $\phi(y) = \frac{\cos(y)}{2} + 0.5$

Linear $\phi(y) = y>1?1:(y<0:y)$

Sine $\phi(y) = \frac{\sin(y)}{2} + 0.5$

Threshold $\phi(y) = y<0?0:1$

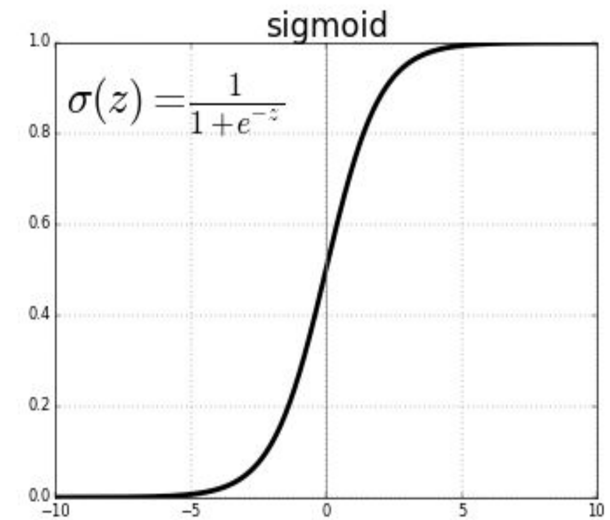


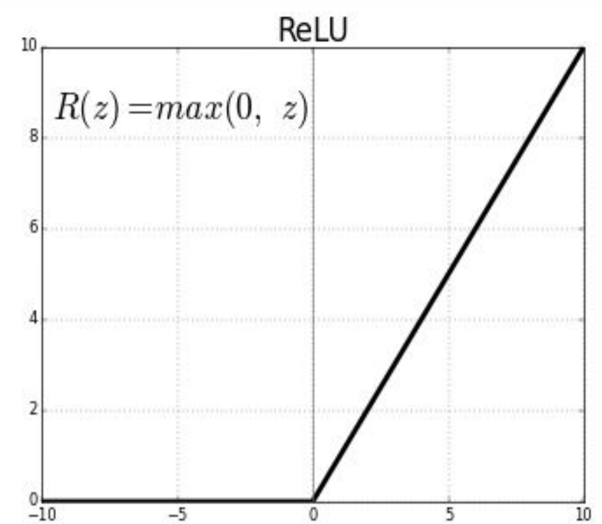


Sloped and Periodic Activation Functions

ReLUs

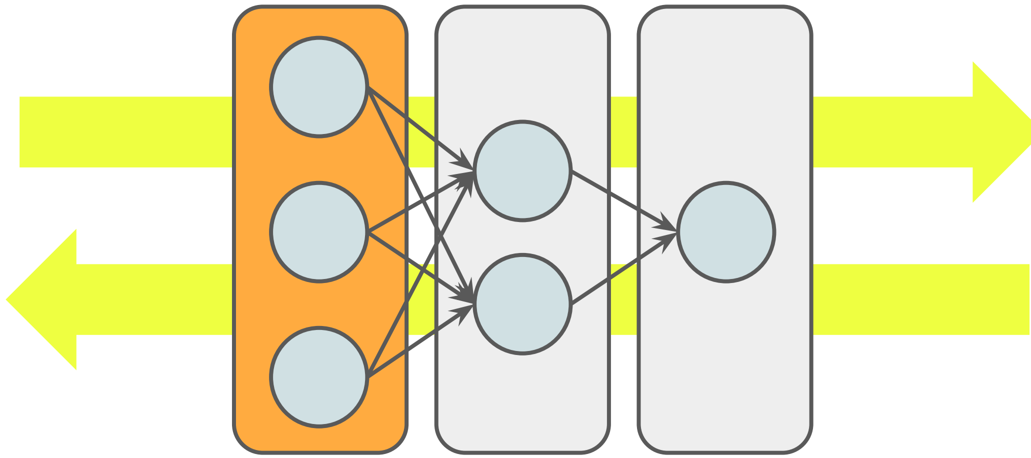
- Generally it makes sense to use a nonlinear activation function, which allows the neural network to model more complex decision spaces.
- Sigmoidal functions are very common, though they can make gradient descent slow when the slope is almost zero.
- For this reason, rectified linear units or “ReLUs,” which output the sum of the weighted inputs (or zero if that sum is negative), have become increasingly popular.





Backpropagation

Neural Networks are usually trained in epochs where each epoch is a complete run through all inputs and error is backpropagated.



Backpropagation

The training algorithm adjusts the weights of the neurons by adjusting them according to some learning rate, α.

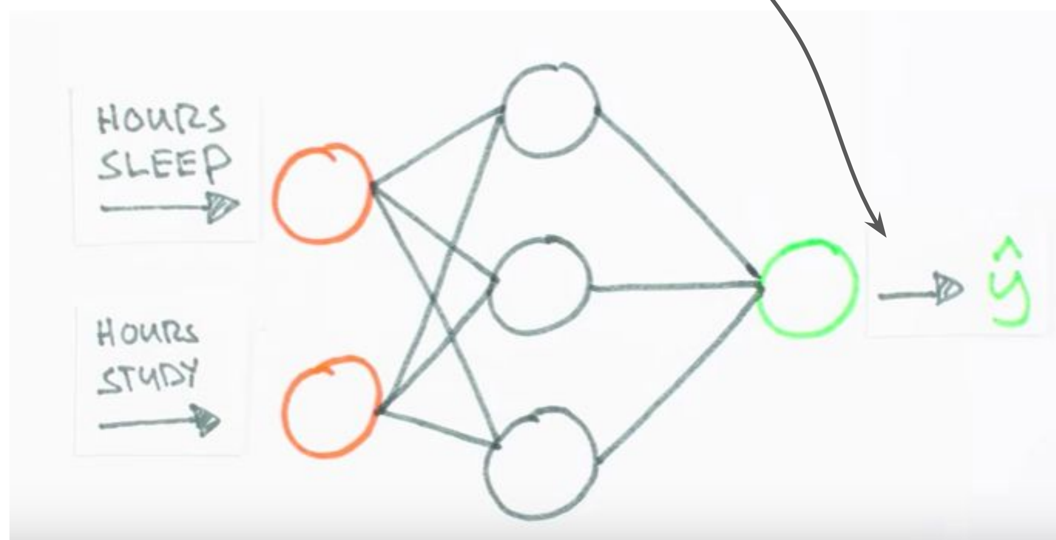
$$\Delta w(t) = -\alpha(t-y)\phi' x_i + \epsilon(t-1)$$

Once the value has been discovered at the end, the error, (expected - actual)^2 is computed, and we use a derivative function to update weights back through the hidden layer.

Each layer is derivable, and therefore the chain rule can be used to find the gradient of an arbitrary number of layers.

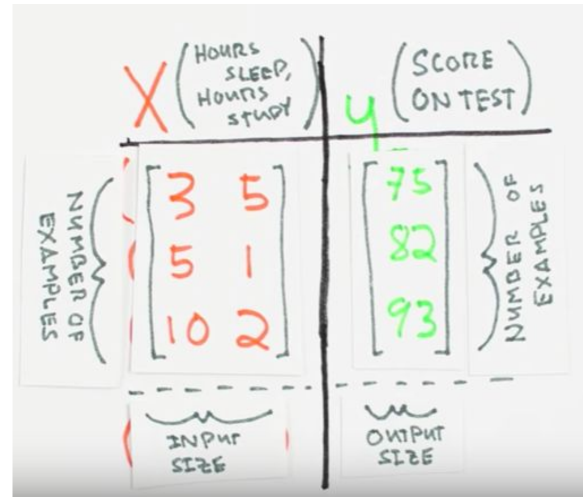
Neural Networks Demystified

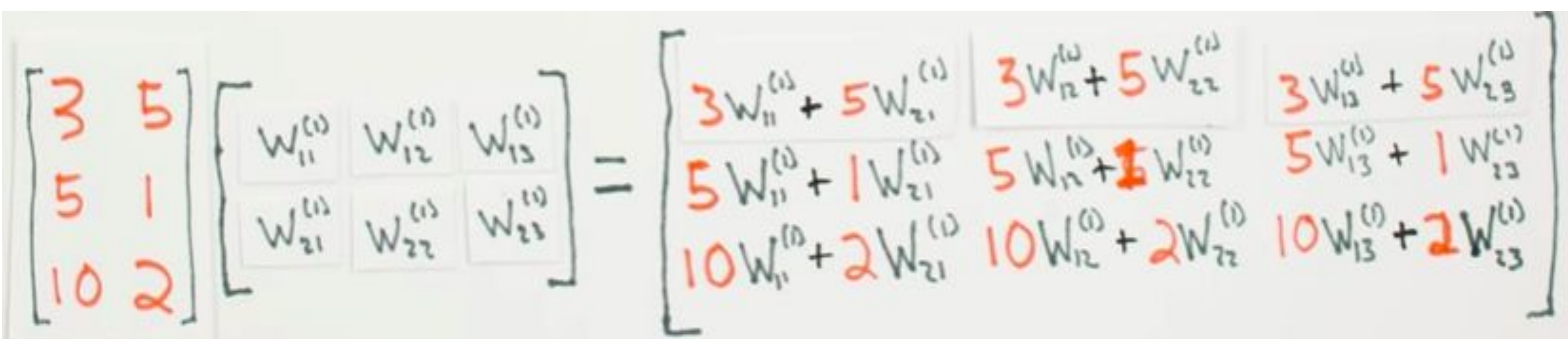
Supervised learning



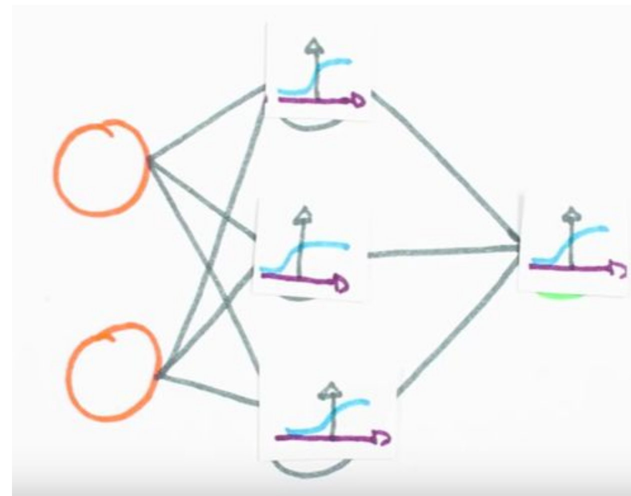
Neural Networks Demystified

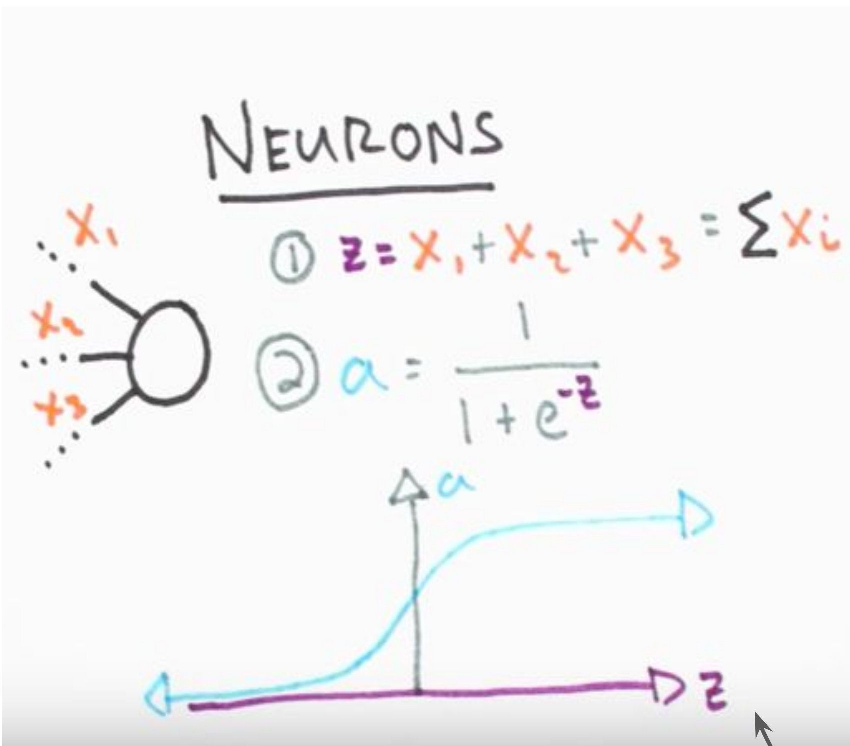
... with matrix multiplication





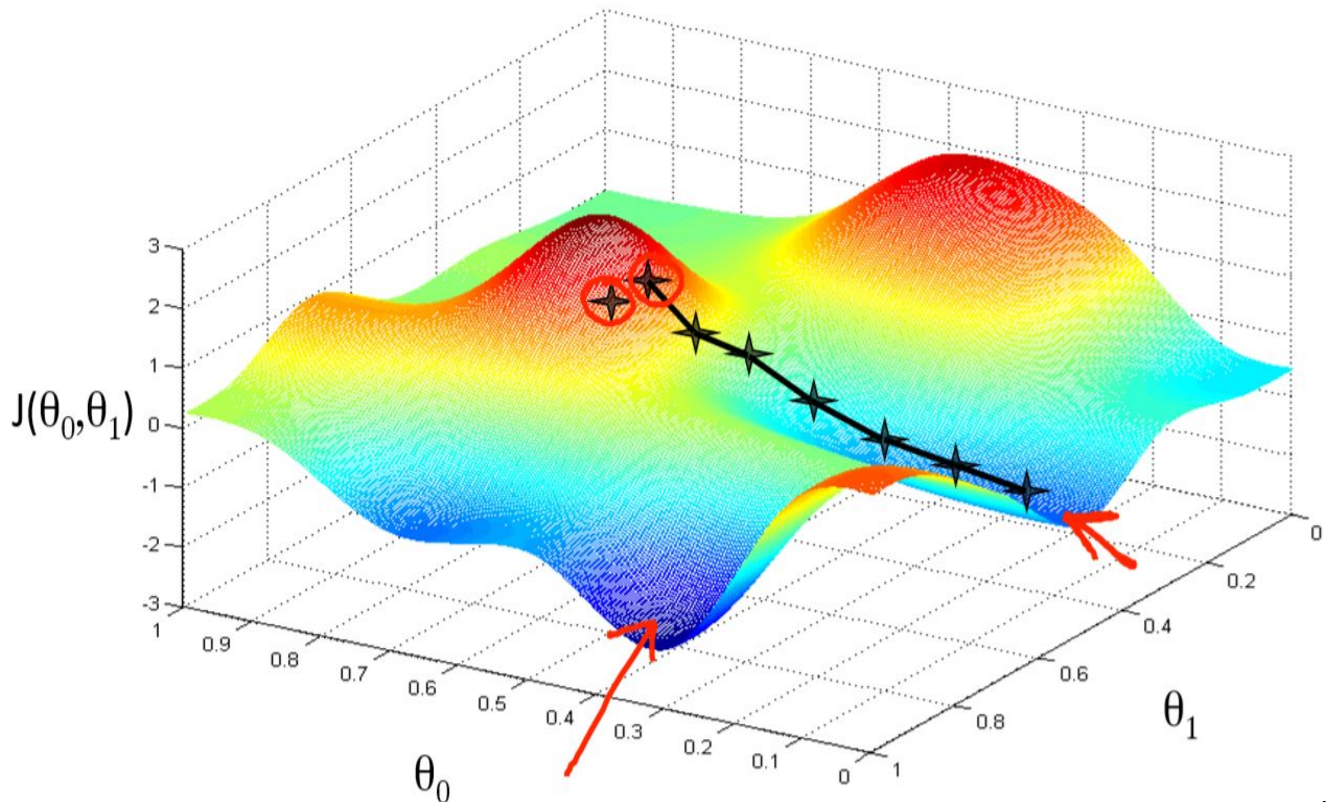
Neural Networks Demystified





... with a non-linear activation function

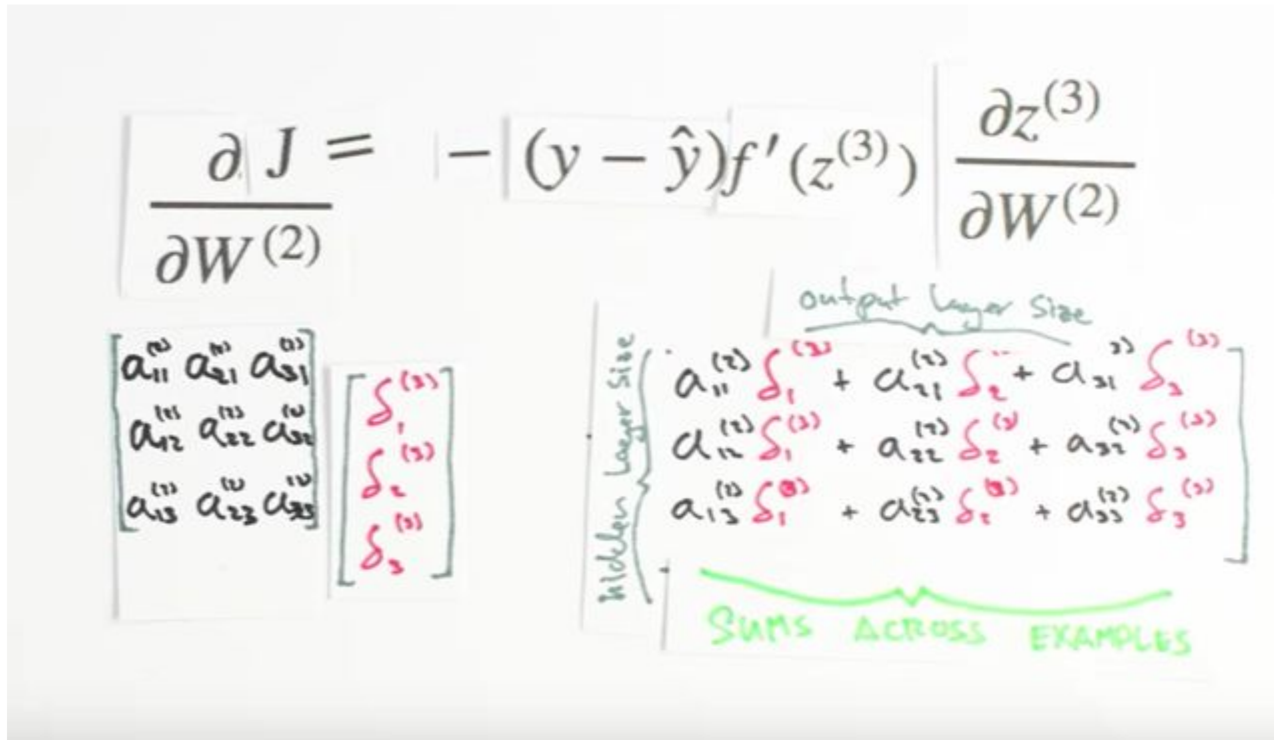
Neural Networks Demystified



... and gradient descent

Neural Networks Demystified

... and a cost function.



Neural Networks Demystified

Part 1: Data and Architecture

Part 2: Forward Propagation

Part 3: Gradient Descent

Part 4: Backpropagation

Part 5: Numerical Gradient Checking

Part 6: Training

Part 7: Overfitting, Regularization, and Testing

Excellent YouTube Series on Neural Networks!

Building Multilayer Perceptrons in Scikit-Learn

Classification

```
from sklearn.neural_network import MLPClassifier


mlp = MLPClassifier()
mlp.fit(X_train, y_train)
mlp.predict(X_test, y_test)
```

MLPClassifer

- Implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation.
- MLP trains on two arrays: array X of size (n_samples, n_features); and array y of size (n_samples,), which holds the (discrete) target values.
- clf.coefs_ contains the weight matrices that constitute the model parameters.
- Supports the cross-entropy loss function; running predict_proba returns a vector of probability estimates for each data point.
- Supports multi-class classification by applying Softmax as the output function.
- Supports multi-label classification in which a sample can belong to more than one class.

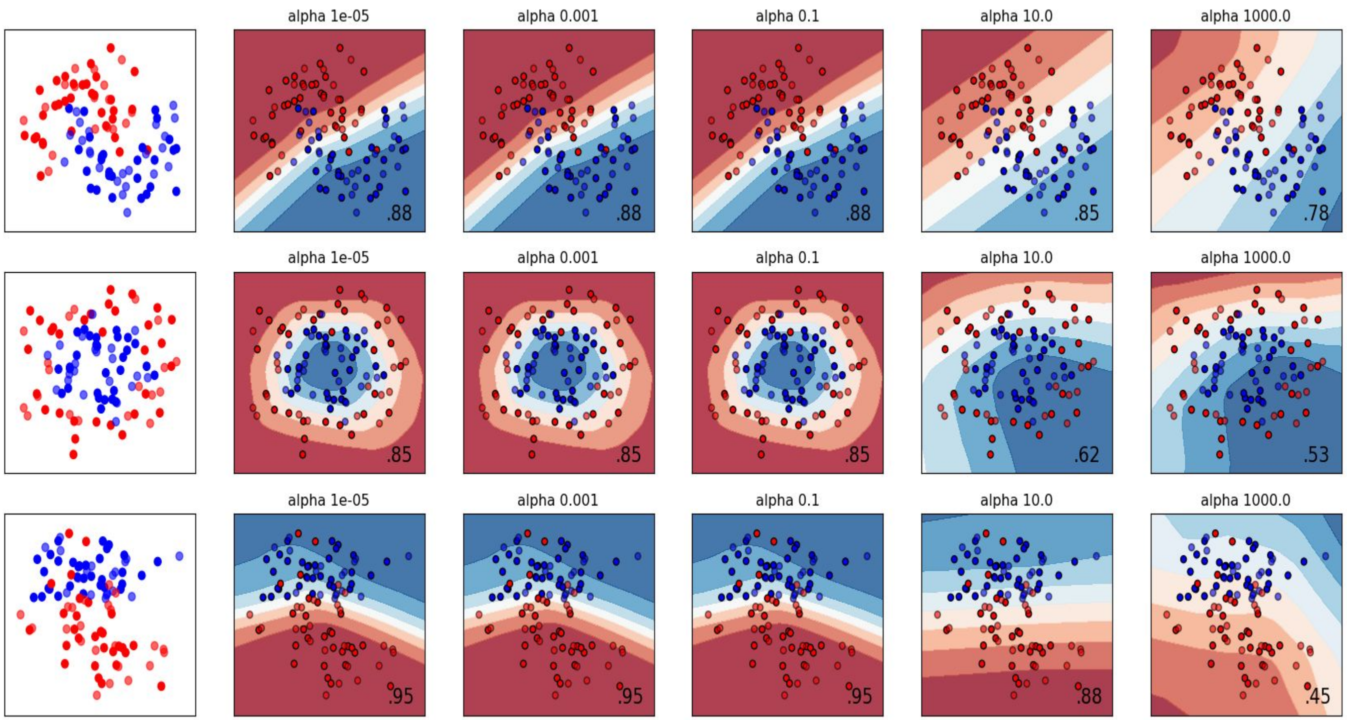
Regression

```
from sklearn.neural_network import MLPRegressor



mlp = MLPRegressor()
mlp.fit(X_train, y_train)
mlp.predict(X_test, y_test)
```

MLPRegressor

- Implements a multi-layer perceptron (MLP) that trains using backpropagation with no activation function in the output layer.
- Also trains on two arrays: array X of size (n_samples, n_features); and array y of size (n_samples,), which holds the (continuous) target values.
- Uses square error as the loss function, and the output is a set of continuous values.
- Also supports multi-output regression, in which a sample can have more than one target.

Regularization



Both MLPRegressor and MLPClassifier use L2 regularization (alpha) to help avoid overfitting by penalizing weights with large magnitudes.

Gradient Descent

- In Scikit-Learn, MLP trains using Stochastic Gradient Descent (SGD) or L-BFGS.
- SGD updates parameters using the gradient of the loss function with respect to the parameter that needs adjusting.

$$w \leftarrow w - \eta(\alpha \frac{\partial R(w)}{\partial w} + \frac{\partial Loss}{\partial w})$$

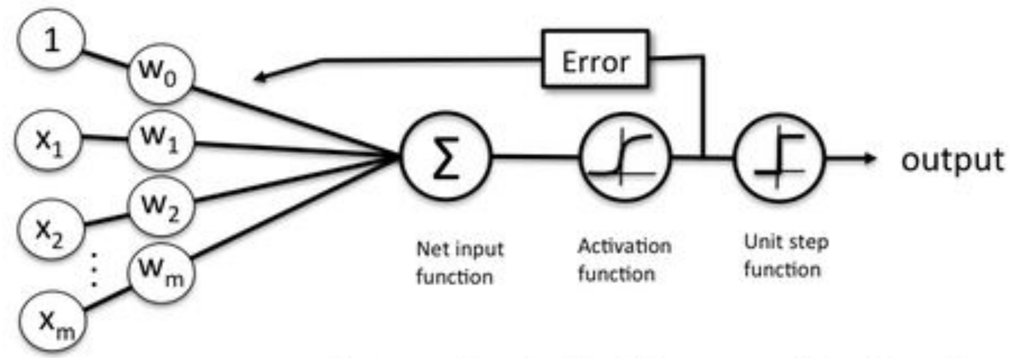
- L-BFGS is a solver that approximates the Hessian matrix which represents the second-order partial derivative of a function. It approximates the inverse of the Hessian matrix to perform parameter updates.

Tips

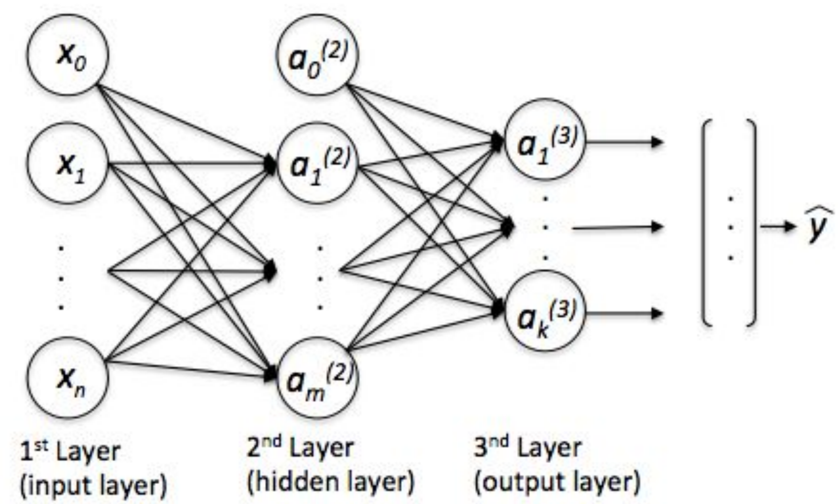
- Start simple: Start with only a few layers and neurons, and add complexity (e.g. add layers, neurons to hidden_layer_sizes, or add training epochs by increasing max_iter) to see if accuracy is increasing evenly across all train and test splits.
- Scale your data: Multi-layer perceptrons are sensitive to feature scaling.
- Tune carefully: MLP with hidden layers have a non-convex loss function where there exists more than one local minimum. Therefore different random weight initializations can lead to different validation accuracy.
- Use for prototyping: The Scikit-Learn implementation is not intended for large-scale applications and offers no GPU support. For faster, GPU-based implementations, use TensorFlow/Keras or PyTorch!

Neural Networks vs. Logistic Regression

- Weights and a sigmoidal function mean that we could think of a logistic regression as a simple neural network.
- A softmax layer is typically used in neural networks as well for classification.
- The logit function can be used as a sigmoidal function for an ANN.



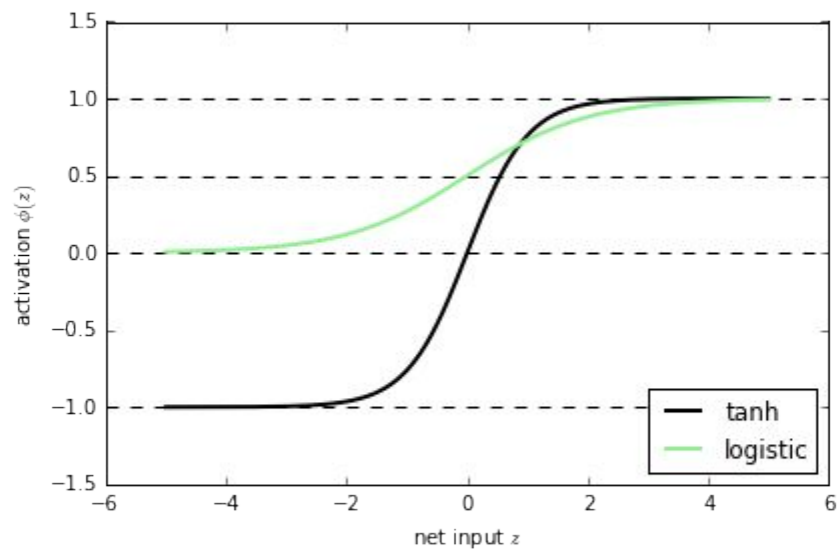
Schematic of a logistic regression classifier.



Schematic of a multi-layer perceptron.

Neural Networks vs. Logistic Regression

- Logits are convex, meaning they are easily optimized and theoretically always provide the global cost minimum.
- Tanh is more often used in ANNs because it can produce non-positive outputs.
- Logistic Regression performs as well as 1-2 layer ANNs (better).



Neural Networks are connected to most other ideas in machine learning - and are also becoming a generalized framework for ML computation.

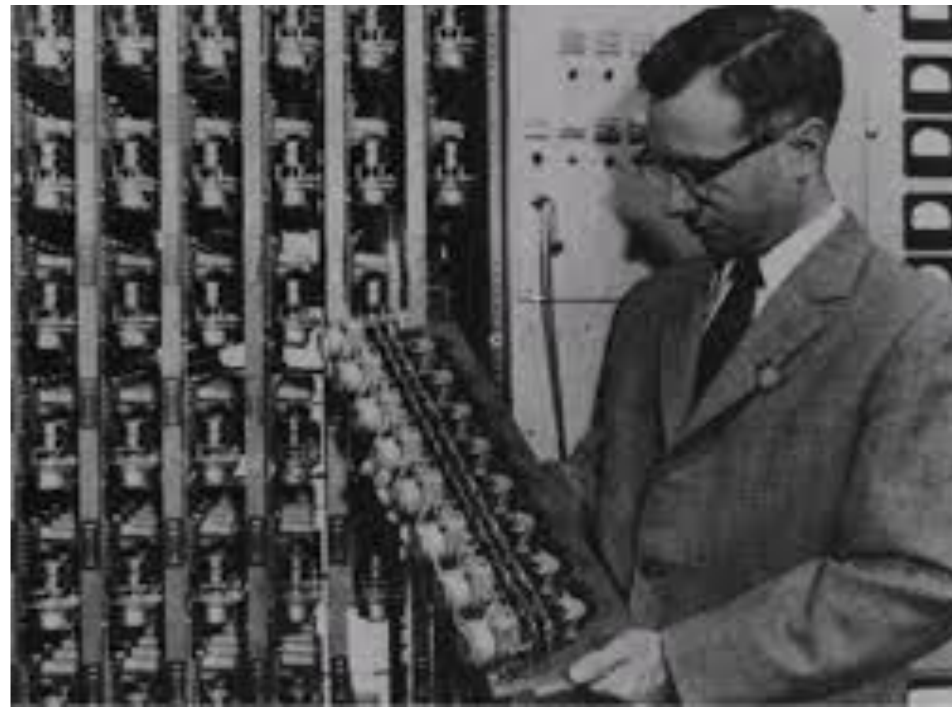
History of Neural Networks

- 1943: Warren S. McCulloch & Walter Pitts develop the ANN: logical calculus of imminent neural activity.
- 1959: Bernard Widrow & Marcian Hoff develop ADALINE and MADALINE used as an adaptive filter to eliminate noise or echo on phone lines (still used in ATC).
- 1962: Widrow & Hoff developed a learning procedure that examines the value before the weight adjusts it.
- 1969: von Neumann Perceptrons
- 1972: Kohonen and Anderson develop matrix algebra representation of a system of weights.
- 1982: Hopfield networks, hybrid networks
- 1986: Multi-layered Machines and Back Propagation

  …
- 2007: Language modeling with ANNs
- 2010: Large scale speech recognition
- 2011: Google Brain (ducks from rabbits)

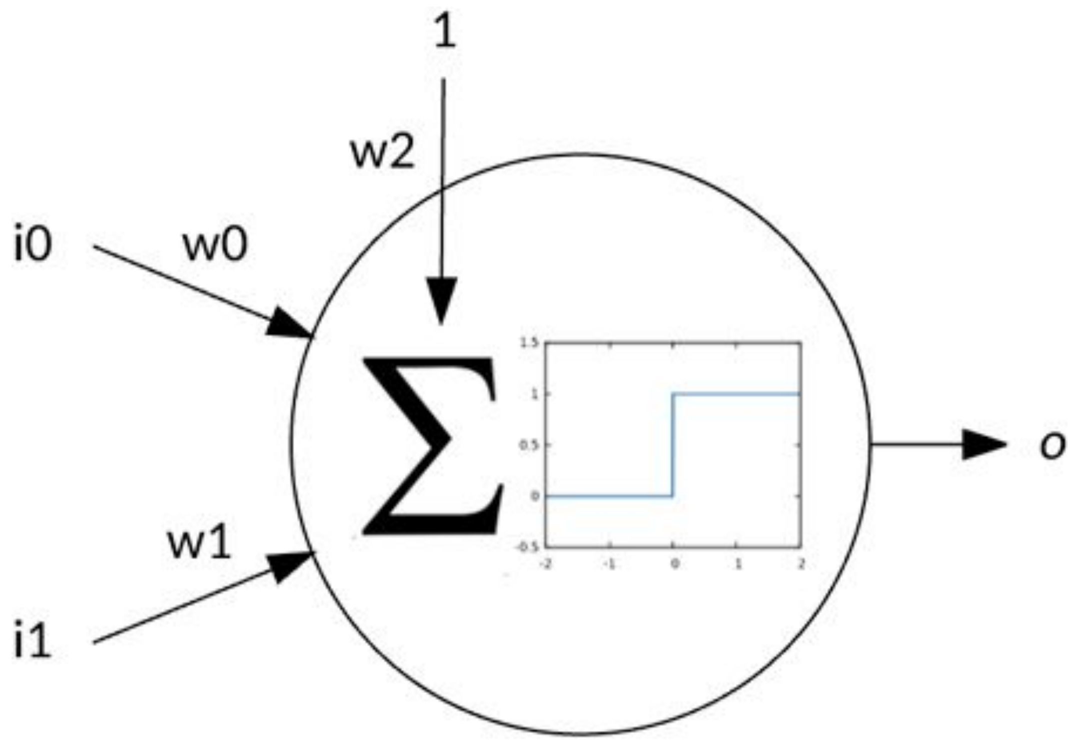
Perceptrons

- In 1957, Frank Rosenblatt invents a piece of hardware designed to help with image recognition tasks, the Perceptron.
- In 1969, Marvin Minsky and Seymour Papert publish *Perceptrons*, a book of mathematical proofs about neural networks.



Perceptrons

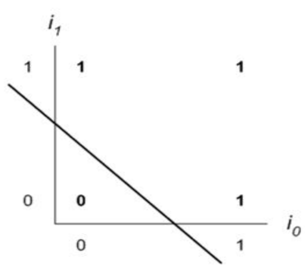
- Modeled on the behavior of the human brain.
- Used for binary classification, making predictions based on a linear function of weights and feature vector.
- Hardware related idea: produce digital signals



Single-Layer Perceptrons

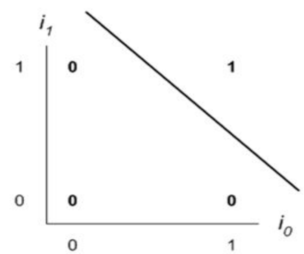
i_0	i_1	o
0	0	0
0	1	1
1	0	1
1	1	1

OR



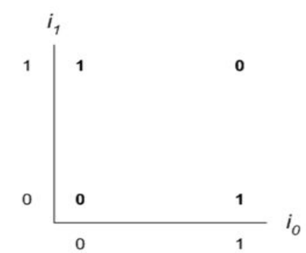
i_0	i_1	o
0	0	0
0	1	0
1	0	0
1	1	1

AND



i_0	i_1	o
0	0	0
0	1	1
1	0	1
1	1	0

XOR



Can learn linearly separable patterns.

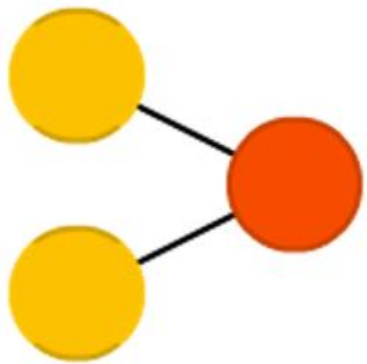
Can't learn an XOR function.

Neural Network Taxonomy

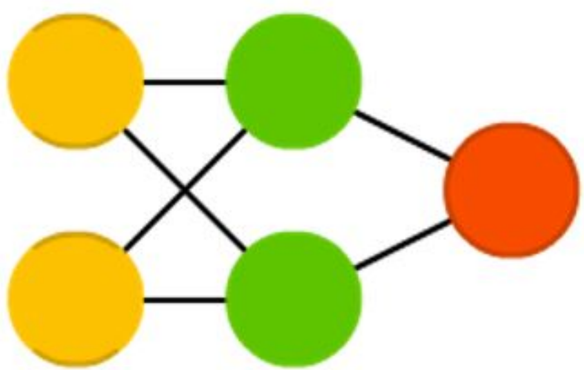
- Tons of acronyms floating around
- Drawing node relationships makes understanding what’s happening in a neural network easier.
- Any model can be made up of an arbitrary network.
- Generalized to tensorflow and Keras.

http://www.asimovinstitute.org/neural-network-zoo/

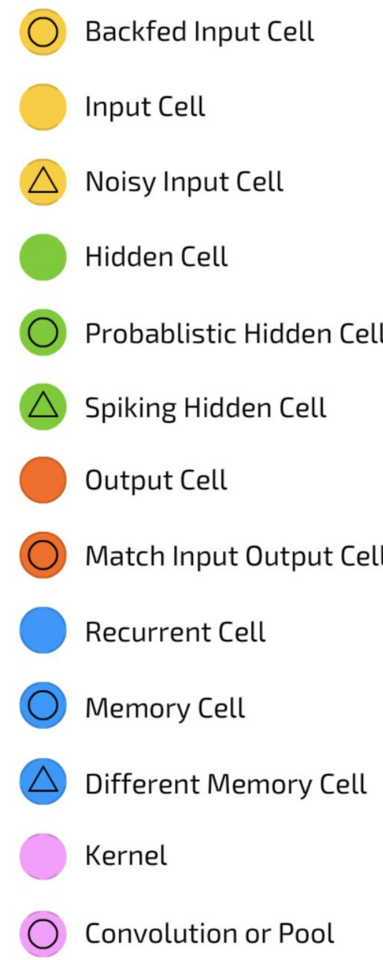
Neural Network Taxonomy



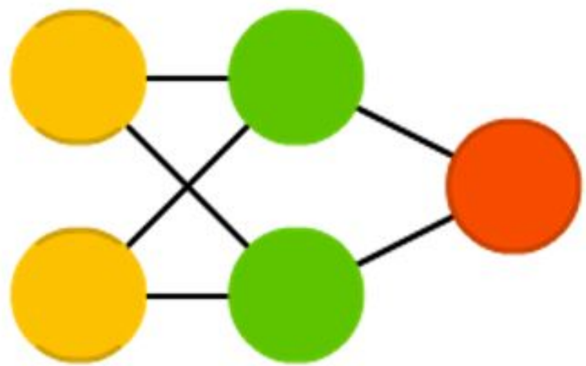
Perceptron



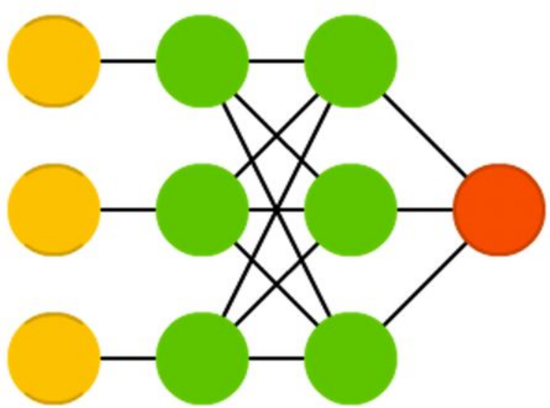
MLP/Feed Forward



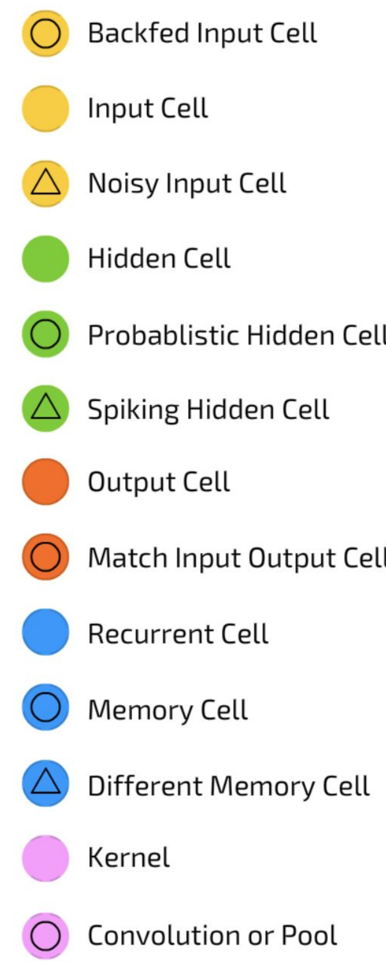
Neural Network Taxonomy



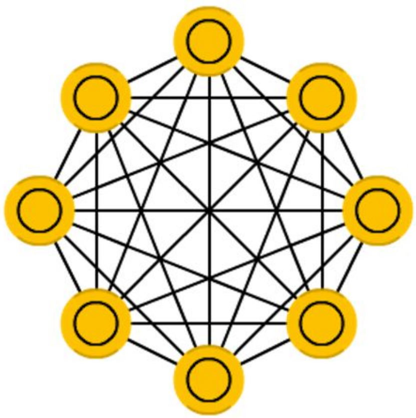
Radial Basis Function



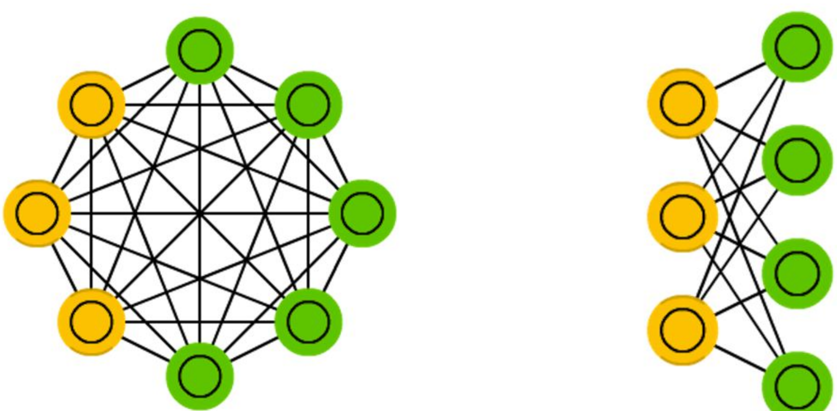
SVM



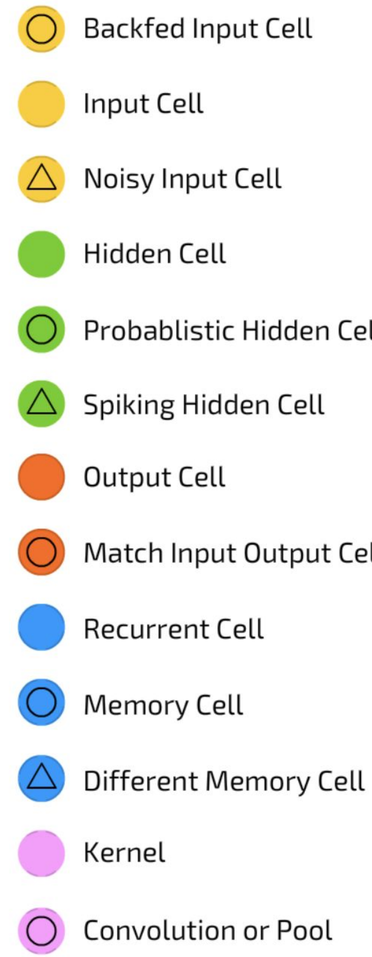
Neural Network Taxonomy



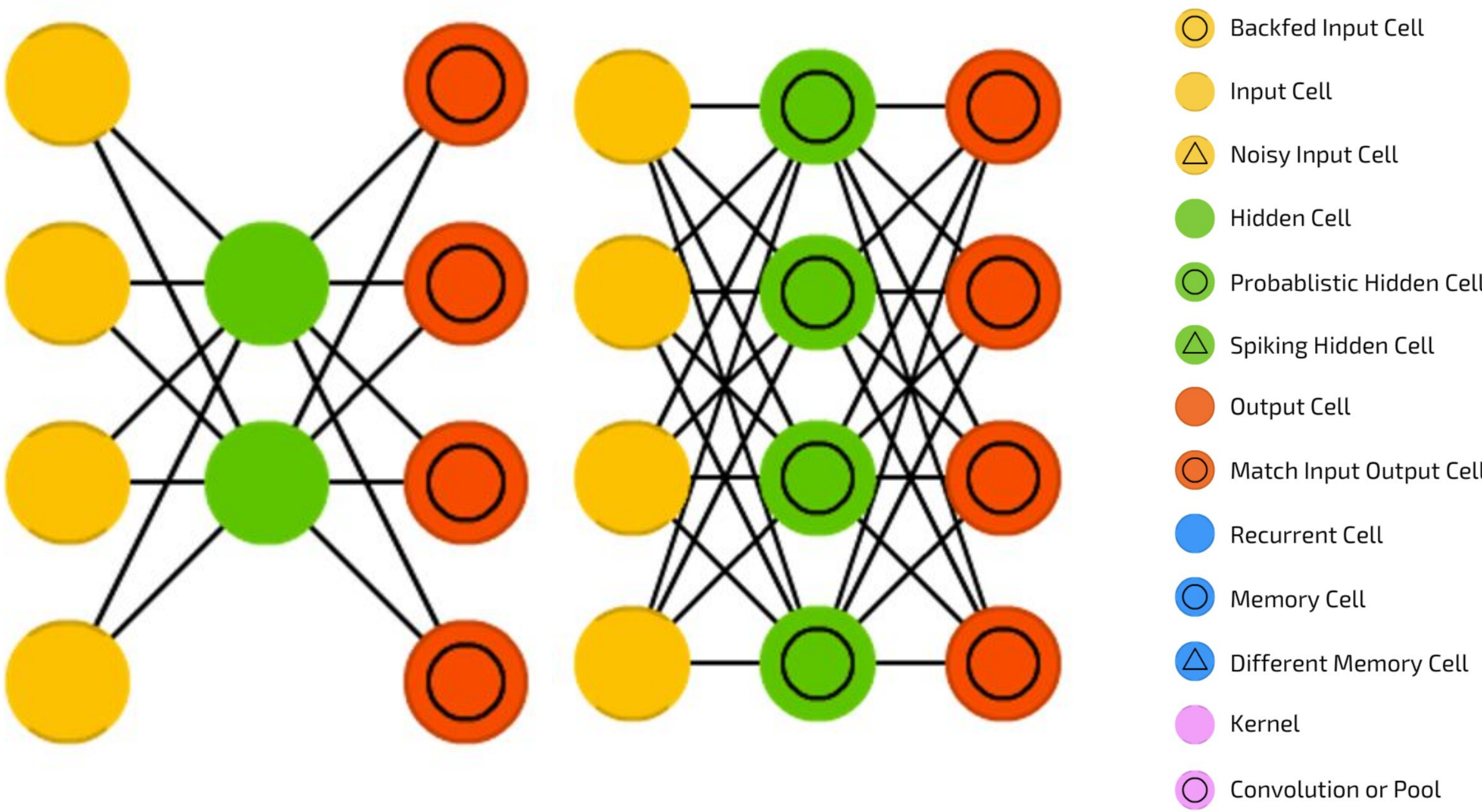
Hopfield Network



Boltzman Machine (Restricted)

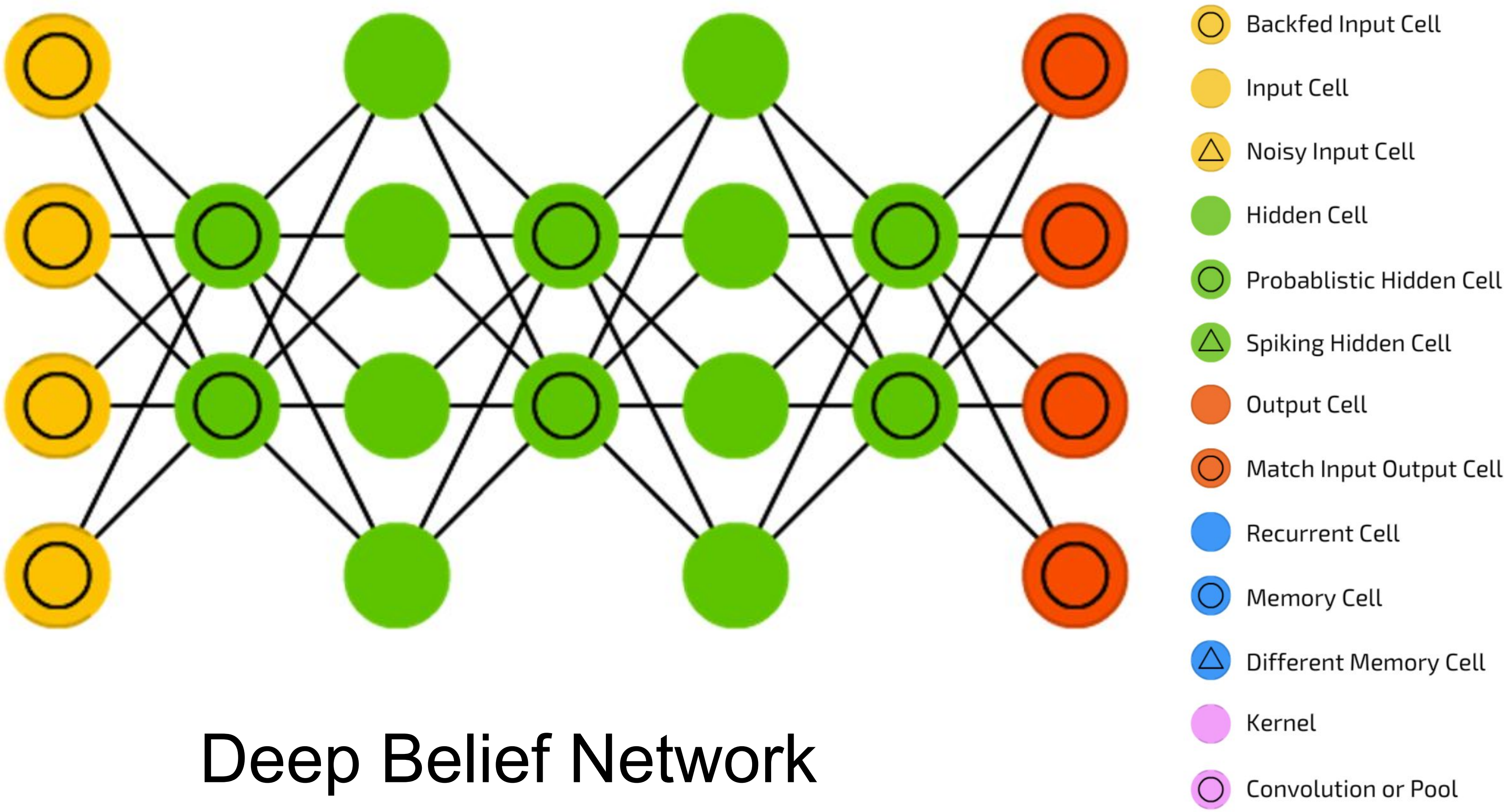


Neural Network Taxonomy



Auto Encoders (Variational)

Neural Network Taxonomy

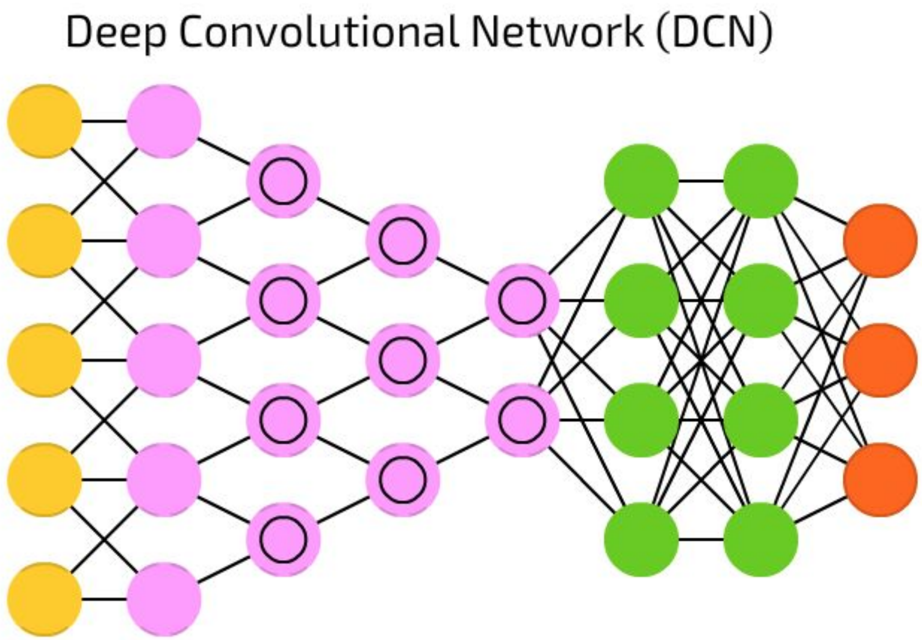


Deep Belief Network

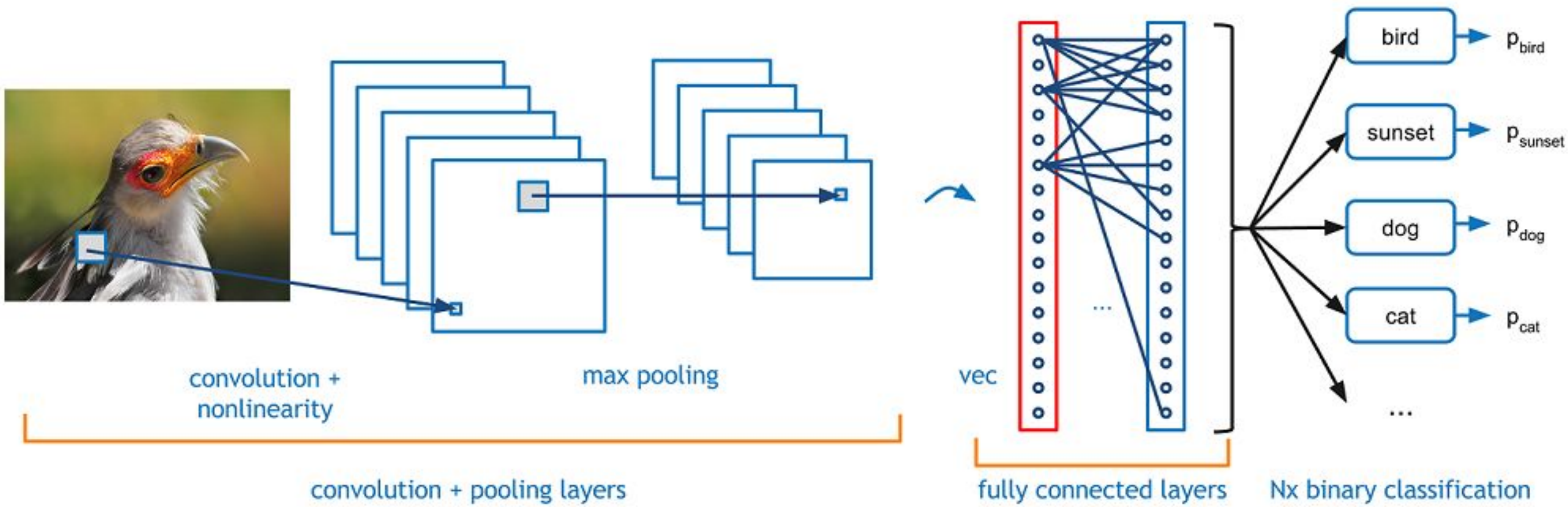
Deep Neural Models

Convolutional Neural Networks

- Combine multilayer perceptrons with a convolutional layer that iteratively builds a map to extract important features
- A pooling stage reduces the dimensionality of the features but preserves their most informative components.
- Highly effective for modeling image data and performing tasks like classification and summarization.

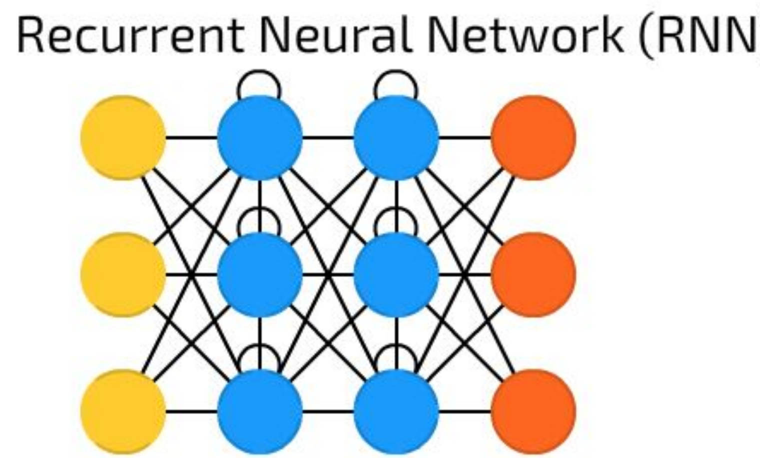


Convolutional Neural Networks



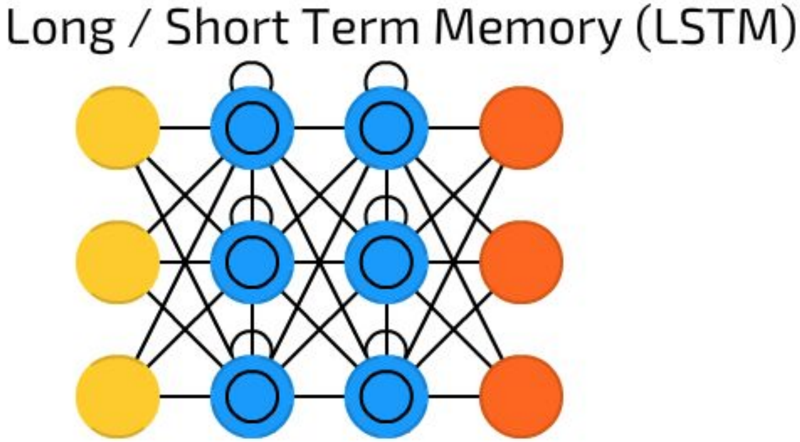
Recurrent Neural Networks

- Allows model to maintain order in a sequence (e.g. words in a sentence) and keep track of long-term dependencies.
- Each recurrent cell internally stores previous value, and is updated like basic cells, but with extra weights.
- Good for modeling sequential data (like language, video, etc)



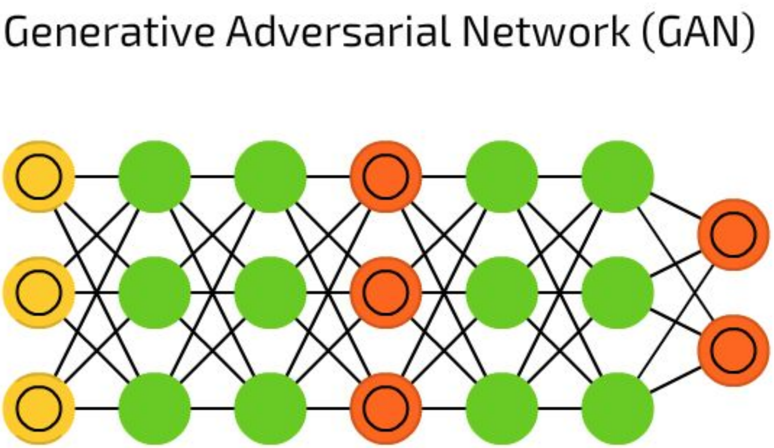
Long Short Term Memory Networks

- Implement 3 logic gates: input, output, and "forget"
- LSTM cells store 4 states (RNNs only store 2):
  - current output value
  - last output value
  - current state of "memory cell"
  - last state of "memory cell"
- Popular for machine translation and natural language generation tasks.



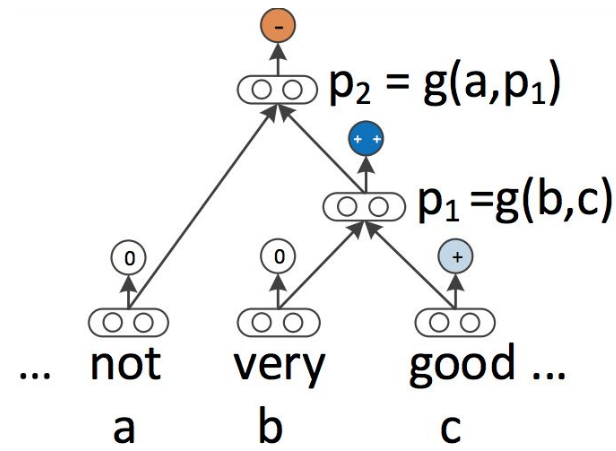
Generative Adversarial Networks

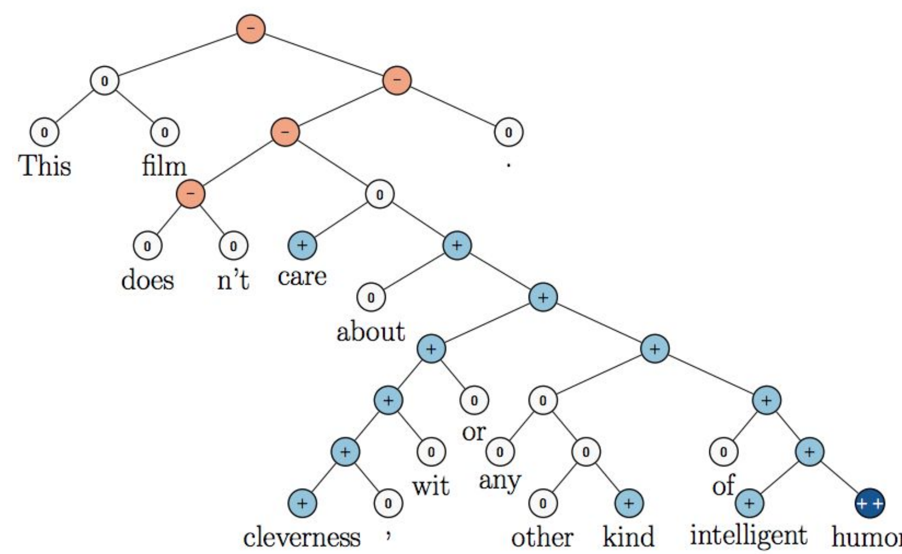
- Predict *features* given *targets* (most supervised learning algorithms try to predict *targets* given *features*).
- Network vs. network:
  - The “generator” network generates new data.
  - The “discriminator” network evaluates each instance to decide if it comes from real training data.
- Commonly used with image data.



Recursive Neural Tensor Networks

- Apply weights recursively to structured input to produce structured predictions.
- Leverage constituency parsing; learn continuous representations (e.g. word embeddings), grouping words into phrases.
- Used for NLP tasks (e.g. sentiment classification).





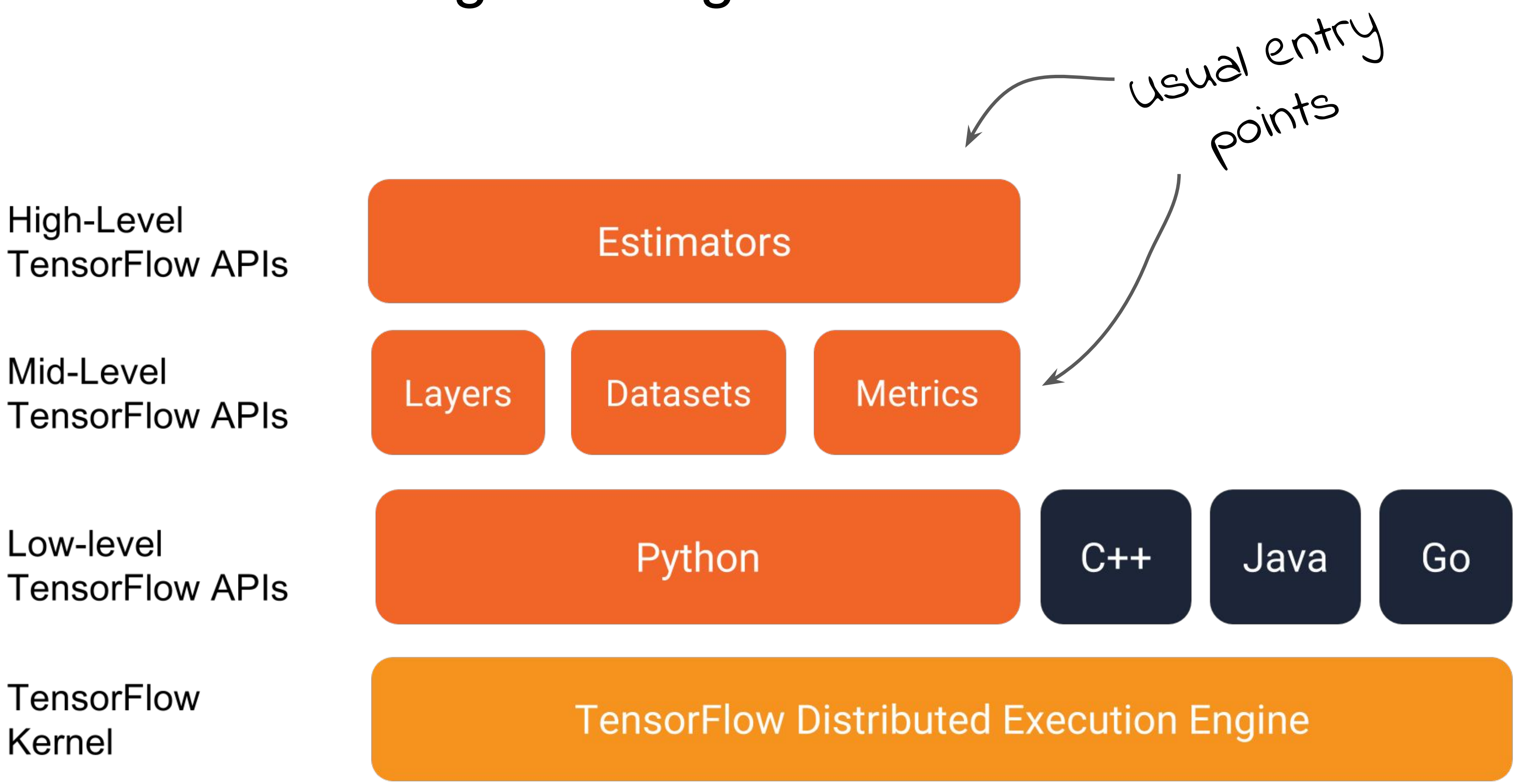
Customized Deep Learning

Tensorflow

What is TensorFlow?

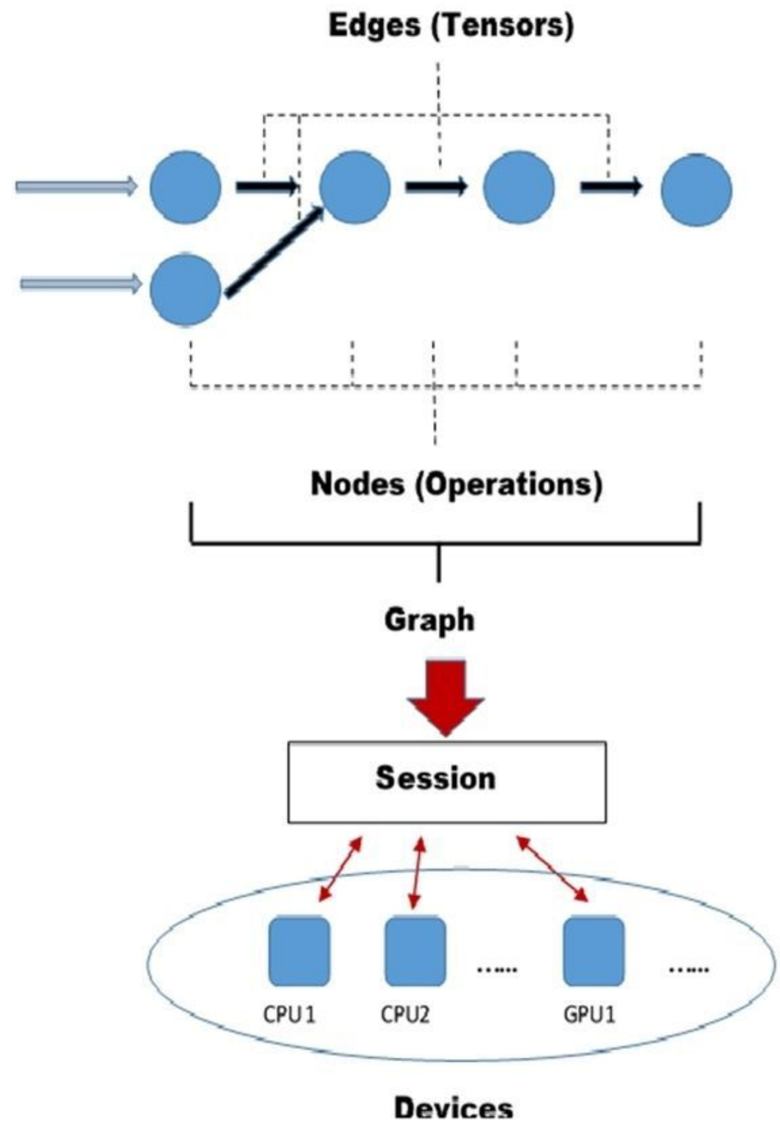
- Open source deep learning framework written in Python, C++, and CUDA (Google).
- Distributed computation; parallelize models across GPUs, networks of machines.
- Build, control, and optimize custom data flow graphs.
- Assumes familiarity with neural network architectures.

TensorFlow Programming Stack

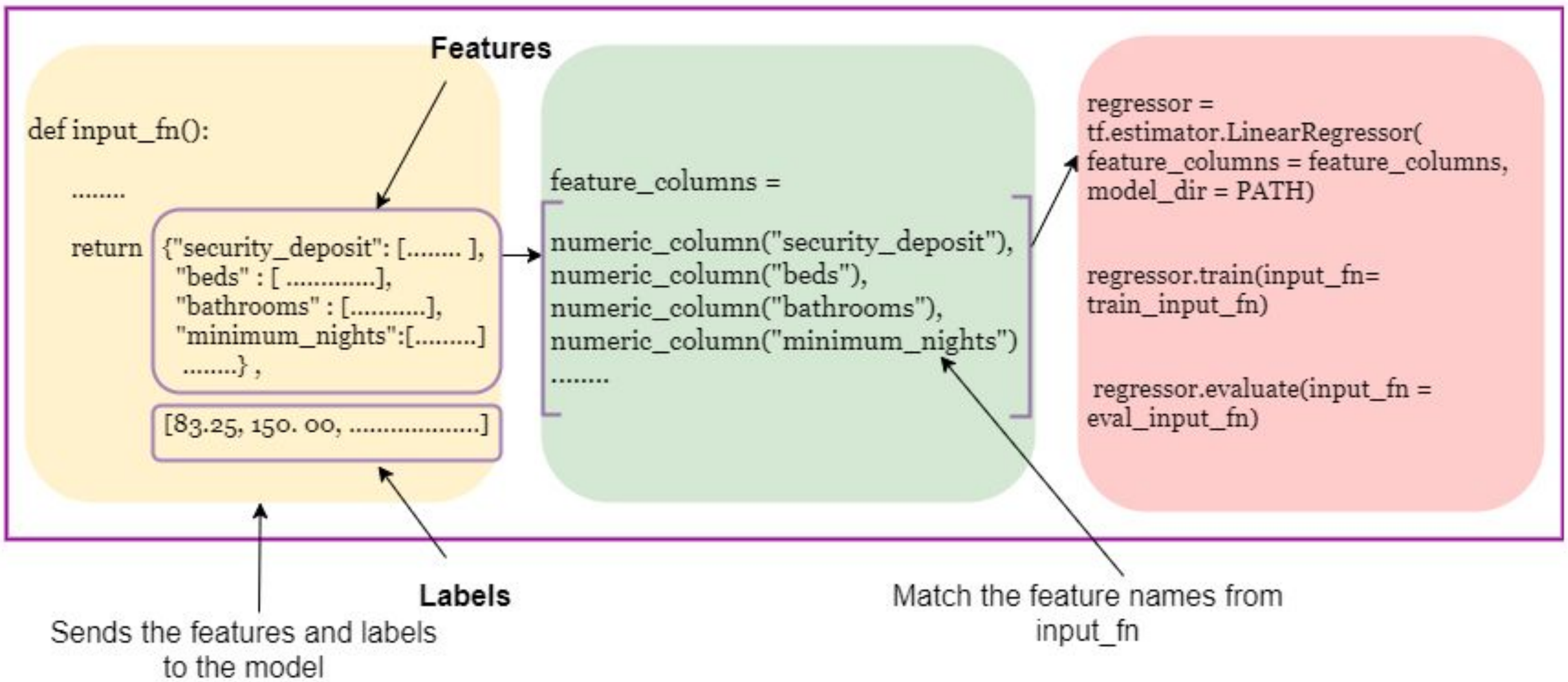


Building Neural Networks with TensorFlow

1. Convert data to tensors.
2. Specify each layer, with hyperparameters.
3. Compile those layers into a static graph.
4. Run a session to begin the training.
5. Evaluate.



Premade TensorFlow Estimators



Recommended Workflow

1. Start with pre-made Estimator to establish a baseline.
2. Build and test your pipeline.
3. Compare with other pre-made Estimators to see which produces the best results.
4. Improve your model by building a custom Estimator, using the pre-made Estimators as blueprints.

Keras

What is Keras?

- Open source Python library for specifying deep learning models.
- Original interface written for Theano backend (theoretically framework-neutral).
- Became default for many TensorFlow users; pulled into TensorFlow core in 2017.

Keras

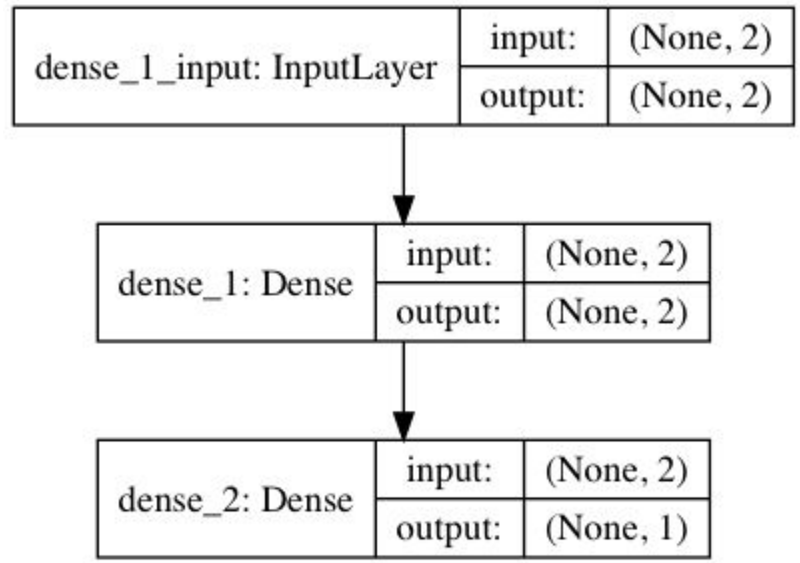
What does Keras add to TensorFlow?

- Everything is an object!
- Convenient for prototyping
- Implements Scikit-Learn API:
    - keras.wrappers.scikit_learn.KerasClassifier
    - keras.wrappers.scikit_learn.KerasRegressor
- Therefore Sequential Keras models can be integrated as part of a Scikit-Learn Pipeline or Gridsearch, used with Scikit-Yellowbrick for diagnostics and tuning.
- Excellent documentation.

Sequential Models with Keras

A Keras Sequential model is a good place to start.

It's just a linear stack of neural network layers.



Sequential Models with Keras

You can instantiate and then add layers one by one via .add():

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

...or by passing a list of layer instances to the constructor:

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

Specifying the Input Shape

The first layer in a Sequential model needs to know what input shape to expect. Pass this in as an argument to the first layer (tuple).

Either `input_shape`:

```
model = Sequential()
model.add(Dense(32, input_shape=(784,)))
```

Or `input_dim`:

```
model = Sequential()
model.add(Dense(32, input_dim=784))
```

(The following layers can do automatic shape inference)

Compiling the Network

After adding all the layers and before training, configure the learning process via the compile method.

Specify (1) an optimizer, (2) a loss function, and (3) a list of metrics, e.g.

for binary classification:

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

for multi-class classification

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```python
from keras.layers import Dense
from keras.models import Sequential


N_FEATURES = 5000
N_CLASSES = 4

def build_network():
    """
    Create a function that returns a compiled neural network
    """
    nn = Sequential()
    nn.add(Dense(500, activation='relu', input_shape=(N_FEATURES,)))
    nn.add(Dense(150, activation='relu'))
    nn.add(Dense(N_CLASSES, activation='softmax'))
    nn.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy']
    )
    return nn
```

Dense Layers

- Regular densely-connected layer.
- Implements `output = activation(dot(input, kernel) + bias)` where:
  - activation is the element-wise activation function passed as the activation argument
  - kernel is a weights matrix
  - bias is a bias vector created by the layer (if use_bias=True).
- Input is an n-dimensional tensor with shape `(batch_size, ..., input_dim)`. Frequently, input is 2D, with shape `(batch_size, input_dim)`. If input has rank > 2, it will be flattened.
- Output is an n-dimensional tensor with shape `(batch_size, ..., units)`. So, for example, our hypothetical 2D input with shape `(batch_size, input_dim)` would output a tensor of shape `(batch_size, units)`.

Many Others!

- `Input:` Instantiate a Keras tensor
- `Activation:` Applies an activation function to an output.
- `Dropout:` Randomly set a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.
- `Flatten:` Flatten input shape (without impacting batch size).
- `Reshape` : Reshape output to target shape.
- `Permute`: Permutes the dimensions of the input according to a given pattern; useful for e.g. connecting RNNs and convnets together.

Many Others!

- `RepeatVector`: Repeats the input n times.
- `Lambda`: Wraps arbitrary expression as a Layer object.
- `ActivityRegularization`: Layer that applies an update to the cost function based on input activity.
- `Masking`: Masks a sequence by using a mask value to skip timesteps.
- `SpatialDropout1D`: Spatial 1D version of Dropout.
- `SpatialDropout2D`: Spatial 2D version of Dropout.
- `SpatialDropout3D`: Spatial 3D version of Dropout.

Pytorch

Pytorch

- Open source deep learning framework written in Python, C++, and CUDA (Facebook).
- Distributed computation; parallelize models across GPUs, networks of machines.
- Build, control, and optimize custom data flow graphs.
- Nice integration with Python ecosystem.

Pytorch

- Define graphs dynamically
- Tensors, numpy-like arrays for GPUs
- Faster ramp-up time
- Good for research
- More Pythonic
- Easier to debug
- Newer, smaller community

Tensorflow

- Define graphs statically
- Tensors
- Better serialization support
- Good for deployment (gRPC server)
- Declarative
- Visualize with Tensorboard
- More established, bigger community

Framework Comparison: Design Choices

Design Choice	Torch.nn	Theano	Caffe	Chainer	MXNet	Tensor-Flow	PyTorch
NN definition	Script (Lua)	Script* (Python)	Data (protobuf)	Script (Python)	Script (many)	Script (Python)	Script (Python)
Backprop	Through graph	Extended graph	Through graph	Through graph	Through graph	Extended graph	Through graph
Parameters	Hidden in operators	Separate nodes	Hidden in operators	Separate nodes	Separate nodes	Separate nodes	Separate nodes
Update formula	Outside of graphs	Part of graphs	Outside of graphs	Outside of graphs	Outside of graphs	Part of graphs	Outside of graphs
Graph construction	Static	Static	Static	Dynamic	Static	Static	Dynamic
Graph Optimization	-	Supported	-	-	-	Supported	-
Parallel computation	Multi GPU*	Multi GPU*	Multi GPU*	Multi GPU**	Multi node Multi GPU	Multi node Multi GPU	Multi GPU**

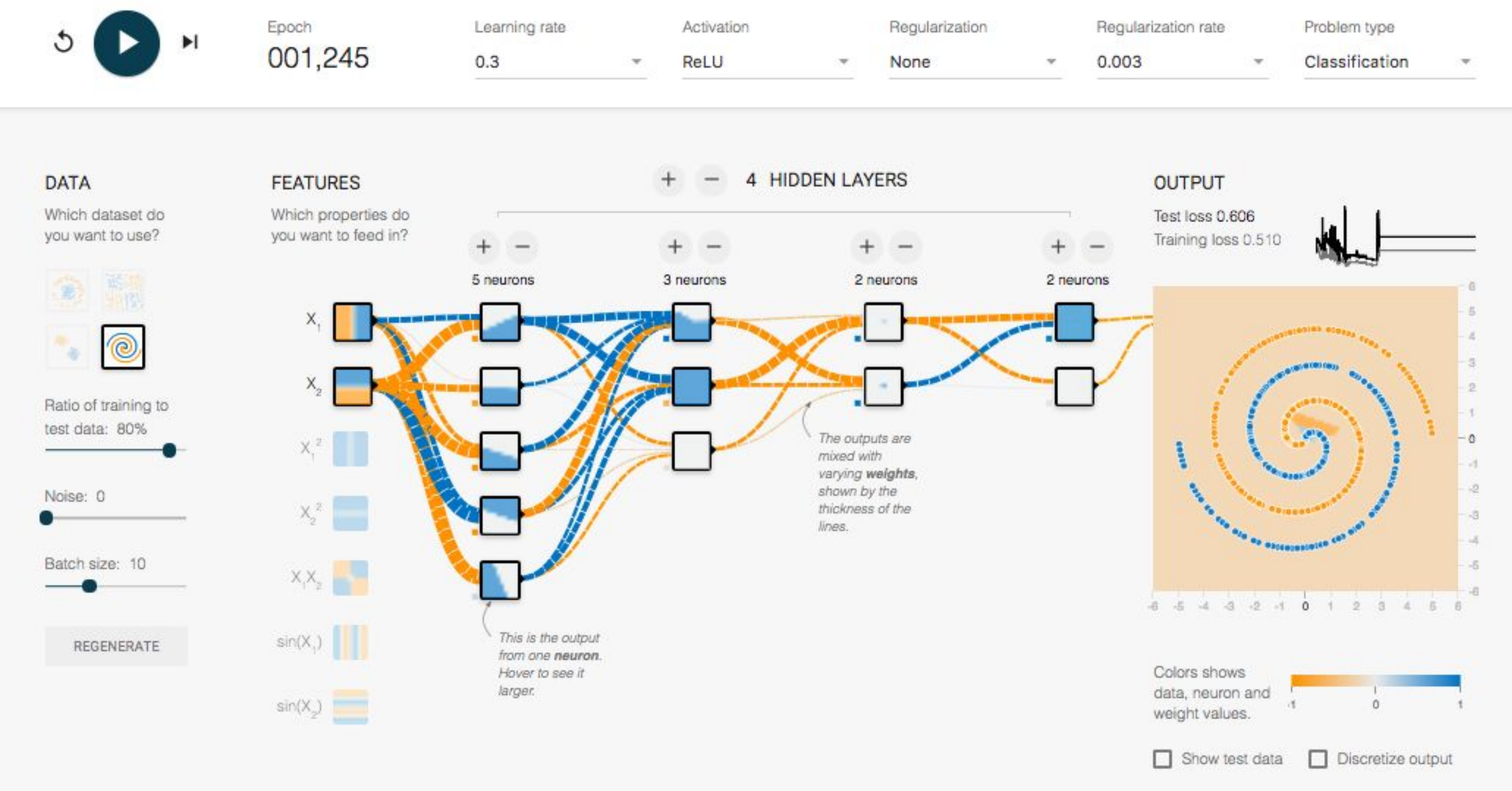
* Third-party multi-node implementations exist
** Planned to release multi-node training support

† Keras has the same capability as its backend (Theano or TensorFlow)

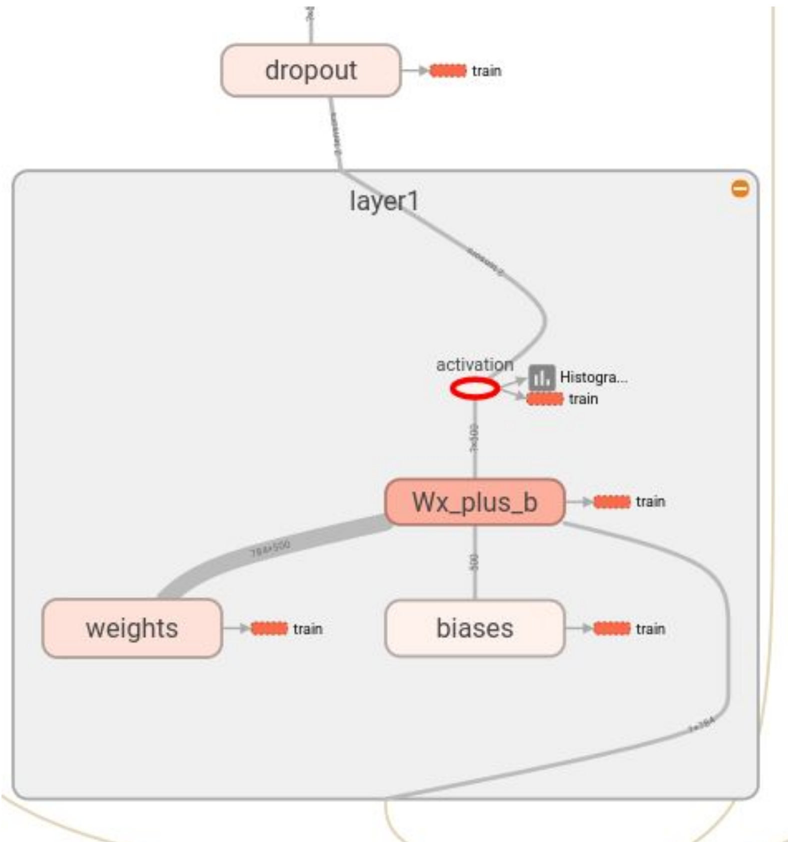
Hyperparameter Tuning and Visualization

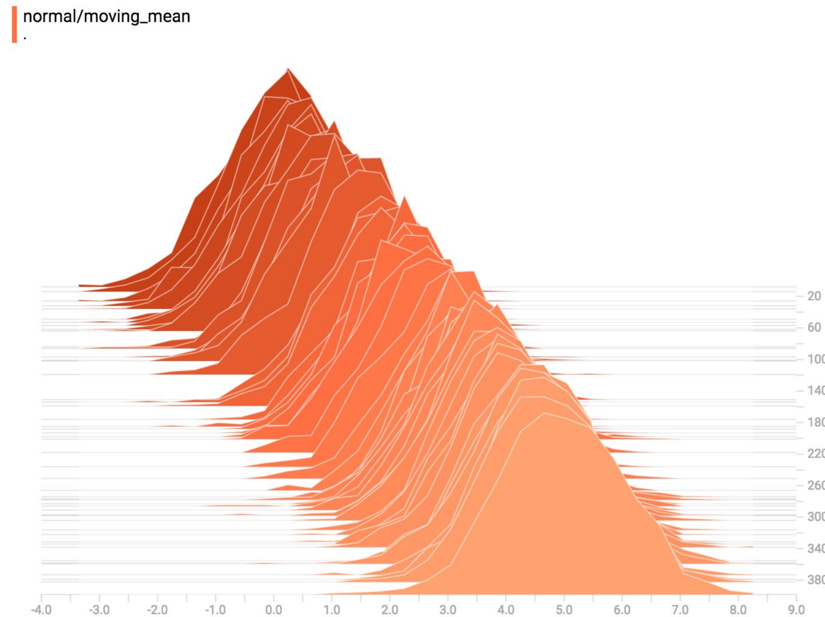
playground.tensorflow.org

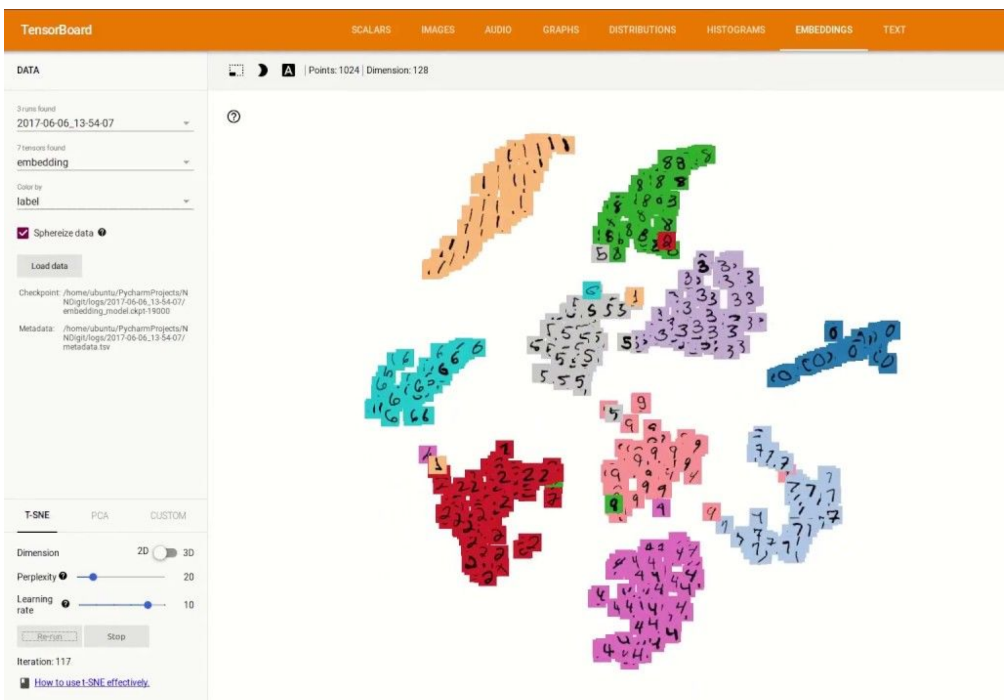
Experiment with different hyperparameters to see how they impact accuracy, speed, overfit.



TensorBoard







A dashboard for deep learning - inspect graphs, visualize training epochs, evaluate models.

A Brief Tangent on Artificial Intelligence


What is Artificial Intelligence?

"Artificial intelligence... is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and other animals.

Colloquially, the term 'artificial intelligence' is applied when a machine mimics 'cognitive' functions that humans associate with other human minds, such as 'learning' and 'problem solving'."

Wikipedia - "Artificial Intelligence"

"Artificial intelligence... is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and other animals.

Colloquially, the term 'artificial intelligence' is applied when a machine mimics 'cognitive' functions that humans associate with other human minds, such as 'learning' and 'problem solving'."

That sure is a lot of scare quotes!

Wikipedia - "Artificial Intelligence"

“[The machine consists of] an unlimited memory capacity obtained in the form of an infinite tape marked out into squares, on each of which a symbol could be printed.

At any moment there is one symbol in the machine; it is called the scanned symbol. The machine can alter the scanned symbol, and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behavior of the machine...”

Alan Turing, 1948

Turing Machine to Add Two Integers

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

Input Tape = B0110110B

Transition Functions

$(q_0, 0) = \{q_1, 0, R\}$

$(q_1, 1) = \{q_1, 1, R\}$

$(q_1, 0) = \{q_2, 0, R\}$

$(q_2, 1) = \{q_3, 0, L\}$

$(q_2, 0) = \{q_5, B, R\} = \{F\}$

$(q_3, 0) = \{q_4, 1, R\}$

$(q_4, 1) = \{q_1, 0, L\}$

$(q_4, 0) = \{q_2 0, R\}$

Computation Trace

$q_0\,0110110 \vdash 0q_1 110110 \vdash 01q_1 10110 \vdash 011q_1 0110$

$\vdash 0110q_2 110 \vdash 011q_3 0010 \vdash 0111q_4 010 \vdash 01110q_2 10$

$\vdash 0111q_3 000 \vdash 01111q_4 00 \vdash 011110q_2 0 \vdash 011110B\,q_5 \vdash \{F\}$

Figure 6

AI: A Historical Perspective

- 1936: Turing first theorizes the Turing Machine
- 1951: Minsky & Edmonds build SNARC, neural net machine
- 1956: Dartmouth workshop creates the academic field of Artificial Intelligence
- 1957: Frank Rosenblatt invents the Perceptron
- 1973: James Lighthill publishes the Lighthill Report, discrediting AI
- 1974-1993 (approx): AI Winter
- 1995: Siegelmann & Vapnik invent the SVM
- 1997: Deep Blue beats Garry Kasparov
- 1998: CNNs can recognize handwritten digits
- 2000's: GPUs become commercially available
- 2004: MapReduce popularized by Google
- 2006: Restricted Boltzmann Machines appear in Science Magazine
- 2015: TensorFlow becomes open source
- ????: Robots take over

Can machines think?

?

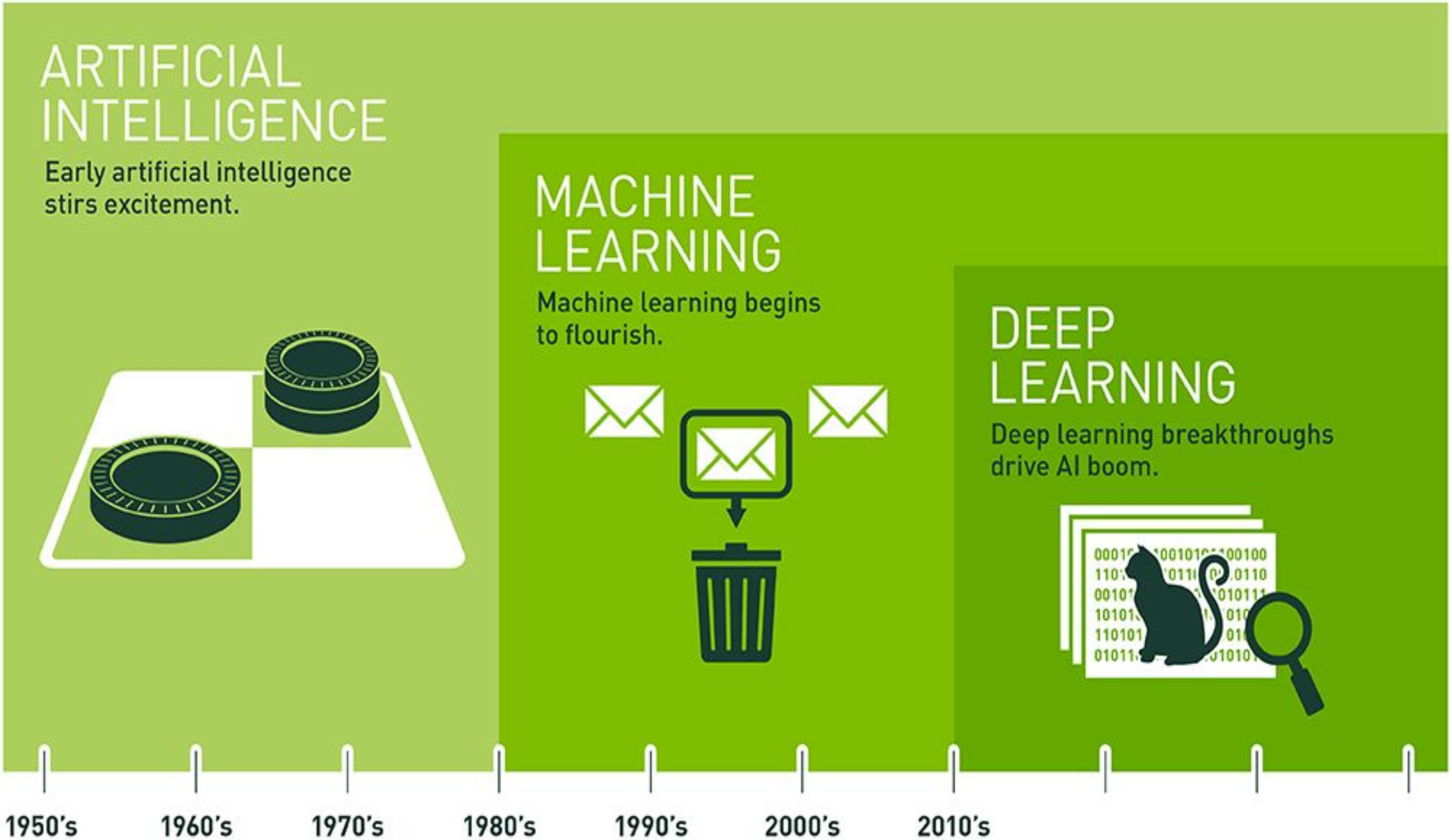
➡

What interesting problems can machines solve?

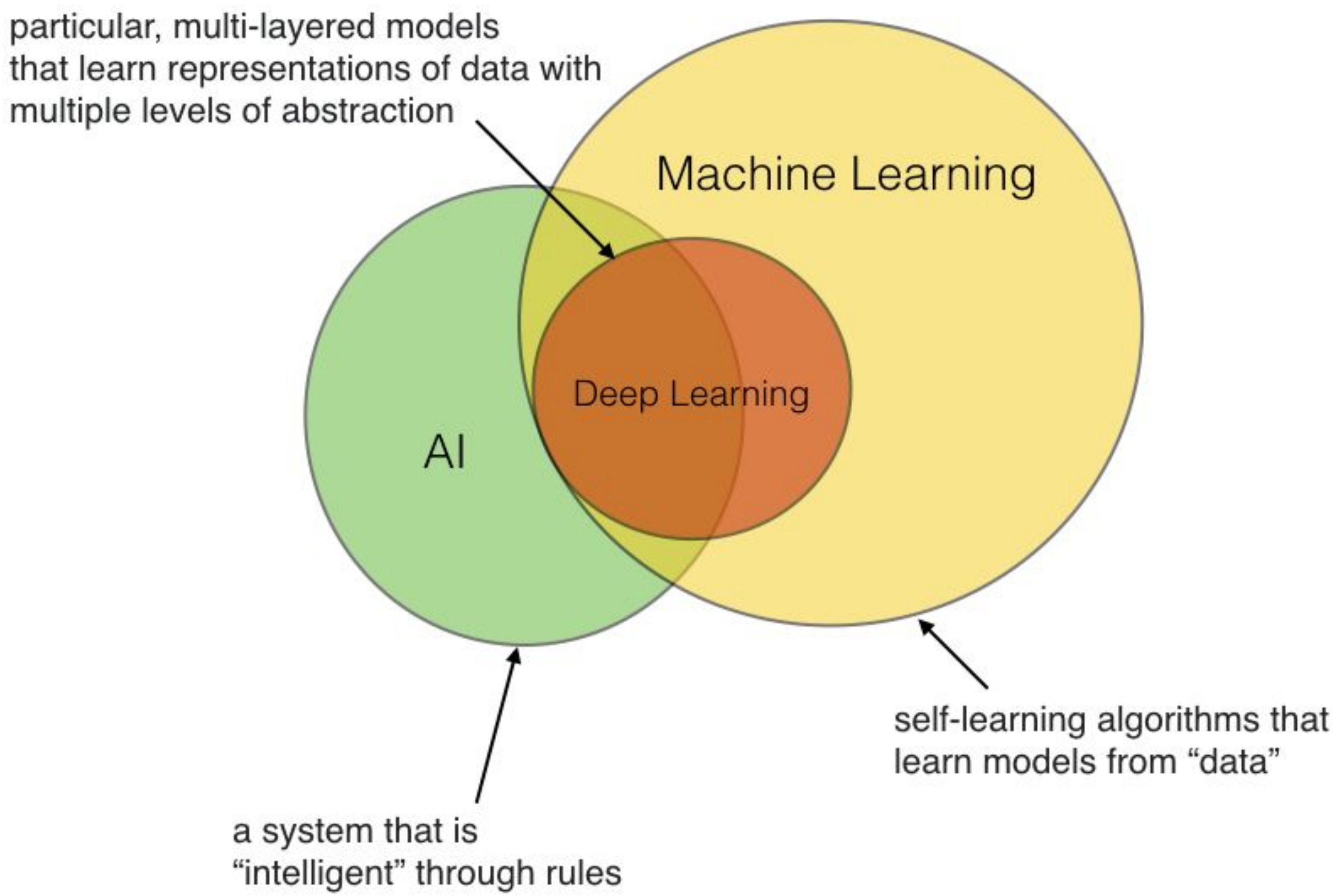
✓

Terminology

Artificial Intelligence or Machine Learning or Deep Learning ?



[Data Science Central - AI vs. ML vs. DL]



[Sebastian Raschka]

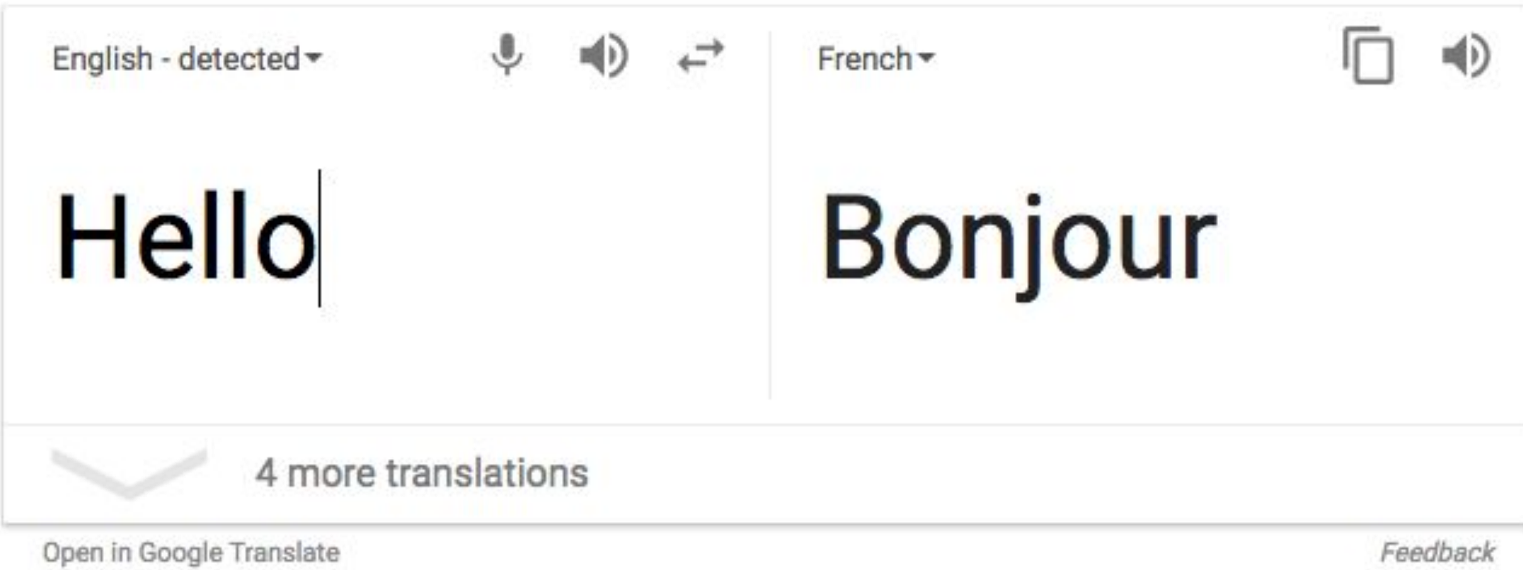
Is AI Different from ML?

Possible answers:

- Yes, AI signals the use of neural models rather than traditional machine learning models.
- Yes, AI is a buzzword, whereas ML is real.
- No, AI and ML are interchangeable ways of referring to building models that can learn from data and make predictions.
- No, AI and ML are both buzzwords, it’s all just statistics under the hood.
- Maybe, but AI is a subset of ML.
- Maybe, but ML is a subset of AI.

Neural Networks in the Wild

Machine Translation



Google Translate

Photo Captioning

picdescbot @picdescbot · Sep 18
a large passenger jet flying through a cloudy sky

commons.wikimedia.org/w/index.php?cu...

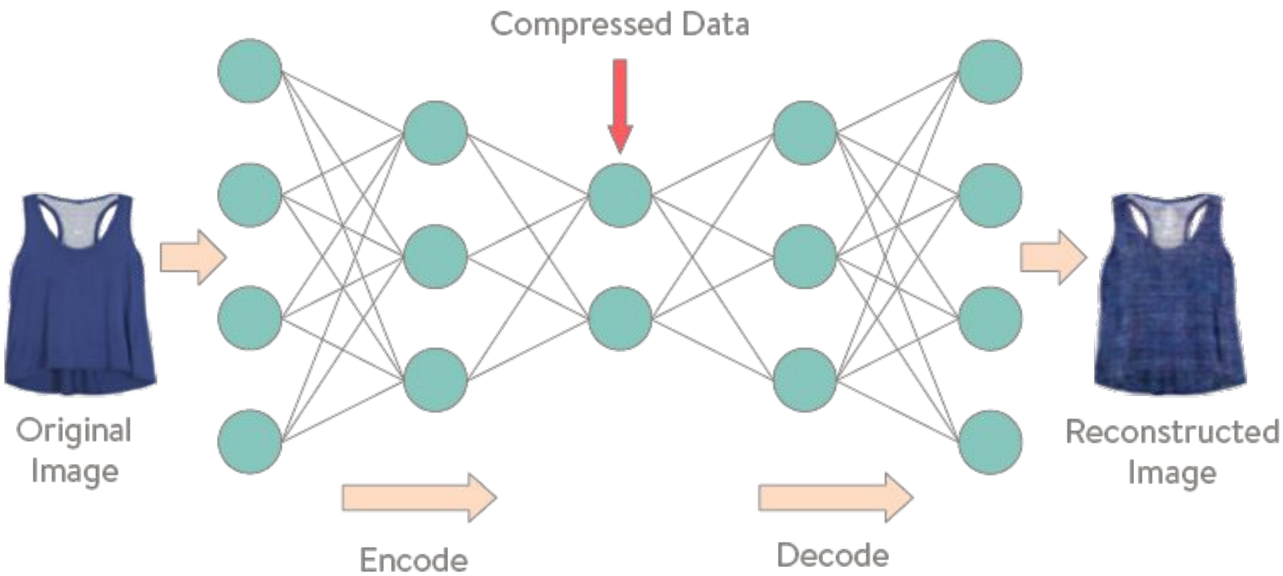
8

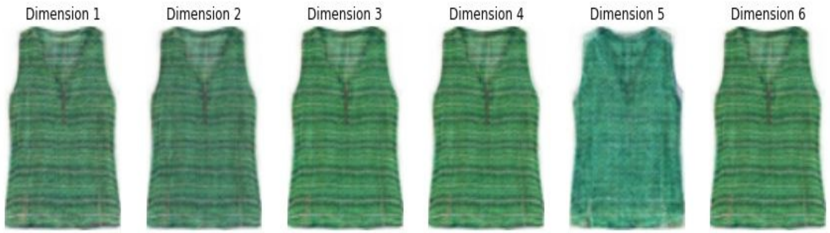
picdescbot @picdescbot · Sep 11
a person wearing sunglasses

commons.wikimedia.org/w/index.php?cu...

4

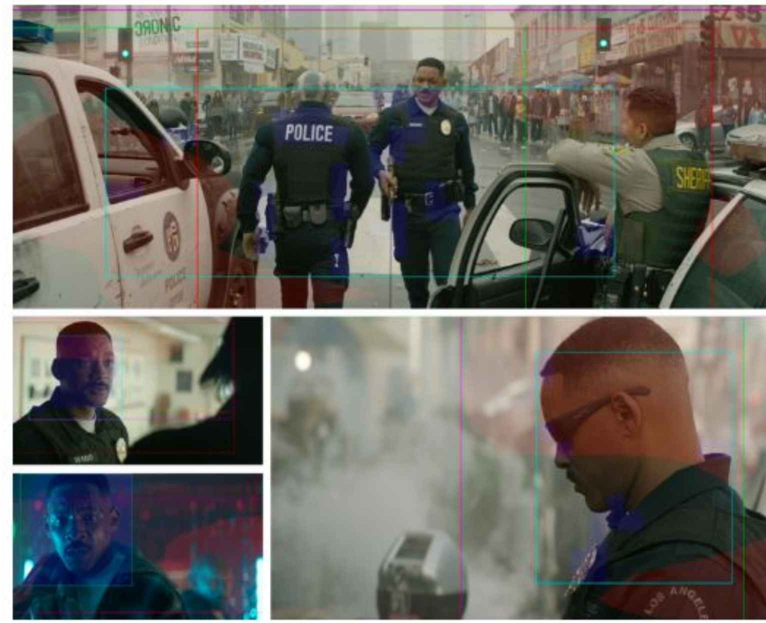
Fashion





[Stitchfix - Deep Style]

Entertainment



[Netflix]()