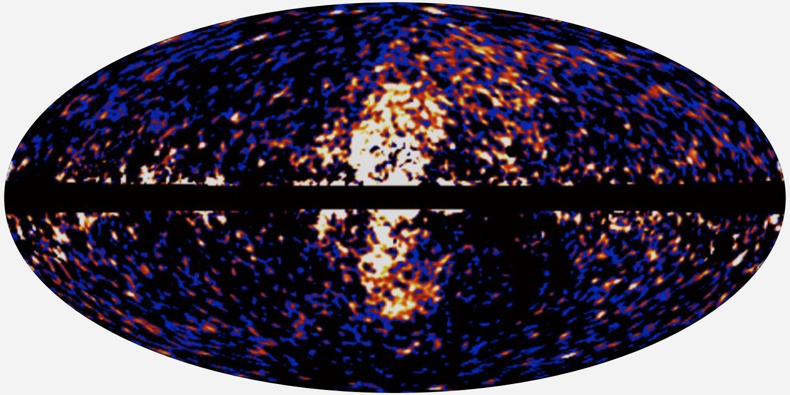
# Machine Learning and Artificial Intelligence in Physics (and beyond)

## Overview and Applications

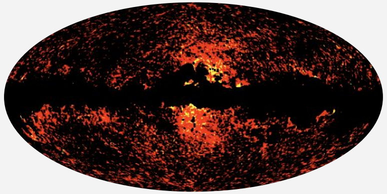
**Greg Dobler**
*Biden School of Public Policy and Administration*
*Department of Physics and Astronomy*
*Data Science Institute*
gdobler@udel.edu
@gregorydobler

## the *Fermi* "Haze/Bubbles"

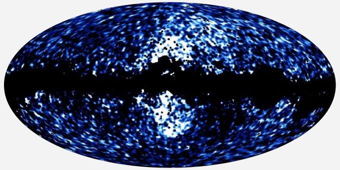


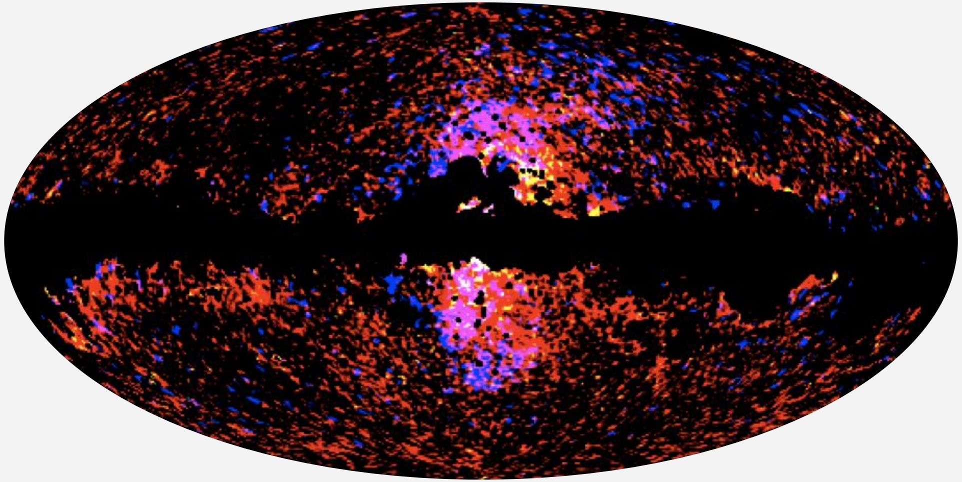
**Dobler**, *et al., 2010. "The Fermi haze: a gamma-ray counterpart to the microwave haze", ApJ, 717, 2*

*Plank* microwaves



*Fermi* gamma-rays





*"Detection of the Galactic haze with Planck"*

*Ade,…,* ***Dobler****, et al, 2013. Planck intermediate results-IX. A&A, 554, A139.*

**Greg Dobler**

*Biden School of Public Policy and Administration*

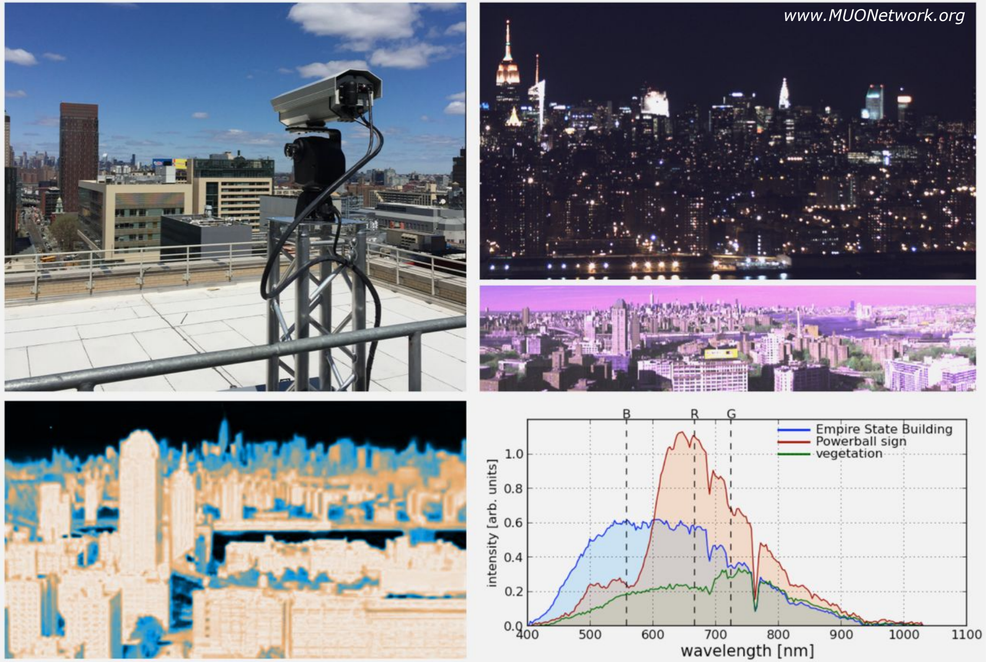
*Department of Physics and Astronomy*

*Data Science Institute*

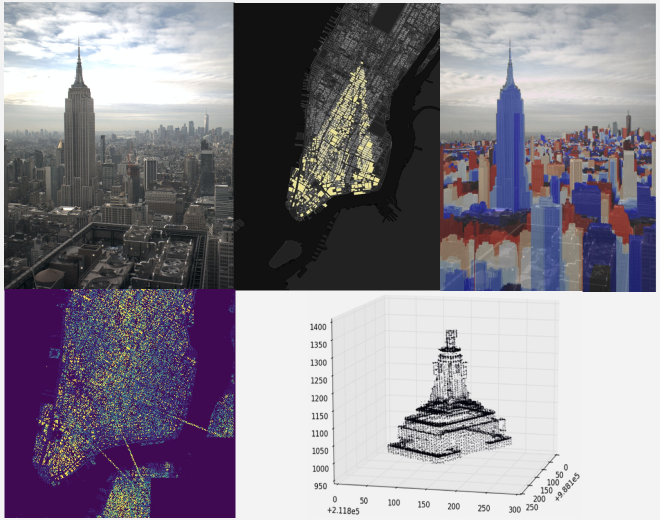
gdobler@udel.edu

@gregorydobler

# The Urban Observatory A Multi-Modal Imaging Platform for the Study of Dynamics in Complex Urban Systems




*Dobler*, et al., 2021. Remote Sensing, 13(8), p.1426.

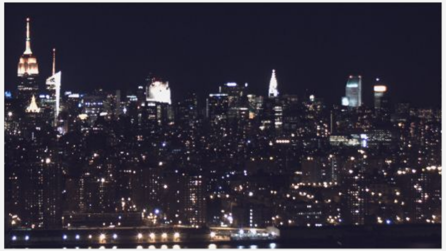


Remote imaging data is fused with available records data via photogrammetric techniques to geo-locate patterns of activity and associated anomalies form minute to diurnal to weekly to yearly timescales.

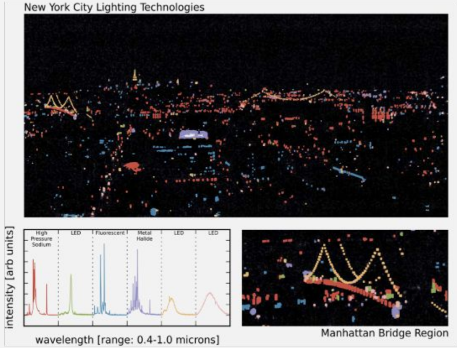
## ENERGY USE, EFFICIENCY, ENVIRONMENT, RESILIENCE, SUSTAINABILITY

# The Urban Observatory *A Multi-Modal Imaging Platform for the Study of Dynamics in Complex Urban Systems*

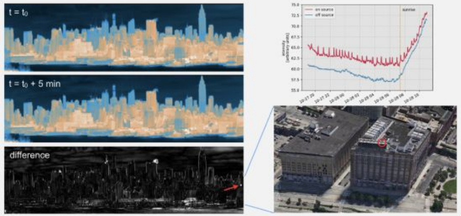
## ENERGY USE, EFFICIENCY, RESILIENCE



**Health of the Power Grid:** The lights in UO images at night flicker due to the 60Hz mains frequency. We can measure the stability of this frequency to monitor the grid in real time for early warning signs of power outages. At the same time, on/off changes in the lighting provide estimates for total energy end use.

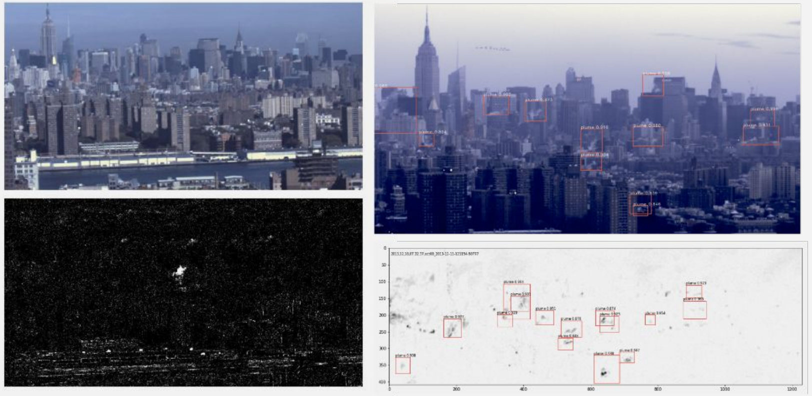



**Lighting Technology Adoption:** The UO camera systems are capable of determining the lighting type (incandescent, LED, fluorescent, etc.) for every bulb in our field of view. This provides information on efficient technology adoption, change-over compliance, target of opportunity identification for lighting upgrades, and measures of lighting inefficiencies.



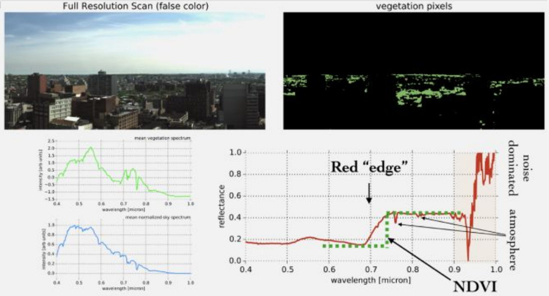

**Remote Building Thermography:** With the UO's infrared imaging capabilities, we can perform thermographic studies of 100s of buildings simultaneously providing key indicators of thermal inefficiencies, energy loss, and assessment of heating/cooling systems.

## ENVIRONMENT AND SUSTAINABILITY



**Remote Detection of Air-Borne Pollutants:** The UO has developed the analytic capability of automatically detecting soot plumes ejected from buildings in near-real time, providing a method for determining the environmental impacts of energy use in cities, monitoring for compliance, and providing situational awareness in the event of a disaster or toxic materials release.

**Urban Vegetative Health:** Using the same technology that we have developed to determine light bulb types at night, daytime imaging allows us to monitor the health of plants in our field of view, correlating with local air quality to ensure robust public green spaces.




**Dobler**, *et al.*, *2021. Remote Sensing, 13(8), p.1426.*

# what is Machine Learning?

Dobler's very broad definition:

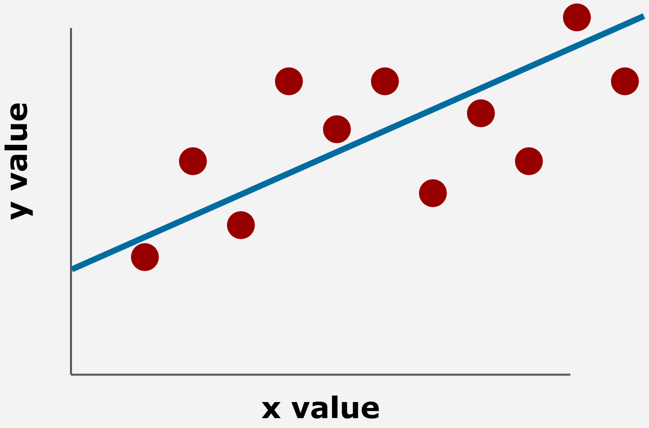
> machine learning is any process by which a parameter is algorithmically determined from data by a computer*

"determine" in this context means that there is some objective metric to be optimized

**advances in machine learning + "the data revolution" → revolutionary shifts in science**
**&**
**society**

** if you've ever fit a straight line to data you've [probably] already executed a machine learning task*

# Linear Regression in 1D




In linear regression, we **model** the relationship between a dependent variable (y value) and independent variable (x value) with a linear function. For example,

$$\mathbf{y_i = m\ x_i + b}$$

In order to "fit" a linear model to data, we must define a **metric** that indicates the goodness-of-fit; the sum of squared differences (ssd) is typical:

$$\mathbf{ssd = \Sigma_i\ (y_i - (m\ x_i + b))^2}$$

With multi-linear regression, the dependent variable is modeled as a linear combination of multiple independent variables,

$$\mathbf{y_i = a_0 + a_1\ x_{1,i} + a_2\ x_{2,i} + a_3\ x_{3,i} + \ldots}$$

the **Normal Equation**

$$\vec{y} = \mathbf{P}\,\vec{a}$$

$$(\mathbf{P}^T\mathbf{P})^{-1} \cdot \mathbf{P}^T\,\vec{y} = \vec{a}$$

# Linear Regression: an Algorithmic Solution

An alternative to the matrix-based solution for determining the best fit parameters of a linear model are numerical, algorithmic solutions.

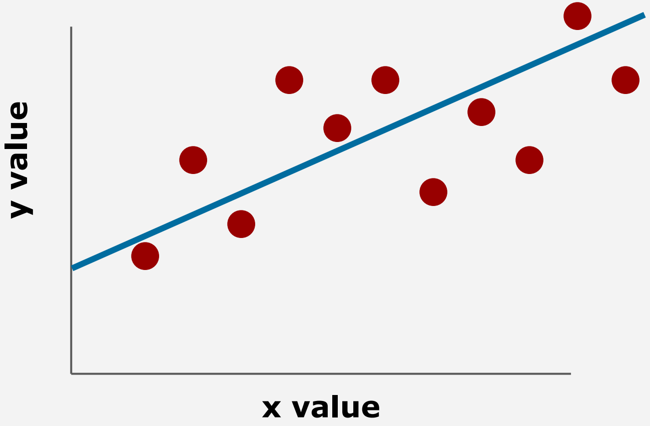
For this framework, we define our **metric** as either,

**Minimize** **COST (or LOSS) FUNCTION** : C = sum of squared differences
(potentially weighted by the squared "errors" as in $\square^2$)

**or**

**Maximize** **LIKELIHOOD** : $\mathscr{L} = e^{-C/2}$

But how do we actually <u>find</u> the values for the parameters (slope and offset) that minimize the cost function?



# Training a Linear Model with Gradient Descent

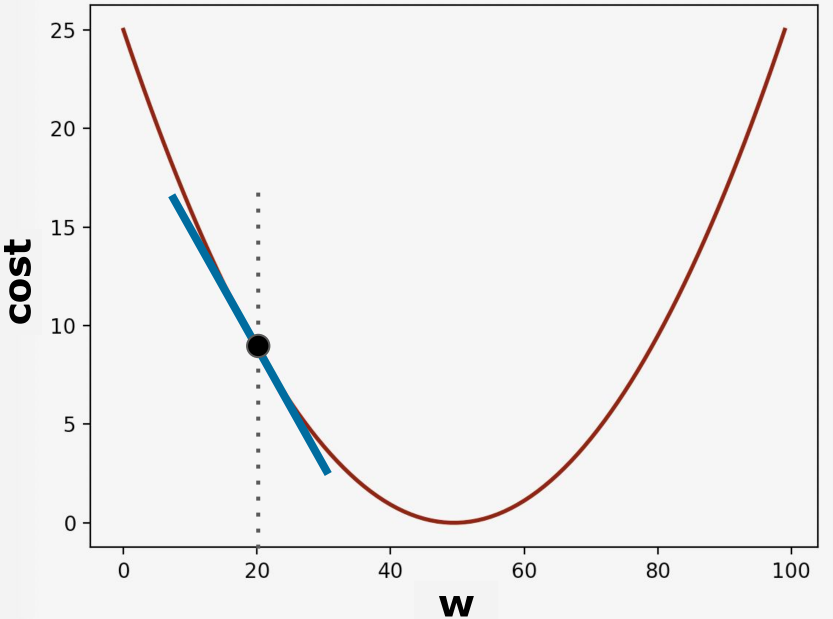
$$a = w \cdot x$$

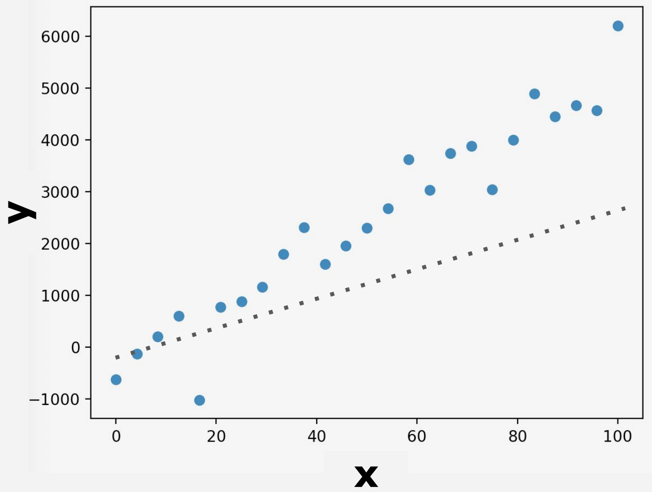
$$C = \sum_i (y_i - a_i)^2 = \sum_i (y_i - w \cdot x_i)^2$$

$$\frac{\partial C}{\partial w} = -2 \sum_i x_i \cdot (y_i - w \cdot x_i)$$

algorithm

1. choose initial $w$
2. update with $w' = w - \eta \frac{\partial C}{\partial w}$
3. repeat step 2 until convergence



# Training a Linear Model with Gradient Descent

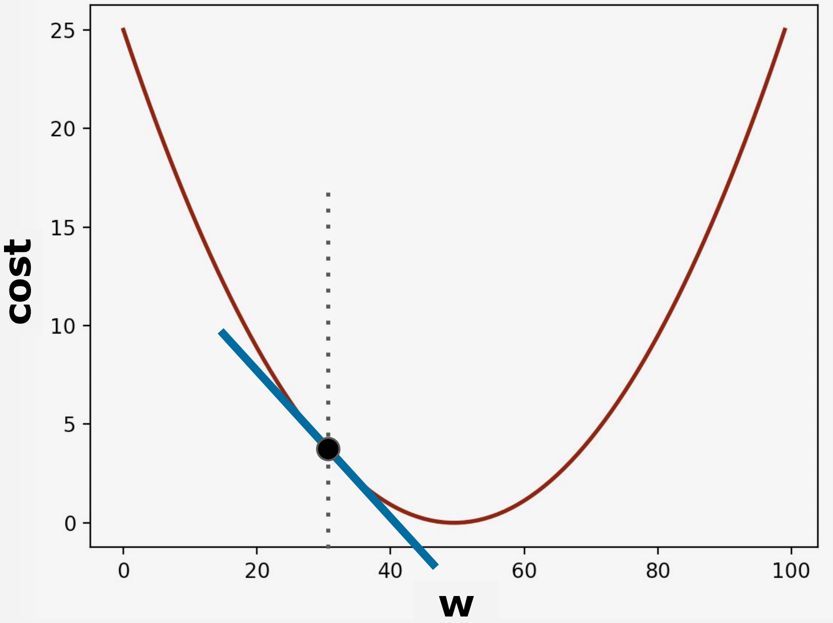
$$a = w \cdot x$$

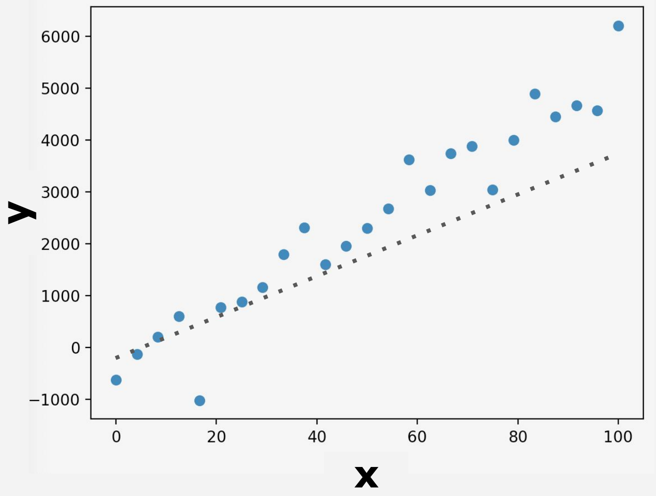
$$C = \sum_i (y_i - a_i)^2 = \sum_i (y_i - w \cdot x_i)^2$$

$$\frac{\partial C}{\partial w} = -2 \sum_i x_i \cdot (y_i - w \cdot x_i)$$

algorithm

1. choose initial $w$
2. update with $w' = w - \eta \frac{\partial C}{\partial w}$
3. repeat step 2 until convergence



# Training a Linear Model with Gradient Descent

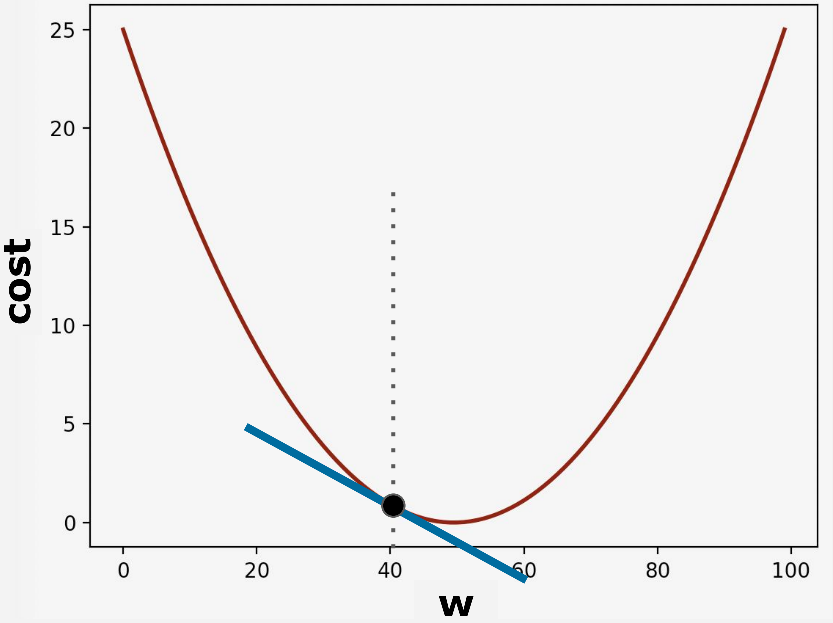
$$a = w \cdot x$$

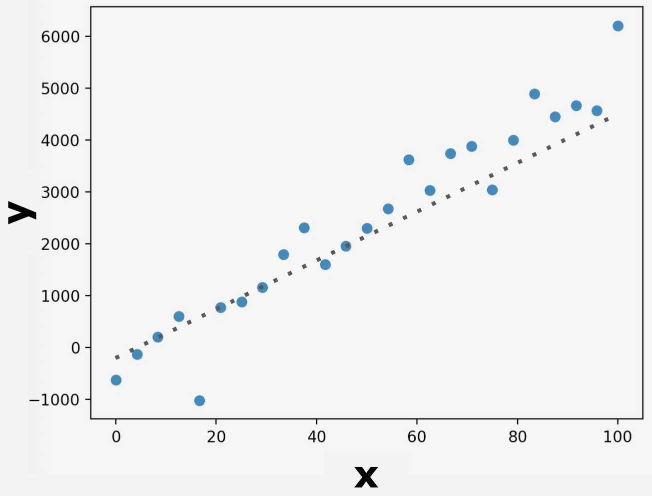
$$C = \sum_i (y_i - a_i)^2 = \sum_i (y_i - w \cdot x_i)^2$$

$$\frac{\partial C}{\partial w} = -2 \sum_i x_i \cdot (y_i - w \cdot x_i)$$

algorithm

1. choose initial $w$
2. update with $w' = w - \eta \frac{\partial C}{\partial w}$
3. repeat step 2 until convergence



# Training a Linear Model with Gradient Descent

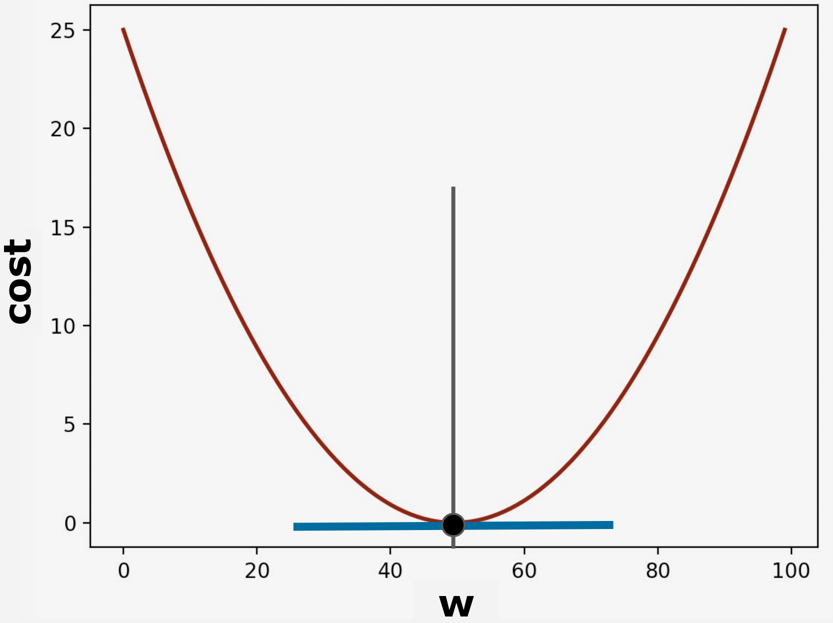
$$a = w \cdot x$$

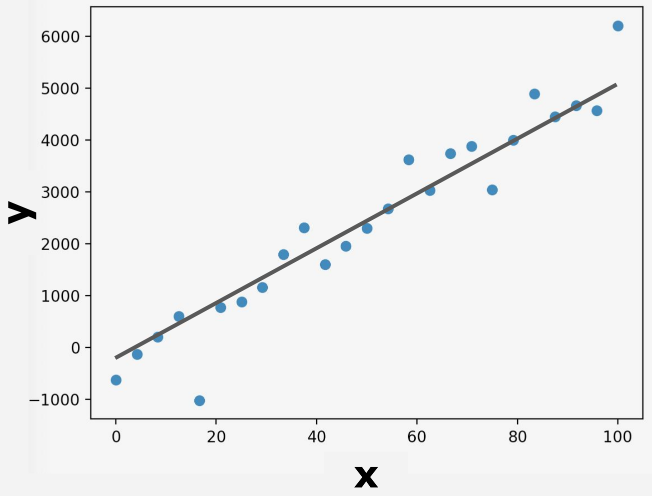
$$C = \sum_i (y_i - a_i)^2 = \sum_i (y_i - w \cdot x_i)^2$$

$$\frac{\partial C}{\partial w} = -2 \sum_i x_i \cdot (y_i - w \cdot x_i)$$

algorithm

1. choose initial $w$
2. update with $w' = w - \eta \frac{\partial C}{\partial w}$
3. repeat step 2 until convergence



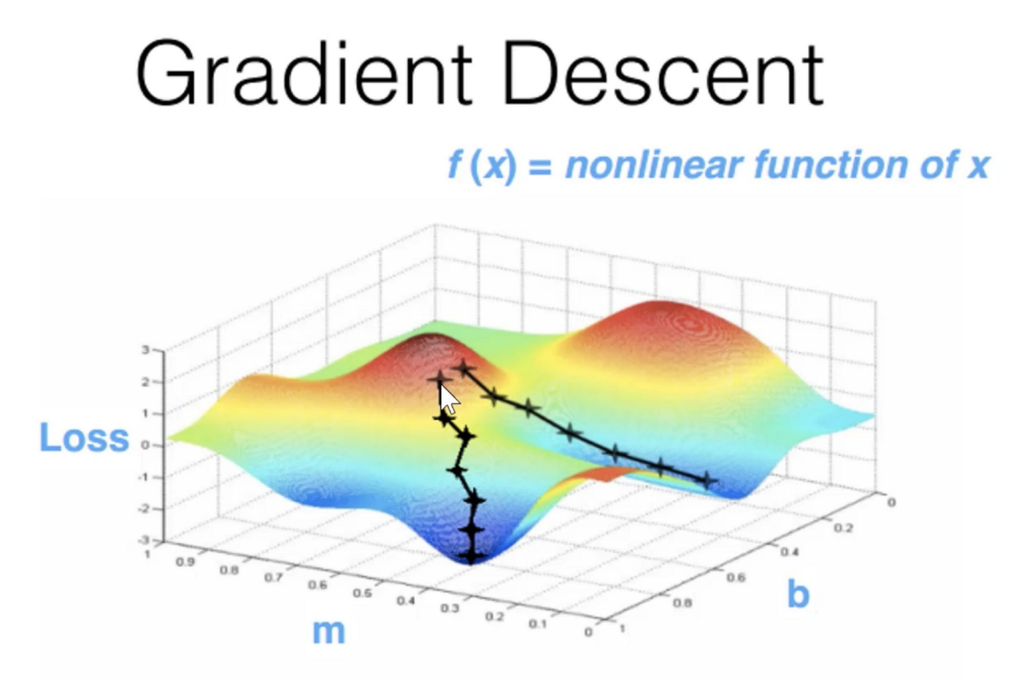



# Learning the slope and offset using SGD

**Stochastic Gradient Descent** (SGD) use a different random subset of data at each iteration

1. choose initial values for m and b
2. calculate the LOSS
3. determine the direction of the steepest gradient
4. step in that direction
5. go back to step 2

https://medium.com/@julian.harris/stochastic-gradient-descent-in-plain-english-9e6c10cdba97




THINGS TO CONSIDER

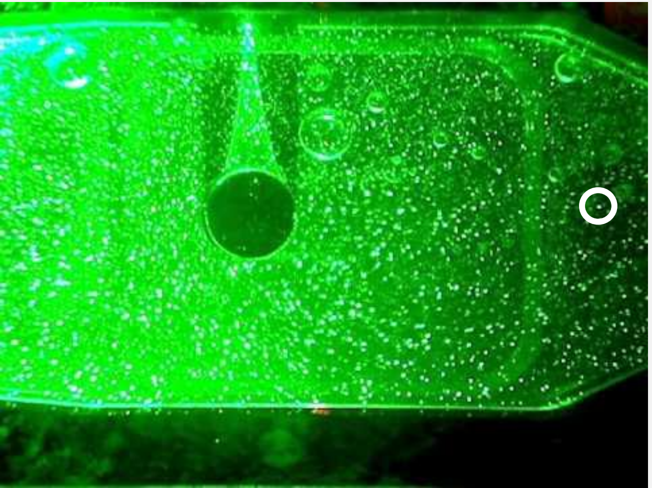
| | |
|---|---|
| choosing starting point? | initialization |
| how far to step? | learning rate |
| take the step? | dealing with local minima |
| when to stop? | stopping criterion change in loss |

# the first Machine Learning paradigm: objects and features

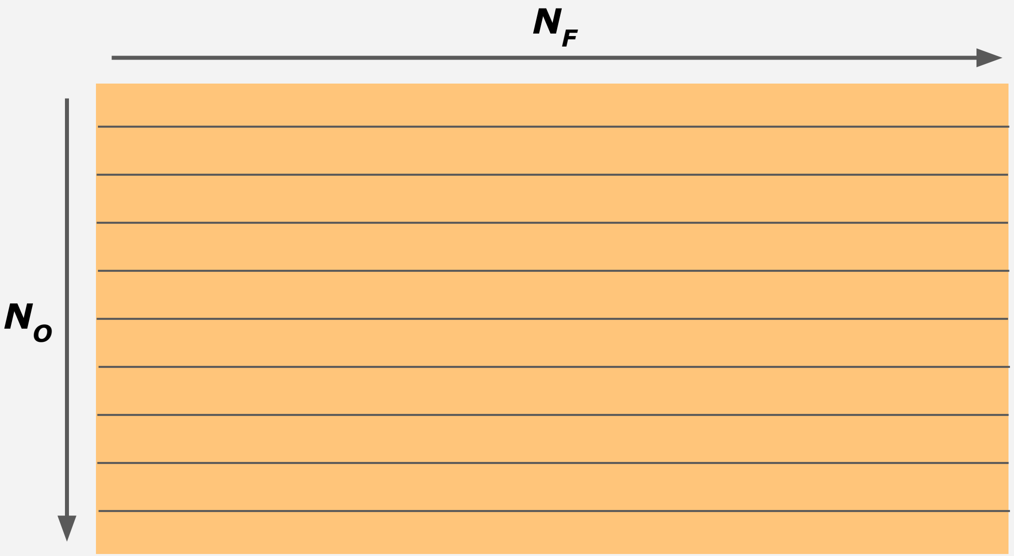
# Objects and Features

in many data analysis tasks in, we consider data as a number as a function of another number characterizing a system, e.g.:

- brightness of SN Ia vs distance
  characterizing the Universe
- height of the oceans vs time
  characterizing the climate
- velocity of tracer particles vs position
  characterizing fluid flow



**in ML, the central data paradigm is one in which the data is represented by objects each of which has associated features**



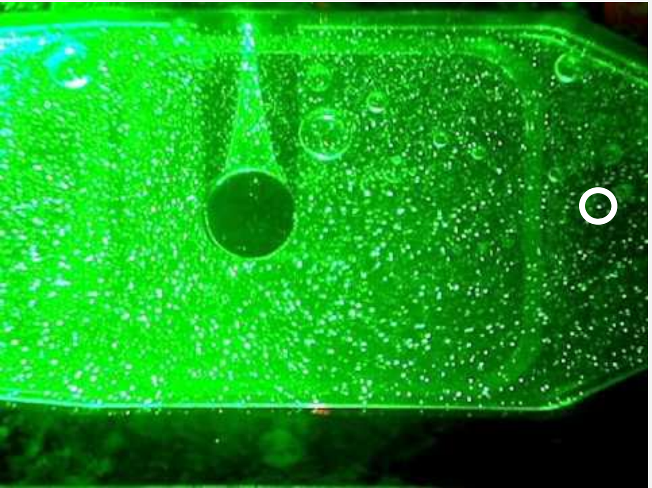

**features:** ***[x, y, $v_x$, $v_y$, radius, charge, material, topology]***

continuous | ordered | categorical

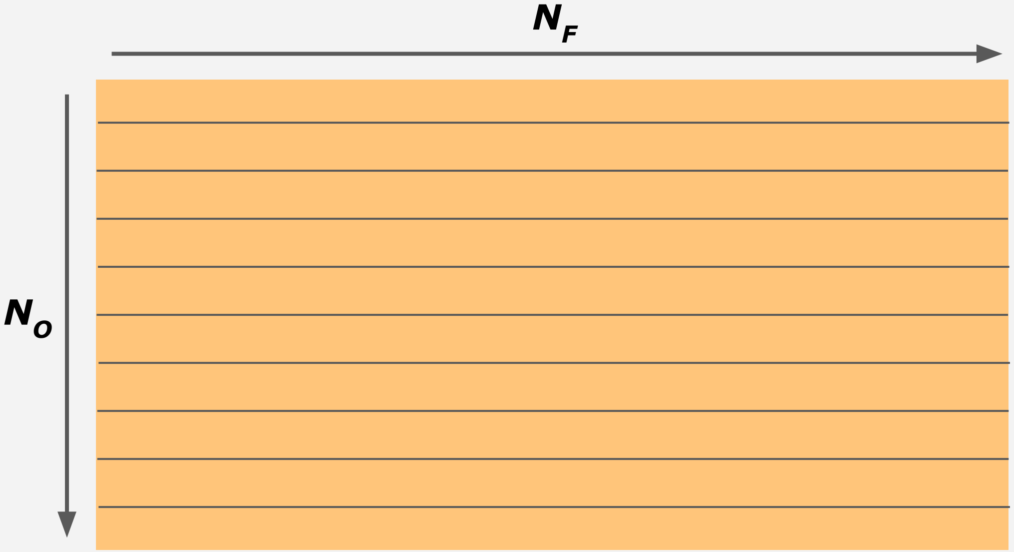
# Objects and Features

in many data analysis tasks in, we consider data as a number as a function of another number characterizing a system, e.g.:

- brightness of SN Ia vs distance
  characterizing the Universe
- height of the oceans vs time
  characterizing the climate
- velocity of tracer particles vs position
  characterizing fluid flow



**in ML, the central data paradigm is one in which the data is represented by objects each of which has associated features**




**features:** $[x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}]$

continuous

# Supervised vs Unsupervised Learning

**Machine Learning algorithms broadly can be split into two categories:**

**supervised learning**: algorithm learns parameters from data using labeled examples to inform the metric for optimization

**unsupervised learning**: algorithm learns parameters from data without labeled examples of “truth”

---

| | **supervised** | **unsupervised** |
|---|---|---|
| **pro** | can generate highly specific, tailored models based on domain knowledge | no need for labeled data means that pattern recognition happens “automatically” |
| **con** | requires a large amount of labeled training data, typically done “by hand” | no guarantee that the outputs describe the data in a useful or relevant way |

# Common ML Models

## Unsupervised example: K-Means clustering

1. choose **k** initial cluster centers
2. assign each object to the nearest cluster center
3. update the cluster centers to be the average of their assigned population
4. calculate *inertia* = $\Sigma_c \Sigma_{j \in C} |\boldsymbol{x}_j - \boldsymbol{x}_c|^2$
5. IF the inertia has not changed, stop
   ELSE go to back to step 2.
6. go back to step 1 choose minimum inertia solution

*https://towardsdatascience.com/clustering-using-k-means-algorithm-81da00f156f6*

THINGS TO CONSIDER

| | |
|---|---|
| how to set k? | number of clusters |
| choosing starting spot? | initialization |
| optimal solution? | dealing with multiple solutions |
| restarting? | re-intializing with fixed k |

# Common ML Models

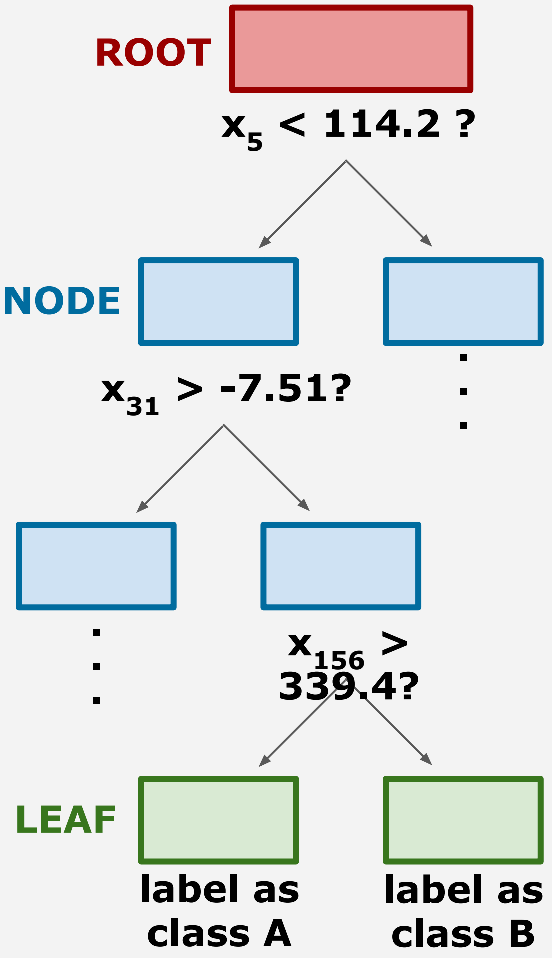
## Supervised example: Decision-Tree Classifier*

** this implementation assumes all features are continuous numerical*

1. for each feature determine the optimal partition threshold to minimize the *Gini "impurity"*, the sum of weighted ratios of various classes should the data be split into sub-populations
2. split the data into sub-populations according to the feature and partition threshold with the minimum impurity
3. for each sub-population, for each feature, determine the optimal partition threshold to minimize impurity should the data be split further
4. split the sub-population into sub-populations according to the feature and partition threshold with the minimum impurity
5. IF sub-populations are 100% pure, stop

   ELSE go to back to step 3.

potentially specify an alternative stopping criterion such as the minimum number of samples in a subpopulation



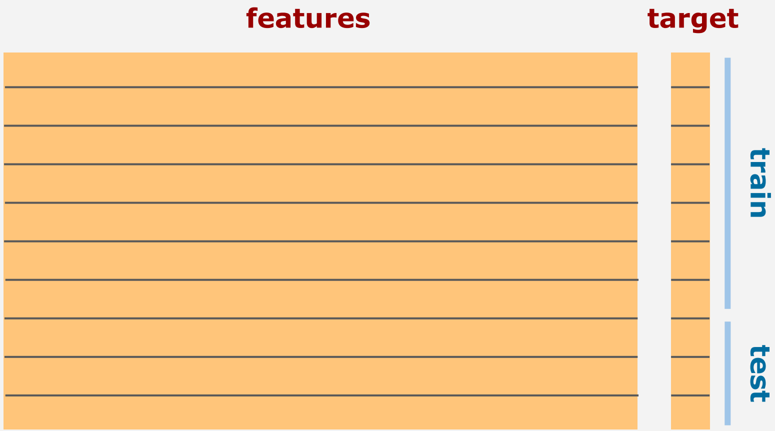
# the second Machine Learning paradigm: training/testing/validation

# Training and Evaluating Models within the ML Paradigm

Decision trees are an important construct, but they tend to suffer from overfitting.*

To understand overfitting, we first need to understand model accuracy for supervised learning models…

Consider a data set:




the most basic method to train a ML model splits the data into two categories,

**training data** (70–80%)

data on which the model parameters are fit by optimizing a metric

**testing data** (30–20%)

data on which the fit model predicts known values of the target

**these subsets are NEVER (ever) to be mixed**

** overfitting occurs when the accuracy on the training data is significantly higher than the accuracy on the testing data*

# Training and Evaluating Models within the ML Paradigm

Things to consider about training/testing sets:

**bias** – randomize <u>before</u> splitting (and be careful) to avoid training on one type of data while testing on another

**noise** – ensure that the noise characteristics are similar between the two data sets

**balance** – the full range of target variables should be represented in both training and testing sets

One of the most common issues is a subtle mixing between training and testing sets leading to invalid accuracy assessment.

the most basic method to train a ML model splits the data into two categories,

**training data** (70–80%)

data on which the model parameters are fit by optimizing a metric

**testing data** (30–20%)

data on which the fit model predicts known values of the target

**these subsets are <u>NEVER</u> (ever) to be mixed**

# Model Accuracy: training/testing with validation

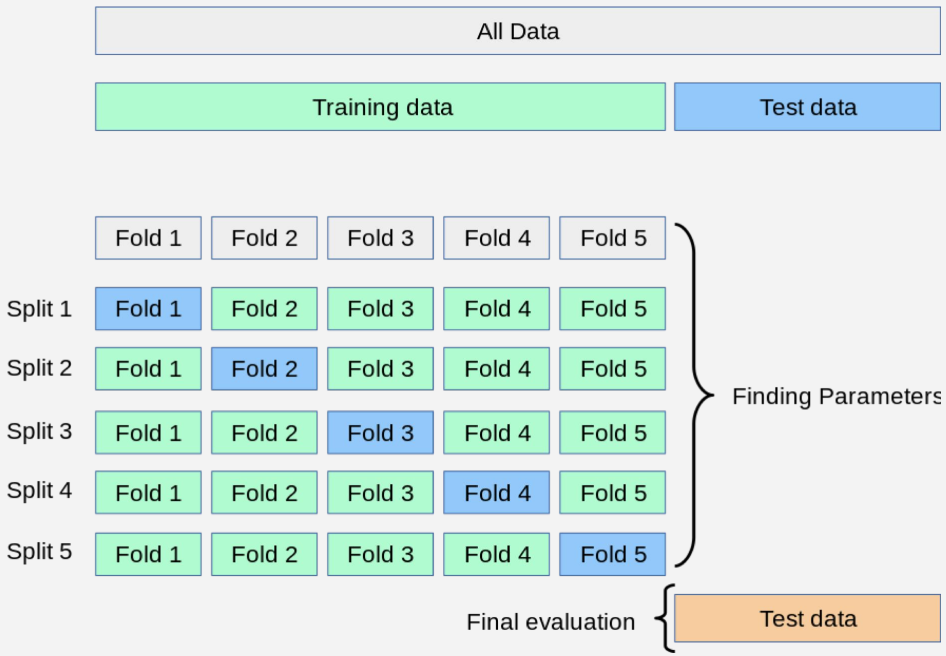
Overfitting can arise for many reasons (small training sets, too many parameters, strong covariance between features, etc.).

There are several model-specific methods to tackle over fitting, but a general technique is to incorporate a validation process in the training of ML models.


## K-fold cross-validation

1. split your training and testing set (e.g., 80/20) and set testing aside.
2. break up training set into K chunks (10 is canonical)
3. loop through the K chunks training on the remaining K−1 chunks and testing on the K-th chunk
4. modify the **hyperparameters** and repeat 3
5. once the best model is found, retrain on the full training set and apply to testing set for final model accuracy.

# Model Accuracy: training/testing with validation




*https://scikit-learn.org/stable/modules/cross_validation.html*

## K-fold cross-validation

1. split your training and testing set (e.g., 80/20) and set testing aside.
2. break up training set into K chunks (10 is canonical)
3. loop through the K chunks training on the remaining K−1 chunks and testing on the K-th chunk
4. modify the **hyperparameters** and repeat 3
5. once the best model is found, retrain on the full training set and apply to testing set for final model accuracy.

# Confusion Matrices

A representation for a classification task that indicates the model's "confusion" between outcomes. The smaller the off-diagonal elements, the more effective the model at correctly labeling classes.

---

Total # of spheres:
*121 + 8 + 14 + 72 = 215*

PRECISION:
fraction of objects labeled as a certain class that actually **are** that class,

$p_{glass} = 121 / (121 + 14) = \mathbf{0.896}$
$p_{plexi} = 72 / (8 + 72) = \mathbf{0.900}$

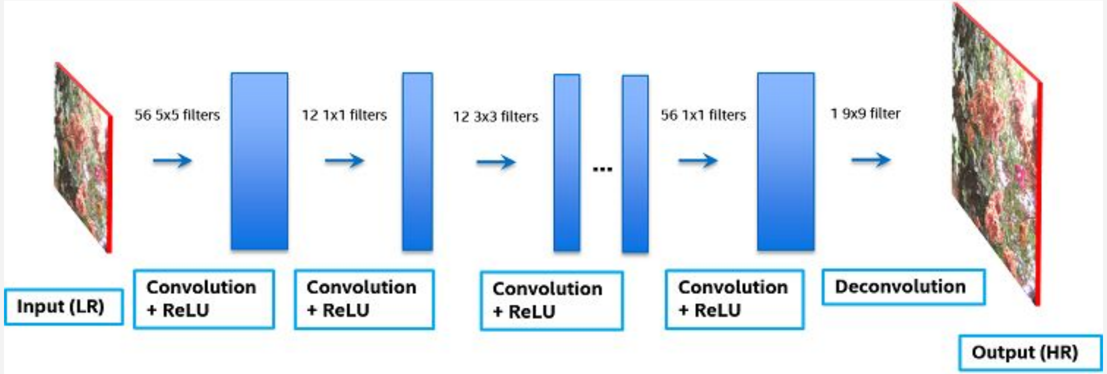
RECALL:
fraction of objects of a certain class that are actually labeled **as that class**,

$r_{glass} = 121 / (121 + 8) = \mathbf{0.938}$
$r_{plexi} = 72 / (14 + 72) = \mathbf{0.837}$

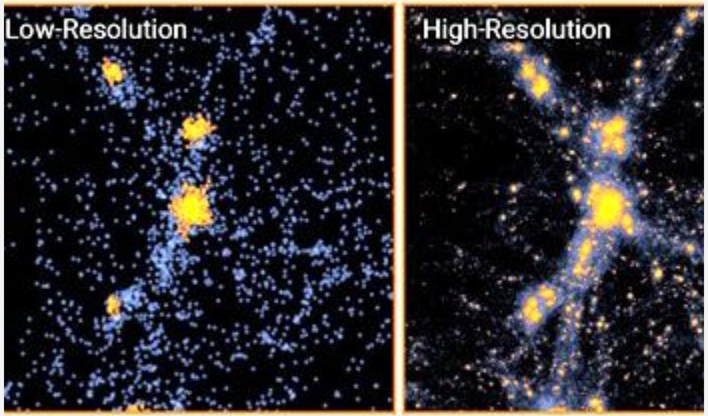
| | | Predicted | |
|---|---|---|---|
| | | glass | plexiglass |
| Actual | glass | 121 | 8 |
| | plexiglass | 14 | 72 |

# an emerging third Machine Learning paradigm: ethics

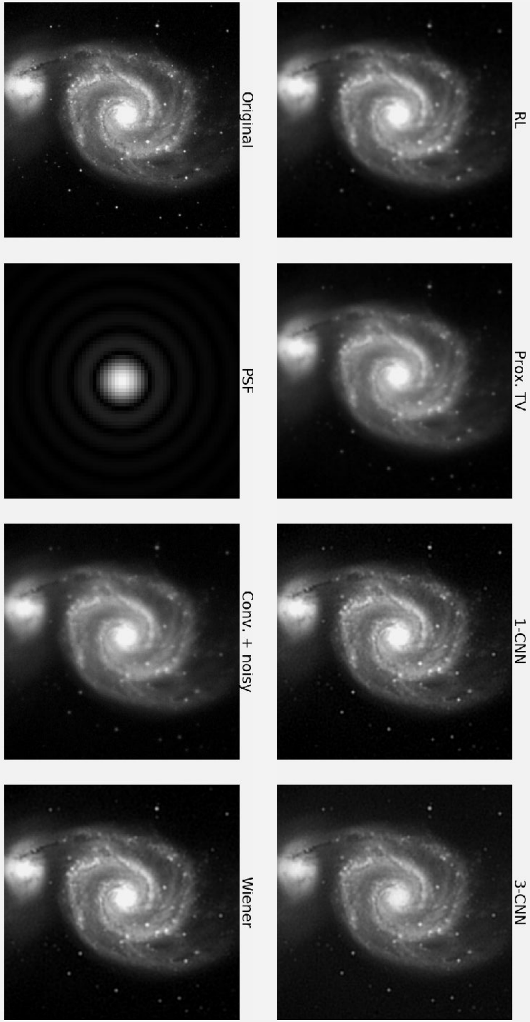
# Superresolution




https://medium.com/datadriveninvestor/using-the-super-resolution-convolutional-neural-network-for-image-restoration-ff1e8420d846

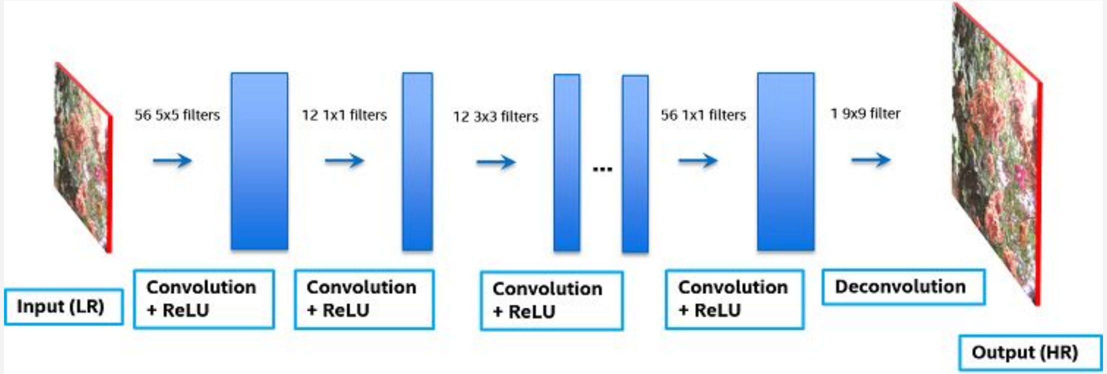



https://www.nsf.gov/discoveries/disc_summ.jsp?cntn_id=302666&org=NSF&from=news




https://ieeexplore.ieee.org/abstract/document/8081654

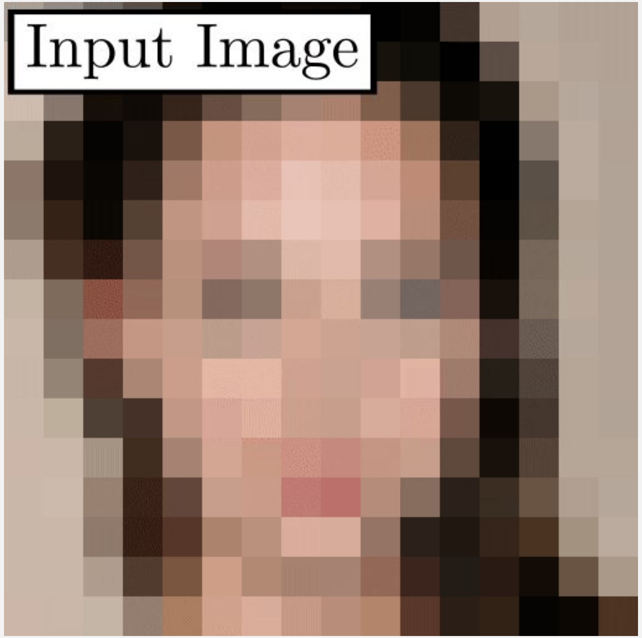
# Superresolution



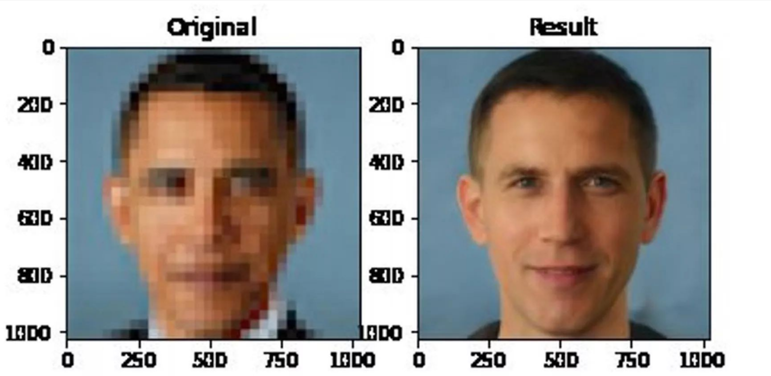

https://medium.com/datadriveninvestor/using-the-super-resolution-convolutional-neural-network-for-image-restoration-ff1e8420d846

PULSE

https://github.com/adamian98/pulse



# Superresolution


**When** is a model failing?
**Why** is a model failing?
**What** are the consequences of failure?
also: metrics? architecture? "good" model?

# Machine Learning in Practice

Tools these days facilitate the rapid creation of machine learning models,
and you ***can*** do machine learning

- without **calculus** (or linear algebra or algebra) and/or **domain knowledge**
- without **training/testing/validation** and **model selection**
- without considerations of the **ethical implications** for the models you build

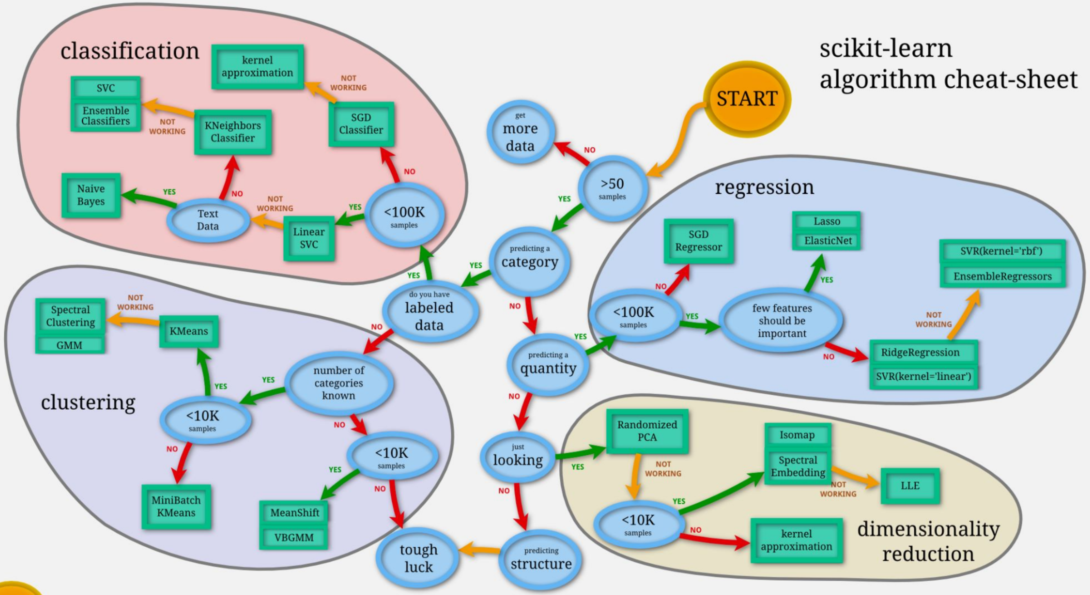
**but without all three, you are doing it poorly…**

```
events = pd.read_csv("icetop_event_detections.csv")[::100].dropna()
dtr = DecisionTreeRegressor(min_samples_leaf=1)
dtr.fit(events[events.columns[2:326]], events["primary_energy"])
acc = r2_score(events["primary_energy"], dtr.predict(events[events.columns[2:326]]))
print("accuracy : {0:.3f}".format(acc))
```

# Machine Learning in Practice

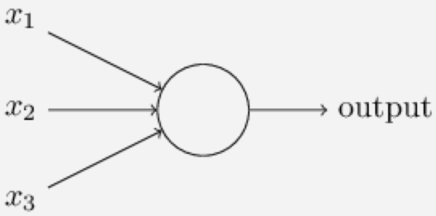
Machine Learning ≠ Data Science ≠ Neural Networks ≠ Artificial Intelligence

"deep" or otherwise
"convolutional" or otherwise




Back

# Neural Networks and Deep Learning



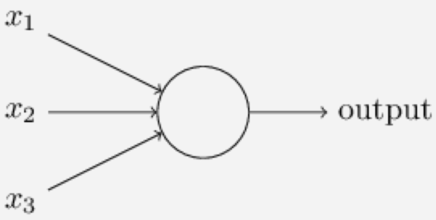

(almost) all images taken from:

**Neural Networks and Deep Learning**

Michael Nielsen

http://neuralnetworksanddeeplearning.com/

# Neural Networks in Public Life




**Autonomous vehicles** – scene awareness and decision making

**Healthcare** – medical imaging, augmentation of diagnosis

**Social media** (and tech of all sorts) – advertisement, automatic tagging, follow recommendations, bot identification

**Finance** – market prediction

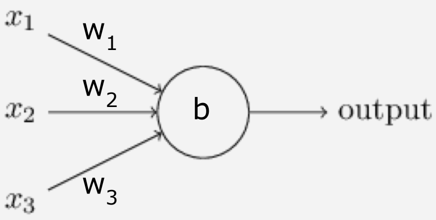
**Translation** – mapping between one language and another

**Security and cybersecurity** – anomaly detection, situational awareness, intrusion, automatic document digitization

**Agriculture** – crop yield prediction

**Speech to text** (and speech recognition) – mapping between audio and free text

# Neurons




A neuron takes a collection of data as input and combines it to generate an output.

The process of combining the data generally starts with a linear weighting,

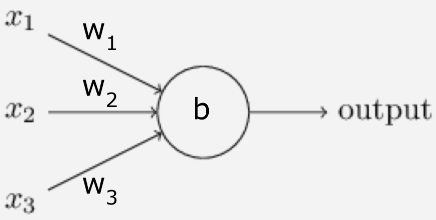
$$\boldsymbol{z = w \cdot x + b}$$

where · is the dot product:

$$\mathbf{w \cdot x = w_1\, x_1 + w_2\, x_2 + w_3\, x_3 + \ldots = \Sigma\, w_i\, x_i}$$

**w** is referred to as the **weights** of the neuron
**b** is referred to as the **bias** of the neuron

# Neurons and Activation Functions




A neuron takes a collection of data as input and combines it to generate an output.

The process of combining the data generally starts with a linear weighting,

$$\boldsymbol{z = w \cdot x + b}$$

where · is the dot product:

$$\mathbf{w \cdot x = w_1\, x_1 + w_2\, x_2 + w_3\, x_3 + \ldots = \Sigma\, w_i\, x_i}$$

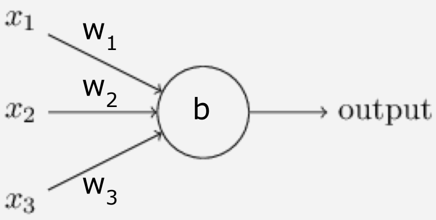
**w** is referred to as the **weights** of the neuron
**b** is referred to as the **bias** of the neuron

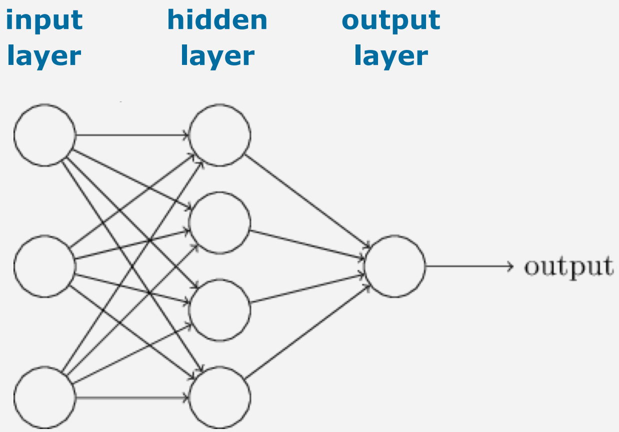
Once **z** is generated, the final step to combine the inputs is the **activation function**,

$$\text{output} = \boldsymbol{a(z)}$$

and **a** can (and will) take many forms.

# Multi-Layer Perceptron

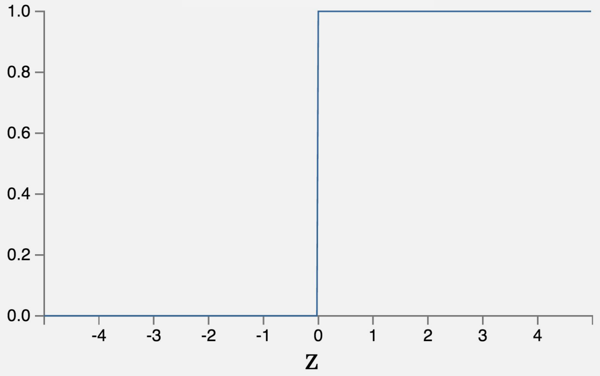


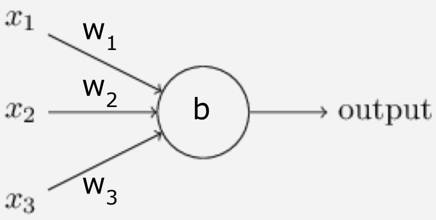

Some of the first neural networks were **multilayer perceptrons** (MLPs).

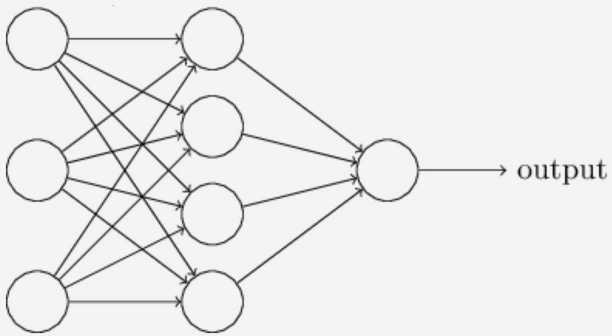
**Perceptron**

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



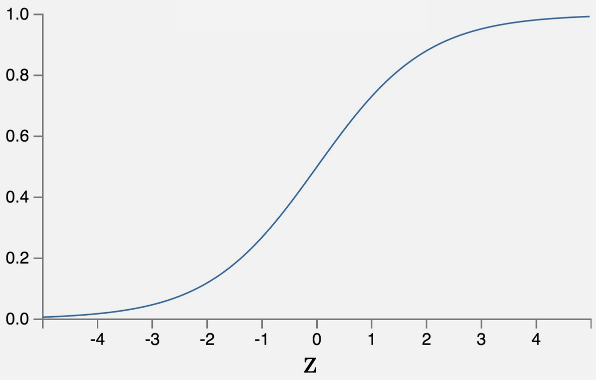
# Activation Functions



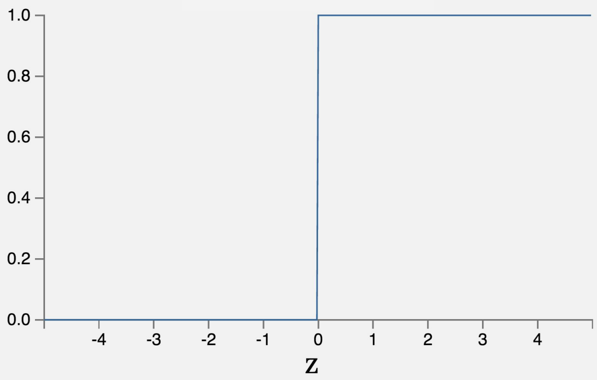

**Sigmoid Activation**

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$

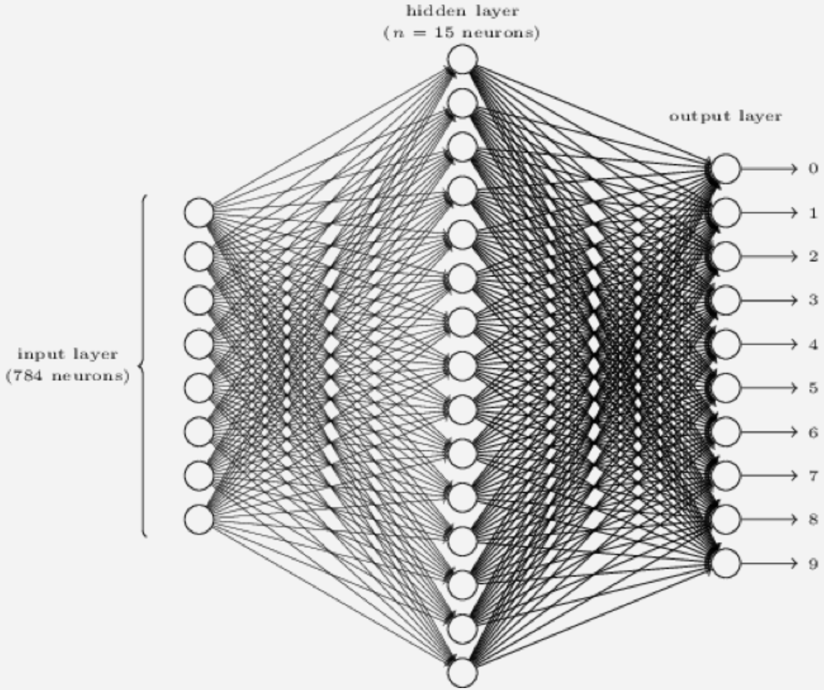


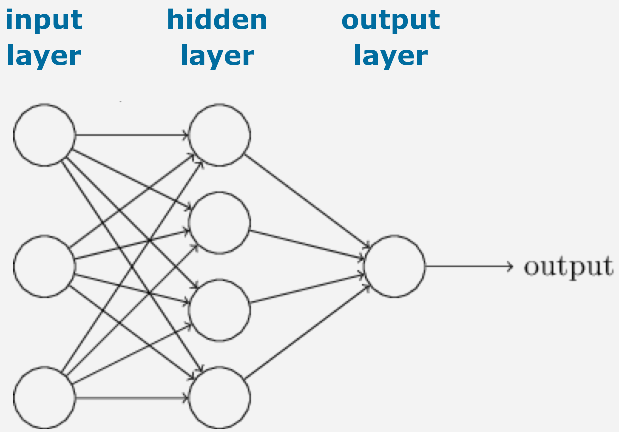

**Perceptron**

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



# Fully-Connected Network



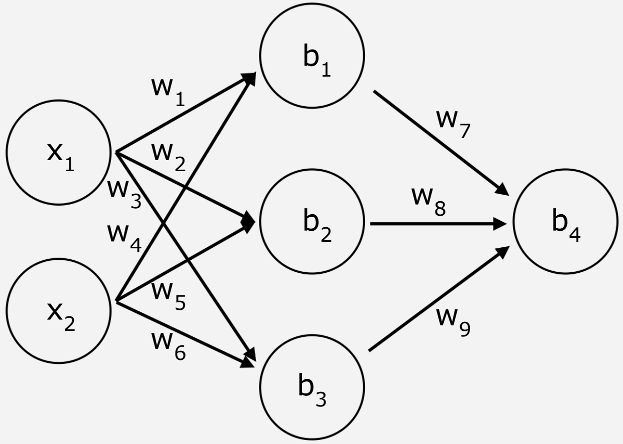




**Fully connected networks** contain links between every neuron in every layer.

The **output layer** can be a single output or multiple output.

note: in this simple example there are already 295 parameters!

# Complexity of Interactions in Neural Networks




$$\text{output} = \frac{1}{1+e^{-w_7 O_1 - w_8 O_2 - w_9 O_3 - b_4}}$$

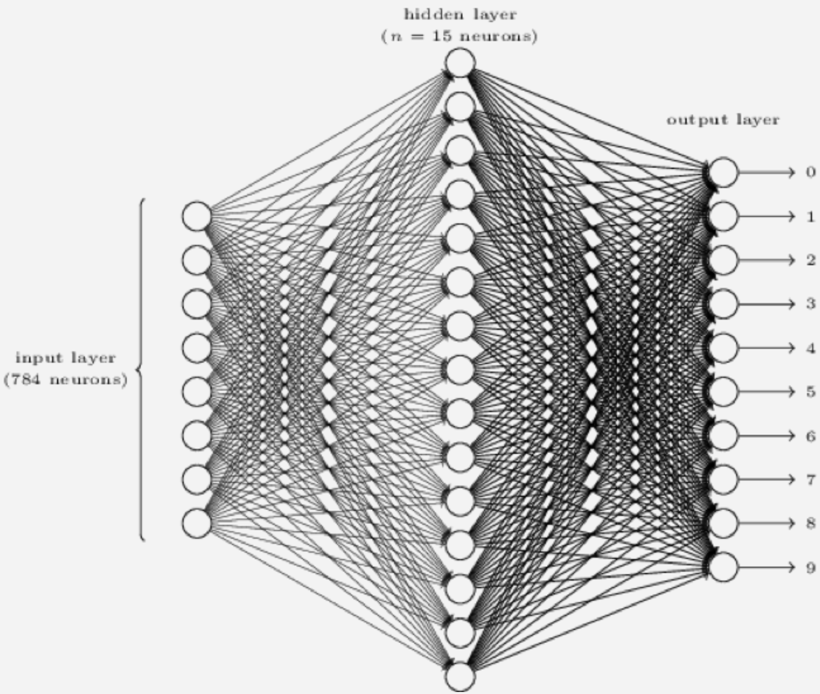
$$O_1 = \frac{1}{1+e^{-w_1 x_1 - w_4 x_2 - b_1}}$$

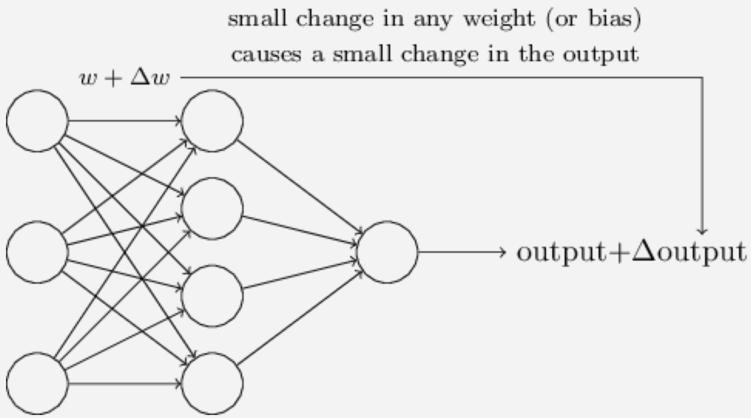
$$O_2 = \frac{1}{1+e^{-w_2 x_1 - w_5 x_2 - b_2}}$$

$$O_3 = \frac{1}{1+e^{-w_3 x_1 - w_6 x_2 - b_3}}$$

$$\text{output} = \frac{1}{1+e^{-\frac{w_7}{1+e^{-w_1 x_1 - w_4 x_2 - b_1}} - \frac{w_8}{1+e^{-w_2 x_1 - w_5 x_2 - b_2}} - \frac{w_9}{1+e^{-w_3 x_1 - w_6 x_2 - b_3}} - b_4}}$$

# Training a Neural Network




Training models with this many parameters requires a lot of care:

. defining the metric
. optimization schemes
. training/validation/testing sets

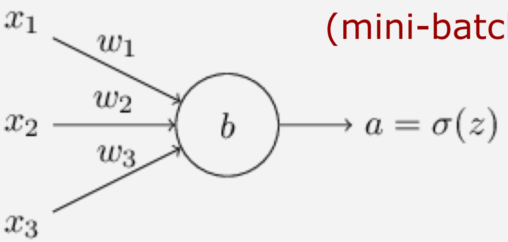
But just like our simple linear regression case, the fact that small changes in the parameters leads to small changes in the output (for the right activation functions) gives us hope!

note: in this simple example there are already 295 parameters!

# Training a Neural Network

$$C(w,b) \equiv \frac{1}{2n} \sum_{x} \|y(x) - a\|^2$$

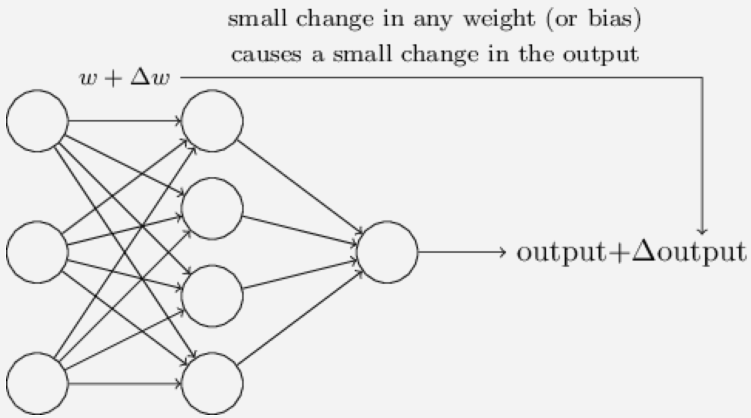
training examples
(mini-batches)




learning rate

$$w_k \rightarrow w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b_l' = b_l - \eta \frac{\partial C}{\partial b_l}.$$

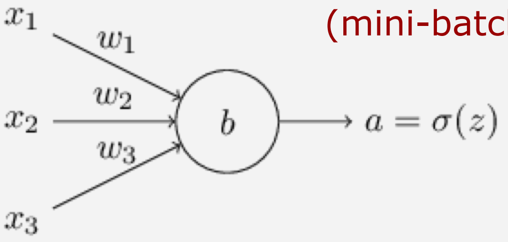



$$\Delta \text{output} \approx \sum_j \frac{\partial \, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial \, \text{output}}{\partial b} \Delta b$$

# Training a Neural Network

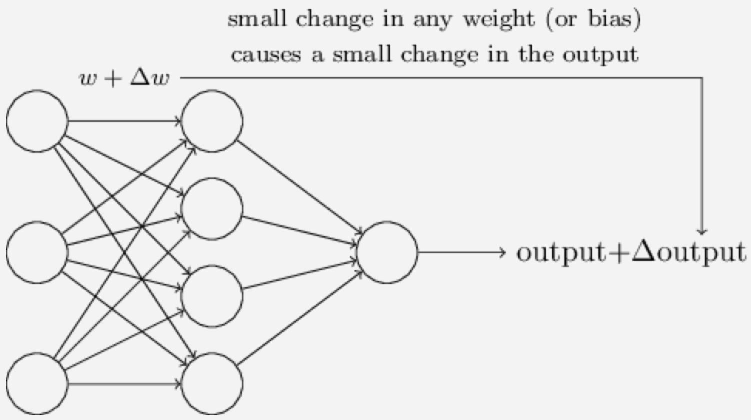
$$C(w, b) \equiv \frac{1}{2n} \sum_{x} \|y(x) - a\|^2$$

training examples
(mini-batches)




$$w_k \to w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \to b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l},$$

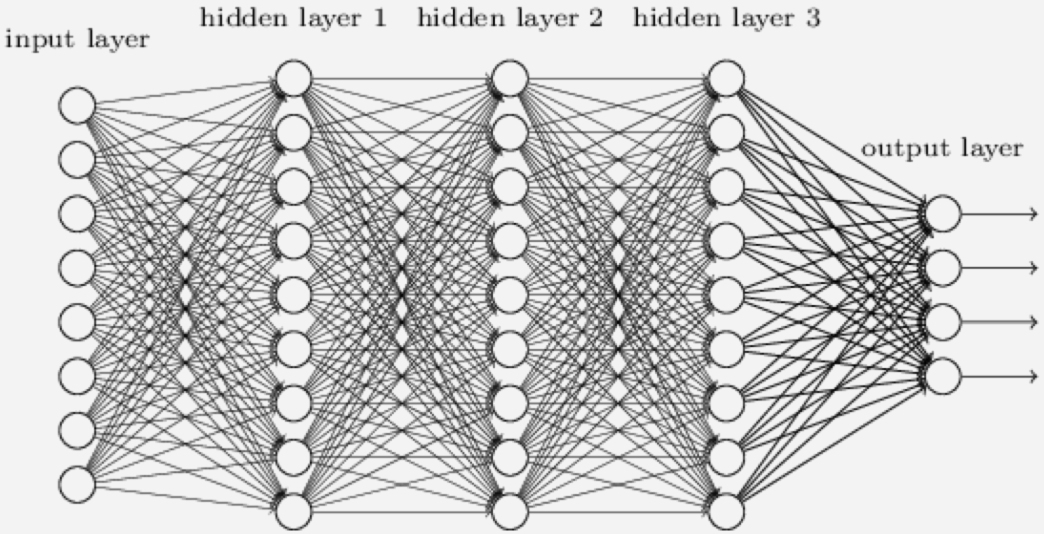



$$\Delta \text{output} \approx \sum_j \frac{\partial \, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial \, \text{output}}{\partial b} \Delta b$$

**Stochastic Gradient Descent**

potentially very difficult to compute

# Training a Neural Network




**feed data forward through network and calculate cost metric** →

**for each layer, calculate effect of small changes on next layer** ←

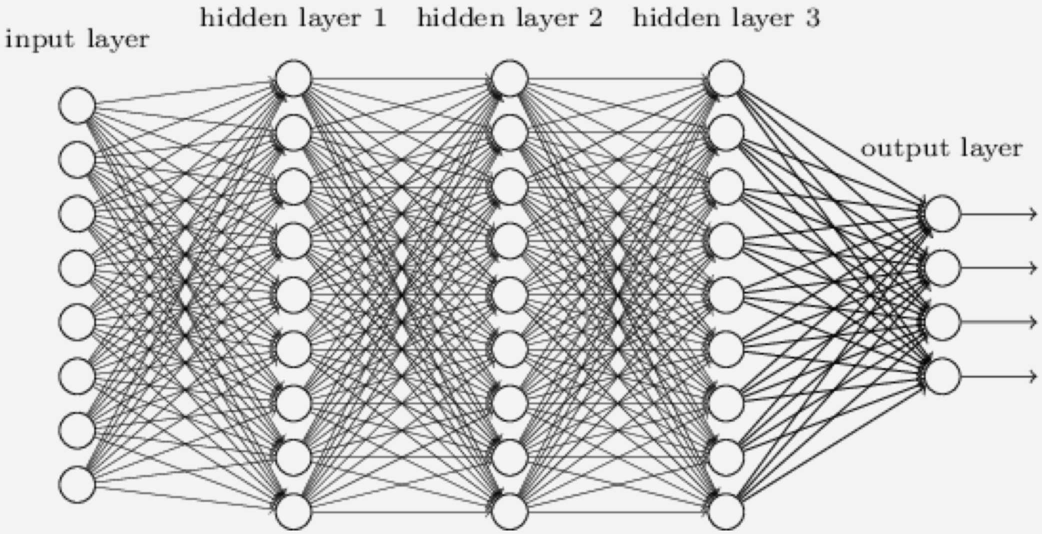
Stochastic gradient descent works well for learning parameters, but…

how to compute which way is "downhill"?

with something like linear regression, it is easy to see the effects on the model as you change w and b. With multivariate regression it's a bit more tricky since w is $w_i$.

With neural networks we need to be able to calculate $\Delta a_k$ given $\Delta w_{ij}$.

# Training a Neural Network




1. Randomly choose all w and b
2. Feed a random subset of data forward through the network
3. Calculate the output error (cost)
4. Determine which “direction” will decrease the cost most efficiently by determining the change in cost at each layer based on changes in parameters at the previous layer
5. Step in that direction
6. Repeat steps 1-5 until convergence

**feed data forward through network and calculate cost metric**

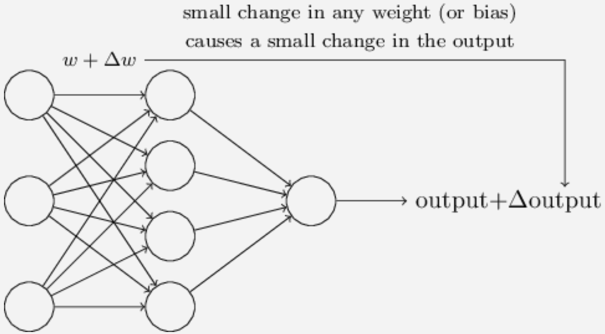
**for each layer, calculate effect of small changes on next layer**

**BACKPROPAGATION**

# Backpropagation

$$C(w,b) \equiv \frac{1}{2n}\sum_x \|y(x)-a\|^2$$ **Quadratic**

$$C = -\frac{1}{n}\sum_x \left[y \ln a + (1-y)\ln(1-a)\right]$$ **Cross-entropy**




$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

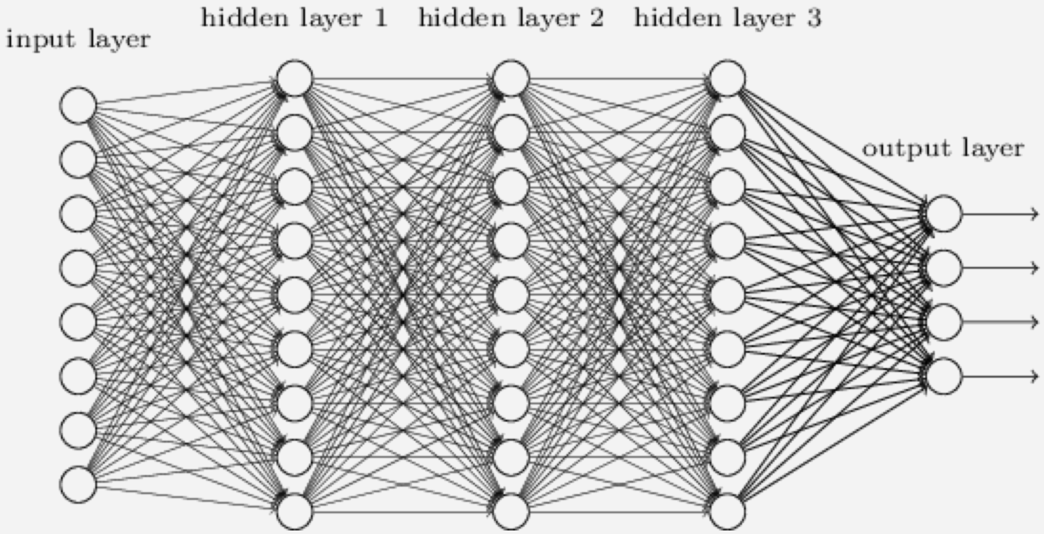
$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

1. **Input $x$:** Set the corresponding activation $a^1$ for the input layer.
2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.
3. **Output error $\delta^L$:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

# Converting Outputs into Probabilities

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$ **Quadratic**

$$C = -\frac{1}{n} \sum_x \left[y \ln a + (1-y) \ln(1-a)\right]$$ **Cross-entropy**




In the multi-output case, we would like to interpret this output layer as a list of probabilities of each outcome.

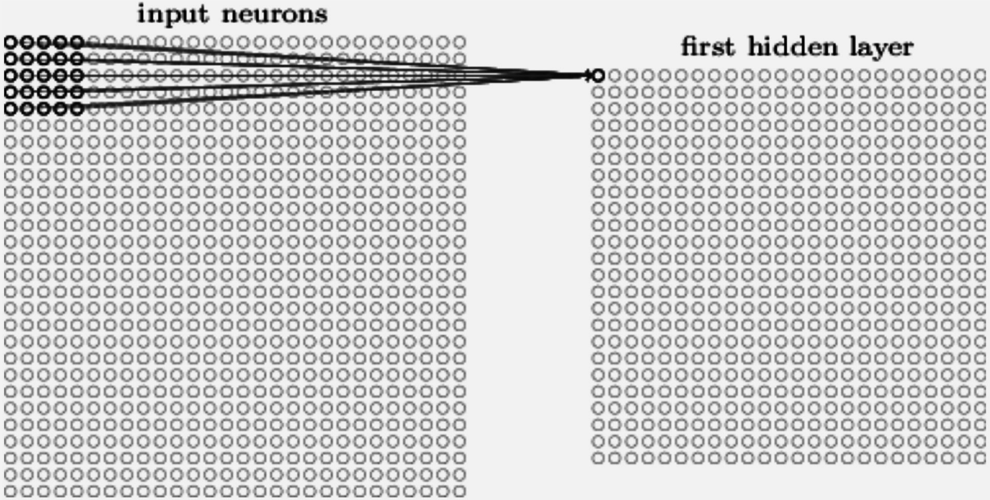
For that, a **softmax activation** is often applied to the output,

$$a_i \rightarrow \frac{e^{a_i}}{\sum_{j=0}^{N} e^{a_j}}$$

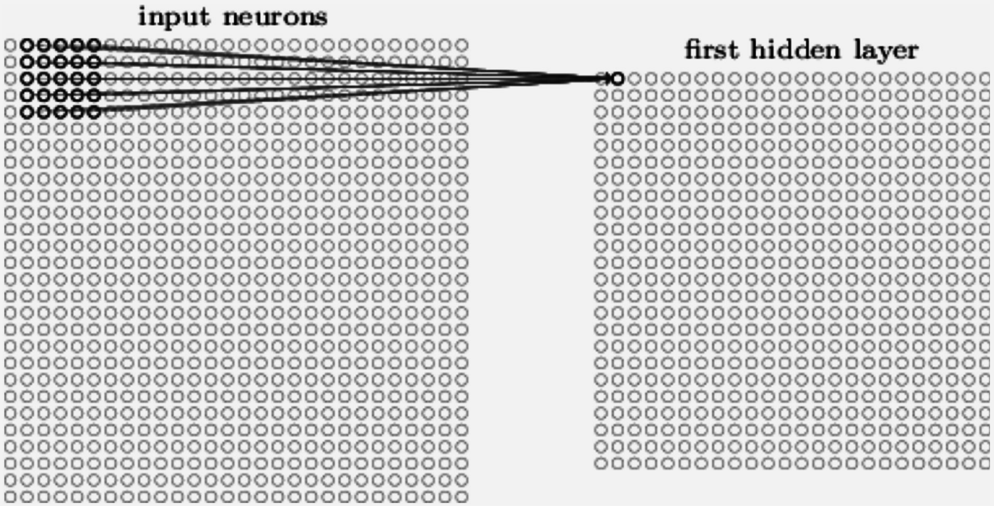
which has the properties that

1.) ***$0 < y_i \leq 1$ for all $i$***
2.) ***$\sum y_i = 1$***

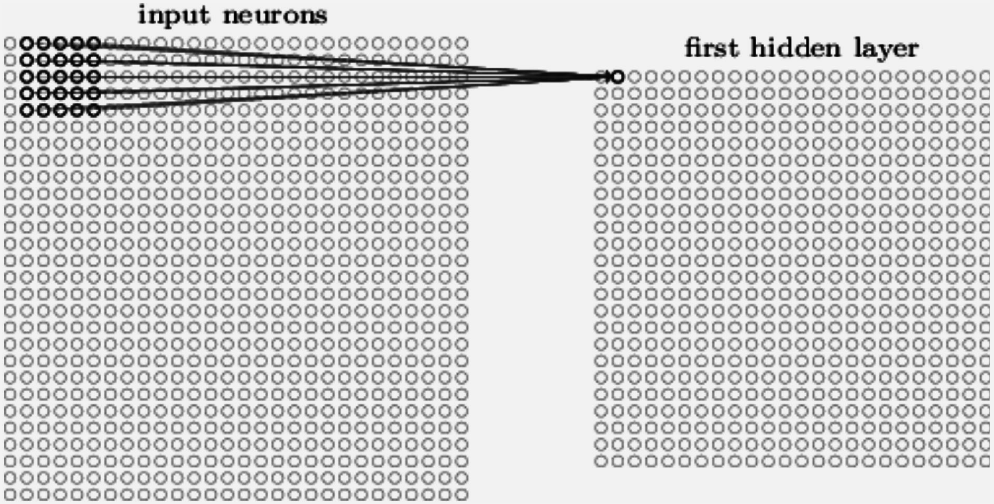
# Convolutional Neural Networks

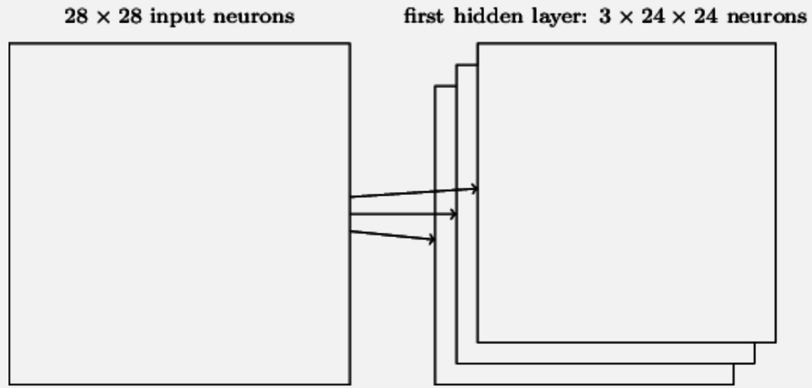


# Convolutional Neural Networks

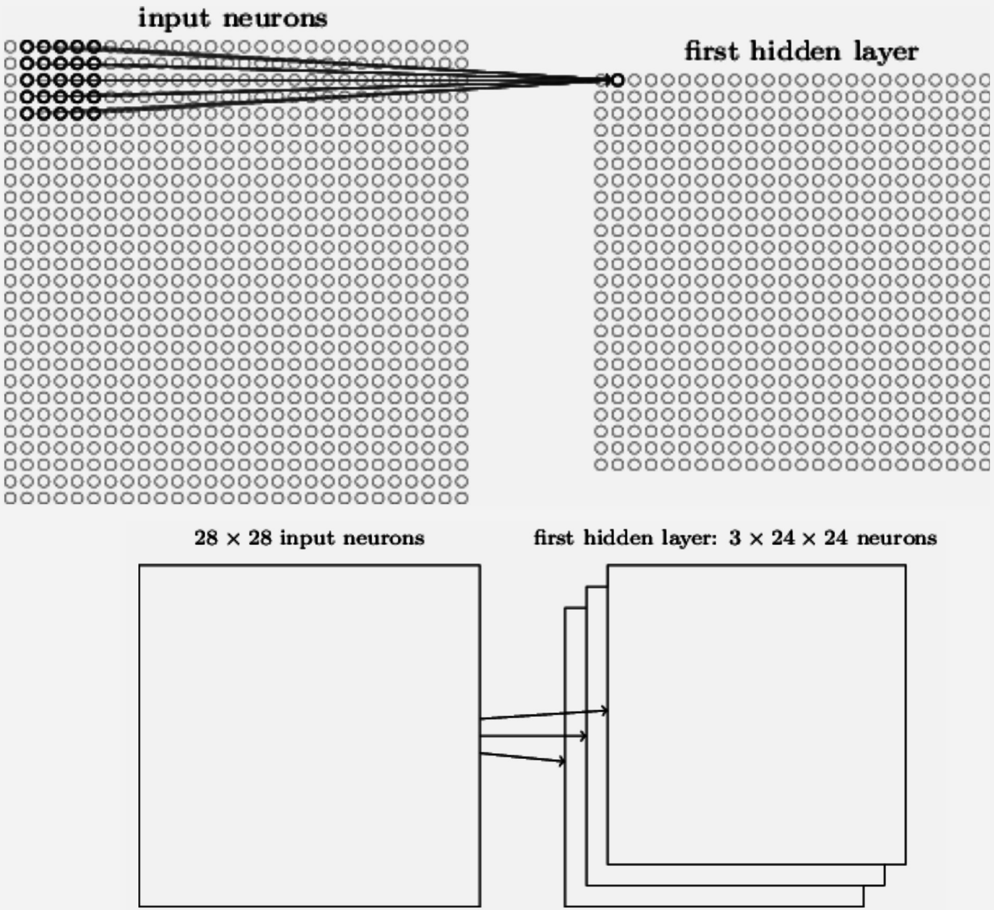


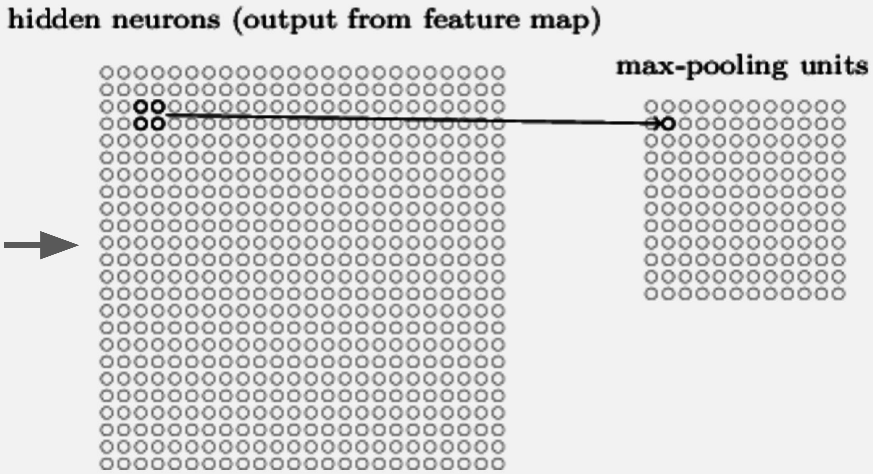
# Convolutional Neural Networks



# Convolutional Neural Networks




**max-pooling** "layers" are a special kind of filter that does not have **w** or **b** (and is not learned)

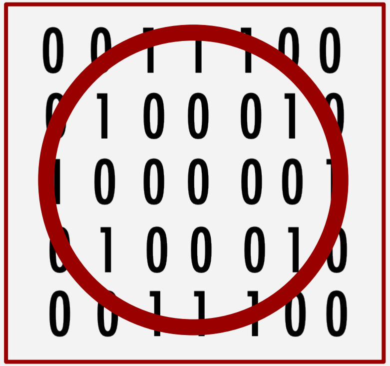
# Convolutional Neural Networks

Bianchi Pista
http://bikeattack.com


0 0 1 1 1 0 0
0 1 0 0 0 1 0
1 0 0 0 0 0 1
0 1 0 0 0 1 0
0 0 1 1 1 0 0

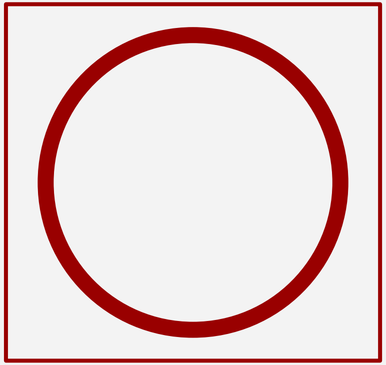
# Convolutional Neural Networks

Bianchi Pista
http://bikeattack.com

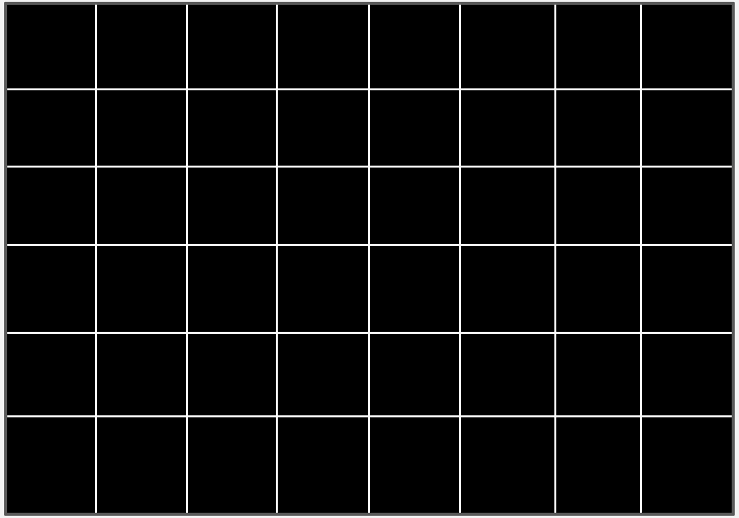
# Convolutional Neural Networks

Bianchi Pista
http://bikeattack.com

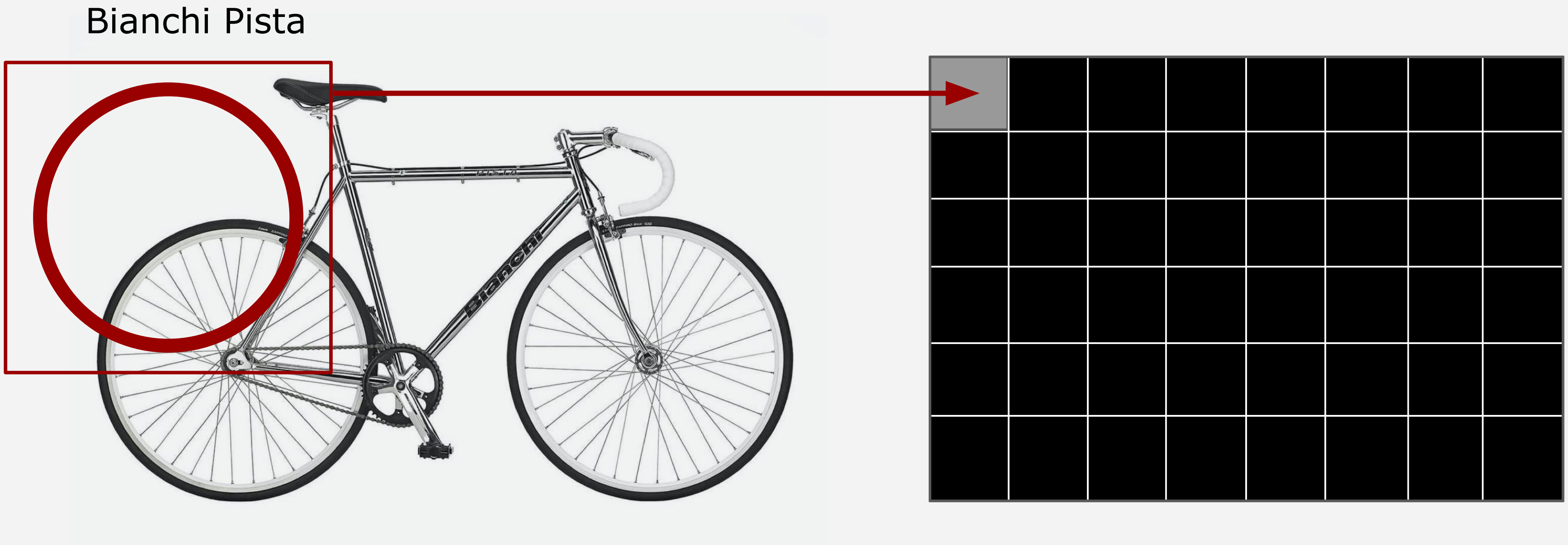



# Convolutional Neural Networks

Bianchi Pista

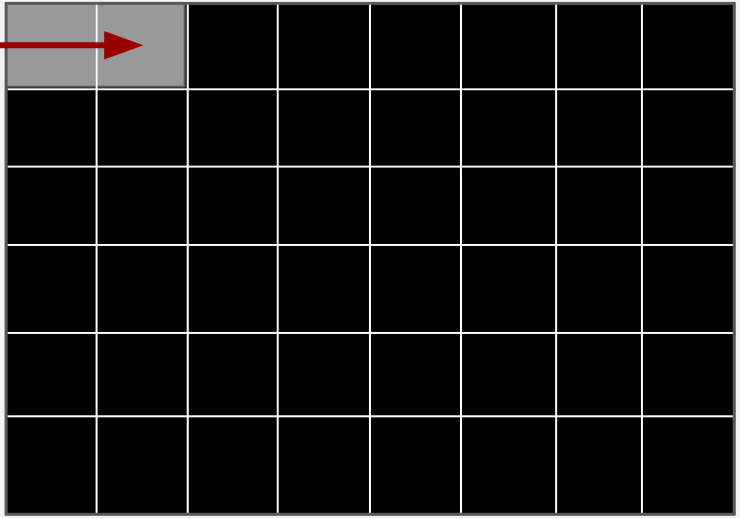



# Convolutional Neural Networks

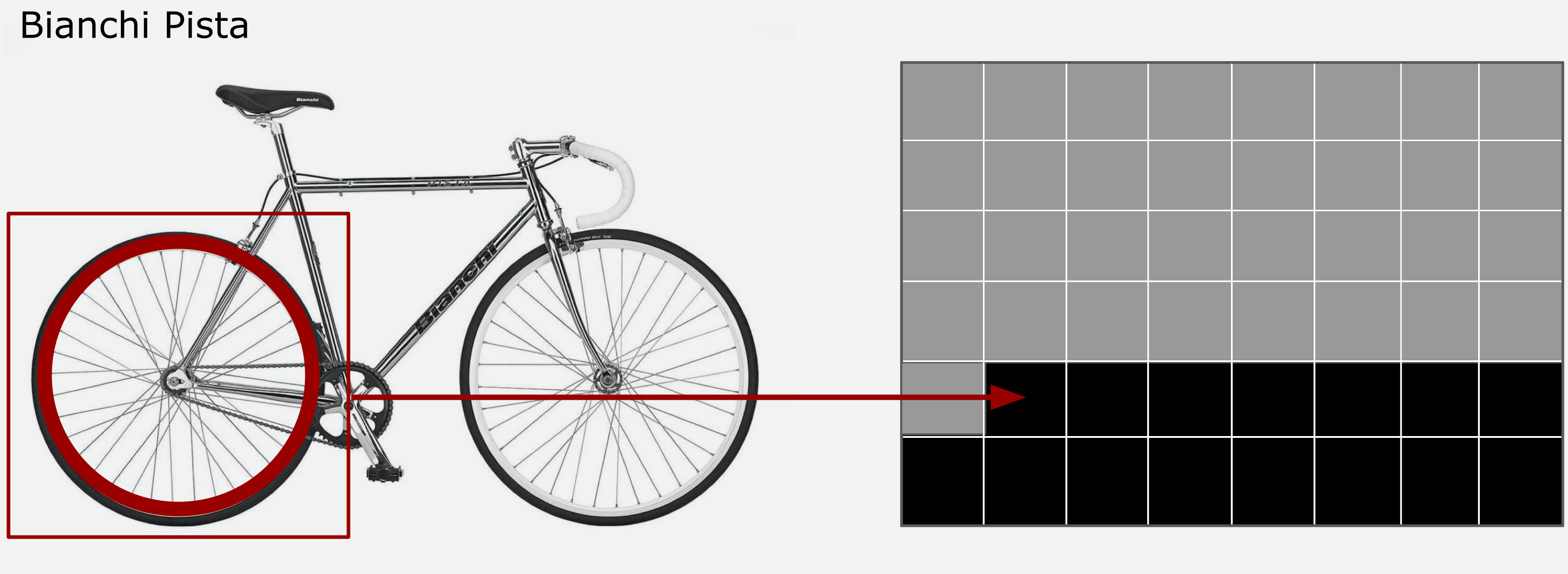


# Convolutional Neural Networks

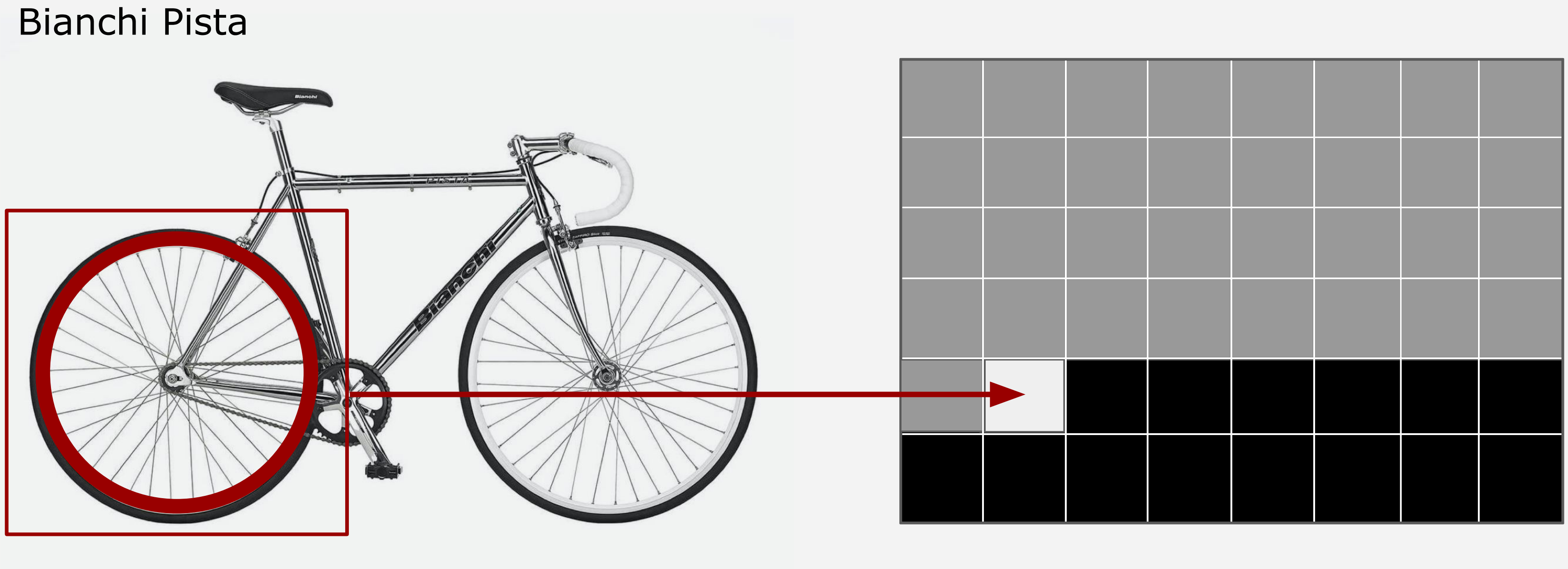
Bianchi Pista

# Convolutional Neural Networks

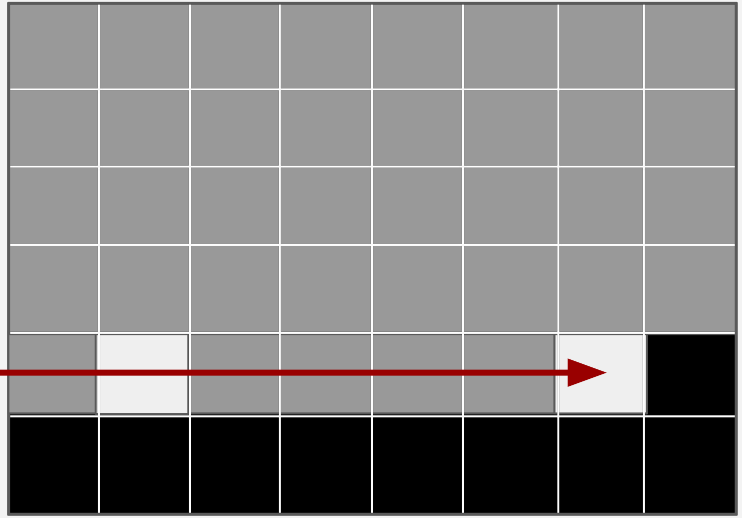


# Convolutional Neural Networks

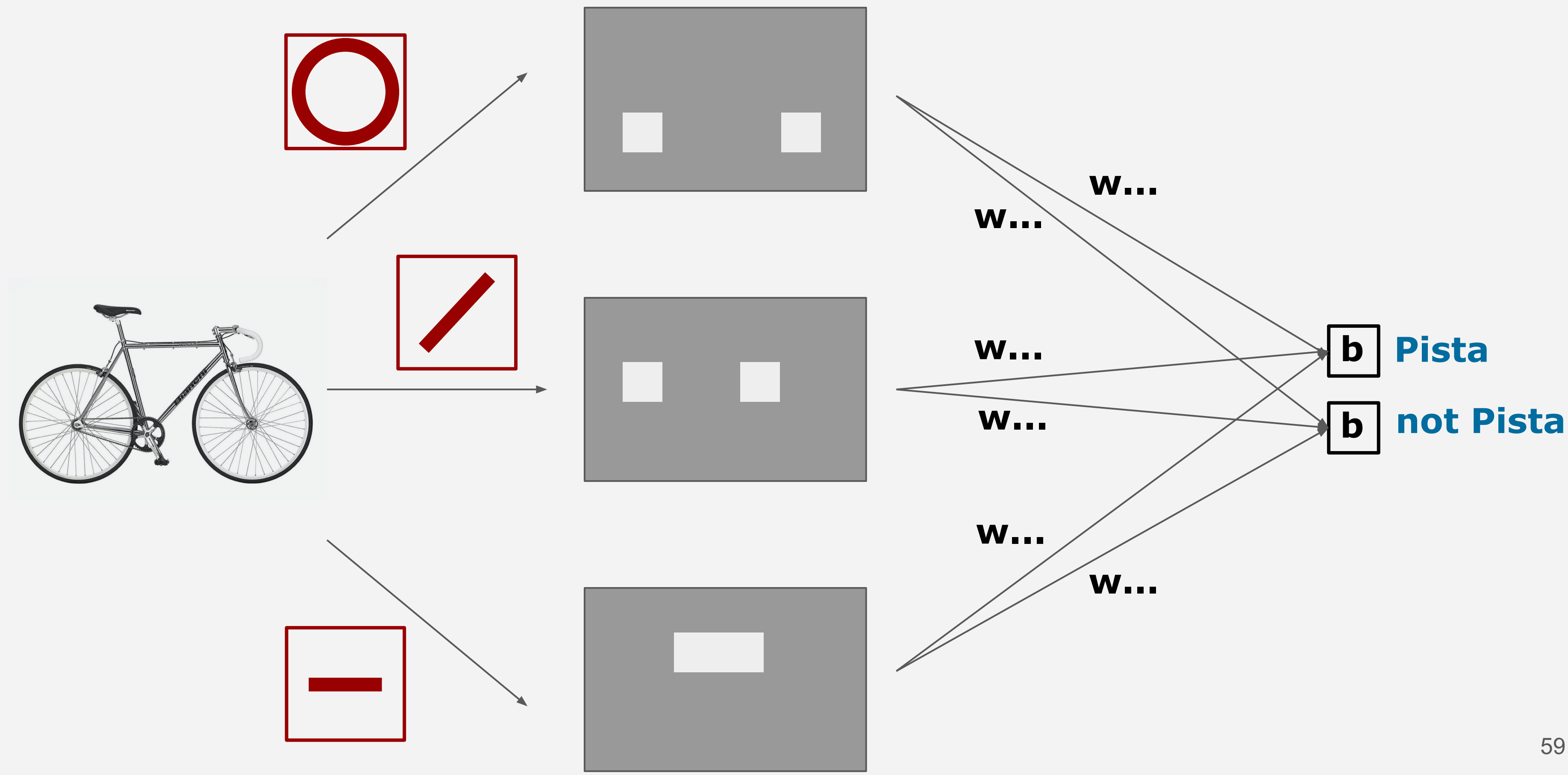


# Convolutional Neural Networks

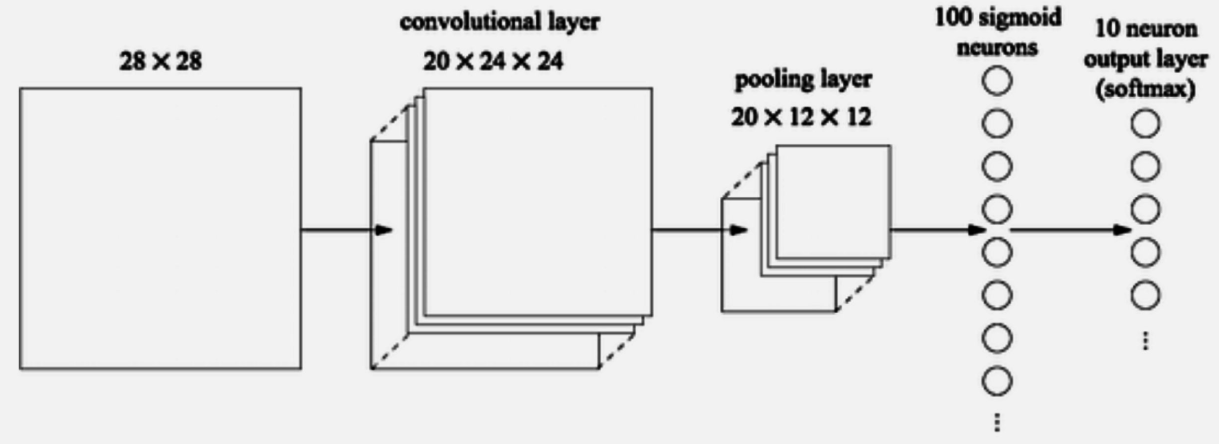
Bianchi Pista

# Convolutional Neural Networks

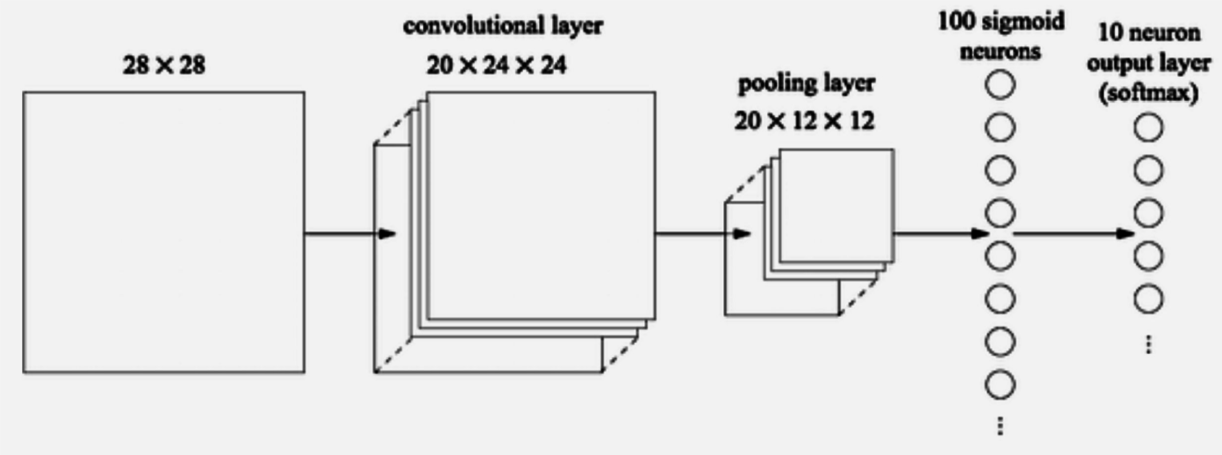


# Convolutional Neural Networks




**DEEP CONVOLUTIONAL NEURAL NETWORK (CNN)**

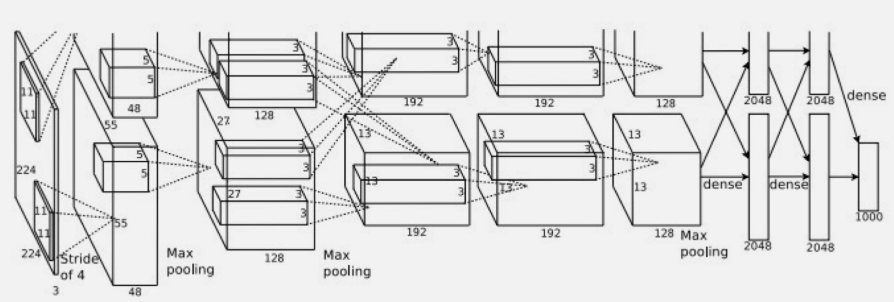
# Convolutional Neural Networks



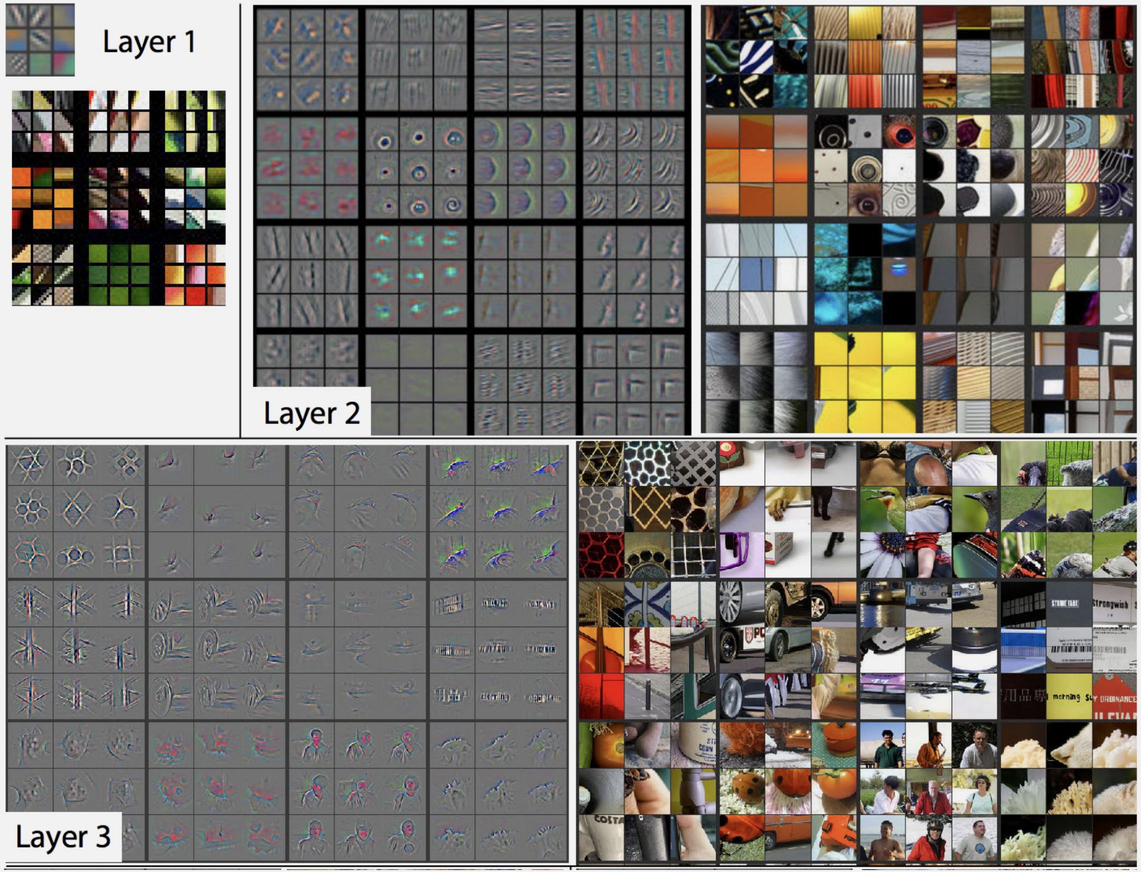

## DEEP CONVOLUTIONAL NEURAL NETWORK (CNN)

“**AlexNet**”

(Krizhevsky, Sutskever, & Hinton, 2012)



# Convolutional Neural Networks




Zeiler & Fergus (2013)