APPLYING THE MAGIC OF NEURAL NETWORKS

MACHINE LEARNING WITH NODEJS

>WHOAMI

LIAN LI

- Dev@Jimdo
- @chimney42
- github.com/chimney42
- Google+ Lian Li

>RANDOM_STRING_OUTPUT

I NEVER PREDICT ANYTHING,
AND I NEVER WILL.

Paul Gascoigne

UNDERSTANDING

NEURAL NETWORKS

>MAN NEURAL_NETWORK

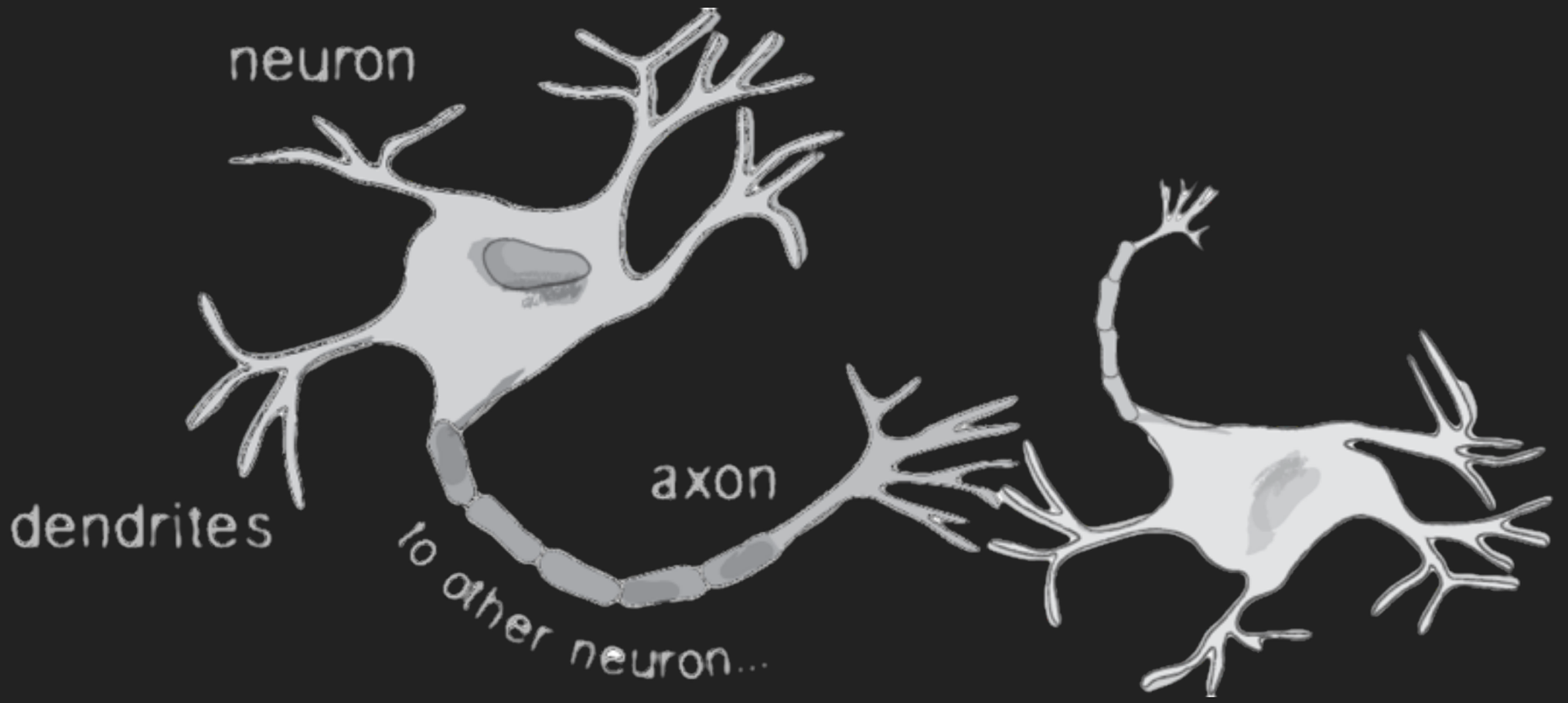
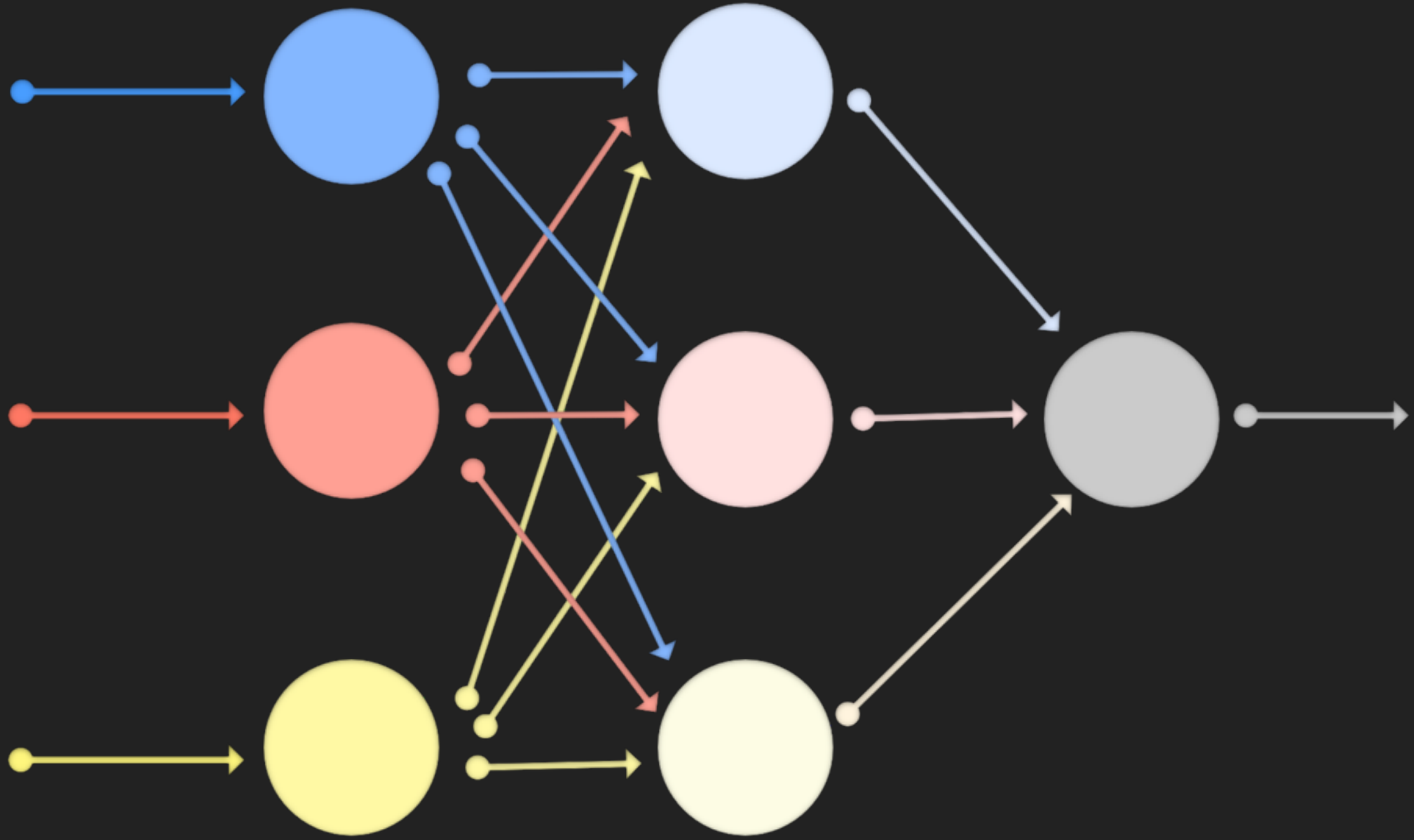


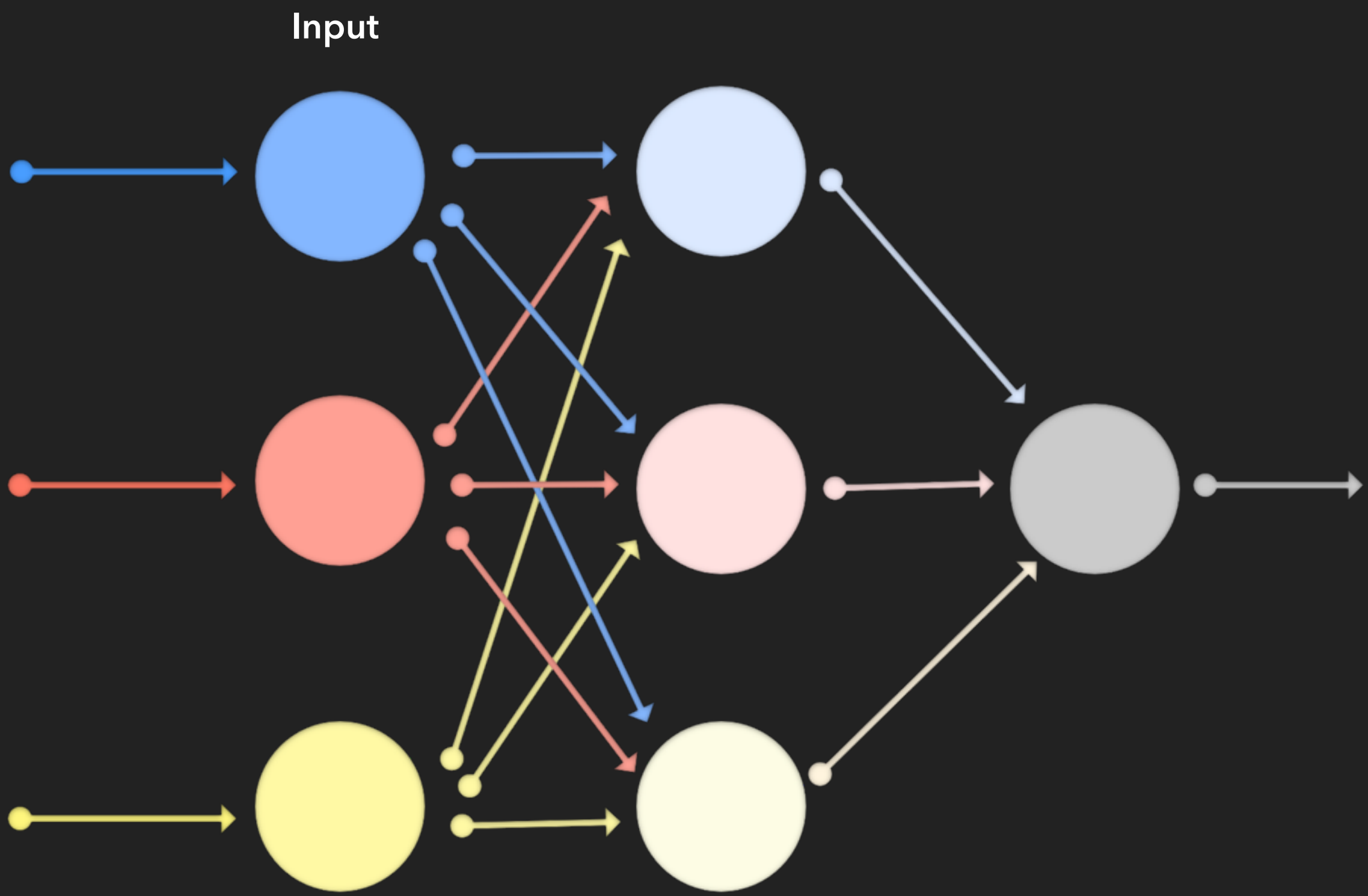
Fig. 10.1
http://natureofcode.com/book/chapter-10-neural-networks/

>MAN NEURAL_NETWORK

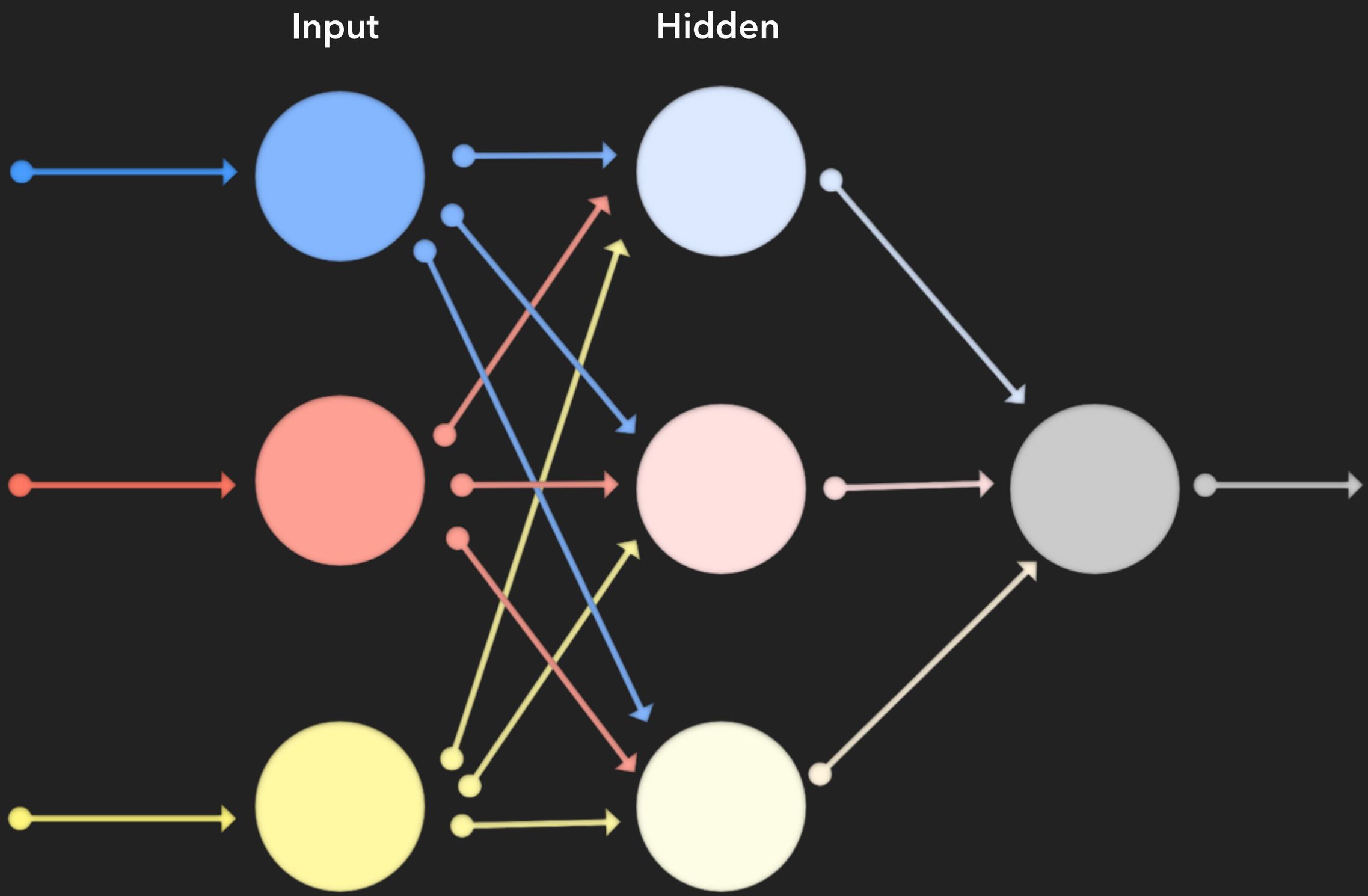
>MAN NEURAL_NETWORK



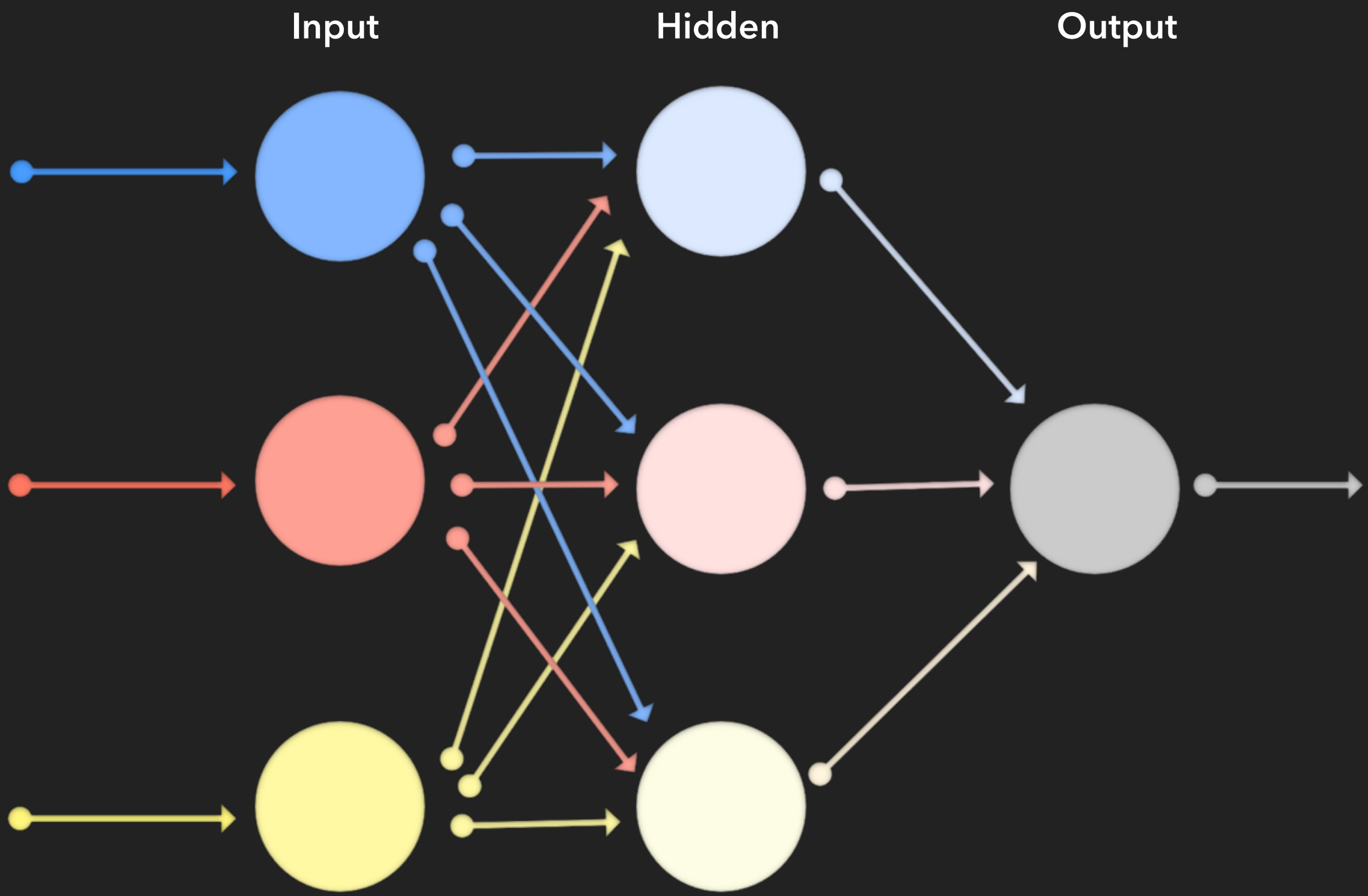
>MAN NEURAL_NETWORK



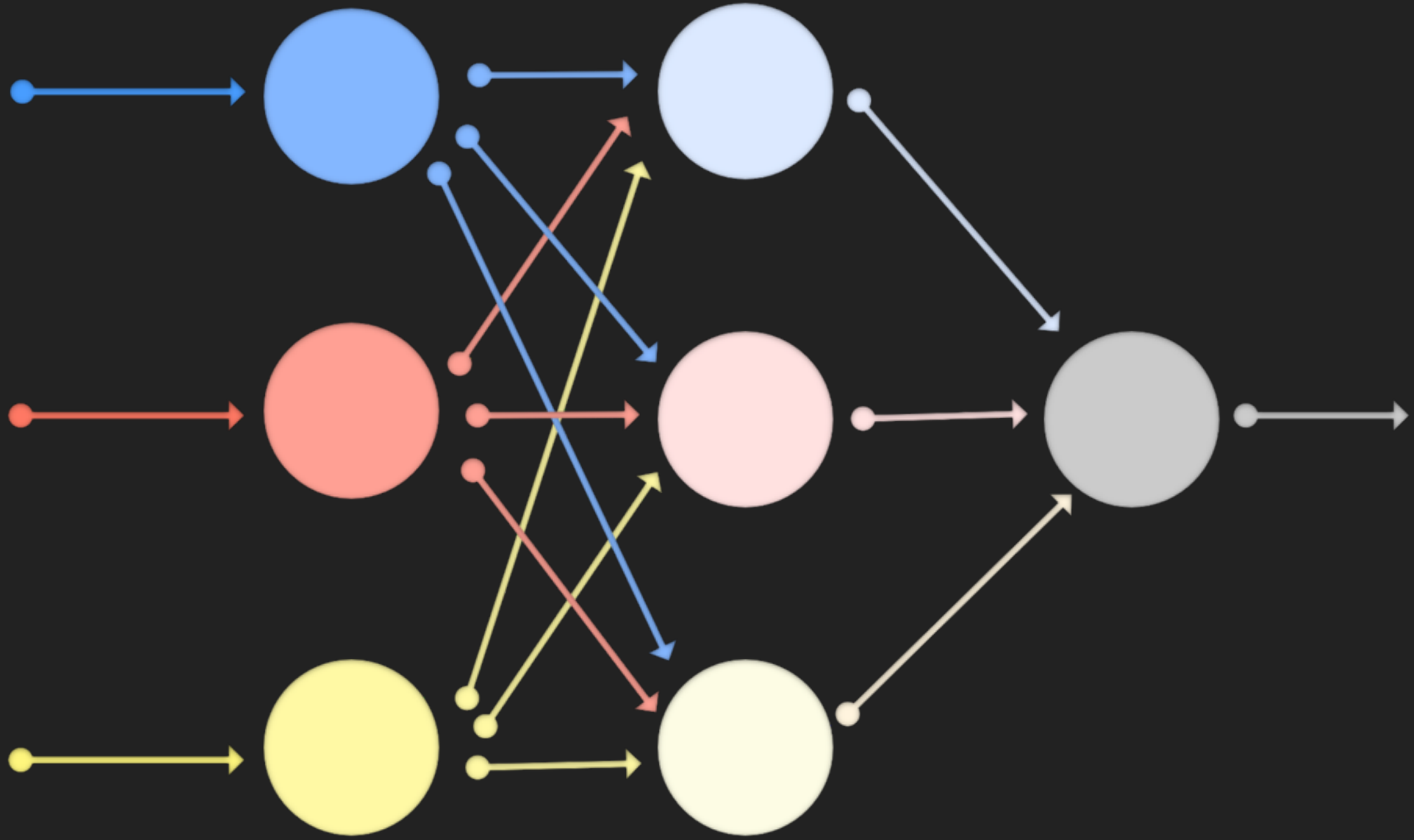
>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK

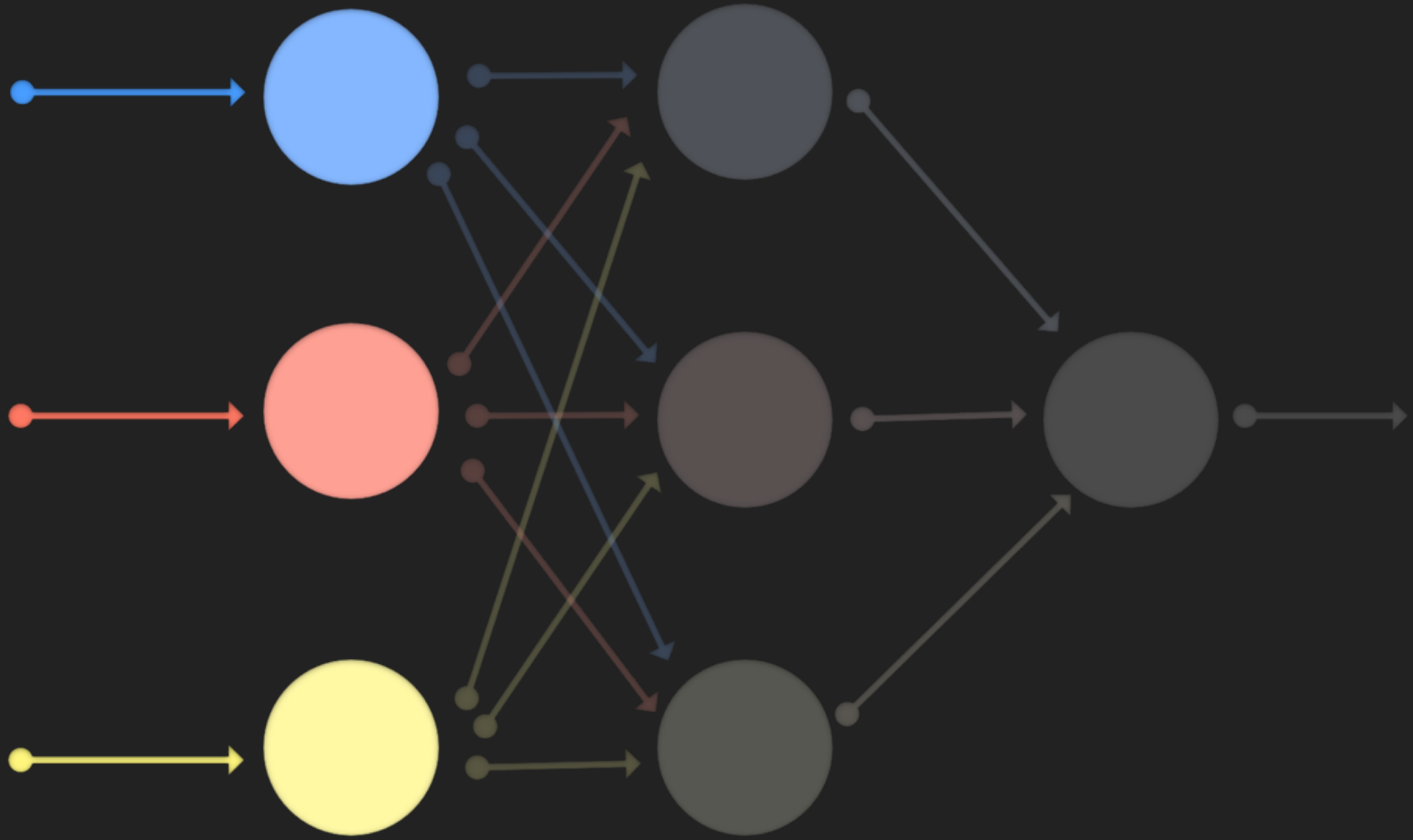


>MAN NEURAL_NETWORK

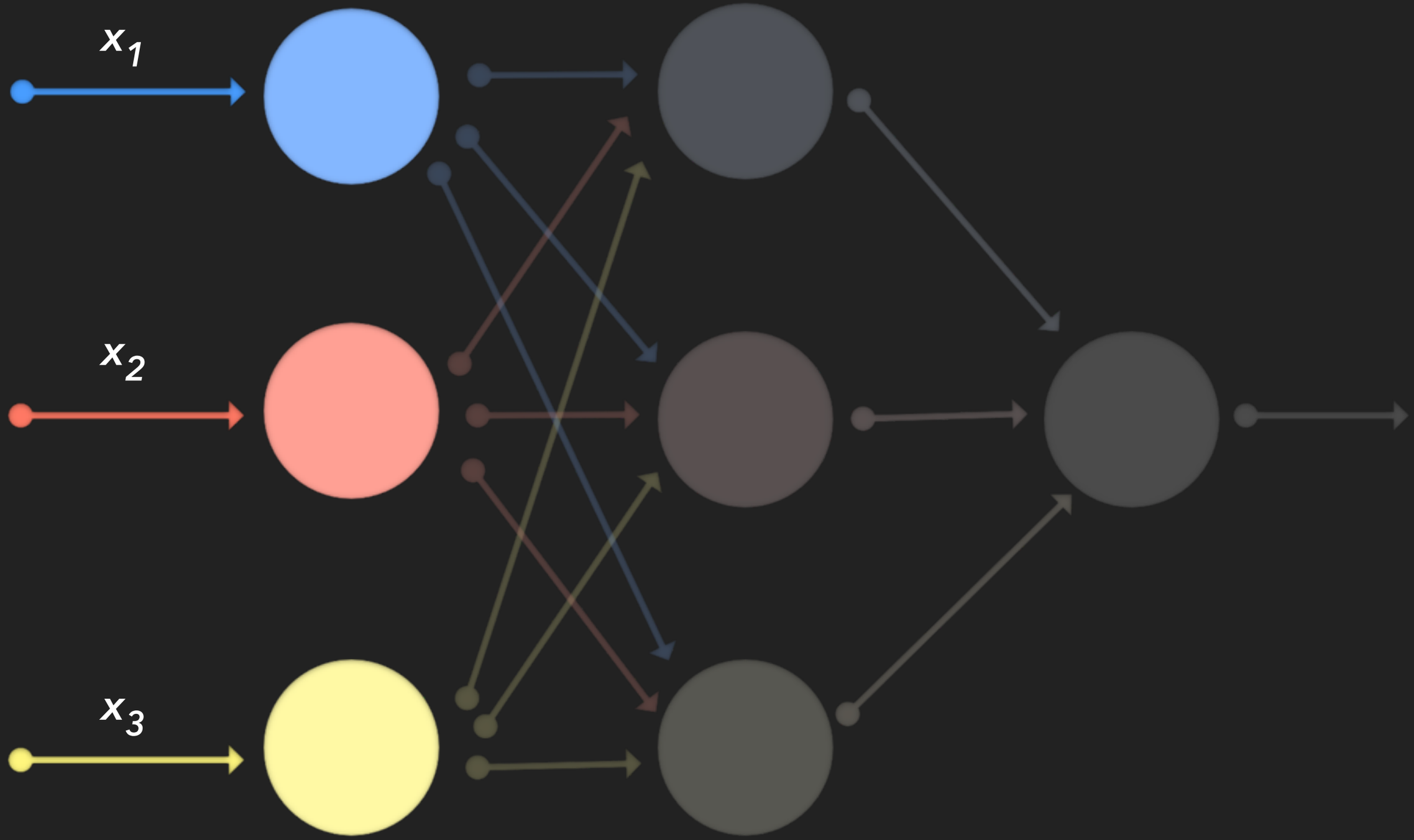


>MAN NEURAL_NETWORK

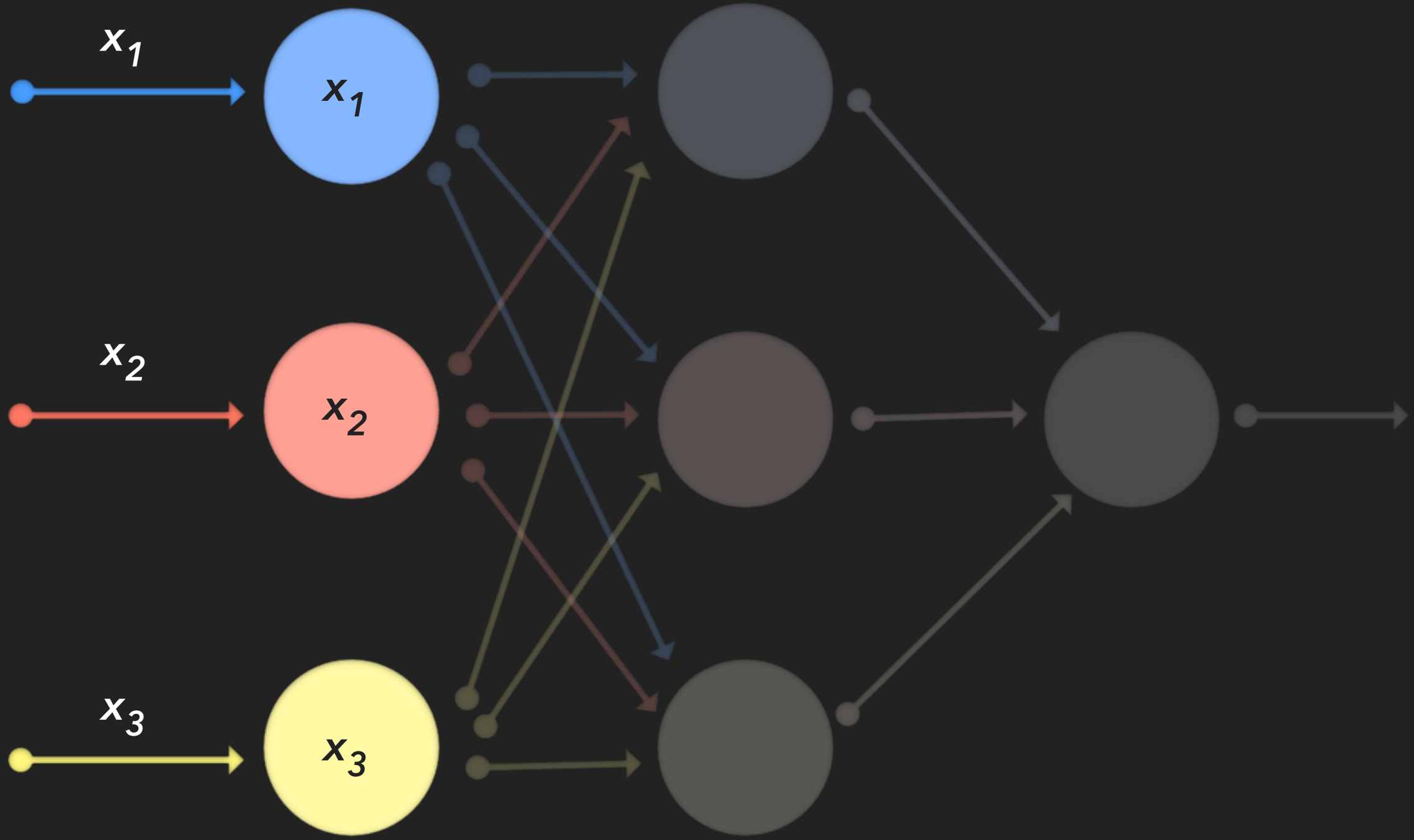
>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK

---

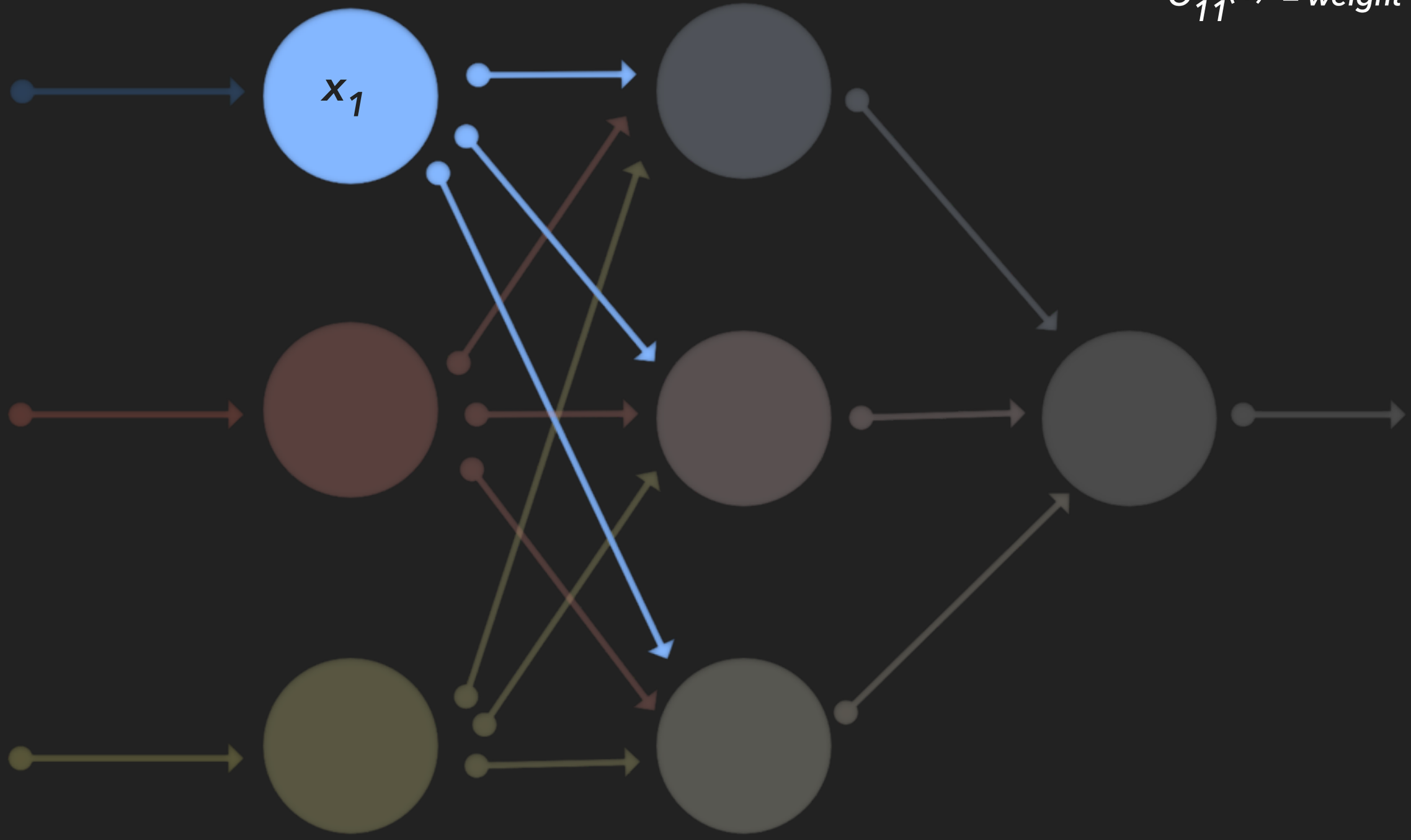
x_1 = *input*

$\Theta_{11}^{(1)}$ = *weight*

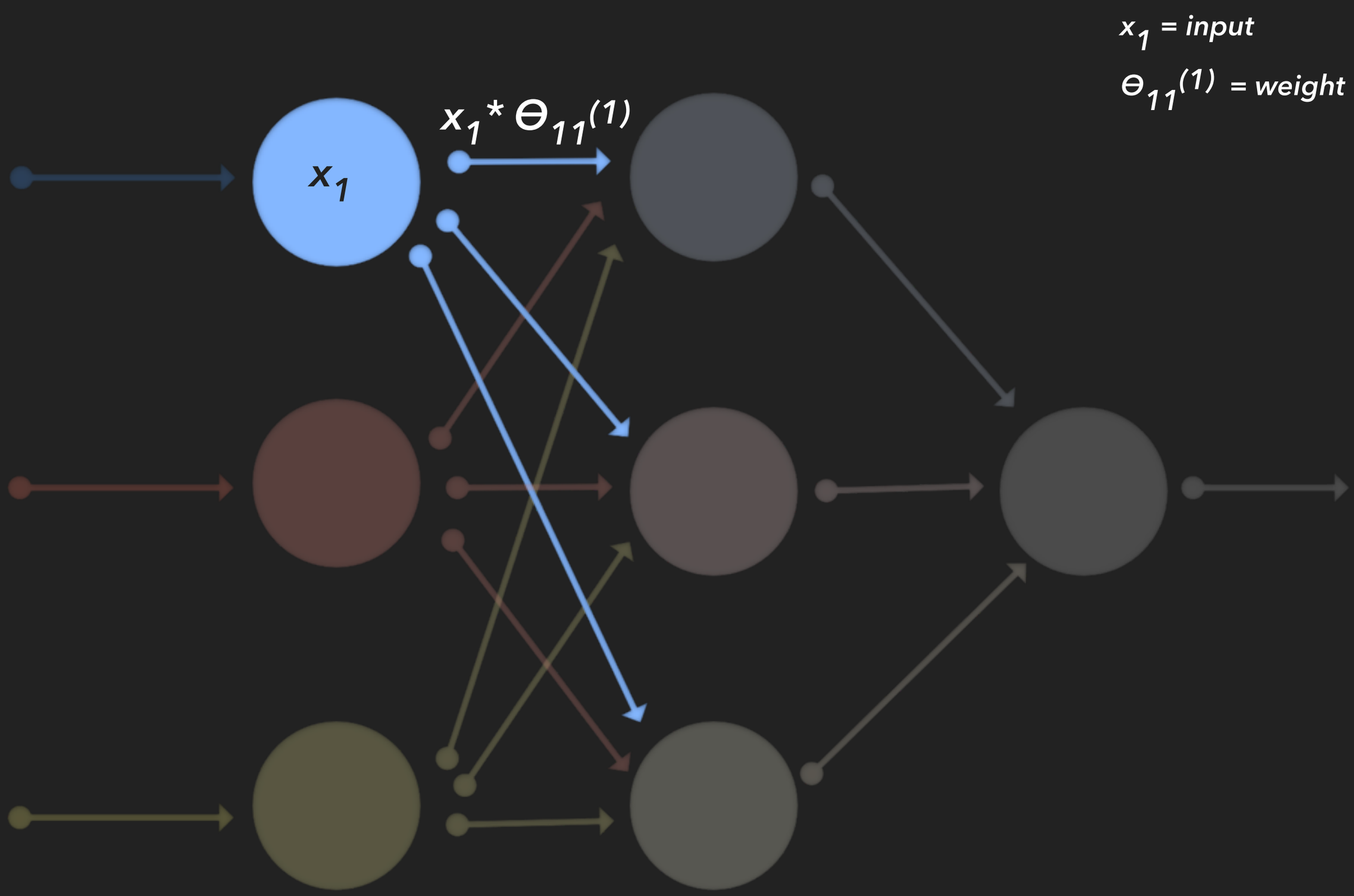
>MAN NEURAL_NETWORK

x_1 = input

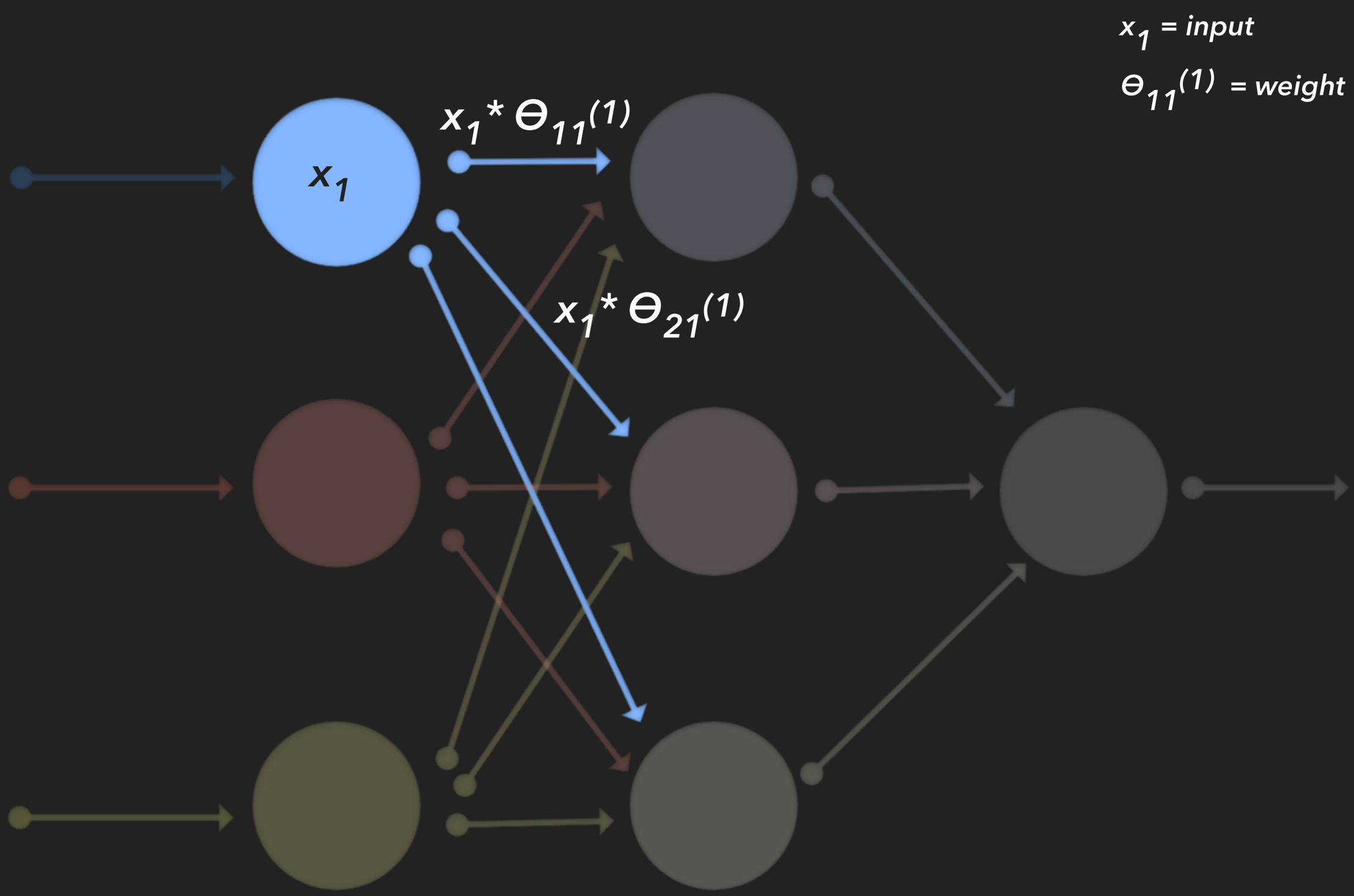
$\Theta_{11}^{(1)}$ = weight



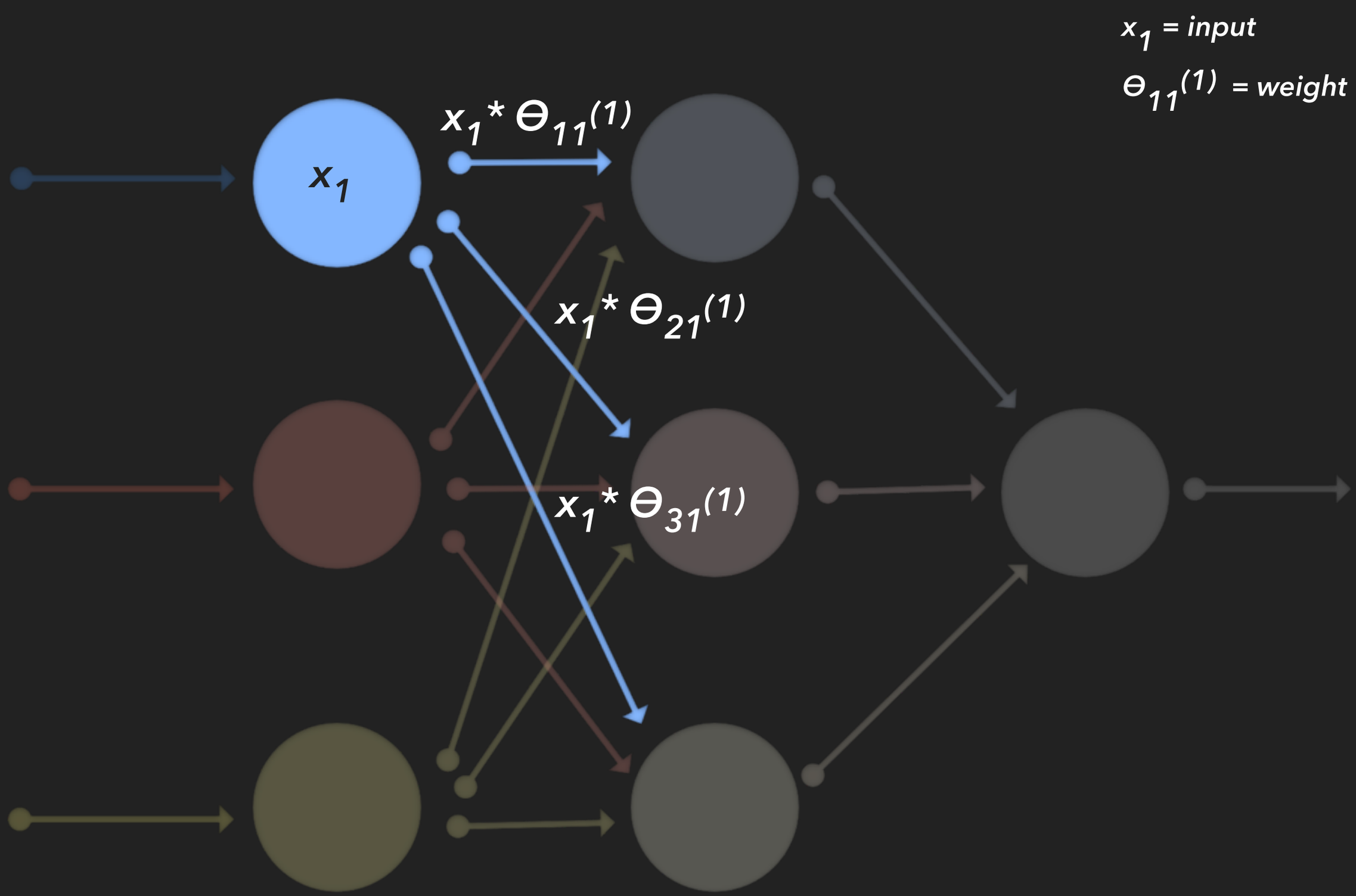
>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK

---

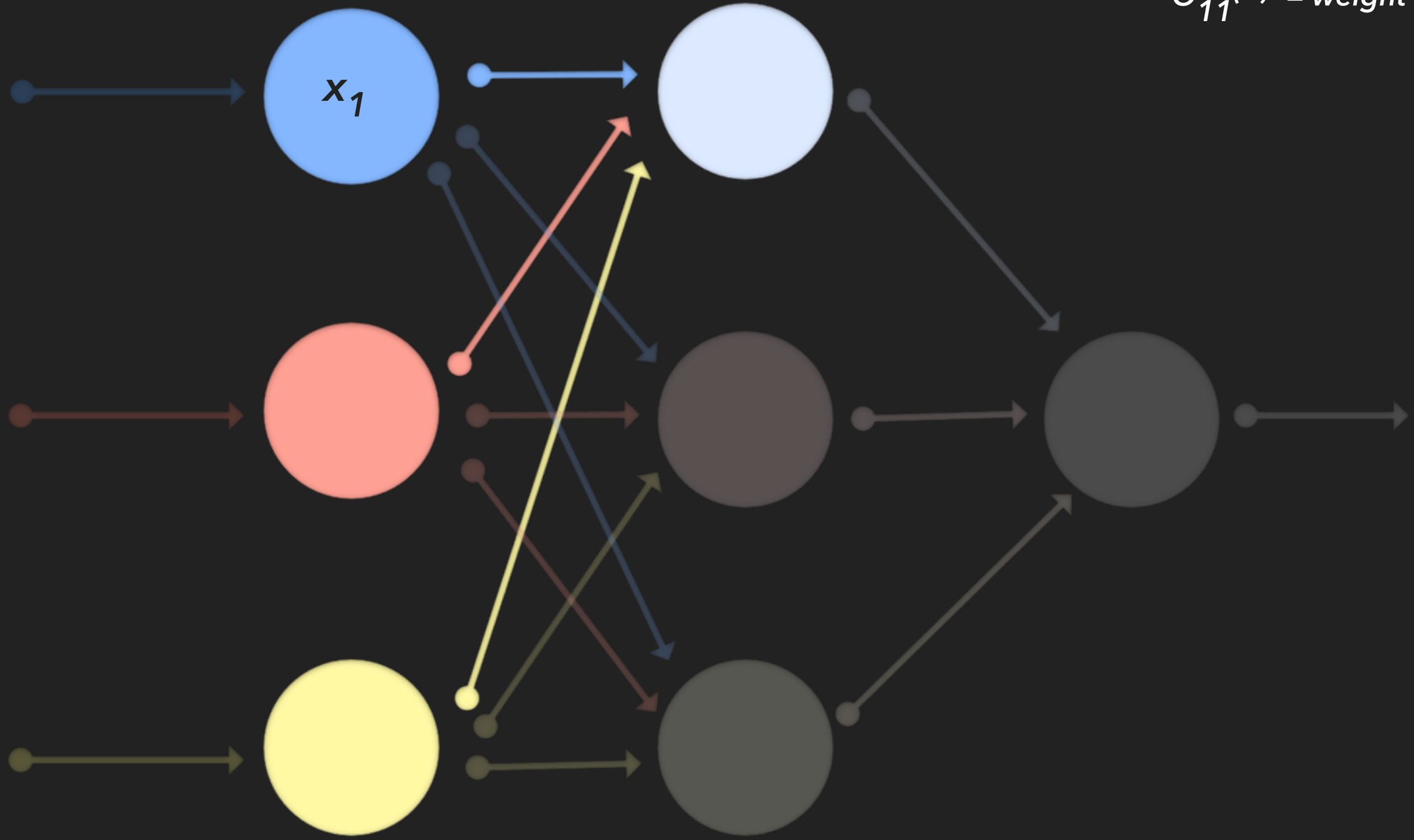
x_1 = *input*

$\Theta_{11}^{(1)}$ = *weight*

>MAN NEURAL_NETWORK

x_1 = input

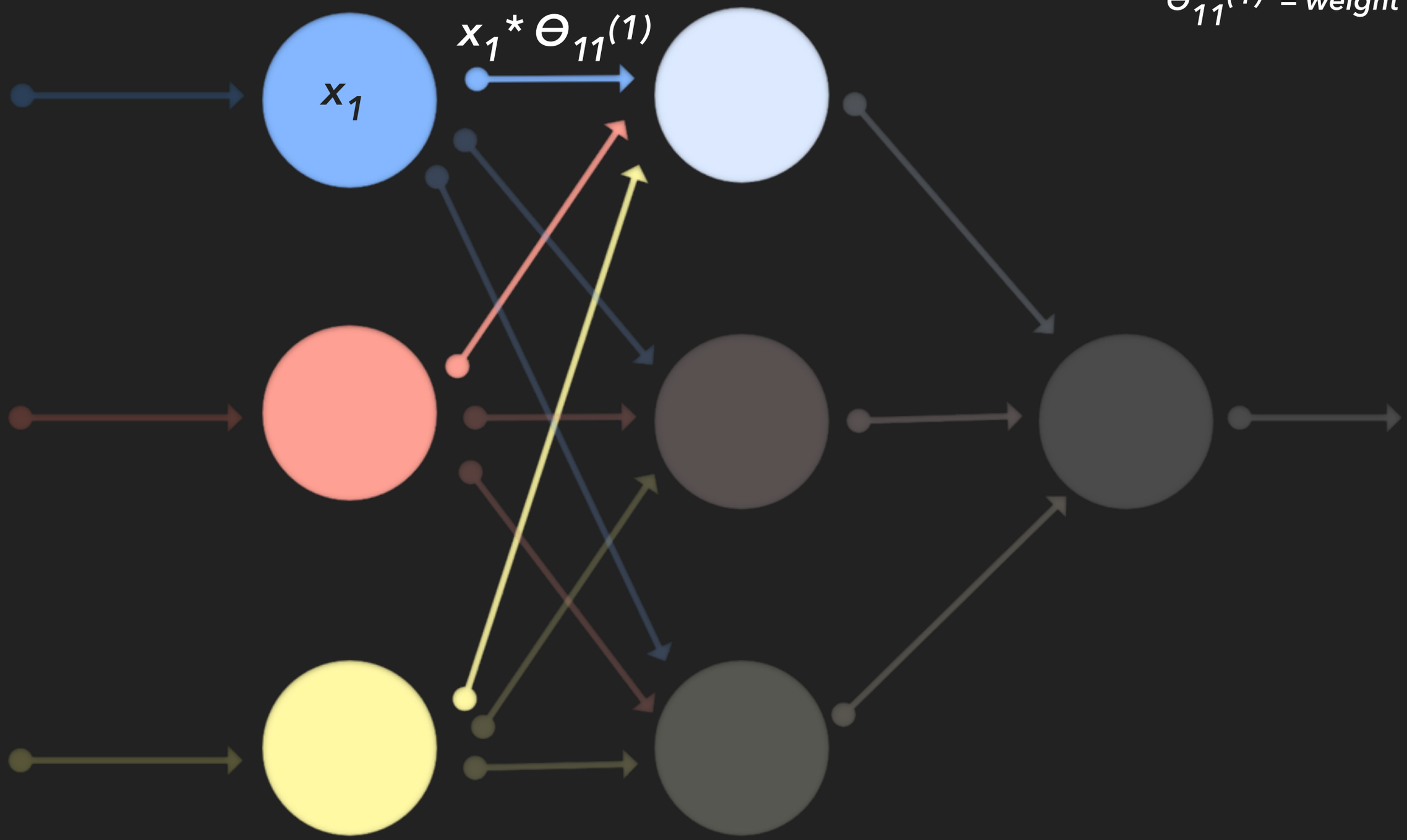
$\Theta_{11}^{(1)}$ = weight



>MAN NEURAL_NETWORK

x_1 = input

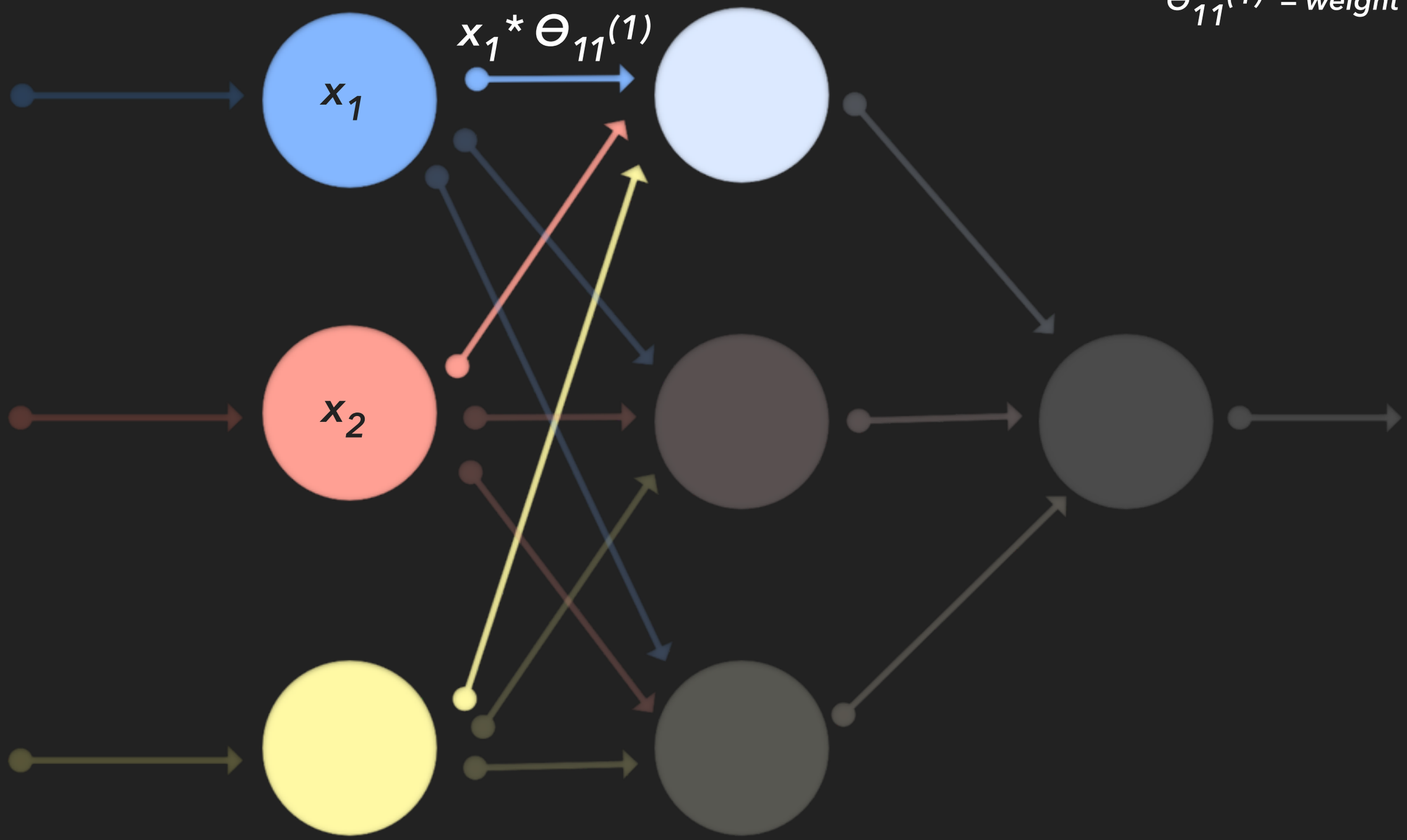
$\Theta_{11}^{(1)}$ = weight



>MAN NEURAL_NETWORK

x_1 = input

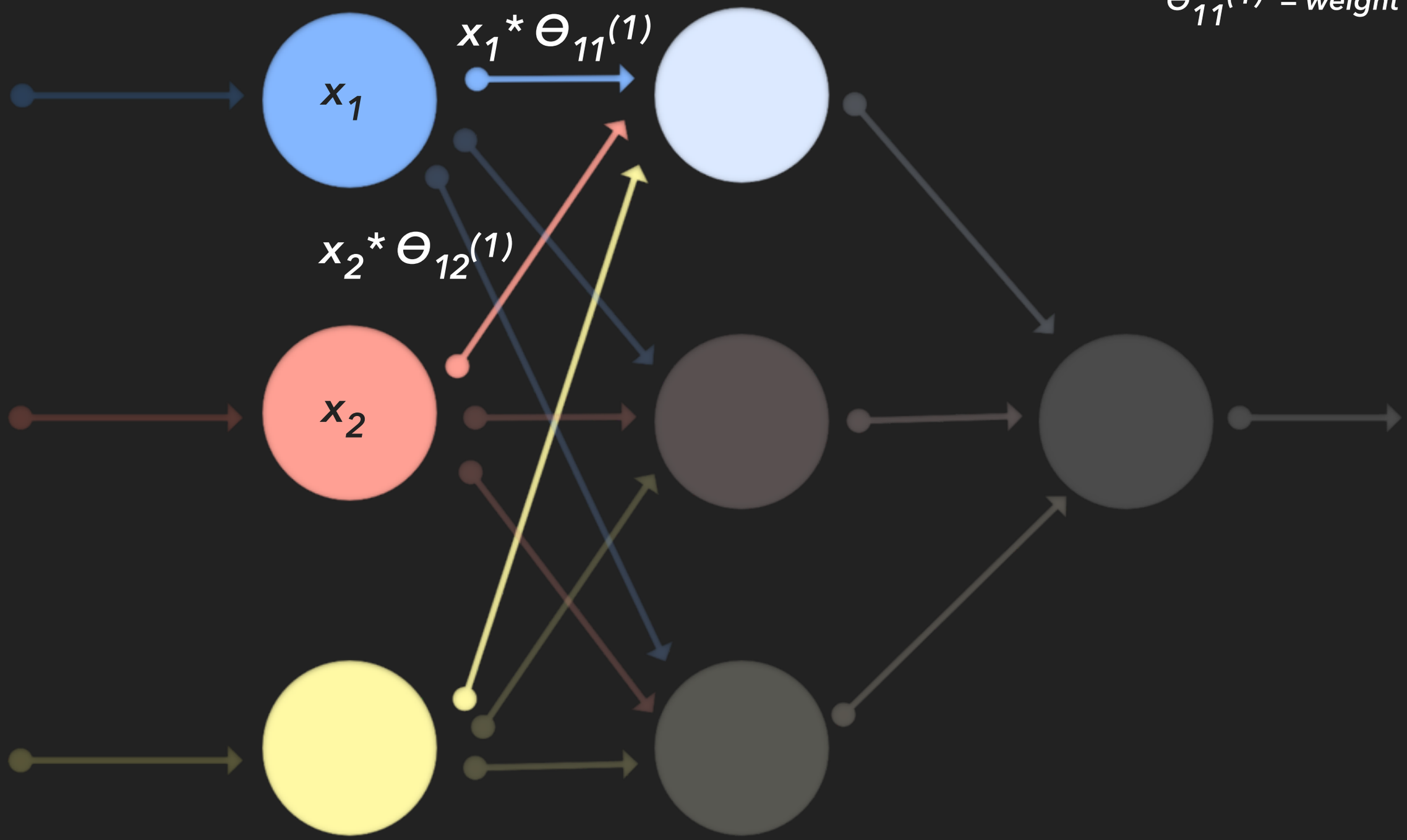
$\Theta_{11}^{(1)}$ = weight



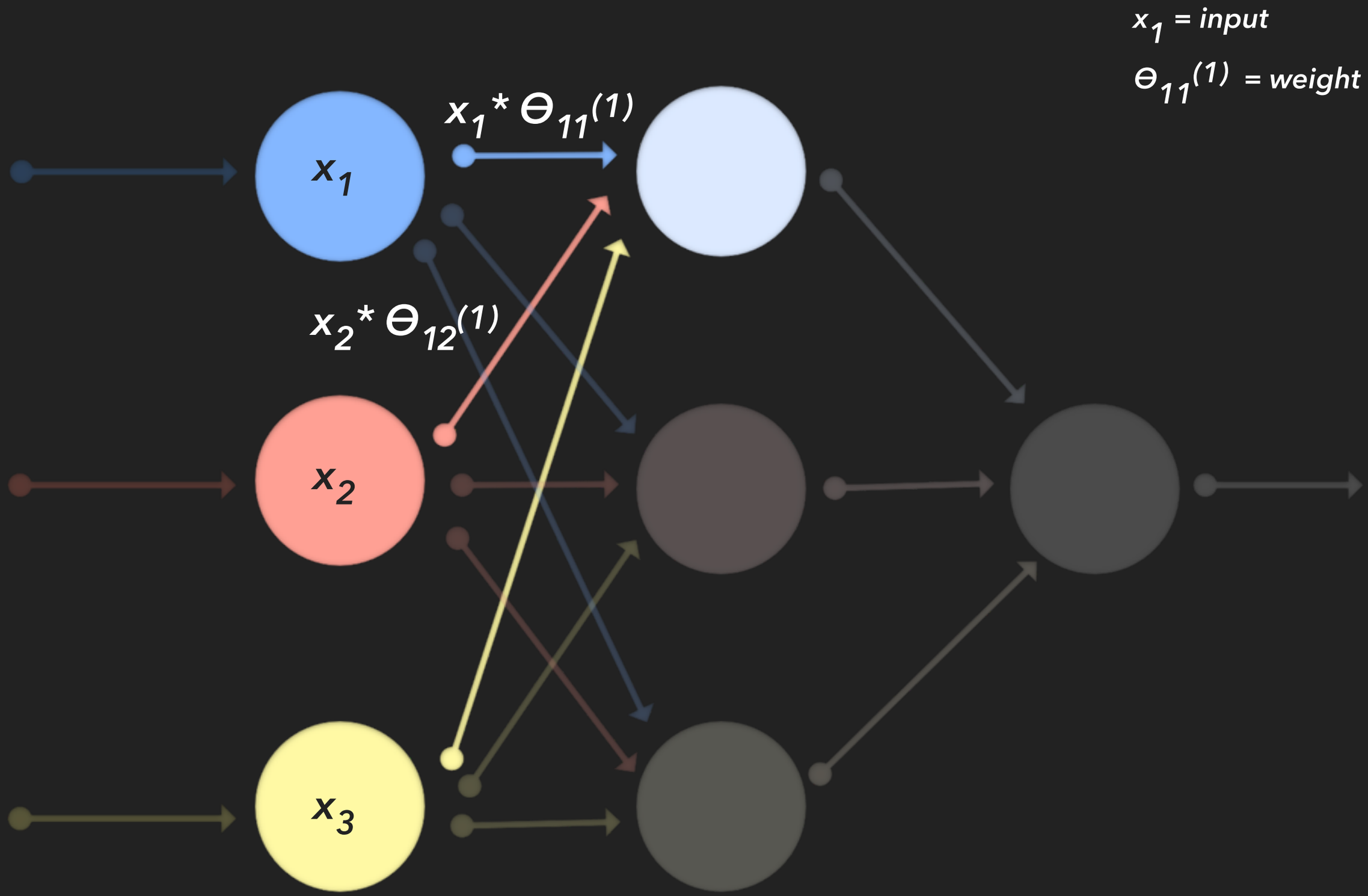
>MAN NEURAL_NETWORK

x_1 = input

$\Theta_{11}^{(1)}$ = weight



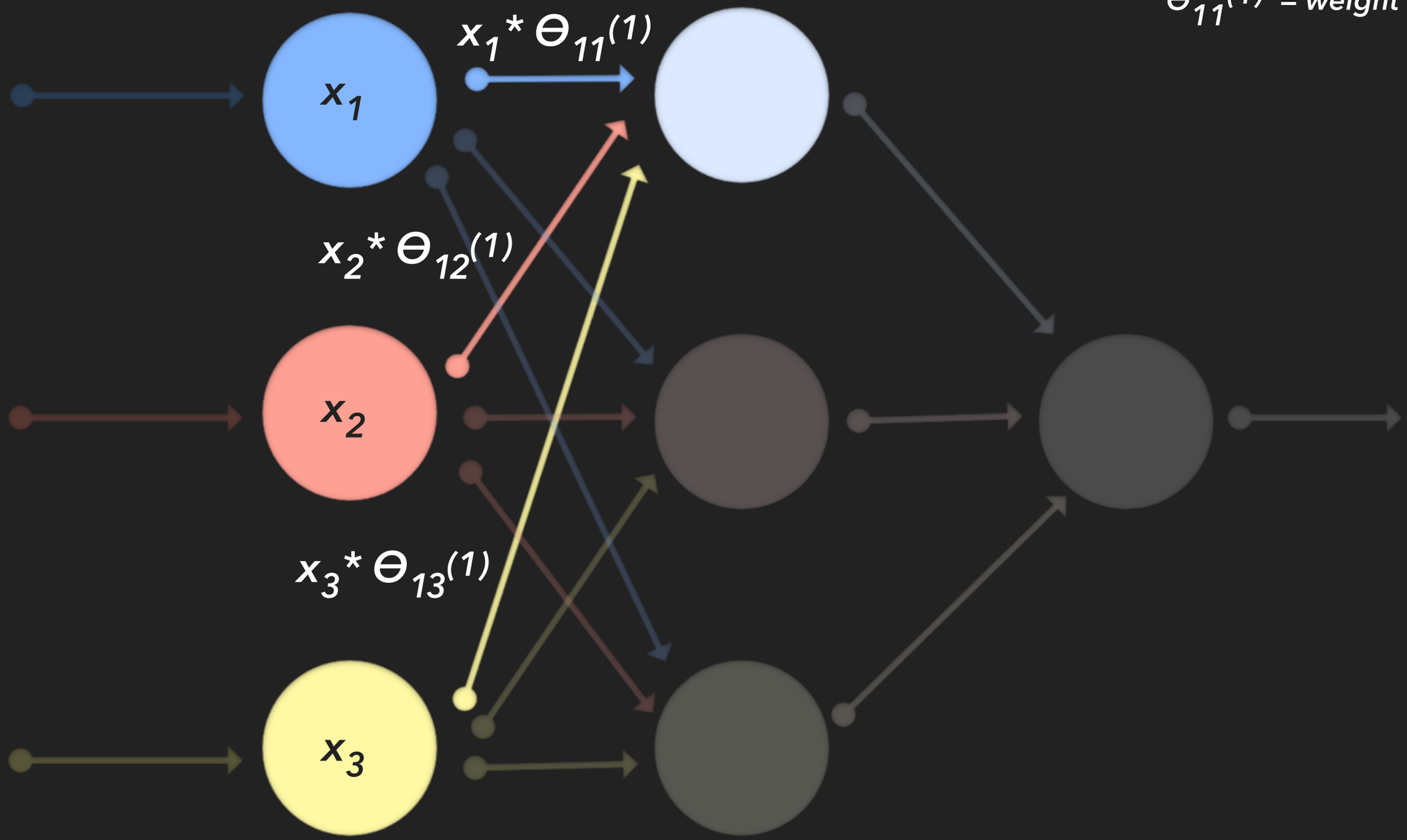
>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK

x_1 = input

$\Theta_{11}^{(1)}$ = weight

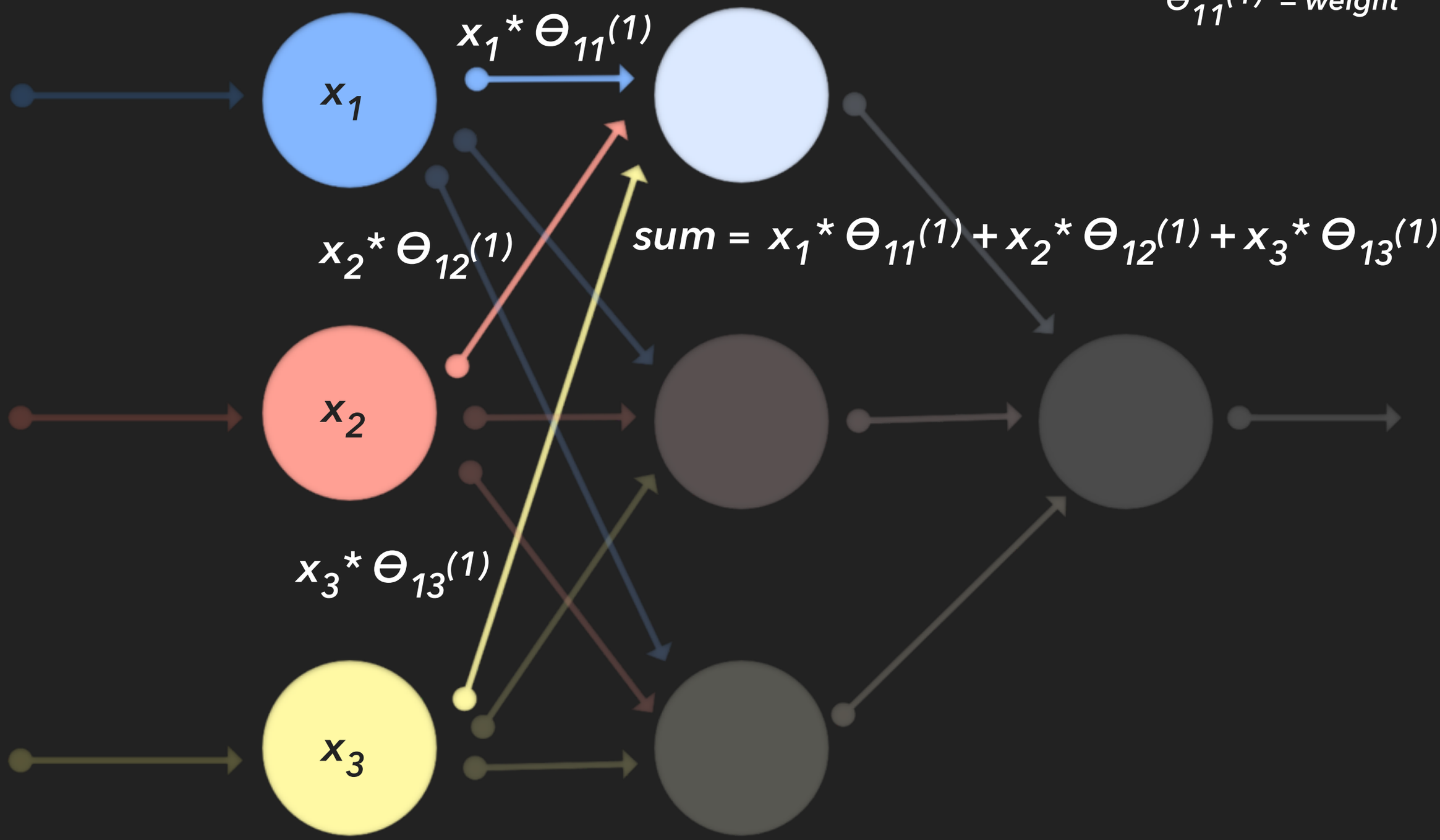


>MAN NEURAL_NETWORK

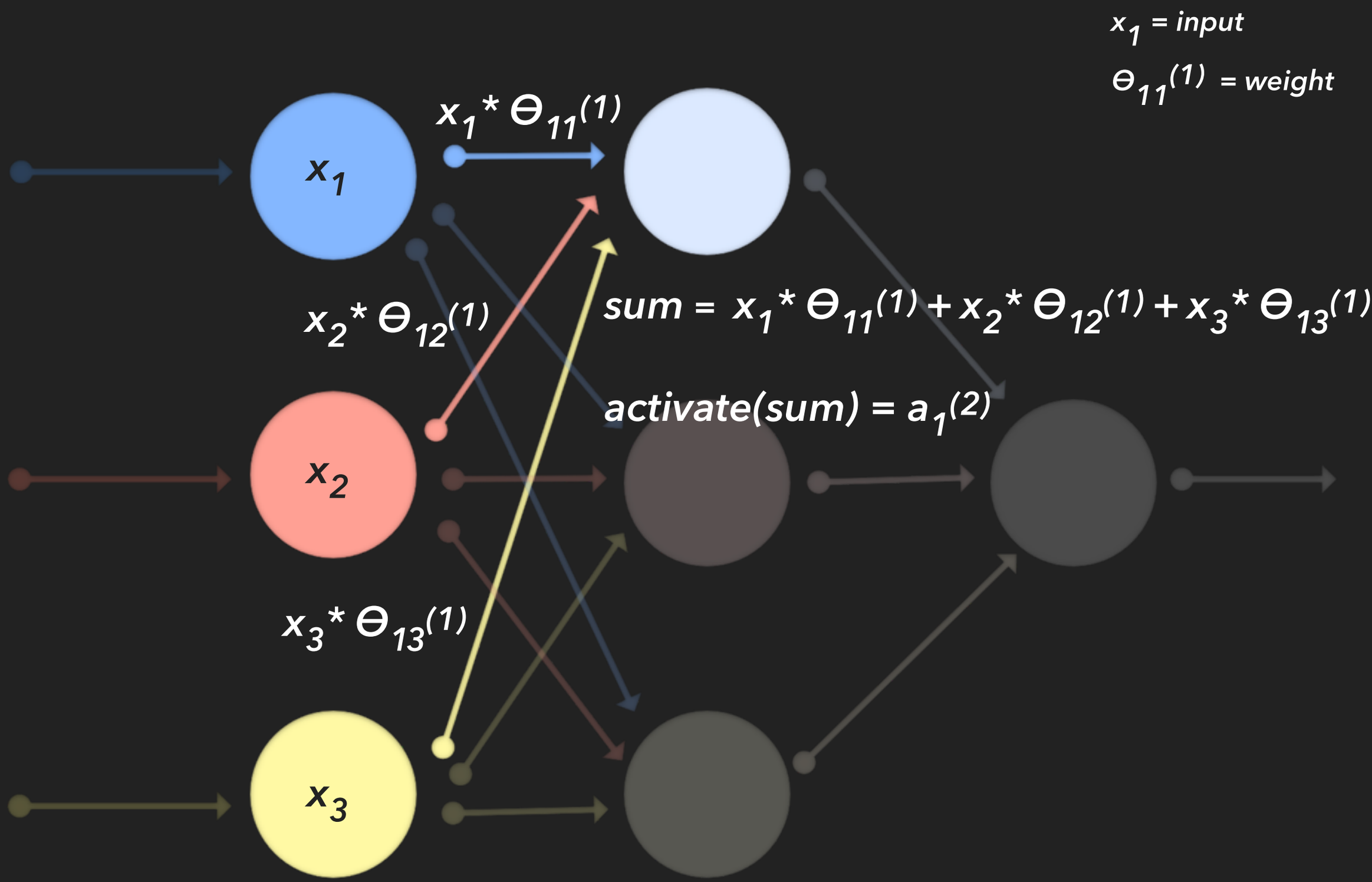
---

x_1 = input

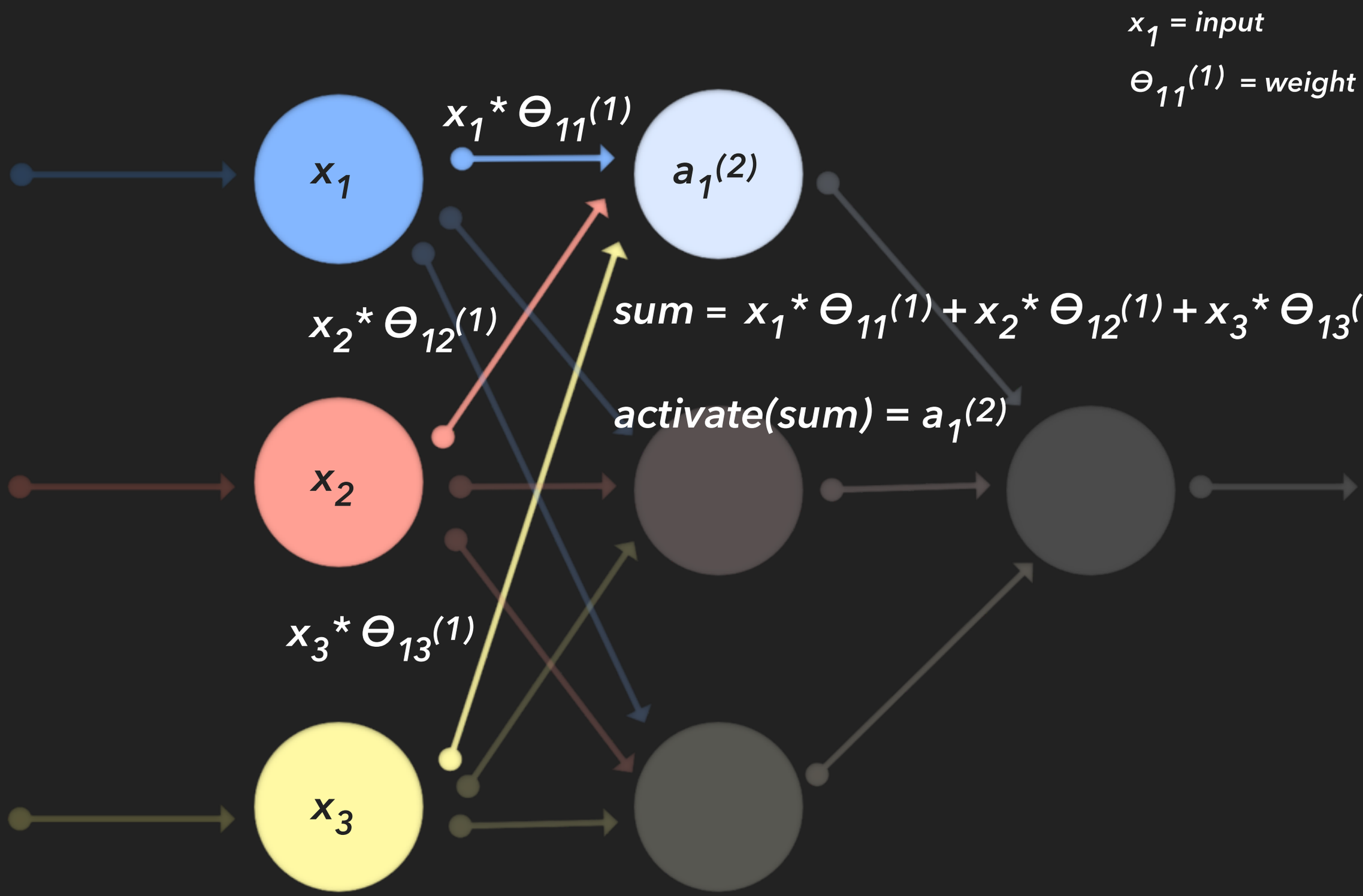
$\Theta_{11}^{(1)}$ = weight



>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK

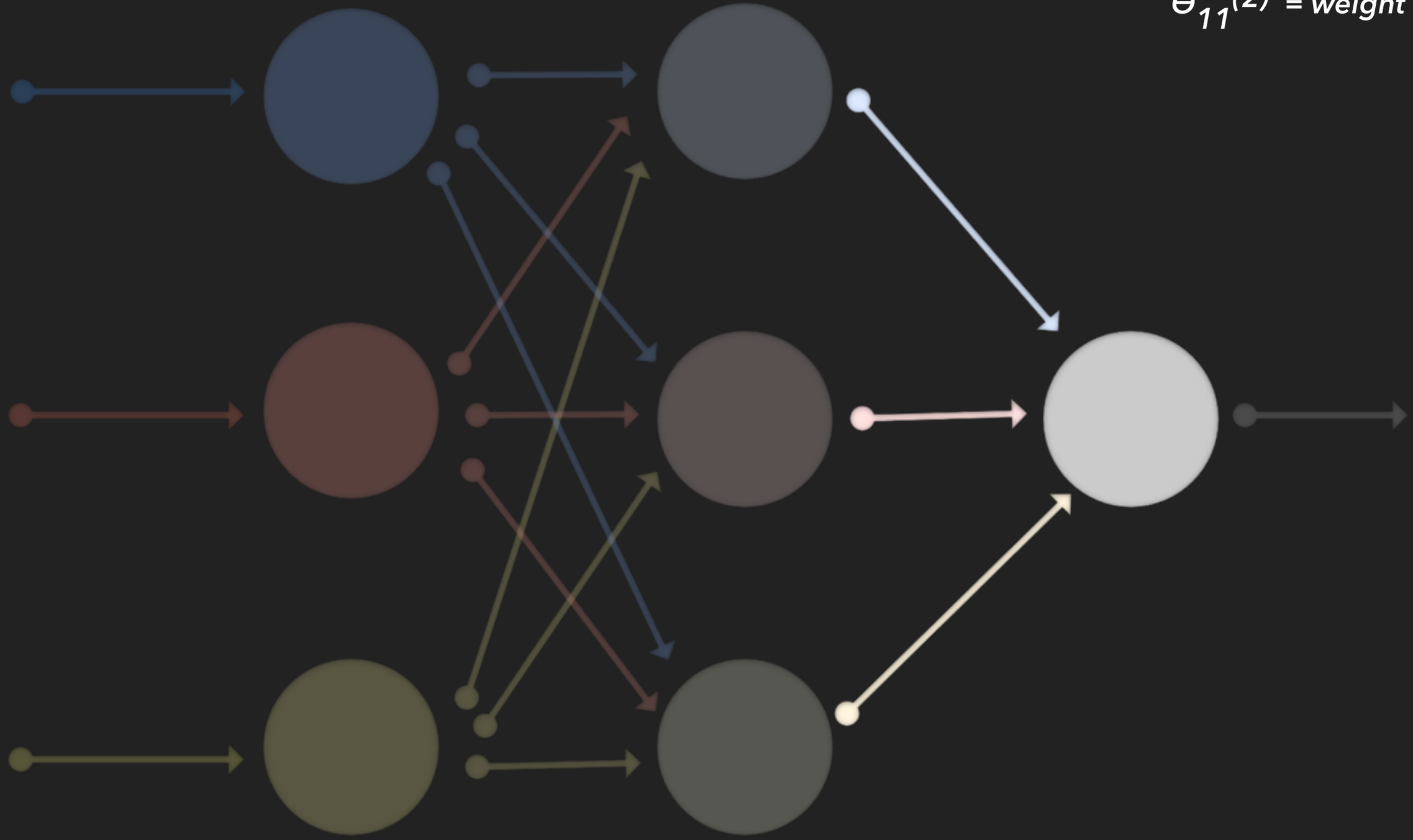
$a_1^{(2)}$ = input

$\Theta_{11}^{(2)}$ = weight

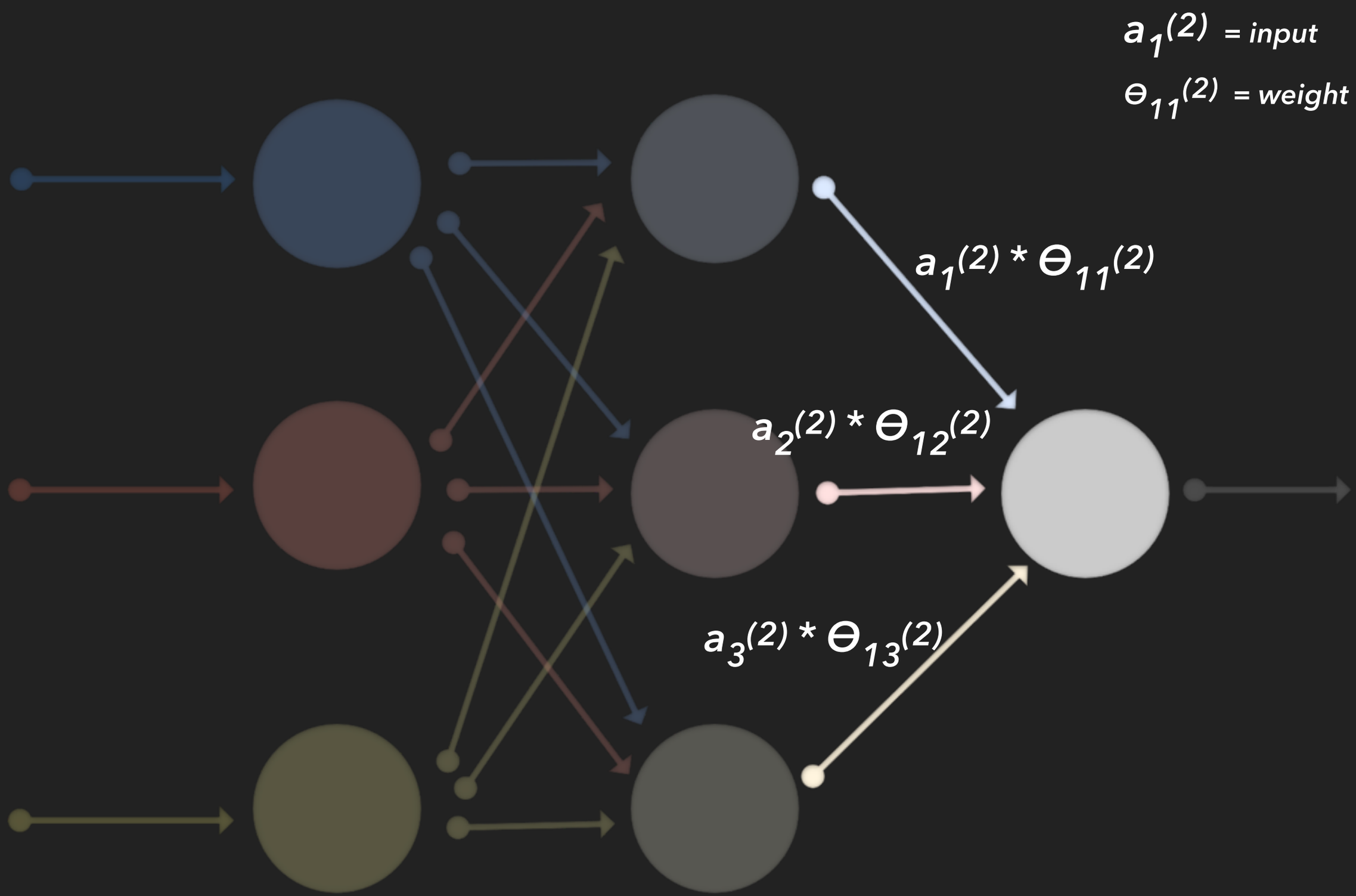
>MAN NEURAL_NETWORK

$a_1^{(2)}$ = input

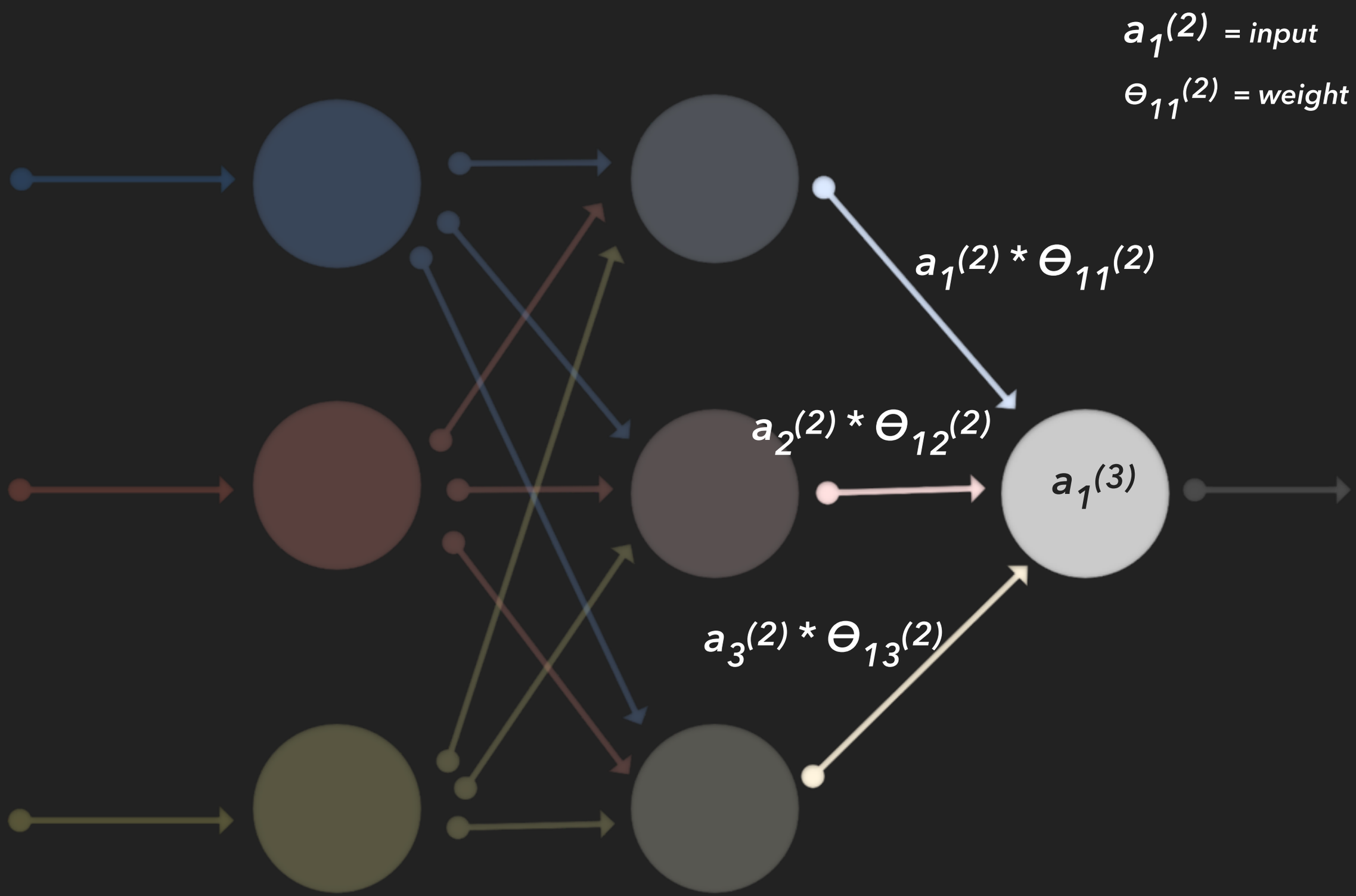
$\Theta_{11}^{(2)}$ = weight



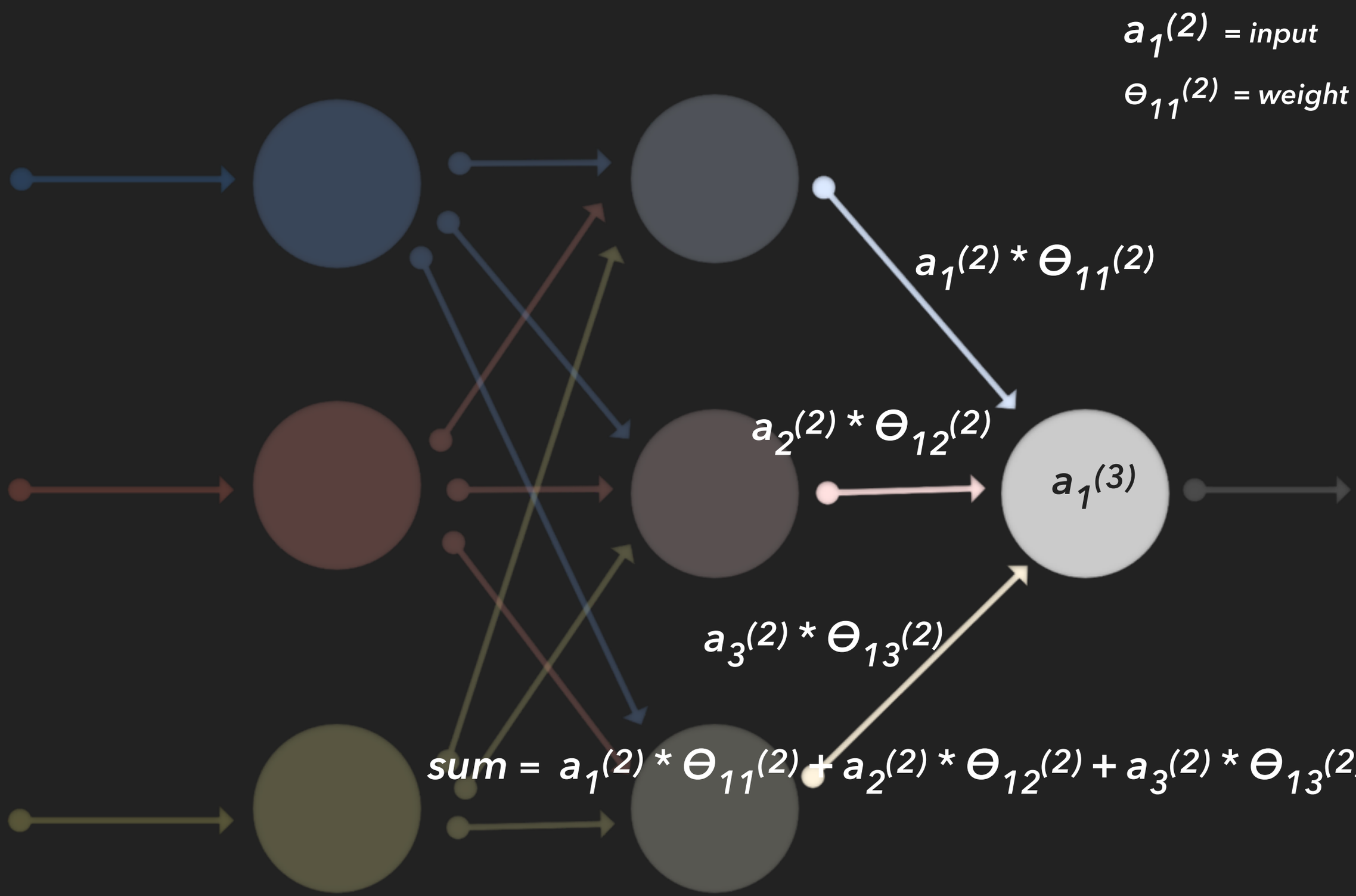
>MAN NEURAL_NETWORK



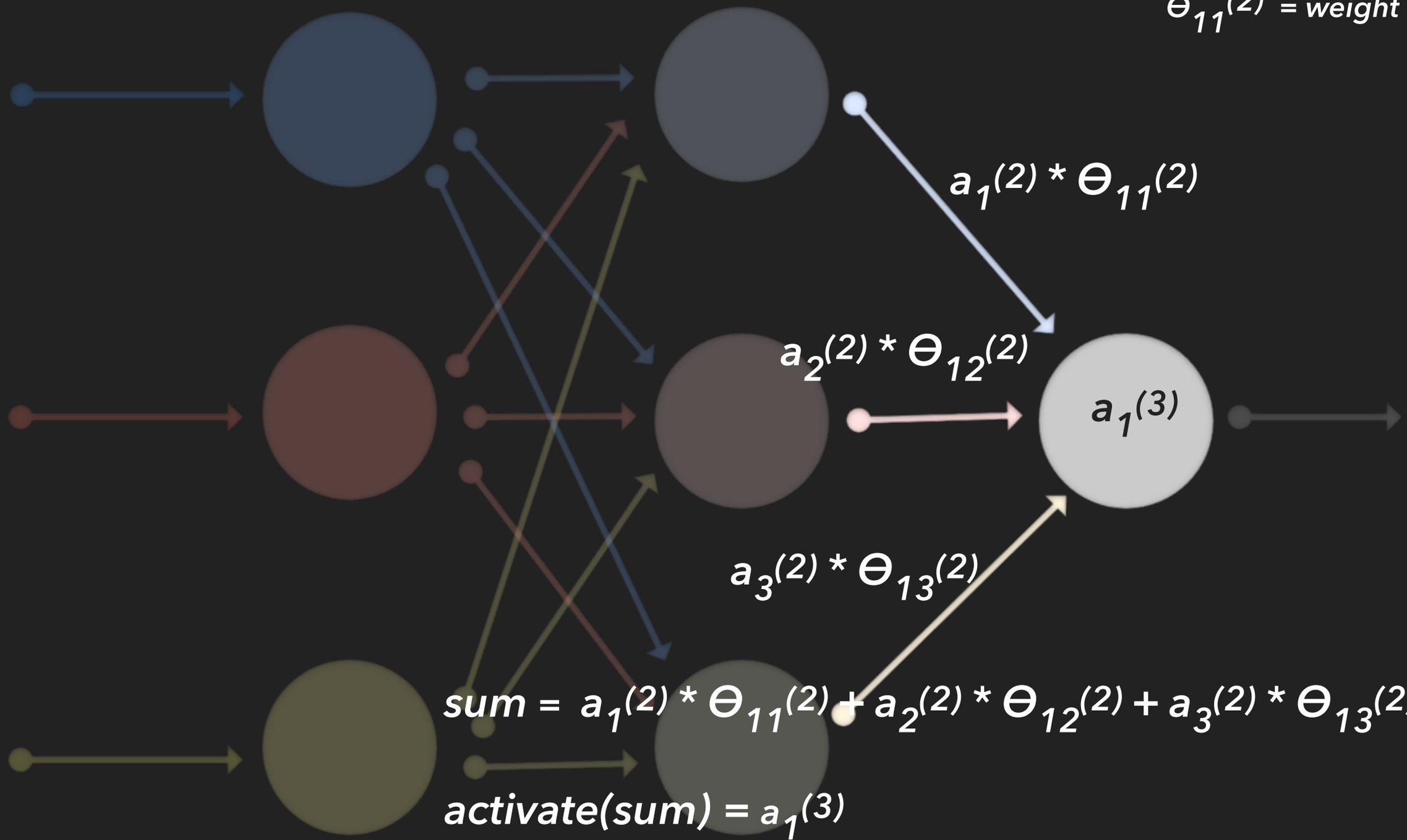
>MAN NEURAL_NETWORK



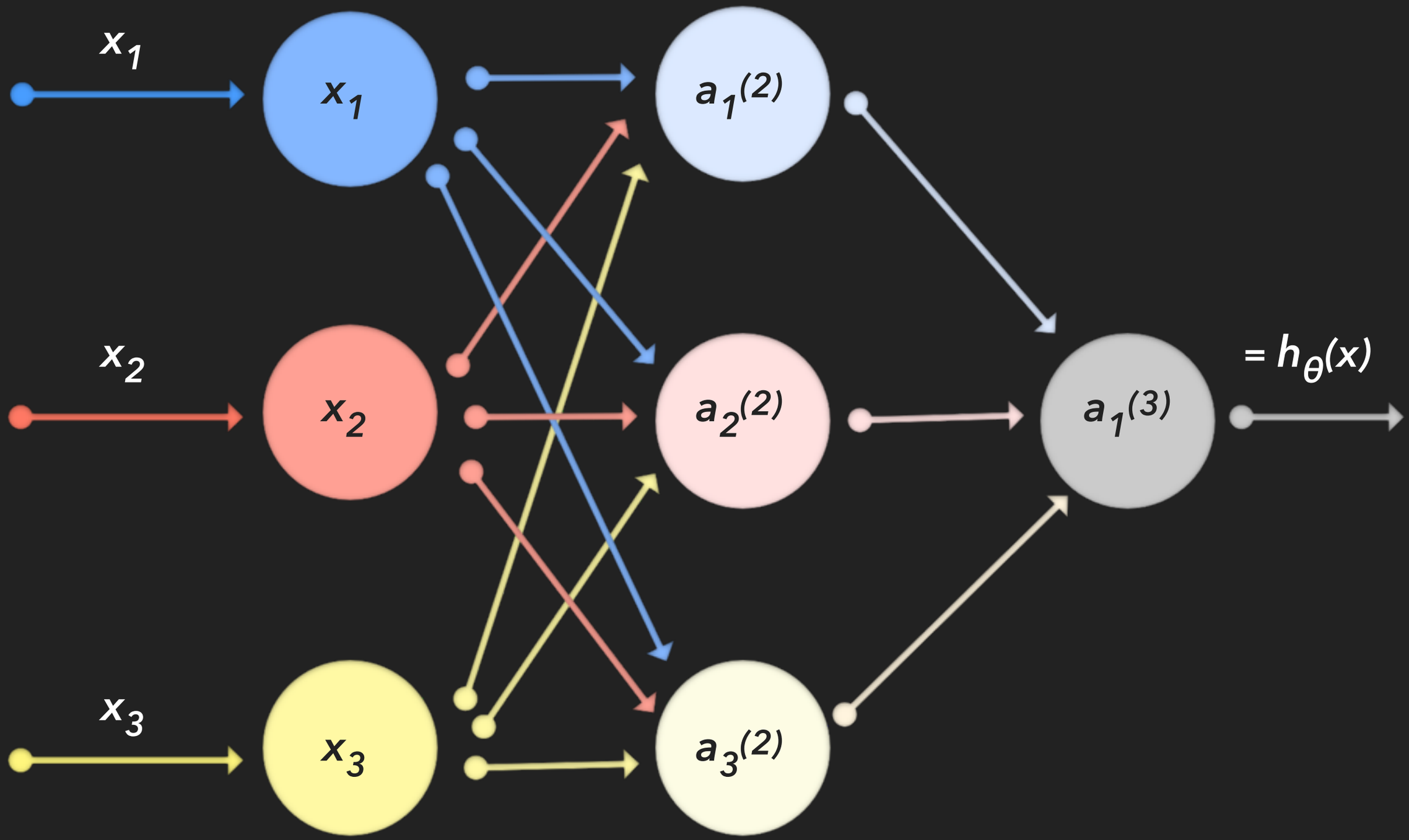
>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK



>RANDOM_STRING_OUTPUT

ALL MODELS ARE WRONG, BUT SOME ARE USEFUL.

George E. P. Box

>MAN NEURAL_NETWORK

>MAN NEURAL_NETWORK

- Supervised learning

>MAN NEURAL_NETWORK

- Supervised learning
  - Quota for betting places

>MAN NEURAL_NETWORK

- Supervised learning
  - Quota for betting places
  - Housing price predictions

>MAN NEURAL_NETWORK

- Supervised learning
  - Quota for betting places
  - Housing price predictions
- Unsupervised learning

>MAN NEURAL_NETWORK

- Supervised learning
  - Quota for betting places
  - Housing price predictions
- Unsupervised learning
  - Netflix recommender System

>MAN NEURAL_NETWORK

>MAN NEURAL_NETWORK

x	y	h(x)
x1 = 10 x2 = 15	43	60
x1 = 6 x2 = 90	26	10
x1 = 30 x2 = 34	54	40
x1 = 25 x2 = 2	32	5

>MAN NEURAL_NETWORK

x	y	h(x)
x1 = 10 x2 = 15	43	60
x1 = 6 x2 = 90	26	10
x1 = 30 x2 = 34	54	40
x1 = 25 x2 = 2	32	5

- $h(x_1) = 60; y_1 = 43$

>MAN NEURAL_NETWORK

x	y	h(x)
x1 = 10 x2 = 15	43	60
x1 = 6 x2 = 90	26	10
x1 = 30 x2 = 34	54	40
x1 = 25 x2 = 2	32	5

- $h(x_1) = 60; y_1 = 43$
- $h(x_1) - y_1 = Cost_1 = J(\theta_1)$

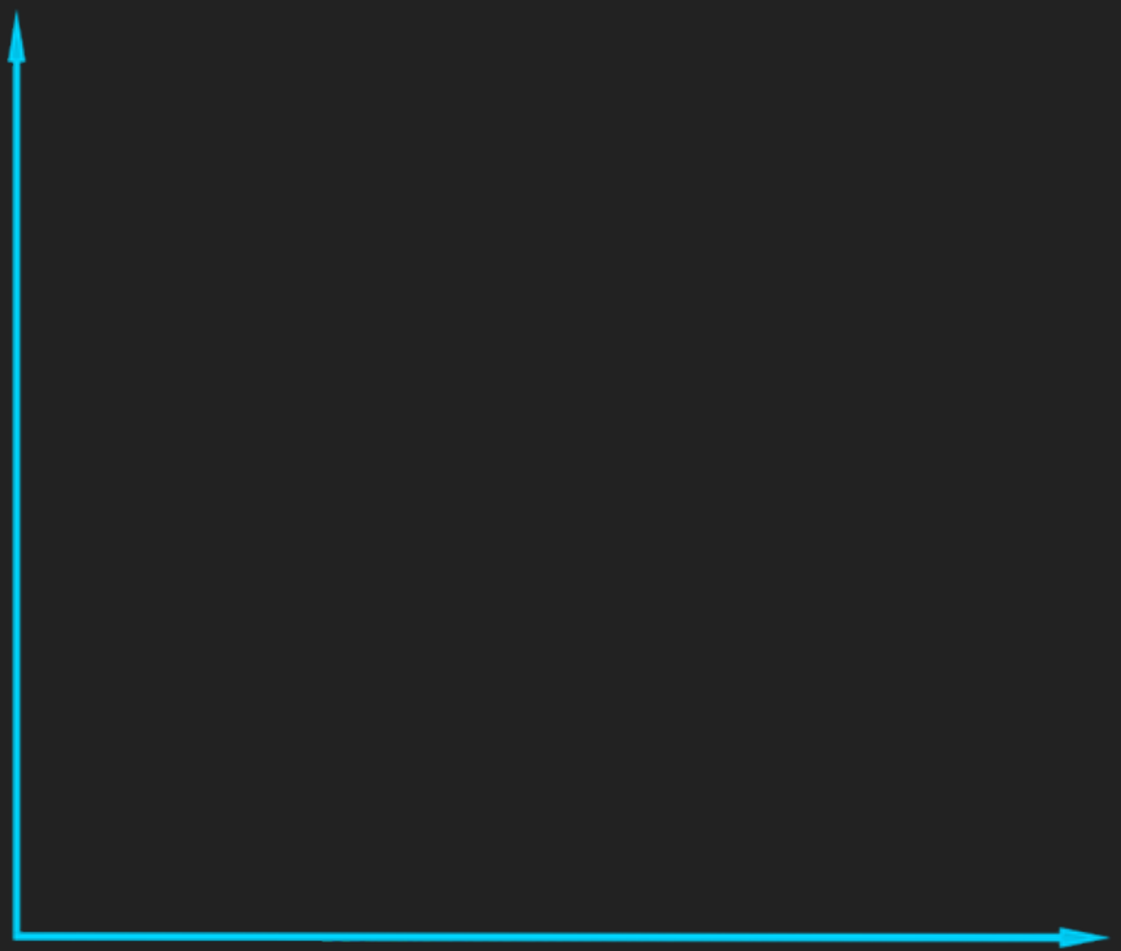
>MAN NEURAL_NETWORK

x	y	h(x)
x1 = 10 x2 = 15	43	60
x1 = 6 x2 = 90	26	10
x1 = 30 x2 = 34	54	40
x1 = 25 x2 = 2	32	5

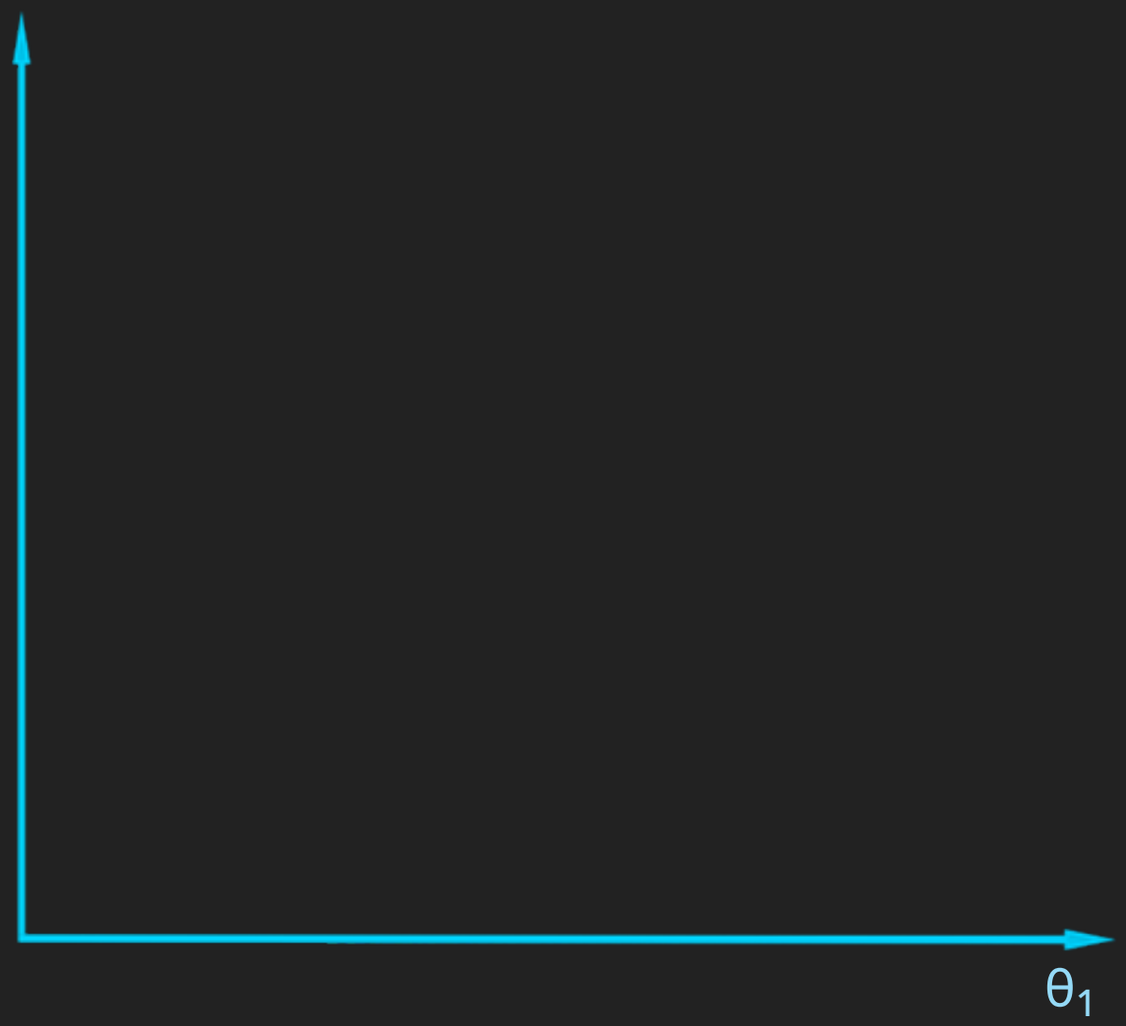
- $h(x_1) = 60; y_1 = 43$
- $h(x_1) - y_1 = Cost_1 = J(\theta_1)$
- Gradient Descent

>MAN NEURAL_NETWORK

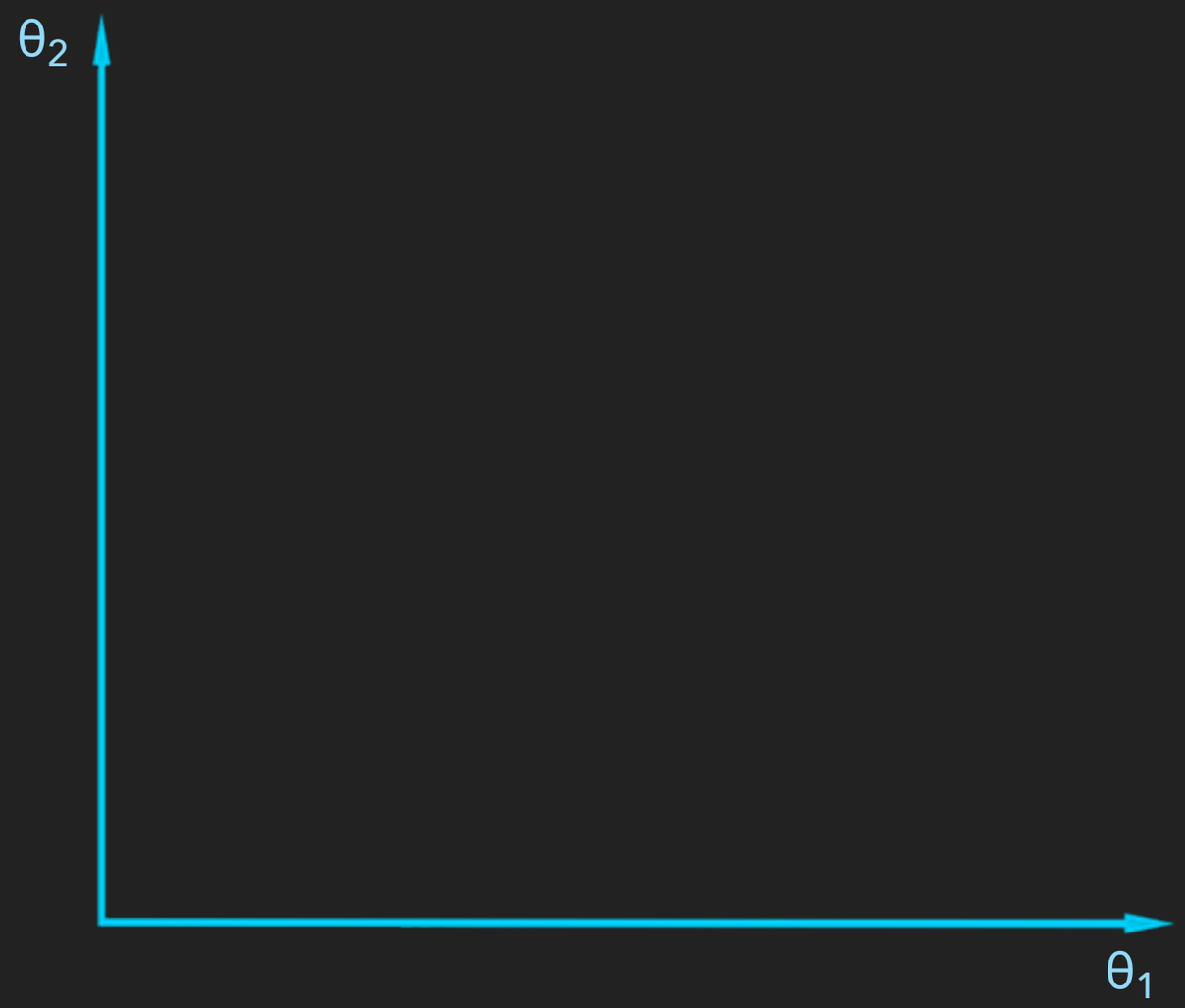
>MAN NEURAL_NETWORK



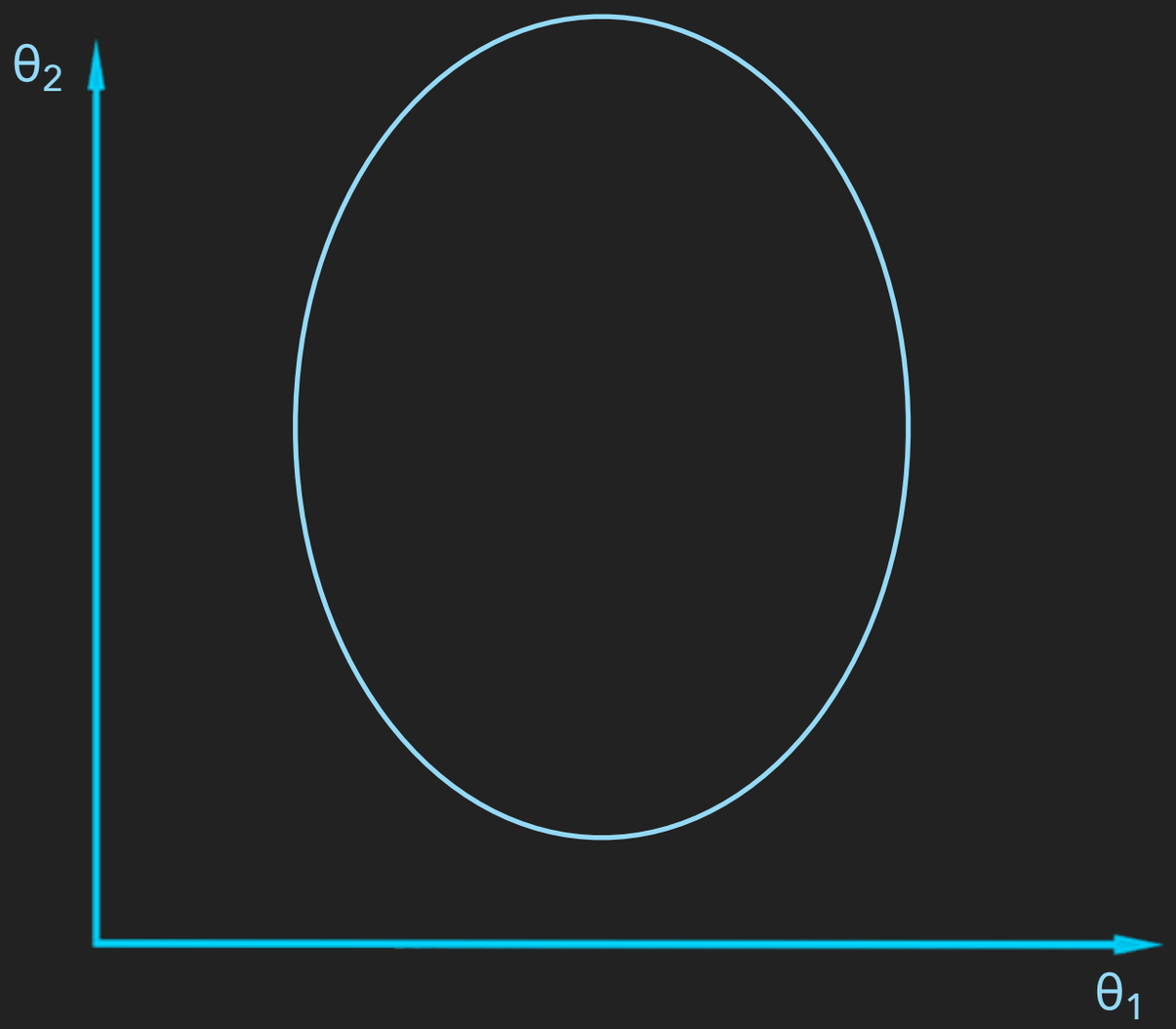
>MAN NEURAL_NETWORK



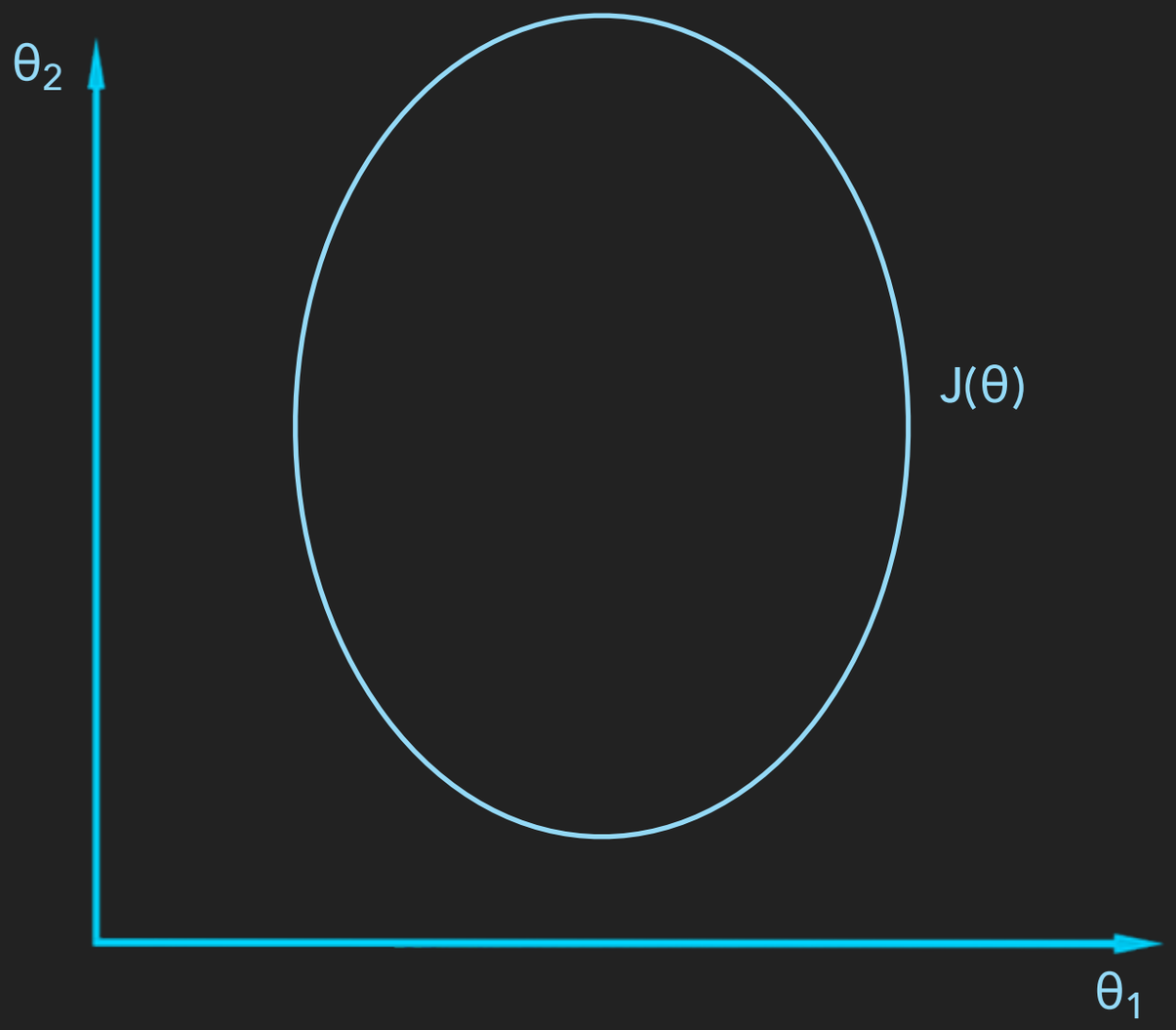
>MAN NEURAL_NETWORK

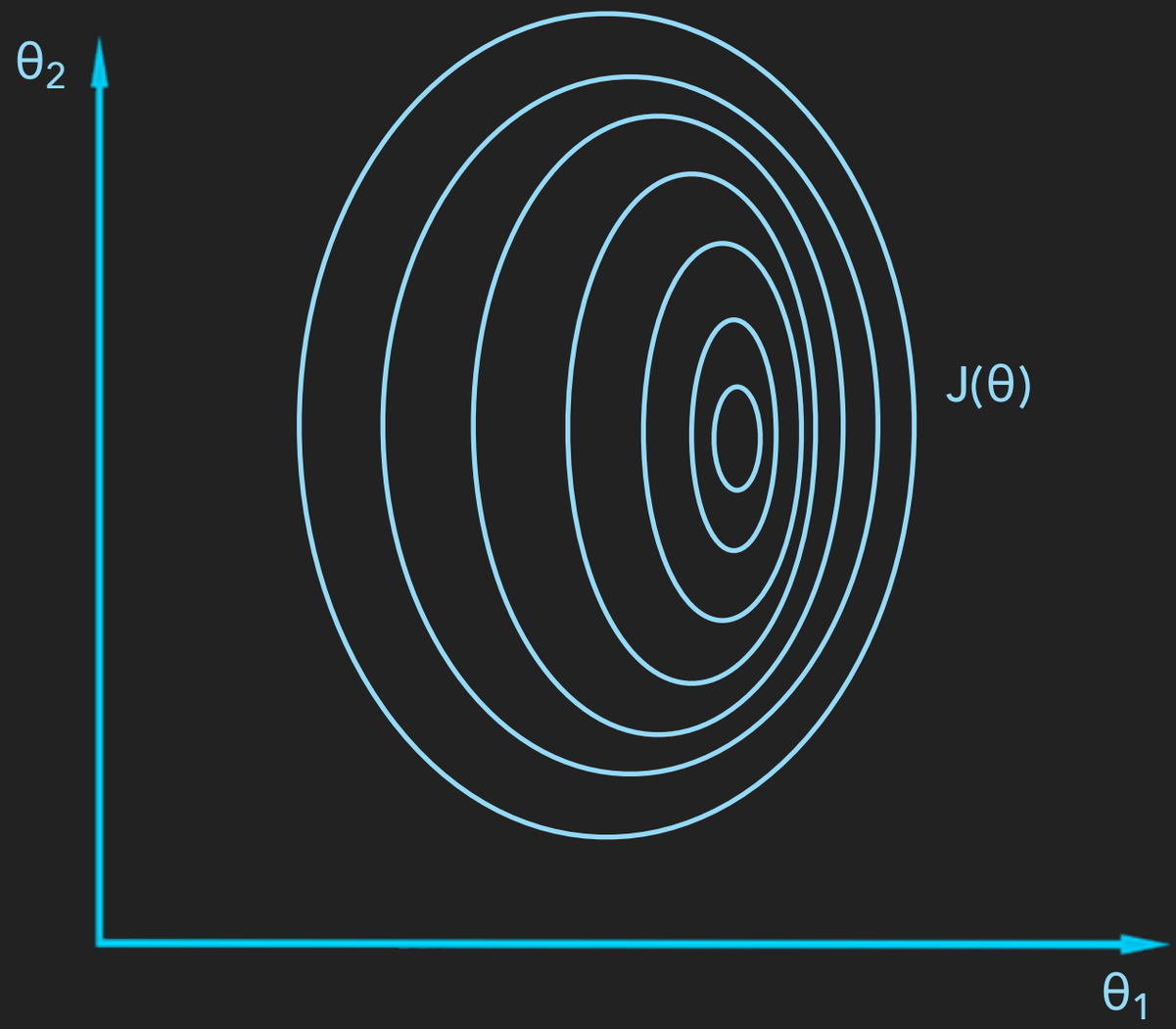


>MAN NEURAL_NETWORK

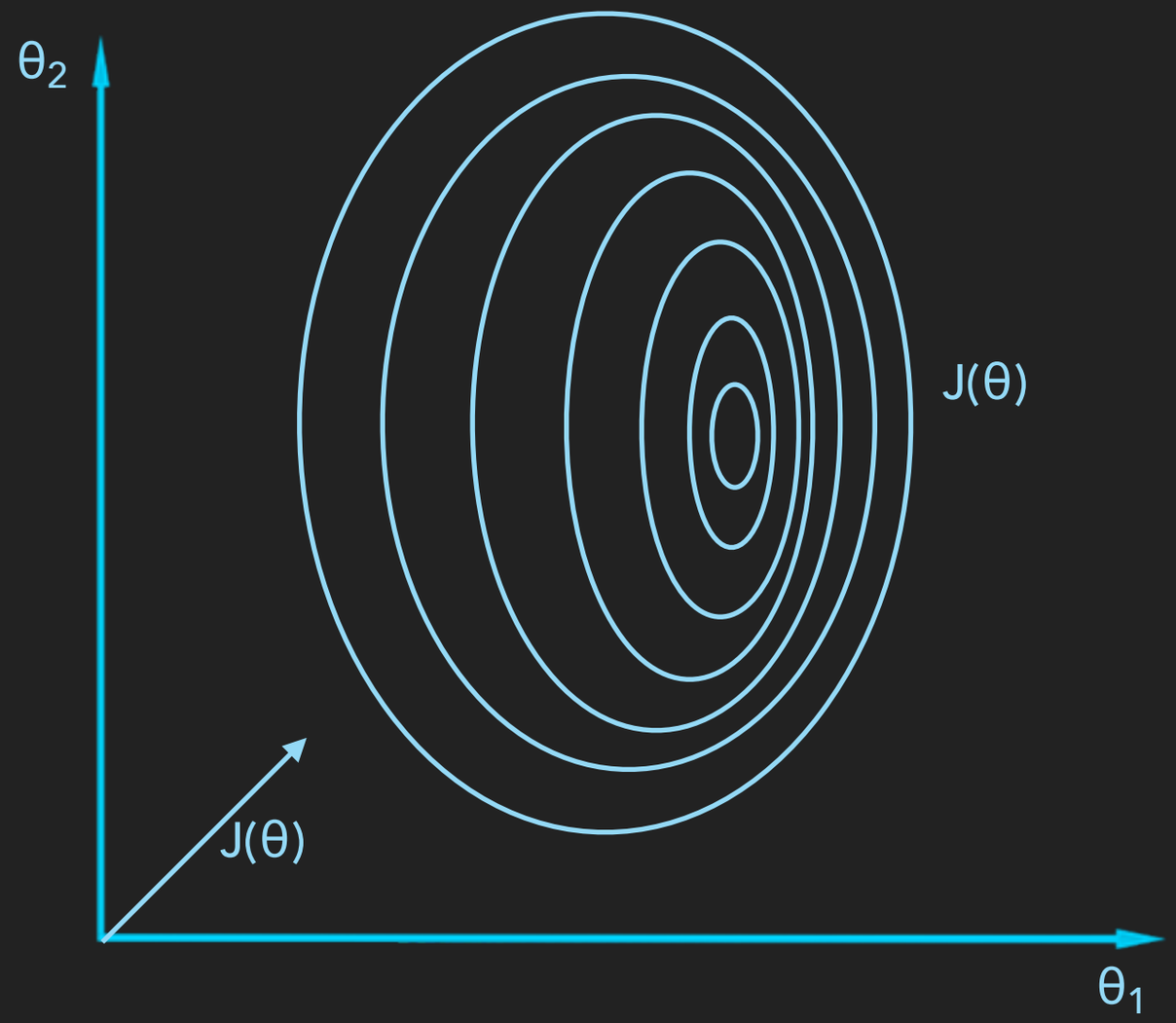


>MAN NEURAL_NETWORK

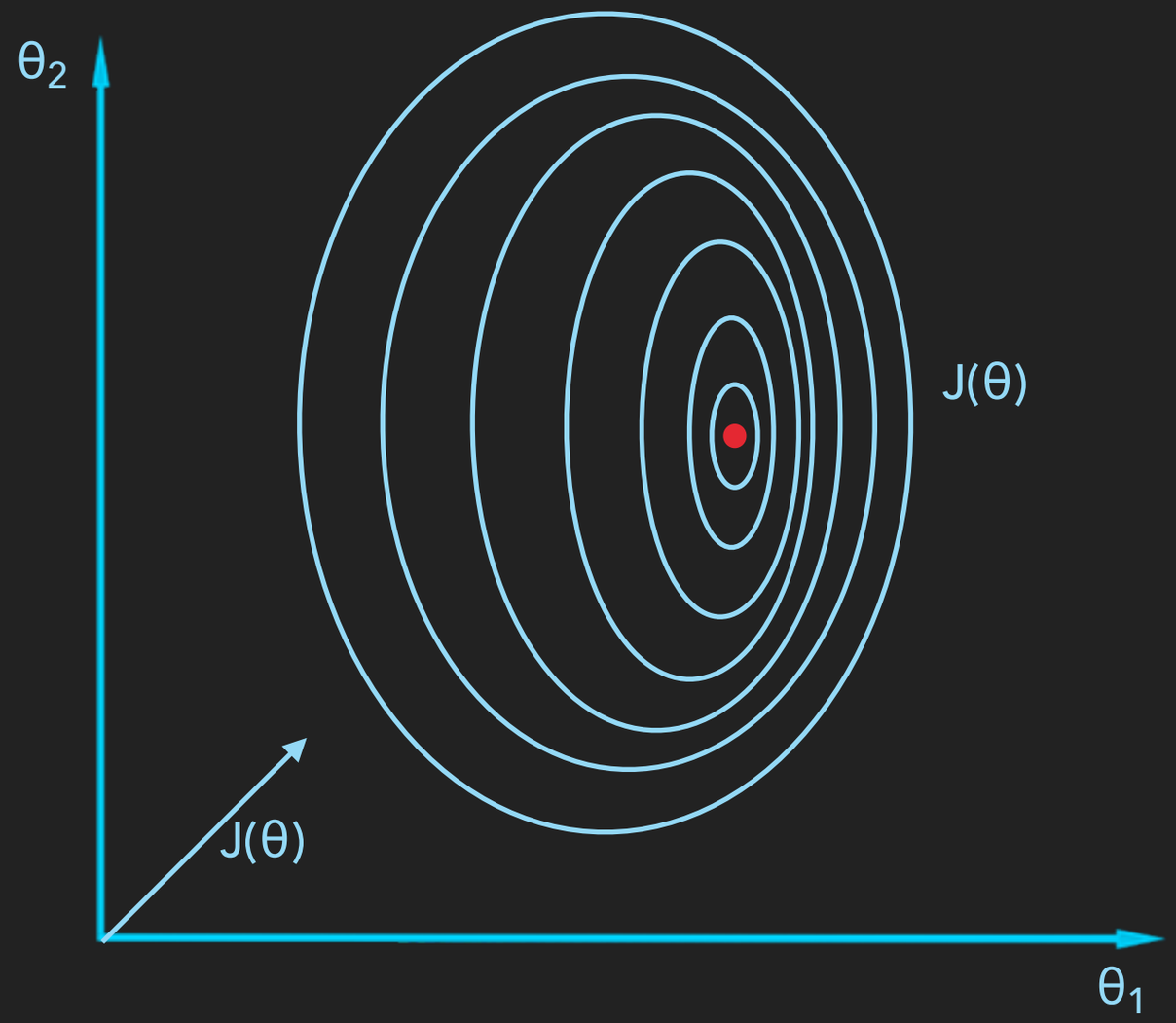



θ2
J(θ)
θ1

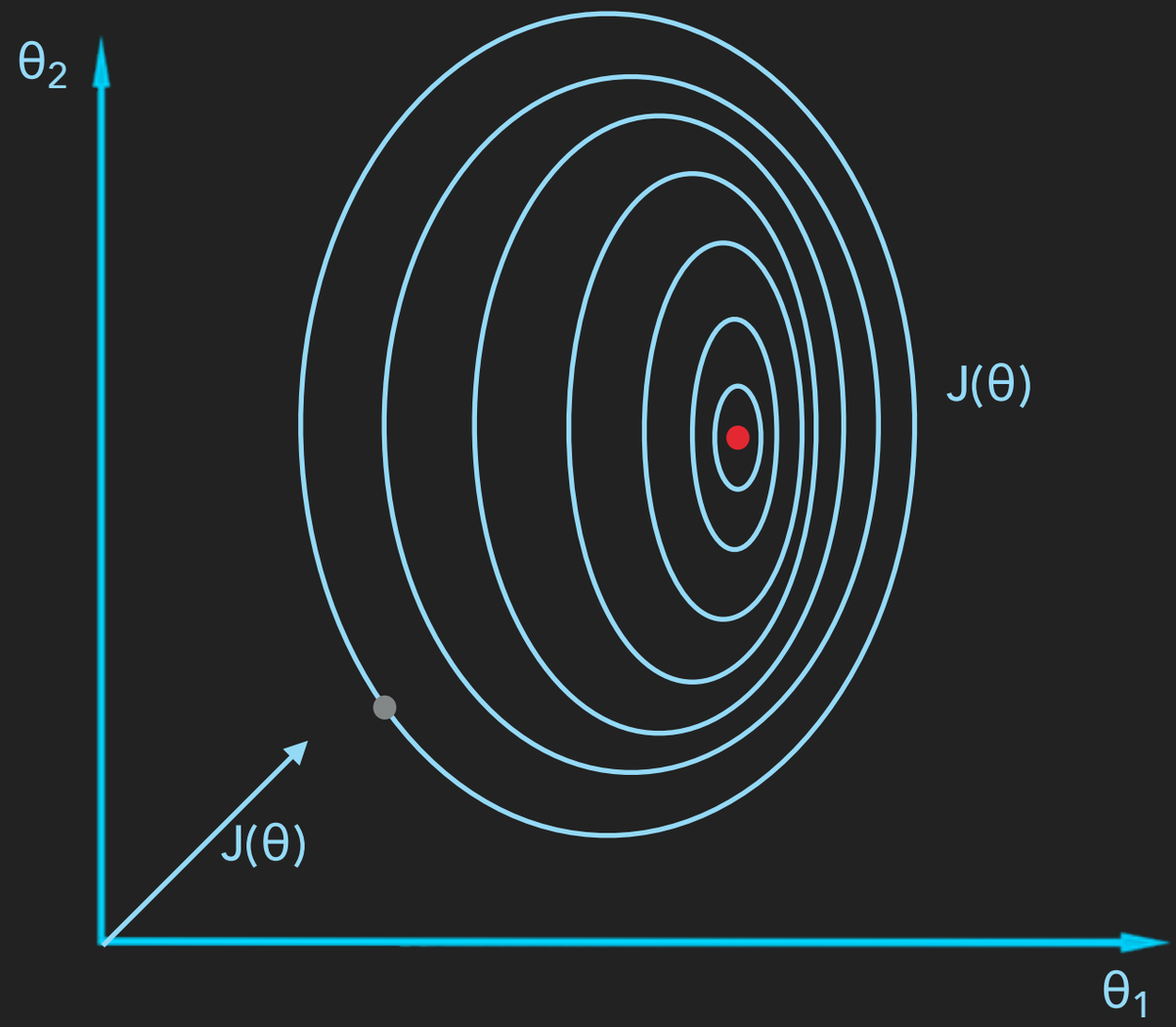
>MAN NEURAL_NETWORK



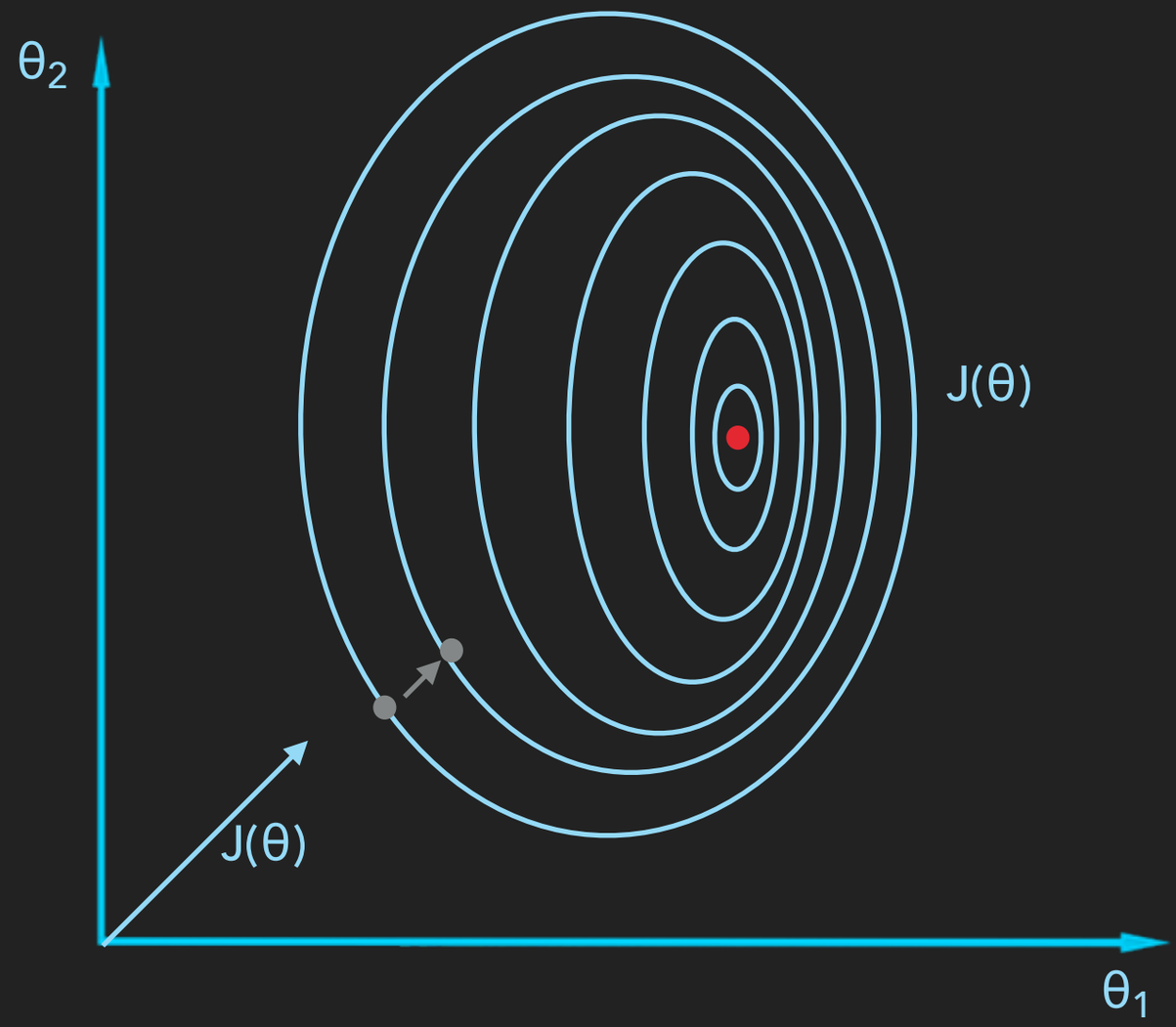
>MAN NEURAL_NETWORK



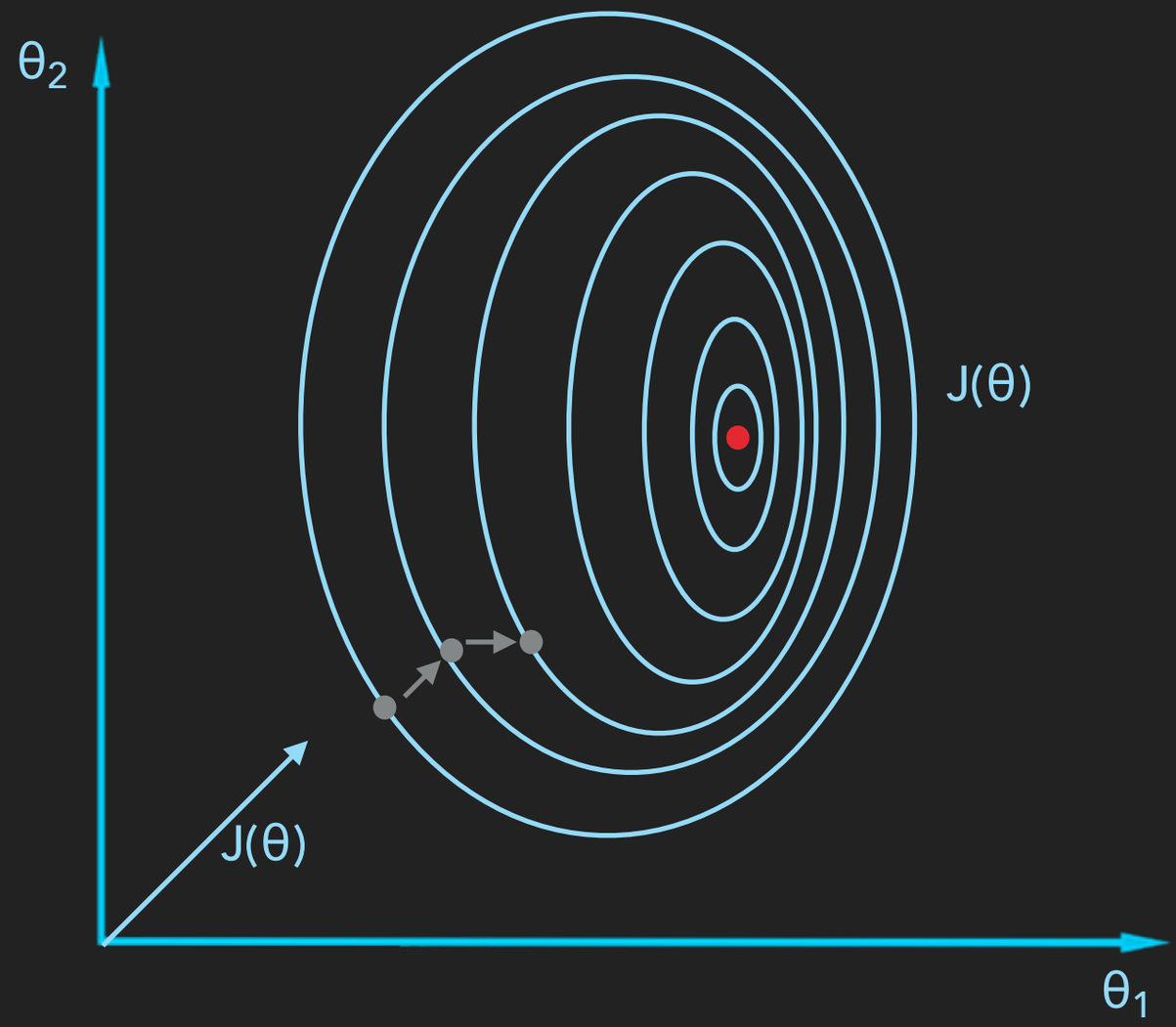
>MAN NEURAL_NETWORK



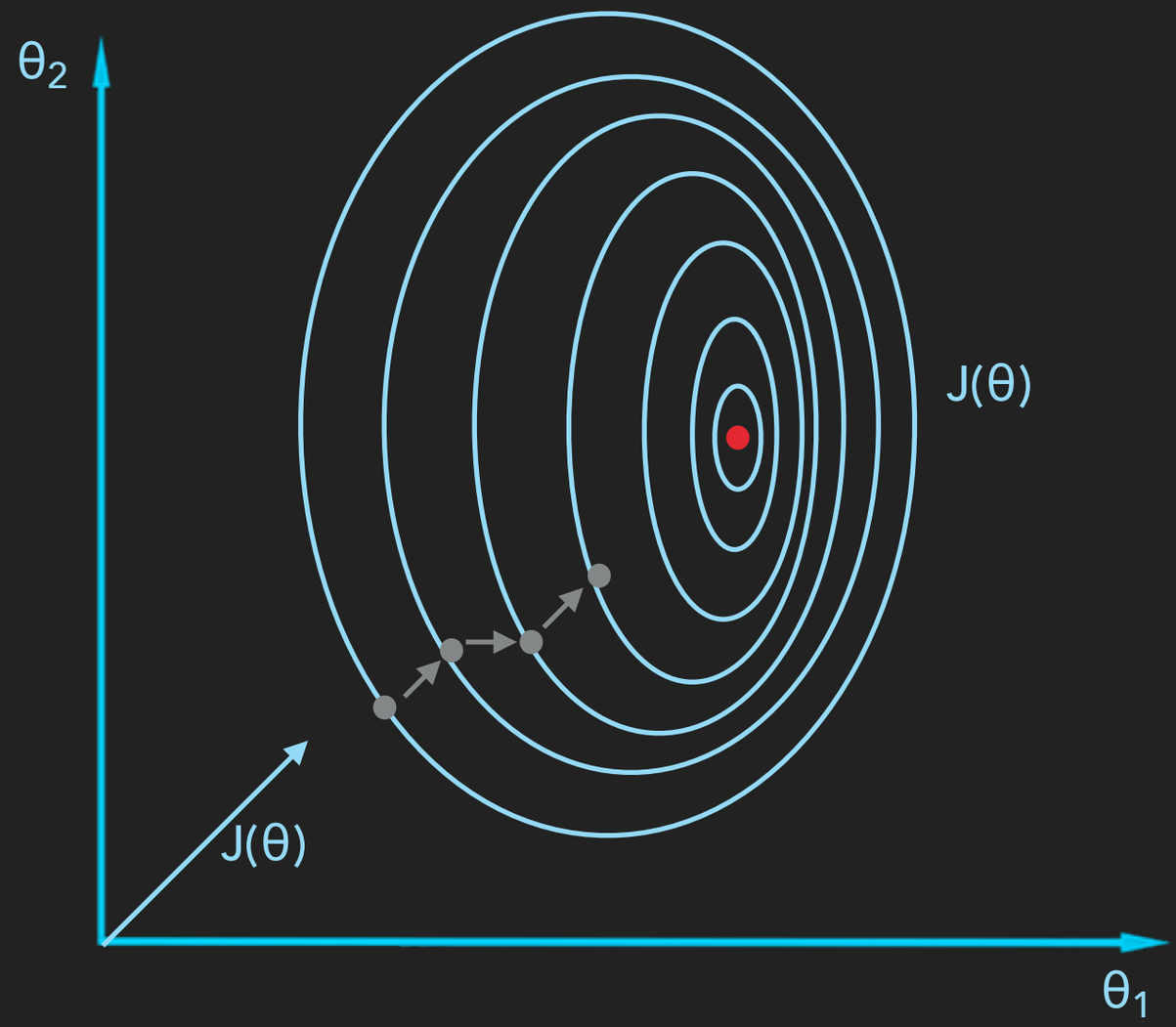
>MAN NEURAL_NETWORK



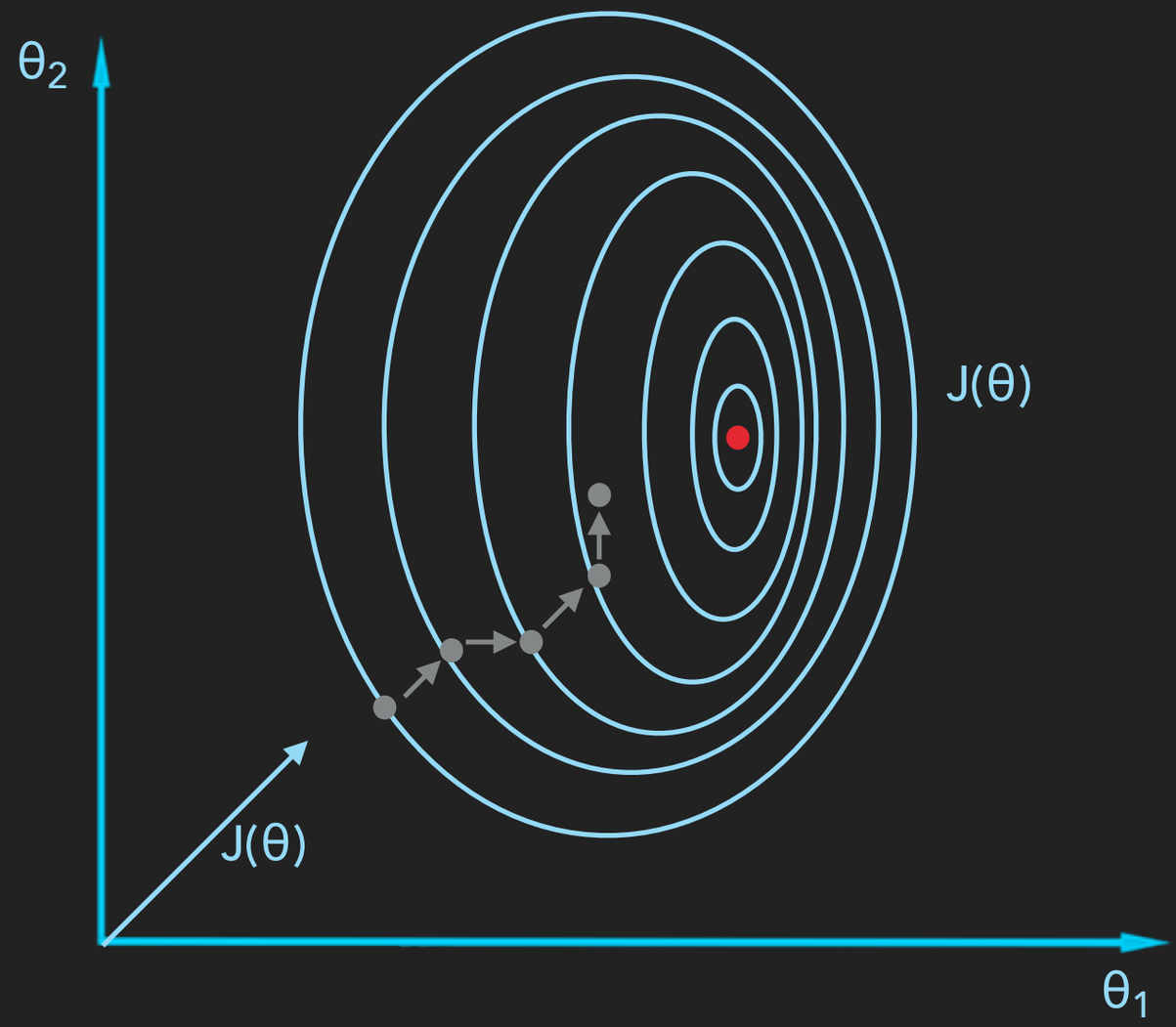
>MAN NEURAL_NETWORK



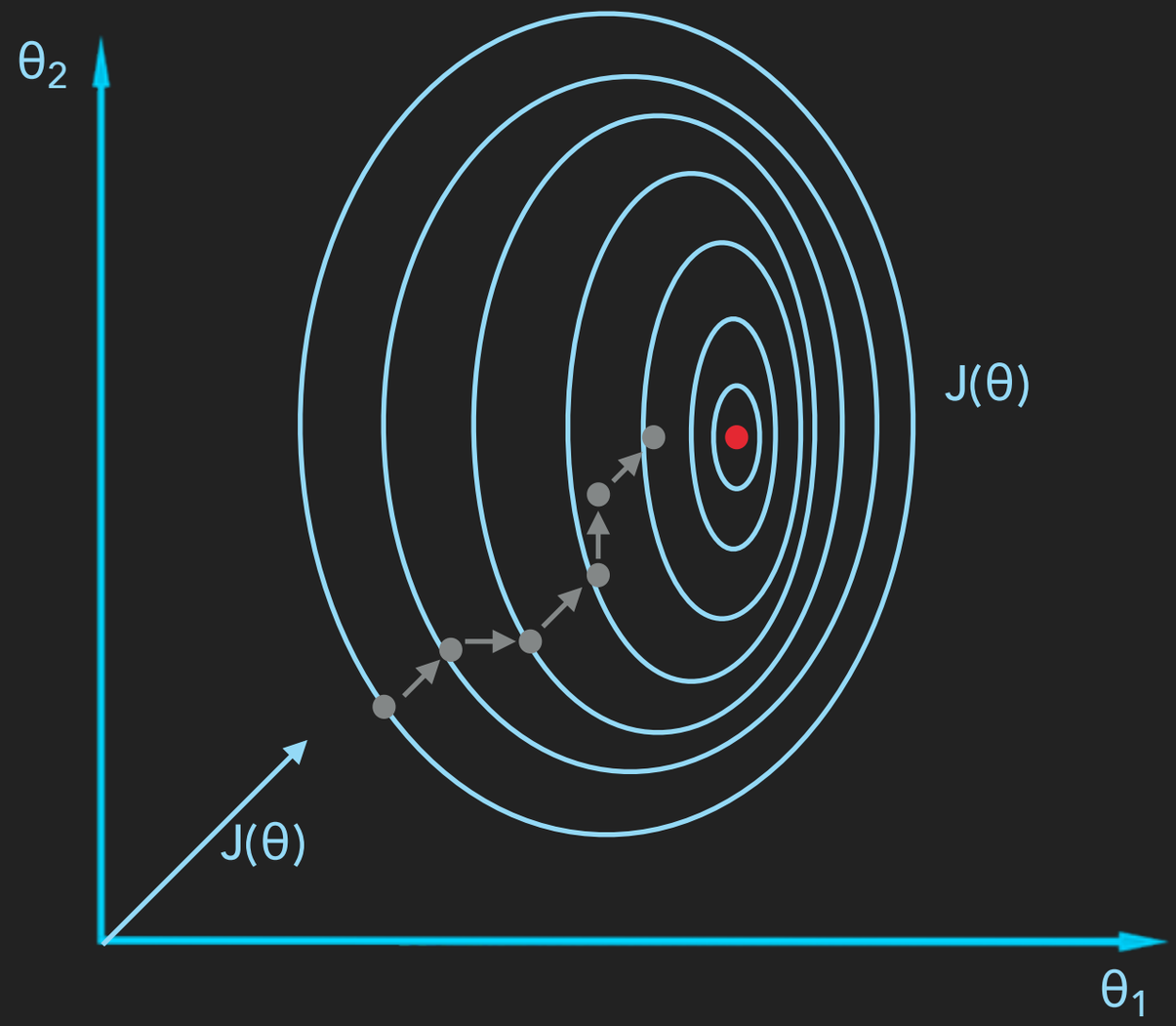
>MAN NEURAL_NETWORK



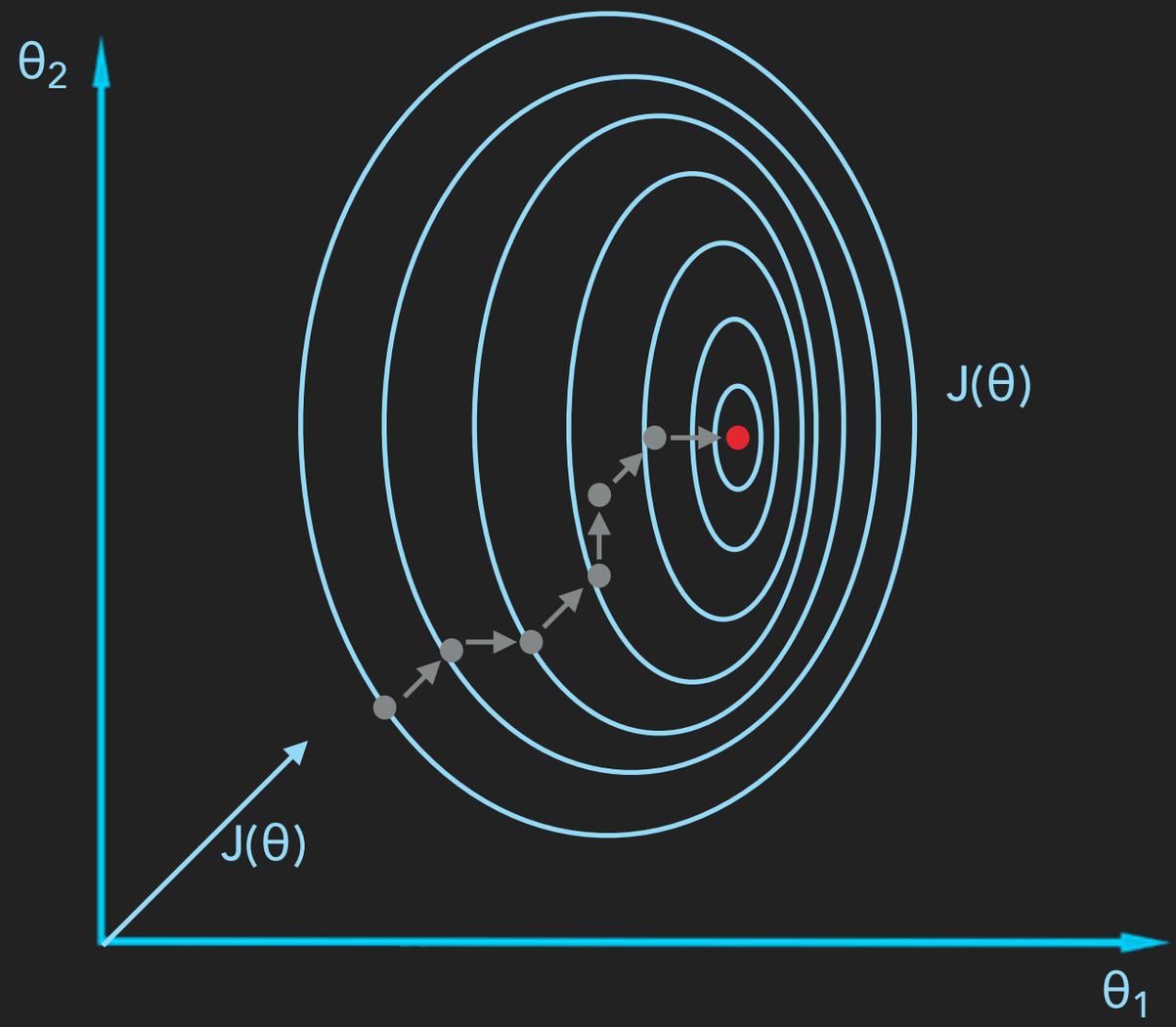
>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK



>MAN NEURAL_NETWORK



ENOUGH THEORY FOR NOW

LET'S IMPLEMENT SOMETHING!

>MAN SYNAPTIC

>MAN SYNAPTIC

- Architect

>MAN SYNAPTIC

- Architect
- Network

>MAN SYNAPTIC

- Architect
- Network
- Trainer

>MAN SYNAPTIC

- Architect
- Network
- Trainer
- Network.activate()

>CAT POSITRONIC-BRAIN

```
var historicMatch = {
  'input': [43.2, 54.2]
  'output' : [1, 0, 0]
};

var matchInTheFuture = {
  'input': [32.42, 634.4]
}

var trainingSet = [historicMatch];
var network = new Architect.Perceptron(historicMatch.input.length, 6, 6, 3);
var trainer = new Trainer(network);

trainer.train(trainingSet, {
    rate: .0003,
    iterations: 100000,
});

var toBePredicted = [matchInTheFuture];
toBePredicted.forEach(function (match) {
    var activations = match.input;
    console.log(activations, network.activate(activations));
});
```

>POSITRONIC-BRAIN

```
lian@Lians-MacBook-Pro ~/Privat/positronic-brain master ● node src/main.js
Training neural network...
[ 321.15, 58.4 ] [ 0.770202754848928, 0.1335316788844524, 0.1022953361060289 ]
[ 71.13, 64.15 ] [ 0.5561042289272905, 0.2215426757492098, 0.227661316237316 ]
[ 58.4, 60.83 ] [ 0.4369071411789803, 0.2477893091651866, 0.313759402211238 5 ]
[ 198.45, 213.65 ] [ 0.3670749232004952 3, 0.2613462956101283 4, 0.3632320018740296 5 ]
[ 182.1, 58.65 ] [ 0.7700340881192606, 0.1336101880111445 2, 0.1023794666097904 3 ]
[ 77.45, 321.15 ] [ 0.3670749207929893 7, 0.261346296076364 4, 0.363232003676395 9 ]
[ 64.15, 25.15 ] [ 0.556126634384865, 0.2215355088723663 7, 0.2277501664707293 2 ]
[ 70.4, 572.55 ] [ 0.03383318461518671, 0.07953362569525316, 0.902653731538891 ]
[ 34.35, 54.23 ] [ 0.367070557254018 2, 0.2613469366090175 3, 0.3632343279875261 4 ]
[ 75.3, 71.13 ] [ 0.5561034823660409, 0.2215428332126973 8, 0.227766689474732 8 ]
[ 172.15, 86.08 ] [ 0.556101688496558 9, 0.2215432115707275 4, 0.227768029910554 03 ]
```

>RANDOM_STRING_OUTPUT

AN APPROXIMATE ANSWER TO THE RIGHT PROBLEM IS WORTH A GOOD DEAL MORE THAN AN EXACT ANSWER TO AN APPROXIMATE PROBLEM.

John Tukey

OK, LOOKS GOOD SO FAR

BUT HOW WELL DOES IT PERFORM?

>TAIL -V ERRORS

SYNAPTIC DATA ERROR

```
14  trainer.train(trainingSet, {
15      rate: .0003,
16      iterations: 100000,
17      schedule: {
18        every: 10000,
19        do: function (data) {
20          console.log("error", data.error);
21        }
22      }
23  });
```

SYNAPTIC DATA ERROR

>TAIL -V ERRORS

SYNAPTIC DATA ERROR

```
lian@Lians-MacBook-Pro ~/Privat/positronic-brain  master ● node src/main.js
Training neural network...
error 0.19007911504127187
error 0.18802092021642586
error 0.18692413681619047
error 0.1854934102252014
error 0.18463050462306577
error 0.18365117232422246
error 0.1829250147744015
error 0.18259779005875912
error 0.18227652566379604
error 0.18187424069529254
```

>TAIL -V ERRORS

CLASSIFICATION ERROR

```
25        do: function (data) {
26          var errors = 0;
27          trainingSet.forEach(function (dataPoint) {
28            var expected = predictFromProbability(dataPoint.output);
29            var prediction = network.activate(dataPoint.input)
30            var predicted = predictFromProbability(prediction);
31
32            if (expected != predicted) {
33              errors++;
34            }
35          });
36          console.log(errors / trainingSet.length);
37        }
```

CLASSIFICATION ERROR

>TAIL -V ERRORS

CLASSIFICATION ERROR

```
lian@Lians-MacBook-Pro ~/Privat/positronic-brain  master ● node src/main.js
Training neural network...
errorRate: 0.4528301886792453
errorRate: 0.4397677793904209
errorRate: 0.444121915820029
errorRate: 0.4281567489114659
errorRate: 0.4426705370101596
errorRate: 0.432510885341074
errorRate: 0.433962264150943 4
errorRate: 0.4513788098693759
errorRate: 0.4426705370101596
errorRate: 0.43831640058055155
```

CROSS VALIDATION ERROR

CROSS VALIDATION ERROR

```
12  var dataSet = [historicMatch];
13  var trainingSet = dataSet.splice(0, threshold);
14  var crossValidationSet = dataSet;
```

CROSS VALIDATION ERROR

```
12  var dataSet = [historicMatch];
13  var trainingSet = dataSet.splice(0, threshold);
14  var crossValidationSet = dataSet;
```

```
25        do: function (data) {
26          var errors = 0;
27          crossValidationSet.forEach(function (dataPoint) {
28            var expected = predictFromProbability(dataPoint.output);
29            var prediction = network.activate(dataPoint.input)
30            var predicted = predictFromProbability(prediction);
31
32            if (expected != predicted) {
33              errors++;
34            }
35          });
36          console.log(errors / crossValidationSet.length);
37        }
```

CROSS VALIDATION ERROR

>TAIL -V ERRORS

CROSS VALIDATION ERROR

```
lian@Lians-MacBook-Pro ~/Privat/positronic-brain  master ● node src/main.js
Training neural network...
errorRate: 0.4724637681159420
errorRate: 0.4869565217391304
errorRate: 0.4927536231884058
errorRate: 0.5043478260869565
errorRate: 0.5072463768115942
errorRate: 0.5130434782608696
errorRate: 0.5101449275362319
errorRate: 0.5014492753623189
errorRate: 0.5188405797101450
errorRate: 0.5130434782608696
```

OK, BUT WHY?

WHAT DOES IT MEAN?

>TAIL -V ERRORS

>TAIL -V ERRORS

- h(x) = [0.3, 0.5, 0.2]

>TAIL -V ERRORS

- h(x) = [0.3, 0.5, 0.2]
- y = [0, 0, 1]

>TAIL -V ERRORS

- h(x) = [0.3, 0.5, 0.2]
- y = [0, 0, 1]

>TAIL -V ERRORS

- h(x) = [0.3, 0.5, 0.2]
- y = [0, 0, 1]

- Data Error

>TAIL -V ERRORS

- h(x) = [0.3, 0.5, 0.2]
- y = [0, 0, 1]

- Data Error
  - error = mean squared error

>TAIL -V ERRORS

- h(x) = [0.3, 0.5, 0.2]
- y = [0, 0, 1]

- Data Error
  - error = mean squared error
  - squared error = || [-0.3, -0.5, 0.8] || ^ 2 = ~0.98

>TAIL -V ERRORS

- h(x) = [0.3, 0.5, 0.2]
- y = [0, 0, 1]

- Data Error
  - error = mean squared error
  - squared error = || [-0.3, -0.5, 0.8] || ^ 2 = ~0.98
- Classification Error

>TAIL -V ERRORS

- h(x) = [0.3, 0.5, 0.2]
- y = [0, 0, 1]

- Data Error
  - error = mean squared error
  - squared error = || [-0.3, -0.5, 0.8] || ^ 2 = ~0.98
- Classification Error
  - error = 1

>TAIL -V ERRORS

- h(x) = [0.3, 0.5, 0.2]
- y = [0, 0, 1]

- Data Error
  - error = mean squared error
  - squared error = || [-0.3, -0.5, 0.8] || ^ 2 = ~0.98
- Classification Error
  - error = 1
- Cross Validation Error

>TAIL -V ERRORS

- h(x) = [0.3, 0.5, 0.2]
- y = [0, 0, 1]

- Data Error
  - error = mean squared error
  - squared error = || [-0.3, -0.5, 0.8] || ^ 2 = ~0.98
- Classification Error
  - error = 1
- Cross Validation Error
  - error =1

>TAIL -V ERRORS

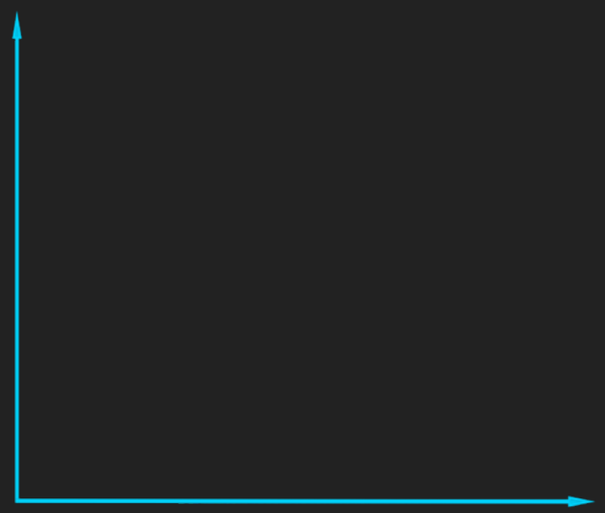
>TAIL -V ERRORS

- Cross Validation Error > Classification Error

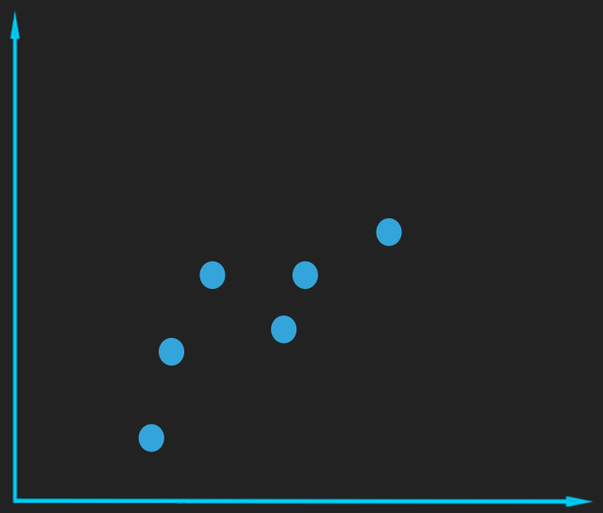
>TAIL -V ERRORS

- Cross Validation Error > Classification Error
  - Overfit / High Variance

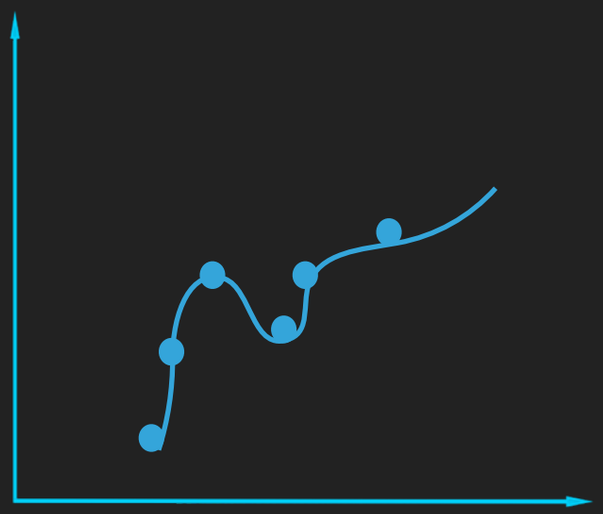
- Cross Validation Error > Classification Error
  - Overfit / High Variance



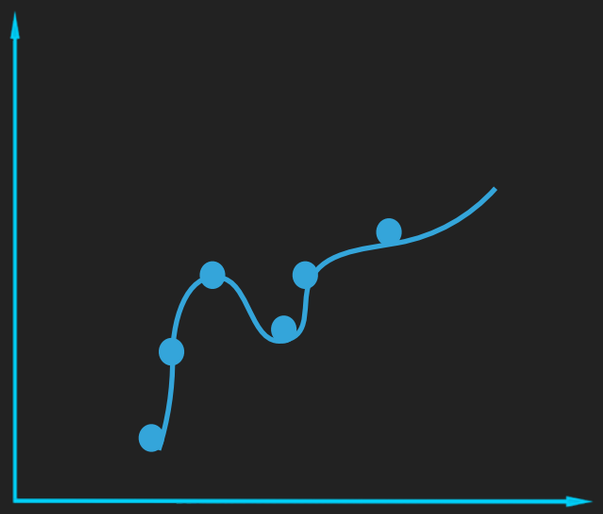
- Cross Validation Error > Classification Error
  - Overfit / High Variance



- Cross Validation Error > Classification Error
  - Overfit / High Variance

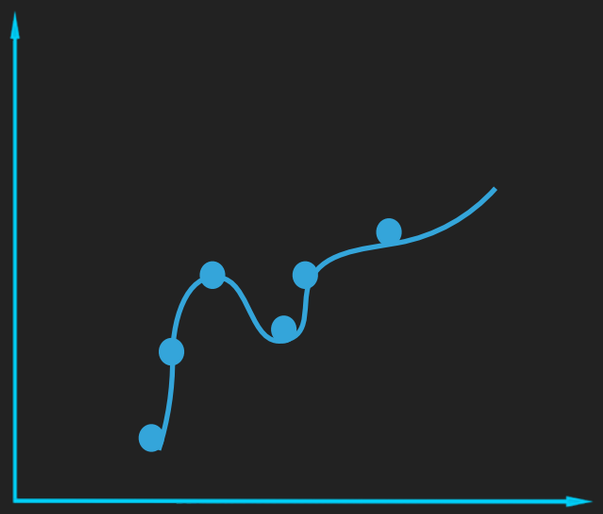


- Cross Validation Error > Classification Error
  - Overfit / High Variance

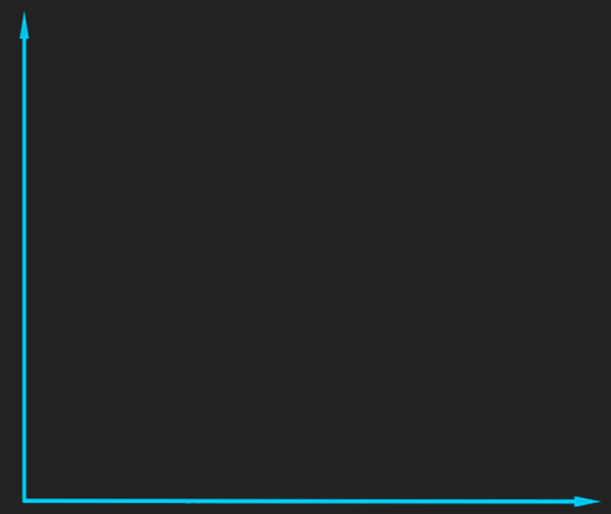


Overfit / High Variance

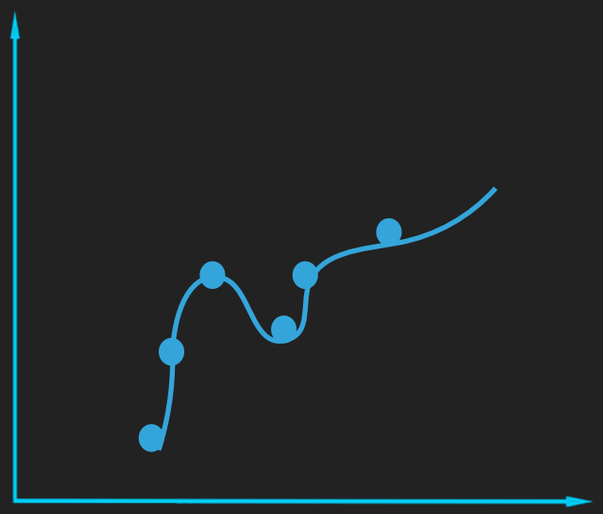
- Cross Validation Error > Classification Error
  - Overfit / High Variance



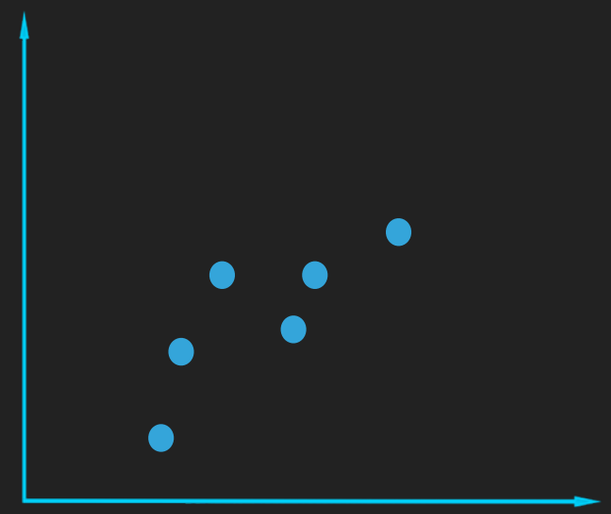
Overfit / High Variance



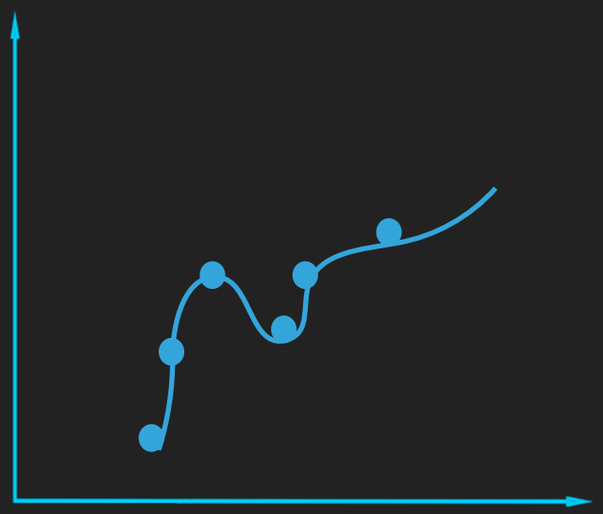
- Cross Validation Error > Classification Error
  - Overfit / High Variance



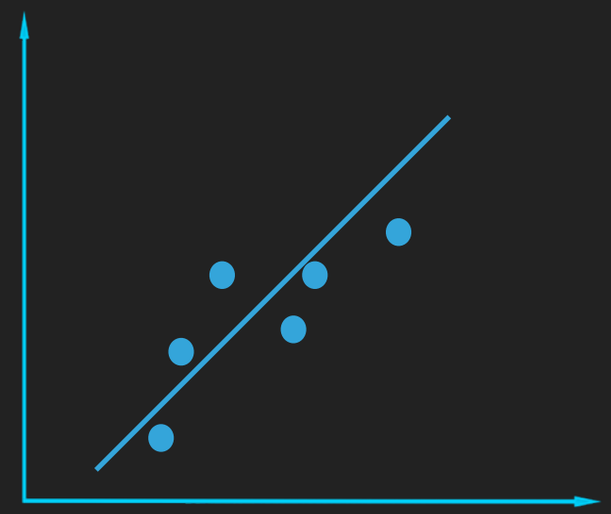
Overfit / High Variance



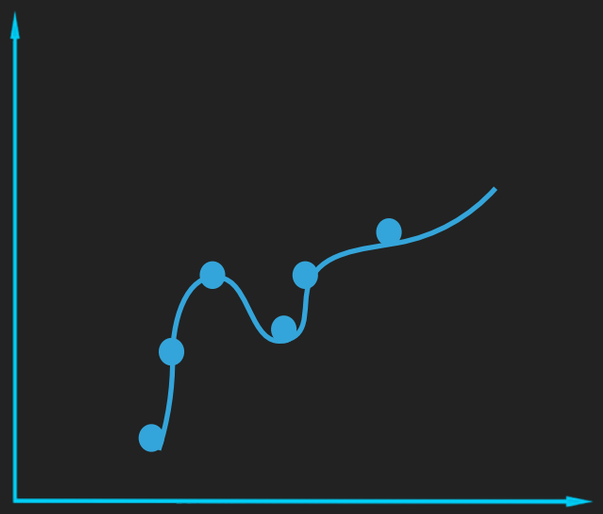
- Cross Validation Error > Classification Error
  - Overfit / High Variance



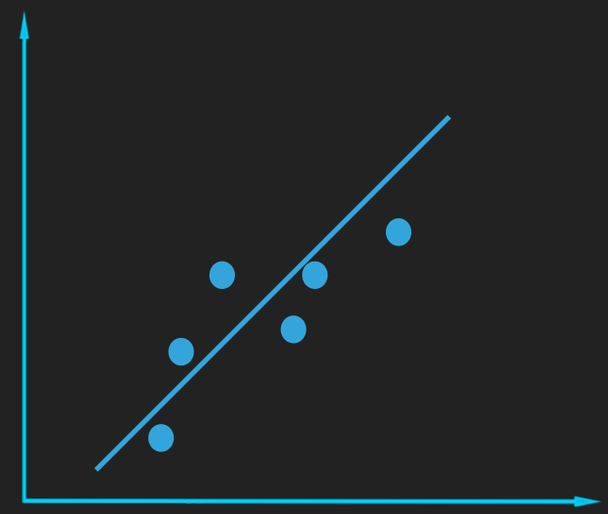
Overfit / High Variance



- Cross Validation Error > Classification Error
  - Overfit / High Variance



Overfit / High Variance



Underfit / High Bias

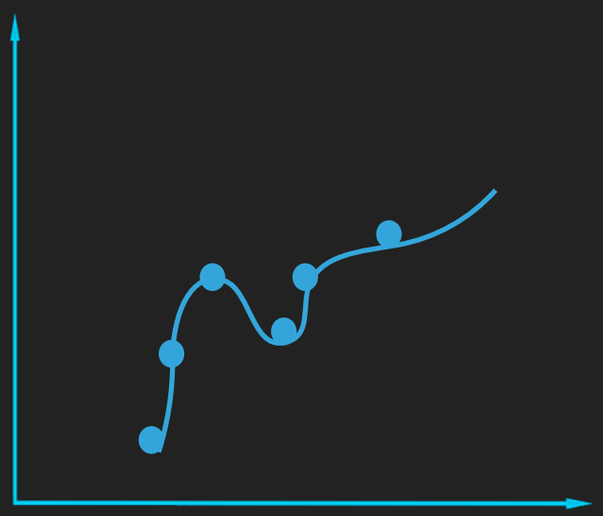
>TAIL -V ERRORS

- More Data / More Features

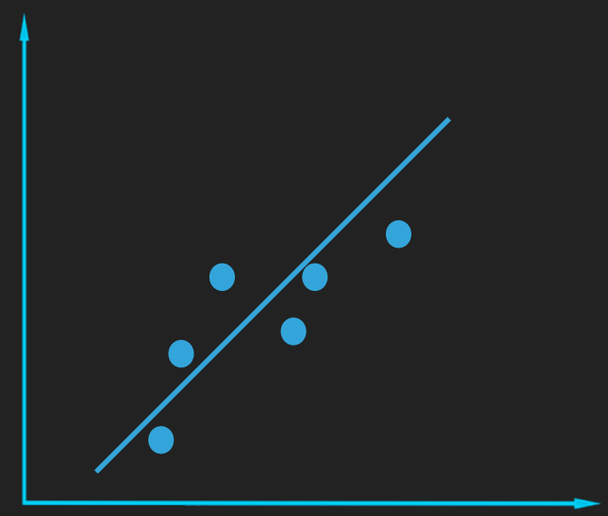
- More Data / More Features
- Regularization

- More Data / More Features
- Regularization
  - beneficial to keep thetas small/big

- More Data / More Features
- Regularization
  - beneficial to keep thetas small/big



Overfit / High Variance



Underfit / High Bias

>POSITRONIC-BRAIN

>POSITRONIC-BRAIN

```
lian@Lians-MacBook-Pro ~/Privat/positronic-brain master ● node src/main.js
Training neural network...
[ 94.03, 76.08 ]
data error: 0.18815276130336966 | classification error: 0.44668587896253603 | crossvalidation error: 0.521613832853026
data error: 0.1865592591273571 | classification error: 0.45244956772334294 | crossvalidation error: 0.5187319884726225
data error: 0.1863054092658534 | classification error: 0.4538904899135447 | crossvalidation error: 0.5273775216138329
data error: 0.184551243143785 | classification error: 0.45244956772334294 | crossvalidation error: 0.521613832853026
data error: 0.184617083088606 | classification error: 0.45677233429394815 | crossvalidation error: 0.5331412103746398
data error: 0.18648172017361 | classification error: 0.430835734870317 | crossvalidation error: 0.5244956772334294
data error: 0.1866602093383748 | classification error: 0.4438040345821326 | crossvalidation error: 0.521613832853026
data error: 0.18725134021059944 | classification error: 0.44812680115273773 | crossvalidation error: 0.5187319884726225
data error: 0.18683408543596147 | classification error: 0.44812680115273773 | crossvalidation error: 0.5187319884726225
data error: 0.18675113540427643 | classification error: 0.44812680115273773 | crossvalidation error: 0.5187319884726225
```

>POSITRONIC-BRAIN

```
lian@Lians-MacBook-Pro ~/Privat/positronic-brain master ● node src/main.js
Training neural network...
[ 94.03, 76.08 ]
data error: 0.18815276130336966 | classification error: 0.44668587896253603 | crossvalidation error: 0.521613832853026
data error: 0.1865592591273571 | classification error: 0.45244956772334294 | crossvalidation error: 0.5187319884726225
data error: 0.1863054092658534 | classification error: 0.4538904899135447 | crossvalidation error: 0.5273775216138329
data error: 0.184551243143785 | classification error: 0.45244956772334294 | crossvalidation error: 0.521613832853026
data error: 0.184617083088606 | classification error: 0.45677233429394815 | crossvalidation error: 0.5331412103746398
data error: 0.18648172017361 | classification error: 0.430835734870317 | crossvalidation error: 0.5244956772334294
data error: 0.1866602093383748 | classification error: 0.4438040345821326 | crossvalidation error: 0.521613832853026
data error: 0.18725134021059944 | classification error: 0.44812680115273773 | crossvalidation error: 0.5187319884726225
data error: 0.18683408543596147 | classification error: 0.44812680115273773 | crossvalidation error: 0.5187319884726225
data error: 0.18675113540427643 | classification error: 0.44812680115273773 | crossvalidation error: 0.5187319884726225
```

```
lian@Lians-MacBook-Pro ~/Privat/positronic-brain master ● node src/main.js
Training neural network...
[ 11, 174.45, 80.1, 2, 14 ]
data error: 0.190937088058995 73 | classification error: 0.4610951008645533 | crossvalidation error: 0.4812680115273775
data error: 0.18988073974106973 | classification error: 0.4538904899135447 | crossvalidation error: 0.49279538904899134
data error: 0.187359737681286 53 | classification error: 0.4510086455331412 | crossvalidation error: 0.4755043227665706
data error: 0.18626499835311 82 | classification error: 0.4438040345821326 | crossvalidation error: 0.4755043227665706
data error: 0.185649362616 39 | classification error: 0.44668587896253603 | crossvalidation error: 0.4956772334293948
data error: 0.18478098795166 44 | classification error: 0.44668587896253603 | crossvalidation error: 0.49279538904899134
data error: 0.18386134162726 75 | classification error: 0.4409221902017291 | crossvalidation error: 0.48703170028818443
data error: 0.18344983410551 08 | classification error: 0.4409221902017291 | crossvalidation error: 0.48703170028818443
data error: 0.184293393158302 95 | classification error: 0.435158501440922 16 | crossvalidation error: 0.5014409221902018
data error: 0.184495975273041 04 | classification error: 0.43948126801152737 | crossvalidation error: 0.521613832853026
```

>RANDOM_STRING_OUTPUT

PREDICTION IS VERY DIFFICULT, ESPECIALLY ABOUT THE FUTURE.

Niels Bohr

NOW YOU KNOW...

NOW YOU KNOW...

- how Neural Networks do their magic

NOW YOU KNOW...

- how Neural Networks do their magic
- how to implement a machine learning algorithm with synaptic

NOW YOU KNOW...

- how Neural Networks do their magic
- how to implement a machine learning algorithm with synaptic
- how to check the performance of your algorithm

NOW YOU KNOW...

- how Neural Networks do their magic
- how to implement a machine learning algorithm with synaptic
- how to check the performance of your algorithm
- how to interpret error rates

>PREDICT MATCHDAY31

- Hamburger SV - Werder Bremen: **Home** 42% => 2:1
- VfL Wolfsburg - 1. FC Augsburg: **Home** 71%
- VfB Stuttgart - Borussia Dortmund: **Away** 69%
- 1. FC Köln - SV Darmstadt: **Home** 40%
- Hertha BSC - FC Bayern München: **Away** 69%
- FC Ingolstadt - Hannover 96: **Home** 42%
- FC Schalke - Bayer Leverkusen: **Home** 60%
- Borussia Mönchengladbach - TSG Hoffenheim: **Home** 71%
- Eintracht Frankfurt - 1. FSV Mainz: **Away** 41%

>CONTINUE

RECOMMENDED READING

- github.com/Chimney42/positronic-brain
- https://slidr.io/Chimney42/machine-learning-with-synaptic#1
- https://www.codingame.com/games/machine-learning
- https://github.com/cazala/synaptic/wiki/Neural-Networks-101
- http://synaptic.juancazala.com/#/
- natureofcode.com/book/chapter-10-neural-networks/

>EXIT 0

THANKS TO

- github.com/PiMastah
- transfermarkt.de
- Jimdo
- all of you for your interest!