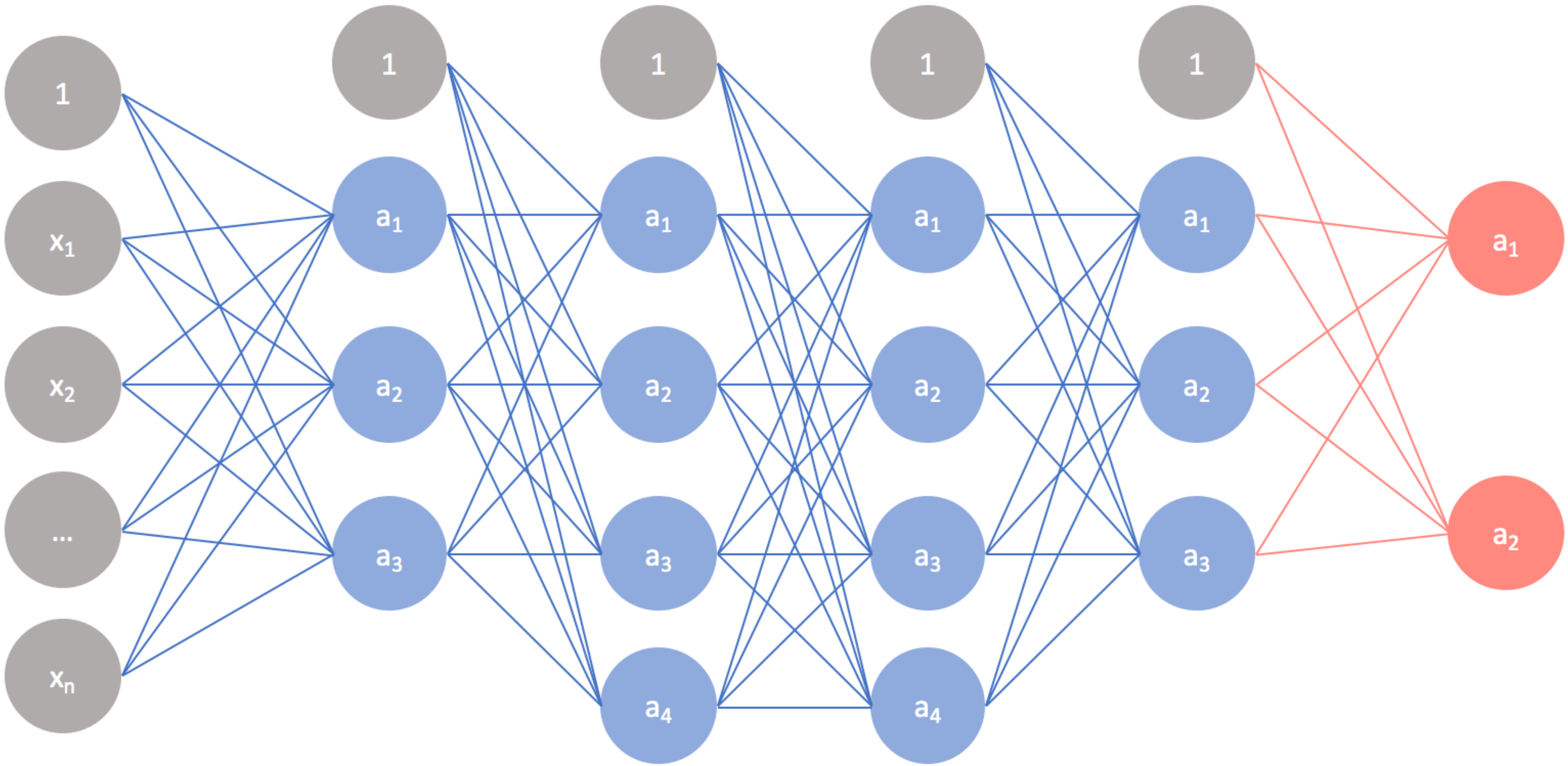
NN basics



References

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

contents

- The classification problem- again
- NN history
- Perceptron
  - Hyperplanes
  - Activation
- Dense layer
- Multi-layer perceptron (MLP)
- Optimization
  - Softmax + cross entropy + loss
  - Gradient descent
- Basic data preprocessing
  - Data normalization
  - Train, validation and test splits

contents

- **The classification problem- again**
- NN history
- Perceptron
  - Hyperplanes
  - Activation
- Dense layer
- Multi-layer perceptron (MLP)
- Optimization
  - Softmax + cross entropy + loss
  - Gradient descent
- Basic data preprocessing
  - Data normalization
  - Train, validation and test splits

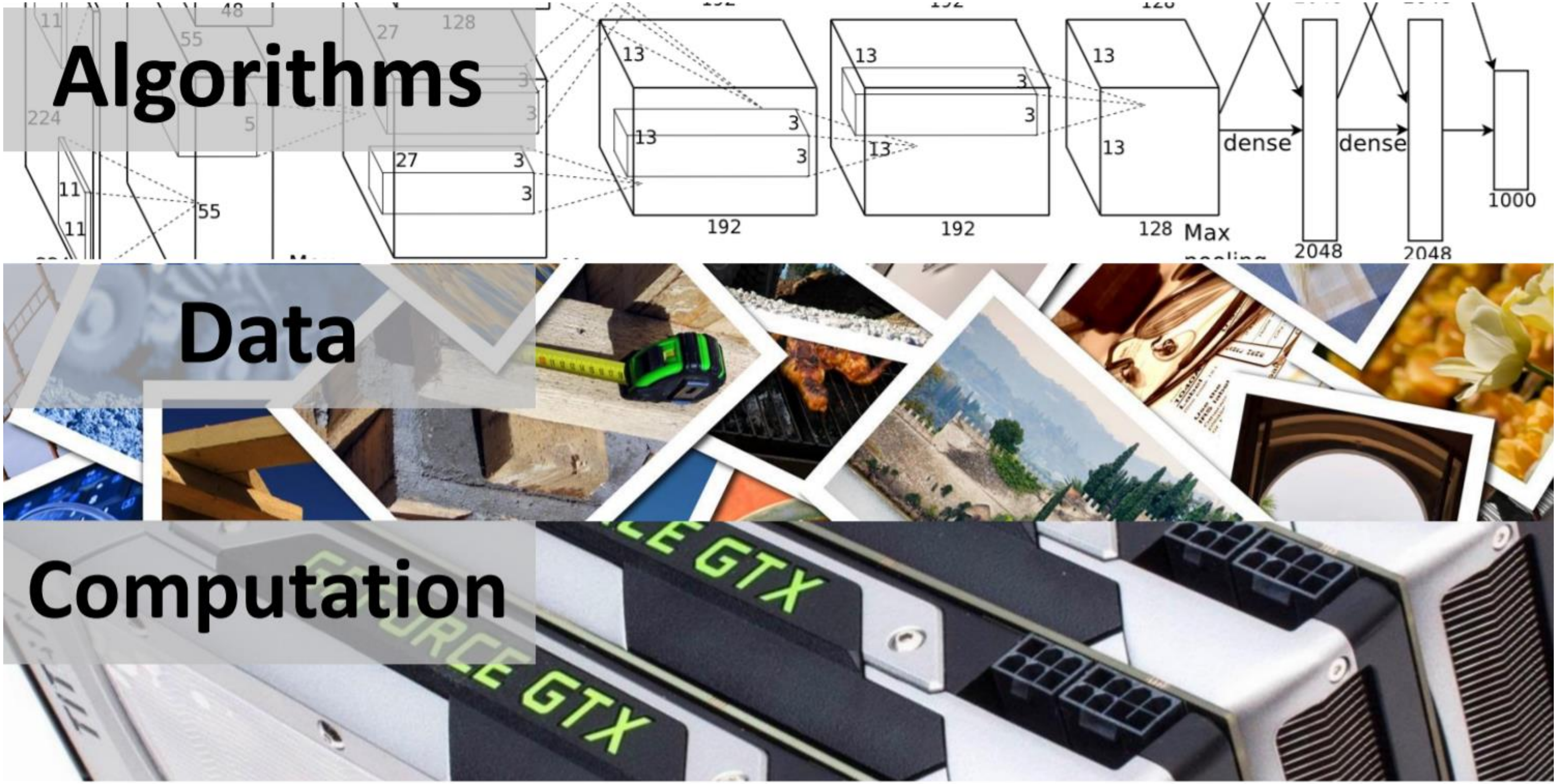
What will we know to do?

- 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?

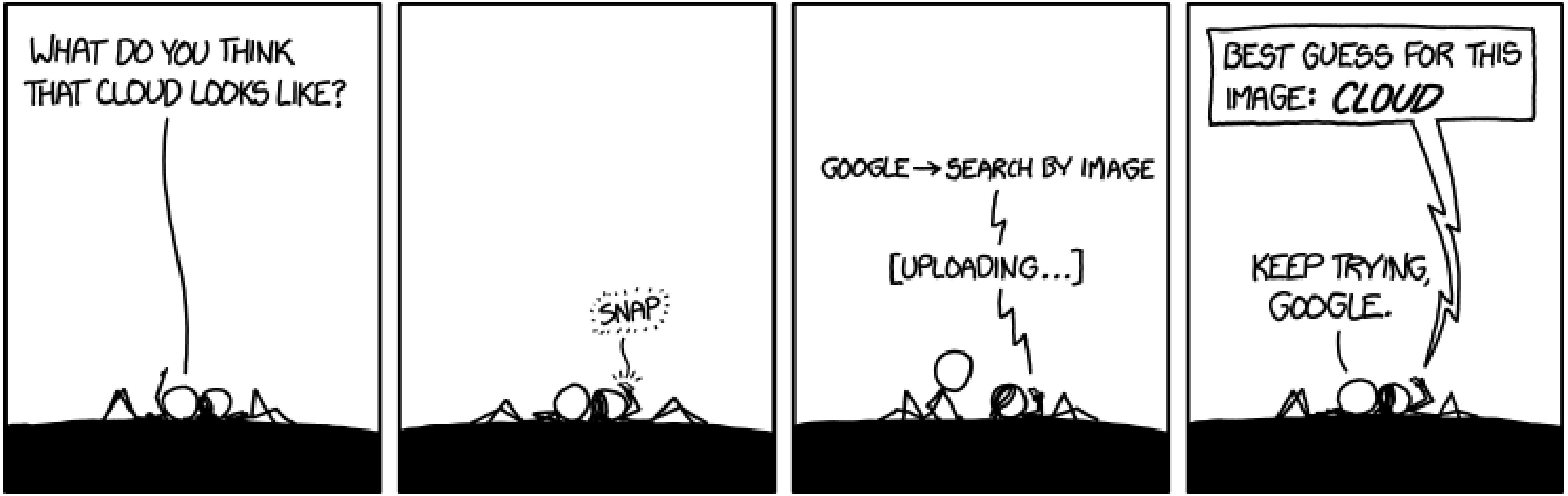


What a neural network can do?

- 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

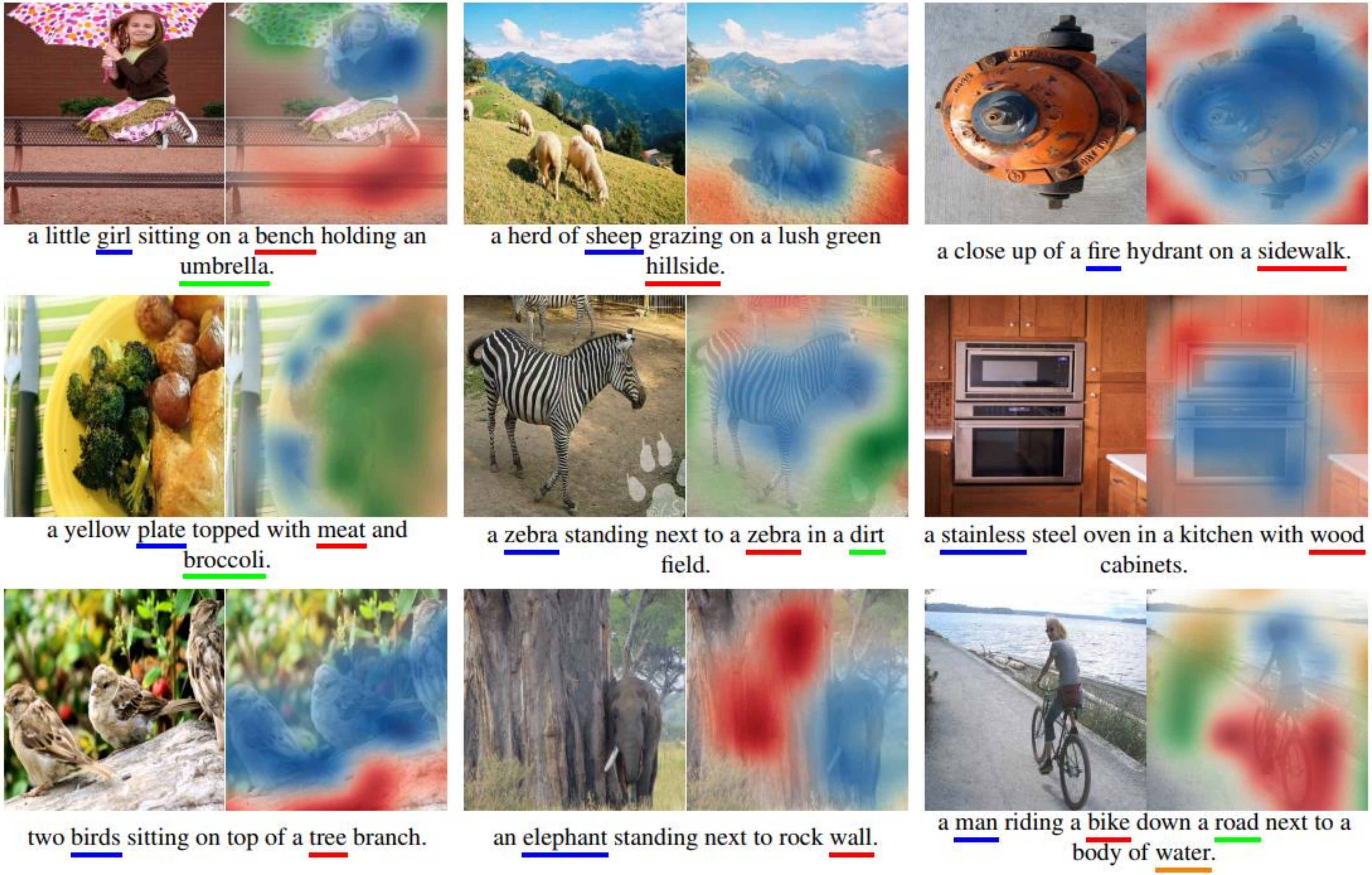
Object recognition



Human pose detection

3D reconstruction from a single image

Image captioning

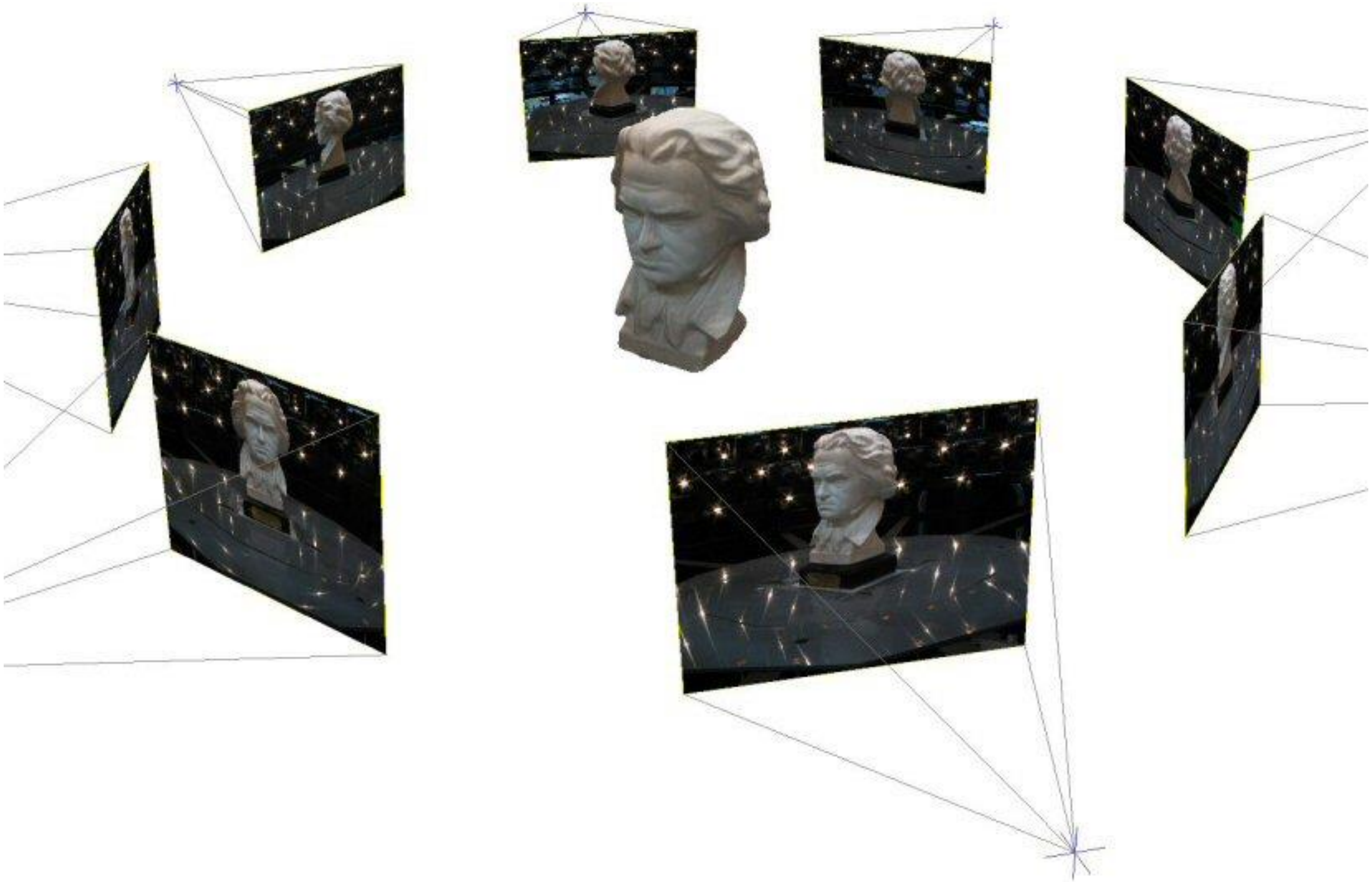


Style transfer

Object recognition challenges

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

Challenge: variable viewpoint



Challenge: variable illumination

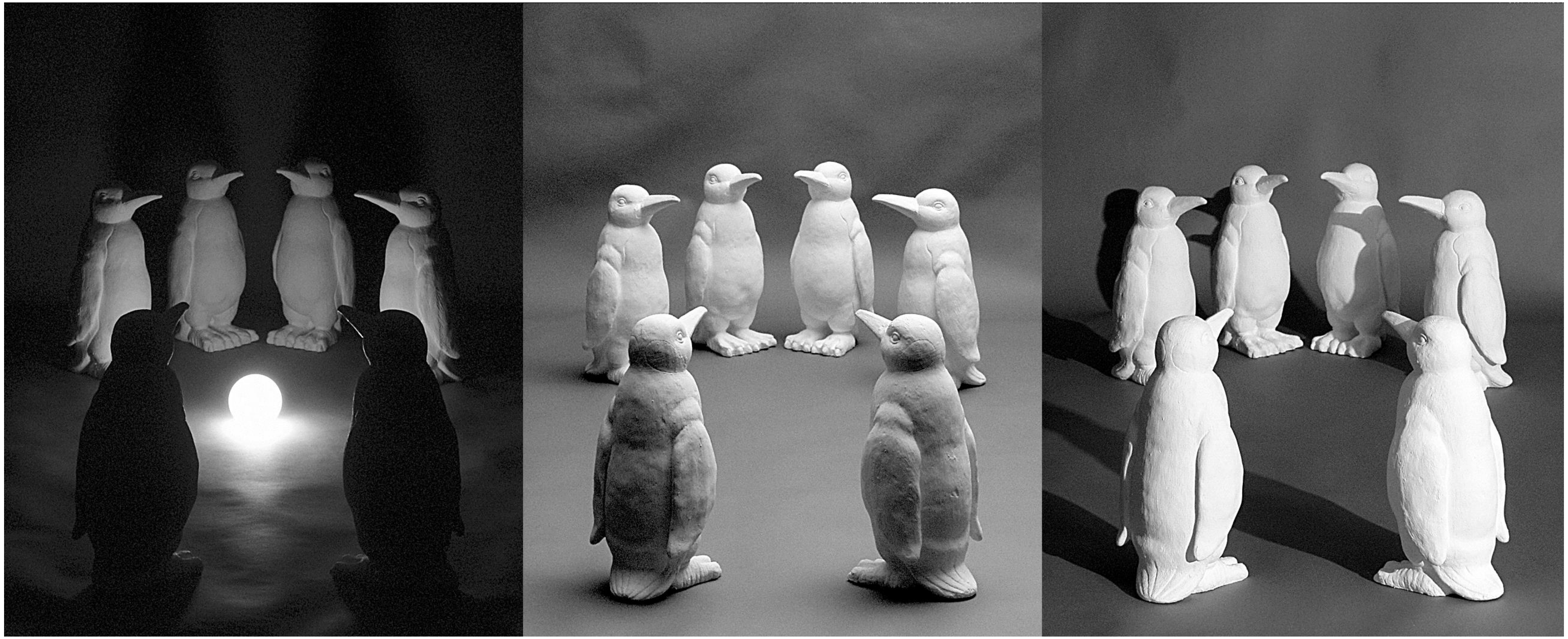


image credit: J. Koenderink

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 win!

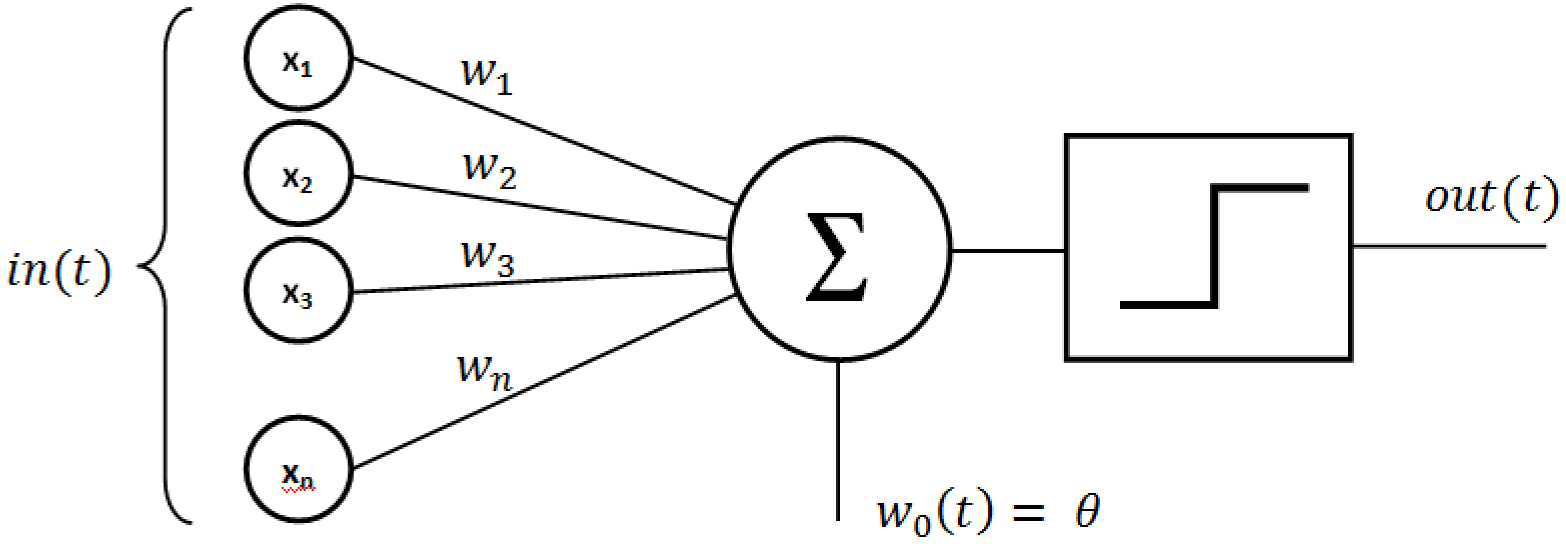


contents

- The classification problem- again
- **NN history**
- Perceptron
  - Hyperplanes
  - Activation
- Dense layer
- Multi-layer perceptron (MLP)
- Optimization
  - Softmax + cross entropy + loss
  - Gradient descent
- Basic data preprocessing
  - Data normalization
  - Train, validation and test splits

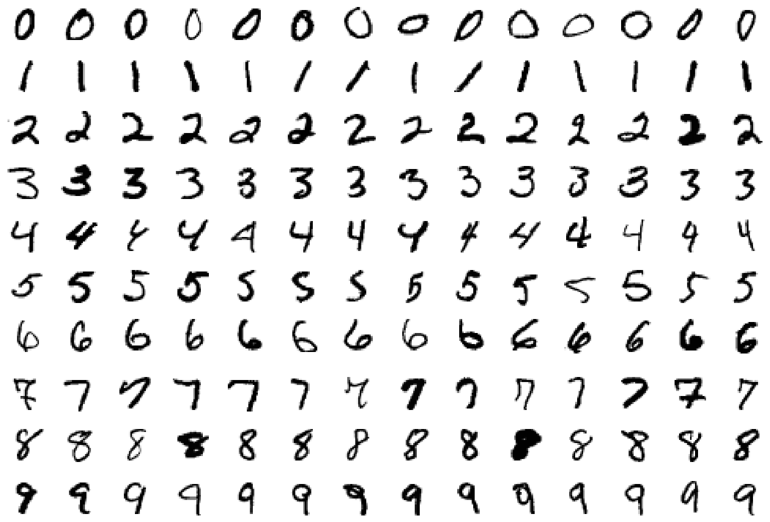
perceptron

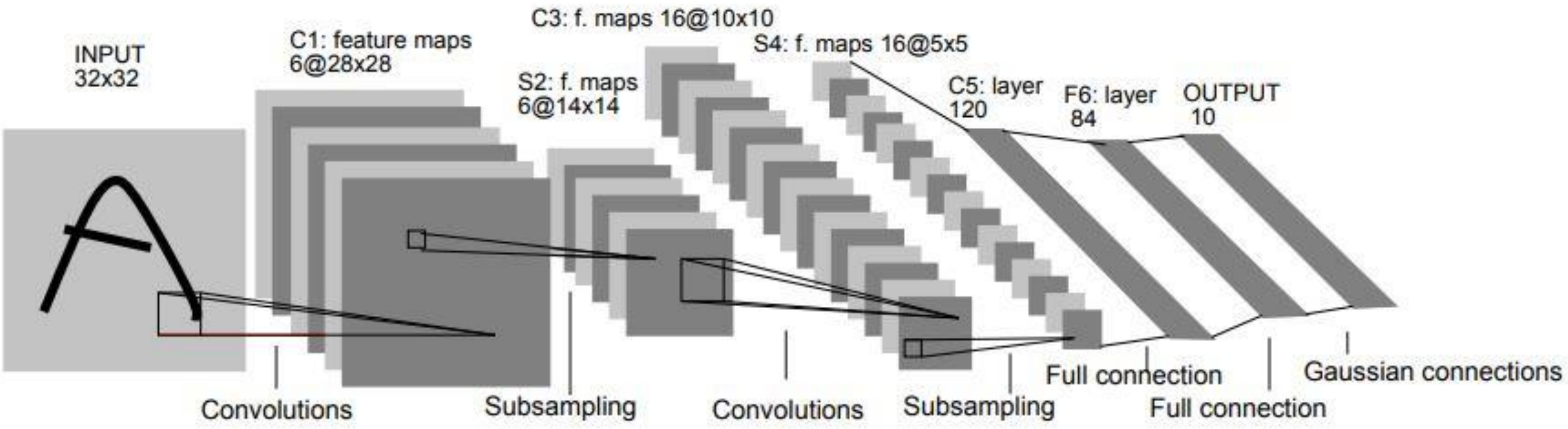
- The basic building block of all NN.
- First introduced in 1958 at Cornell Aeronautical Laboratory by Frank Rosenblatt.
- We will talk more about it in a moment…



MNIST + LeNet-5

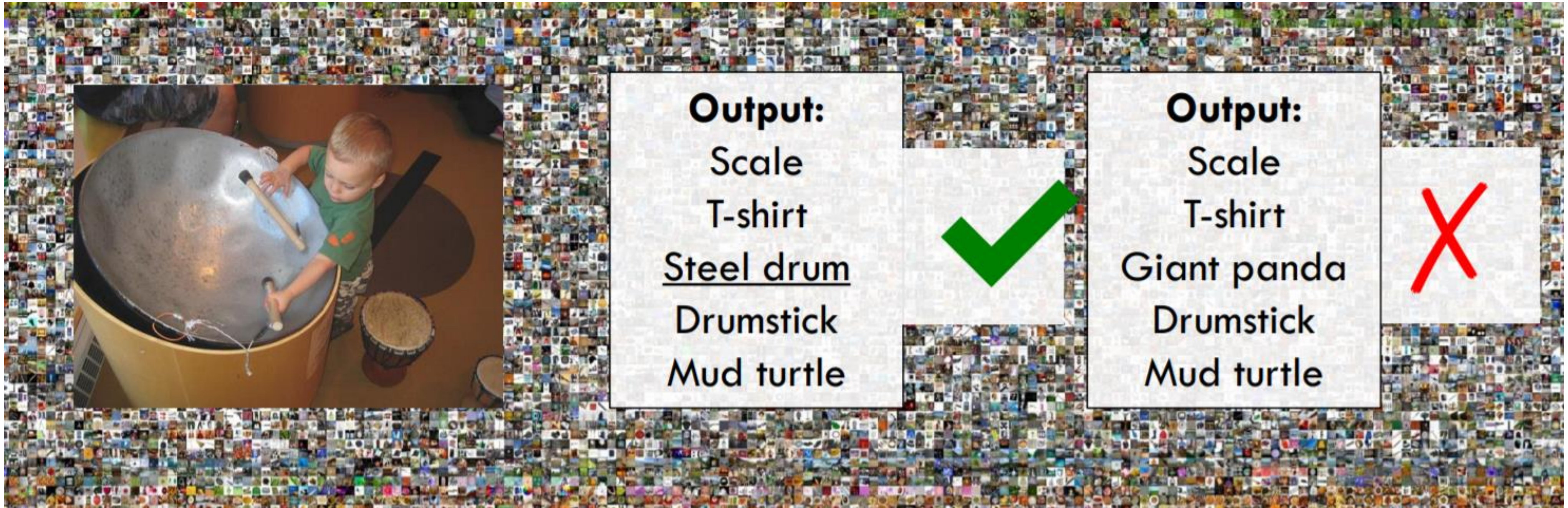
- MNIST is a large dataset of handwritten digits used in training of LeNet-5.
- LeNet-5 is the first known NN to solve a major computer vision problem:
  - Classifies digits, was applied by several banks to recognize hand-written numbers on checks.
  - Used 7 trainable layers with a total of **60K**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



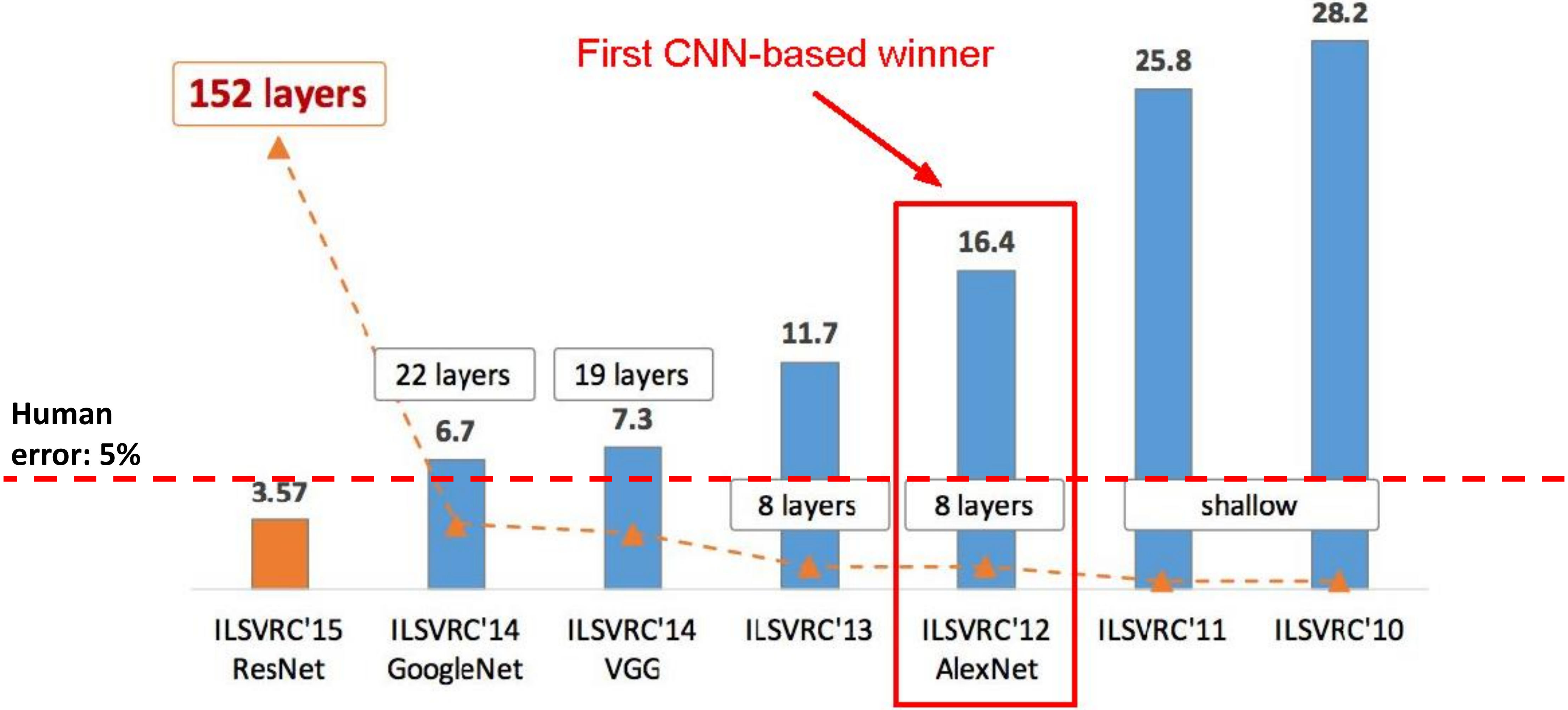


IMAGENET Large Scale Visual Recognition Challenge (ILSVRC)

- 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

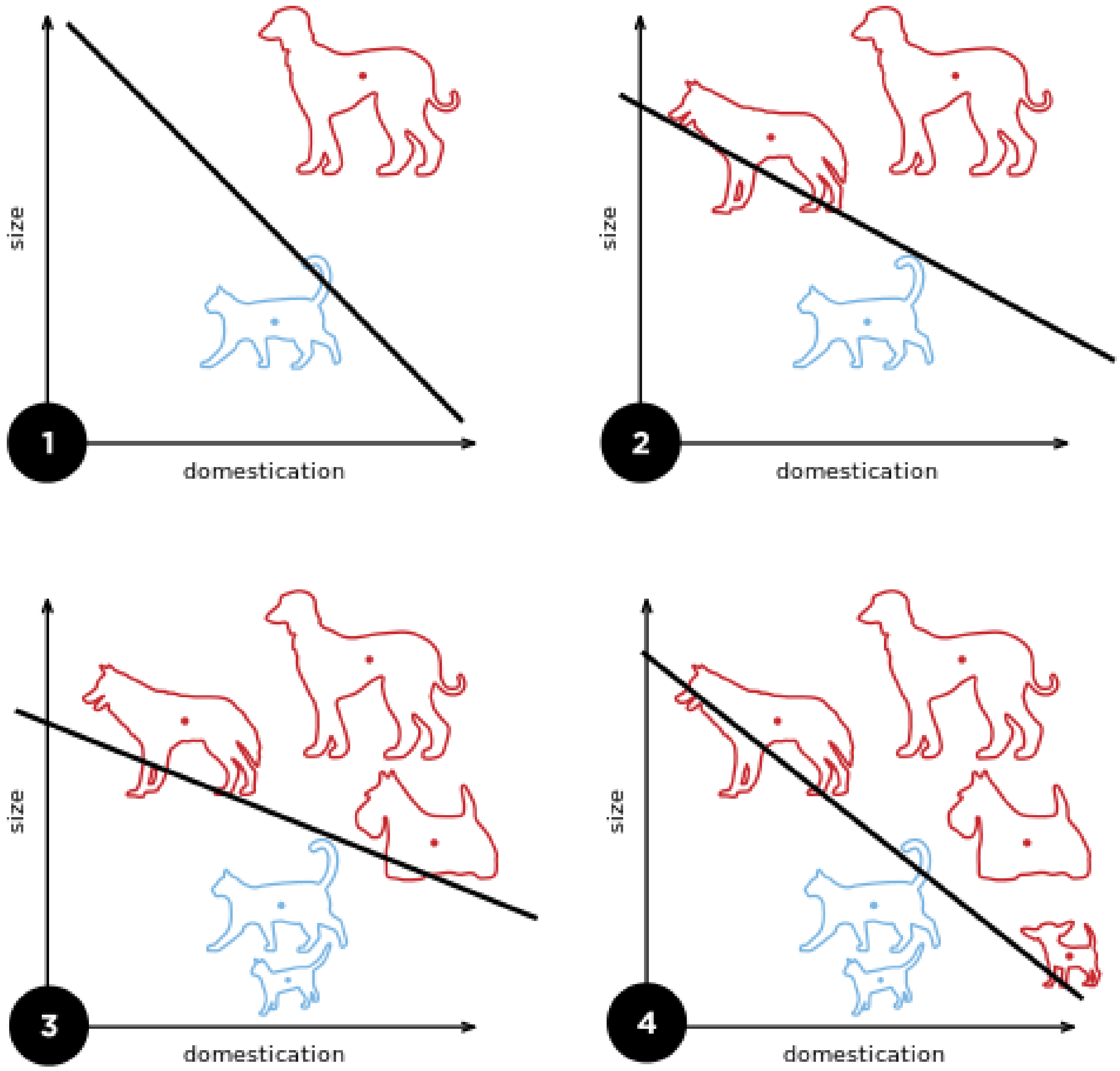


contents

- The classification problem- again
- NN history
- **Perceptron**
  - Hyperplanes
  - Activation
- Dense layer
- Multi-layer perceptron (MLP)
- Optimization
  - Softmax + cross entropy + loss
  - Gradient descent
- Basic data preprocessing
  - Data normalization
  - Train, validation and test splits

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

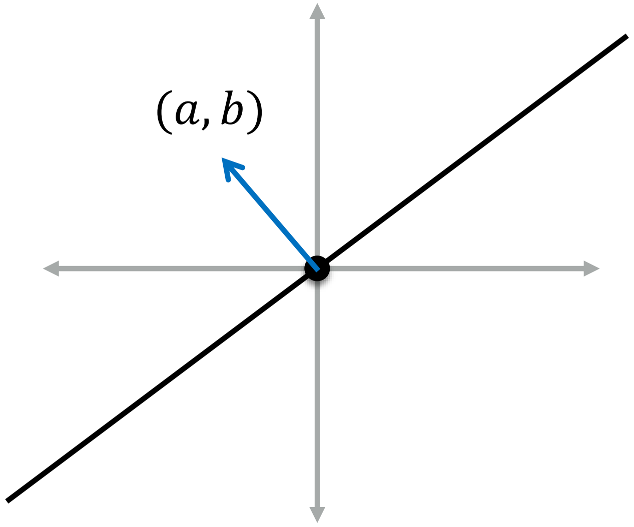
- Paramtrization of a line in 2D:

$$ax + by + c = 0$$

  - 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

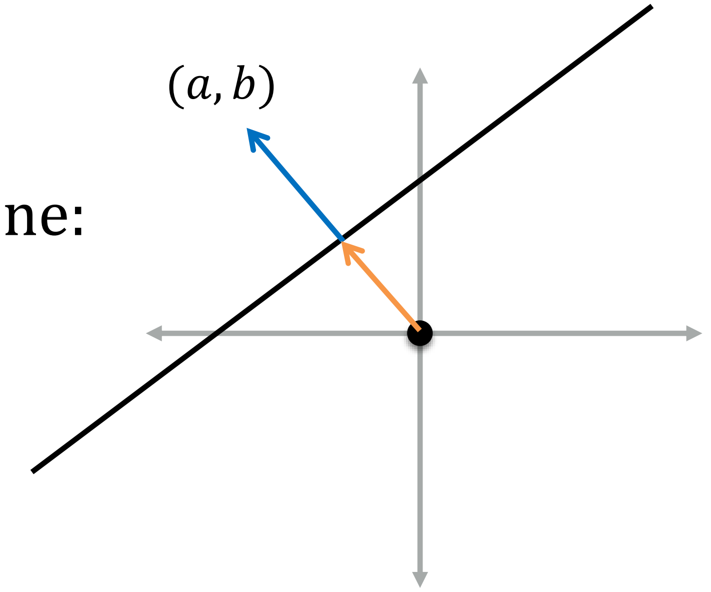
- Paramtrization of a line in 2D:

$$ax + by + c = 0$$

  - 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: $\mathrm{d} = \frac{|ax+by+c|}{\sqrt{a^2+b^2}}$
    - $bias = \frac{|c|}{\sqrt{a^2+b^2}}$



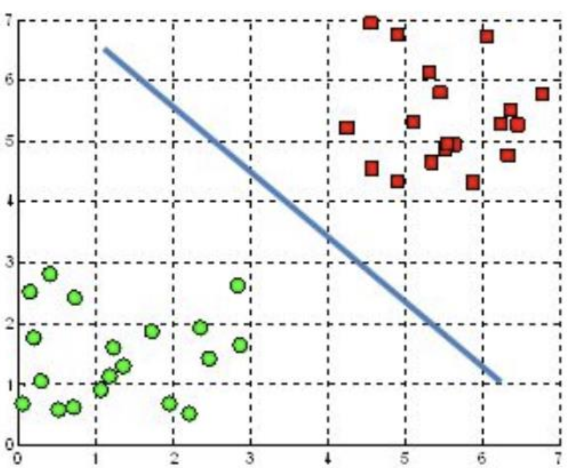
hyperplane

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

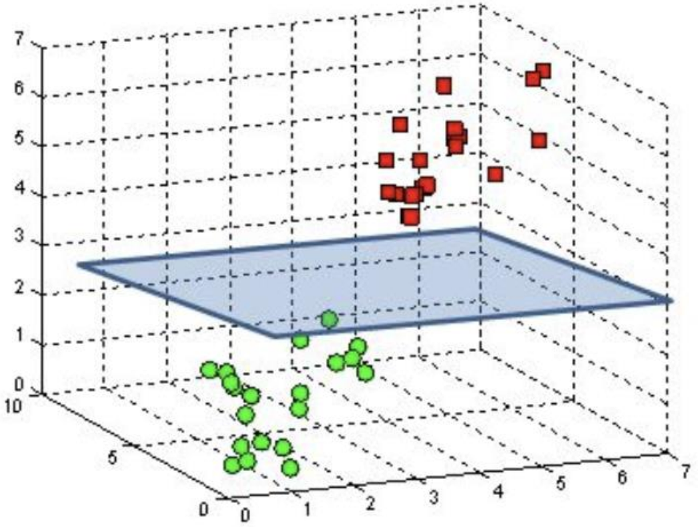
$$ax + by + cz + d = 0$$

  - (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.**

A hyperplane in $\mathbb{R}^2$ is a line



A hyperplane in $\mathbb{R}^3$ is a plane



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 $w^T x < 0$.

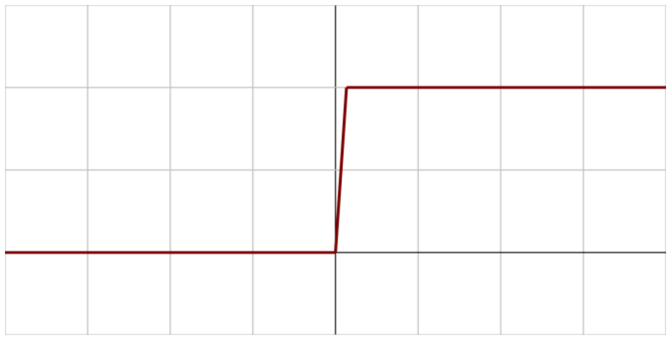
Activation function

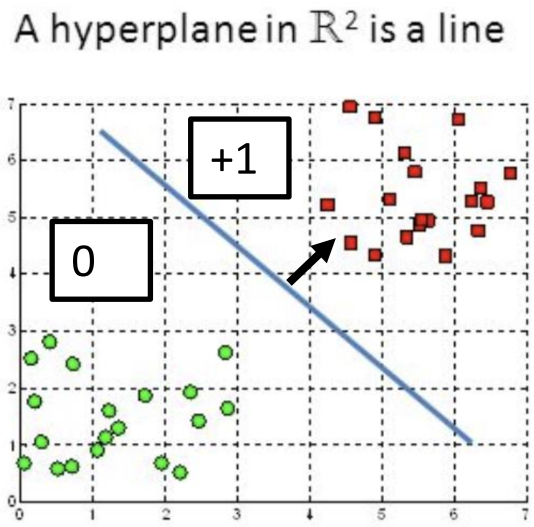
- A non-linear function $f()$ that appends the perceptron's hyperplane equation

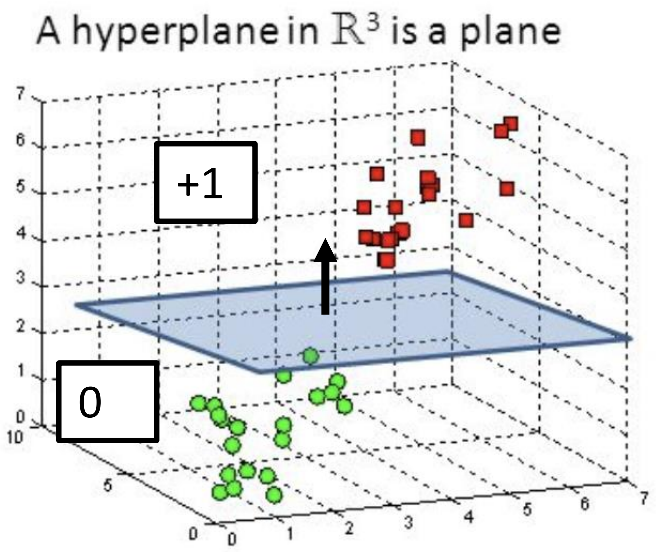
$$y = f(Wx).$$

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

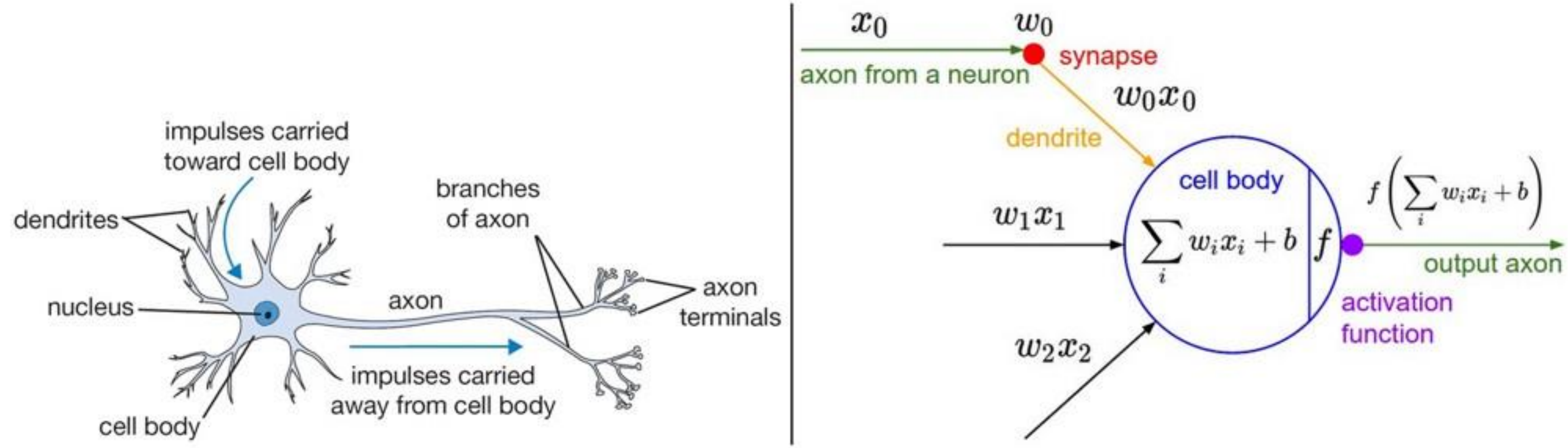
$$f(x) = step(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$







perceptron: Inspiration from Biology

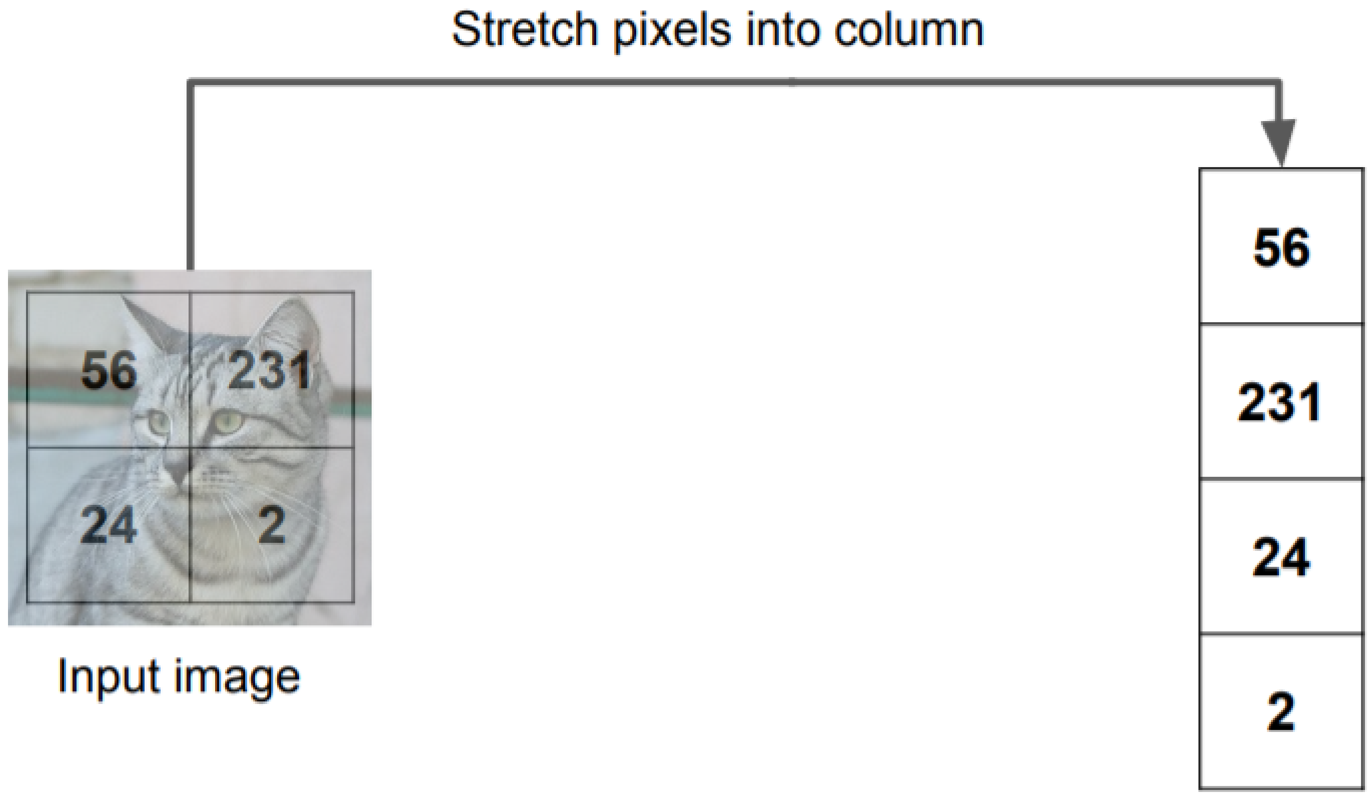


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

- 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.

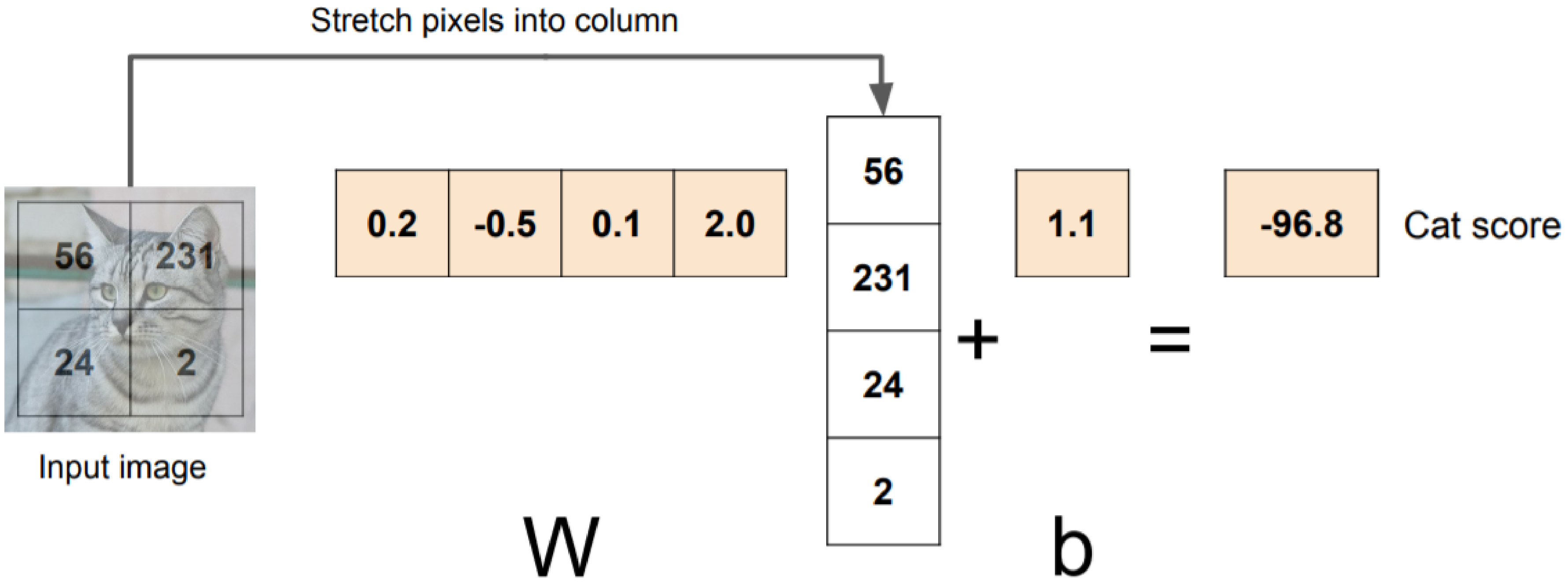
Images as inputs

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



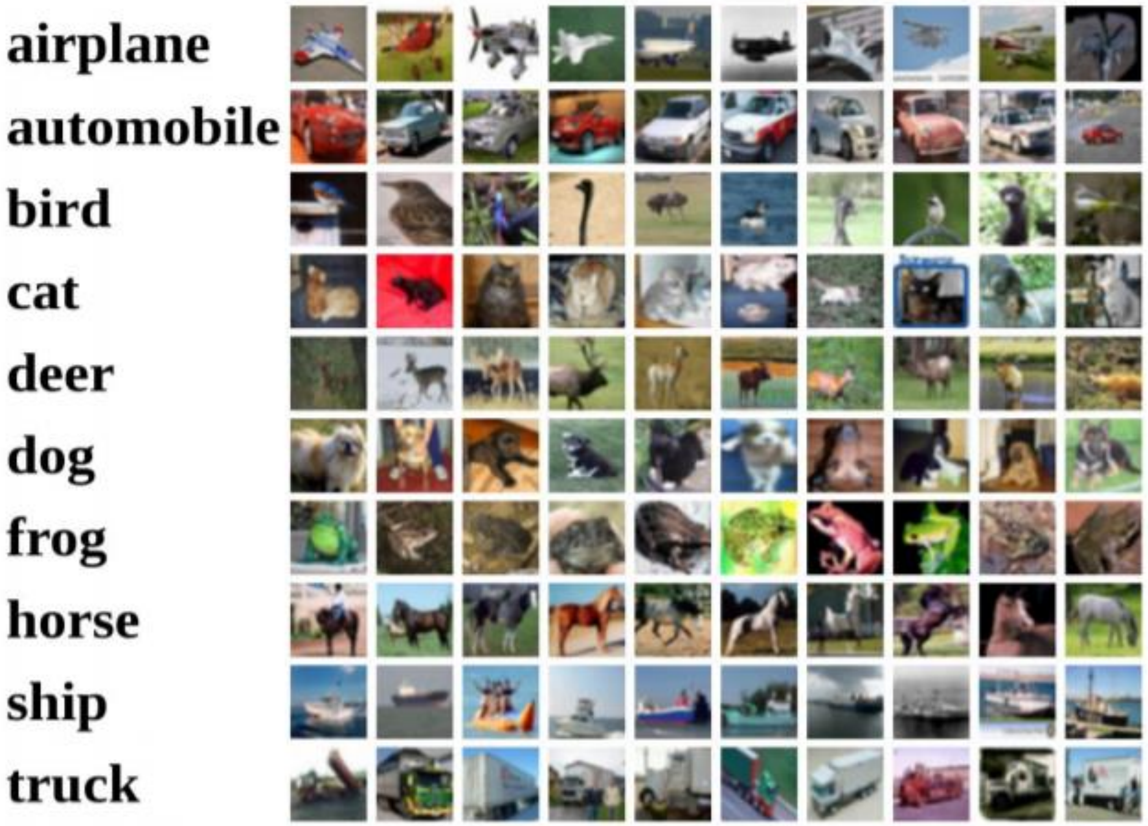
Images as inputs

- 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.



CIFAR10 dataset

- CIFAR10 (Canadian Institute For Advanced Research) is a known dataset of 10 classes of small images.



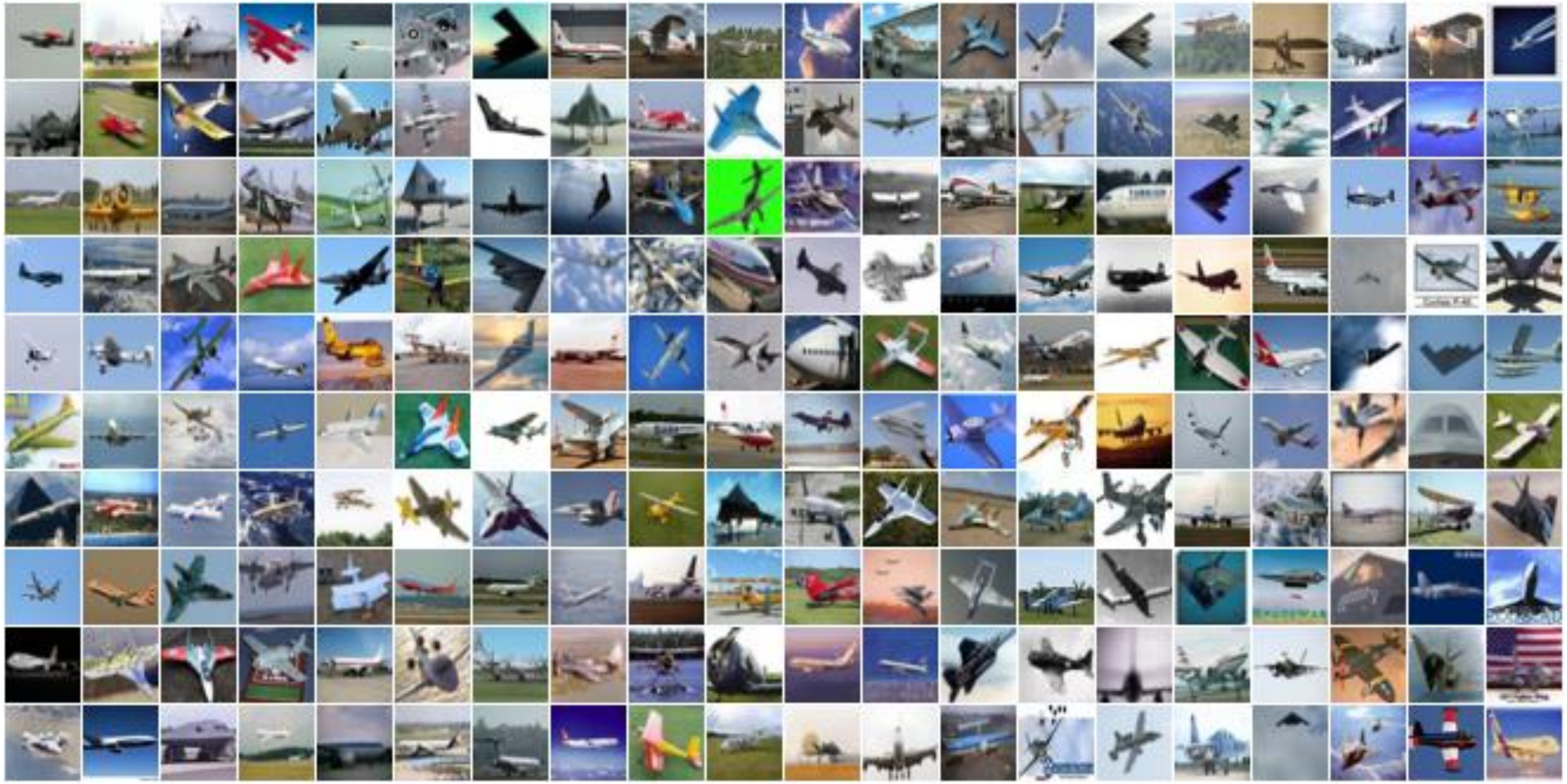
10 classes

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

10,000 test images.

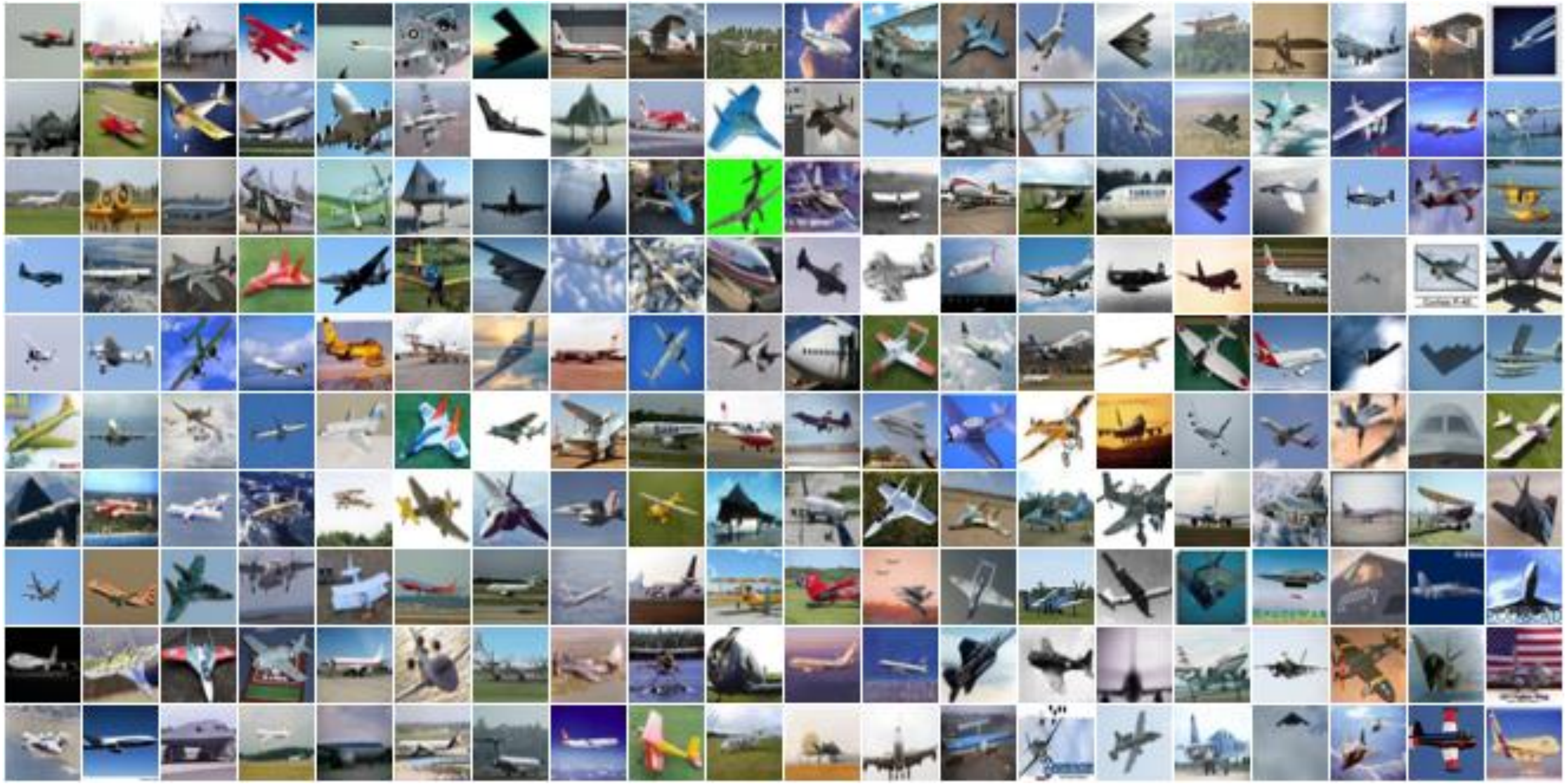
What is the best separator for such data?

- Assume the pixels values are [0,255] -> [-127,128]



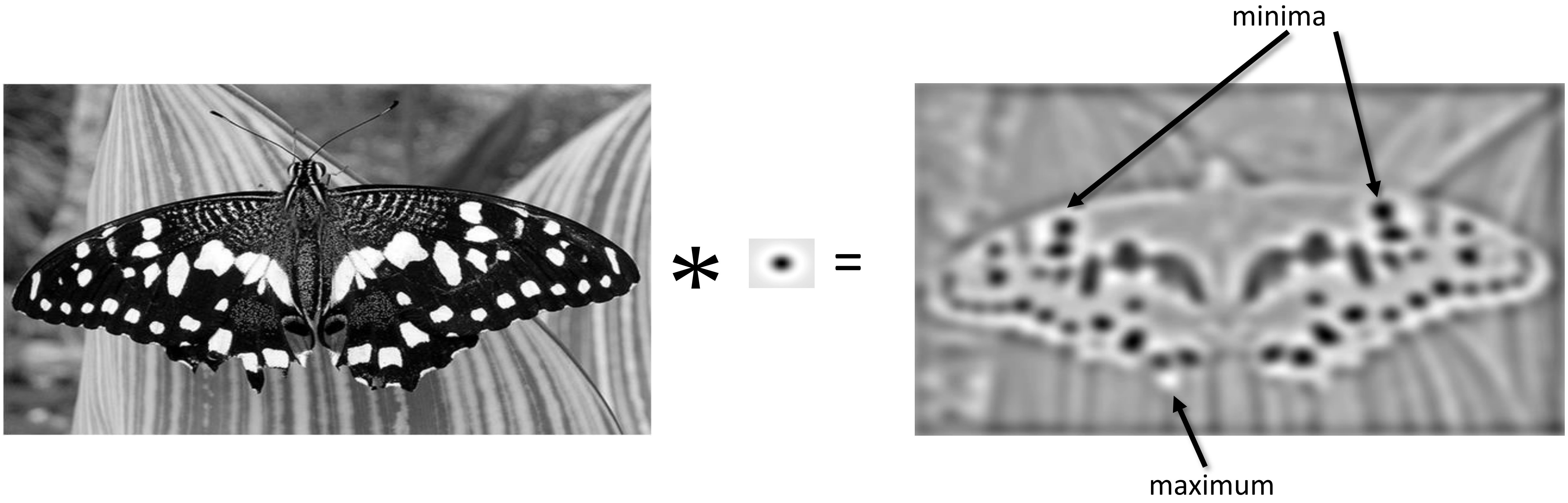
What is the best separator for such data?

- We can try and take the mean image per class.



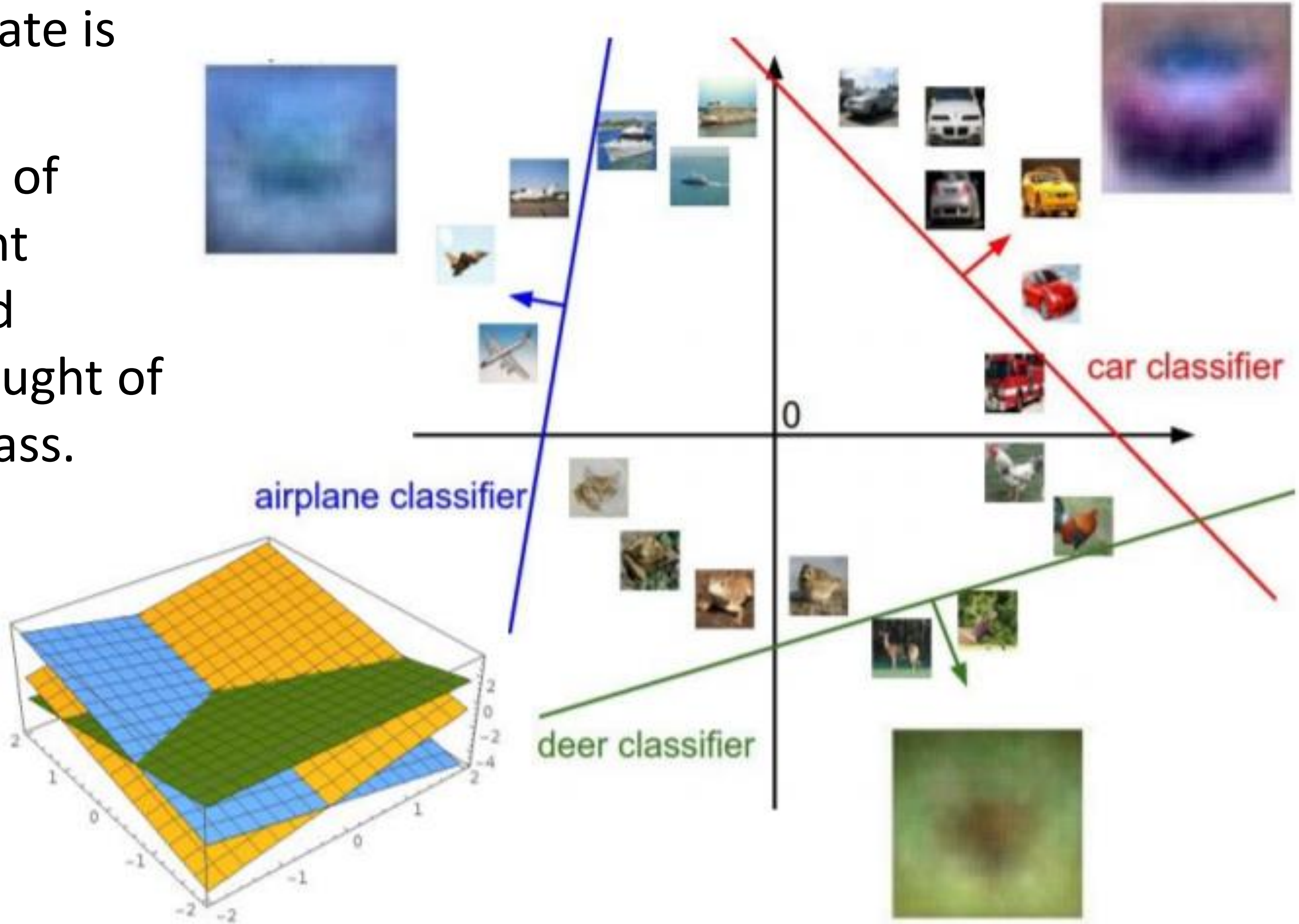
Perceptron: template matching interpretation

- We can think about the optimized weights as an image sized template in zero mean cross correlation (ZMCC) algorithm.
  - We get a strong positive response when the template matches the image area.

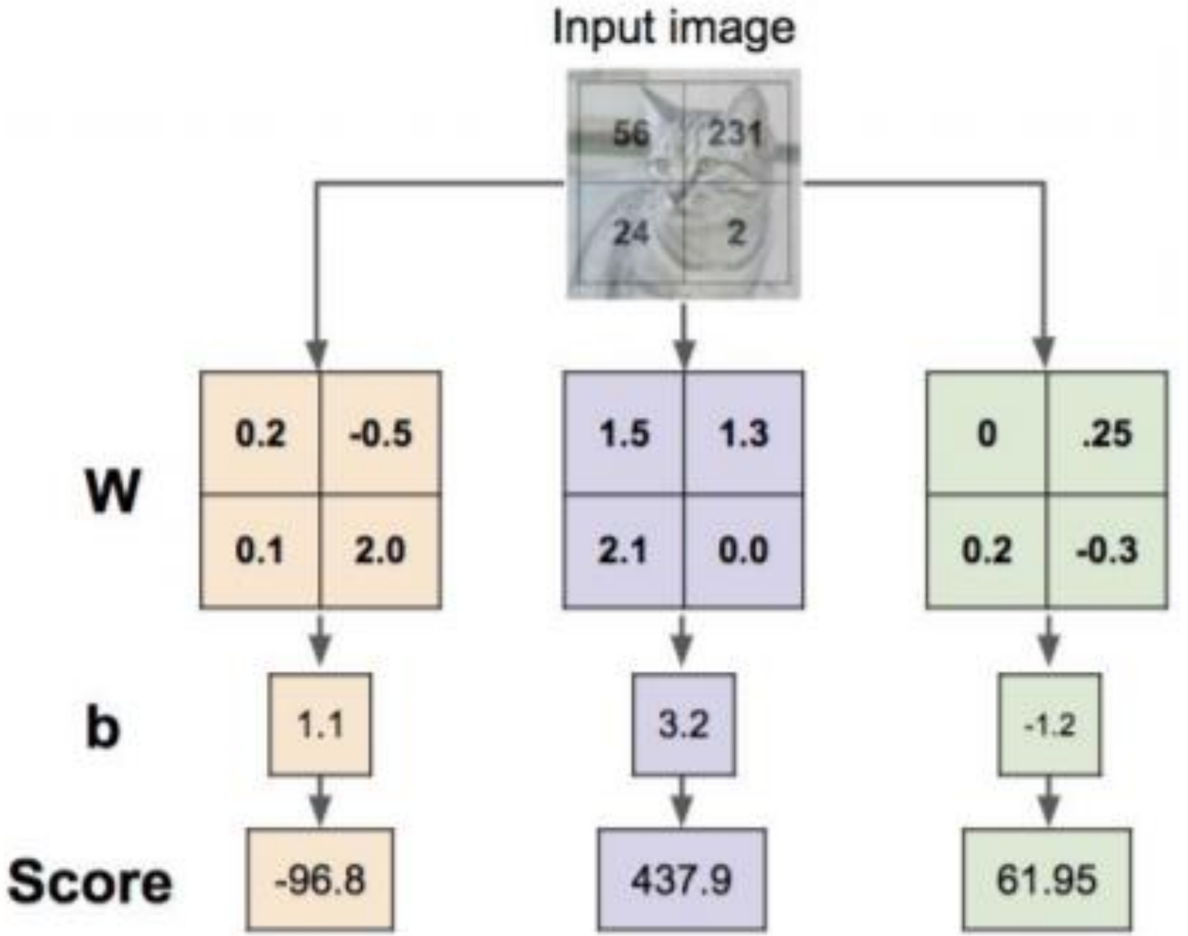


Perceptron: template matching interpretation

- 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

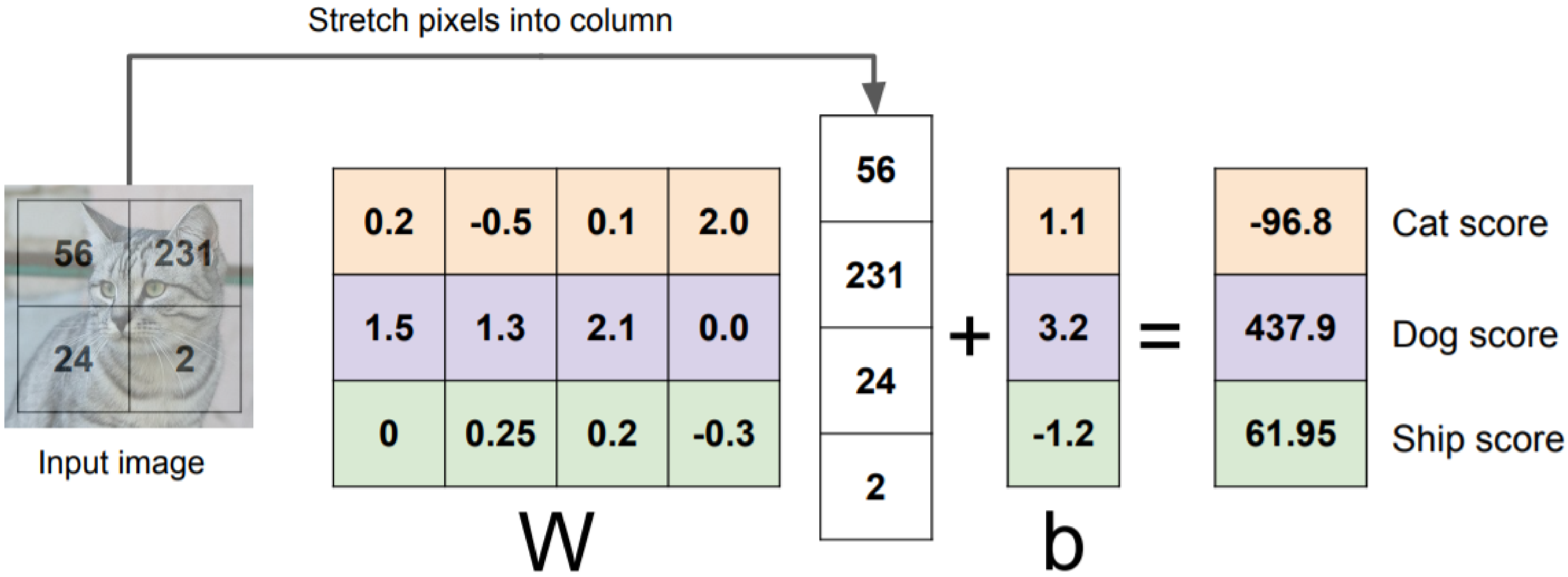


contents

- The classification problem- again
- NN history
- Perceptron
  - Hyperplanes
  - Activation
- **Dense layer**
- Multi-layer perceptron (MLP)
- Optimization
  - Softmax + cross entropy + loss
  - Gradient descent
- Basic data preprocessing
  - Data normalization
  - Train, validation and test splits

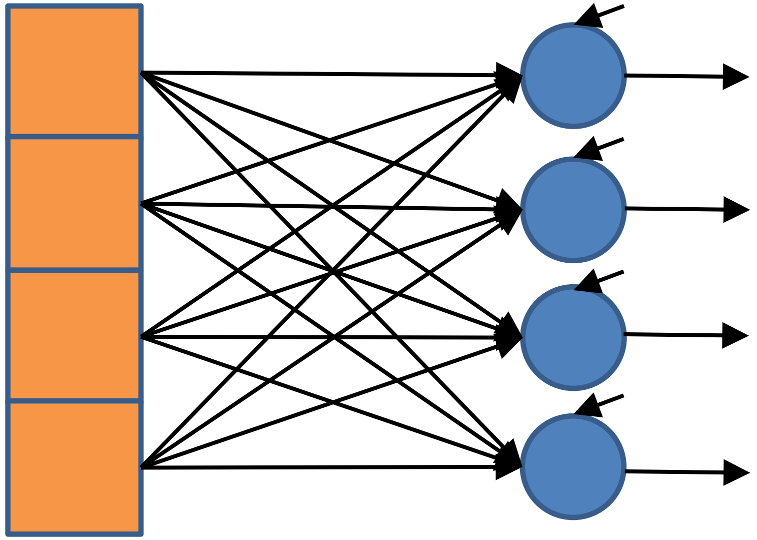
Hyperplanes and image classification

- 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.



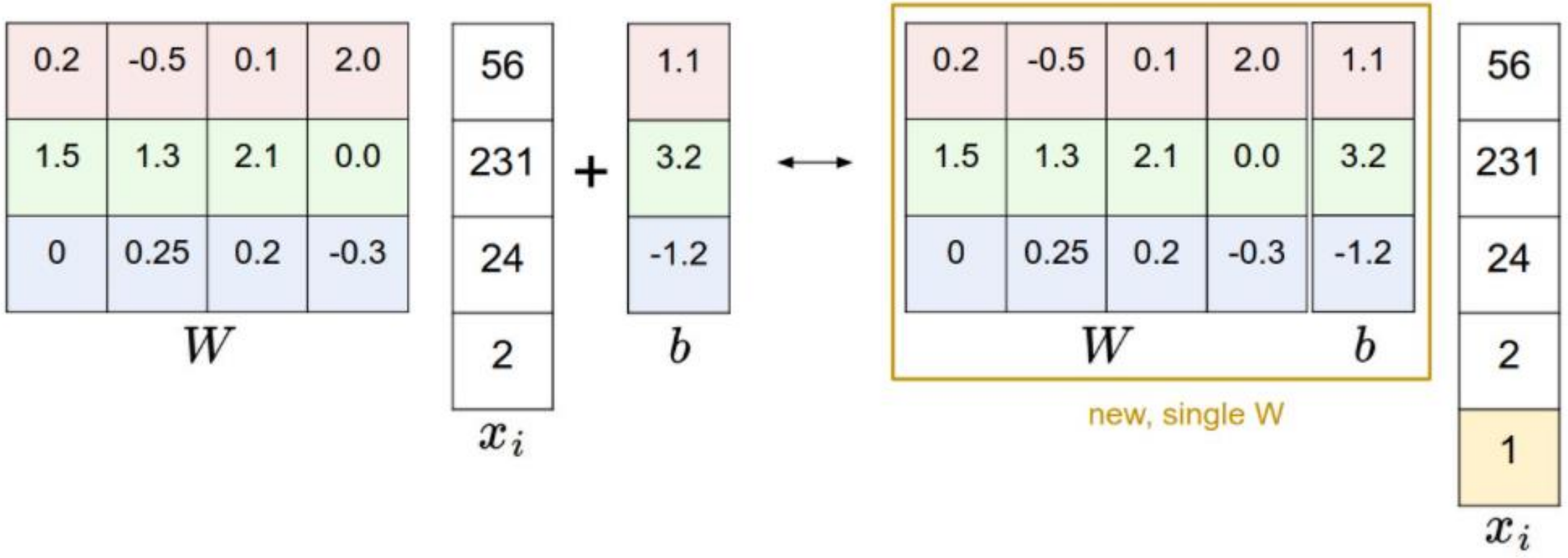
Dense layer

- This is the first NN layer we encounter- all inputs are going through multiple perceptrons at the same time.
- This layer is called **dense layer** or **fully-connected layer**.



Dense layer

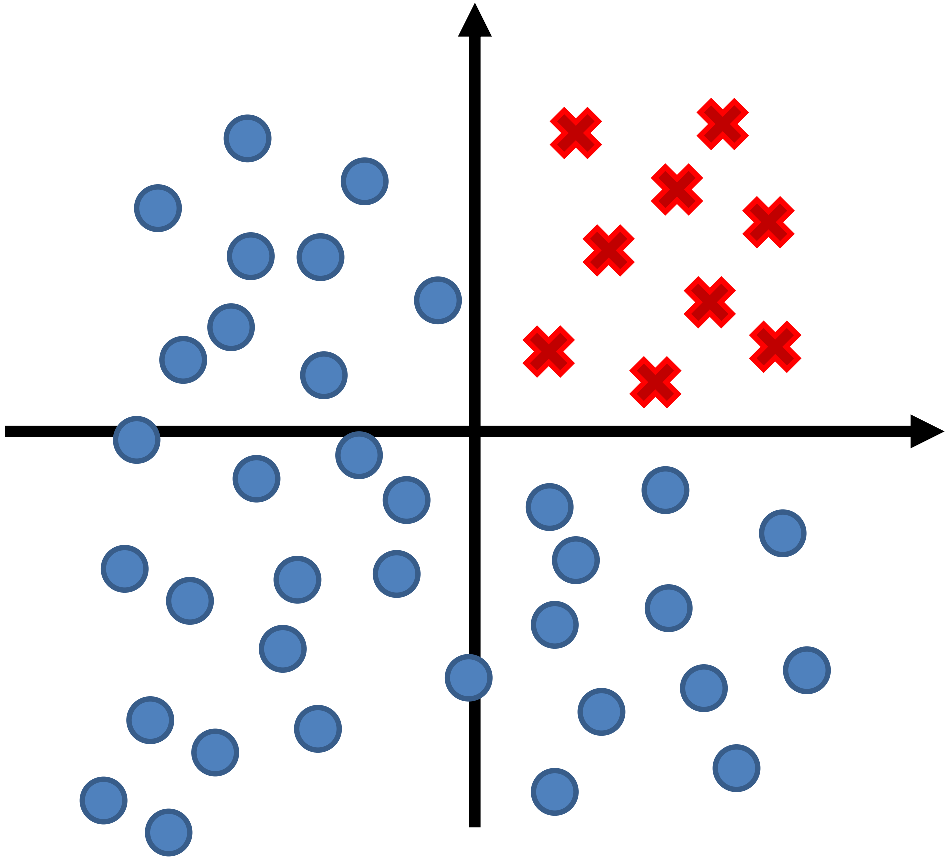
- Sometimes you can see W and b concatenated like this:



contents

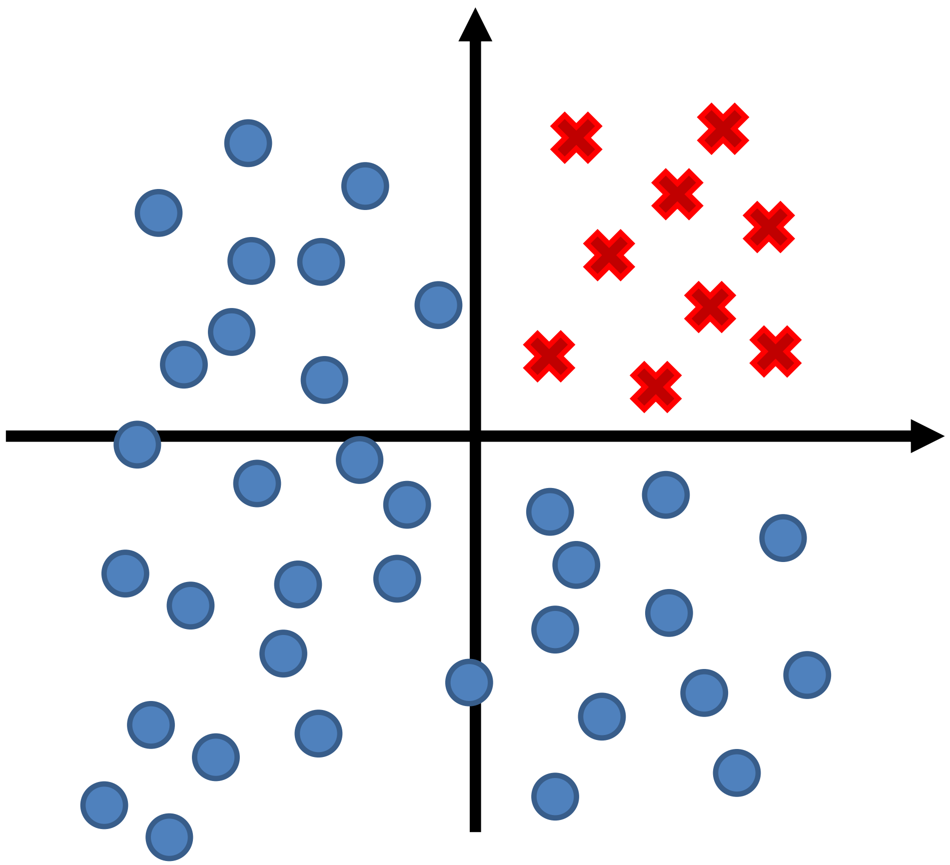
- The classification problem- again
- NN history
- Perceptron
  - Hyperplanes
  - Activation
- Dense layer
- **Multi-layer perceptron (MLP)**
- Optimization
  - Softmax + cross entropy + loss
  - Gradient descent
- Basic data preprocessing
  - Data normalization
  - Train, validation and test splits

- Try to solve classification of RAVIA 1 with 2 layers perceptrons and step activation.
- Interpretation of feature transformation + usage of other activations.



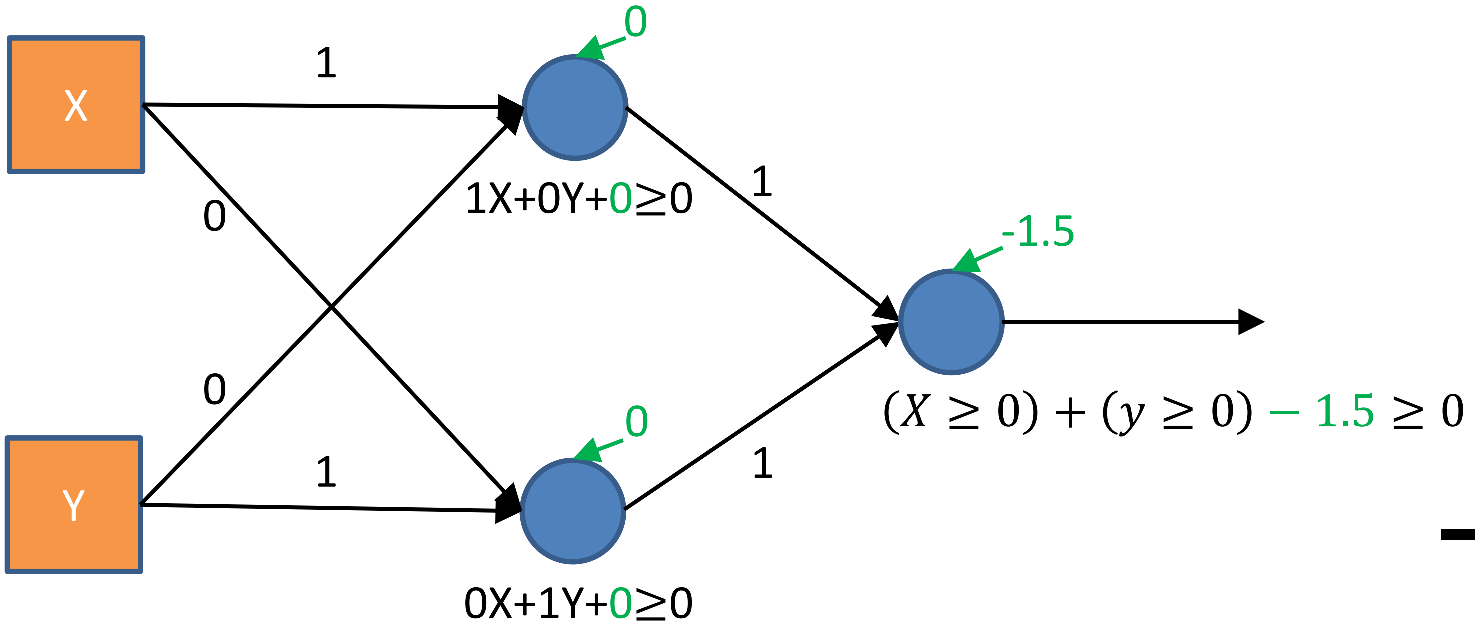
Hyperplane classification is not enough

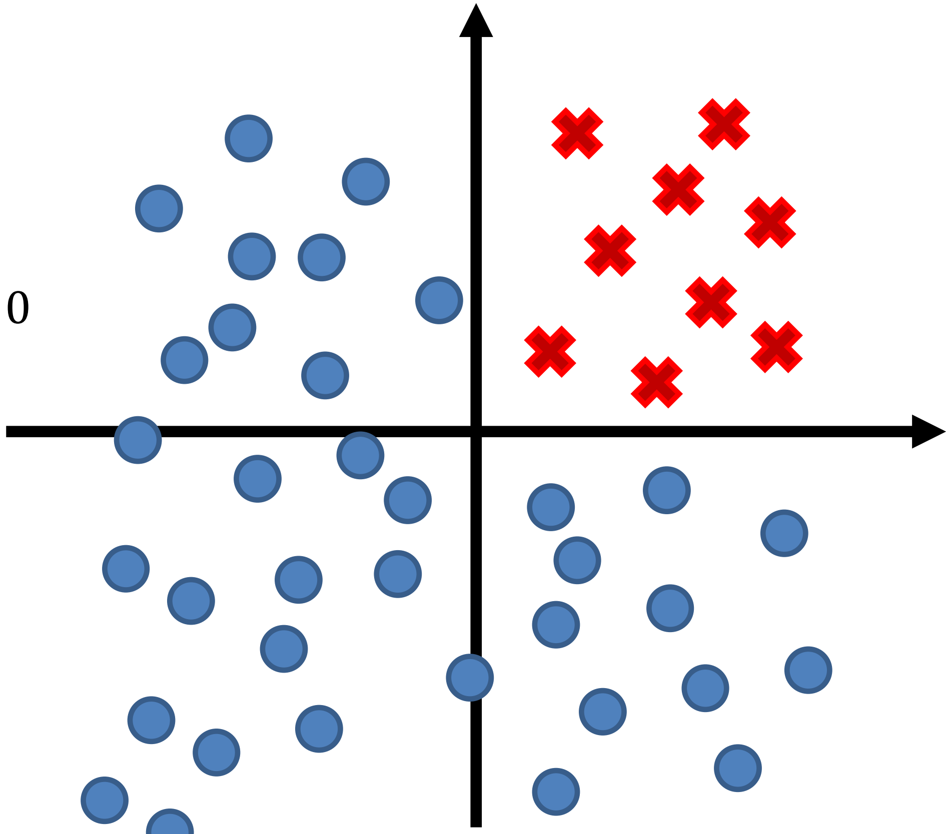
- Not all datasets can be linearly separable.
- Try classify ✖ as the positive +1 class using a perceptron.



Hyperplane classification is not enough

- multi-layer perceptron (MLP), or in a more common name- **neural network**, is a better approach to try to handle this data.



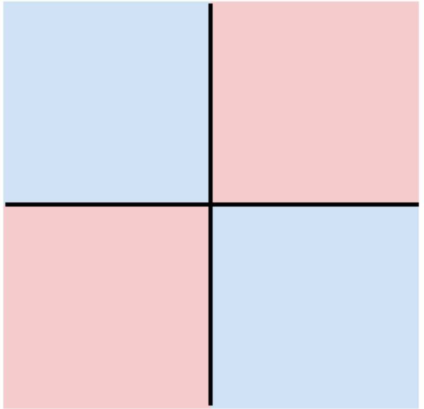


Feature transformation

- We can build “new” features from the data and the classify it!
- Another example without perceptrons:

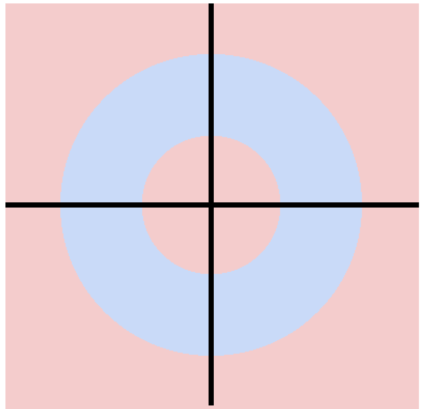
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



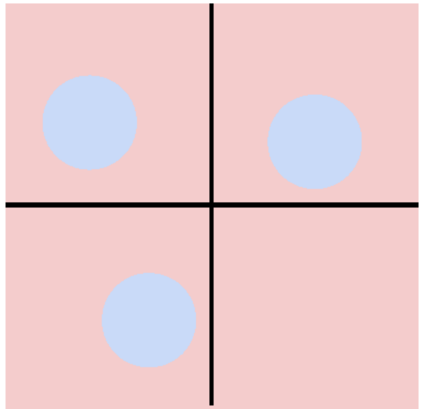
Class 1:
1 <= L2 norm <= 2

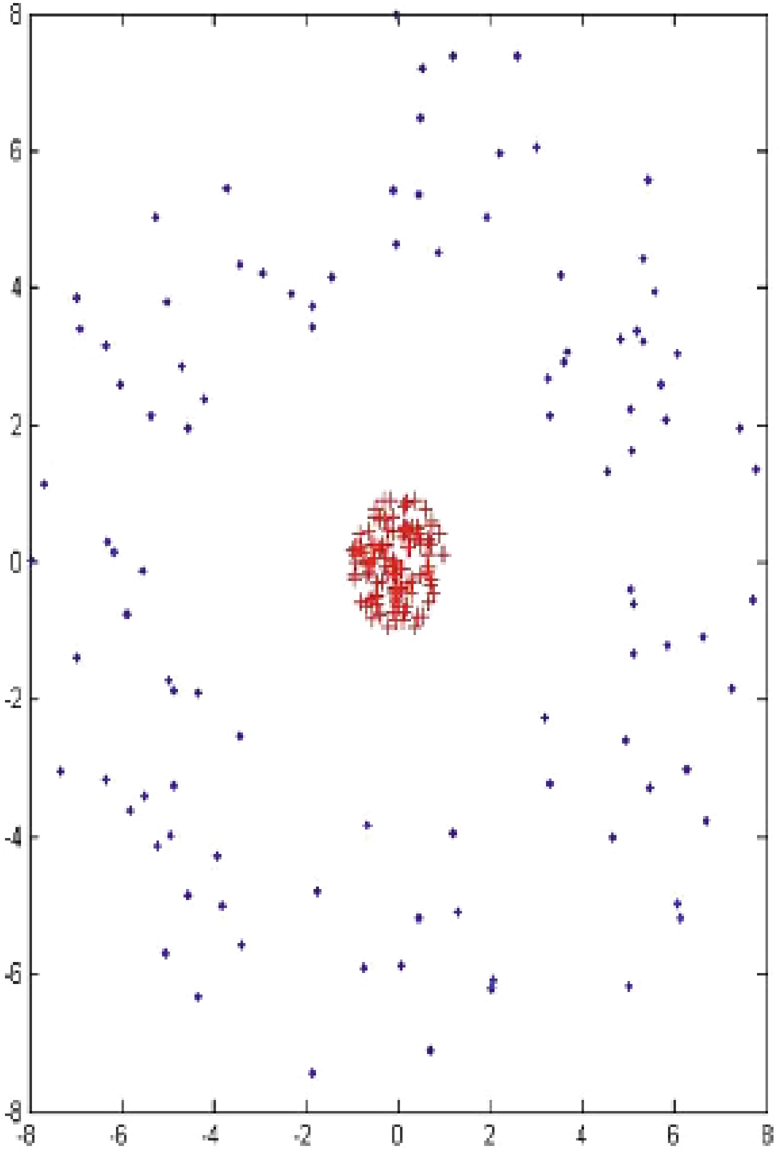
Class 2:
Everything else



Class 1:
Three modes

Class 2:
Everything else



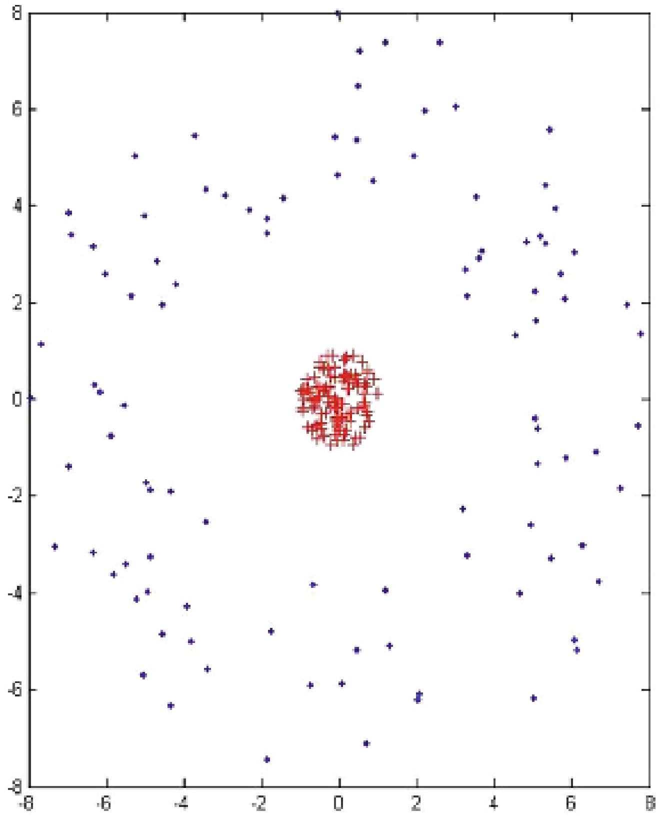


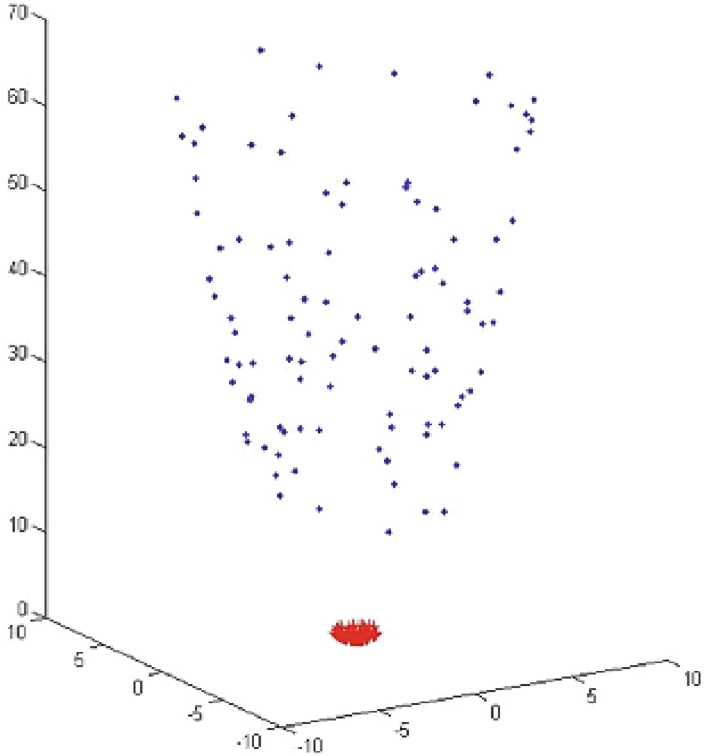
Hyperplane classification is not enough

- We can build a transformation function from our feature data $[x, y]$ to a new linearly separable dataset:

$$T([x, y]) \rightarrow [x, y, x^2 + y^2]$$

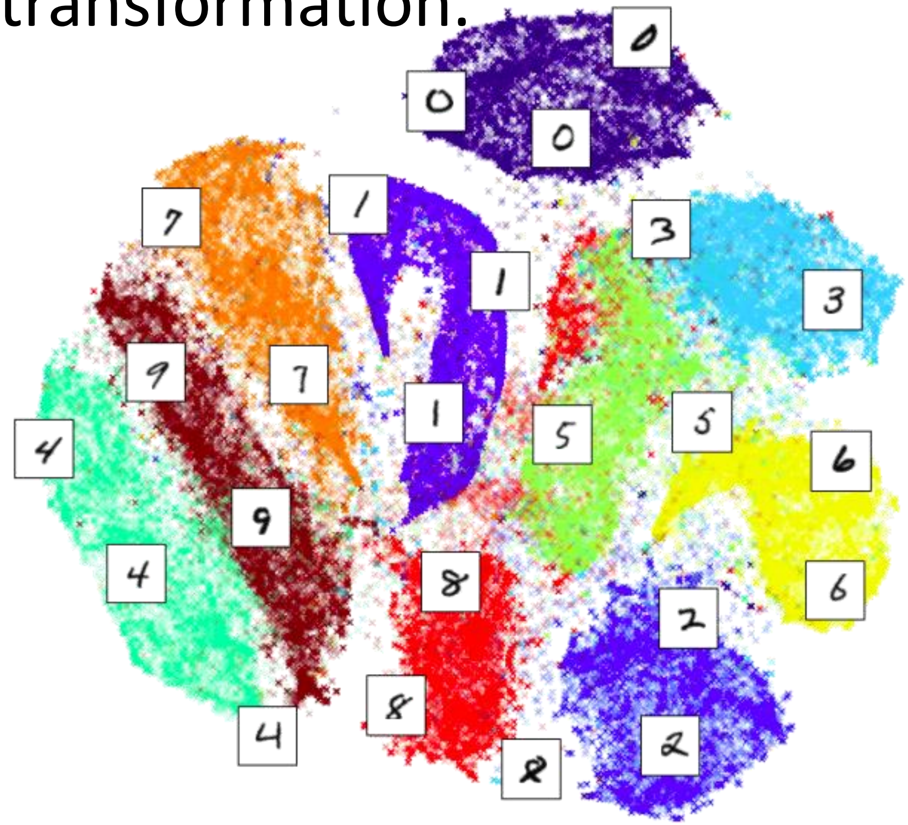
- Now we can separate the dataset with the plane $Z = R^2 = 4$ for example.





Feature transformation

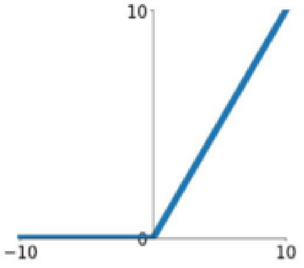
- By using multiple layers with varying number of neurons in each we can try and transform any input data to some space where it can be linearly separable.
- https://www.youtube.com/watch?v=Uf3wnBNXV4k
- The activation function is really important for the transformation.



Activation functions

- Here are some more activation functions.
- <u>The most used</u> is the **rectified linear unit** (**ReLU**) function:

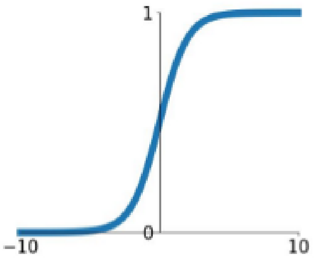
$$f(x) = \max(x, 0) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$



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

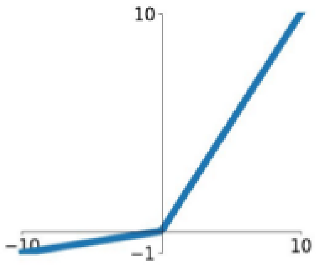
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



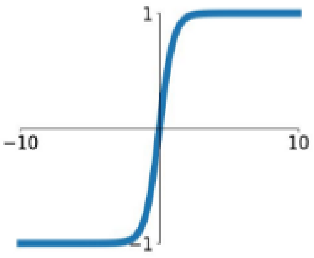
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$



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?

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

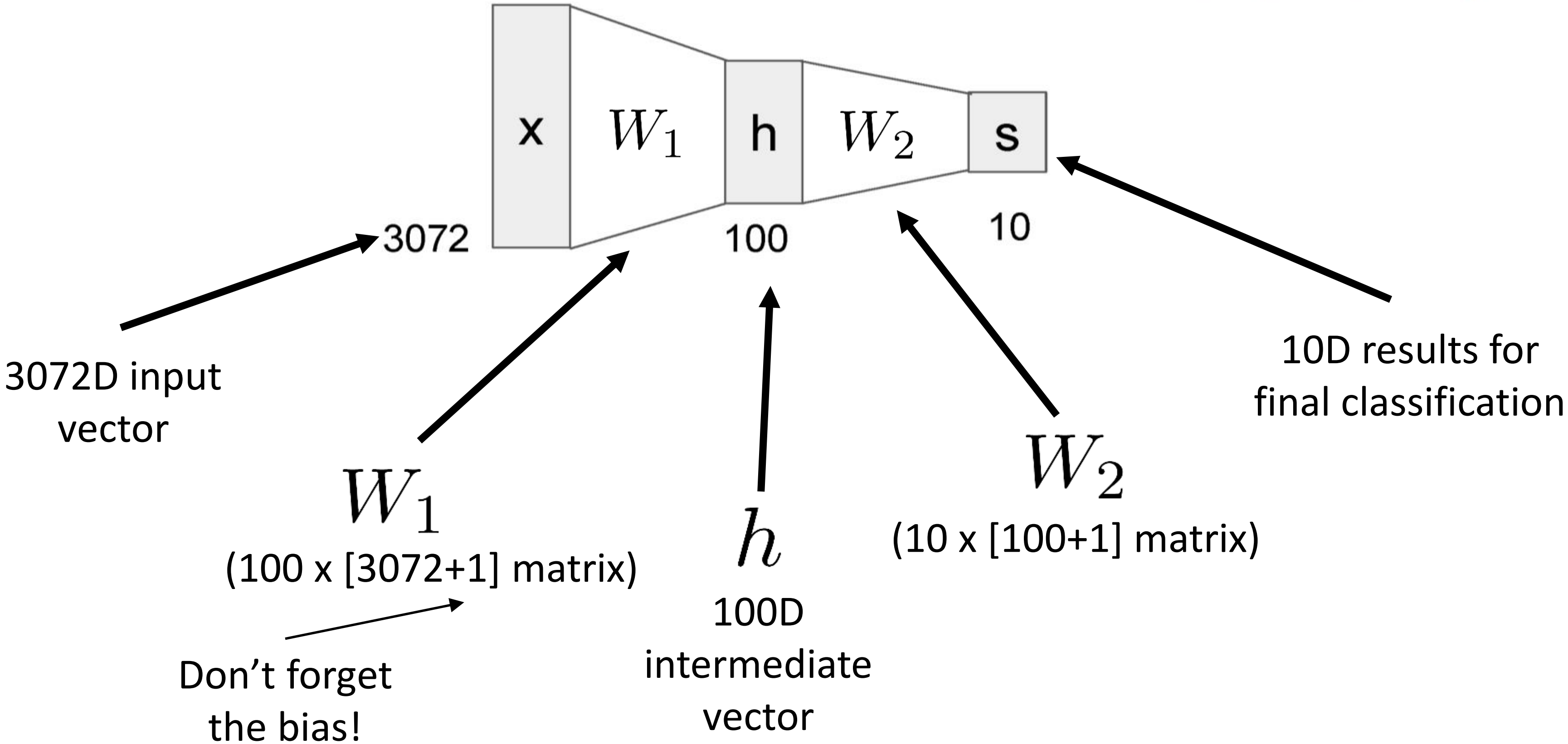
- We've gotten a linear separator again… not good.
- Remember the activation function!

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



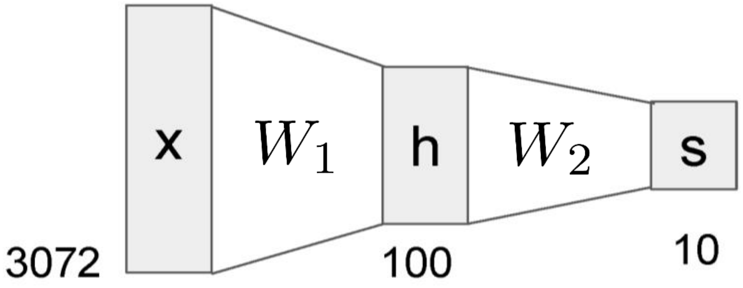
Multi-layer NN

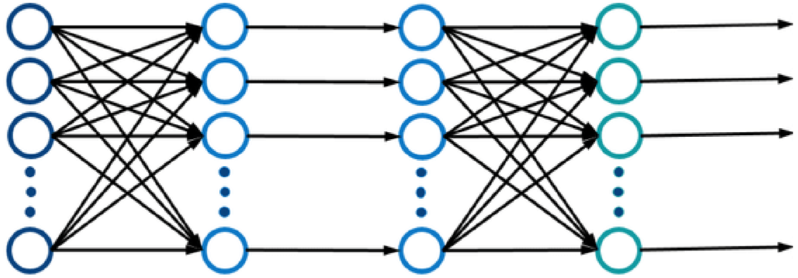
- Total number of weights to learn:

  [3,072+1] x 100 + [100+1] x 10 = 308,310

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

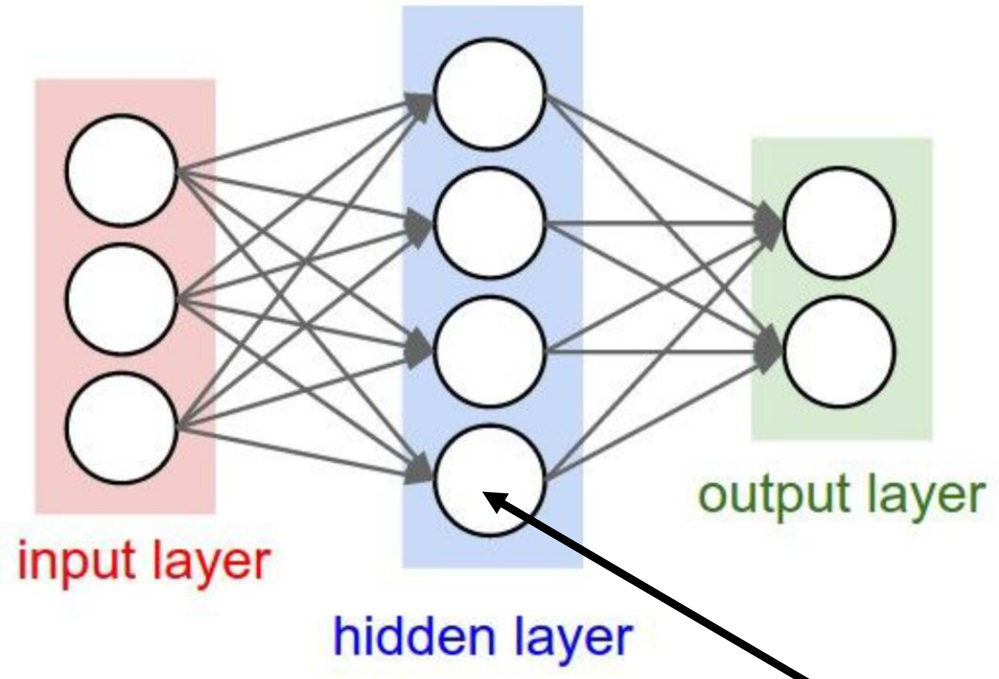
(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$





Neural network architecture

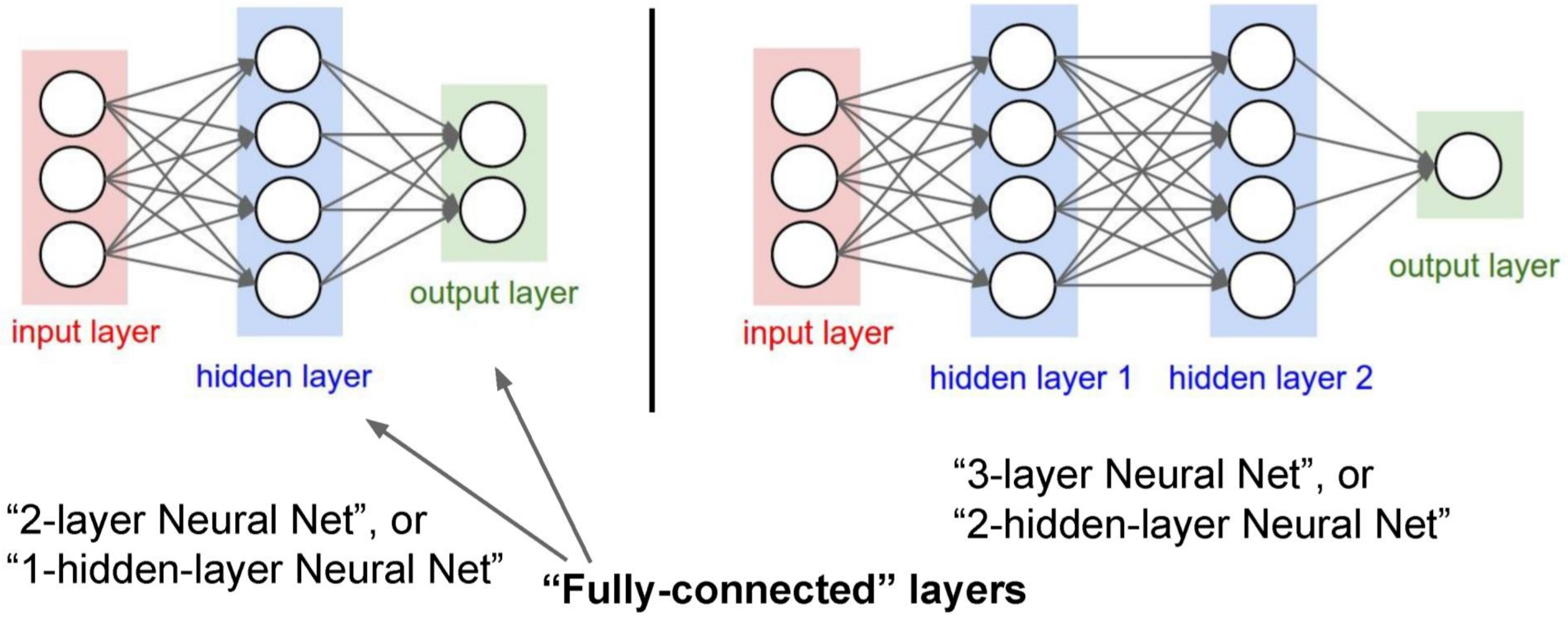
- Computation graph for a **2-layer neural network.**
  - Only count layers with tunable weights (so don't count the input layer).
  - Each layer is built from perceptrons: weights + bias + activation function.



One Neuron/ perceptron

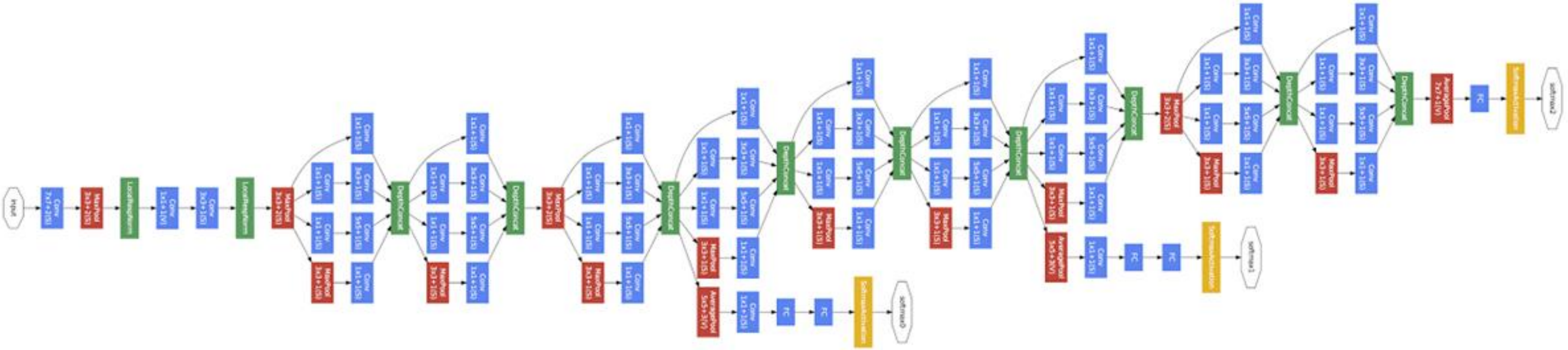
Neural network architecture

- **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 (GoogLeNet, Szegedy et al, 2015)
- 22 layers

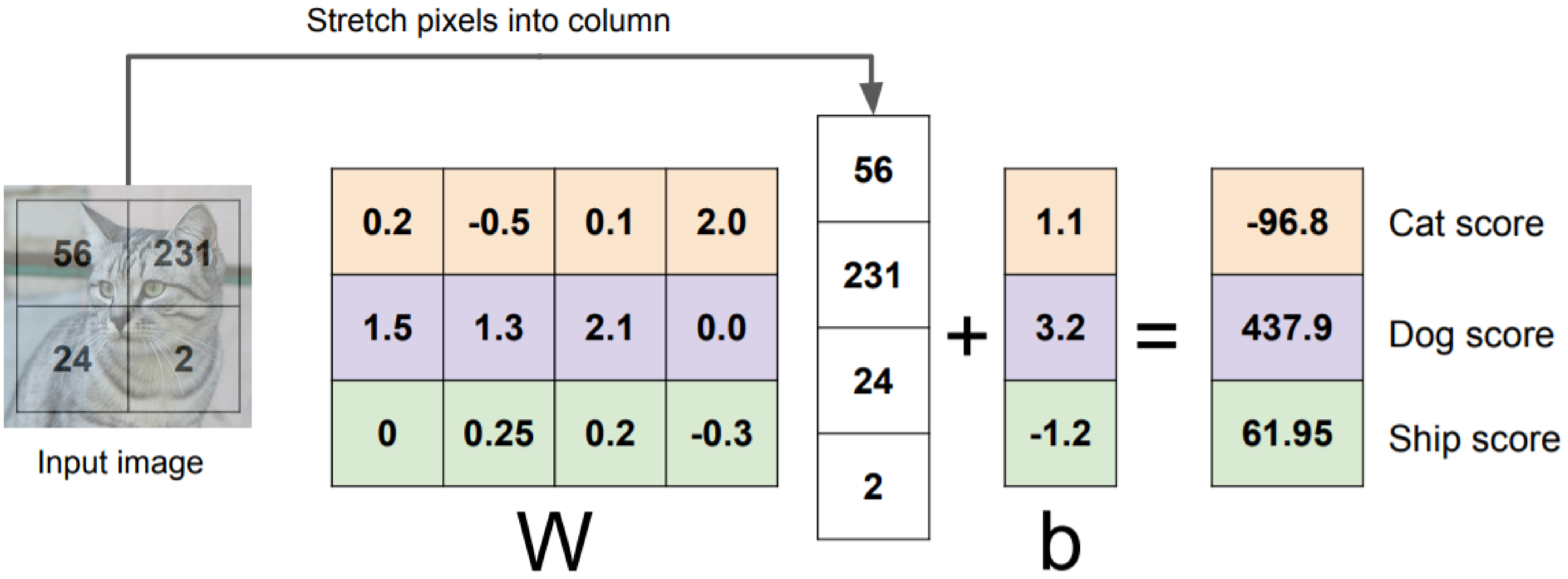


contents

- The classification problem- again
- NN history
- Perceptron
  - Hyperplanes
  - Activation
- Dense layer
- Multi-layer perceptron (MLP)
- **Optimization**
  - **Softmax + cross entropy + loss**
  - Gradient descent
- Basic data preprocessing
  - Data normalization
  - Train, validation and test splits

Optimizing the weights

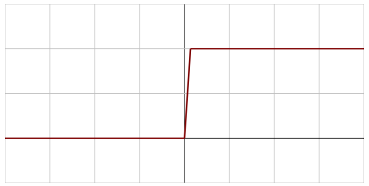
- We have this results for each possible label.
- which is the best result currently? Which should be the best result?



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.

-96.8	Cat score
437.9	Dog score
61.95	Ship score

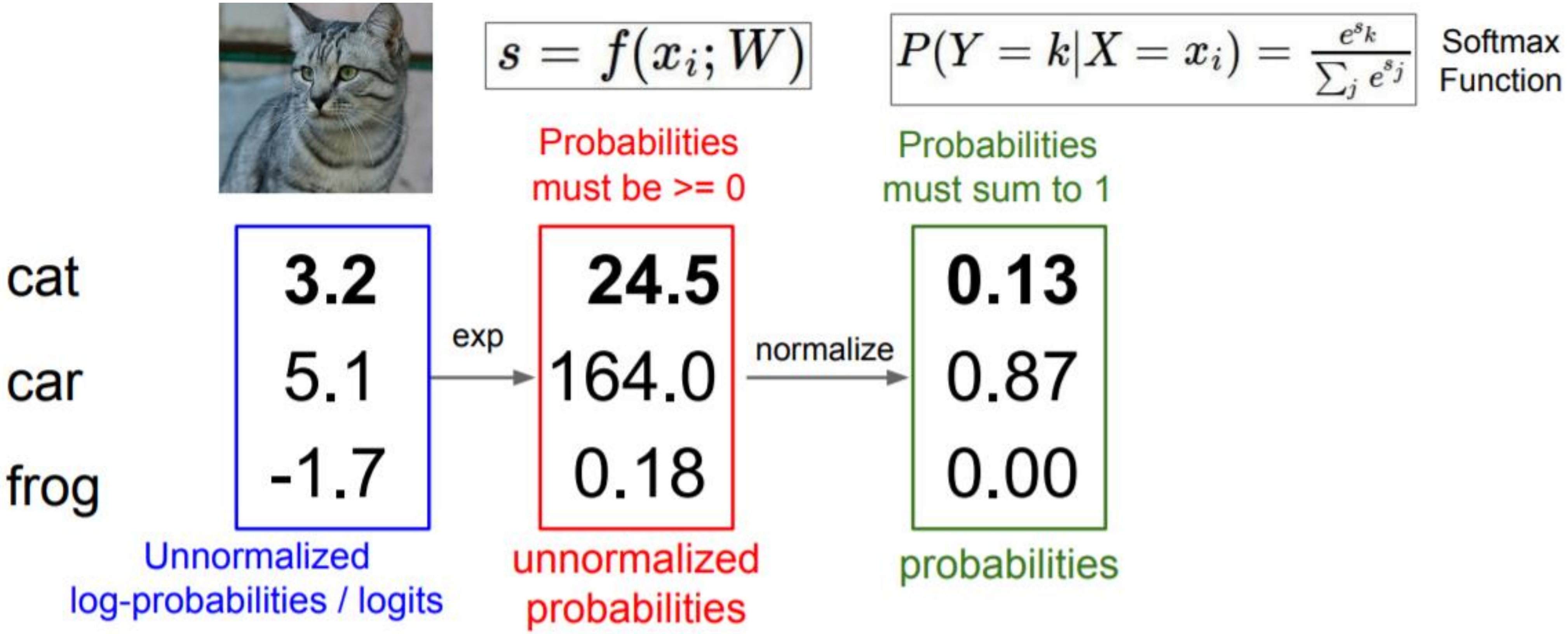


0
1
1

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

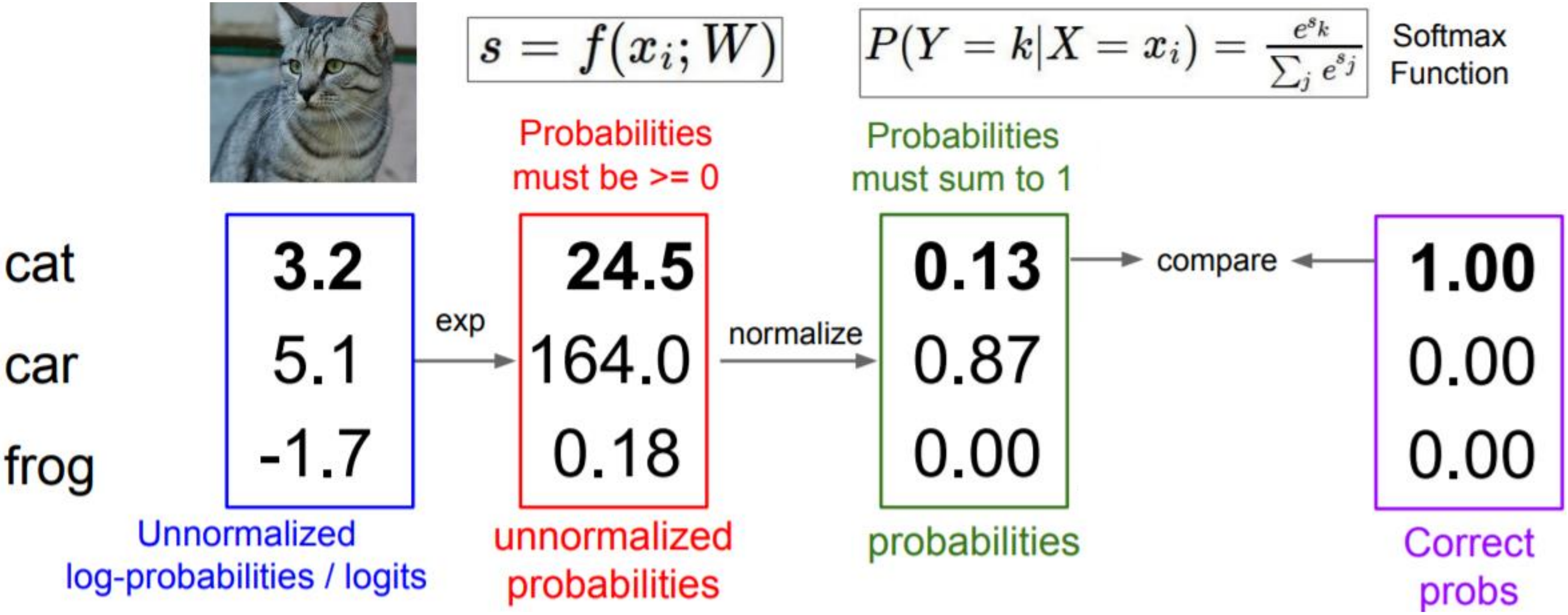
Softmax layer

- The softmax layer normalizes all the results so that you get a percentage of correctness for each label and **in use with the classification problem**.
- 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

- **Only during training time**, we need to define an error of the given probabilities and the correct (wanted) probabilities.
- A known loss function for the classification problem is called **cross entropy loss.**



Cross entropy loss + softmax

- Cross entropy is a way to measure "distance" between the wanted distribution of results p and given distribution of results q:

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

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

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

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

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

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 $\boldsymbol{W}$ that minimizes $\boldsymbol{L}$.**
- 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: $\nabla_W L$

contents

- The classification problem- again
- NN history
- Perceptron
  - Hyperplanes
  - Activation
- Dense layer
- Multi-layer perceptron (MLP)
- Optimization
  - Softmax + cross entropy + loss
  - **Gradient descent**
- Basic data preprocessing
  - Data normalization
  - Train, validation and test splits

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 in order to naively find all extremum points.
- An approximate solution to find best W is called **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.**

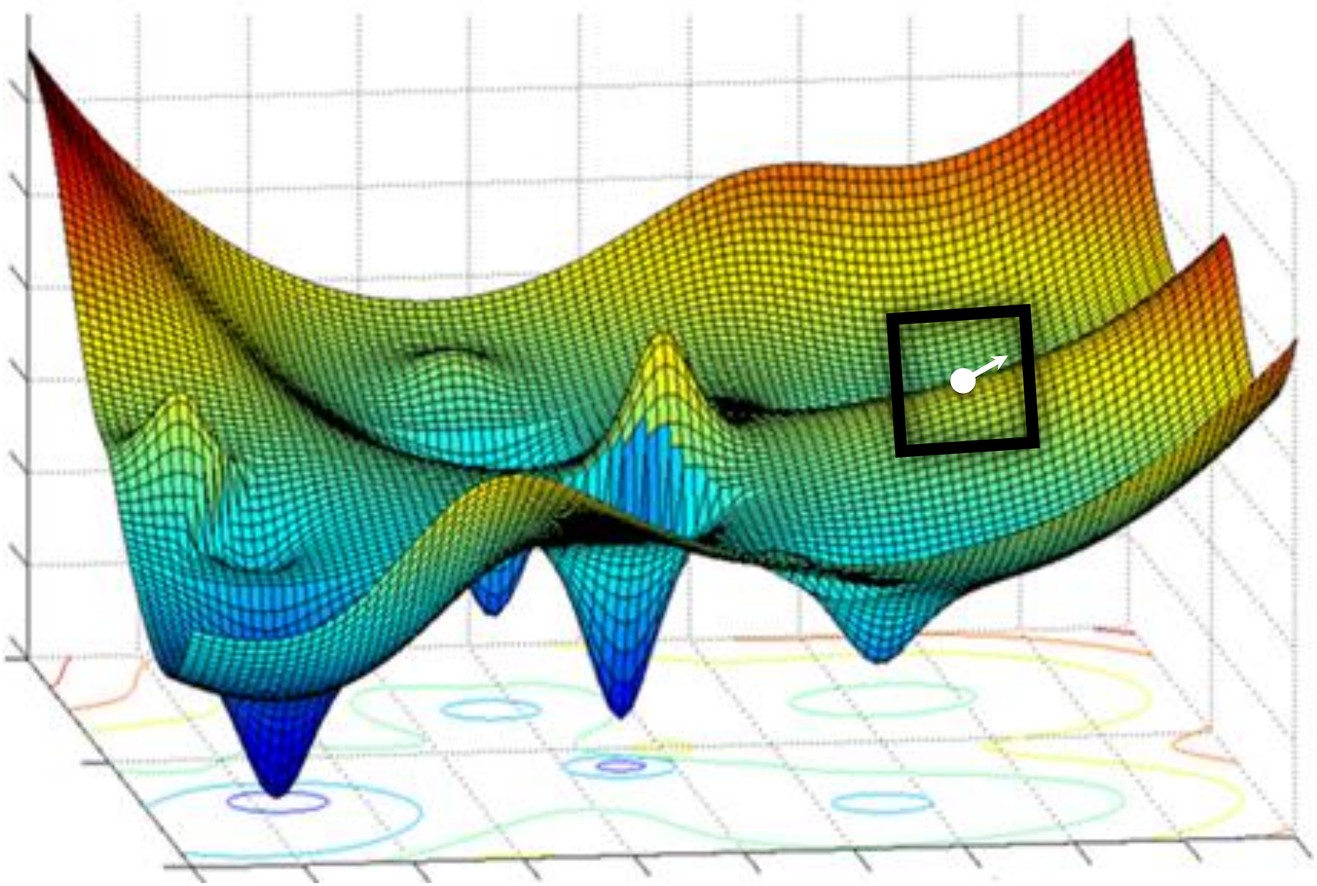
Finding the best W

- 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 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:

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

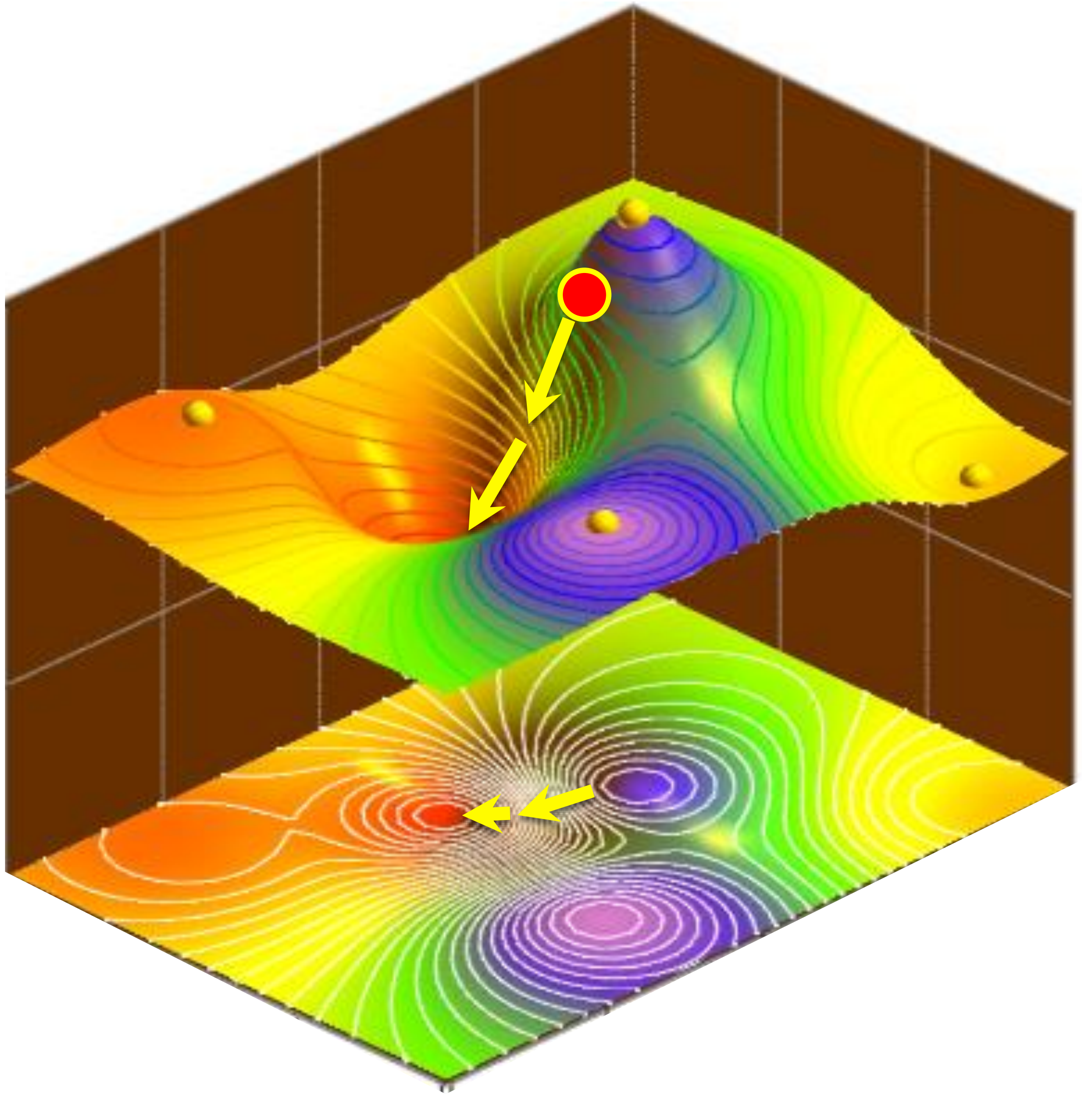


Gradient descent

- 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:

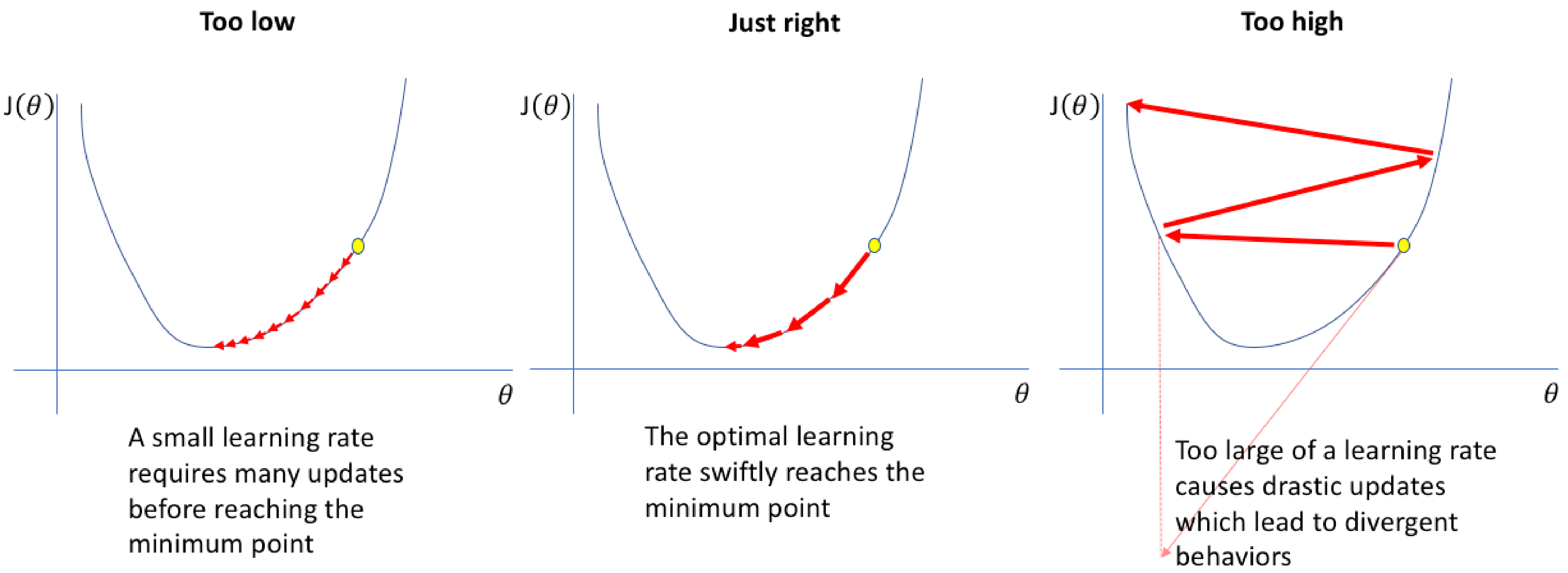
$$x_{n+1} = x_n - \eta \cdot \nabla f(x_n)$$

  – "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**.
- This is known as a **hyperparameter:** an unknown variable that is configured by the user (unlike the weights W which the system "learns").
- The learning rate can change over time- after several steps you can make the step size smaller for finer results (this is known as **learning rate decay**).



Examples of learning rates

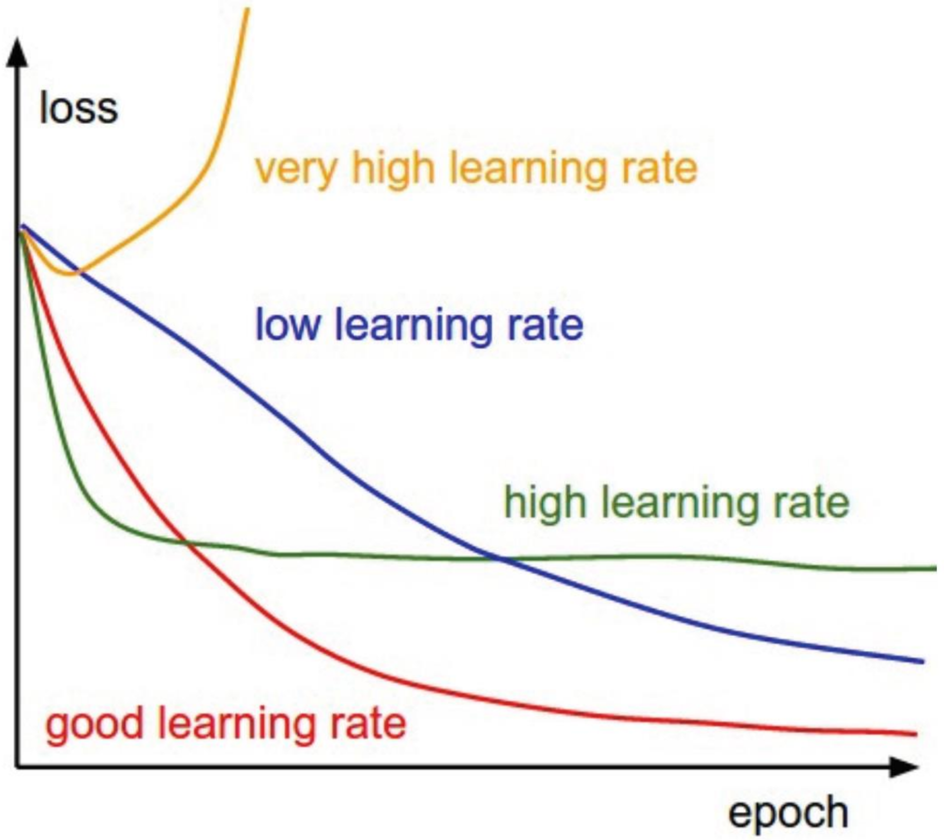


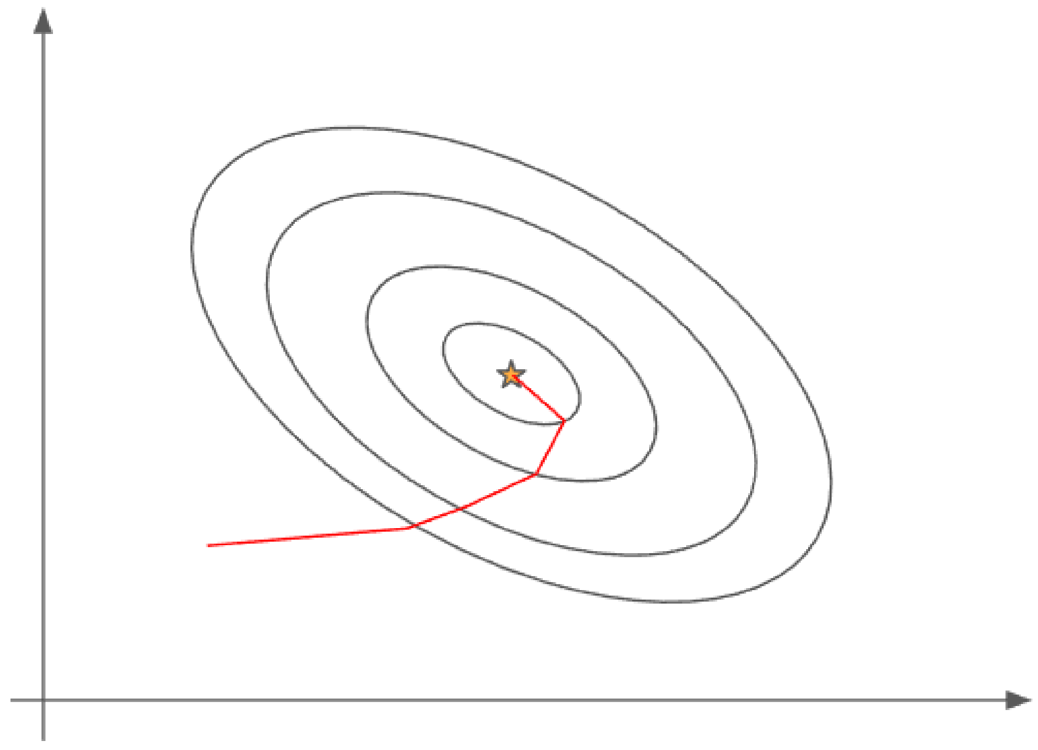
Figure: Andrej Karpathy

Gradient descent- local minima

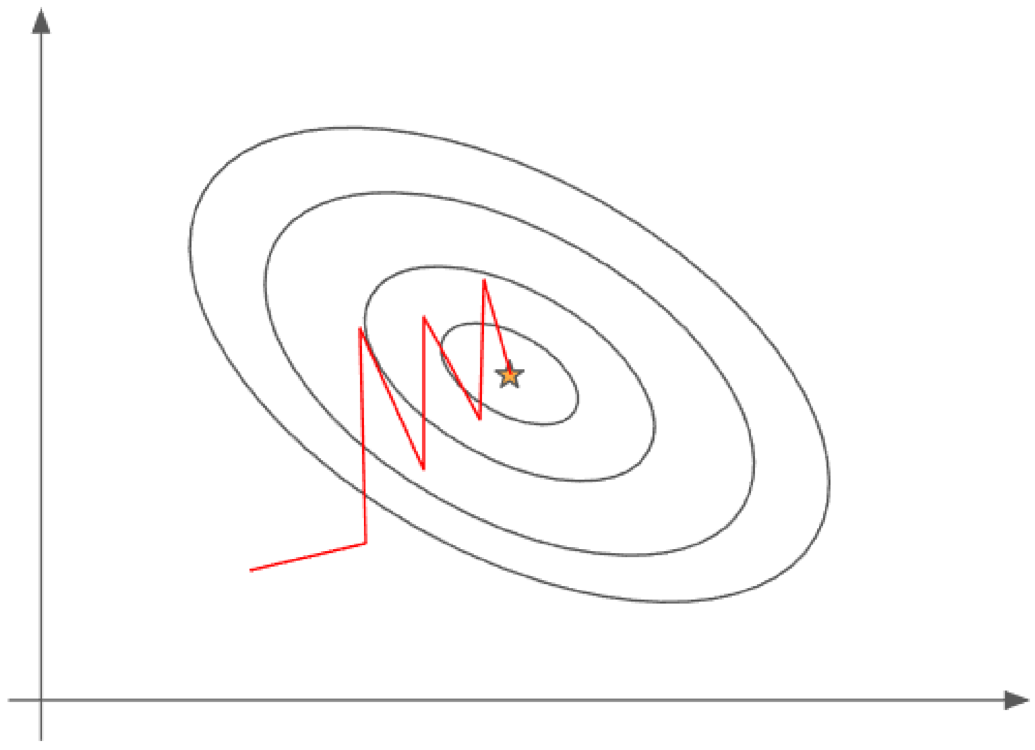
- An iterative algorithm for finding local minima of functions.
- we can initiate this procedure several times from several random staring points and take the minimum of all output minimum points- this way we can get a better result.

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

Loss noise

Typical training loss:

Why is it varying so rapidly?

The width of the curve is related to the batchsize — if too noisy, increase the batch size

Possibly too linear
(learning rate too small)

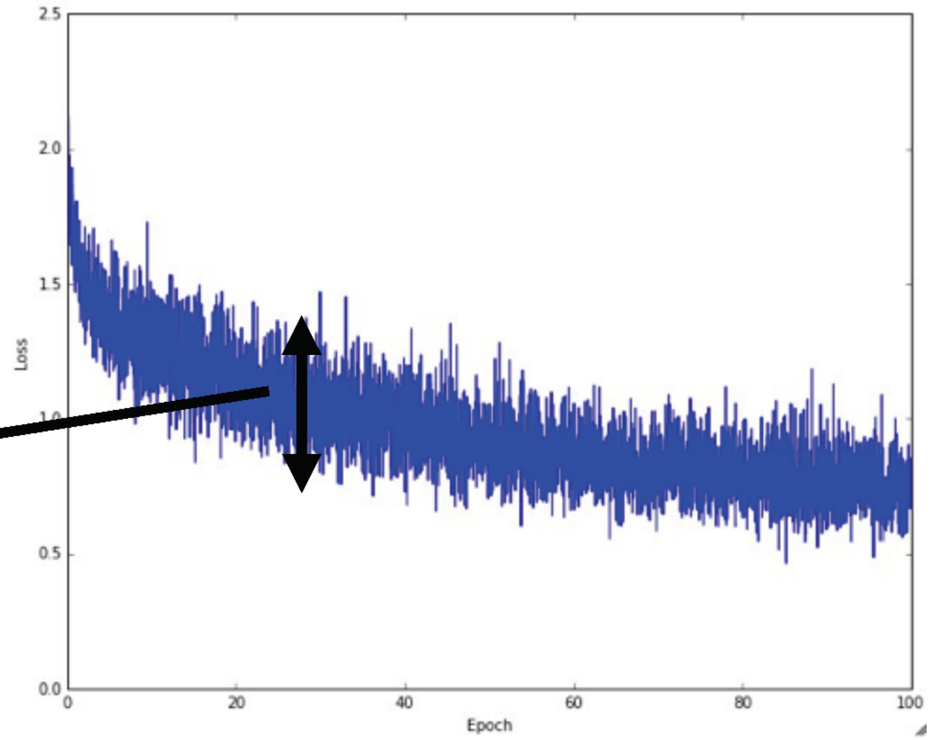


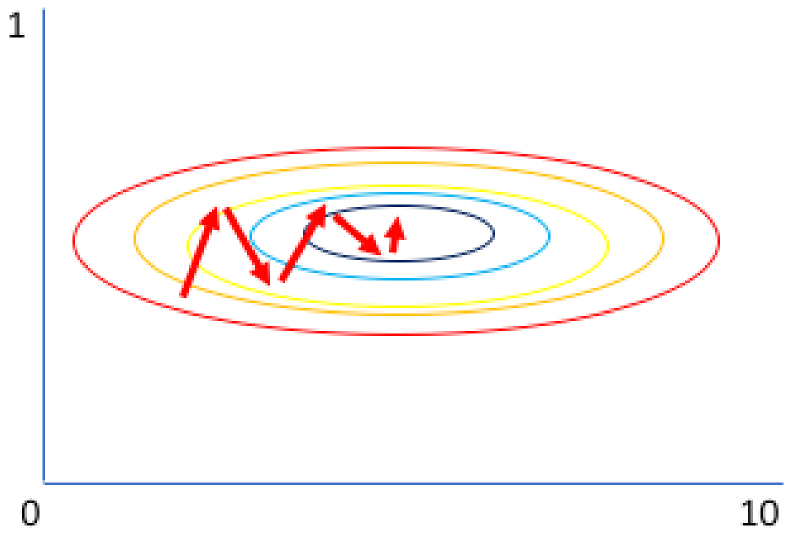
Figure: Andrej Karpathy

contents

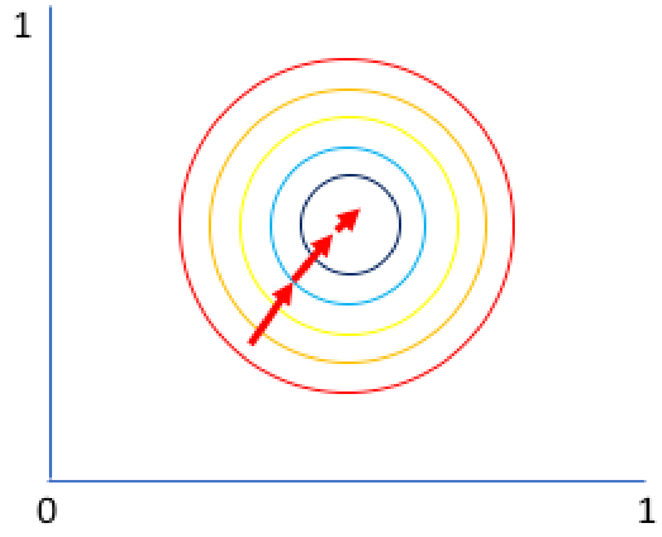
- The classification problem- again
- NN history
- Perceptron
  - Hyperplanes
  - Activation
- Dense layer
- Multi-layer perceptron (MLP)
- Optimization
  - Softmax + cross entropy + loss
  - Gradient descent
- **Basic data preprocessing**
  - Data normalization
  - Train, validation and test splits

Data normalization

- Assuming 2D input data with different scales ($x_1 \in [0,1], x_2 \in [0,1000]$)
- The weights needed to make x_1 significant as x_2 are much larger and hence make the loss function ellipsoid in one direction.
- This will cause the gradient descent method to converge in more steps than if the two axis where at the same scale.



Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

Data normalization

- In order to overcome this, we shall normalize the data before the entrance to the NN:

$$\mu = \frac{1}{m}\sum_{i=0}^{m} x_i \,, \sigma^2 = \frac{1}{m-1}\sum_{i=0}^{m}(x_i - \mu)^2$$

$$\widetilde{x_i} = \frac{x_i - \mu}{\sigma^2}$$

- **This should be done for each dimension of the input vector independently.**
- The test data should be normalized with the same variables found in the train data.
- This is a common practice to do even if the data are at the same scale for all dimensions since the default hyperparameters for all NN are based on such normalized data.

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**.

- Fully connected colab