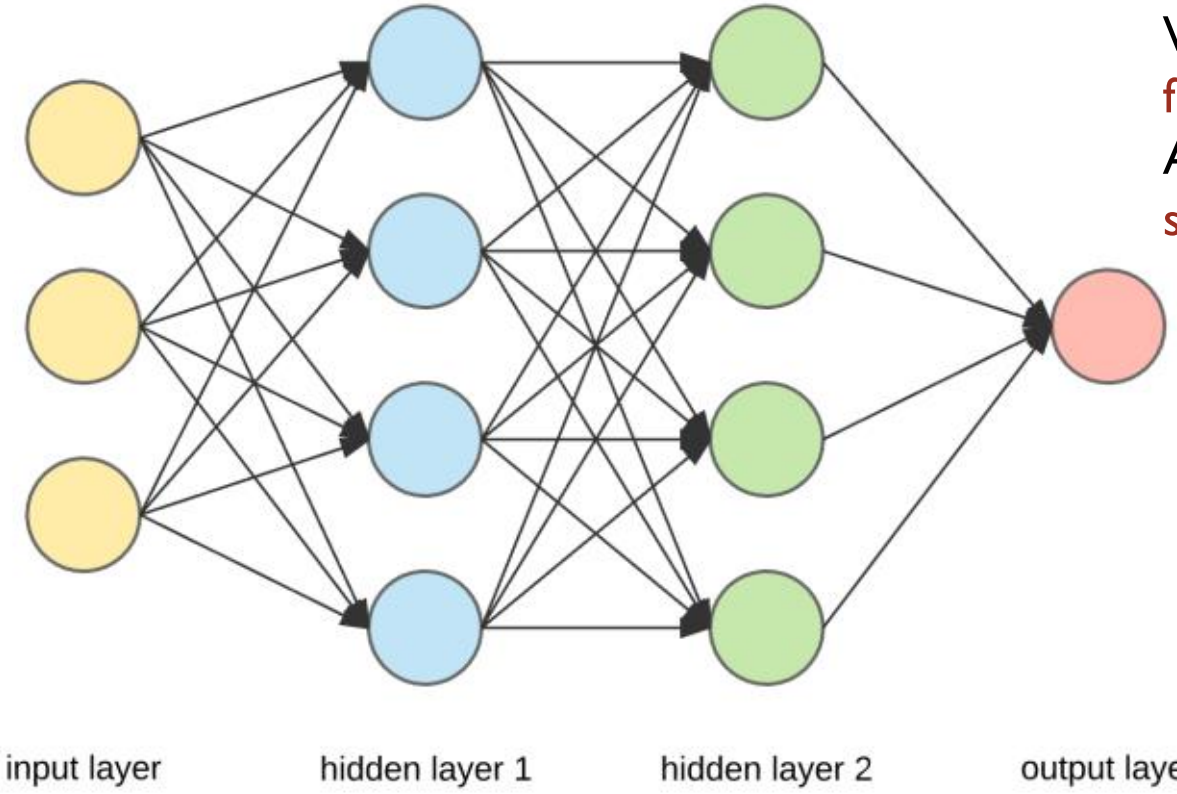
Introduction to Machine Learning

Introduction to neuronal network

Michel.Riveill@univ-cotedazur.fr

Neural Networks: Basic Definition

- An Artificial Neural Network (ANN) consists of a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections.



We learn this year only
feed forward neural network
All connections are in the
same direction

Appropriate Problems

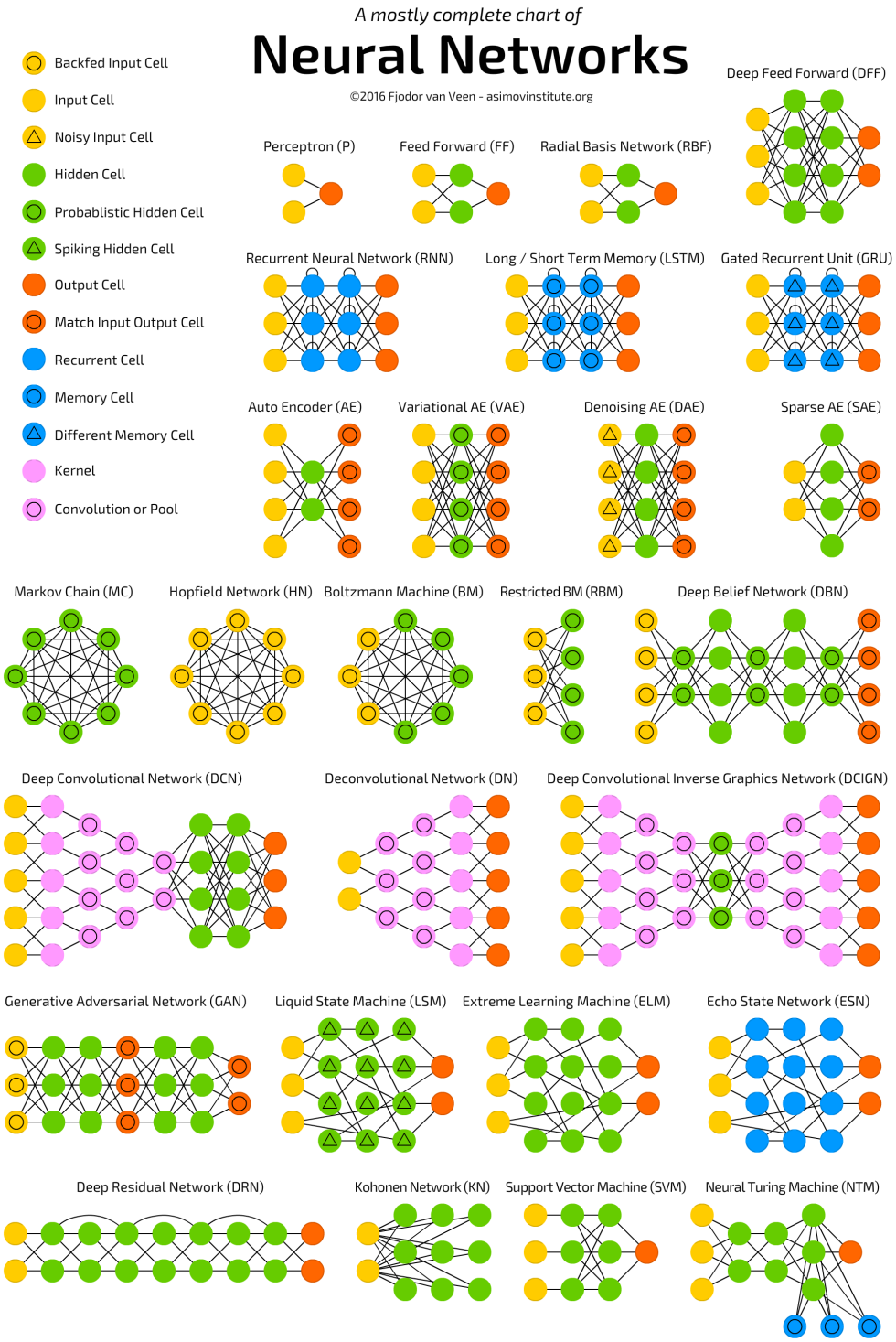
- ANN learning well-suit to problems which the training data corresponds to noisy, complex data (inputs from cameras or microphones for instance)

- Often used where data or functions are uncertain
  - Goal is to learn from a set of training data
  - And to generalize from learned instances to new unseen data

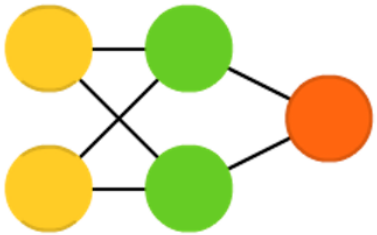
- Neural networks can be used in different fields. The tasks to which artificial neural networks are applied tend to fall within the following broad categories:
  - Function approximation, or regression analysis, including time series prediction and modeling.
  - Classification, including pattern and sequence recognition, novelty detection and sequential decision making.
  - Data processing, including filtering, clustering, blind signal separation and compression.

Topologies of Neural Networks

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



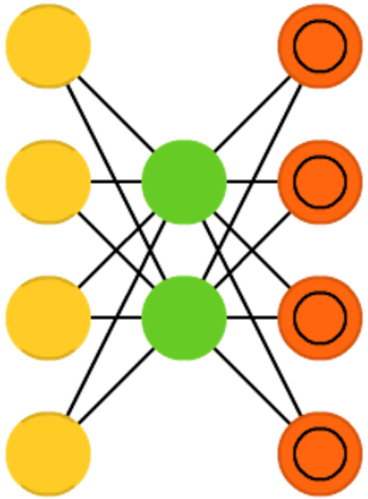
Topologies of Neural Networks



Feed forward or Perceptrons (P)
Original paper: *Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6 (1958): 386.*

- Feedforward: No loops, input → hidden layers → output
- Supervised networks use a "teacher"
  - The desired output for each input is provided by user

Topologies of Neural Networks



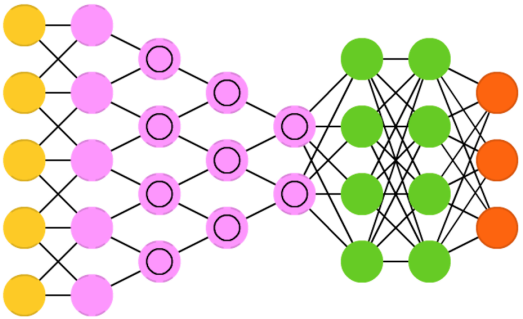
Autoencoders (AE) The basic idea behind autoencoders is to compress information automatically

Original paper: *Bourlard, Hervé, and Yves Kamp. "Auto-association by multilayer perceptrons and singular value decomposition." Biological cybernetics 59.4-5 (1988): 291-294.*

Unsupervised networks find hidden statistical patterns in input data

Topologies of Neural Networks

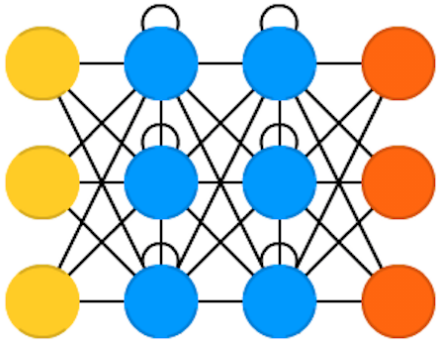
- In mathematics (in particular, functional analysis), convolution is a mathematical operation on two functions (f and g) that produces a third function (f * g) that expresses how the shape of one is modified by the other.
- The term convolution refers to both the result function and to the process of computing it.



Convolutional neural networks (CNN) are quite different from most other networks. They are primarily used for image processing but can also be used for other types of input such as audio or text. Each node only concerns itself with close neighbouring cells.

Original paper: *LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.*

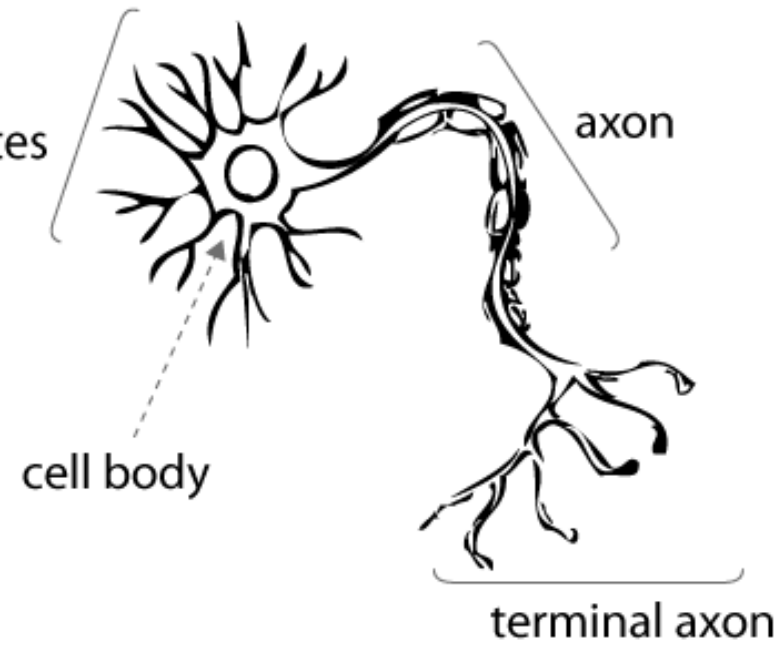
Topologies of Neural Networks



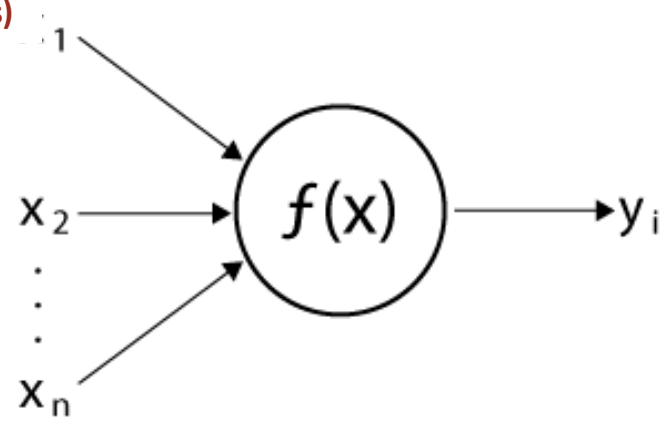
Recurrent neural networks (RNN) they have connections between passes, connections through time. Neurons are feed information not just from the previous layer but also from themselves from the previous pass.
Original paper: *Elman, Jeffrey L. "Finding structure in time." Cognitive science 14.2 (1990): 179-211.*

The Neuron Metaphor

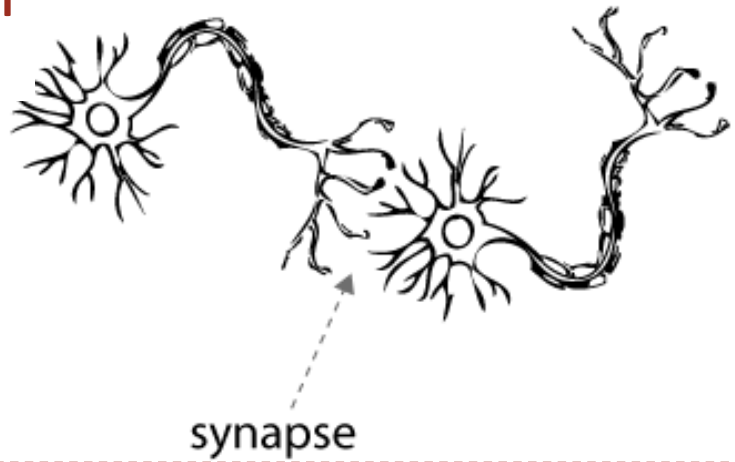
Human neuron



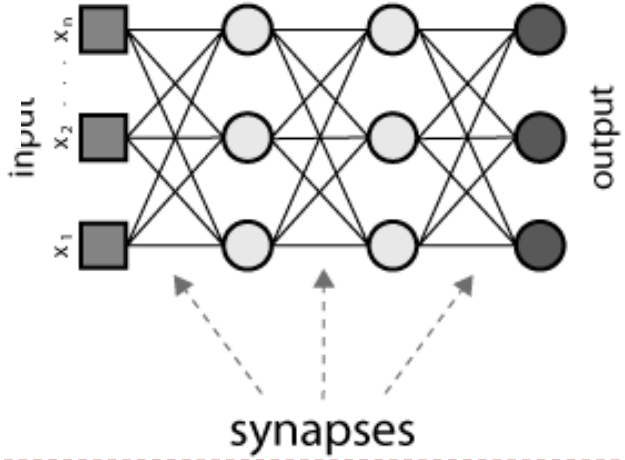
Artificial neuron
(n features)



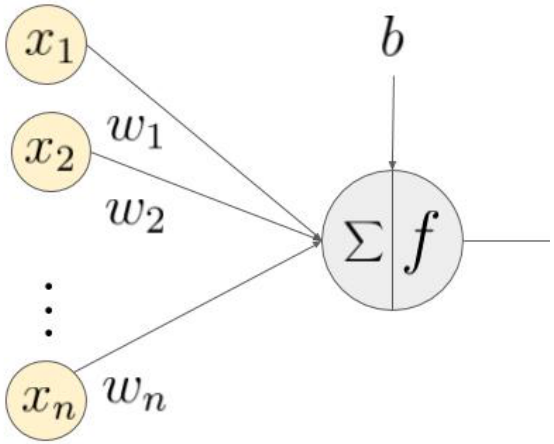
Biological synapse



Artificial Network
(n features)



Artificial neuron



$$\sigma\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

n: number of features

- σ: output function
  - Linear: $\sigma(x) = x$ → I linear neuron is équivalent to Linear Regression
  - Perceptron: $\sigma(x) = 1\ if\ x > 0\ else\ 0$
  - Relu: $\sigma(x) = \mathrm{x}\ if\ x > 0\ else\ 0$
  - Sigmoid: $\sigma(x) = \frac{1}{1 - e^{-x}}$ → I sigmoid neuron is équivalent to Logistic Regression
  - …
  - Full list: https://en.wikipedia.org/wiki/Activation_function

Multilayer Neural Networks

Linearly separable

- A single-layer perceptron network can classify linearly separable sets
  - Perceptron: $\sigma(output) = -1 \; if \; output < 0 \; else \; output$

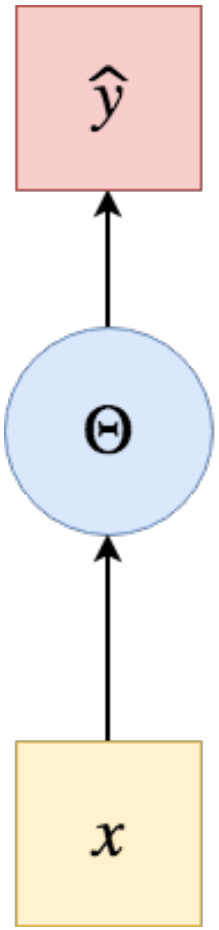
x	Not x
1	0
0	1

Solution: $\beta_1 = -1$ and $\beta_0 = 0.5$

if $\beta_1 x + \beta_0 > 0$

then output = 1

else output = 0



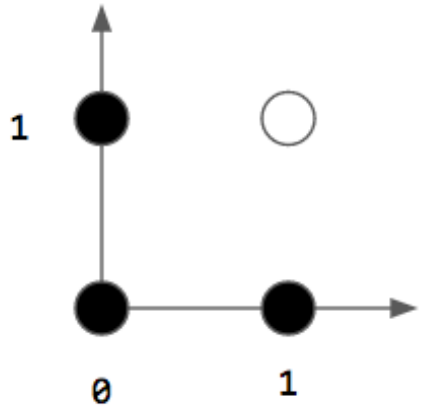
Linearly separable

- A single-layer perceptron network can classify linearly separable sets

x_1	x_2	x_1 and x_2
1	1	1
0	1	0
1	0	0
0	0	0

SOLUTION: $\beta_1 = 1, \beta_2 = 1, \beta_0 = -1.5$

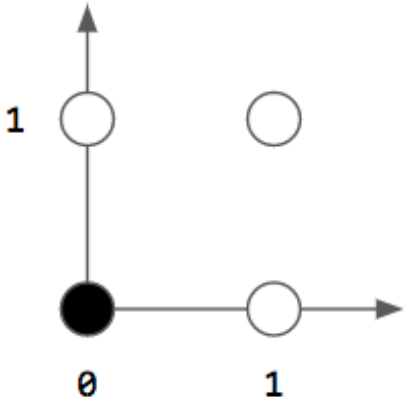
AND

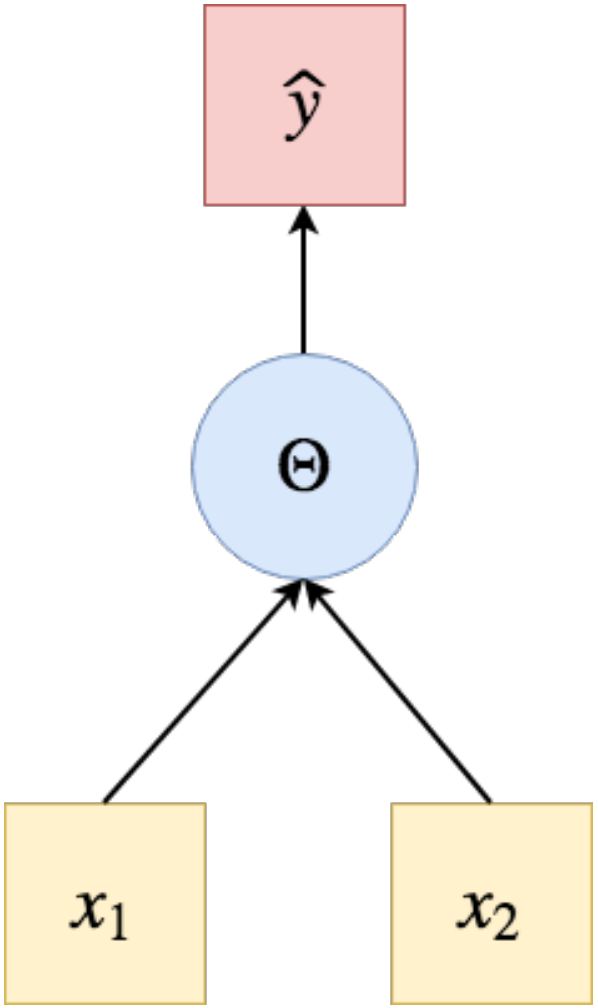


x_1	x_2	x_1 or x_2
1	1	1
0	1	1
1	0	1
0	0	0

SOLUTION: $\beta_1 = 1, \beta_2 = 1, \beta_0 = -0.5$

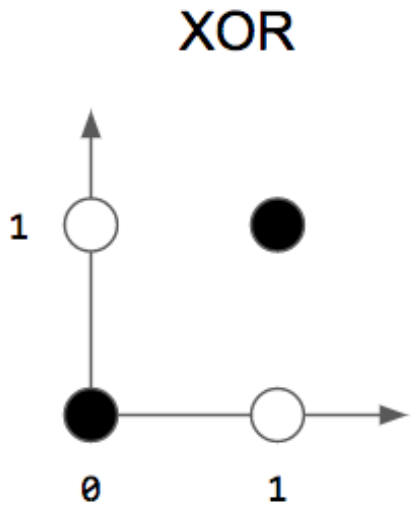
OR

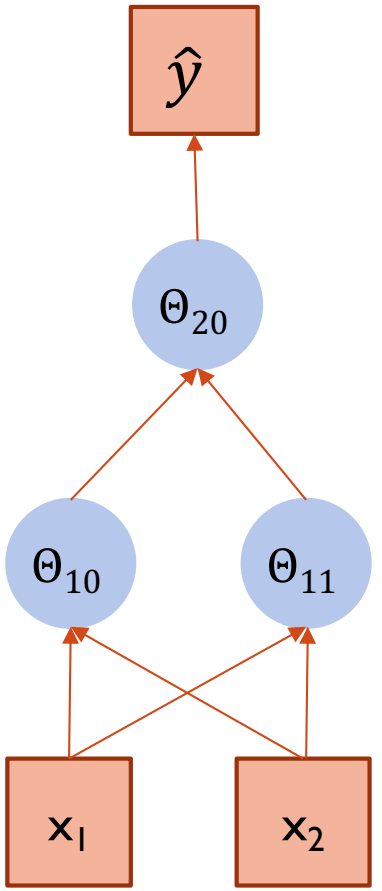




Non Linearly separable

x_1	x_2	x_1 and x_2
1	1	1
0	1	0
1	0	0
0	0	0





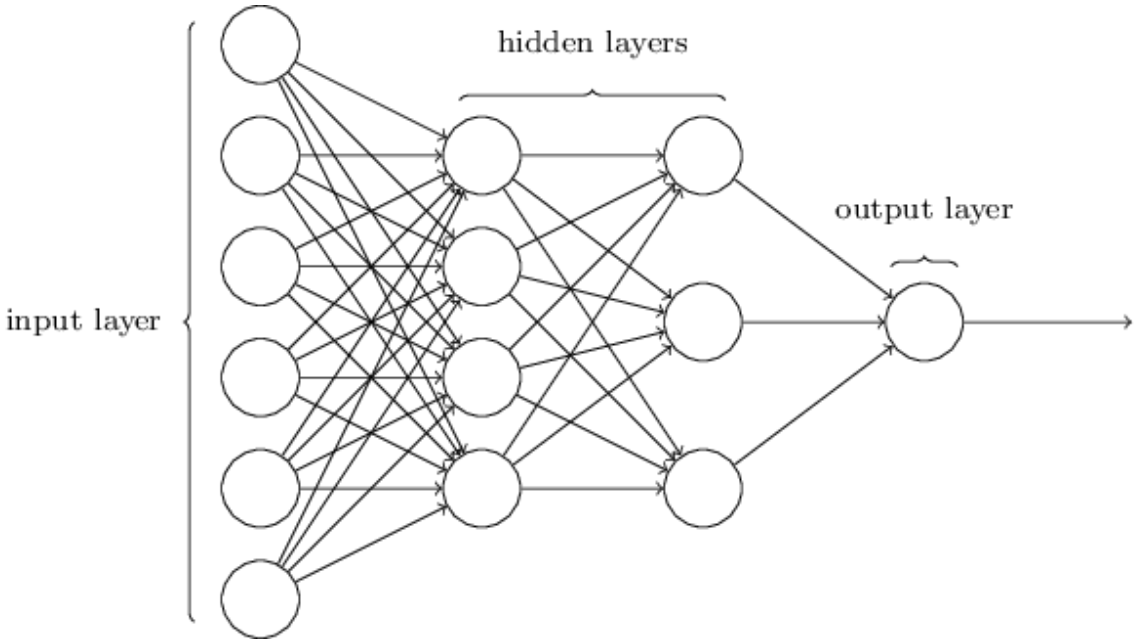
SOLUTION: ß = [[1, 1, -0.5],
[1, 1, -1.5],
[1, -2, -0.5]

i.e. Output is 1 if and only if

(x1+x2-0.5>0) + -2*(x1+x2-1.5>0) - 0.5 > 0

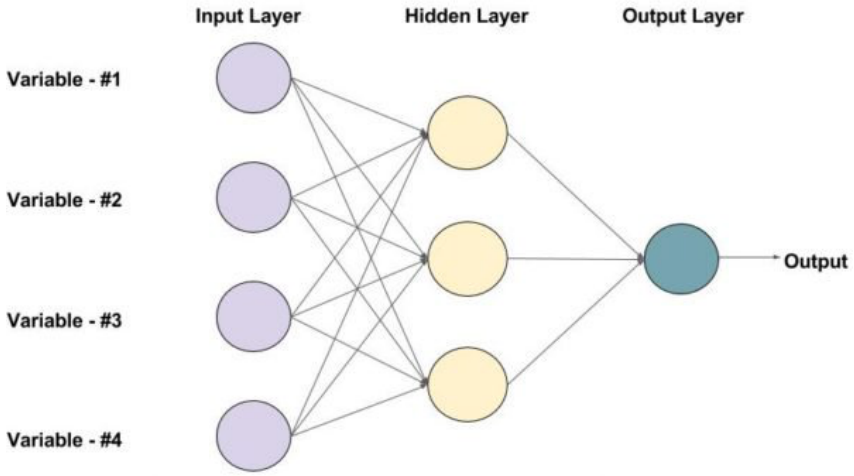
Multilayer networks

- Cascade neurons together
  - The output from one layer is the input to the next
  - Each Layer has its own sets of weights
  - A L-layer network corresponds to a network with L layers having some weights to compute (we do not count the input layer but we count the output layer)
  - The layers between the input layer and the output layer are called the hidden layers.



Linear regression neural networks

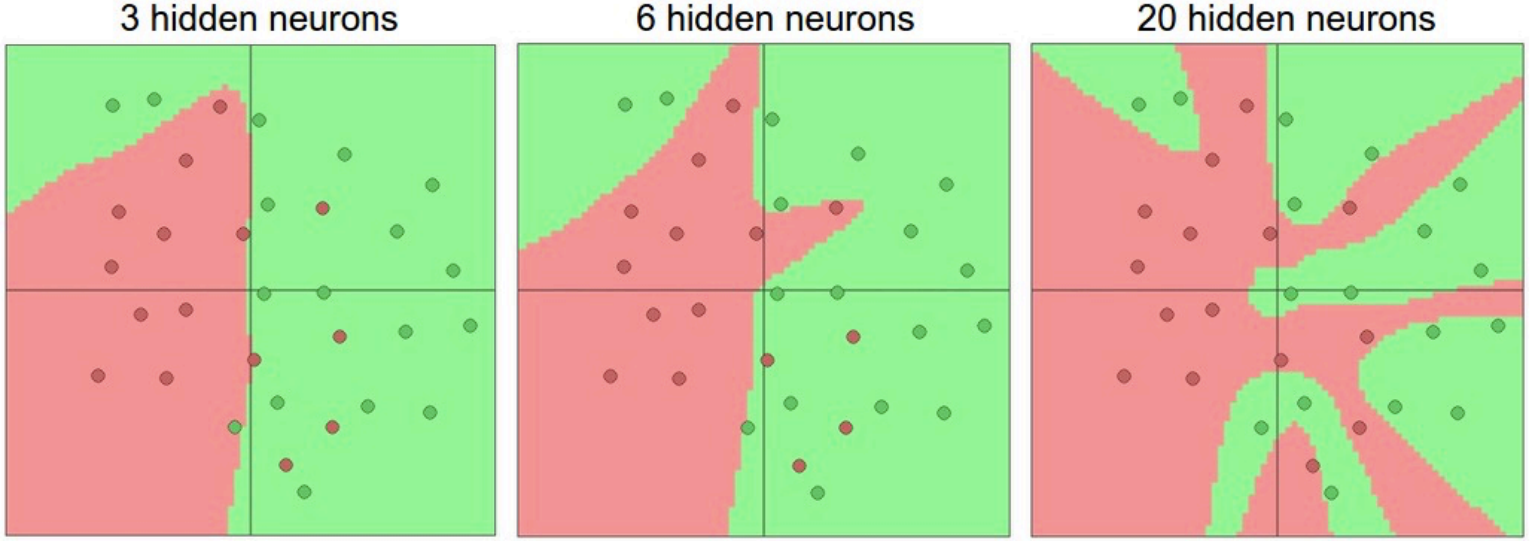
- What happens when we arrange **linear neurons** in a multilayer network?
  - Linear neurons: $\sigma(x) = x$
  - Nothing special happens.
  - The product of two linear transformations is itself a linear transformation



- $output = \sum_j w_j^{(2)} \left(\sum_i w_i^{(1)} x_i \right) = \sum_j \sum_i w_j^{(2)} w_i^{(1)} x_i = \sum_i \sum_j w_j^{(2)} w_i^{(1)} x_i$
- $output = \sum_i (\sum_j w_j^{(2)} w_i^{(1)}) x_i = \sum_i \widehat{w}_i x_i = \widehat{w}.x$

Neural Networks

- We must introduce non-linearities to the network
  - Non-linearities allow a network to identify complex regions in space
  - replace linear active function for non linear active function
- A one-layer cannot handle XOR
  - More layers can handle more complicated spaces – but require more parameters
  - Each node splits the feature space with a hyperplane
- A two-layer network can represent any convex region
  - provided that the number of neurons is large enough in the hidden layer

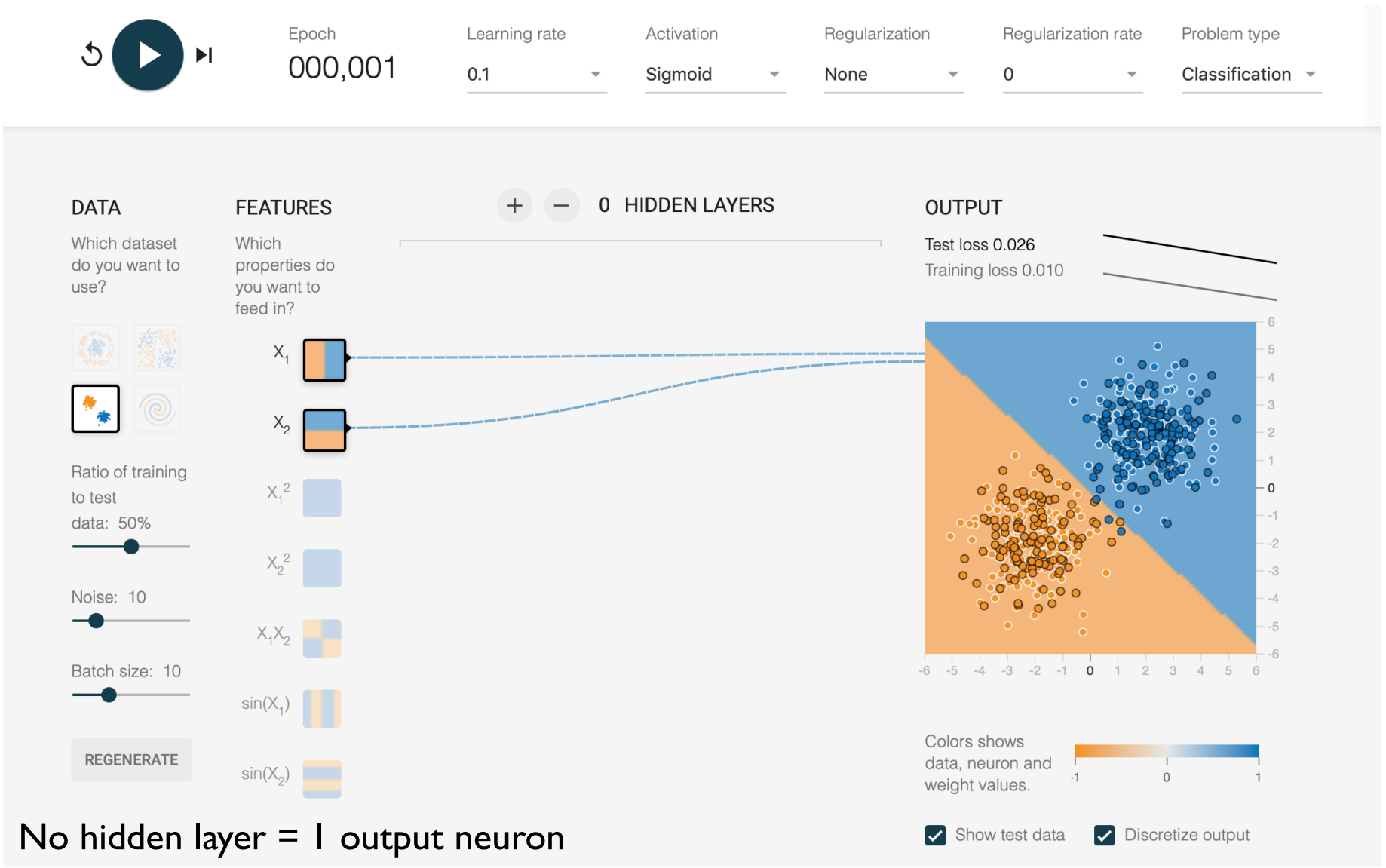


It's time to play

- Go to playground: http://playground.tensorflow.org
  - Choose classification problem
  - Fix:
    - Ratio: 50%
    - Noise: 10%
    - Regularization: None
    - Batch size: 10
    - Other paramters: Learning rate, Activation
  - When you fit the model, stop training when test loss < 0.05
    - make a screenshot (or note epoch number, training loss and test loss)

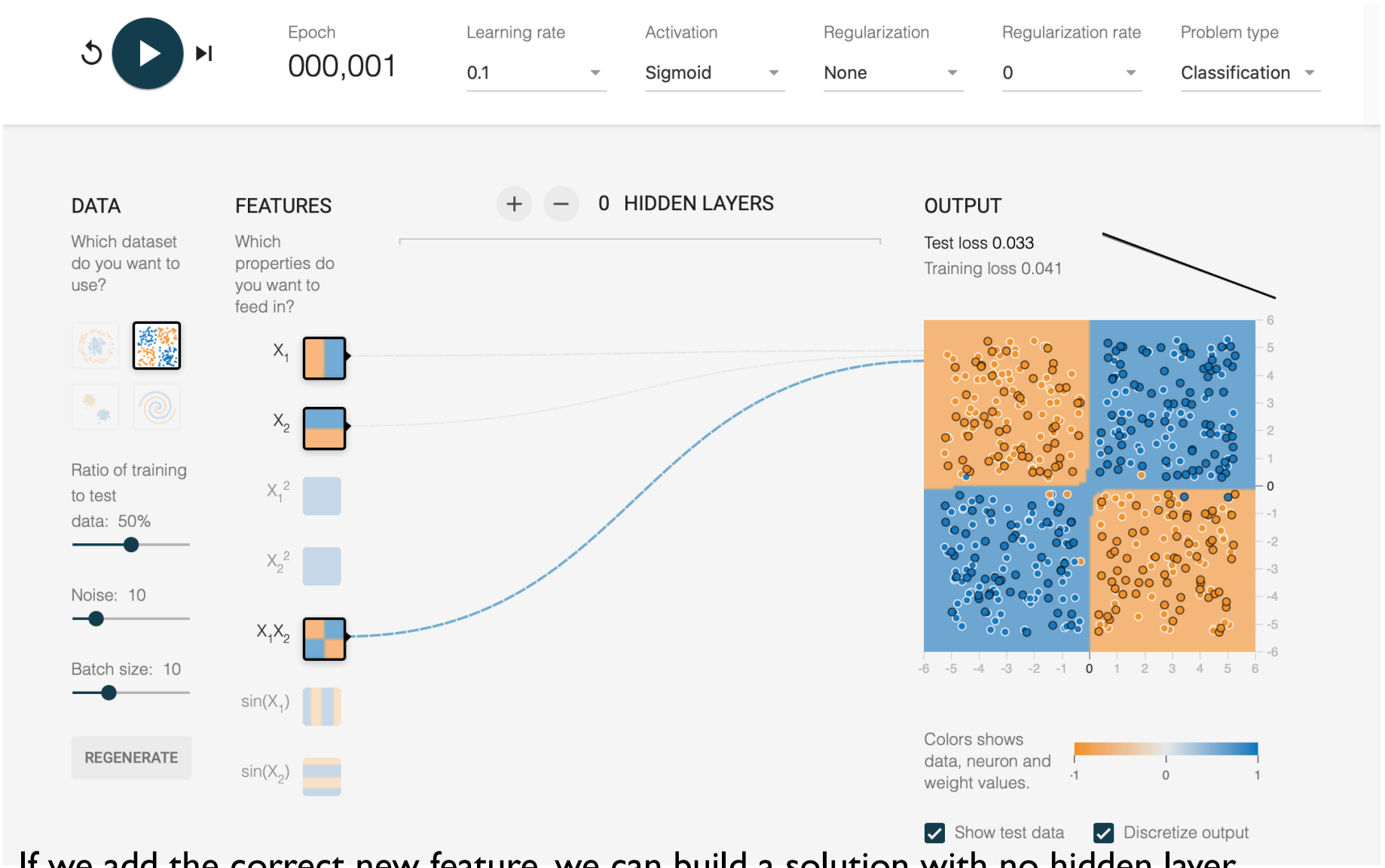
1. Choose a linearly separable dataset
   - Build the smaller network to fit them
2. Choose a non linearly separable dataset
   1. Add some extra feature in order to fit them with only one neuron
      - extra feature=x^2, xy, sin(x)
   2. Remove the extra feature and try to build the smaller network to fit them
   3. With the same example
      - Increase learning (0,1 → 1) rate and then, decrease learning (0,1 → 0,01)

linearly separable dataset



No hidden layer = 1 output neuron

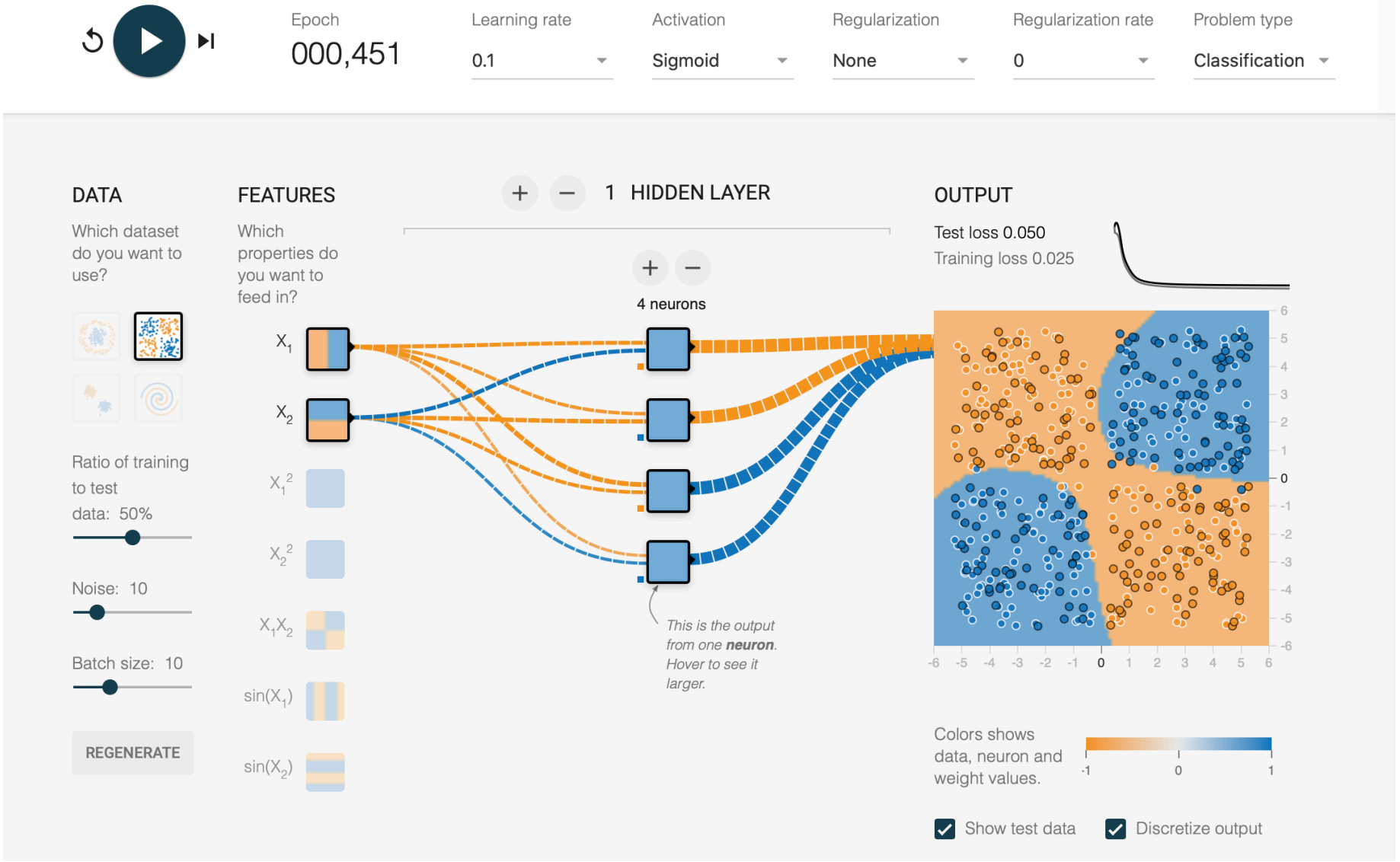
Non linearly separable dataset



If we add the correct new feature, we can build a solution with no hidden layer

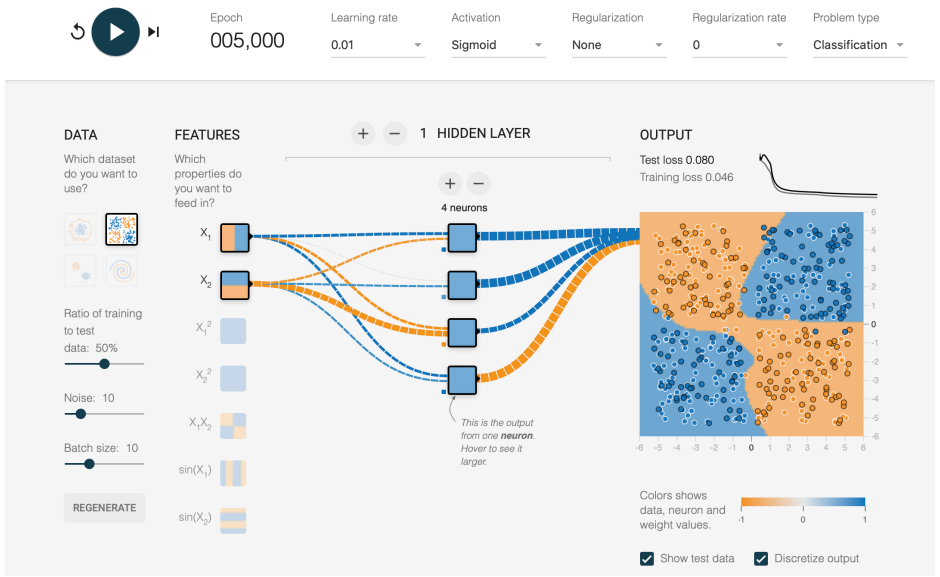
i.e. with only 1 output neuron

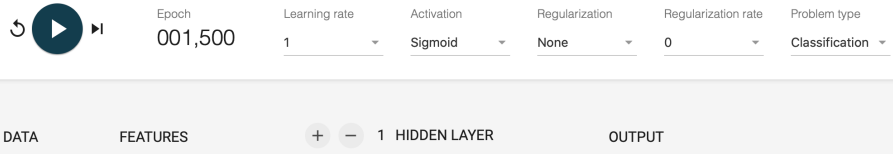
Non linearly separable dataset



Non linearly separable dataset

- Same example
  - Lr = 0,01 → after 5.000 epoch, loss = 0.08
  - Lr = 0,1 → 450 epochs, loss = 0.05
  - Lr = 1 → 1.500 epochs, loss around 1.3 (no convergence)

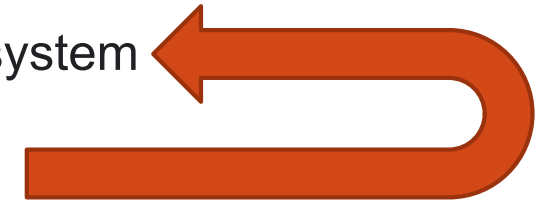




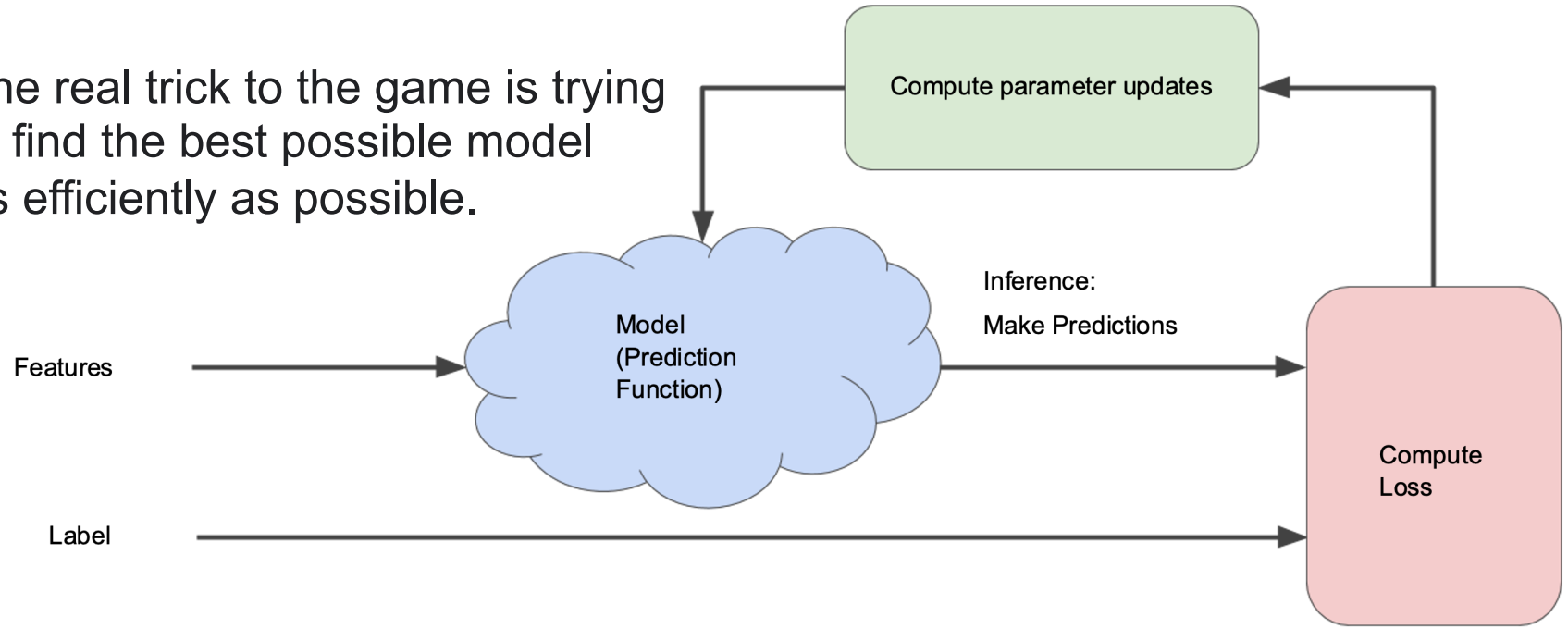
Feed-forward network
How does the network learn?

How does the network learn?

- Iterative learning = "Hot and Cold" kid's game for finding a hidden object.
- In this game, the "hidden object" is the best possible model.
- You choose a random value
- You submit the value to the model and wait for the system to tell you what the loss is.
- Then, you'll correct the initial value and try
- OK, you're getting warmer. Actually, if you play this game right, you'll usually be getting warmer.

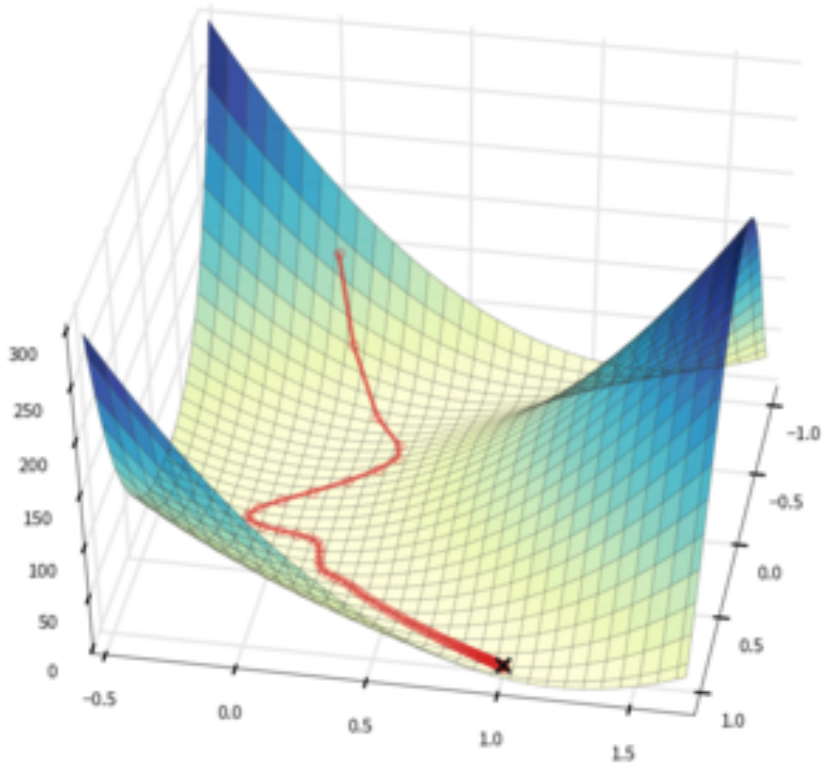


- The real trick to the game is trying to find the best possible model as efficiently as possible.



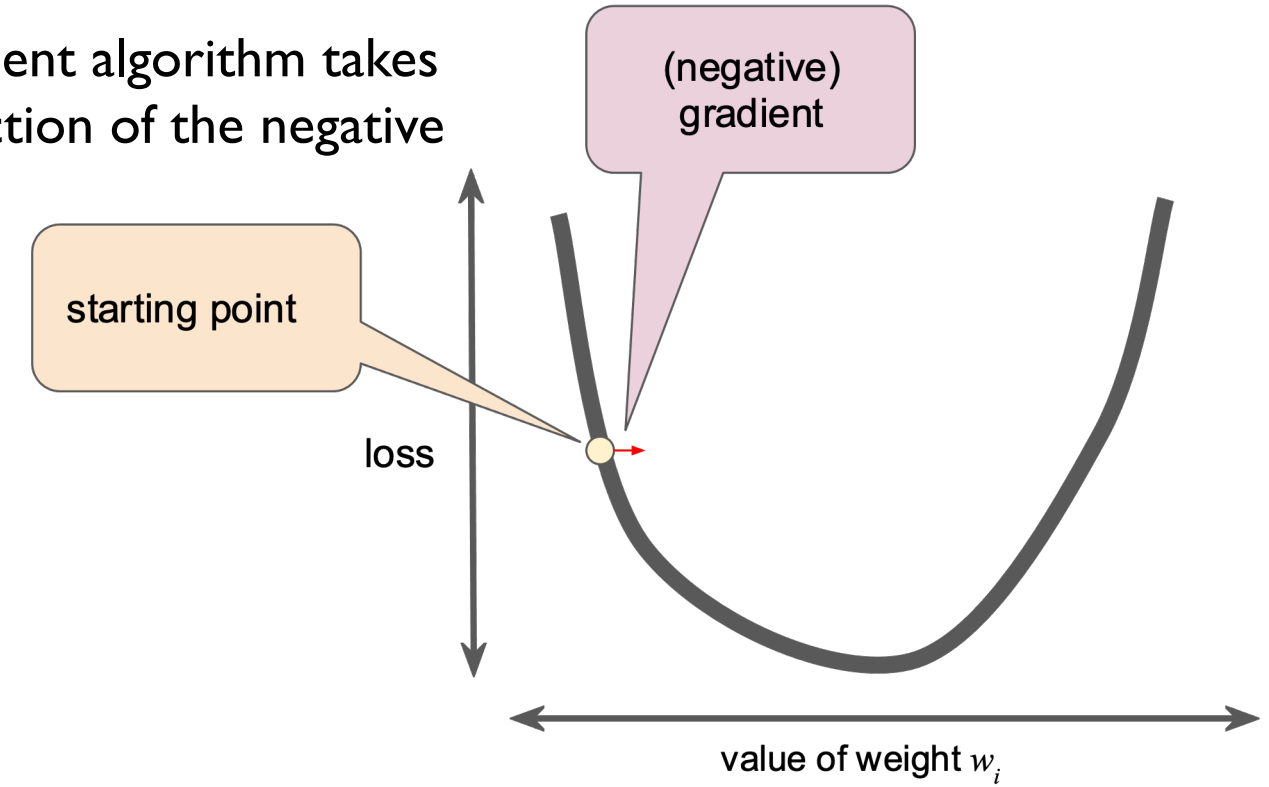
How does the network learn?

- In order to play this game we use "Gradient Descent"
  - Gradient Descent = help us to find the minimum of a function
- In Neural Network
  - Find the minimum of the loss function
  - Regression : MAE, MSE
  - Classification : CrossEntropy



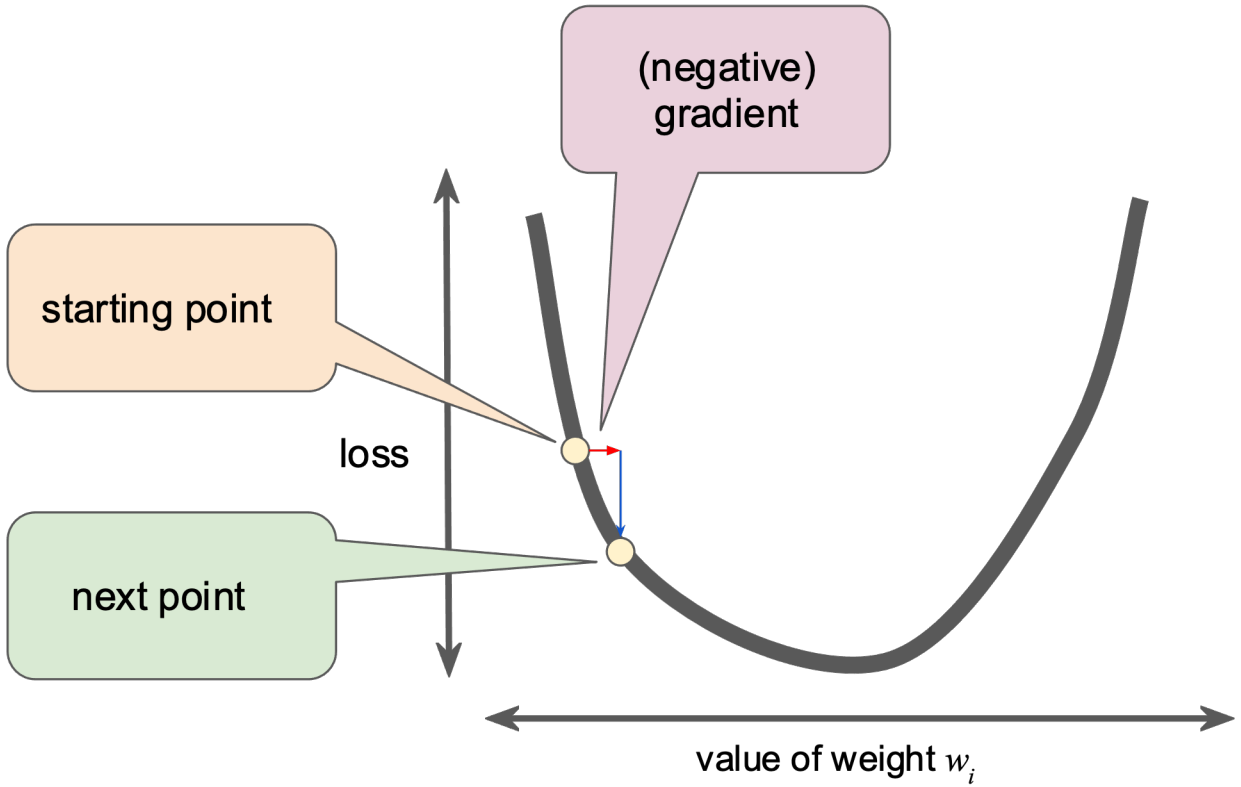
Gradient descent

- A gradient is a vector, so it has both of the following characteristics:
  - a direction
  - a magnitude
- The gradient always points in the direction of steepest increase in the loss function.
- The gradient descent algorithm takes a step in the direction of the negative gradient in order to reduce loss as quickly as possible.



Gradient descent

- To determine the next point along the loss function curve, the gradient descent algorithm adds some fraction of the gradient's magnitude to the current point.
- The gradient descent then repeats this process, edging ever closer to the minimum.



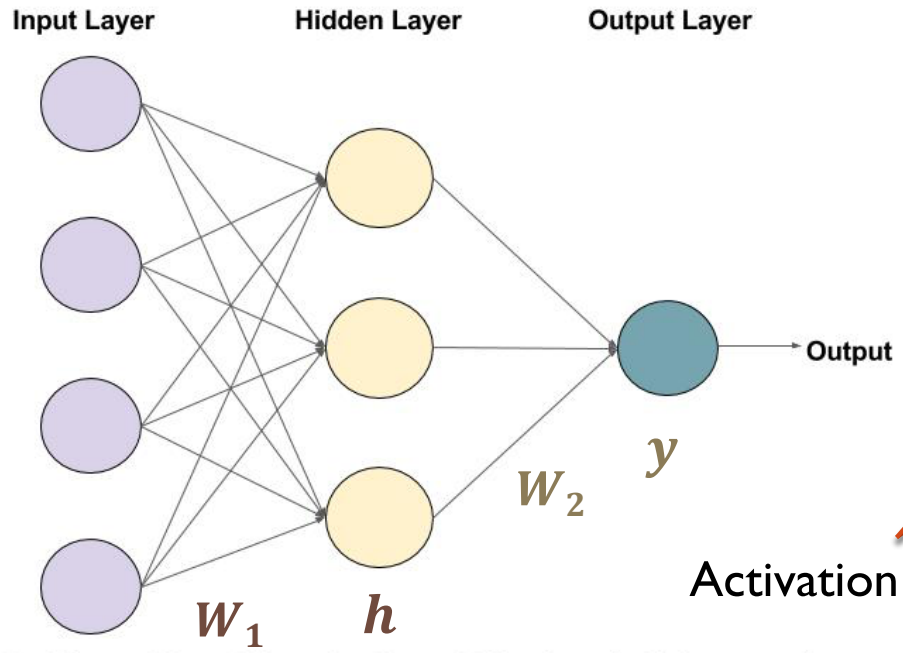
Gradient descent

- Objective: find a minimum of a differentiable function

- Main variables
  - Choose the initial point x_0
  - Choose the learning rate $\alpha \geq 0$ -- for the new point
  - Choose the tolerance level $\varepsilon \geq 0$ -- stop the algorithm

- Execute the following algorithm
  1. Calculate $\nabla f(x_k)$
  2. Stop if $\|\nabla f(x_k)\| \leq \varepsilon$
  3. Calculate the new value of x: $x_{k+1} = xk - \alpha \nabla f(x_k)$
     - α is correct, if $f(x_{k+1}) < f(x_k)$

How a neural network learns with gradient descent



Predictions are fed forward through the network to classify

Weights

$$h_i = \sigma_1(W_{1i}x + b_{1i}) = \sigma_1(W_{1i}.x)$$

$$y = \sigma_2(W_2 h + b_2) = \sigma_2(W_2 . h)$$

$$y = \sigma_2\left(\sum_i \sigma_1(W_{1i}.x)\right) = \sigma_W(x)$$

Activation functions

h = 4 neurons = 3 x (input +1) weights = 3 x (4 + 1) = 15
y = 1 x (previous level + 1) = 1 x (3 + 1) = 4

---

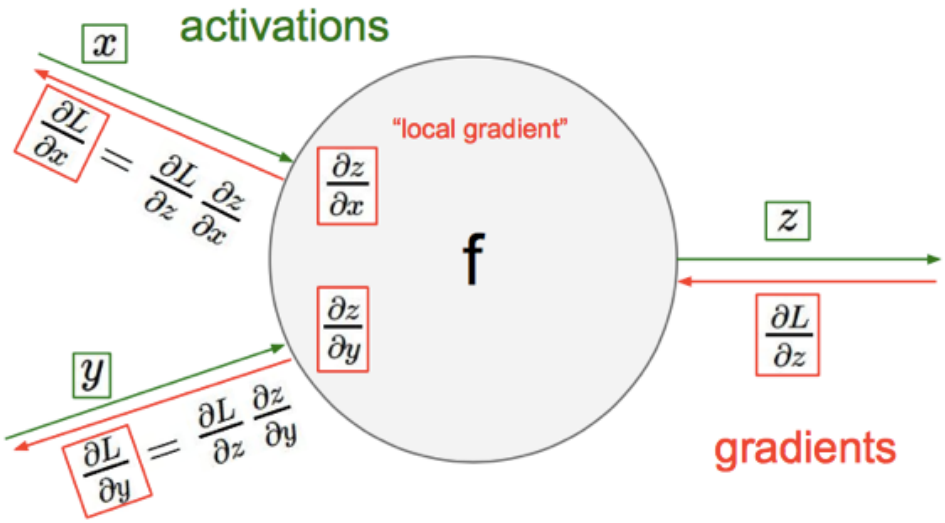
19 learnable **parameters**

How a neural network learns with gradient descent

- If we have only one neuron → it's like a regression
- We need a cost/loss function
  - For regression: MSE
  - MSE = $J_W(y_i, xi) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \sigma_W(x_i)\right)^2$
  - *It could be rewrited in*
    - $J_W(y_i, xi) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, xi, W)^2$
    - *For a given dataset depen only of the weights*
      - $J_W(y_i, xi) = J_{(y_i, x_i)}(W)$
- the good model is the one that minimizes this function,
  - for the dataset that we own
  - this can be done using the gradient descent approach

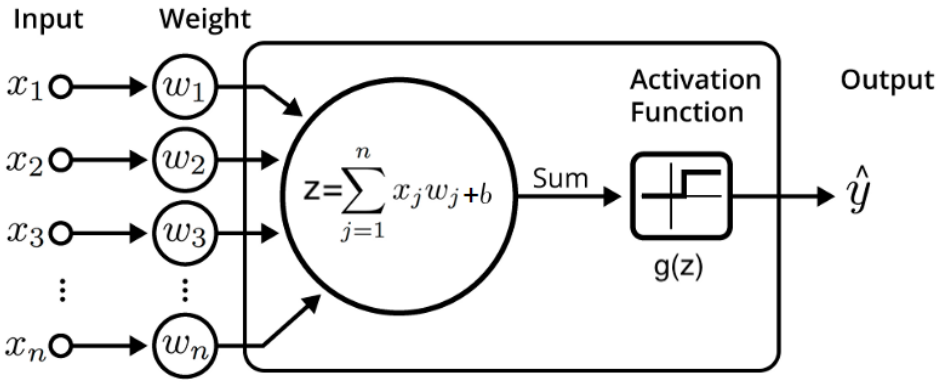
Gradient Descent for neural network

- A network is define by
  - X: input (vector of features
  - y; target output
  - W: a set of weigths
  - σ: a set of activation function
  - J: loss function or cost function



- Step 1.
  - Compute the prediction: $\hat{y} = \sigma(W.X)$
  - Error calculation : $J(y, \hat{y}) = \frac{1}{N}(y - \hat{y})2$ // mean squared error, for exemple
- Step 2
  - computes the gradient in weight space with respect to a loss function: $\frac{\mathrm{d}}{\mathrm{d}W_j^{old}} J(W)$
- Step 3
  - Adjust weigth: $W_j^{new} = W_j^{old} - \alpha \frac{\mathrm{d}}{\mathrm{d}W_j^{old}} J(W)$

Gradient descent for deep learning
Case of one neuron



Notation:

- $x_0 = 1$ and $b = w_0\,(bias)$
- $z = \sum_{i=0}^{n} x_i w_i$
- $\hat{y} = a = g(z)$

- $J(y, \hat{y}) = (y - \hat{y})^2$ A cost function must be derivable
- The objective of course is to minimize the cost function and therefore to find b and w
- In order to do that, we use a chain rule
  - $\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial a} * \frac{\partial a}{\partial z} * \frac{\partial z}{\partial wi}$

Gradient descent for deep learning Case of one neuron

- Let's consider earch term alone:
  - $\frac{\partial J}{\partial a} = \frac{\partial (y-\hat{y})^2}{\partial a} = \frac{\partial (y-a)^2}{\partial a} = -2(y-a) = 2(\hat{y}-y)$
    - $\frac{\partial J}{\partial a}$ = derivative of the cost function
  - $\frac{\partial a}{\partial z} = \frac{\partial g(z)}{\partial z} = g'(z) = z(1-z)$, for sigmoid activation function
    - $\frac{\partial a}{\partial z}$ = derivative of the activation function
  - $\frac{\partial z}{\partial wi} = \frac{\partial \sum_{i=0}^{n} x_i w_i}{\partial wi} = \frac{\sum_{i=0}^{n} x_i w_i}{\partial wi} = \frac{x_i w_i}{\partial wi} = x_i$
    - $\frac{\partial z}{\partial wi} = x_i$
- Replacing each term by its value will give
  - $\frac{\partial J}{\partial wi} = 2(\hat{y}-y) * g'(z) * x_i = \delta * xi$ - δ have the same value for all $\frac{\partial J}{\partial wi}$
  - $\nabla J(X,\ W) = \delta * X$
  - $W_{t+1} = Wt - \alpha . \nabla J(X,\ Wt) = W_t - \alpha . \delta_t * X$

Gradient descent for deep learning Case of one neuron

- Algorithm
  - Choose initial weights,
  - Choose stopping condition $(\varepsilon > 0)$
  - Choose learning rate α

  - Compute $\hat{y}$ -- propagation step
  - Compute $\delta_t = J'(y, \hat{y}) * g'(\sum w_i x_i)$
  - Stop if $\|\delta_t * X\| < \varepsilon$
  - Update weight -- back propagation step
    - $W_{t+1} = Wt - \alpha . \delta_t * X$

Gradient descent for deep learning

- For a chain of one neuron
  - It's exactly the same approach with a longer derivative chain.
- For many hidden layers withs many neurons
  - It is necessary to adapt the "chain rule" to take into account the contribution of each neuron of the previous level.
- If you are interested, read:
  - https://github.com/maxim5/cs224n-winter-2017/blob/master/lecture_notes/cs224n-2017-gradient-notes.pdf

Gradient descent for deep learning
Back propagation algorithm

- Input: set the input in layer #1
- Forward: for each layer $l = 2, 3, \dots, L$
  - Compute
    - $z^l = w^l a^{l-1}$ (remember $a_0^{l-1} = 1 = bias$ and w^l is a matrix)
    - $a^l = g^l(z^l)$
- Compute output error for final layer and the gradient
  - $\delta^L = \nabla aJ \odot g^{L\prime}(z^L)$
- Stop if $\|\delta^L * X\| < \varepsilon$
- Back propagation: for each layer $l = L-1, L-2, \dots, 2$
  - Compute
    - $\delta^l = \left(\left(w^{l+1}\right)^T * \delta^{l+1}\right) \odot g^{l\prime}(z^l)$
  - Update gradient
    - $w^l_{ir,t+1} = w^l_{ir,t} - \alpha * \delta^l_i * a^{l-1}_r$ where $a^l_0 = 1$ (biais)

Deep learning in Python

Which library ?

- Some deep learning libraries under Python
  - **Theano**: provides an API for writing digital algorithms using the GPU
    - Not specialized in deep learning even if it offers many features for this purpose
    - kind of numpy GPU
  - **Tensorflow**: Originally developed by researchers and engineers from the Google
    - Strong support for deep learning
  - **Pytorch**
  - **Keras**: an over-layer to different at deep learning library
    - I've choose Keras+Tensoflow as backend

- Comparatifs: github.com/zer0n/deepframeworks

Keras properties

- Keras is an API designed for human beings
  - Minimizes the number of user actions required for common use cases
  - Offers consistent & simple APIs
  - Keras is the official high-level API of TensorFlow
- Keras is multi-platform, multi-backend,
  - Develop in Python, R
  - On Unix, Windows, OSX
  - Run the same code with... – TensorFlow, CNTK, Theano, …
- Allow access to low-level API
  - Full access to TensorFlow API
- Keras is easy to learn and easy to use
  - Easy to use models in production
- Three API
  - Sequential
  - Functional ← our favorite one
  - Subclassing

Sequential API
(only for sequential model)

```
1. import keras
2. from keras.models import Sequential
3. from keras.layers import Dense

4. model = keras.Sequential()
5. model.add(layers.Dense(16, input_dim=N_features, activation='relu'))
6. model.add(layers.Dense(8, activation='relu'))
7. model.add(layers.Dense(1))

8. model.compile(optimizer='sgd', loss='mean_squared_error’)

9. model.fit(X_train, y_train, epochs=100, validation_split=0.33)

10. y_pred = model.predict(X_test)
```

Functional API (our favorite API)

```
1. import keras
2. from layers import Input, Dense

3. inputs = keras.Input(shape=(N_features, ))
4. x = Dense(16, activation='relu')(inputs)
5. x = Dense(8, activation='relu')(x)
6. outputs = Dense(1)(x)

7. model = keras.Model(inputs, outputs)
8. model.compile(optimizer='sgd', loss='mean_squared_error')

9. model.fit(X_train, y_train, epochs=100, validation_split=0.33)

10. y_pred = model.predict(X_test)
```

Model subclassing (You can work without)

```
import keras
from keras.layers import Dense

class MyModel(keras.Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = Dense(16, activation='relu')
        self.dense2 = Dense(8, activation='relu')
        self.dense3 = Dense(1)
    def call(self, inputs):
        x = self.dense1(inputs)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.compile(optimizer='sgd', loss='mean_squared_error')

model.fit(X_train, y_train, epochs=100, validation_split=0.33)

y_pred = model.predict(X_test)
```

Practical issues

Choose the network architecture

- **Input layer → number of feature**
  - inputs = keras.Input(**shape=(N_features,)**)

- Hidden layer → number of layers / number of neurons by layer
  → activation funtion: relu / sigmoid / tanh
  - x = Dense(**16**, activation='**relu**')(inputs)
  - x = Dense(**8**, activation='**relu**')(x)

Choose the network architecture

- **Output layer**

→ **Regression**

→ One neuron by function to fit

→ activation funtion: Linear

- outputs = Dense(**1**, activation='**linear**')(x)

→ **Classification**

→ 2 classes = 1 neuron

→ → activation funtion: sigmoid

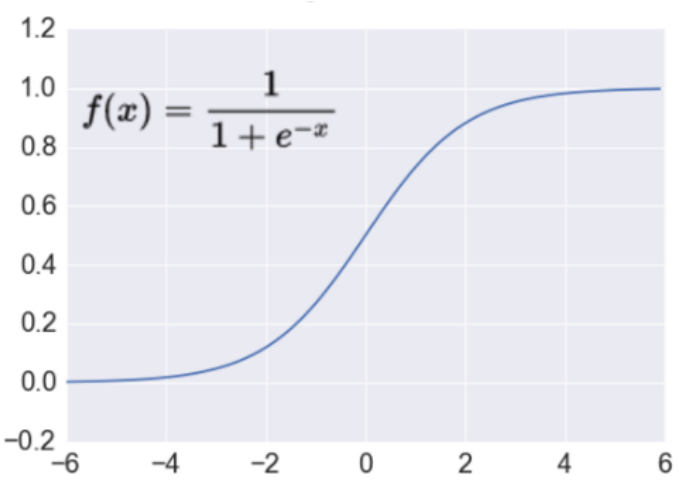
- outputs = Dense(**1**, activation='**sigmoid**')(x)

→ N classes = N neurons

→ activation funtion: sofmax

- outputs = Dense(**n_classes**, activation='**softmax**')(x)

Activation: Sigmoid



http://adilmoujahid.com/images/activation.png

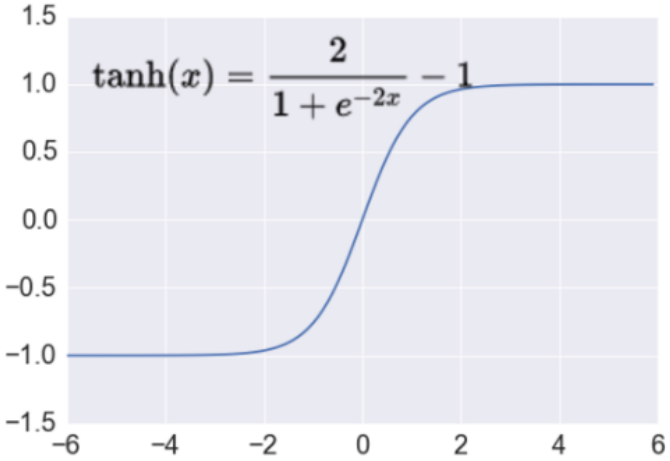
Takes a real-valued number and
"squashes" it into range between 0 and 1.

$$R^n \rightarrow [0,1]$$

\+ Nice interpretation as the **firing rate** of a neuron
- 0 = not firing at all
- 1 = fully firing

\- Sigmoid neurons **saturate** and **kill gradients**, thus NN will barely learn
- when the neuron's activation are 0 or 1 (saturate)
  gradient at these regions almost zero
  almost no signal will flow to its weights
  if initial weights are too large then most neurons would saturate

Activation: Tanh



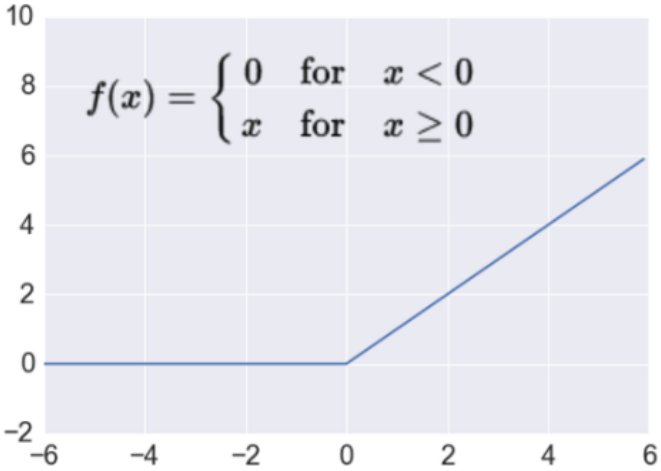
http://adilmoujahid.com/images/activation.png

Takes a real-valued number and
"squashes" it into range between -1 and 1.

$$R^n \rightarrow [-1,1]$$

- 🙁Like sigmoid, tanh neurons **saturate**
- 🙂Unlike sigmoid, output is **zero-centered**
- Tanh is a **scaled sigmoid**: $\tanh(x) = 2sigm(2x) - 1$

Activation: ReLU



http://adilmoujahid.com/images/activation.png

Takes a real-valued number and thresholds it at zero $\mathrm{f}(x) = \max(0, x)$

Most Deep Networks use ReLU nowadays

Trains much **faster**

- accelerates the convergence of SGD
- due to linear, non-saturating form

Less expensive operations

- compared to sigmoid/tanh (exponentials etc.)
- implemented by simply thresholding a matrix at zero

More **expressive**

Prevents the **gradient vanishing problem**

Hyper-parameters fitting

- There are many other parameters to set in order to efficiently converge the network to the slightest error while maintaining a strong generalization capacity.

- Batch size
- Optimizer
- Number of epochs
- Loss / Metrics
- Deal with underfitting and overfitting
- Vanishing gradient
- Etc.

Next week lecture

- We'll discuss these items next week

Conclusion

- Neural Network: a flexible model but also a black box model
  - Difficult to interpret
  - Regression
    - $output \in]-\infty, +\infty[$, an activation is not necessary
    - $output \in]-1, +1[$, you can use also tanh function
  - Classification
    - Output function = sofmax
    - 2 classes (sofmax = sigmoid)
      - Interpret $\hat{y}$ as probability of belonging to the class: P(C=c|X)
    - More than 2 classes, do the same i.e.
      - Sofmax: normalize the output probabilities: $\sum P(C = ci|X) = 1$
- Many libraries in many programming languages
- Tuning is crucial and not obvious