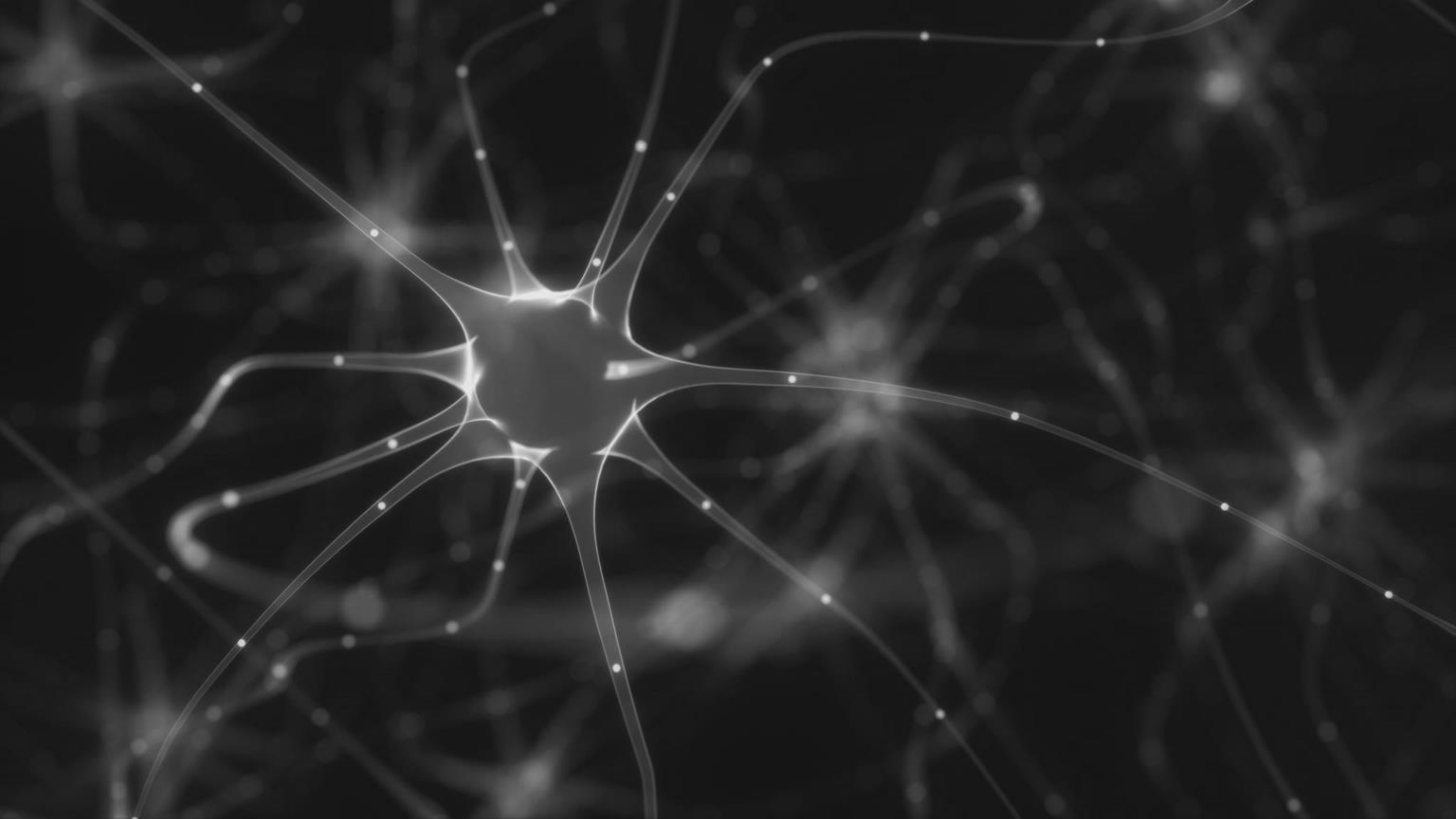
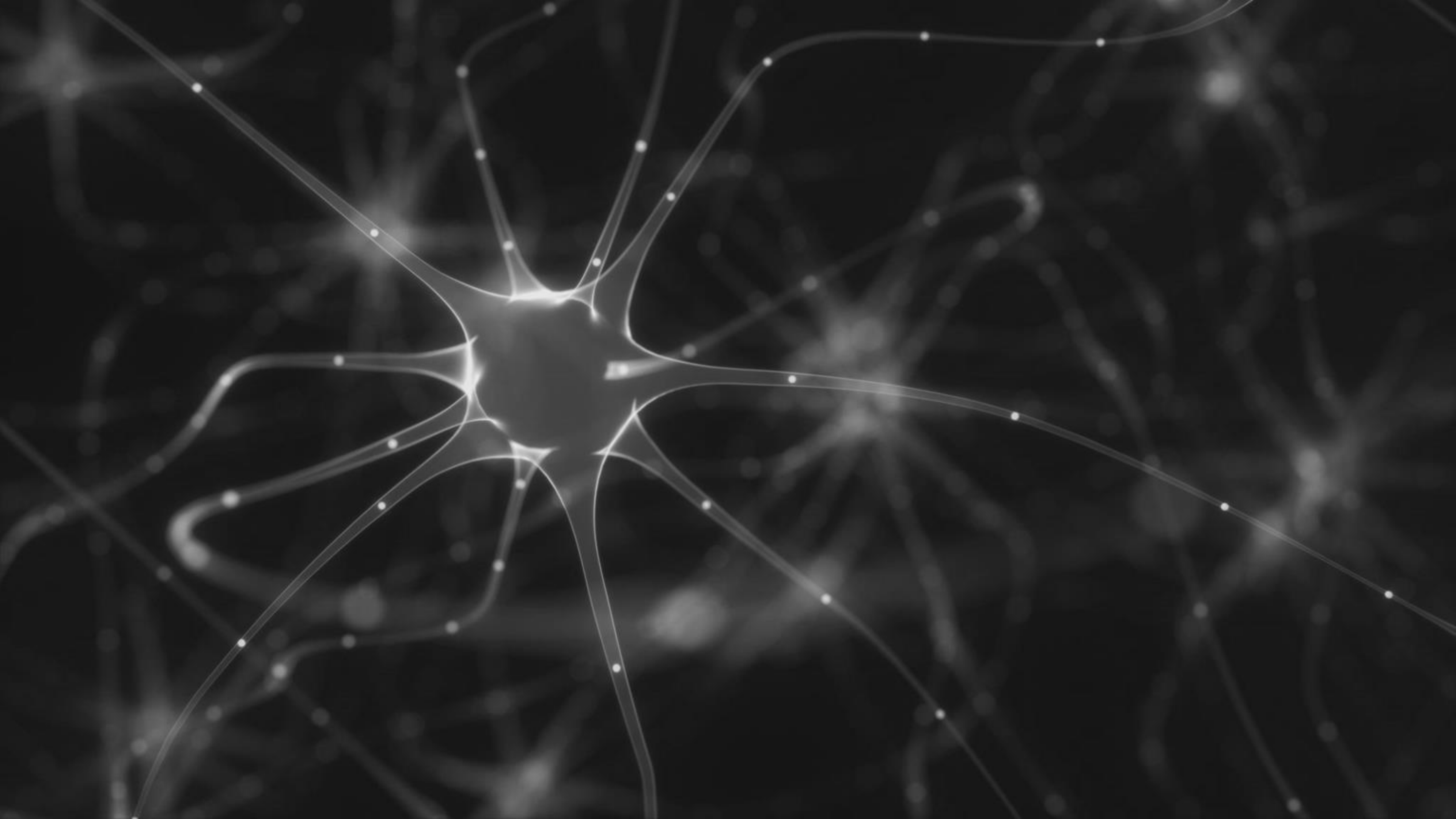


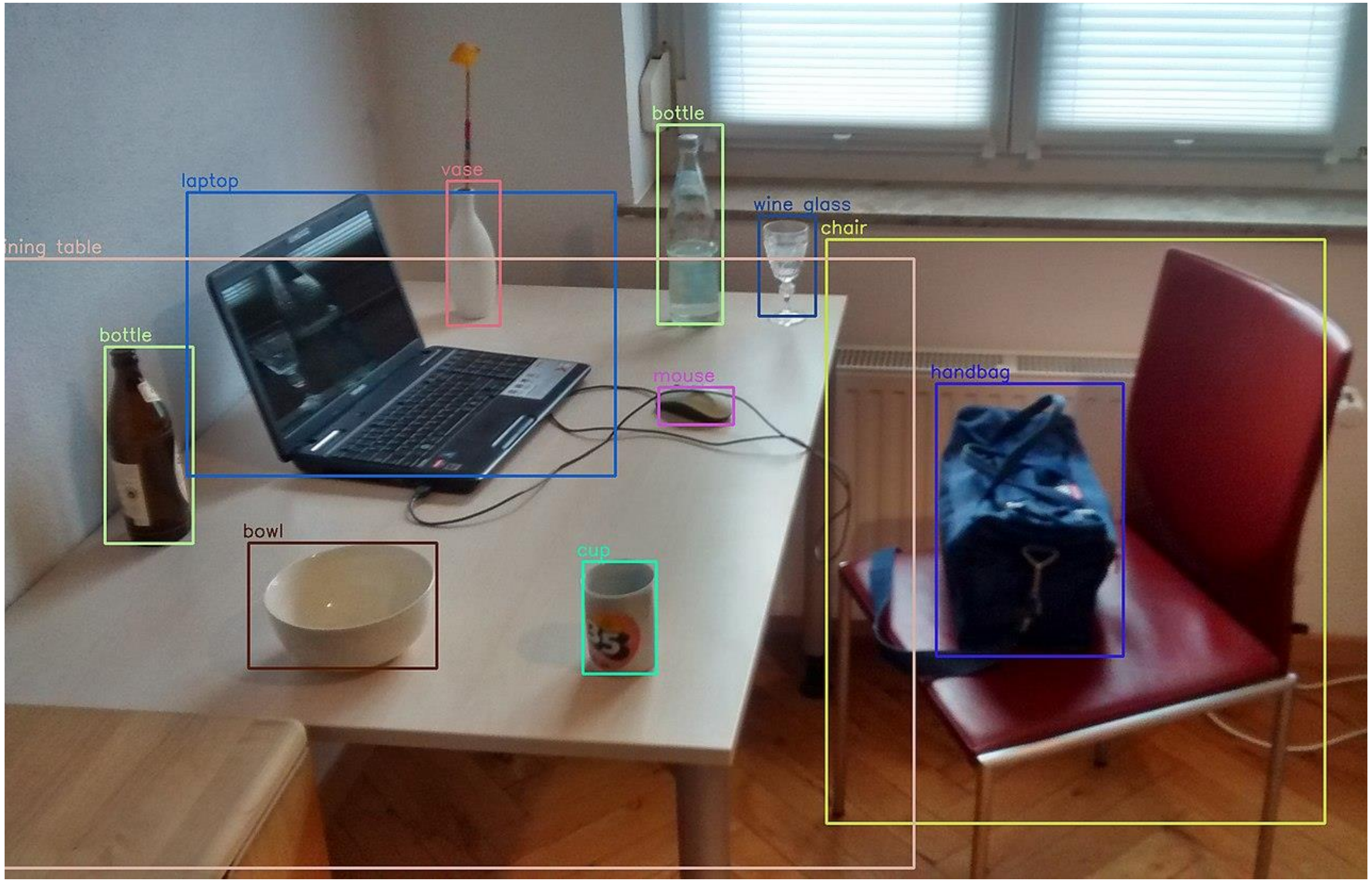
AGENDA

- State-of-the-art
- What changed?
- Fundamentals
- Deep Learning
- Examples / Practice

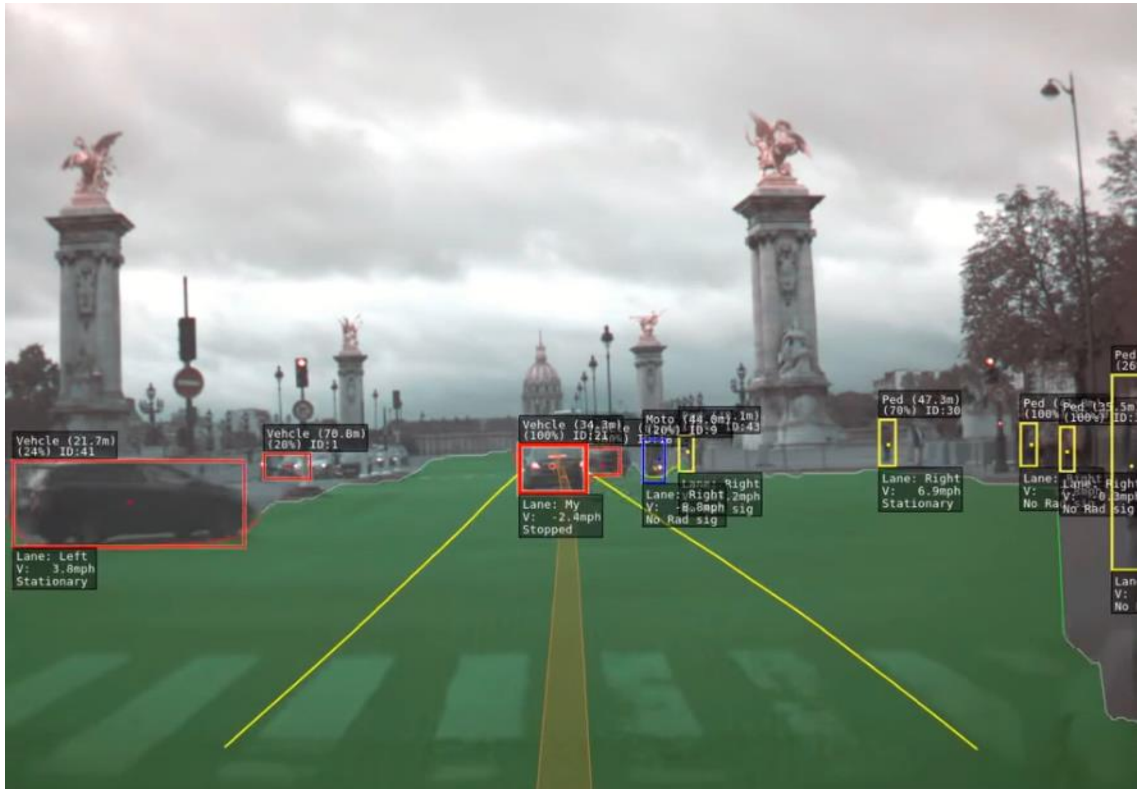


COMPUTER VISION

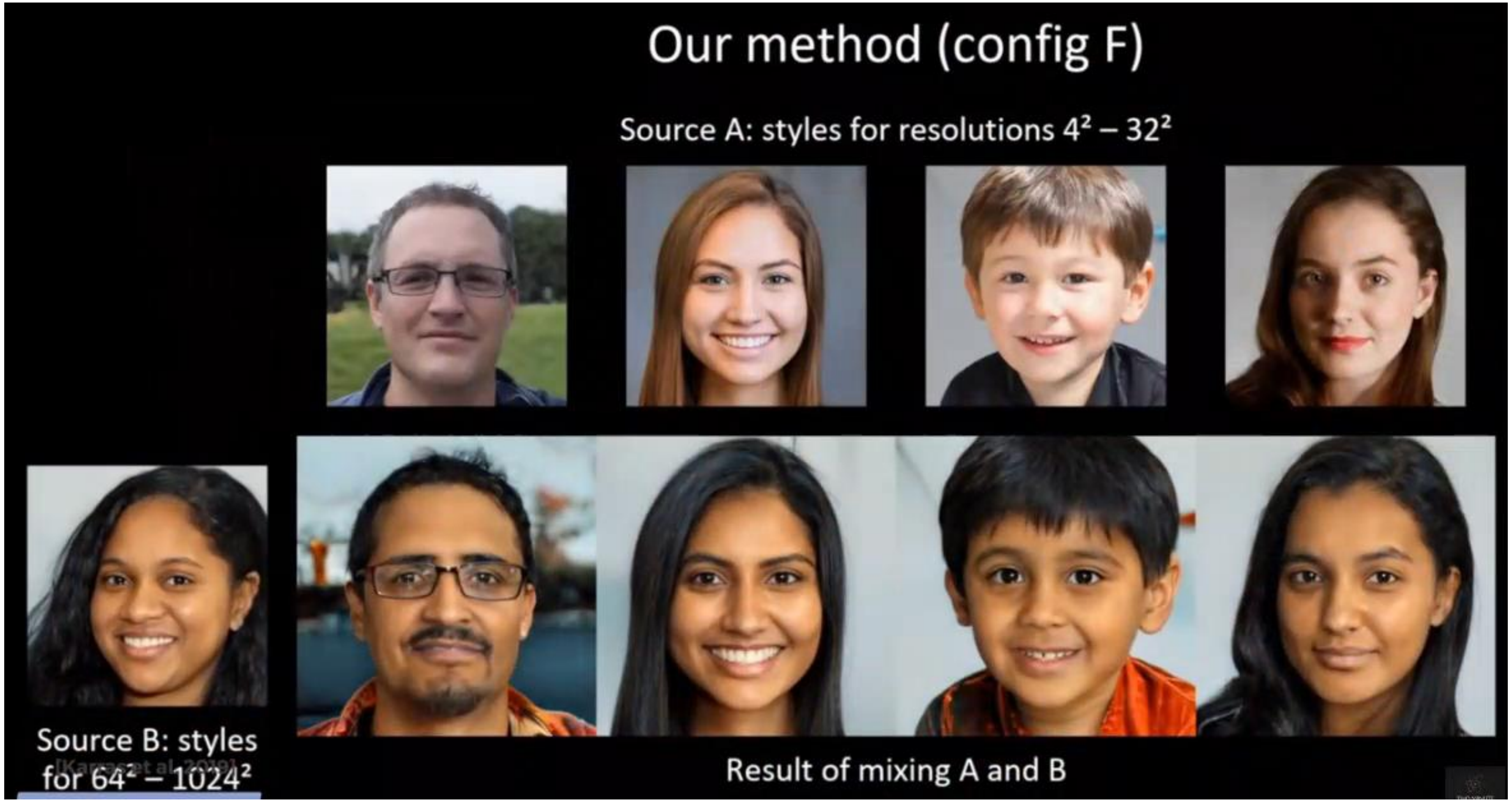
SEMANTIC SEGMENTATION AND OBJECT DETECTION



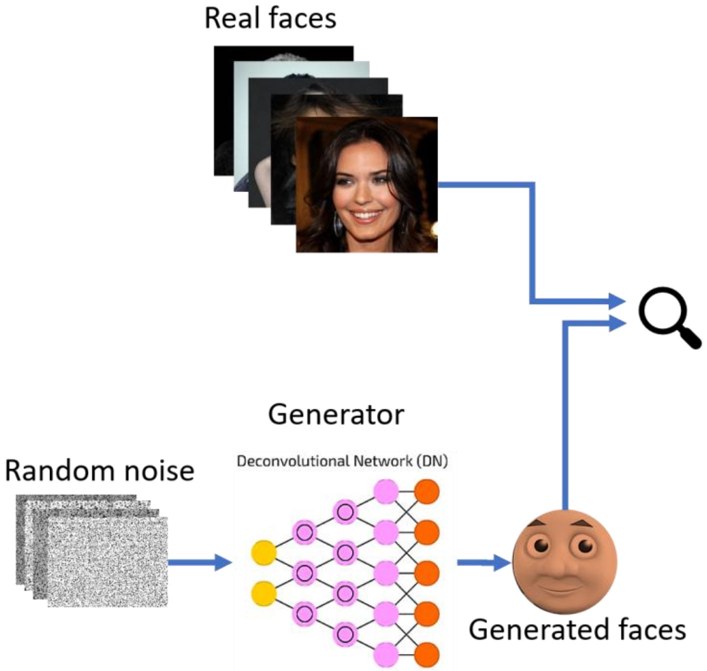
SEMANTIC SEGMENTATION AND OBJECT DETECTION

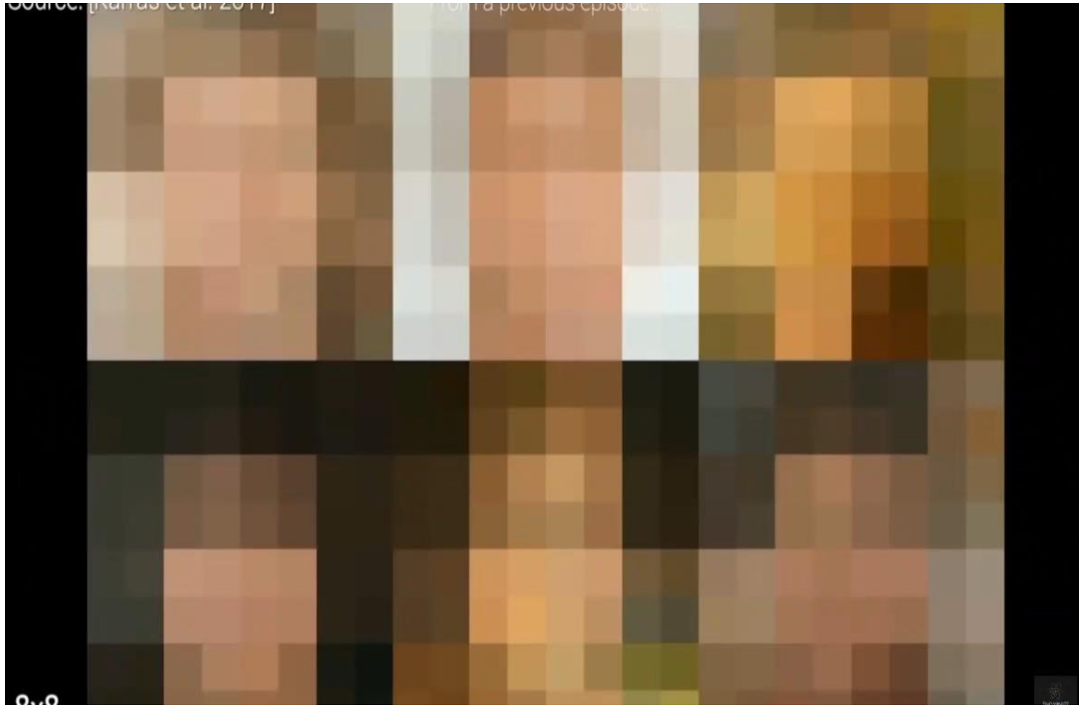


GENERATIVE ADVERSIAL NETWORKS (GANs)



GENERATIVE ADVERSIAL NETWORKS (GANs)





GENERATIVE ADVERSIAL NETWORKS (GANs)

NATURAL LANGUAGE PROCESSING

NATURAL LANGUAGE PROCESSING (NLP)

Write With Transformer gpt2

The meaning of life is not what I think it is, it's what I do to make it.

The limits of deep learning are still in the process of being figured out.

Most important person in the history of deep learning is probably Andrew Ng.

Lex Fridman's best quality is that he's smart, but I think he gets more attention than he deserves.

SYSTEM PROMPT (HUMAN-WRITTEN)

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

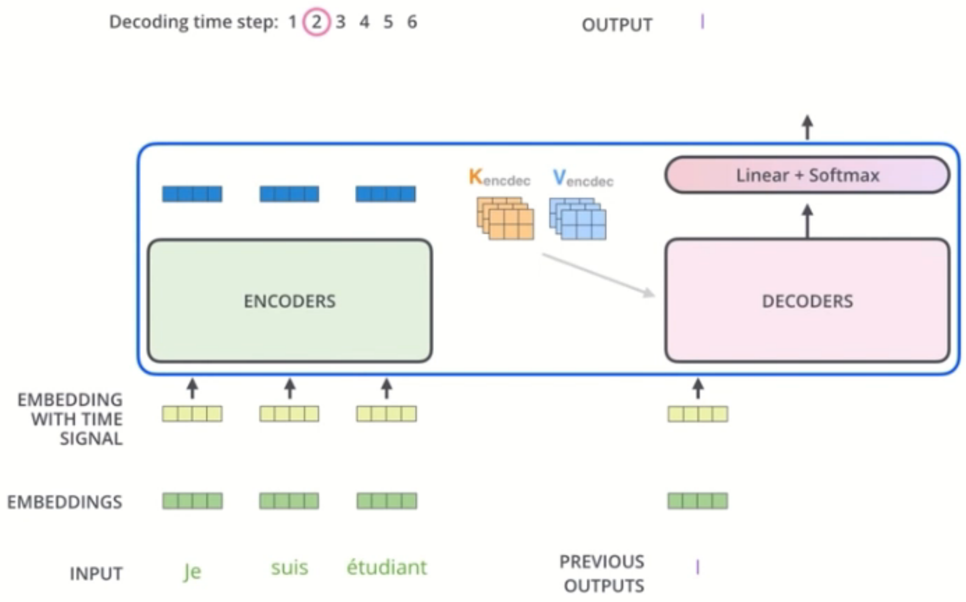
While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Pérez stated, "We can see, for example, that they have a common 'language,' something like a dialect or dialectic."

Dr. Pérez believes that the unicorns may have originated in Argentina, where the animals were believed to be descendants of a lost race of people who lived there before the arrival of humans in those parts of South America.

NATURAL LANGUAGE PROCESSING (NLP)

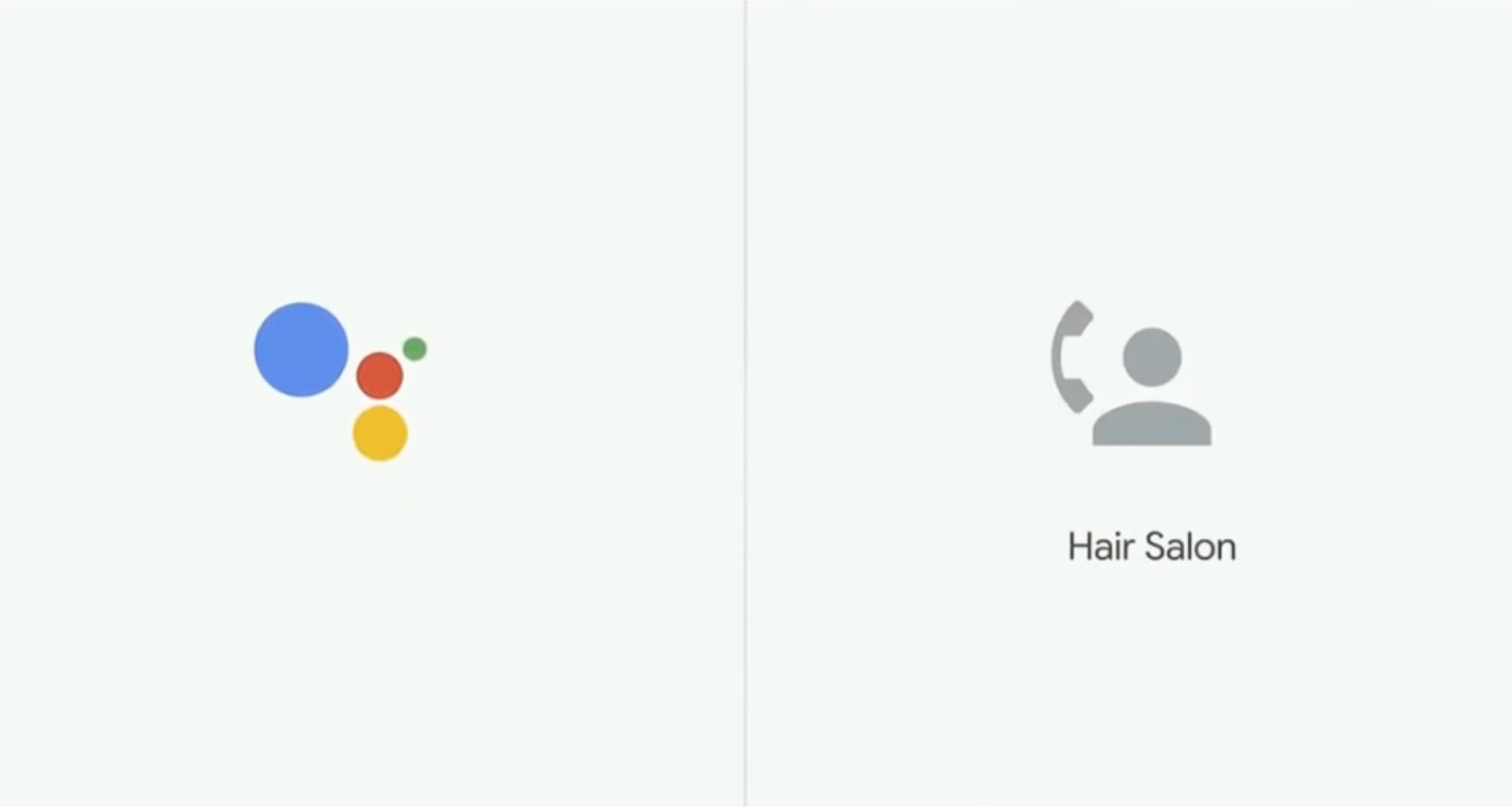
Question:	While eating a hamburger with friends, what are people trying to do?
Choices:	**have fun**, tasty, or indigestion
CoS-E:	Usually a hamburger with friends indicates a good time.
Question:	After getting drunk people couldn't understand him,it was because of his what?
Choices:	lower standards,**slurred speech**, or falling down
CoS-E:	People who are drunk have difficulty speaking.
Question:	People do what during their time off from work?
Choices:	**take trips**, brow shorter, or become hysterical
CoS-E:	People usually do something relaxing, such as taking trips,when they don't need to work.

Transformer



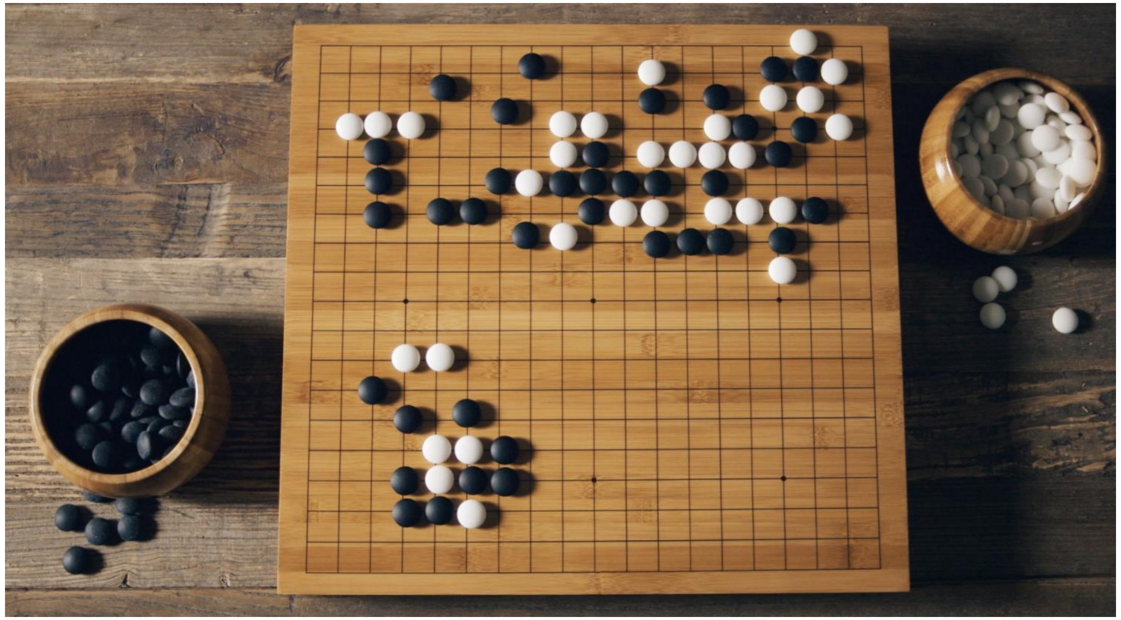
Vaswani et al. "Attention is all you need." *Advances in Neural Information Processing Systems*. 2017.

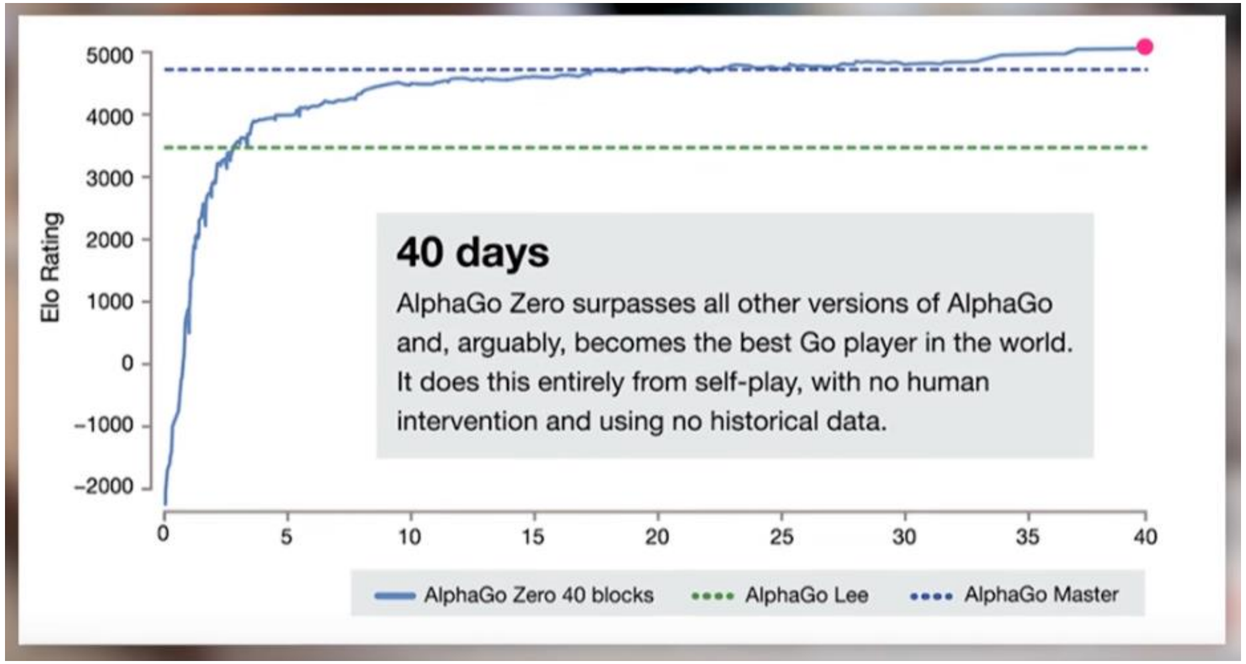
NATURAL LANGUAGE PROCESSING (NLP)



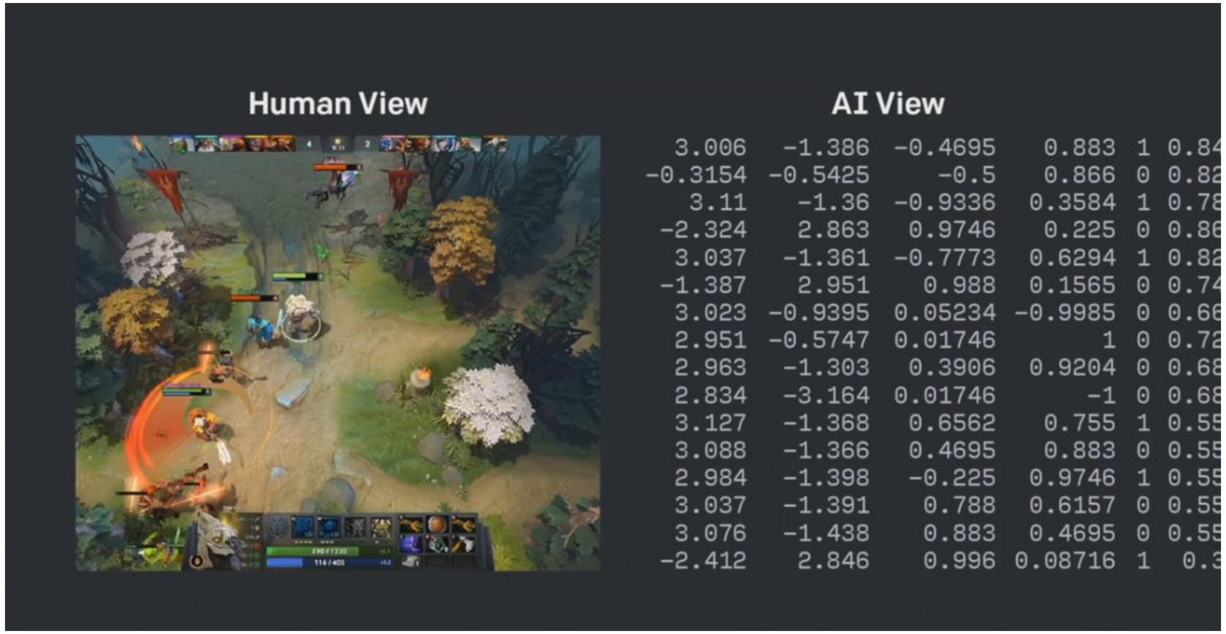
REINFORCEMENT LEARNING

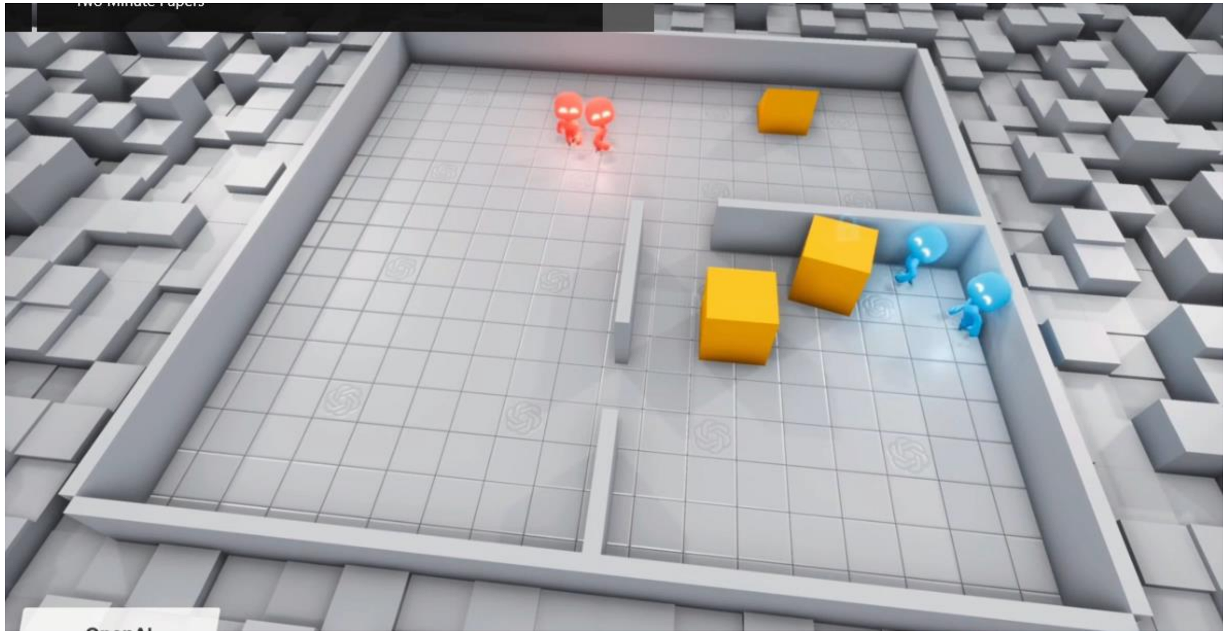
REINFORCEMENT LEARNING

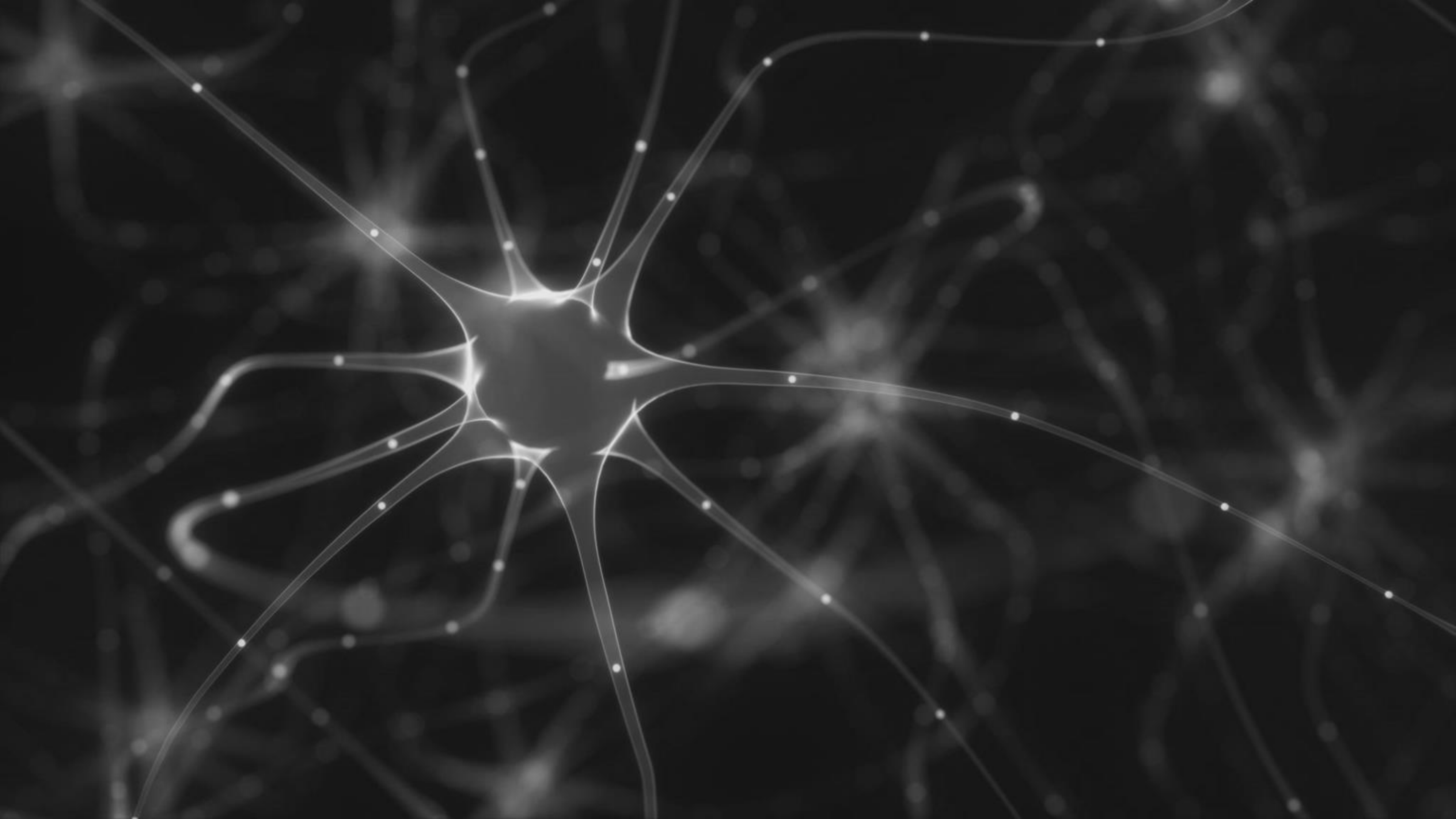




REINFORCEMENT LEARNING

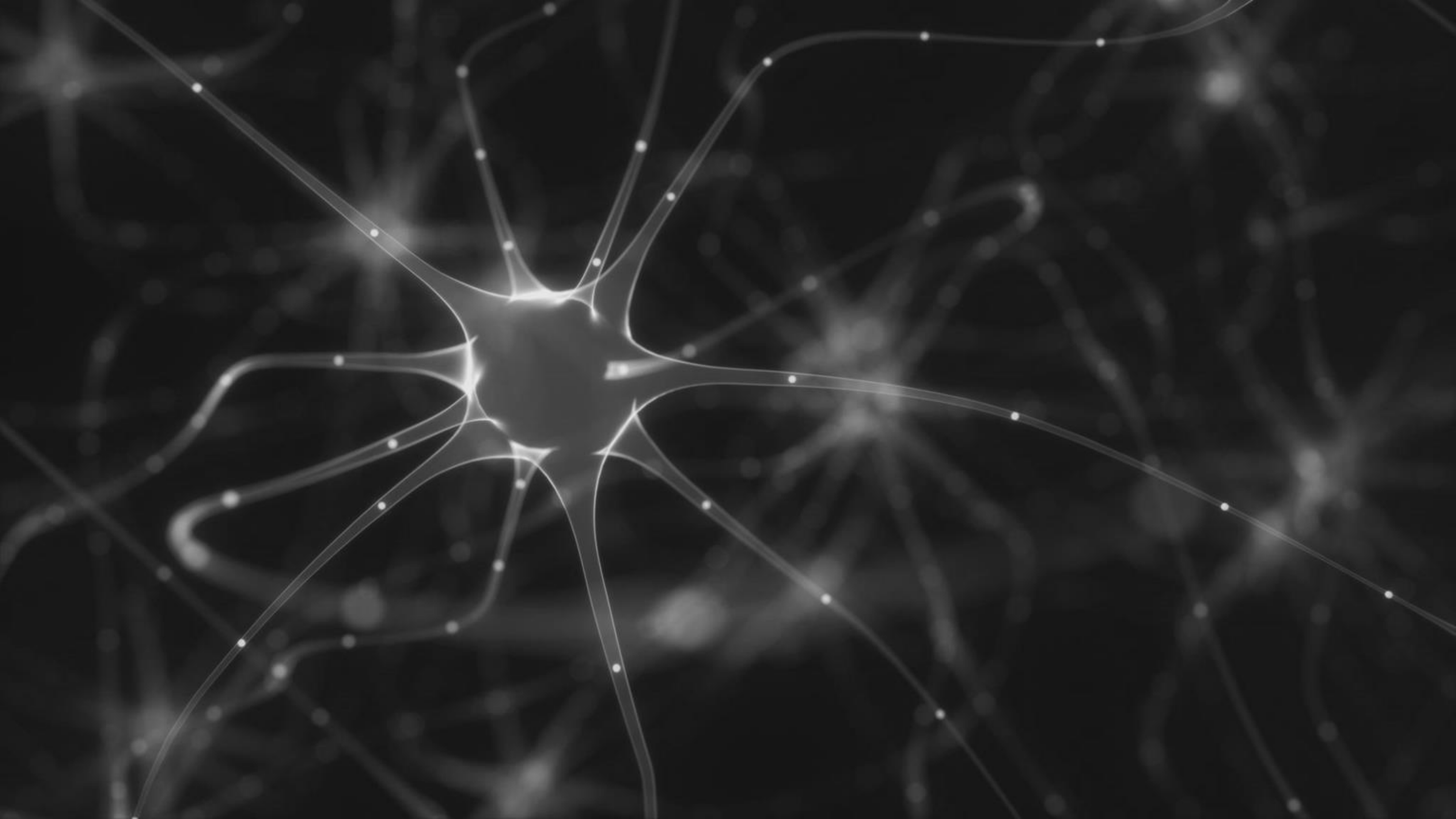




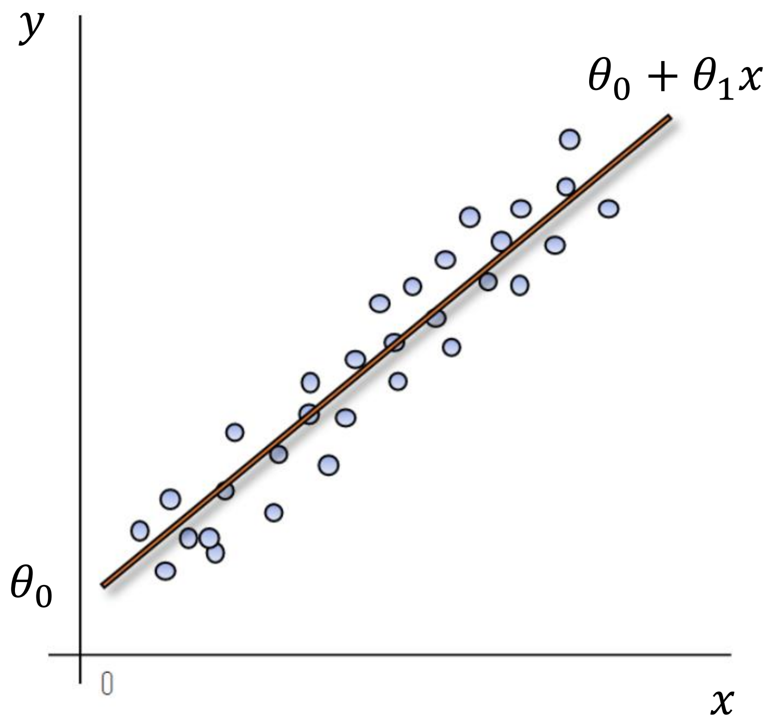


WHAT CHANGED?

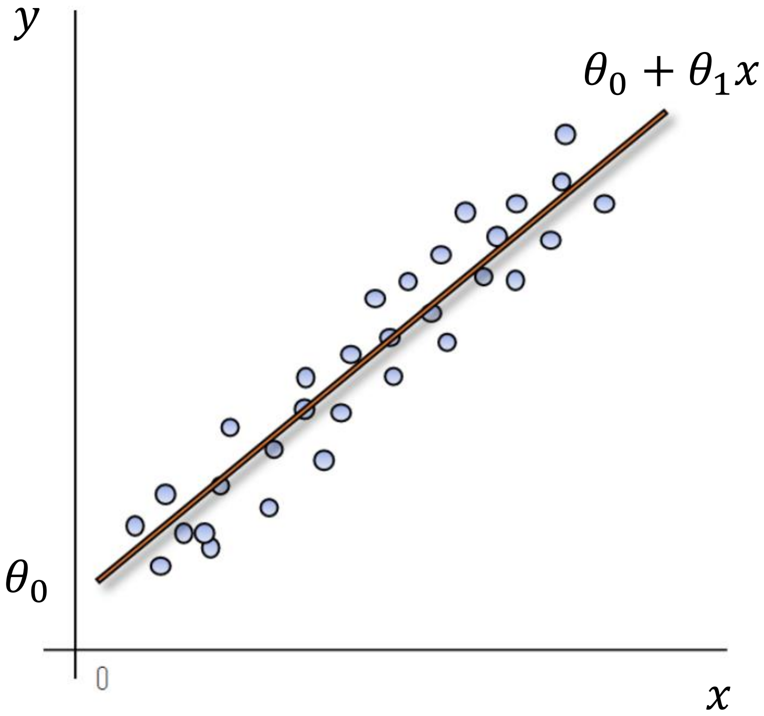
- Availability of high quality labelled data
- Quantity of data
- Advances in computational resources
- Better understanding of deep neural networks
- Solved issues in DNN learning process
    - Dropout/regularization
    - Batch Normalization
    - Vanishing/Exploding Gradients



LINEAR REGRESSION



LINEAR REGRESSION



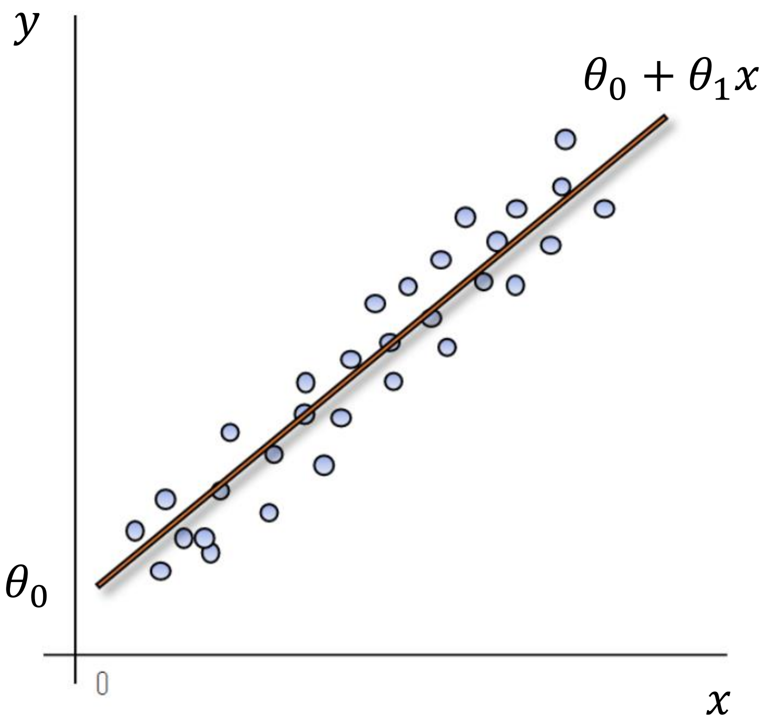
Univariate

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Multivariate

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n$$

LINEAR REGRESSION



Univariate

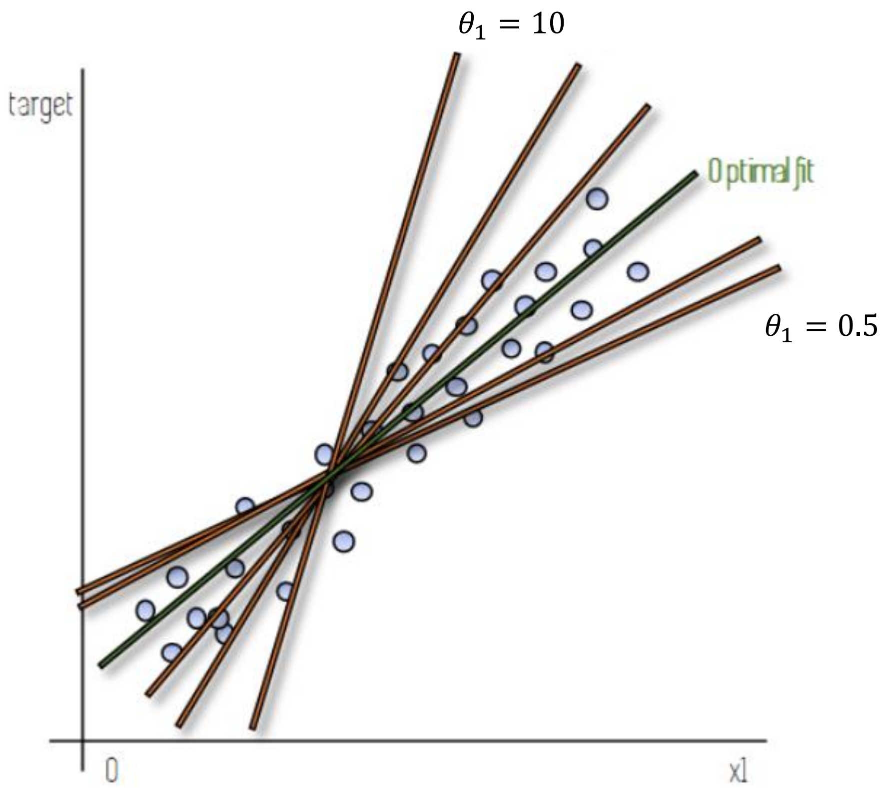
$$h_\theta(x) = \theta_0 + \theta_1 x$$

Multivariate

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n$$

$$h_\theta(x) = \theta_0 + \theta . x, \text{ with } \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \text{ and } x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

LINEAR REGRESSION



Graphical representation of the different iterations of a linear regression model with one feature (x1)

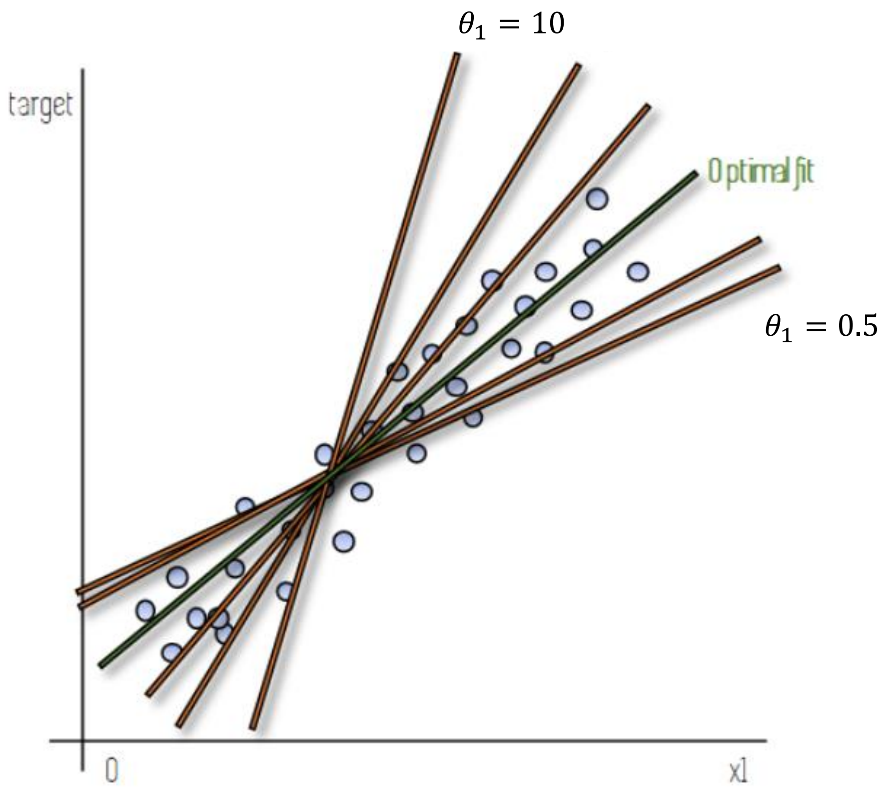
$$h_\theta(x) = \theta_0 + \theta_1 x$$

$$\theta_1 \; search \; space \in [0.5; \; 10]$$

Iteratively explore all options in space and compare them using an evaluation metric (for example Mean Absolute Error)

$$optimal \; \theta_1 = 1$$

LINEAR REGRESSION



Graphical representation of the different iterations of a linear regression model with one feature (x1)

$$h_\theta(x) = \theta_0 + \theta_1 x$$

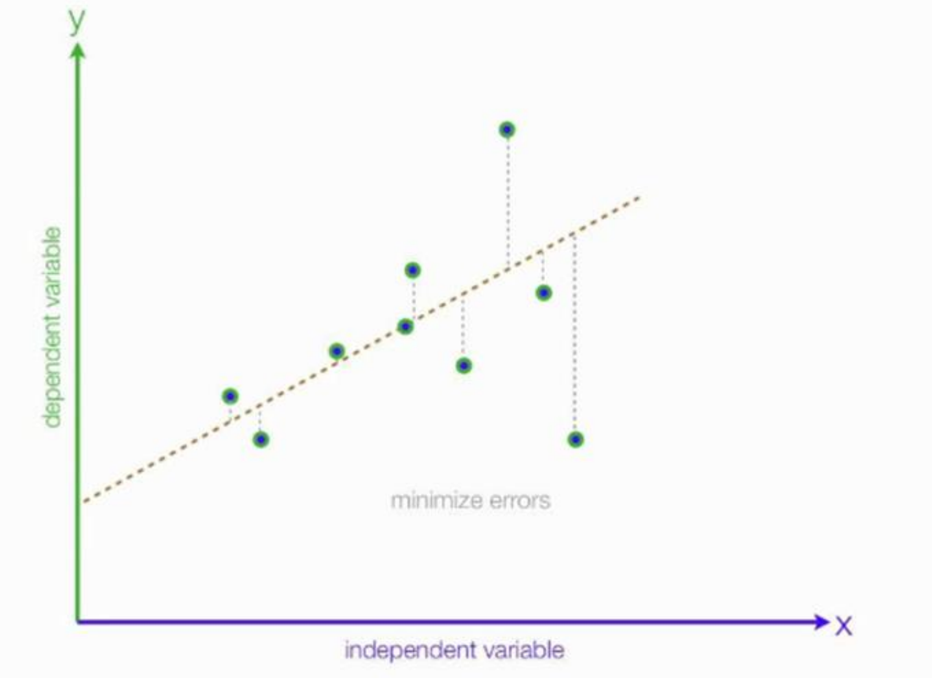
θ_1 $search\ space \in [0.5;\ 10]$

Iteratively explore all options in space and compare them using an evaluation metric (for example Mean Absolute Error)

$optimal\ \theta_1 = 1$

Optimization Problem!

GRADIENT DESCENT
OBJECTIVE FUNCTION



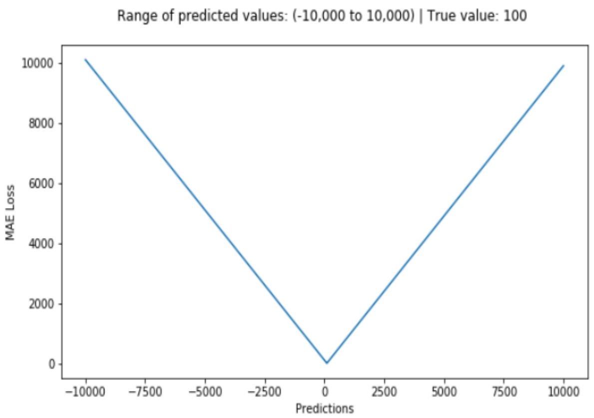
$$MAE = \frac{1}{m}\sum_{i=1}^{m}|h_\theta(x_i) - y_i| \qquad MSE = \frac{1}{m}\sum_{i=1}^{m}|h_\theta(x_i) - y_i|^2$$

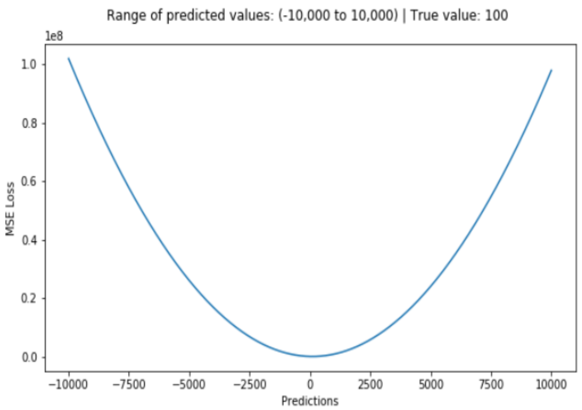
GRADIENT DESCENT

OBJECTIVE FUNCTION

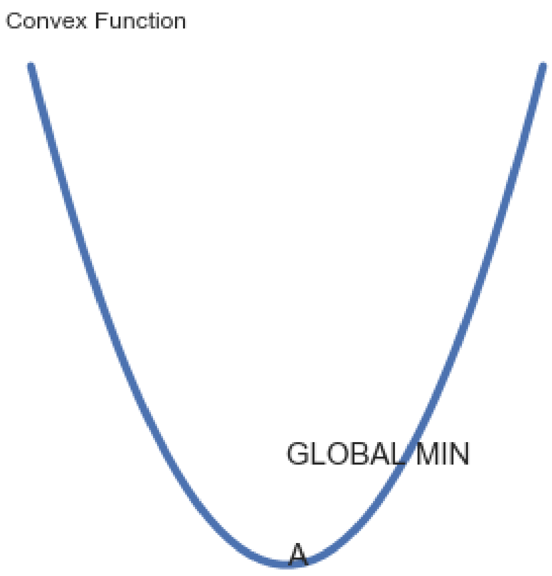
$$MAE = \frac{1}{m}\sum_{i=1}^{m}|h_\theta(x_i) - y_i|$$

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$



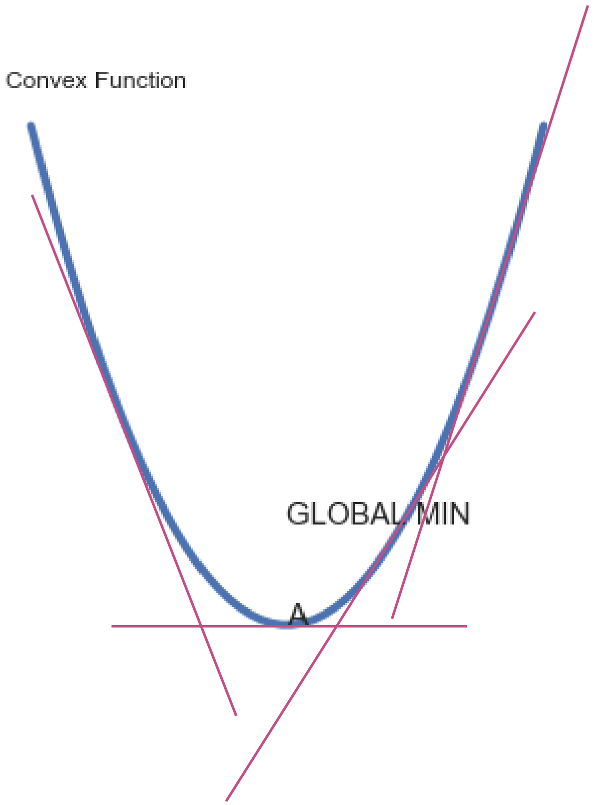


GRADIENT DESCENT
WEIGHTS UPDATING



$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

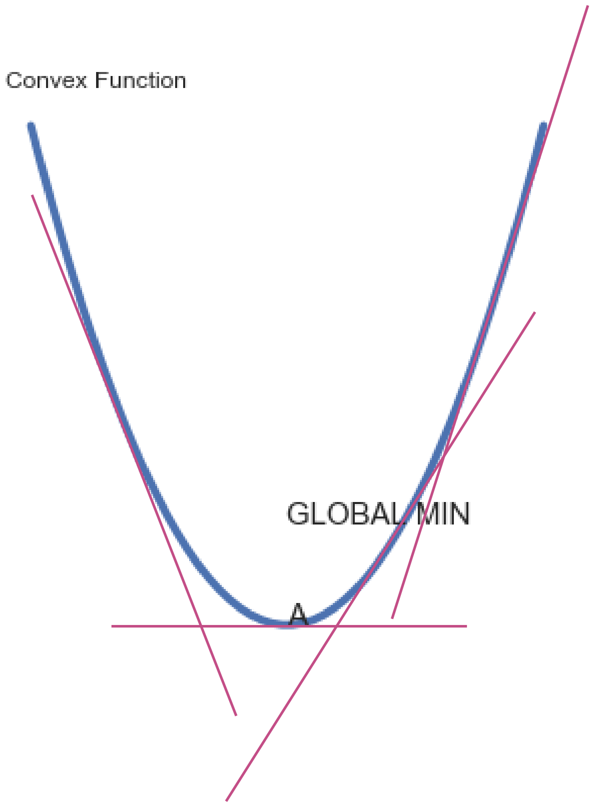
GRADIENT DESCENT
WEIGHTS UPDATING



$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

Minimum when $\frac{\partial J(\theta)}{\partial \theta_i} = 0$

GRADIENT DESCENT
WEIGHTS UPDATING

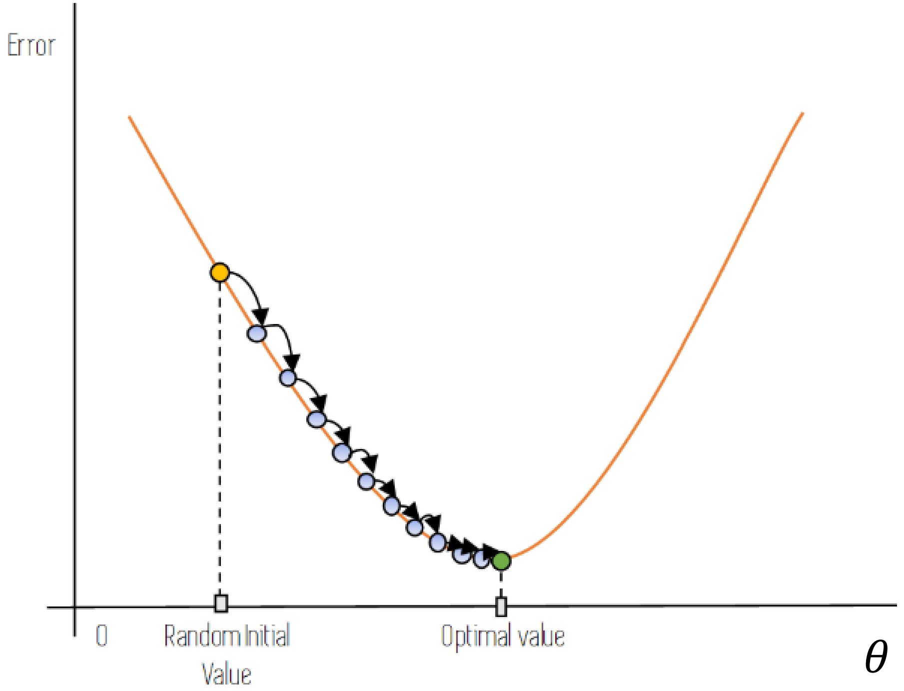


$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

Minimum when $\dfrac{\partial J(\theta)}{\partial \theta_i} = 0$

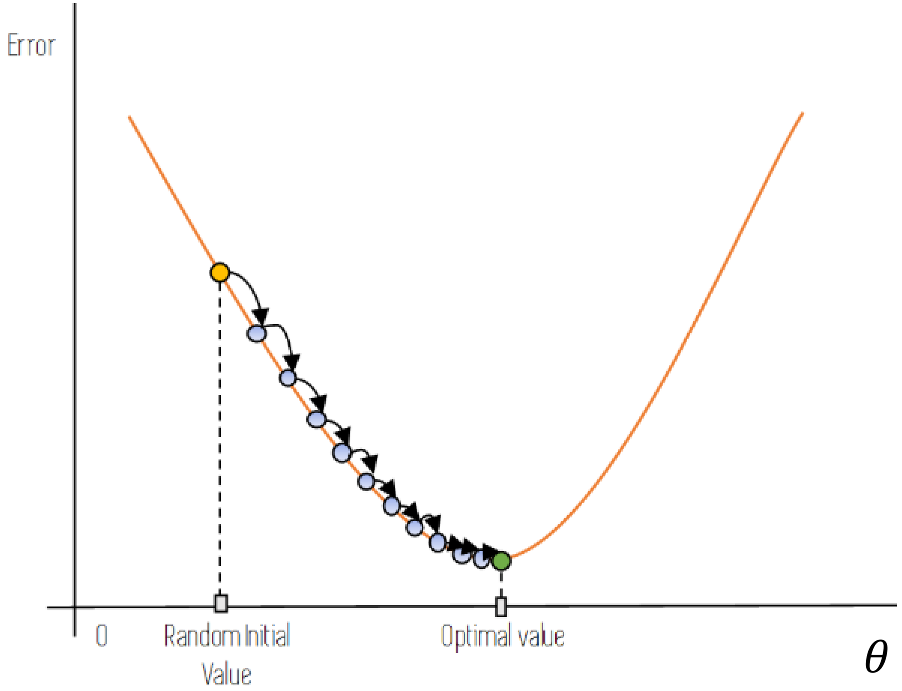
minimize $J(\theta) \equiv$ minimize $\text{const} * J(\theta) \equiv$ minimize $\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$

GRADIENT DESCENT
WEIGHTS UPDATING



Iteratively update θ and compute $J(\theta)$ until hopefully we get convergence on the global minimum

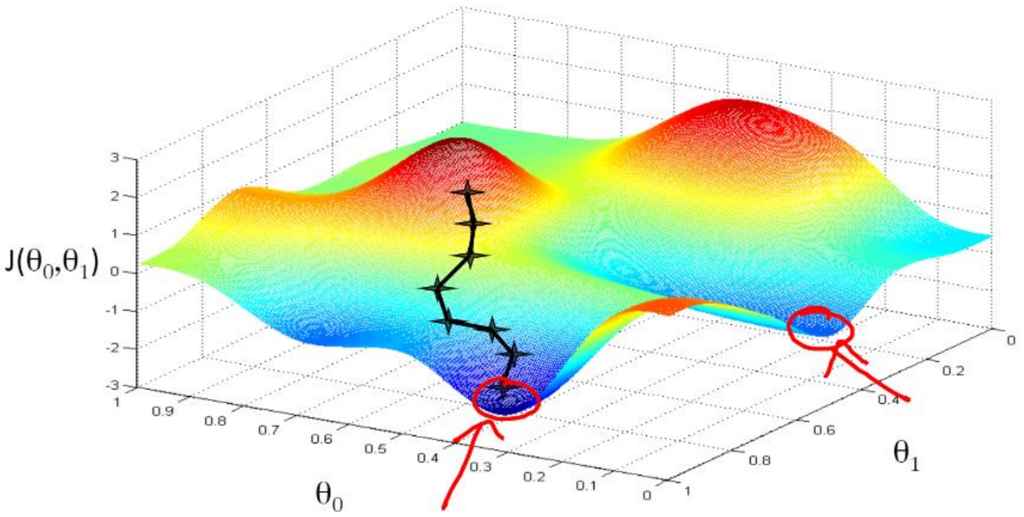
GRADIENT DESCENT
WEIGHTS UPDATING

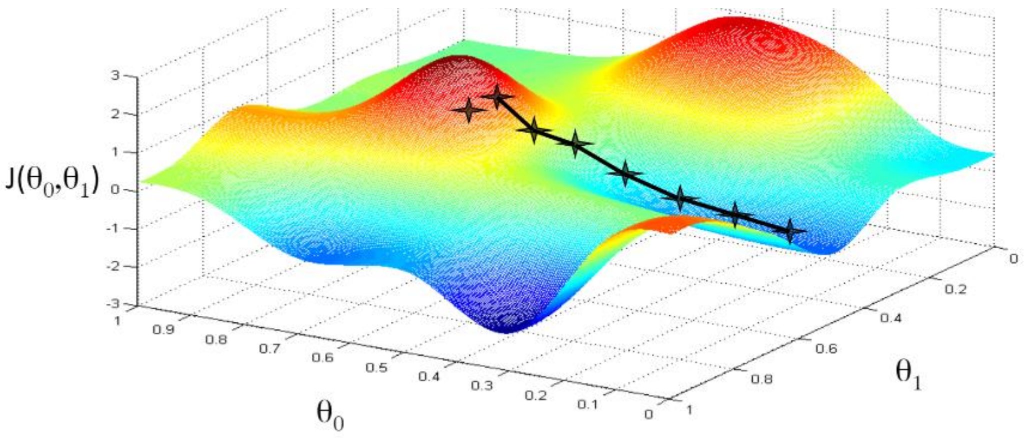


Iteratively update θ and compute $J(\theta)$ until hopefully we get convergence on the global minimum

Gradient descent can converge to a local minimum

GRADIENT DESCENT
WEIGHTS UPDATING



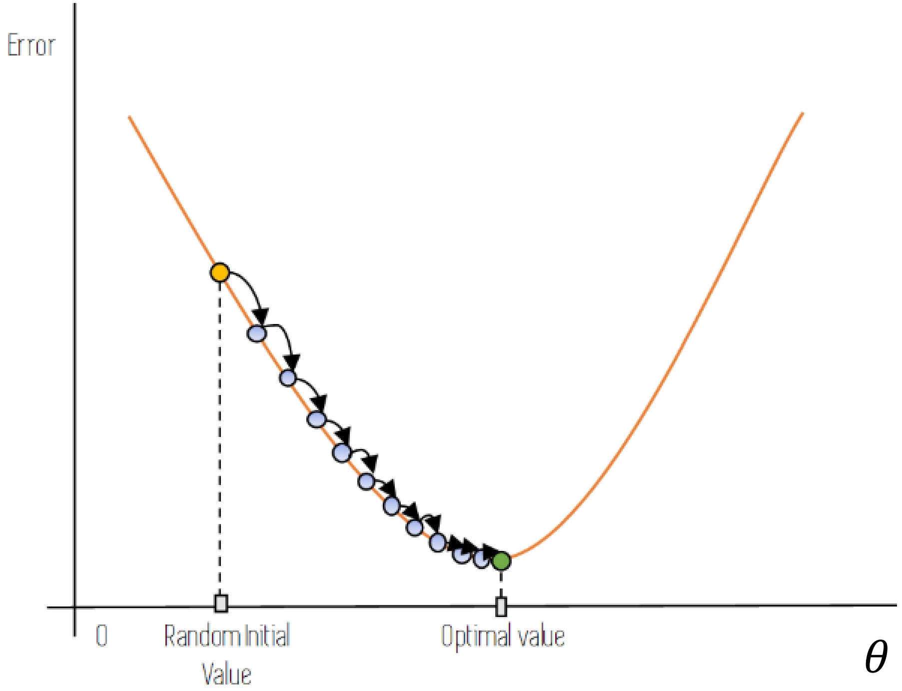


Iteratively update θ and compute $J(\theta)$ until hopefully we get convergence on the global minimum

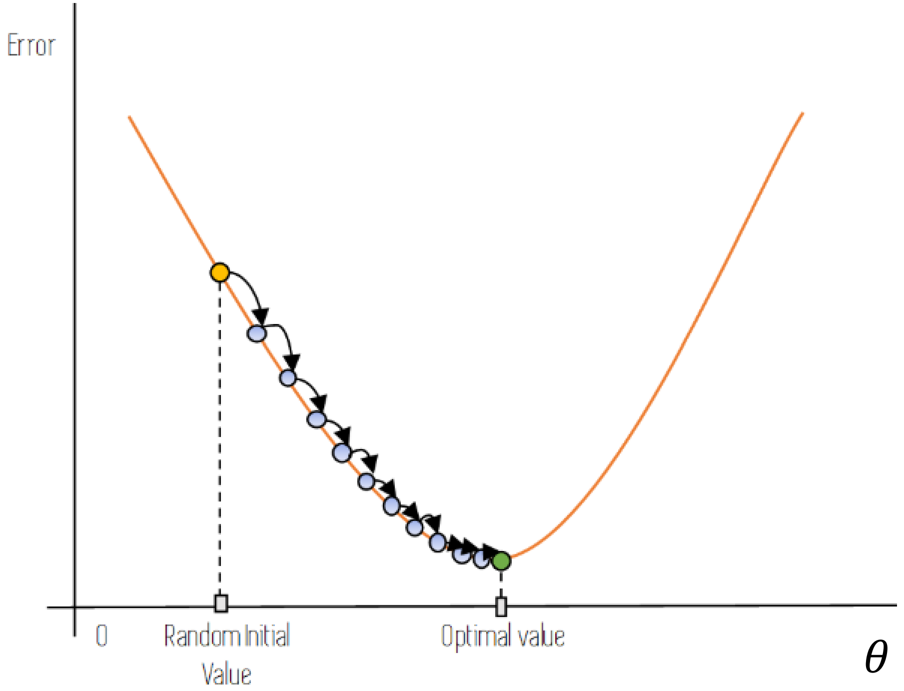
Gradient descent can converge to a local minimum

Weights initialization (and optimizer parameters) must be chosen carefully

GRADIENT DESCENT
WEIGHTS UPDATING

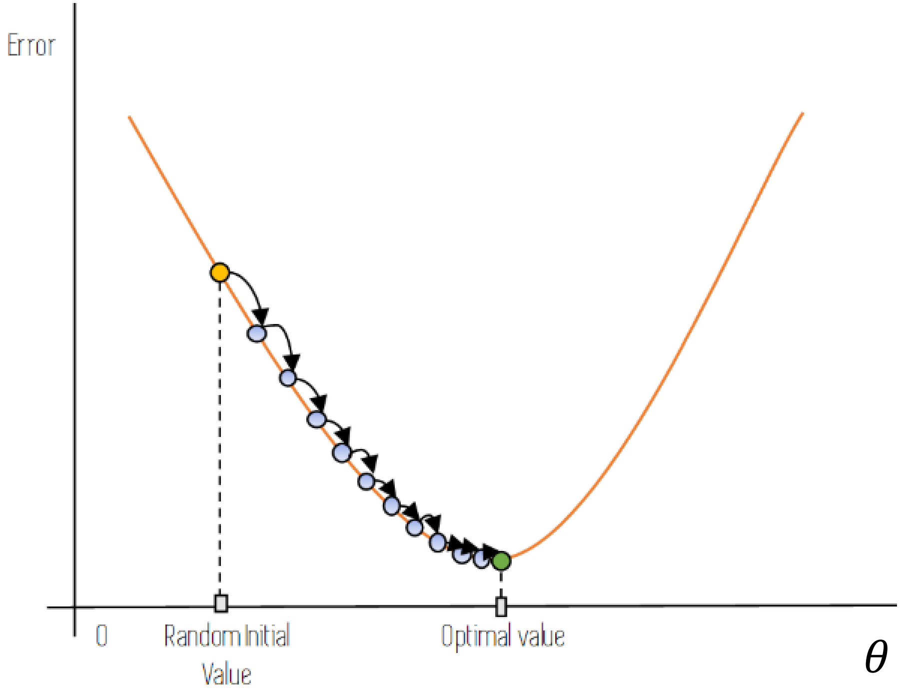


GRADIENT DESCENT
WEIGHTS UPDATING



$$\theta_j \coloneqq \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

GRADIENT DESCENT
WEIGHTS UPDATING

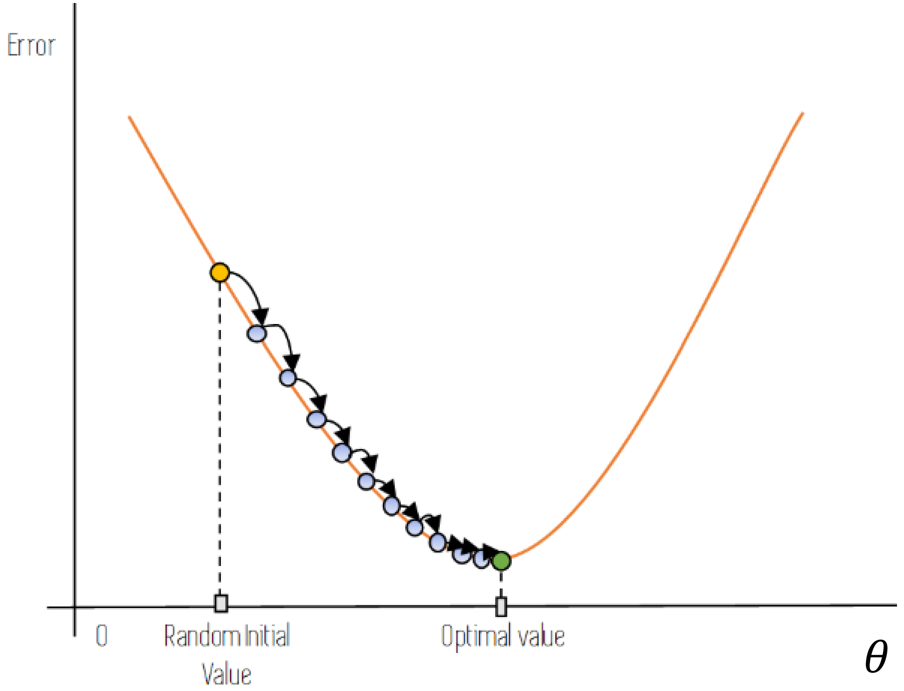


learning rate

$$\boxed{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)}$$

GRADIENT DESCENT

WEIGHTS UPDATING

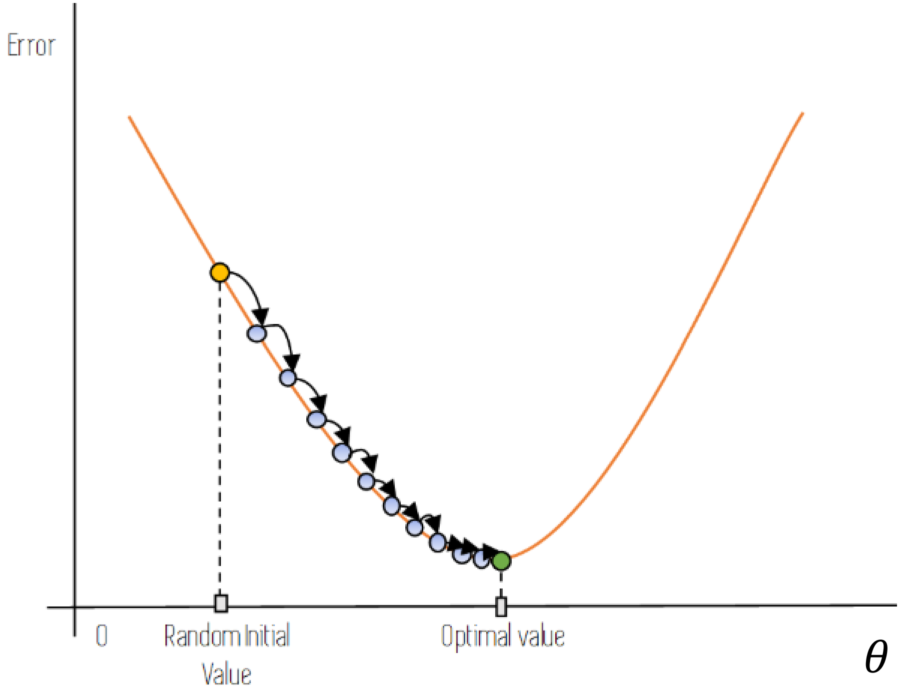


learning rate

$$\boxed{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)}$$

All θ_j must be updated simultaneous, otherwise the calculation of $\frac{\partial}{\partial \theta_j} J(\theta)$ will change within the same optimization step

GRADIENT DESCENT
WEIGHTS UPDATING



learning rate

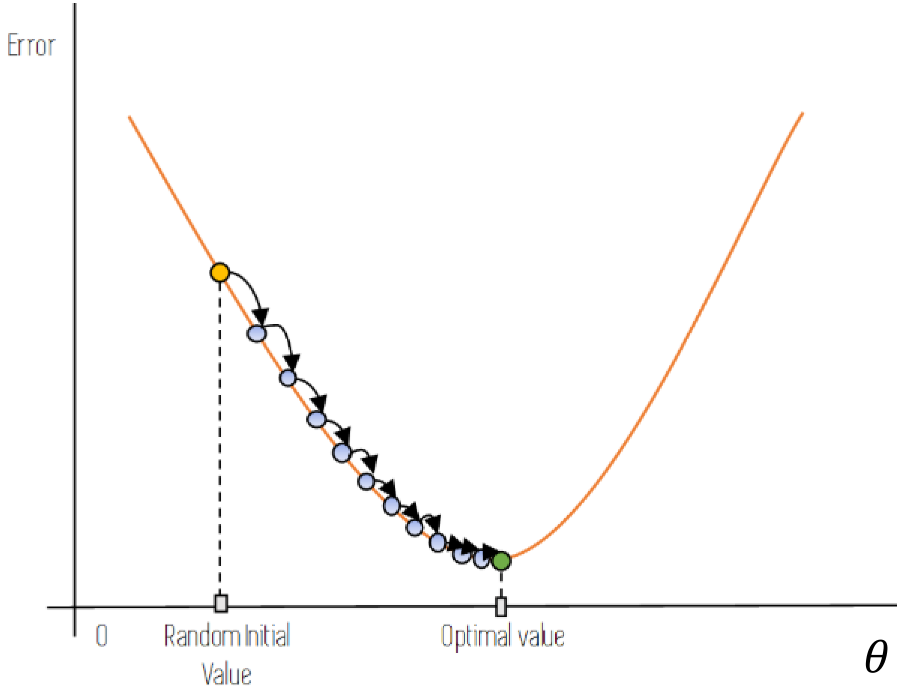
$$\boxed{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)}$$

All θ_j must be updated simultaneous, otherwise the calculation of $\frac{\partial}{\partial \theta_j} J(\theta)$ will change within the same optimization step

Derivative term decreases with closeness to the minimum, adaptively reducing the intensity of the updates

This allows for the learning rate to be fixed

GRADIENT DESCENT
WEIGHTS UPDATING



learning rate

$$\boxed{\theta_j \coloneqq \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)}$$

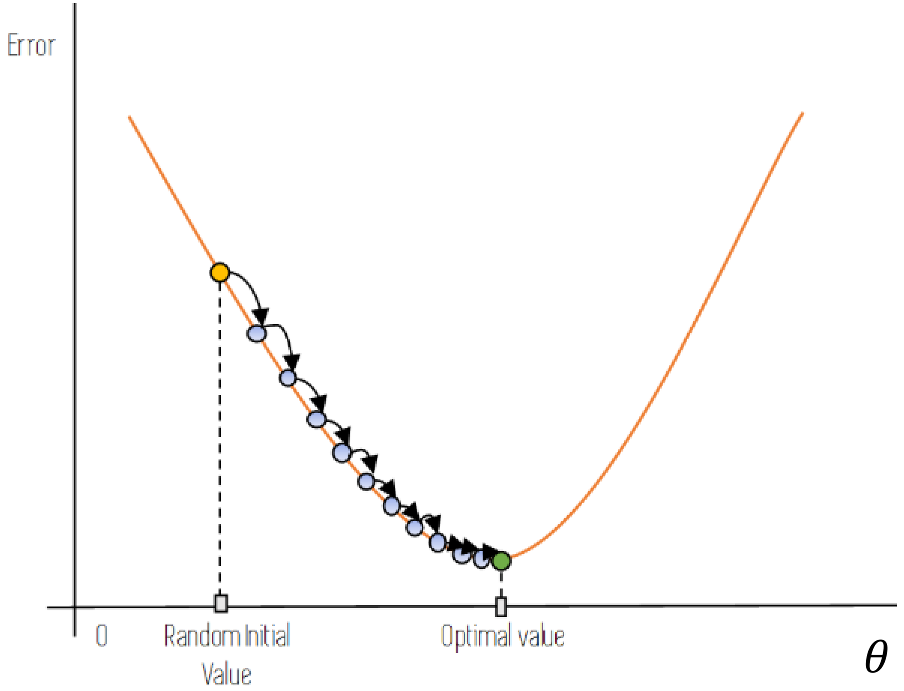
Learning rate choice is very important!

Too small step size and algorithm will be slow

Too big step size and algorithm will overshoot, which can lead to not reaching the minimum or even diverge

GRADIENT DESCENT

WEIGHTS UPDATING



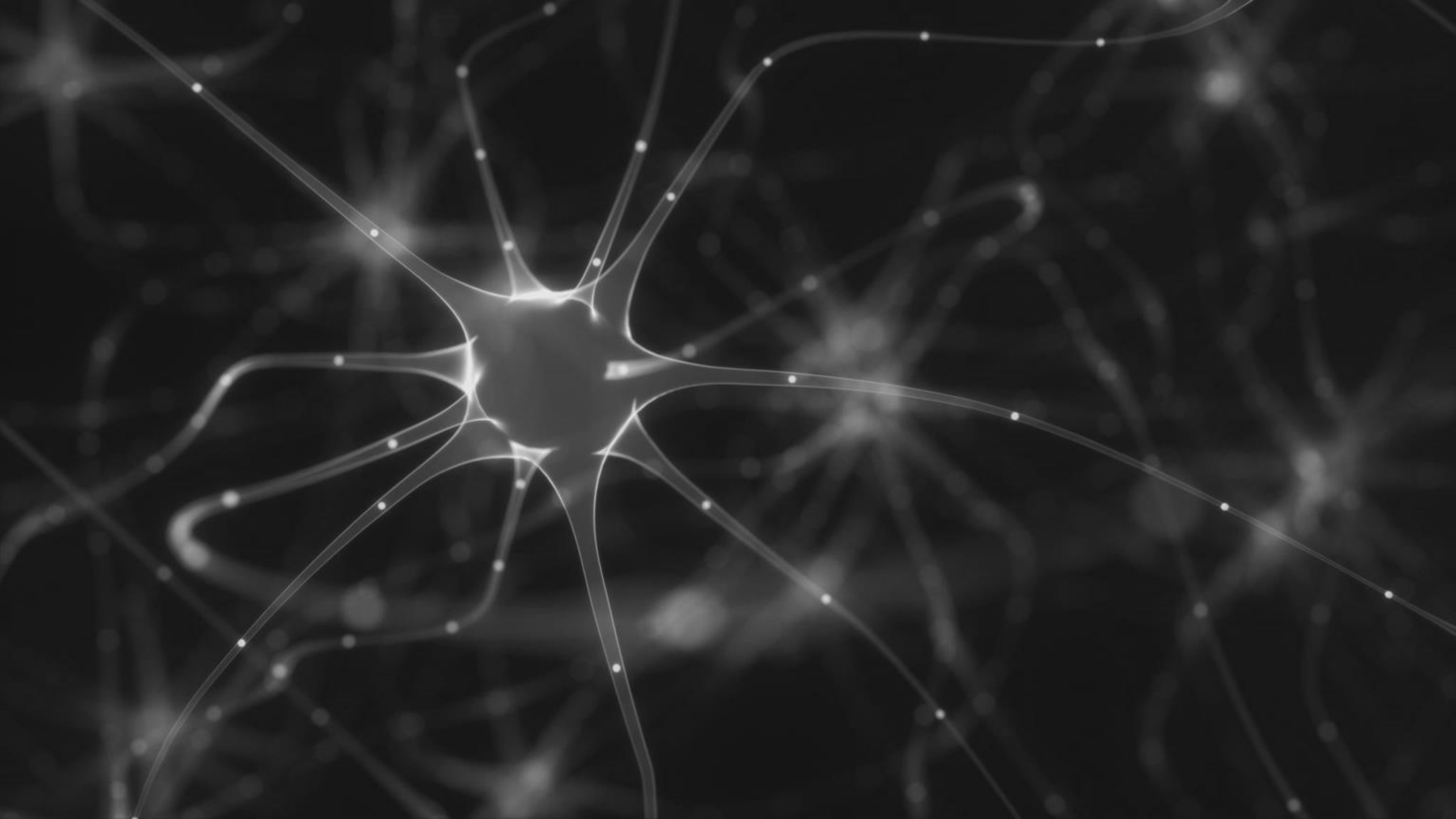
learning rate

$$\boxed{\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)}$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right).\, x_j^{(i)}$$

SO FAR

- Machine learning model is about fitting a function to several examples
- To fit this function, it is necessary to solve an optimization problem regarding the weights of our function
- To solve the optimization problem, a loss/objective function must be defined, as well as an updating strategy
- Gradient descent is a specific strategy that can be used to train machine learning models
- Gradient descent (and other learning strategies) can be susceptible to local minimums
- Good initializations of weights and parameters can be crucial to achieve a good outcome



LOGISTIC REGRESSION

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

$h_\theta(x) = \theta^T x$, with $x_0 = 1$

LOGISTIC REGRESSION

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

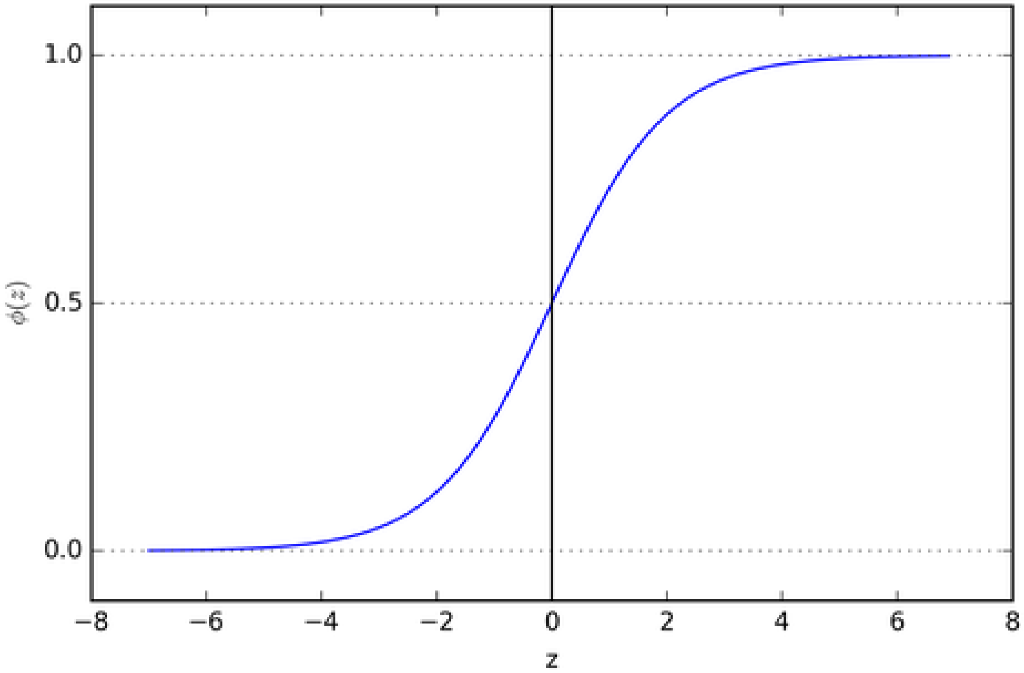
$h_\theta(x) = \theta^T x$, with $x_0 = 1$

For classification problems it is useful that the results of $h_\theta(x)$ be either 0 or 1 (doesn’t belong or belong to class)

Then we need to change our hypothesis.

LOGISTIC REGRESSION
SIGMOID

Sigmoid function: $g(z) = \frac{1}{1+e^{-z}}$



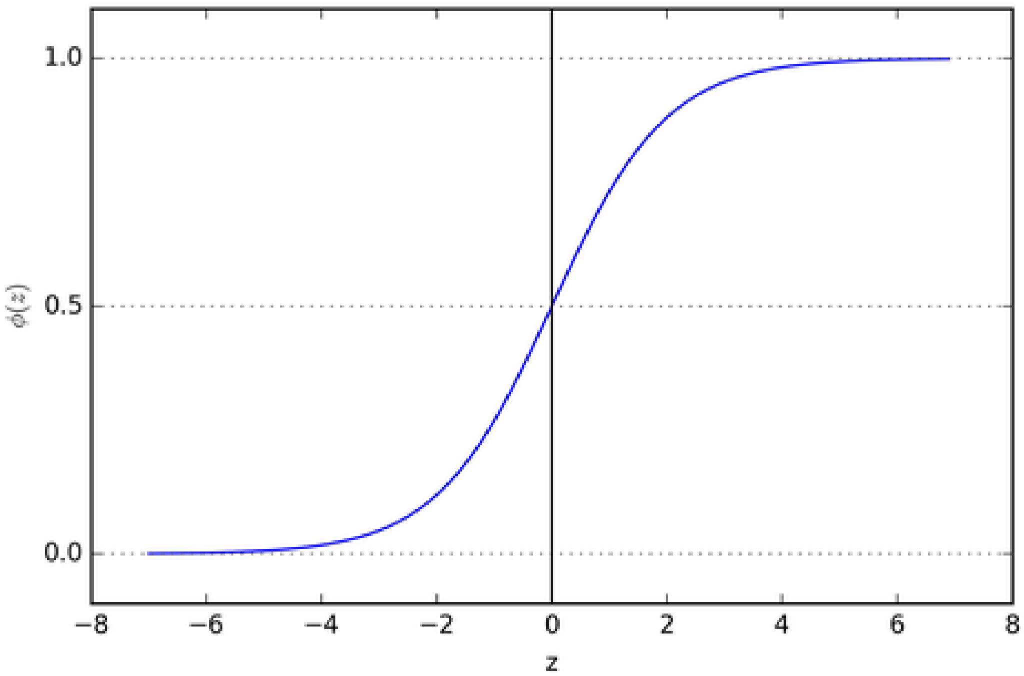
LOGISTIC REGRESSION
SIGMOID

Sigmoid function: $g(z) = \frac{1}{1+e^{-z}}$

Logistic regression: $h_\theta(x) = g(\theta^T x)$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$0 \le h_\theta(x) \le 1$$



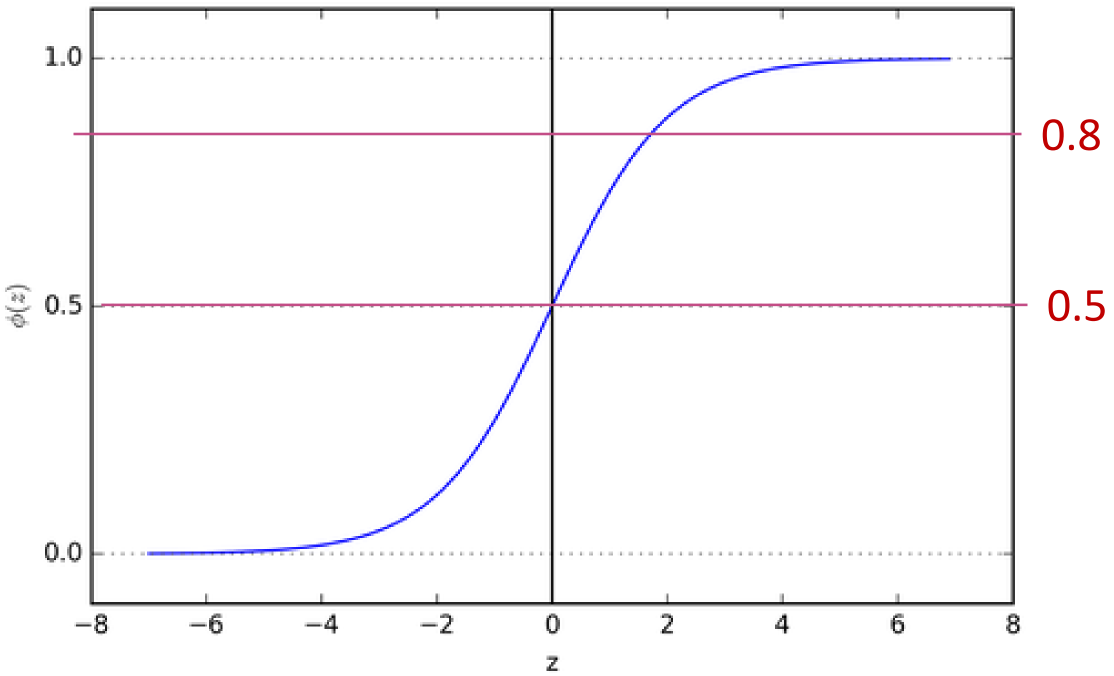
LOGISTIC REGRESSION
SIGMOID

Sigmoid function: $g(z) = \frac{1}{1+e^{-z}}$

Logistic regression: $h_\theta(x) = g(\theta^T x)$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$0 \leq h_\theta(x) \leq 1$$

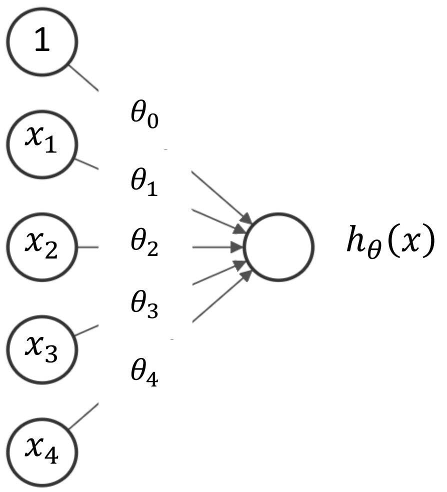


Predict 1 if $h_\theta(x) \geq 0.5$ defines similar probability for both events

Predict 1 if $h_\theta(x) \geq 0.8$ states that we only predict 1 if probability above 80%

LOGISTIC REGRESSION
INTUITION

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

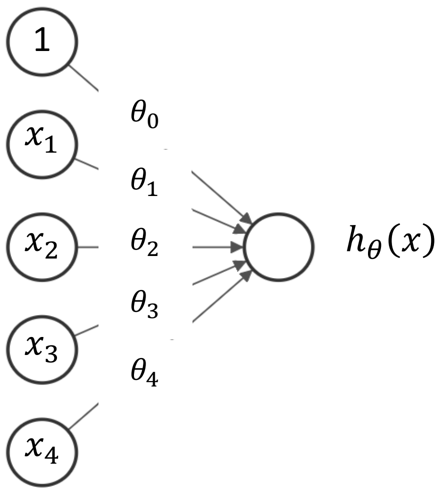


LOGISTIC REGRESSION
INTUITION

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

Also valid for logistic regression!

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$



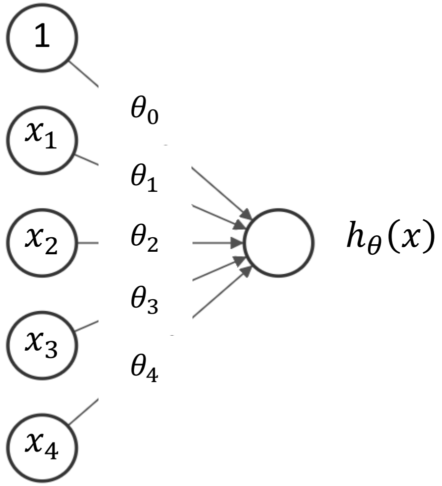
LOGISTIC REGRESSION
INTUITION

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

Also valid for logistic regression!

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

And $h_\theta(x)$ defines then the probability of the neuron to be activated



LOGISTIC REGRESSION
OBJECTIVE FUNCTION

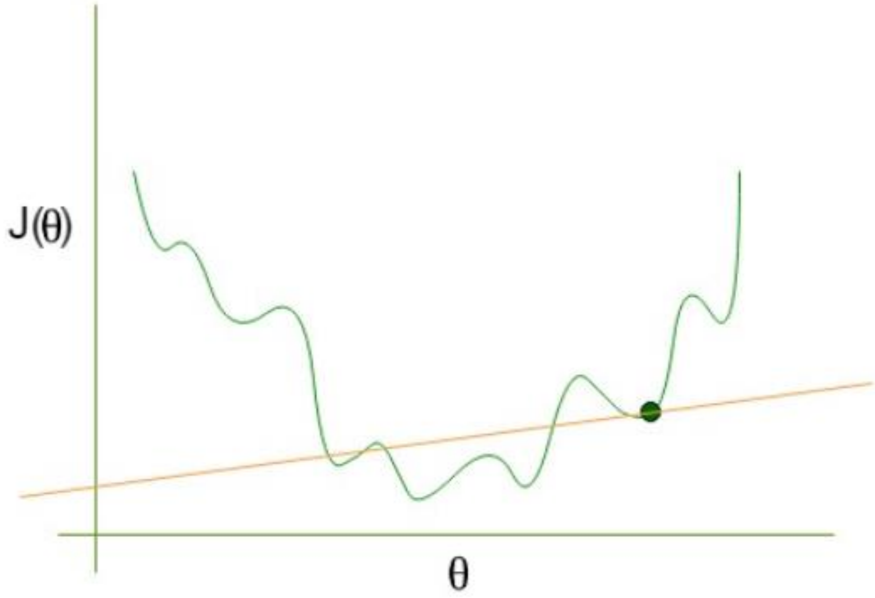
Can logistic regression be optimized in the same fashion as linear regression?

LOGISTIC REGRESSION
OBJECTIVE FUNCTION

Can logistic regression be optimized in the same fashion as linear regression?

Application of linear regression cost function will result in a non-convex function.

What should be the cost function for logistic regression then?

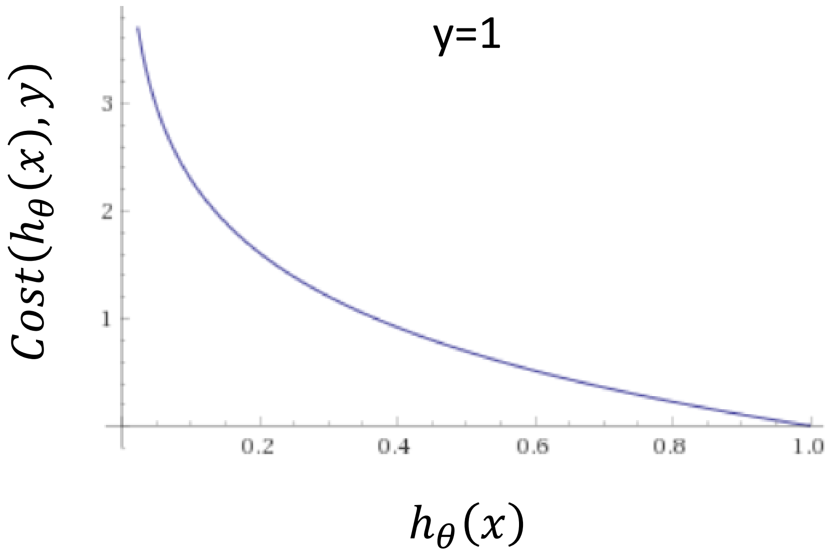


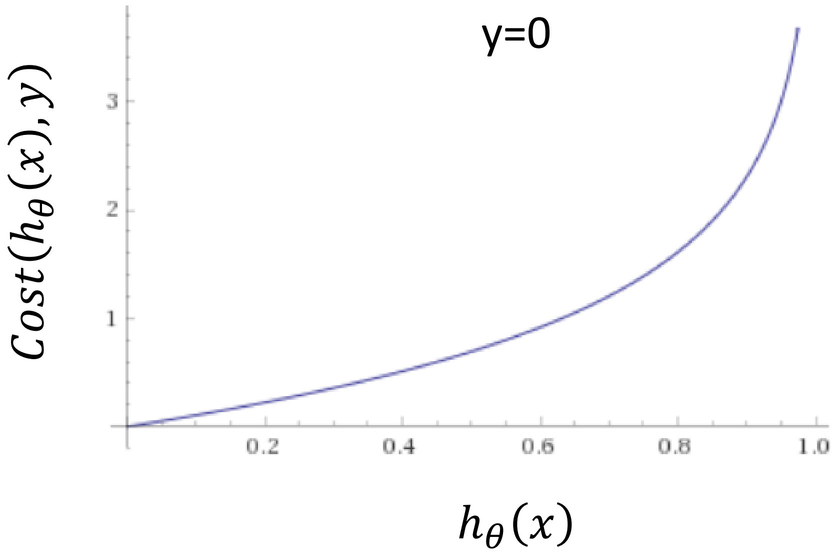
LOGISTIC REGRESSION
OBJECTIVE FUNCTION

$$Cost(h_\theta(x), y) = \begin{cases} -\log\big(h_\theta(x)\big), & if\ y = 1 \\ -\log\big(1 - h_\theta(x)\big), & if\ y = 0 \end{cases}$$

LOGISTIC REGRESSION
OBJECTIVE FUNCTION

$$Cost(h_\theta(x), y) = \begin{cases} -\log\big(h_\theta(x)\big), & if\ y = 1 \\ -\log\big(1 - h_\theta(x)\big), & if\ y = 0 \end{cases}$$





LOGISTIC REGRESSION
OBJECTIVE FUNCTION

$$Cost(h_\theta(x), y) = \begin{cases} -\log\big(h_\theta(x)\big), & if\ y = 1 \\ -\log\big(1 - h_\theta(x)\big), & if\ y = 0 \end{cases}$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\big(h_\theta\big(x^{(i)}\big), y^{(i)}\big)$$

LOGISTIC REGRESSION
OBJECTIVE FUNCTION

$$Cost(h_\theta(x), y) = \begin{cases} -\log\big(h_\theta(x)\big), & if\ y = 1 \\ -\log\big(1 - h_\theta(x)\big), & if\ y = 0 \end{cases}$$

Rewritable as

$$Cost(h_\theta(x), y) = -\mathrm{y}.\log\big(h_\theta(x)\big) - (1 - y).\log\big(1 - h_\theta(x)\big)$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\big(h_\theta\big(x^{(i)}\big), y^{(i)}\big) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\Big(h_\theta\big(x^{(i)}\big)\Big) + \big(1 - y^{(i)}\big).\log\Big(1 - h_\theta\big(x^{(i)}\big)\Big)\right]$$

LOGISTIC REGRESSION

WEIGHTS UPDATE

$$Cost(h_\theta(x), y) = \begin{cases} -\log\big(h_\theta(x)\big), & if\ y = 1 \\ -\log\big(1 - h_\theta(x)\big), & if\ y = 0 \end{cases}$$

Rewritable as

$$\boxed{Cost(h_\theta(x), y) = -\mathrm{y}.\log\big(h_\theta(x)\big) - (1 - y).\log\big(1 - h_\theta(x)\big)}$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\big(h_\theta\big(x^{(i)}\big), y^{(i)}\big) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log\Big(h_\theta\big(x^{(i)}\big)\Big) + \big(1 - y^{(i)}\big).\log\Big(1 - h_\theta\big(x^{(i)}\big)\Big)\right]$$

To apply gradient descent, we just need to calculate the derivative of the new cost function, and plug it into the weights updating formula

$$\theta_j \coloneqq \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

LOGISTIC REGRESSION

WEIGHTS UPDATE

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)), & if\ y = 1 \\ -\log(1 - h_\theta(x)), & if\ y = 0 \end{cases}$$

Rewritable as

$$Cost(h_\theta(x), y) = -\mathrm{y}.\log(h_\theta(x)) - (1 - y).\log(1 - h_\theta(x))$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right).\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right]$$

To apply gradient descent, we just need to calculate the derivative of the new cost function, and plug it into the weights updating formula

$$\theta_j \coloneqq \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta) = \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right).x_j^{(i)}$$

LOGISTIC REGRESSION

WEIGHTS UPDATE

$$Cost(h_\theta(x), y) = \begin{cases} -\log\left(h_\theta(x)\right), & if\ y = 1 \\ -\log\left(1 - h_\theta(x)\right), & if\ y = 0 \end{cases}$$

Rewritable as

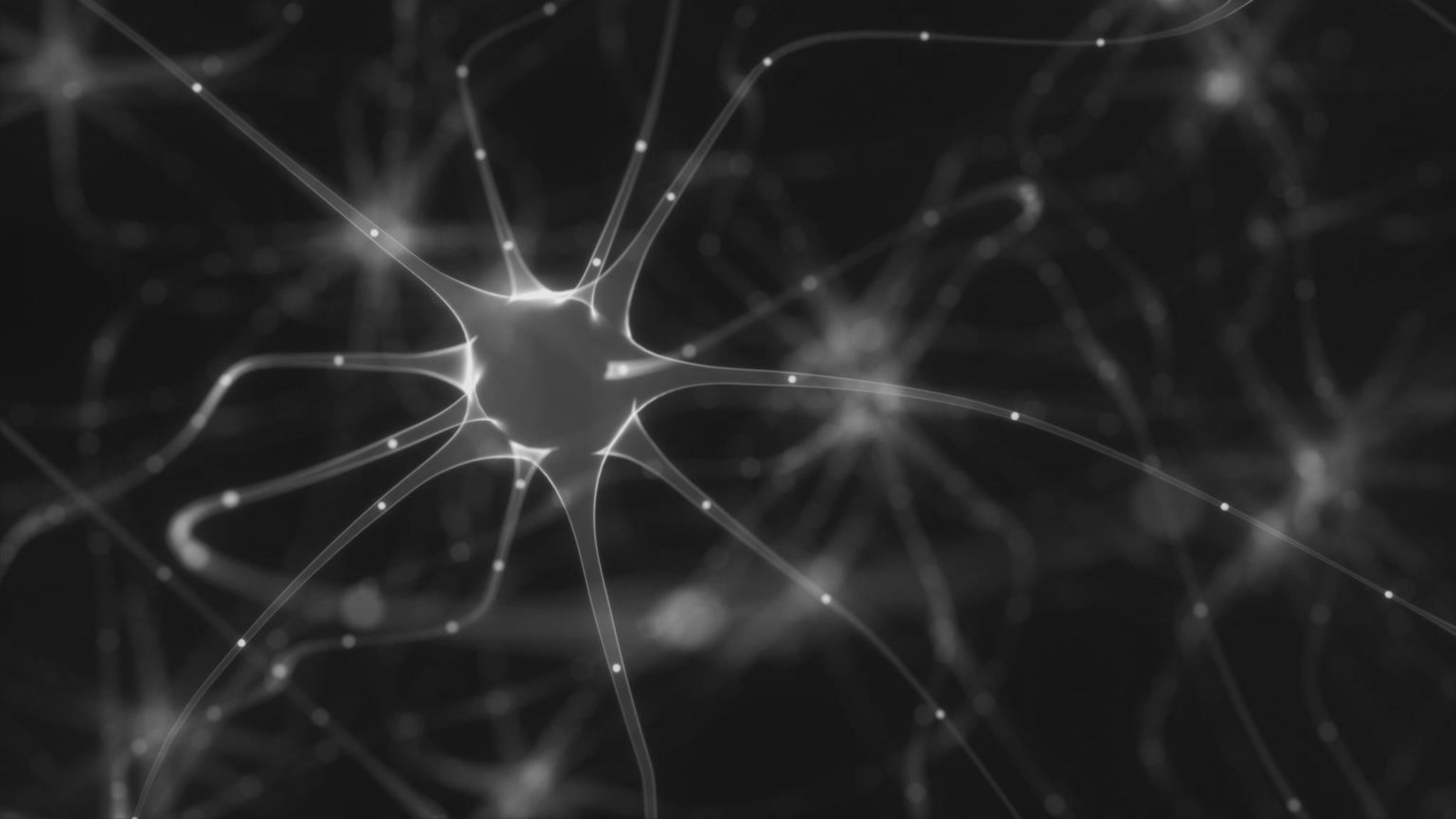
$$\boxed{Cost(h_\theta(x), y) = -\mathrm{y}.\log\left(h_\theta(x)\right) - (1 - y).\log\left(1 - h_\theta(x)\right)}$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right).\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right]$$

To apply gradient descent, we just need to calculate the derivative of the new cost function, and plug it into the weights updating formula

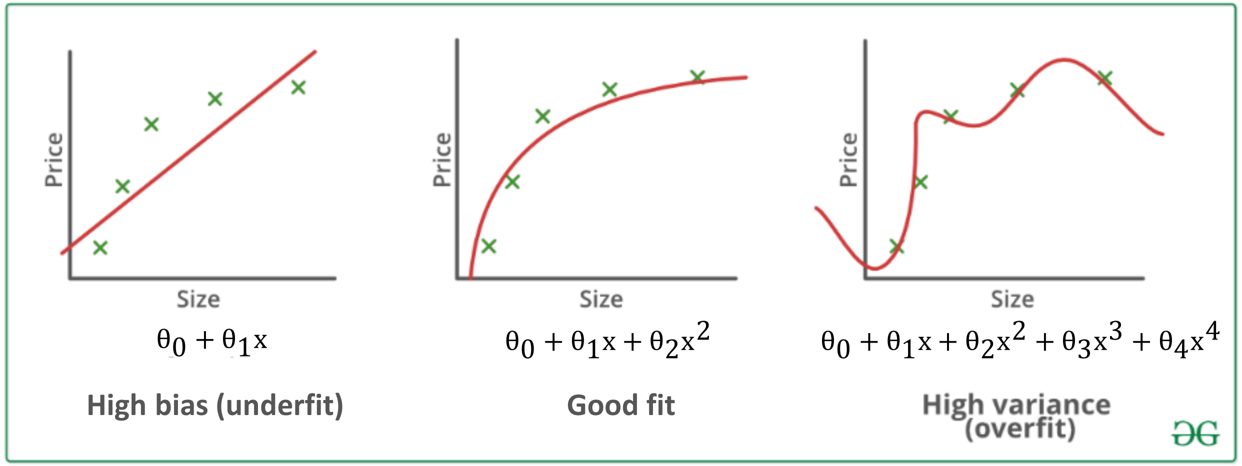
$$\theta_j \coloneqq \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) = \boxed{\theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right).x_j^{(i)}}$$

Same as for linear regression!
Only $h_\theta(x)$ is different!

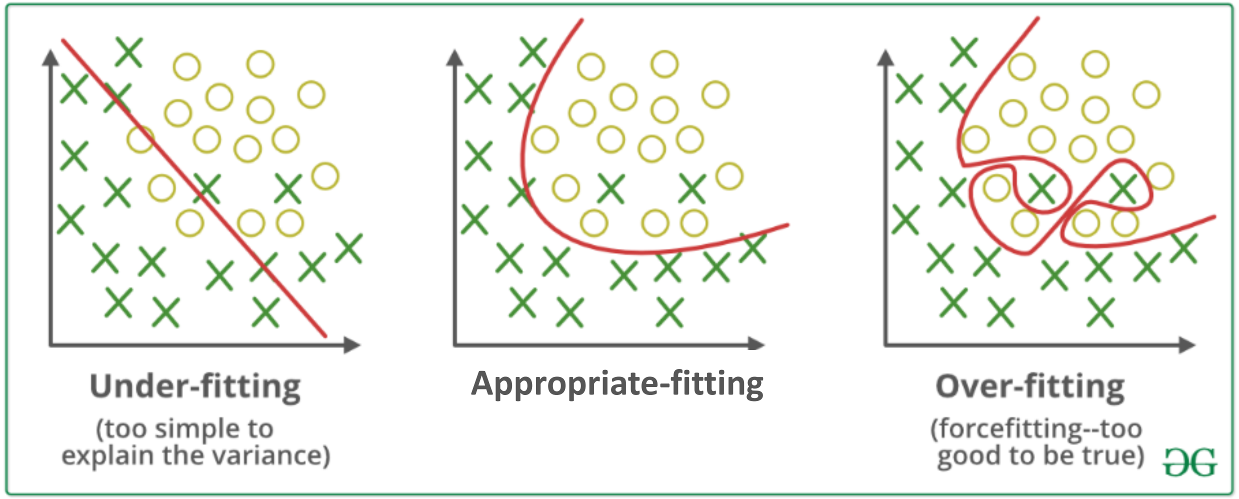


UNDERFIT/OVERFIT

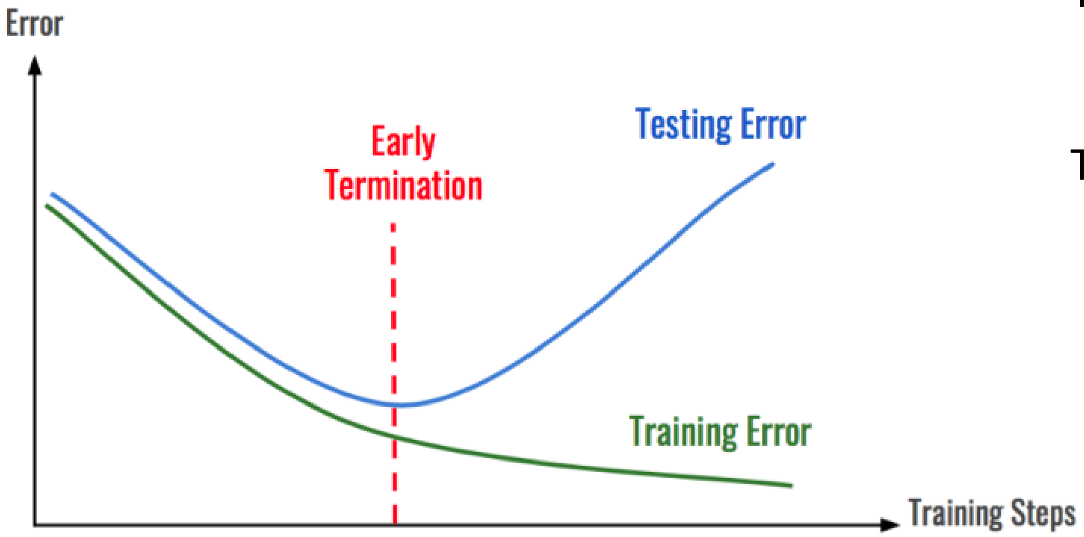
Linear Regression



Logistic Regression



UNDERFIT/OVERFIT



When the **testing error** starts to **increase**, it's time to stop!

Too few iterations => model can still be underfit

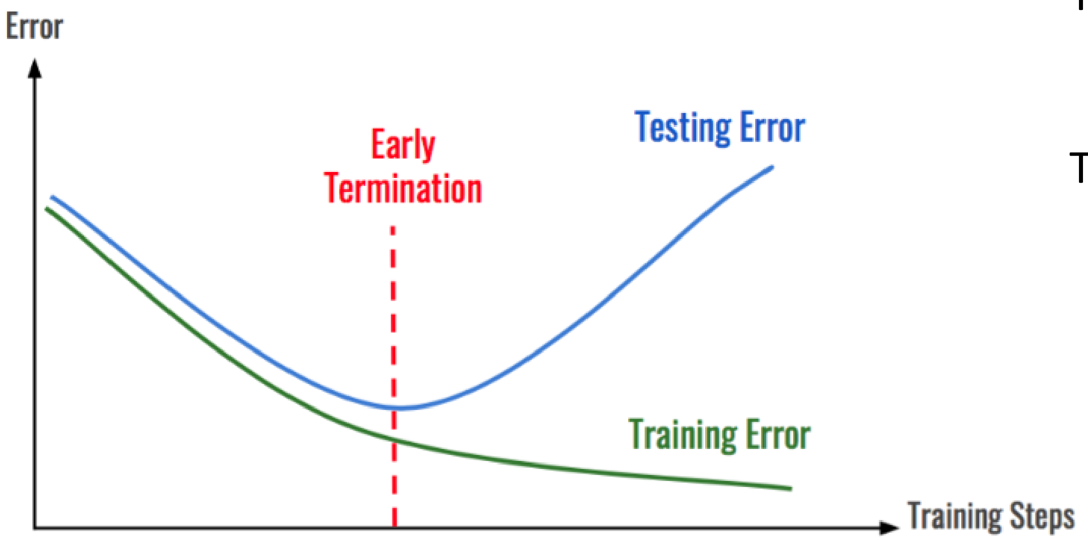
There is still potential to improve on the validation set

Too many iteration => model can be overfit

Training error continues to decrease, as model memorizes samples

Testing error increases has model loses capability of generalization

UNDERFIT/OVERFIT



When the **testing error** starts to **increase**, it's time to stop!

Too few iterations => model can still be underfit

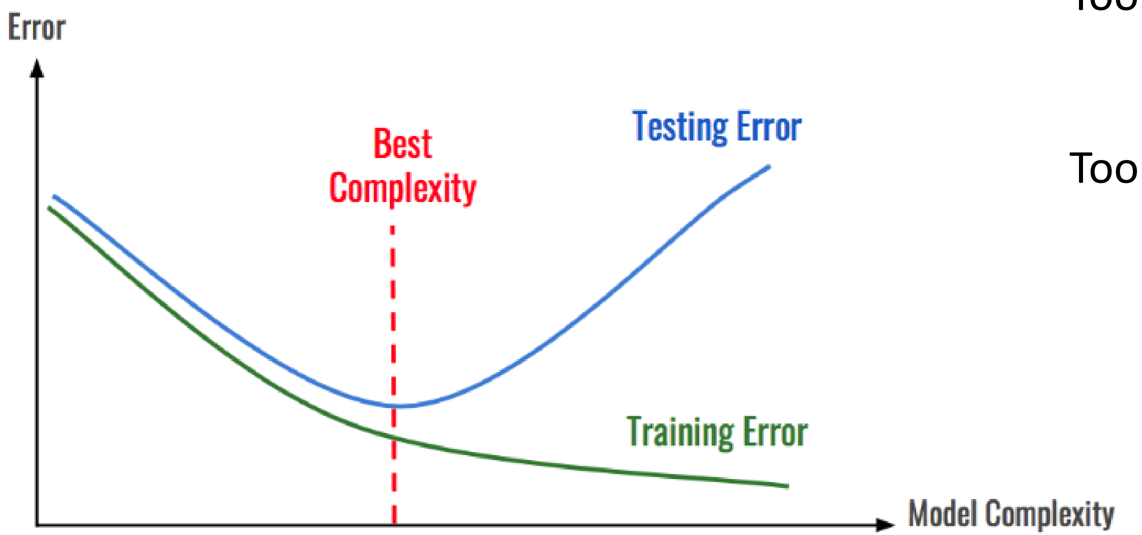
- There is still potential to improve on the validation set

Too many iterations => model can overfit

- Training error continues to decrease, as model memorizes samples
- Testing error increases has model loses capability of generalization

Good number of iterations depends on hyperparameters and model complexity!

UNDERFIT/OVERFIT



On the left, the model is too simple. On the right it overfits.

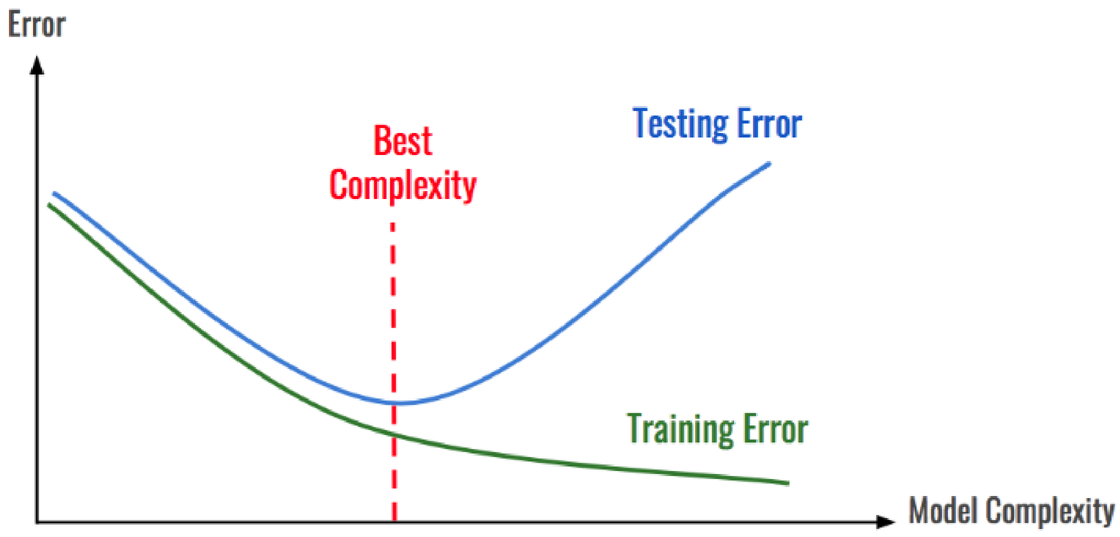
Too simple a model => model can still be underfit

- There is still potential to improve on the validation set

Too complex of a model => model can overfit

- Training error continues to decrease, as model memorizes samples
- Testing error increases has model loses capability of generalization

UNDERFIT/OVERFIT



On the left, the model is too simple. On the right it overfits.

simple a model => model can still be underfit

There is still potential to improve on the validation set

complex of a model => model can overfit

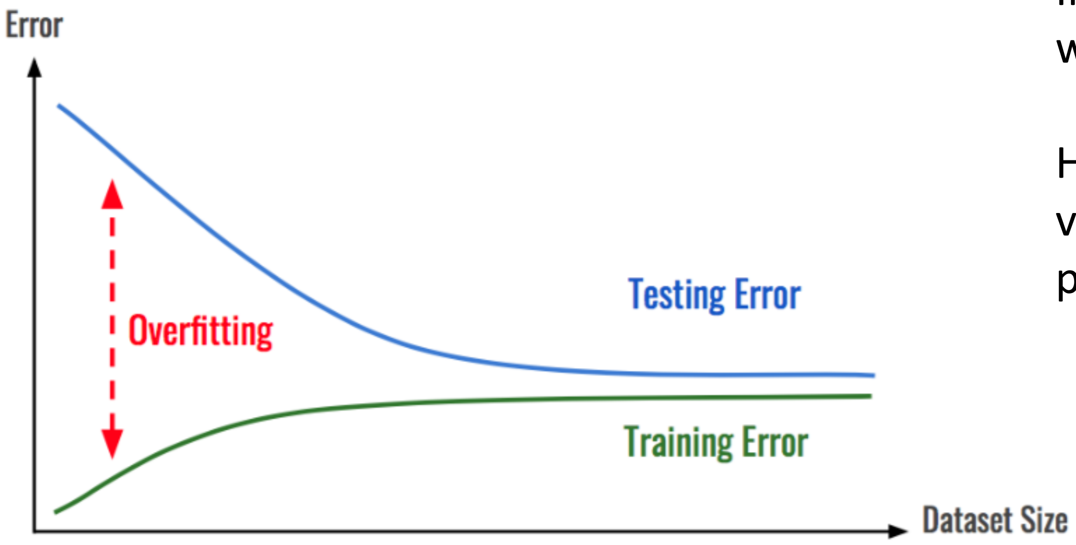
Training error continues to decrease, as model memorizes samples

Testing error increases has model loses capability of generalization

Choice of model complexity depends greatly on complexity of the data.

Typically one must try different architectures/complexities and adapt the model complexity in order to avoid underfit/overfit

UNDERFIT/OVERFIT



The **more data** you get, the **less** likely the model is to **overfit.**

Increasing the dataset size tends to reduce overfitting as there will exist more data to learn on and conditioning the learning

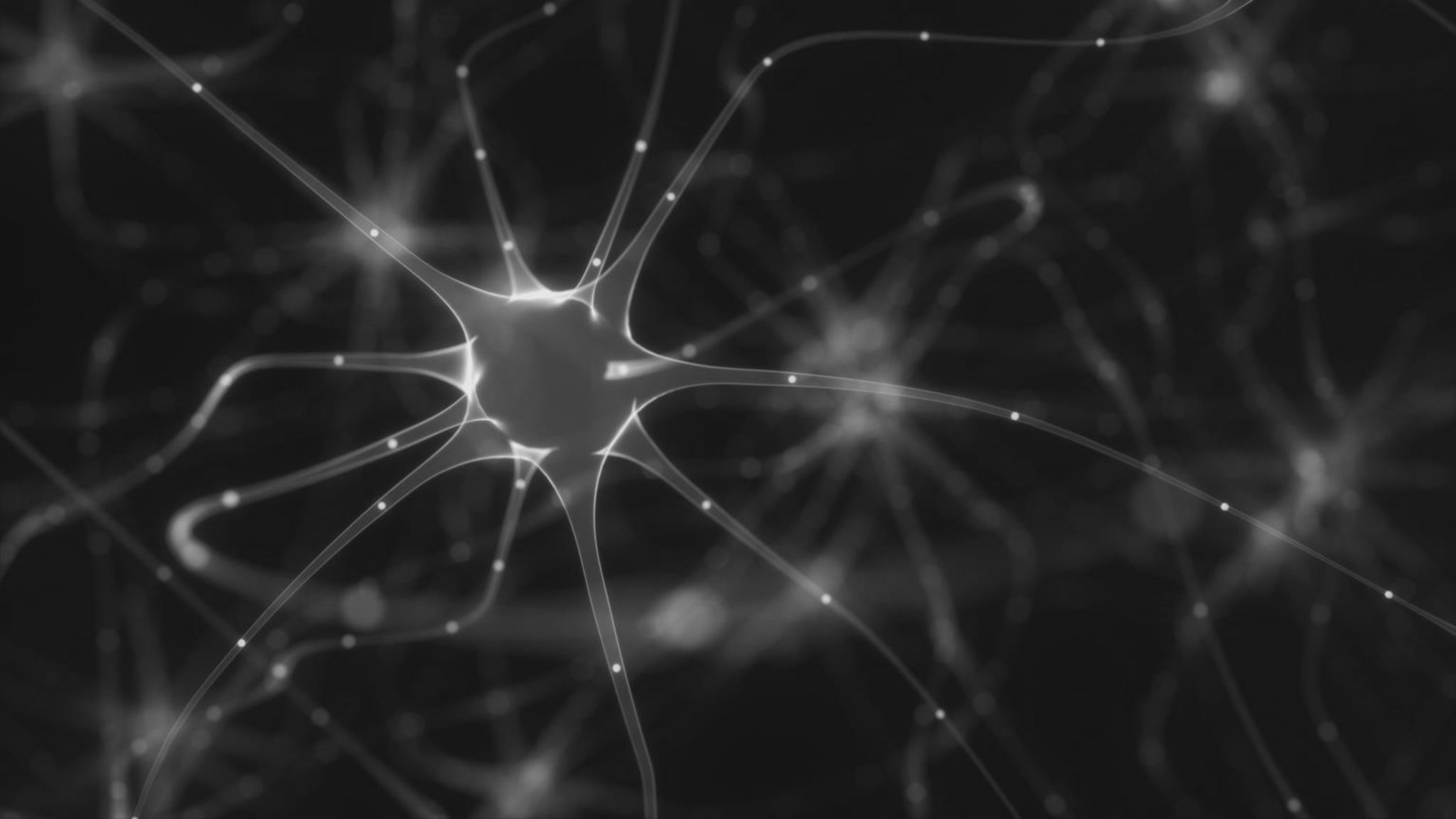
However, if increase in dataset also increases greatly the variance of patterns existing in the data, we can easily end in a problem that is to complex for our current model complexity.

To address overfitting:

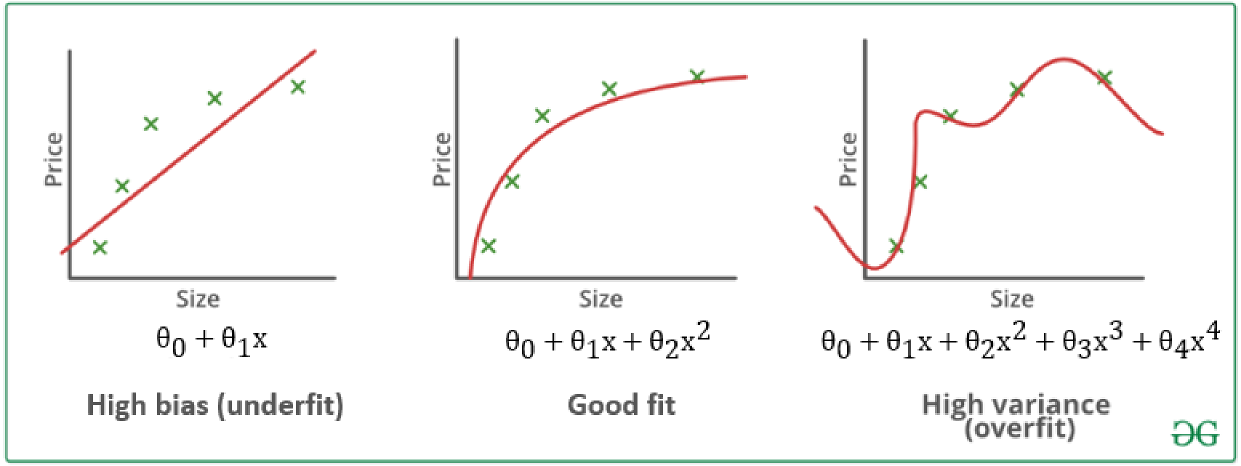
- Get more data
- Reduce the number of features
- Regularization
- Reduce the number of iterations and/or learning rate

To address underfit:

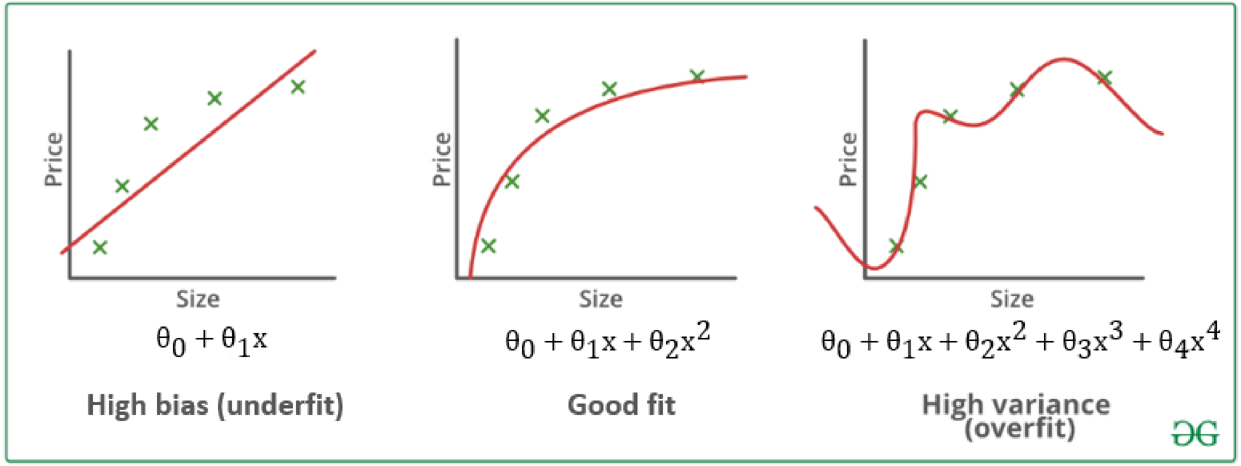
- Increase model complexity
- Get/engineer more features
- Decrease strength of regularization
- Increase the number of iterations (can be time-expensive)



REGULARIZATION
INTUITION

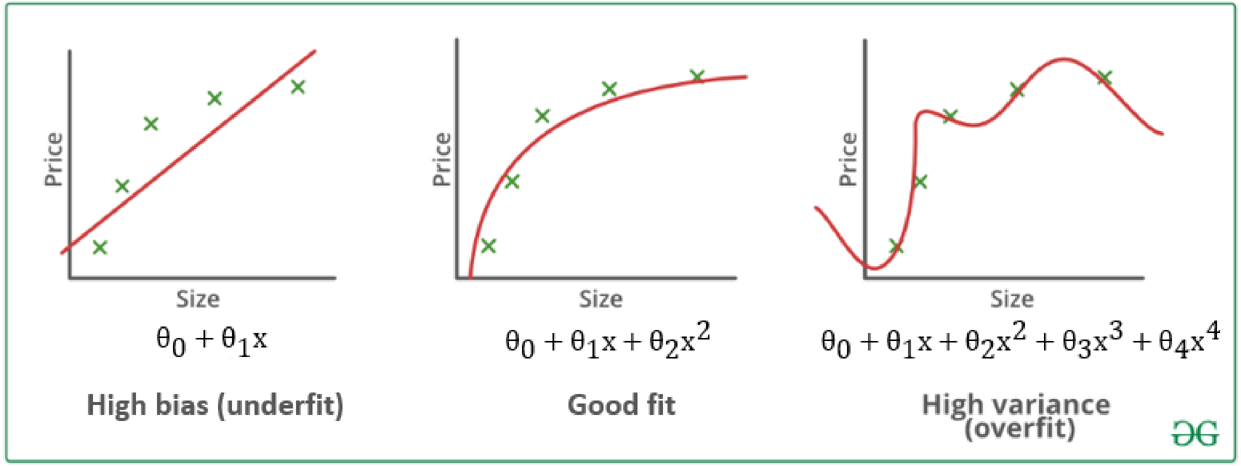


REGULARIZATION
INTUITION



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4\, x^4 \quad ==> \quad h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

REGULARIZATION
INTUITION



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4\, x^4 \qquad ==> \qquad h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

Drive θ_3 and θ_4 to be ≈ 0

REGULARIZATION

INTUITION

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

REGULARIZATION
INTUITION

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \; + bigM_1.\theta_3 + bigM_2.\theta_4$$

REGULARIZATION

INTUITION

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \ + \underbrace{bigM_1 . \theta_3}_{\theta_3 \approx 0} + \underbrace{bigM_2 . \theta_4}_{\theta_4 \approx 0}$$

REGULARIZATION
GENERALIZATION

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

Regularization parameter

By tuning λ it is possible to control the strength of the regularization and thus control the underfitting/overfitting

Notice that θ_0 is not regularized, and so, if λ is very high and all $\theta_1 \dots \theta_n$ are very small, then $h_\theta(x_i) \approx \theta_0$ (model underfit)

On the other hand, if $\lambda \approx 0$ we go back to the original situation and can have overfit

Lasso (l1) regularization: $\sum_{j=1}^{n} |\theta_j|$

Ridge (l2) regularization: $\sum_{j=1}^{n} \theta_j^2$

Ridge coefficients can get very small, but never exactly zero

On the other hand, Lasso coefficients can be zero, and consequently perform one kind of feature selection

However, Ridge tends to be computationally less intensive than Lasso

Updated gradients (with L2):

$$\begin{cases} \theta_0 \coloneqq \theta_0 - \alpha\left[\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}(h_\theta\left(x^{(i)}\right) - y^{(i)}).\, x_0^{(i)}\right], \text{for } j = 0 \\ \theta_j \coloneqq \theta_j - \alpha\left[\dfrac{1}{m}\displaystyle\sum_{i=1}^{m}(h_\theta\left(x^{(i)}\right) - y^{(i)}).\, x_j^{(i)} + \dfrac{\lambda}{m}\theta_j\right], \text{for } j \neq 0 \end{cases}$$

Update is the same for linear and logistic regression. Only $(h_\theta\left(x^{(i)}\right)$ is different

REGULARIZATION

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \; + \lambda \sum_{j=1}^{n} \theta_j^2$$

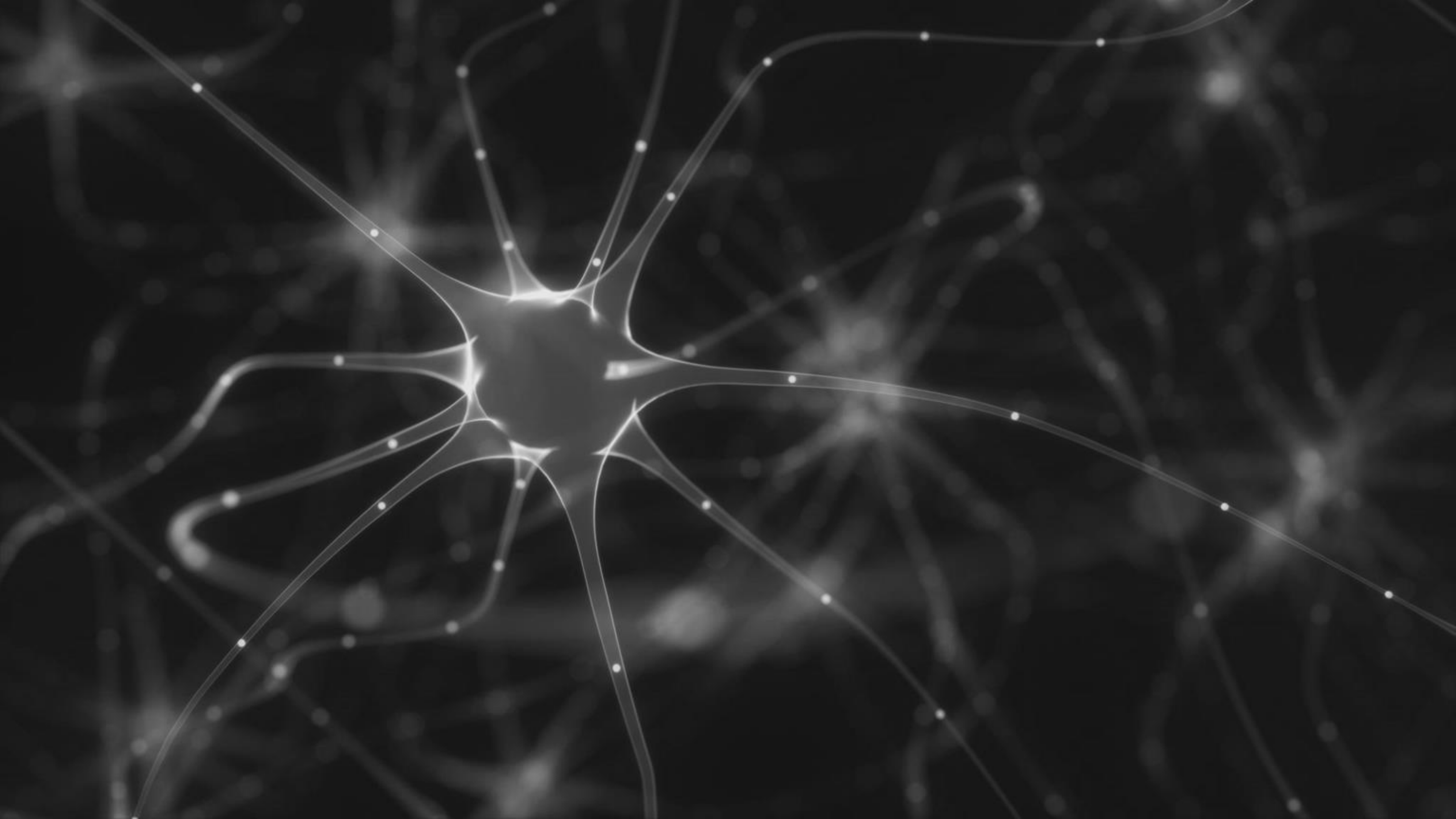
How to choose lambda, such that we don't underfit/overfit?

REGULARIZATION

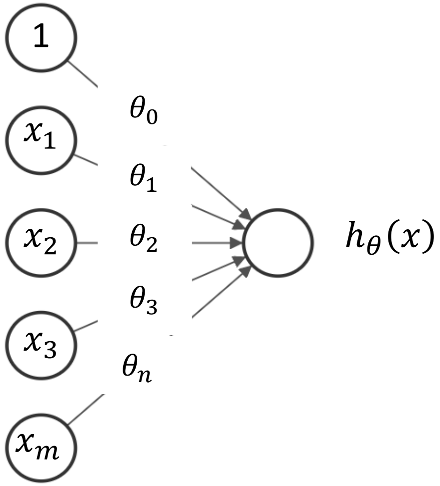
$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \; + \lambda \sum_{j=1}^{n} \theta_j^2$$

How to choose lambda, such that we don't underfit/overfit?

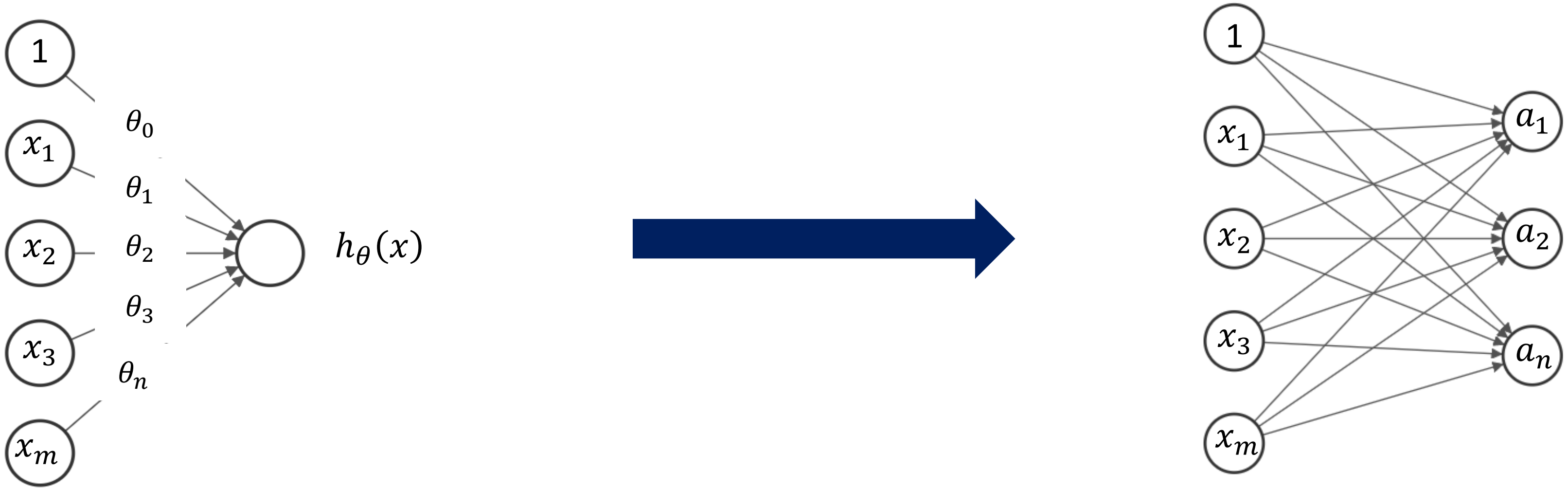
- Perform a grid search on different order of magnitudes with reduced, different, datasets (coarse search)
- Plot the error vs Model Complexity or vs Lambda
- Around the selected order of magnitude perform a finer search with the full dataset



SHALLOW NETWORK
INTUITION



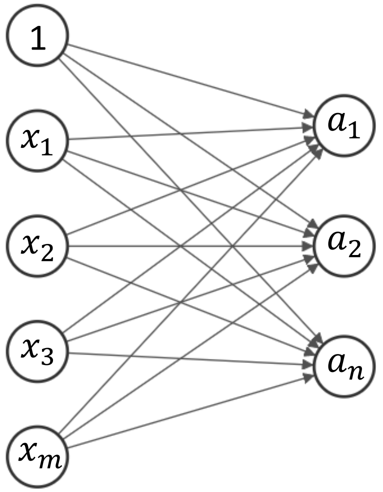
SHALLOW NETWORK
INTUITION



SHALLOW NETWORK
INTUITION

$$a_n = g(\theta_n^T x) = g(\theta_{n0} + \theta_{n1}x_1 + \theta_{n2}x_2 + \theta_{n3}x_3 + \theta_{n4}x_4)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$



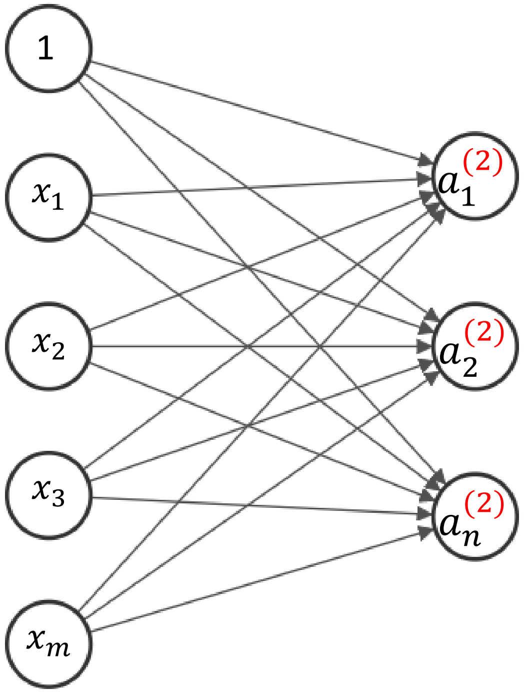
SHALLOW NETWORK

INTUITION

$$a_n = g(\theta_n^T x) = g(\theta_{n0} + \theta_{n1}x_1 + \theta_{n2}x_2 + \theta_{n3}x_3 + \theta_{n4}x_4)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$a_n^{(2)} = g(\Theta_{n0}^{(1)} + \Theta_{n1}^{(1)}x_1 + \Theta_{n2}^{(1)}x_2 + \cdots + \Theta_{nm}^{(1)}x_m)$$



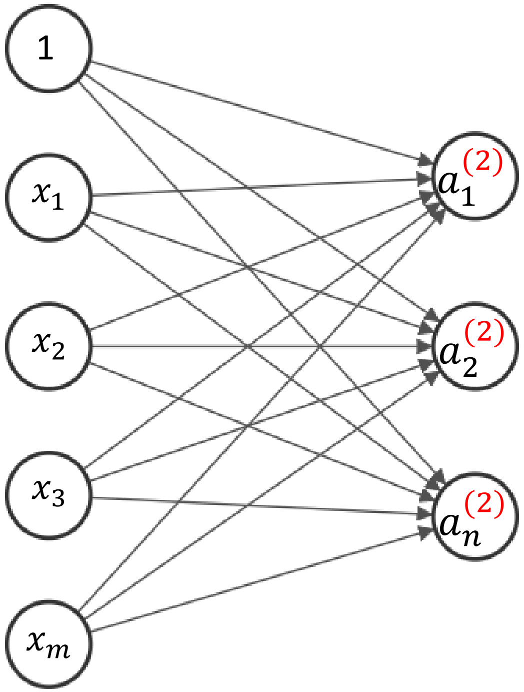
m – number of input features
n – number of neurons in current layer

SHALLOW NETWORK
INTUITION

$$a_n = g(\theta_n^T x) = g(\theta_{n0} + \theta_{n1}x_1 + \theta_{n2}x_2 + \theta_{n3}x_3 + \theta_{n4}x_4)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

$$a_n^{(2)} = g(\Theta_{n0}^{(1)} + \Theta_{n1}^{(1)}x_1 + \Theta_{n2}^{(1)}x_2 + \cdots + \Theta_{nm}^{(1)}x_m)$$



m – number of input features
n – number of neurons in current layer

Θ_{nm} is a matrix!

If $m = 4$ then $\dim(\Theta_{nm}) = (3, 5)$

m is 0 based due to the bias unit!

SHALLOW NETWORK

GENERALIZATION

$$a_n^{(2)} = g(\Theta_{n0}^{(1)} + \Theta_{n1}^{(1)} x_1 + \Theta_{n2}^{(1)} x_2 + \cdots + \Theta_{nm}^{(1)} x_m)$$

n – number of neurons in next layer
m – number of neurons in current layer

$$a^{(1)} = x$$

SHALLOW NETWORK

GENERALIZATION

$$a_n^{(2)} = g(\Theta_{n0}^{(1)} + \Theta_{n1}^{(1)} x_1 + \Theta_{n2}^{(1)} x_2 + \cdots + \Theta_{nm}^{(1)} x_m)$$

$$a^{(1)} = x$$

n – number of neurons in next layer
m – number of neurons in current layer
l – number of current layer

$$a_n^{(l+1)} = g(\Theta_{n0}^{(l)} + \Theta_{n1}^{(l)} a_1^{(l)} + \Theta_{n2}^{(l)} a_2^{(l)} + \cdots + \Theta_{nm}^{(l)} a_m^{(l)})$$

SHALLOW NETWORK

GENERALIZATION

$$a_n^{(2)} = g(\Theta_{n0}^{(1)} + \Theta_{n1}^{(1)} x_1 + \Theta_{n2}^{(1)} x_2 + \cdots + \Theta_{nm}^{(1)} x_m)$$

$$a^{(1)} = x$$

n – number of neurons in next layer
m – number of neurons in current layer
l – number of current layer

$$a_n^{(l+1)} = g(\Theta_{n0}^{(l)} + \Theta_{n1}^{(l)} a_1^{(l)} + \Theta_{n2}^{(l)} a_2^{(l)} + \cdots + \Theta_{nm}^{(l)} a_m^{(l)})$$

l – number of current layer
$s_l \equiv m$
$s_{l+1} \equiv n$

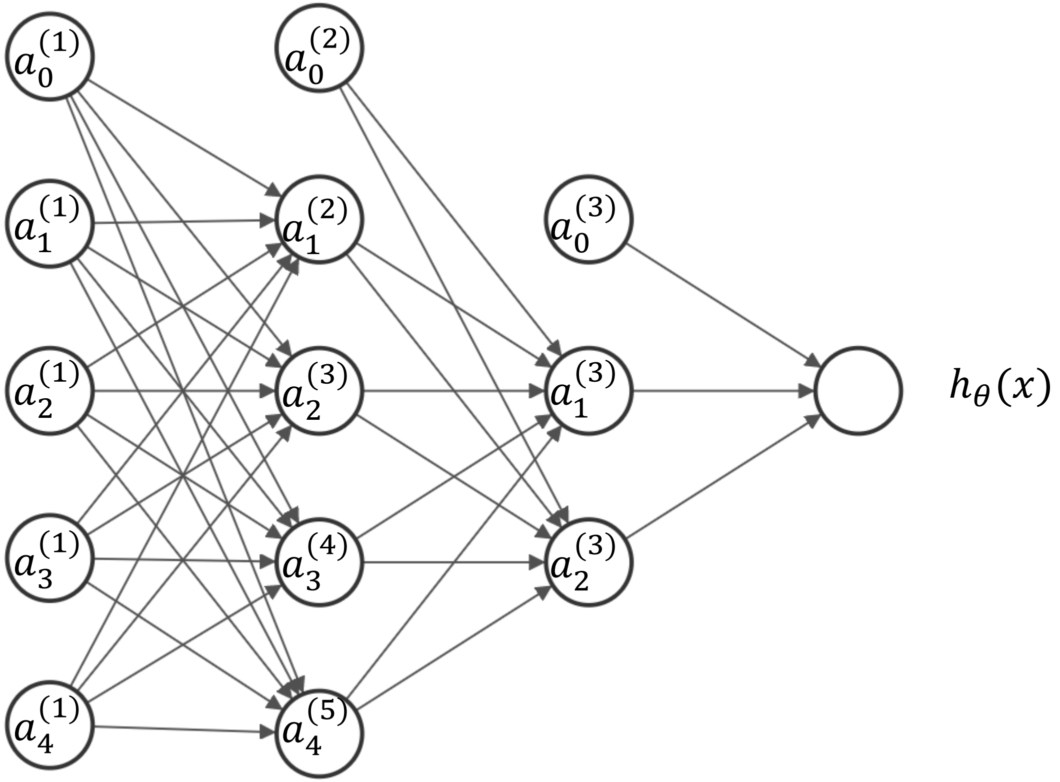
$$a_{s_{l+1}}^{(l+1)} = g(\Theta_{s_{l+1},0}^{(l)} + \Theta_{s_{l+1},1}^{(l)} a_1^{(l)} + \Theta_{s_{l+1},2}^{(l)} a_2^{(l)} + \cdots + \Theta_{s_{l+1},s_l}^{(l)} a_{s_l}^{(l)})$$

DEEP NETWORKS

GENERALIZATION

$$a_{s_{l+1}}^{(l+1)} = g(\Theta_{s_{l+1},0}^{(l)} \overset{=1}{a_0^{(l)}} + \Theta_{s_{l+1},1}^{(l)} a_1^{(l)} + \Theta_{s_{l+1},2}^{(l)} a_2^{(l)} + \cdots + \Theta_{s_{l+1},s_l}^{(l)} a_{s_l}^{(l)})$$

s_{l+1} – number of neurons in next layer
s_l – number of neurons in current layer
l – number of current layer

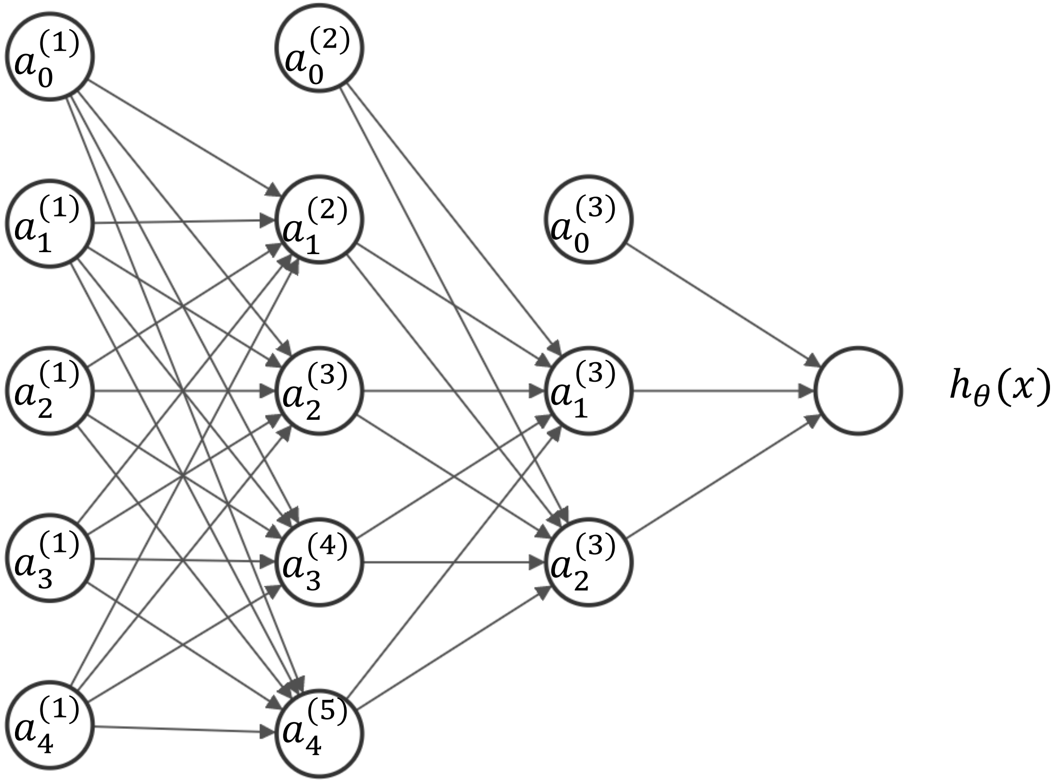


DEEP NETWORKS

GENERALIZATION

$$a_{s_{l+1}}^{(l+1)} = g(\Theta_{s_{l+1},0}^{(l)}\underbrace{a_0^{(l)}}_{=1} + \Theta_{s_{l+1},1}^{(l)}a_1^{(l)} + \Theta_{s_{l+1},2}^{(l)}a_2^{(l)} + \cdots + \Theta_{s_{l+1},s_l}^{(l)}a_{s_l}^{(l)})$$

s_{l+1} – number of neurons in next layer
s_l – number of neurons in current layer
l – number of current layer



$$a^{(1)} = x$$

$$a^{(2)} = g(\Theta^{(1)}.a^{(1)})$$

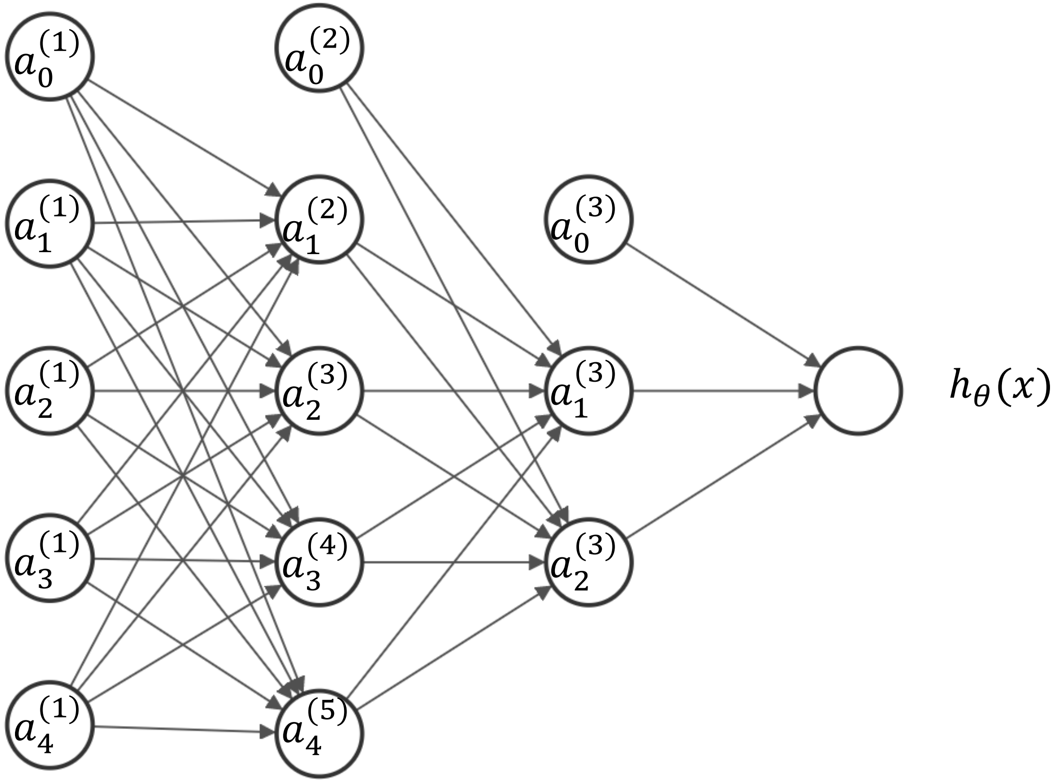
$$a^{(3)} = g(\Theta^{(2)}.a^{(2)})$$

$$h_\theta(x) = g(\Theta^{(3)}.a^{(3)})$$

DEEP NETWORKS
GENERALIZATION

$$a_{s_{l+1}}^{(l+1)} = g(\Theta_{s_{l+1},0}^{(l)} \underbrace{a_0^{(l)}}_{=1} + \Theta_{s_{l+1},1}^{(l)} a_1^{(l)} + \Theta_{s_{l+1},2}^{(l)} a_2^{(l)} + \cdots + \Theta_{s_{l+1},s_l}^{(l)} a_{s_l}^{(l)})$$

s_{l+1} – number of neurons in next layer
s_l – number of neurons in current layer
l – number of current layer



$$a^{(1)} = x$$

Add $a_0^{(1)} = x_0 = 1$

…

$$a^{(2)} = g(\Theta^{(1)} . a^{(1)})$$

Add $a_0^{(2)} = 1$

…

$$a^{(3)} = g(\Theta^{(2)} . a^{(2)})$$

Add $a_0^{(3)} = 1$

…

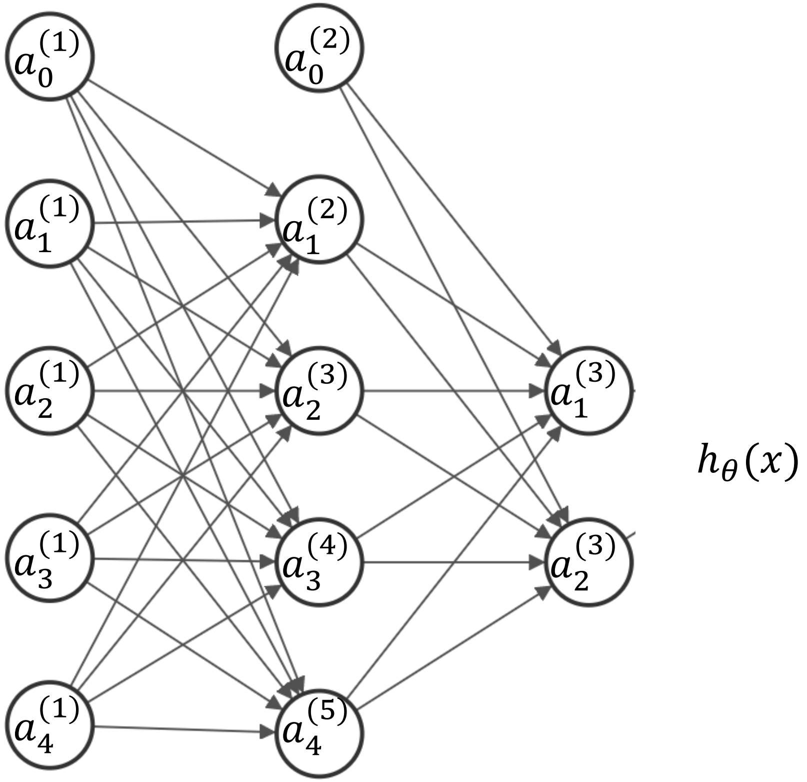
$$h_\theta(x) = g(\Theta^{(3)} . a^{(3)})$$

DEEP NETWORKS
GENERALIZATION

MULTICLASSIFICATION

$$a^{(l+1)}_{s_{l+1}} = g(\Theta^{(l)}_{s_{l+1},0}\underbrace{a^{(l)}_0}_{=1} + \Theta^{(l)}_{s_{l+1},1}a^{(l)}_1 + \Theta^{(l)}_{s_{l+1},2}a^{(l)}_2 + \cdots + \Theta^{(l)}_{s_{l+1},s_l}a^{(l)}_{s_l})$$

s_{l+1} – number of neurons in next layer
s_l – number of neurons in current layer
l – number of current layer



$$a^{(1)} = x$$

...

$$a^{(2)} = g(\Theta^{(1)}.a^{(1)})$$

...

$$h_\theta(x) = a^{(3)} = g(\Theta^{(2)}.a^{(2)})$$

Add $a^{(1)}_0 = x_0 = 1$

Add $a^{(2)}_0 = 1$

Logistic Regression

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right).\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta^2$$

Neural Network

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right)_k + \left(1 - y_k^{(i)}\right).\log\left(1 - h_\theta\left(x^{(i)}\right)\right)_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

Logistic Regression

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right).\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta^2$$

Neural Network

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right)_k + \left(1 - y_k^{(i)}\right).\log\left(1 - h_\theta\left(x^{(i)}\right)\right)_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}} \left(\Theta_{ji}^{(l)}\right)^2$$

For K outputs

Logistic Regression

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right).\log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta^2$$

Neural Network

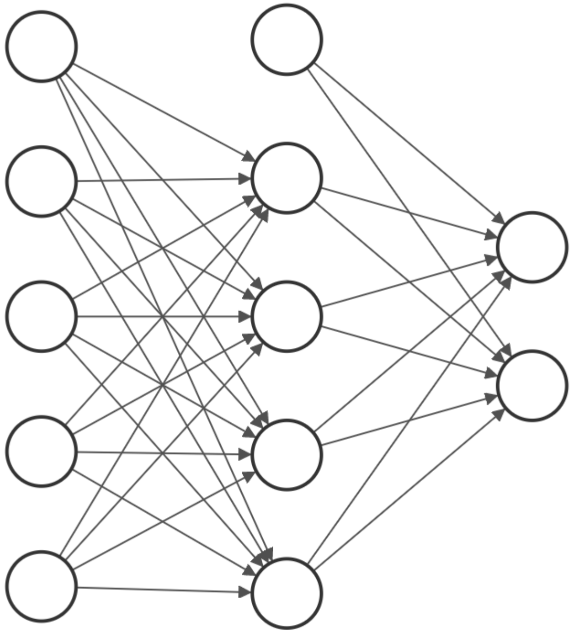
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\theta\left(x^{(i)}\right)\right)_k + \left(1 - y_k^{(i)}\right).\log\left(1 - h_\theta\left(x^{(i)}\right)\right)_k\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2$$

For K outputs

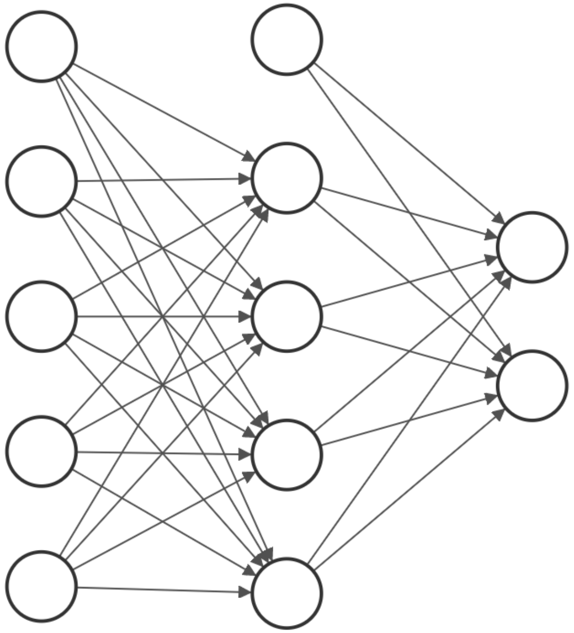
Sum of regularization term for all weights, on all layers

DEEP NETWORKS
BACKPROPAGATION

$$\delta_j^{(l)} = \text{"}error\text{"}\ of\ node\ j\ in\ layer\ l$$



DEEP NETWORKS
BACKPROPAGATION



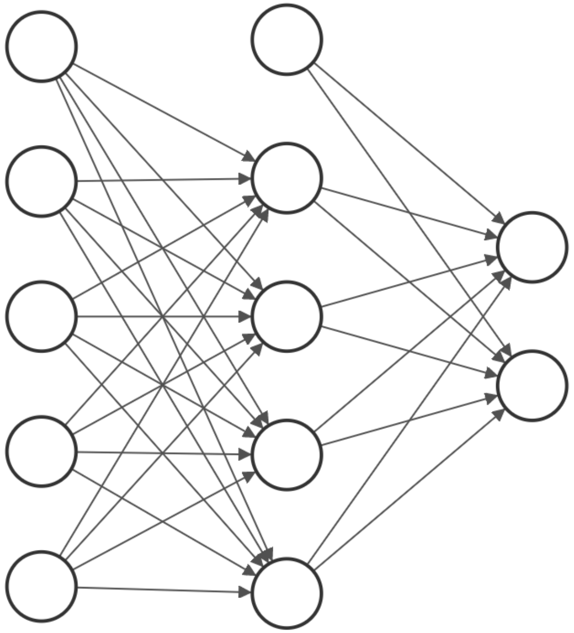
$$\delta_j^{(l)} = \text{"}error\text{"}\ of\ node\ j\ in\ layer\ l$$

$$\delta_j^{(3)} = a_j^{(3)} - y_j$$

DEEP NETWORKS
BACKPROPAGATION

$$\delta_j^{(l)} = \text{"error" of node } j \text{ in layer } l$$

$$\delta_j^{(3)} = a_j^{(3)} - y_j$$



If we use the chain rule to manipulate the derivative of the cost function it is possible to obtain the following expression to compute $\delta_j^{(l-1)}$:

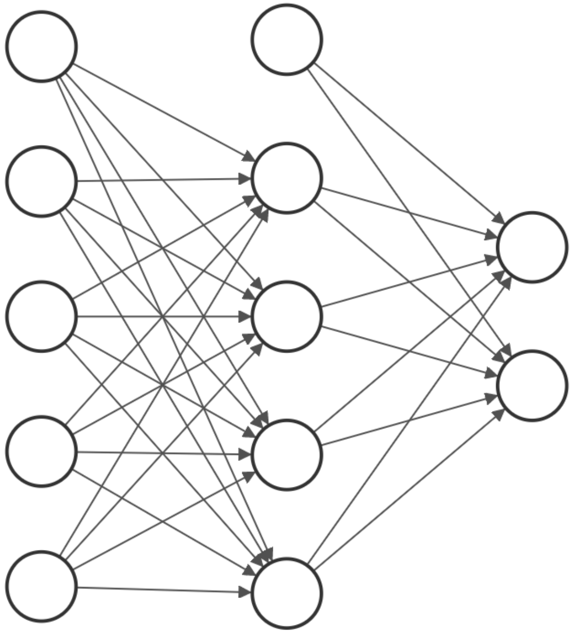
$$\delta_j^{(l-1)} = (\Theta^{(l-1)})^T \delta_j^{(l)} .* g'\left(z^{(l-1)}\right)$$

with

$$z^{(l-1)} = \Theta^{(l-2)} . a^{(l-2)}$$

Computed on the forward pass

DEEP NETWORKS
BACKPROPAGATION



$$\delta_j^{(l)} = \text{"}error\text{"}\ of\ node\ j\ in\ layer\ l$$

$$\delta_j^{(3)} = a_j^{(3)} - y_j$$

$$\delta_j^{(l-1)} = (\Theta^{(l-1)})^T \delta_j^{(l)} .* \, g'\left(z^{(l-1)}\right)$$

$$\boxed{z^{(l-1)} = \Theta^{(l-2)} . a^{(l-2)}}$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} = D_{ij}^{(l)}$$

How to perform updates over all the training examples ?

For each epoch (or alternatively batch):

Initialize an error matrix $\Delta_{ij}^{(l)} = 0$, (for all i, j, l)

For each example in epoch/batch

Compute prediction

Compute error ($\delta^{(L)} = a^{(L)} - y$)

Compute all $\delta_j^{(l)}, 1 \leq l \leq L-1$

Update $\Delta_{ij}^{(l)}$ as $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Average $\Delta_{ij}^{(l)}$ over number of examples used: $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$

If using regularization, add the regularization term:

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \; if \; j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \; if \; j = 0$$

Update weights on the entire network with:

$$\Theta_{ij}^{(l)} := \Theta_{ij}^{(l)} - \alpha \frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) := \Theta_{ij}^{(l)} - \alpha D_{ij}^{(l)}$$

$$\delta_j^{(3)} = a_j^{(3)} - y_j$$

$$\delta_j^{(l-1)} = (\Theta^{(l-1)})^T \delta_j^{(l)} .* \, g'\left(z^{(l-1)}\right)$$

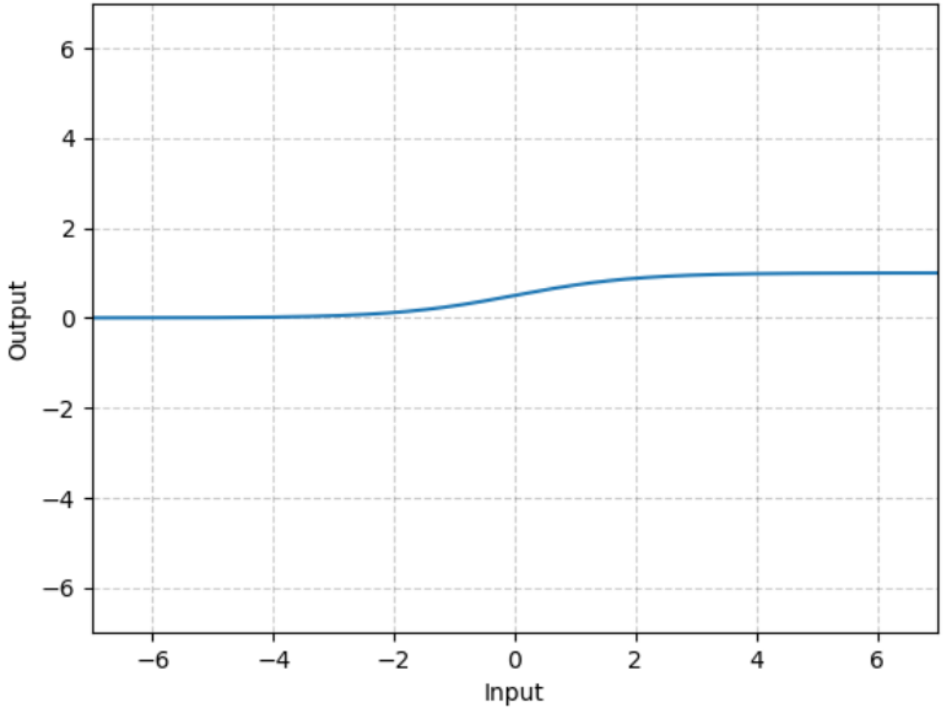
$$\boxed{z^{(l-1)} = \Theta^{(l-2)} . a^{(l-2)}}$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} = D_{ij}^{(l)}$$

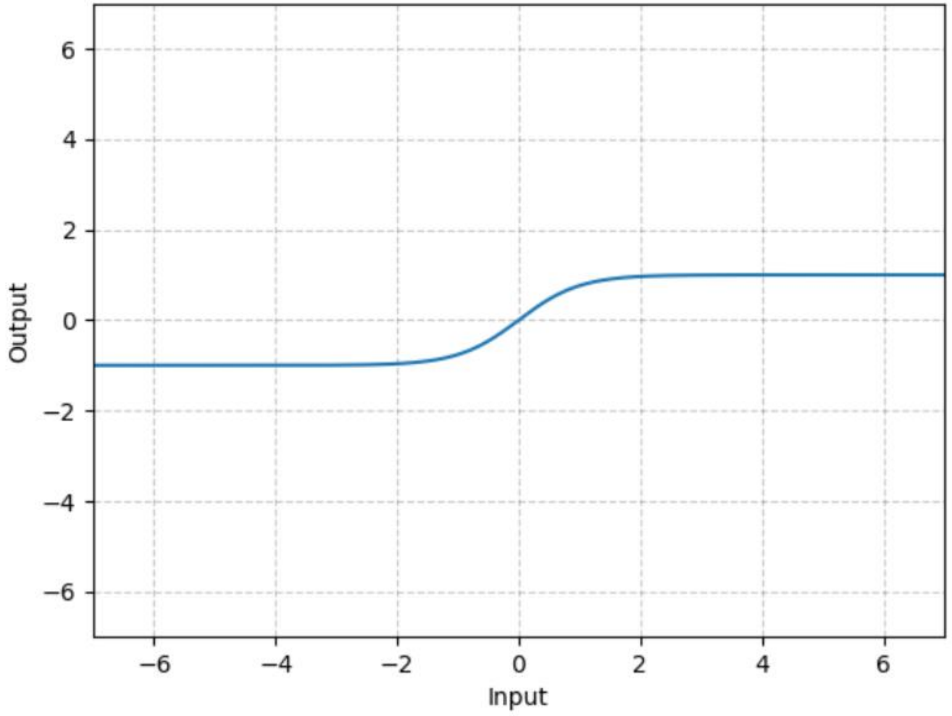
For a single example! If several examples are to be used to compute one update, the value must be averaged, as at each example the errors are added cumulatively

ACTIVATION FUNCTIONS

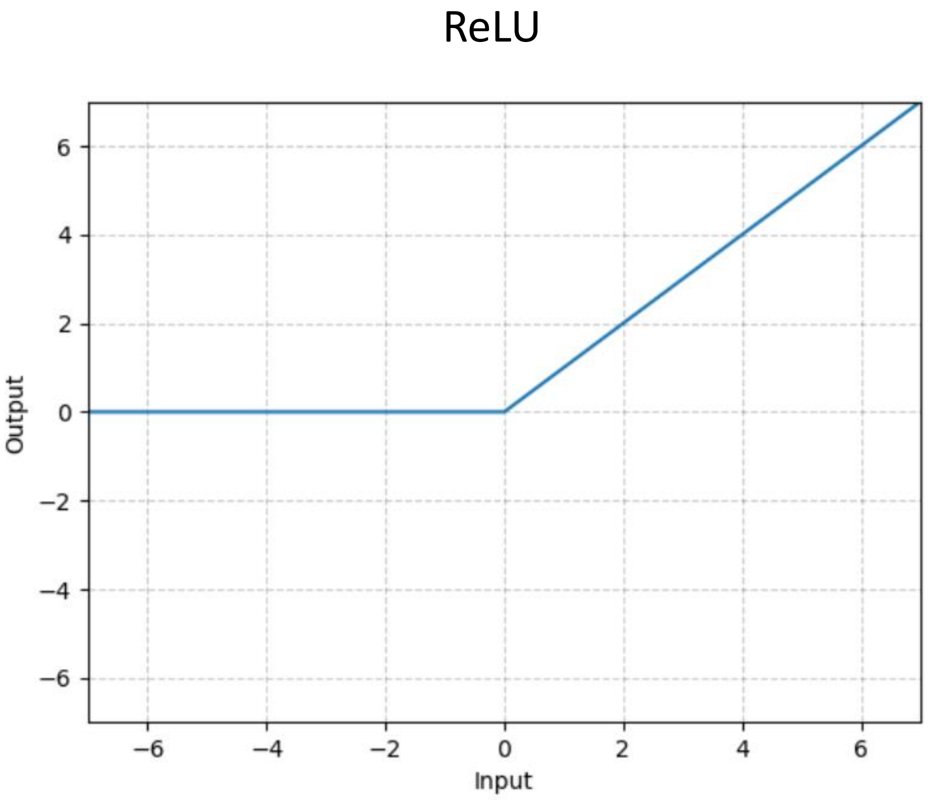
Sigmoid

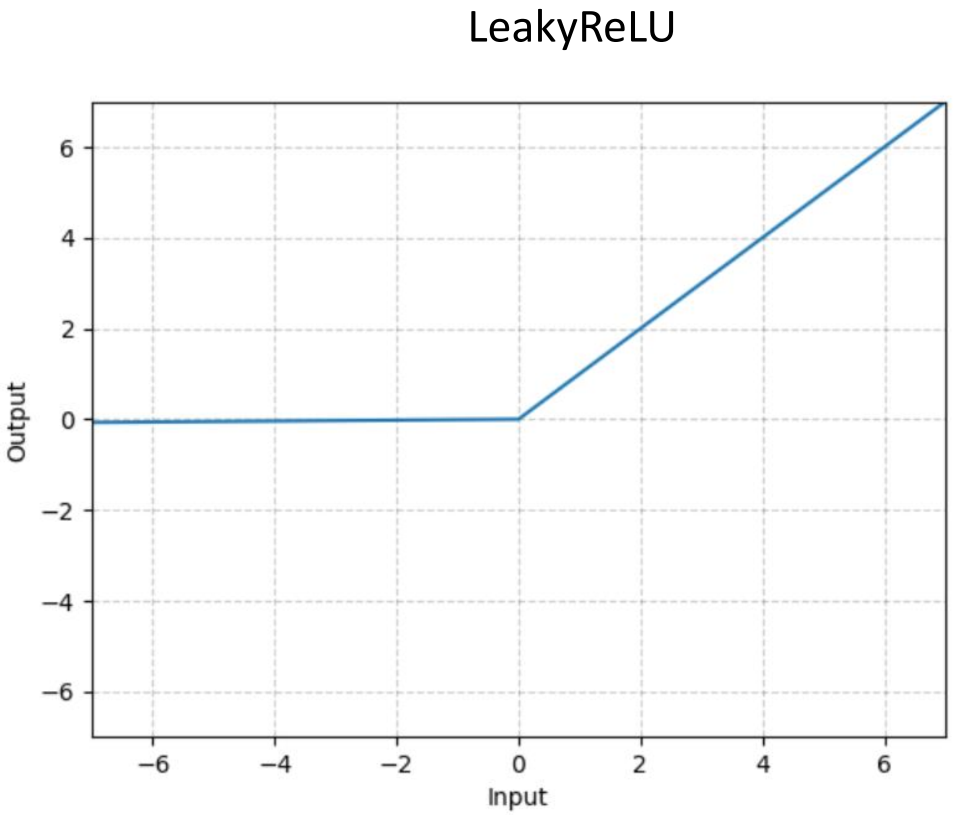


Tanh



ACTIVATION FUNCTIONS





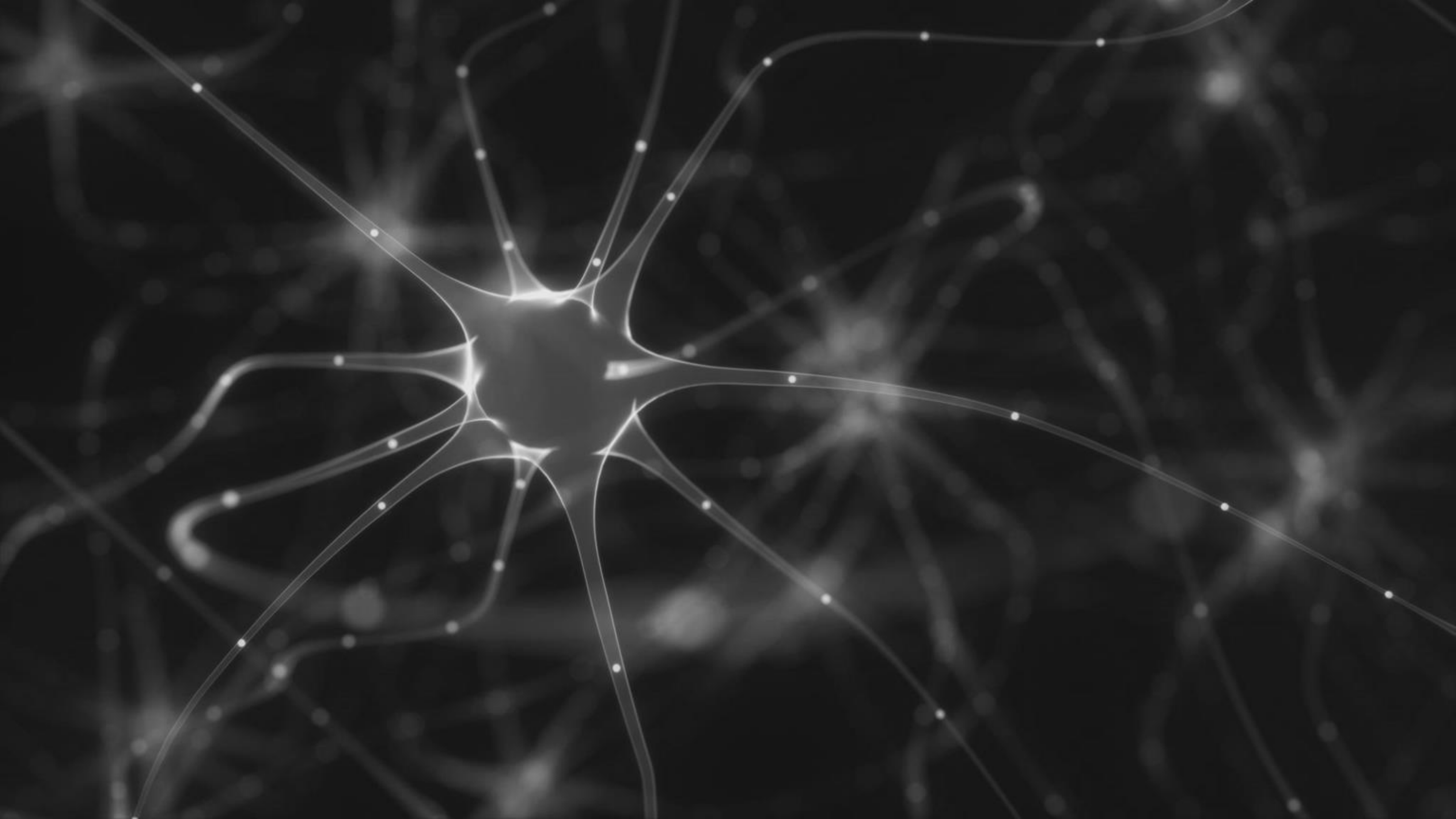
https://pytorch.org/docs/stable/nn.html

INPUT NORMALIZATION/STANDARDIZATION

- Inputs should be on same scale (order of magnitude)
- When magnitudes are different, weights values can also be very different
- Consequently, updates on weights based on the derivatives can rapidly make some weights explode or switch signal or practically don’t update at all
- It can also generate exploding or vanishing gradients, resulting that the last layers can still learn with the errors, but the error backpropagation don’t reach the initial layers
- This can make the network to not converge (or even diverge), despite the learning rate chosen

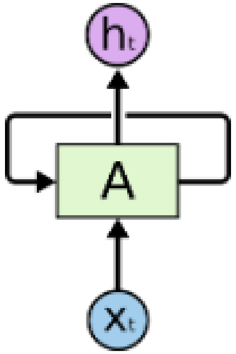
REGULARIZATION TECHNIQUES ON NEURAL NETWORKS

- L1 and L2 regularization is possible
- Additionally, it is possible to introduce dropout or batch normalization layers between hidden layers
- Dropout layers turn off a set number of neurons in the previous layer (given as a ratio). The neurons are chosen randomly.
- Batch normalization layers perform normalization of the outputs of the previous layer, resetting the outputs of that layer to a range within 0 to 1. This is especially effective when using activation functions with a linear behavior.
- Tuning the batch size can also have a regularization effect. Usually higher batch sizes tend to oblige the network to perform more generalist updates



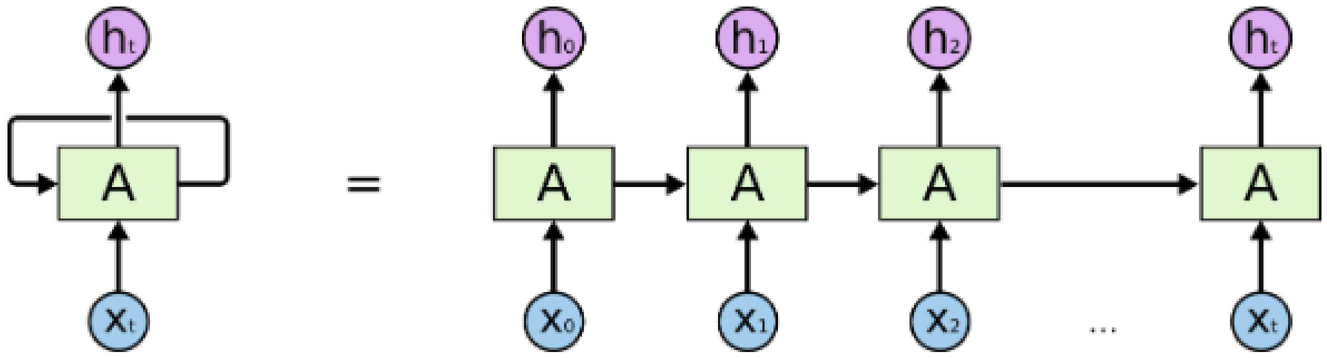
OTHER ARCHITECTURES

RECURRENT NEURAL NETWORK (RNN)



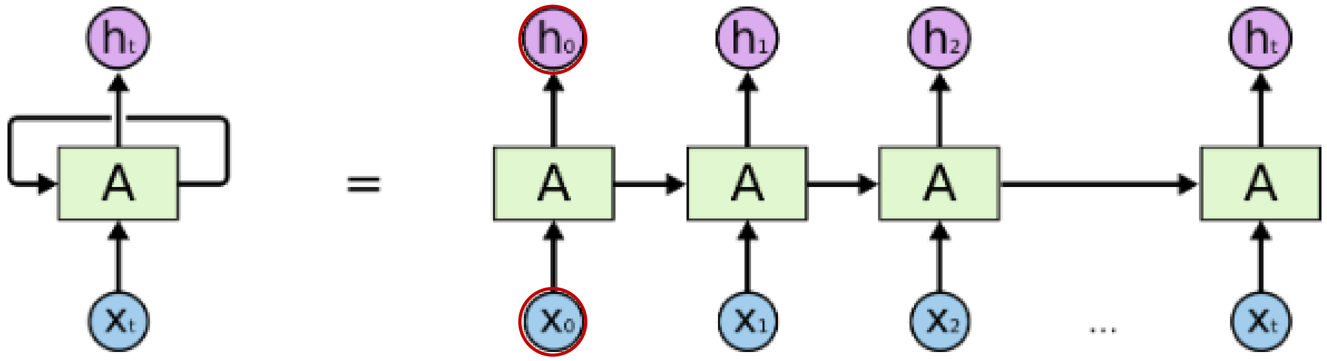
OTHER ARCHITECTURES

RECURRENT NEURAL NETWORK (RNN)

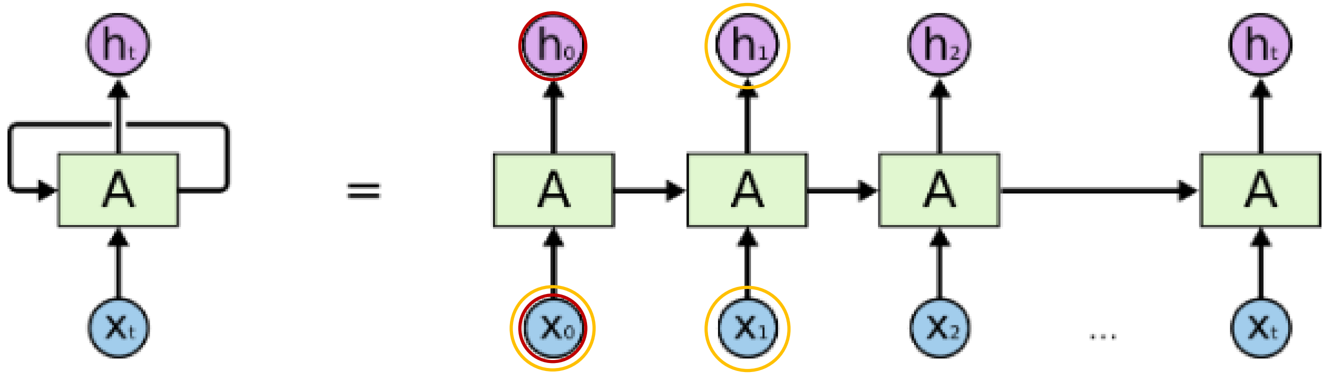


OTHER ARCHITECTURES

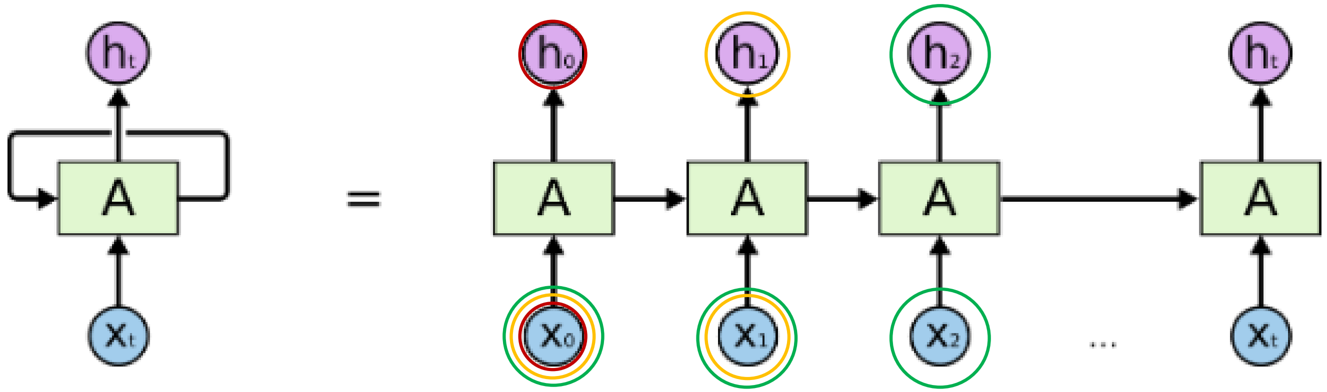
RECURRENT NEURAL NETWORK (RNN)



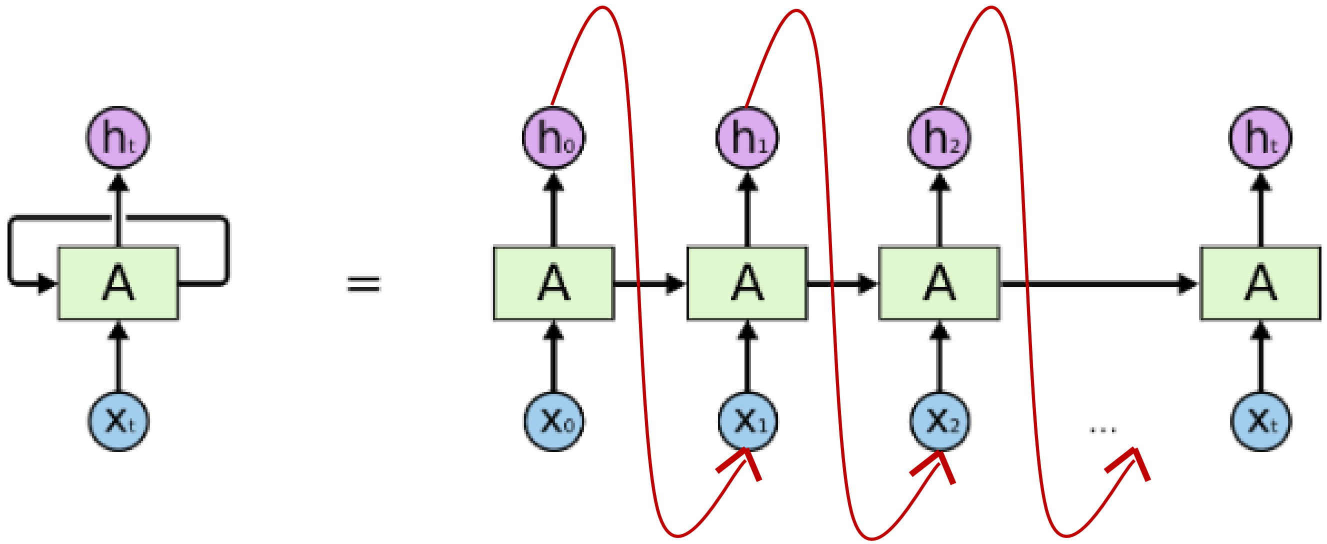
OTHER ARCHITECTURES
RECURRENT NEURAL NETWORK (RNN)



OTHER ARCHITECTURES
RECURRENT NEURAL NETWORK (RNN)



OTHER ARCHITECTURES
RECURRENT NEURAL NETWORK (RNN)



OTHER ARCHITECTURES

CONVOLUTIONAL NEURAL NETWORKS (CNN)

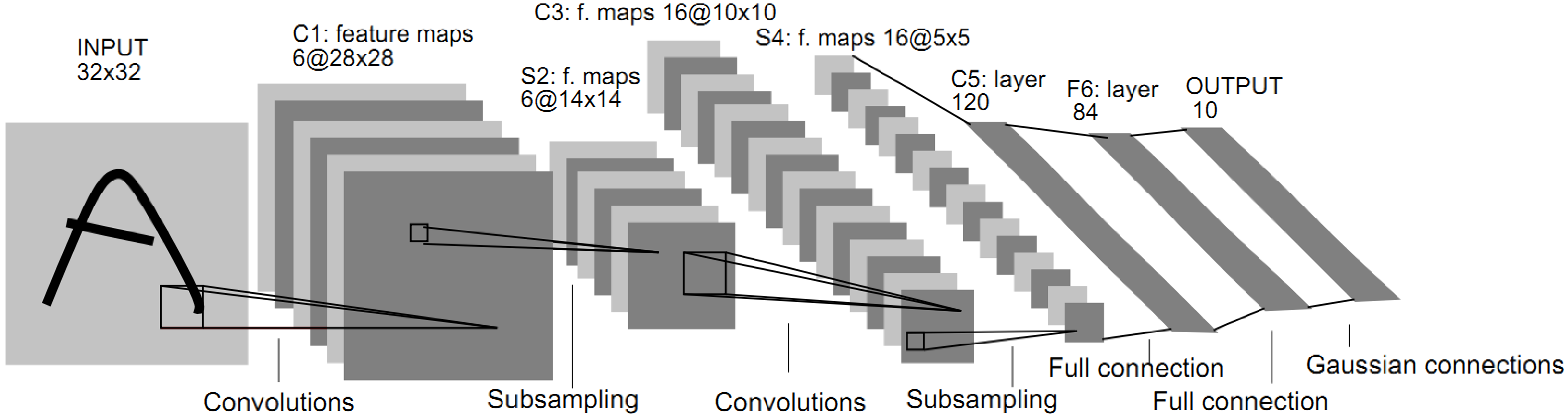
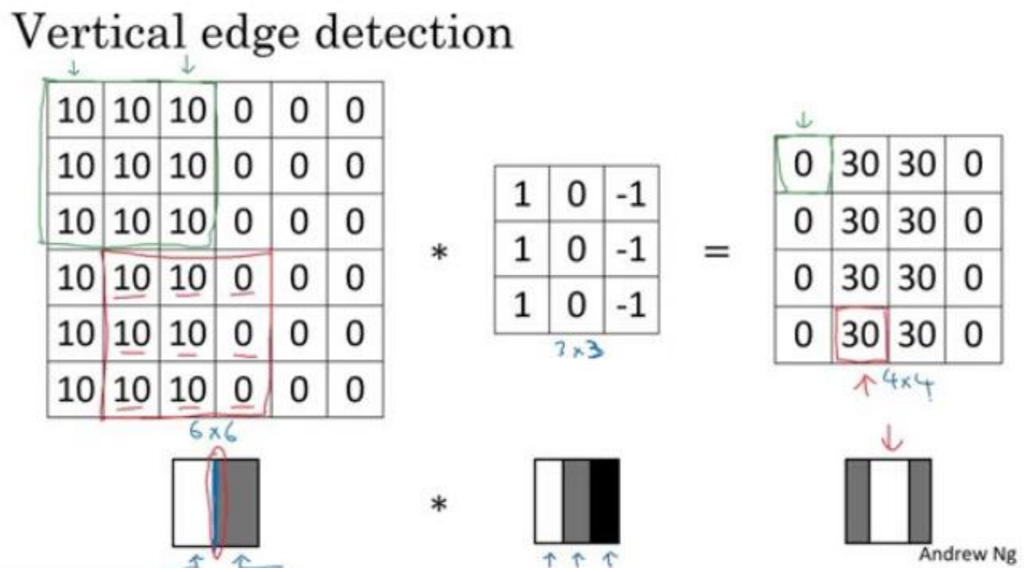
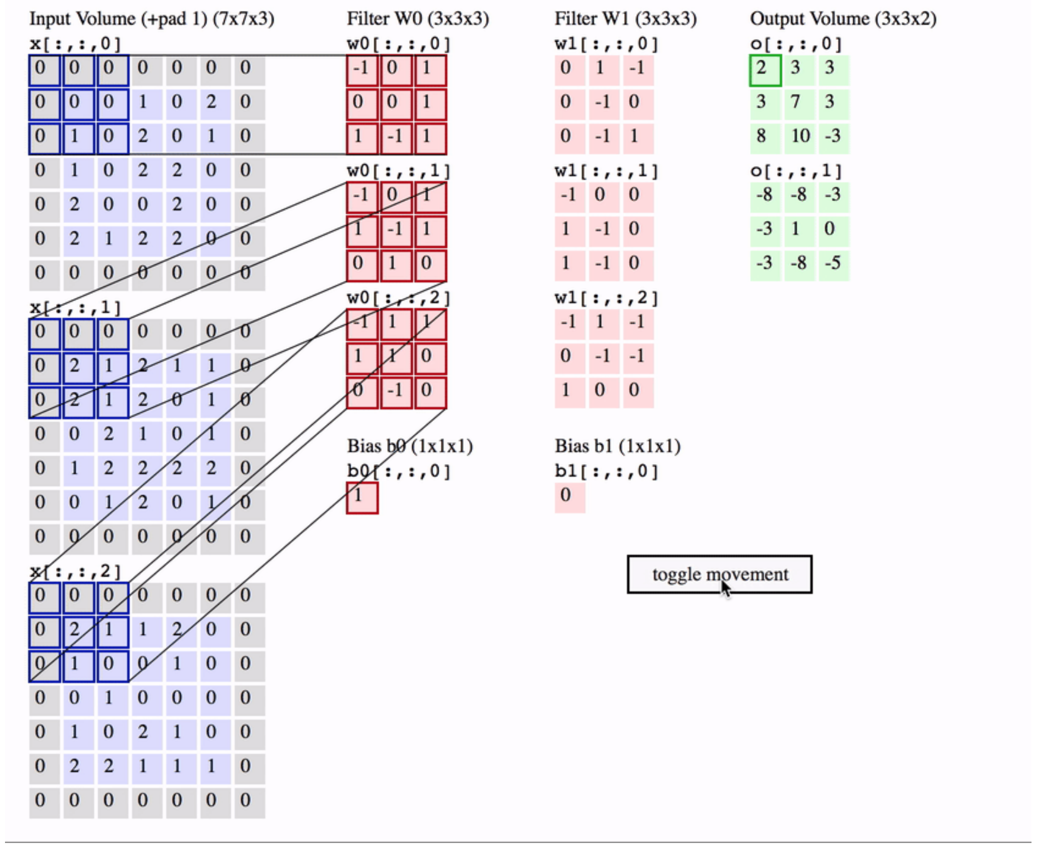


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

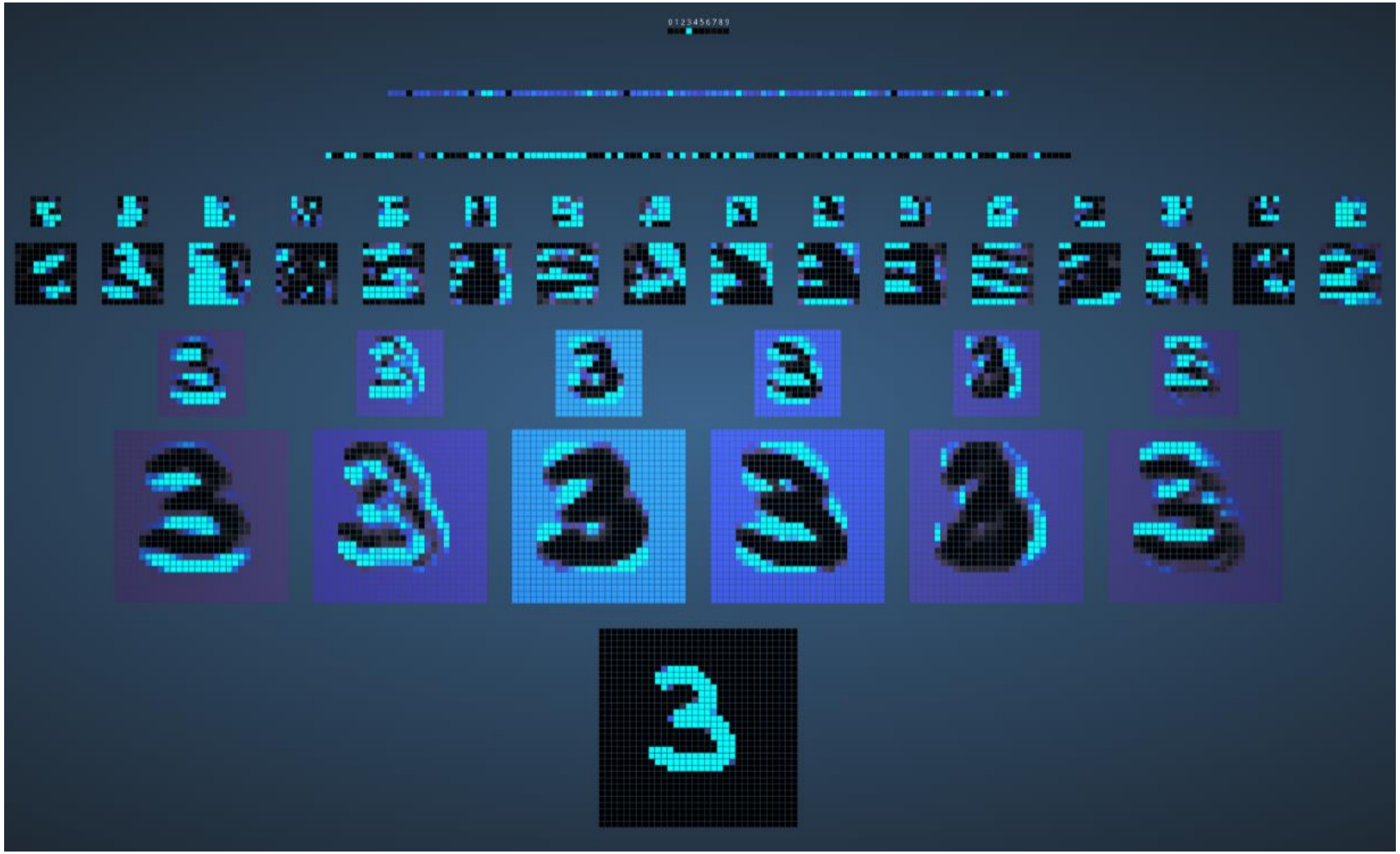
OTHER ARCHITECTURES

CONVOLUTIONAL NEURAL NETWORKS (CNN)

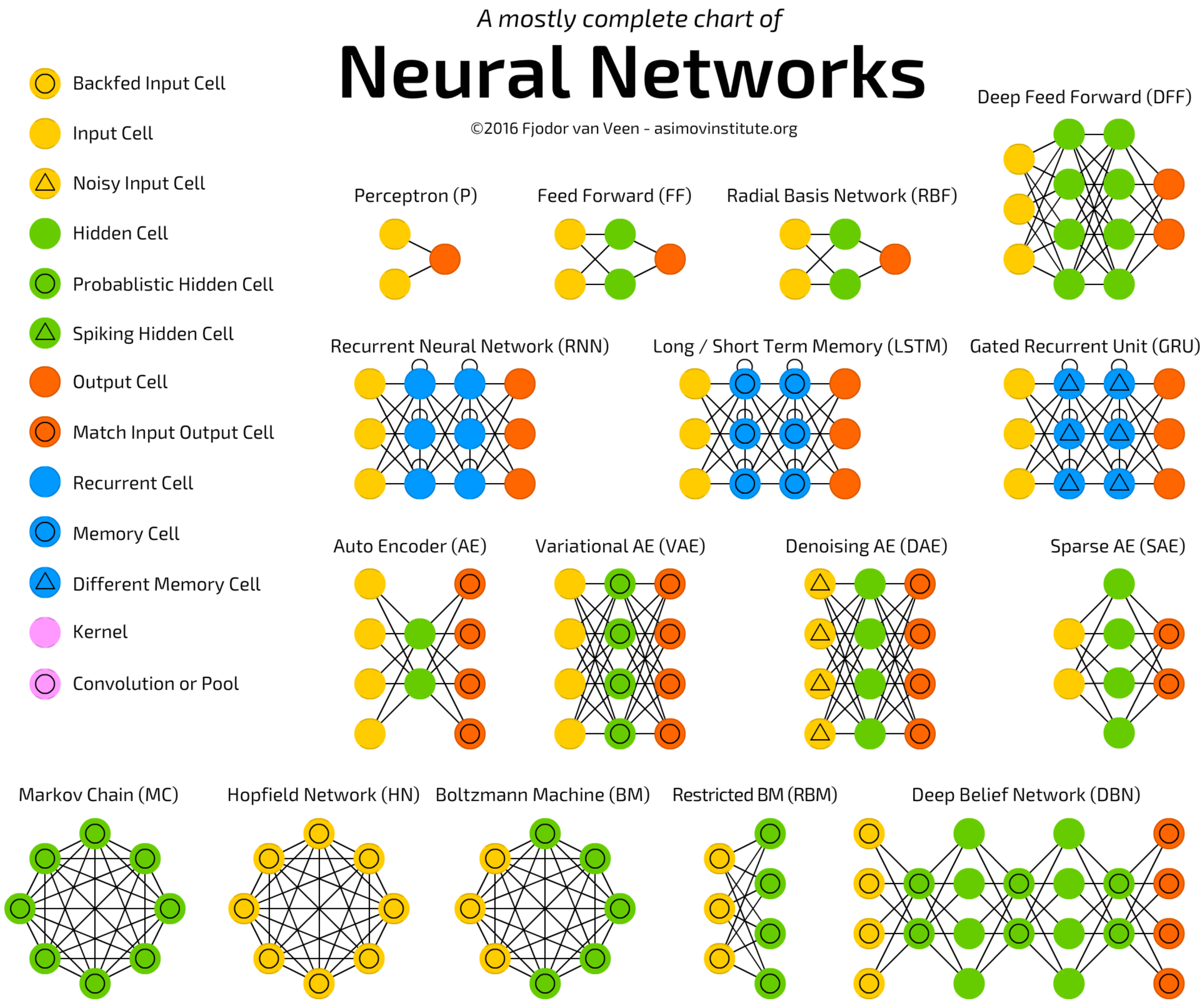


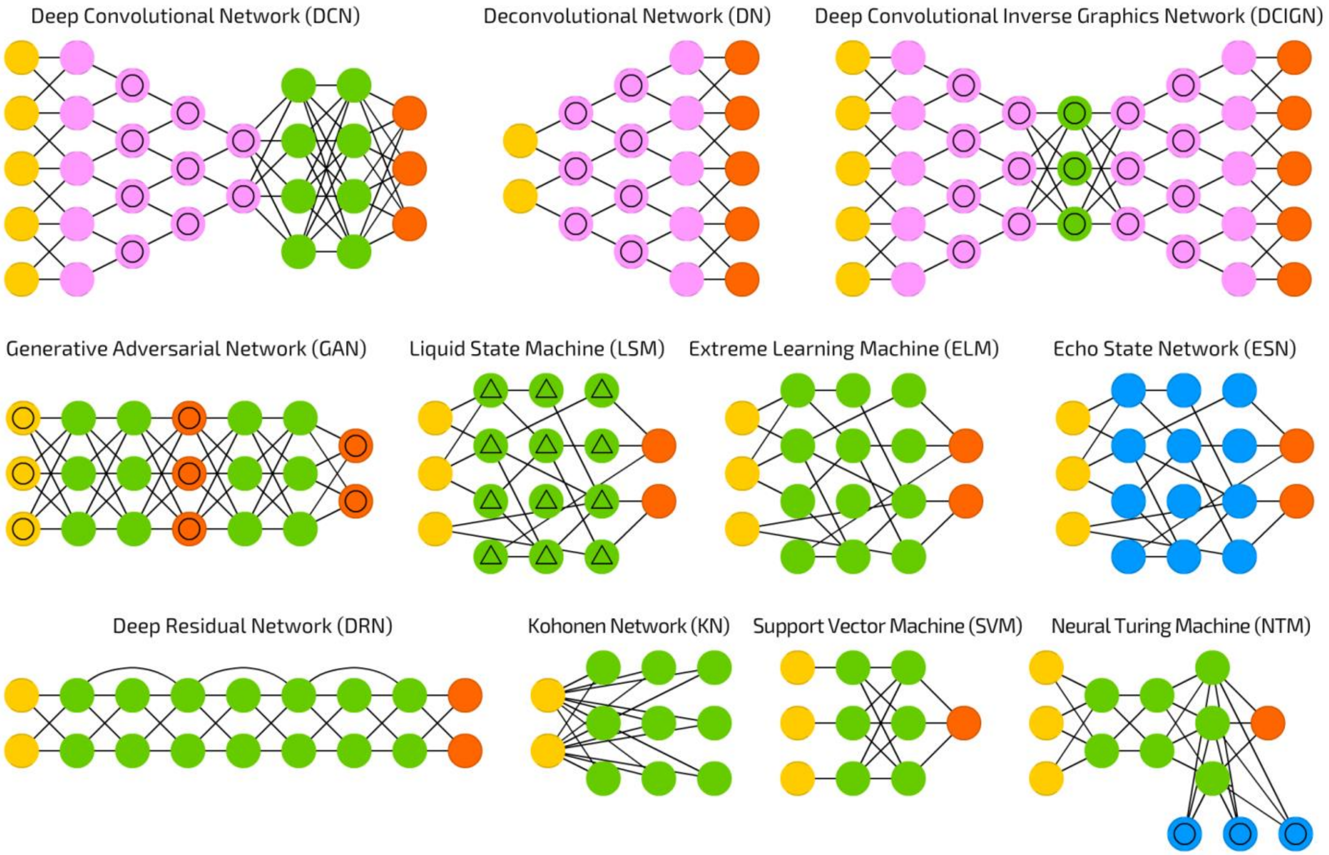


OTHER ARCHITECTURES
CONVOLUTIONAL NEURAL NETWORKS (CNN)



https://www.cs.ryerson.ca/~aharley/vis/conv/flat.html


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)



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