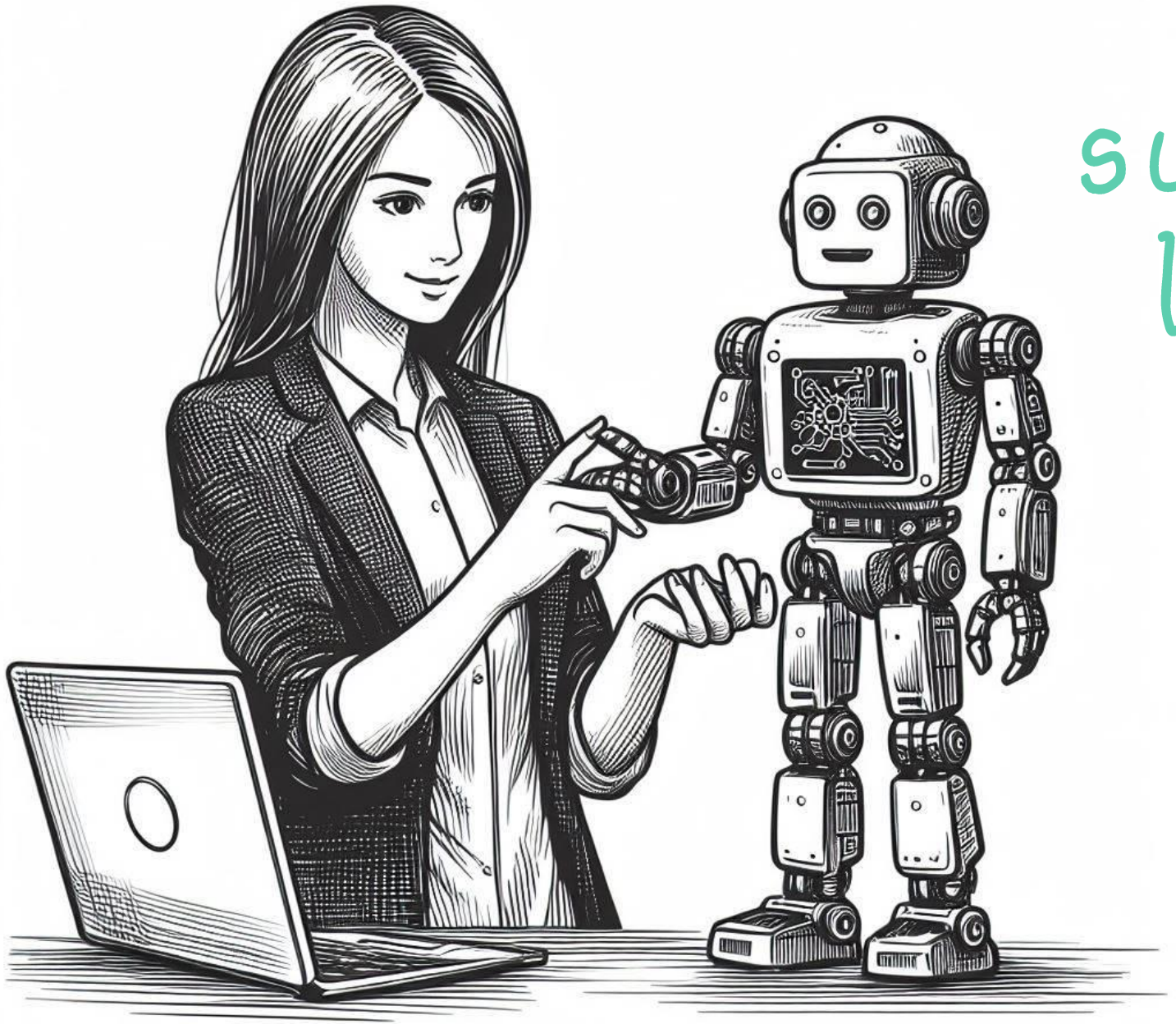


supervised learning

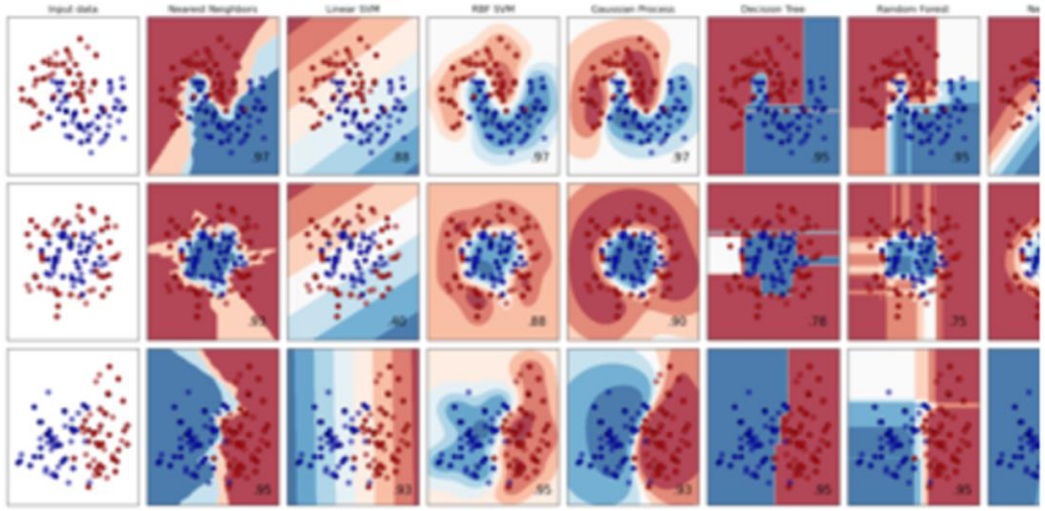
ADULT
SUPERVISION
REQUIRED

any case where you know some truth

Classification

Identifying which category an object belongs to.

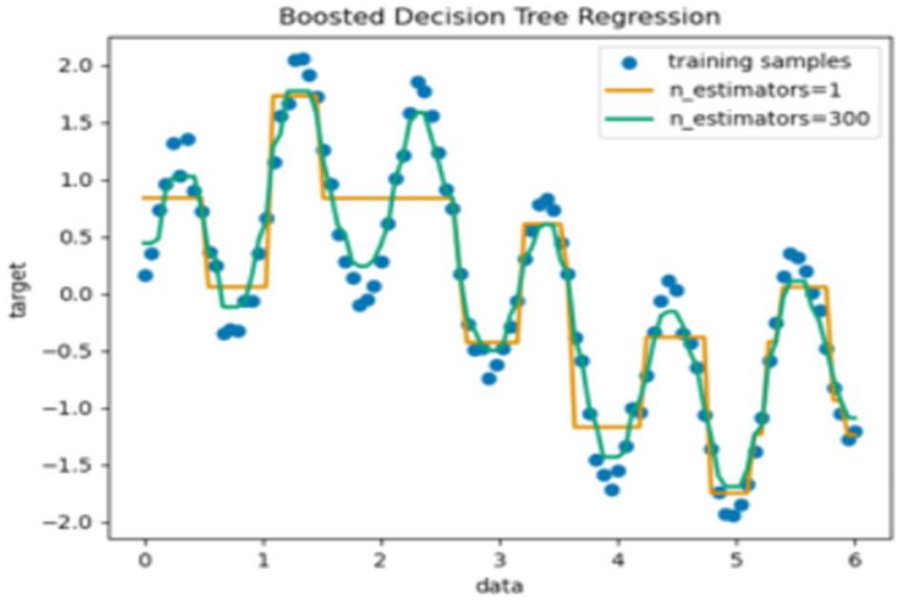
Applications: Spam detection, image recognition.
Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, and more...



Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: Gradient boosting, nearest neighbors, random forest, ridge, and more...



carefully curated

your data

images, spectra, measurements, whole simulations, anything that you measured and trust dearly

labels

galaxy images
(or measurements of)
mass, type, redshift

sims with cosmologies X_i and X_{i+1}
sim with cosmology $X_{i+\frac{1}{2}}$

lensed objects
reconstruction

pairs of similar images
similarity score

carefully curated

your data

images, spectra, measurements, whole simulations, anything that you measured and trust dearly

labels

galaxy images
(or measurements of)
mass, type, redshift

sims with cosmologies X_i and X_{i+1}
sim with cosmology $X_{i+\frac{1}{2}}$

lensed objects
reconstruction

pairs of similar images
similarity score

carefully curated

your data

images, spectra, measurements, whole simulations, anything that you measured and trust dearly

labels

well-understood

new data

very similar data you your curated dataset, except you don't have the labels!

carefully curated

your data

images, spectra, measurements, whole simulations, anything that you measured and trust dearly

labels

machine learning

well-understood

new data

very similar data you your curated dataset, except you don't have the labels!

predicted labels

supervised learning

regression

continuous variable

more difficult
(more room for error)

traces spectrum/evolution

like traditional science

classification

discrete variable

easier to optimize

finds distinct categories/cases

similar to how humans operate?

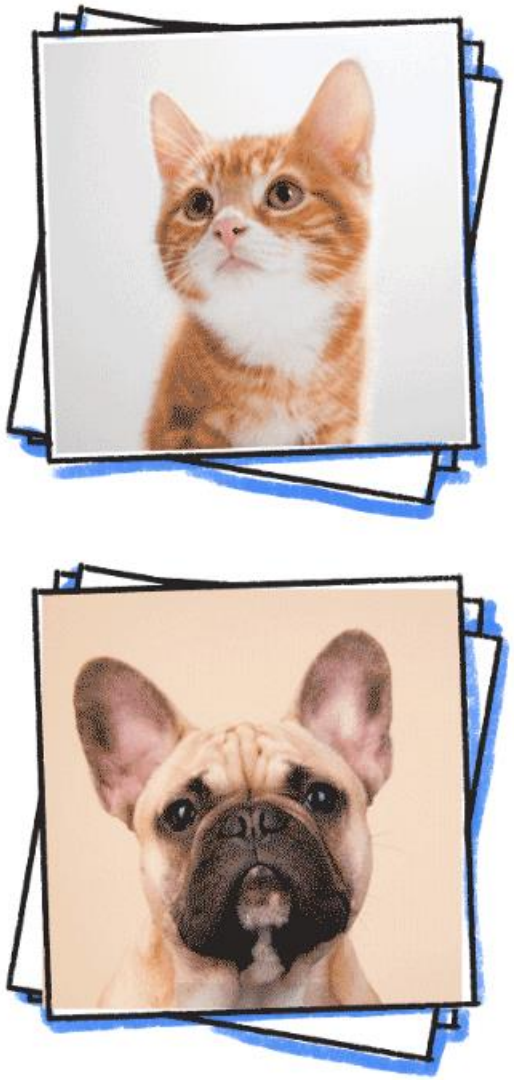
Euclid preparation. Measuring detailed galaxy morphologies for *Euclid* with Machine Learning

Euclid Collaboration: B. Aussel[⋆1], S. Kruk[2], M. Walmsley[3], M. Huertas-Company[4,5,6,7], M. Castellano[8], C. J. Conselice[3], M. Delli Veneri[9], H. Domínguez Sánchez[10], P.-A. Duc[11], U. Kuchner[12], A. La Marca[13,14], B. Margalef-Bentabol[13], F. R. Marleau[15], G. Stevens[16], Y. Toba[17], C. Tortora[18], L. Wang[13,14], N. Aghanim[19], B. Altieri[2], A. Amara[20], S. Andreon[21], N. Auricchio[22], M. Baldi[23,22,24], S. Bardelli[22], R. Bender[25,26], C. Bodendorf[25], D. Bonino[27], E. Branchini[28,29,21], M. Brescia[30,18,9], J. Brinchmann[31], S. Camera[32,33,27], V. Capobianco[27], C. Carbone[34], J. Carretero[35,36], S. Casas[37], S. Cavuoti[18,9], A. Cimatti[38], G. Congedo[39], L. Conversi[40,2], Y. Copin[41], F. Courbin[42], H. M. Courtois[43], M. Cropper[44], A. Da Silva[45,46], H. Degaudenzi[47], A. M. Di Giorgio[48], J. Dinis[46,45], F. Dubath[47], X. Dupac[2], S. Dusini[49], M. Farina[48], S. Farrens[50], S. Ferriol[41], S. Fotopoulou[51], M. Frailis[52], E. Franceschi[22], P. Franzetti[34], M. Fumana[34], S. Galeotta[52], B. Garilli[34], B. Gillis[39], C. Giocoli[22,53], A. Grazian[54], F. Grupp[25,55], S. V. H. Haugan[56], W. Holmes[57], I. Hook[58], F. Hormuth[59], A. Hornstrup[60,61], P. Hudelot[62], K. Jahnke[63],

CAT

(LABELED PHOTOS)

DOG

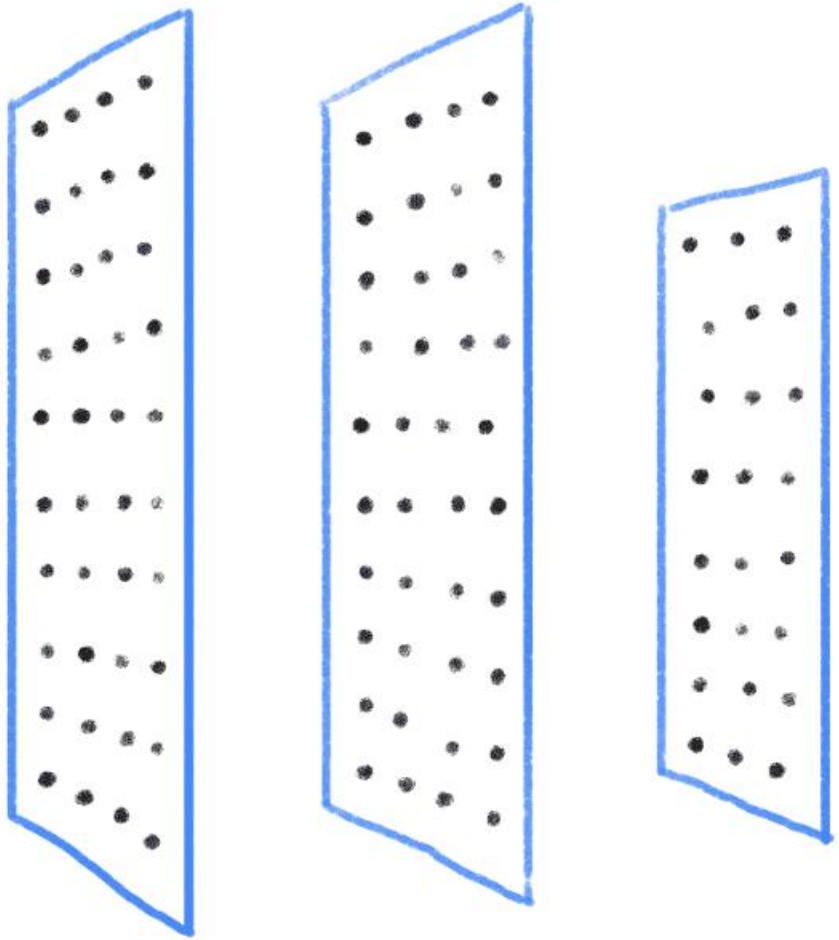


OUTPUT

CAT

(LABELED PHOTOS)

DOG



output $\in (0,1)$

OUTPUT

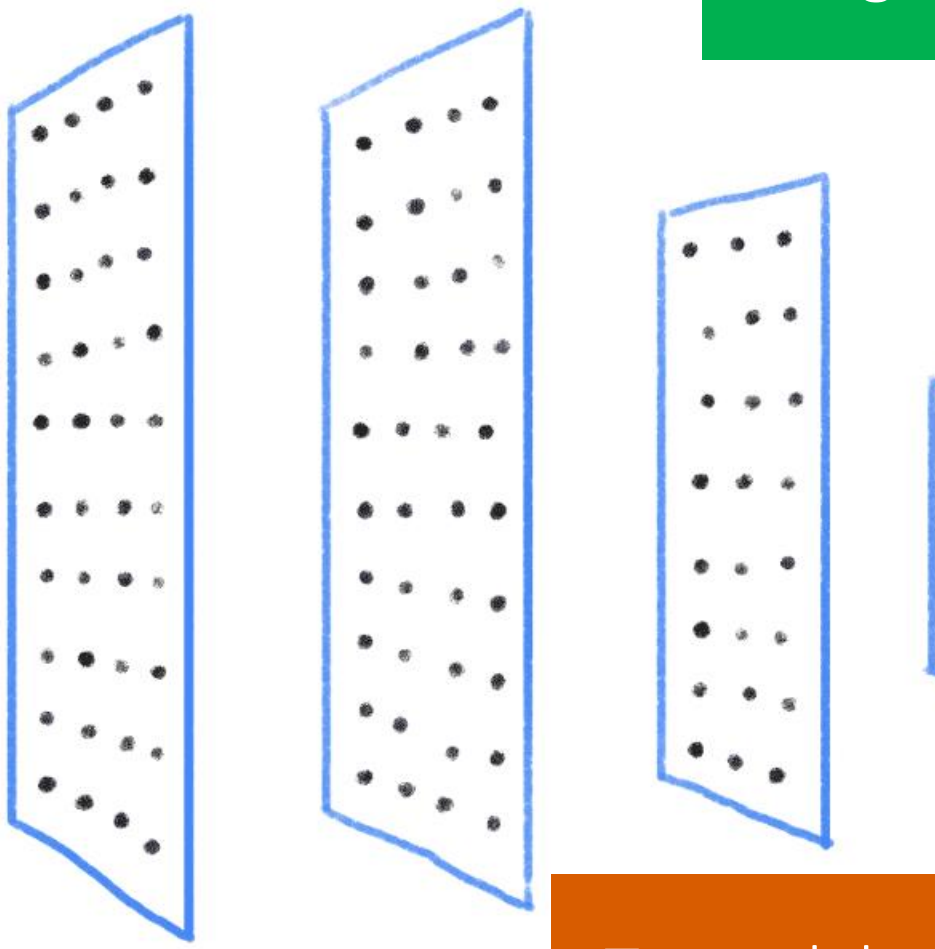
classification

Target label: 0 or 1

CAT

(LABELED PHOTOS)

DOG

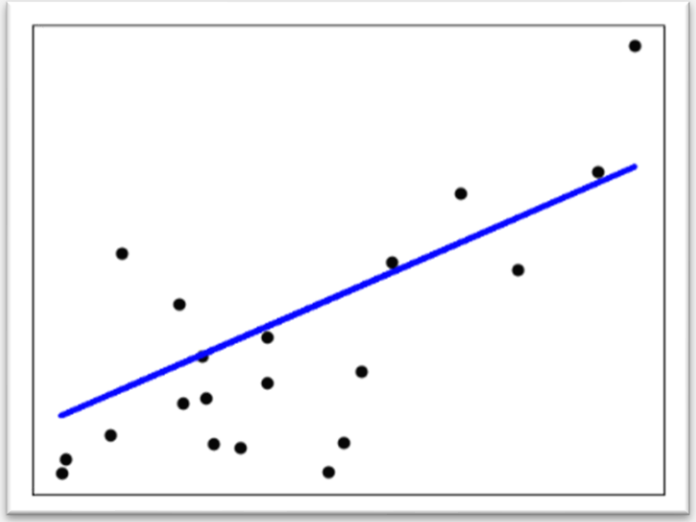


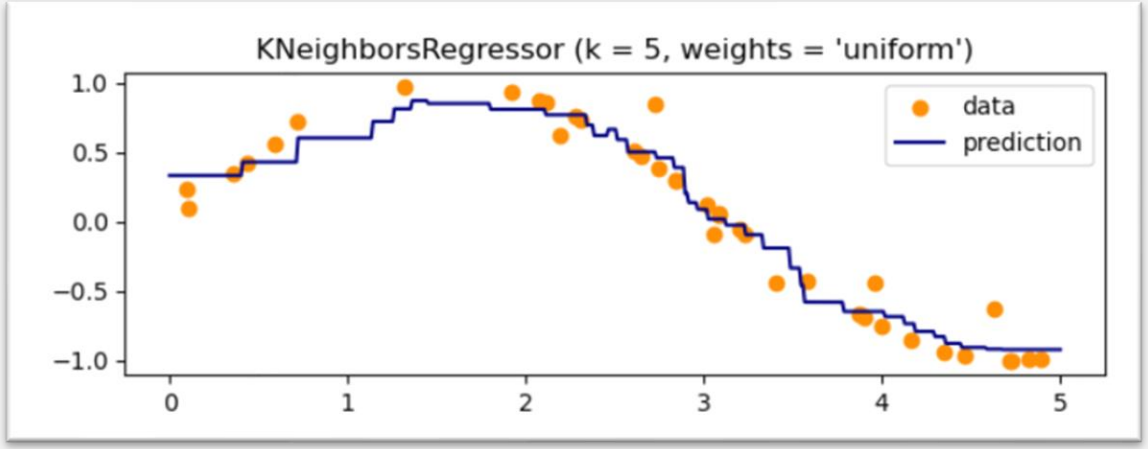
OUTPUT

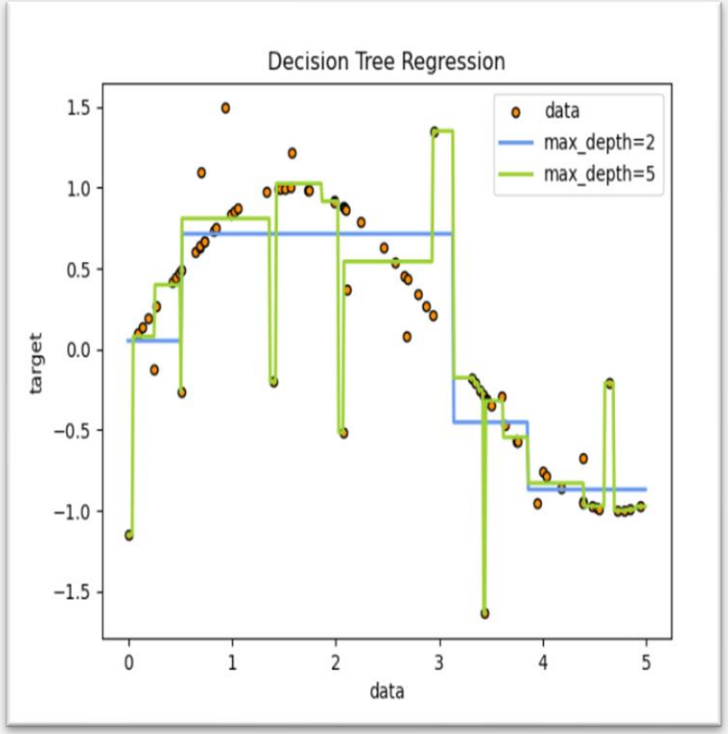
Target label: p(cat)∈ (0,1)

regression

regression

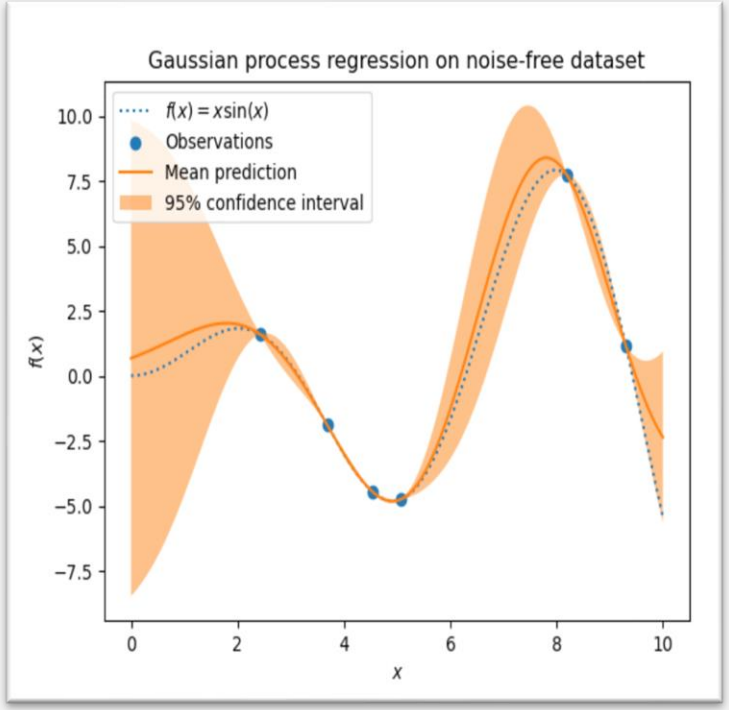




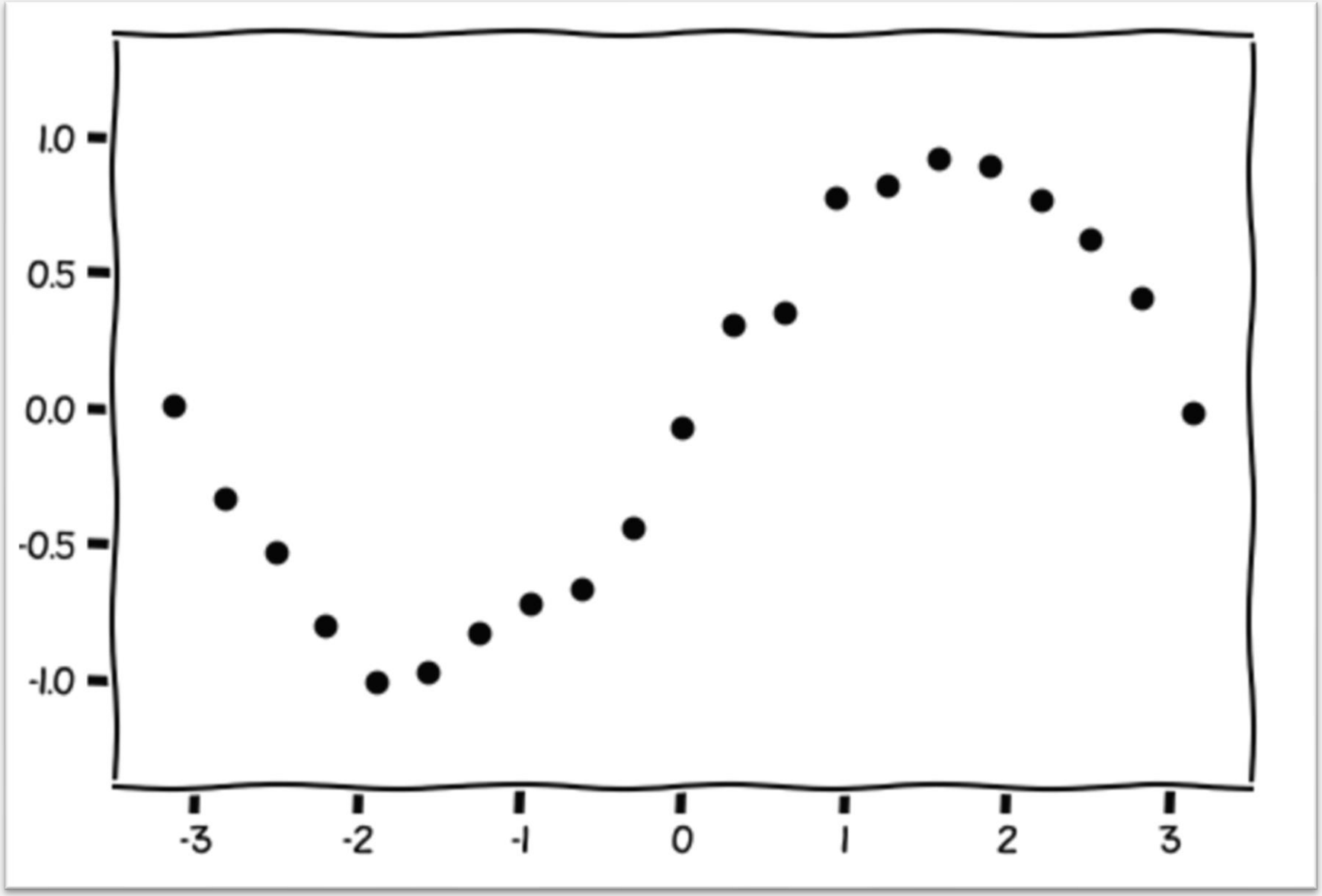


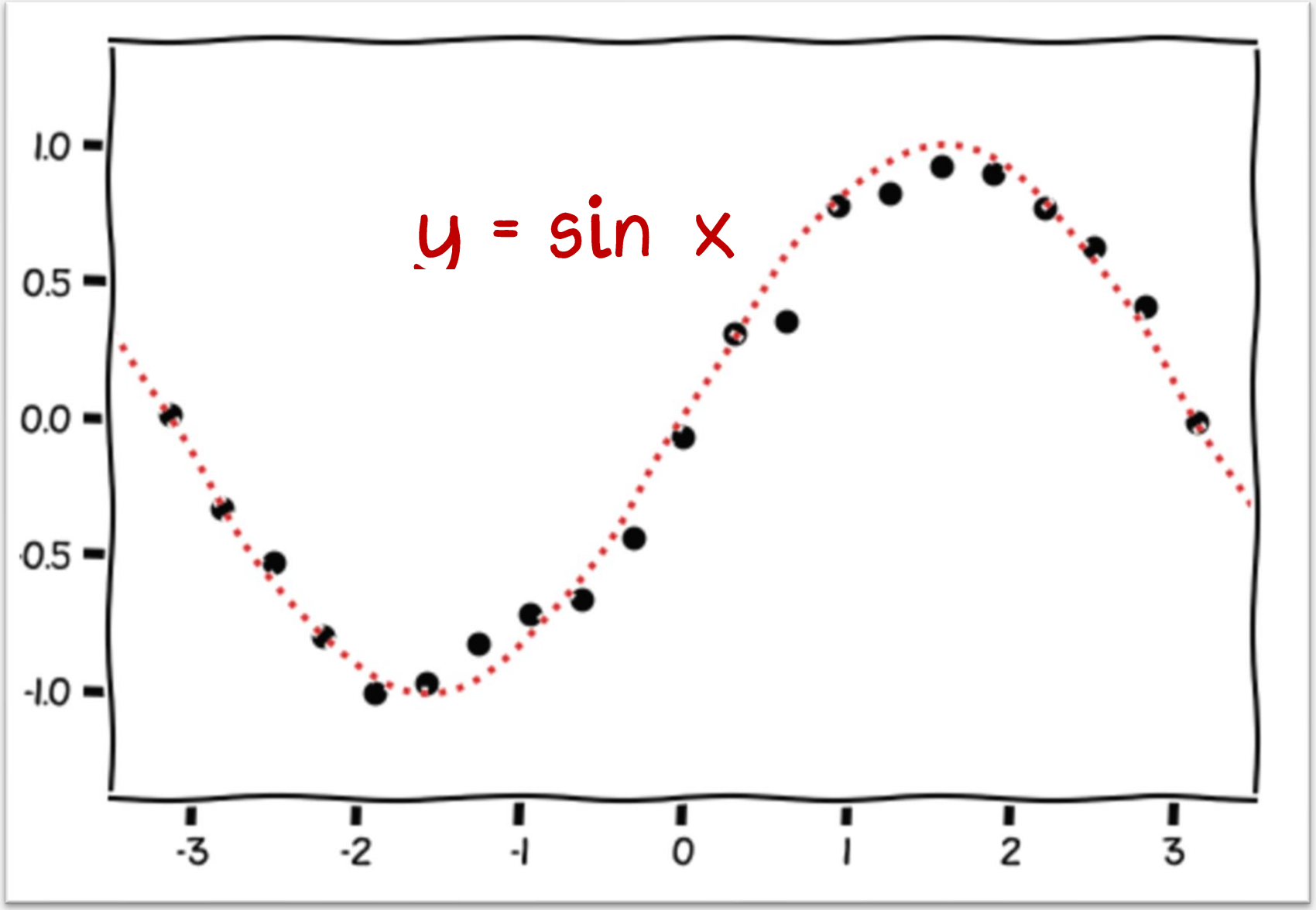
regression

given $\vec{x}$ predict $\vec{y} = f(\vec{x})$



linear regression


1.0
0.5
0.0
-0.5
-1.0
-3
-2
-1
0
1
2
3


y = sin x
1.0
0.5
0.0
-0.5
-1.0
-3
-2
-1
0
1
2
3

linear

regression

$$\vec{y} = \alpha_1 f_1(\vec{x}) + \alpha_2 f_2(\vec{x}) + \cdots + \alpha_n f_n(\vec{x})$$

linear regression

$$\vec{y} = \alpha_1 f_1(\vec{x}) + \alpha_2 f_2(\vec{x}) + \cdots + \alpha_n f_n(\vec{x})$$

you can do whatever you want to your dependent variables, but the coefficients must be linear

$$y = a\,x_1^b$$

$$y = e^{a+bx}$$

$$y = e^a x$$

$$y = ax + b$$

$$ax^2 + by^2 = 1$$

$$y = (ax_1 + bx_2)^2$$

$$y = a \sin x_1^2 + \frac{b}{x_2}$$

$$y = ax + a^2x^2$$

$$y = \sin ax$$

- **Linearity**. This means that the mean of the response variable is a linear combination of the parameters (regression coefficients) and the predictor variables. Note that this assumption is much less restrictive than it may at first seem. Because the predictor variables are treated as fixed values (see above), linearity is really only a restriction on the parameters. The predictor variables themselves can be arbitrarily transformed, and in fact multiple copies of the same underlying predictor variable can be added, each one transformed differently. This technique is used, for example, in polynomial regression, which uses linear regression to fit the response variable as an arbitrary polynomial function (up to a given degree) of a predictor variable. With this much flexibility, models such as polynomial regression often have "too much power", in that they tend to overfit the data. As a result, some kind of

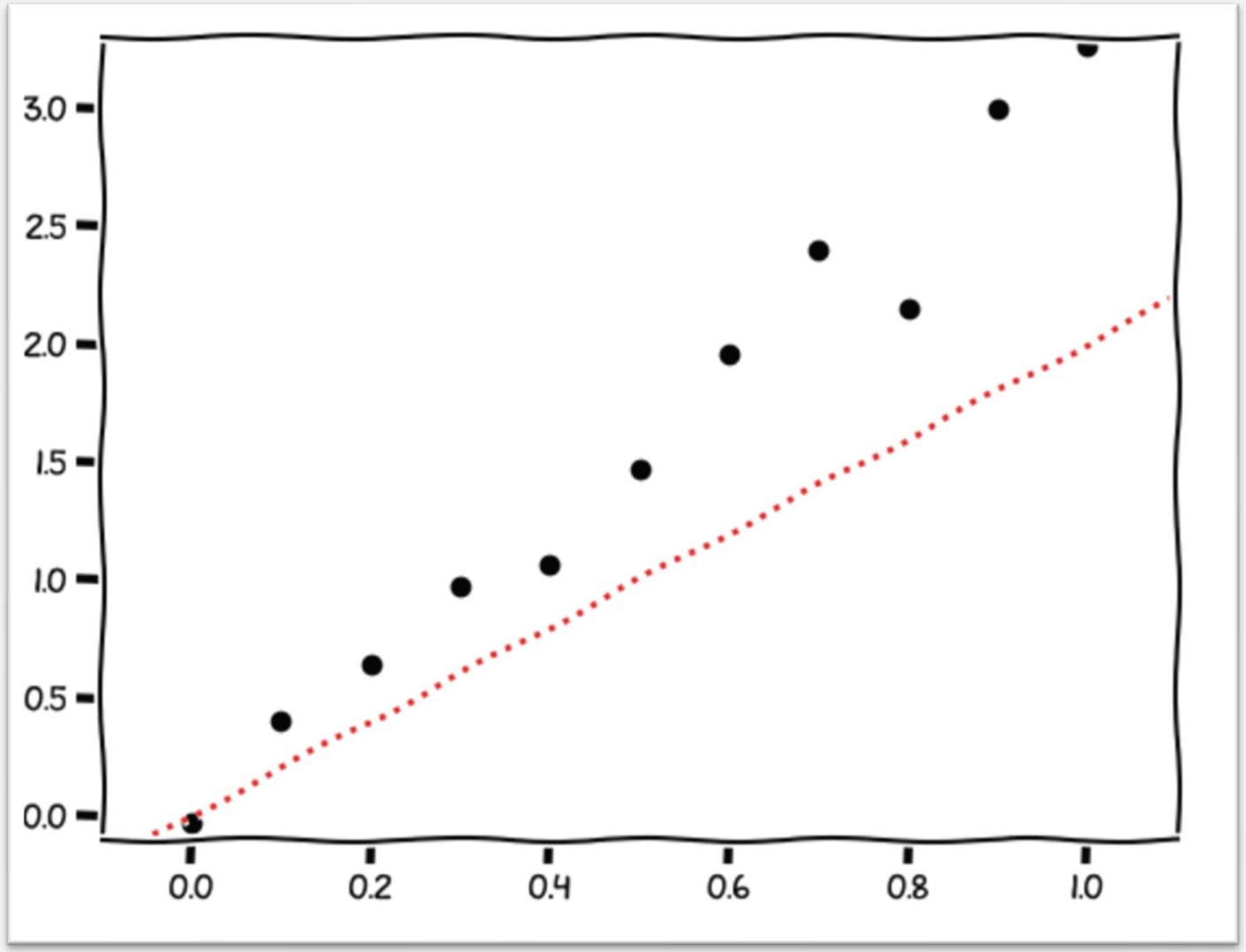
linear regression

fitting a line $y = ax$

1. Try some guess a_0
2. Calculate loss

   RMSE = 0.5

```
from sklearn.metrics import mean_squared_error
```



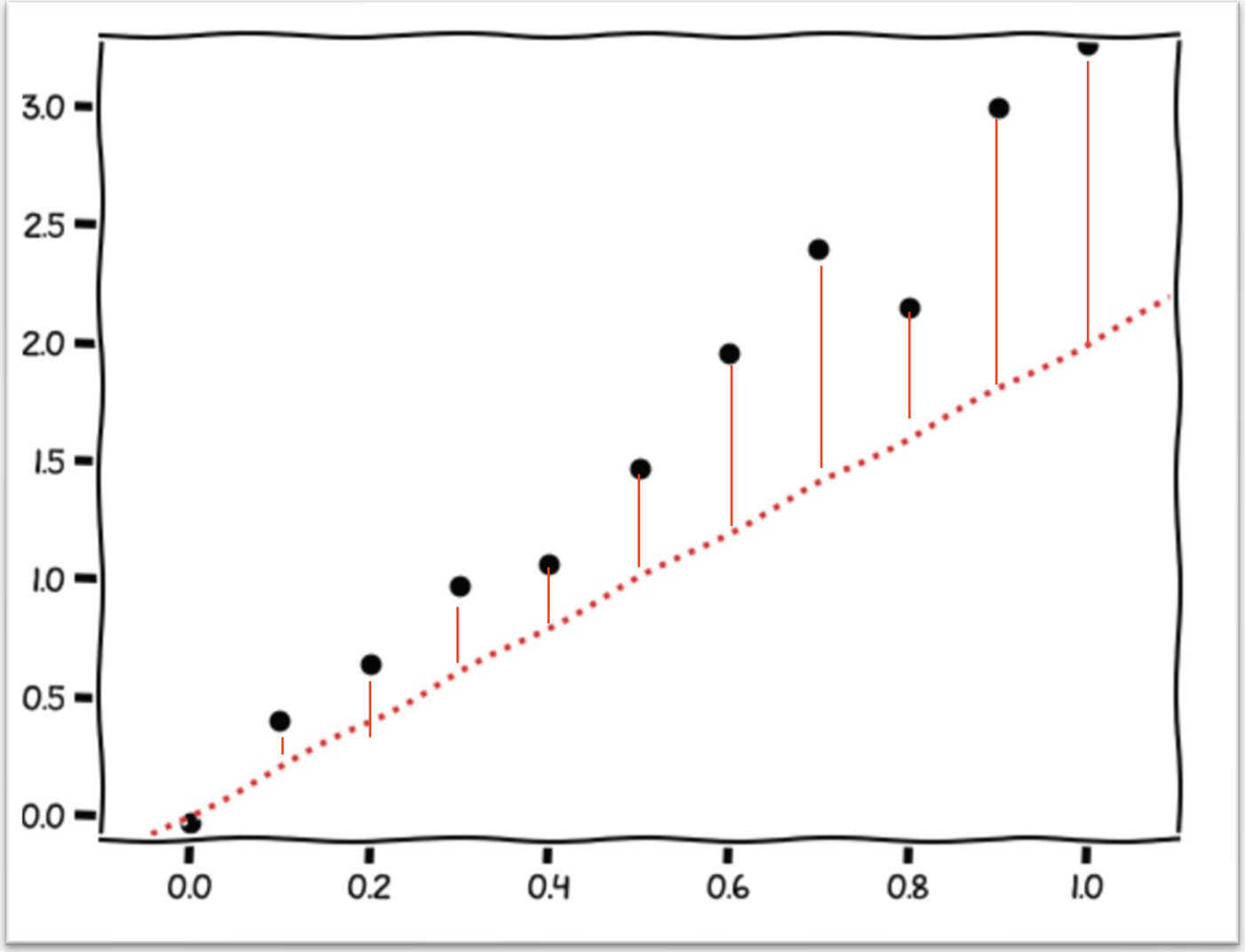
linear regression

fitting a line $y = ax$

1. Try some guess a_0
2. Calculate loss

   RMSE = 0.5

```
from sklearn.metrics import mean_squared_error
```



linear regression

fitting a line $y = ax$

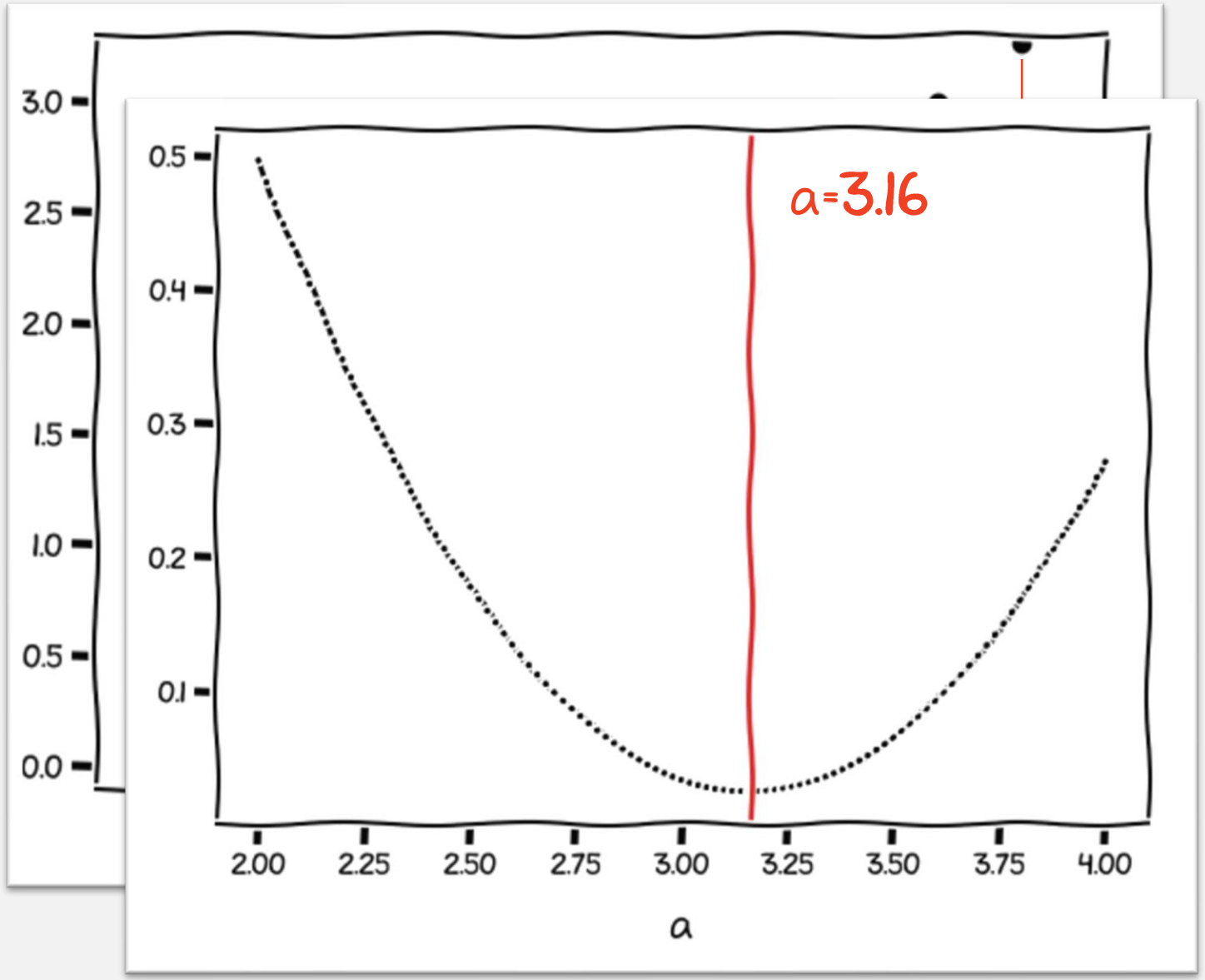
1. Try some guess a_0
2. Calculate loss

   RMSE = 0.5

```
from sklearn.metrics import mean_squared_error
```

3. Find the minimum of the loss function

   E.g., through gradient descent



linear regression

fitting a line $y = ax$

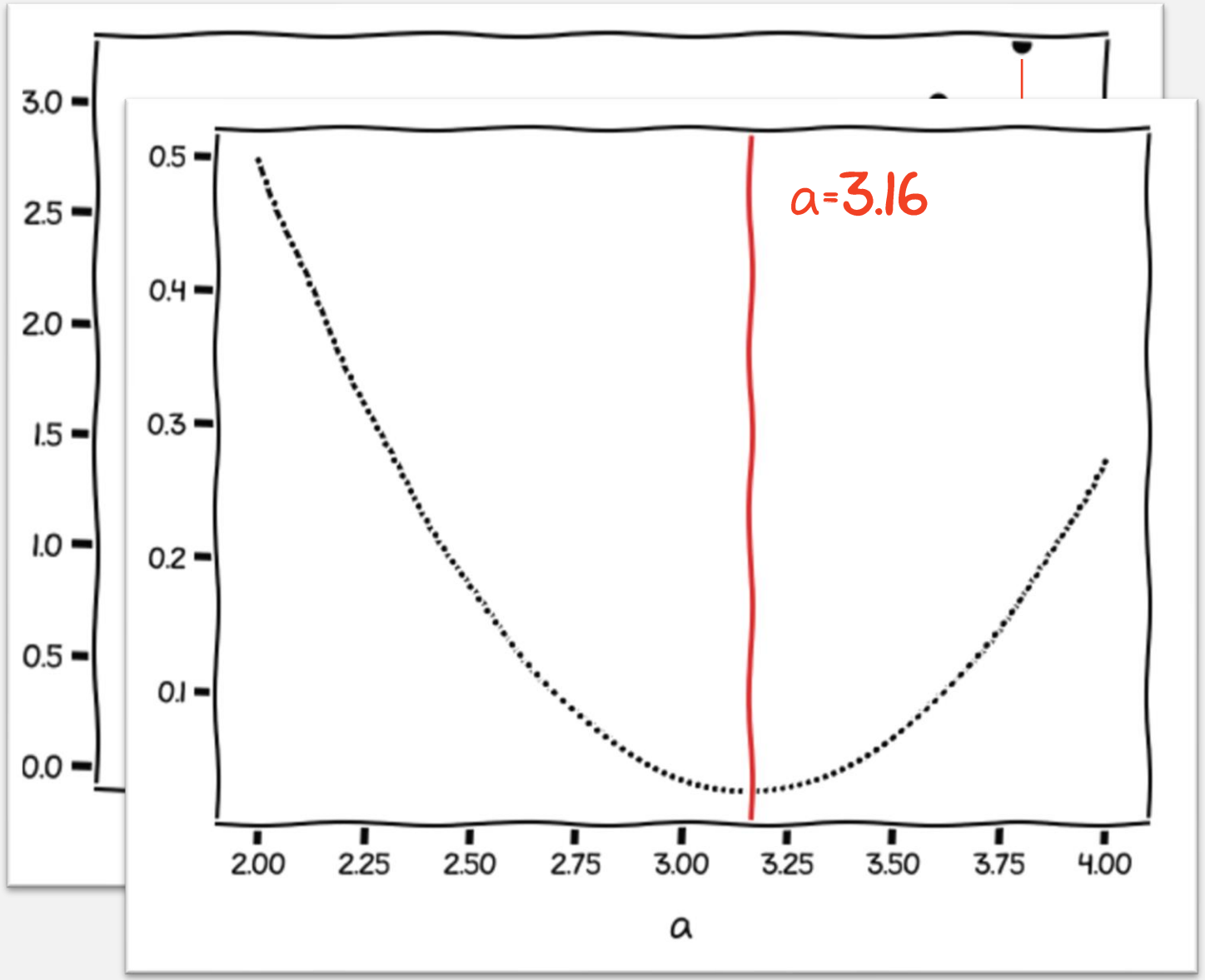
1. Try some guess a_0
2. Calculate loss

   RMSE = 0.5

```
from sklearn.metrics import mean_squared_error
```

3. Find the minimum of the loss function

   E.g., through gradient descent



linear regression

fitting a line $y = ax$

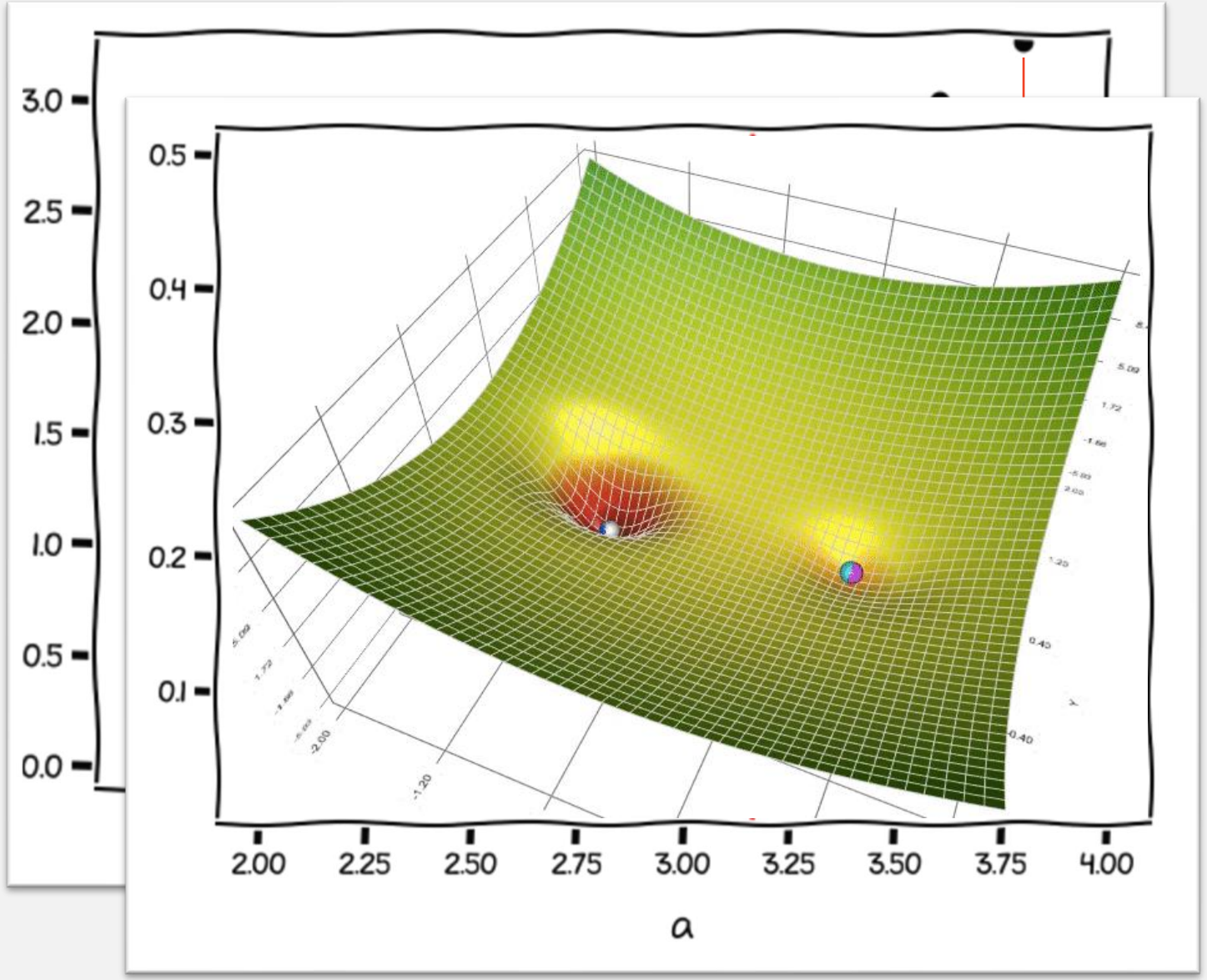
1. Try some guess a_0
2. Calculate loss

   RMSE = 0.5

```
from sklearn.metrics import mean_squared_error
```

3. Find the minimum of the loss function

   E.g., through gradient descent



linear regression

fitting a line $y = ax$

1. Try some guess a_0
2. Calculate loss

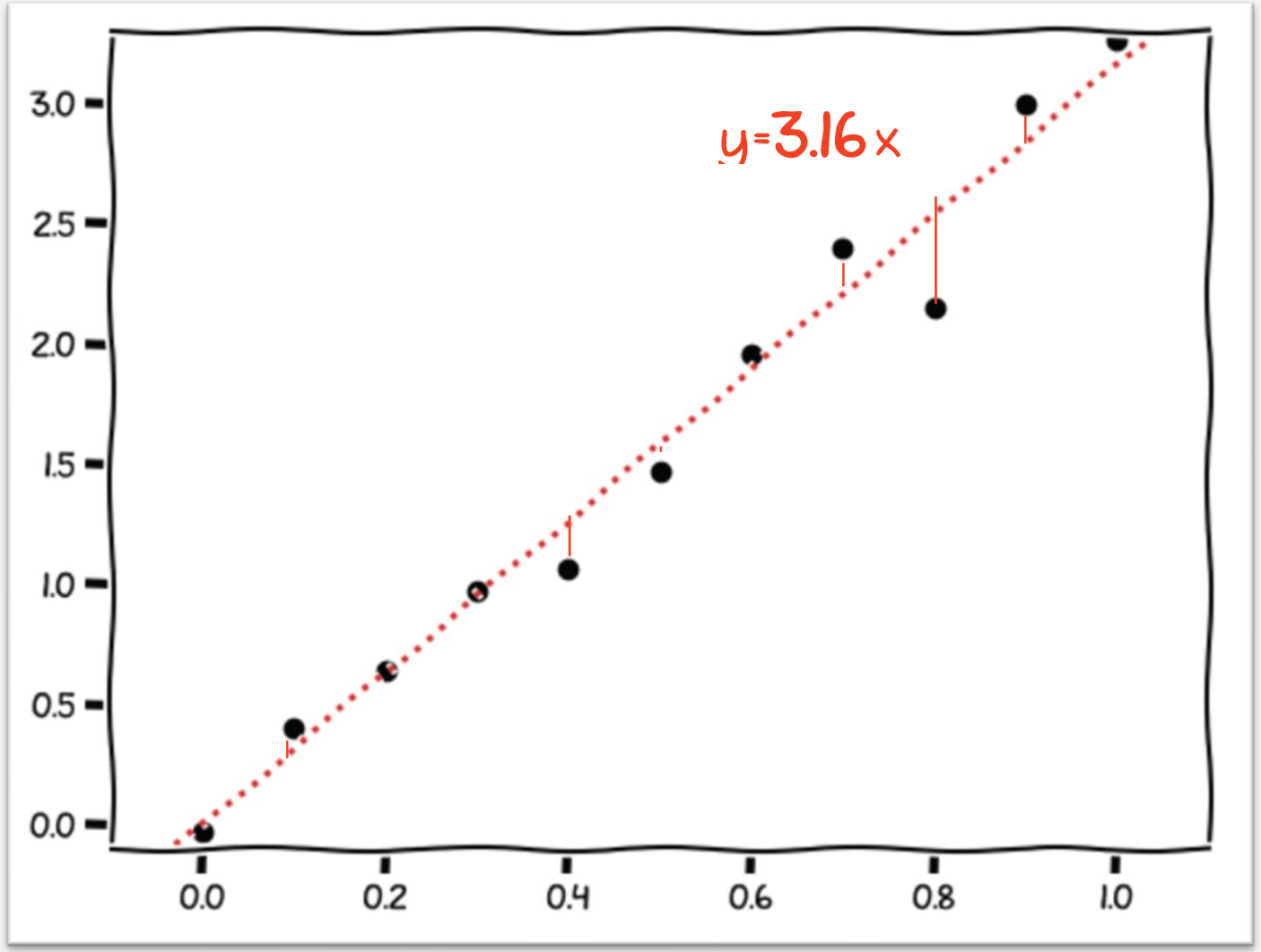
   RMSE = 0.5

```
from sklearn.metrics import mean_squared_error
```

3. Find the minimum of the loss function

   E.g., through gradient descent

4. Celebrate!



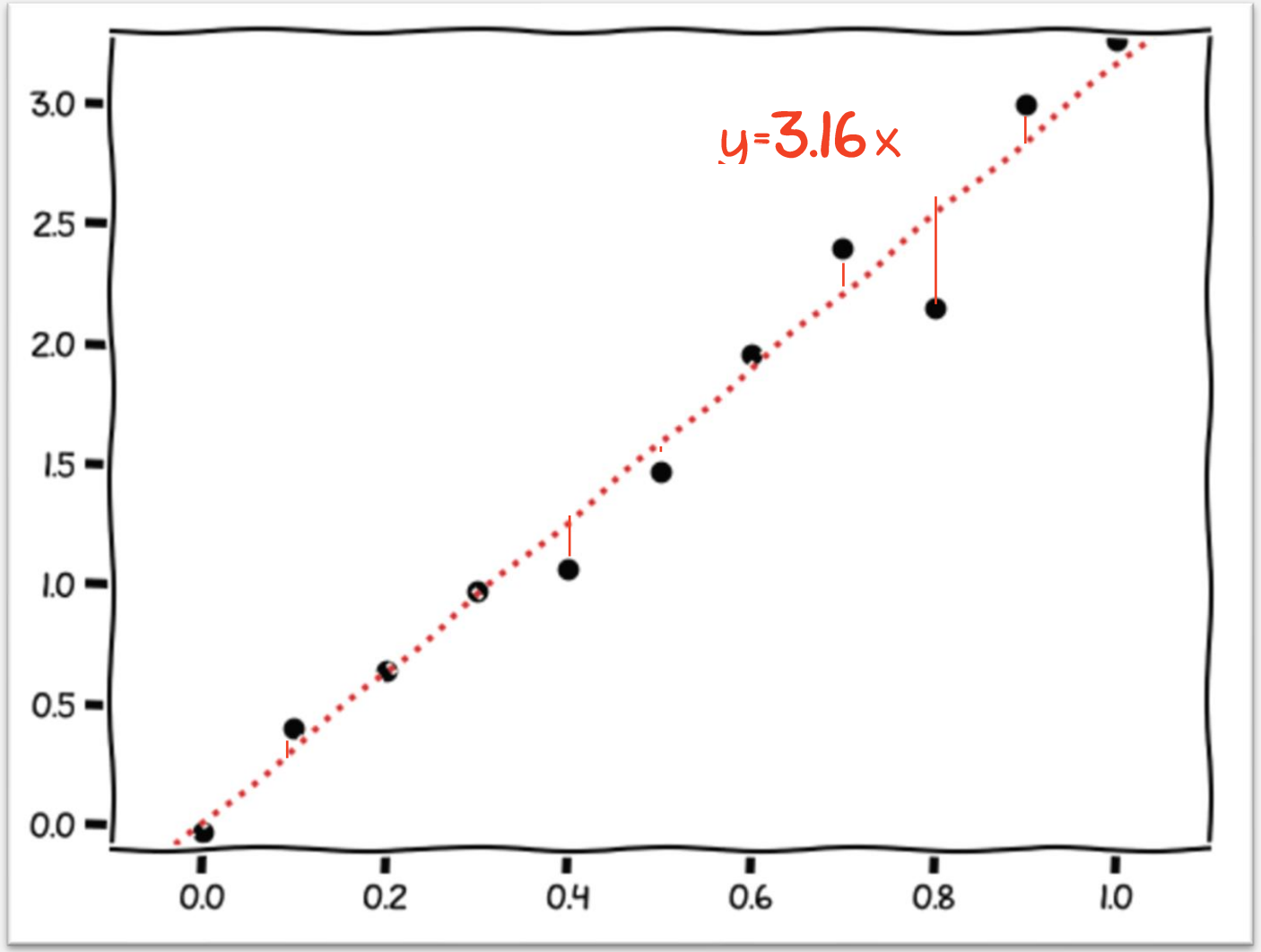
`sklearn.linear_model`.LinearRegression

linear regression

fitting a line $y = ax$

This assumes y depends on x!

If you want a *correlation* you will get a different answer



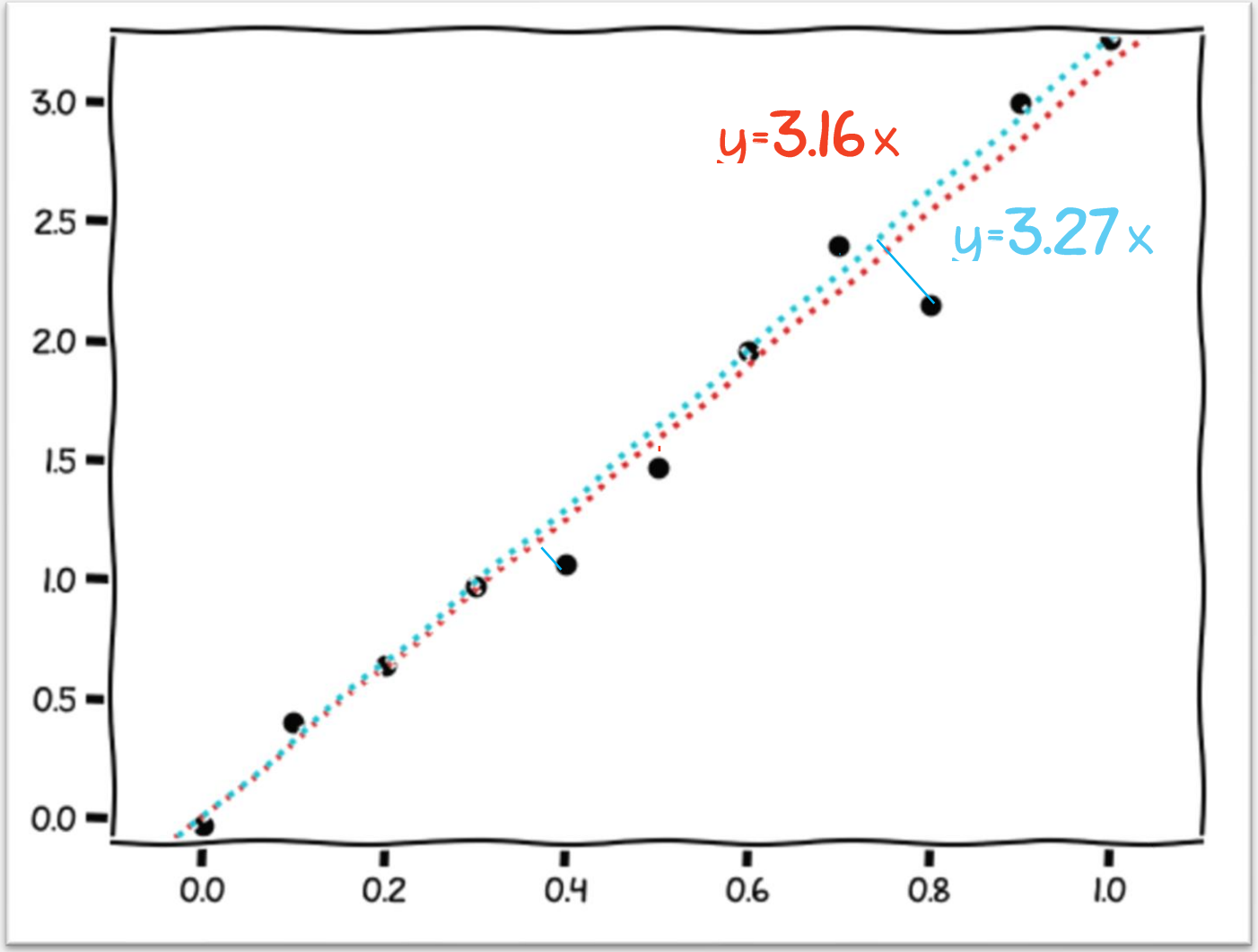
`sklearn.linear_model`.LinearRegression

linear regression

fitting a line $y = ax$

This assumes y depends on x!

If you want a *correlation* you will get a different answer



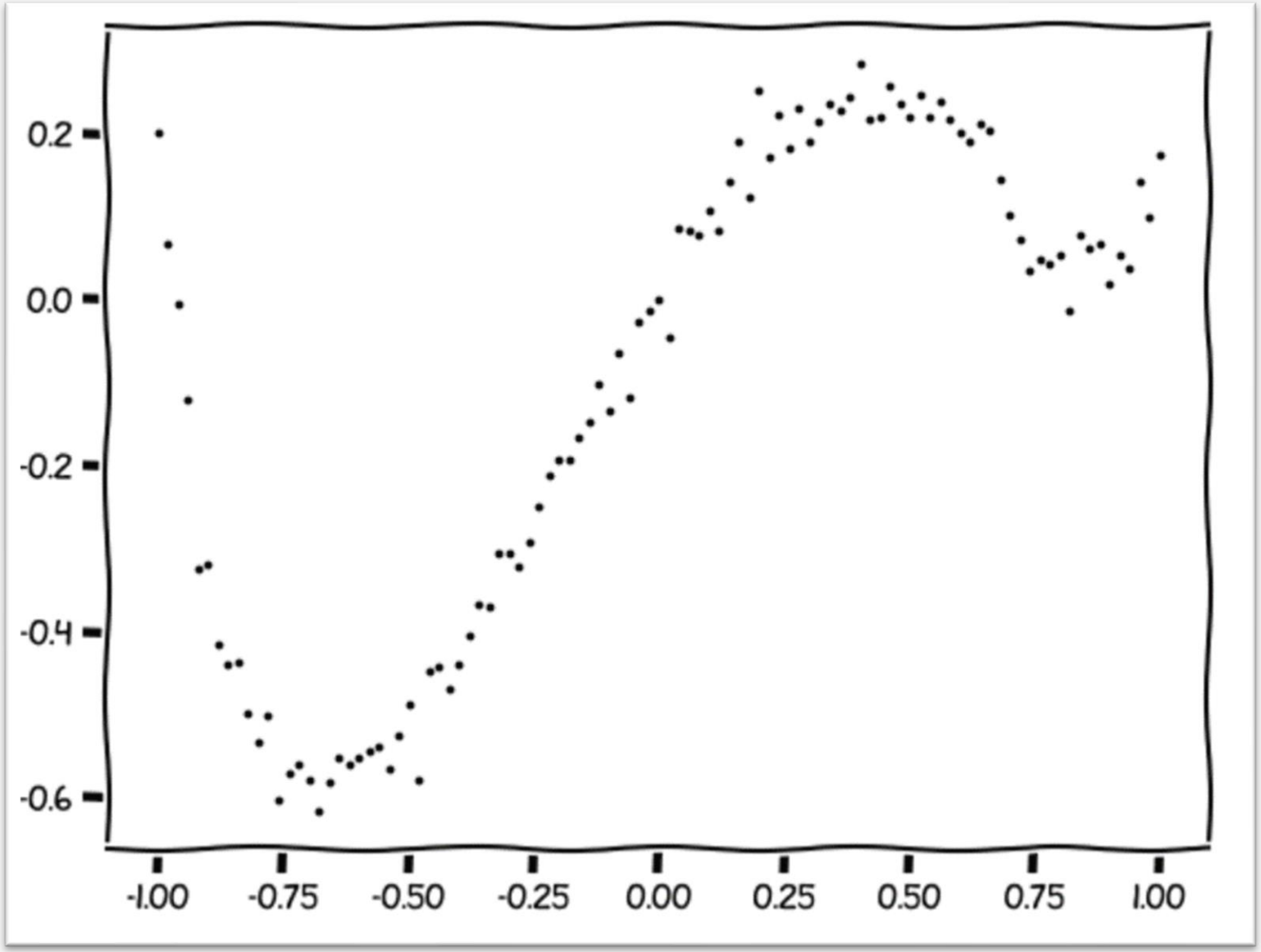
`sklearn.linear_model`.LinearRegression

`sklearn.decomposition`.PCA

lasso regression

`sklearn.linear_model.Lasso`

regression

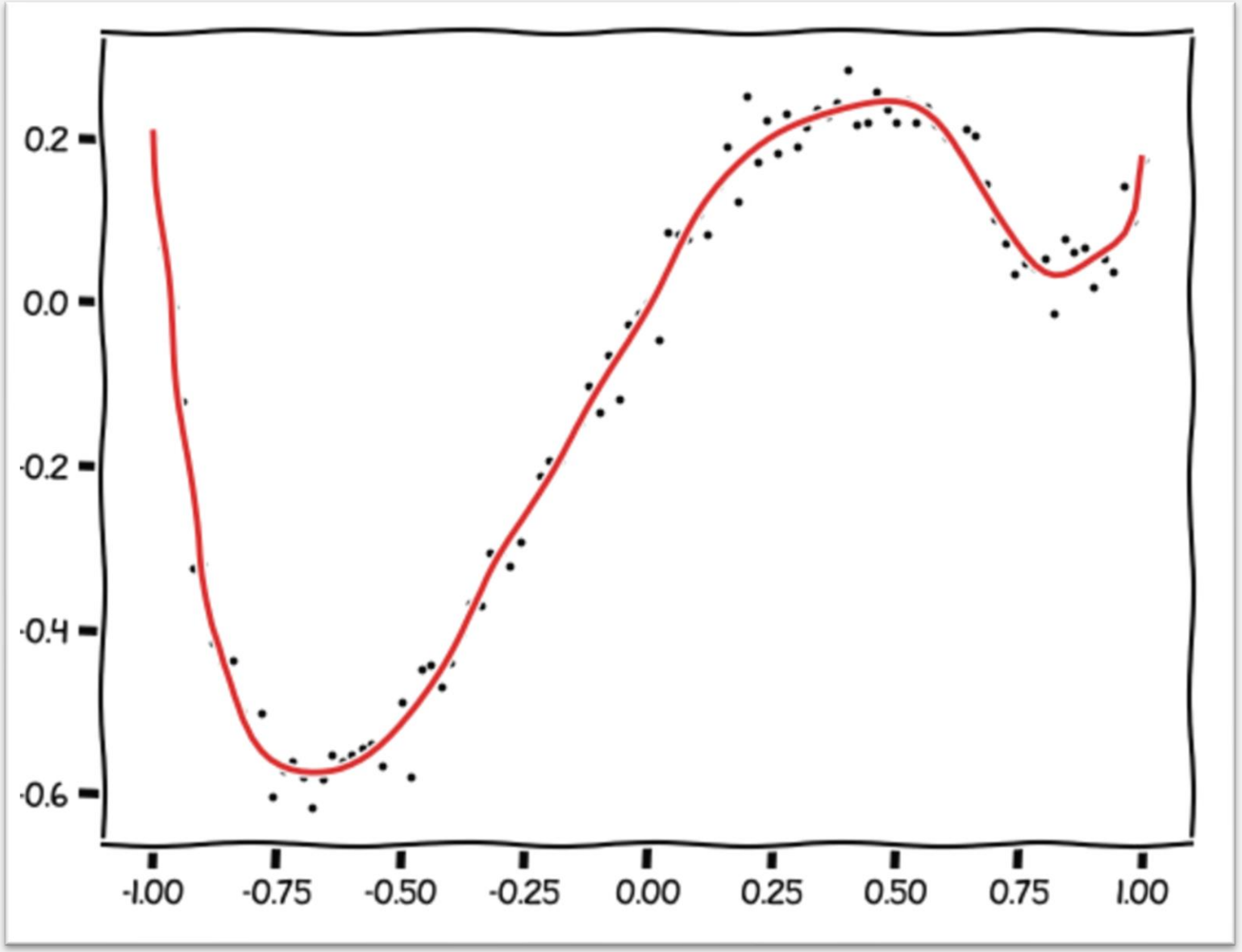


regression

$$y = \Sigma($$

-0.002	x
1	x^2
0.05	x^3
-4	x^4
-10	x^5
28	x^6
63	x^7
-100	x^8
-178	x^9
173	x^{10}
256	x^{11}
-140	x^{12}
-181	x^{13}
43	x^{14}
50	x^{15}

$)$

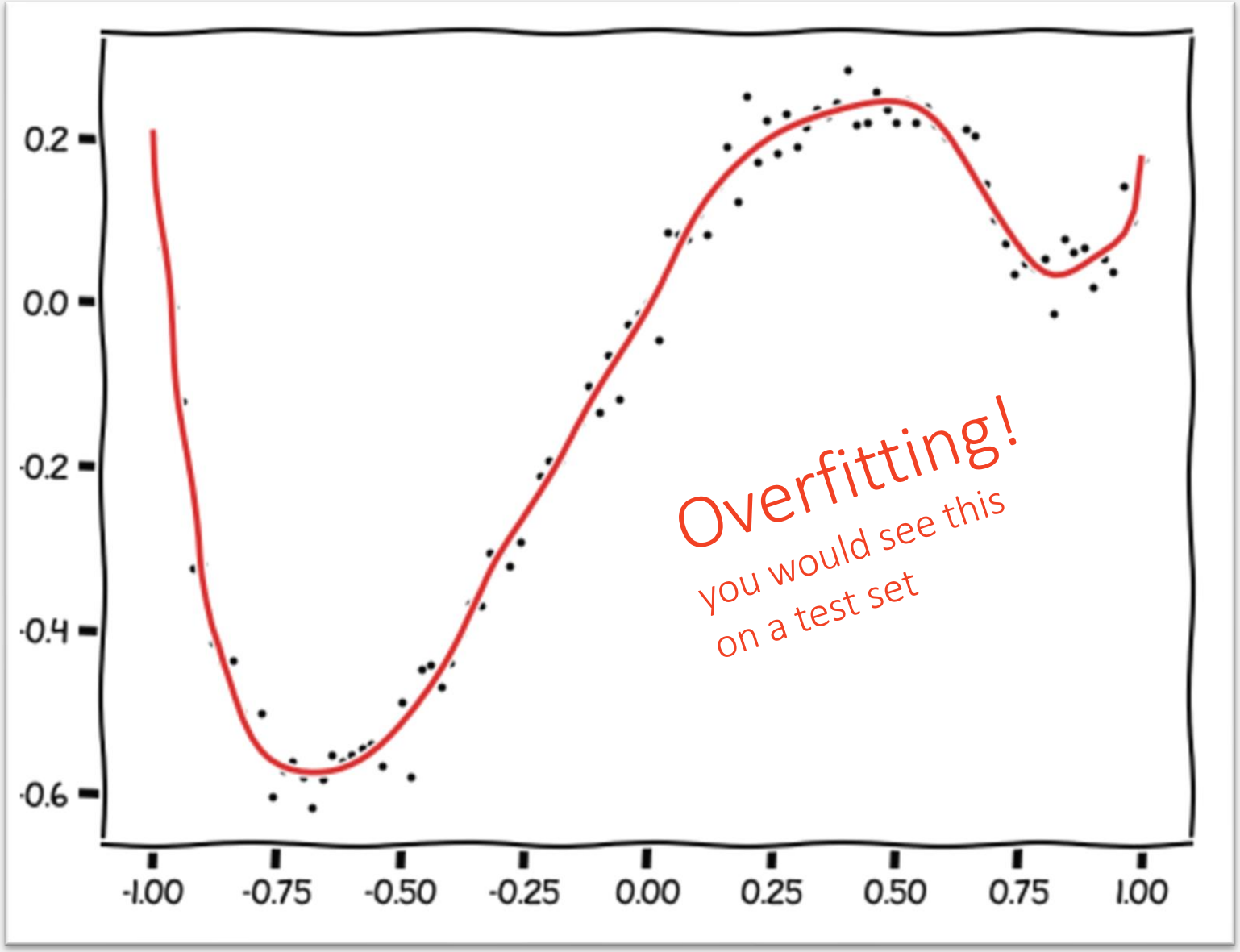


regression

$$y = \Sigma($$

-0.002	x
1	x^2
0.05	x^3
-4	x^4
-10	x^5
28	x^6
63	x^7
-100	x^8
-178	x^9
173	x^{10}
256	x^{11}
-140	x^{12}
-181	x^{13}
43	x^{14}
50	x^{15}

$$)$$



lasso regression

"least absolute shrinkage and selection operator"

is a regularization technique

linear coefficients

$$\mathcal{L} = \underbrace{\frac{|y - \vec{\alpha} \cdot \vec{x}|^2}{N}}_{\text{least squares loss}} + \lambda \sum |\vec{\alpha}|$$

lasso parameter (lagrange multiplier)

Our first fancy loss function!

lasso regression

"least absolute shrinkage and selection operator"

is a regularization technique

linear coefficients

$$\mathcal{L} = \underbrace{\frac{|y - \vec{\alpha} \cdot \vec{x}|^2}{N}}_{\text{least squares loss}} + \lambda \sum |\vec{\alpha}|$$

lasso parameter (lagrange multiplier)

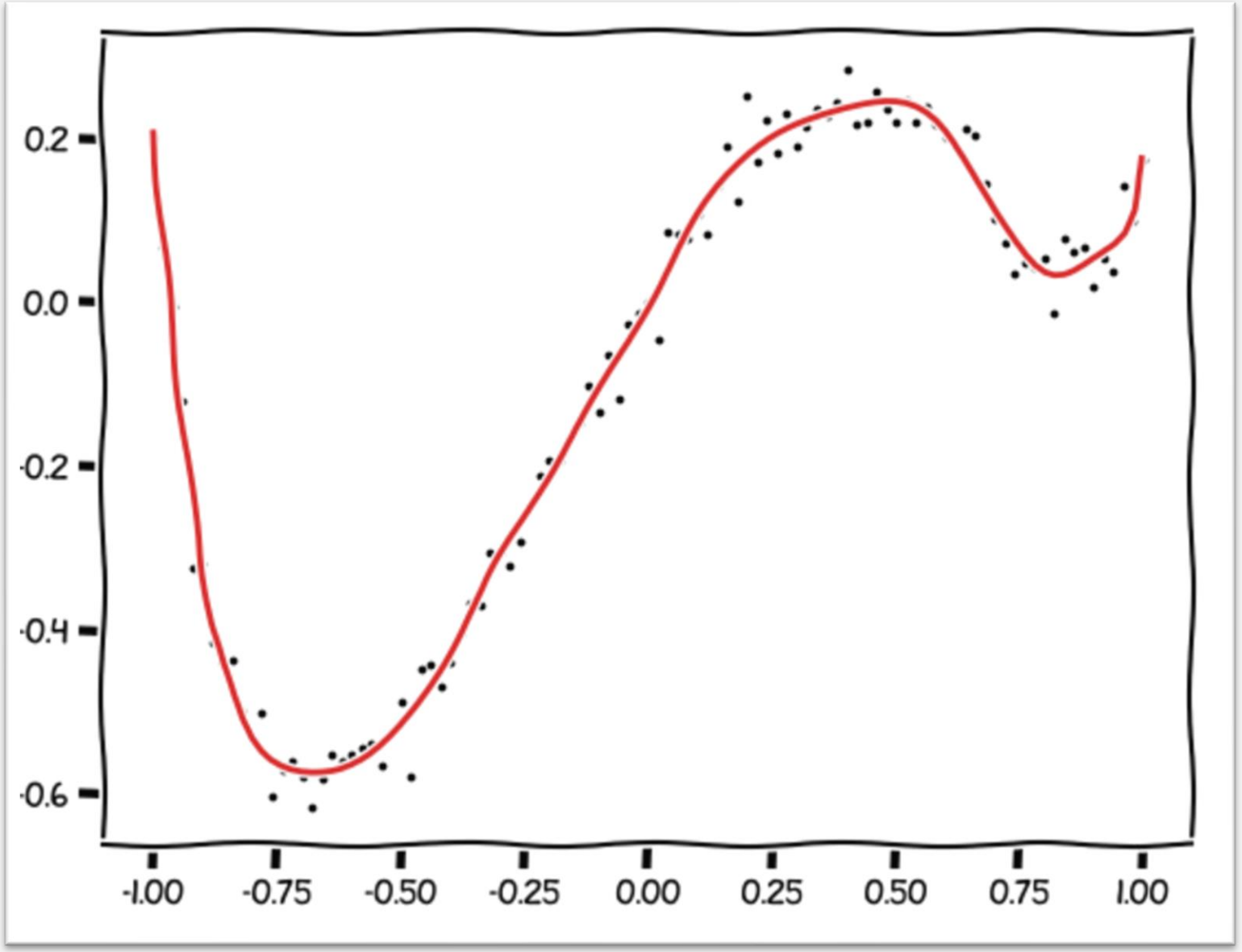
This thing is called alpha in scikit-learn to make your life harder!

Our first fancy loss function!

$$y = \Sigma($$

-0.002	x
1	x^2
0.05	x^3
-4	x^4
-10	x^5
28	x^6
63	x^7
-100	x^8
-178	x^9
173	x^{10}
256	x^{11}
-140	x^{12}
-181	x^{13}
43	x^{14}
50	x^{15}

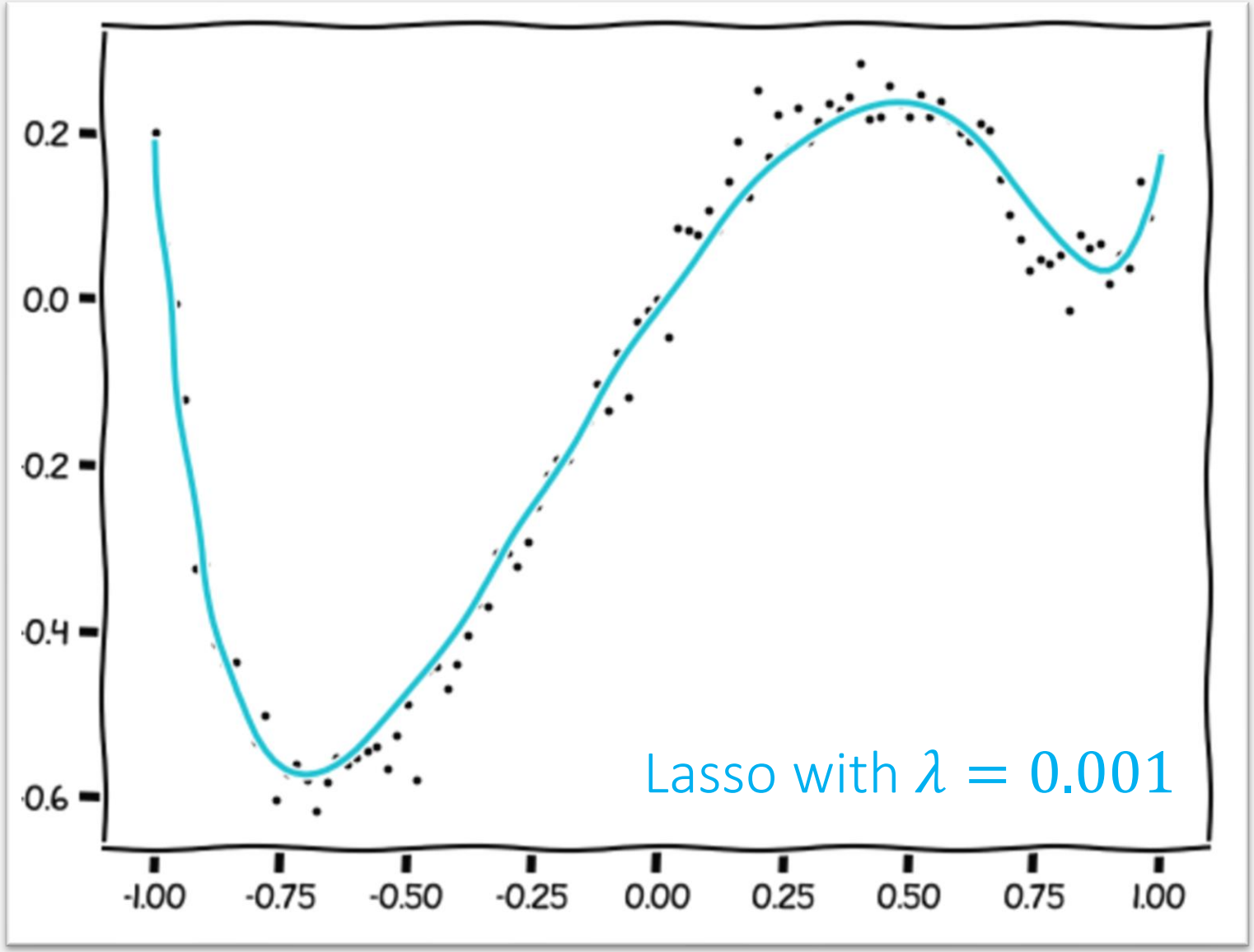
$)$



$$y = \Sigma($$

-0.01	x
0.9	x^2
-0.4	x^3
-0.6	x^4
0	x^5
-0.3	x^6
0	x^7
0	x^8
0	x^9
0	x^{10}
0.6	x^{11}
0	x^{12}
0.03	x^{13}
0	x^{14}
0	x^{15}

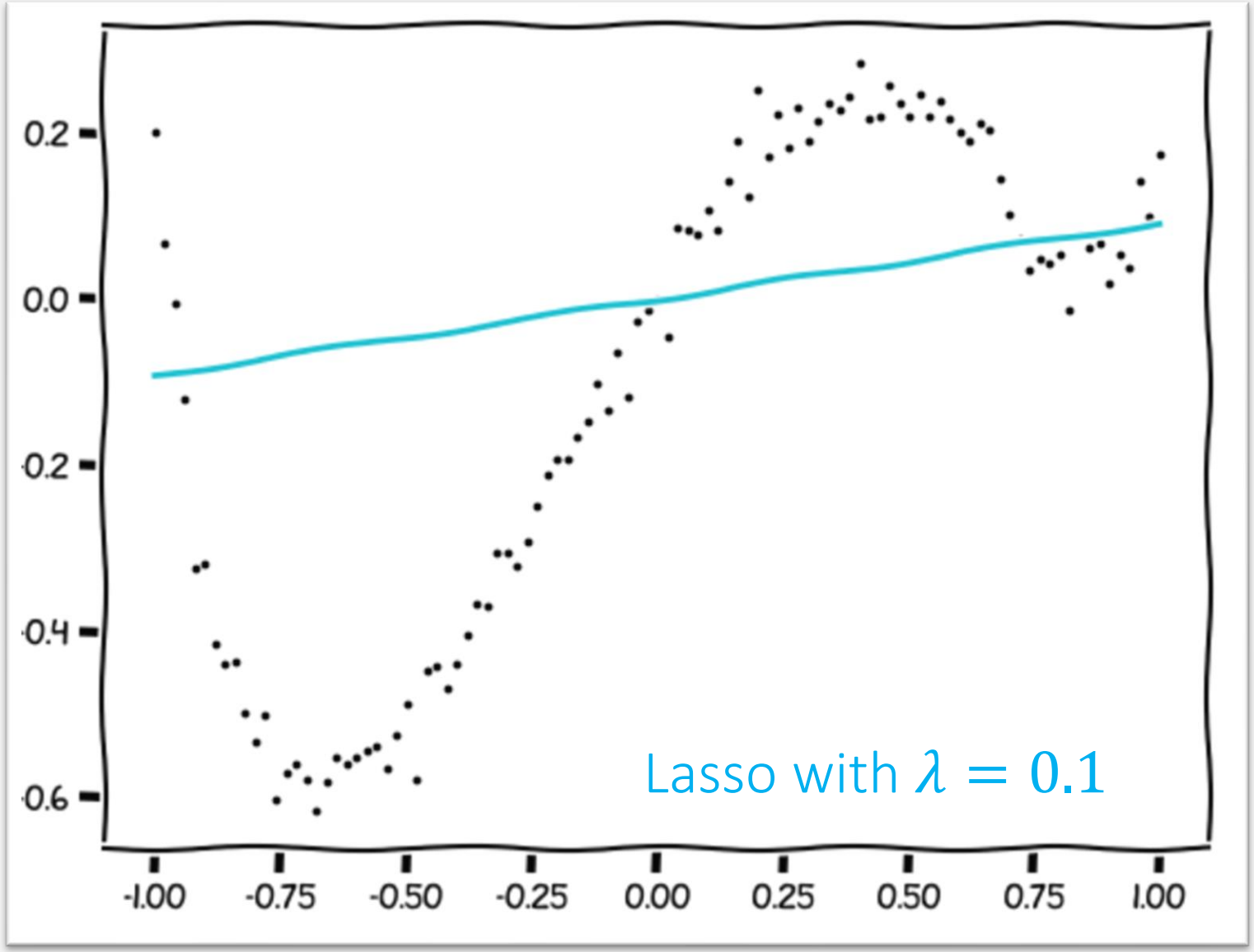
$)$



$$y = \Sigma($$

-0.01	x
0.09	x^2
-0.4	x^3
-0.6	x^4
0	x^5
-0.3	x^6
0	x^7
0	x^8
0	x^9
0	x^{10}
0.6	x^{11}
0	x^{12}
0.03	x^{13}
0	x^{14}
0	x^{15}

$$)$$



lasso regression

"least absolute shrinkage and selection operator"

is a regularization technique

pros

very easy (my usual pro)
good at reducing # of dimensions and selecting important features
easy to implement to other techniques: just change your loss
Helps to avoid overfitting – amazing!

cons

Need to tune λ – see `sklearn.linear_model.LassoCV`
Not stable: different resamples will output different coefficients
Has no knowledge of complexity by default
All of this is worse if coefficients are correlated

`sklearn.linear_model.Ridge`

ridge regression

Aka Tikhonov regularization

another regularization technique

linear coefficients

$$\mathcal{L} = \frac{|y - \vec{\alpha} \cdot \vec{x}|^2}{N} + \lambda \sum |\vec{\alpha}|^2$$

ridge parameter
(lagrange multiplier)

least squares
loss

`sklearn.linear_model.Ridge`

ridge regression

Aka Tikhonov regularization

another regularization technique

linear coefficients

$$\mathcal{L} = \frac{|y - \vec{\alpha} \cdot \vec{x}|^2}{N} + \lambda \sum |\vec{\alpha}|^2$$

pros more stable than Lasso

cons does not set parameters to 0 – not good for feature selection

If you ask me why I can probably explain but I don't want to

classification

decision trees!

`sklearn.tree`.DecisionTreeClassifier

lots of input parameters

can be heterogeneous, unnormalized, continuous, categorical, whatever really

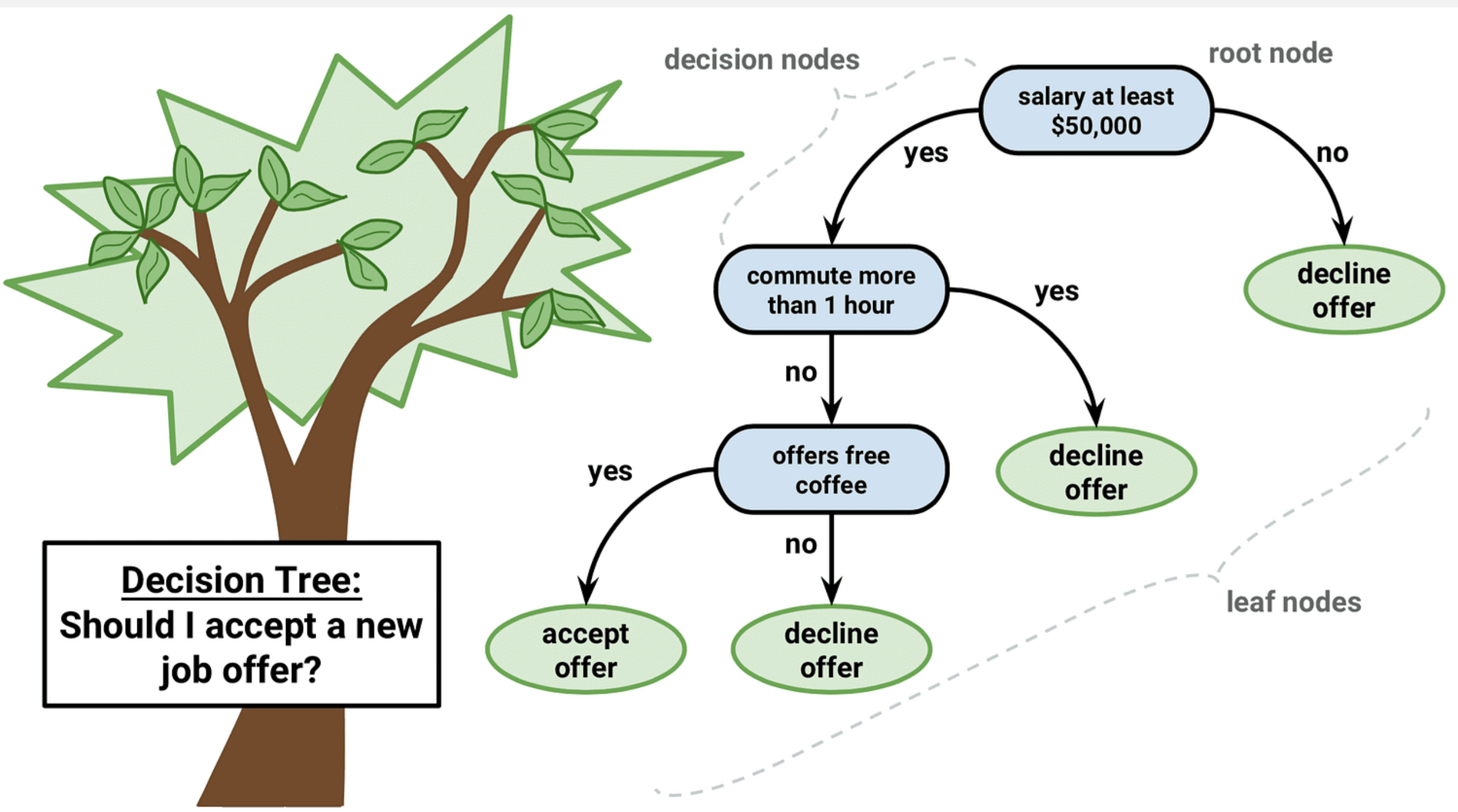
single categorical label

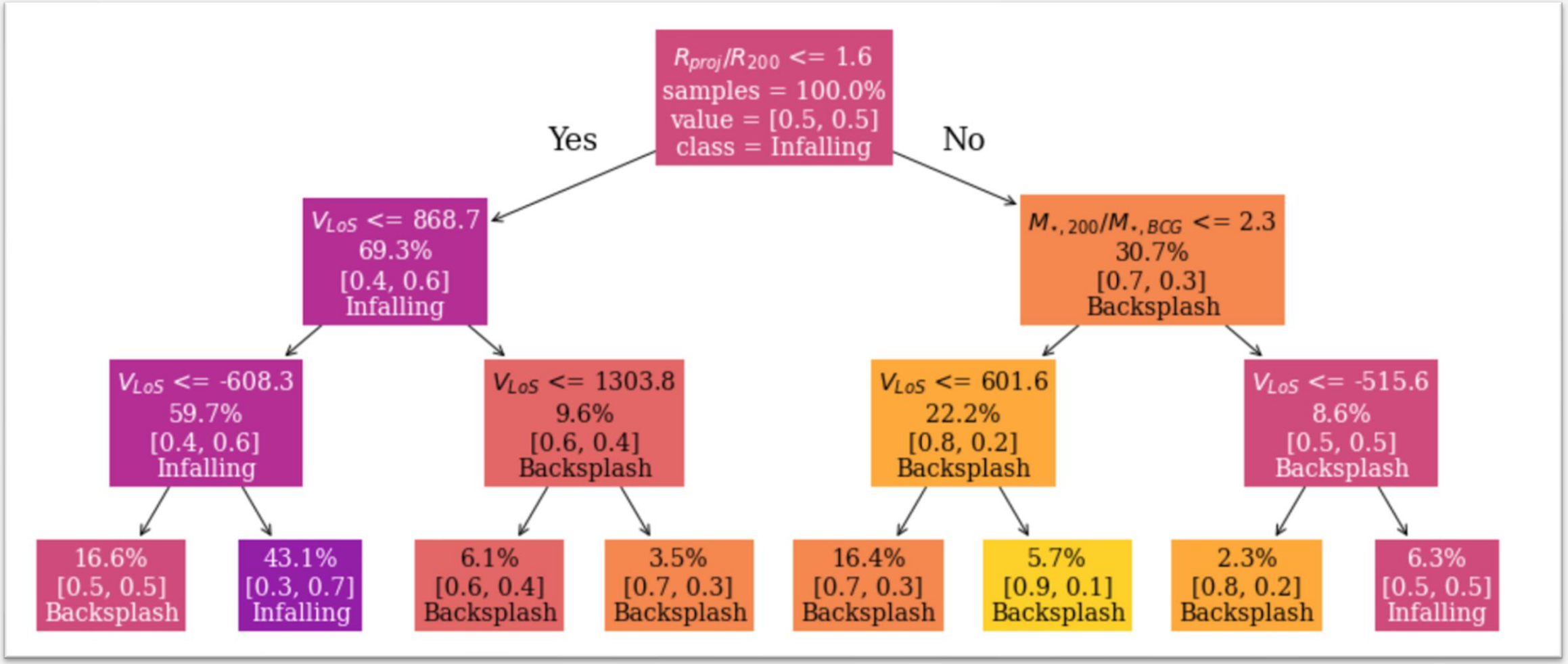
lots of input parameters

can be heterogeneous, unnormalized, continuous, categorical, whatever really

the reason they are great

single categorical label

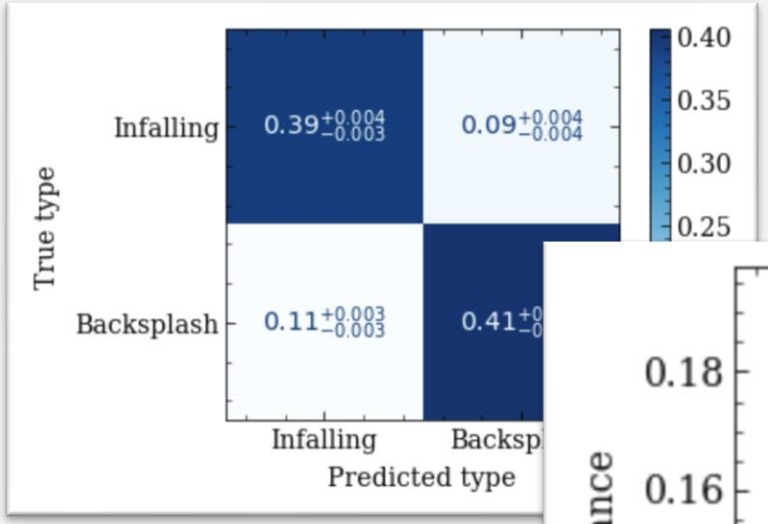

Decision Tree:
Should I accept a new job offer?
decision nodes
root node
salary at least $50,000
yes
no
commute more than 1 hour
yes
decline offer
no
offers free coffee
decline offer
yes
no
accept offer
decline offer
leaf nodes



Make nice interpretable figures!

If your trees are simple

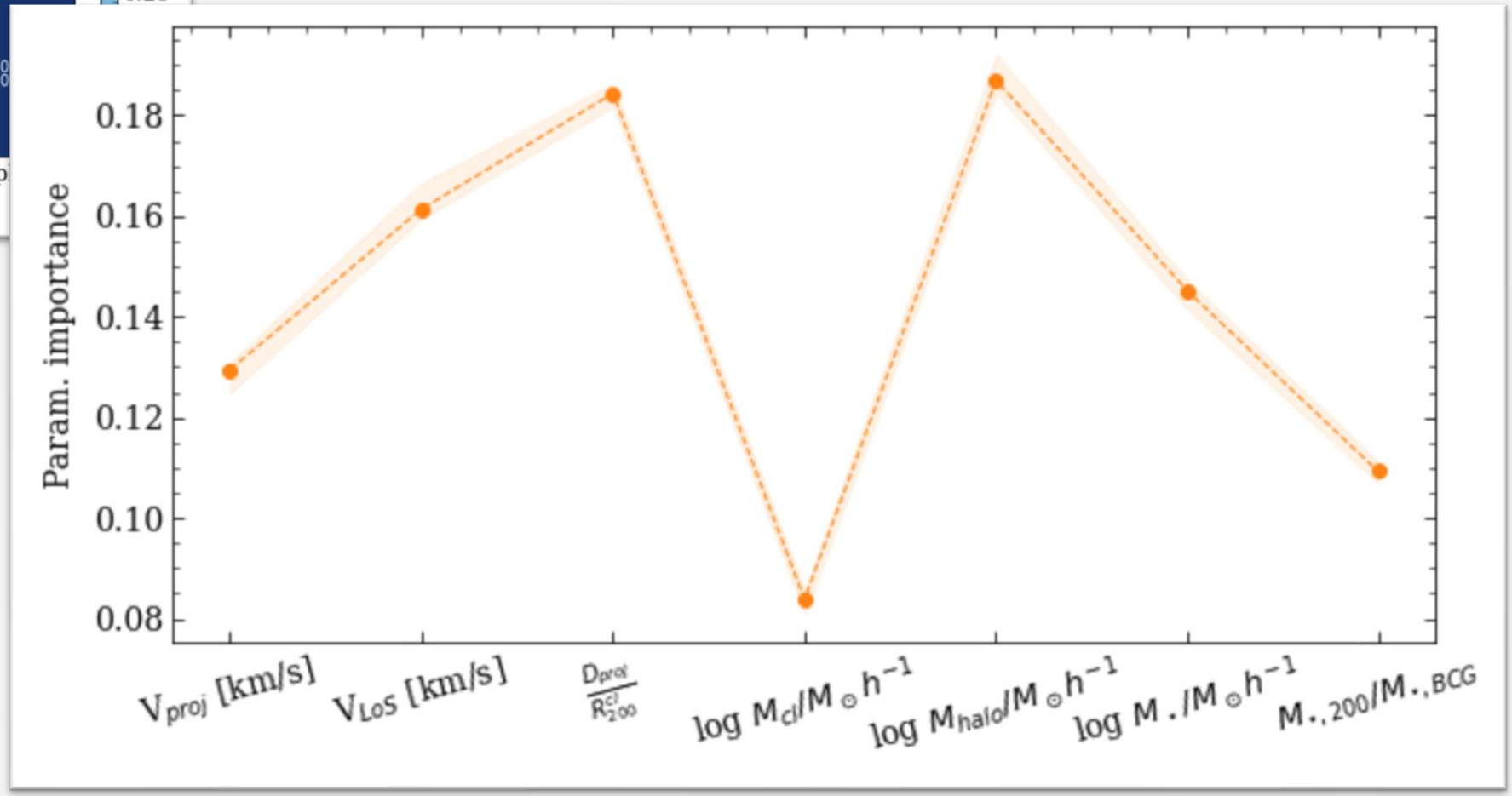
`sklearn.tree.plot_tree`¶



`sklearn.metrics`.confusion_matrix

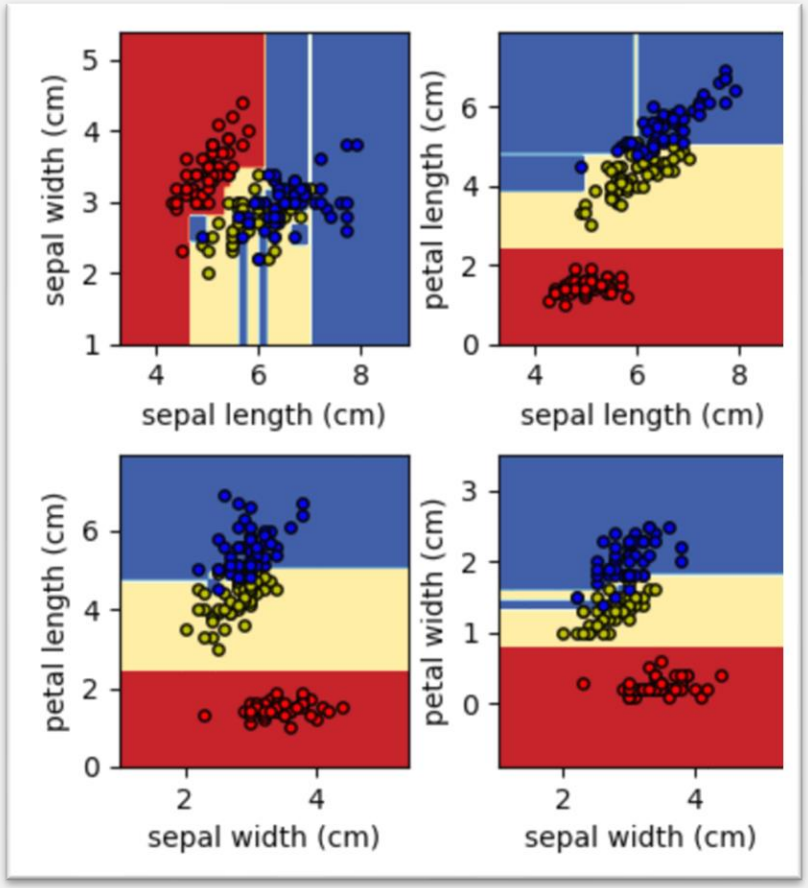
`sklearn.metrics`.ConfusionMatrixDisplay

Get great results & see which parameters matter most!

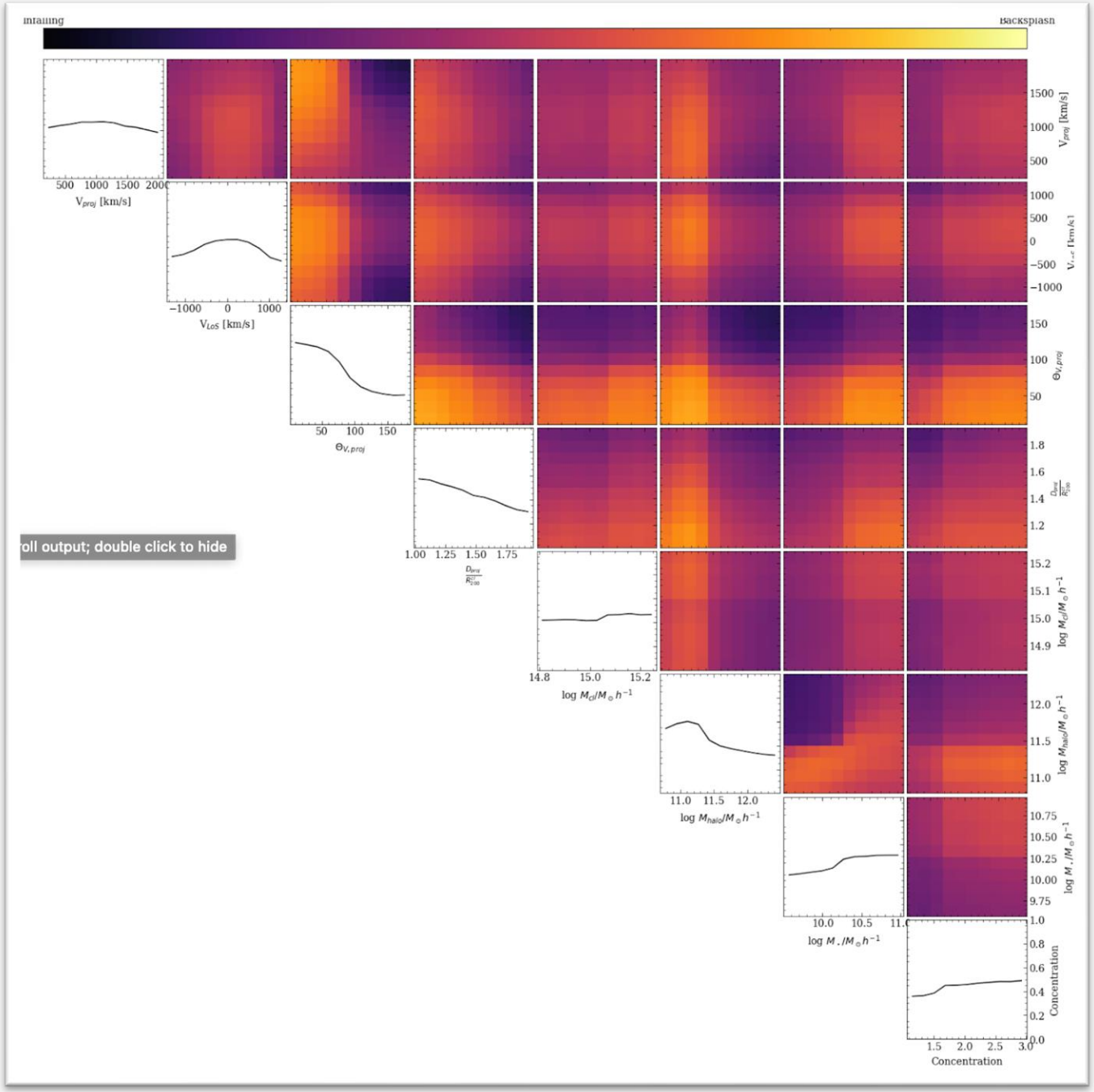


property `feature_importances_`

Show non-linear decision boundaries!

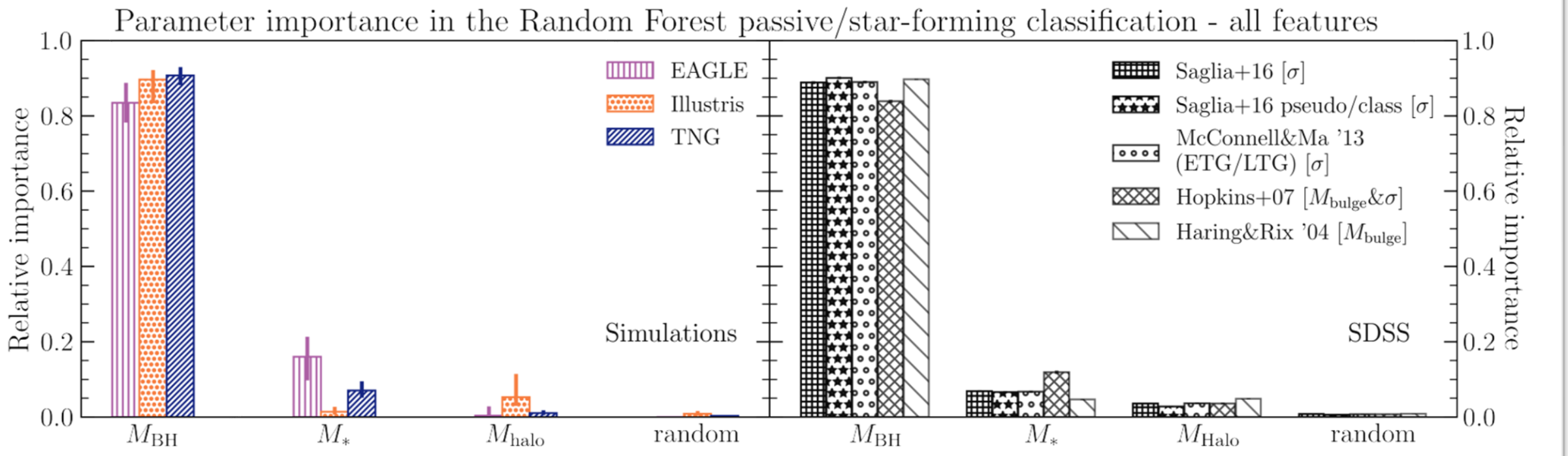


`sklearn.inspection`.DecisionBoundaryDisplay



Can even prove causation over correlation?

Piotrowska et al., 2022



https://arxiv.org/pdf/2112.07672.pdf

lesson 2

decision trees!

`sklearn.tree`.DecisionTreeClassifier

pros

Trees!
Extremely interpretable
Can capture non-linear relationships
Feature importances – causation vs correlation?
Works with basically any input data w/out worries

cons

Very unstable
Require ***a lot*** of tuning
Not good on imbalanced datasets (like most things)
Cannot do anything except “yes” or “no” classification
Neural networks beat them on accuracy

pros

Trees!
Extremely interpretable
Can capture non-linear relationships
Feature importances – causation vs correlation?
Works with basically any input data w/out worries

cons

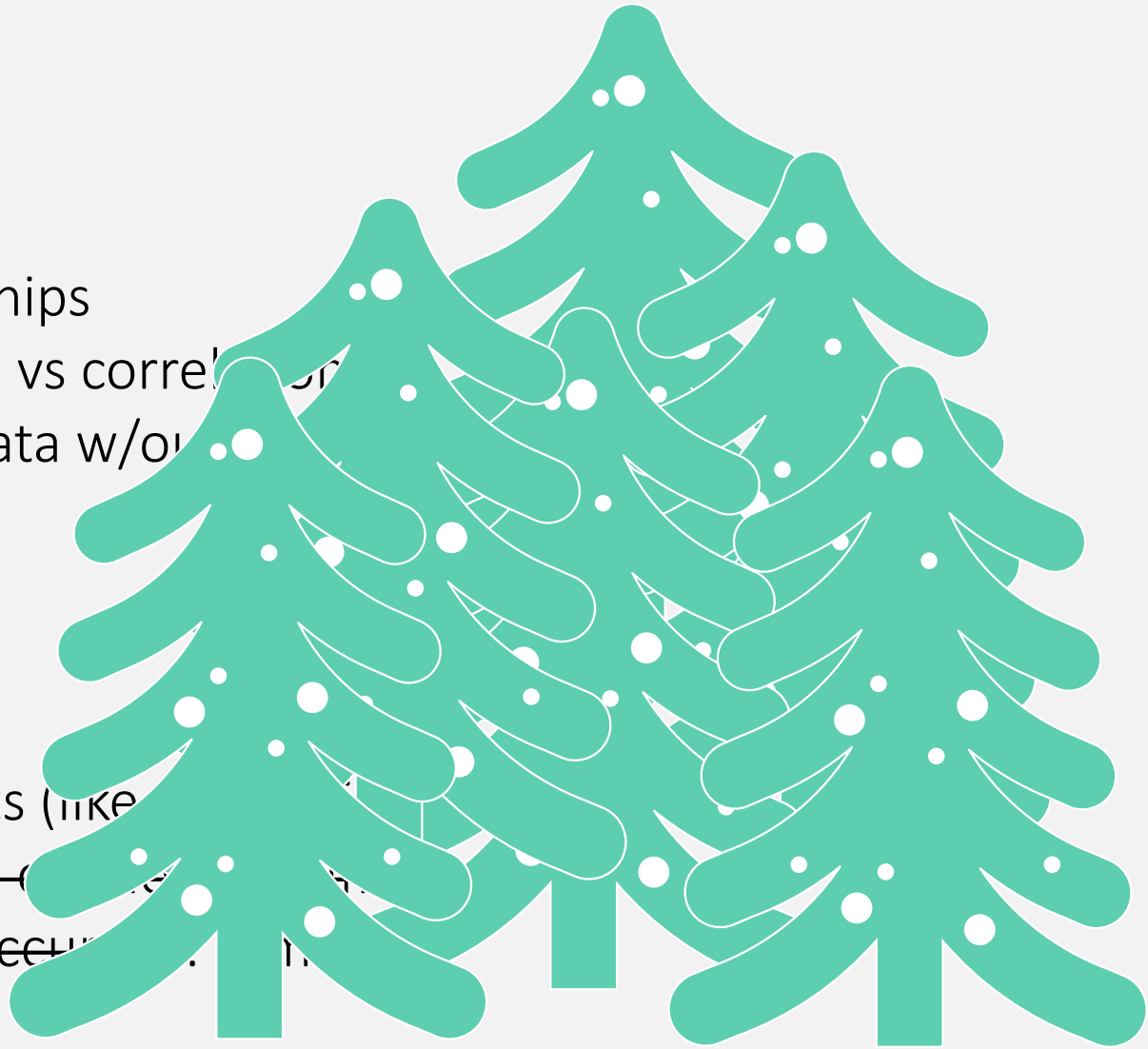
~~Very unstable~~
~~Require ***a lot*** of tuning~~
Not good on imbalanced datasets (like most things)
~~Cannot do anything except “yes” or “no” classification~~
~~Neural networks beat them on accuracy~~???maybe

pros

Trees!
Extremely interpretable
Can capture non-linear relationships
Feature importances – causation vs corre
Works with basically any input data w/o

cons

~~Very unstable~~
~~Require ***a lot*** of tuning~~
Not good on imbalanced datasets (like
~~Cannot do anything except "yes"~~
~~Neural networks beat them on accu~~



random forests

lots of input parameters

can be heterogeneous, unnormalized, continuous, categorical, whatever really

single categorical label

lots of input parameters

can be heterogeneous, unnormalized, continuous, categorical, whatever really

fraction of “yes” votes

is this a cat?

no

NO

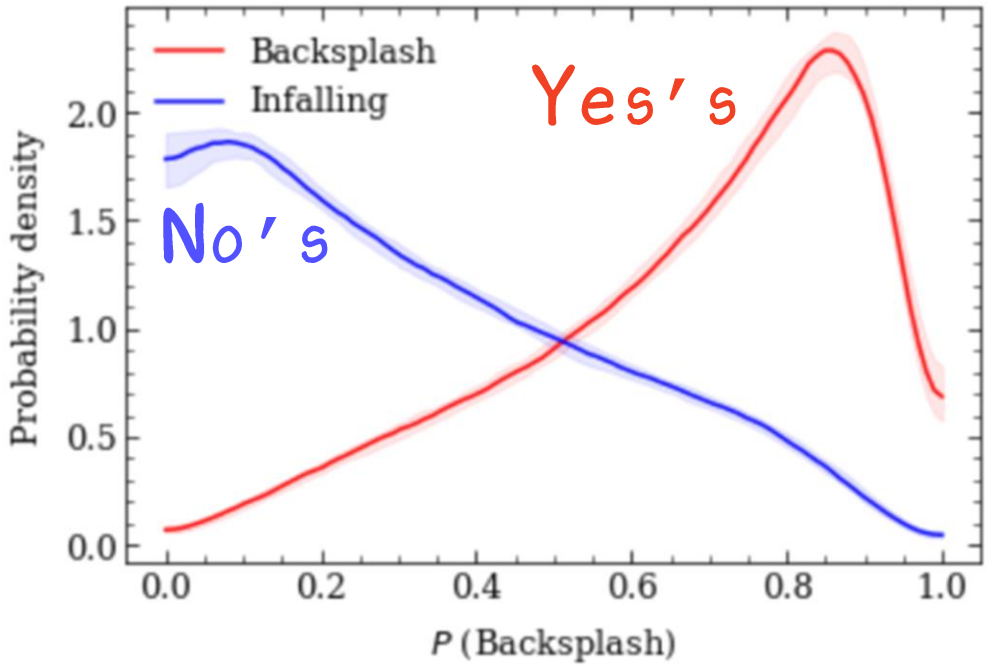
is this a cat?

yes

no

yes

p(cat)=0.67

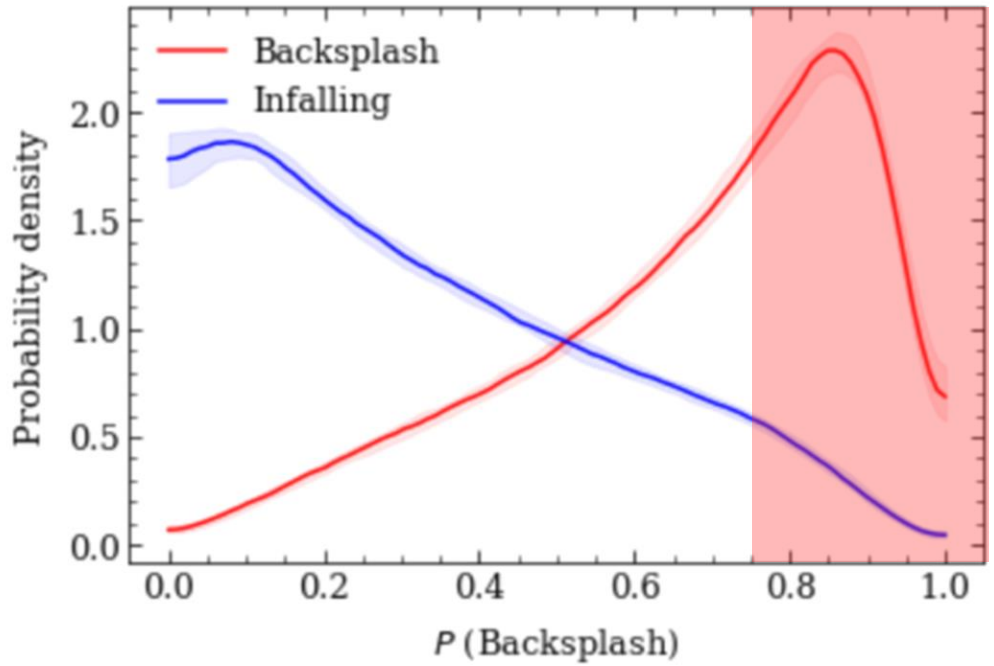


Probability density distribution of infalling and backsplash galaxies, as a function of **predicted backsplash probability**.

Infalling galaxies are more likely to be misclassified as backsplash than vice versa

p(yes)

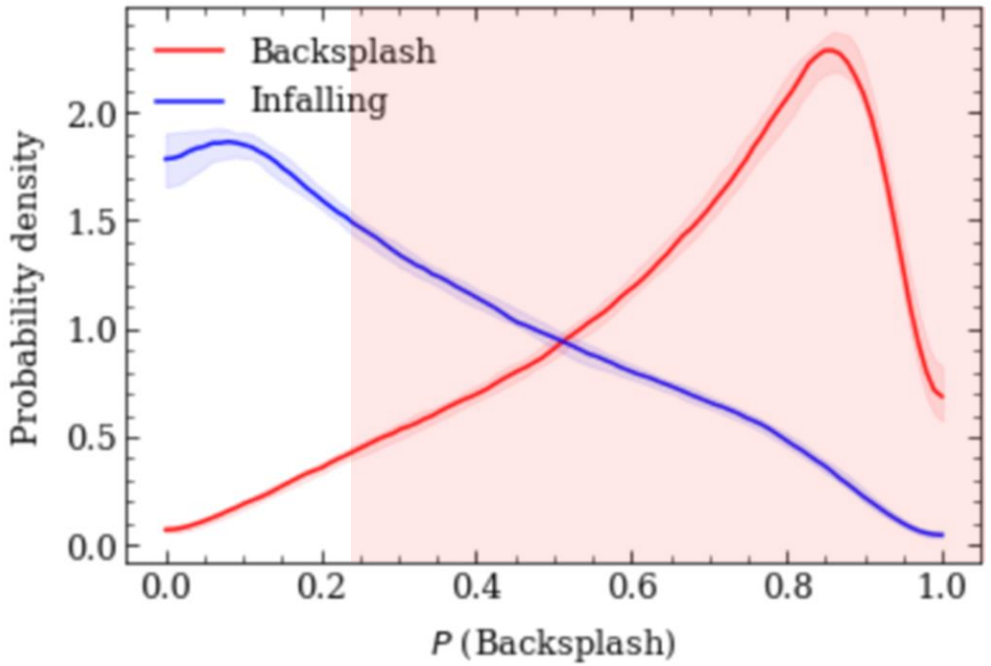
pure sample



Probability density distribution of infalling and backsplash galaxies, as a function of **predicted backsplash probability**.

Infalling galaxies are more likely to be misclassified as backsplash than vice versa

complete sample



Probability density distribution of infalling and backsplash galaxies, as a function of **predicted backsplash probability**.

Infalling galaxies are more likely to be misclassified as backsplash than vice versa

Galaxy Zoo

20 citizen scientists classify each galaxy, giving p(smooth)

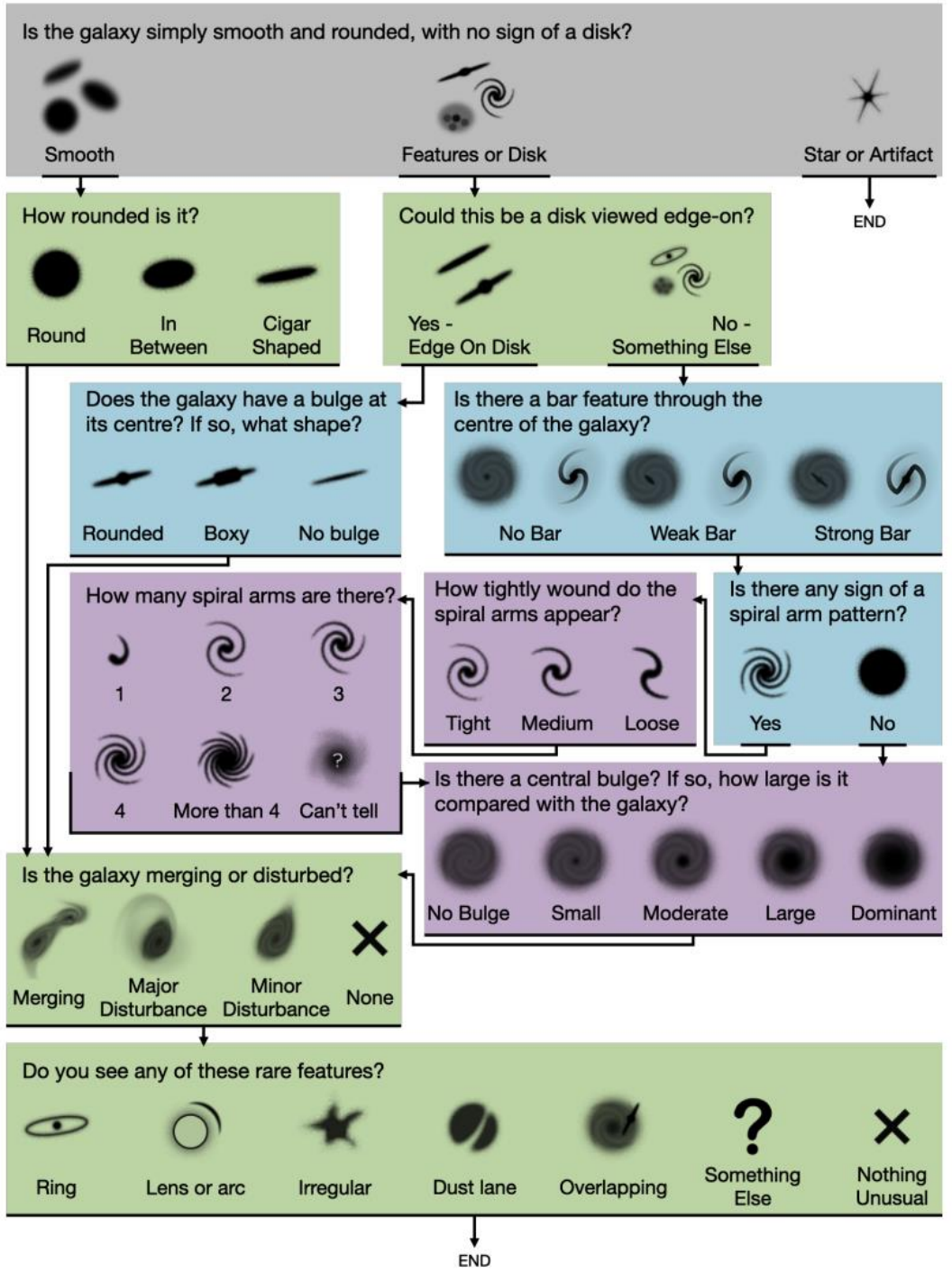
TASK | TUTORIAL

Is the galaxy simply smooth and rounded, with no sign of a disk?

Smooth

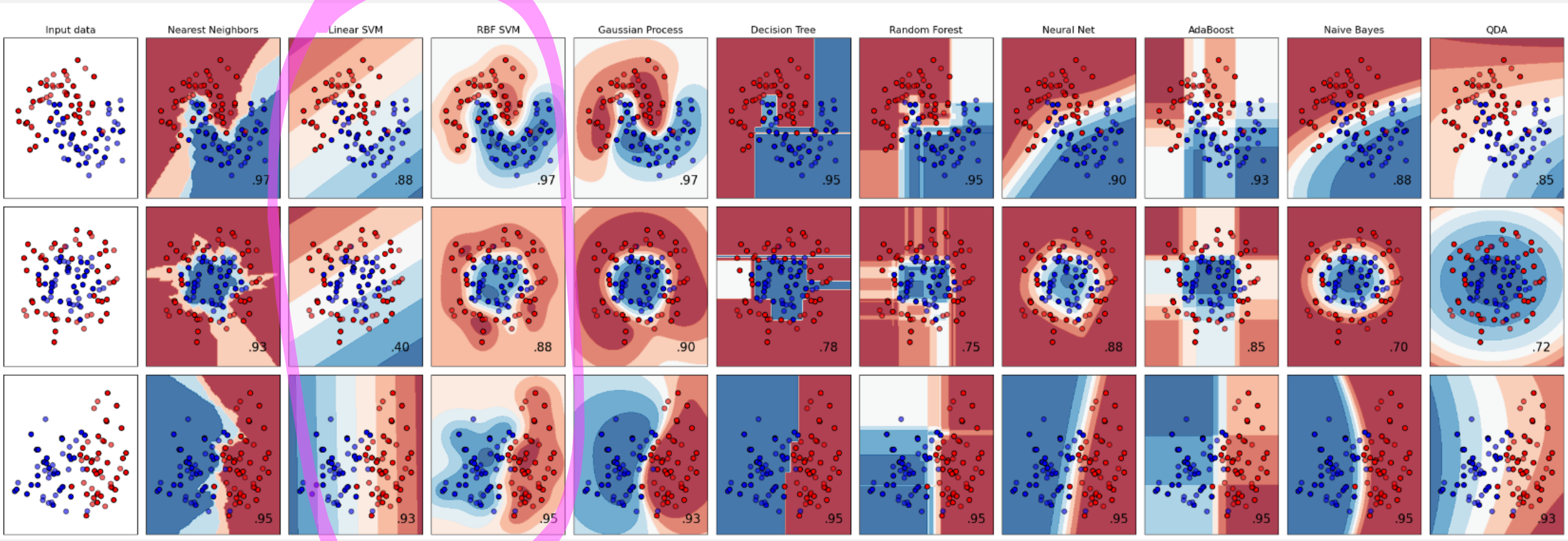
Features or Disk

Star, Artifact, or Bad Zoom



A random forest = a natural loss function!

Support Vector Machines



sklearn.svm.SVC

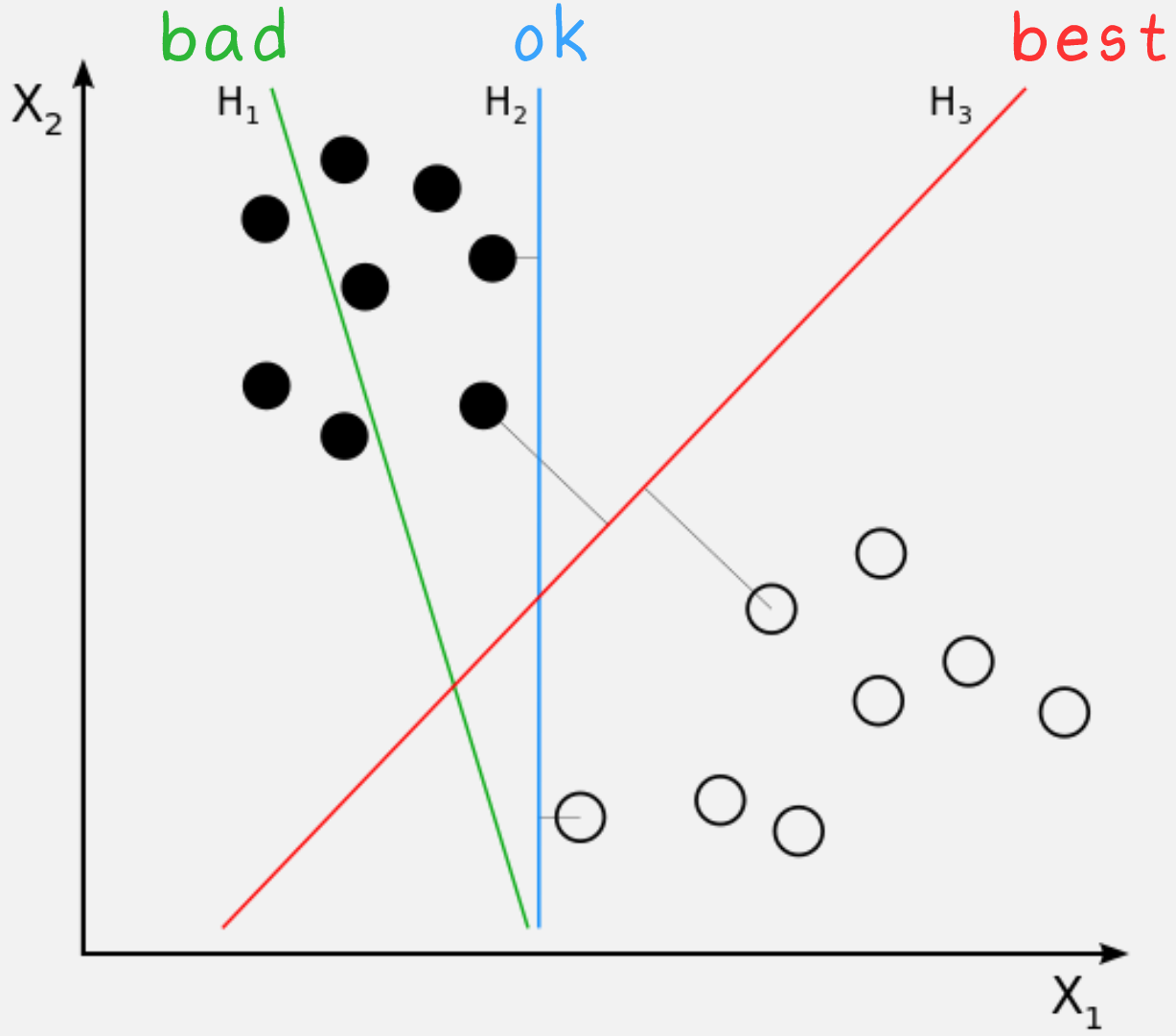
SVMs (SVCs)

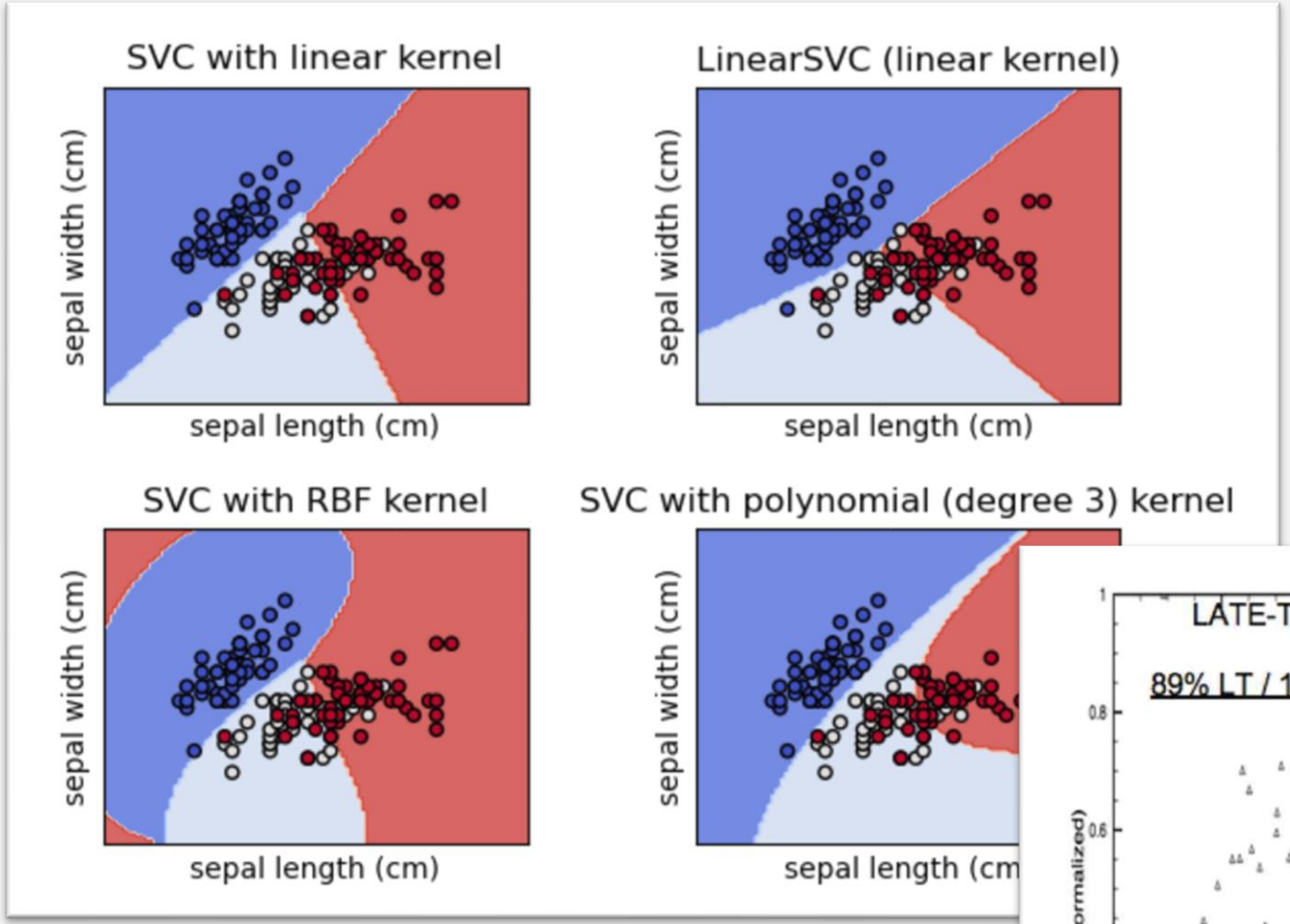
Think of them as linear regression but for classification

Don't quote me on this

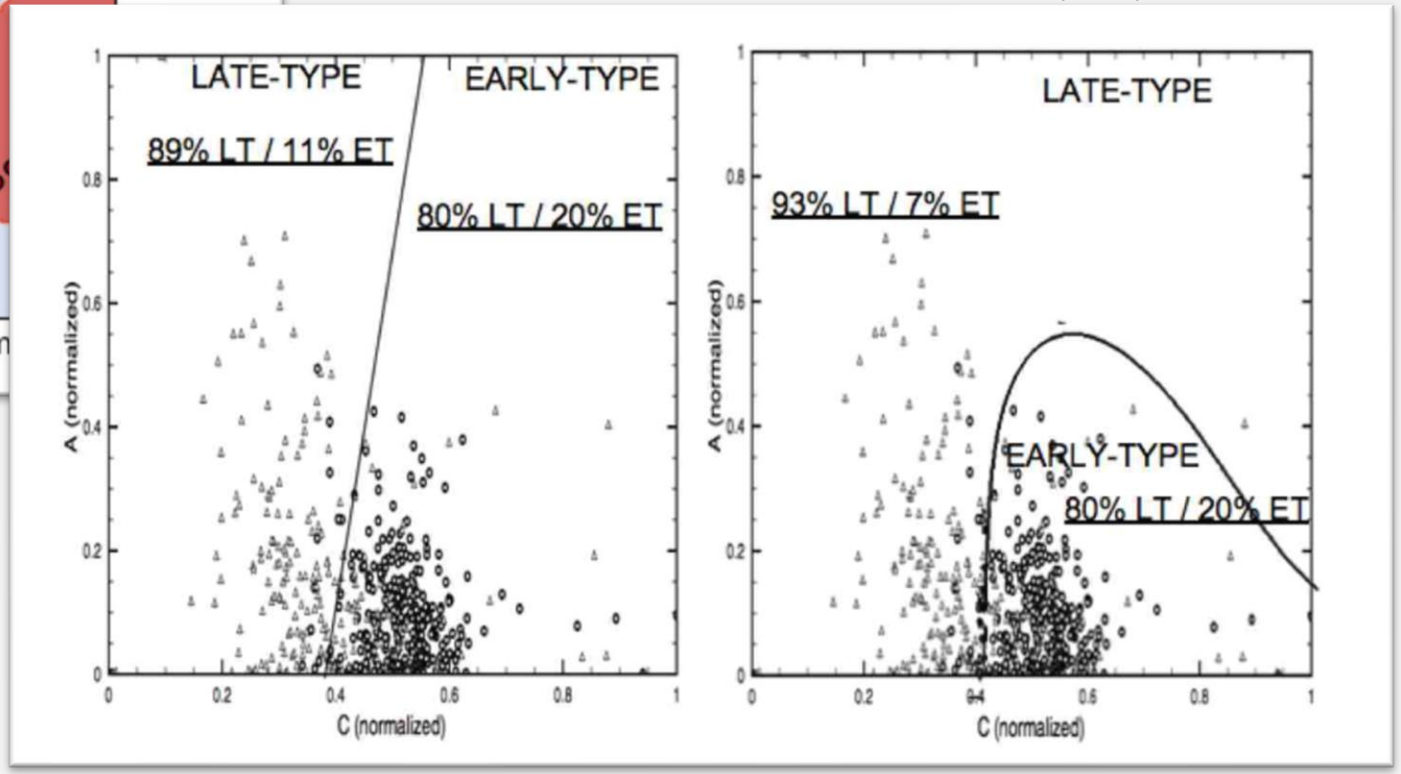
Fits some function (kernel or hyperplane) to get the **maximum separation** between classes

Distance to the **nearest point** on each side is maximized





Huertas-Company et al. 2008



SVMs (SVCs)

pros

very versatile (with RBF/polynomial kernels)
give you a meaningful formula you can tell other people
good in many-dimensional spaces
good for small datasets (even if $N_{dim} > N_{samples}$)
multi-class classifier!

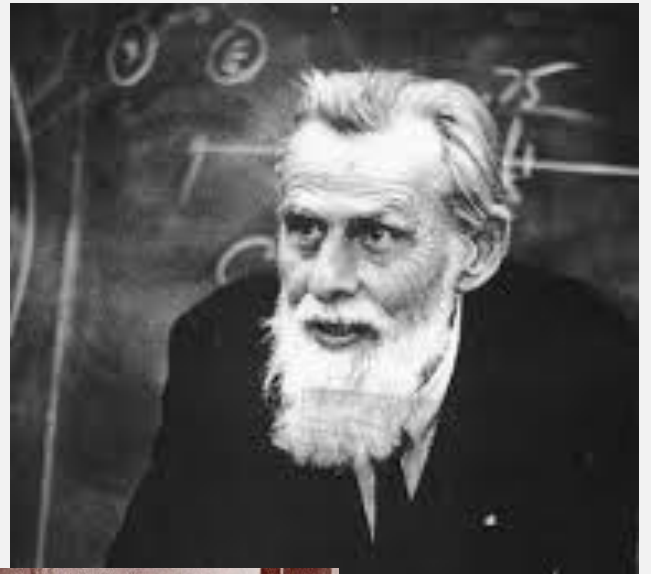
cons

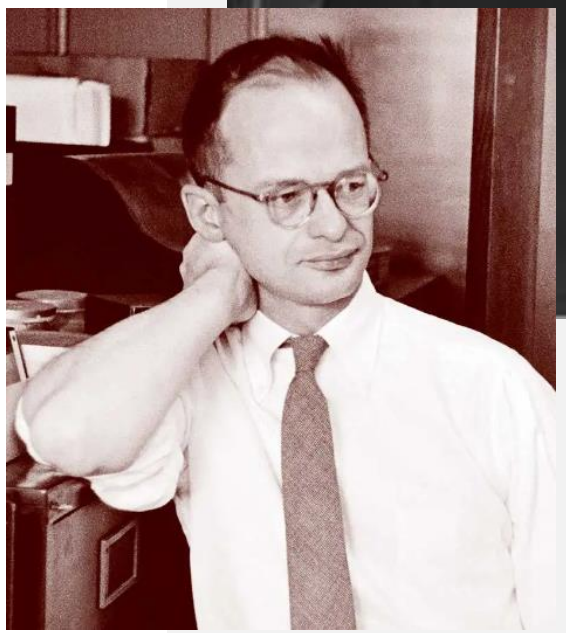
prone to overfitting (like most things in life)
can be slow
not great if classes are very overlapping (e.g., Gaussian mixtures)
need to be careful if classes are not balanced

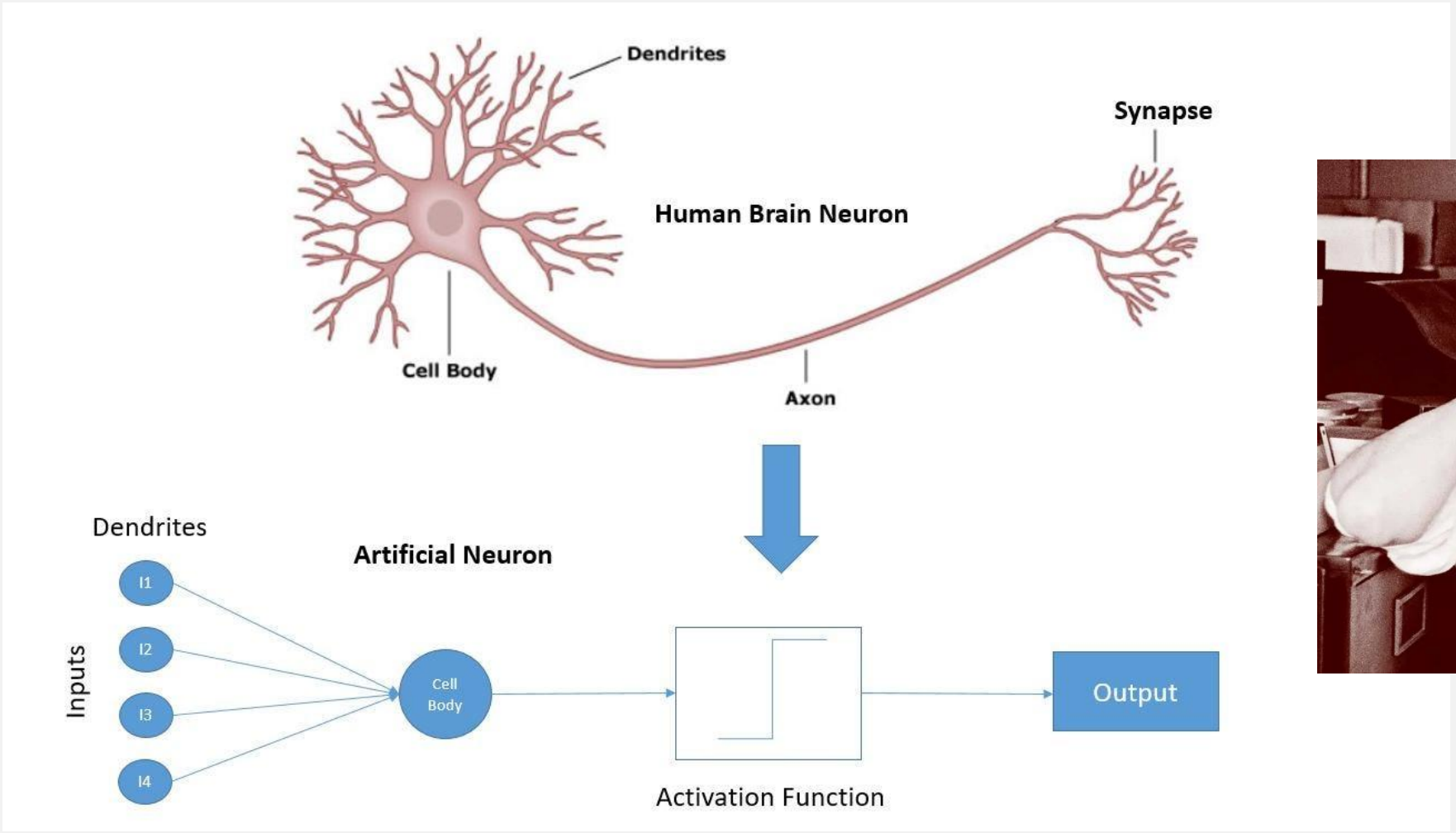
Deep Learning

1943 McCulloch & Pitts

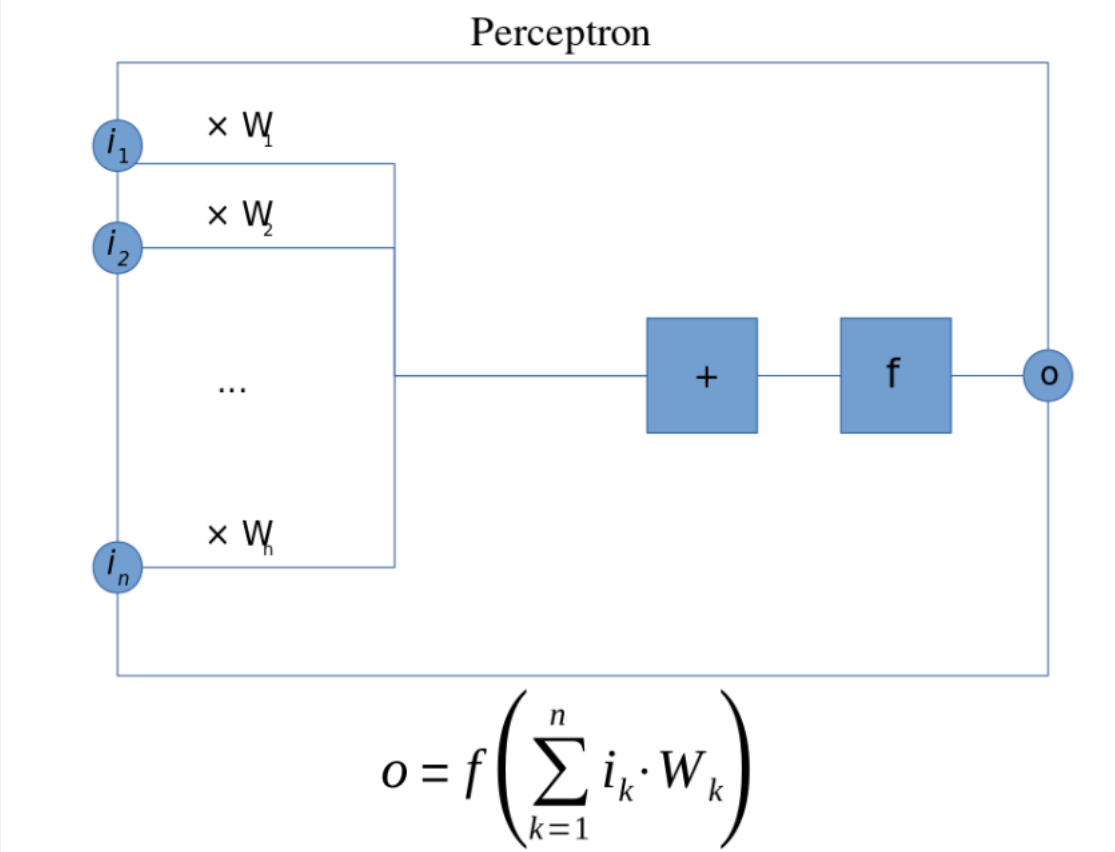
First mathematical model
of an artificial neuron





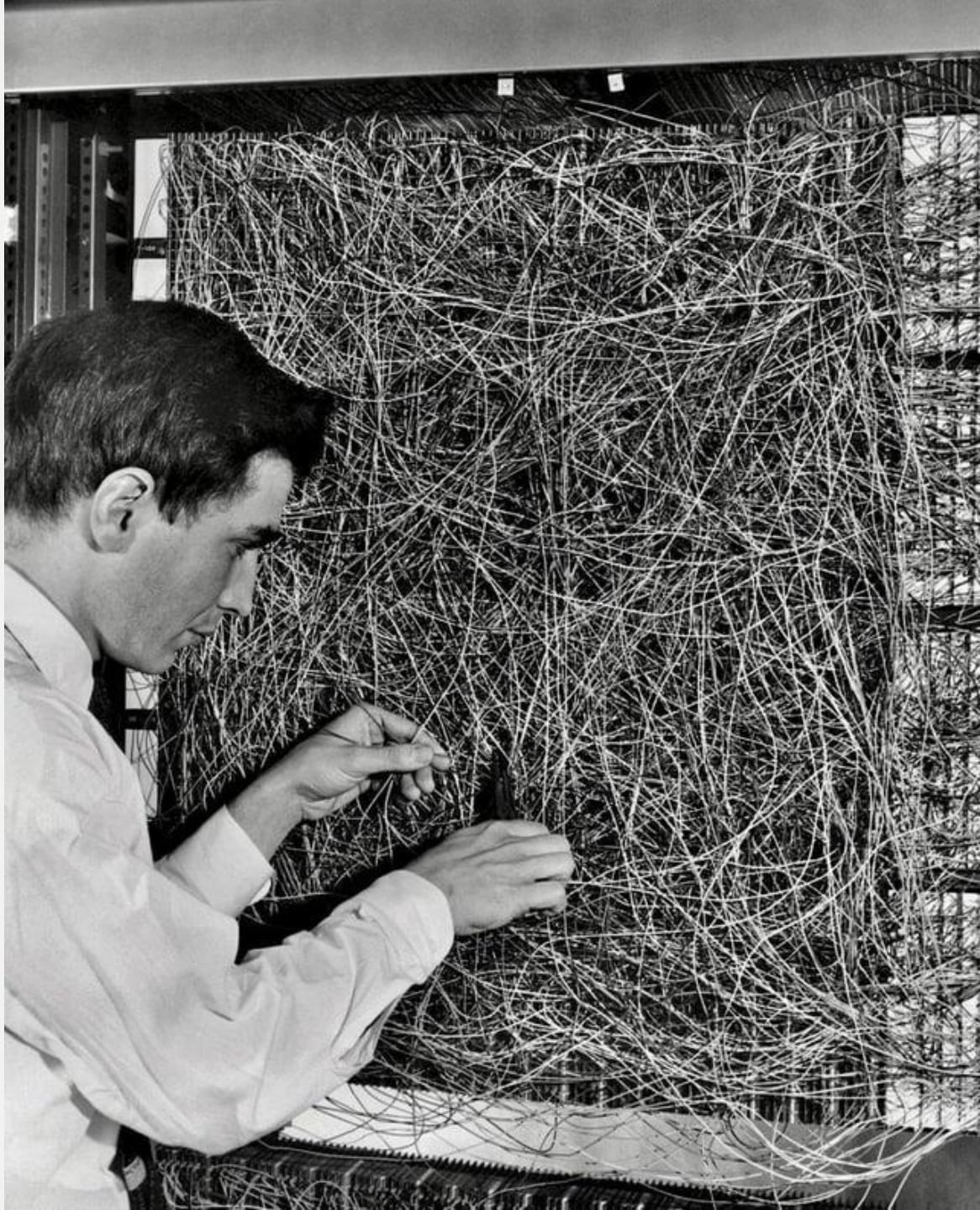


1958 Frank Rosenblatt
Perceptron

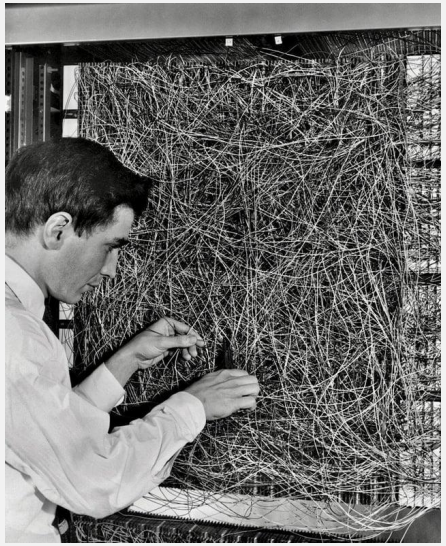


$$o = f\left(\sum_{k=1}^{n} i_k \cdot W_k\right)$$

`sklearn.neural_network`.MLPClassifier



1958 Minsky & Papert Perceptrons



Minsky and Papert write a book called “Perceptrons”

They show that a single layer of these devices was unable to learn some critical Mathematical functions, such as XOR

In the same book, they also show that using multiple layers of these devices would allow these limitations to be addressed

ML Breakthroughs

1986

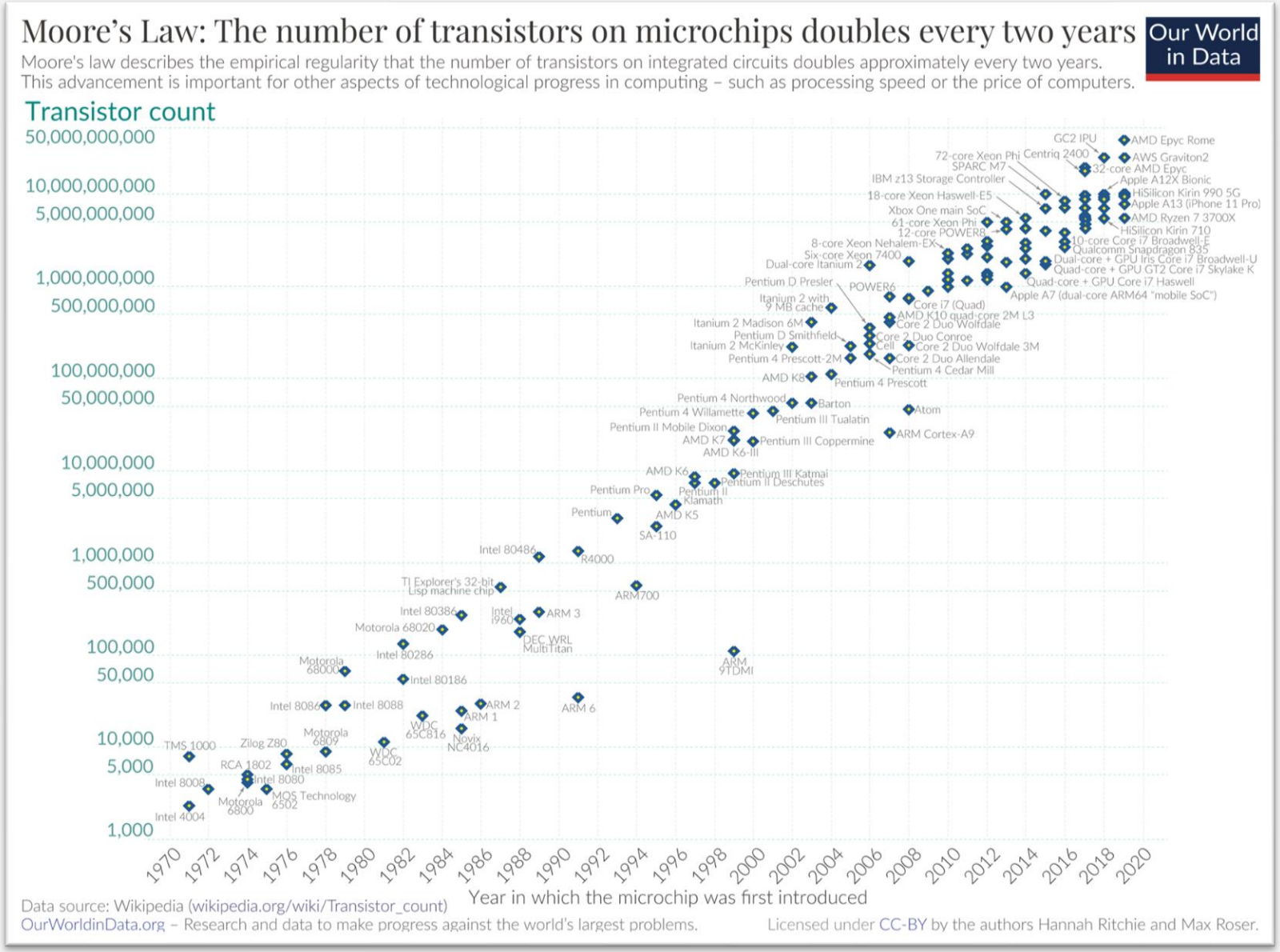
Back-propagation

1989

Universal Approximation Theorem

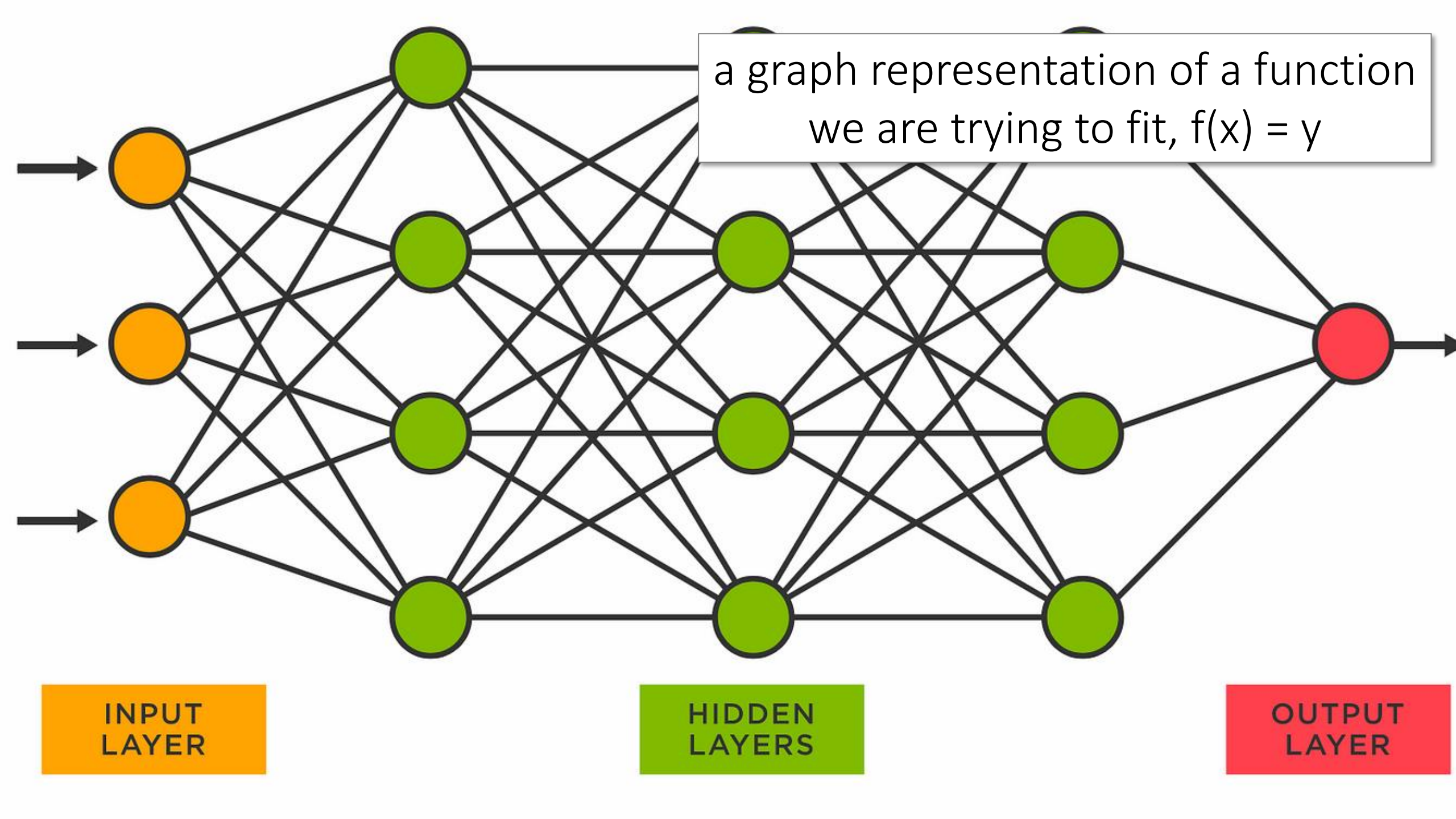
2000+

Moore’s Law

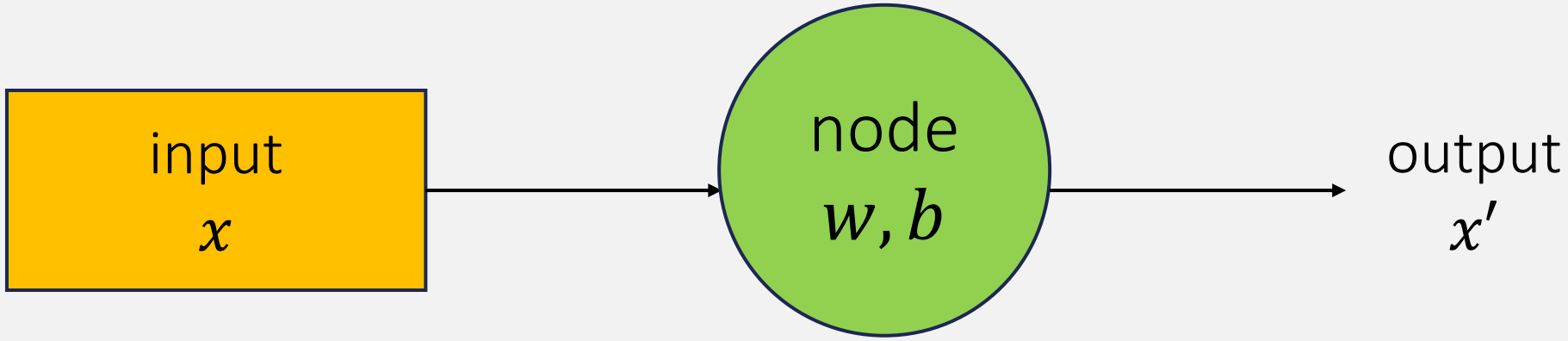


Machine Leaning 101

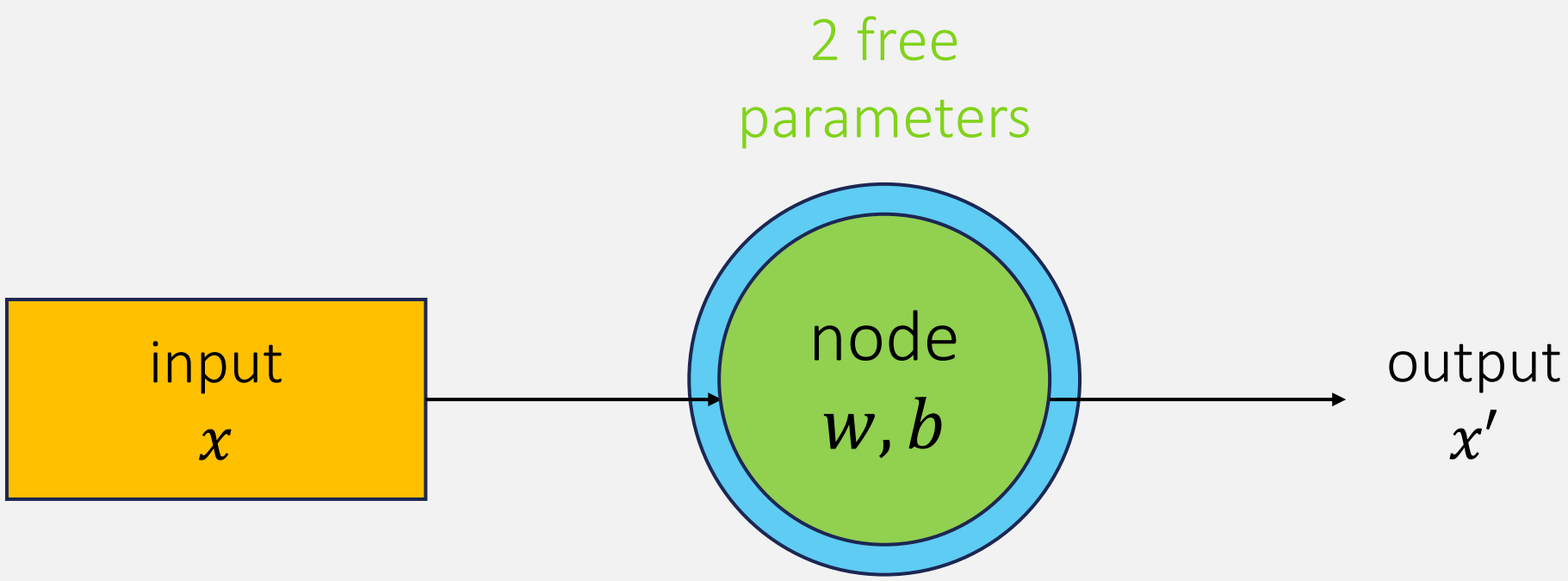
deepish neural nets


a graph representation of a function we are trying to fit, f(x) = y
INPUT LAYER
HIDDEN LAYERS
OUTPUT LAYER

2 free
parameters

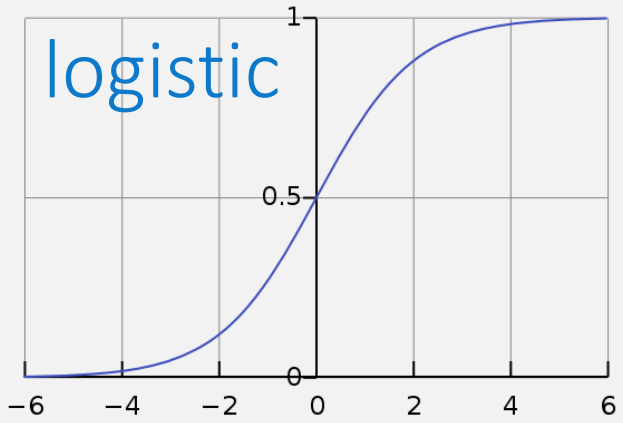


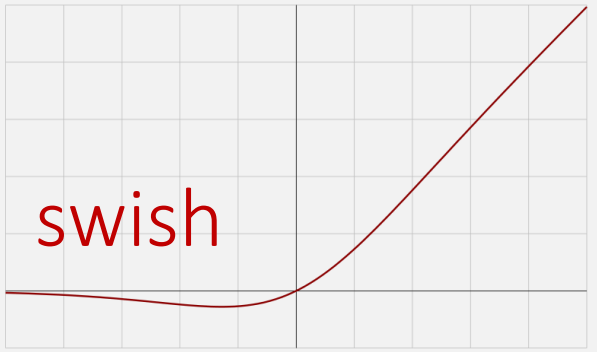
$$x' = wx + b$$



$$x' = f(wx + b)$$

activation function



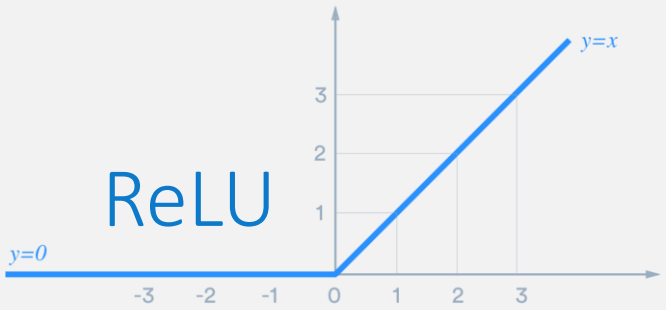


2 free parameters

input
x

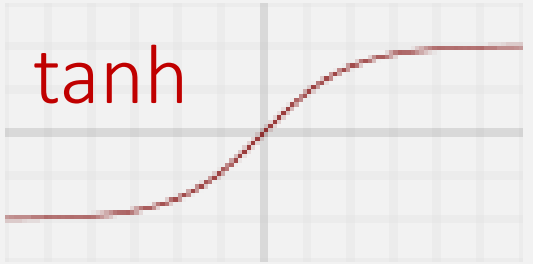
node
w, b

output
x'

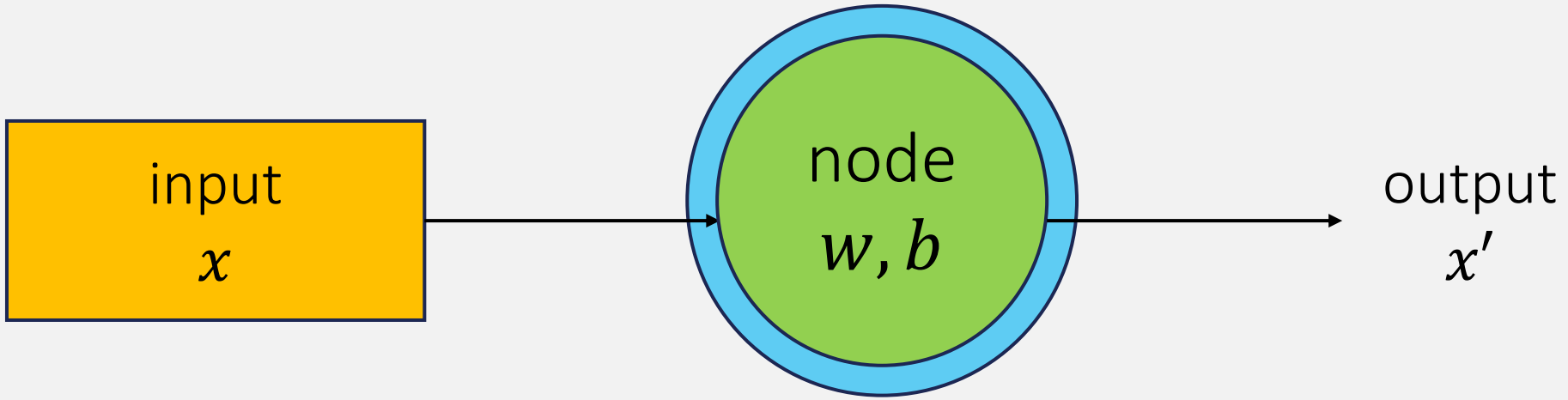


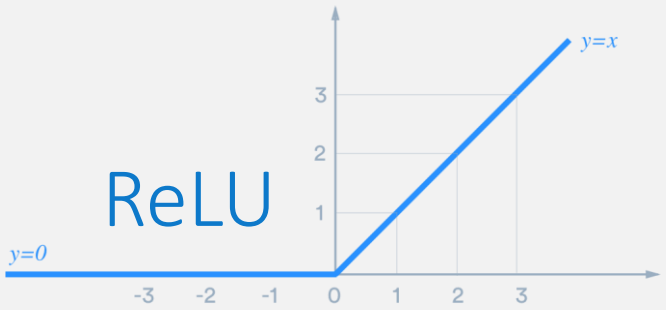
$$x' = f(wx + b)$$

activation function



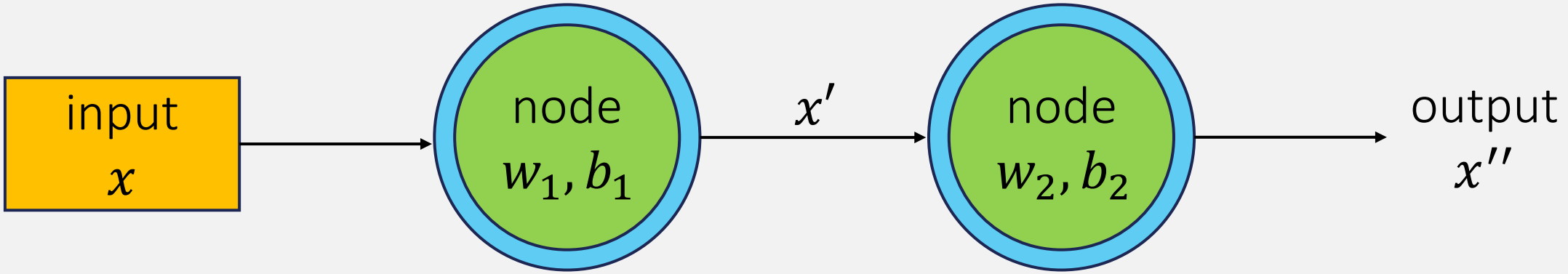
2 free parameters





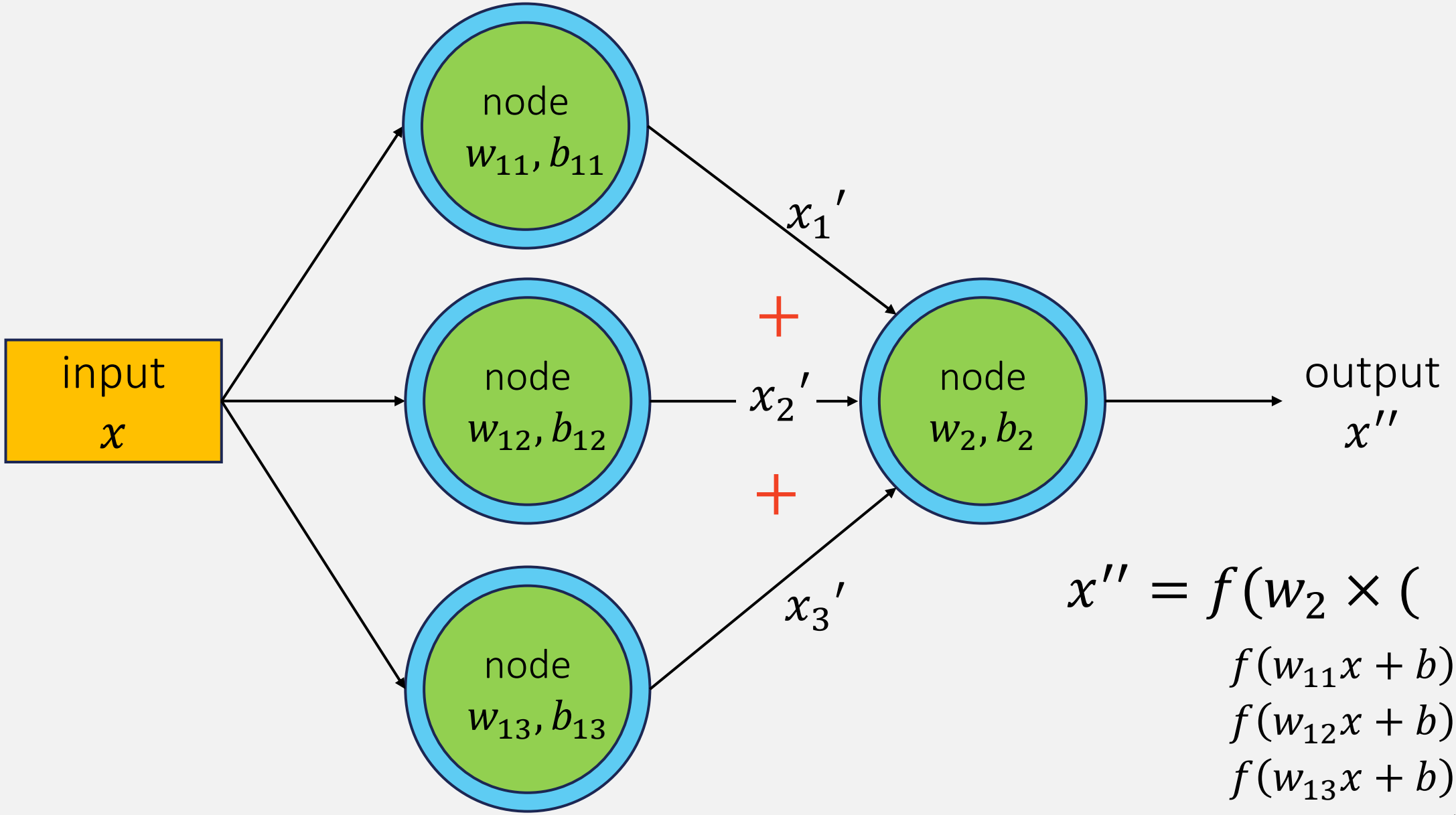
$$x' = f(wx + b)$$

activation function

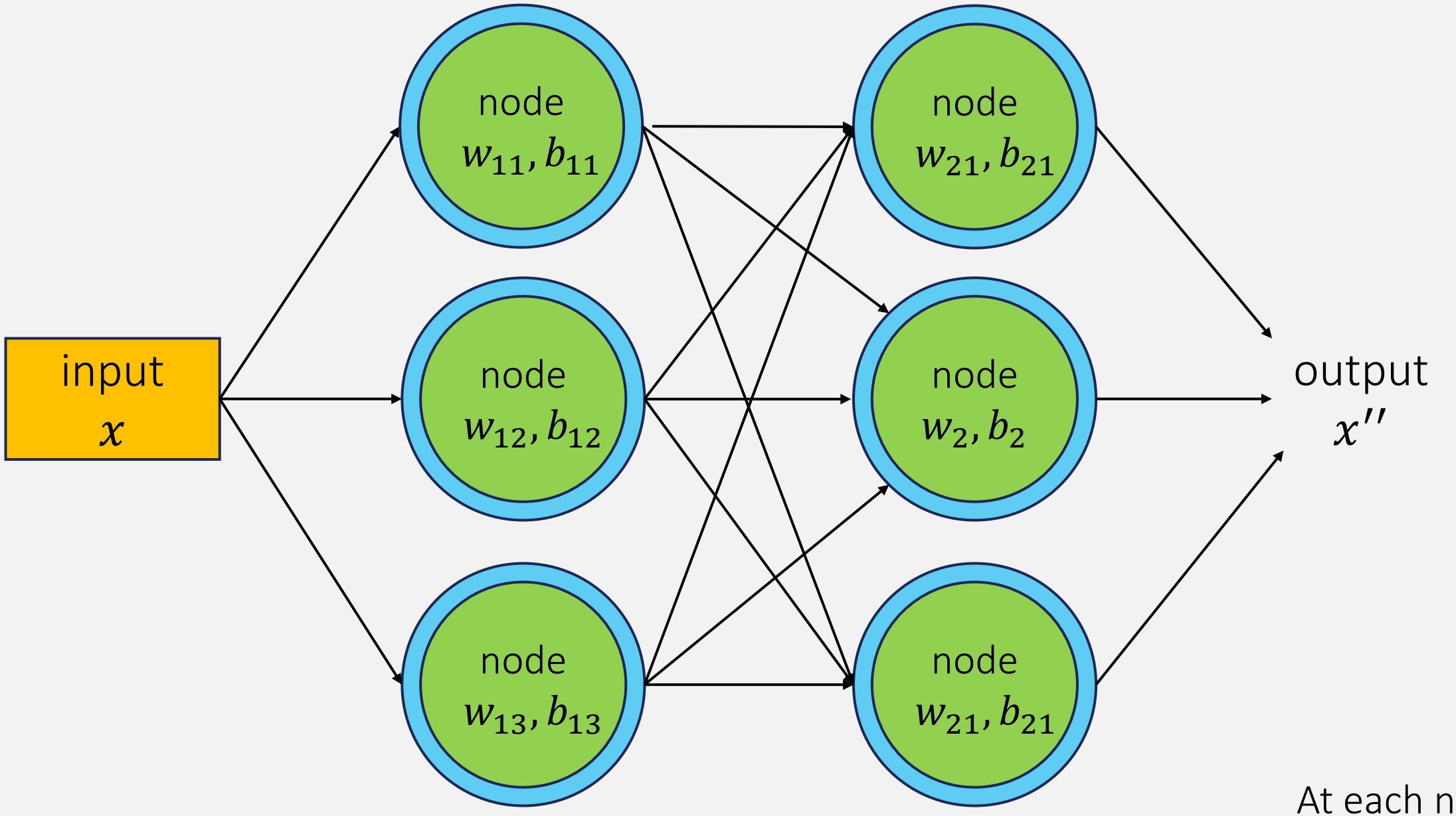


$$x'' = f(w_2 x' + b_2)$$

$$= f(w_2 f(w_1 x + b_1) + b_2)$$

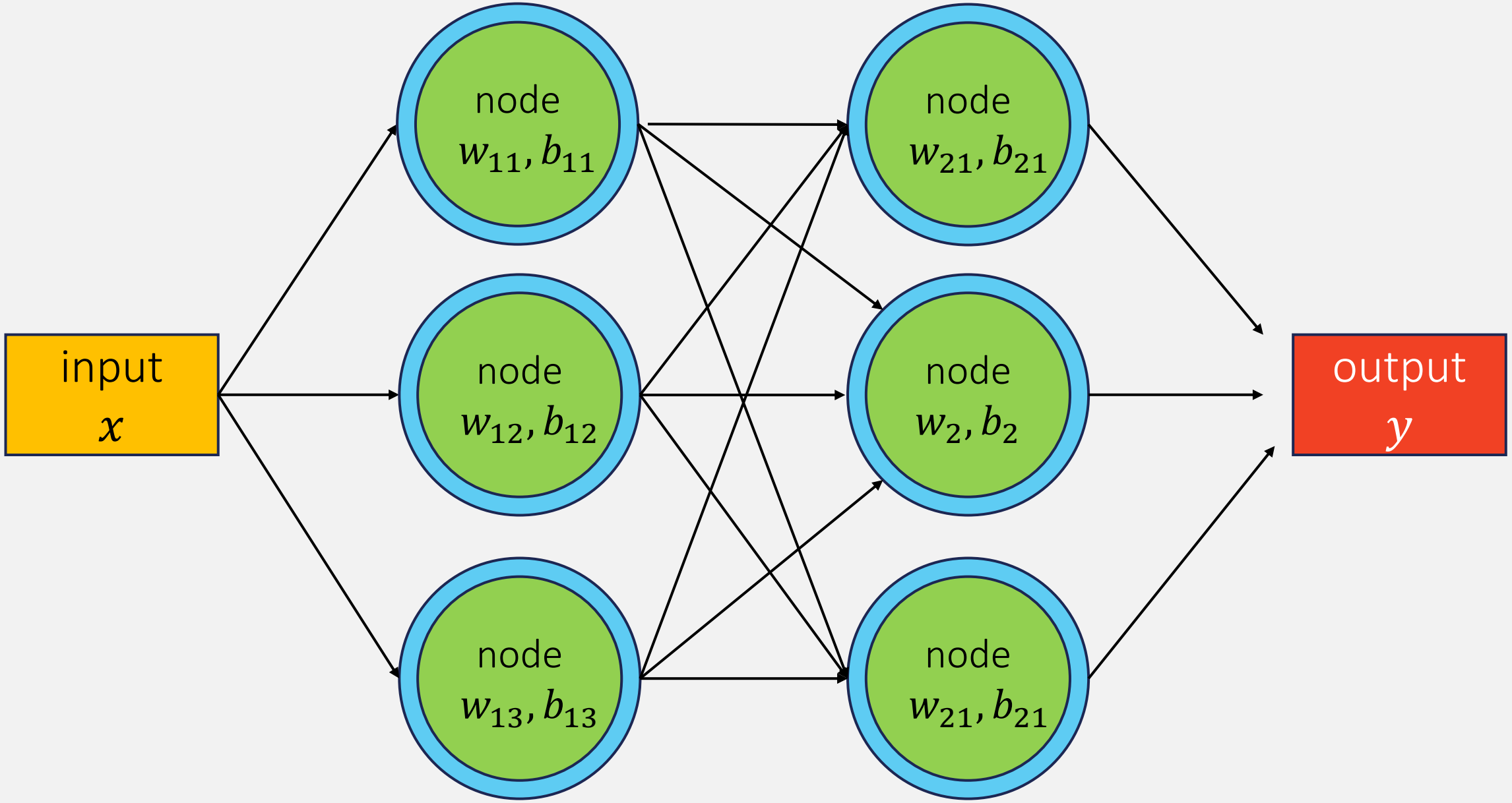


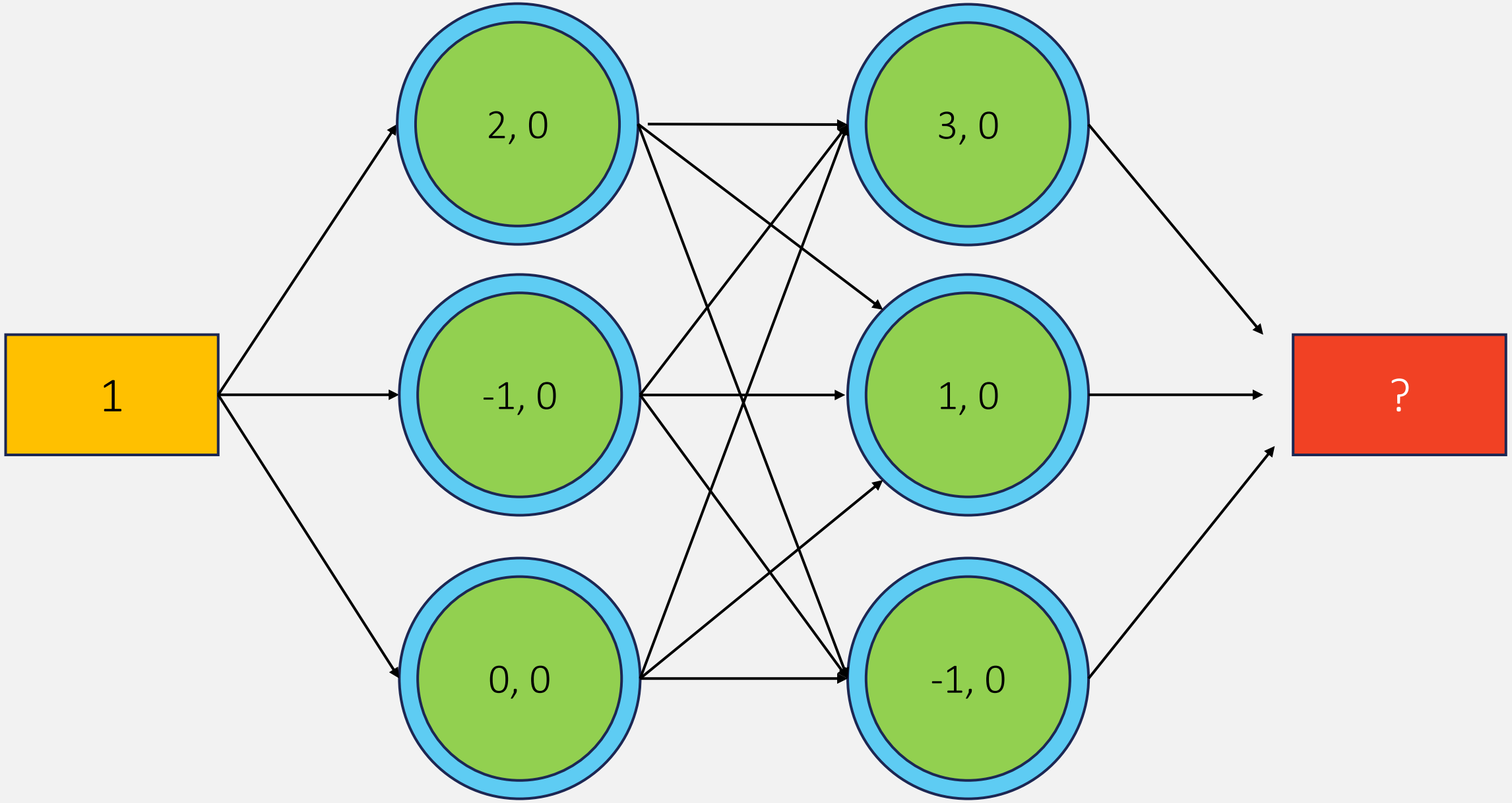
$$x'' = f(w_2 \times (f(w_{11}x + b) + f(w_{12}x + b) + f(w_{13}x + b) +) + b)$$

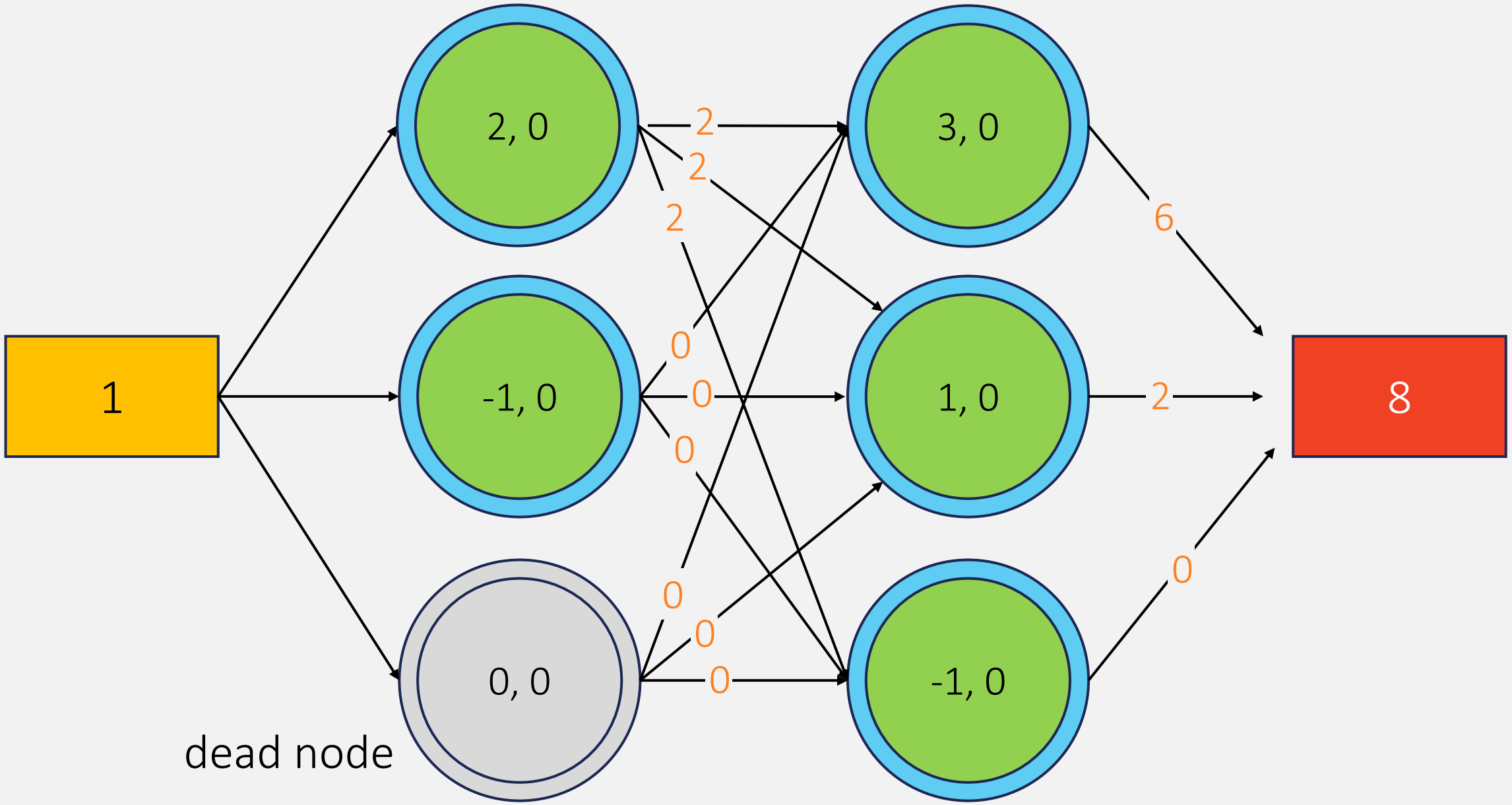


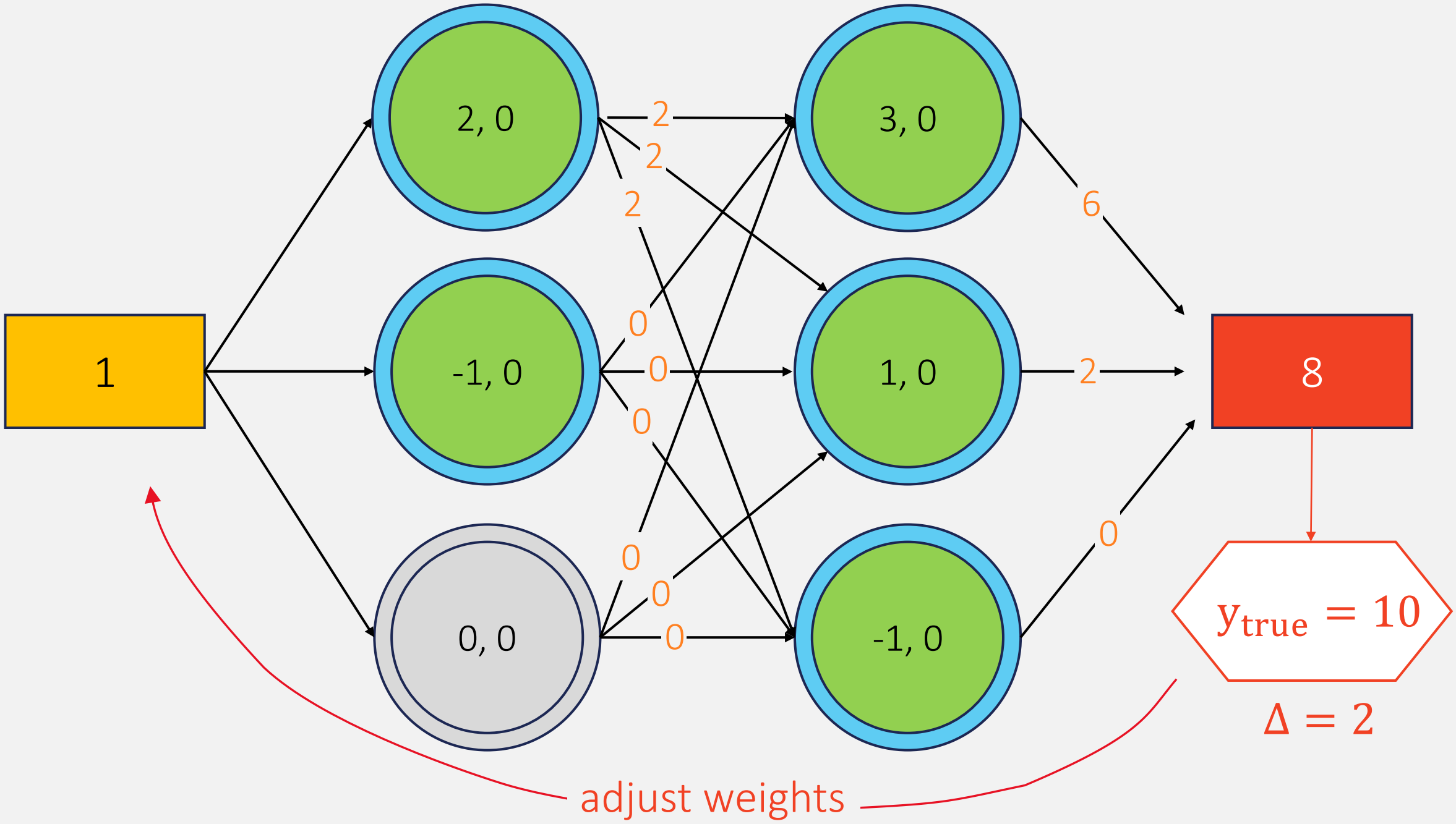
At each node:

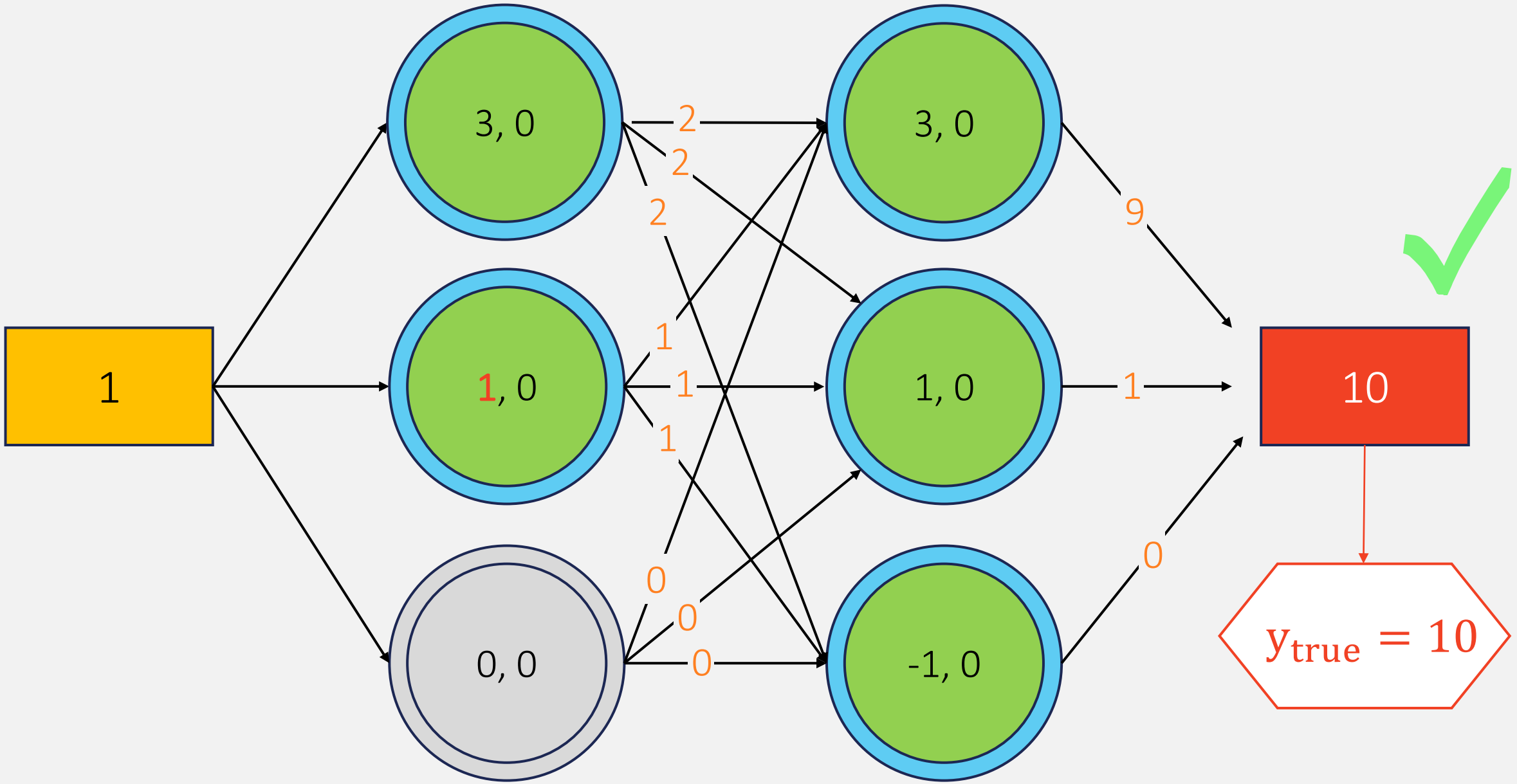
$y = f(wx + b)$

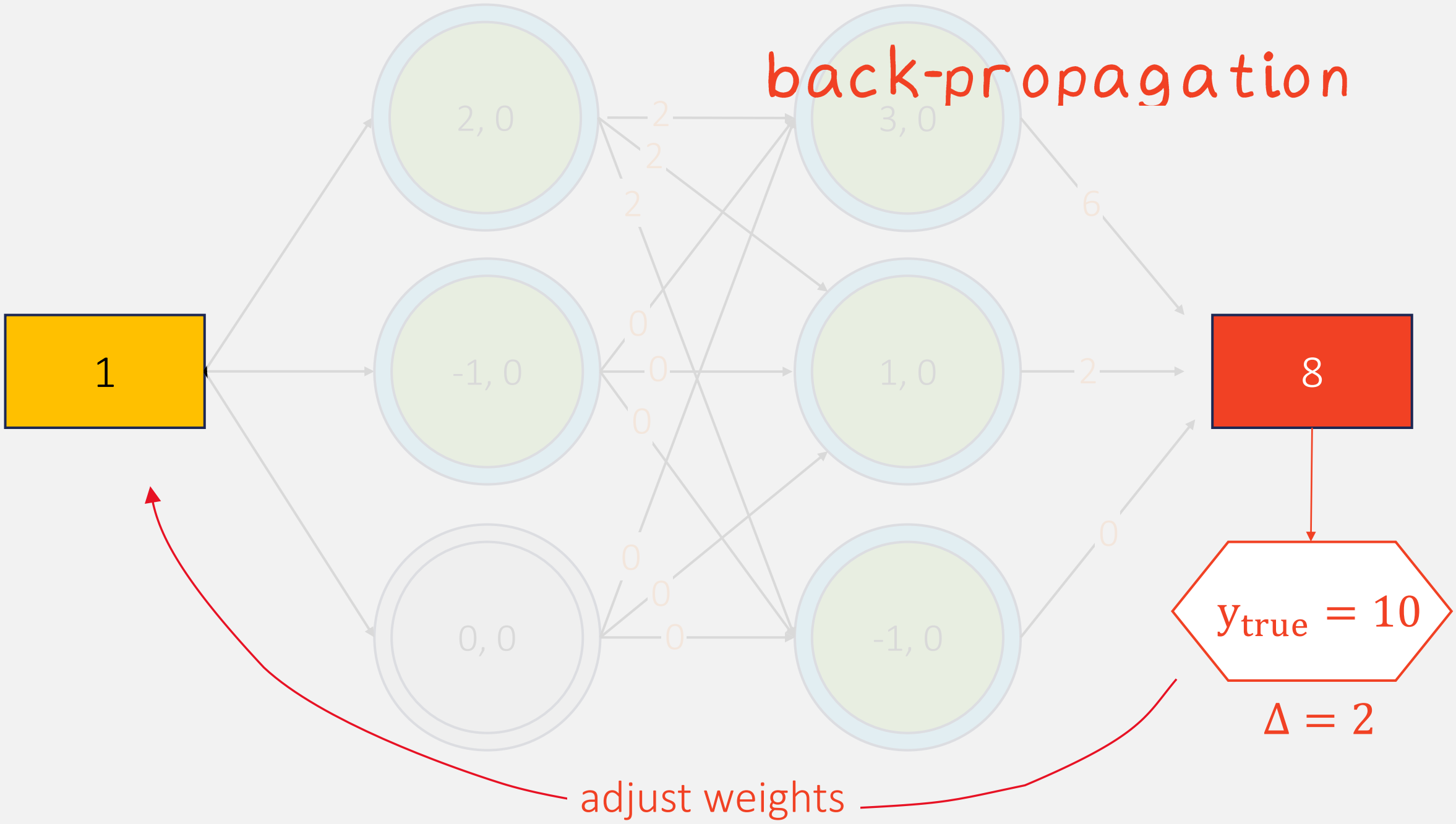

input
x
node
w_{11}, b_{11}
node
w_{12}, b_{12}
node
w_{13}, b_{13}
node
w_{21}, b_{21}
node
w_2, b_2
node
w_{21}, b_{21}
output
y


1
2, 0
-1, 0
0, 0
3, 0
1, 0
-1, 0
?

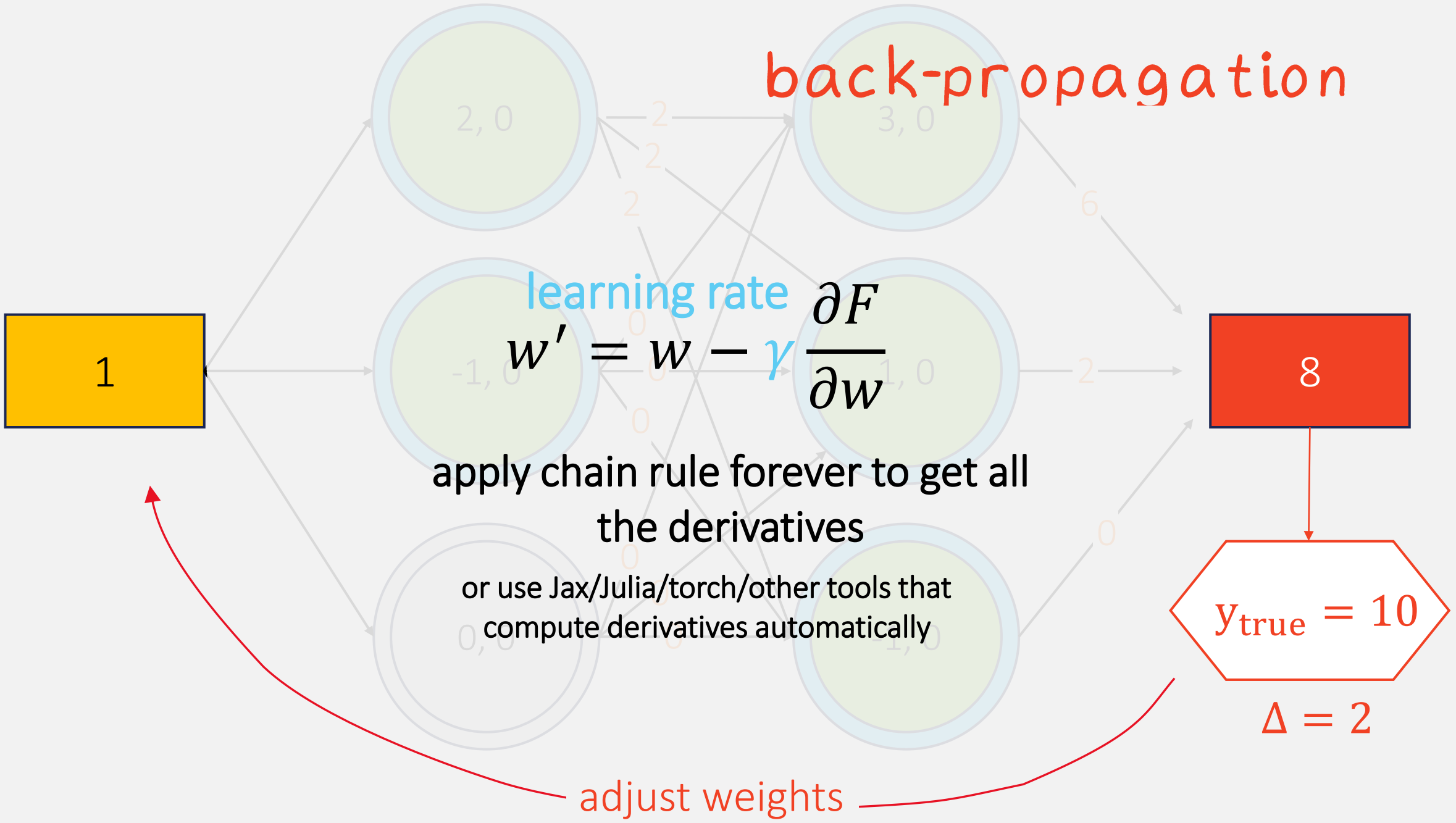

1
2, 0
-1, 0
0, 0
3, 0
1, 0
-1, 0
8
2
2
2
0
0
0
0
0
0
6
2
0
dead node


1
2, 0
-1, 0
0, 0
3, 0
1, 0
-1, 0
8
2
2
2
0
0
0
0
0
0
6
2
0
$y_{true} = 10$
$\Delta = 2$
adjust weights


1
3, 0
1, 0
0, 0
3, 0
1, 0
-1, 0
10
2
2
2
1
1
1
0
0
0
9
1
0
$y_{true} = 10$


back-propagation
1
2, 0
-1, 0
0, 0
3, 0
1, 0
-1, 0
8
2
2
2
0
0
0
0
0
0
6
2
0
$y_{true} = 10$
$\Delta = 2$
adjust weights

back-propagation



Different optimization choices

- Stochastic gradient descent
- Learning rate optimization (Adam, Adamax, …)

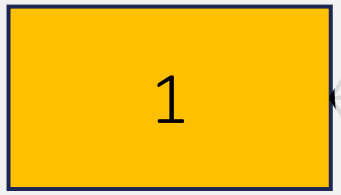
back-propagation

learning rate

$$w' = w - \gamma \frac{\partial F}{\partial w}$$

apply chain rule forever to get all the derivatives

or use Jax/Julia/torch/other tools that compute derivatives automatically



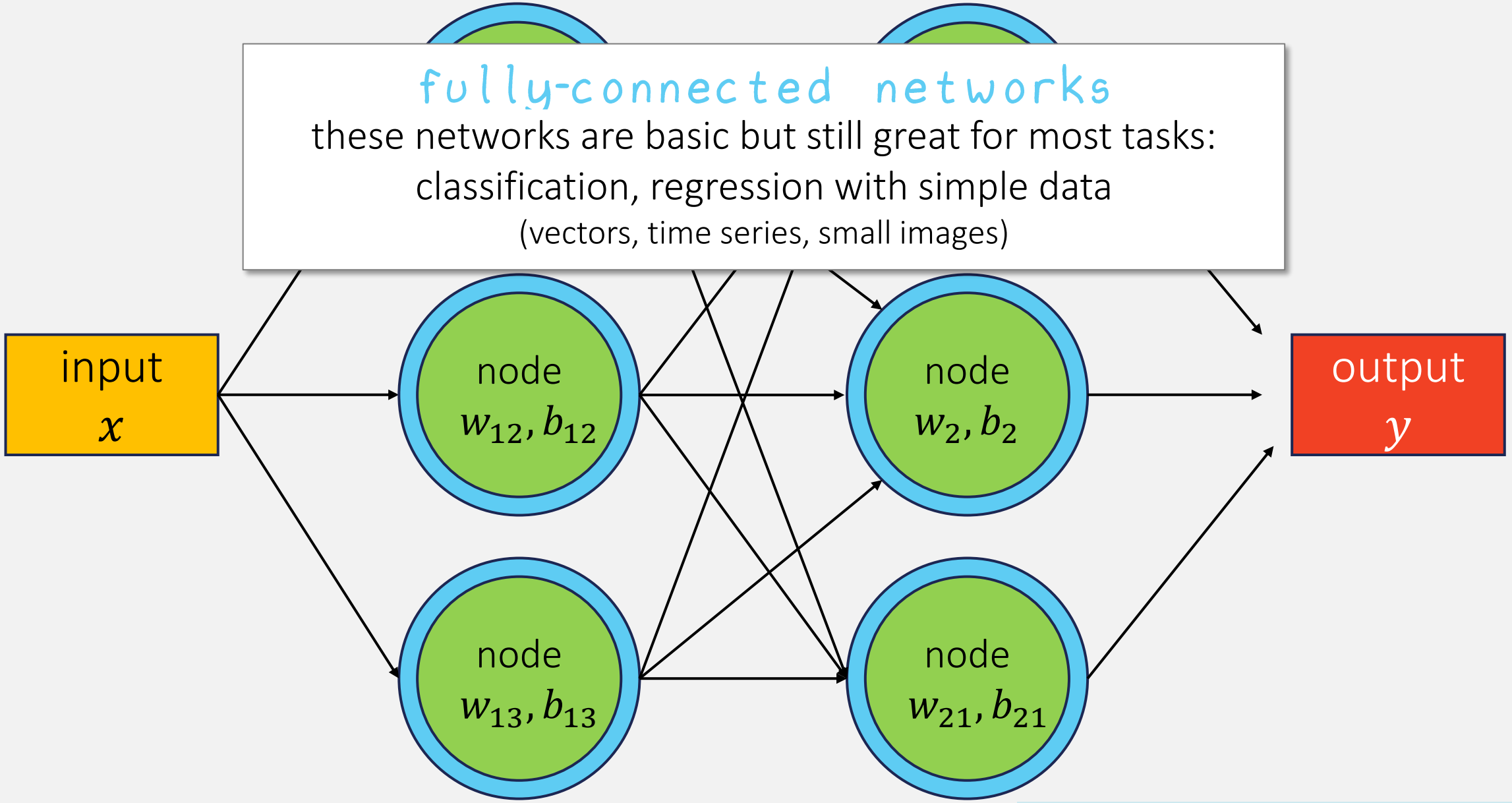
$y_{\text{true}} = 10$

$\Delta = 2$

adjust weights

fully-connected networks

these networks are basic but still great for most tasks:
classification, regression with simple data
(vectors, time series, small images)



`sklearn.neural_network`.MLPClassifier

`sklearn.neural_network`.MLPRegressor

fully-connected networks

these networks are basic but still great for most tasks:
classification, regression with simple data
(vectors, time series, small images)

pros

Few ($\mathcal{O}(100s)$) free parameters
Fast: no need for a GPU
Easy to implement
Still capture a lot of complexity
Easy to adapt for different tasks
Can be pre-trained

cons

Not as powerful as complex NNs
Still prone to overfitting

`sklearn.neural_network`.MLPClassifier
`sklearn.neural_network`.MLPRegressor

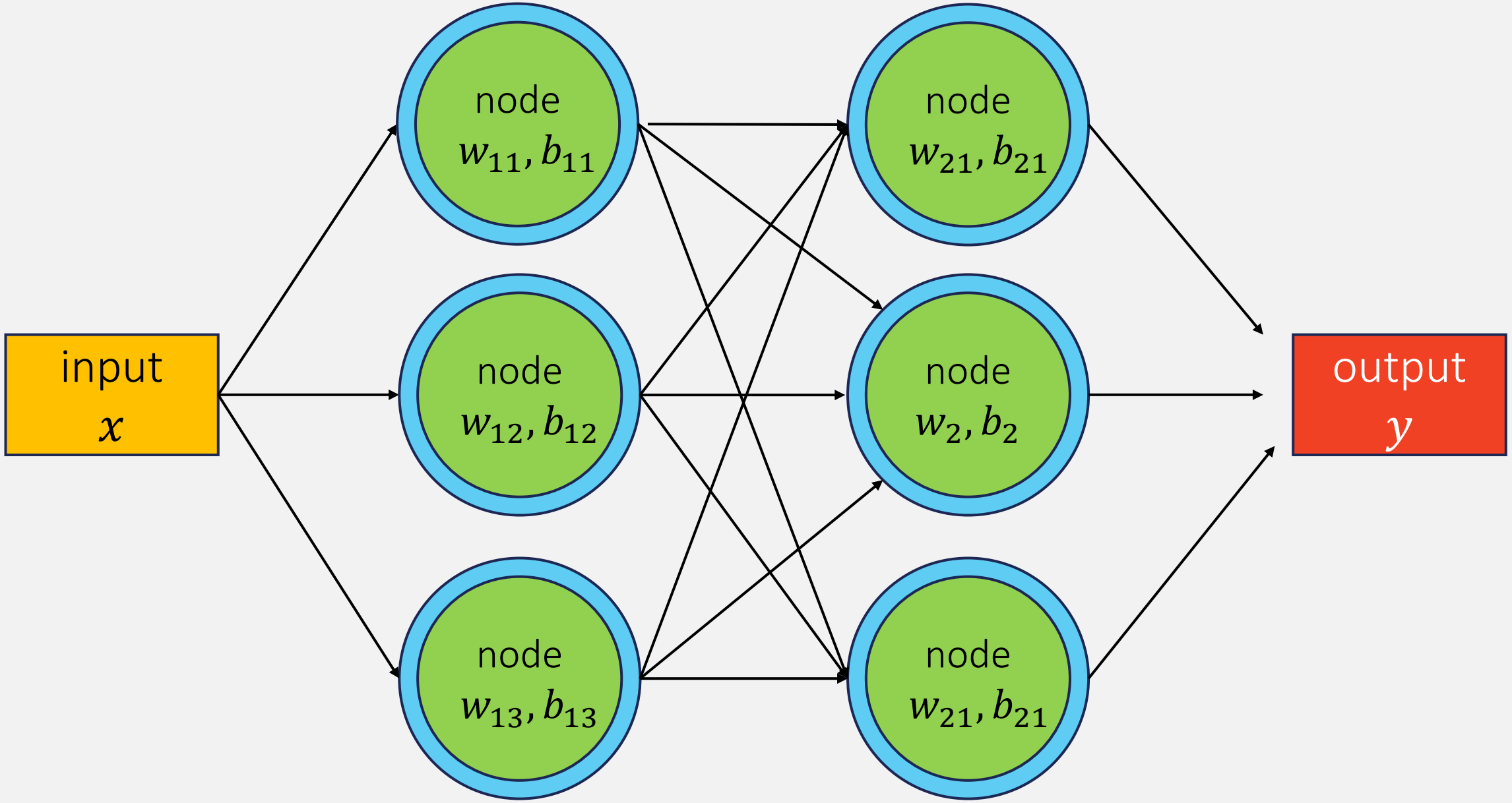
overfitting

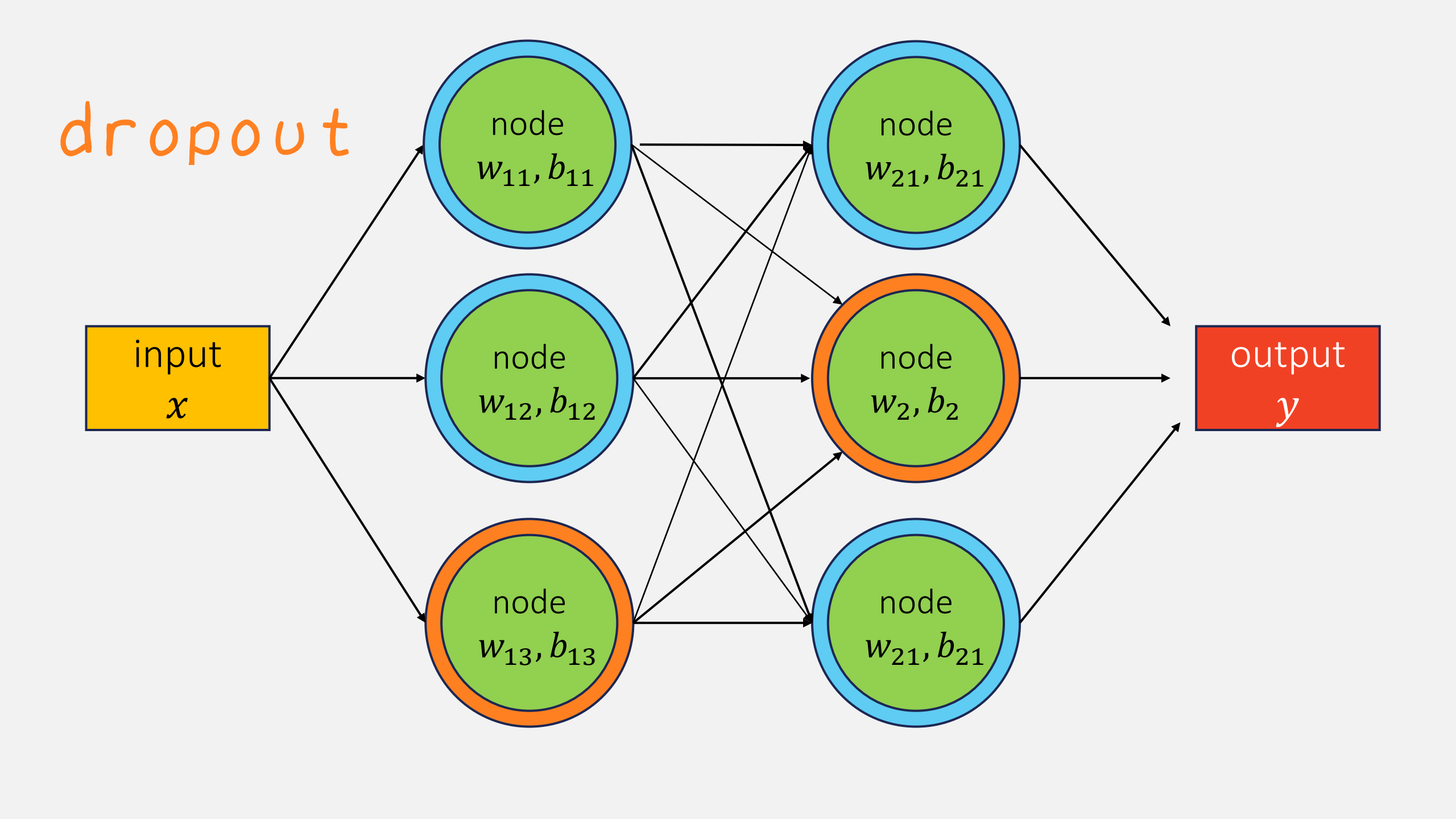
NNs have 100s to $\mathcal{O}(10^6)$ free parameters – overfitting can be a real problem

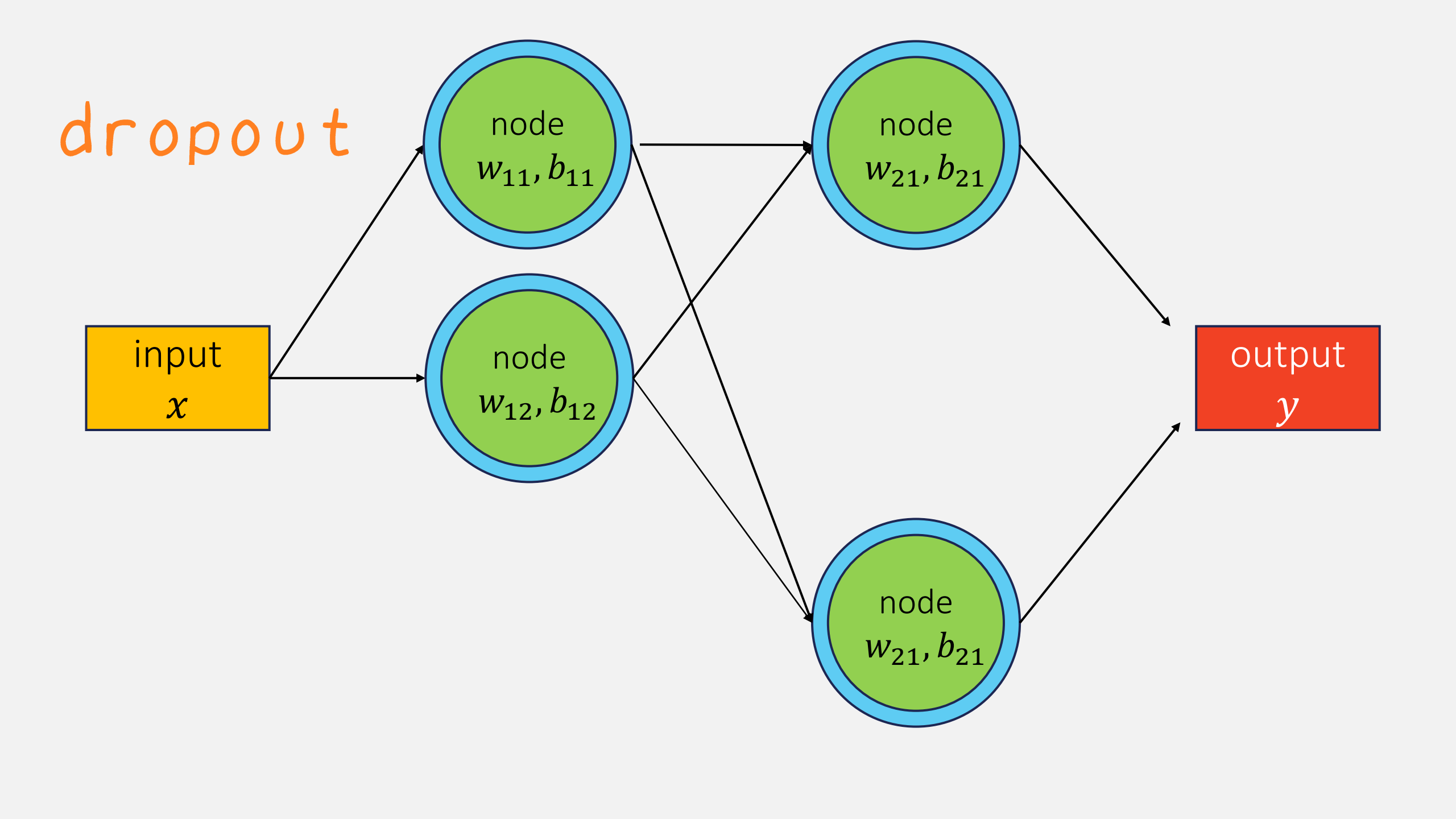
1. Always keep your train/test sets separate
   Easy to mix them up if data becomes more complex or is generated on the fly

2. Neural networks train in batches
   Take a subsample → run inference → calculate loss → adjust weights
   Sometimes a subsample is just one input vector!

3. Dropout
   Adds stochasticity to the network


input
x
node
w_{11}, b_{11}
node
w_{12}, b_{12}
node
w_{13}, b_{13}
node
w_{21}, b_{21}
node
w_2, b_2
node
w_{21}, b_{21}
output
y


dropout
input
x
node
w_{11}, b_{11}
node
w_{12}, b_{12}
node
w_{13}, b_{13}
node
w_{21}, b_{21}
node
w_2, b_2
node
w_{21}, b_{21}
output
y


dropout
node
w_{11}, b_{11}
node
w_{21}, b_{21}
input
x
node
w_{12}, b_{12}
output
y
node
w_{21}, b_{21}

dropout

1. Assign probability of dropout, $p(\text{drop}) \in (0,1)$ to the nodes you want
   Usually this will range between 0.1 and 0.5. Higher values = more generalized, longer to train

2. At each training step (each batch), randomly drop nodes
   Different nodes will be dropped at different iterations

3. The model should generalize better
   As different nodes won't learn to overcompensate each other

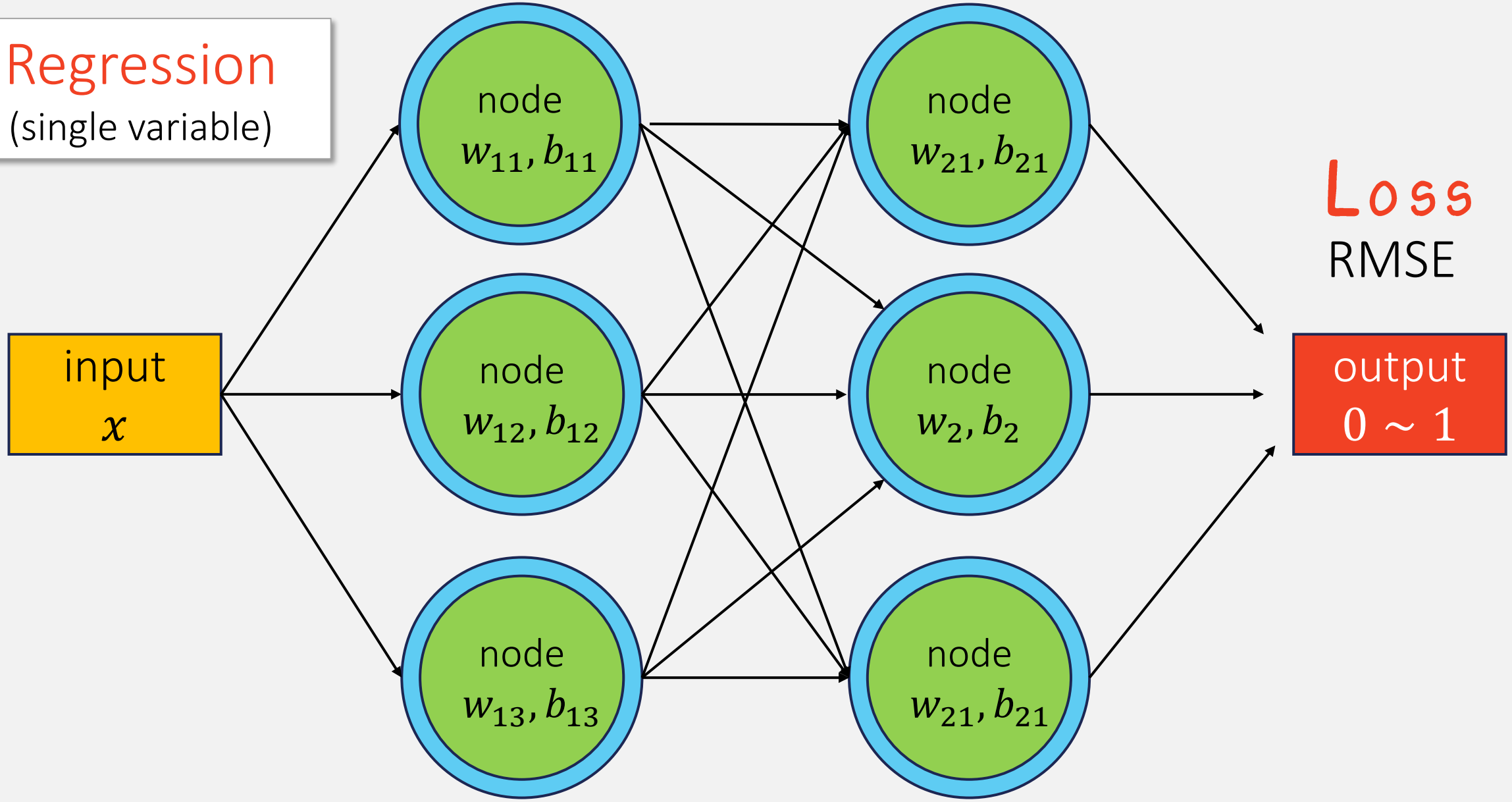
4. Turn dropout off when running inference
   Produces consistent results

NN use cases

what the network learns to do depends mostly on the labels, the output layer, and the loss function

Regression

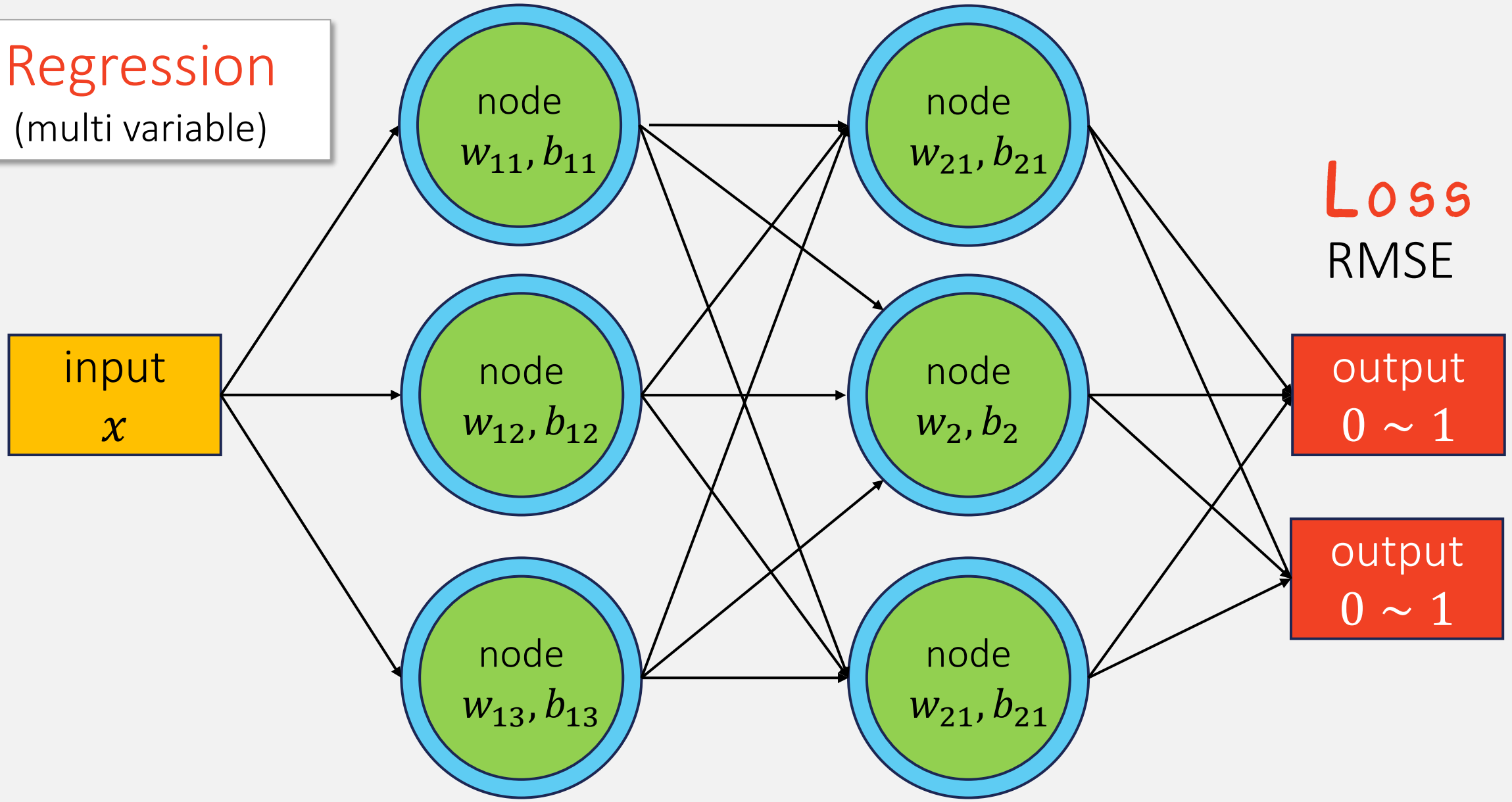
(single variable)



Loss

RMSE

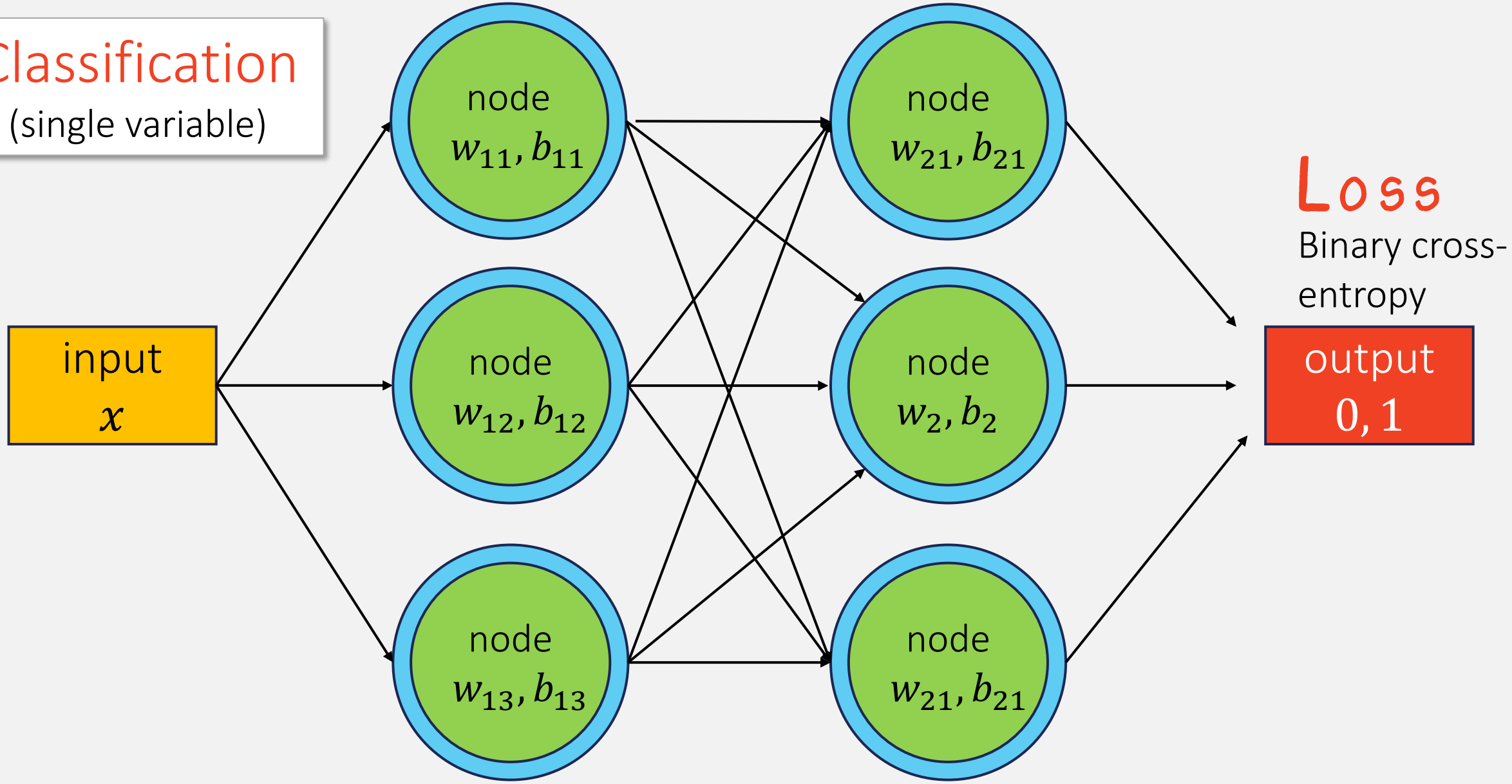
Regression
(multi variable)



Loss
RMSE

Classification

(single variable)

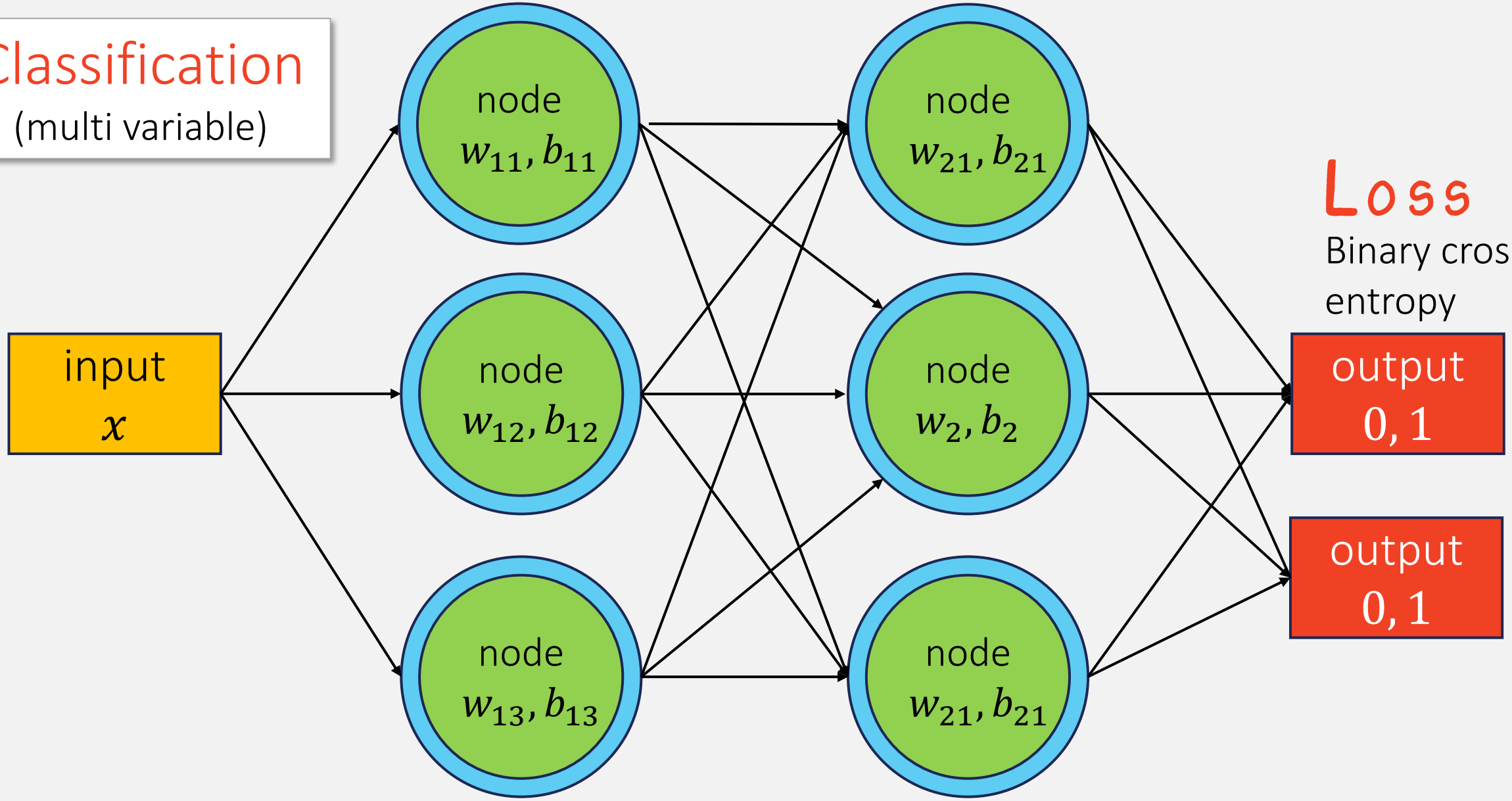


Loss

Binary cross-entropy

Classification

(multi variable)



Loss

Binary cross-entropy

NN use cases

what the network learns to do depends mostly on the labels, the output layer, and the loss function

+ Easy to adapt the same model for different tasks

+ No need to re-train much if the task is similar (fine-tuning)

Other things to know about

1

Normalization

Your input data should be normalized!!! Networks perform **best** if the data is in (0,1) range
Everything is additive: if some values cap at 1 and some at 10^5, the 10^5 ones will dominate
Learning rate: the same rate used for all nodes, so it can't change based on data amplitude

2

Balanced training sets

Still important – networks will not learn about data it hasn't seen, or seen little of

3

Uncertainty estimation

How to get the error on the prediction?

Other things to know about

1

Normalization

Your input data should be normalized!!! Networks perform **best** if the data is in (0,1) range
Everything is additive: if some values cap at 1 and some at 10^5, the 10^5 ones will dominate
Learning rate: the same rate used for all nodes, so it can't change based on data amplitude

2

Balanced training sets

Still important – networks will not learn about data it hasn't seen, or seen little of

3

Uncertainty estimation

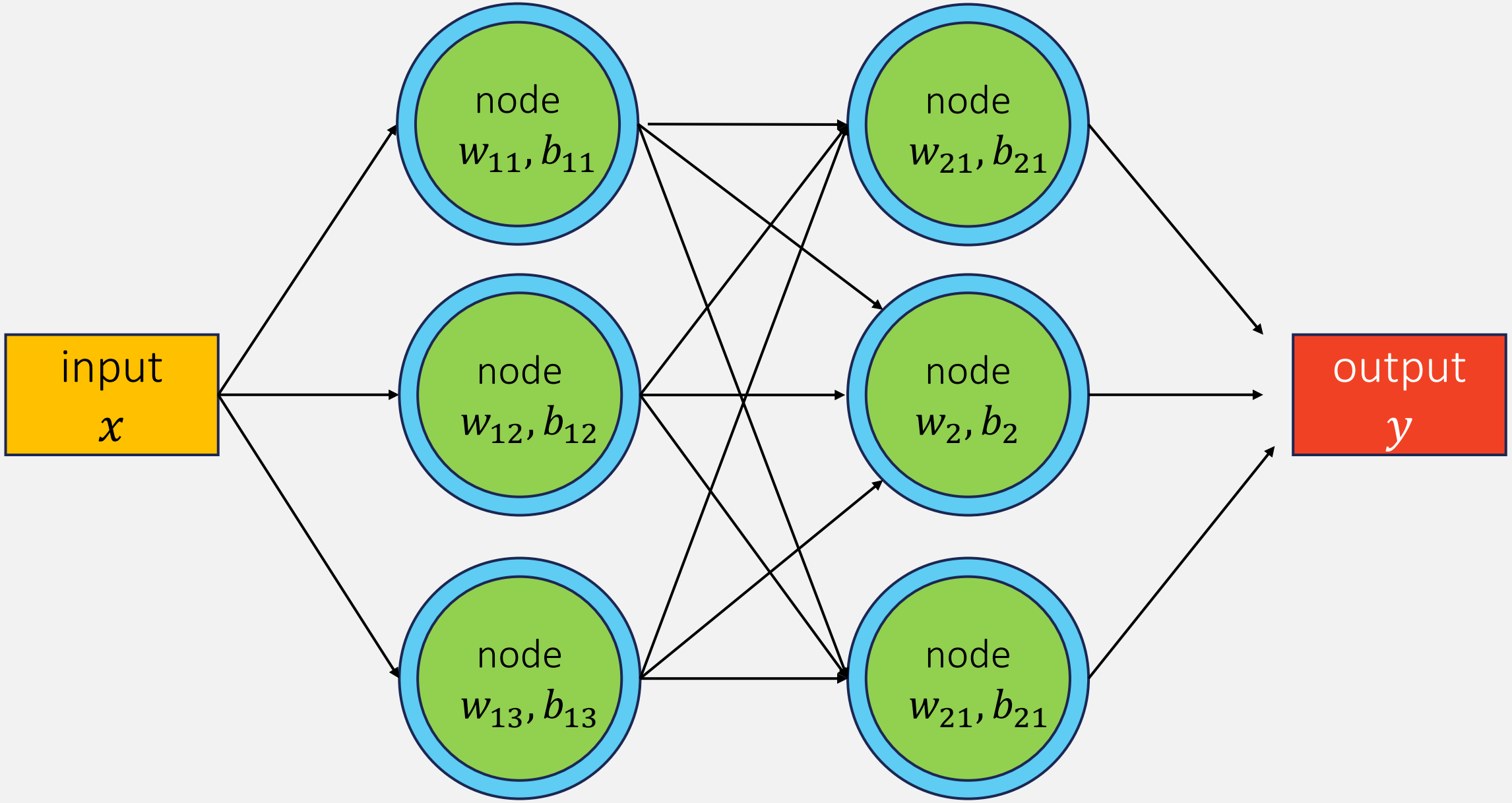
How to get the error on the prediction?

Uncertain input data: resample within uncertainties
Dropout: turn the dropout on during inference!

} re-run the prediction N times to get uncertainties

Machine Leaning 201

Convolutional neural nets


input
x
node
w_{11}, b_{11}
node
w_{12}, b_{12}
node
w_{13}, b_{13}
node
w_{21}, b_{21}
node
w_2, b_2
node
w_{21}, b_{21}
output
y

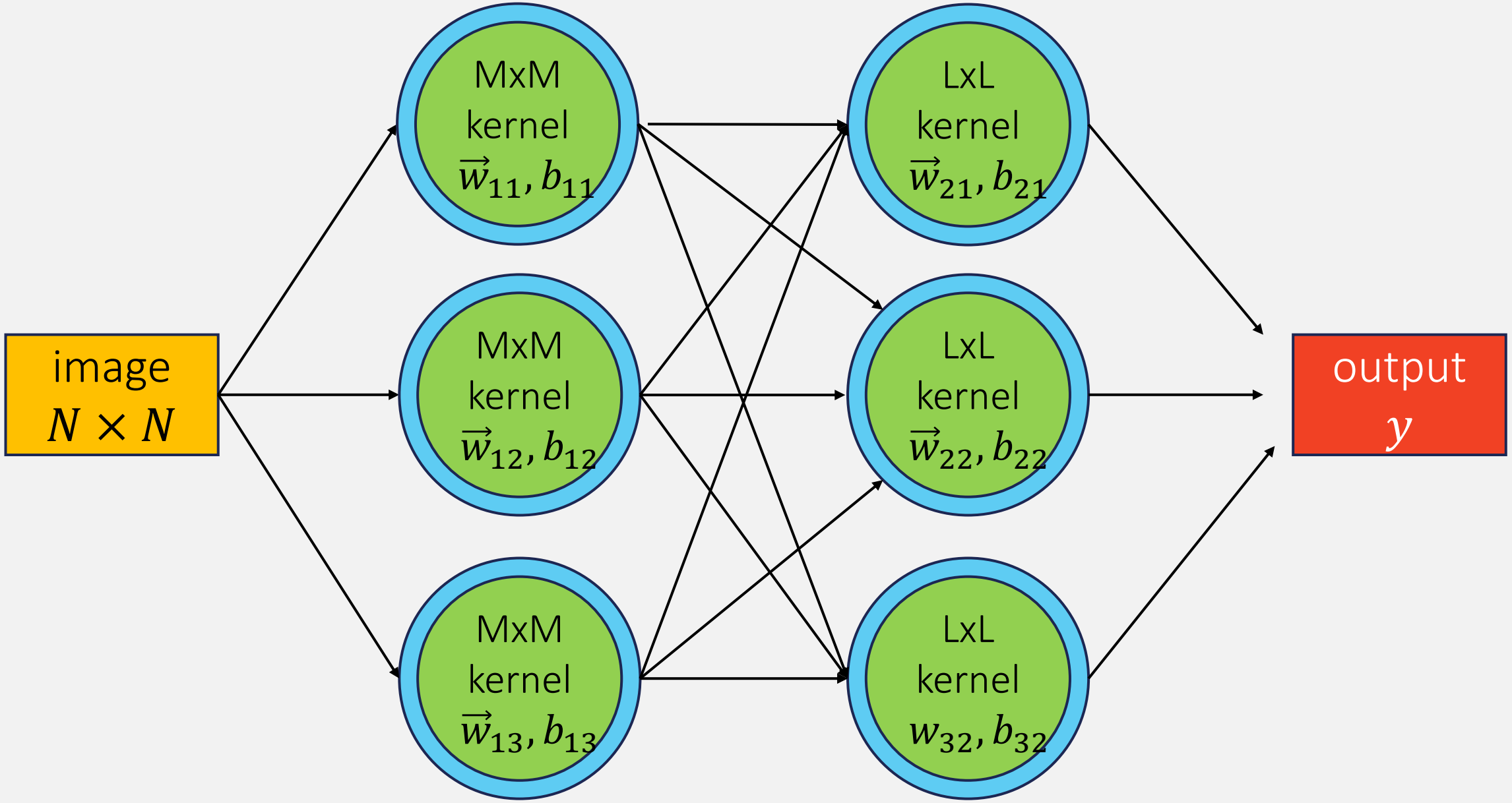


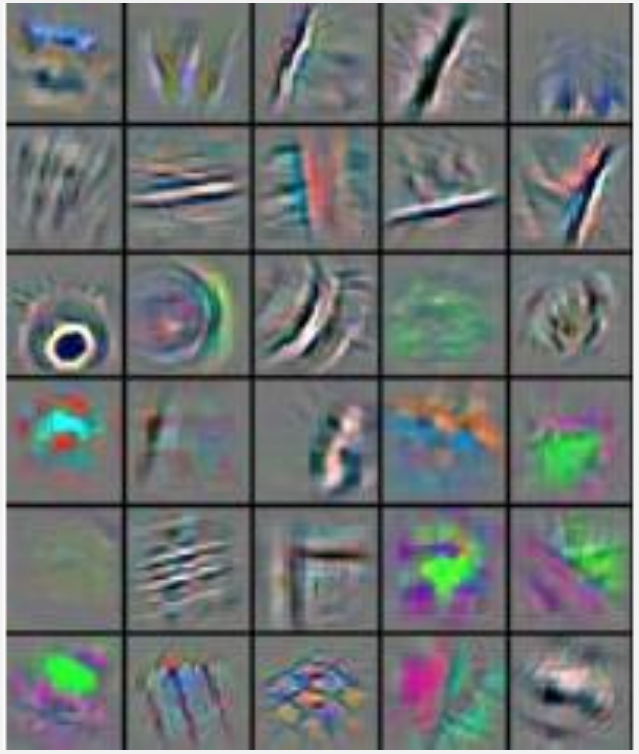
image
$N \times N$
MxM
kernel
$\vec{w}_{11}, b_{11}$
MxM
kernel
$\vec{w}_{12}, b_{12}$
MxM
kernel
$\vec{w}_{13}, b_{13}$
LxL
kernel
$\vec{w}_{21}, b_{21}$
LxL
kernel
$\vec{w}_{22}, b_{22}$
LxL
kernel
w_{32}, b_{32}
output
y

Convolution kernels

Simpler patterns

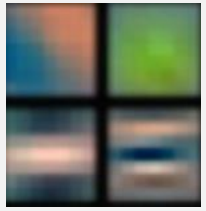
Complex patterns

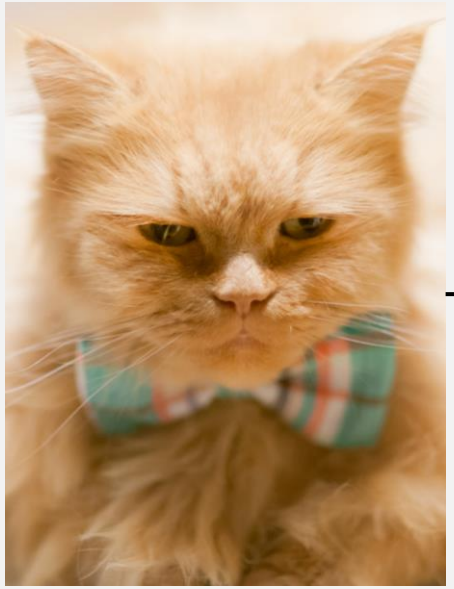
Layer 1

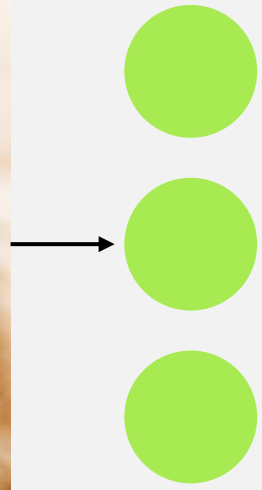


Layer 2

Layer 3







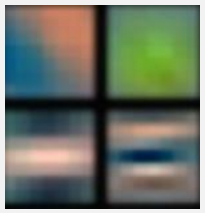
Layer 1

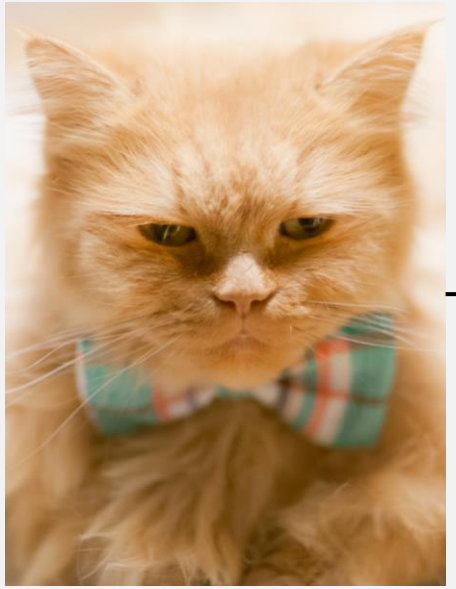
5x5 kernels

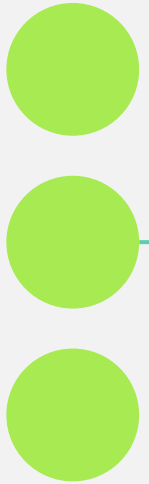
re-sample the image to lower resolution by some factor (2)

maxpooling: take the max. of each 2x2 slice

avgpooling: take the mean

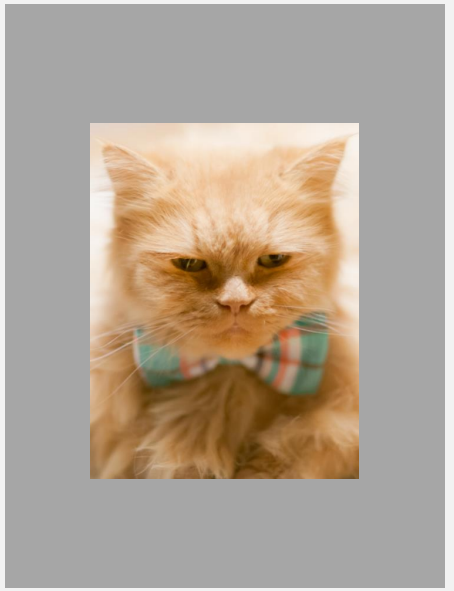






maxpooling

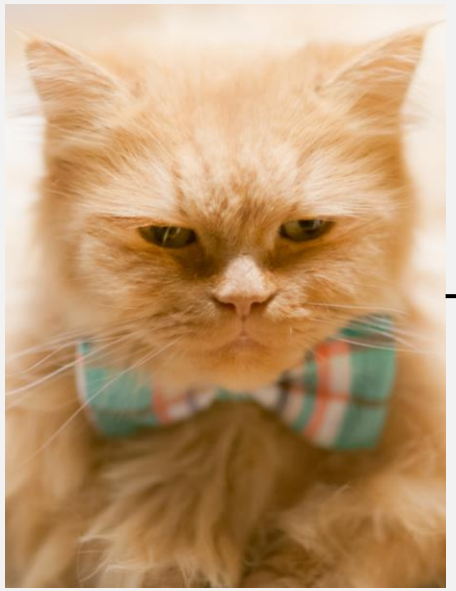
or [other]pooling

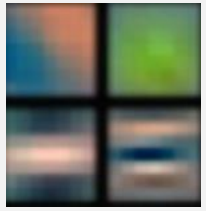


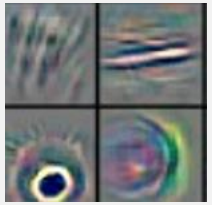
Layer 1

5x5 kernels

then pad it with 0s to keep the shape the same







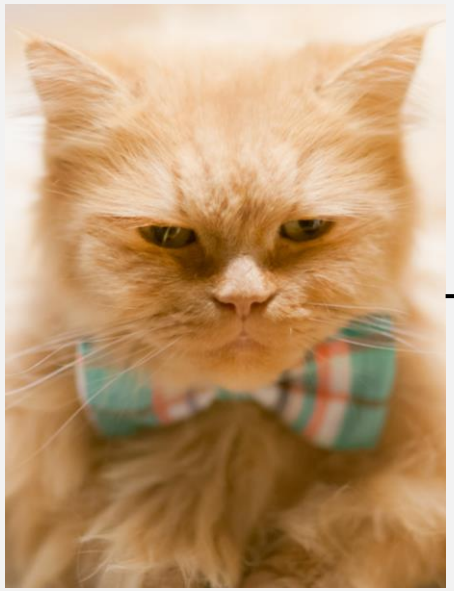
maxpool

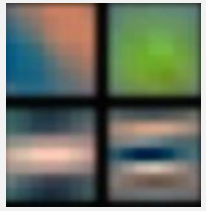
Layer 1

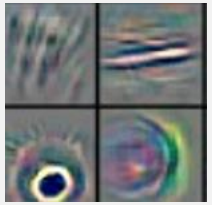
5x5 kernels

Layer 2

5x5 kernels





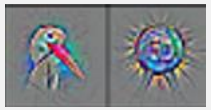


maxpool

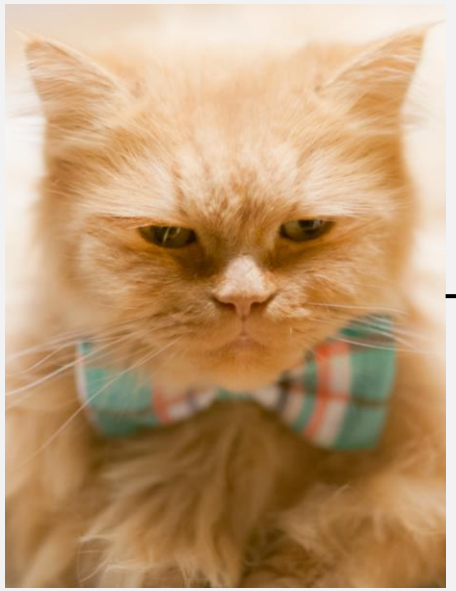
maxpool

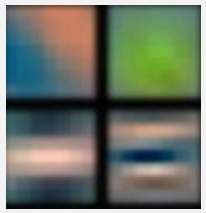
Layer 1
5x5 kernels

Layer 2
5x5 kernels

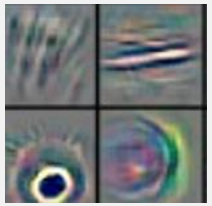


Layer 3
5x5 kernels





maxpool



maxpool

flatten

100s of nodes each

usually dropout is applied here

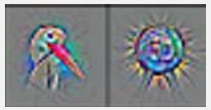
But you can add dropout to the conv layers too

Layer 1

5x5 kernels

Layer 2

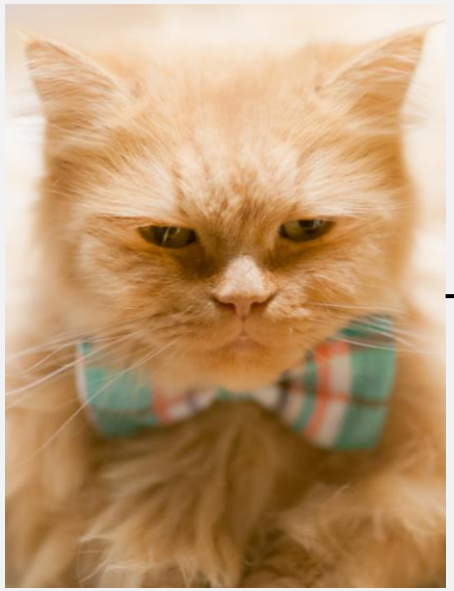
5x5 kernels

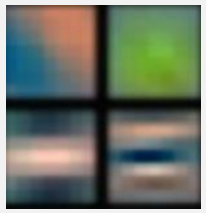


Layer 3

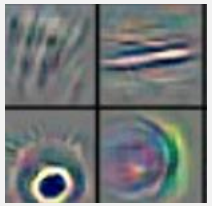
5x5 kernels

Fully connected layers





maxpool



maxpool

flatten

100s of nodes each

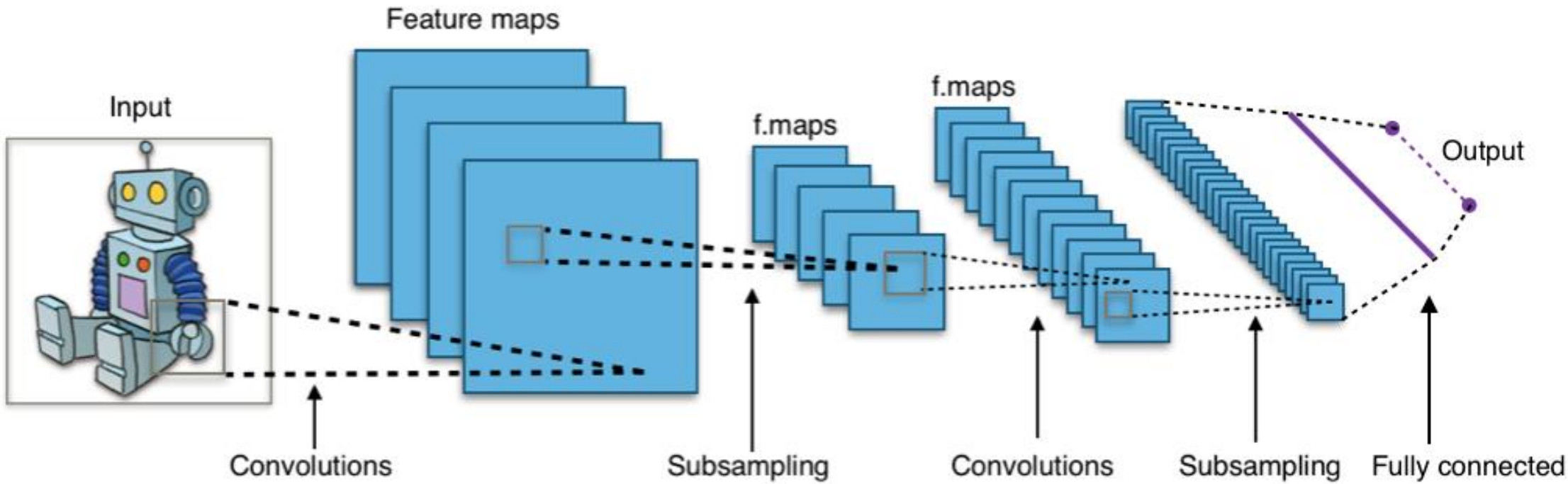
Output 0,1

Layer 1
5x5 kernels

Layer 2
5x5 kernels

Layer 3
5x5 kernels

Fully connected layers


Input
Feature maps
f.maps
f.maps
Output
Convolutions
Subsampling
Convolutions
Subsampling
Fully connected

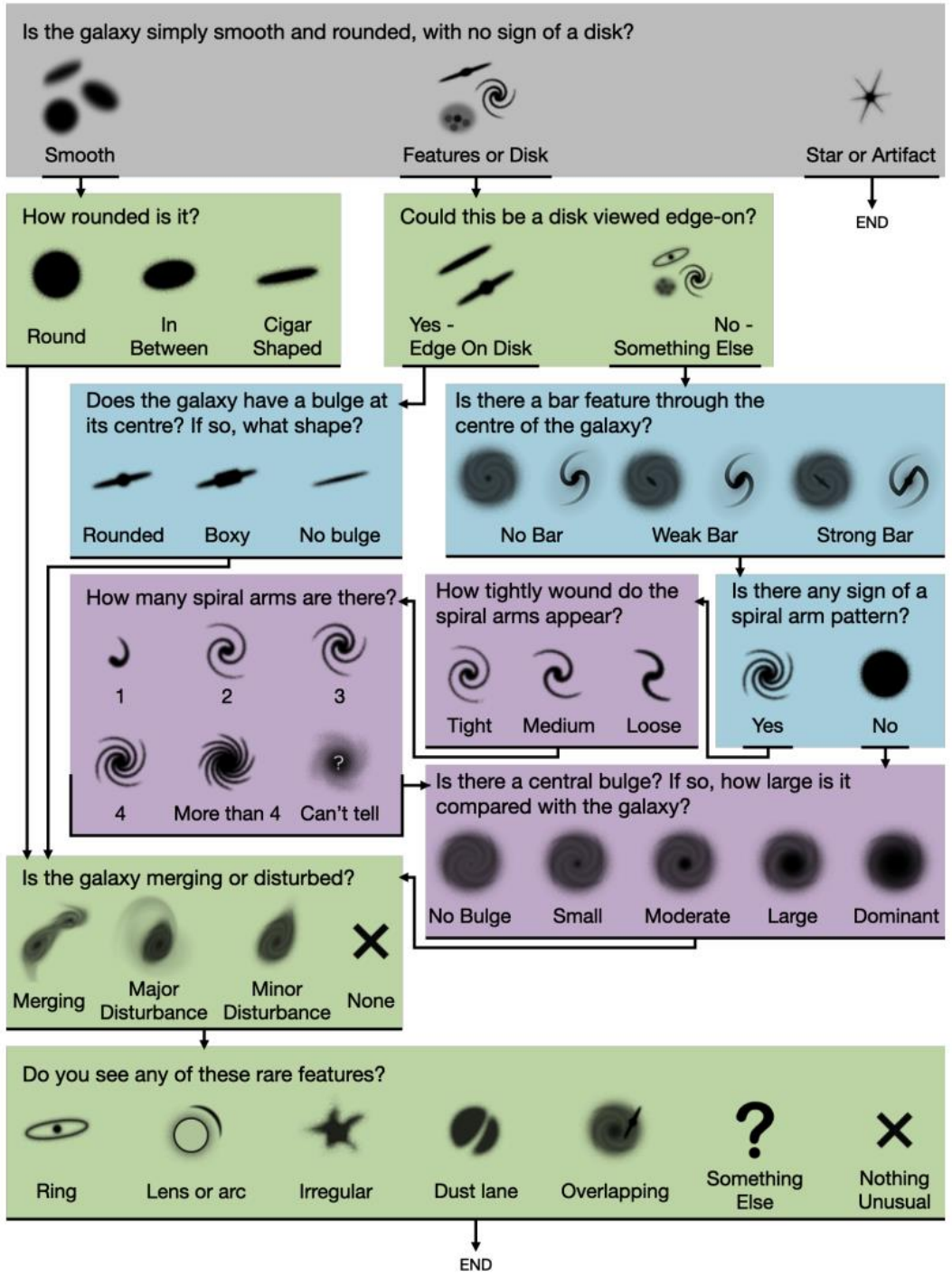
Galaxy Zoo

20 citizen scientists classify each galaxy, giving p(smooth)

TASK | TUTORIAL

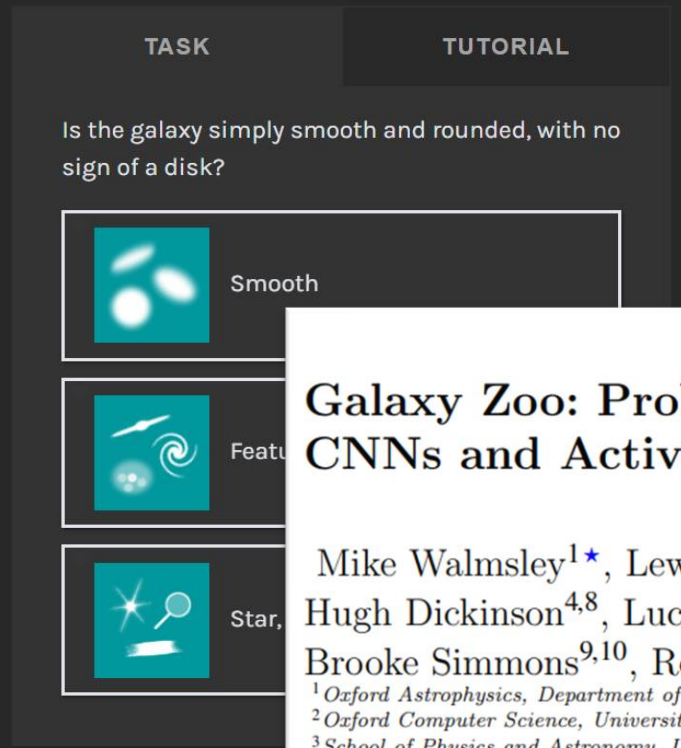
Is the galaxy simply smooth and rounded, with no sign of a disk?

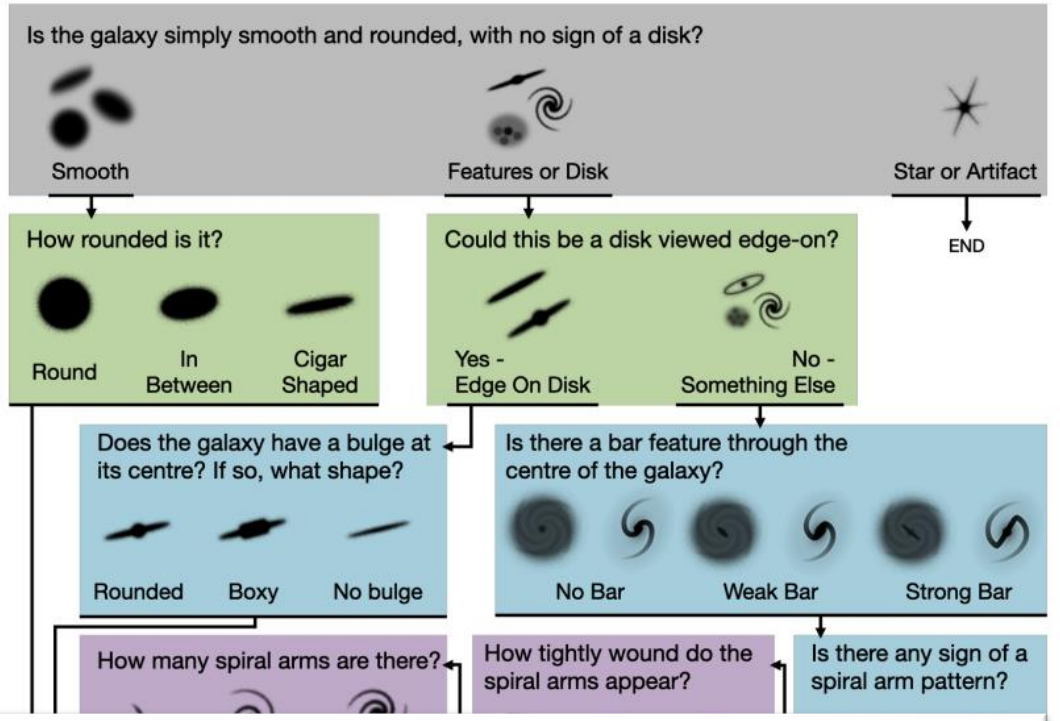
- Smooth
- Features or Disk
- Star, Artifact, or Bad Zoom



Galaxy Zoo

20 citizen scientists classify each galaxy, giving p(smooth)





Galaxy Zoo: Probabilistic Morphology through Bayesian CNNs and Active Learning

Mike Walmsley[1]*, Lewis Smith[2], Chris Lintott[1], Yarin Gal[2], Steven Bamford[3], Hugh Dickinson[4,8], Lucy Fortson[4,8], Sandor Kruk[5], Karen Masters[6,7], Claudia Scarlata[4,8] Brooke Simmons[9,10], Rebecca Smethurst[1], Darryl Wright [4,8]

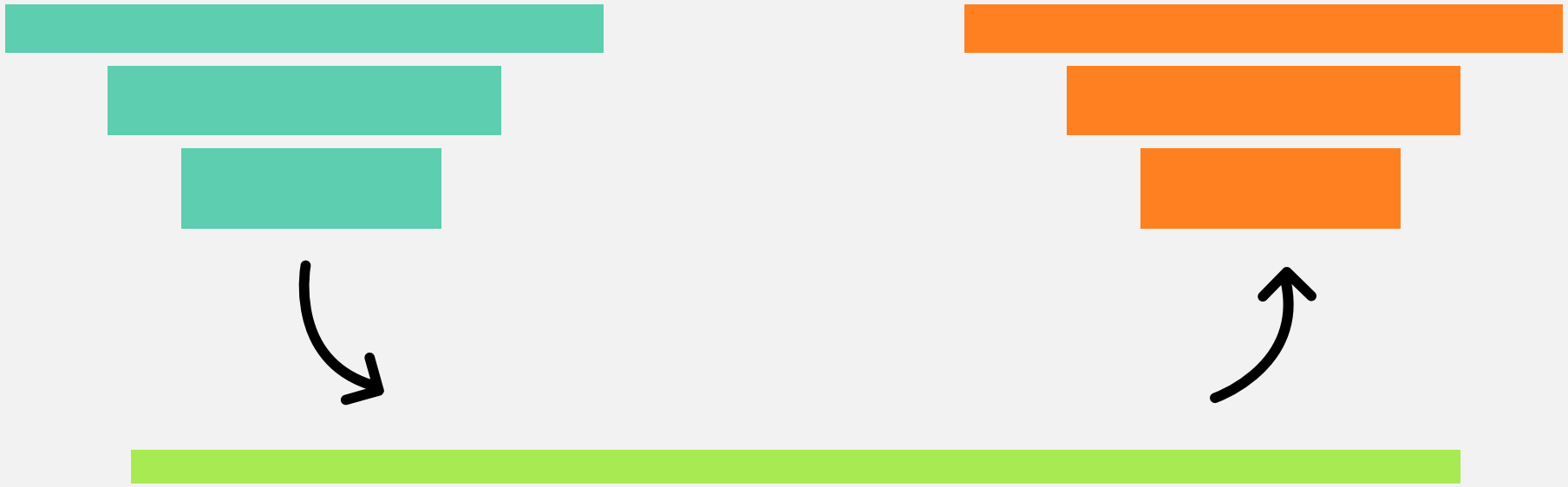
[1] *Oxford Astrophysics, Department of Physics, University of Oxford, Denys Wilkinson Building, Keble Road, Oxford, OX1 3RH, UK*
[2] *Oxford Computer Science, University of Oxford, 15 Parks Rd, Oxford, OX1 3QD, UK*
[3] *School of Physics and Astronomy, University of Nottingham, University Park, Nottingham NG7 2RD, UK*
[4] *School of Physics and Astronomy, University of Minnesota, 116 Church St SE, Minneapolis, MN 55455, USA*
[5] *European Space Agency, ESTEC, Keplerlaan 1, NL-2201 AZ, Noordwijk, The Netherlands*
[6] *Haverford College, Department of Physics and Astronomy, 370 Lancaster Avenue, Haverford, Pennsylvania 19041, USA*
[7] *Institute for Cosmology and Gravitation, University of Portsmouth, Dennis Sciama Building, Burnaby Road, Portsmouth, PO1 3FX, UK*
[8] *Minnesota Institute for Astrophysics, University of Minnesota, Minneapolis, MN 55455, USA*
[9] *Physics Department, Lancaster University, Lancaster, LA1 4YB, UK*
[10] *Center for Astrophysics and Space Sciences (CASS), Department of Physics, University of California, San Diego, CA 92093, USA*

Zoobot: uses dropout at inference to get 20 network votes on each class

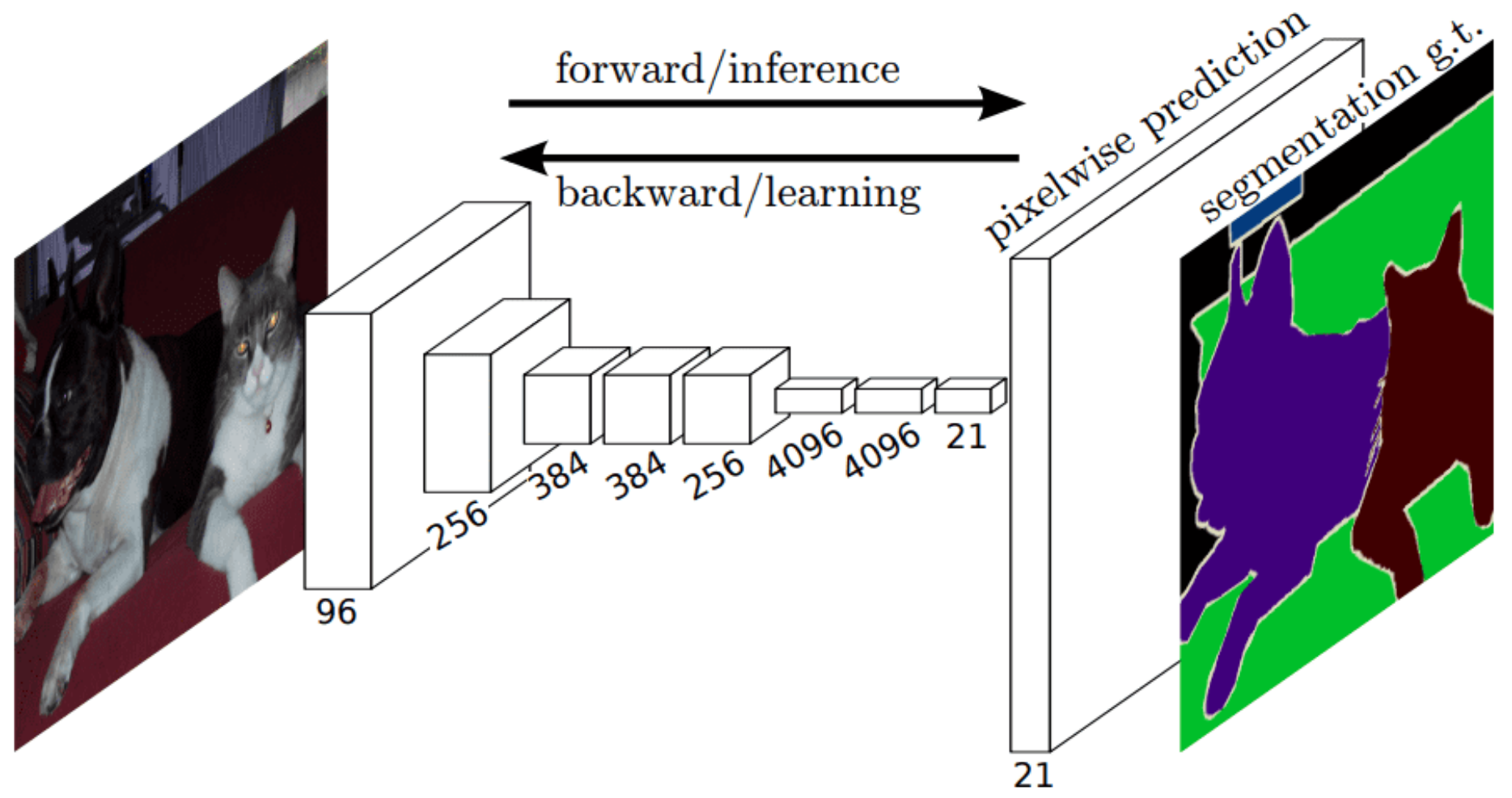
Machine Leaning 301

GANs, contrastive nets, U-nets

encoder-decoder
Or U-net

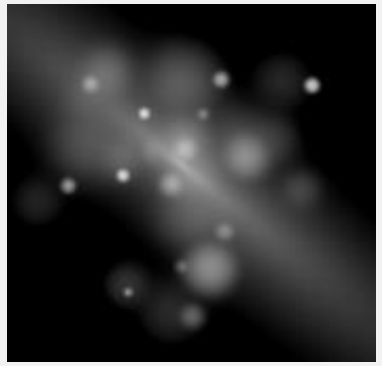


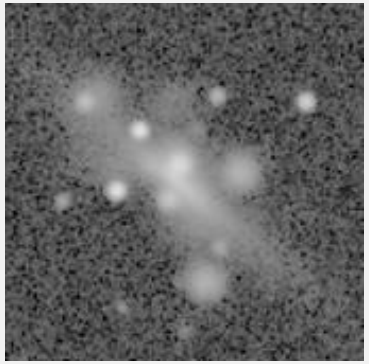
U-nets: image segmentation

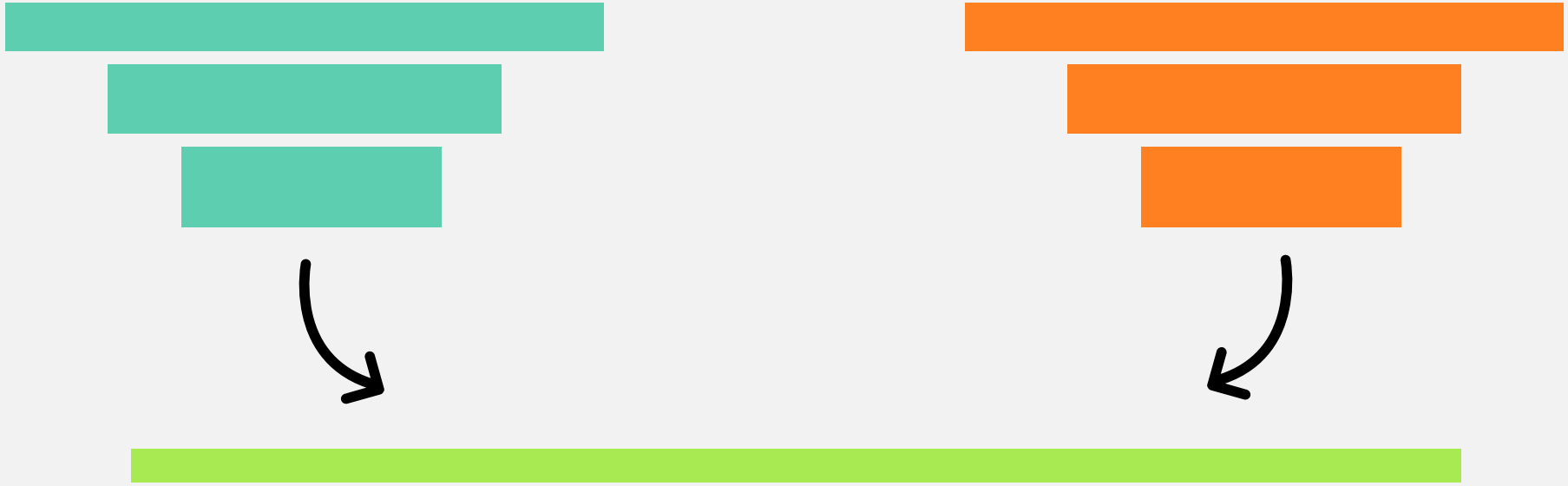


Loss: still similar to binary cross-entropy

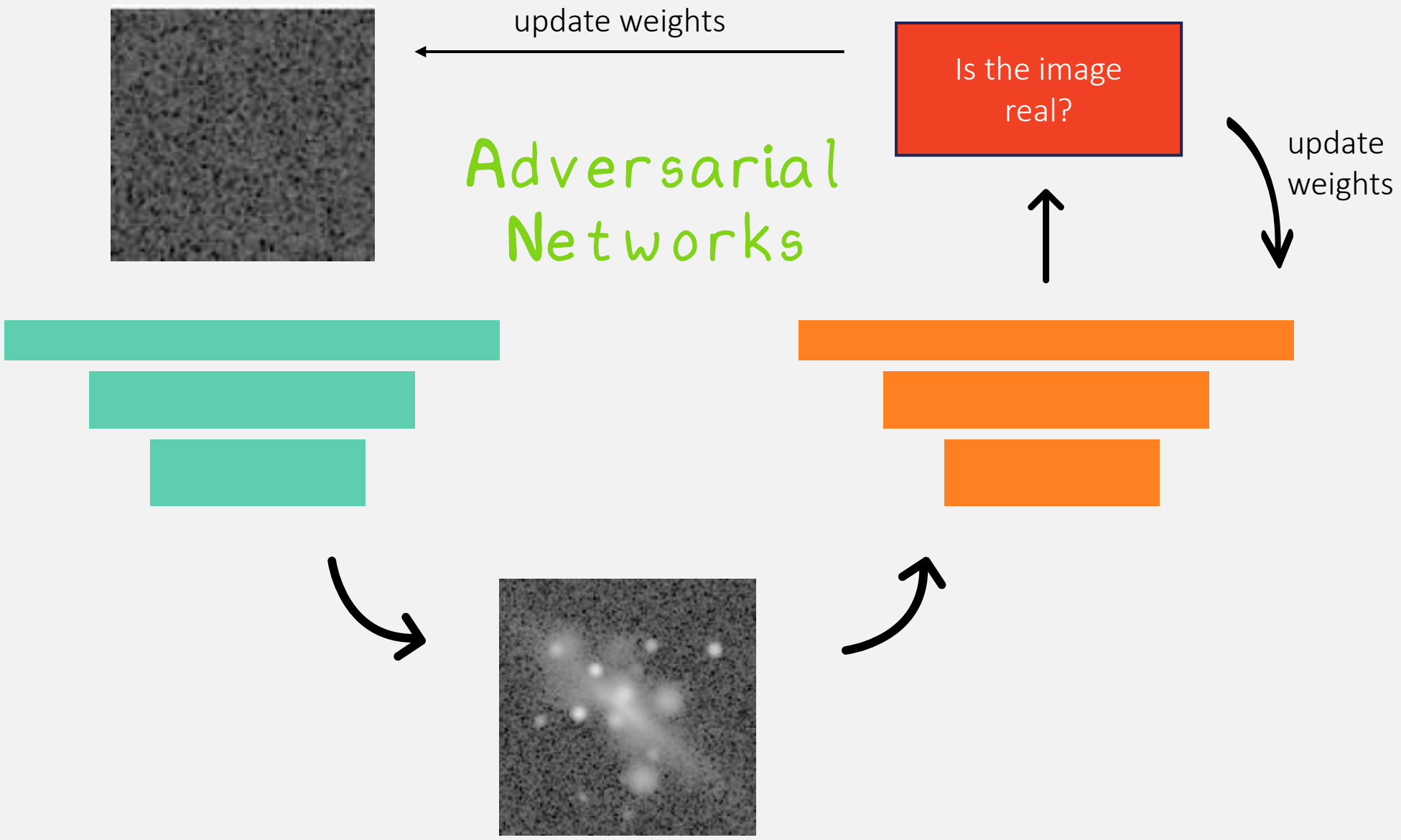
Contrastive Learning





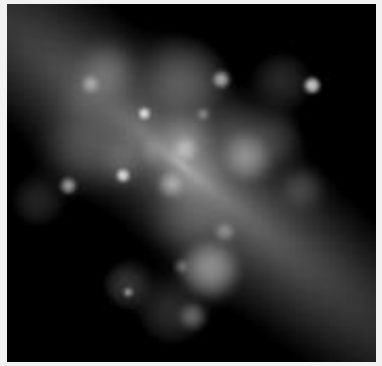


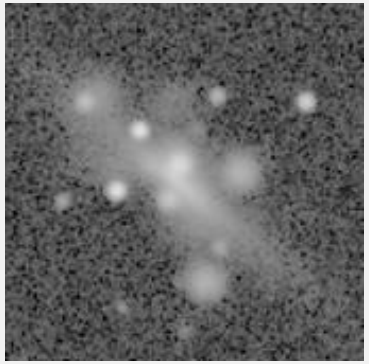
Loss: similarity score



Loss: similarity score

Contrastive Learning





latent space

Loss: similarity score

Latent space

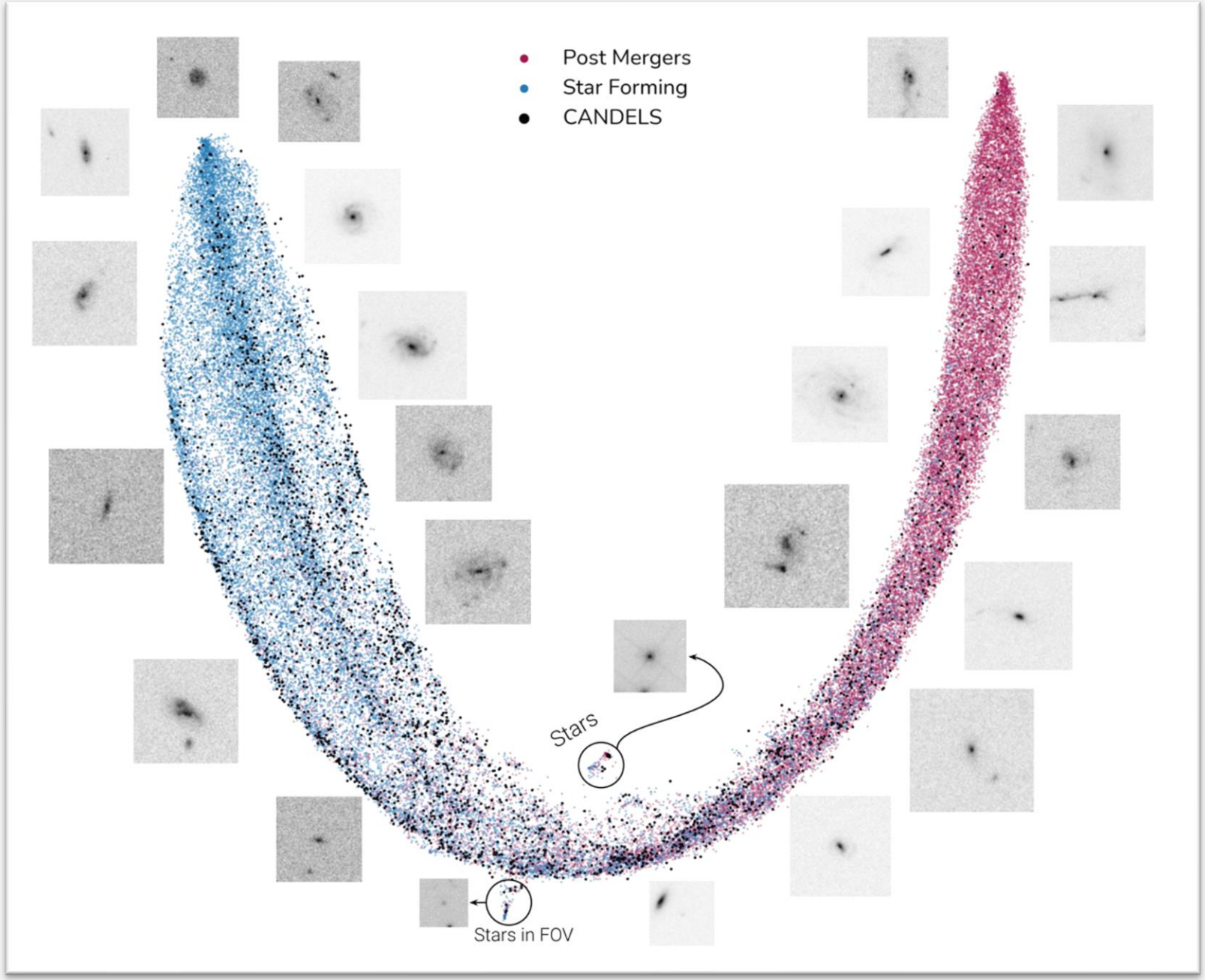
1024-dim vector that represents *something* about the data

Optimized to learn the real properties and be invariant to noise, etc.

Latent space

1024-dim vector that represents *something* about the data

Optimized to learn the real properties and be invariant to noise, etc.



Ferreira et al. 2022