

# Machine Learning crash course

(Part-4)

Ahmed Hammad

*Deep learning and feedforward  
deep neural network*

# Fully connected deep learning

Neural network playground

	Epoch 000,313	Learning rate 0.0001	Activation ReLU	Regularization None	Regularization rate 0	Problem type Classification
--	------------------	-------------------------	--------------------	------------------------	--------------------------	--------------------------------

## DATA

Which dataset do you want to use?



Ratio of training to test data: 50%

Noise: 20

Batch size: 6

**REGENERATE**

## FEATURES

Which properties do you want to feed in?

$x_1$ ,  $x_2$ ,  $x_1^2$ ,  $x_2^2$ ,  $x_1x_2$ ,  $\sin(x_1)$ ,  $\sin(x_2)$

## 6 HIDDEN LAYERS

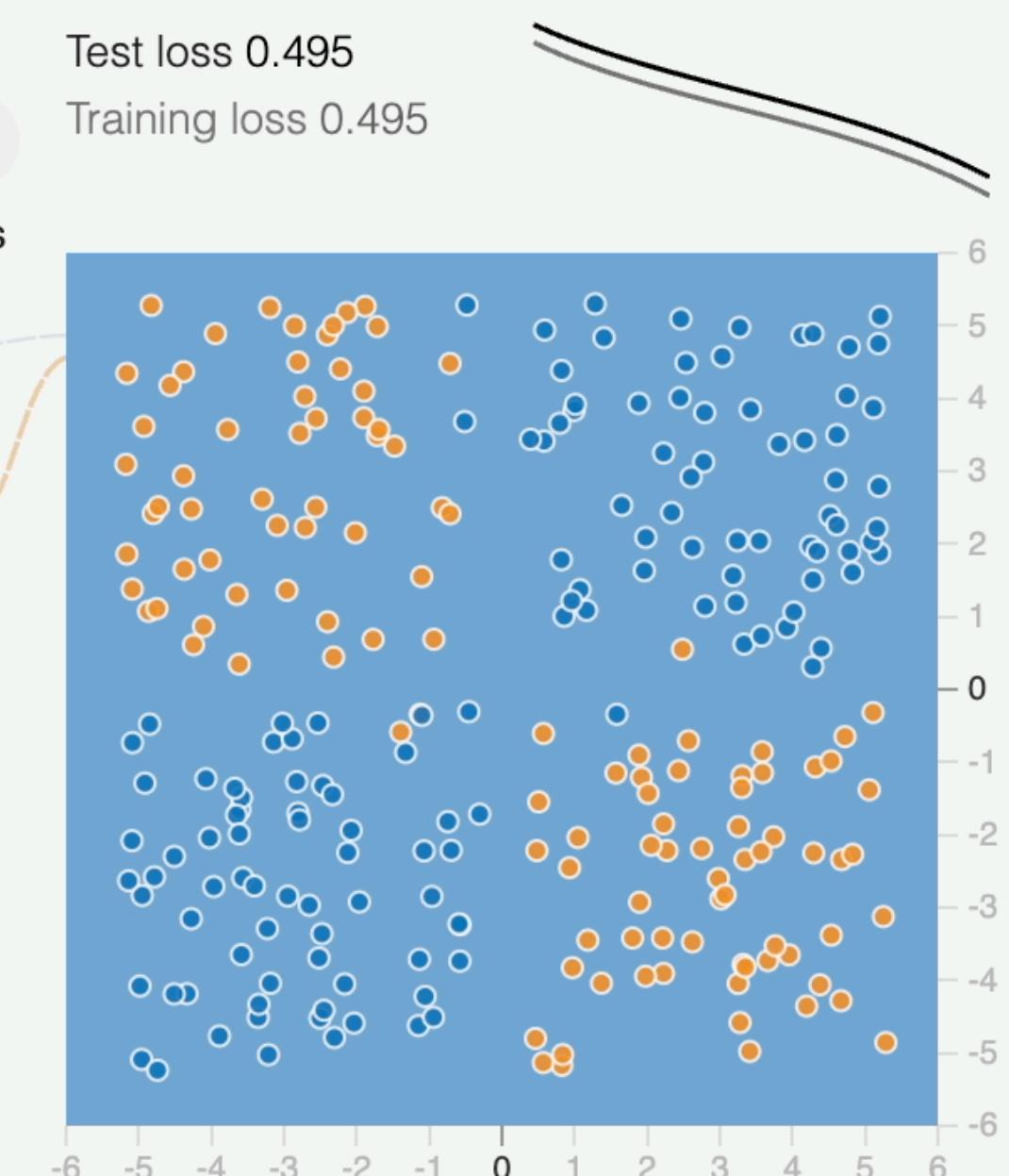
+

-

## 6 HIDDEN LAYERS

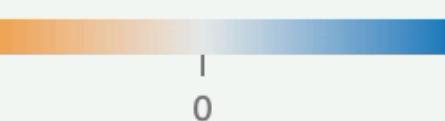
## OUTPUT

Test loss 0.495  
Training loss 0.495



This is the output from one **neuron**. The outputs are mixed with varying **weights**, shown by the thickness of the lines.

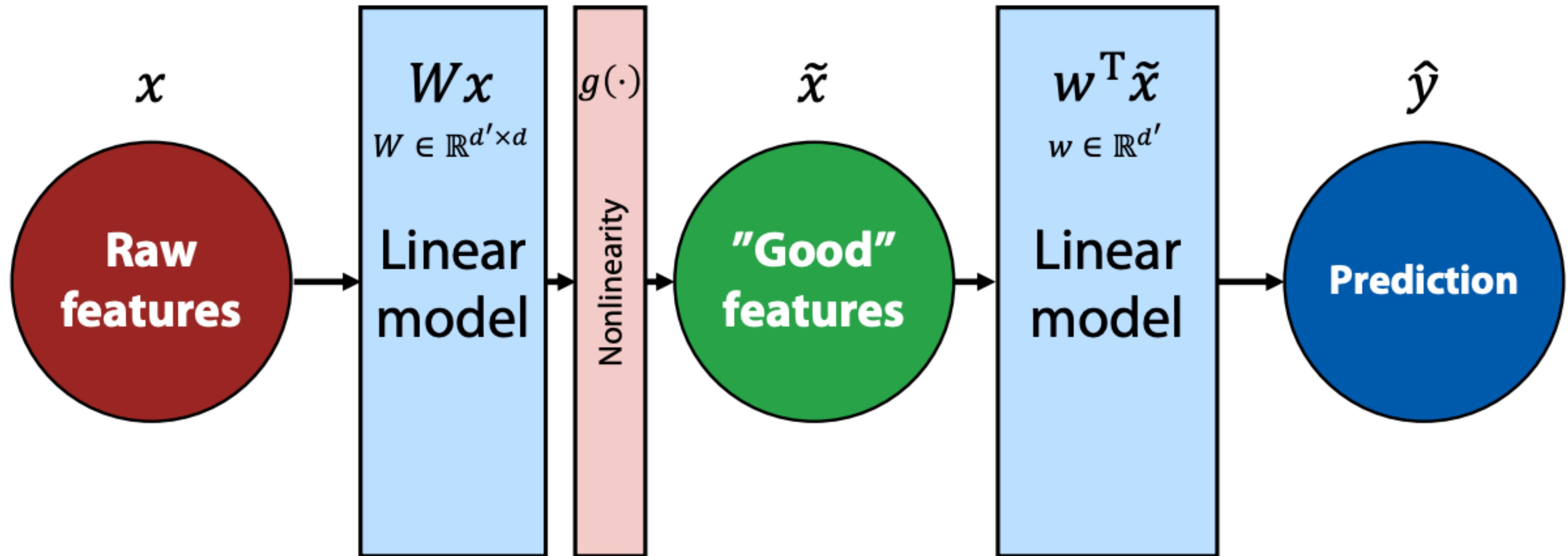
Colors shows data, neuron and weight values.



Show test data

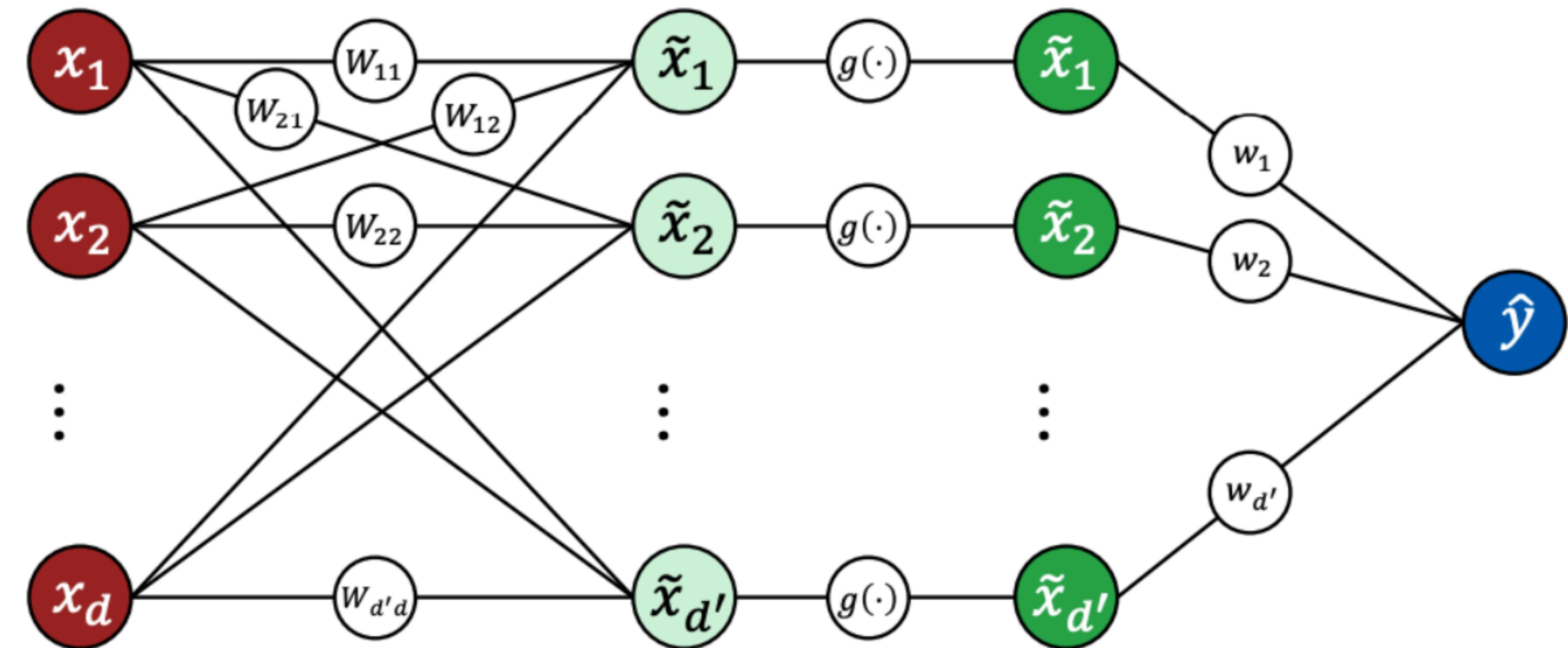
Discretize output

# Fully connected deep learning



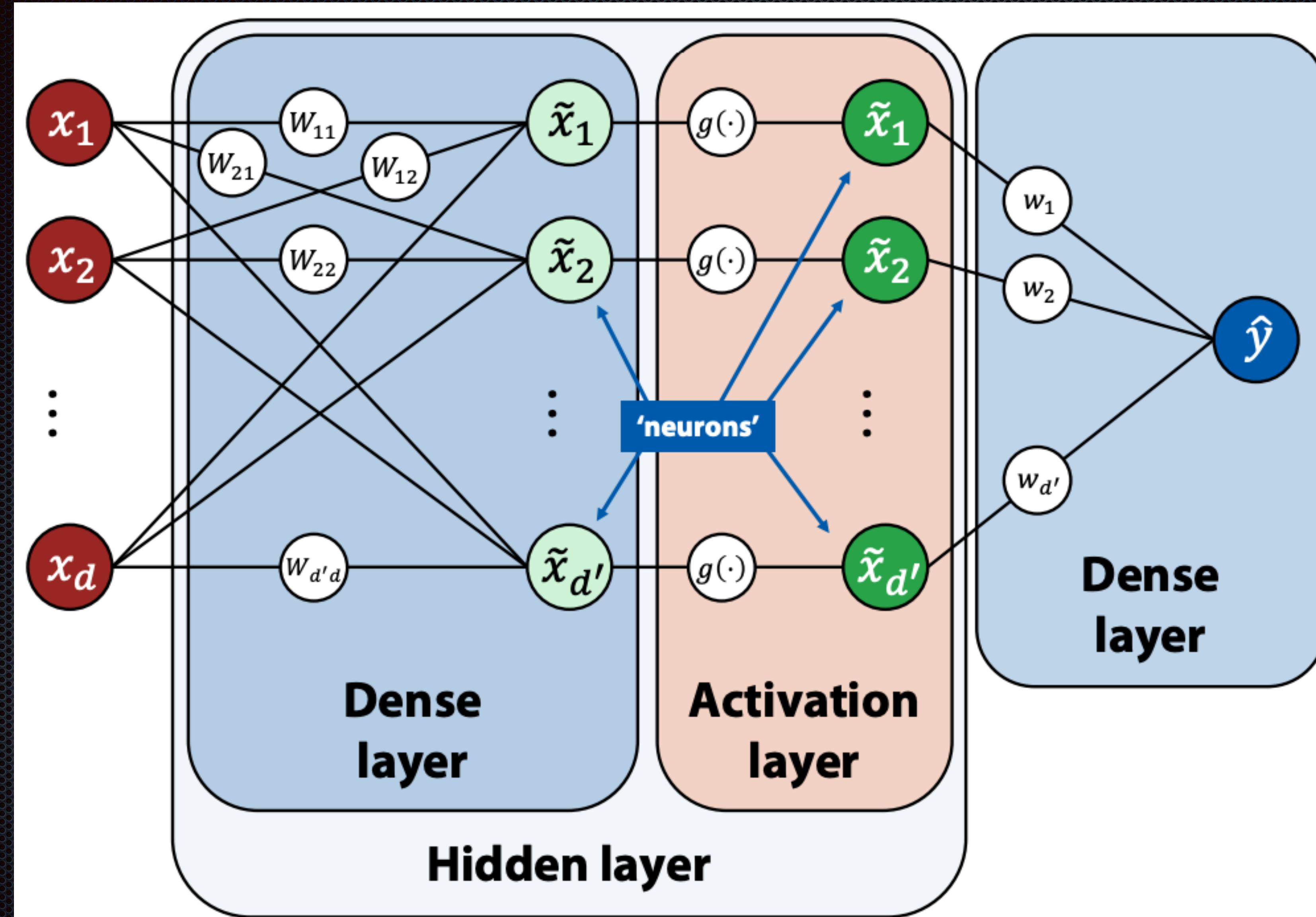
$$\hat{y} = w^T \tilde{x} = w^T g(Wx)$$

# Fully connected deep learning

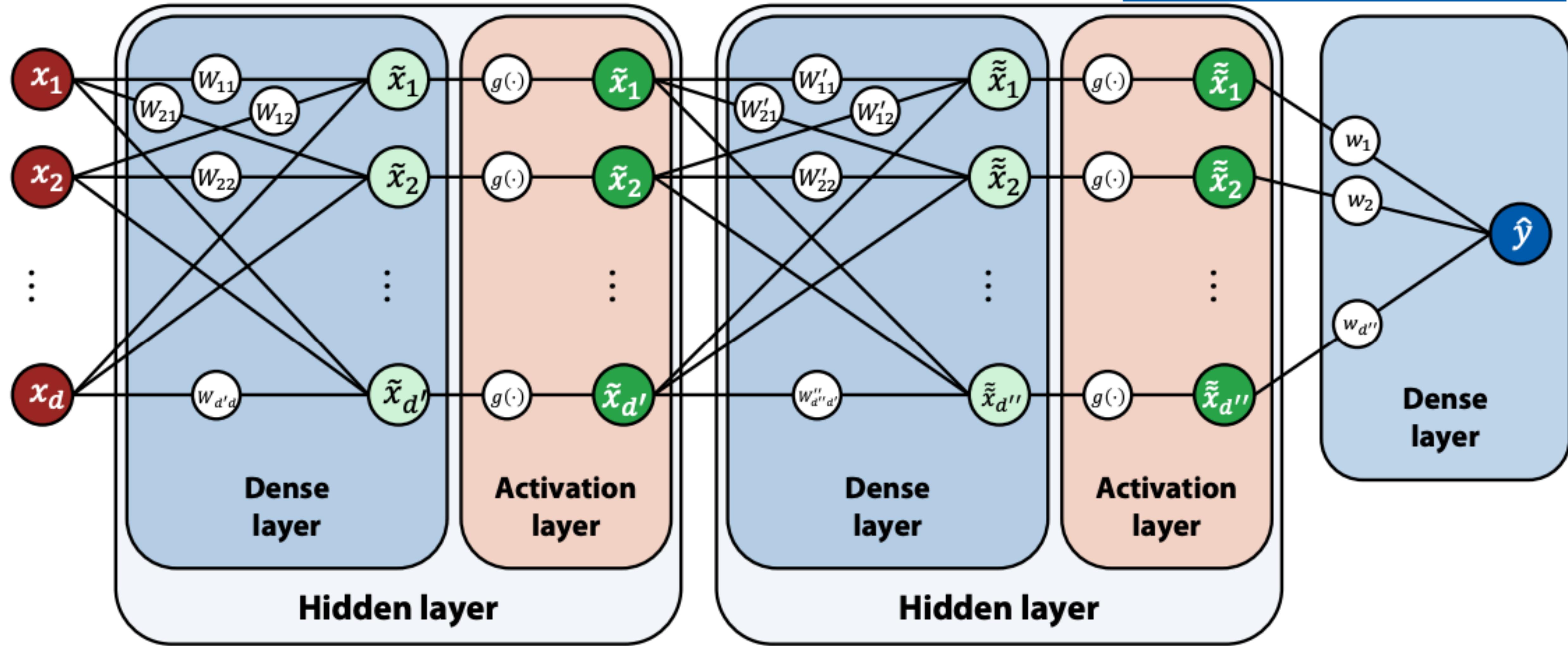


$$\hat{y} = w^T \tilde{x} = w^T g(Wx) = \sum_j [w_j g \left( \sum_i W_{ji} x_i \right)]$$

# Single layer neural network



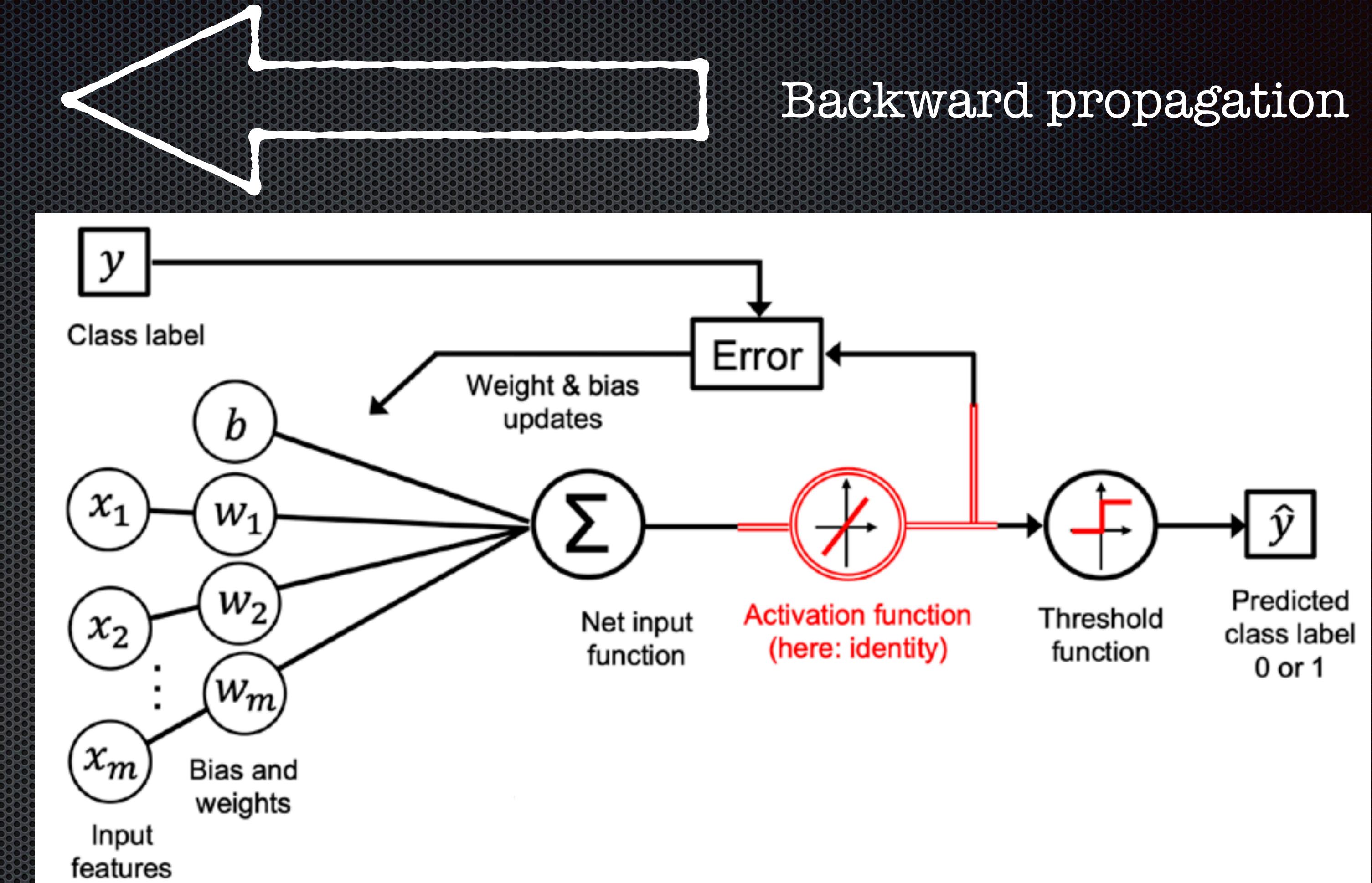
# Multilayer neural network



# Training deep learning models

The training loop consists of forward and backward propagation

Each iteration is called “epoch”

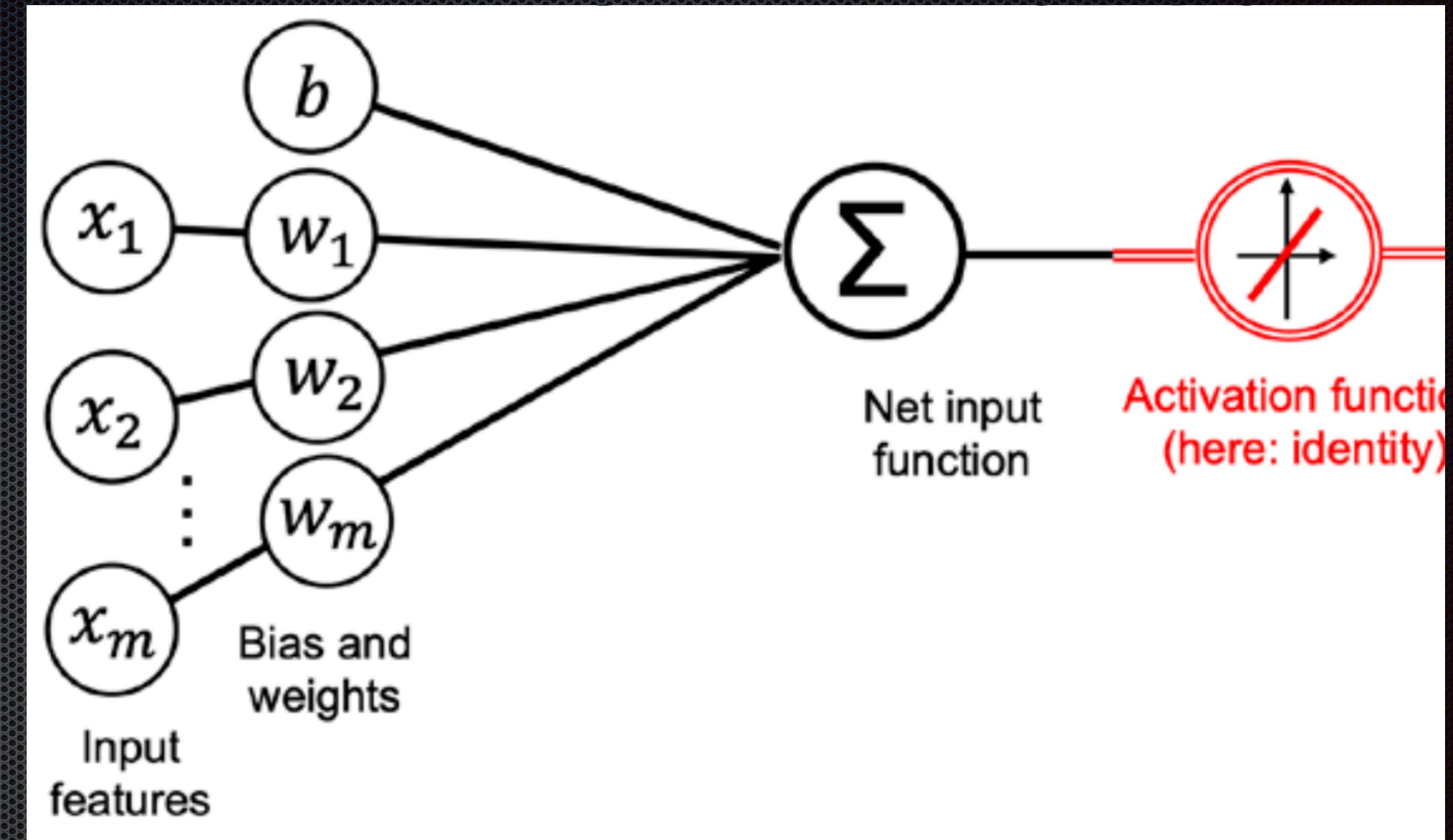


Forward propagation

Backward propagation

# Forward propagation-single layer

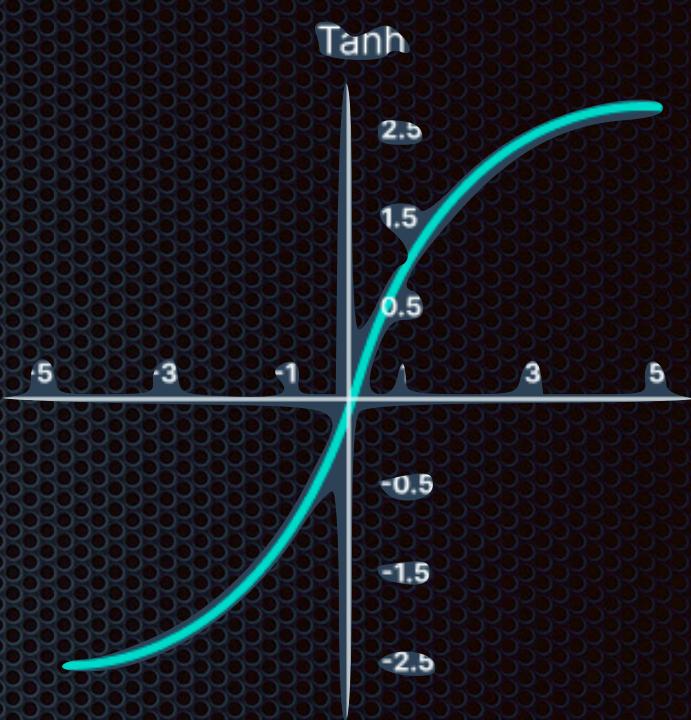
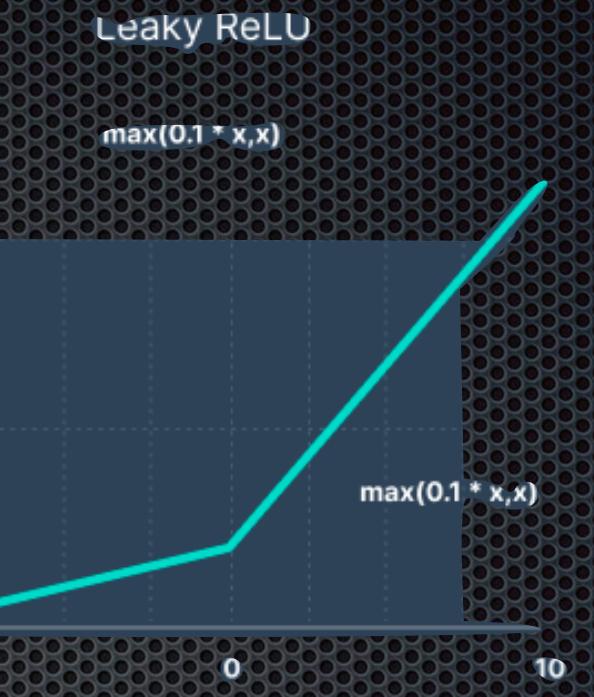
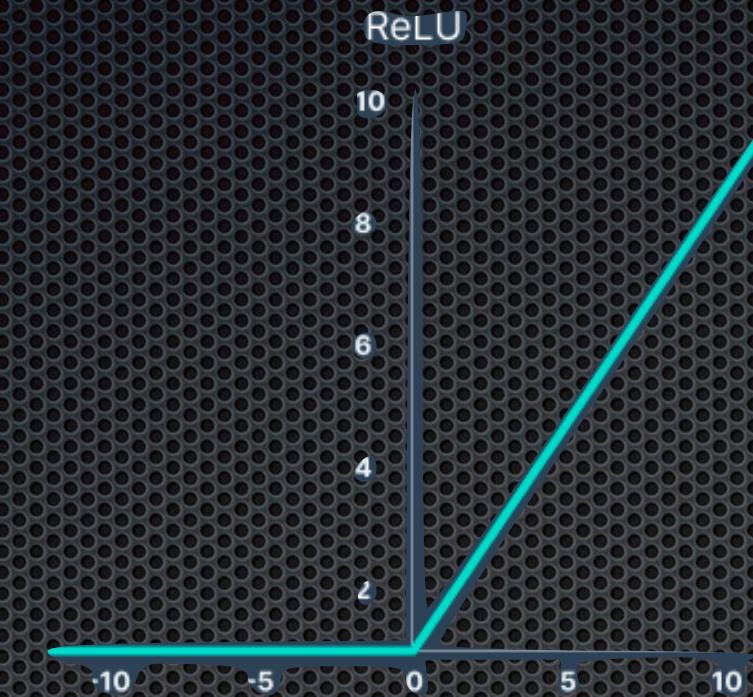
Non-linear activation function is needed to add nonlinearity to the approximated learned classification boundaries



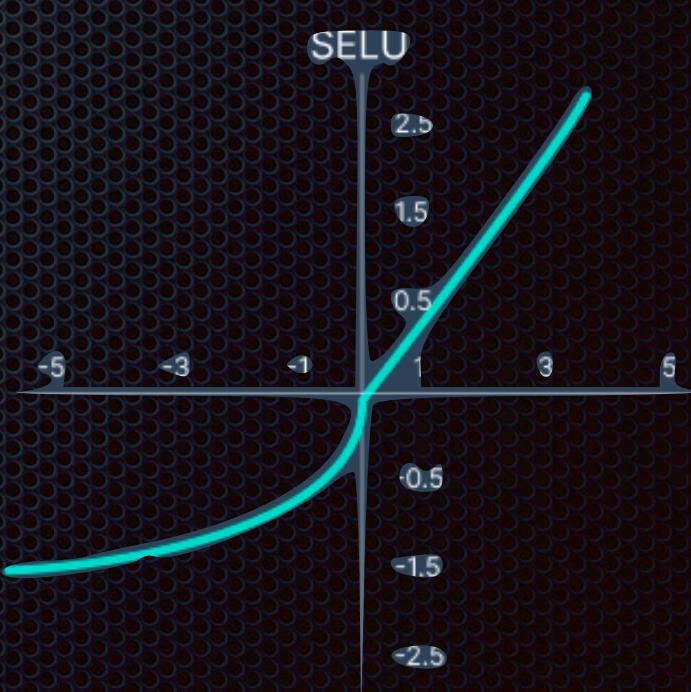
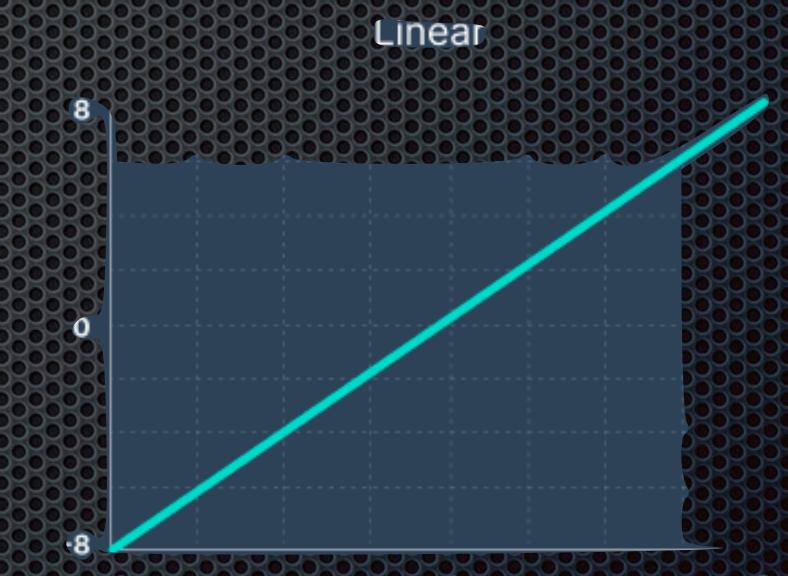
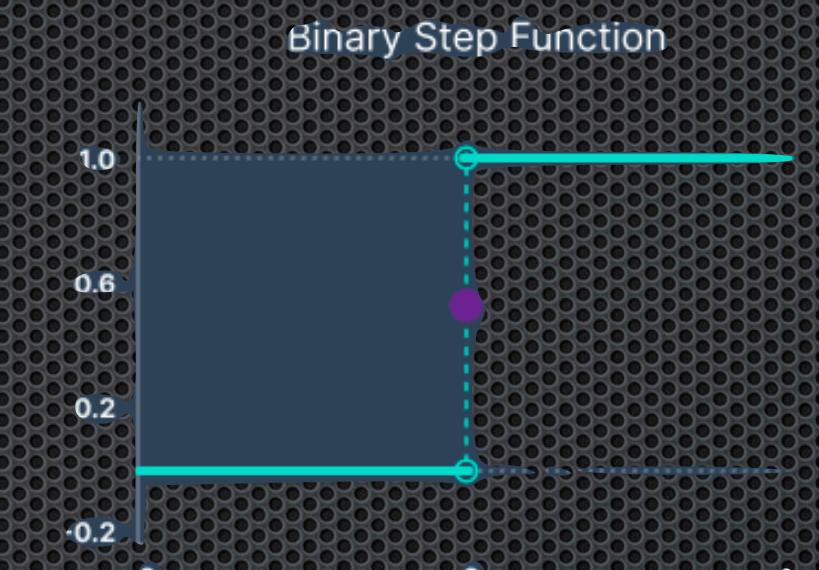
$$\text{Neuron Output} = \text{Activation function} \left( \sum_m \omega_m x_m + b \right)$$

# Activation functions

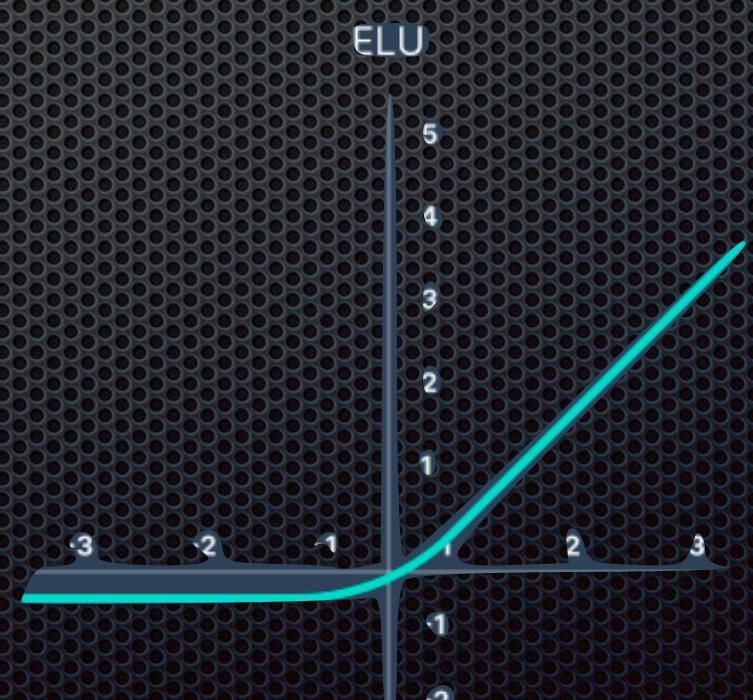
$$\max(0, x)$$



The choice of the activation function at each layer depends on the problem at hand

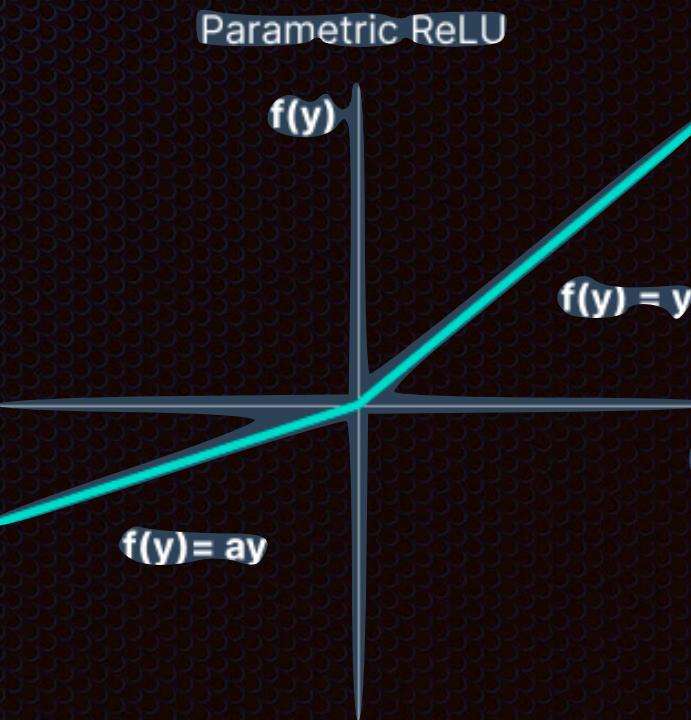
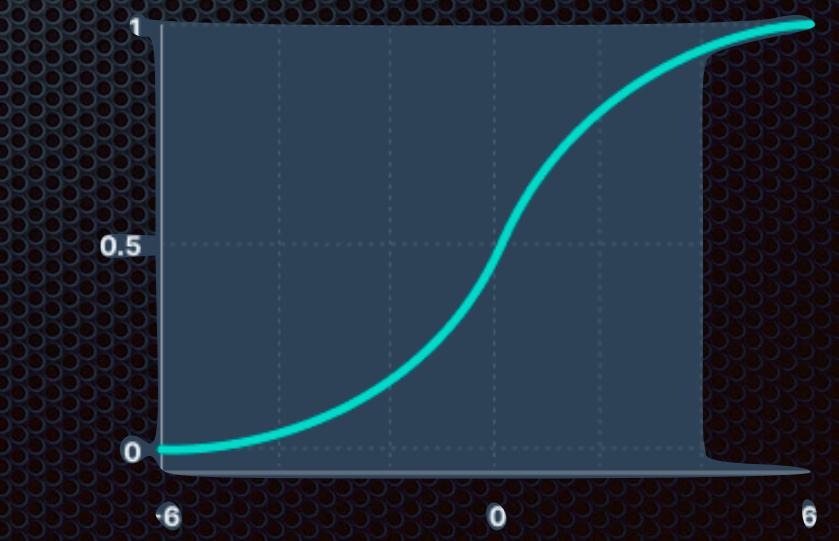


$$\max(\alpha, x)$$



$$\frac{1}{1 + e^{-x}}$$

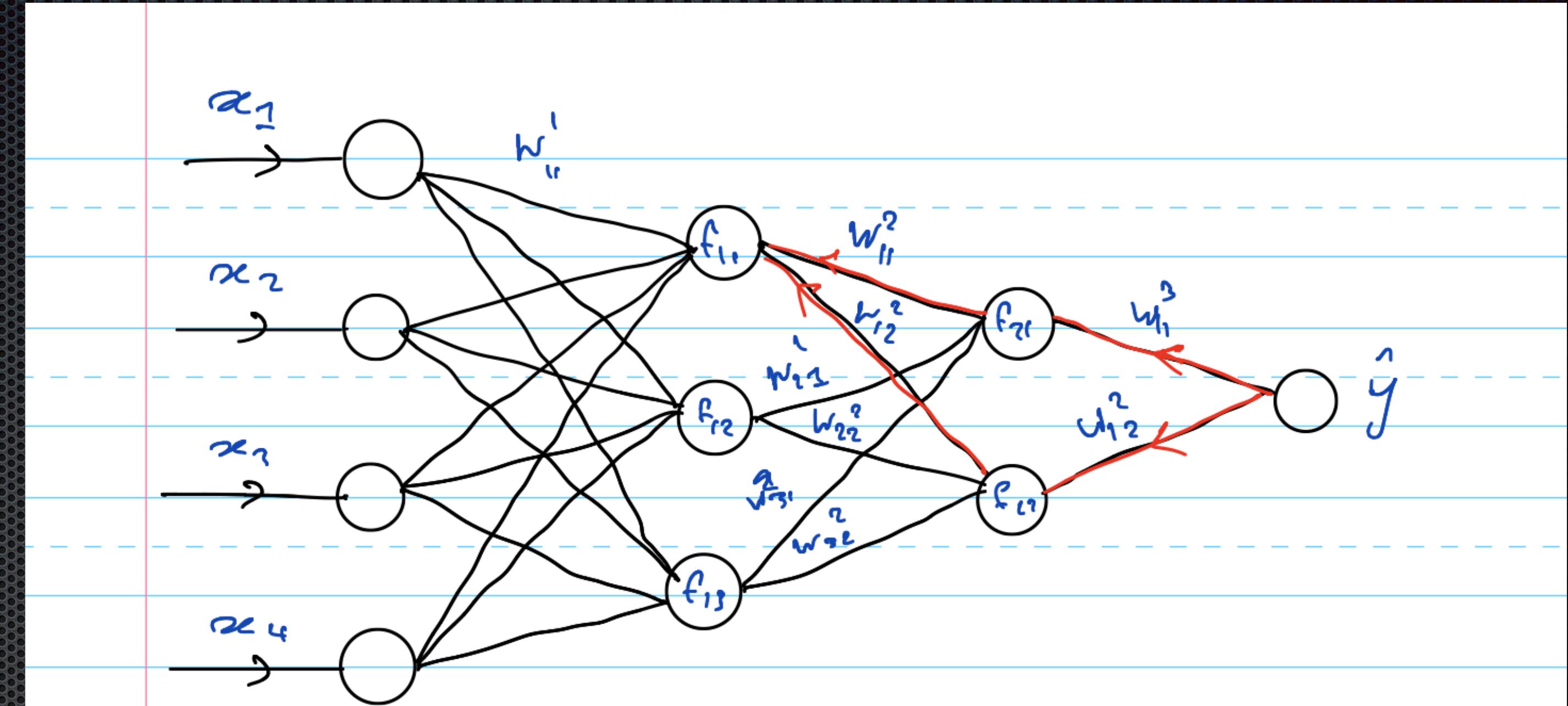
Sigmoid / Logistic



The choice of the activation functions can be optimized using the “Grid search optimization”

# Back propagation-single layer

As discussed before, to update the weights using the gradient descent method we need to compute the derivatives



We use the gradient descent method

$$w_{\text{new}}^i = w_{\text{old}}^i - \eta \nabla L(w_{\text{old}}^i)$$

to update the weights in backward mode

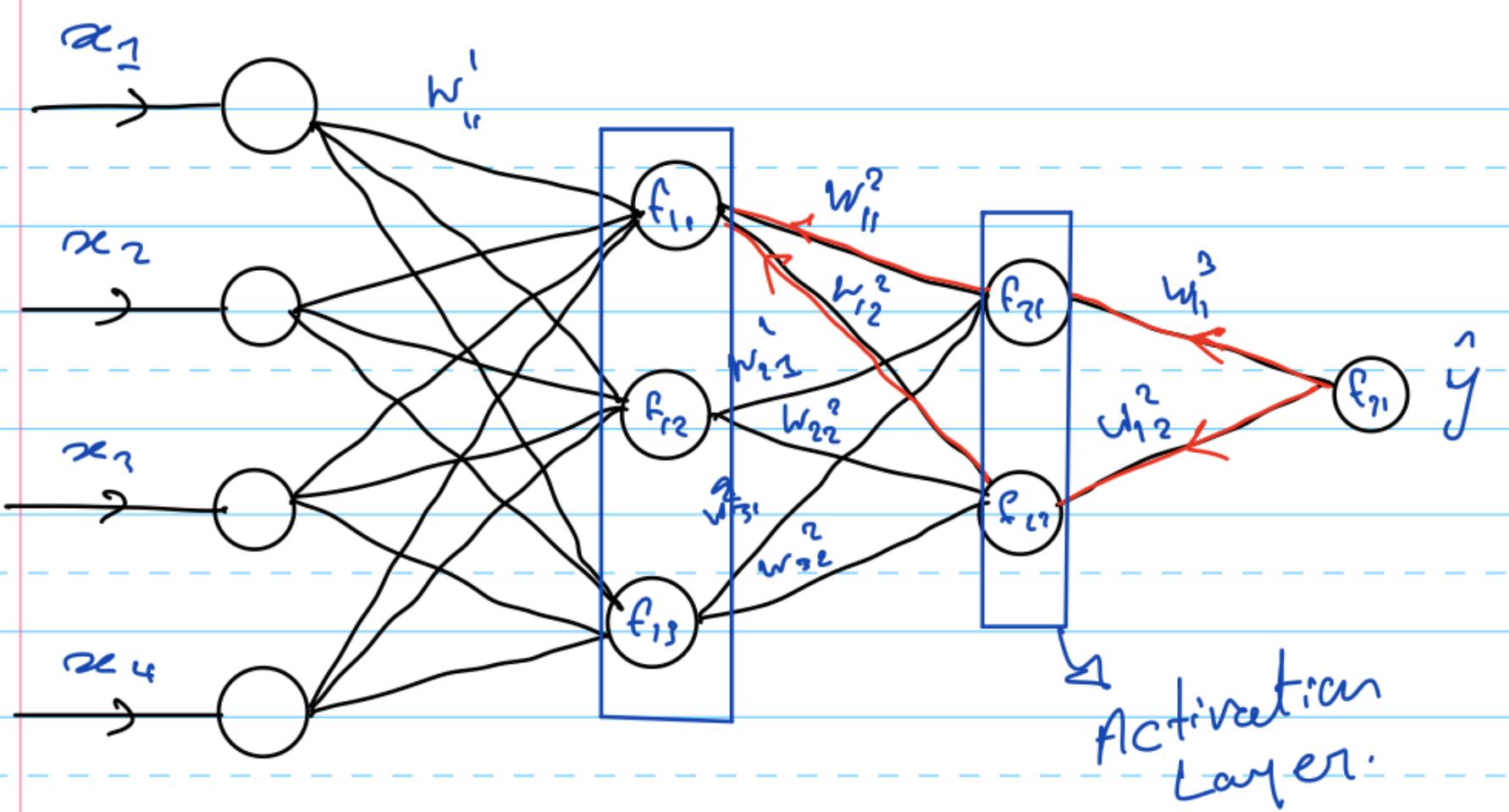
and minimize the loss function.

# Back propagation-single layer

Very tedious work for deeper NN, can you do it yourself ?

To compute the derivatives for the tunable weights, computers use the “Automatic differentiation”

Please take a look at the auto-diff methods in TensorFlow or PyTorch



$$L = \frac{1}{2} (y - \hat{y})^2, \quad w_{new}^i = w_{old}^i - \gamma \nabla L (w_{old}^i)$$

Example:

Find the updated weight for  $w_{11}^3$ :

$$\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial f_{31}} \frac{\partial f_{31}}{\partial w_{11}^3}$$

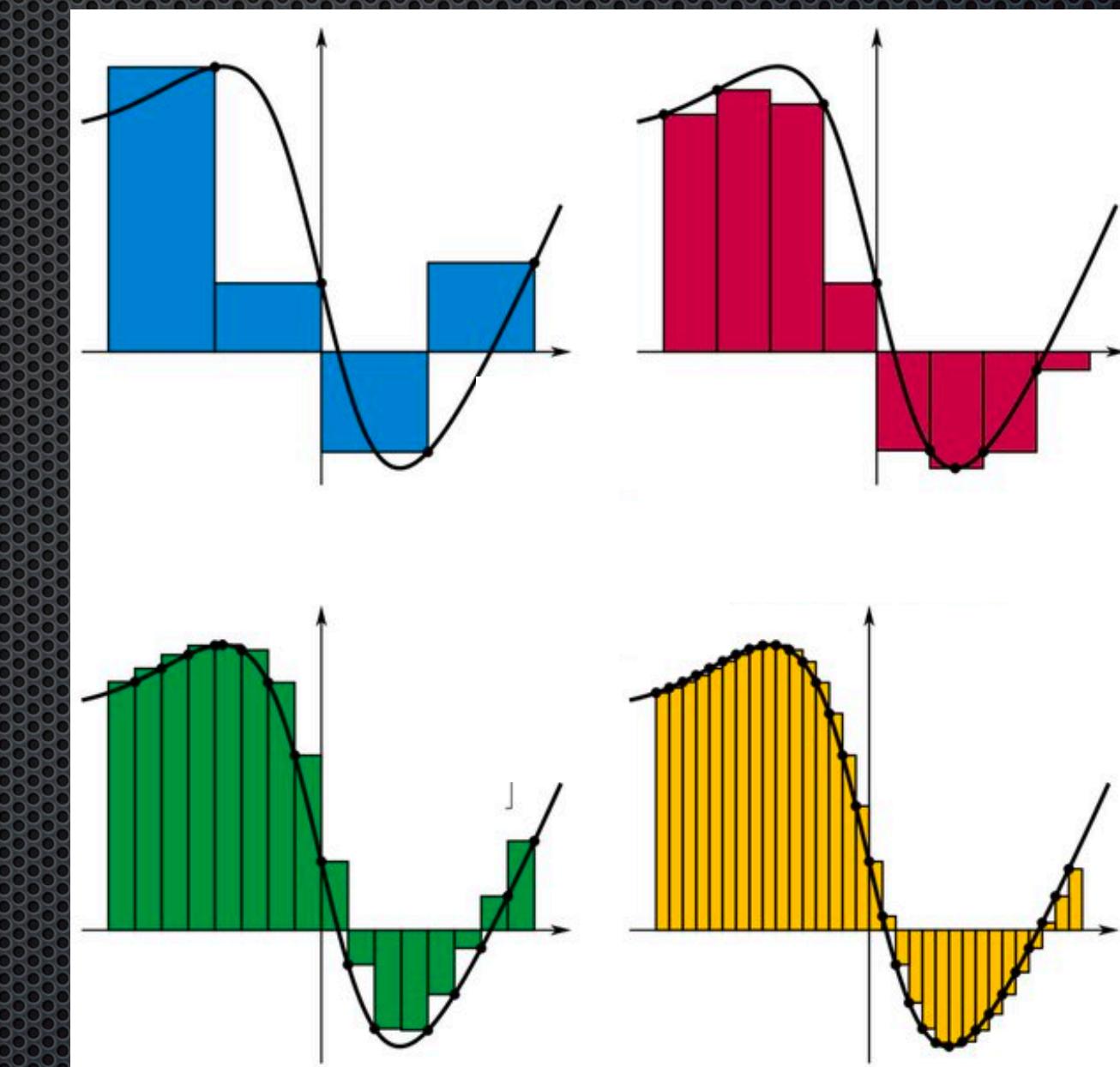
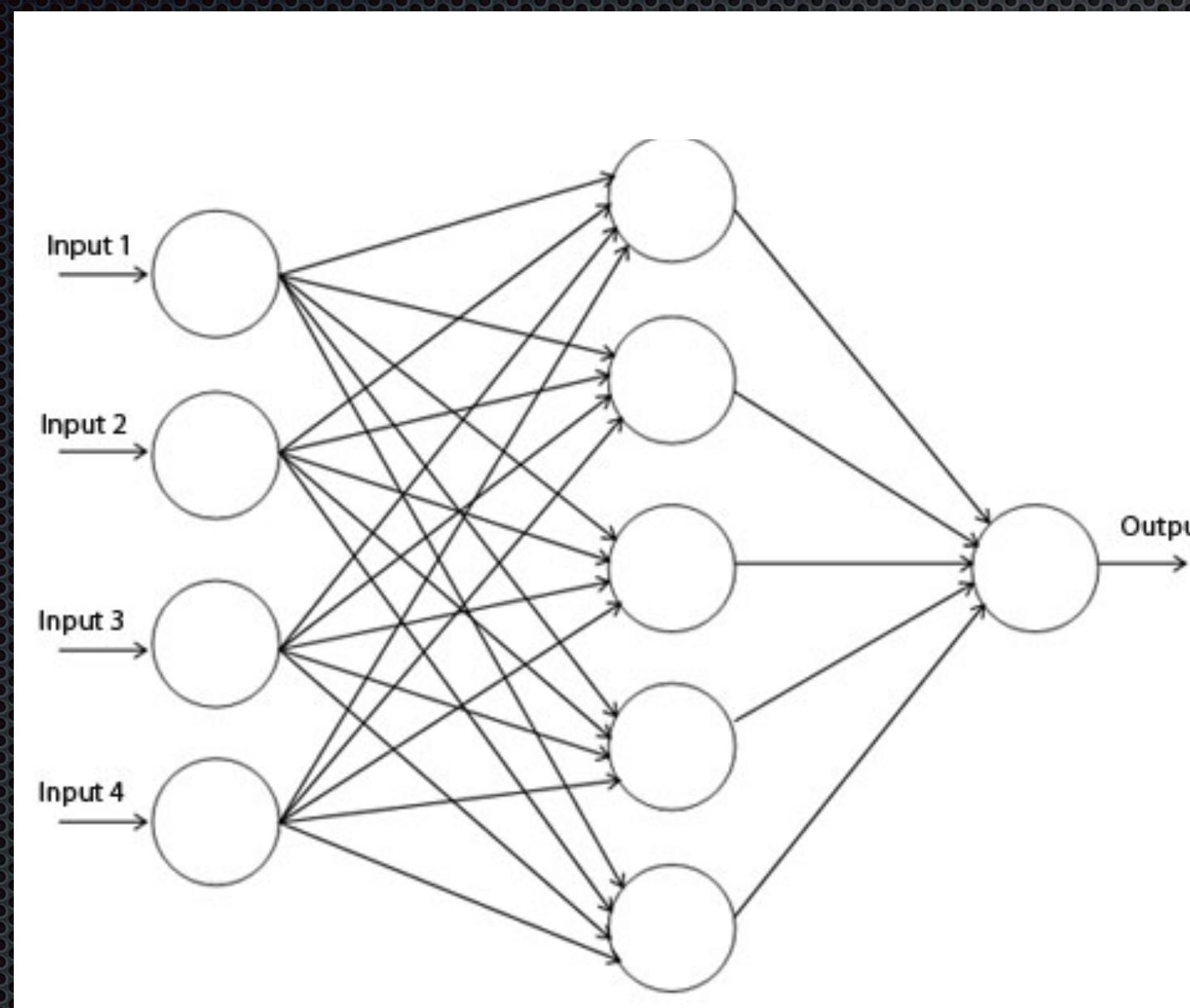
$$w_{11, new}^3 = w_{11}^3 - \gamma * \left[ \frac{\partial L}{\partial f_{31}} \frac{\partial f_{31}}{\partial w_{11}^3} \right]$$

\* For  $w_{11}^2$ :

$$\frac{\partial L}{\partial w_{11}^2} = \left[ \frac{\partial L}{\partial f_{31}} \frac{\partial f_{31}}{\partial w_{11}^3} \frac{\partial f_{21}}{\partial w_{11}^2} \right] + \left[ \frac{\partial L}{\partial f_{31}} \frac{\partial f_{31}}{\partial w_{12}^3} \frac{\partial f_{22}}{\partial w_{11}^2} \right]$$

# Universal approximation theorem

Given any continuous function, no matter how complicated it is, there is always exist a network that can approximate this function

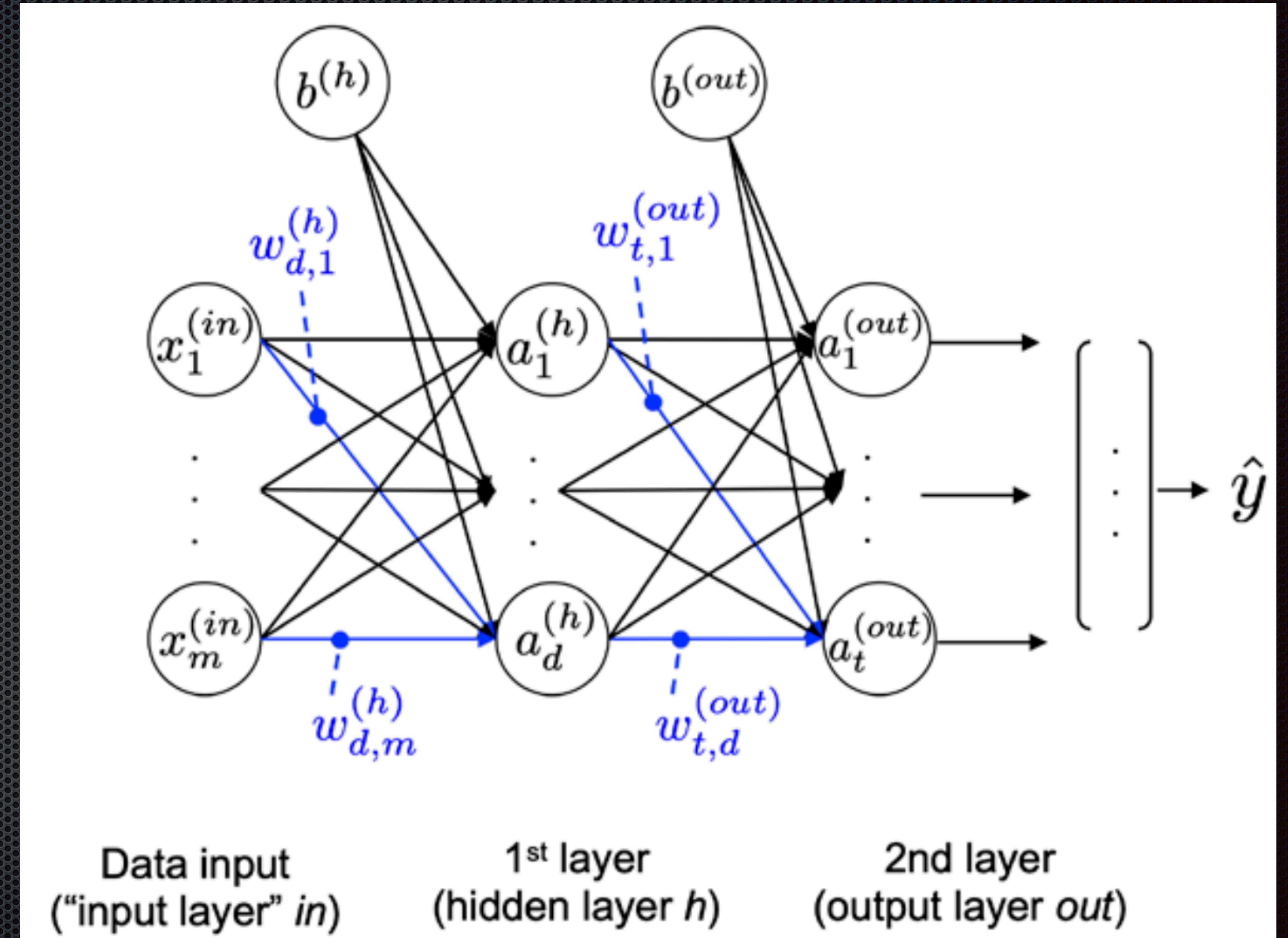


What about nonlinear discontinuous high dimensional function ?

# Multi-layer perceptron (MLP)

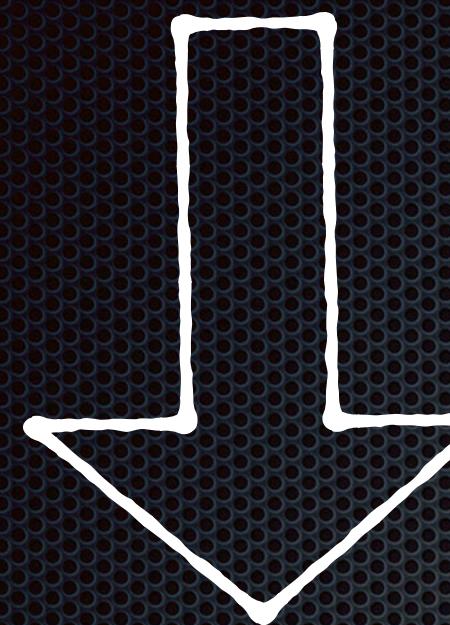
Adding more hidden layers increases the model ability to approximate more complicated functions

Are we free to use any number of hidden layers?

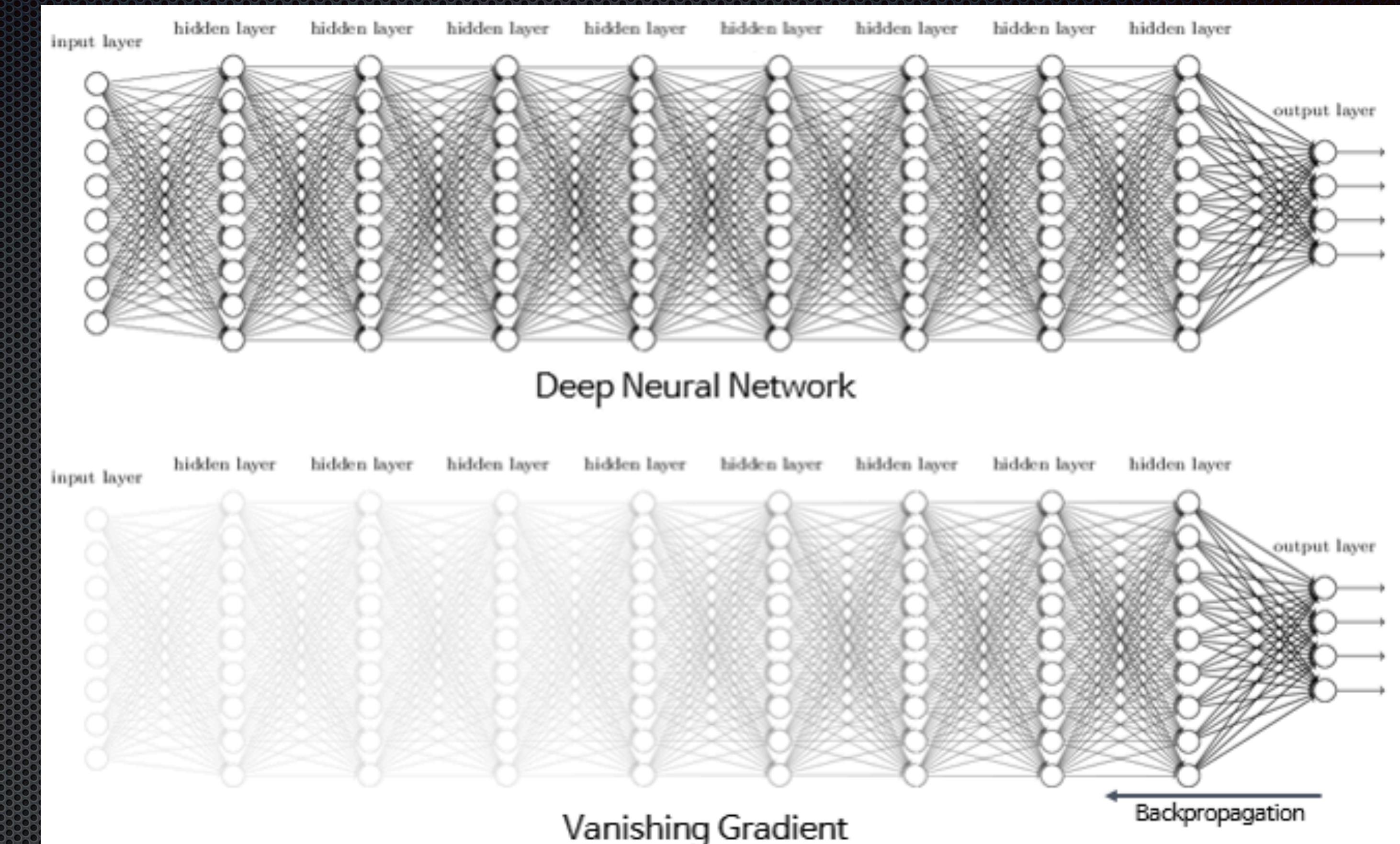


# Vanishing gradient

For very deep models, during the backpropagation  
the gradient of the initial layers vanishes

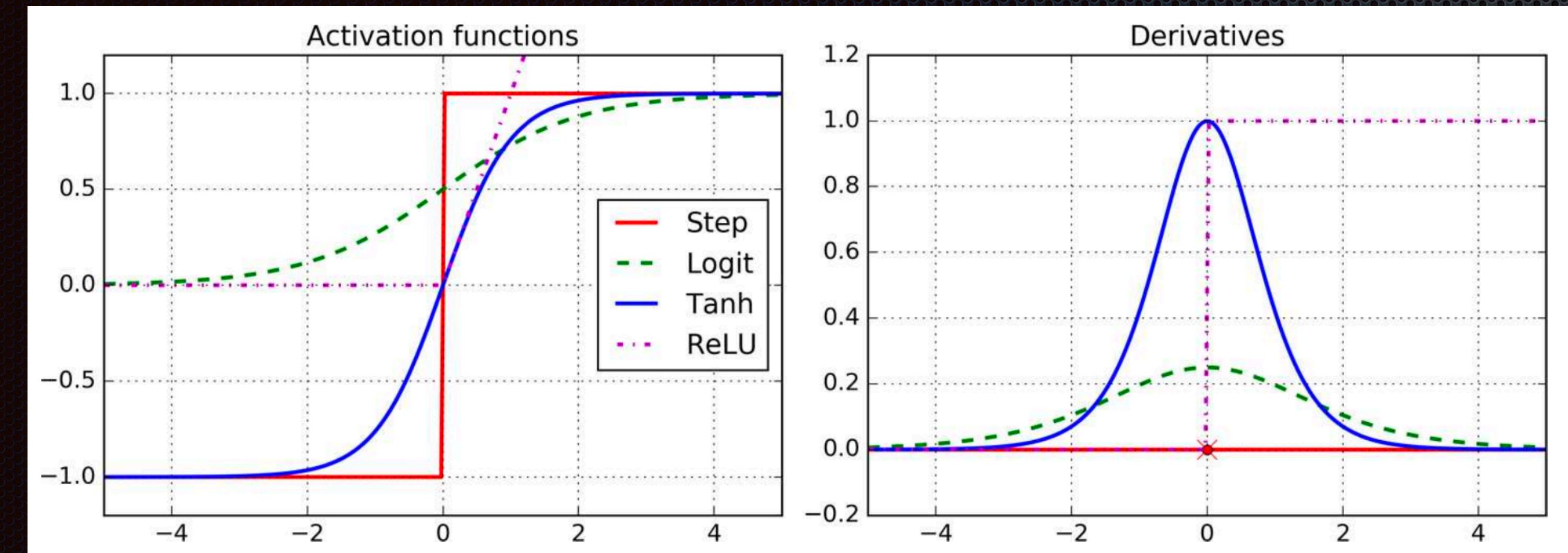


$$\omega_i^{\text{new}} = \omega_i^{\text{old}} - \eta \nabla L(\omega_i^{\text{old}}) \approx \omega_i^{\text{old}}$$



Why vanishing gradients takes place?

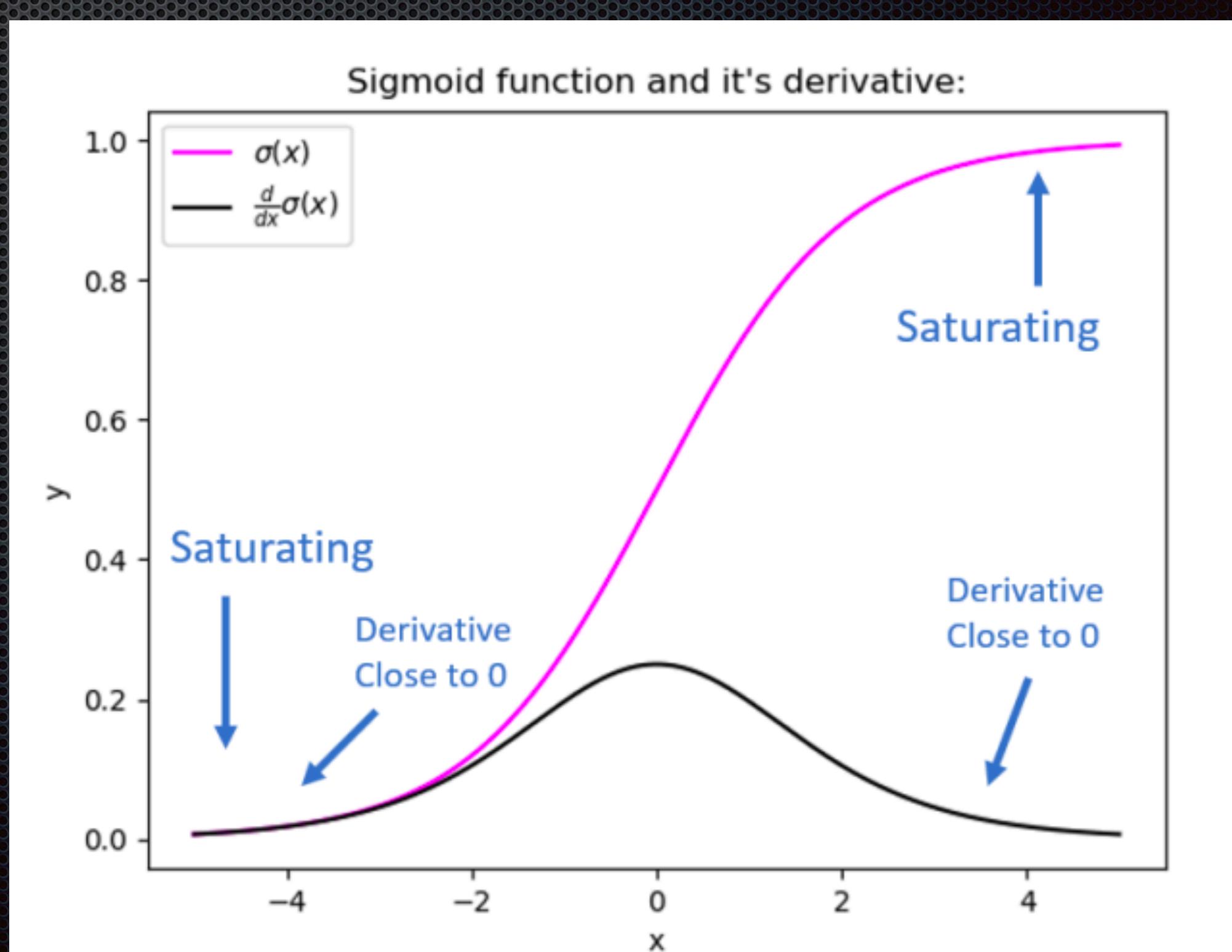
# Vanishing gradient



As long as we keep the gradients for so long  
the derivatives of the activation functions vanish

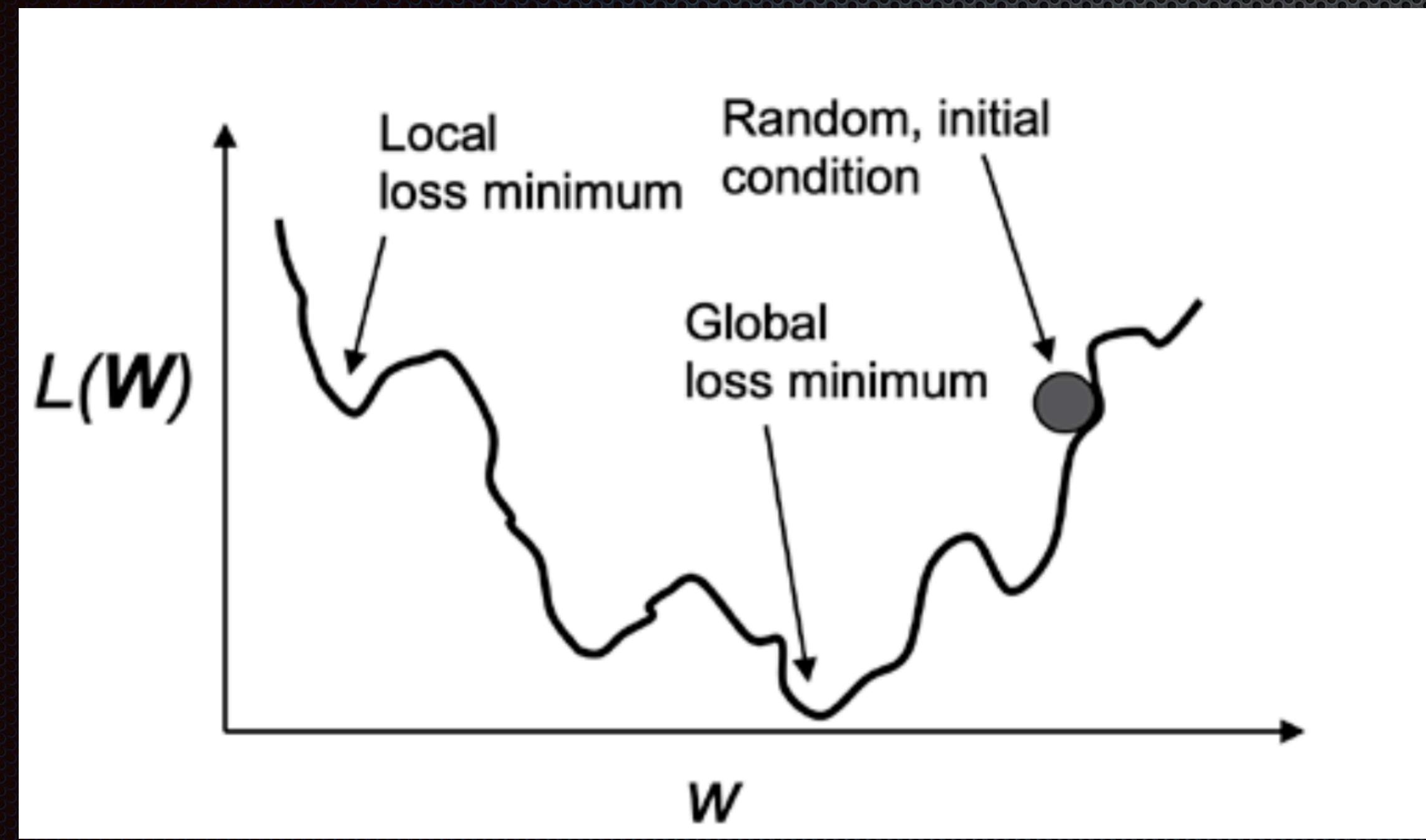
## Solutions:

- Using regularized activation functions, e.g. Leaky ReLU
- Using batch normalization Layer
- Adjust the learning rate
- Reduce the number of the hidden layers
- Clip the weights



# Convergence of the deep models

In multilayer NNs, we typically have hundreds, thousands, or even billions of weights that we need to optimize. Unfortunately, the output function has a rough surface, and the optimization algorithm can easily become trapped in local minima

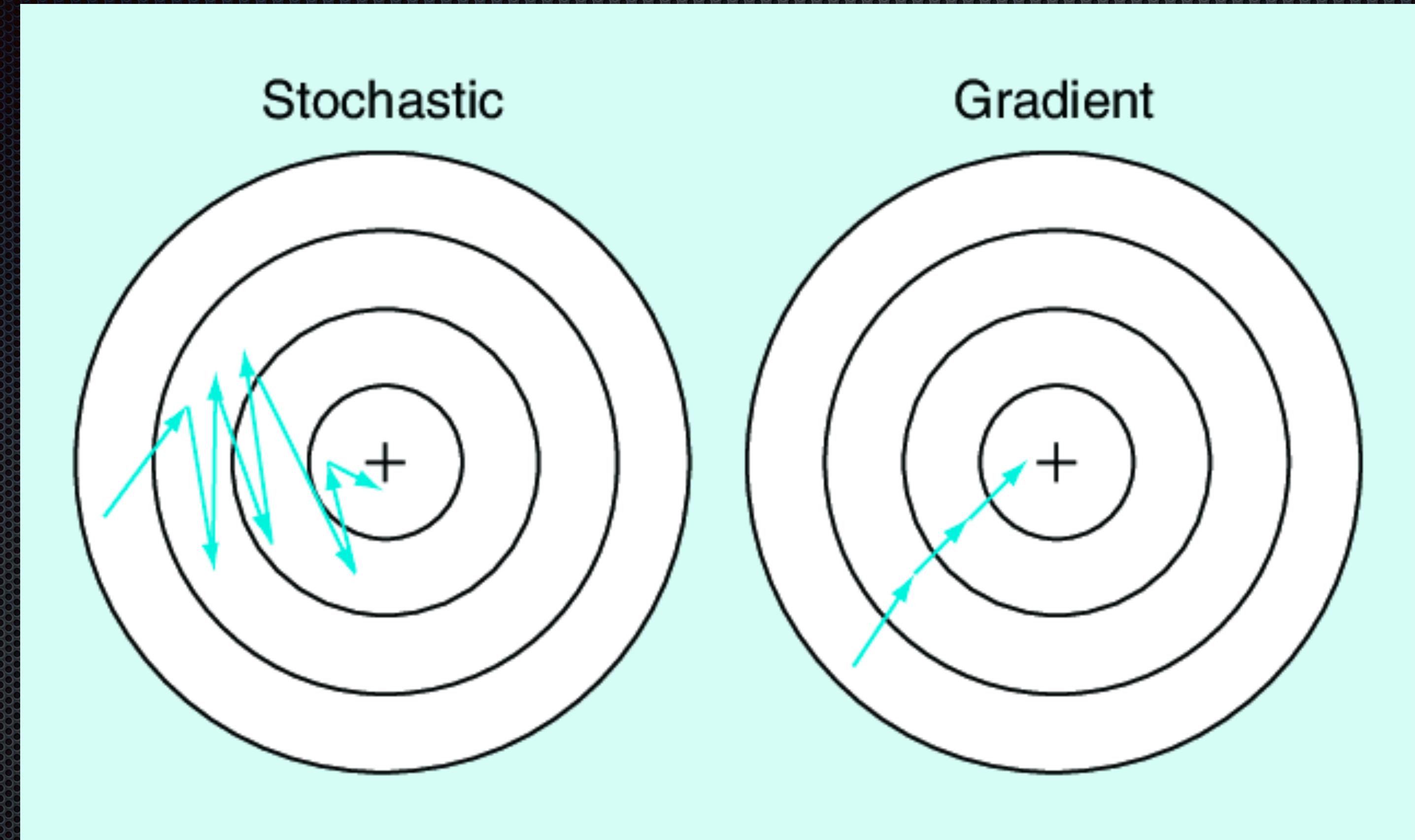


Conventionally, we initialize the weights randomly. If the initialized weight is on the right it can reach the global minimum faster than if it lies in the left.

To avoid this problem we train the model on randomly selected batches of the input

# *Stochastic gradient descent*

Same as the gradient descent but we choose random inputs to update their weights



Stochastic gradient descent oscillates due to the random choice and may not converge easily

## Stochastic gradient descent with momentum

Momentum is the exponential weighted average to reduce the fluctuation

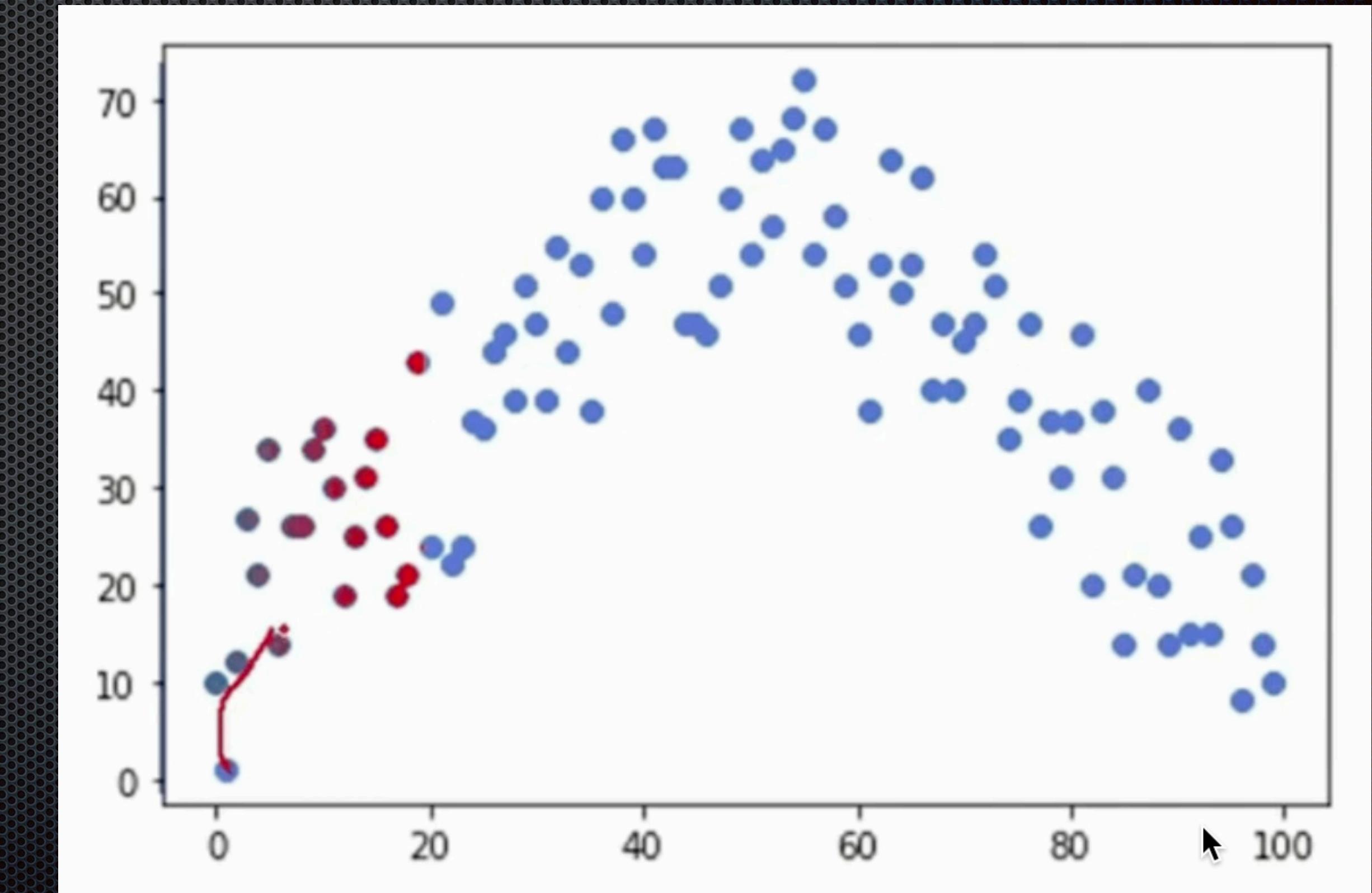
Exponential weighted moving average

$$V_t = \beta V_{t-1} + (1 - \beta) F_t$$

$$V_{t-1} = \beta V_{t-2} + (1 - \beta) F_{t-1}$$

$$V_{t-2} = \beta V_{t-3} + (1 - \beta) F_{t-2}$$

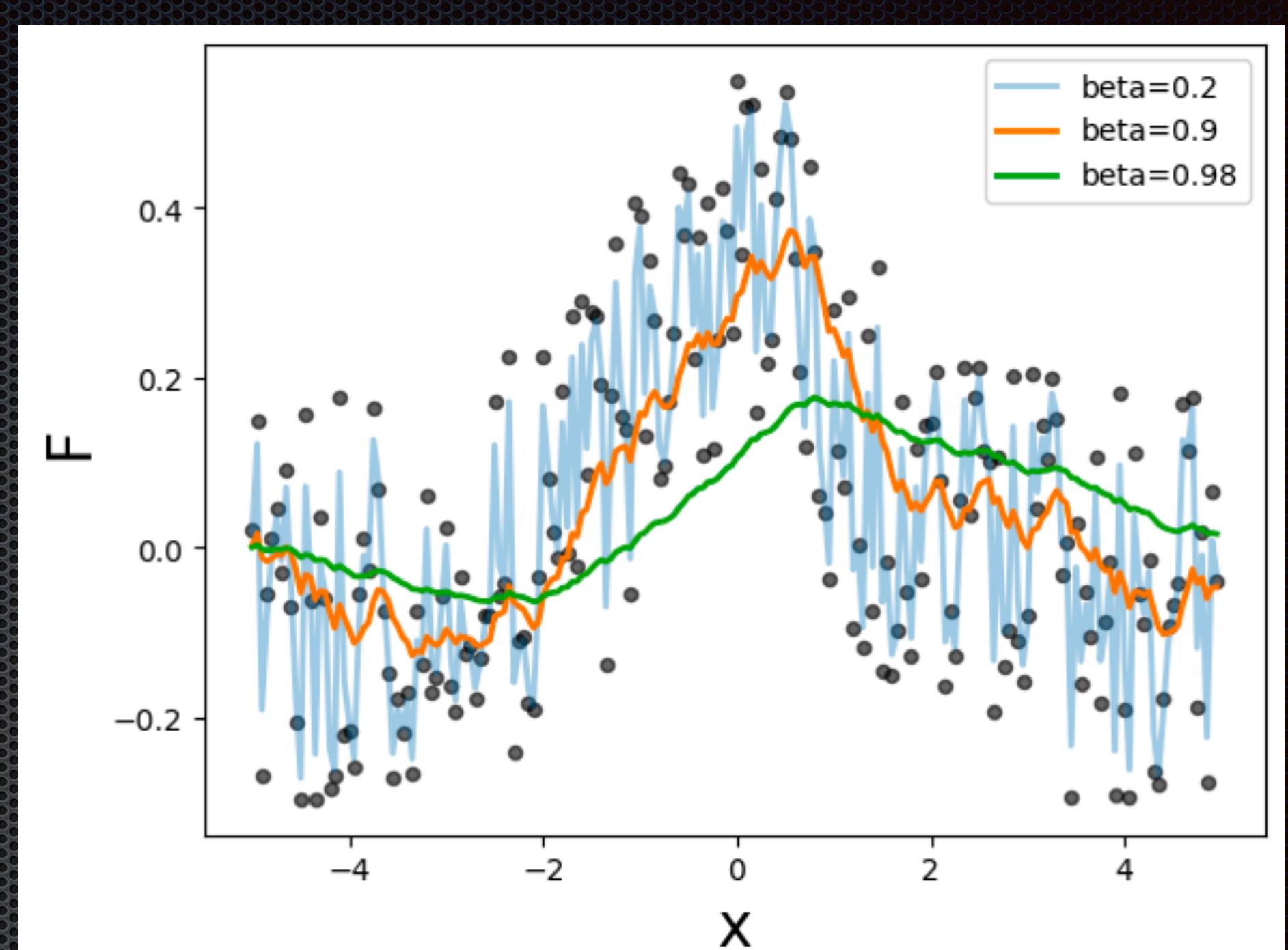
$$\beta \in [0, 1]$$



# Stochastic gradient descent with momentum

The larger the beta the smoother curve

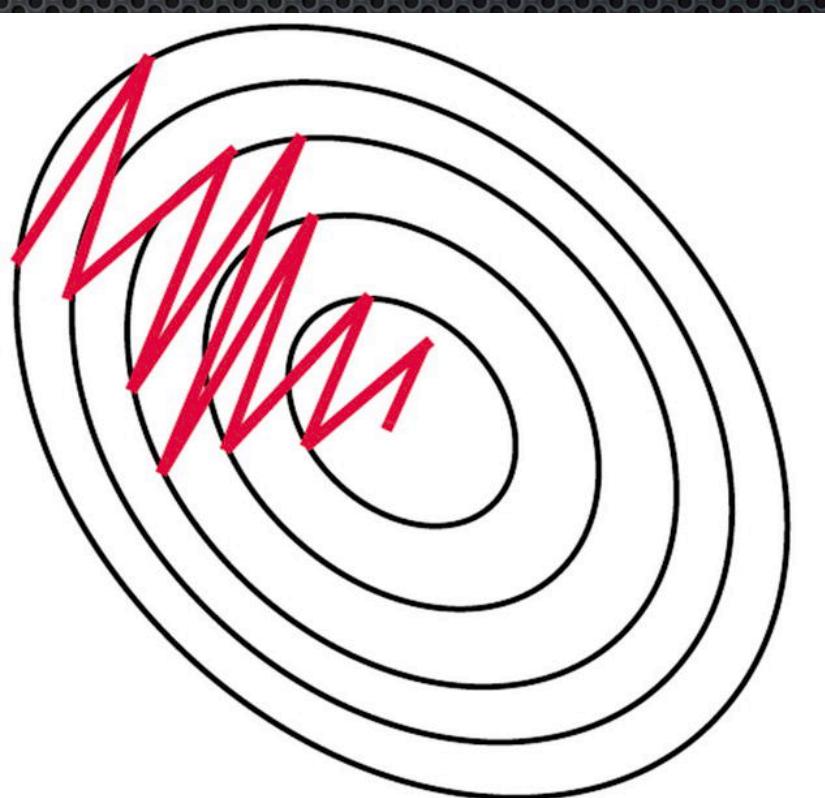
For fixed beta value and less variation, F collapses around zero



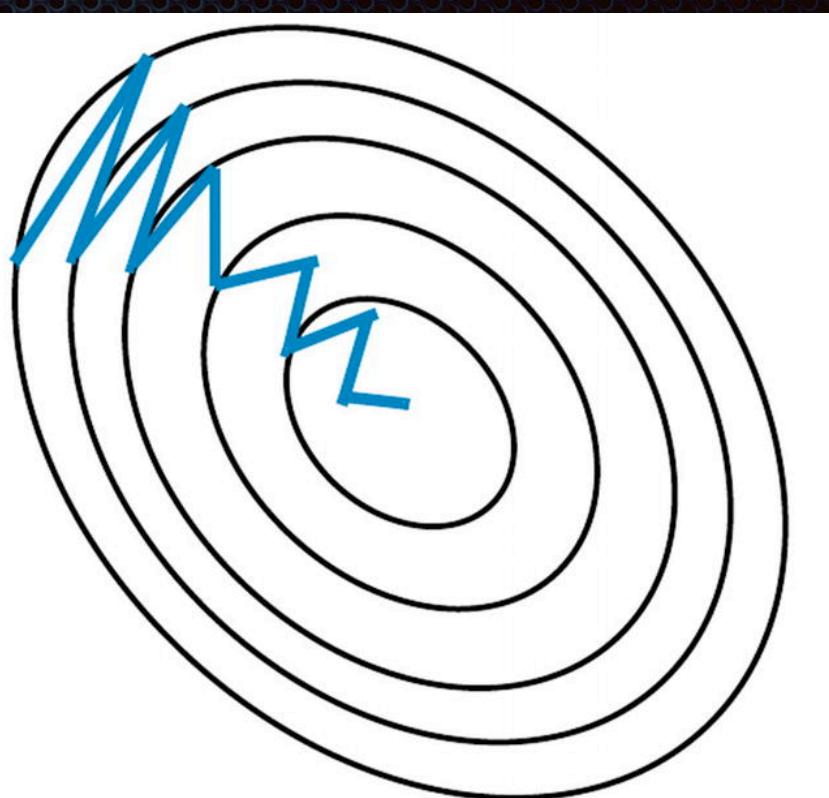
Stochastic gradient descent with momentum:

$$V_t = \beta V_{t-1} + (1 - \beta) \nabla L(\omega)$$
$$\omega^{\text{new}} = \omega^{\text{old}} - \eta V_t$$

Conventionally in ML we use  $\beta = 0.9$



Stochastic Gradient  
Descent **without**  
Momentum



Stochastic Gradient  
Descent **with**  
Momentum

# Adaptive gradient descent (AdaGrad)

<https://optimization.cbe.cornell.edu/index.php?title=AdaGrad>

Whatever the optimizer we learned till SGD with momentum,  
the learning rate remains constant.

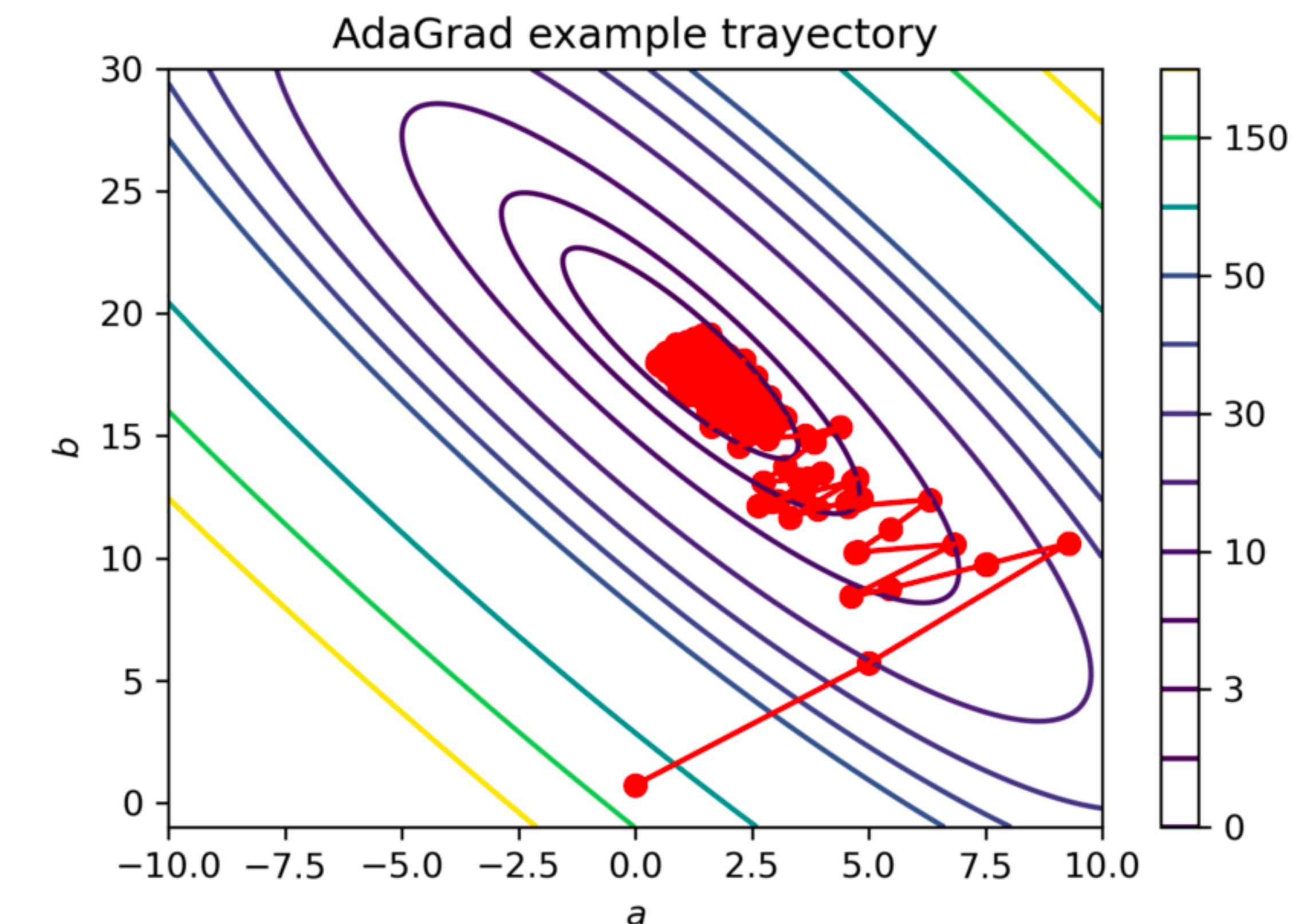
In Adagrad optimizer, there is no momentum concept so,  
it is much simpler compared to SGD with momentum

$$\omega^{\text{new}} = \omega^{\text{old}} - \eta' \nabla L(\omega)$$

$$\eta' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$

Initial chosen learning rate

Regularization parameter  
To avoid the divergence  
In case of vanishing gradient



The learning rate will automatically decrease  
because the summation of the previous gradient square will always  
keep on increasing after every iteration.

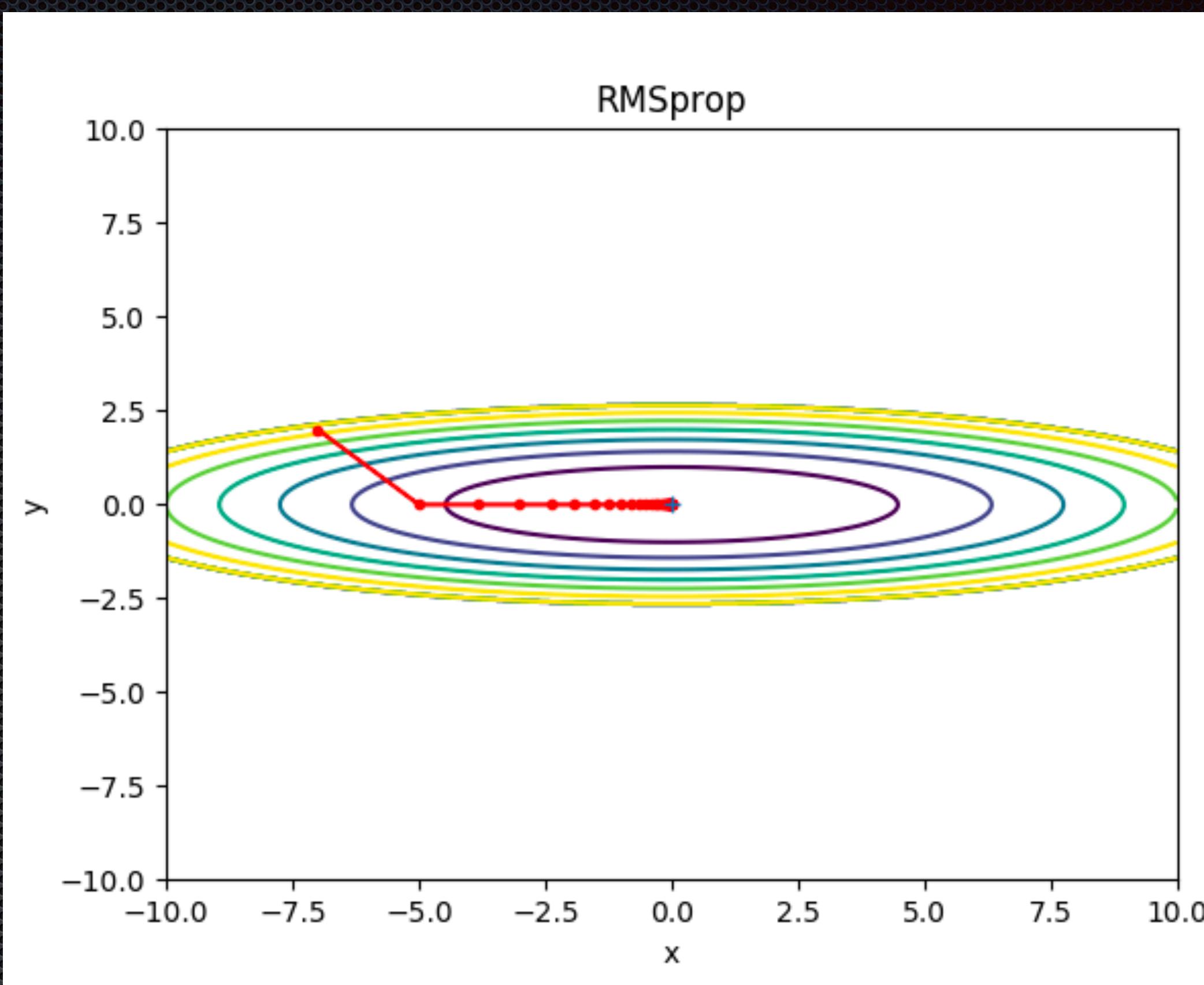
# Root Mean Squared Propagation (RMSprop)

The problem of AdaGrad, however, is that it is incredibly slow.  
this is because the sum of gradient squared only  
grows and never shrinks.

$$\omega_t = \omega_{t-1} - \eta' \nabla L(\omega)$$

$$\eta' = \frac{\eta}{\sqrt{\beta V_t + (1 - \beta)(\nabla L)^2 + \epsilon}}$$

More precisely, the sum of gradient squared is actually the decayed sum of gradient squared. The decay rate is saying only recent gradient<sup>2</sup> matters, and the ones from long ago are basically forgotten.

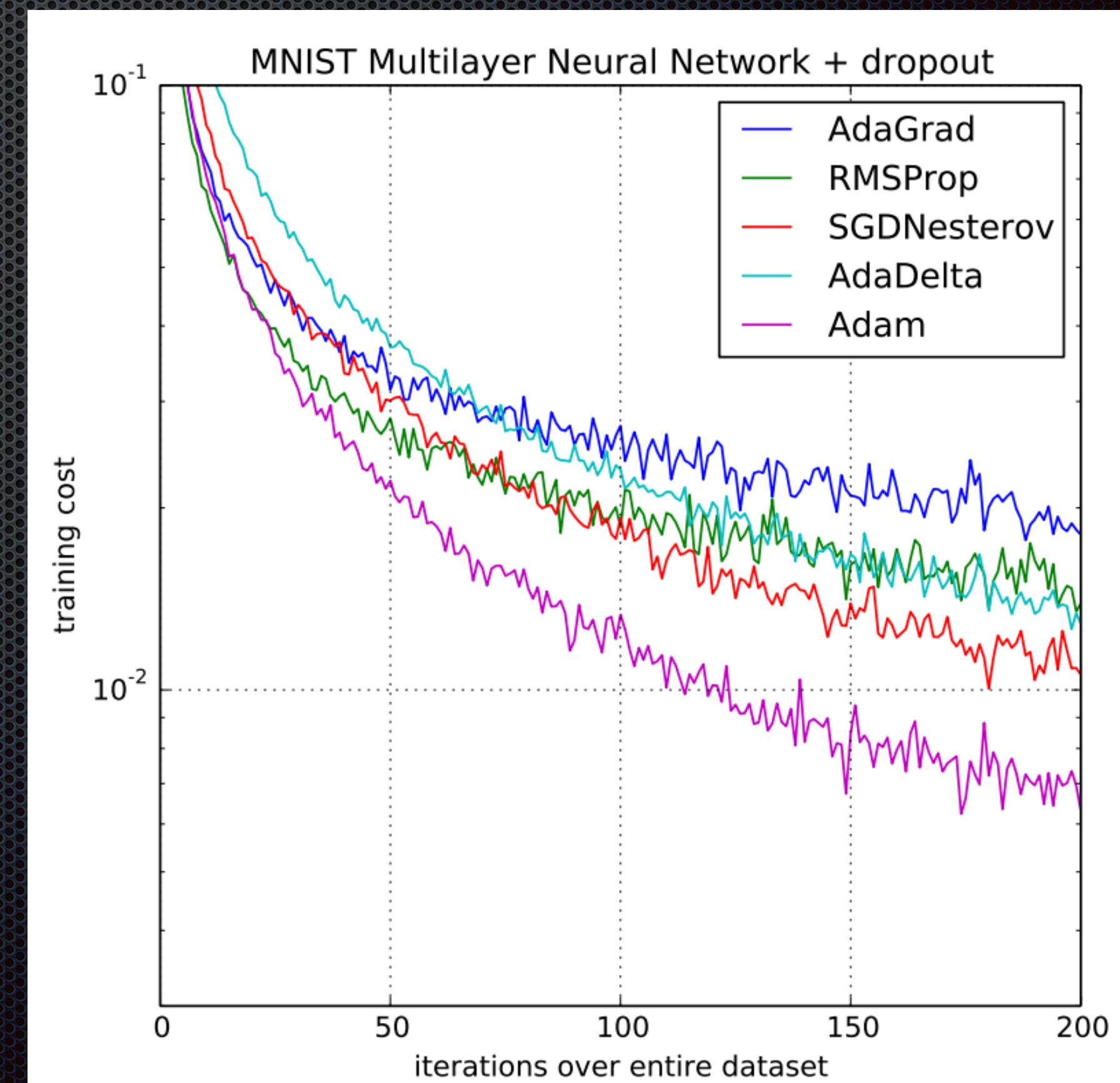


# Adaptive Moment Estimation (Adam)

Adam takes the best of both worlds of Momentum and RMSProp. Adam empirically works well, and thus in recent years, it is commonly the go-to choice of deep learning problems.

arXiv:1412.6980v

$$\omega_t = \omega_{t-1} - \frac{\eta (\beta_1 V_{t-1} + (1 - \beta_1) \nabla L)}{\sqrt{\beta_2 V_{t-1} + (1 - \beta_2)(\nabla L)^2} + \epsilon}$$



# Deep models convergence

Code:

[https://github.com/lilipads/gradient\\_descent\\_viz/tree/master](https://github.com/lilipads/gradient_descent_viz/tree/master)

## Function with local minimum

— Gradient descent

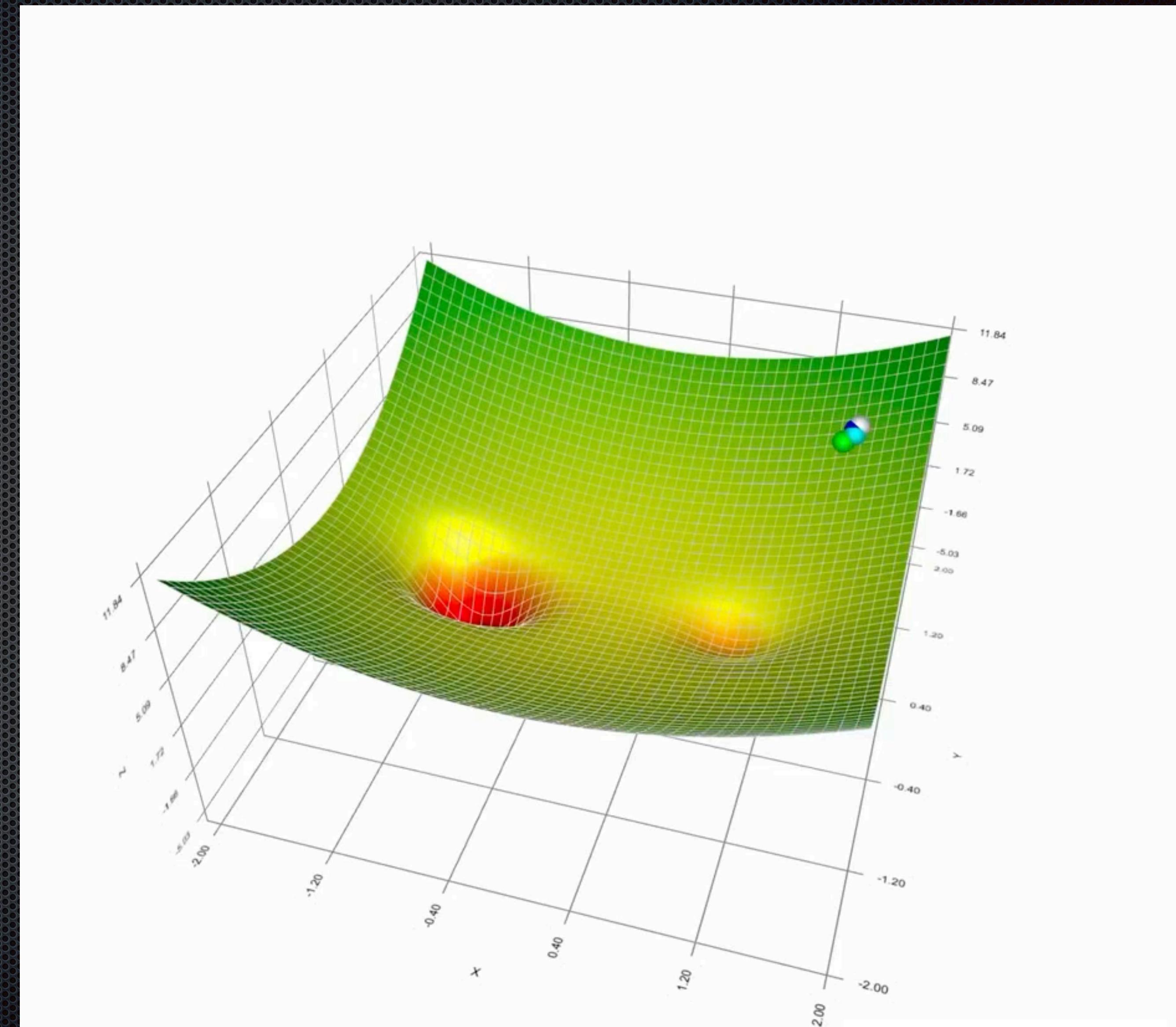
— Gradient descent with momentum

— Adagrad

— RMSPROP

— Adam

Initial Learning rate = 0.01



# Deep models convergence

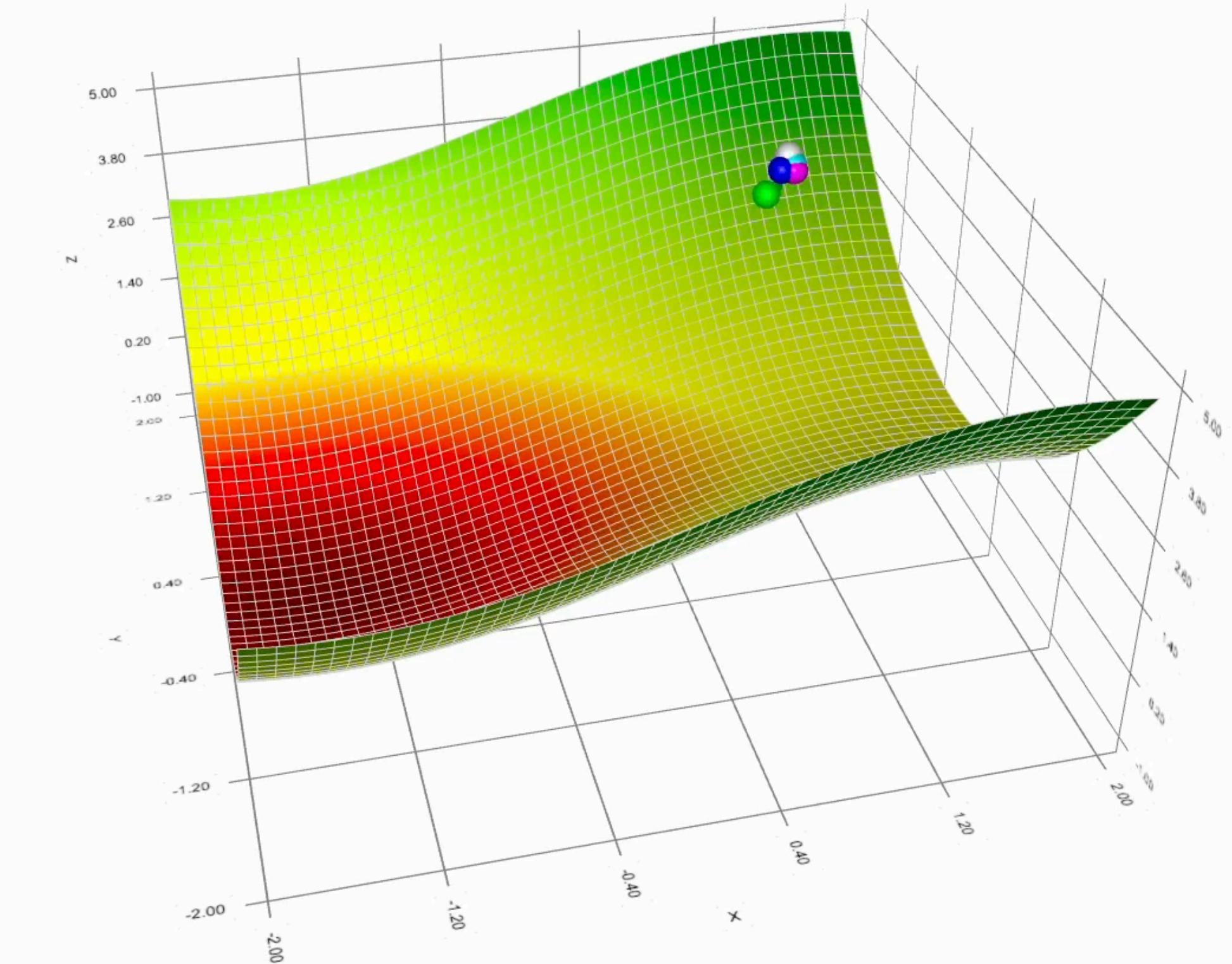
Code:

[https://github.com/lilipads/gradient\\_descent\\_viz/tree/master](https://github.com/lilipads/gradient_descent_viz/tree/master)

## Function with saddle point

- Gradient descent
- Gradient descent with momentum
- Adagrad
- RMSProp
- Adam

Initial Learning rate = 0.01



# Deep models convergence

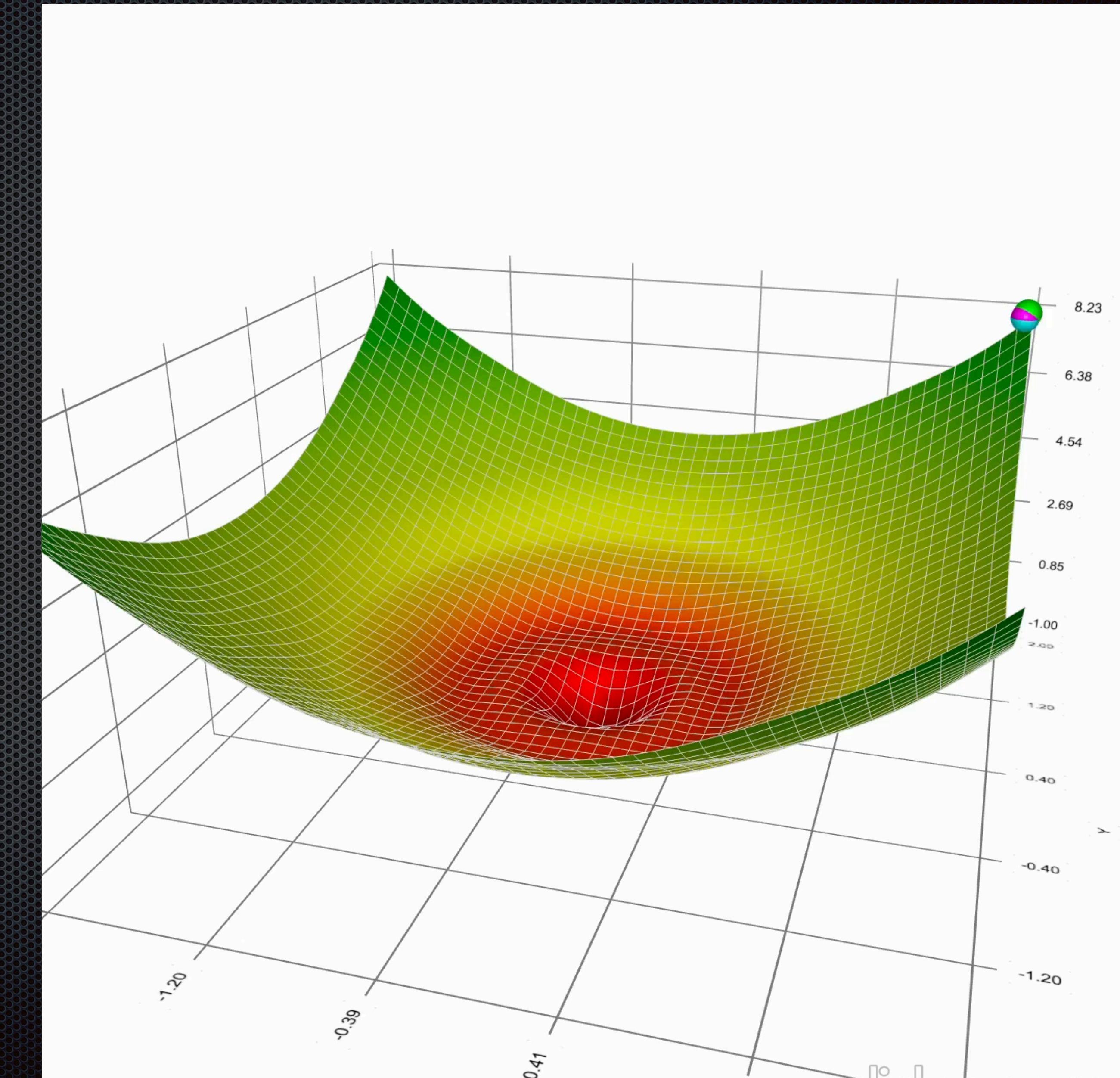
Code:

[https://github.com/lilipads/gradient\\_descent\\_viz/tree/master](https://github.com/lilipads/gradient_descent_viz/tree/master)

## Function with plateau

- Gradient descent
- Gradient descent with momentum
- Adagrad
- RMSPROP
- Adam

Initial Learning rate = 0.01



# Deep models convergence

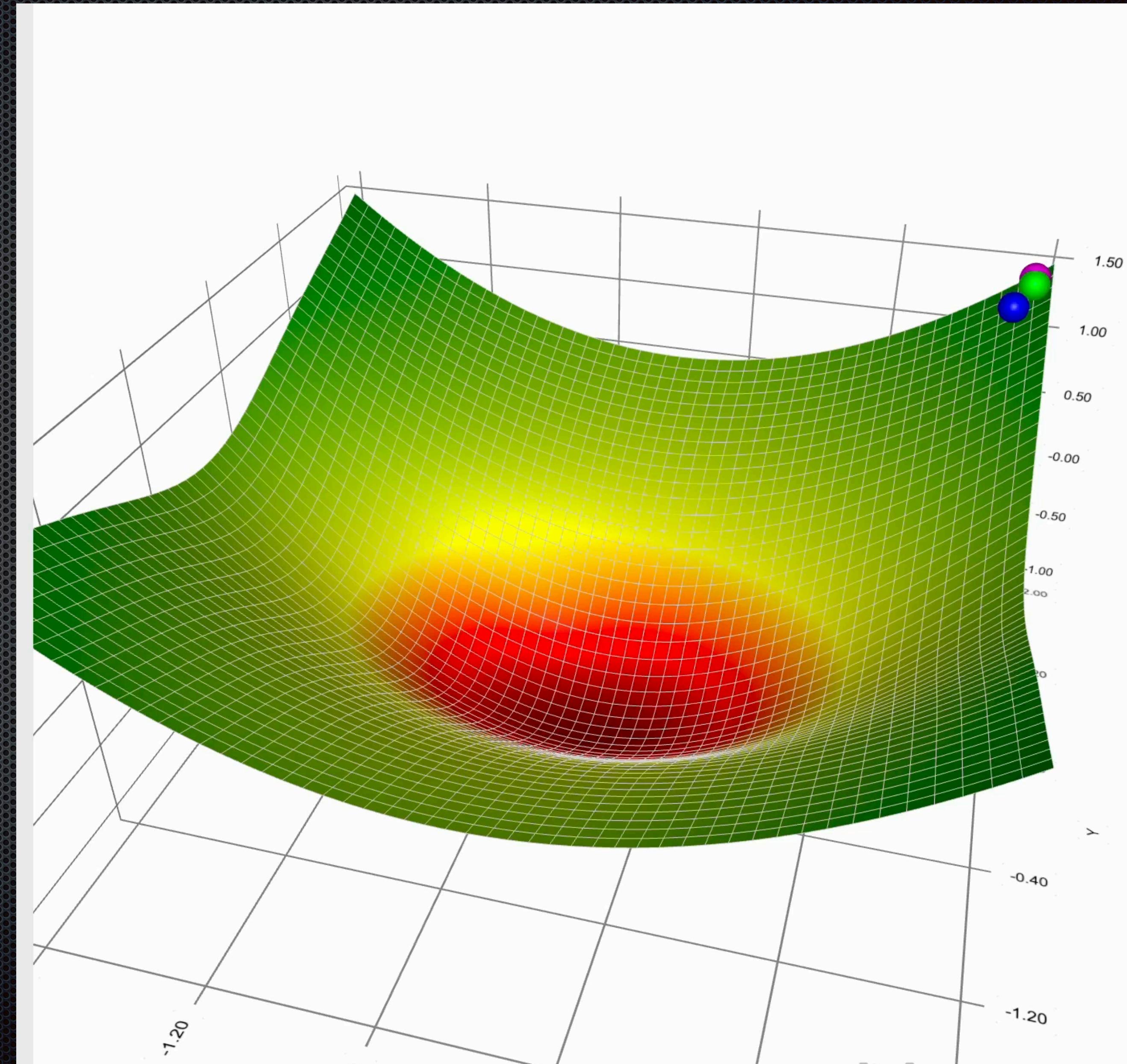
Code:

[https://github.com/lilipads/gradient\\_descent\\_viz/tree/master](https://github.com/lilipads/gradient_descent_viz/tree/master)

## Function with Ecliptic Bowl

- Gradient descent
- Gradient descent with momentum
- Adagrad
- RMSPROP
- Adam

Initial Learning rate = 0.01



# *Exploratory data analysis (EDA)*

*Prepare the data for deep learning models*

- Data Cleaning
- Handling text and categorical attributes
- Features scaling

# Data cleaning

- Handle the missing data

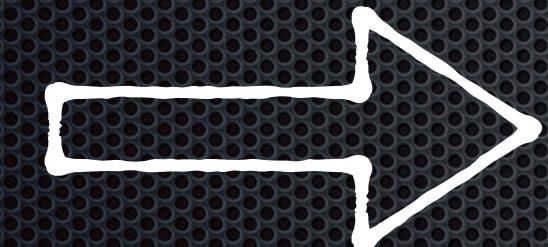
Working on real life data, most of the time the data are not standard.

Most of the collected real data suffer from missing information

As ML models cannot process with missing values

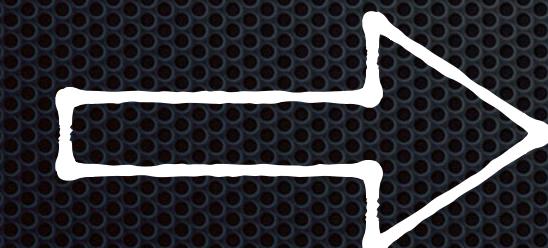
Easy to be handled in Pandas

Pandas.isnull()



Find the missing values

Pandas.fillna()



Fill the missing values by extrapolation between the last values

Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
3	Braund, Mr. Owen Harris	male	22.0	1	0.0	A/5 21171	7.2500		S
1	Cumings, Mrs. John Bradley (Florence Briggs Th... er)	female	38.0	1	0.0	3101282	71.2833	C85	C
3	Heikkinen, Miss. Laina	female		0	0.0				S
1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0.0	113803	53.1000	C123	S
3	Allen, Mr. William Henry	male	35.0	0	0.0	373450	8.0500		S
2	Montvila, Rev. Juozas	male	27.0	0	0.0	211536	13.0000		S
1	Graham, Miss. Margaret Edith	female	19.0	0	0.0	112053	30.0000	B42	S
3	Johnston, Miss. Catherine Helen "Carrie"	female		1	2.0	W/C. 6607	23.4500		S
1	Behr, Mr. Karl Howell	male	26.0	0	0.0	111369	30.0000	C148	C
3	Dooley, Mr. Patrick	male	32.0	0	0.0	370376	7.7500		Q

# Data cleaning

- Feature importance

Deep models learn from large data and keeping more redundant information into the data consume the computational resources and may hinder the model accuracy

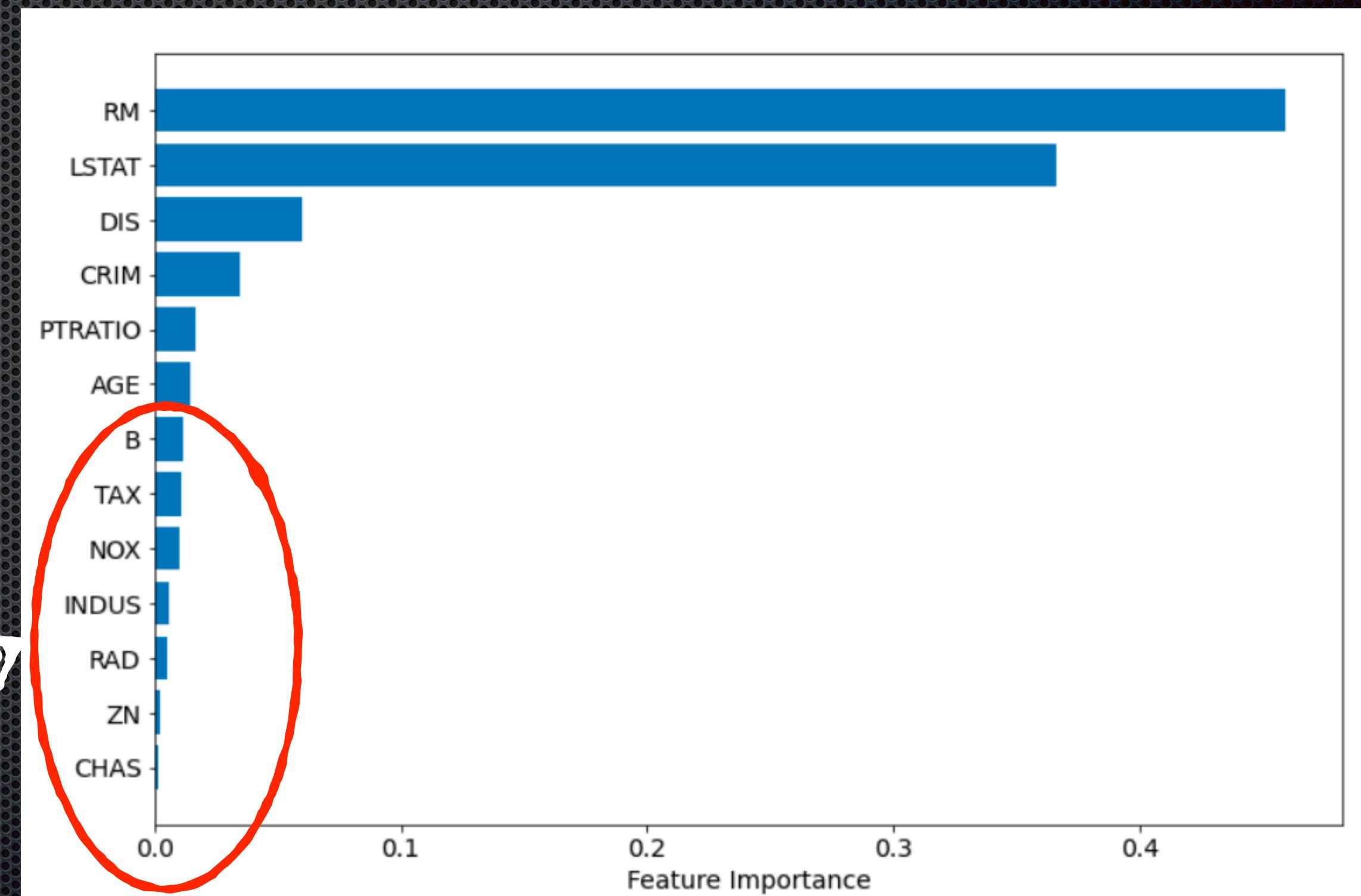
Two ways for the selection

Forward selection

Reverse selection



These features can be removed



# Handling text data

- One hot encoding

Real life data mostly contains, texts, numbers, characters, etc  
ML models deal with numbers only.

Color	Red	Yellow	Green
Red	1	0	0
Red	1	0	0
Yellow	0	1	0
Green	0	0	1
Yellow	0	0	1

For multi-class classification problem with cross entropy loss Labels (targets) need to be one hot encoded

3 4 2 1 9 5 6 2 1 8  
8 9 1 2 5 0 0 6 6 4  
6 7 0 1 6 3 6 3 7 0  
3 7 7 9 4 6 6 1 8 2  
2 9 3 4 3 9 8 7 2 5  
1 5 9 8 3 6 5 7 2 3  
9 3 1 9 1 5 8 0 8 4  
5 6 2 6 8 5 8 8 9 9  
3 7 7 0 9 4 8 5 4 3  
7 9 6 4 7 0 6 9 2 3

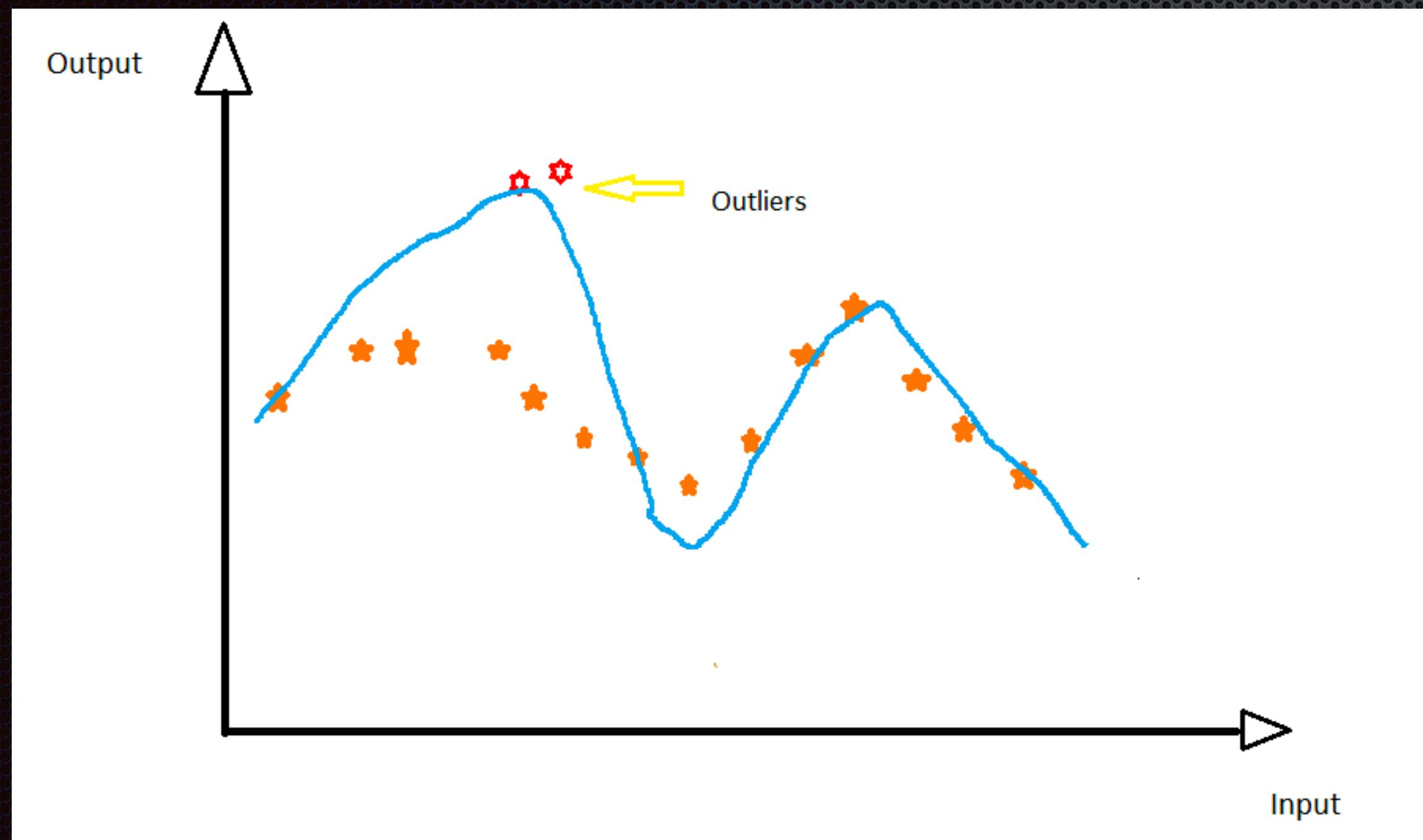
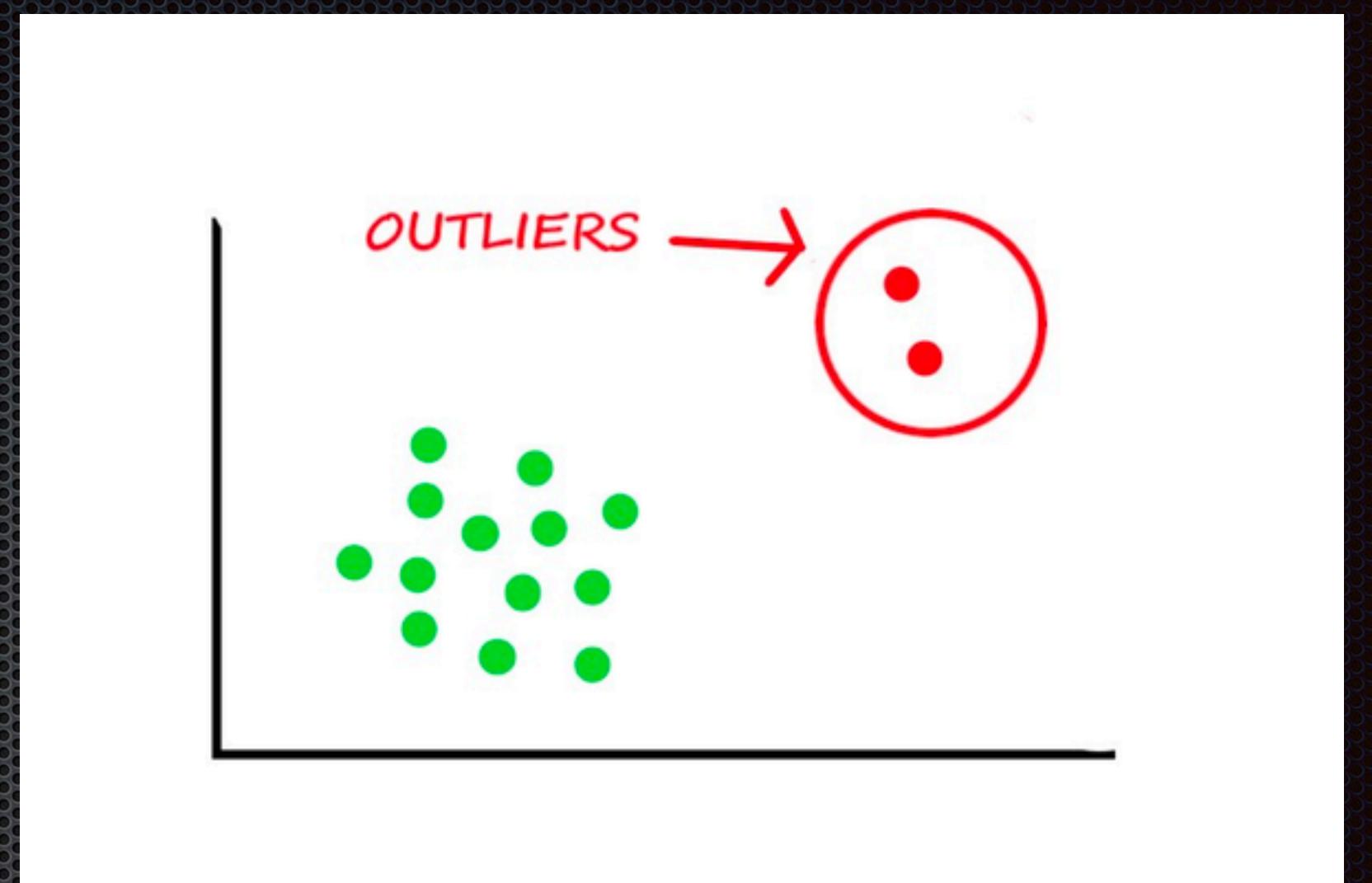
Labels

	0	1	2	3	4	5	6	7	8	9	label
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	5
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	4
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	9
5	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2
6	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
7	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	3
8	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
9	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	4

# Features scaling

- Outliers

Input data contains different scales and mostly contains outliers

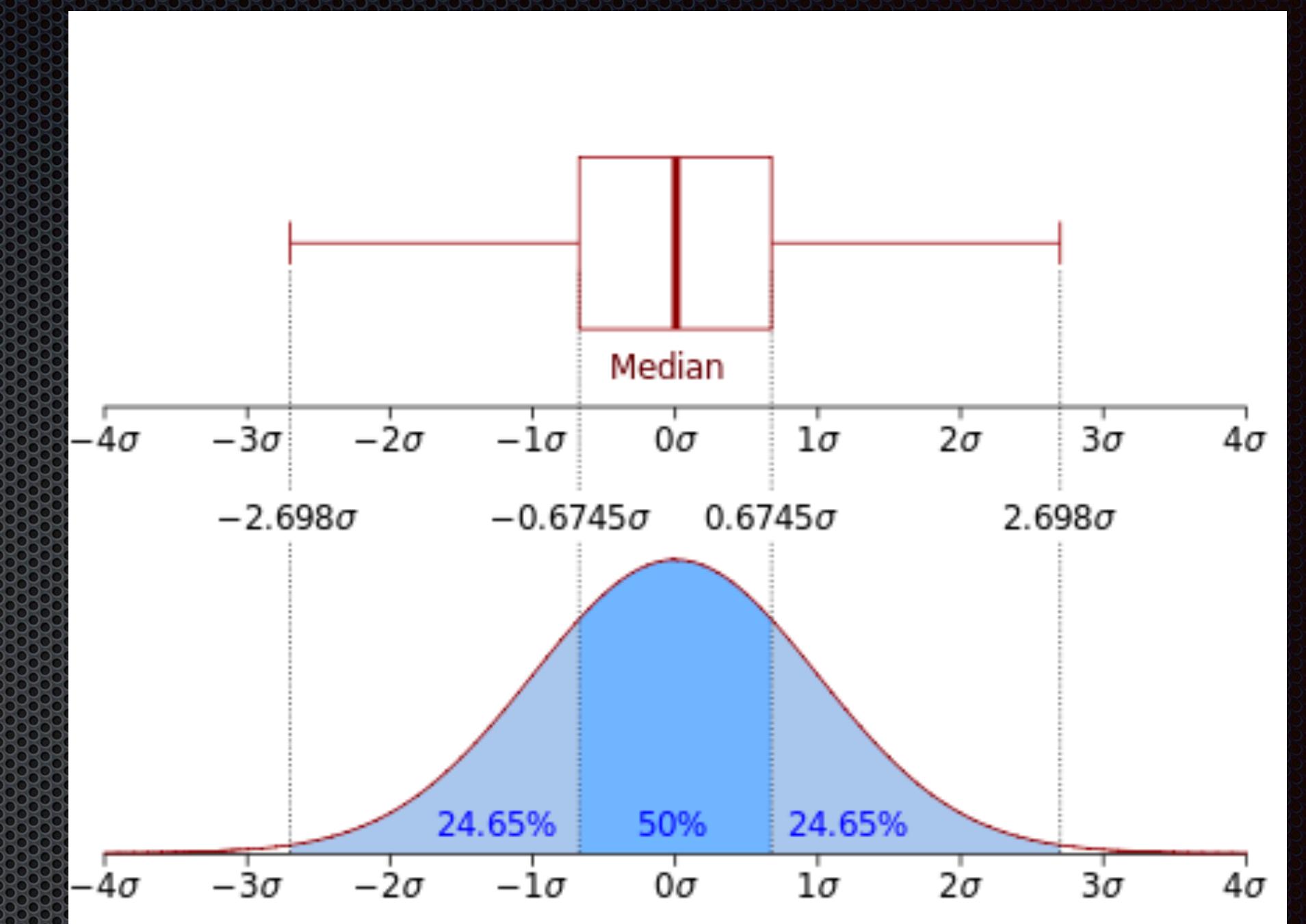
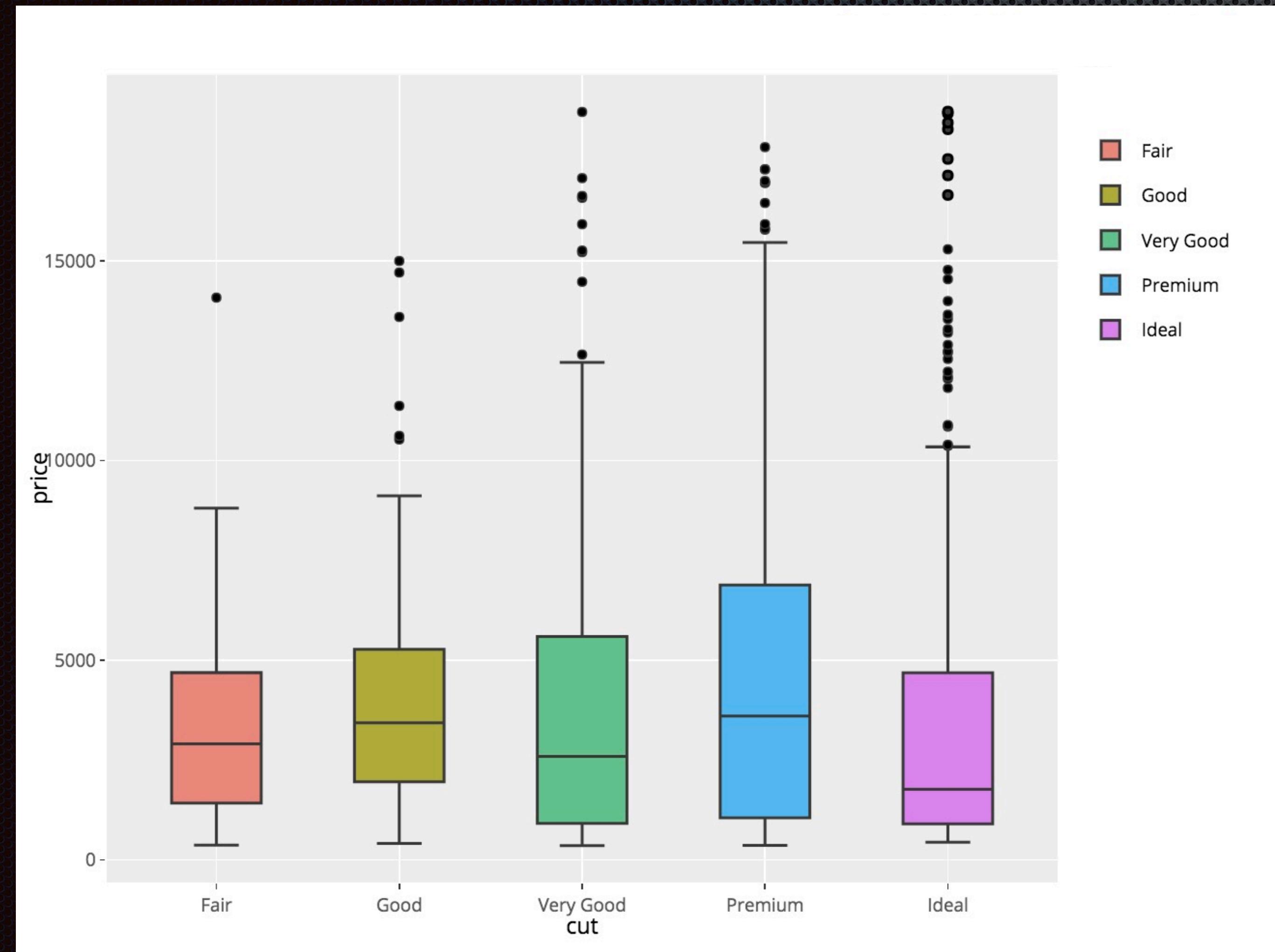


Data with outliers the ML model focuses  
in learning the features of the outlier points  
Not the features of the whole dataset

# Features scaling

- Handling the Outliers

If the size of the outliers is small one can easily remove them



Box diagram for outliers detection

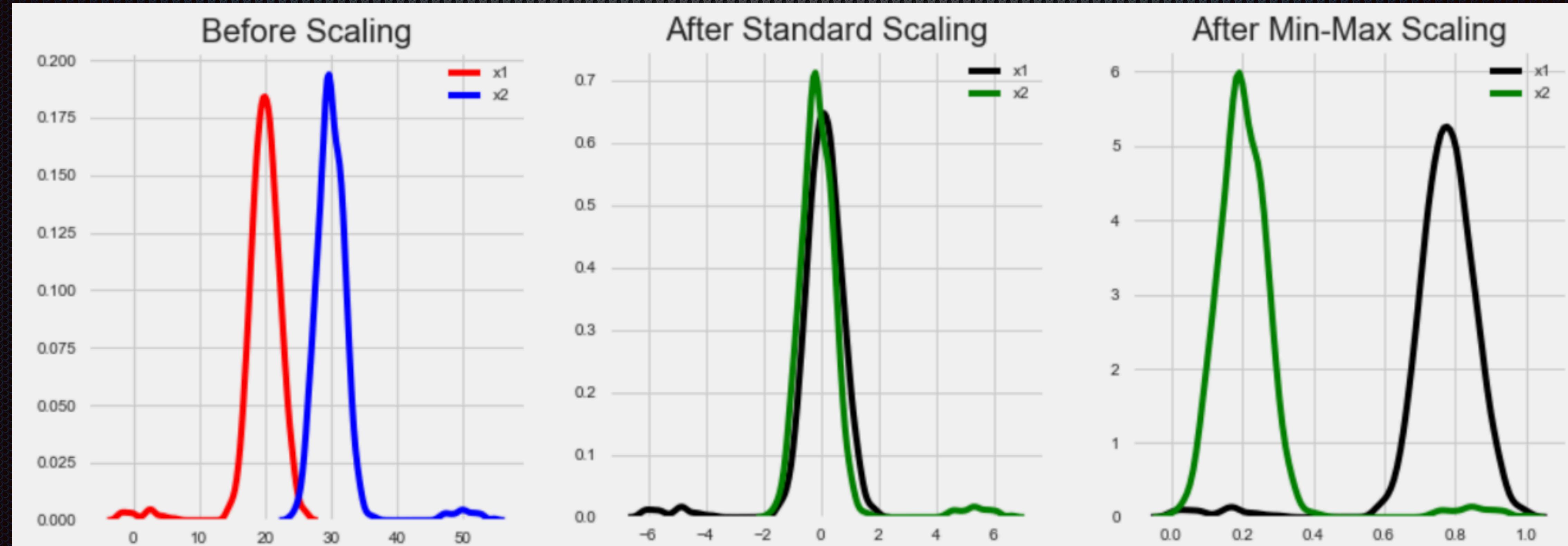
# Features scaling

- Handling the Outliers

For large amount of outliers, scaling is needed

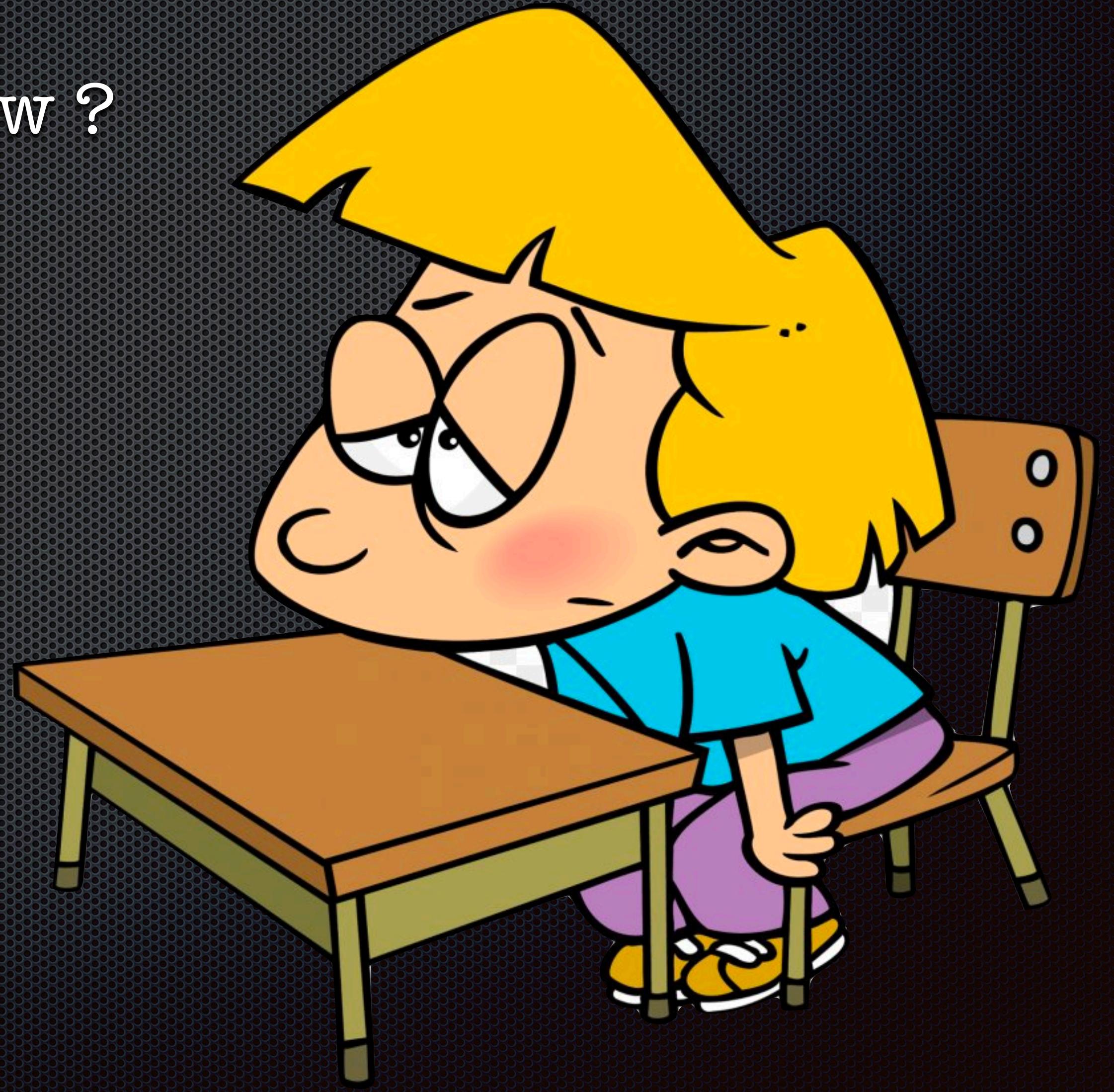
$$x^{\text{stand}} = \frac{x - \mu}{\sigma}$$

$$x^{\text{minmax}} = \frac{x - x^{\text{min}}}{x^{\text{max}} - x^{\text{min}}}$$

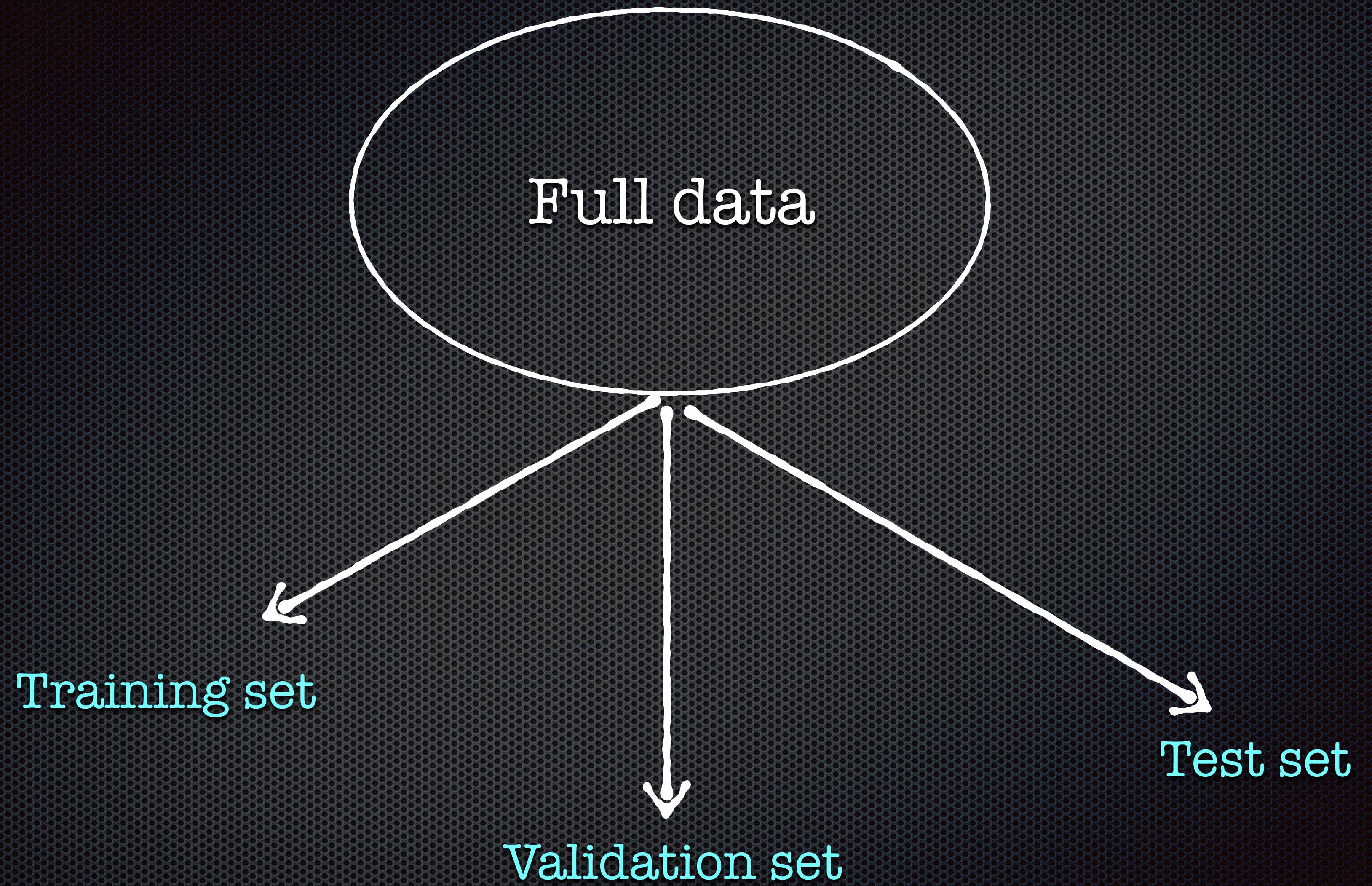


# Deep learning

Well, can we train the DL model now ?



# Data Splitting



# *Imbalanced training set*

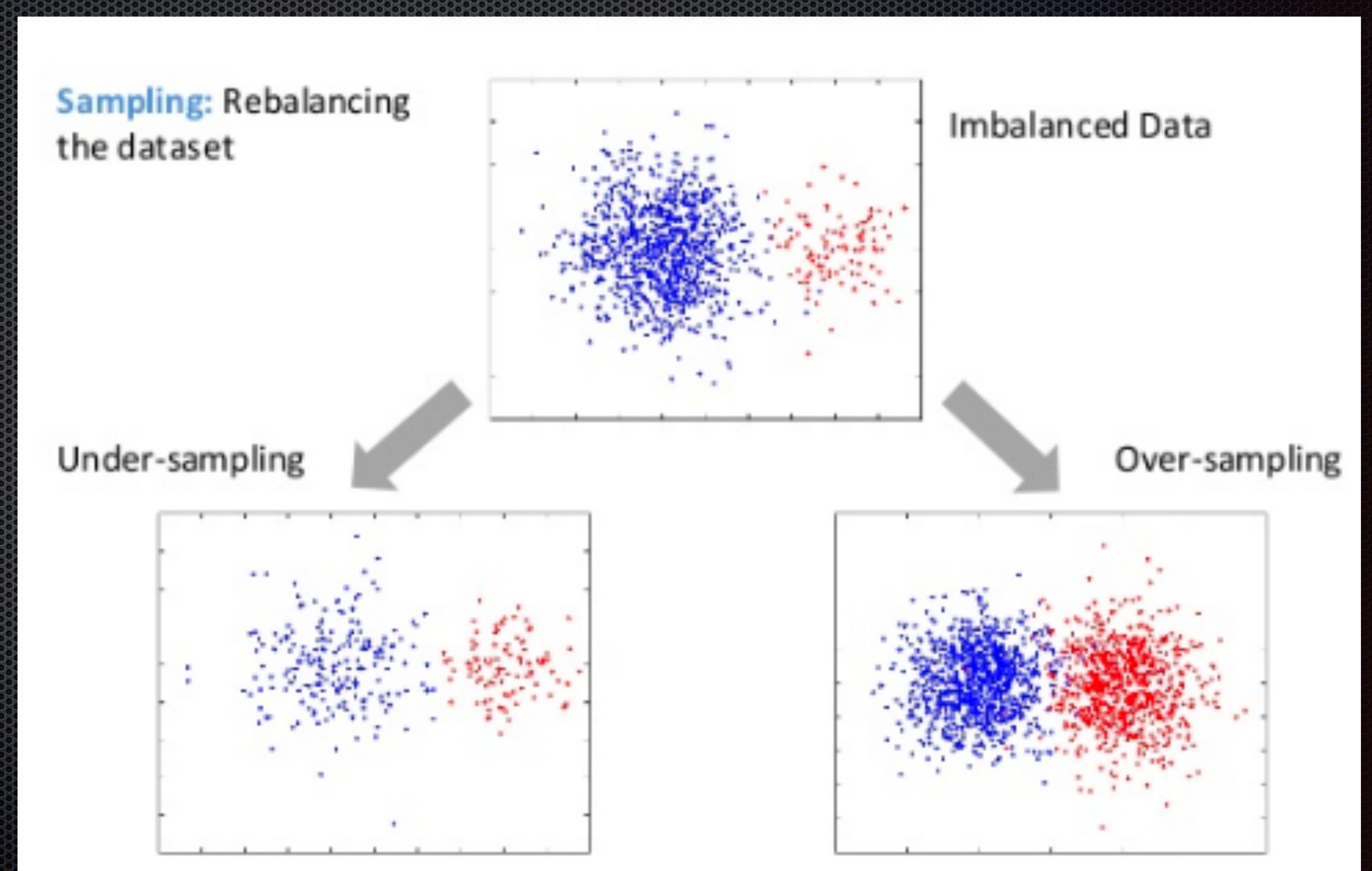
For the classification problem we need to train the model on equal size data from each class. For imbalanced data sets the model learns the features of the majority class over the minority one

Undersampling the majority class by  
Randomly removing its data

We loose important information  
about the majority class

Upsampling the minority class by  
Randomly copying its data

We add redundant information !!

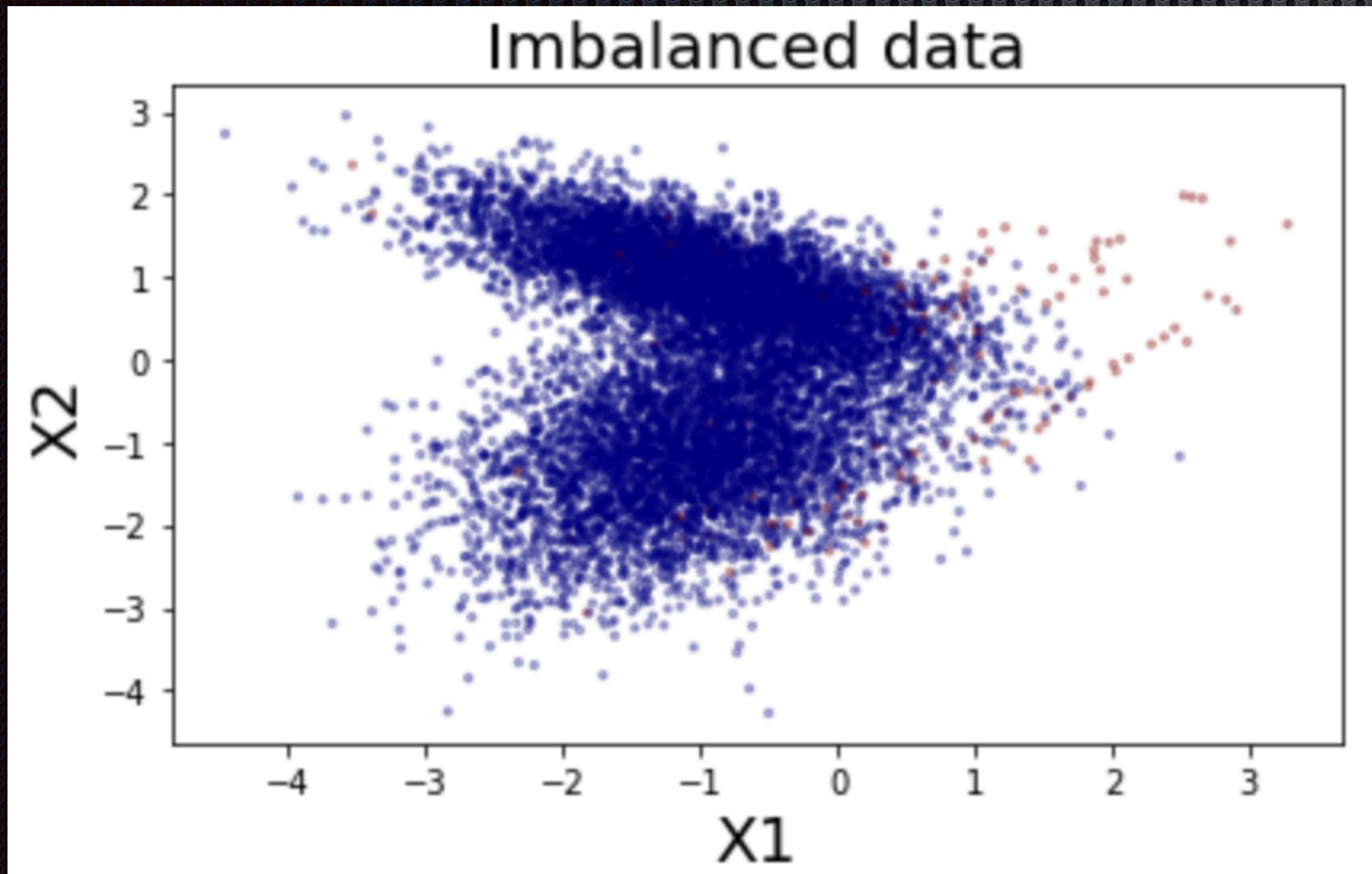


# *Imbalanced training set*

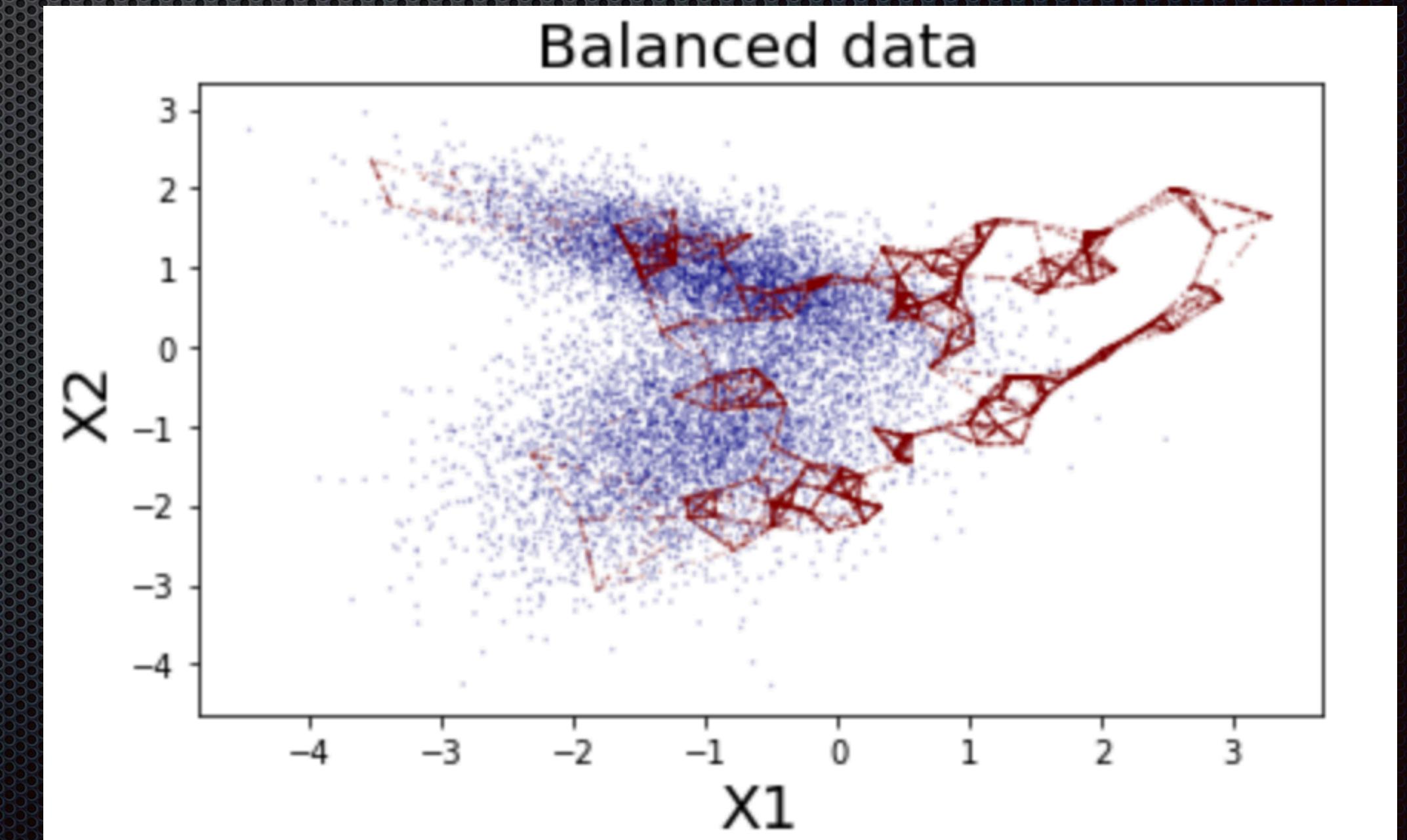
## Synthetic Minority Over-sampling Technique (**SMOTE**)

SMOTE upsample the minority class by generating synthetic points by generating new points along the joint lines between the minor data

Before SMOTE

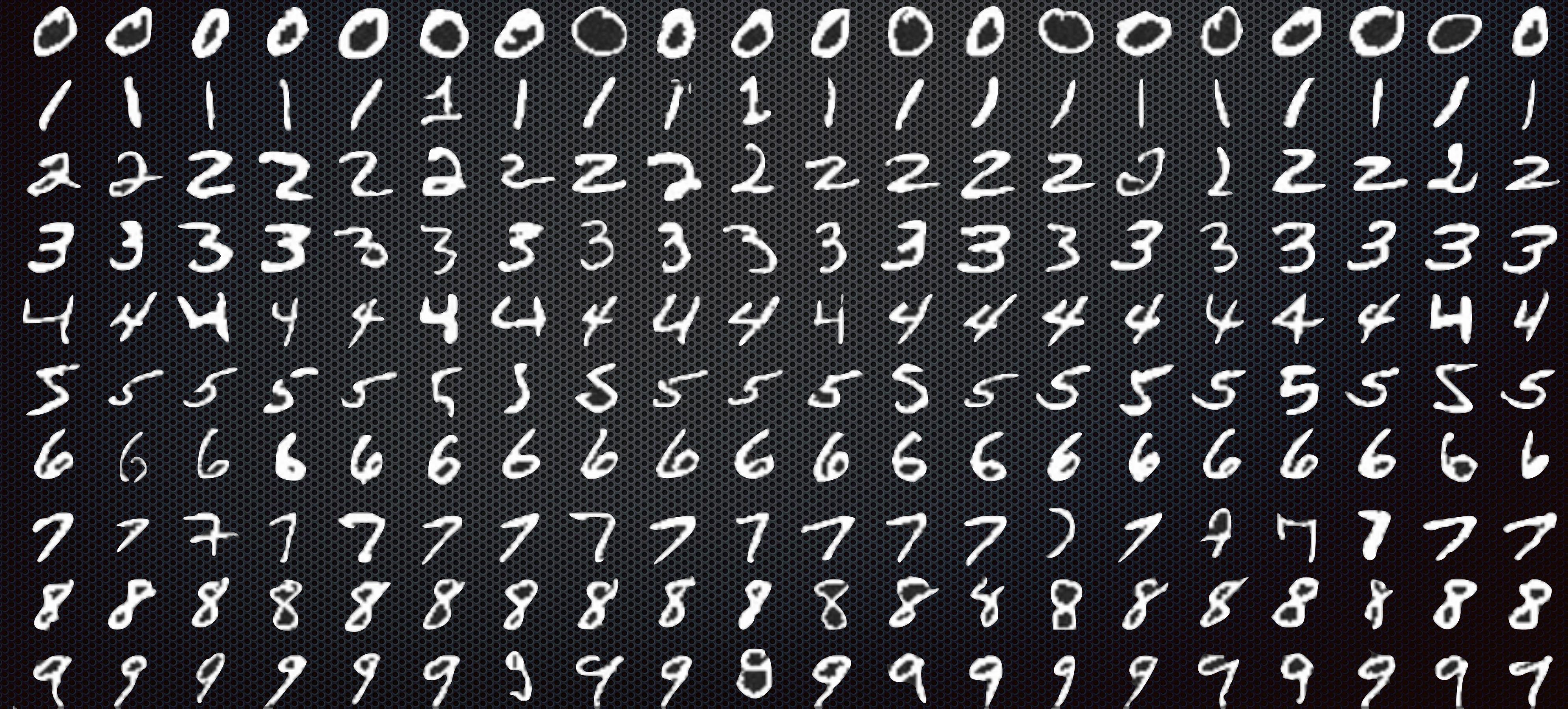


After SMOTE

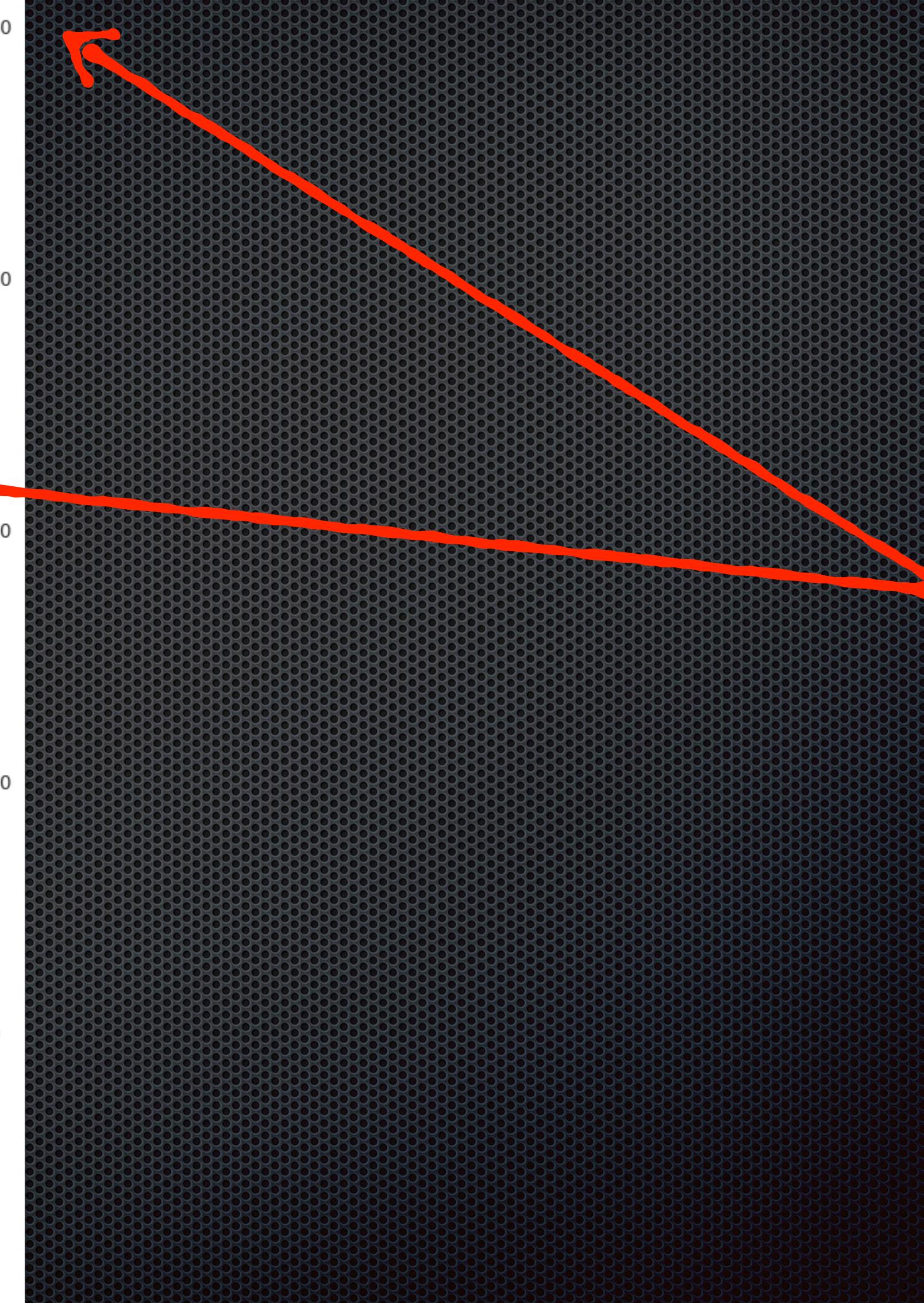
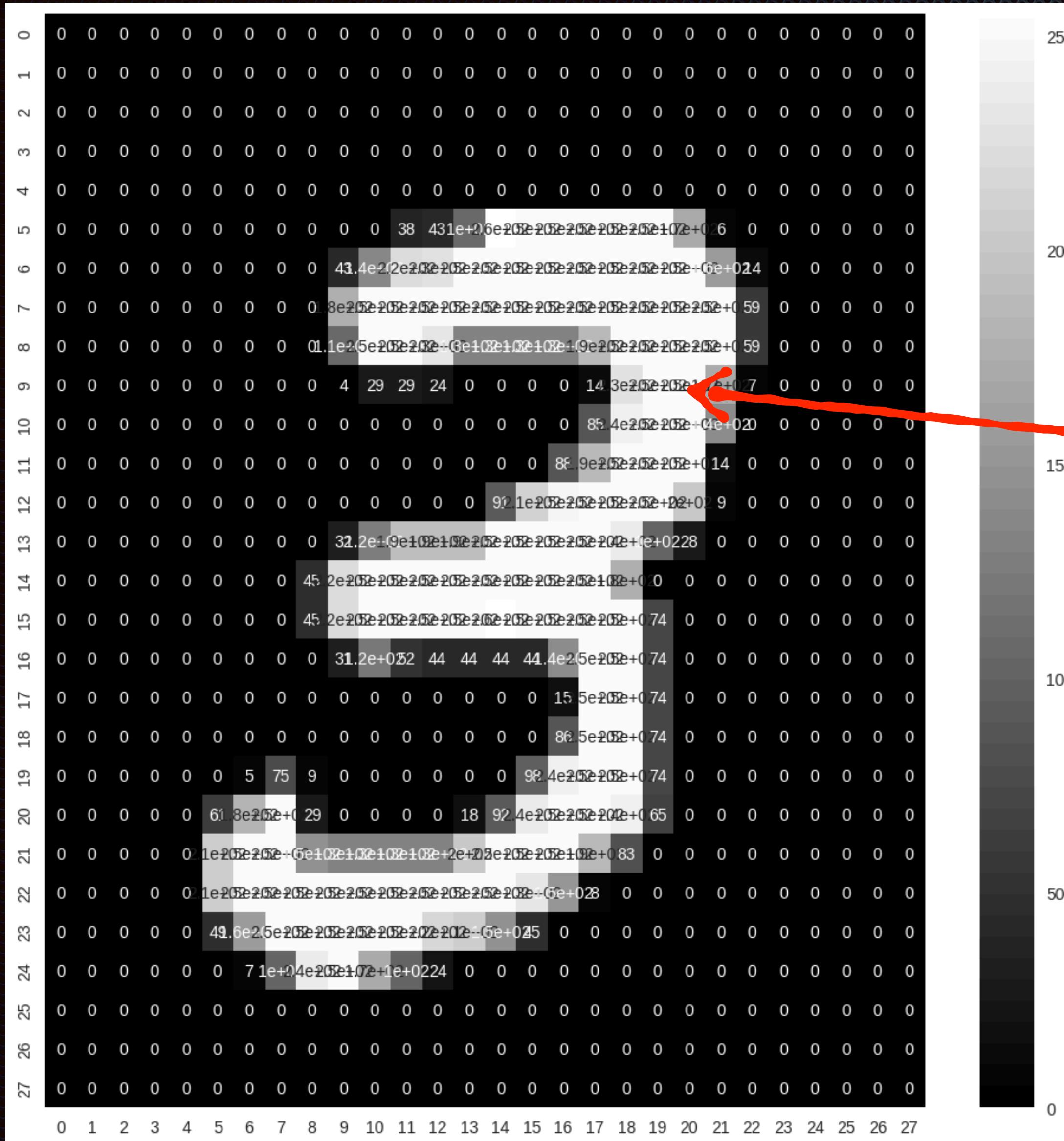


# Example - MNIST data

Task: Construct DNN model to classify the hand written digits

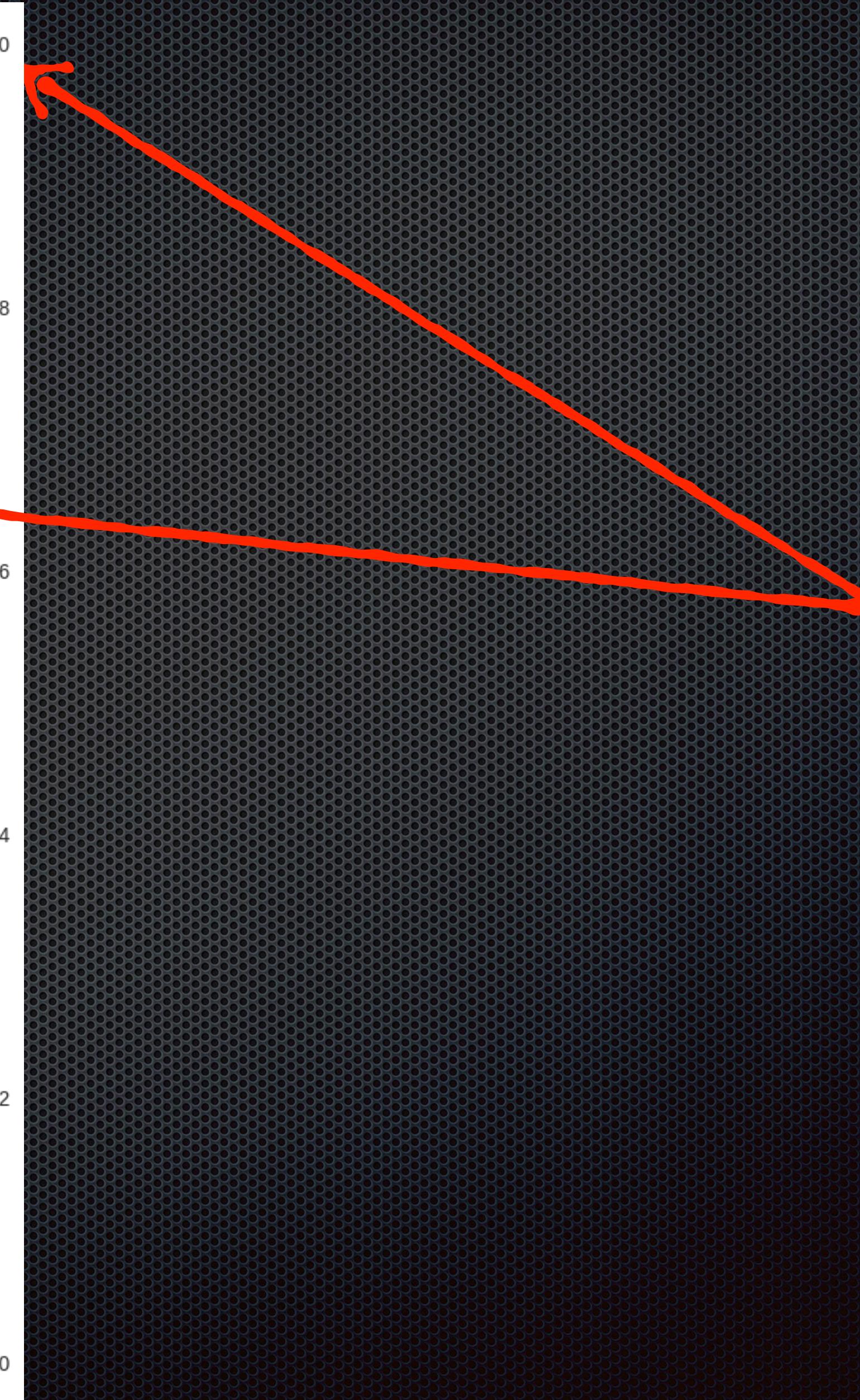
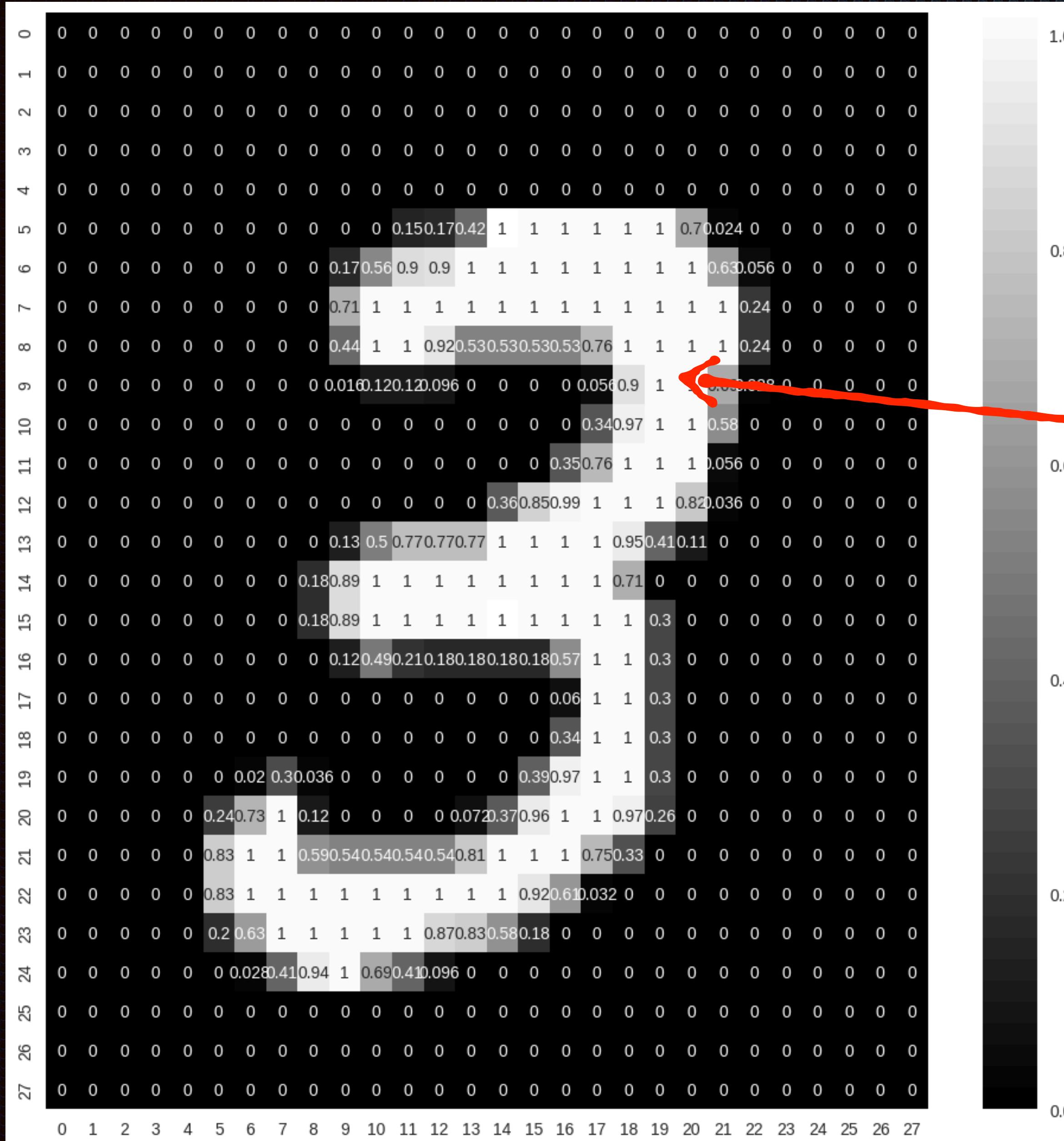


# Example - MNIST data



Different scale  
Of the pixels

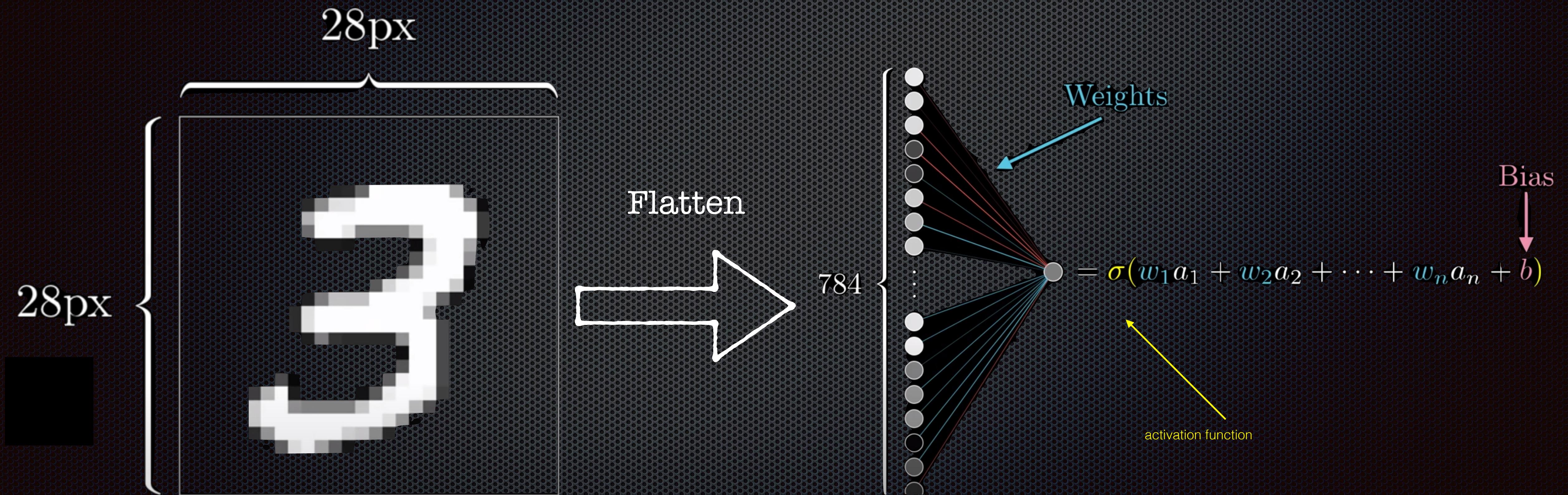
# Example - MNIST data



# After min-max Normalization

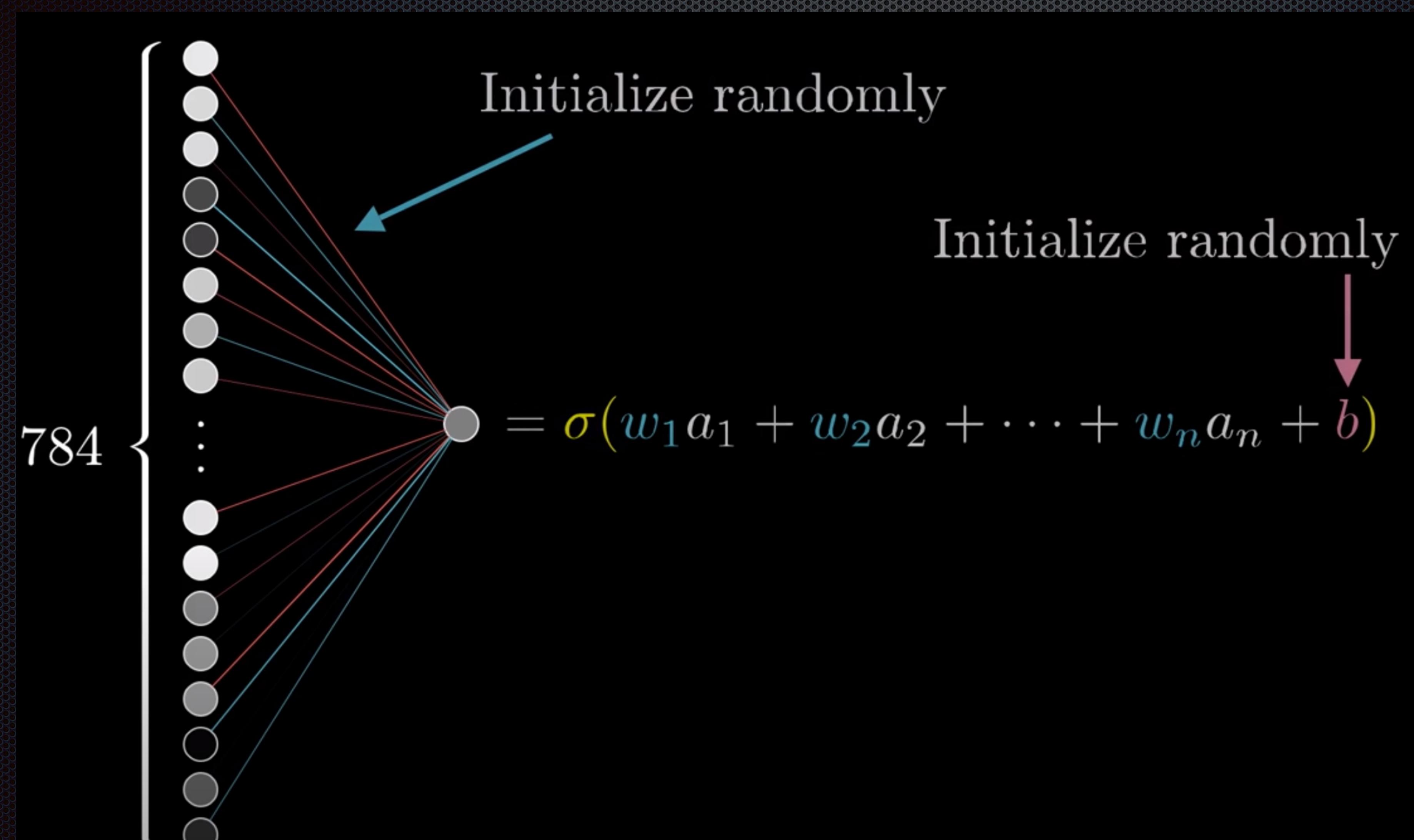
# Example - MNIST data

Construct the first neuron in the first layer



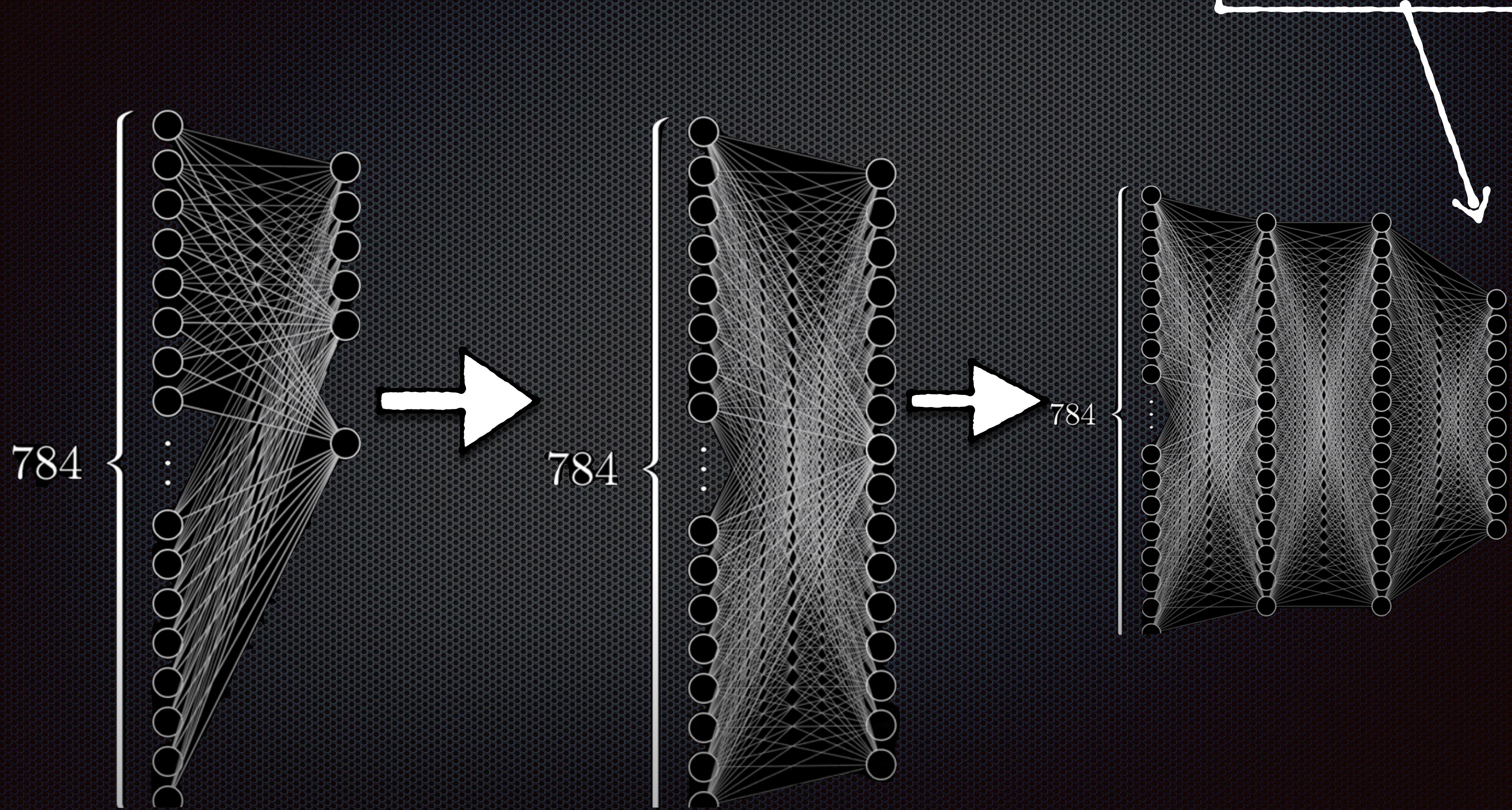
# Example - MNIST data

For the first iteration only: Initialize the weights and bias randomly



# Example - MNIST data

Keep constructing one layer after another



# Example - MNIST data

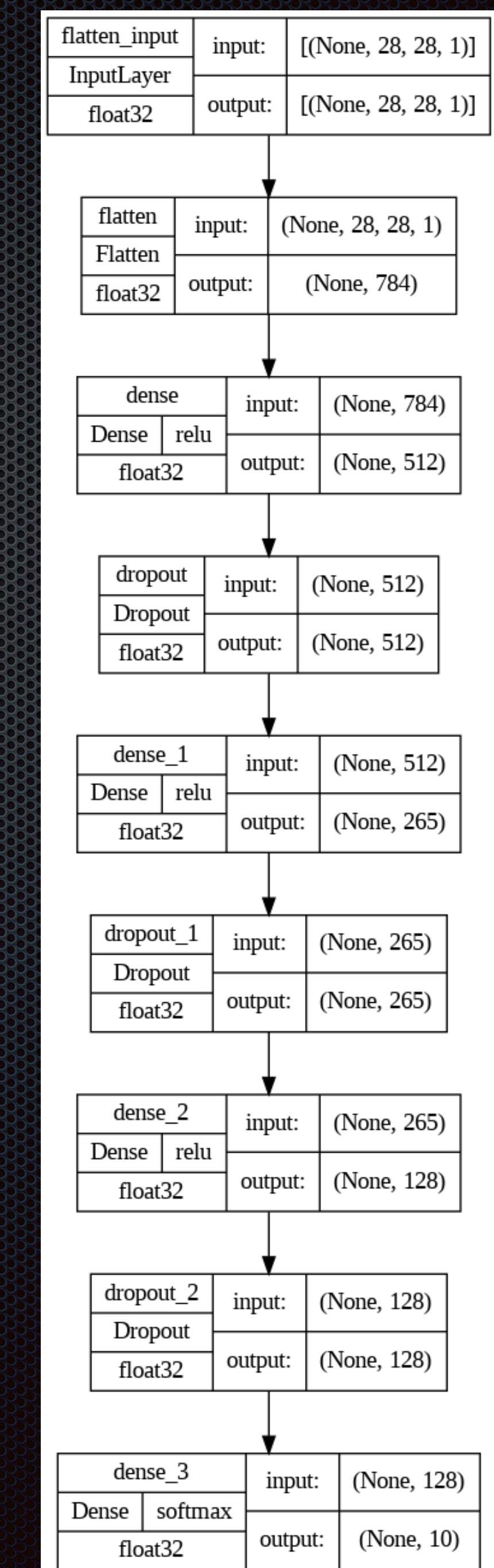
## Model summary

```
Model: "sequential"

Layer (type)          Output Shape       Param #
=====              ======           =====
flatten (Flatten)      (None, 784)        0
dense (Dense)          (None, 512)       401920
dropout (Dropout)      (None, 512)        0
dense_1 (Dense)        (None, 265)       135945
dropout_1 (Dropout)    (None, 265)        0
dense_2 (Dense)        (None, 128)        34048
dropout_2 (Dropout)    (None, 128)        0
dense_3 (Dense)        (None, 10)         1290

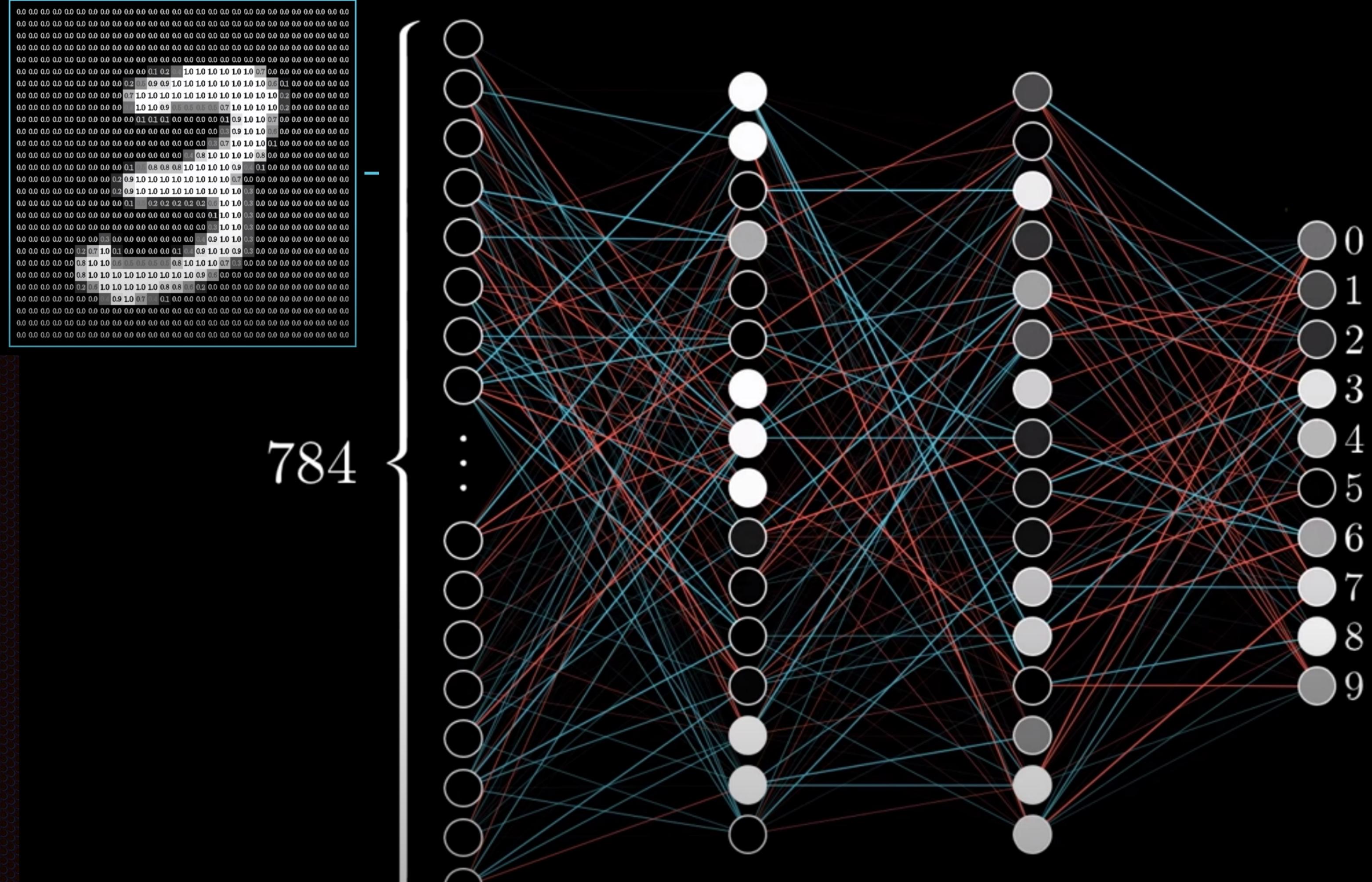
Total params: 573,203
Trainable params: 573,203
Non-trainable params: 0
```

More detailed



# Example - MNIST data

Forward propagation of the inputs



Model output

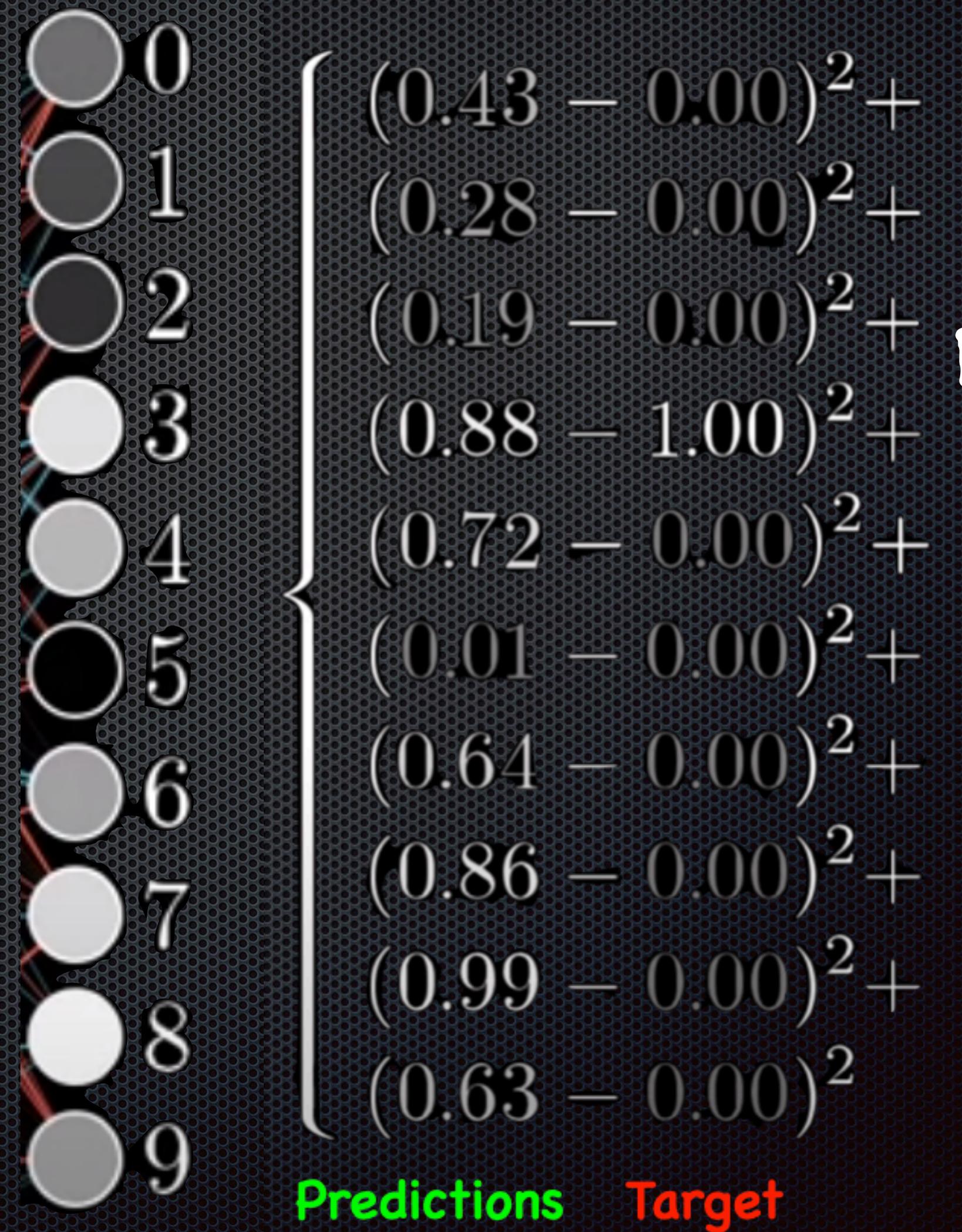
Prediction probability

$$P \in [0, 1]$$

# Example - MNIST data

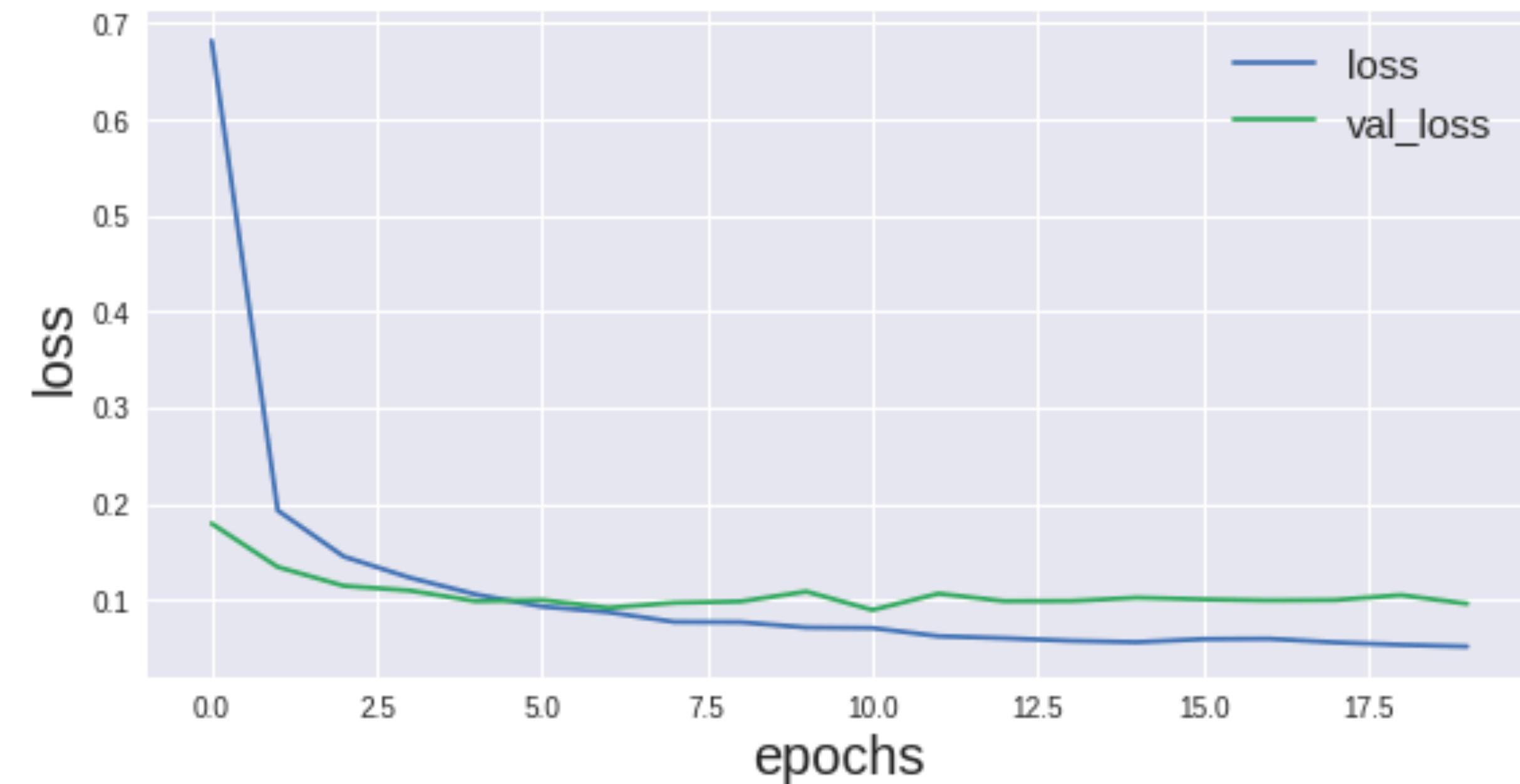
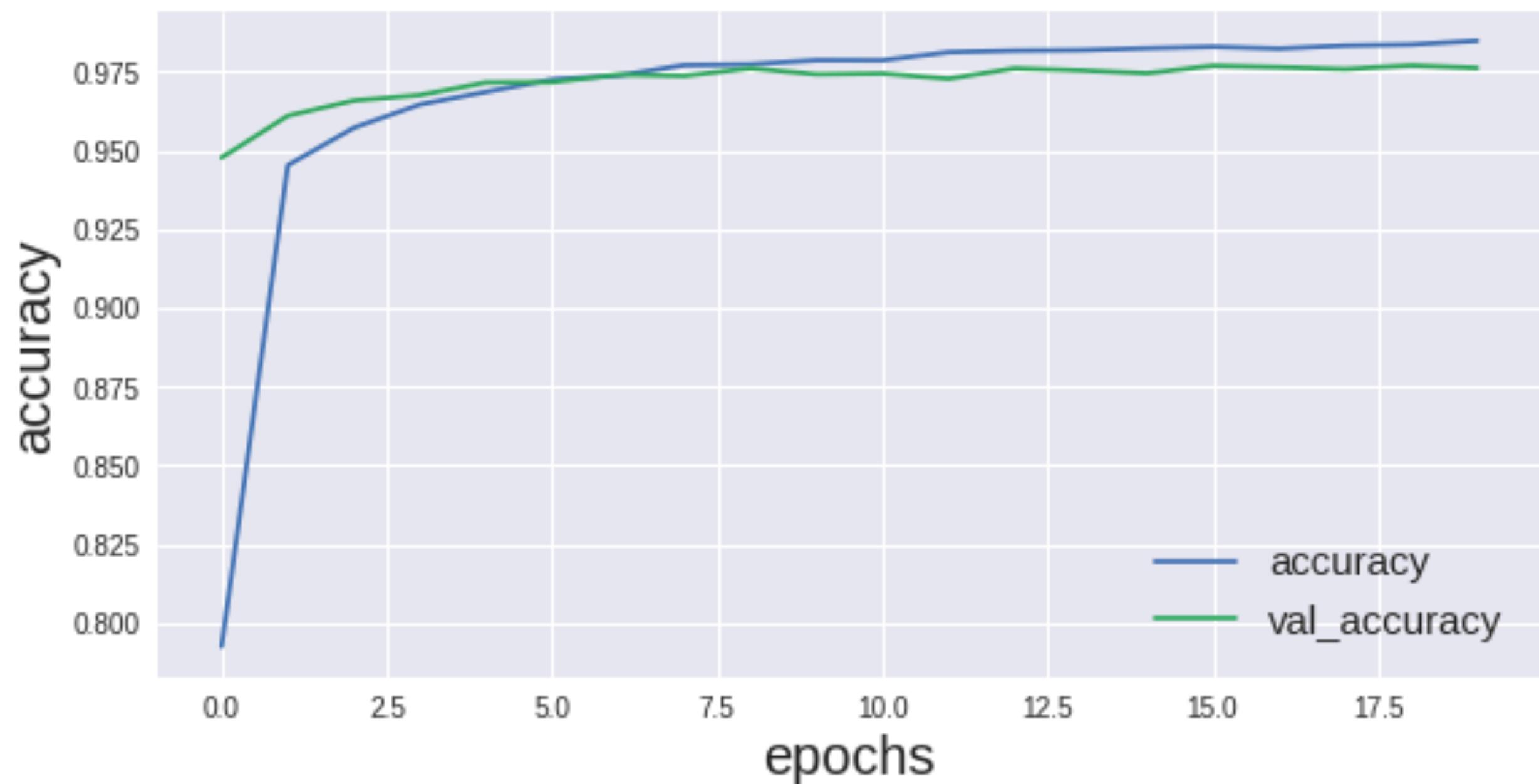
On-hot encoded  
Labels

- Compute the loss function
- Back propagation to update the weights
- Repeat the iteration until to reach the desires accuracy



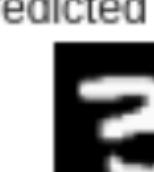
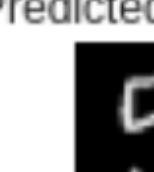
# Example - MNIST data

Test the model on the hold out validation sample



# Example - MNIST data

Do some predictions

Predicted value=7	Predicted value=2	Predicted value=1	Predicted value=0	Predicted value=4	Predicted value=1	Predicted value=4	Predicted value=9	Predicted value=6	Predicted value=9
									
Predicted value=0	Predicted value=6	Predicted value=9	Predicted value=0	Predicted value=1	Predicted value=5	Predicted value=9	Predicted value=7	Predicted value=3	Predicted value=4
									
Predicted value=9	Predicted value=6	Predicted value=6	Predicted value=5	Predicted value=4	Predicted value=0	Predicted value=7	Predicted value=4	Predicted value=0	Predicted value=1
									
Predicted value=3	Predicted value=1	Predicted value=3	Predicted value=4	Predicted value=7	Predicted value=2	Predicted value=7	Predicted value=1	Predicted value=2	Predicted value=1
									
Predicted value=1	Predicted value=7	Predicted value=4	Predicted value=2	Predicted value=3	Predicted value=5	Predicted value=1	Predicted value=2	Predicted value=4	Predicted value=4
									
Predicted value=6	Predicted value=3	Predicted value=5	Predicted value=5	Predicted value=6	Predicted value=0	Predicted value=4	Predicted value=1	Predicted value=9	Predicted value=5
									

To be continued...