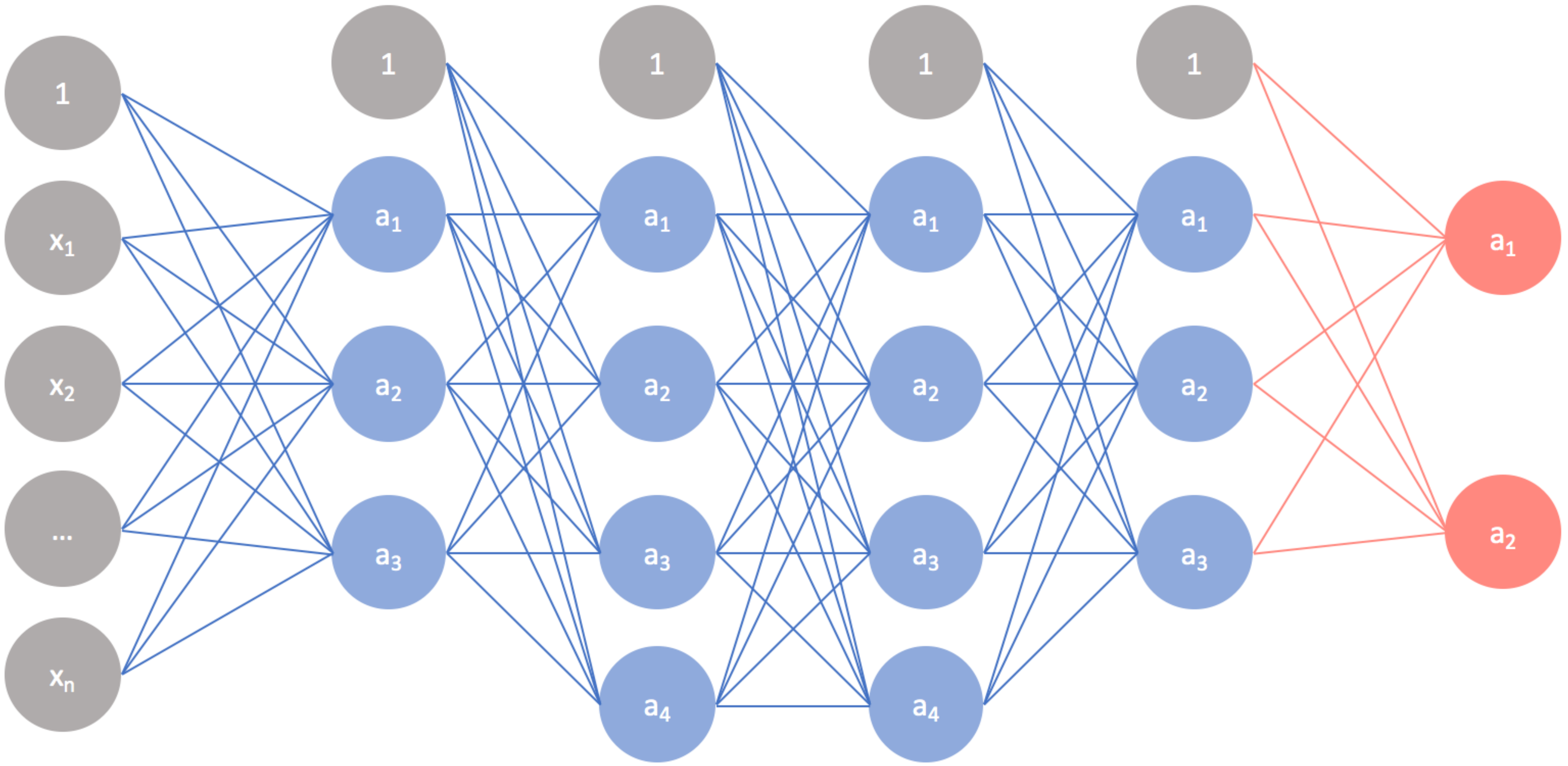
NN basics



References

- http://cs231n.stanford.edu/index.html
- http://www.cs.cornell.edu/courses/cs5670/2019sp/lectures/lectures.html
- http://www.cs.cmu.edu/~16385/

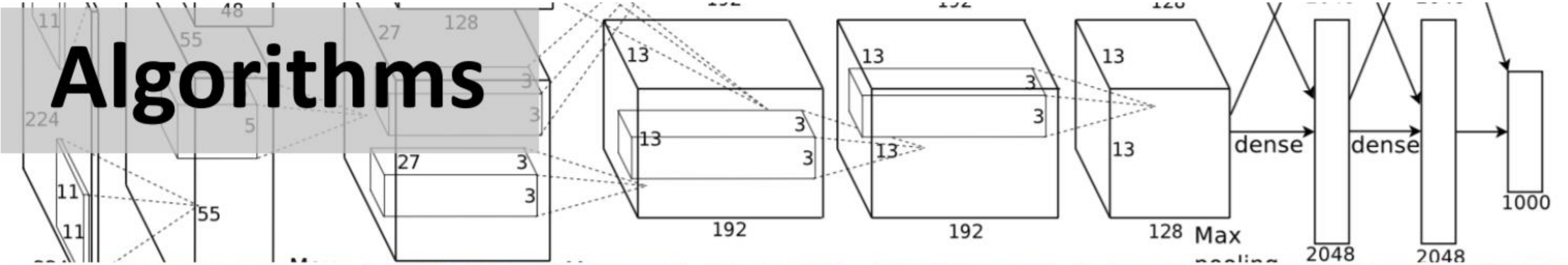
What will we know to do?

- Hopefully by the end of the course:
- https://teachablemachine.withgoogle.com/

What is a neural network

- **Artificial neural networks** (**ANN** / **NN**) are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules.
  - [Wikipedia]

What does a NN needs?

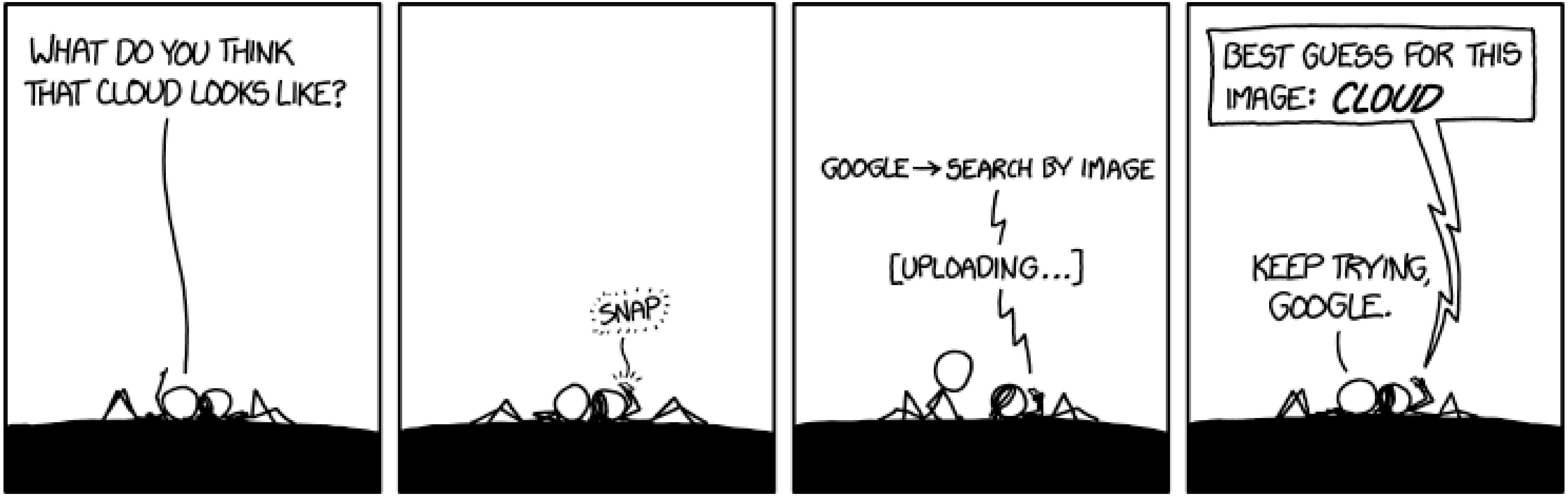


What a neural network can do?

- Image based:
  - Object recognition
  - Human pose detection
  - 3D reconstruction from a signal image
  - Image captioning
  - Style transfer
- Non image based:
  - Language translation
  - Game playing
- And much-much more…

Object recognition

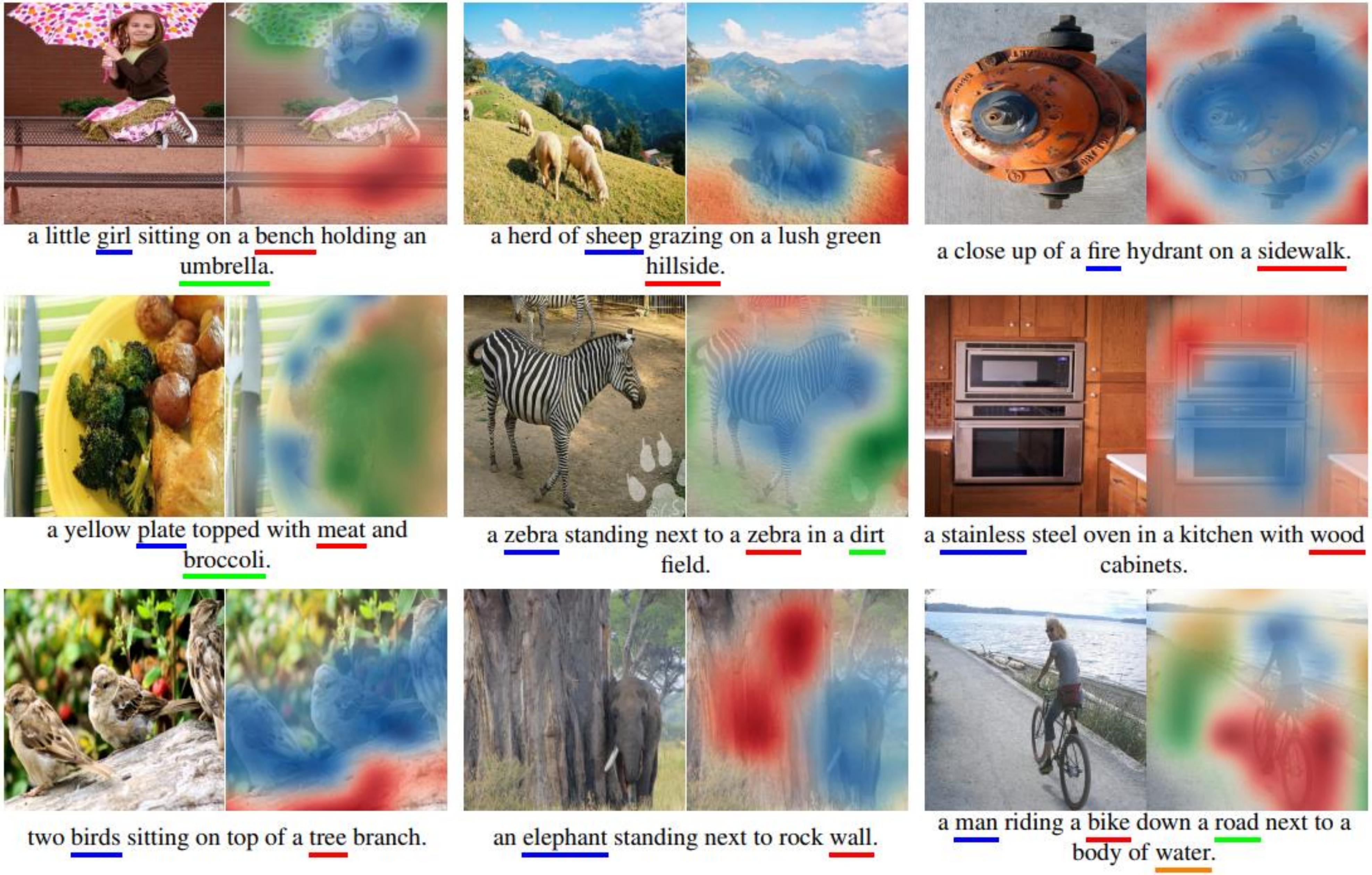
Object recognition



Human pose detection

3D reconstruction from a single image

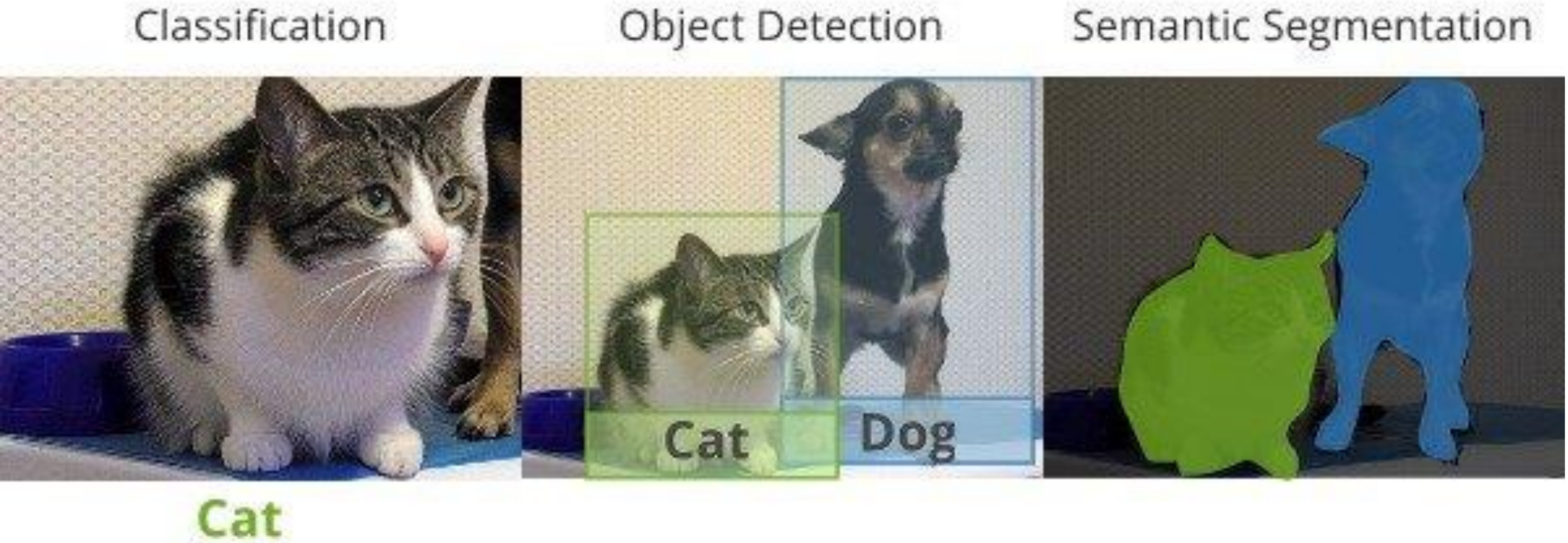
Image captioning



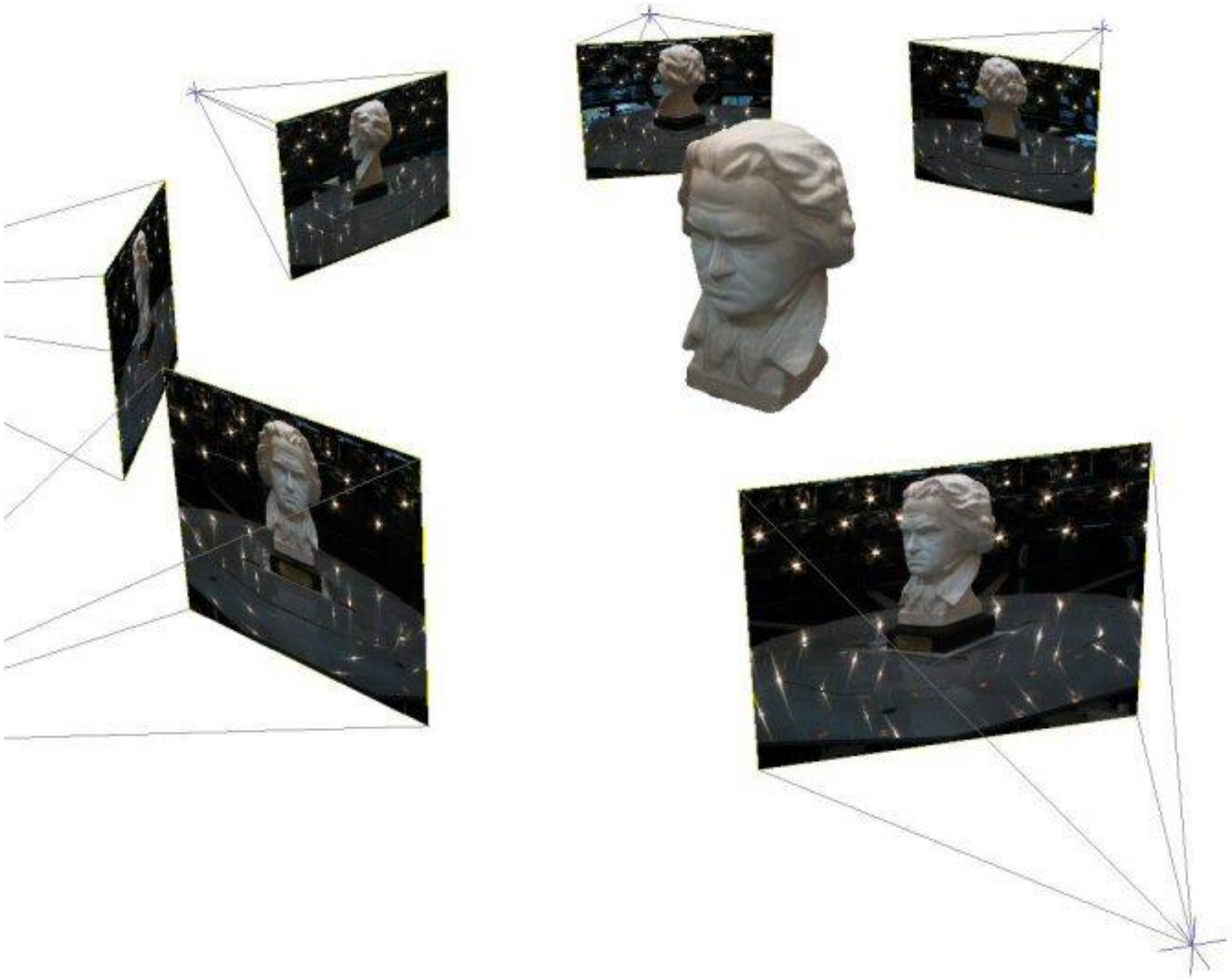
Style transfer

Object recognition challenges

- As we've seen before- object recognition is hard!



Challenge: variable viewpoint



Challenge: variable illumination

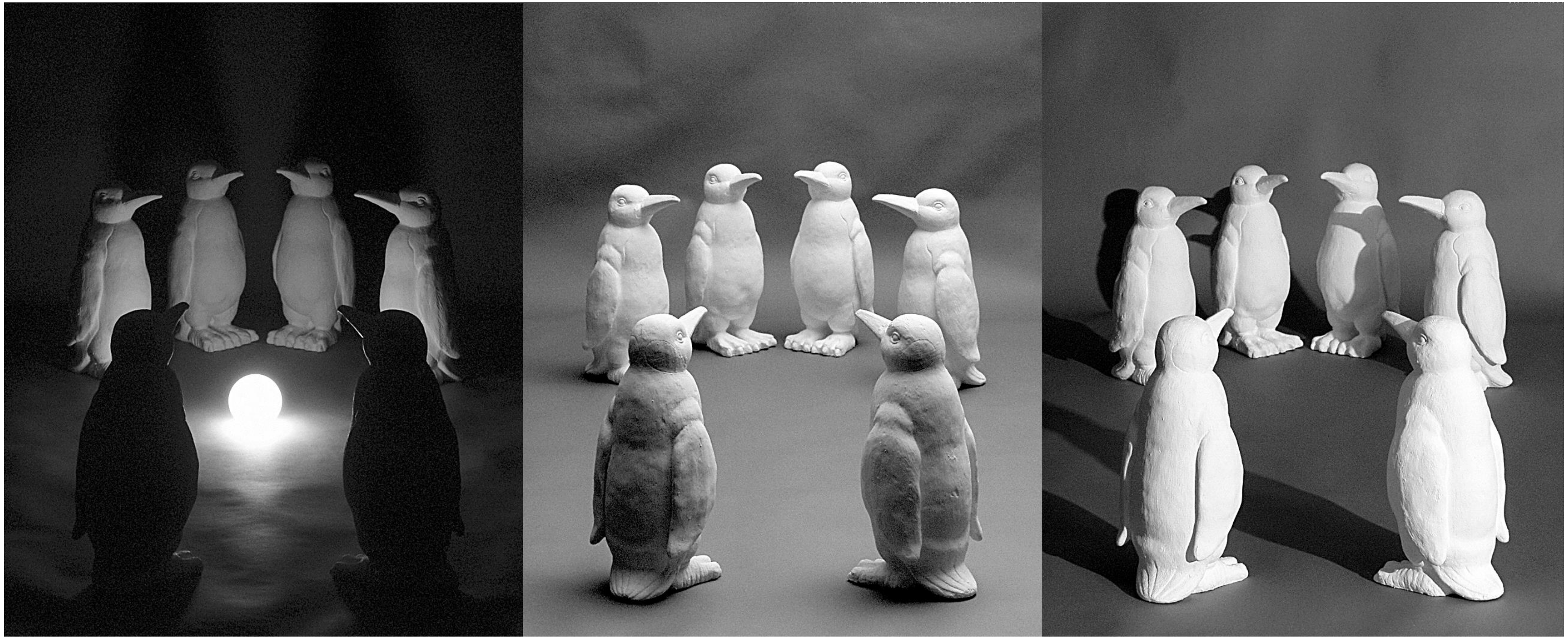


image credit: J. Koenderink

Challenge: scale

Challenge: deformation

Challenge: occlusion

Challenge: background clutter

Challenge: intra-class variations

Svetlana Lazebnik

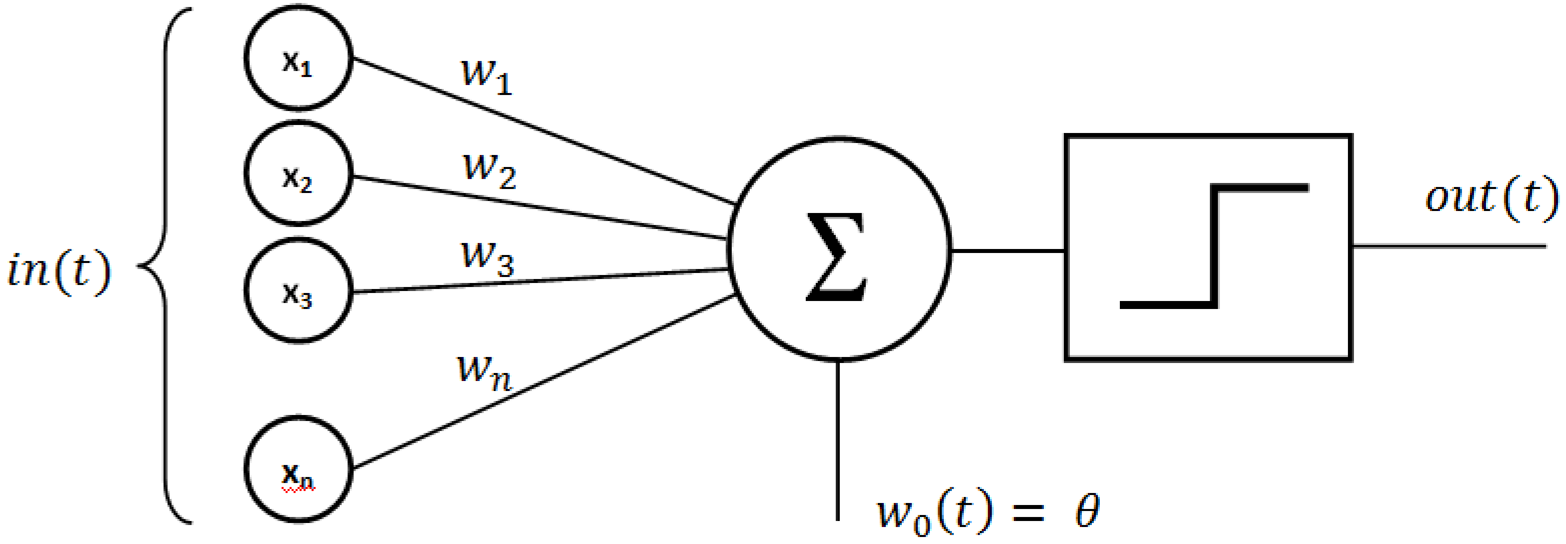
Object recognition challenges

- We’ve already seen that this is a hard problem to tackle with “classic” CV algorithms like SIFT and template matching.
  - Template matching does a relatively good job to find the same template instance in an image.
  - SIFT can extend this to find the instance with changing viewpoint/scale/illumination and rotation.
- What happens when want to find similar object that are not the same?
  - NN for the saving!

history

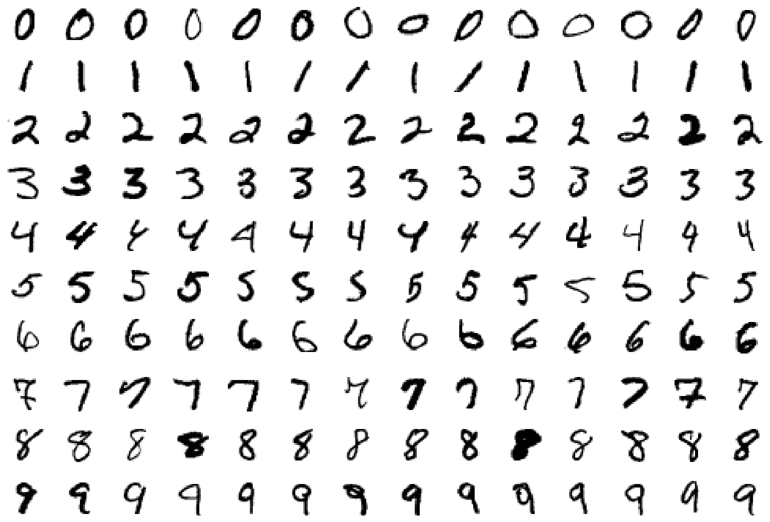
perceptron

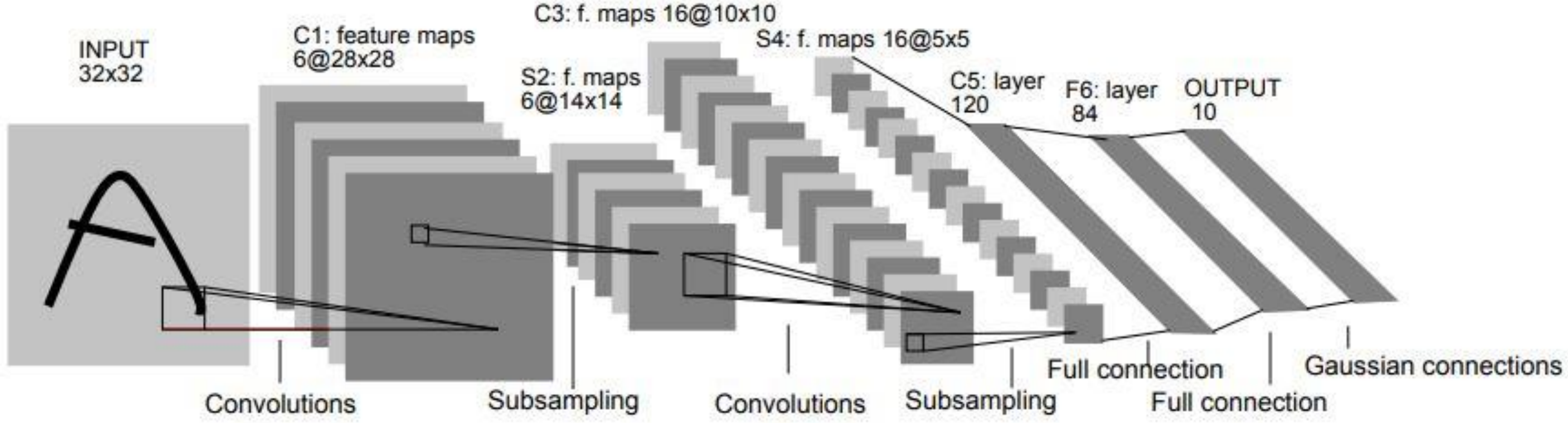
- The basic building block of all NN.
- First introduced in 1958 at Cornell Aeronautical Laboratory by Frank Rosenblatt.
- We will talk more about it in a moment…



MNIST + LeNet-5

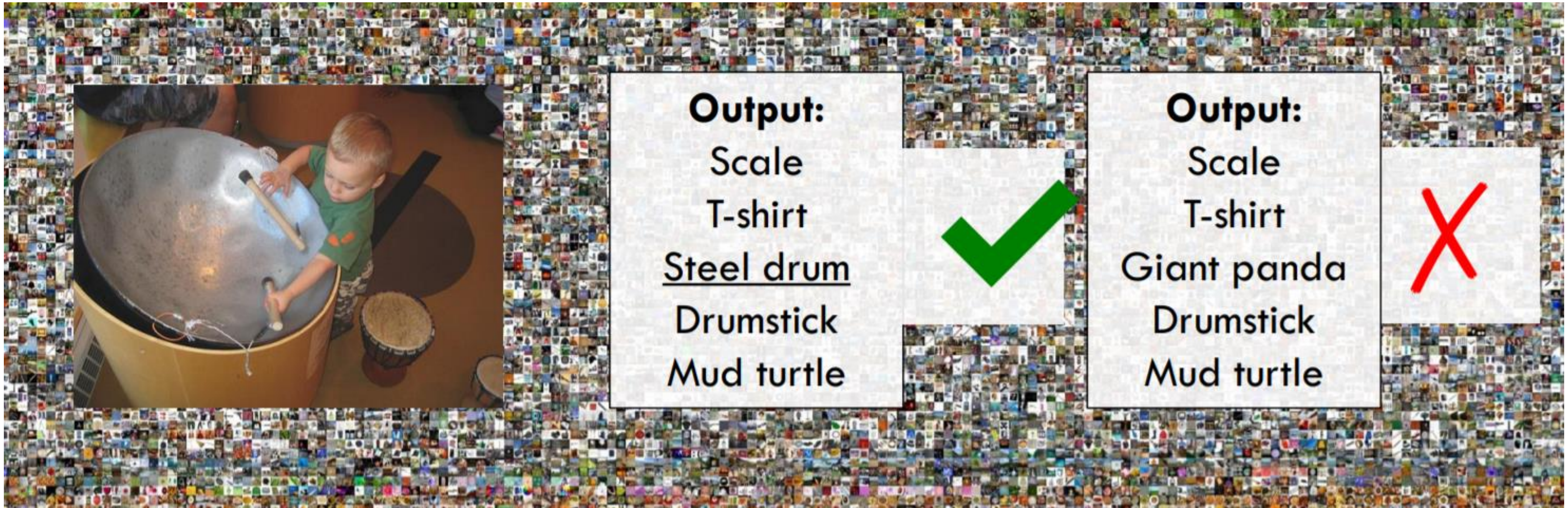
- MNIST is a large dataset of handwritten digits used in training of LeNet-5.
- LeNet-5 is the first known NN to solve a major computer vision problem:
  - Classifies digits, was applied by several banks to recognize hand-written numbers on checks.
  - Used 7 trainable layers with a total of **60K** params (sounds a lot?).
  - Yann LeCun at el., 1998, 23000 citations.



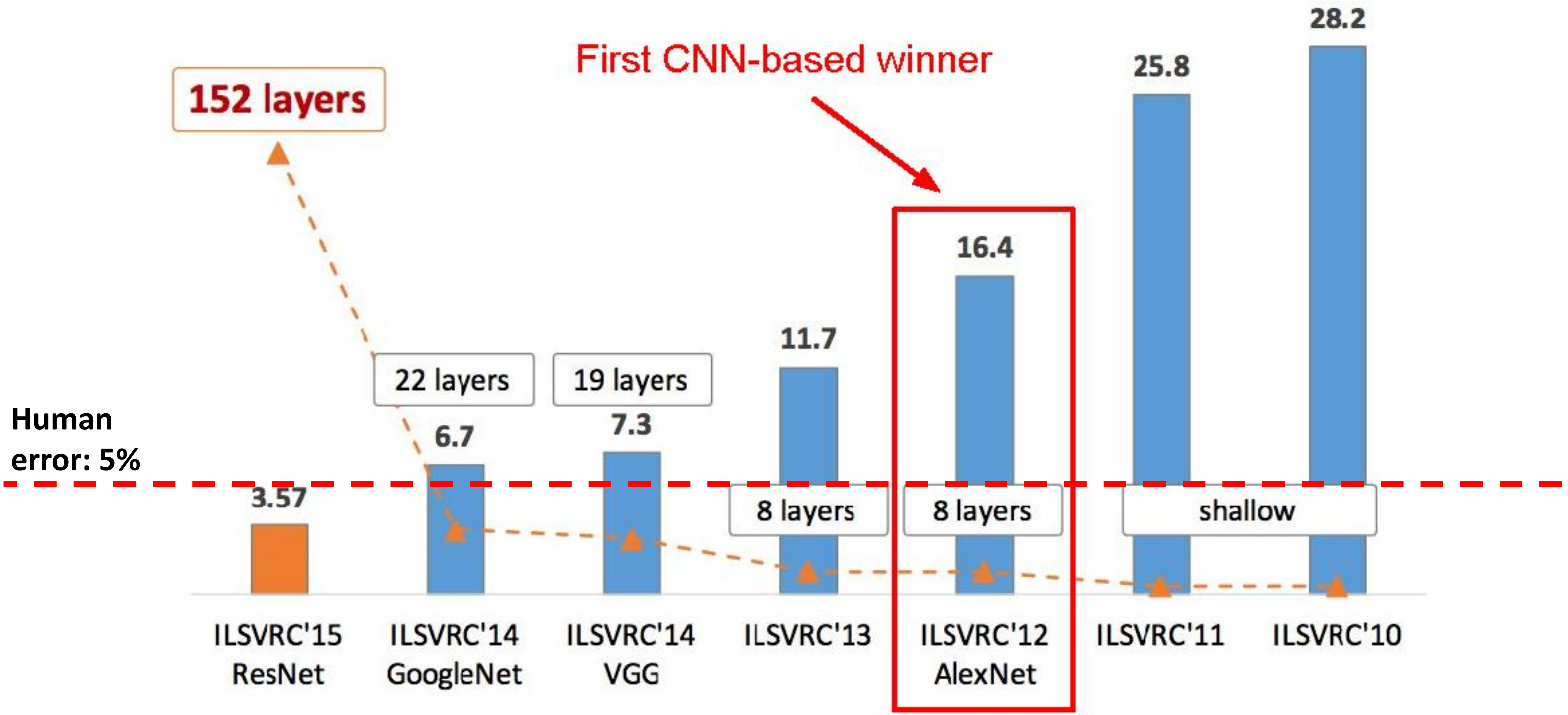


IMAGENET Large Scale Visual Recognition Challenge (ILSVRC)

- ImageNet is an image database most known for its ILSVRC challenge, and specifically for the image classification contest:
  - 1000 object classes
  - 1,431,167 images
  - Winner has the minimum mean labeling error out of 5 gausses for a given unknown test set.



ILSVRC winners

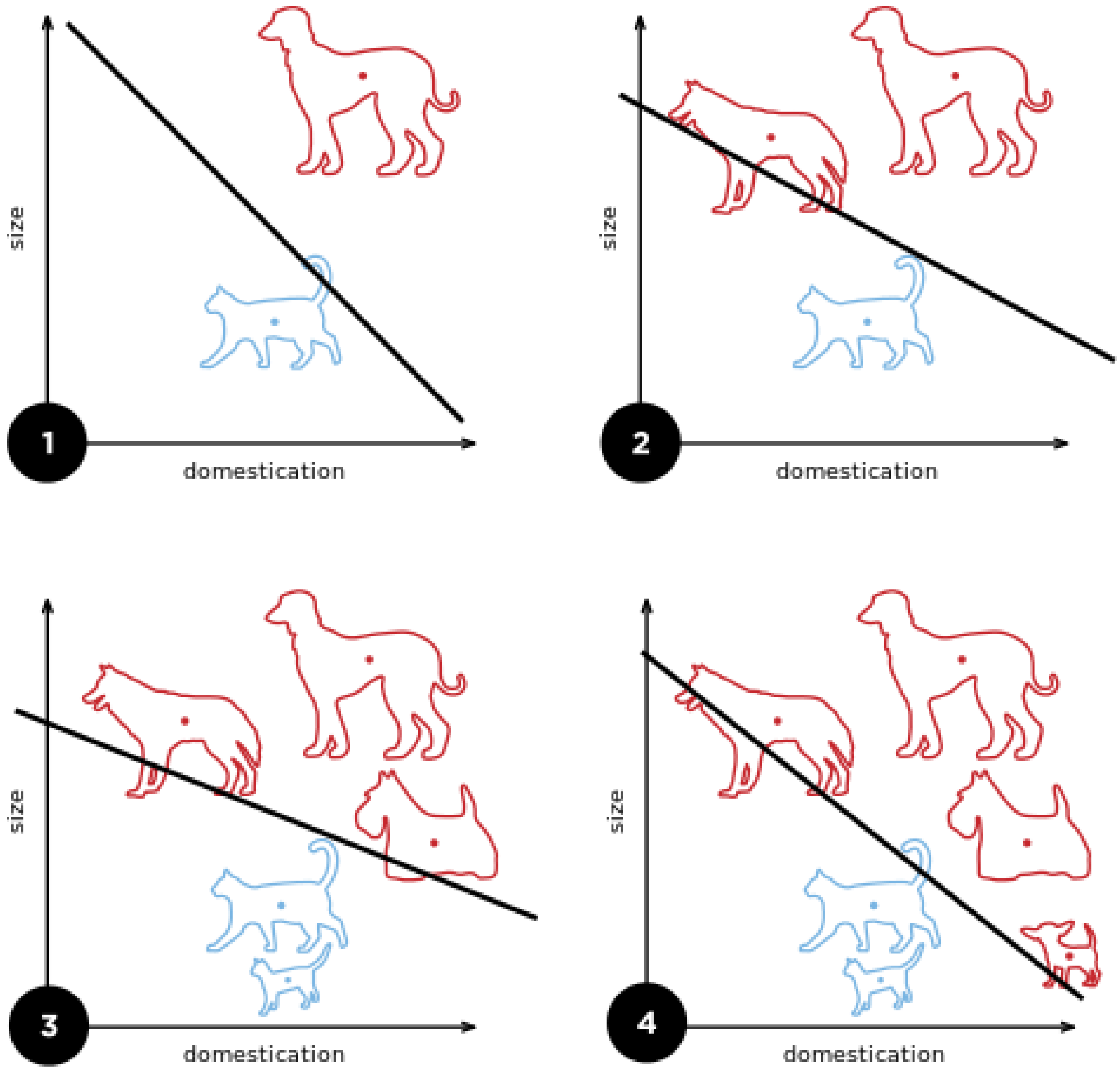


The classification problem

- Let’s first try to solve it with a perceptron.

Perceptron

- the **perceptron** is an algorithm for supervised learning of binary classifiers.
  - The perceptron determines a hyperplane separator which is determined by a set of weights (W).
  - A feature vector is the representation of the object to be classified which the perceptron receives as input ($\boldsymbol{x}$).
- The weights (W) determine the separator are what we need to learn in order to optimize the classification.



hyperplane

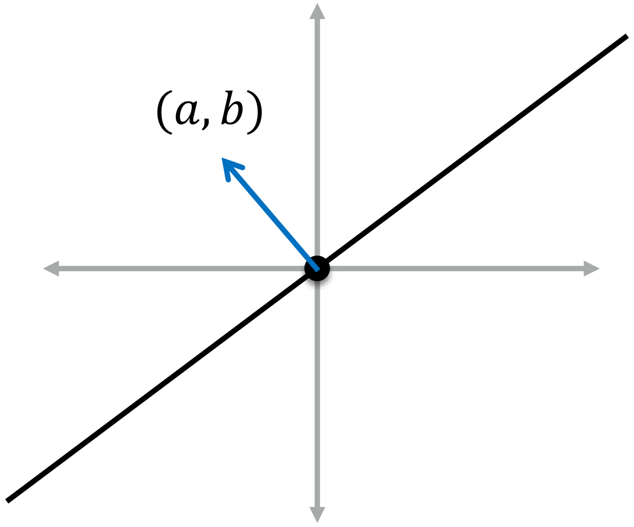
- Paramtrization of a line in 2D:

$$ax + by + c = 0$$

  - if $c = 0$:

$$ax + by = 0 \leftrightarrow (a, b) \cdot (x, y) = 0 \leftrightarrow (a, b) \perp (x, y)$$

    - (a, b) defines the normal to the line



hyperplane

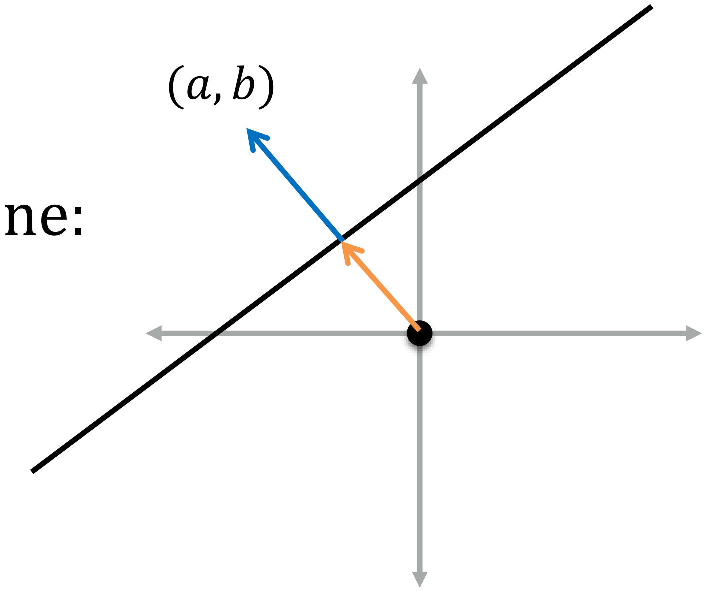
- Paramtrization of a line in 2D:

$$ax + by + c = 0$$

  - if $c = 0$:

$$ax + by = 0 \leftrightarrow (a, b) \cdot (x, y) = 0 \leftrightarrow (a, b) \perp (x, y)$$

    - (a, b) defines the normal to the line
  - if $c \neq 0$:
    - This is the **bias** factor.
    - Defines the distance of $(0,0)$ from the line:
      - Point-line distance: $\mathrm{d} = \frac{|ax+by+c|}{\sqrt{a^2+b^2}}$
      - $bias = \frac{|c|}{\sqrt{a^2+b^2}}$

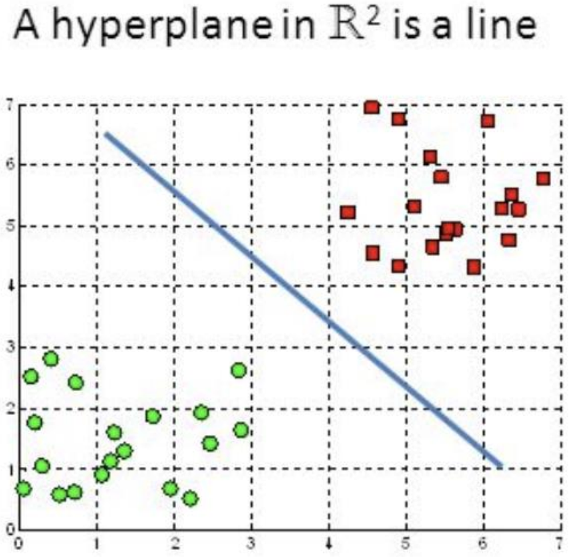


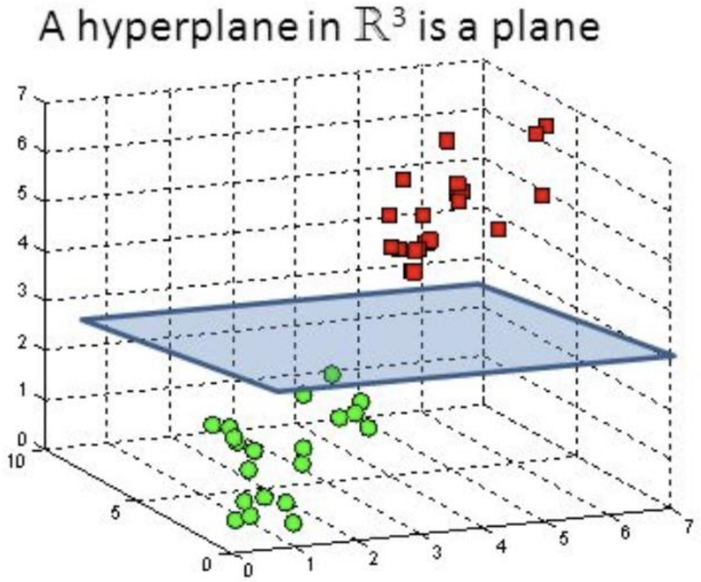
hyperplane

- This is the same for 3D representation of a plane as well:

$$ax + by + cz + d = 0$$

  - (a, b, c) defines the normal to the plane, d defines the bias of the plane from $(0,0,0)$.
- And the same representation can be done for ND space. The ND plane is called a **hyperplane.**





hyperplane

- Writing the hyperplane representation vector vise will result the equation below:

$$[w_1 \ \cdots w_n]\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + b = w^T x + b = 0$$

- Points x above the hyperplane (in the direction of the normal) will result in $w^T x + b > 0$, and points x below the hyperplane will result in $w^T x + b < 0$.

hyperplane

- **Another option** is to write the hyperplane representation with **homogenous vectors,** this will result with the (more compact) equation below:

$$[w_1 \;\cdots w_n \;\; b]\begin{bmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{bmatrix} = w^T x = 0$$

- Points x above the hyperplane (in the direction of the normal) will result in $w^T x > 0$, and points x below the hyperplane will result in $w^T x < 0$.

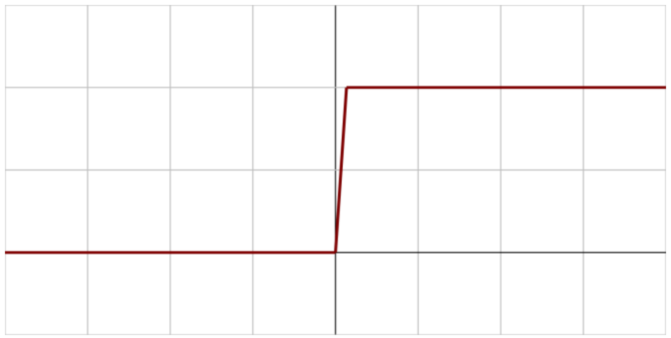
Activation function

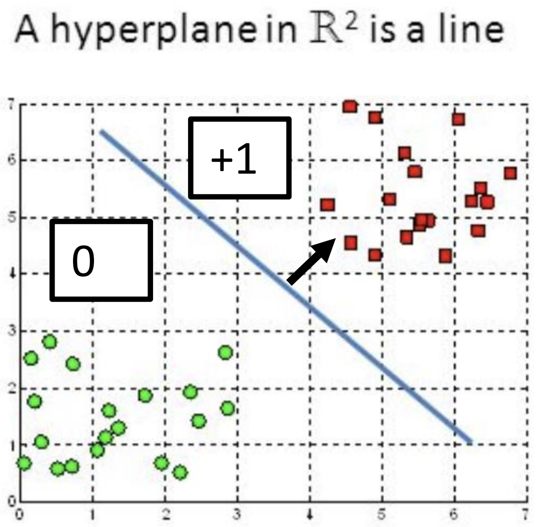
- A non-linear function $f()$ that appends the perceptron's hyperplane equation

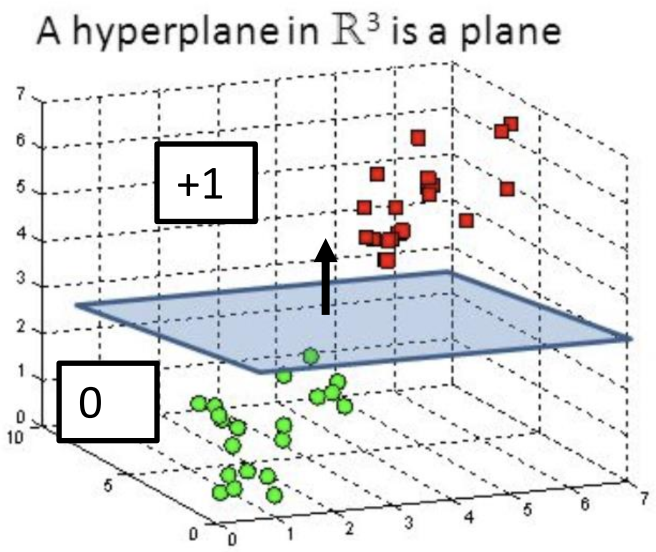
$$y = f(Wx).$$

- If we have a problem of classifying two groups with a single hyperplane, we can use a step activation function:

$$f(x) = step(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$





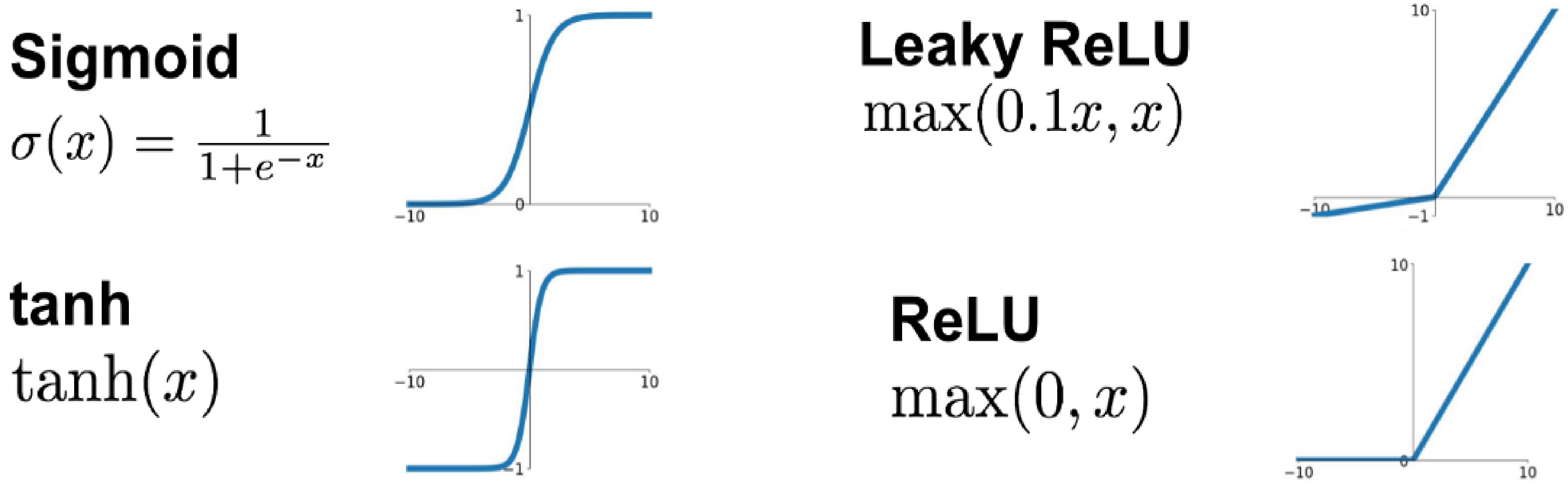


Activation function

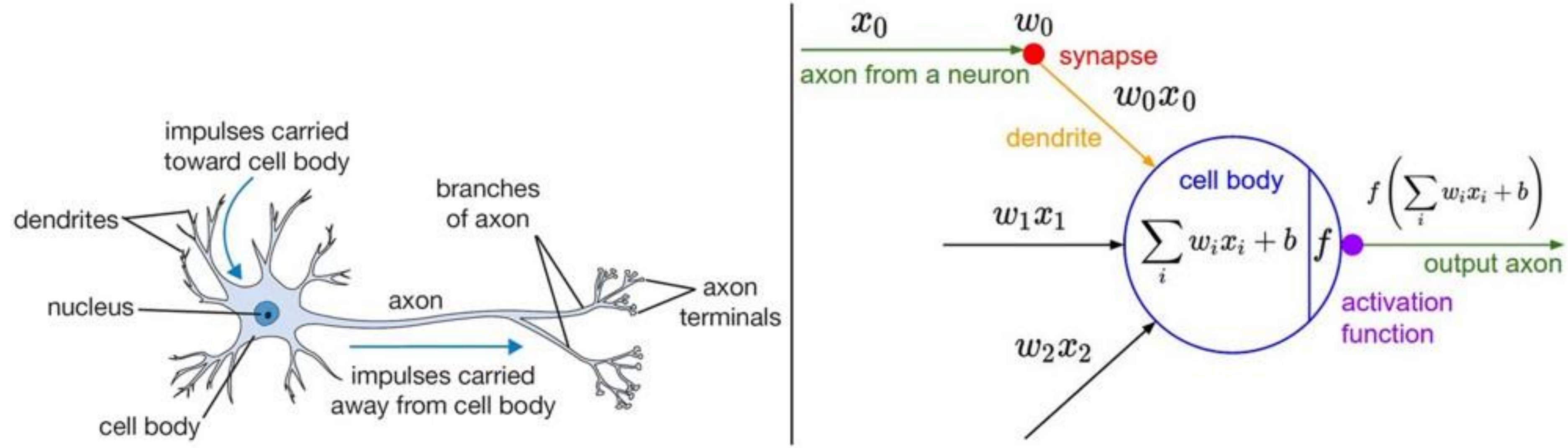
- Later we will use more common activation functions.
- One of them is the **rectified linear unit** (**ReLU**) function:

$$f(x) = \max(x, 0) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

- Other known activation functions: sigmoid, tanh, leaky ReLU.



perceptron: Inspiration from Biology

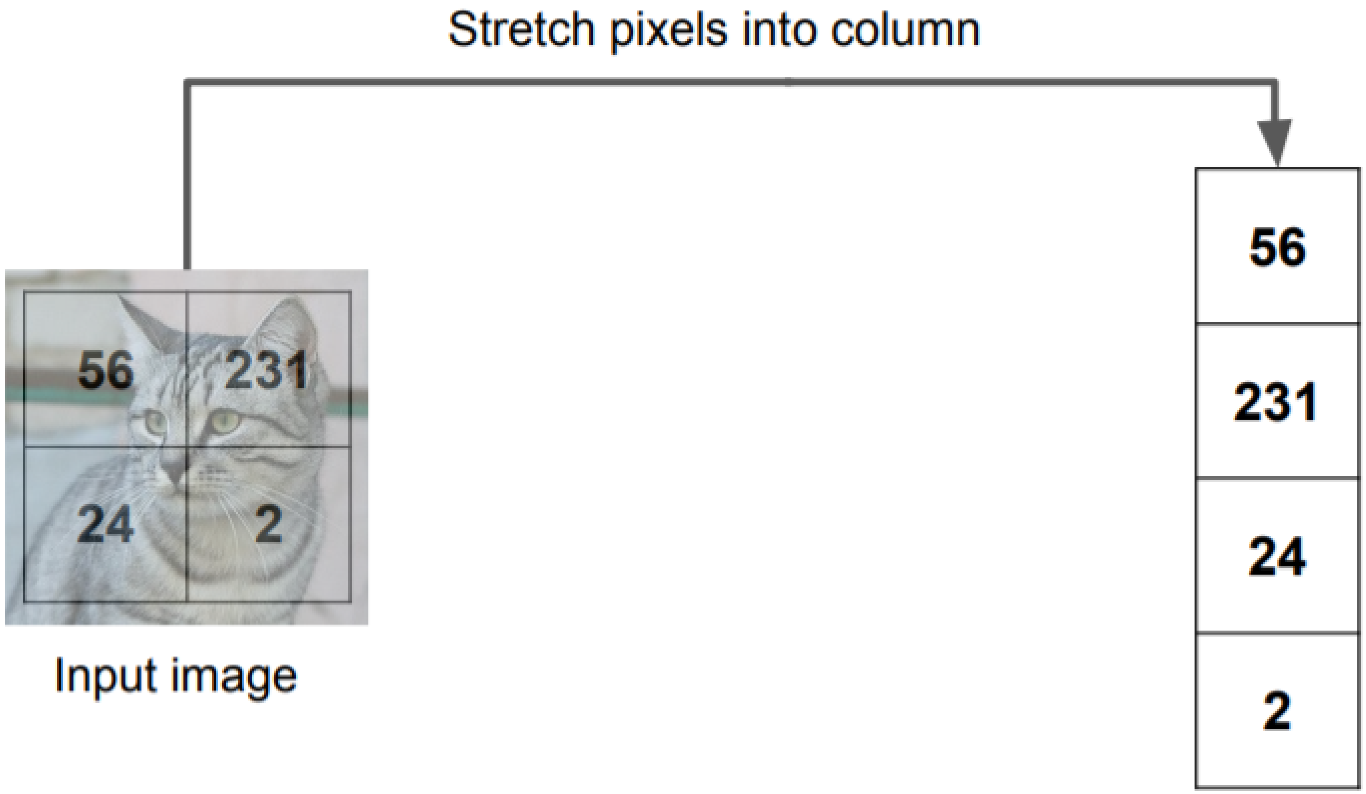


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

- Neural nets/perceptrons are **loosely** inspired by biology.
- But they certainly are **not** a proper model of how the brain works, or even how neurons work.

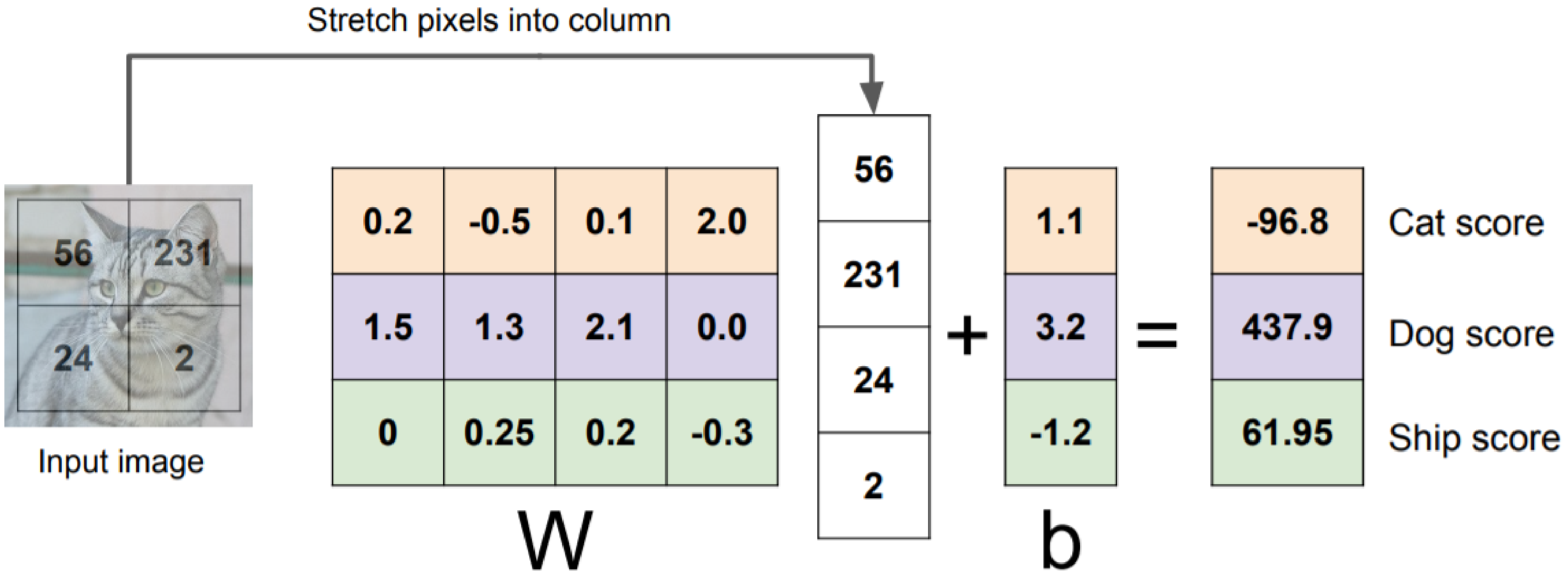
Hyperplanes and image classification

- In images, the pixels can be the input feature vector.



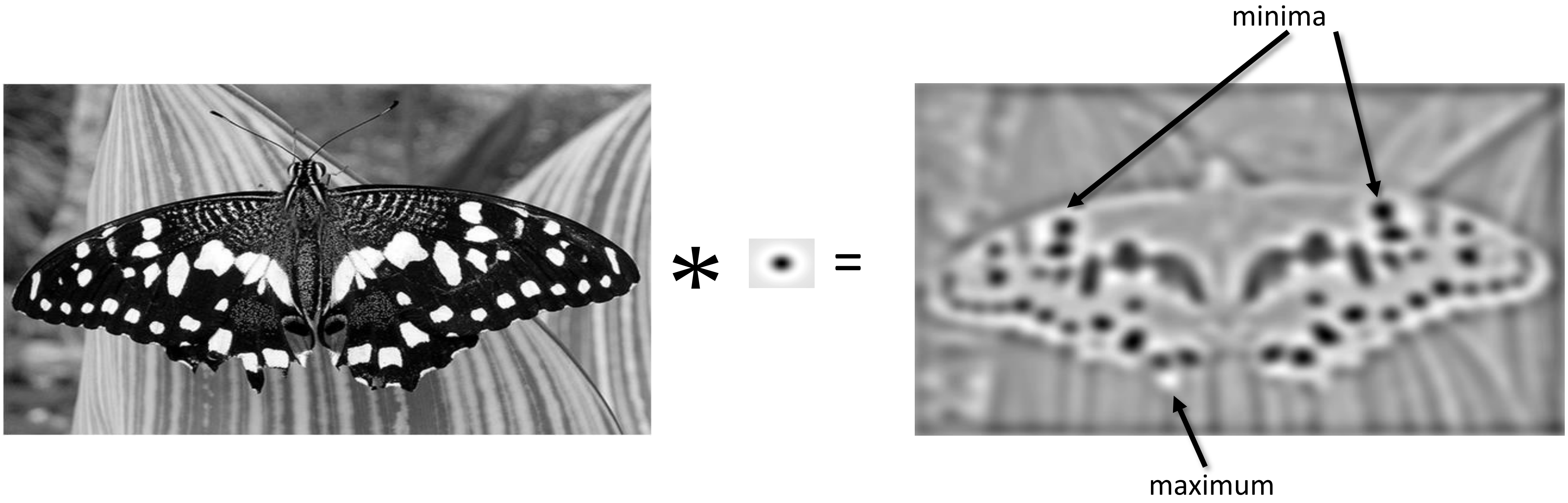
Hyperplanes and image classification

- We want to find a hyperplane in 4D space that puts all cats' vectors in one side of it, and all other images in the other side.
  - Let's assume there are 2 more classes. In total: cats, dogs and ships. Now, W is a matrix rather than a vector
  - Find 3 separating planes, one for each class.



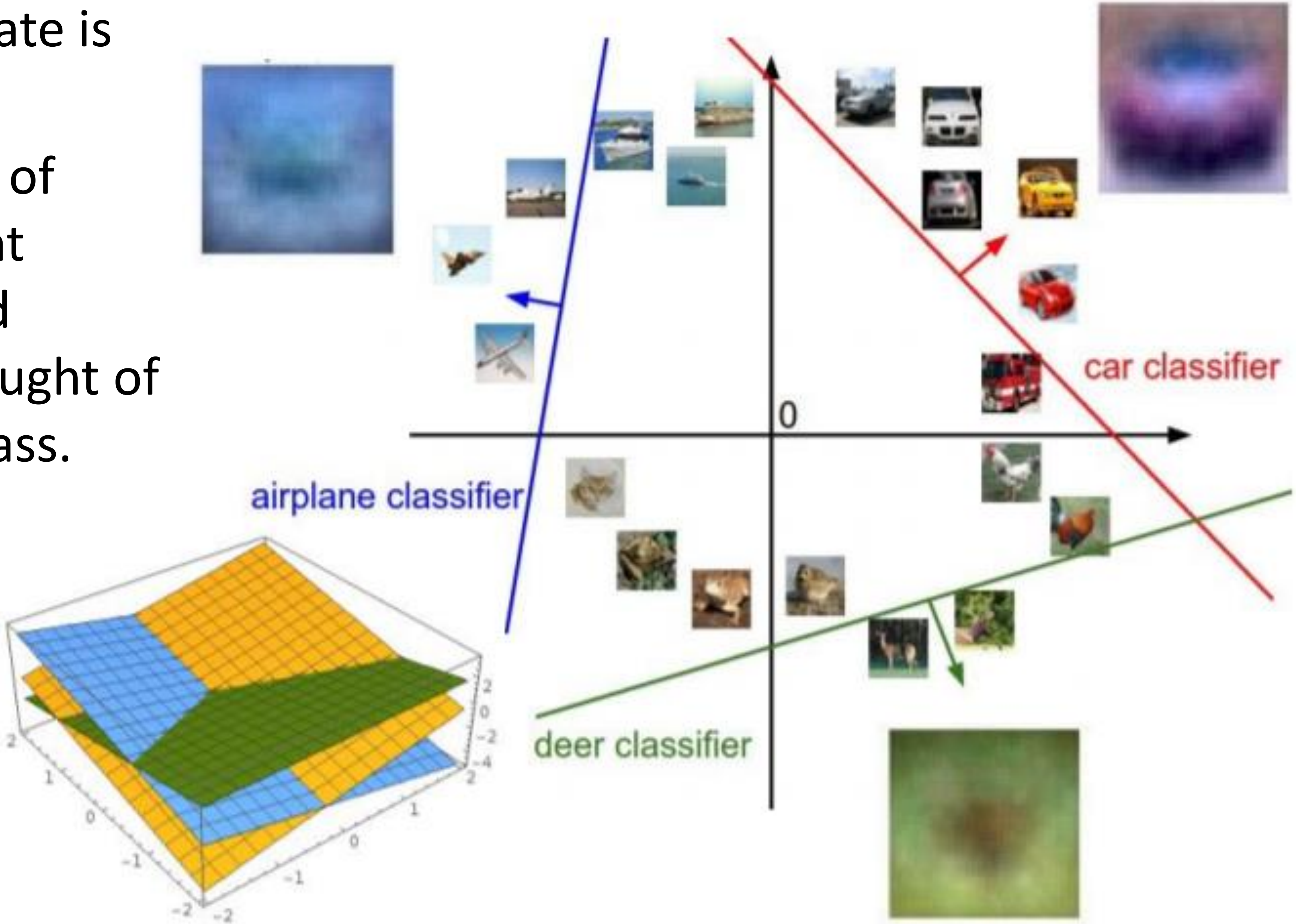
Perceptron: template matching interpretation

- We can think about the optimized weights as a template in template matching cross correlation algorithm.
  - We get a strong positive response when the template matches the image area.

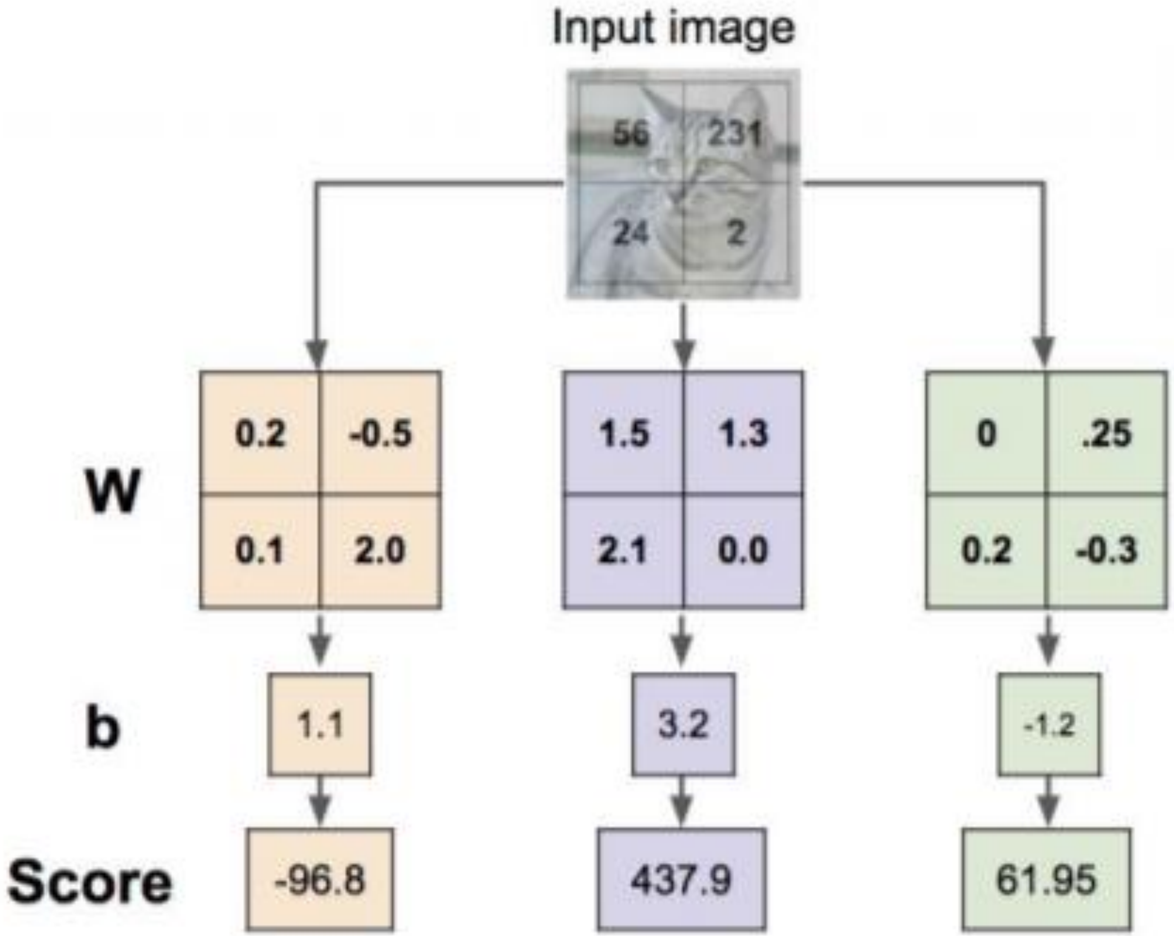


Perceptron: template matching interpretation

- In our case the template is the size of the image.
- We can see examples of templates for different groups- the optimized template can bee thought of as the mean of the class.



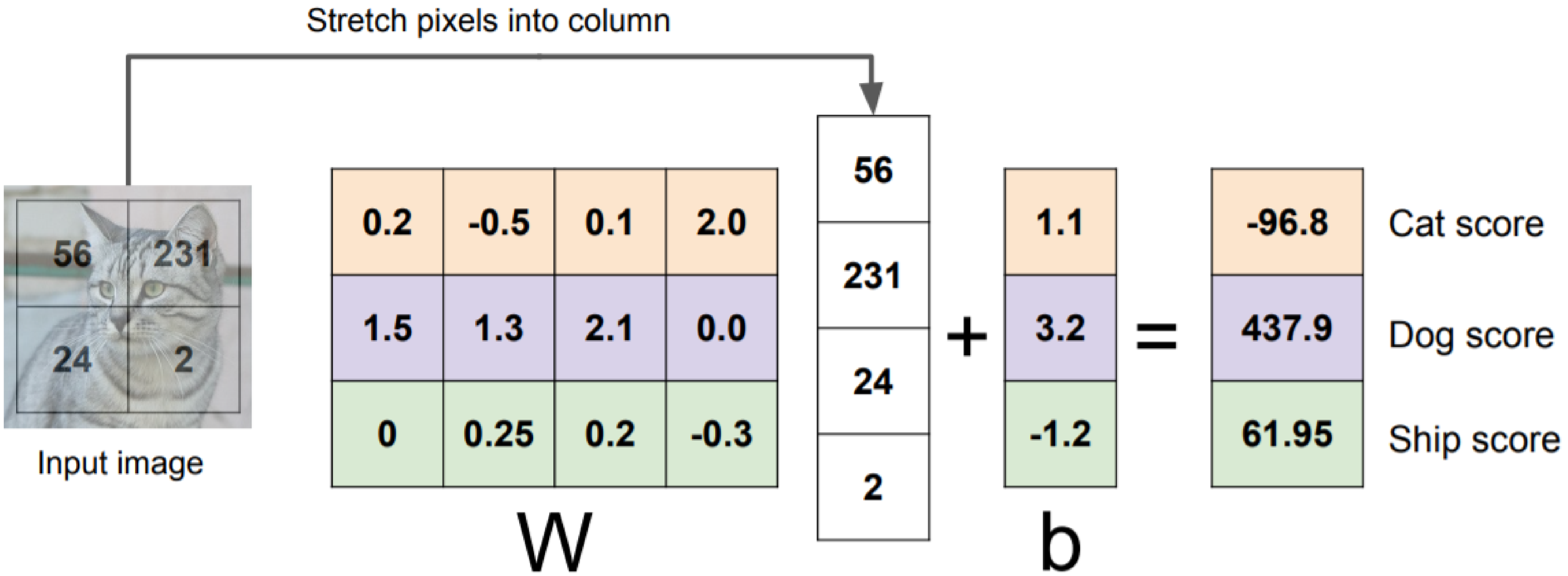
Perceptron: template matching interpretation



optimization

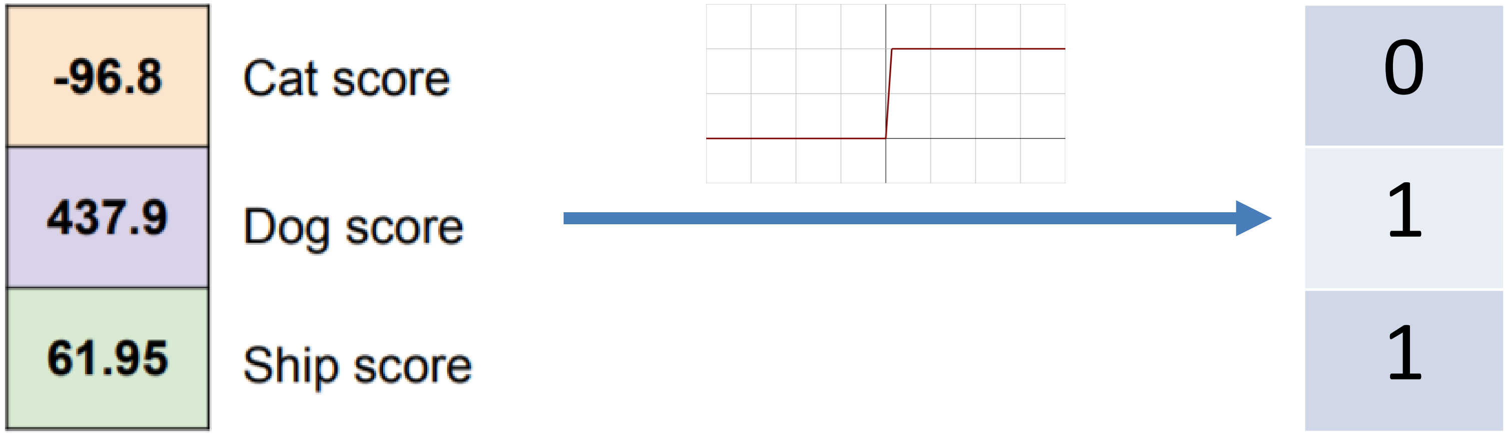
Optimizing the weights

- We have this results for each possible label.
- which is the best result currently? Which should be the best result?



Optimizing the weights- first try

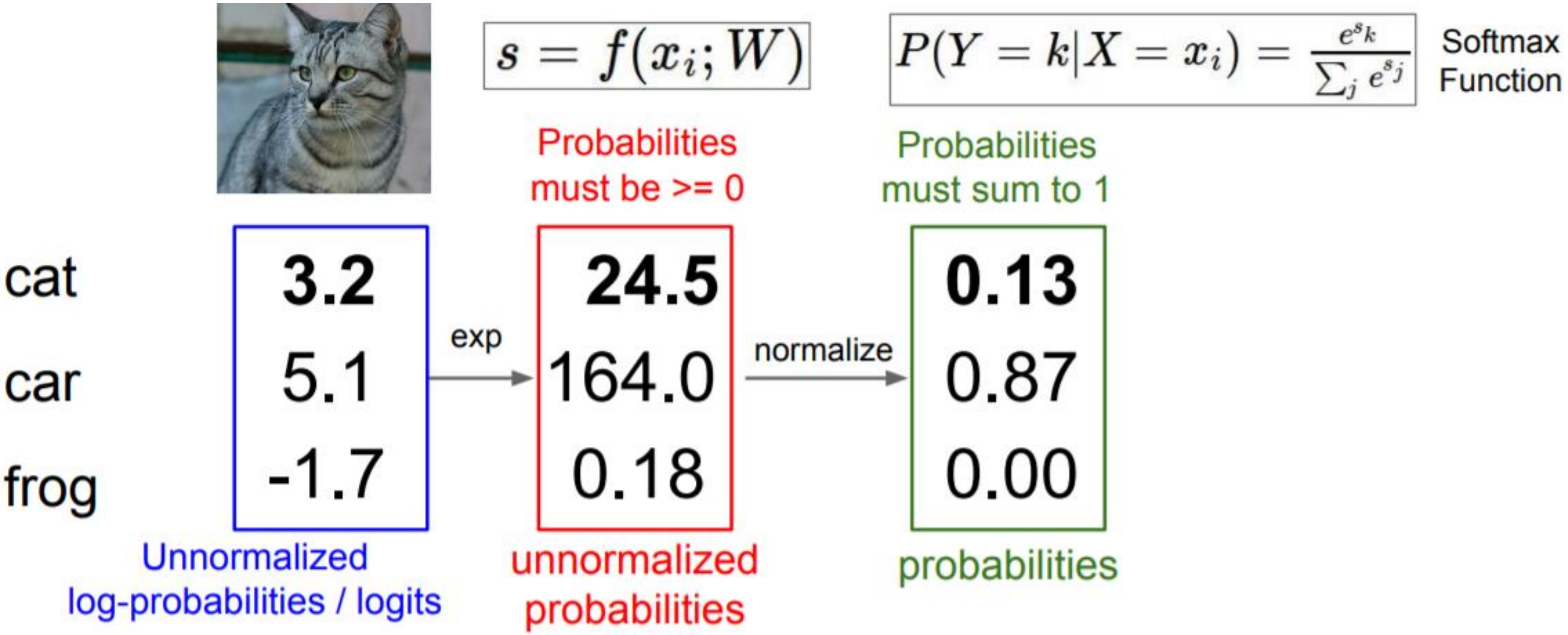
- We have this results for each possible label.
- which is the best result currently? Which should be the best result?
  - Let’s use our step activation function from before.



- Can’t tell us which class is better… not good enough.
  - We need a way to quantify the results as more/less likely.

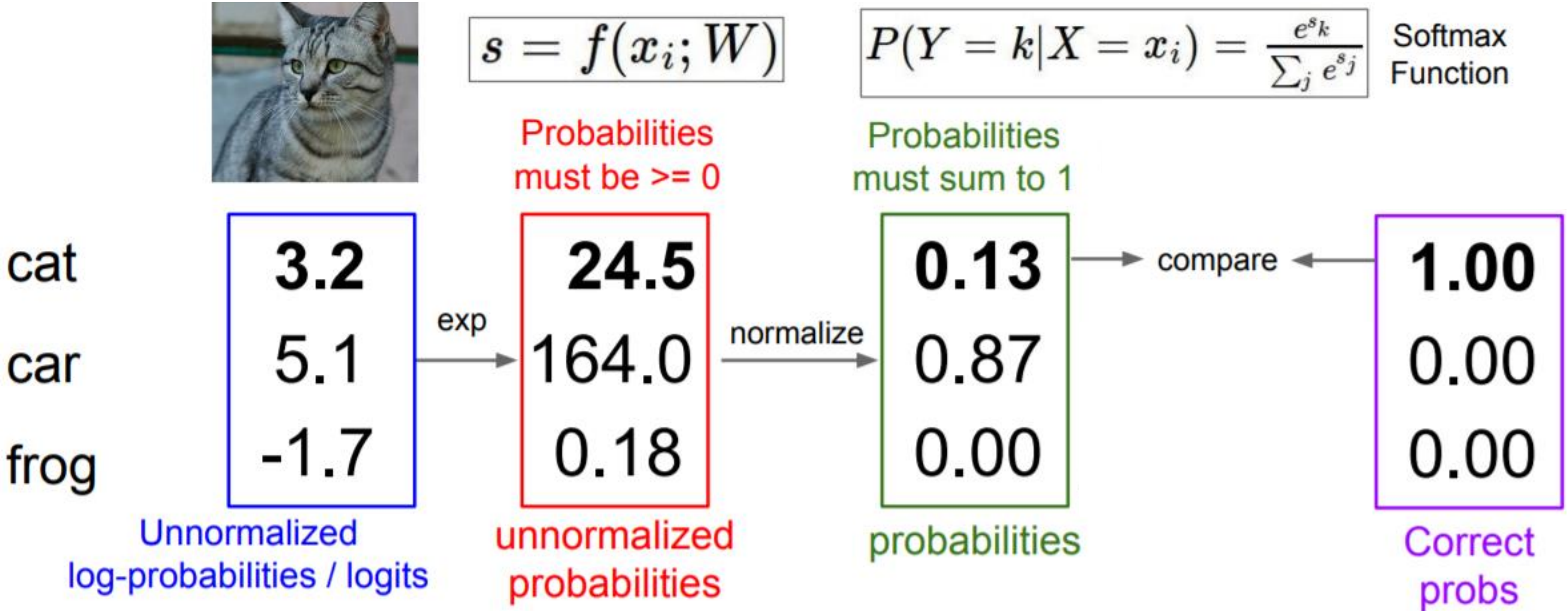
Softmax layer

- The softmax layer normalizes all the results so that you get a percentage of correctness for each label.
- The softmax is usually added as the last layer in a NN to normalize the results instead of an activation function.



Cross entropy loss function

- We need to define an error of the given probabilities and the correct (wanted) probabilities.
- A known loss function for this problem is called **cross entropy loss.**



Cross entropy loss + softmax

- The cross entropy of the distribution q (output results) relative to a distribution p (wanted results) over a given set is defined as follow：

$$L_i = -\sum_{j \in labels} p(j) \log q(j)$$

$$\xrightarrow[]{\begin{cases} p(j) = 1 \text{ if } j = y_i \text{ (right label)} \\ p(j) = 0 \ \forall\ j \neq y_i \end{cases}}$$

$$L_i = -\log q(y_i)$$

$$\xrightarrow{\text{plug in with softmax classifier}}$$

$$L_i = -\log q\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Total loss

- This L_i is the loss of a single **given** input image x_i.
- Let's say we have all possible images in the world, so the **total loss** will be:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i$$

  - A mean of all possible losses, where N is number of images.
- We want to find the best W that minimizes L.
- How do we do this?

Total loss

- This L_i is the loss of a single **given** input image x_i.
- Let's say we have all possible images in the world, so the **total loss** will be:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i$$

  – A mean of all possible losses, where N is number of images.
- **We want to find the best $\boldsymbol{W}$ that minimizes $\boldsymbol{L}$.**
- How do we do this?
  – Derive over W: $\nabla_W L$

Finding the best W

- How do we do this?
  - Derive over W: $\nabla_W L$
- Problems:
  - We don't have all images, and even if we do, it will take forever…
  - No one said L is a convex function.
  - It's sometimes hard to compute the analytic derivative of the function L for all possible $\boldsymbol{x}$ in order to naively find all extremum points.
- An approximate solution to find best W is called **mini-batch gradient descent.**

Mini-batch Gradient descent

Finding the best W

- How do we do this?
  - Derive over W: $\nabla_W L$
- Problems:
  - **We don't have all images, and even if we do, it will take forever…**
  - No one said L is a convex function.
  - It's sometimes hard to compute the analytic derivative of the function L for all possible $\boldsymbol{x}$ in order to naively find all extremum points.
- An approximate solution to find best W is called **mini-batch gradient descent.**

Mini-batch

- In mini-batch gradient descent we take only a small subset of images and compute their average loss:

$$\tilde{L} = \frac{1}{\tilde{N}} \sum_{i=1}^{\tilde{N}} L_i$$

  – A mean of the subset losses, where $\tilde{N}$ is the size of images subset.
- This approximation of the loss function is **faster to compute but less accurate.**

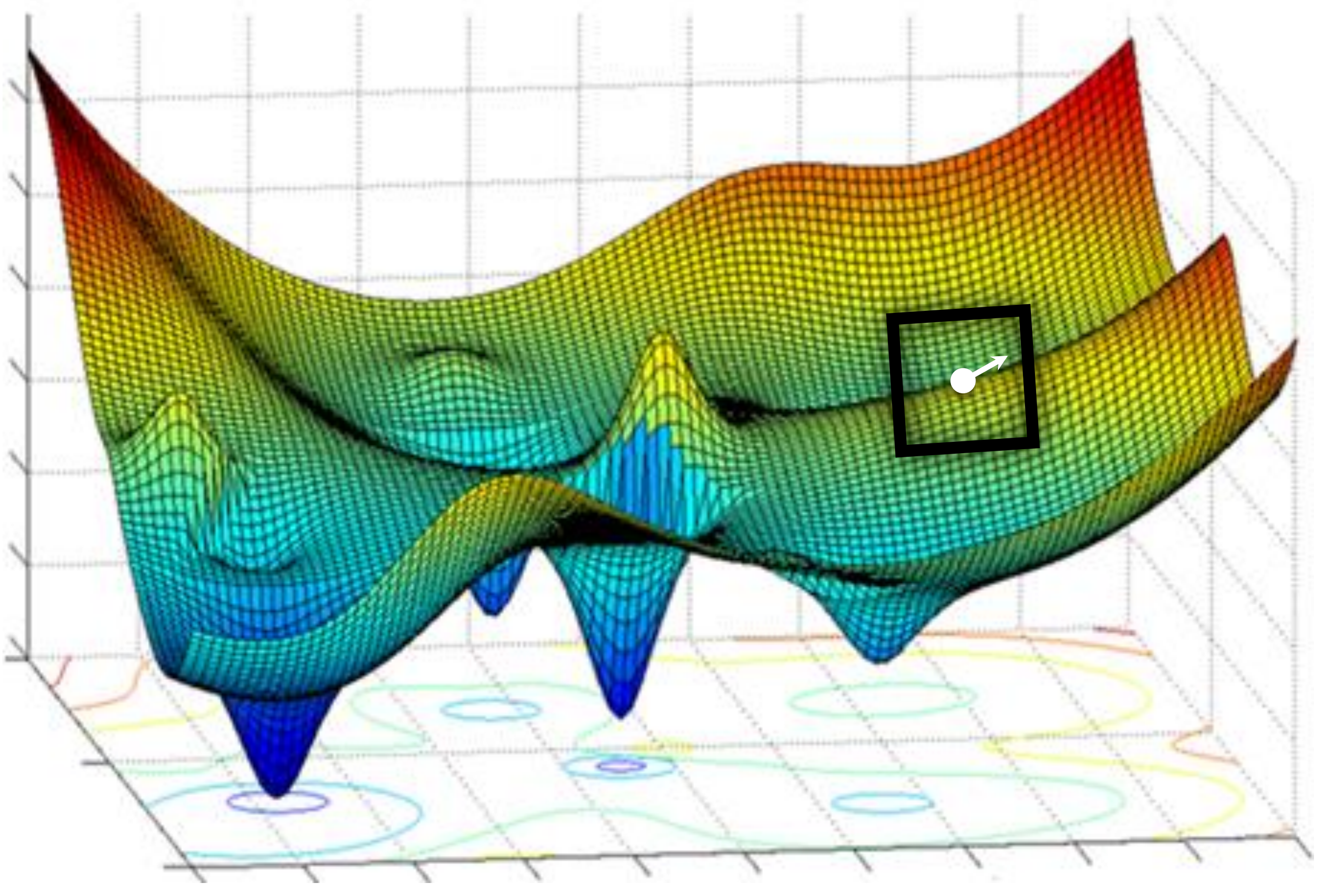
Finding the best W

- How do we do this?
  - Derive over W: $\nabla_W L$
- Problems:
  - We don’t have all images, and even if we do, it will take forever…
  - **No one said L is a convex function.**
  - **It’s sometimes hard to compute the analytic derivative of the function L for all possible x in order to naively find all extremum points.**
- An approximate solution to find best W is called **mini-batch gradient descent.**

What is a gradient?

- describes the direction and magnitude of the fastest increase around a point $\boldsymbol{x}$.
- Example: gradient of a function of 2 variables:

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \left[\frac{\partial f(\boldsymbol{x})}{\partial x}, \frac{\partial f(\boldsymbol{x})}{\partial y}\right]$$

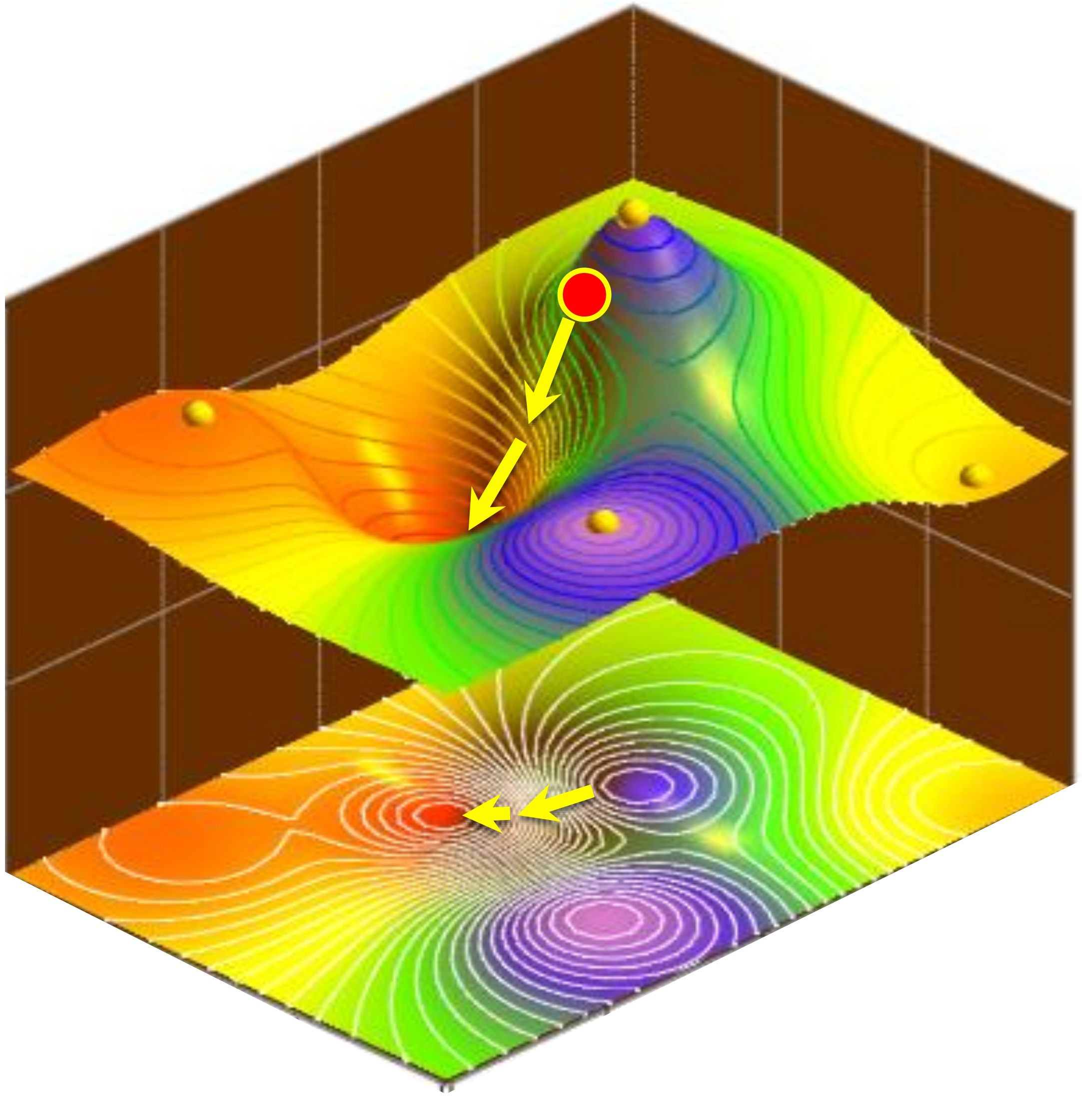


Gradient descent

- An iterative algorithm for finding local minima of functions.
- starts at a random point and moves step-by-step in the direction and proportional magnitude of the negative of the gradient of the point he is currently in:

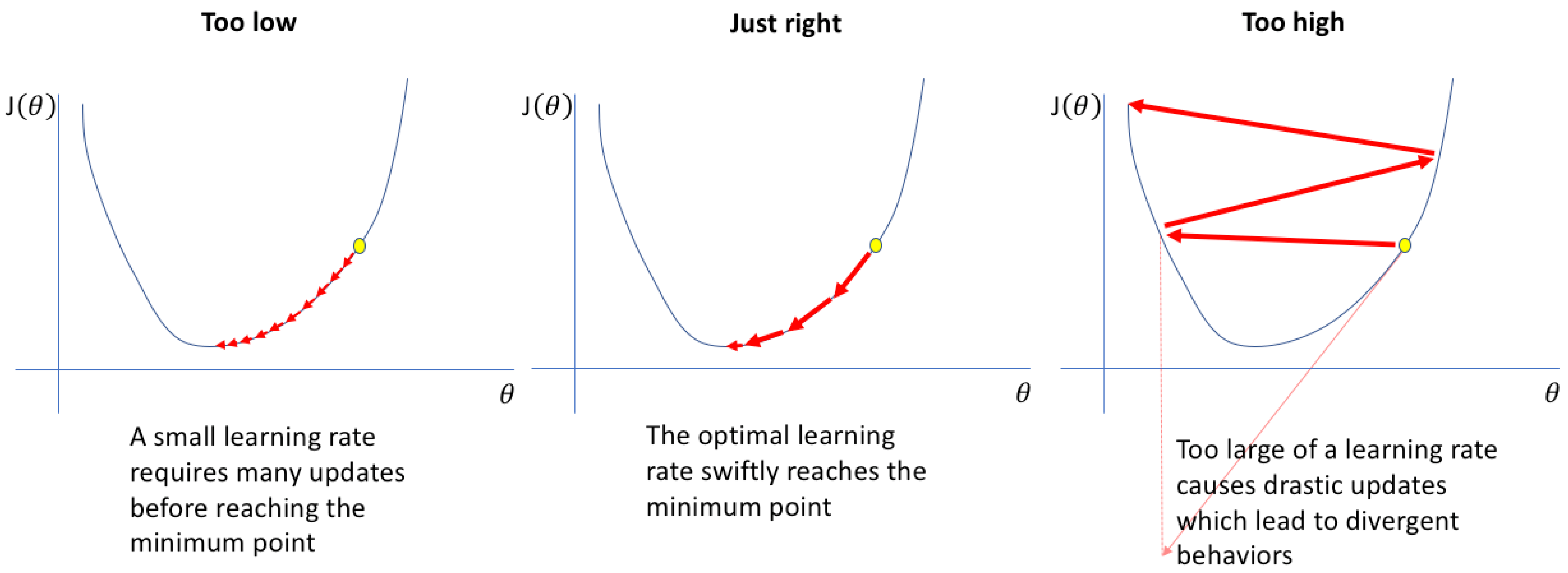
$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \eta \cdot \nabla f(\boldsymbol{x}_n)$$

  – “proportional magnitude” == step size η.
- In “proper use” this algorithm converges to a local minimum which is depended on the starting point.



Gradient descent- step size

- Also known as **learning rate**.
- Choosing the right step size is important.
- This is known as a **hyperparameter:** an unknown variable that is configured by the user (unlike the weights W which the system "learns").

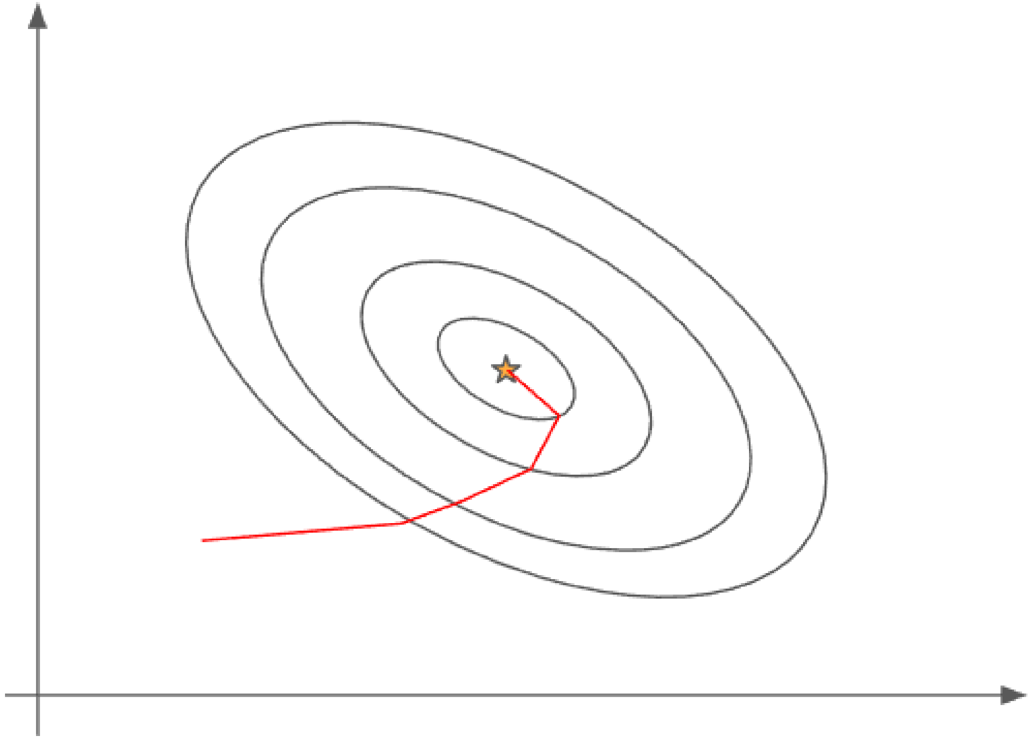


Gradient descent- local minima

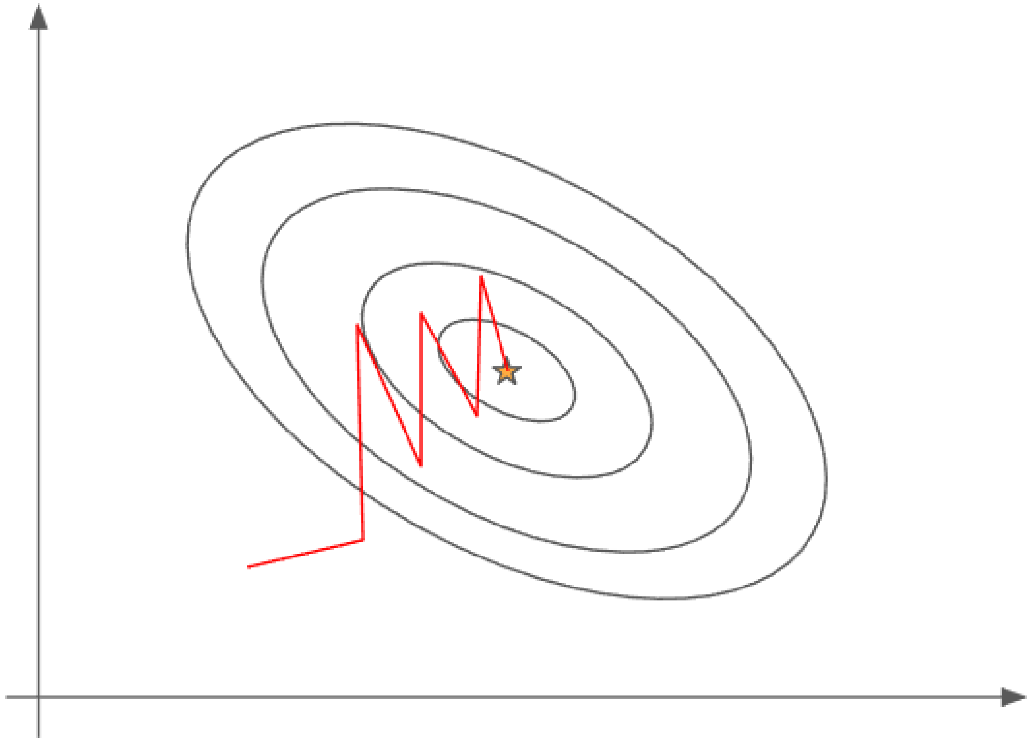
- An iterative algorithm for finding local minima of functions.
- we can initiate this procedure several times from several random staring points and take the minimum of all output minimum points- this way we can get a better result.

Mini-batch gradient descent

- Combining the two methods is called **Mini-batch gradient descent.**
- Almost always mis-called **stochastic gradient descent (SGD)**…
  - This is the name only if the batch size is 1.



Gradient Descent



Stochastic Gradient Descent

Testing the results

Testing the results

- NN frameworks are build on learning from examples, so the data is important.
- Usually we split the data to 3 different datasets:
  - Train: to train the weights.
  - Validation: test the resulted NN with specific architecture on unseen data.
  - Test: compare different types of NN architectures/ change in hyperparameters which are not learned.
- If we don't have a validation dataset, we will eventually change the architecture/ hyperparameters so they will fit the test data- basically learning on the unseen dataset- **not good**.

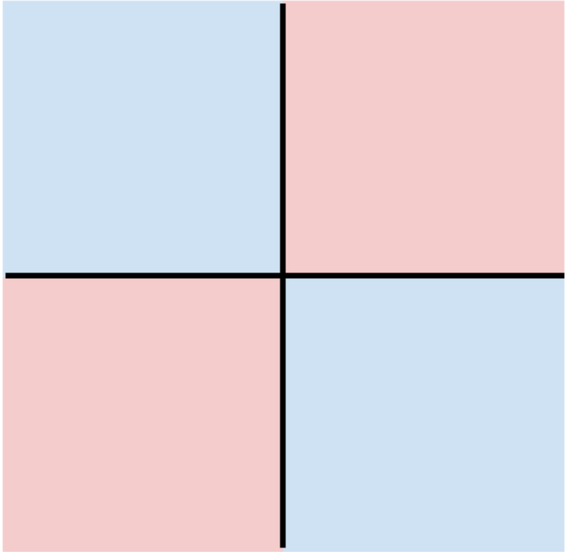
Multi-layer perceptron

Multi-layer perceptron

- Perceptron plane separation is not enough for all data sets- some are not linearly separable.
- multi-layer perceptron (MLP), or in a more common name- **neural network**, is a better approach to try to handle this data.

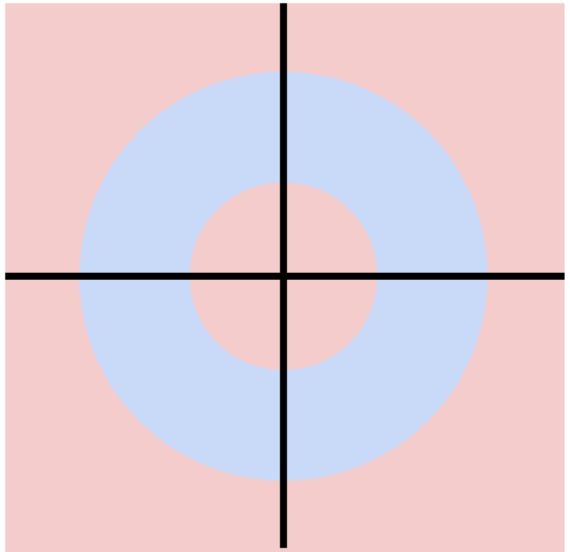
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



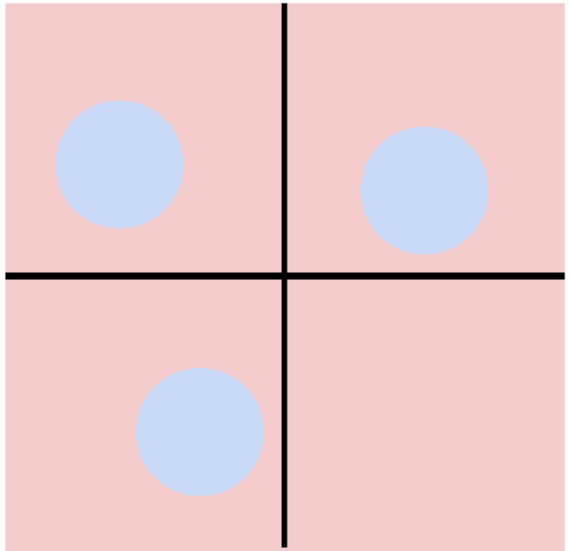
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

Class 2:
Everything else



CIFAR10 dataset

- CIFAR10 (Canadian Institute For Advanced Research) is a known dataset of 10 classes of small images.
- 32X32X3=3072 DOFs in this problem, and images vary a lot. This is not possible to linearly separate.

10 classes

50,000 training images
each image is **32x32x3**

10,000 test images.

Multi-layer NN: intuition

- We can use the data of **all** the responses to all “templates” of weights from the first layer to better represent the result.
- In this way, instead of one best fit for a template, we can use all the responses to all templates of the first layer to learn a better classification.
- This is also correct for any number of layers in an NN.

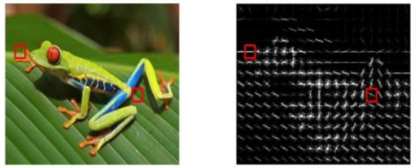
(**Before**) Linear score function: $f = Wx$

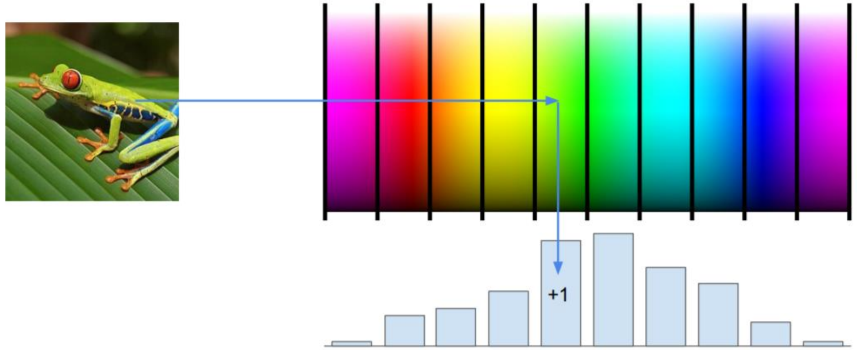
(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
or 3-layer Neural Network

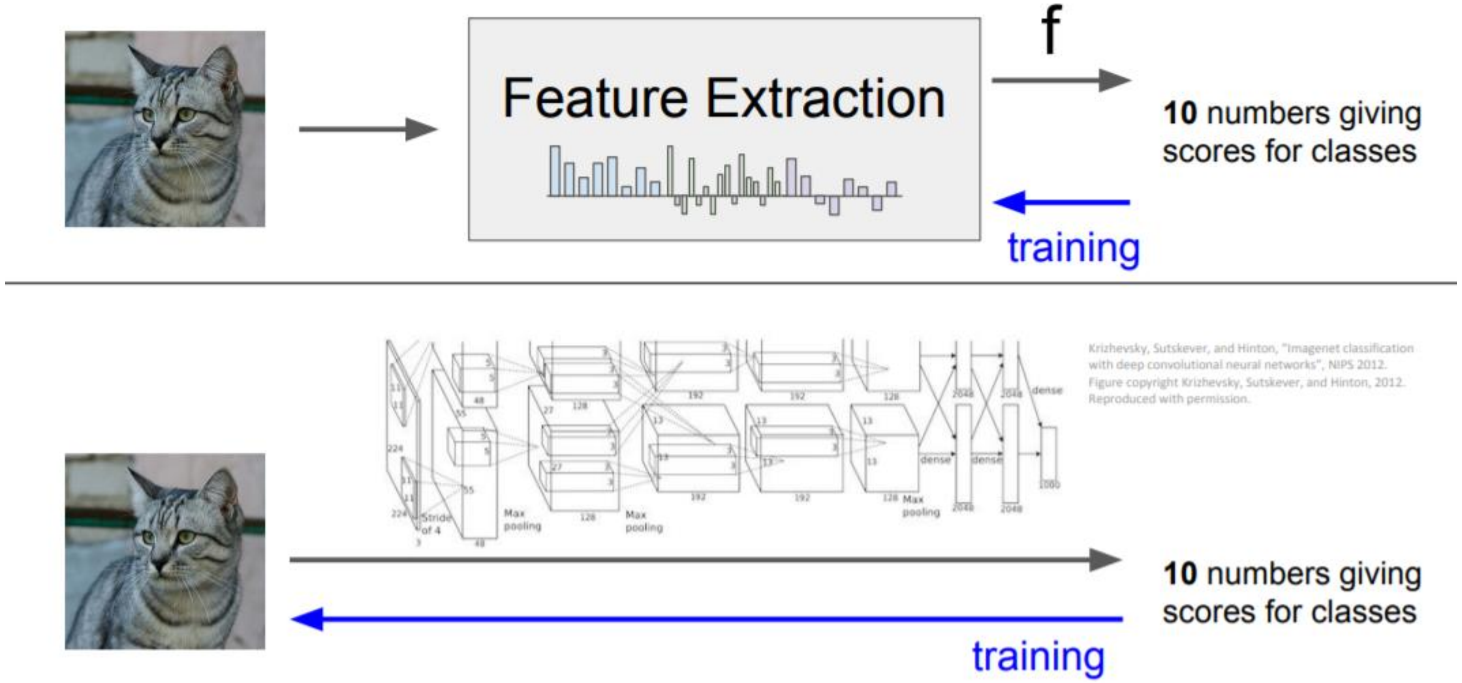
$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Multi-layer NN: intuition

- Before: human “hand engineered” features as input into a machine learning (ML) framework.
  – Examples of features we’ve seen: SIFT, HOG, color histograms.
- Now: the NN finds best features.





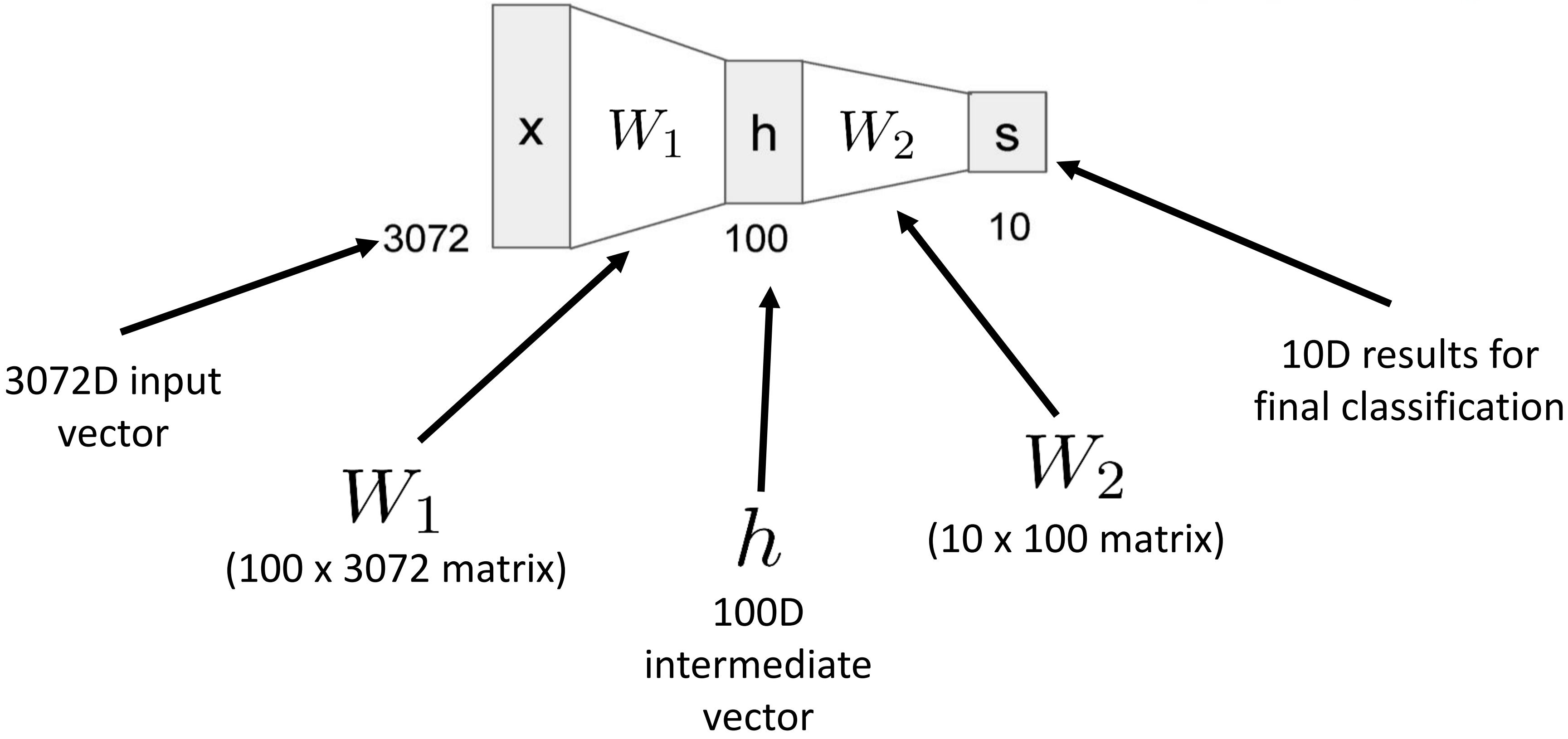


Multi-layer NN

- 2-layer NN example: Learned 100 different templates in the first layer and input them into a second layer for final classification.

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$



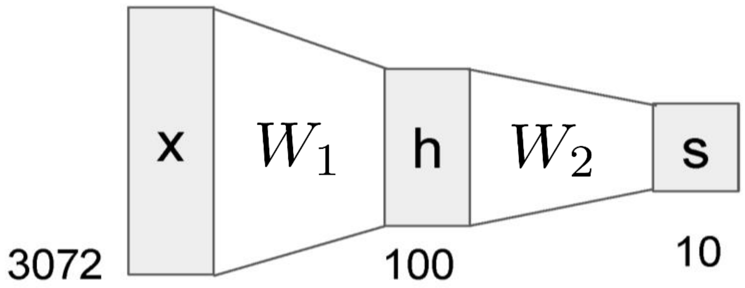
Multi-layer NN

- Total number of weights to learn:

  3,072 x 100 + 100 x 10 = 308,200

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$



Multi-layer NN

- What happens if we remove the non-linear activation?

$$f = W_2 \max(0, W_1 x)$$

Multi-layer NN

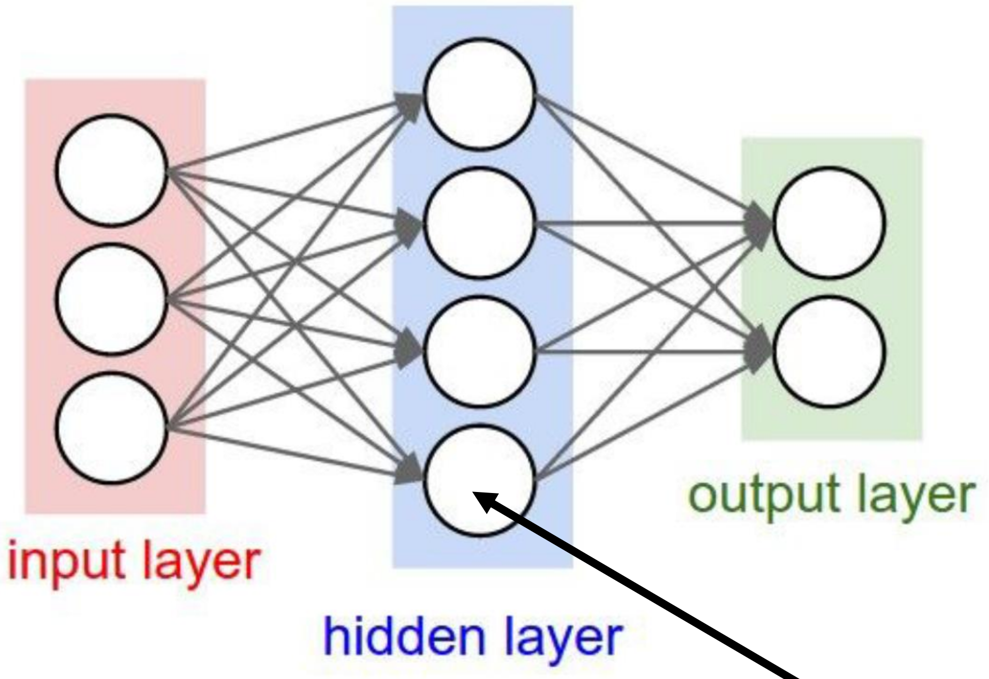
- What happens if we remove the non-linear activation?

$$f = W_2 \max(0, W_1 x) \rightarrow W_2 W_1 x = \widetilde{W} x$$

- We've gotten a linear separator again… not good.
- Remember the activation function!

Neural network architecture

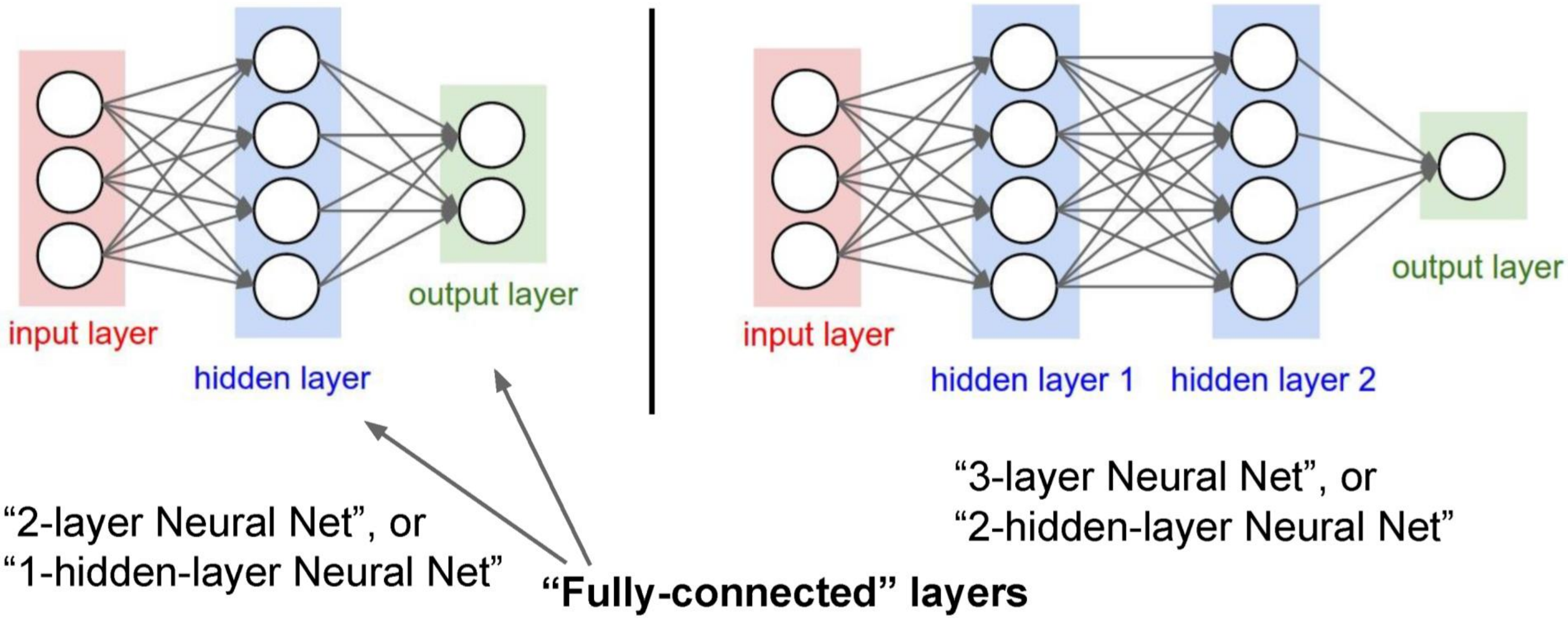
- Computation graph for a **2-layer neural network.**
  - Only count layers with tunable weights (so don’t count the input layer).
  - Each layer is built from perceptrons: weights + activation function.



One Neuron/ perceptron

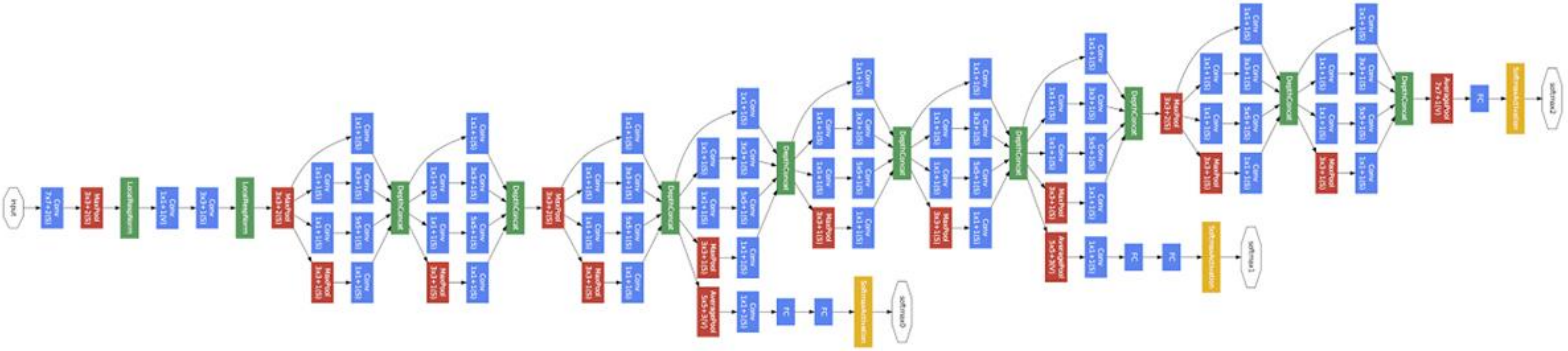
Neural network architecture

- **Deep** networks typically have many layers and potentially millions of parameters.
- **Fully connected layer** is a layer in which all inputs are multiplied for each perceptron with different weights. (this is what we saw until now).



Neural network architecture

- Example of a deep NN: Inception network (Szegedy et al, 2015)
- 22 layers



A good fully connected example

- https://playground.tensorflow.org/#activation=tanh&batchSize=10&dataset=spiral&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=8,8,8&seed=0.68609&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=true&xSquared=true&ySquared=true&cosX=false&sinX=true&cosY=false&sinY=true&collectStats=false&problem=classification&initZero=false&hideText=false