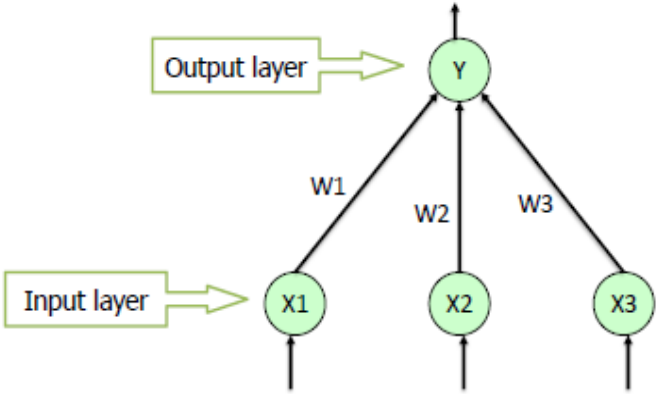
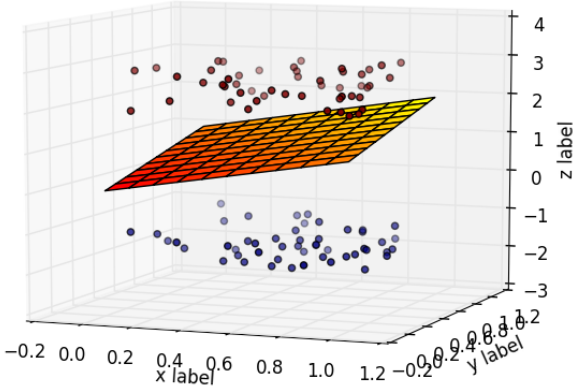
Introduction to Machine Learning

Neural Networks

Linearly separable

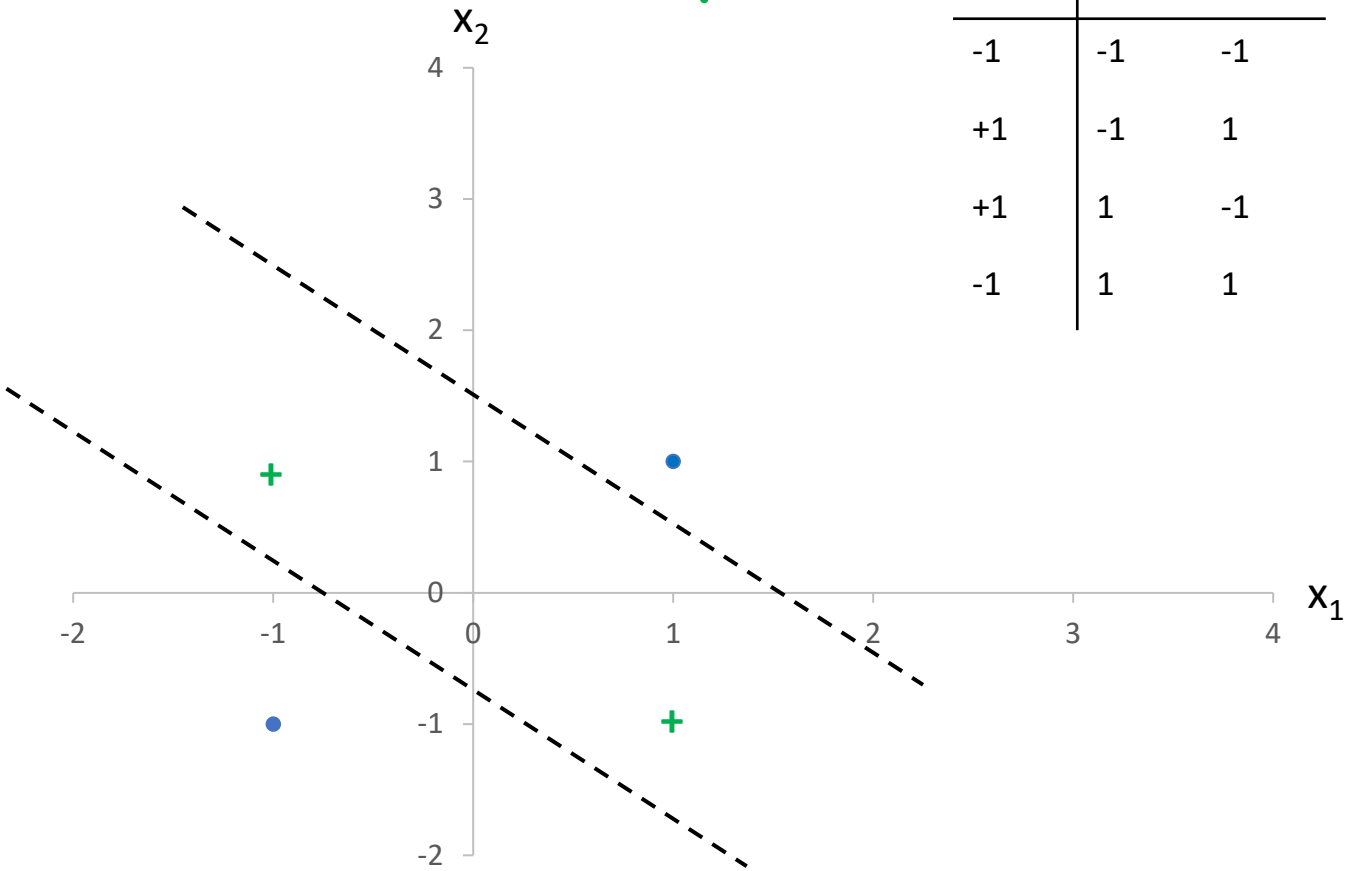
- If the classes can be separated by a hyperplane, then they are linearly separable



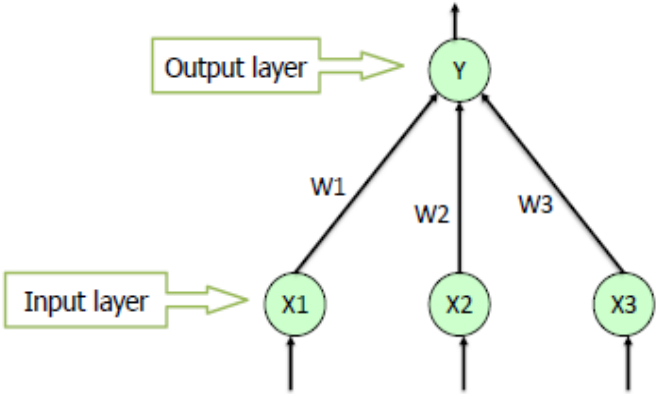


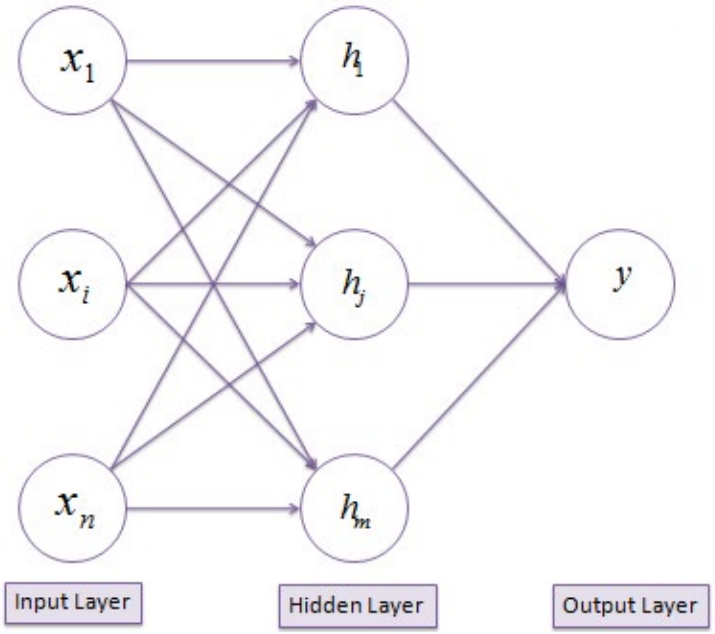
XOR decision boundary

y	x_1	X_2
-1	-1	-1
+1	-1	1
+1	1	-1
-1	1	1



Adding layers



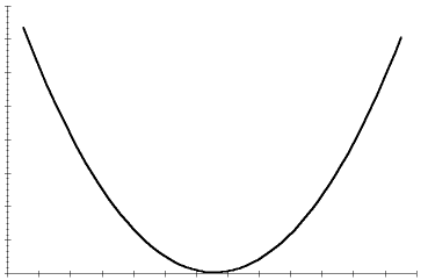


Activation function and gradient descent

$$h_i = f\left(\boldsymbol{w}_i \cdot \boldsymbol{x}\right)$$

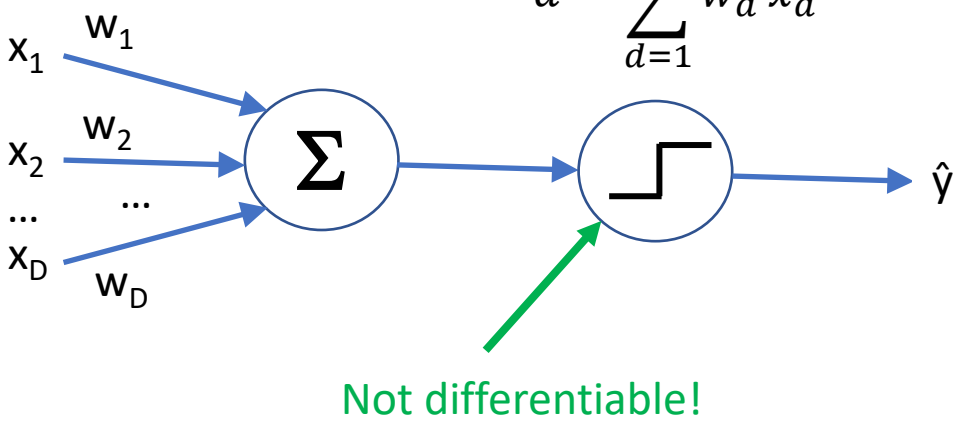
Algorithm 21 GradientDescent($\mathcal{F}$, K, η_1, ...)

```
z^(0) ← ⟨0, 0, ..., 0⟩                    // initialize variable we are optimizing
for k = 1 ... K do
    g^(k) ← ∇_z F|_{z^(k-1)}              // compute gradient at current location
    z^(k) ← z^(k-1) − η^(k) g^(k)         // take a step down the gradient
end for
return z^(K)
```



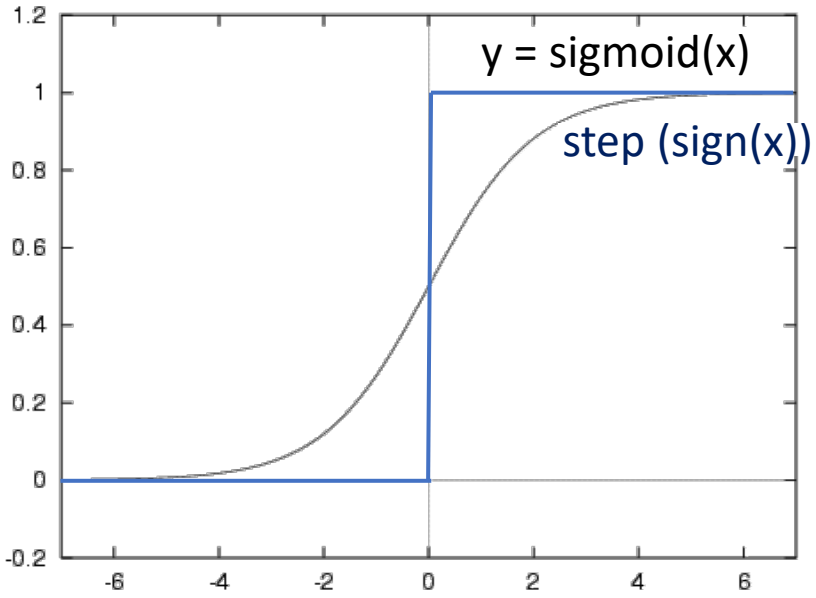
Perceptron neuron

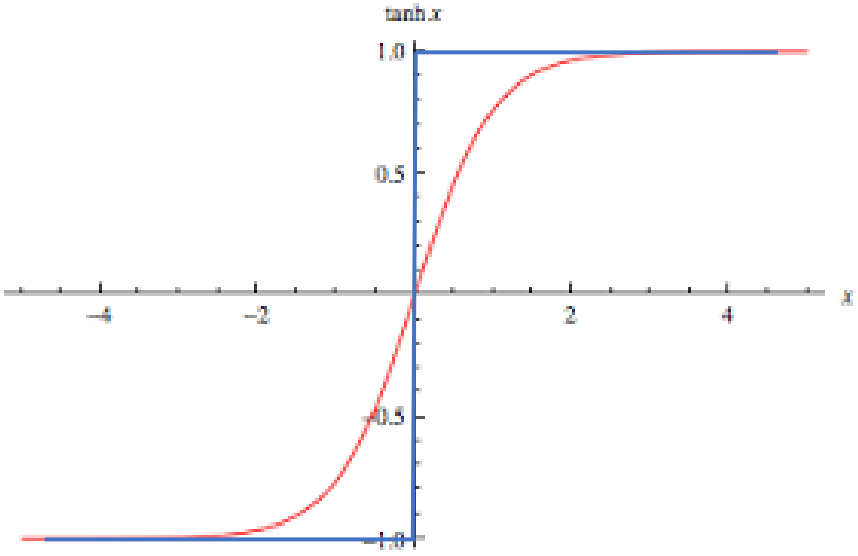
$$a = \sum_{d=1}^{D} w_d \, x_d$$



derivative = infinity here

Hyperbolic tangent





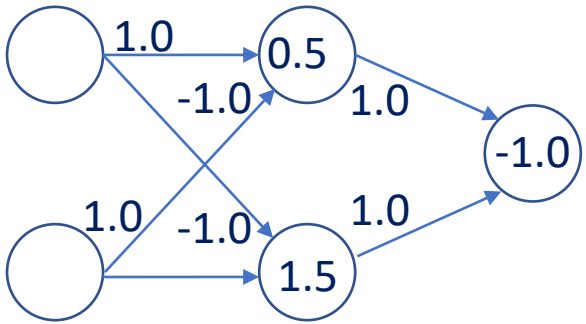
Algorithm 25 TwoLayerNetworkPredict($\mathbf{W}$, v, $\hat{x}$)

```
for i = 1 to number of hidden units do
    h_i ← tanh(w_i · x̂)                    // compute activation of hidden unit i
end for
return v · h                                // compute output unit
```

Inductive bias

XOR (with step functions)

y	x1	x2
-1	-1	-1
+1	-1	1
+1	1	-1
-1	1	1



$$a(hidden) = \sum w \cdot x + b$$

h = tanh(a)

$$y = \sum v \cdot h$$

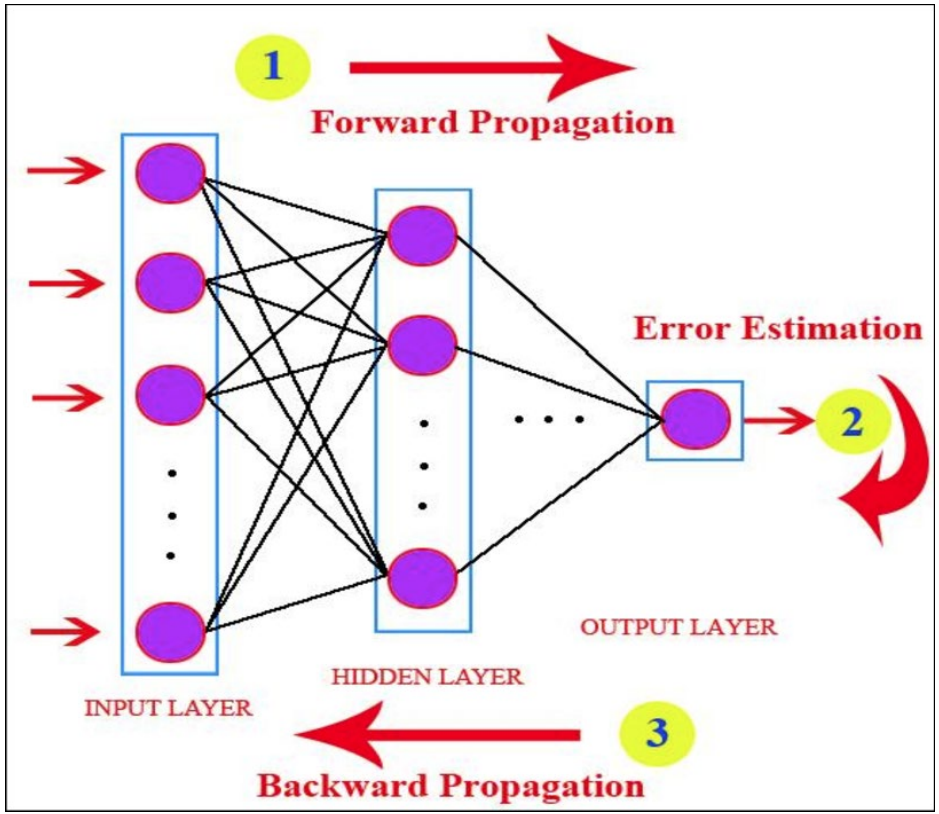
How big of a network do I need?

Let F be a continuous function on a bounded subset of D-dimensional space. Then there exists a two-layer neural network $\hat{F}$ with a finite number of hidden units that approximate F arbitrarily well. Namely, for all **x** in the domain of F, $|F(x) - \hat{F}(x)| < \varepsilon$.

Backpropagation

- We know how to compute output
- How do we learn weights?

backpropagation = gradient descent + chain rule


1
Forward Propagation
Error Estimation
2
OUTPUT LAYER
HIDDEN LAYER
INPUT LAYER
3
Backward Propagation

Optimization function

- Based on loss function: Squared error

$$\min_{\mathbf{W},v} \quad \sum_n \frac{1}{2}\left(y_n - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x}_n)\right)^2$$

Adjusting weights to output node

- Differentiate objective with respect to v

$$\min_{\mathbf{W},v} \quad \sum_n \frac{1}{2}\left(y_n - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x}_n)\right)^2$$

$$\nabla_v = -\sum_n e_n \boldsymbol{h}_n$$

Adjusting weights to hidden node

- Only compensate for *portion* of error for that hidden node
- Not trying to produce specific output value
- Just trying to produce activations that get fed to output layer
- Change depends on how output layer uses the weight

Adjusting weights to hidden node

$$\mathcal{L}(\mathbf{W}) = \frac{1}{2}\left(y - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x})\right)^2$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_i} = \frac{\partial \mathcal{L}}{\partial f_i}\frac{\partial f_i}{\partial \boldsymbol{w}_i}$$

$$\frac{\partial \mathcal{L}}{\partial f_i} = -\left(y - \sum_i v_i f(\boldsymbol{w}_i \cdot \boldsymbol{x})\right) v_i = -e v_i$$

$$\frac{\partial f_i}{\partial \boldsymbol{w}_i} = f'(\boldsymbol{w}_i \cdot \boldsymbol{x})\boldsymbol{x}$$

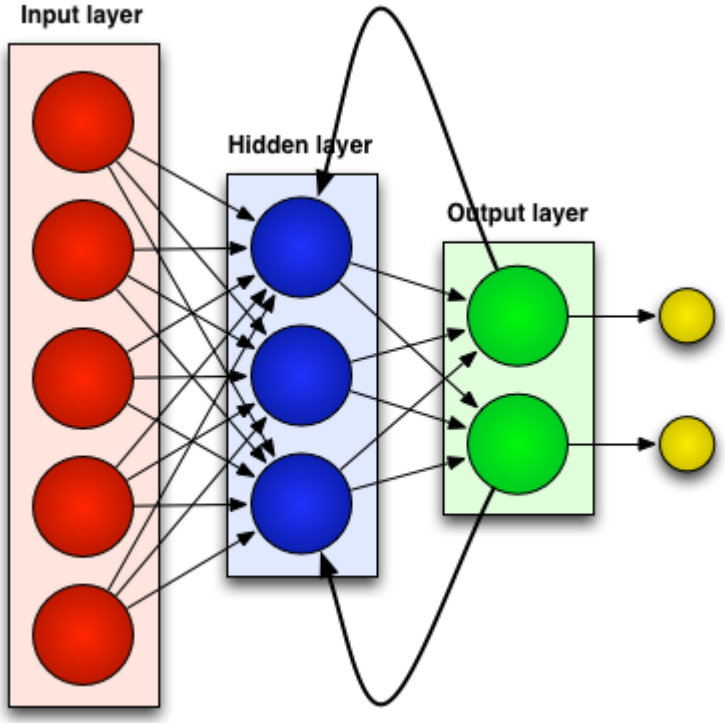
Algorithm 26 TwoLayerNetworkTrain($\mathbf{D}$, η, K, $MaxIter$)

```
W ← D×K matrix of small random values          // initialize input layer weights
v ← K-vector of small random values            // initialize output layer weights
for iter = 1 ... MaxIter do
    G ← D×K matrix of zeros                    // initialize input layer gradient
    g ← K-vector of zeros                      // initialize output layer gradient
    for all (x,y) ∈ D do
        for i = 1 to K do
            a_i ← w_i · x̂
            h_i ← tanh(a_i)                    // compute activation of hidden unit i
        end for
        ŷ ← v · h                              // compute output unit
        e ← y − ŷ                              // compute error
        g ← g − eh                             // update gradient for output layer
        for i = 1 to K do
            G_i ← G_i − e v_i (1 − tanh^2(a_i)) x   // update gradient for input layer
        end for
    end for
    W ← W − ηG                                 // update input layer weights
    v ← v − ηg                                 // update output layer weights
end for
return W, v
```

Hyperparameters

Let's try this out

Recurrent neural networks



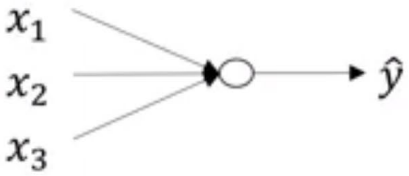
Recurrent neural networks

- Sequence models

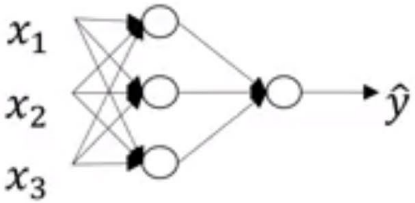
Speech recognition		→	"The quick brown fox jumped over the lazy dog."
Music generation	∅	→	
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger.

[Ng 2018]

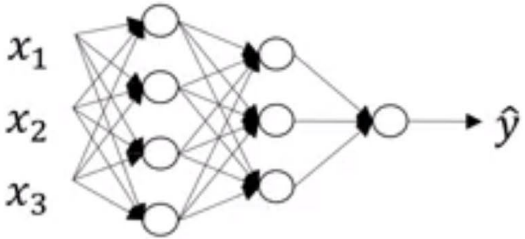
Deep neural networks



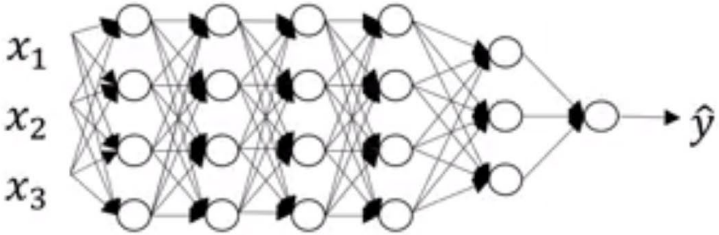
logistic regression



1 hidden layer



2 hidden layers



5 hidden layers

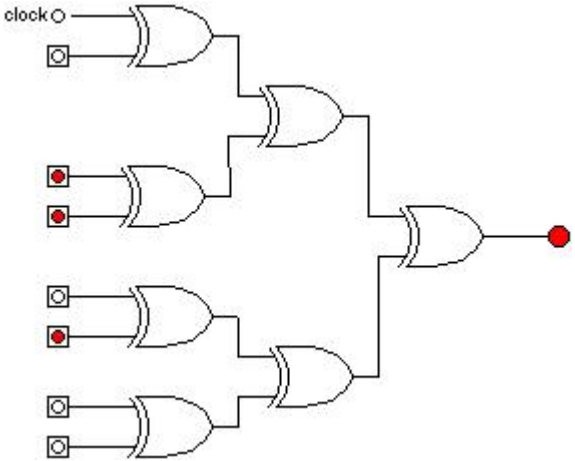
Why deep?

- Some functions are more easily learned by deep (and narrow) versus broad (and shallow)

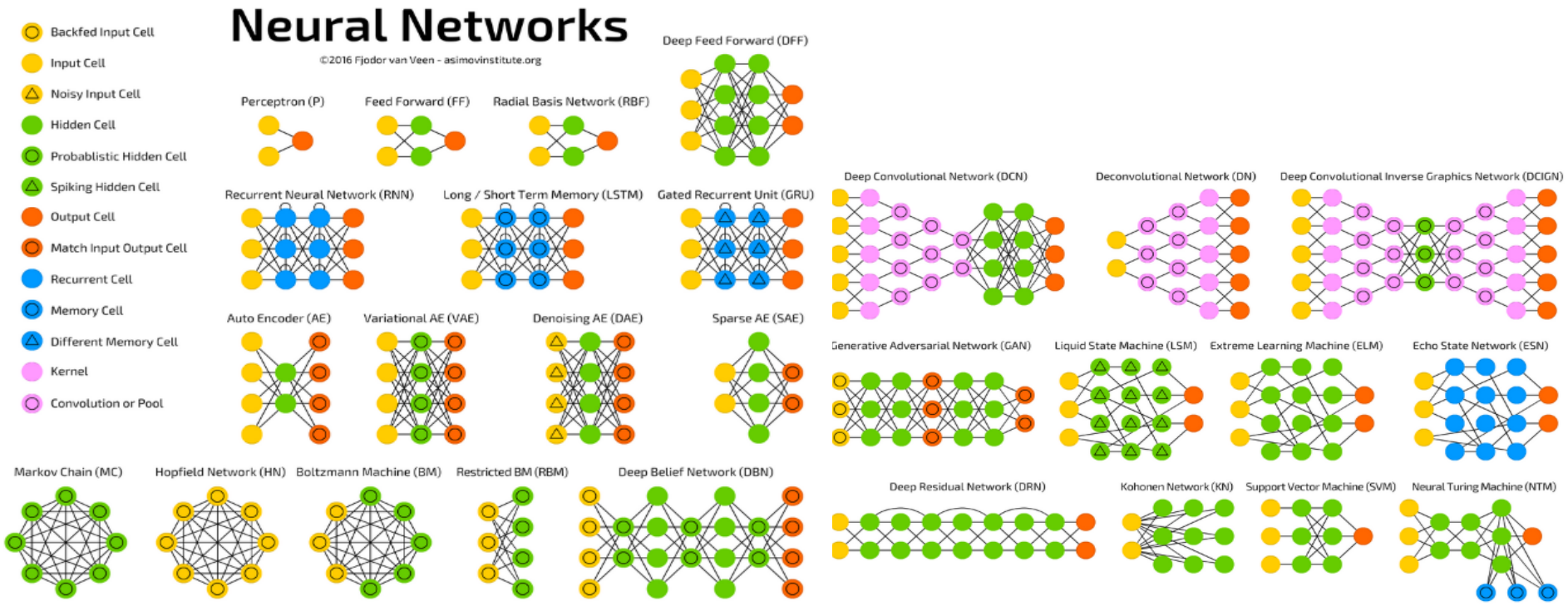
- Parity

Parity function complexity

- Define circuit with XOR gates

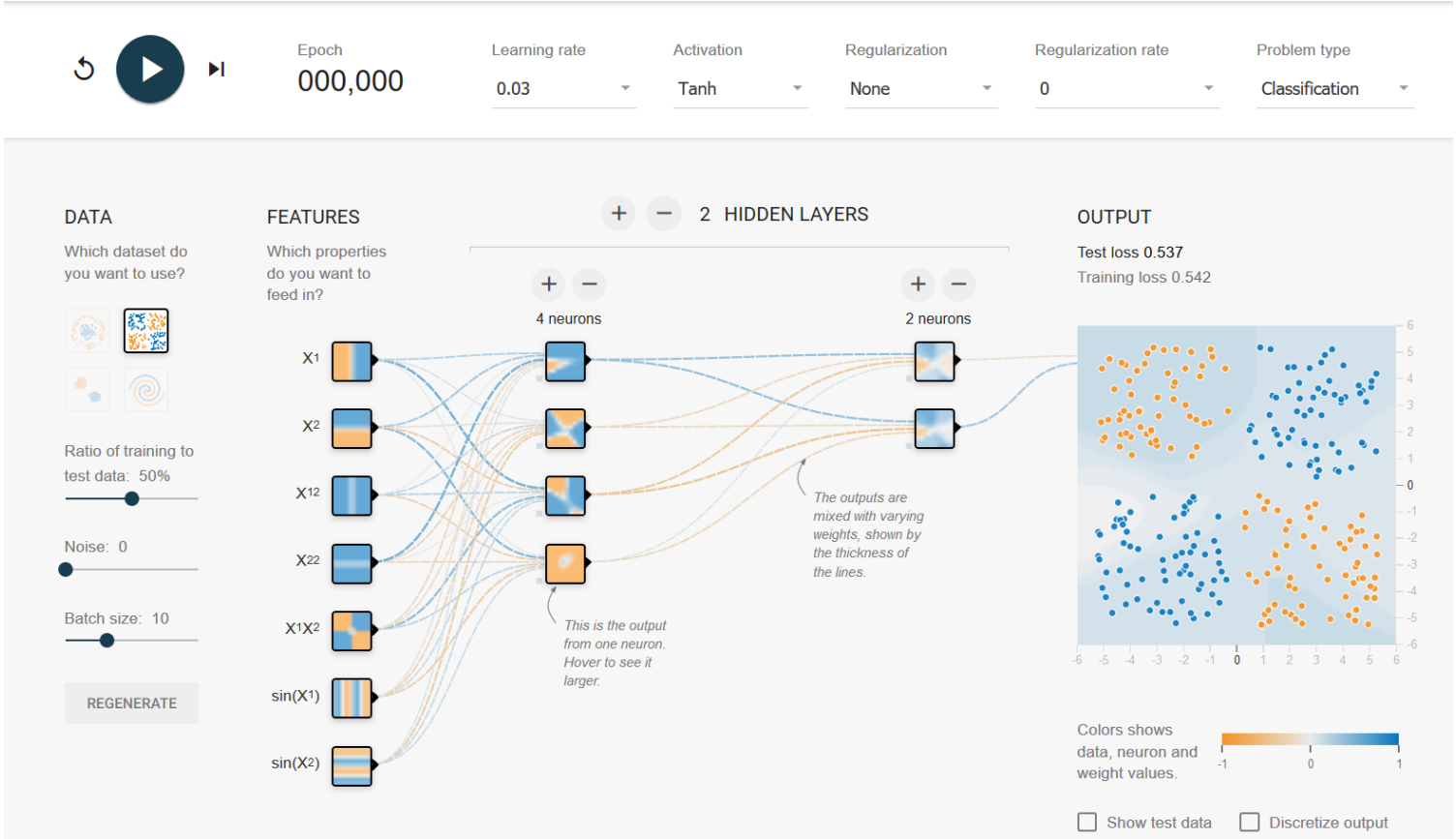


Zoo of neural network types

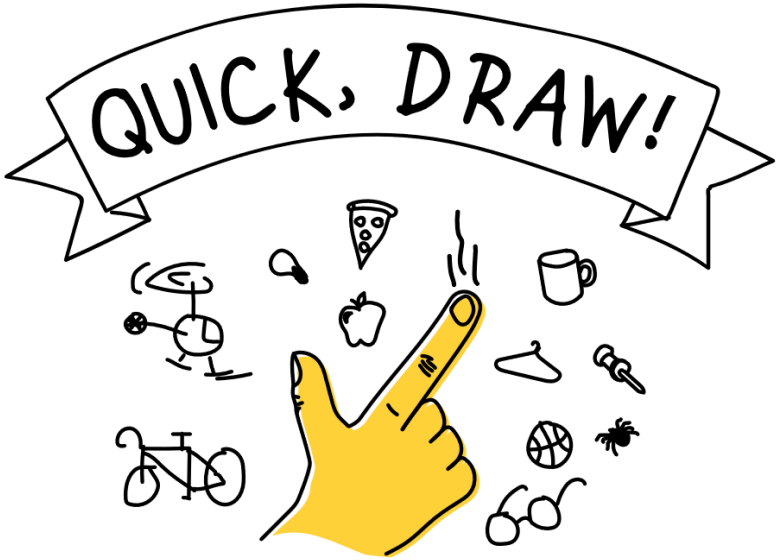


https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

Let's play with this



Deep learning applications



Can a neural network learn to recognize doodling?

Help teach it by adding your drawings to the world's largest doodling data set, shared publicly to help with machine learning research.