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

7 neurons | 6 neurons | 7 neurons | 8 neurons | 6 neurons | 2 neurons

*This is the output from one **neuron**.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

OUTPUT

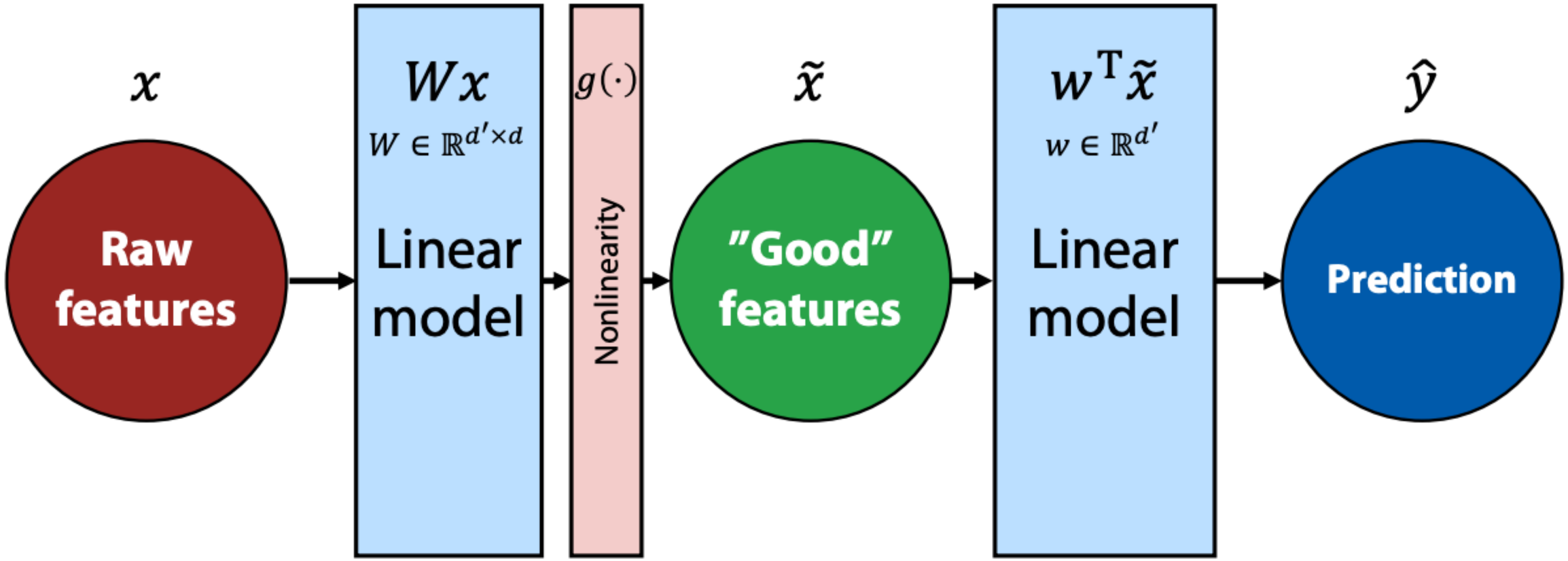
Test loss 0.495
Training loss 0.495

Colors shows data, neuron and weight values.

-1 0 1

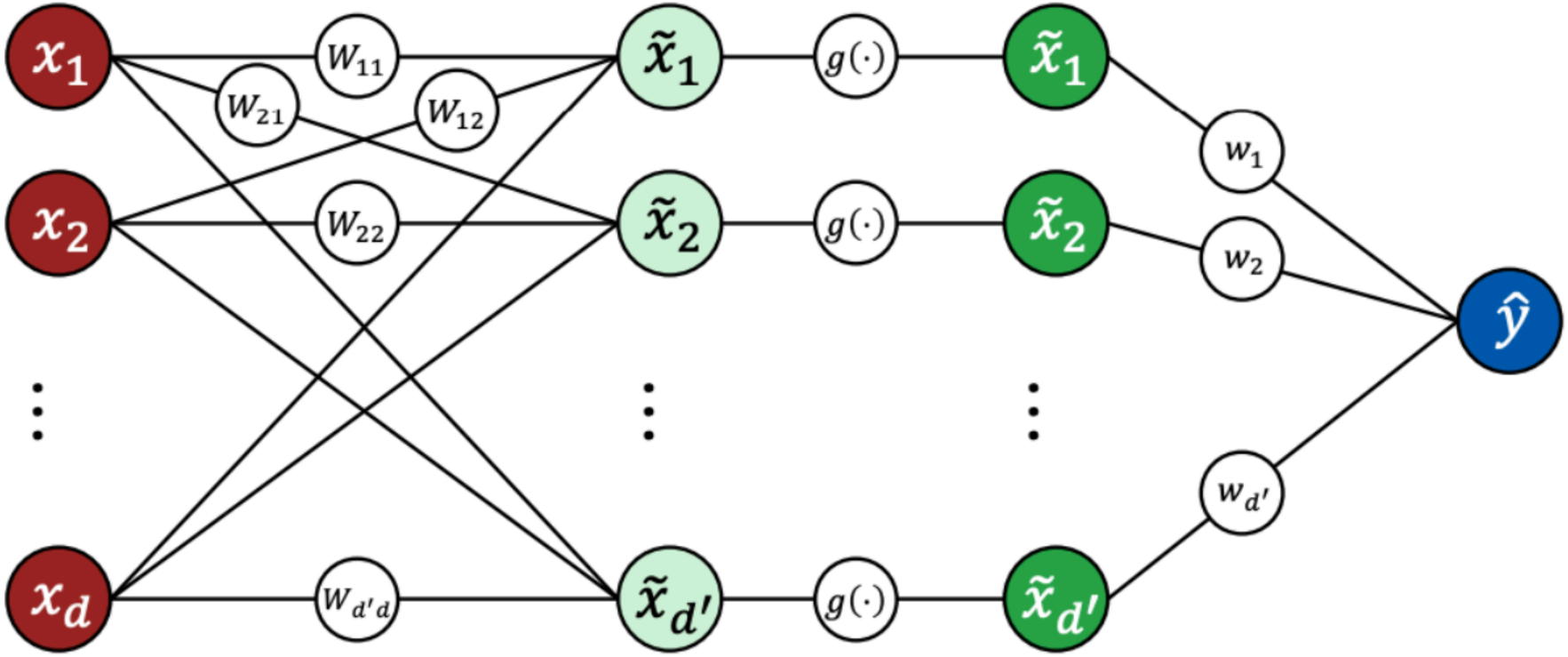
Show test data | Discretize output

Fully connected deep learning



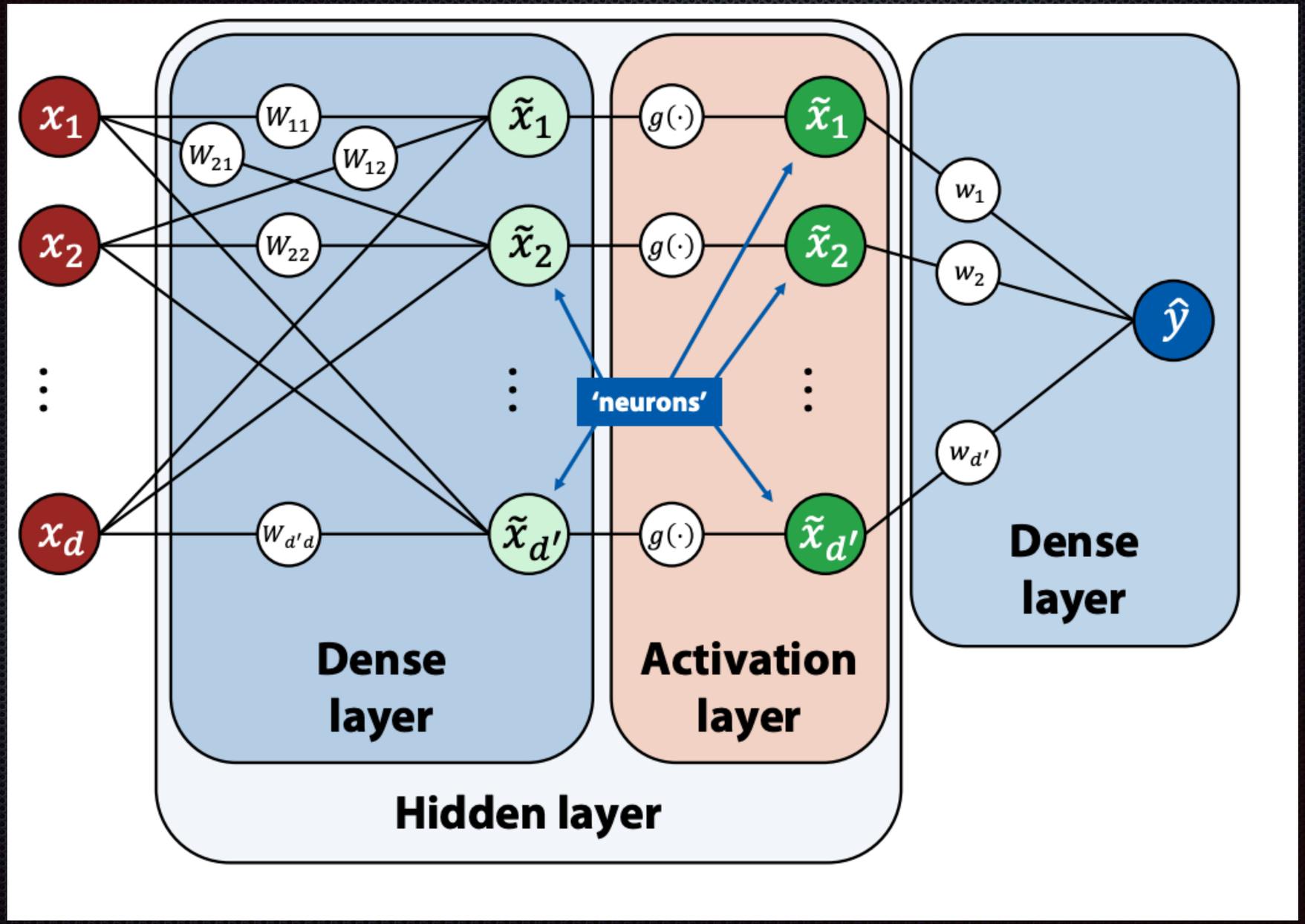
$$\hat{y} = w^T\tilde{x} = w^T g(Wx)$$

Fully connected deep learning

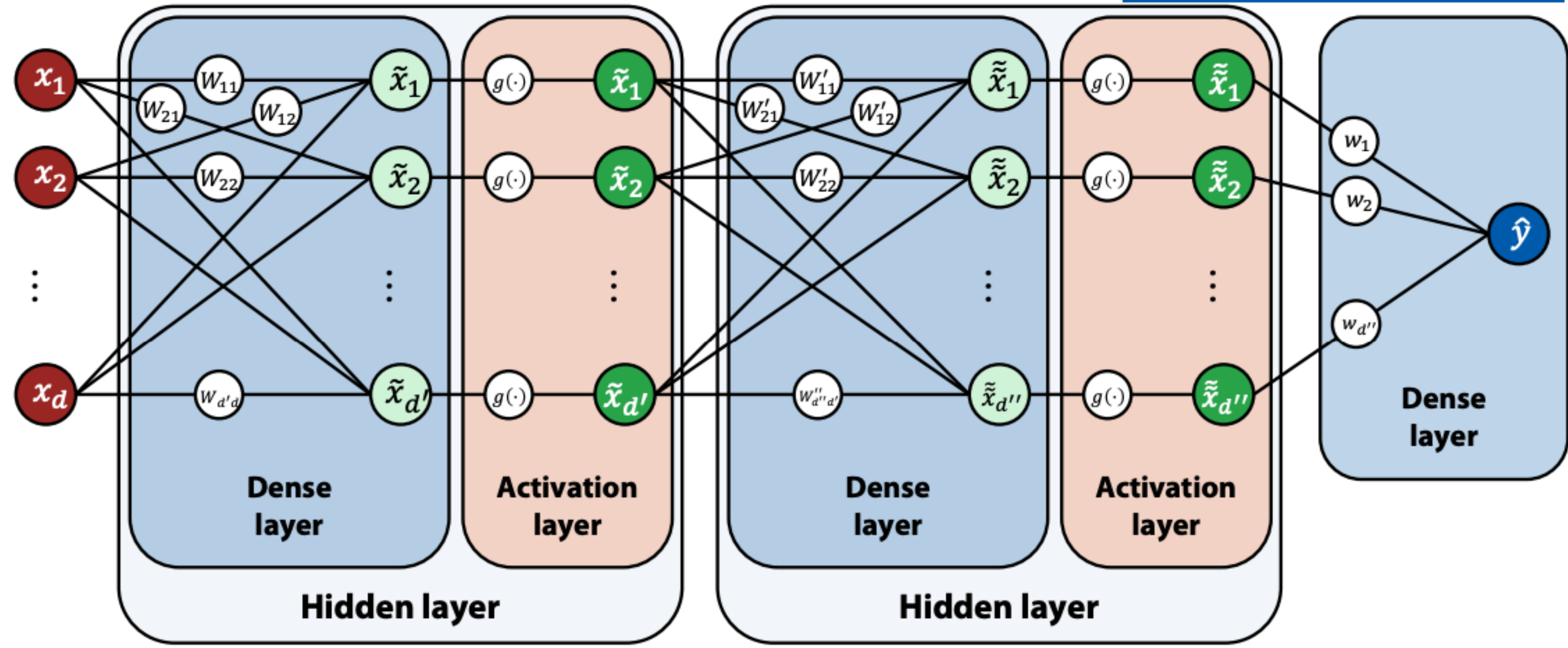


$$\hat{y} = w^{\mathrm{T}}\tilde{x} = w^{\mathrm{T}}g(Wx) = \sum_j \left[w_j g\left(\sum_i W_{ji} x_i \right) \right]$$

Single layer neural network



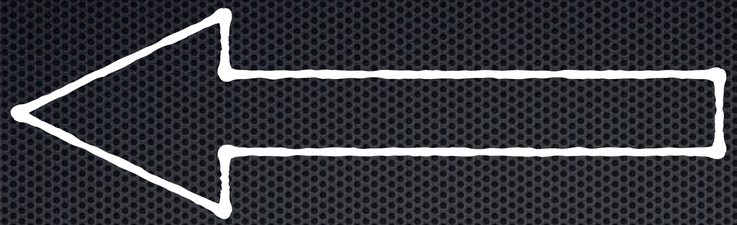
Multilayer neural network



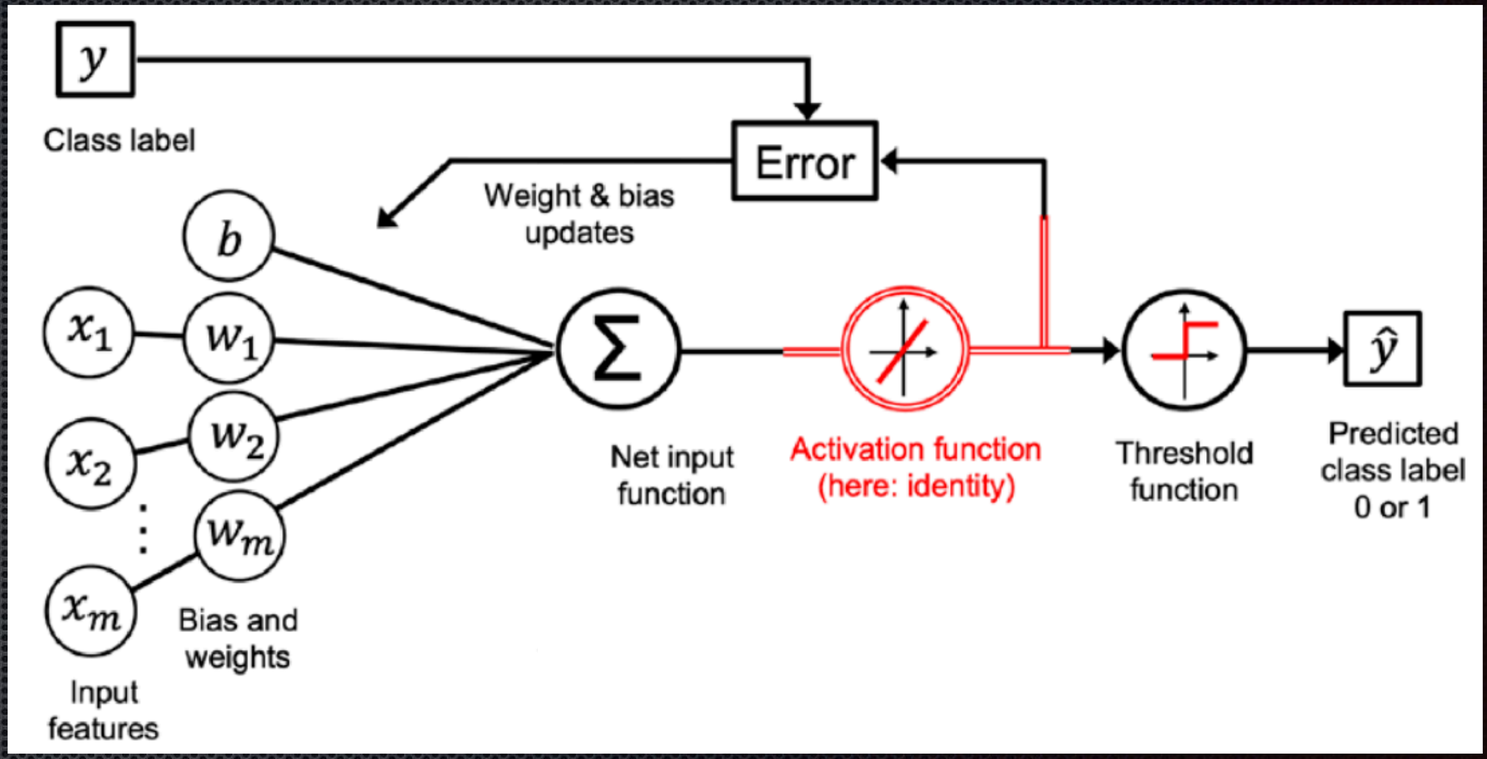
Training deep learning models

The training loop consists of forward and backward propagation

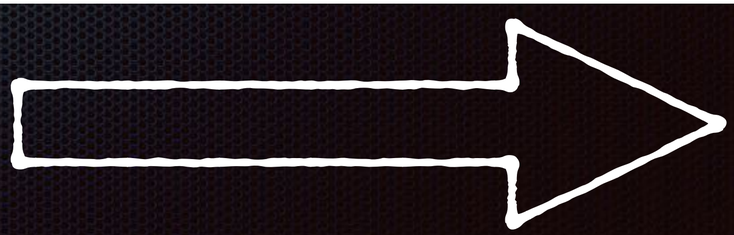
Each iteration is called “epoch”



Backward propagation

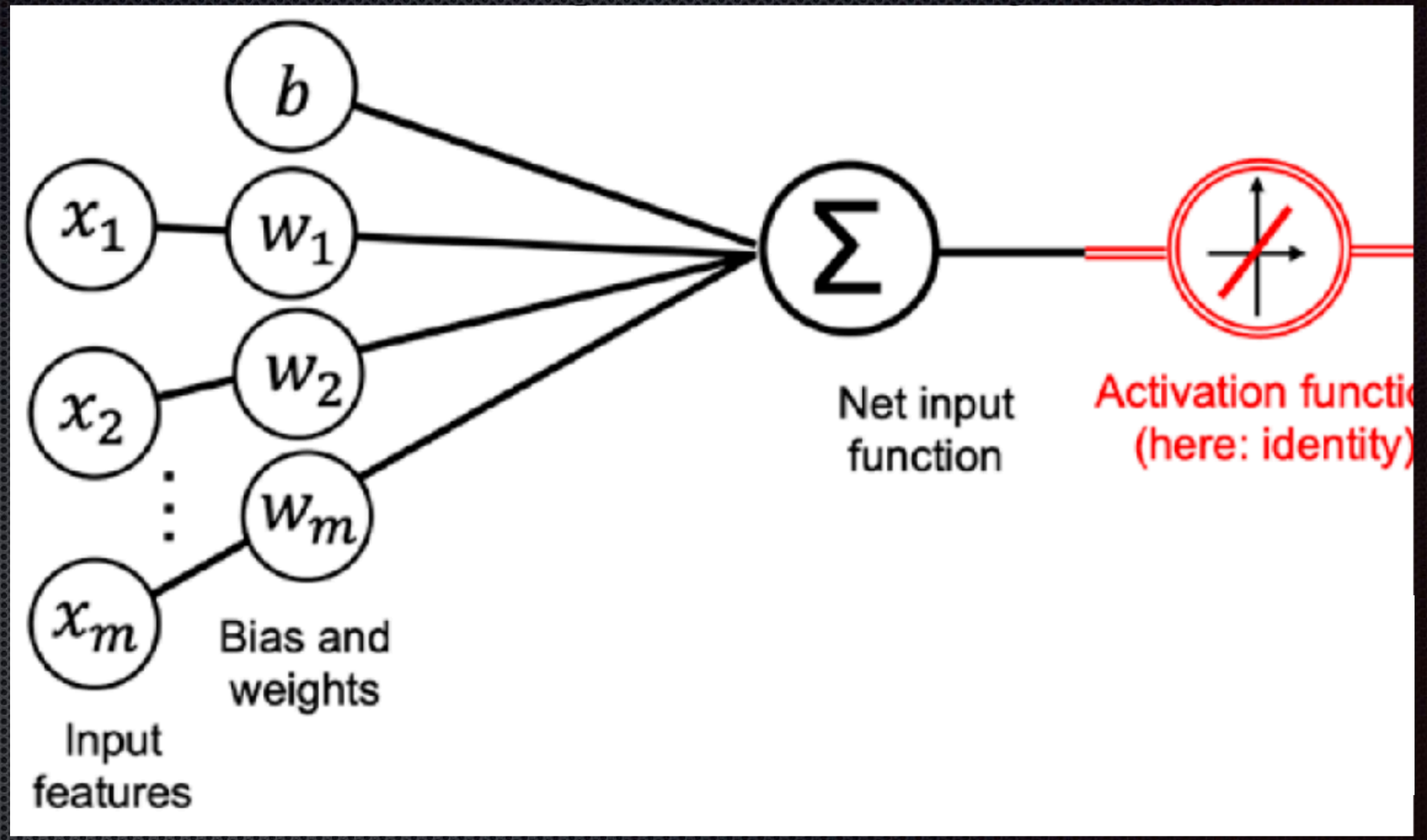


Forward propagation



Forward propagation–single layer

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



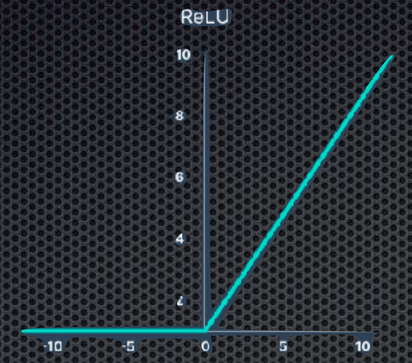
$$\text{Neuron Output} = \text{Activation function}\left(\sum_m \omega_m \, x_m + b\right)$$

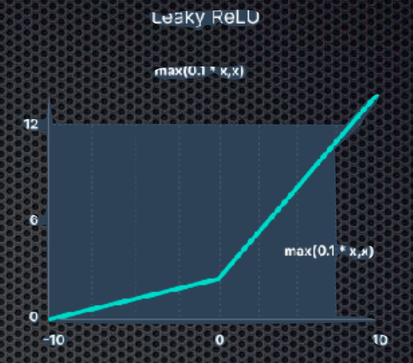
Activation functions

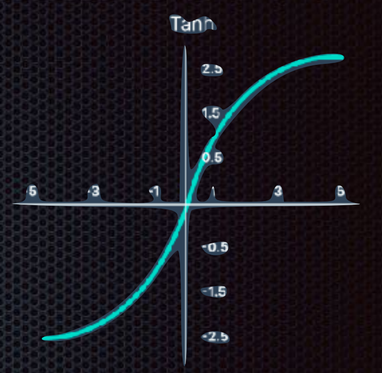
The choice of the activation function at each layer depends no the problem at hand

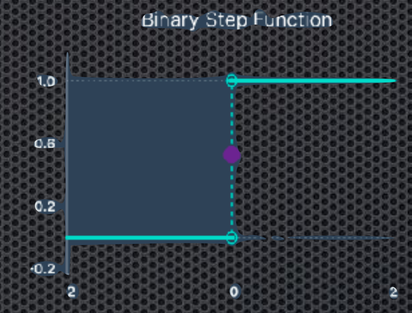
The choice of the activation functions can be optimized using the “Grid search optimization”

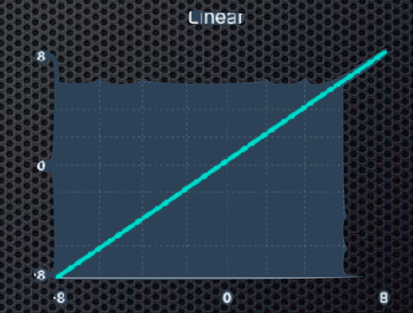
$\max(0, x)$

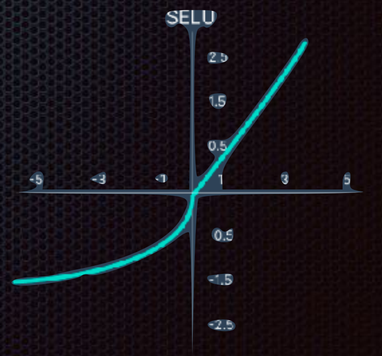




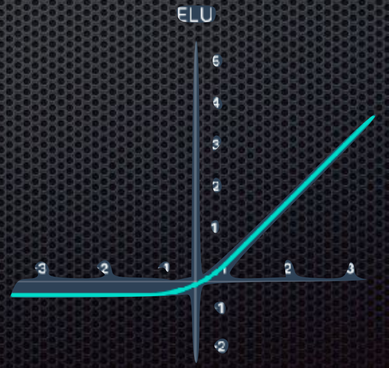




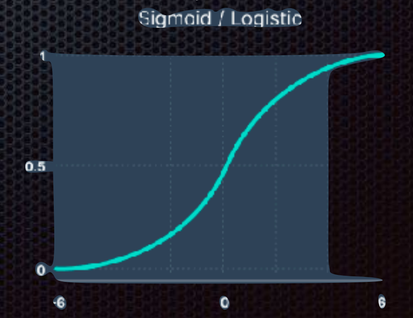


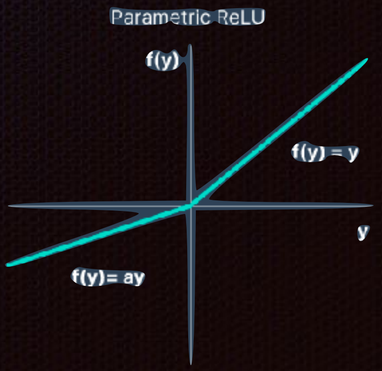


$\max(\alpha, x)$



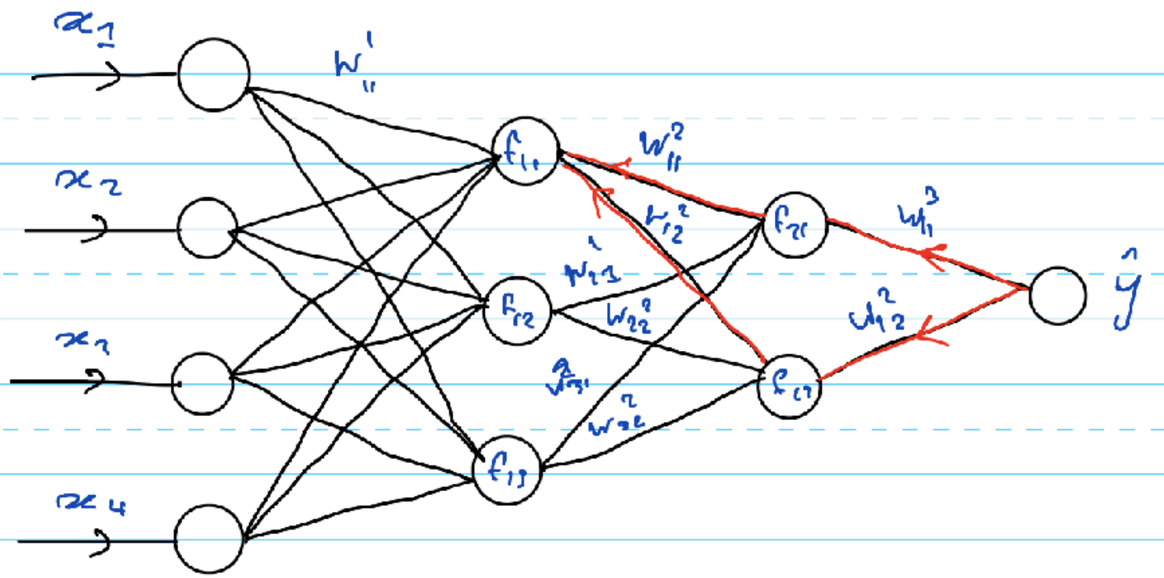
$$\frac{1}{1 + e^{-x}}$$





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

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

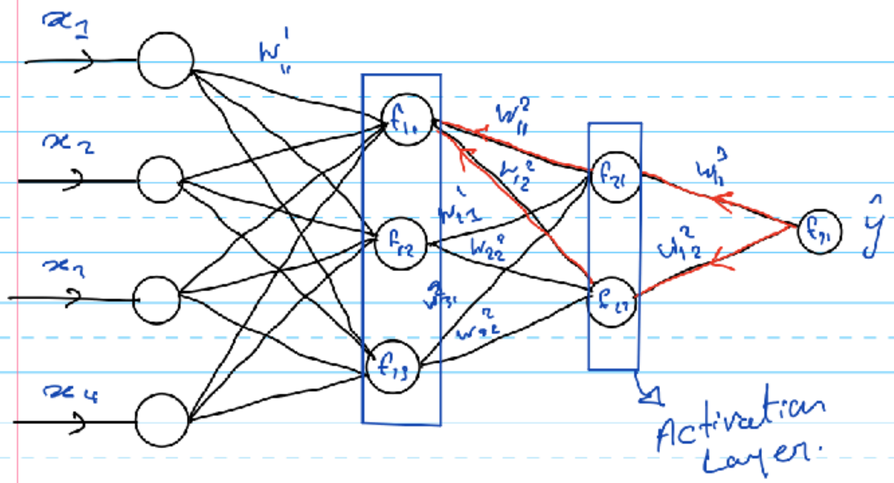
to update the weights in backward mode and minimize the loss function.

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



$L = \frac{1}{2}|y-\hat{y}|^2$, $w^i_{new} = w^i_{old} - \eta \nabla L(w^i_{old})$

Example:

Find the updated weight for w^3_{11}:

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

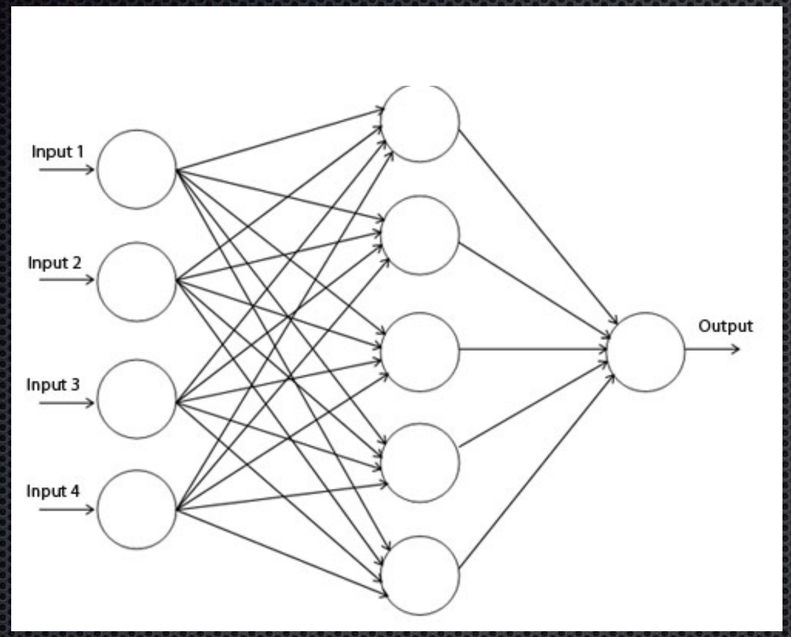
$$w^3_{11,new} = w^3_{11} - \eta * \left[\frac{\partial L}{\partial F_{31}} \frac{\partial f_{31}}{\partial w^3_{11}}\right]$$

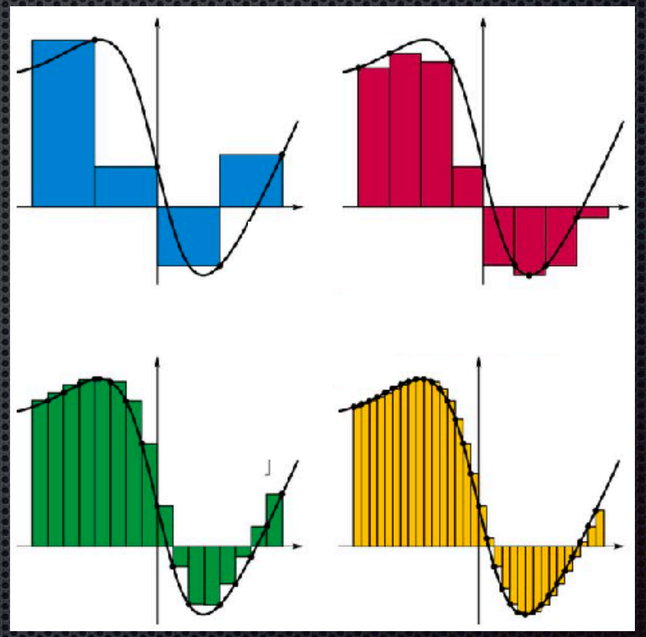
* For w^2_{11}:

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

Universal approximation theorem

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



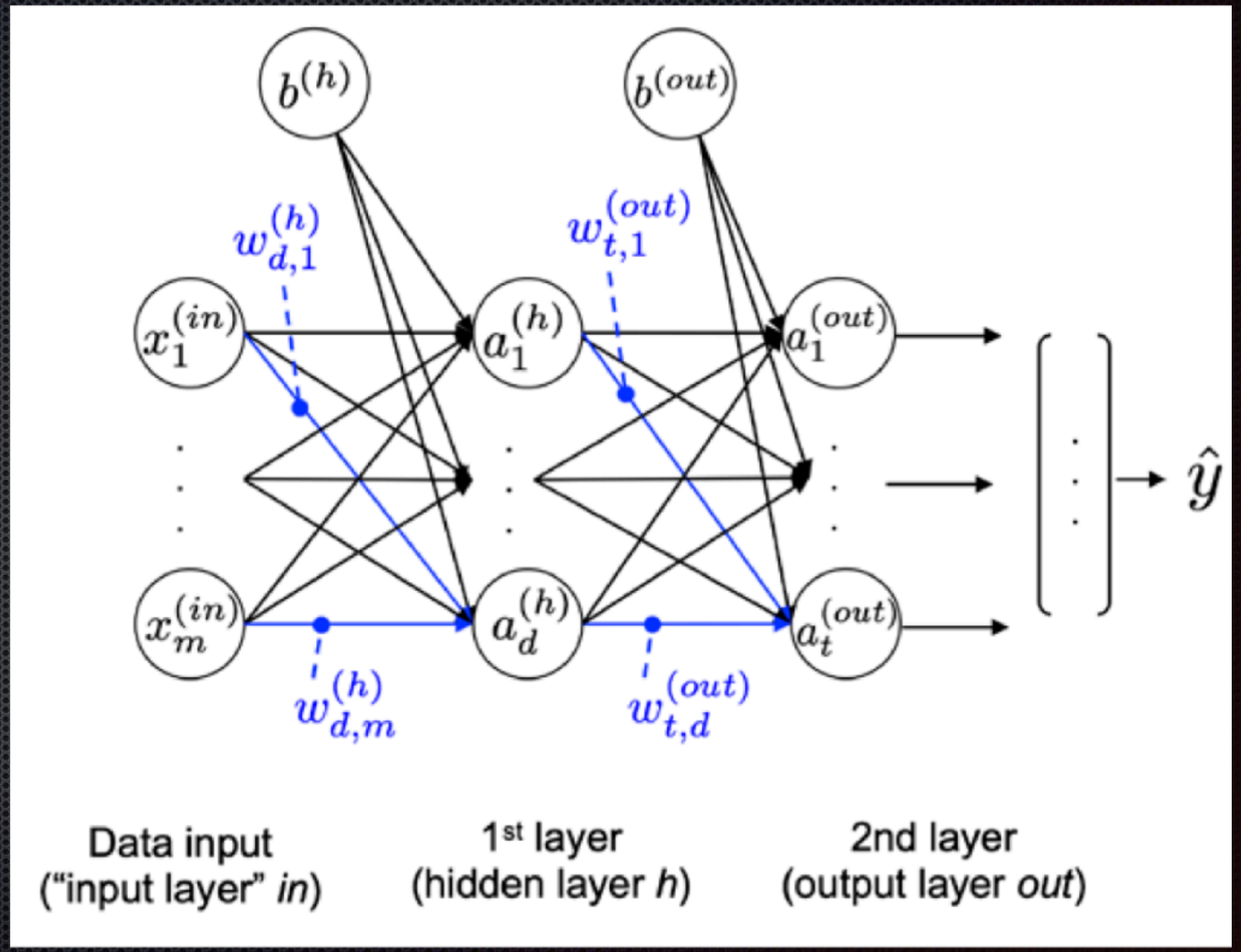


What about nonlinear discontinuous high dimensional function ?

Multi-layer perceptron (MLP)

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

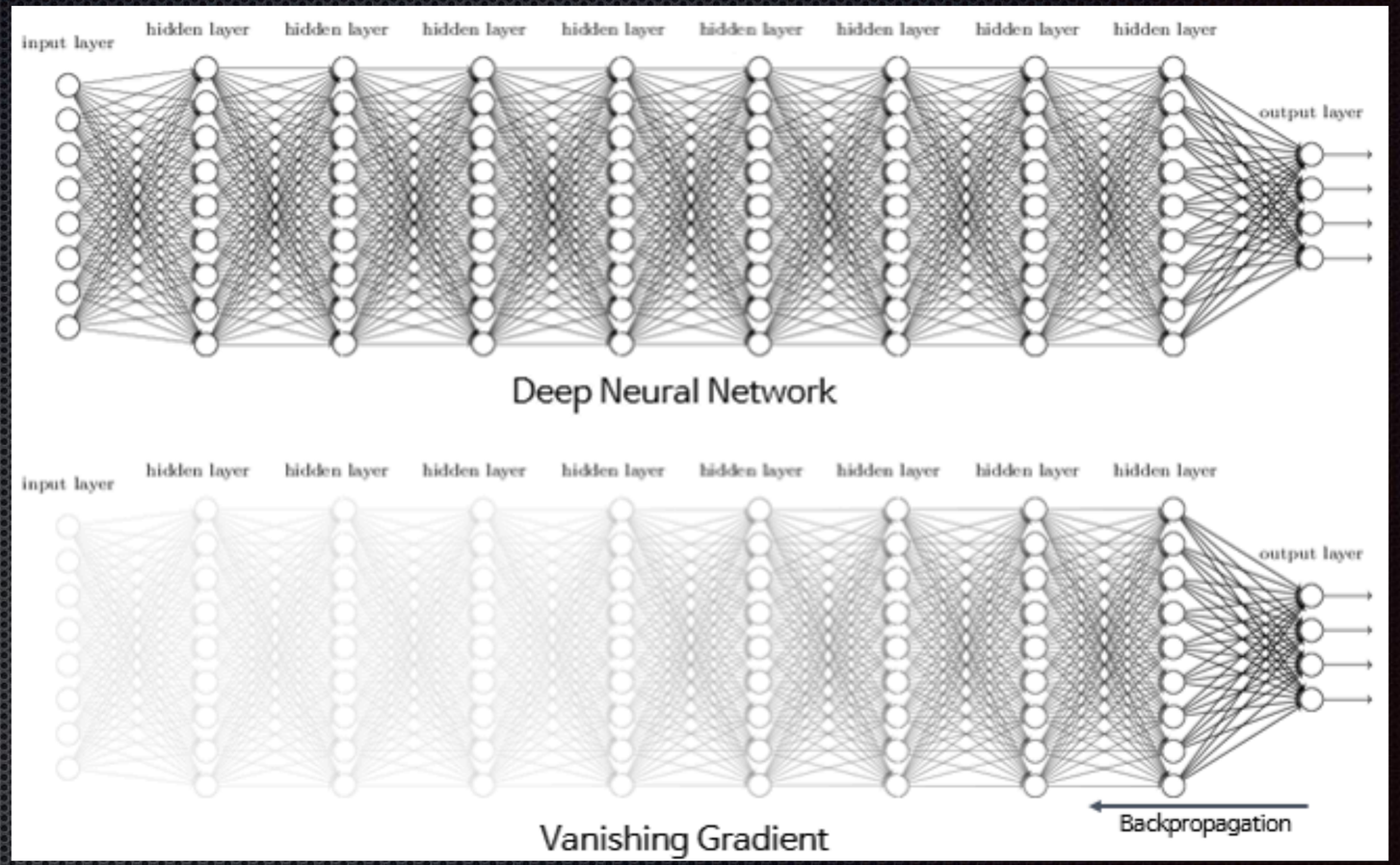
Are we free to use any number of hidden layers?



Vanishing gradient

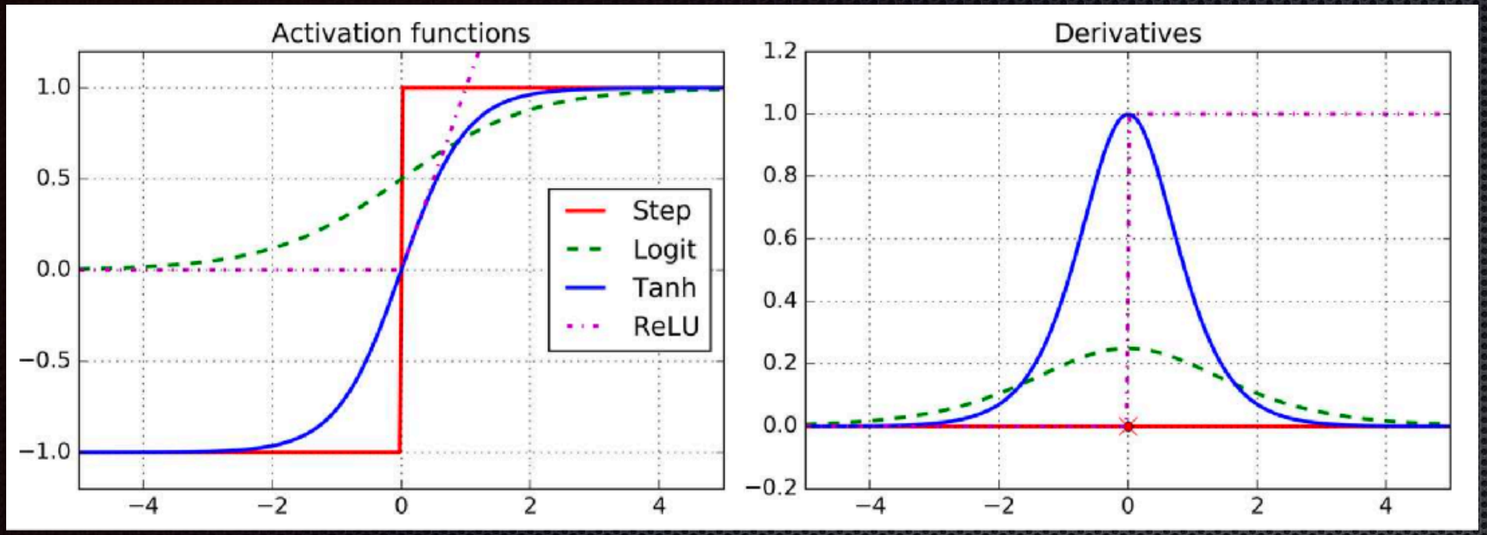
For very deep models, during the backpropagation the gradient of the initial layers vanishes

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



Why vanishing gradients takes place?

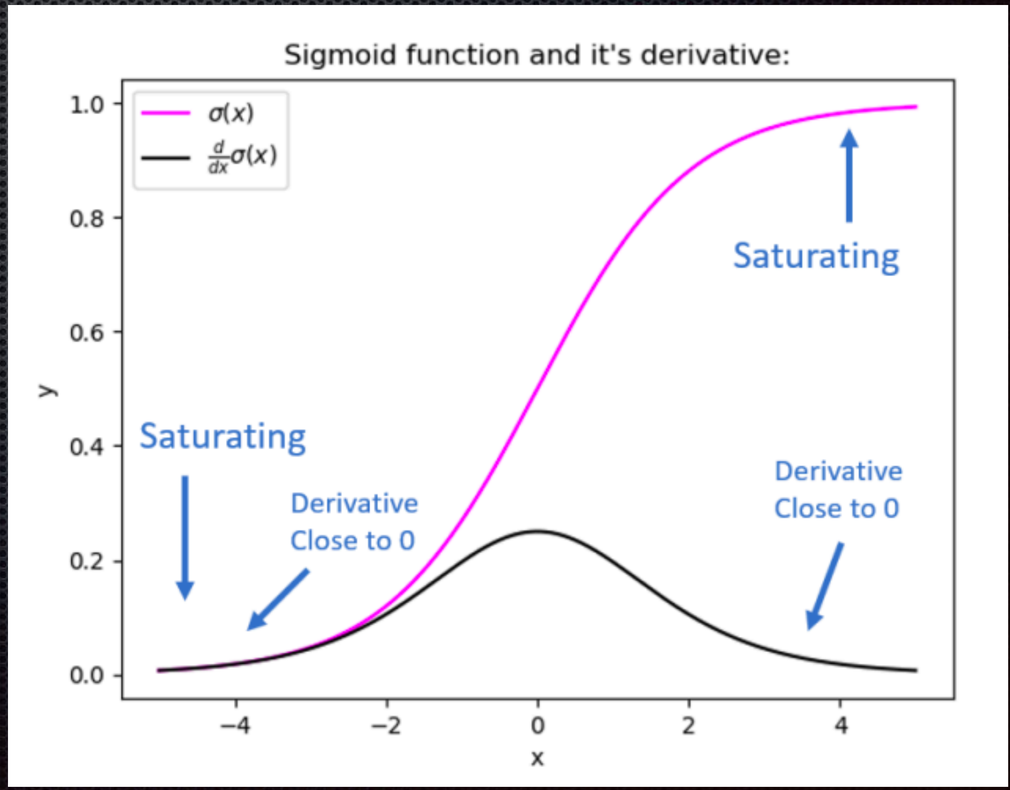
Vanishing gradient



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

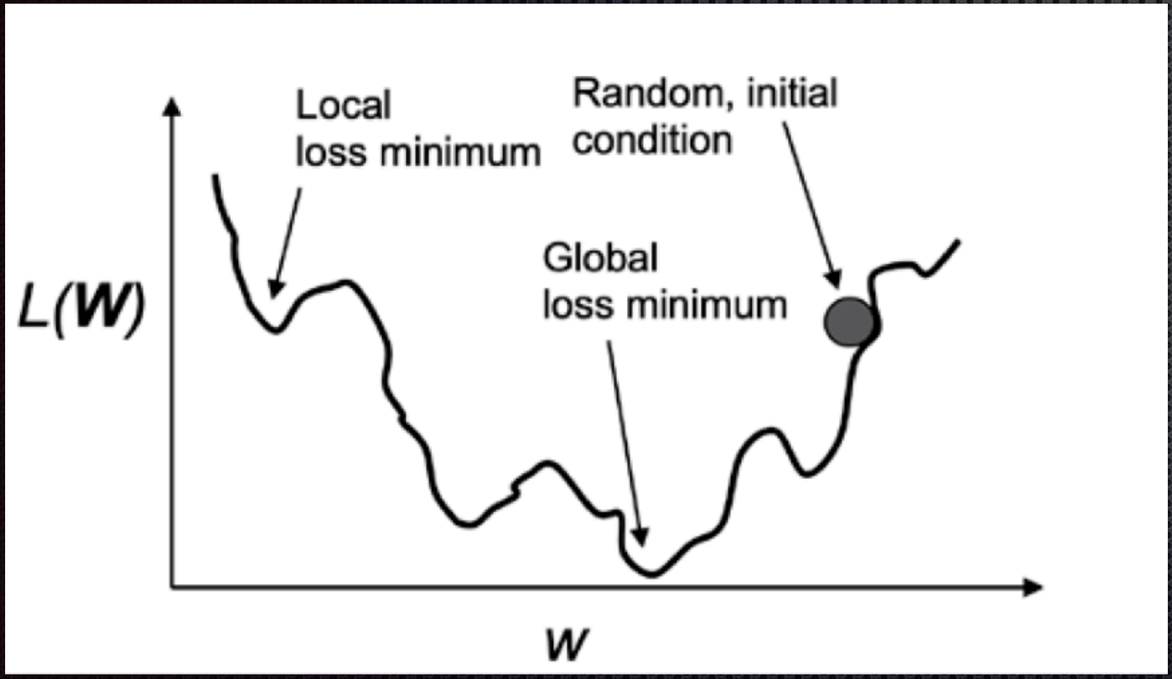
Solutions:

- 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

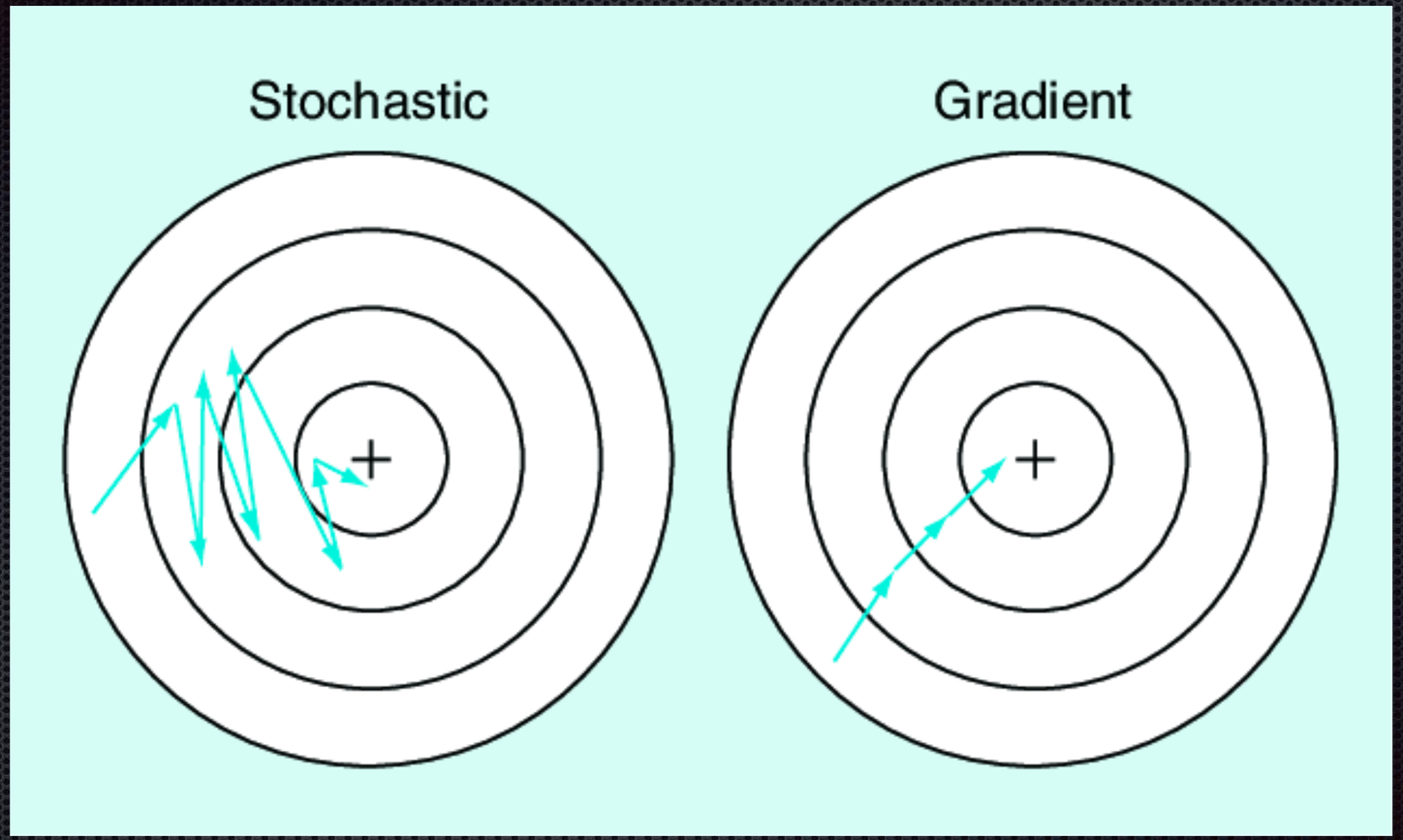


Conventionally, we initialize the weights randomly
If the initialized weight is on the right it can reach the global minimum faster than if it lies in the left.

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

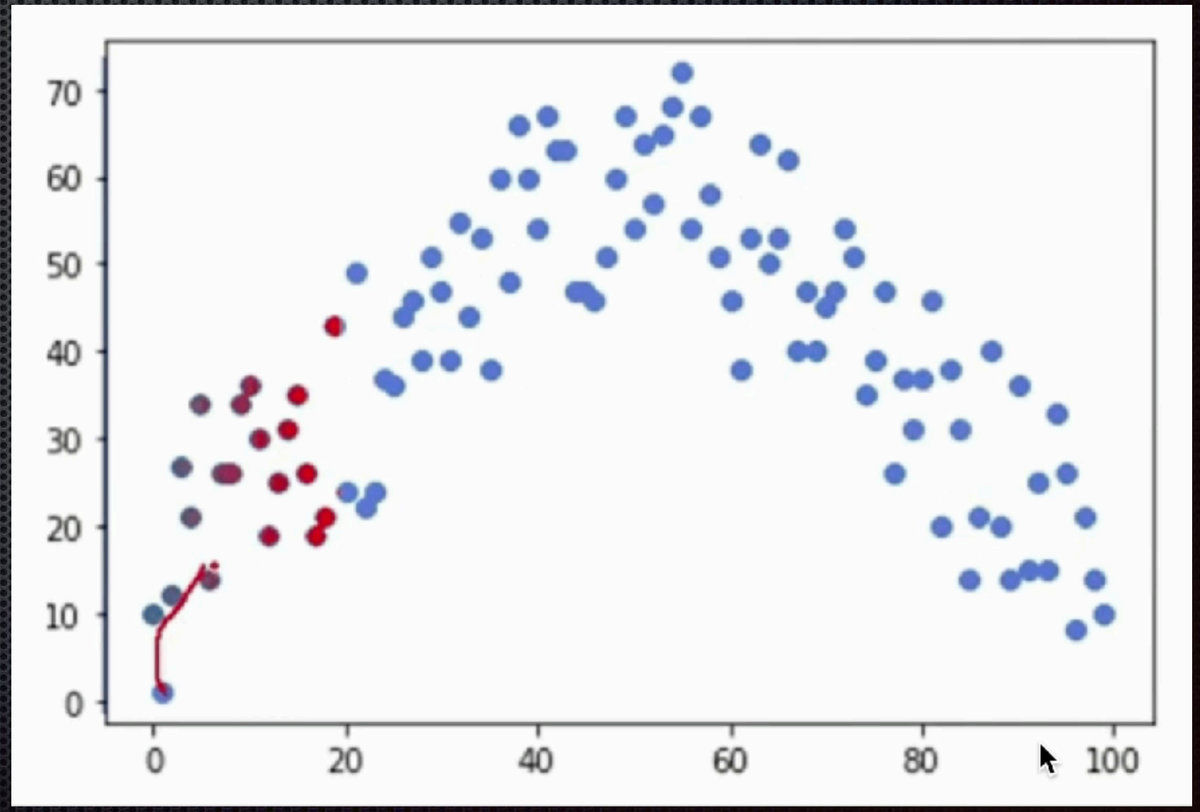
Stochastic gradient descent with momentum

Momentum is the exponential weighted average to reduce the fluctuation

Exponential weighted moving average

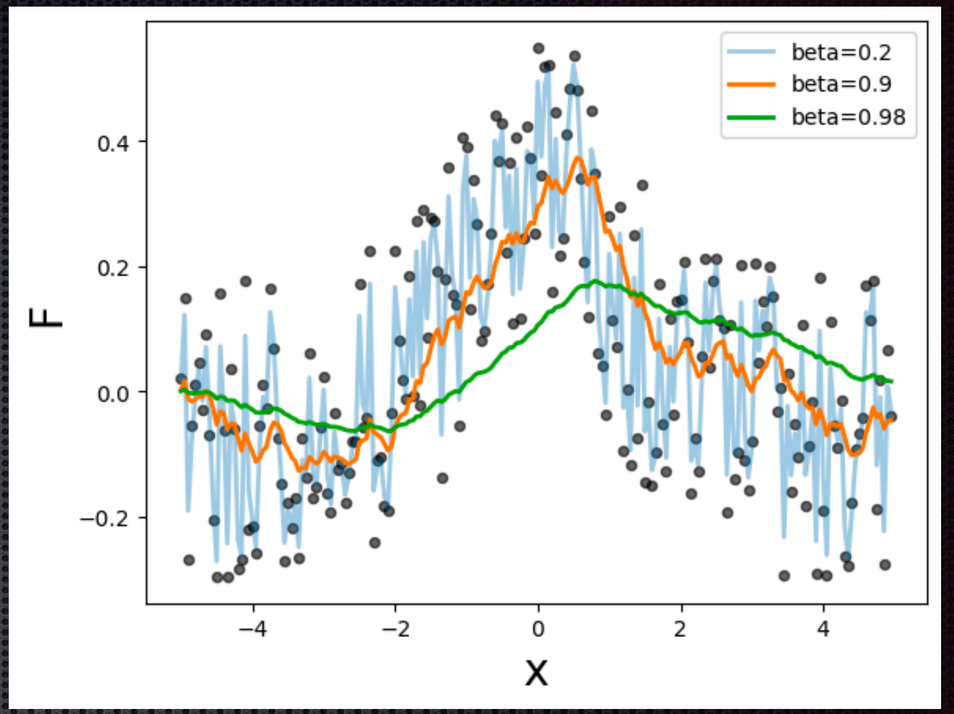
$$V_t = \beta V_{t-1} + (1-\beta) F_t$$
$$V_{t-1} = \beta V_{t-2} + (1-\beta) F_{t-1}$$
$$V_{t-2} = \beta V_{t-3} + (1-\beta) F_{t-2}$$

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



Stochastic gradient descent with momentum

The larger the beta the smoother curve

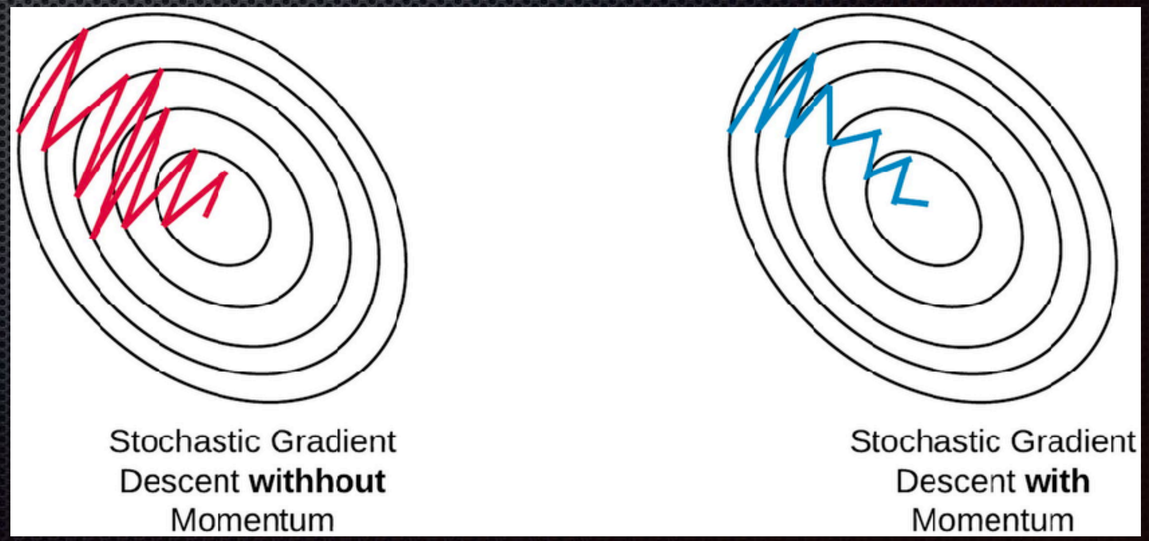


For fixed beta value and less variation, F collapses around zero

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



Adaptive gradient descent (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

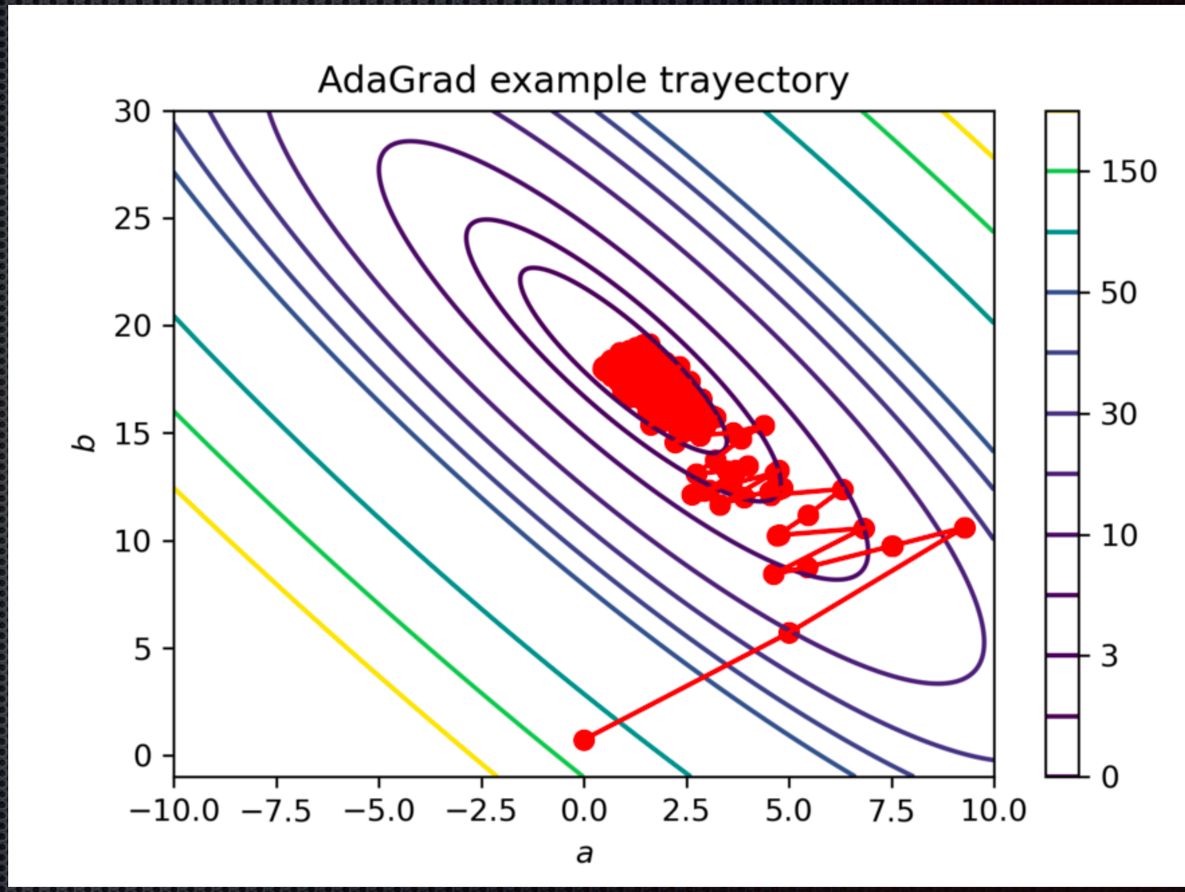
$$\omega^{\text{new}} = \omega^{\text{old}} - \eta' \nabla L(\omega)$$

$$\eta' = \frac{\eta}{\sqrt{\alpha_t + \epsilon}} \qquad \alpha_t = \sum^{t} (\nabla L(\omega))^2$$

Initial chosen learning rate

Regularization parameter
To avoid the divergence
In case of vanishing gradient

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



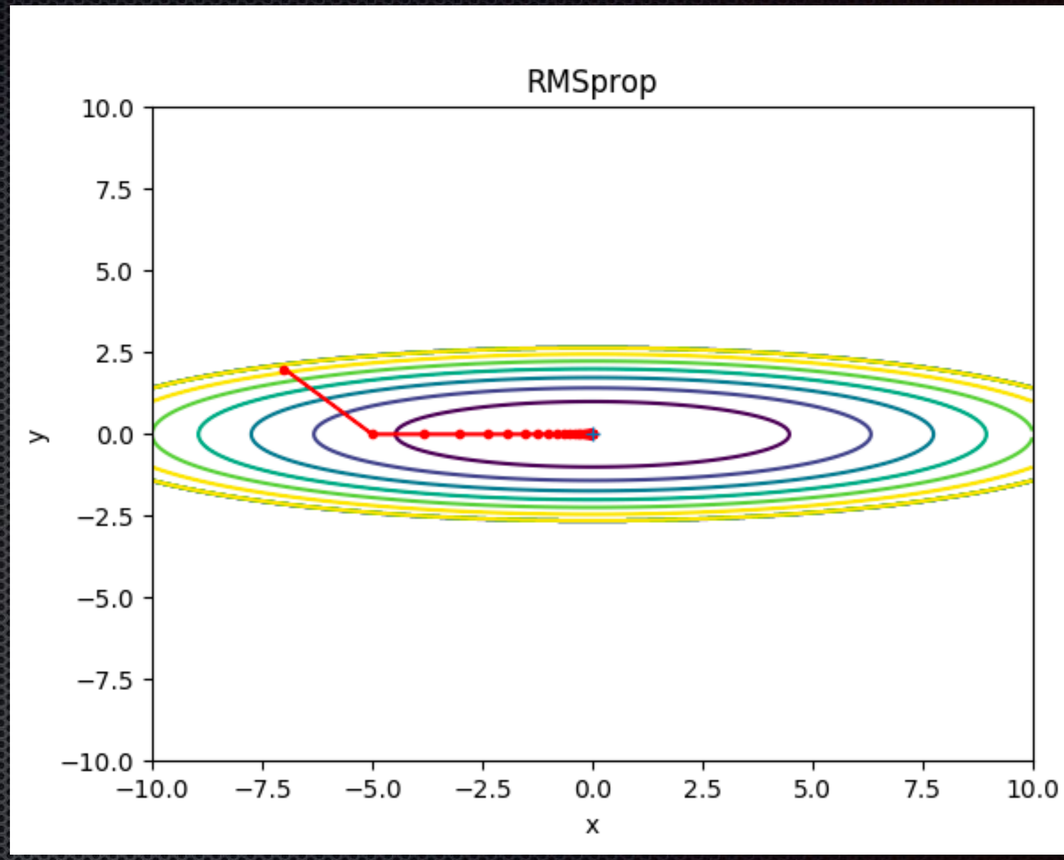
The learning rate will automatically decrease because the summation of the previous gradient square will always keep on increasing after every iteration.

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.

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

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



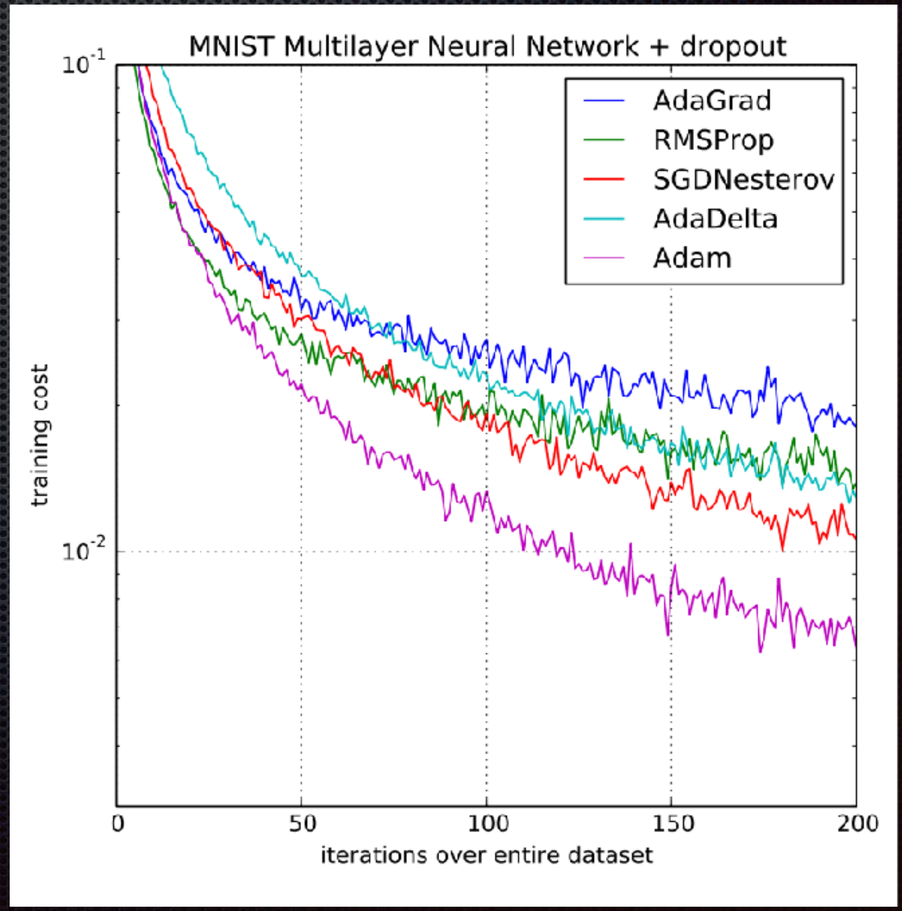
More precisely, the sum of gradient squared is actually the decayed sum of gradient squared. The decay rate is saying only recent gradient² matters, and the ones from long ago are basically forgotten.

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.

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

arXiv:1412.6980v



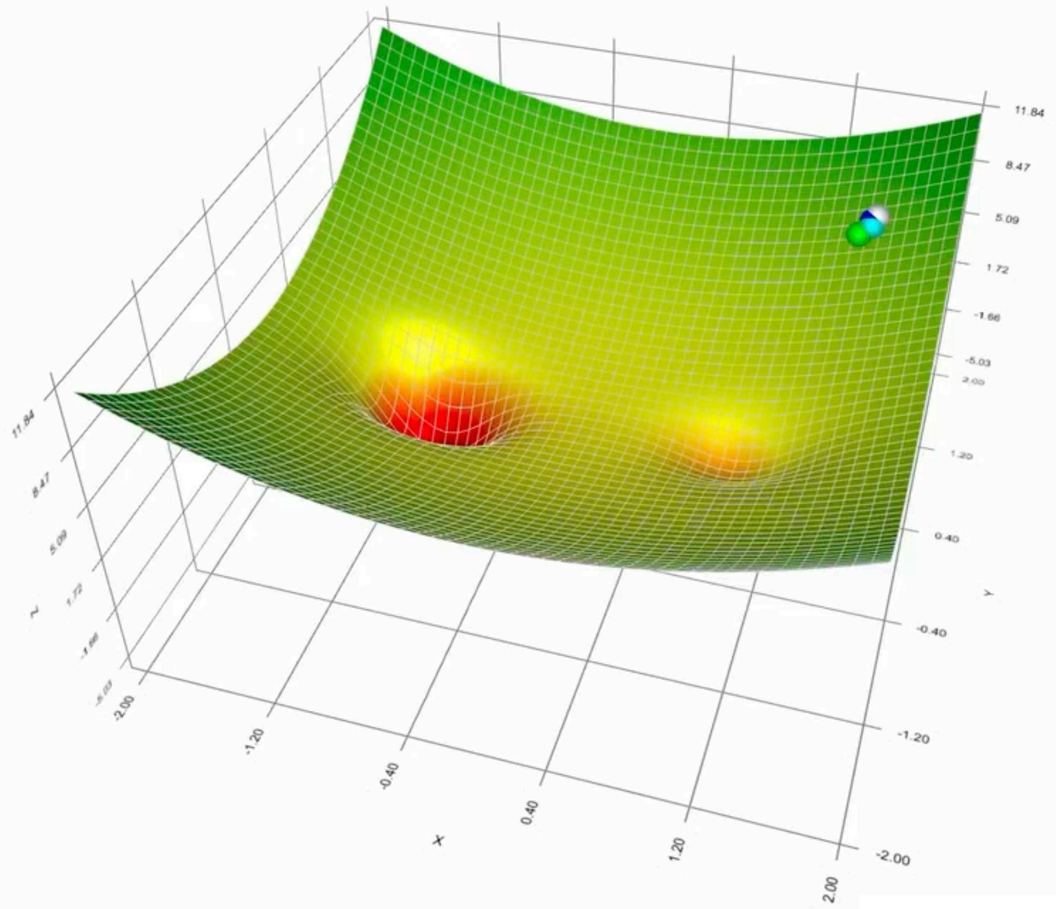
Deep models convergence

Code:
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



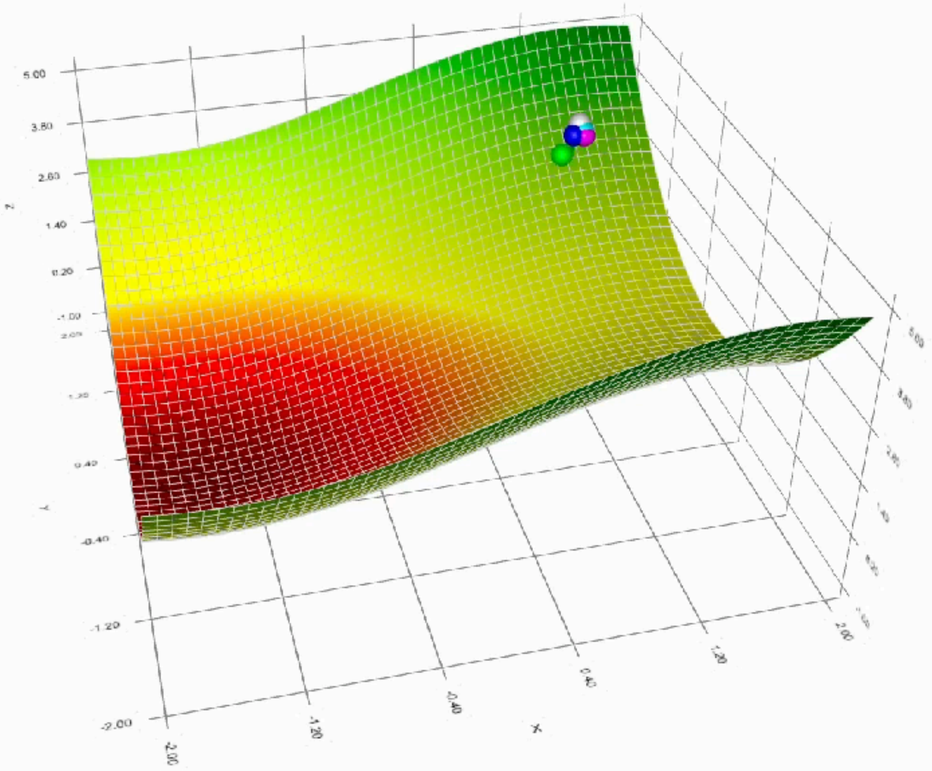
Deep models convergence

Code:
https://github.com/lilipads/gradient_descent_viz/tree/master

Function with saddle point

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

Initial Learning rate = 0.01



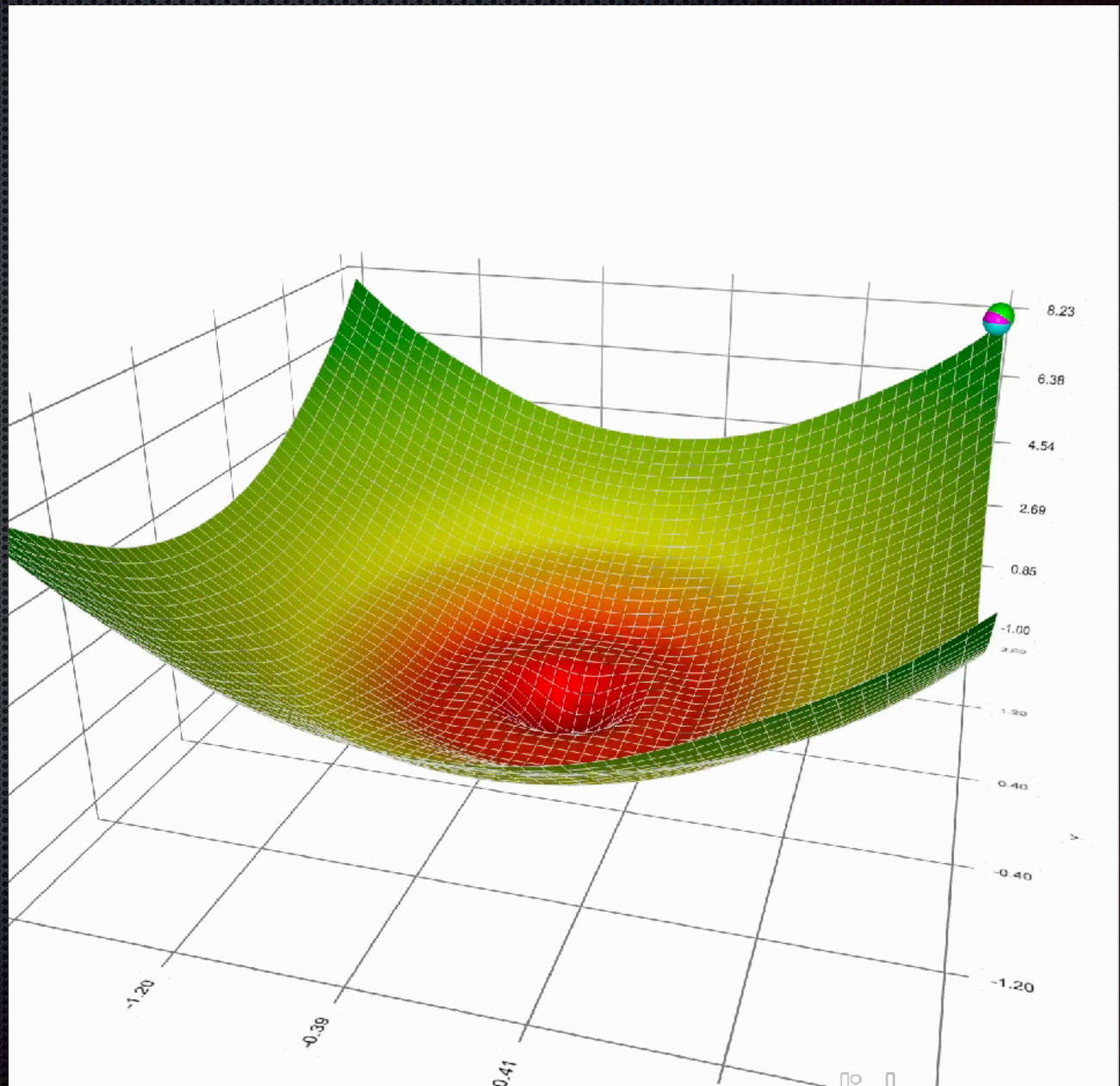
Deep models convergence

Code:
https://github.com/lilipads/gradient_descent_viz/tree/master

Function with plateau

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

Initial Learning rate = 0.01



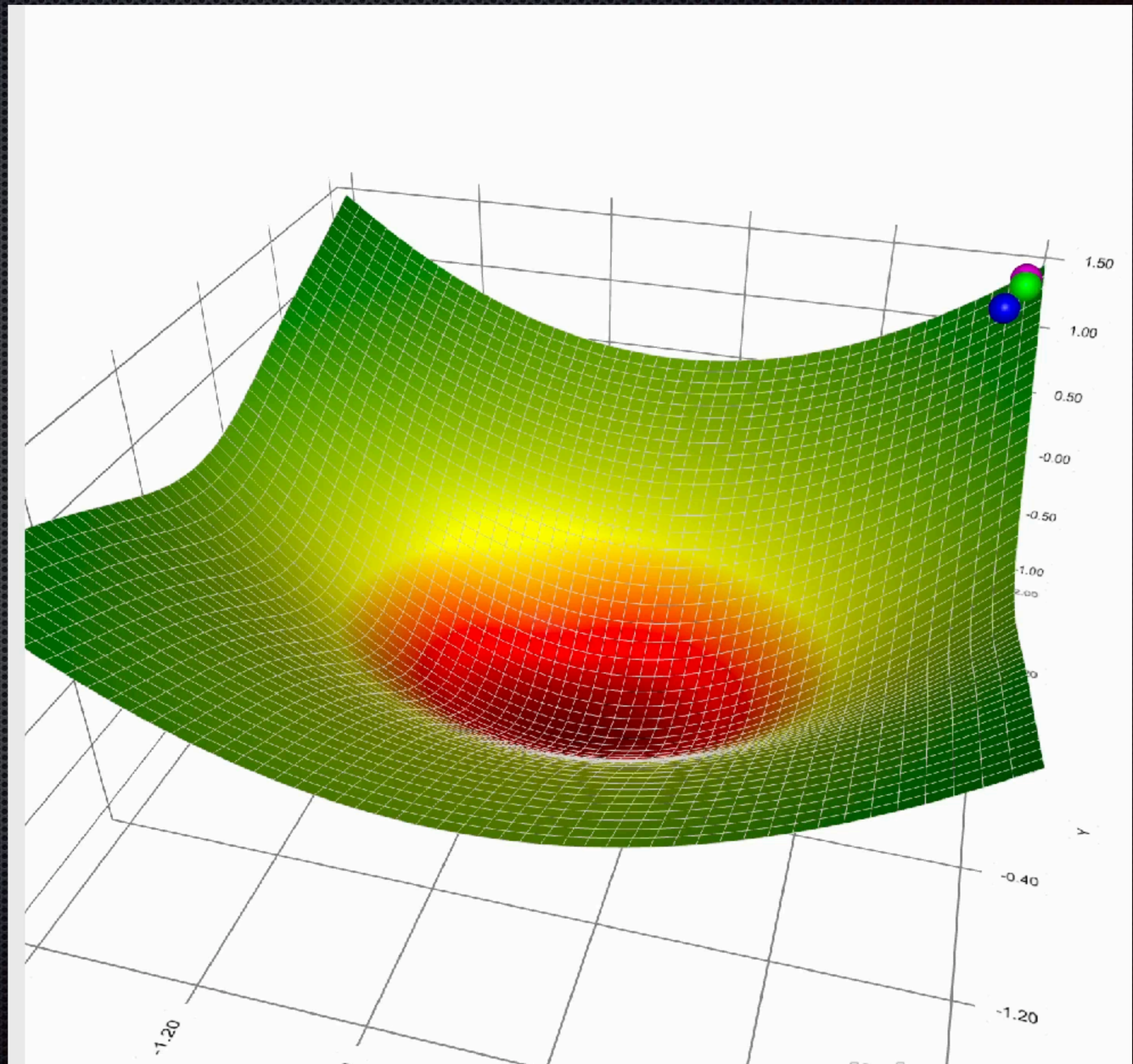
Deep models convergence

Code:
https://github.com/lilipads/gradient_descent_viz/tree/master

Function with Ecliptic Bowl

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

Initial Learning rate = 0.01



No free lunch theorem

All the optimization methods and ML models are similar under constrains

There is no guarantee that one optimization method is the best among all other optimization methods. Also, there is no ML model can solve all the problems

No Free Lunch Theorems for Search

SFI-TR-95-02-010

David H. Wolpert (dhw@santafe.edu)
William G. Macready (wgm@santafe.edu)
The Santa Fe Institute
1399 Hyde Park Road
Santa Fe, NM, 87501

February 23, 1996

Abstract

We show that all algorithms that search for an extremum of a cost function perform exactly the same, according to any performance measure, when averaged over all possible cost functions. In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A. Starting from this we analyze a number of the other a priori characteristics of the search problem, like its geometry and its information-theoretic aspects. This analysis allows us to derive mathematical benchmarks for assessing a

Exploratory data analysis (EDA)

Prepare the data for deep learning models

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

Data cleaning

- Handel 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

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...	female	38.0	1			71.2833	C85	C
3	Heikkinen, Miss. Laina	female		0	0.0	STON/O2. 3101282	7.9250		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

Easy to be handled in Pandas

Pandas.isnull() ⇒ Find the missing values

Pandas.fillna() ⇒ Fill the missing values by extrapolation between the last values

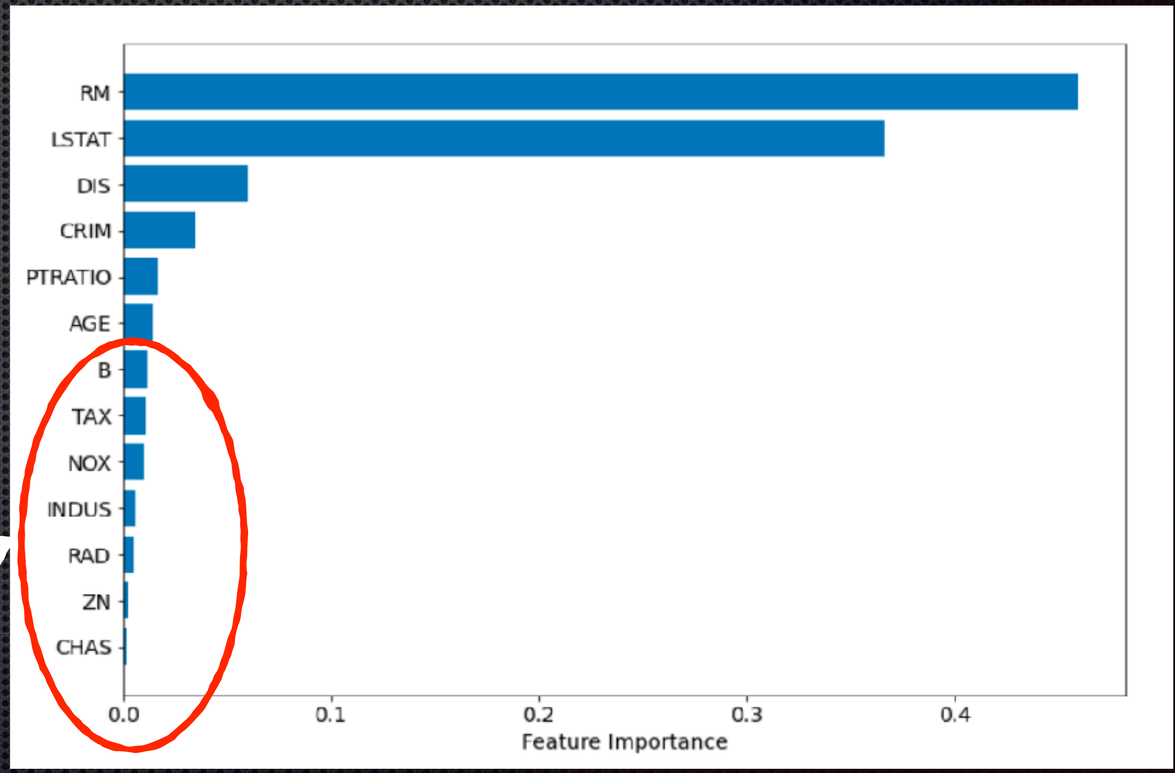
Data cleaning

- Feature importance

Deep models learn from large data and keeping more redundant information into the data consume the computational resources and may hinders 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
Red
Yellow
Green
Yellow

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

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

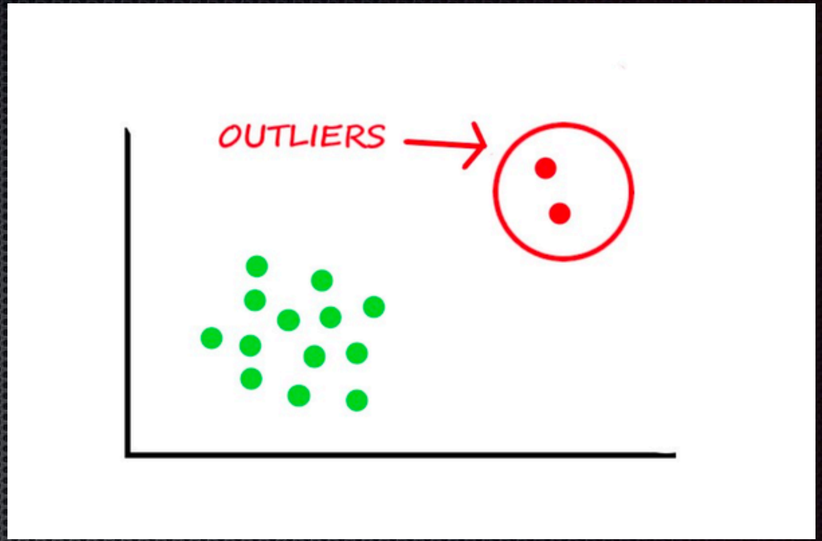
Labels

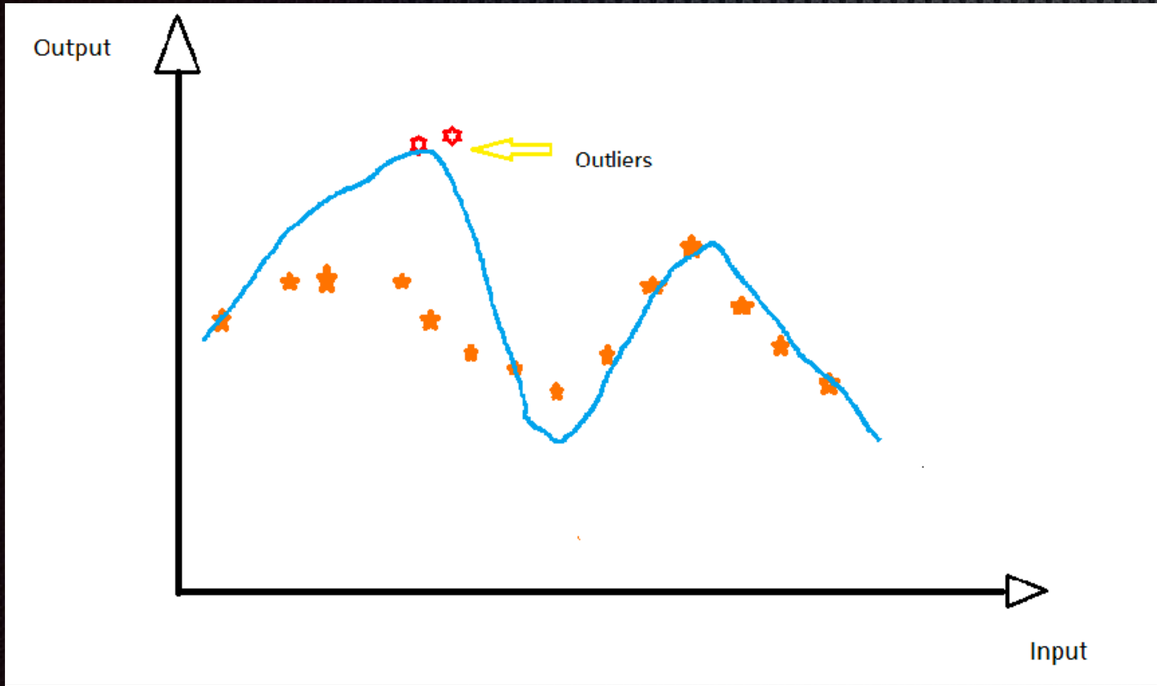
	0	1	2	3	4	5	6	7	8	9	label
0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	5
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	4
3	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	9
5	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2
6	0.0	1.0	0.0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



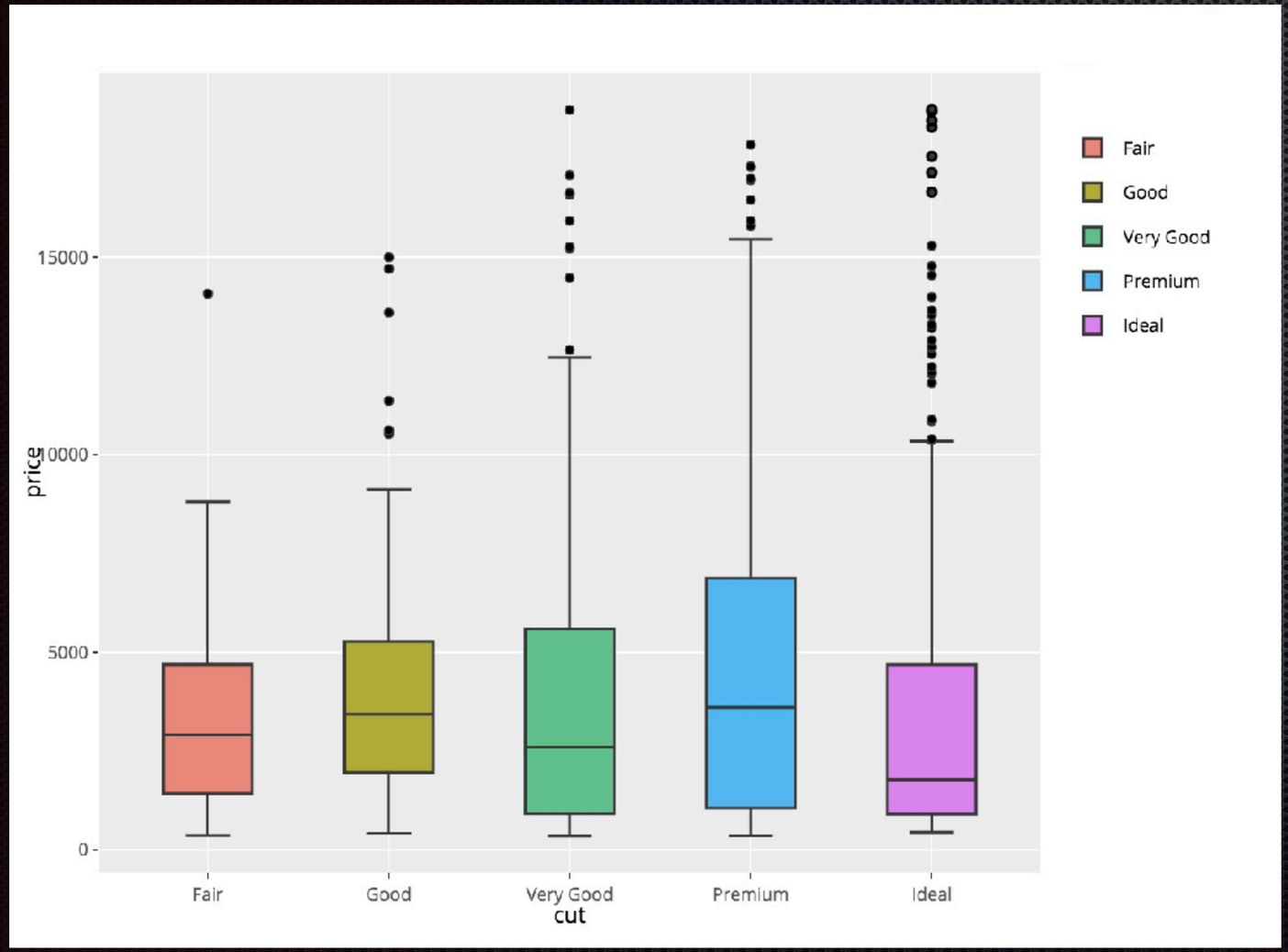


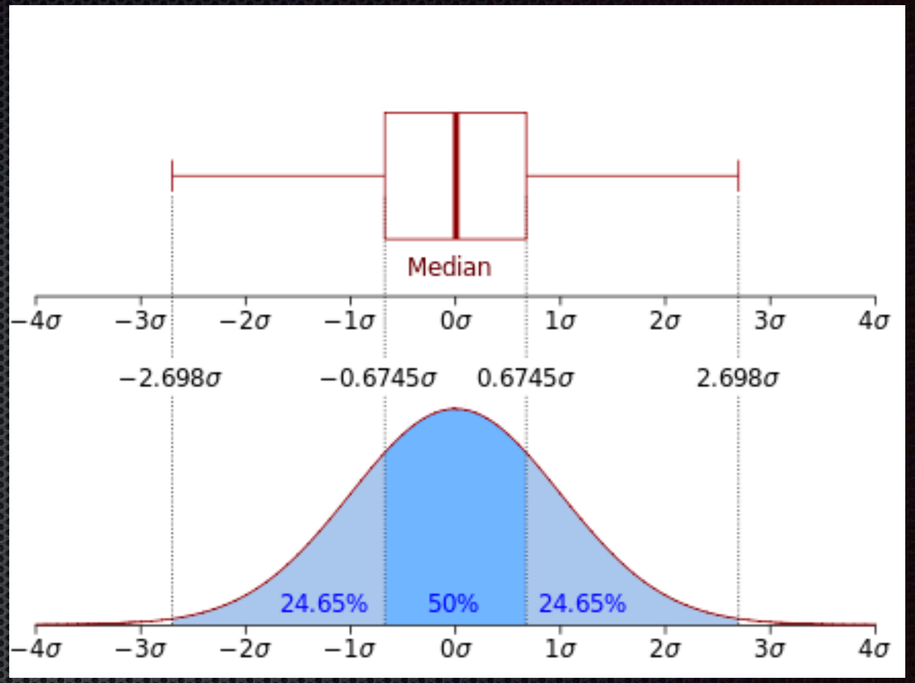
Data with outliers the ML model focuses in learning the features of the outlier points Not the features of the whole dataset

Features scaling

- Handling the Outliers

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





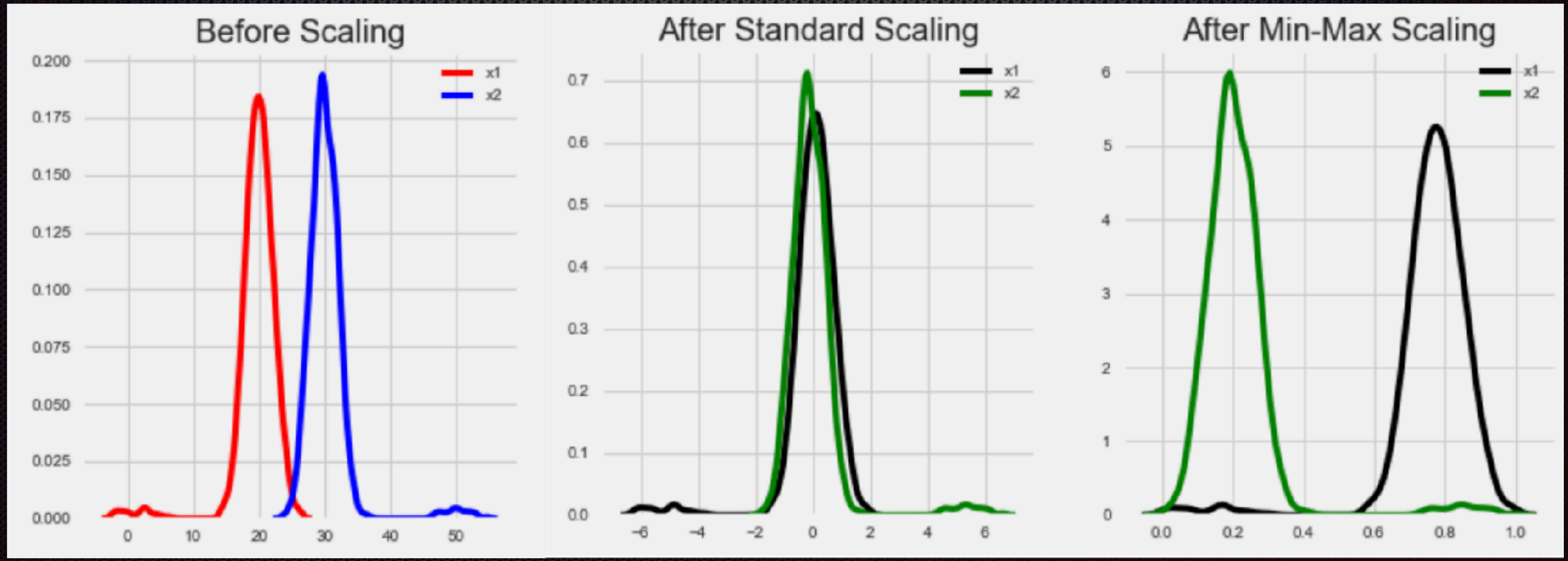
Box diagram for outliers detection

Features scaling

- Handling the Outliers

For large amount of outliers, scaling is needed

$$x^{\text{stand}} = \frac{x - \mu}{\sigma} \qquad x^{\text{minmax}} = \frac{x - x^{\min}}{x^{\max} - x^{\min}}$$

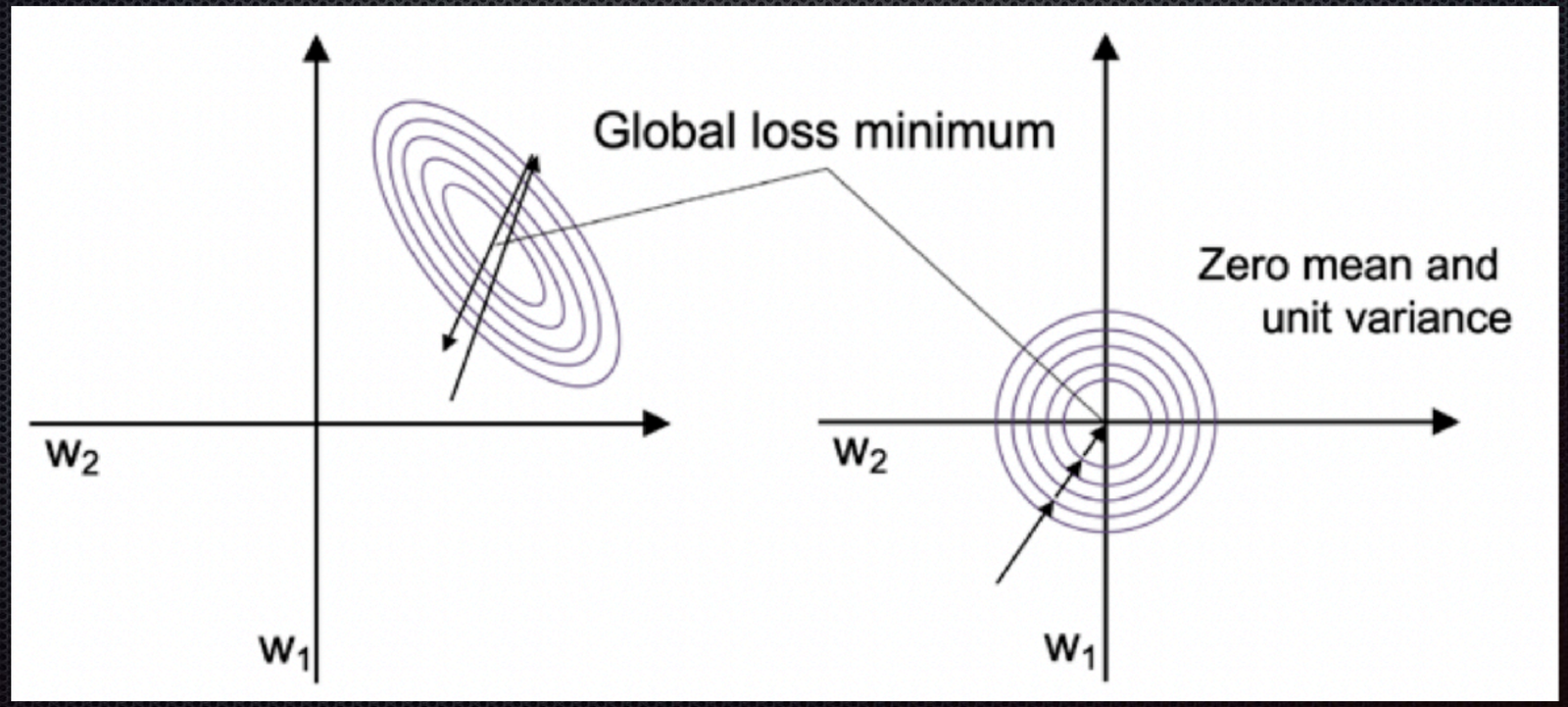


Features scaling

- Handling the Outliers

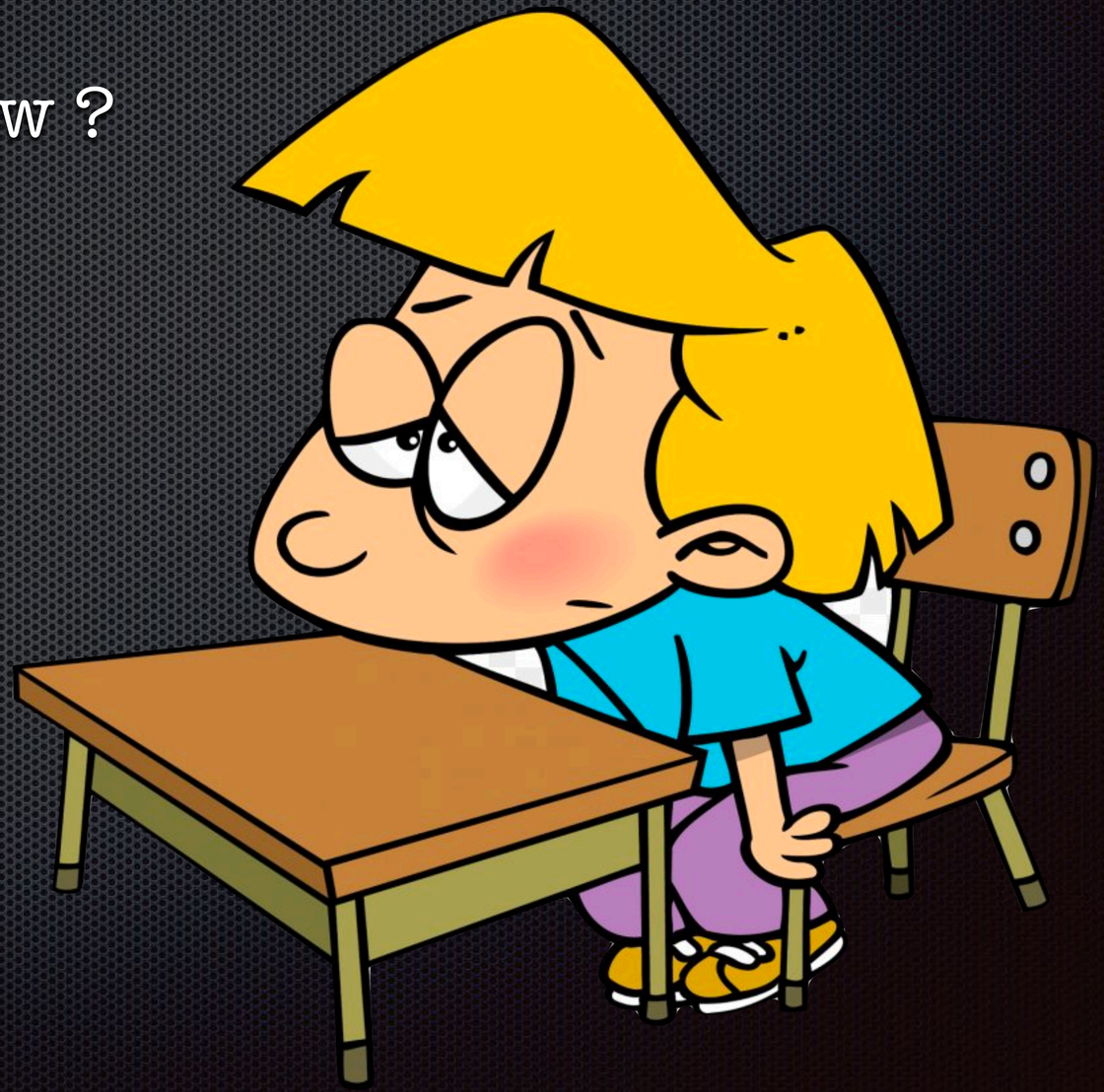
Data normalization is mandatory with gradient descent method.

If the features are on vastly different scales, a learning rate that works well for updating one weight might be too large or too small to update the other weights equally well.

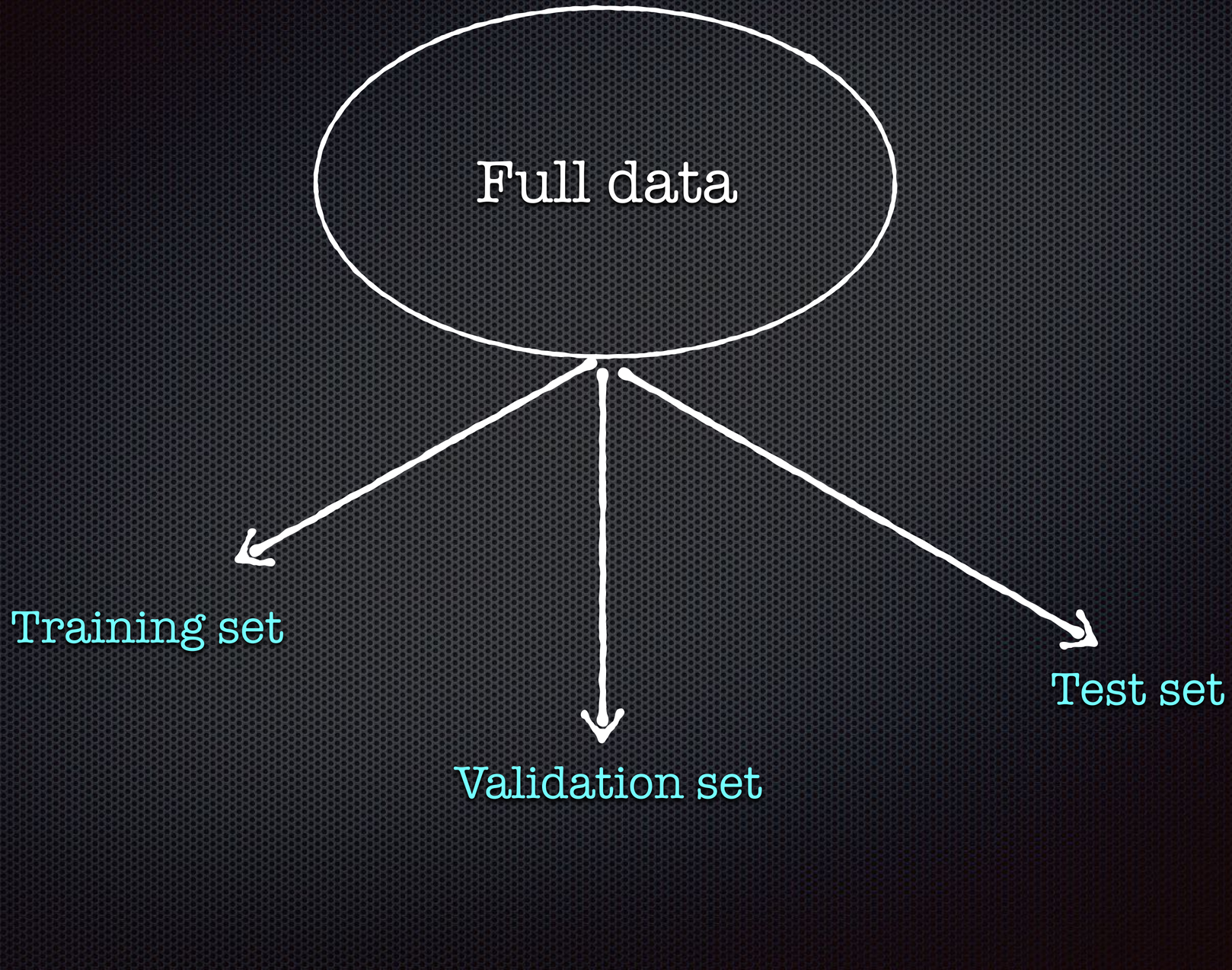


Deep learning

Well, can we train the DL model now ?



Data Splitting



Imbalanced training set

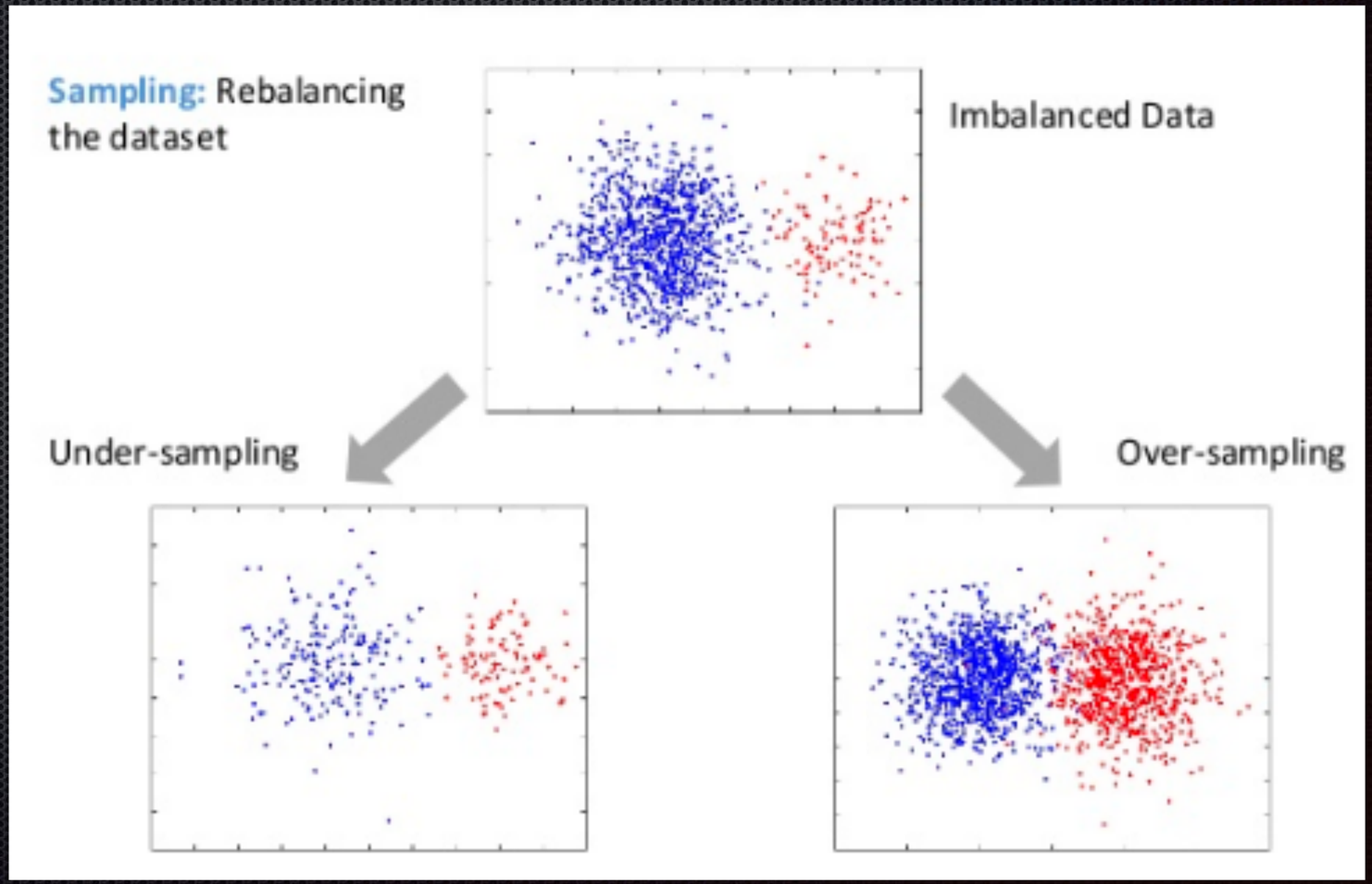
For the classification problem we need to train the model on equal size data from each class. For imbalanced data sets the model learns the features of the majority class over the minority one

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 !!

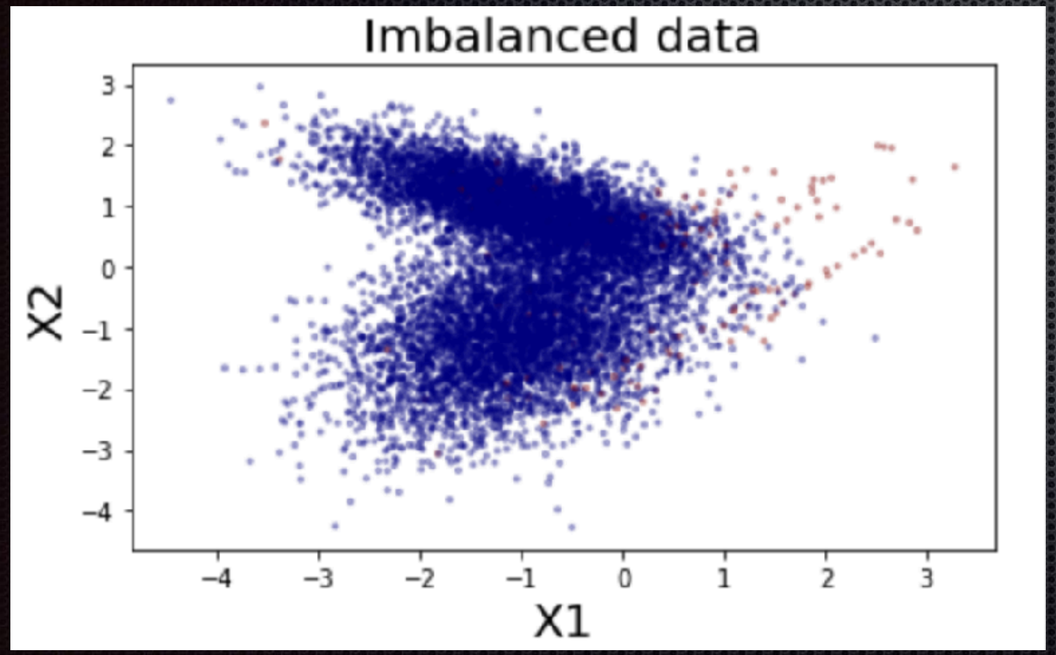


Imbalanced training set

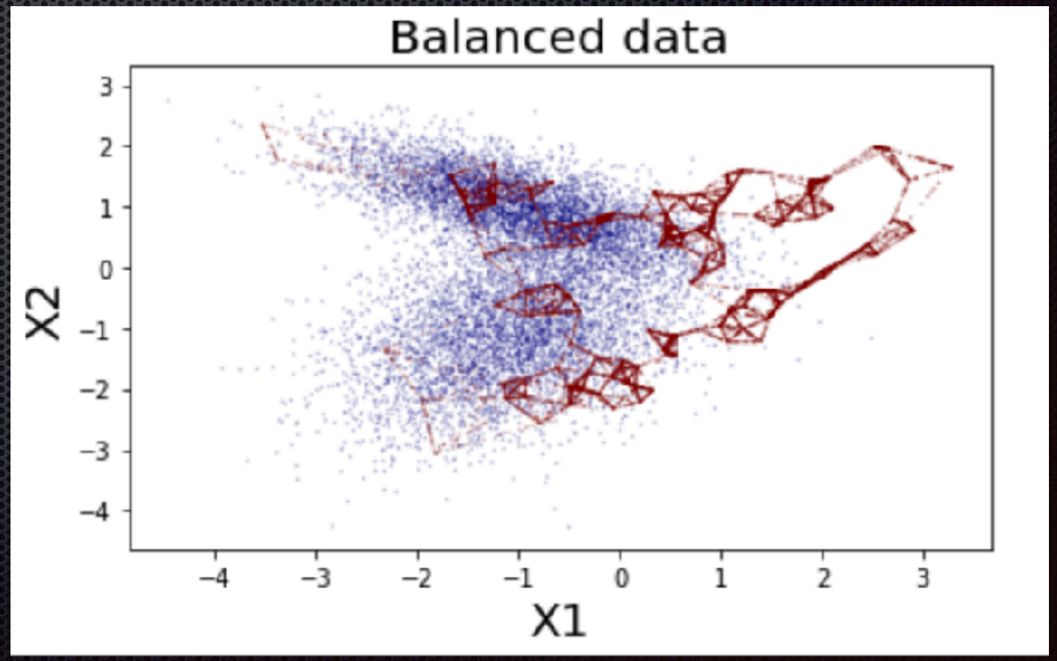
Synthetic Minority Over-sampling Technique (SMOTE)

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

Before SMOTE

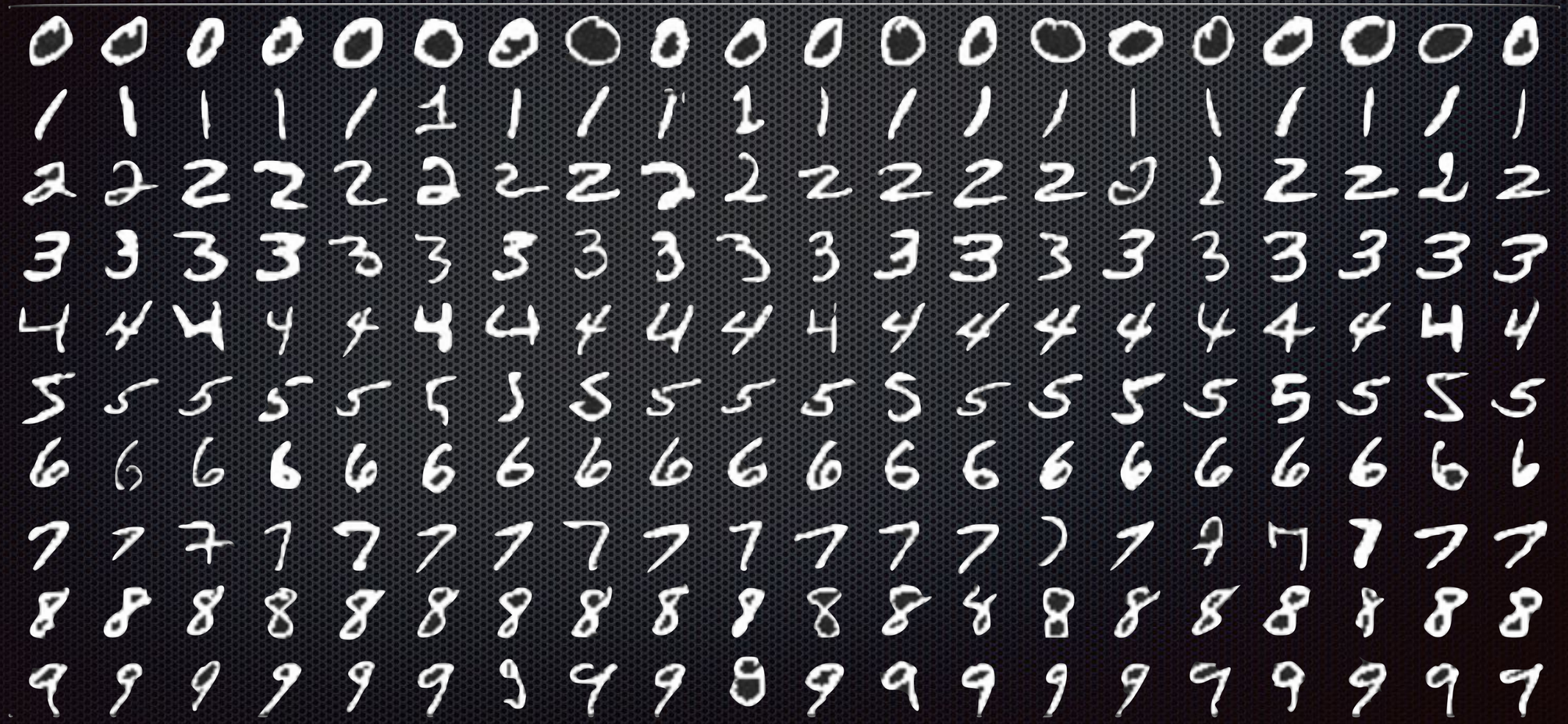


After SMOTE

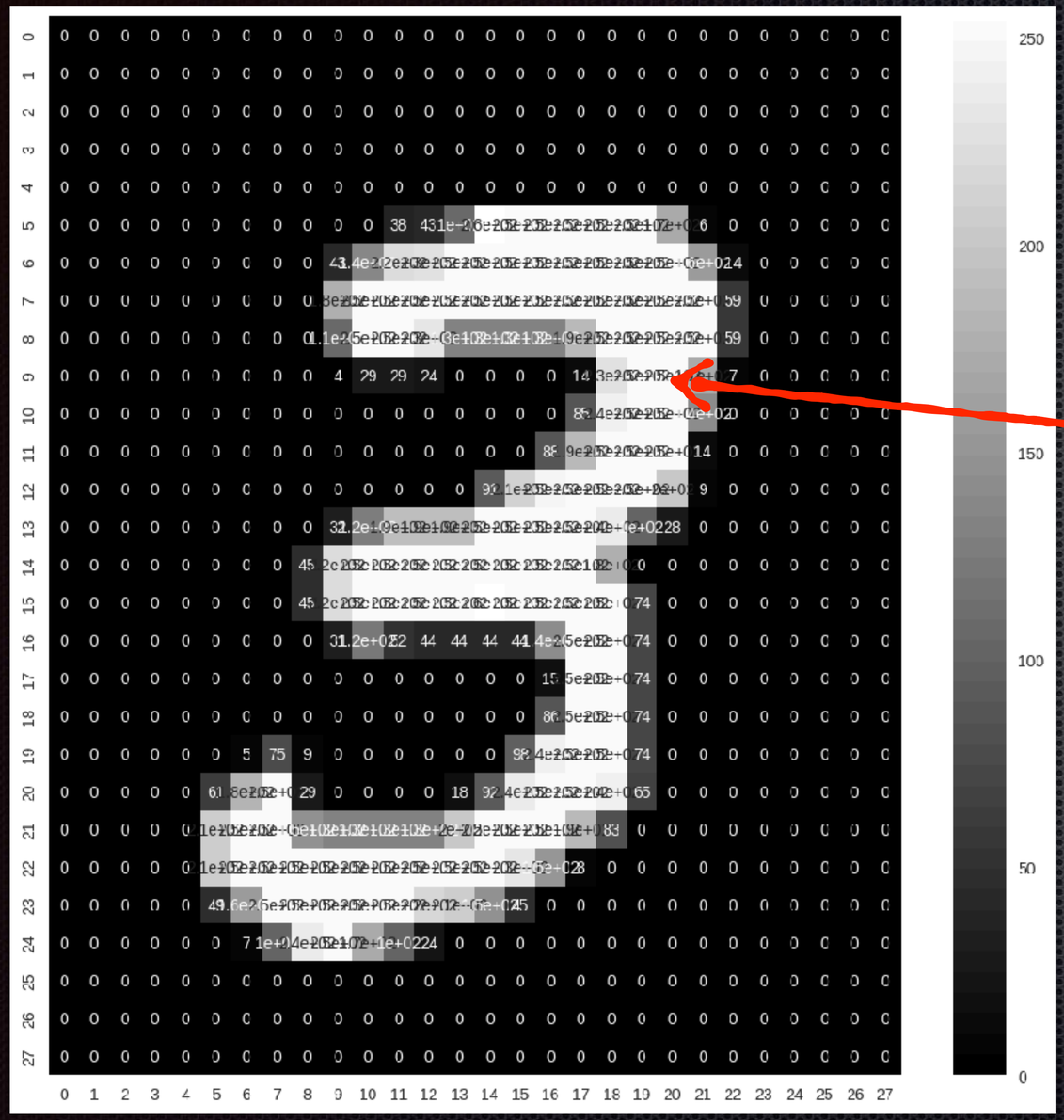


Example – MNIST data

Task: Construct DNN model to classify the hand written digits



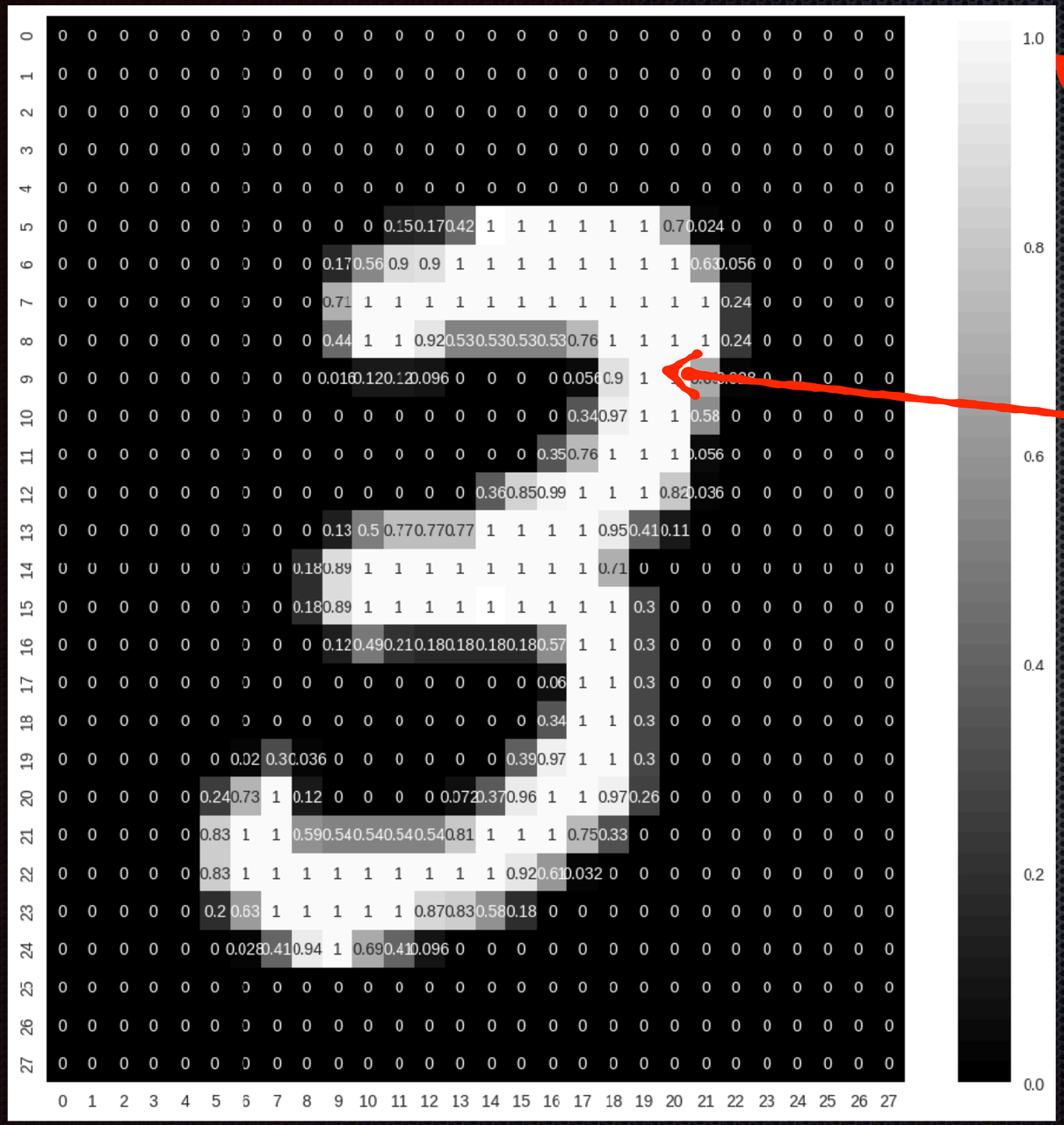
Example – MNIST data



Different scale

Of the pixels

Example – MNIST data

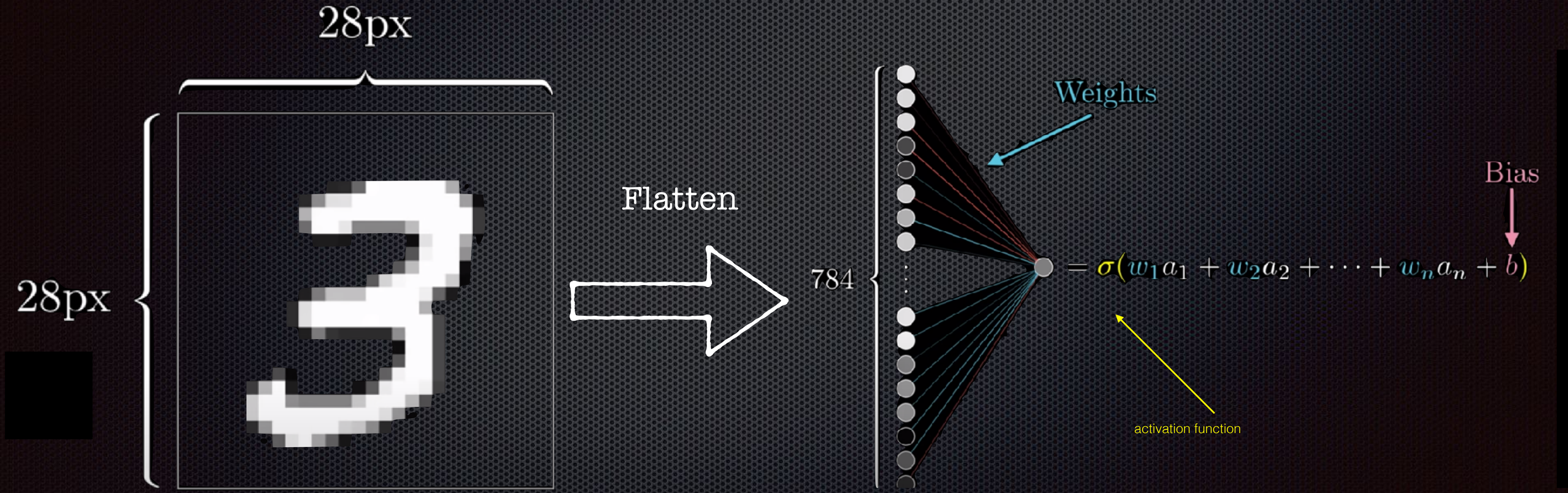


After min-max

Normalization

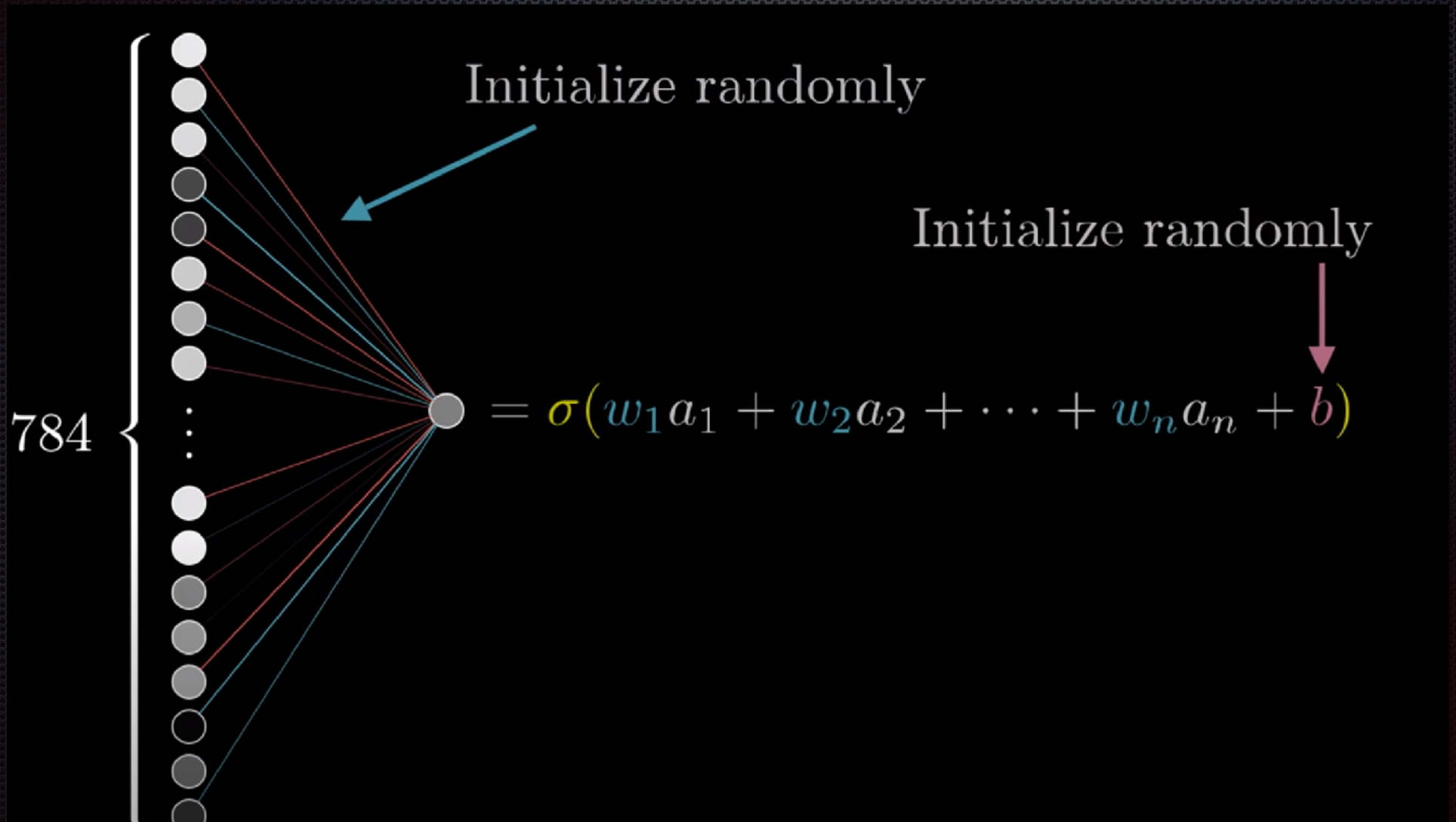
Example – MNIST data

Construct the first neuron in the first layer



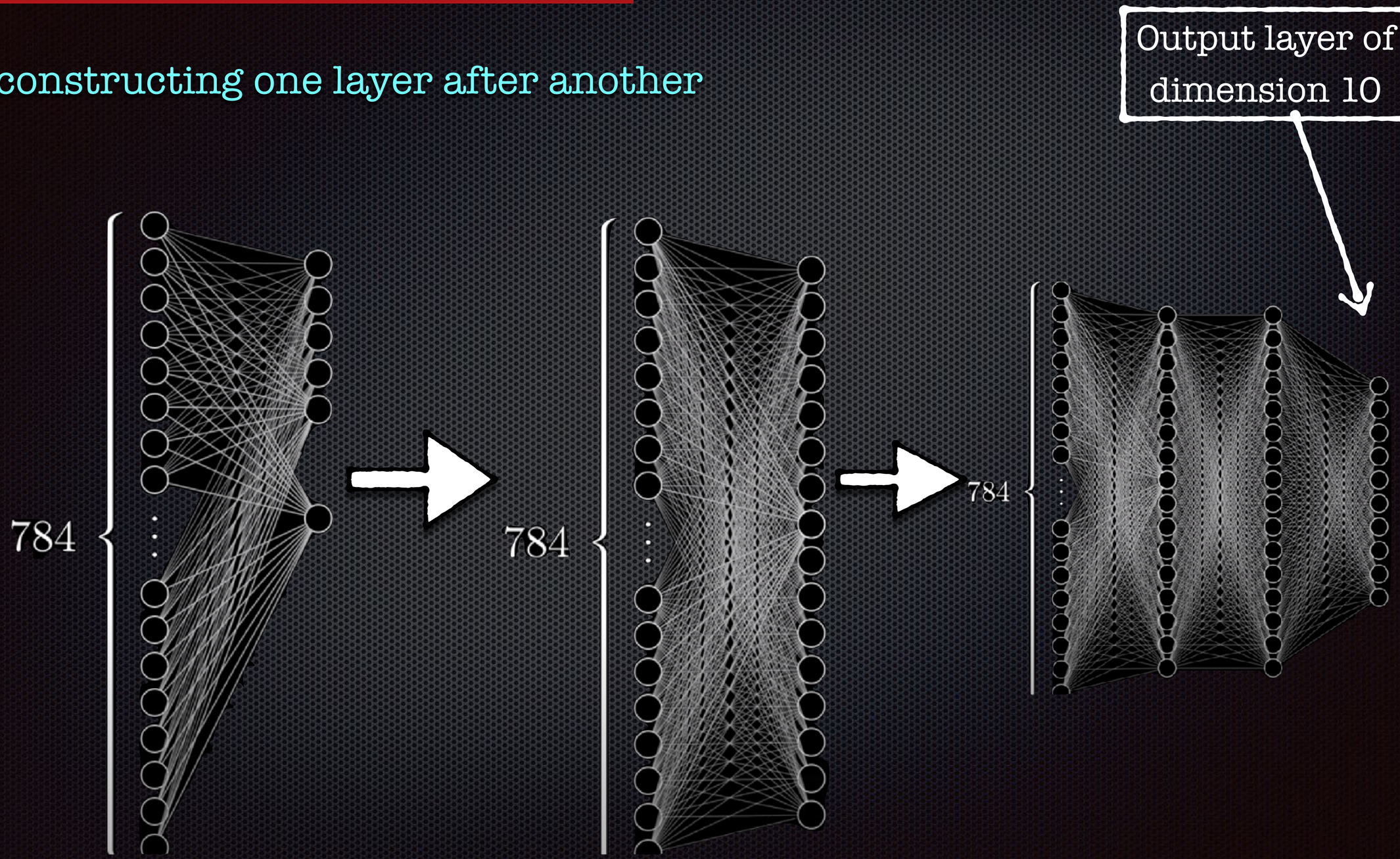
Example – MNIST data

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



Example – MNIST data

Keep constructing one layer after another

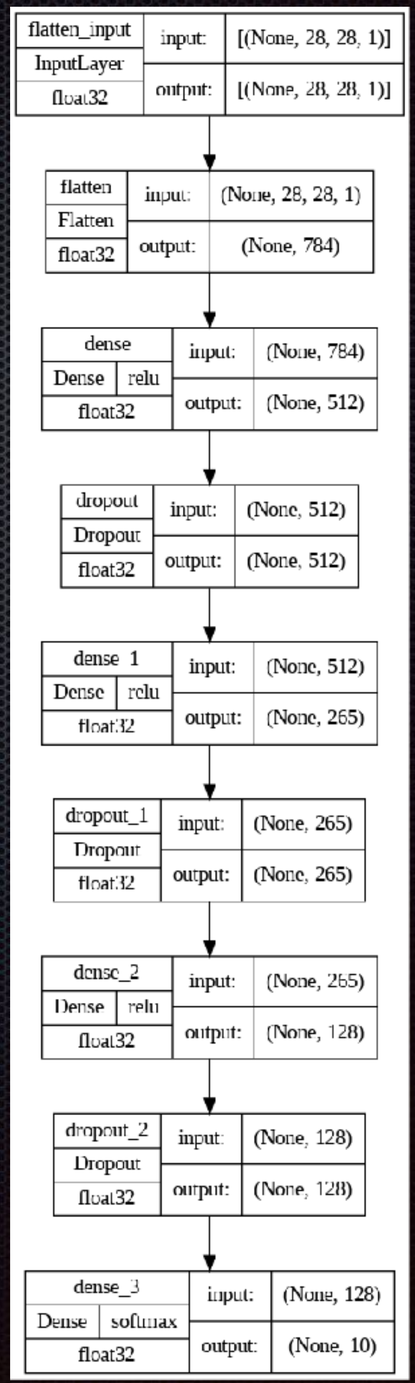


Example – MNIST data

Model summary

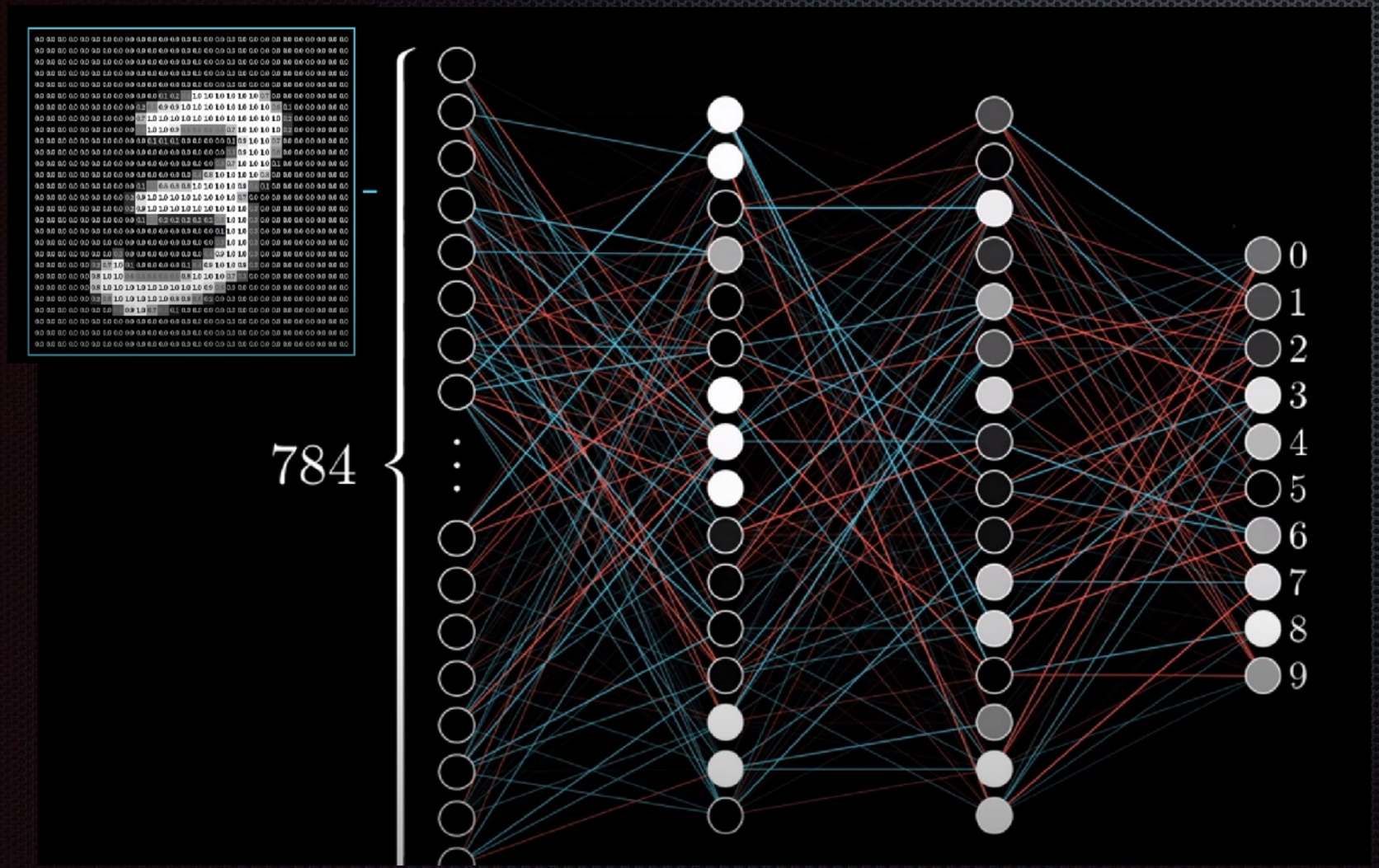
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 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



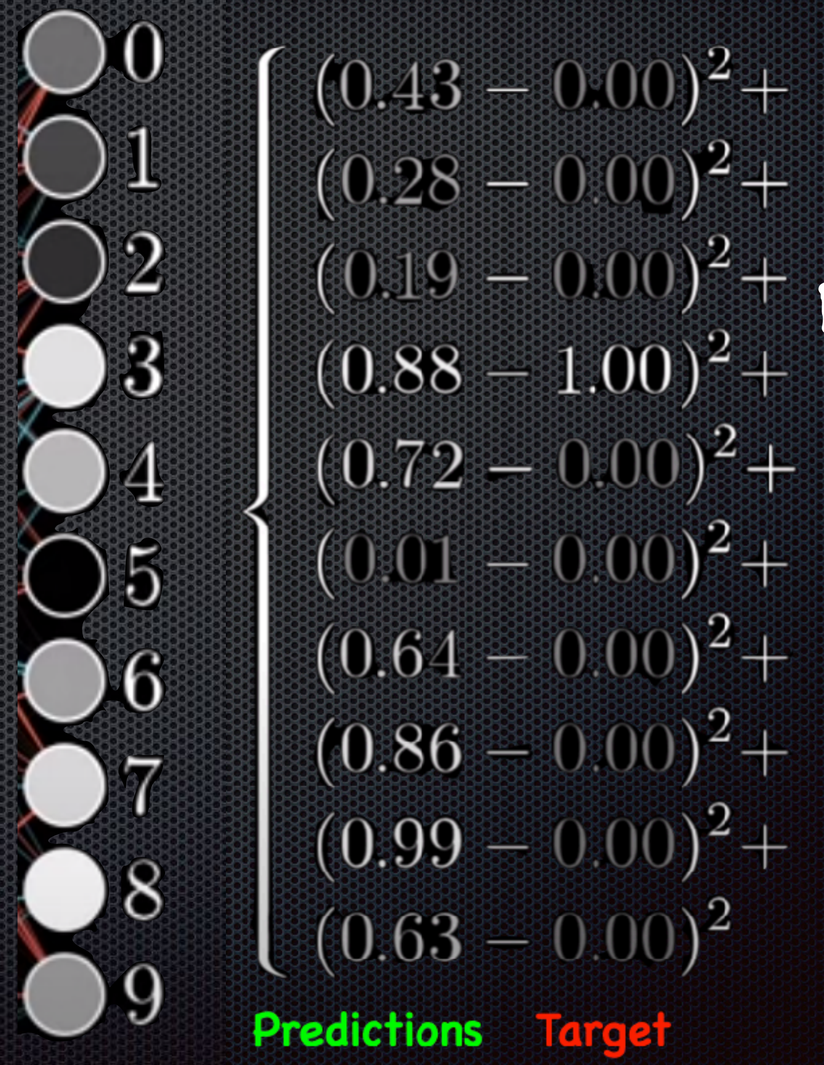
Model output

Prediction probability

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

Example – MNIST data

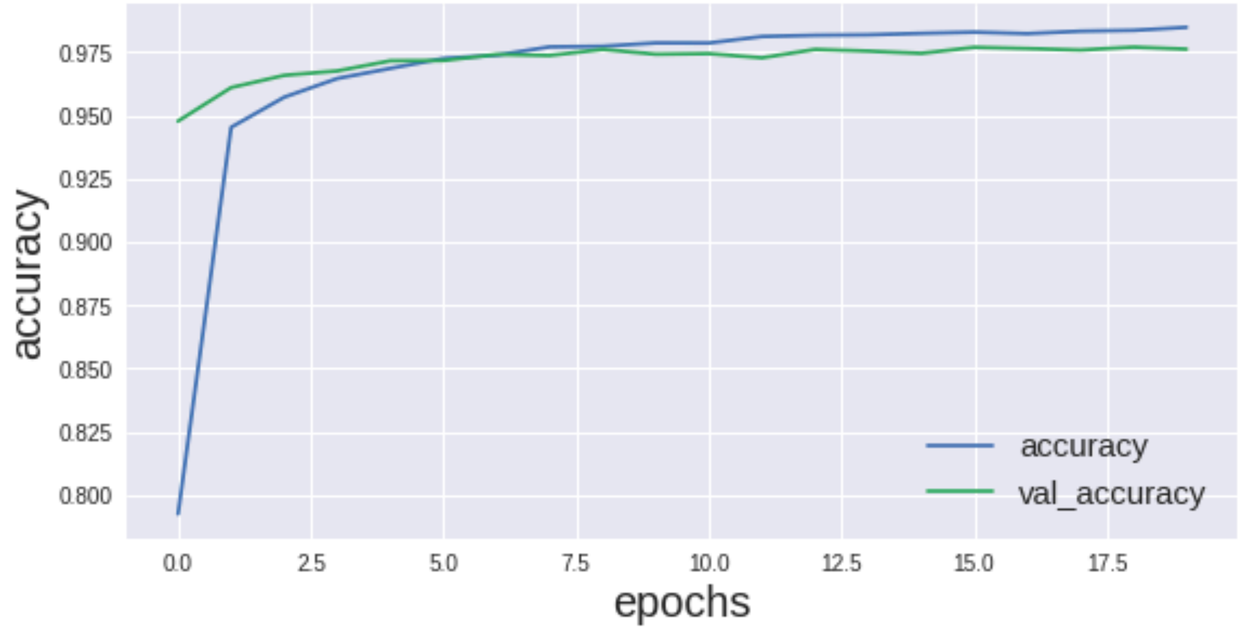
- Compute the loss function
- Back propagation to update the weights
- Repeat the iteration until to reach the desires accuracy

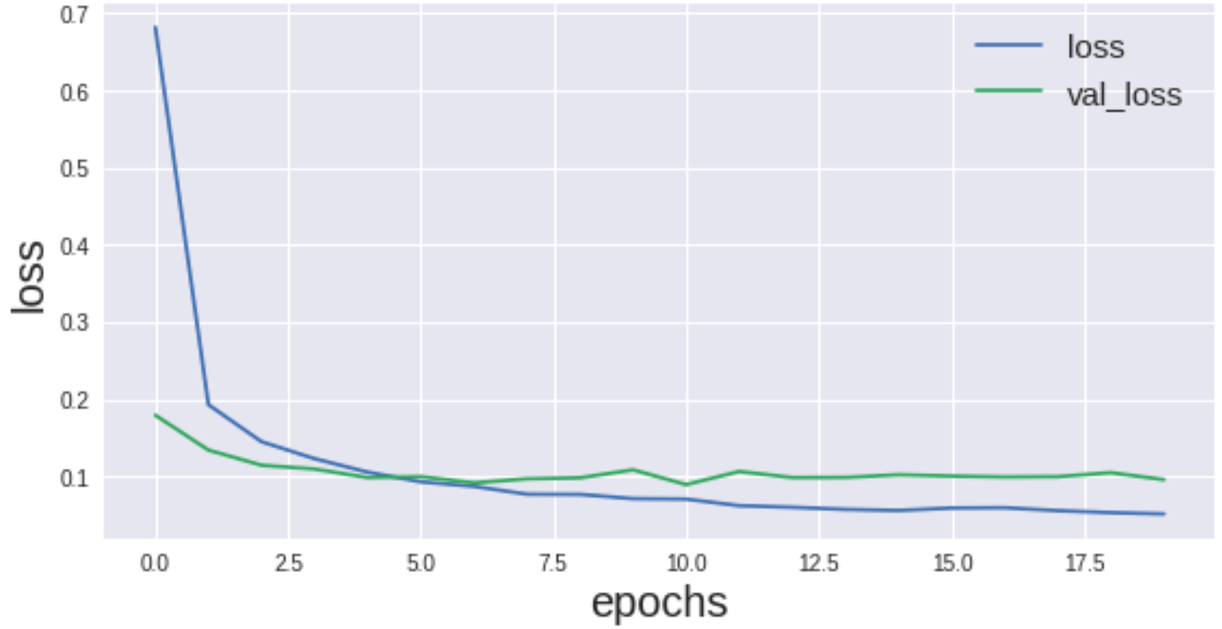


On-hot encoded Labels

Example – MNIST data

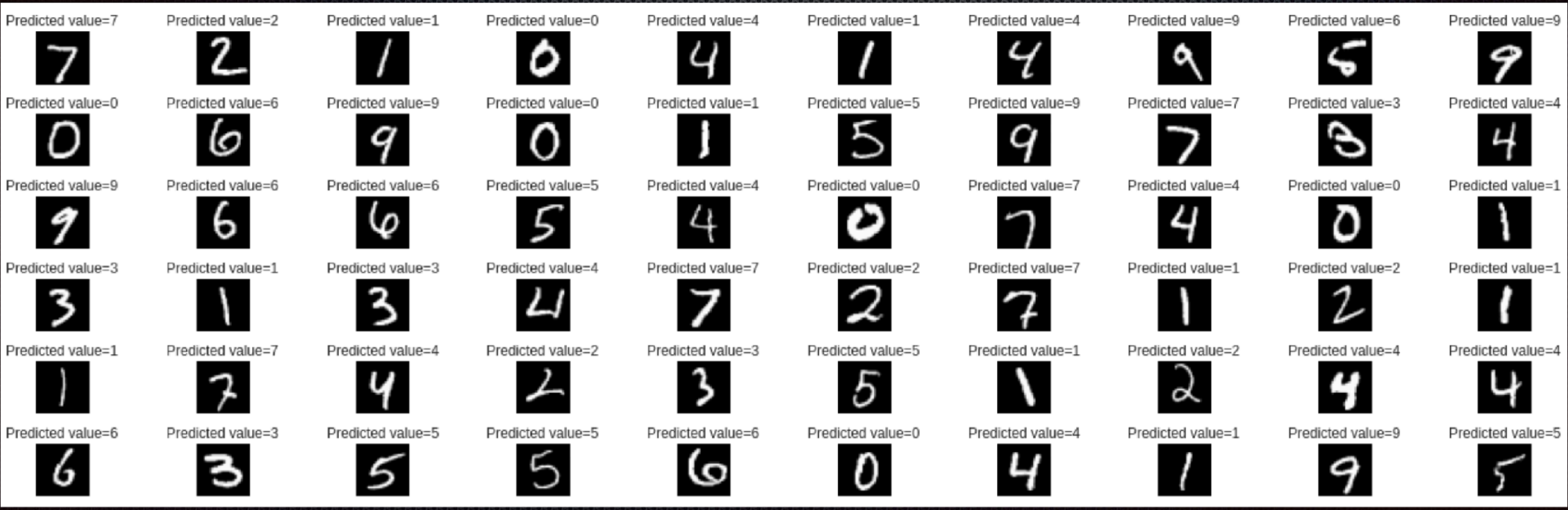
Test the model on the hold out validation sample





Example – MNIST data

Do some predictions



To be continued…