ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING THE STEEPEST DESCENT

ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING THE STEEPEST DESCENT

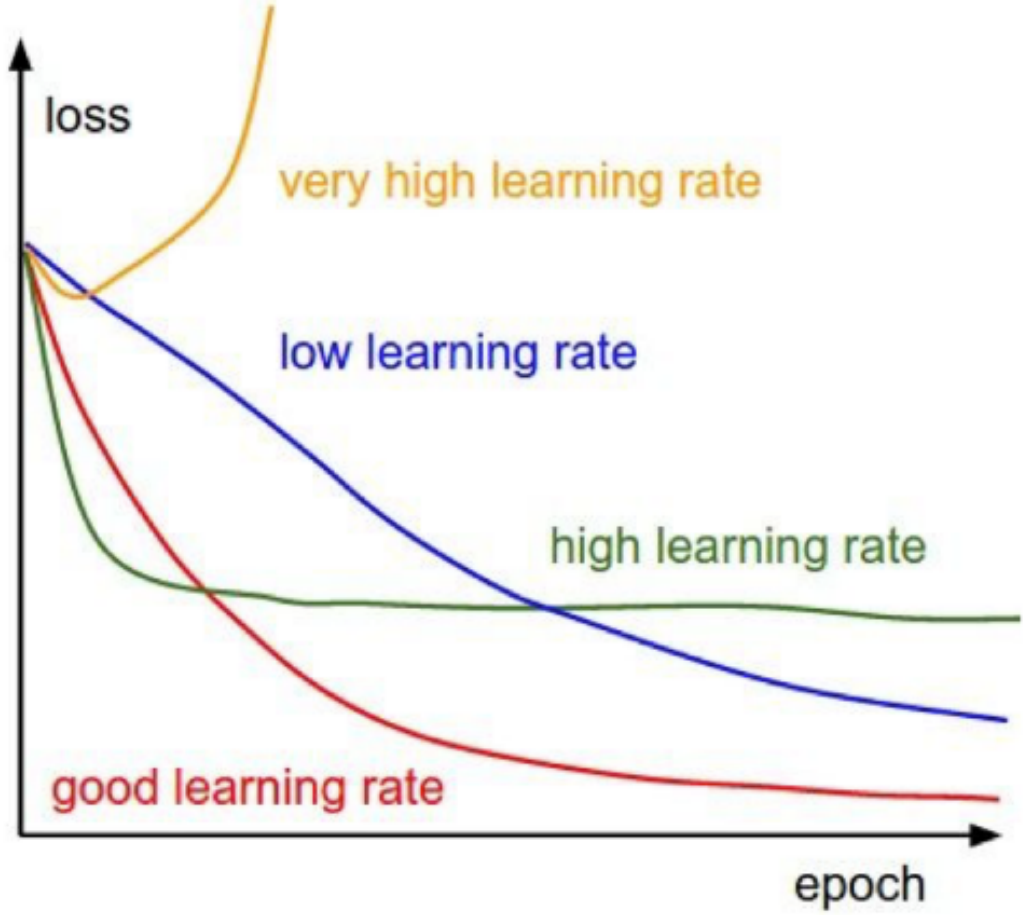
THE CHOICES OF THE INITIAL WEIGHTS AND THE LEARNING RATES ARE IMPORTANT

ONE KEY PROBLEM WITH GRADIENT DESCENT IS THAT IT EASILY CONVERGES TO LOCAL MINIMA BY FOLLOWING THE STEEPEST DESCENT

THE CHOICES OF THE INITIAL WEIGHTS AND THE LEARNING RATES ARE IMPORTANT

WE WILL TALK ABOUT THIS LATER

LEARNING RATES



Credit:

LEARNING RATES

$$W_{t+1} = W_t - \lambda \nabla f(W_t)$$

THERE ARE DIFFERENT WAYS
TO UPDATE THE LEARNING RATE

Credit:

LEARNING RATES

$$W_{t+1} = W_t - \lambda \nabla f(W_t)$$

THERE ARE DIFFERENT WAYS
TO UPDATE THE LEARNING RATE

ADAGRAD:

THE LEARNING RATE IS SCALED DEPENDING ON THE HISTORY OF PREVIOUS GRADIENTS

$$W_{t+1} = W_t - \frac{\lambda}{\sqrt{G_t + \epsilon}} \nabla f(W_t)$$

G IS A MATRIX CONTAINING ALL PREVIOUS GRADIENTS. WHEN THE GRADIENT BECOMES LARGE THE LEARNING RATE IS DECREASED AND VICE VERSA.

$$G_{t+1} = G_t + (\nabla f)^2$$

Credit:

LEARNING RATES

$$W_{t+1} = W_t - \lambda \nabla f(W_t)$$

THERE ARE DIFFERENT WAYS
TO UPDATE THE LEARNING RATE

RMSPROP:

THE LEARNING RATE IS SCALED DEPENDING ON THE HISTORY OF PREVIOUS GRADIENTS

$$W_{t+1} = W_t - \frac{\lambda}{\sqrt{G_t + \epsilon}} \nabla f(W_t)$$

SAME AS **ADAGRAD** BUT G IS CALCULATED BY EXPONENTIALLY DECAYING AVERAGE

$$G_{t+1} = \lambda G_t + (1 - \lambda)(\nabla f)^2$$

Credit:

ADAM [Adaptive moment estimator]:

SAME IDEA, USING FIRST AND SECOND ORDER MOMENTUMS

$$G_{t+1} = \beta_2 G_t + (1-\beta_2)(\nabla f)^2 \quad M_{t+1} = \beta_1 M_t + (1-\beta_1)(\nabla f)$$

$$W_{t+1} = W_t - \frac{\lambda}{\sqrt{\hat{G}_t + \epsilon}} \hat{M}_t$$

with: $\hat{M}_{t+1} = \frac{M_t}{1-\beta_1}$ $\hat{G}_{t+1} = \frac{G_t}{1-\beta_2}$

ADAM [Adaptive moment estimator]:

SAME IDEA, USING FIRST AND SECOND ORDER MOMENTUMS

$$G_{t+1} = \beta_2 G_t + (1-\beta_2)(\nabla f)^2 \quad M_{t+1} = \beta_1 M_t + (1-\beta_1)(\nabla f)$$

ONLY FOR YOUR RECORDS

$$\sqrt{G_t + \epsilon}$$

with: $\hat{M}_{t+1} = \dfrac{M_t}{1-\beta_1}$ $\hat{G}_{t+1} = \dfrac{G_t}{1-\beta_2}$

IN KERAS:

RMSprop [source]

```
keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
```

RMSProp optimizer.

It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned).

This optimizer is usually a good choice for recurrent neural networks.

Arguments

- **lr**: float >= 0. Learning rate.
- **rho**: float >= 0.
- **epsilon**: float >= 0. Fuzz factor. If `None`, defaults to `K.epsilon()`.
- **decay**: float >= 0. Learning rate decay over each update.

References

- rmsprop: Divide the gradient by a running average of its recent magnitude

IN KERAS:

Adam [source]

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

Adam optimizer.

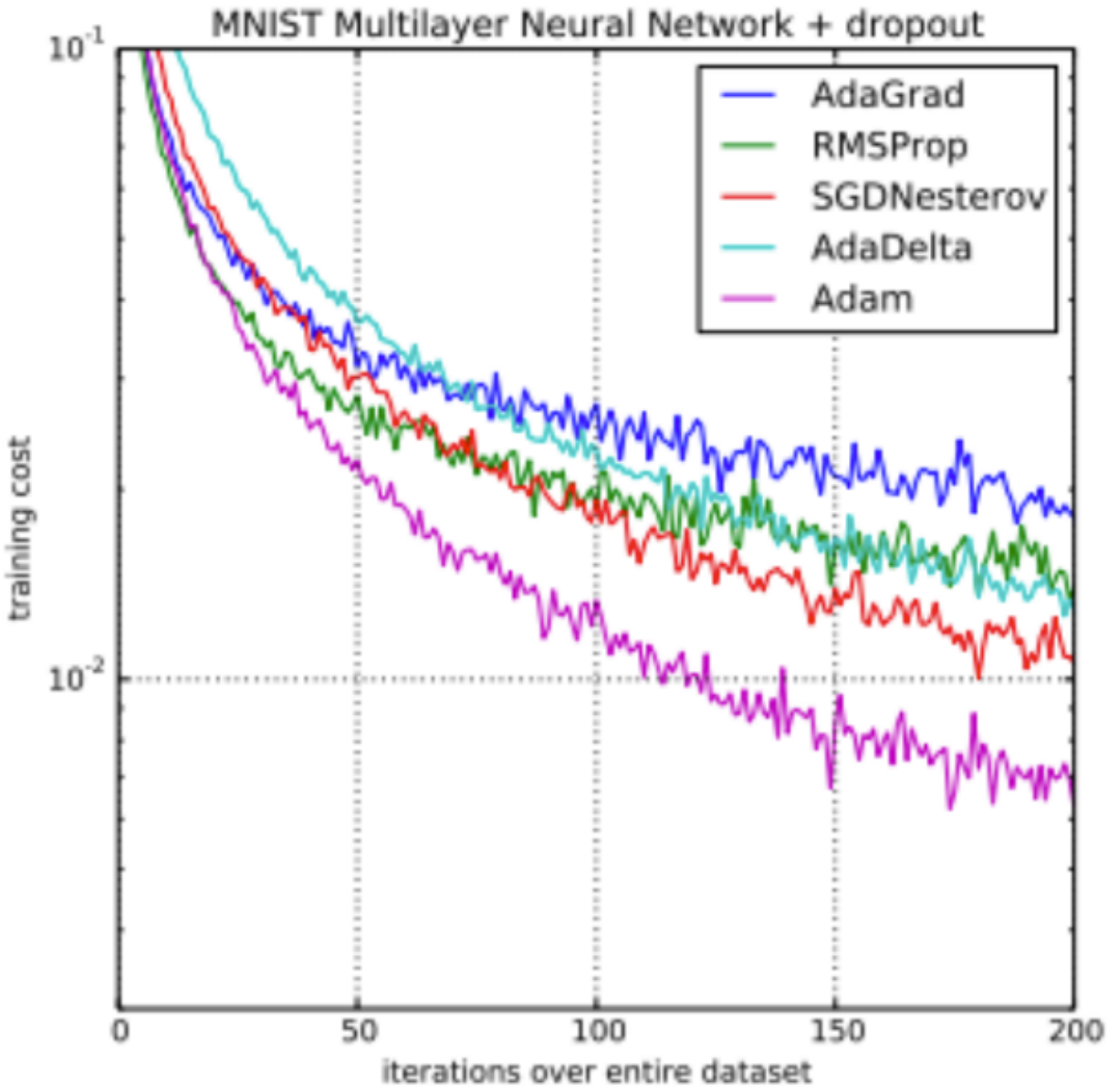
Default parameters follow those provided in the original paper.

Arguments

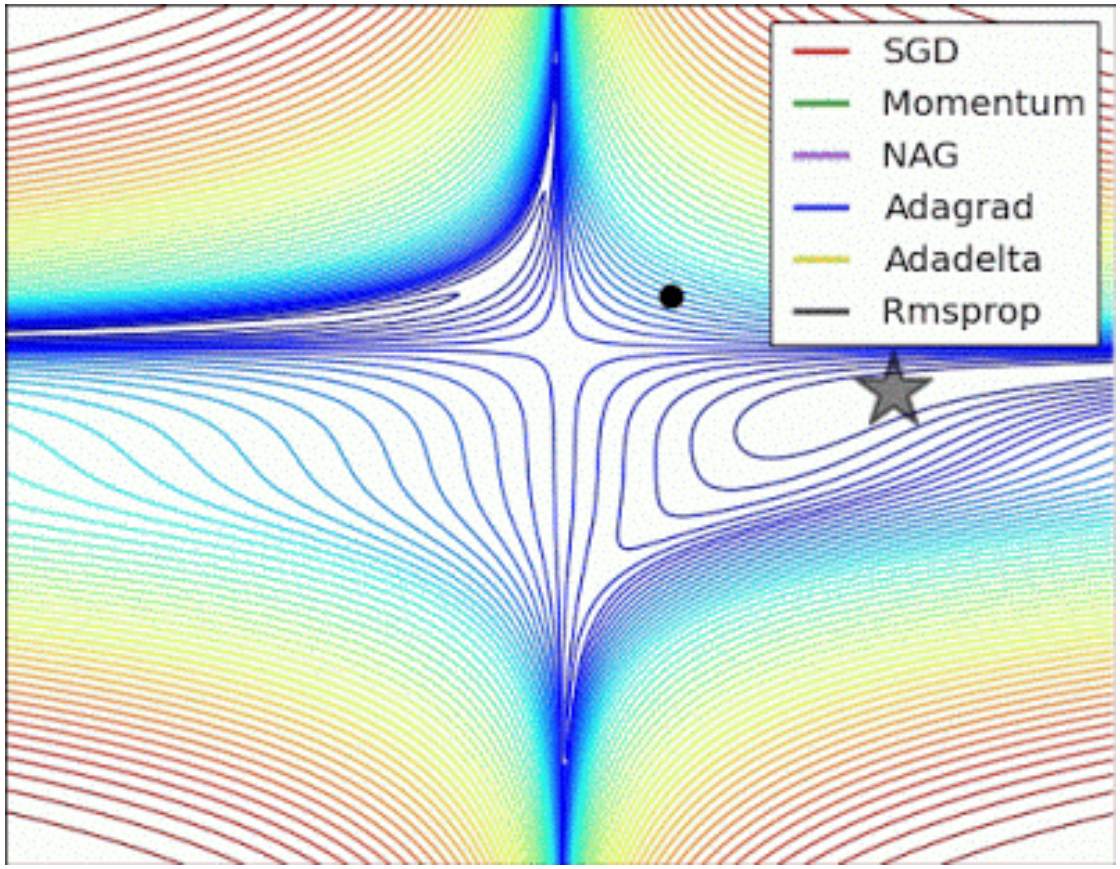
- **lr**: float >= 0. Learning rate.
- **beta_1**: float, 0 < beta < 1. Generally close to 1.
- **beta_2**: float, 0 < beta < 1. Generally close to 1.
- **epsilon**: float >= 0. Fuzz factor. If `None`, defaults to `K.epsilon()`.
- **decay**: float >= 0. Learning rate decay over each update.
- **amsgrad**: boolean. Whether to apply the AMSGrad variant of this algorithm from the paper "On the Convergence of Adam and Beyond".

References

- Adam - A Method for Stochastic Optimization
- On the Convergence of Adam and Beyond


MNIST Multilayer Neural Network + dropout
AdaGrad
RMSProp
SGDNesterov
AdaDelta
Adam
training cost
iterations over entire dataset
10^{-1}
10^{-2}
0
50
100
150
200

Credit



Credit

BATCH GRADIENT DESCENT

LOCAL MINIMA CAN ALSO BE AVOIDED BY COMPUTING THE GRADIENT IN SMALL BATCHES INSTEAD OF OVER THE FULL DATASET

BATCH GRADIENT DESCENT

LOCAL MINIMA CAN ALSO BE AVOIDED BY COMPUTING THE GRADIENT IN SMALL BATCHES INSTEAD OF OVER THE FULL DATASET

MINI-BATCH GRADIENT DESCENT

$$V_{t+1/num} = W_t - \lambda_t \nabla f(W_t; x^{(i,i+b)}, y^{(i,i+b)})$$

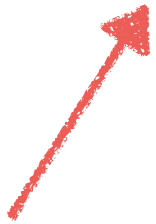
THE GRADIENT IS COMPUTED OVER A BATCH OF SIZE B

STOCHASTIC GRADIENT DESCENT

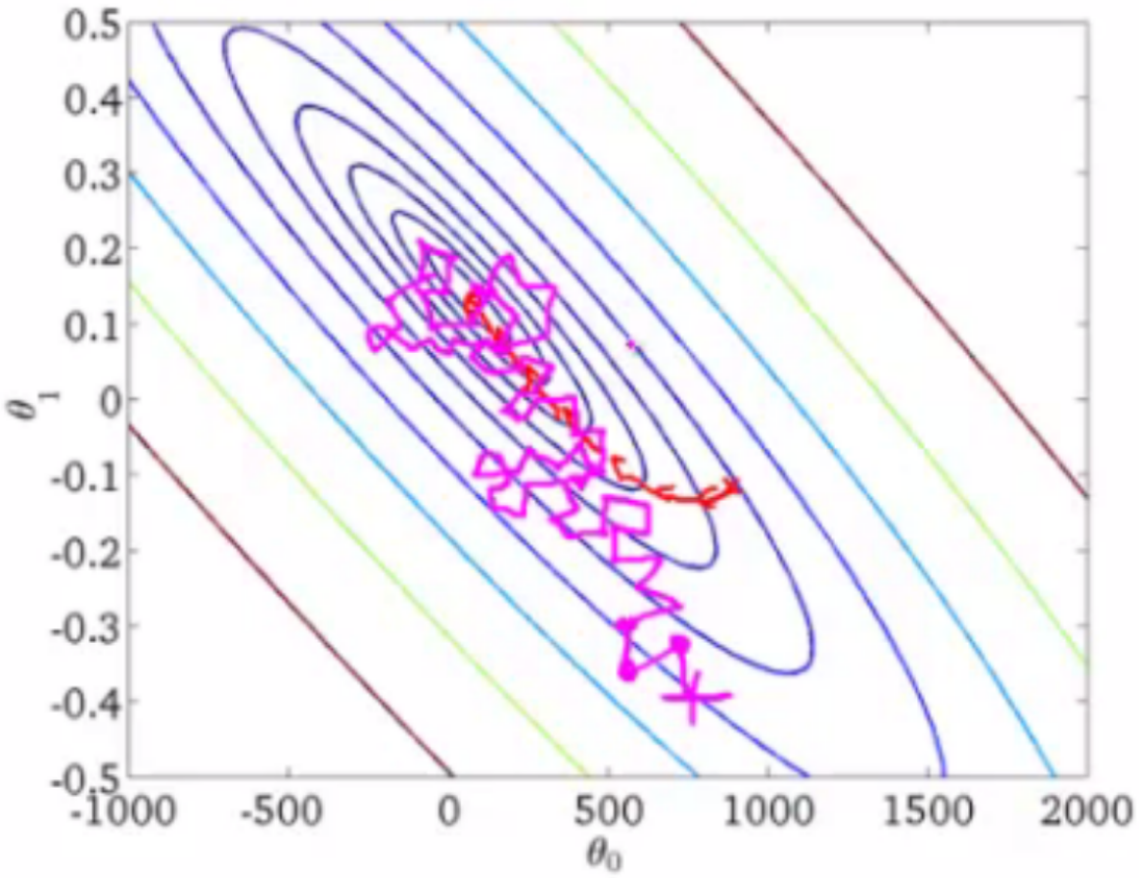
THE EXTREME CASE IS TO COMPUTE THE GRADIENT ON EVERY TRAINING EXAMPLE.

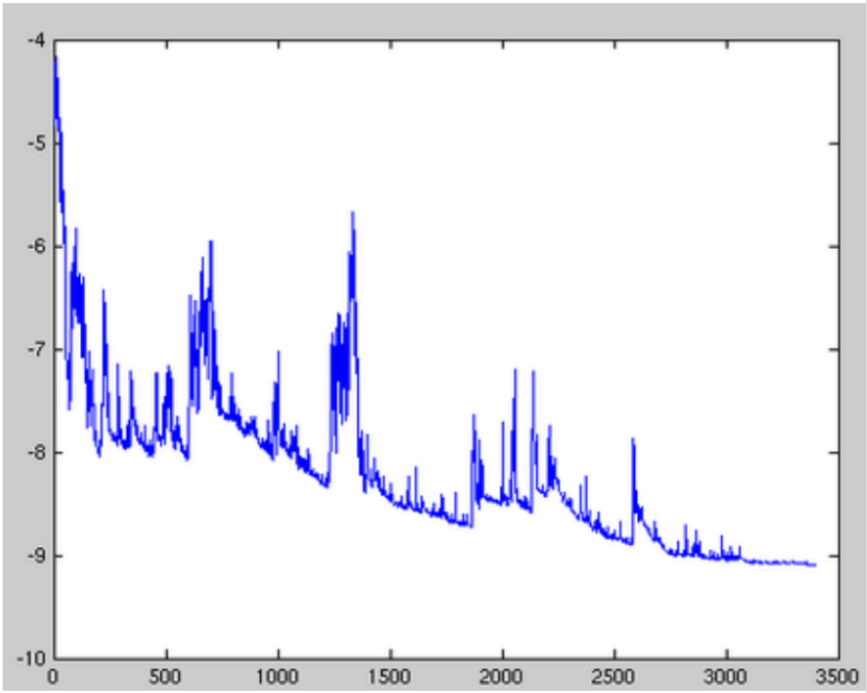
STOCHASTIC GRADIENT DESCENT

$$V_{t+1/num} = W_t - \lambda_t \nabla f(W_t; x^{(i,i+b)}, y^{(i,i+b)})$$



b=1





Fluctuations in the total objective function as gradient steps with respect to mini-batches are taken.

Credit

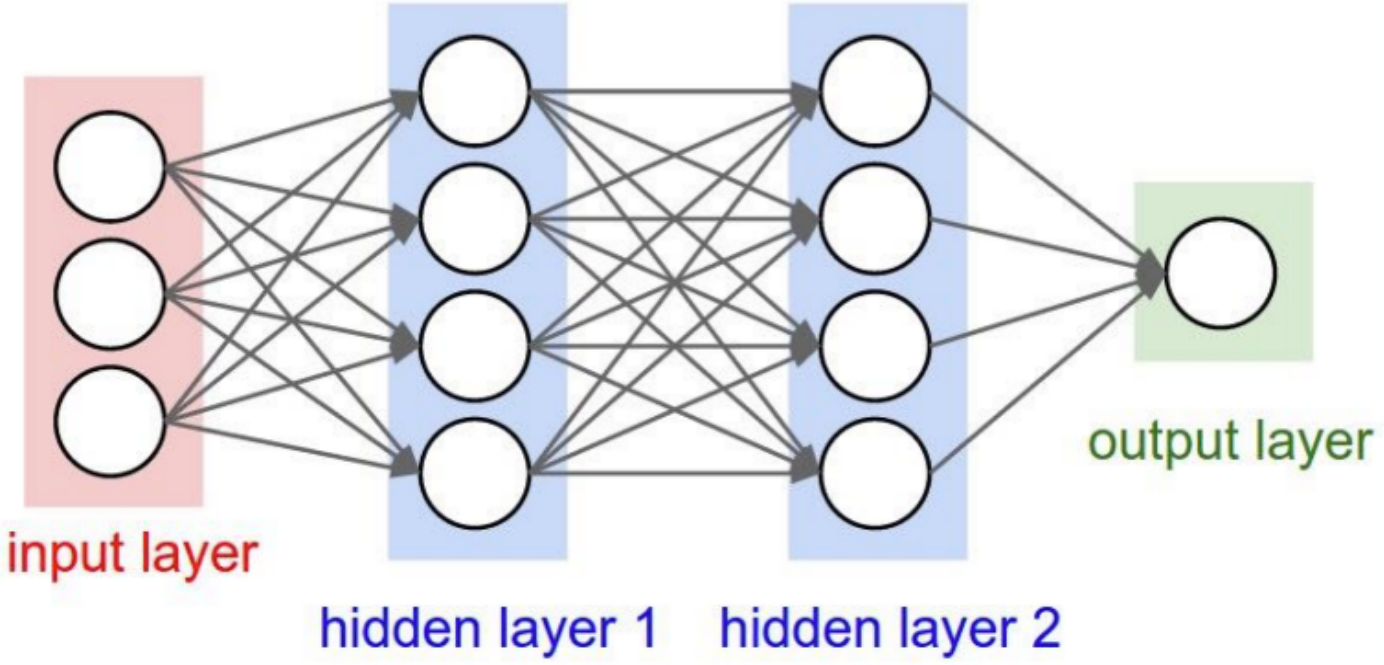
“CLASSICAL” MACHINE LEARNING

$$f_{\vec{W}}(\vec{x}) = \vec{y}$$

LABEL

Q , SF

REPLACE THIS BY A GENERAL NON LINEAR FUNCTION WITH SOME PARAMETERS W



$$p = g_3(W_3 g_2(W_2 g_1(W_1 \vec{x}_0)))$$

NETWORK FUNCTION

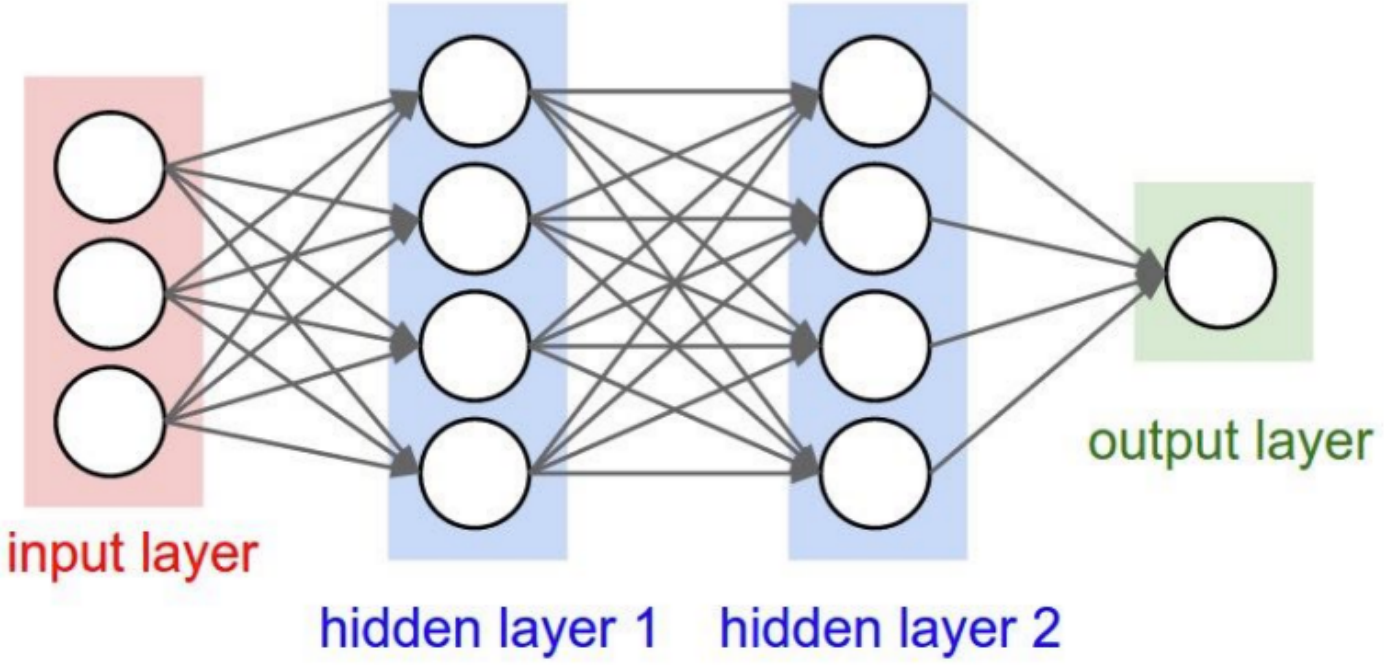
“CLASSICAL” MACHINE LEARNING

$$f_{\vec{W}}(\vec{x}) = \vec{y}$$

LABEL

Q , SF

REPLACE THIS BY A GENERAL NON LINEAR FUNCTION WITH SOME PARAMETERS W



$$p = g_3(W_3 g_2(W_2 g_1(W_1 \vec{x}_0)))$$

NETWORK FUNCTION

LET'S TALK LOSS FUNCTIONS….

THE CHOICE OF THE LOSS FUNCTION IS CRITICAL AND WILL DETERMINE THE BEHAVIOR OF THE ALGORITHM (*REMEMBER: LOSS IS WHAT IS MINIMIZED BY THE OPTIMIZATION ALGORITHM*)

SOME ALGORITHMS SUCH AS RFs HAVE LESS FLEXIBILITY. ONE **ADVANTAGE OF NNs IS THAT THE LOSS FUNCTION CAN BE CHANGED VERY EASILY**.

IT IS IMPORTANT TO CHOOSE THE LOSS FUNCTION FOR YOUR PROBLEM

NEURAL NETWORKS AS STATISTICAL MODELS

Let's have a look at the "binary cross-entropy loss". Where does it come from?

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i.log(p(y_i)) + (1-y_i).log(1-p(y_i))$$

GROUND TRUTH

NETWORK OUTPUT (SOFTMAX)

GROUND TRUTH

NETWORK OUTPUT (SOFTMAX)

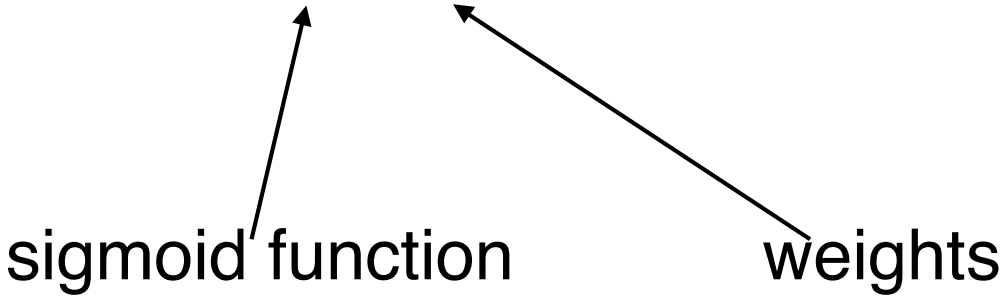
we have a set of independent realizations:

$$(x_i, y_i); i = 1, ..., N$$

and want to find the underlying probability distribution of y given x:

Since y can only take 0 / 1 values, we assume it can be parametrized with a Bernouilli distribution:

$$P(y_i = 0|x_i) = 1 - sigm(f_w(x_i)) \qquad P(y_i = 1|x_i) = sigm(f_w(x_i))$$



So the goal is to find the values of w (weights) that generate a Bernouilli distribution for y given a set of N independent observations

This can be achieved via Maximum Likelihood estimation:

The likelihood of a given observation under the Bernouilli assumption can be written as:

$$L(w; x_i, y_i) = [sigm(f_w(x_i))]^{y_i} [1 - sigm(f_w(x_i))]^{1-y_i}$$

which is equal to $[sigm(f_w(x_i))]$ if y=1 and $[1 - sigm(f_w(x_i))]$ if y=0

And the likelihood of the entire sample is the product of the likelihoods:

$$L(w; x_i, y_i) = \prod_{i=1}^{N} [sigm(f_w(x_i))]^{y_i} [1 - sigm(f_w(x_i))]^{1-y_i}$$

So, the log-likelihood:

$$l(w; x_i, y_i) = \sum_{i=1}^{N} y_i \times \log[sigm(f_w(x_i))] + (1 - y_i) \times \log[1 - sigm(f_w(x_i))]$$

EREFORE EQUIVALENT TO THE CROSS-ENTROPY LOSS:

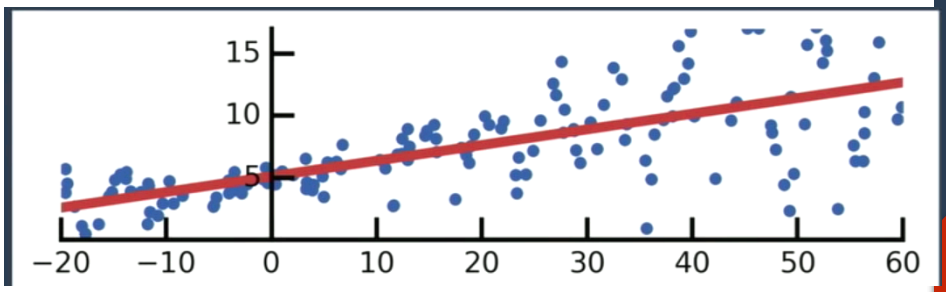
$$l(w; x_i, y_i) = \sum_{i=1}^{N} y_i \times \log[sigm(f_w(x_i))] + (1 - y_i) \times \log[1 - sigm(f_w(x_i))]$$

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^{N} y_i.log(p(y_i)) + (1 - y_i).log(1 - p(y_i))$$

WHEN YOU DO A BINARY CLASSIFICAITON WITH NEURAL NETWORKS YOU ARE SIMPLY FINDING THE WEIGHTS OF YOUR MODEL THAT MAXIMIZE THE LIKELIHOOD OF A BERNOUILLI DISTRIBUTION

NOTE THAT SINCE fw IS FIXED (THE ARCHITECTURE), WE FIND A SOLUTION AMONG A POSSIBLE SET OF SOLUTIONS

WHAT ABOUT REGRESSIONS?




```
model = tf.keras.Sequential([
  tf.keras.layers.Dense(hidden_units, ...),
  tf.keras.layers.Dense(1),
])
model.compile(loss=tf.keras.mean_squared_error)
```

WHEN USING THE MEAN SQUARED ERROR, WE ARE ASSUMING THAT THE OUTPUT POINTS FOLLOW A NORMAL DISTRIBUTION. OUR ESTIMATOR IS THEREFORE THE MEAN OF THE NORMAL PDF THAT MAXIMIZES THE LIKELIHOOD.

$$L(w, \sigma^2; y, X) = (2\pi\sigma^2)^{-N/2} \exp(-\frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - f_w(x_i))^2)$$

weights

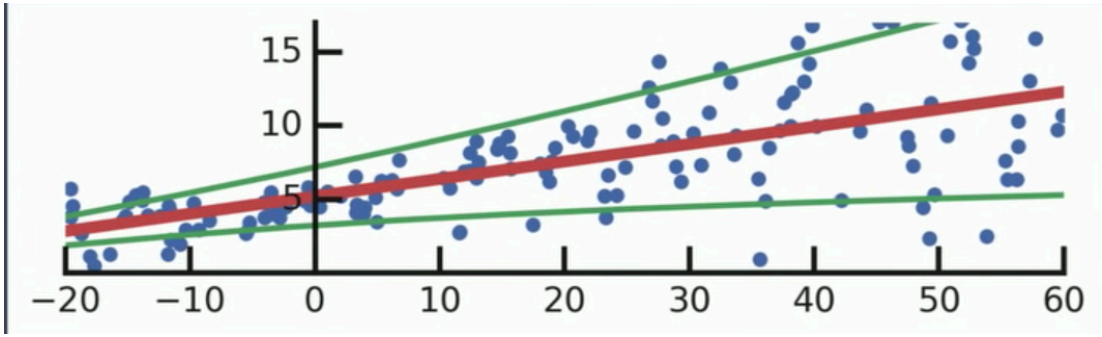
neural network

SO, IN A NUTSHELL, WE ARE DOING BAYESIAN VARIATIONAL INFERENCE ASSUMING THEREFORE A PDF FOR OUR OUTPUT VARIABLE (BERNOUILLI FOR CLASSIFICATION, NORMAL FOR REGRESSION)

Tensorflow probability allows now to generalize this to any distribution:

Guess what! *You can*.

```
model = tf.keras.Sequential([
  tf.keras.layers.Dense(hidden_units, ...),
  tf.keras.layers.Dense(1),
  tfp.layers.DistributionLambda(lambda t:
    tfd.Normal(loc=t[..., 0],
               scale=1)),
```



TensorFlow

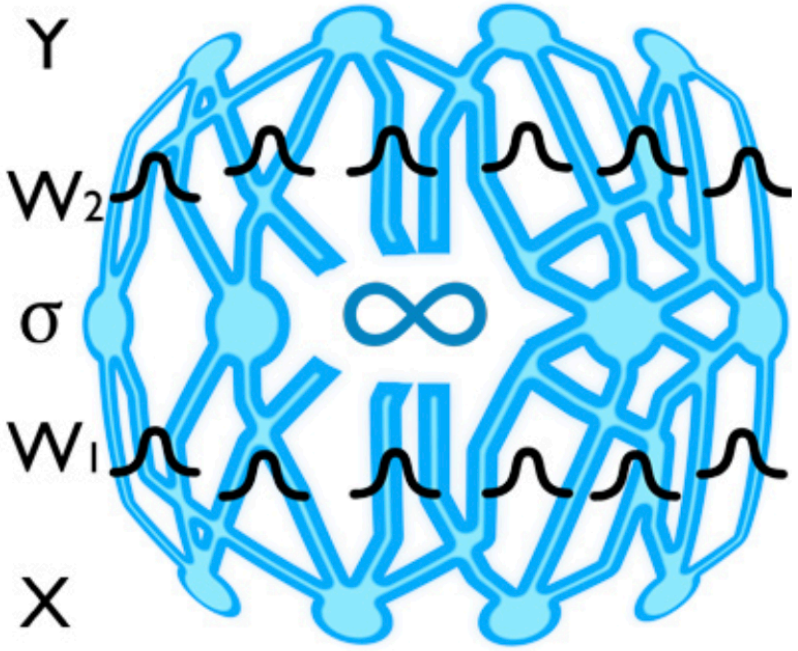
DEV SUMMIT 2019

```
~~model.compile(loss=tf.keras.mean_squared_error)~~
model.compile(loss=lambda y, rv_y: -rv_y.log_prob(y))
yhat = model(x_tst) # A Distribution, not a Tensor!
```

} Our wish.

SEE THIS GREAT TALK: https://www.youtube.com/watch?v=BrwKURU-wpk

BUT REMEMBER, WE ARE CHOOSING ONE MODEL AND ONE VALUE FOR THE WEIGHTS



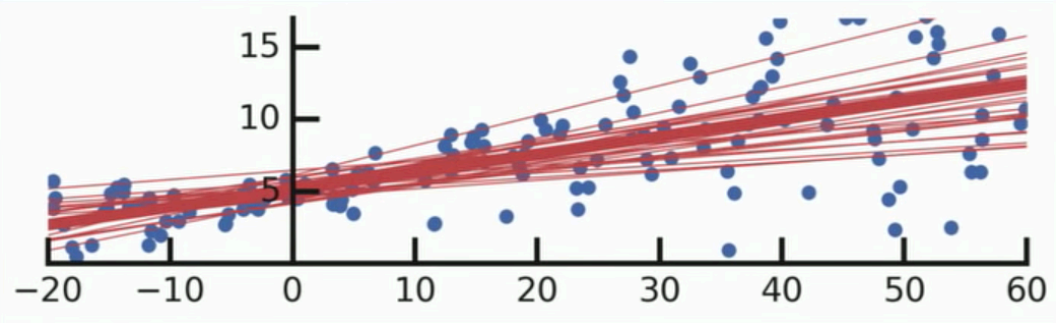
CAPTURING "MODEL UNCERTAINTY"

```
model = tf.keras.Sequential([
  tfp.layers.DenseVariational(hidden_units, ...),
  tfp.layers.DenseVariational(1),
  tfp.layers.DistributionLambda(lambda t:
    tfd.Normal(loc=t[..., 0],
               scale=1)),
])
```

"Bayesian Weights"

Linear Regression

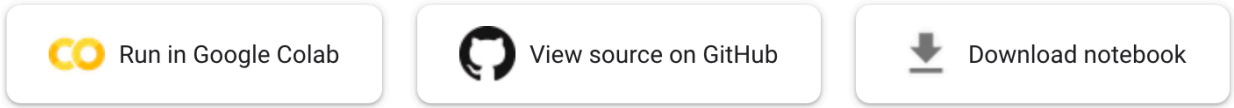
IDEALLY, WE WOULD LIKE TO MARGINALIZE OVER ALL MODELS AND OVER ALL POSSIBLE WEIGHT VALUES



BNNs ADD A PRIOR DISTRIBUTION TO EACH WEIGHT - HARD TO TRAIN

AND YOU STILL ARE LEFT WITH THE CHOICE OF THE MODEL ARCHITECTURE:

Introduction to the Keras Tuner



Overview

The Keras Tuner is a library that helps you pick the optimal set of hyperparameters for your TensorFlow program. The process of selecting the right set of hyperparameters for your machine learning (ML) application is called *hyperparameter tuning* or *hypertuning*.

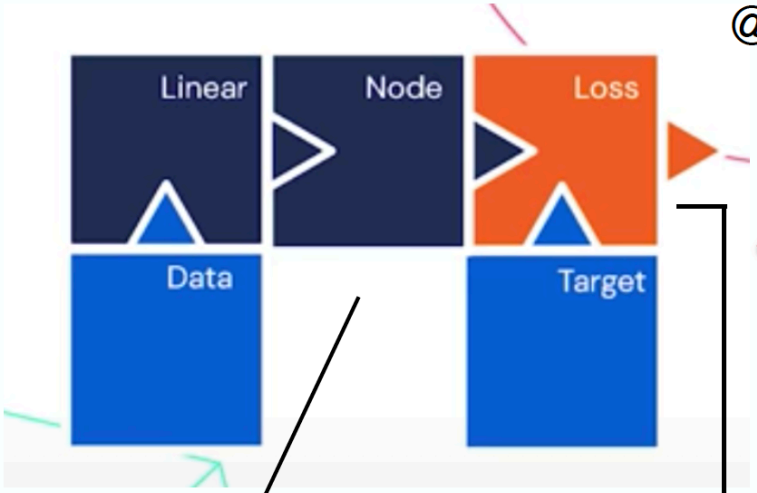
Hyperparameters are the variables that govern the training process and the topology of an ML model. These variables remain constant over the training process and directly impact the performance of your ML program. Hyperparameters are of two types:

1. **Model hyperparameters** which influence model selection such as the number and width of hidden layers
2. **Algorithm hyperparameters** which influence the speed and quality of the learning algorithm such as the learning rate for Stochastic Gradient Descent (SGD) and the number of nearest neighbors for a k Nearest Neighbors (KNN) classifier

In this tutorial, you will use the Keras Tuner to perform hypertuning for an image classification application.

RECAP:

(deepmind @Czarnecki)

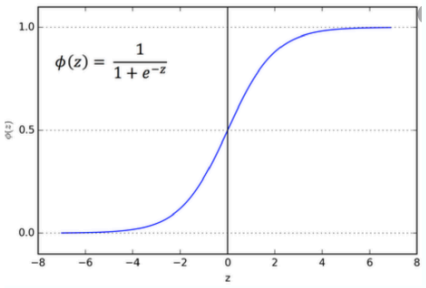


$$f_W(\vec{x}) = \vec{y}$$

$$\vdots$$

$$g_3(W_3 g_2(W_2 g_1(W_1 \vec{x_0})))$$

activation function



$$\frac{1}{N}\sum_{i=1}^{N}(y_i - f_w(x))^2$$

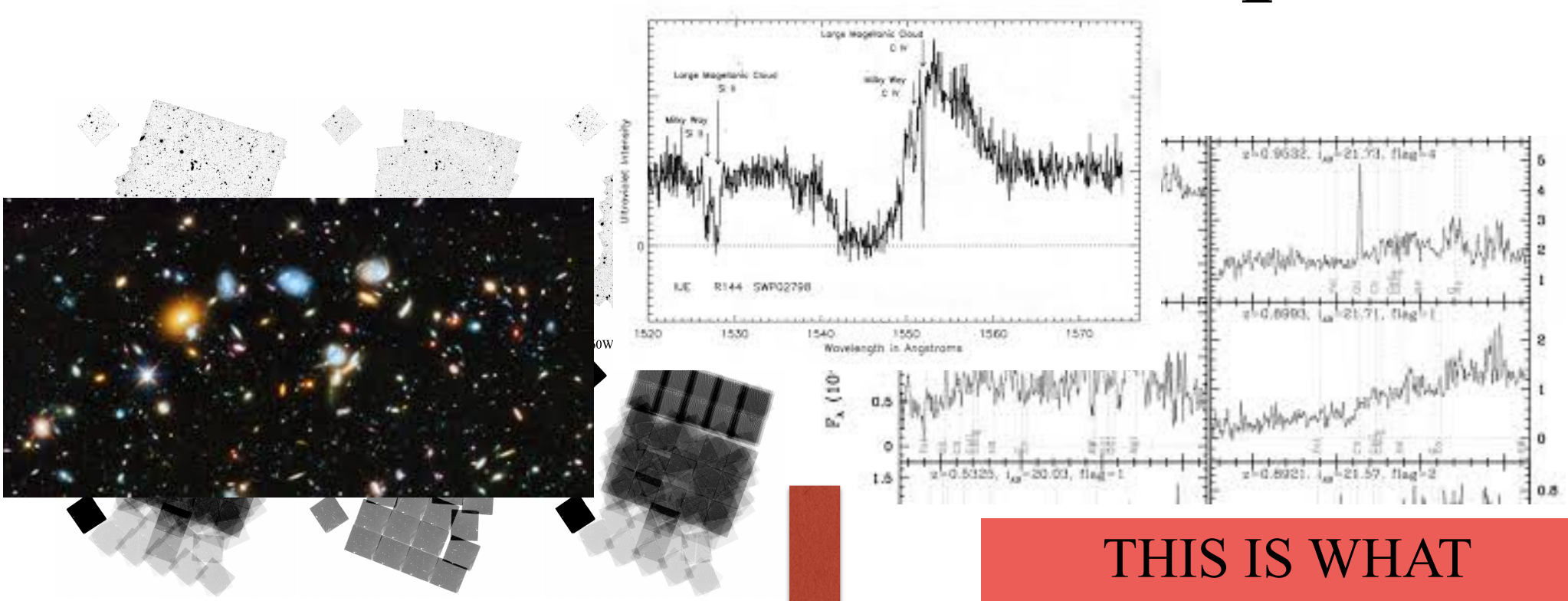
Minimized through gradient descent (backpropagation)

CAN WE GO DEEP NOW?

CAN WE GO DEEP NOW?

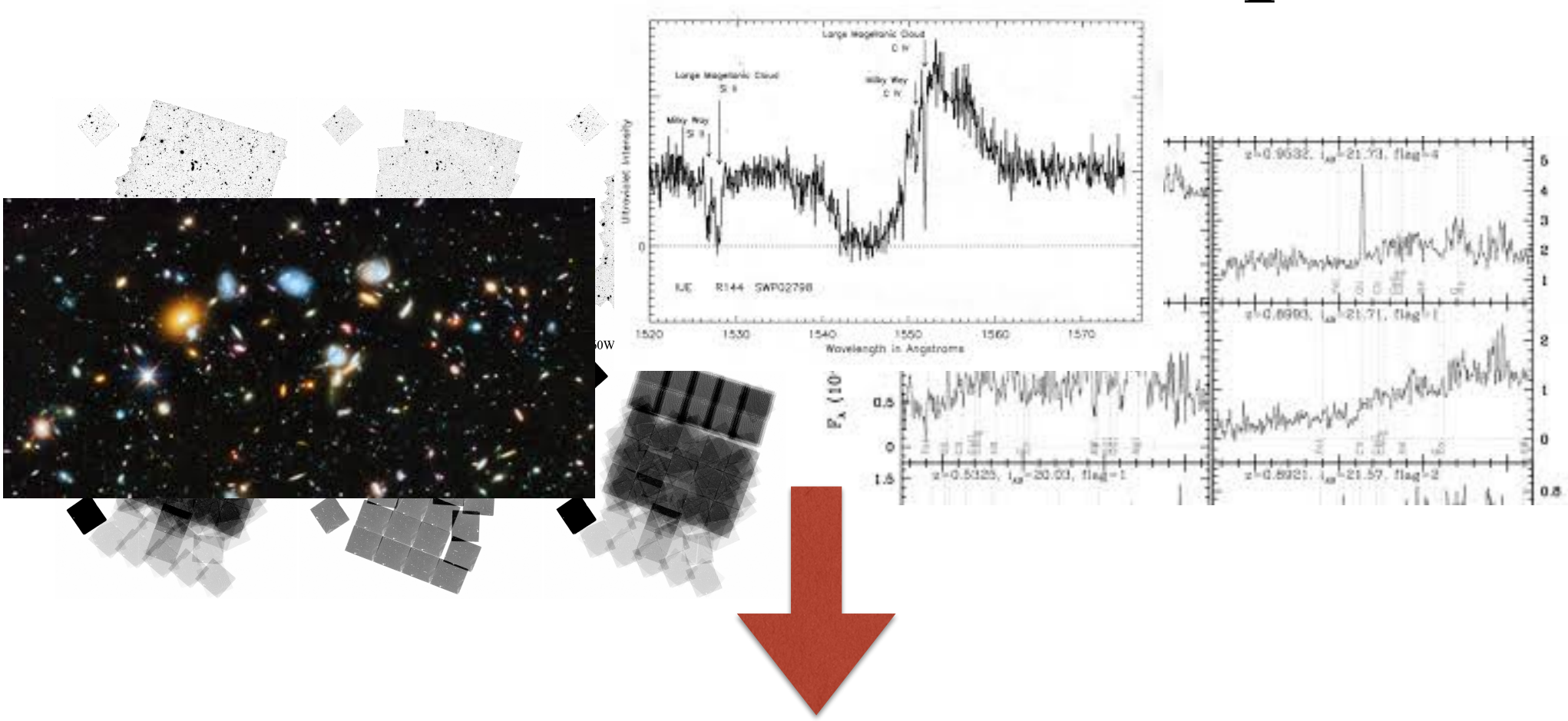
ALMOST THERE…LET'S THINK FOR A MOMENT ABOUT WHAT WE PUT AS INPUT…

What do we put as input?



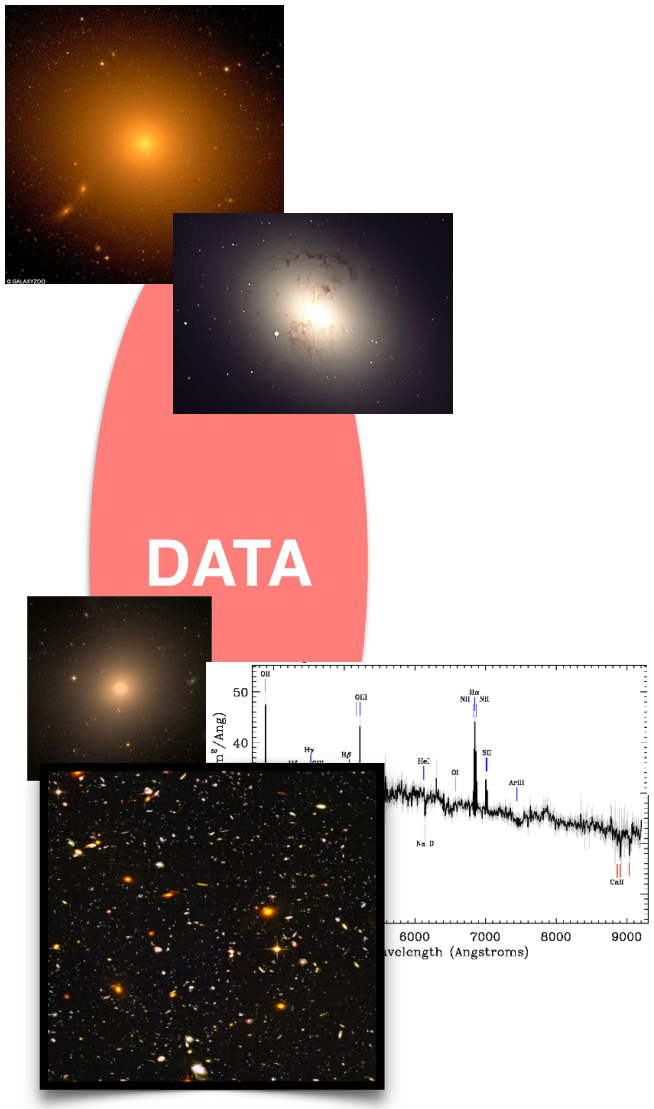
THIS IS WHAT MACHINES SEE

What do we put as input?



PRE-PROCESS DATA TO EXTRACT MEANINGFUL INFORMATION

THIS IS GENERALLY CALLED **FEATURE EXTRACTION**



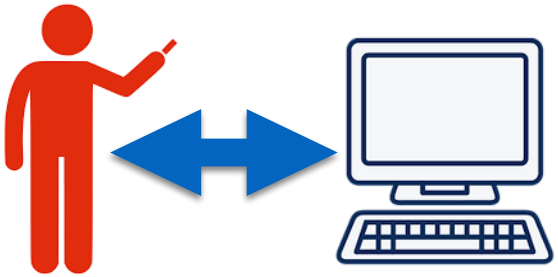
Spiral!

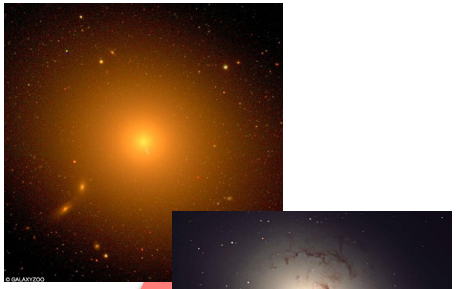
Emission line!

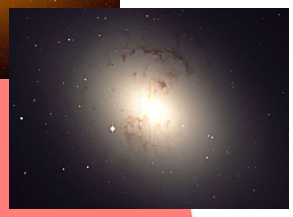
Merger!

Clump!

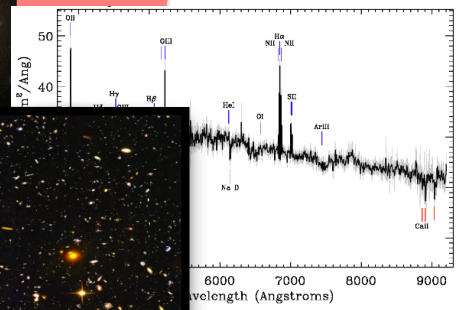
AGN!







DATA



Spiral!

Emission line!

Merger!

Clump!

AGN!

$$f_{\vec{W}}(\vec{x}) = \vec{y}$$

LABEL

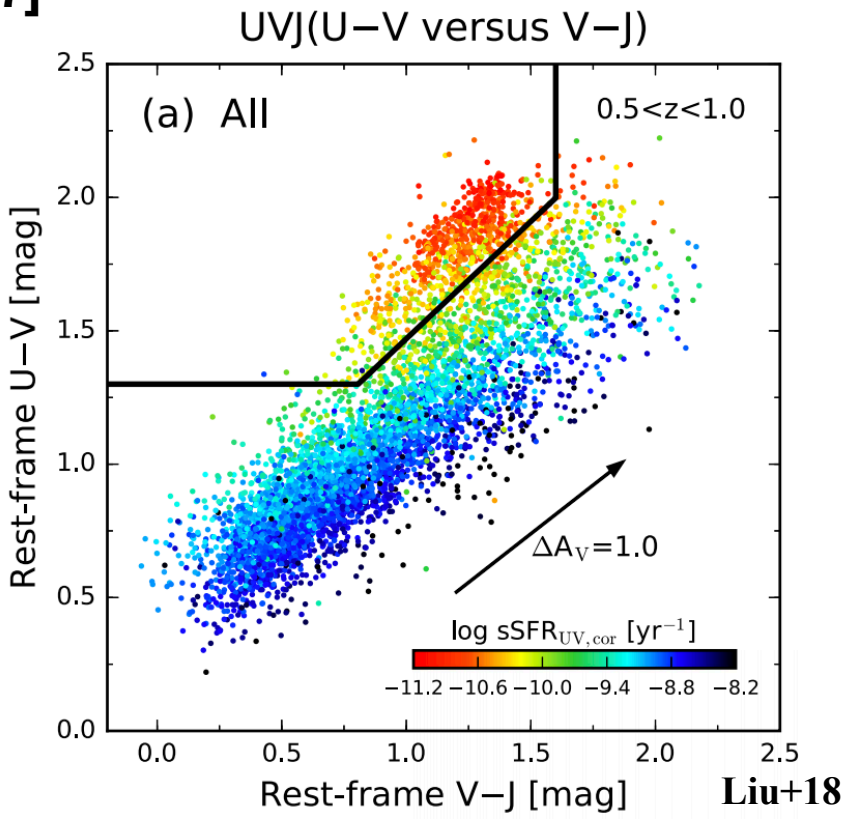
Q(0) , SF(1)

NETWORK FUNCTION

(U-V, V-J) FEATURES

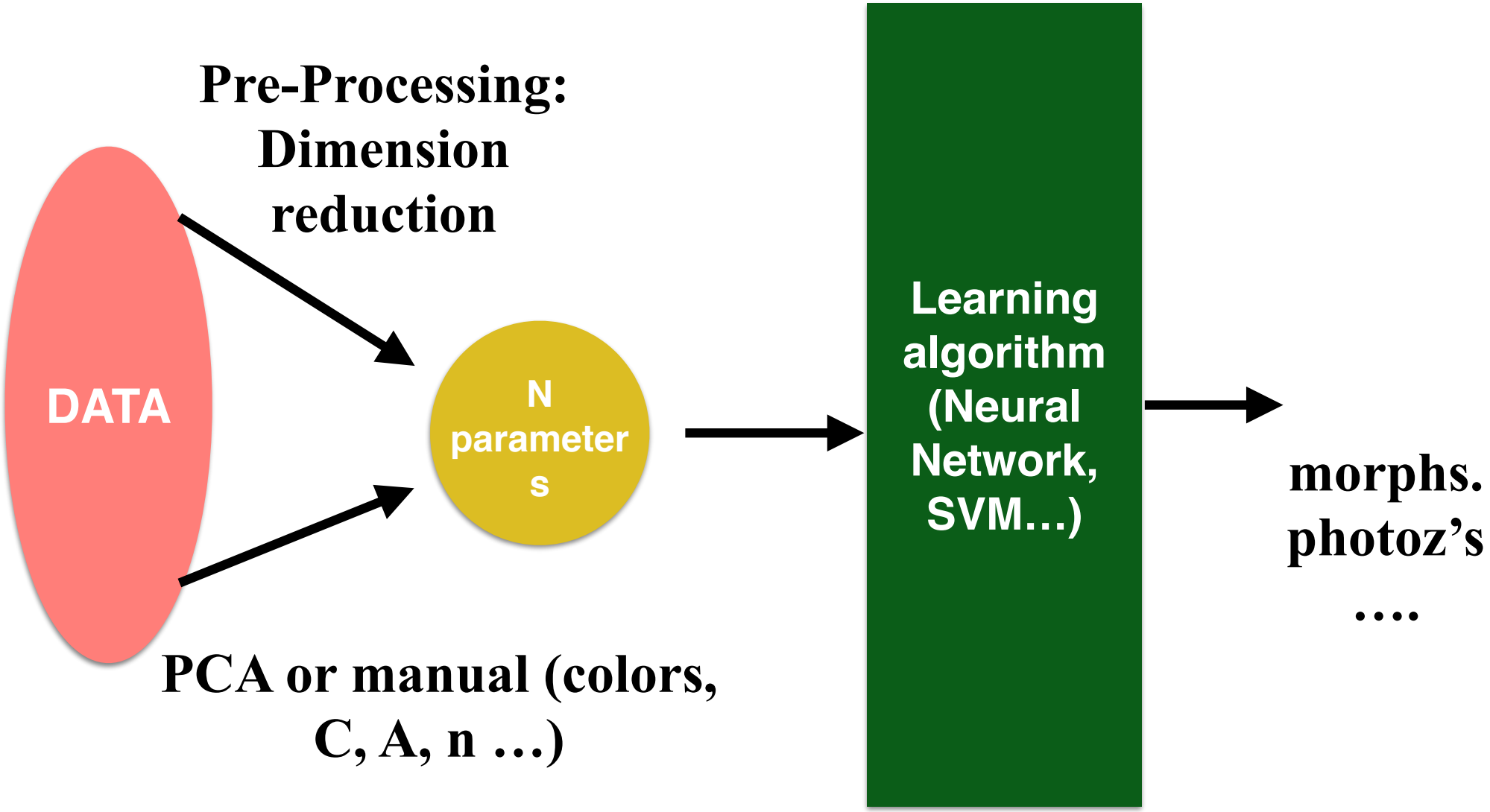
sgn[(u-v)-0.8*(v-j)-0.7]

WEIGHTS

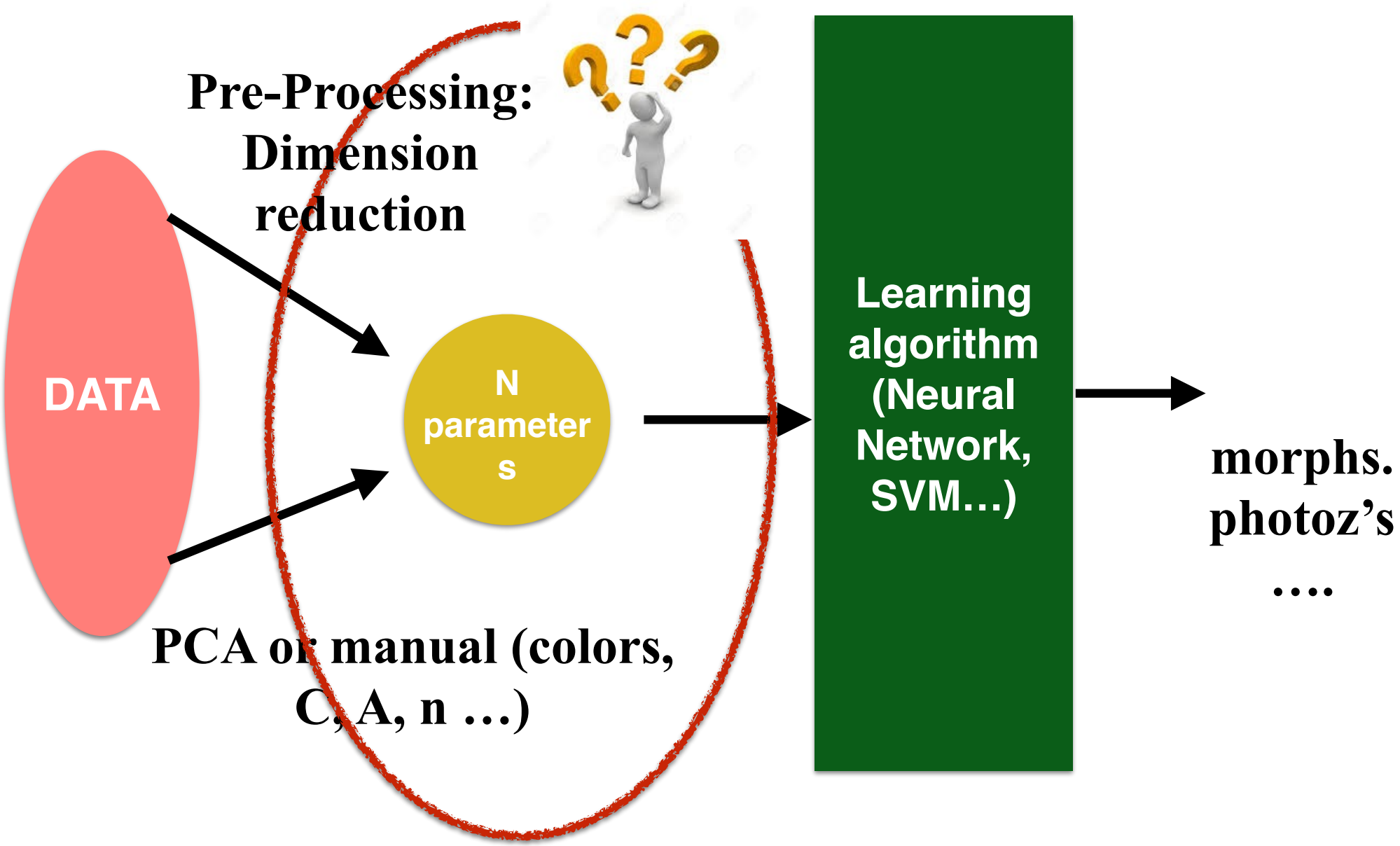


Liu+18

THE "CLASSICAL" APPROACH

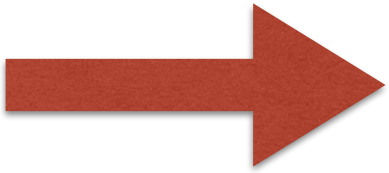


"CLASSICAL" MACHINE LEARNING



In Astronomy

- Colors, Fluxes
- Shape indicators
- Line ratios, spectral features
- Stellar Masses, Velocity Dispersions



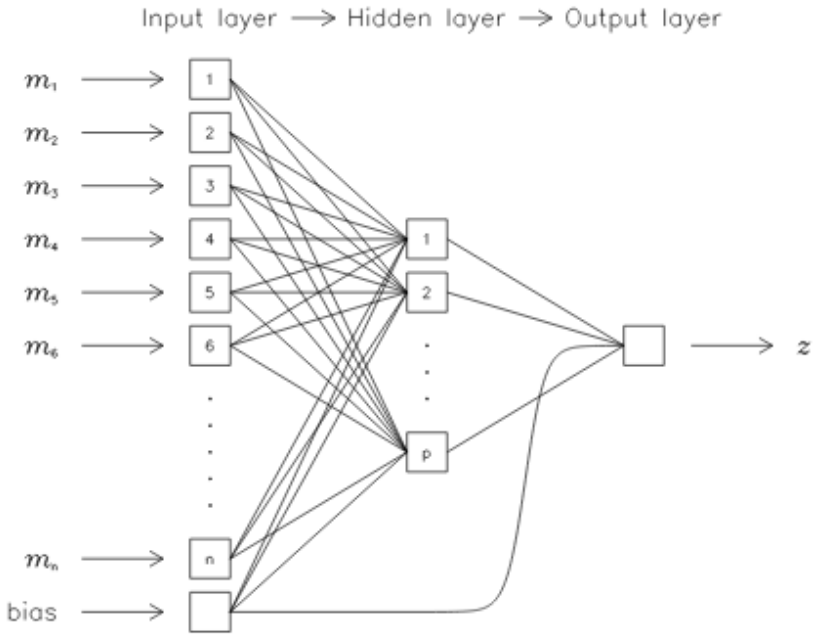
Requires specialized software before feeding the machine learning algorithm

IT IMPLIES A DIMENSIONALITY REDUCTION!

PHOTOMETRIC REDSHIFTS

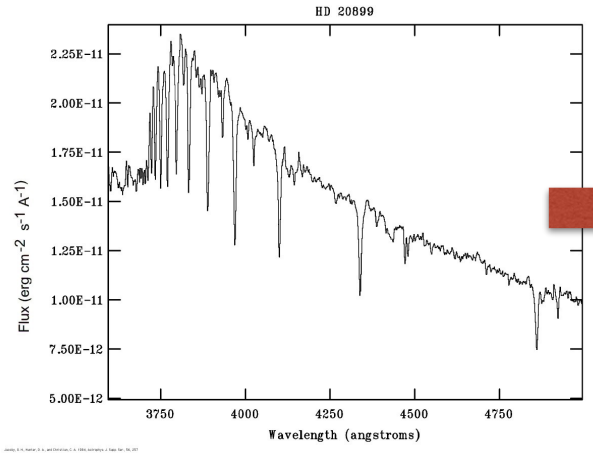
SDSS

g
r
i
z

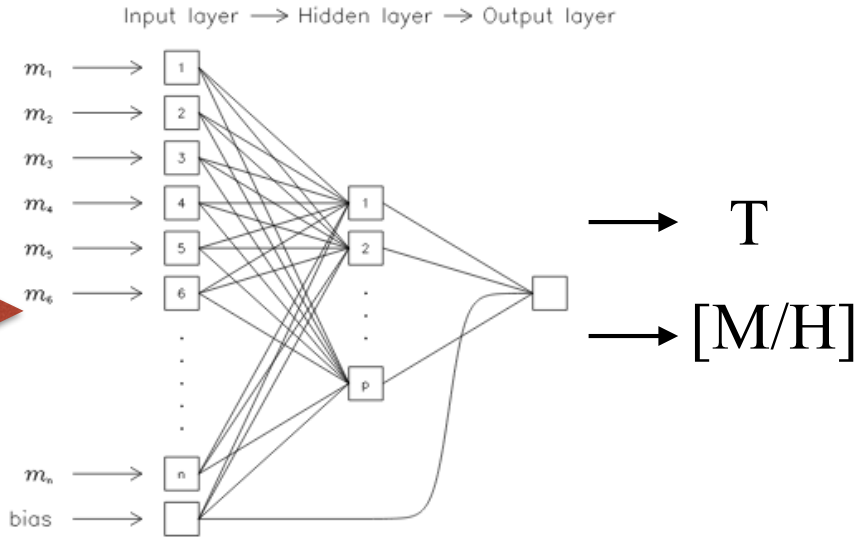


Collister+08

STELLAR PARAMETERS FROM MEDIUM BAND FILTERS



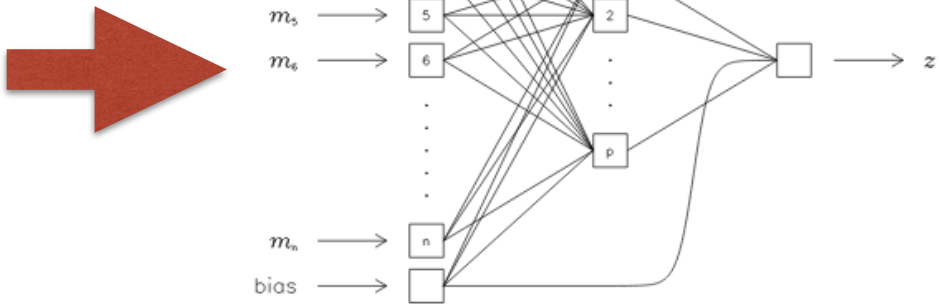
MEDIUM BAND FLUXES



Bailer-Jones+00

No.	Symbol	Description	Scale[a]
1	CVD	Central velocity dispersion	~1 kpc
2	M_{bulge}	Bulge stellar mass	0.5–4 kpc
3	Re	Bulge effective radius	0.5–4 Kpc
4	B/T	Bulge-to-total stellar mass ratio	0.5–8 kpc
5	M_*	Total stellar mass	2–8 kpc
6	M_{disc}	Disc stellar mass	4–10 kpc
7	M_{halo}	Group halo mass	0.1–1 Mpc
8	δ_5	Local density parameter	0.5–3 Mpc

Notes. [a]Approximate 1σ range from centre of galaxy. For photometric quantities half-light radii are used.



HEAVILY PROCESSED DATA

Teimiroonia+16

Other general computer vision features [for images!]

- Pixel Concatenation
- Color histograms
- Texture Features
- Histogram of Gradients
- SIFT

FOR MANY YEARS COMPUTER VISION RESEARCHERS HAVE BEEN TRYING TO FIND THE MOST GENERAL FEATURES

Other general computer vision features [for images!]

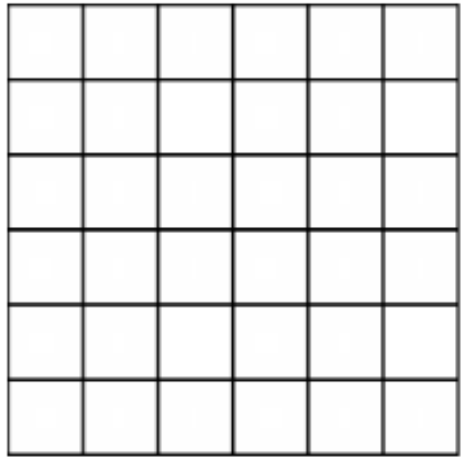
- Pixel Concatenation
- Color histograms
- Texture Features
- Histogram of Gradients
- SIFT

FOR MANY YEARS COMPUTER VISION RESEARCHERS HAVE BEEN TRYING TO FIND THE MOST GENERAL FEATURES

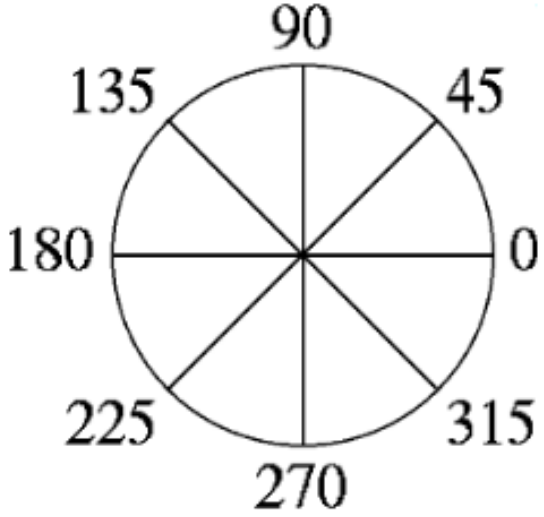
THE BEST CLASSICAL SOLUTION [BEFORE 2012] WHERE BASED **ON LOCAL FEATURES**

HISTOGRAM OF ORIENTED GRADIENTS (HoG)

1. DIVIDE IMAGE INTO SMALL SPATIAL REGIONS CALLED CELLS

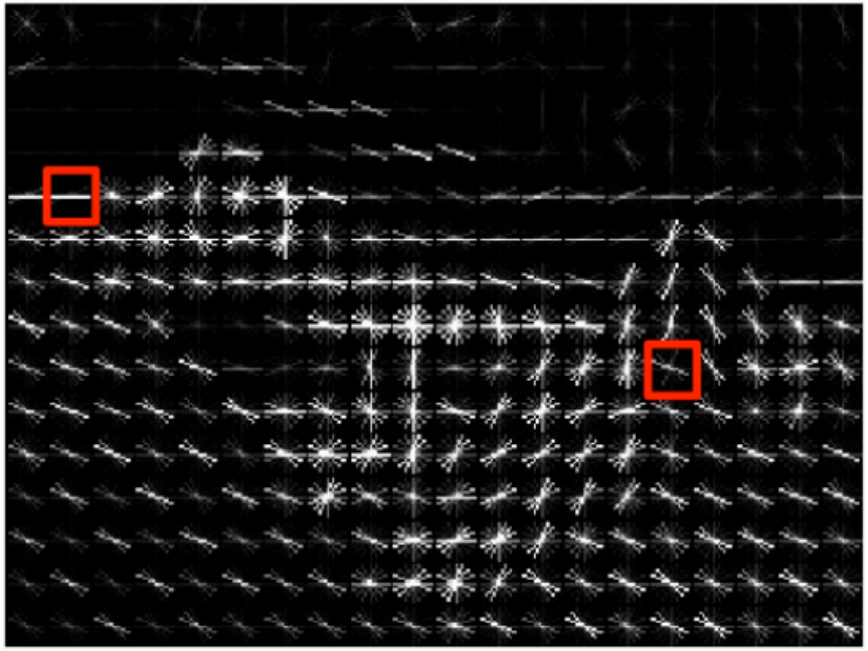


2. COMPUTE INTENSITY GRADIENTS OVER N DIRECTIONS [TYPICALLY 9 FOR IMAGE]



3. COMPUTE WEIGHTED 1-D HISTOGRAM OF ALL DIRECTIONS. A CELL IS REDUCED TO N NUMBERS

HISTOGRAM OF ORIENTED GRADIENTS (HoG)



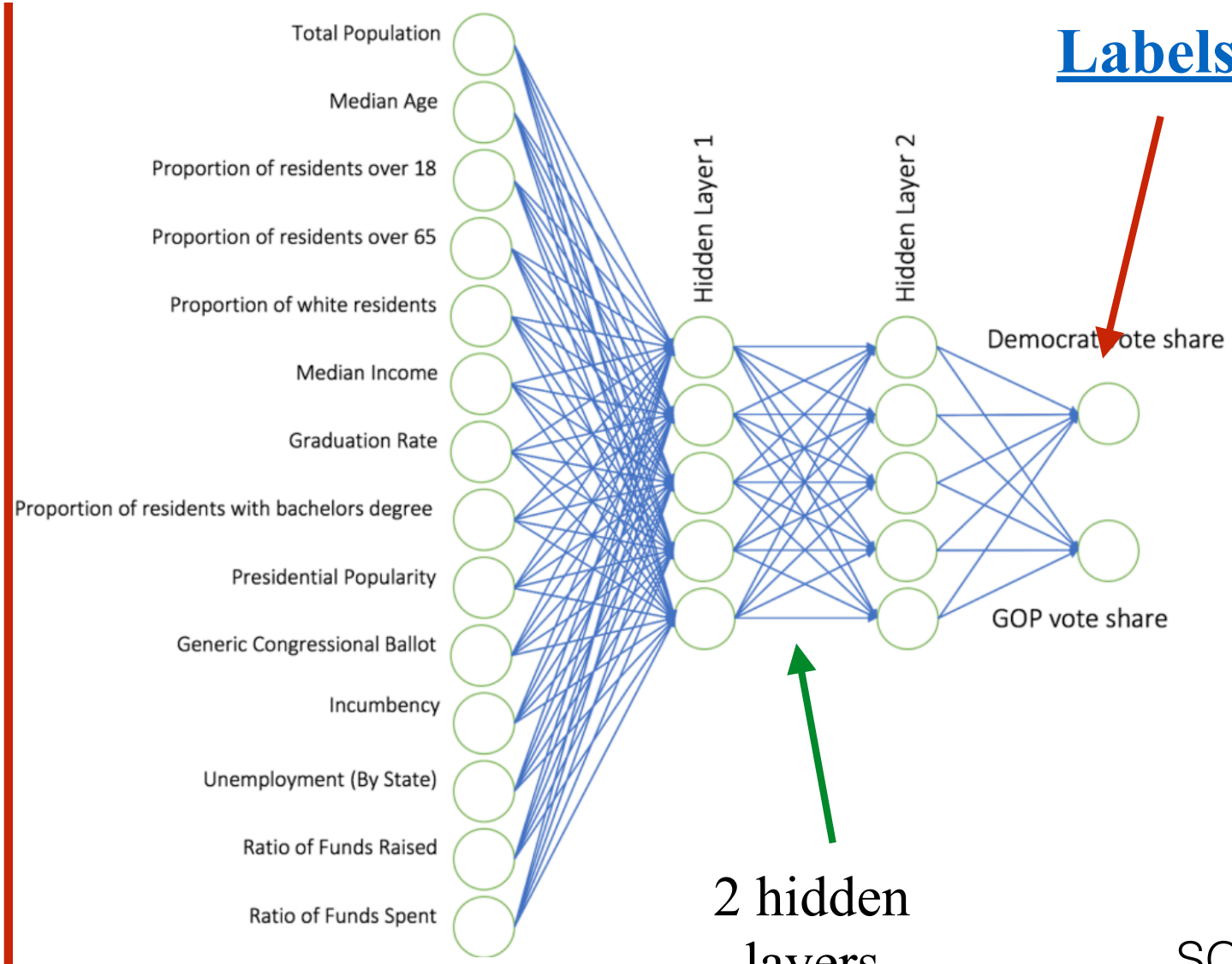
EVERYTHING IS IN THE FEATURES….WHAT IF I IGNORED SOME IMPORTANT FEATURES?

EVERYTHING IS IN THE FEATURES….WHAT IF I IGNORED SOME IMPORTANT FEATURES?

NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS

$$p = g_3(W_3 g_2(W_2 g_1(W_1 \vec{x_0})))$$

Features (x)



Labels (y)

2 hidden layers

SOURCE

NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS

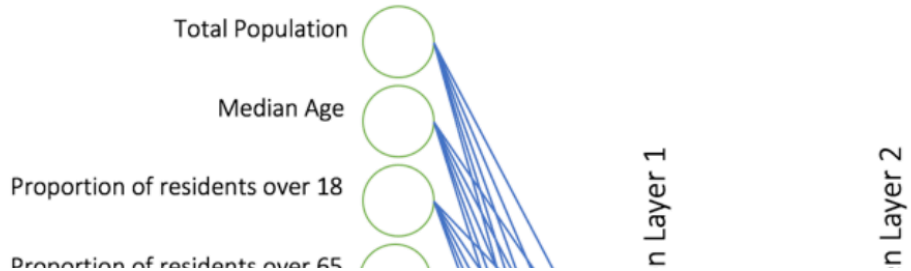
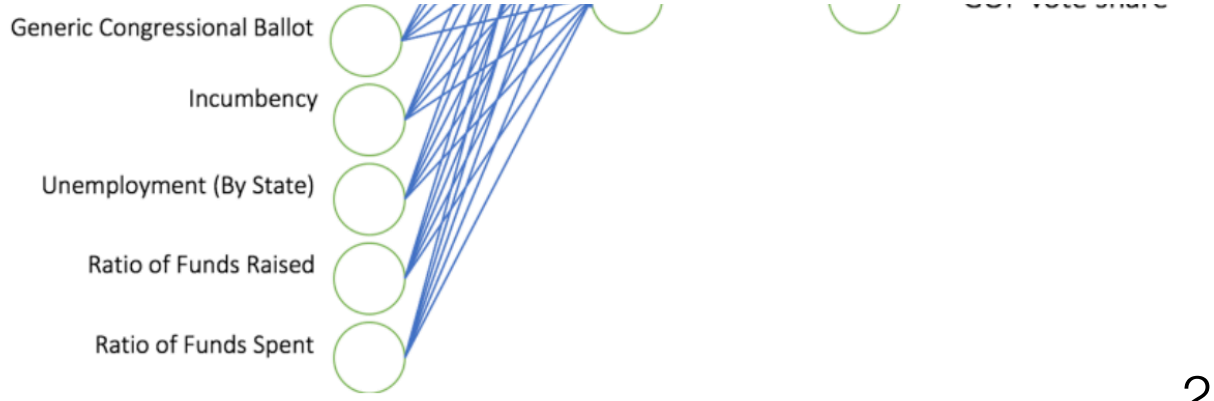


Table 2 – Results of both Models:

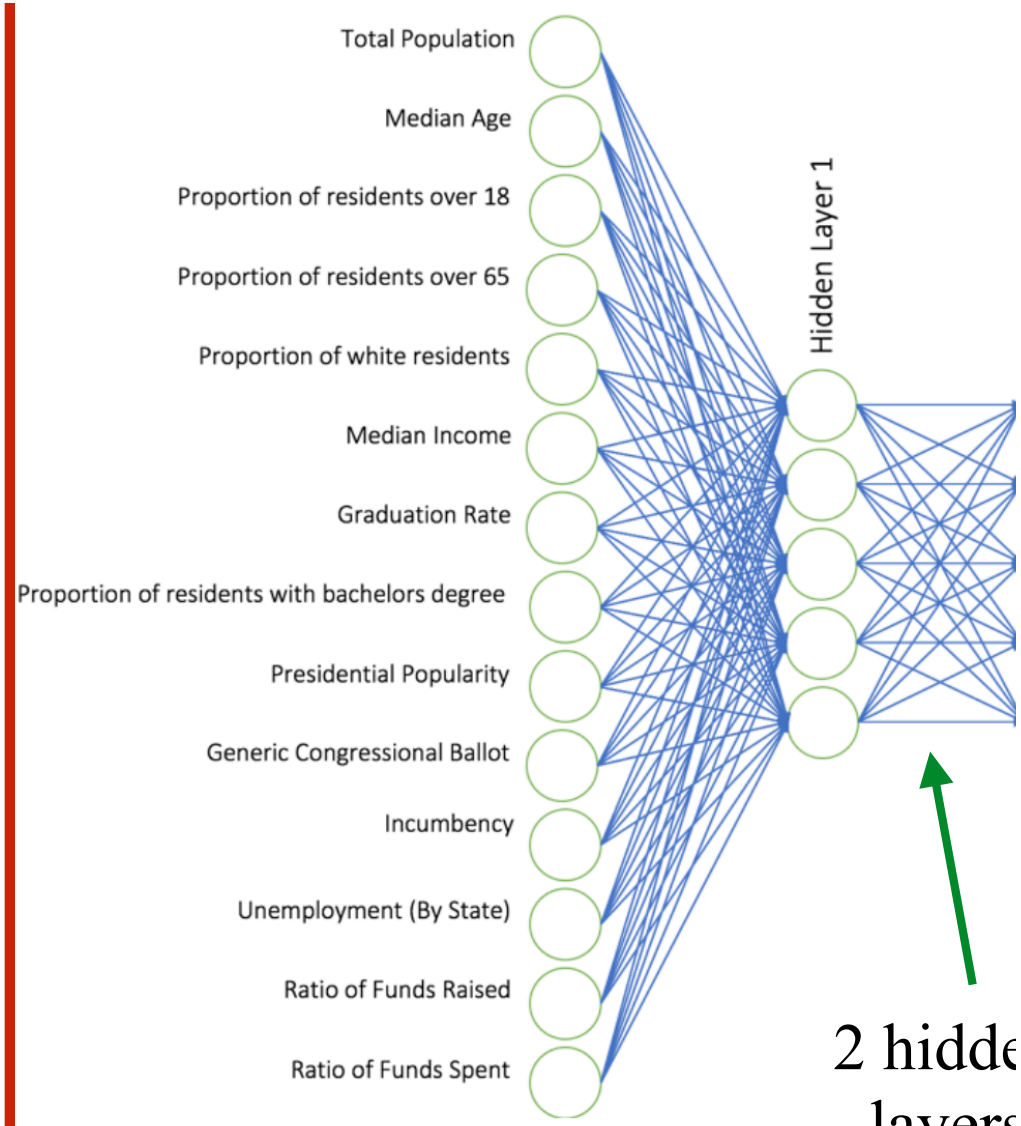
Model not including past elections (Model A)	***Model including past elections (Model B)***
D +3	***D +17***
Democrat Seats: 219	***Democrat Seats: 226***
Republican Seats: 216	***Republican Seats: 209***
33% chance of Republicans keeping house*	**0.3% chance of Republicans keeping house***



232+198

NEURAL NETWORK TO PREDICT RESULTS OF MIDTERM ELECTIONS

Features (x)



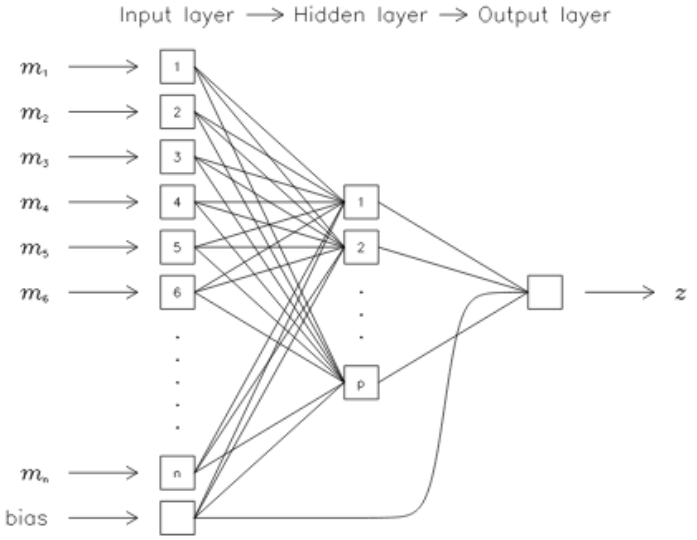
Bad Weather, Known to Lower Turnout, Will Greet Many Voters

Rain can decrease voter numbers, which studies show tends to help Republicans. “I hope it rains hard tomorrow,” one Republican candidate said.

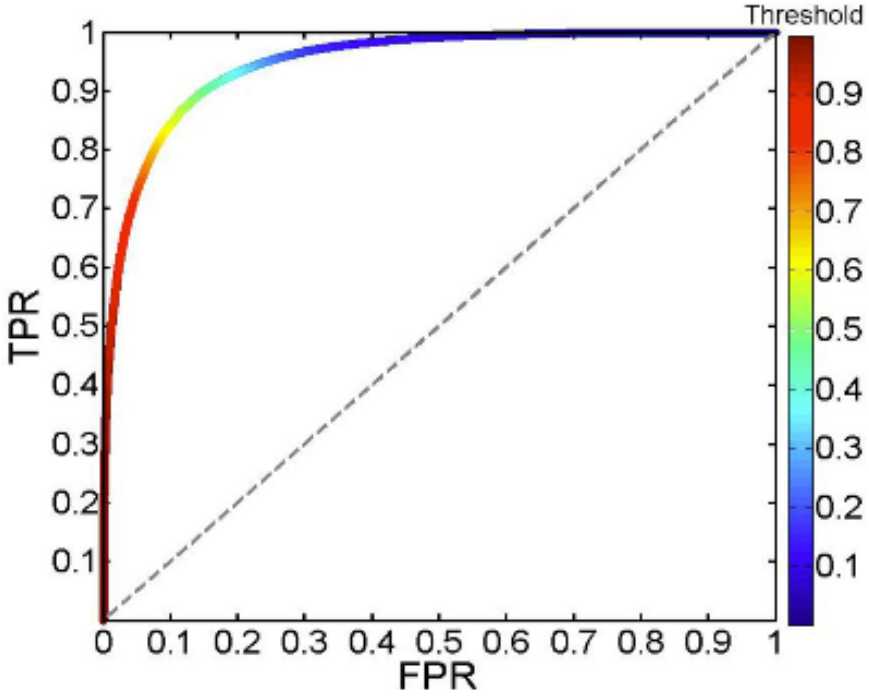
10h ago

No.	Symbol	Description	Scale[a]
1	CVD	Central velocity dispersion	~1 kpc
2	M_{bulge}	Bulge stellar mass	0.5–4 kpc
3	Re	Bulge effective radius	0.5–4 Kpc
4	B/T	Bulge-to-total stellar mass ratio	0.5–8 kpc
5	M_*	Total stellar mass	2–8 kpc
6	M_{disc}	Disc stellar mass	4–10 kpc
7	M_{halo}	Group halo mass	0.1–1 Mpc
8	δ_5	Local density parameter	0.5–3 Mpc

Notes. [a]Approximate 1σ range from centre of galaxy. For photometric quantities half-light radii are used.

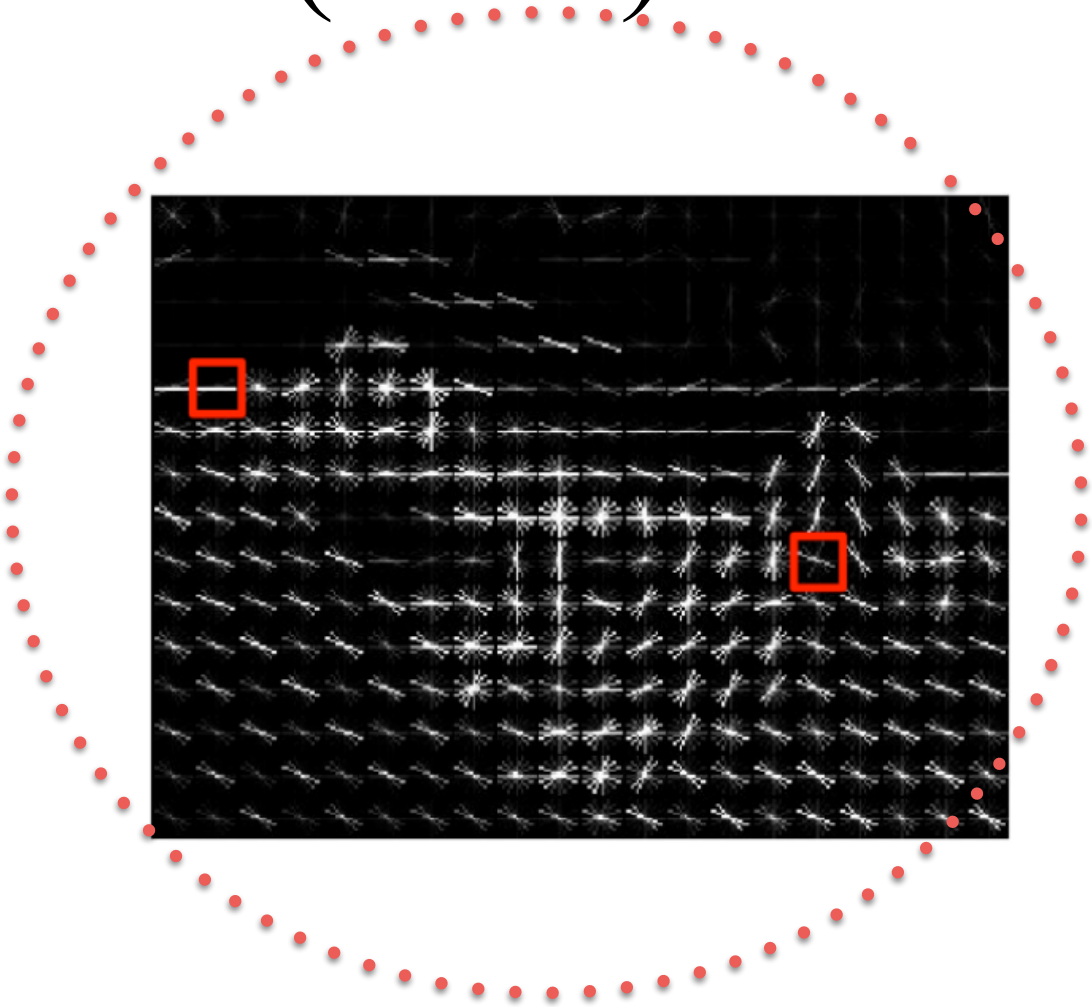


HEAVILY PROCESSED DATA

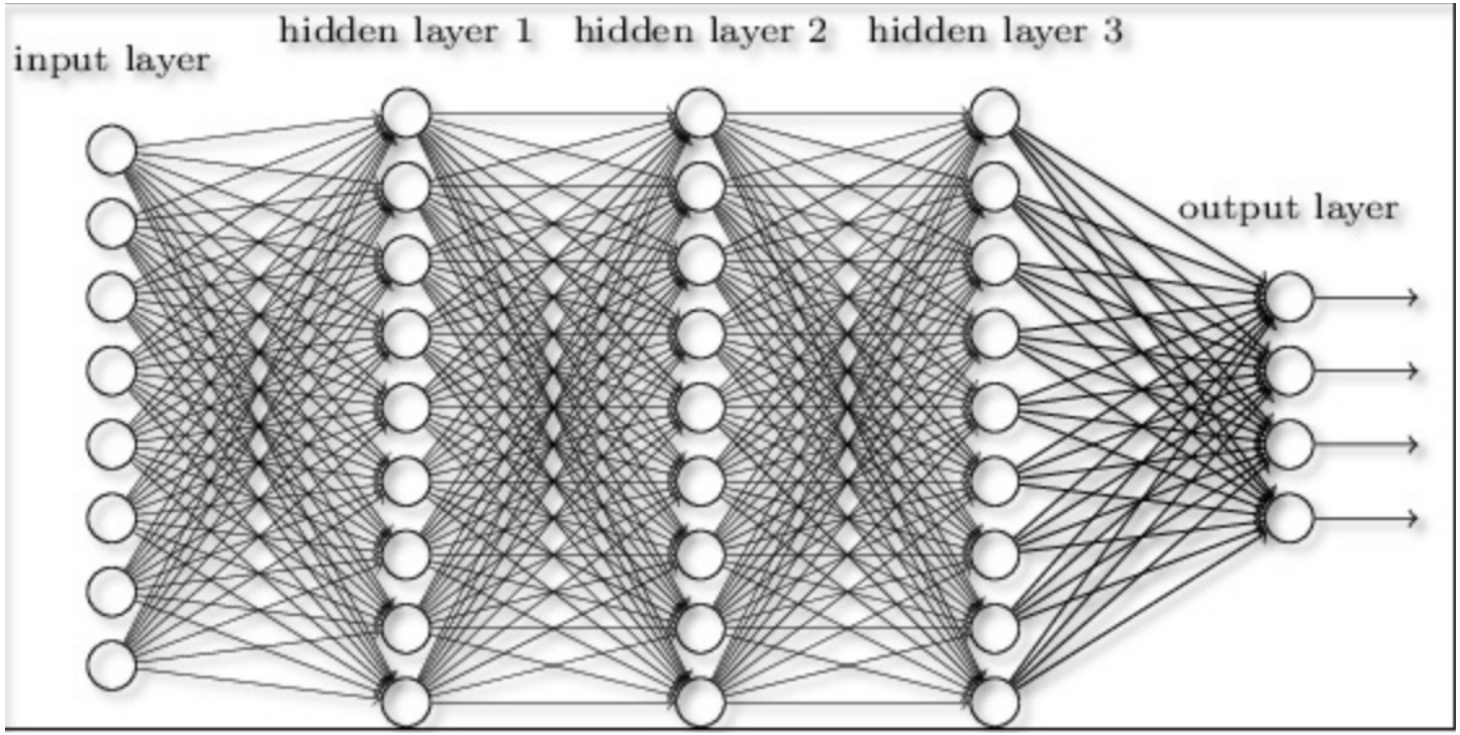


Teimiroonia+16

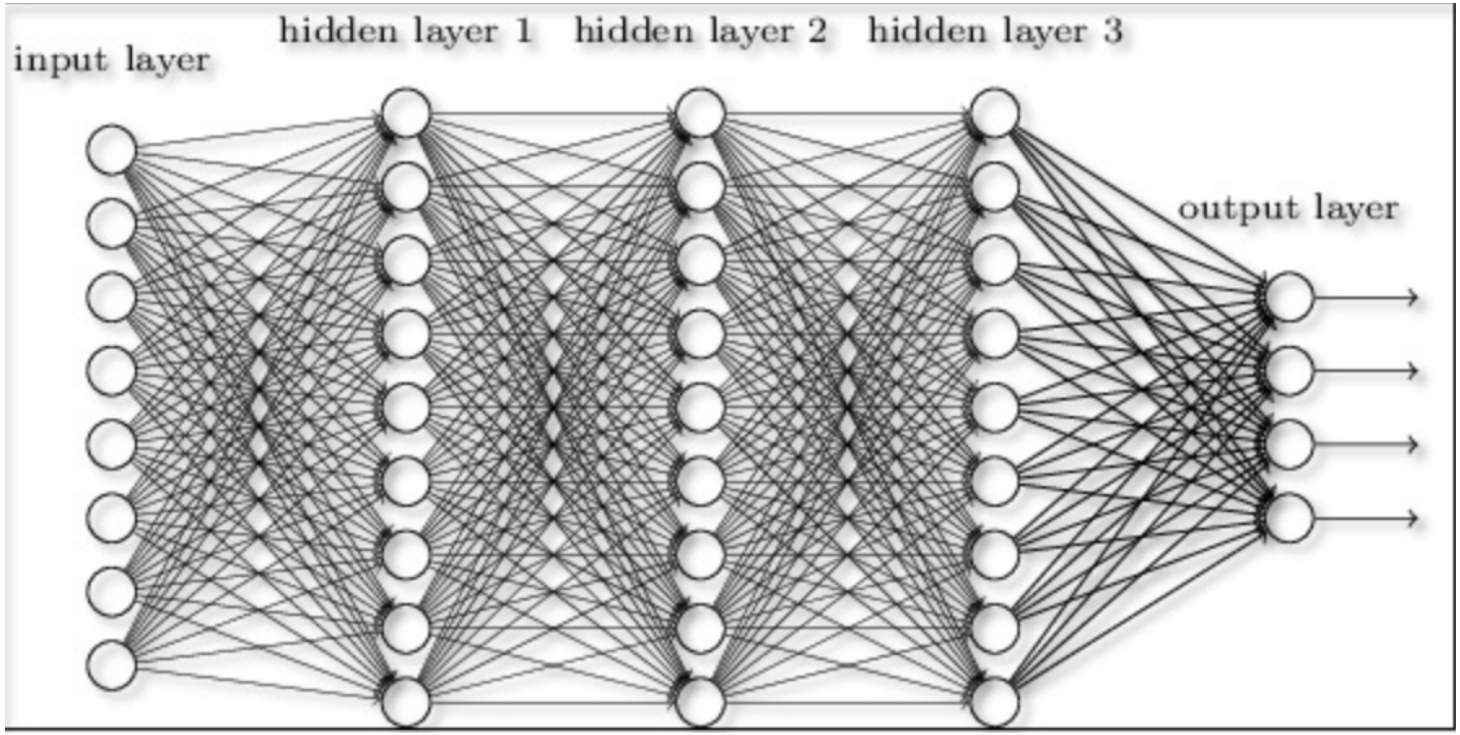
HISTOGRAM OF ORIENTED GRADIENTS (HoG)



KEEP THIS IMAGE IN MIND FOR LATER…

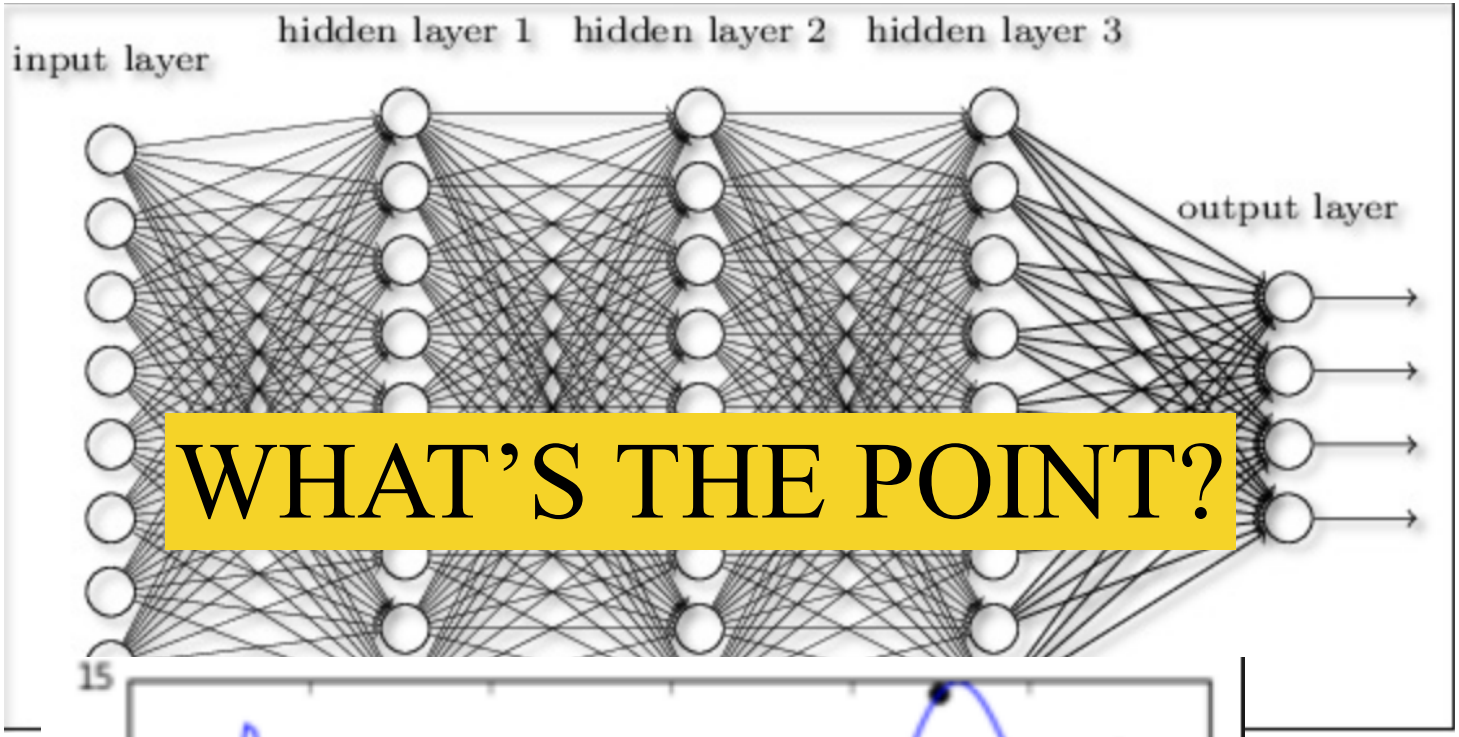


DEEPER —> INCREASING NUMBER OF PARAMETERS



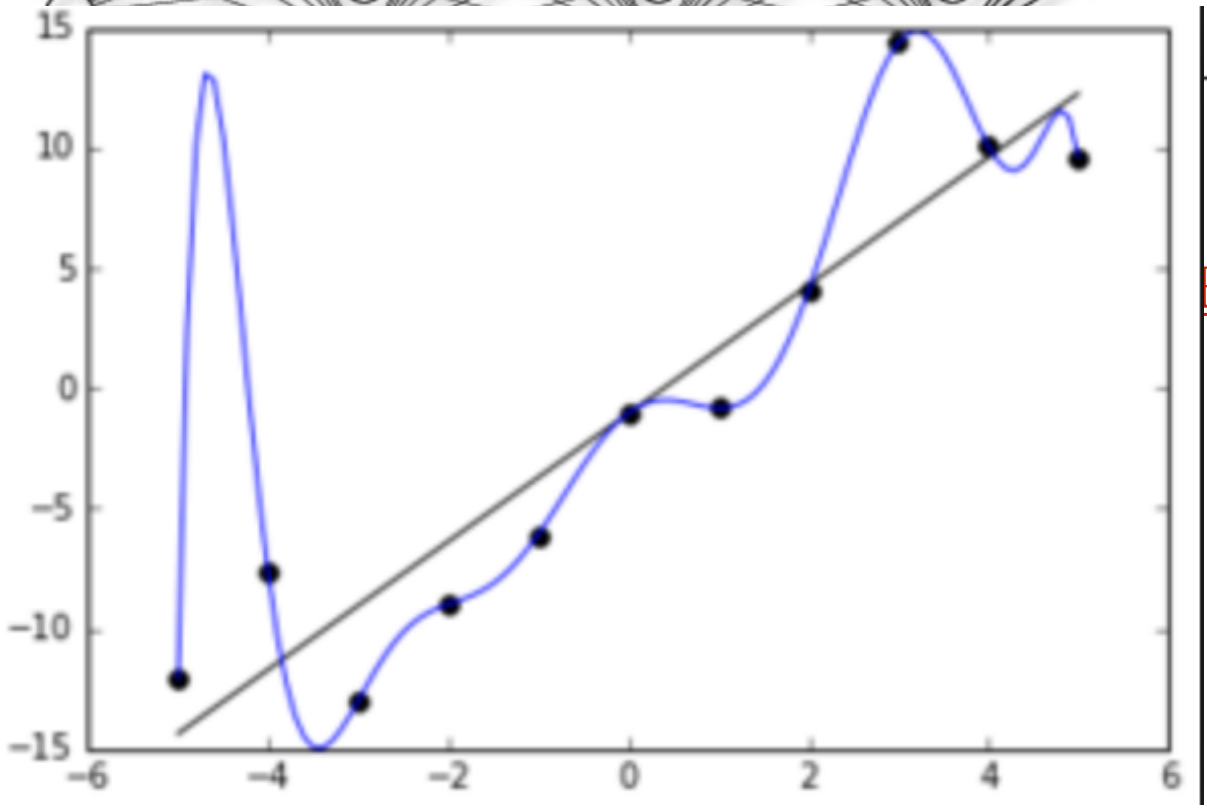
DEEPER —> INCREASING NUMBER OF PARAMETERS

IF OUR INPUT CONTAINS A HANDFUL
OF PARAMETERS [colors, sizes, lines etc..]



WHAT'S THE POINT?

DEEP ETERS



WHAT ABOUT USING RAW DATA?

ALL INFORMATION IS IN THE INPUT DATA

WHY REDUCING ?

LET THE NETWORK FIND THE INFO

WHAT ABOUT USING RAW DATA?

ALL INFORMATION IS IN THE INPUT DATA

WHY REDUCING ?

LET THE NETWORK FIND THE INFO

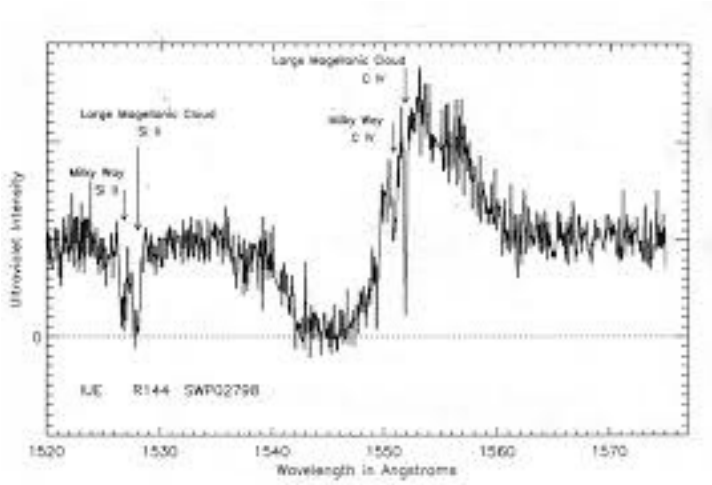
LARGE DIMENSION SIGNALS SUCH AS IMAGES OR SPECTRA WOULD REQUIRE TREMENDOUSLY LARGE MODELS

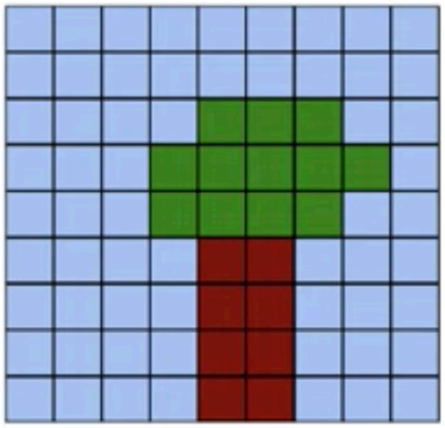
A 512x512 image as input of a fully connected layer producing output of same size:

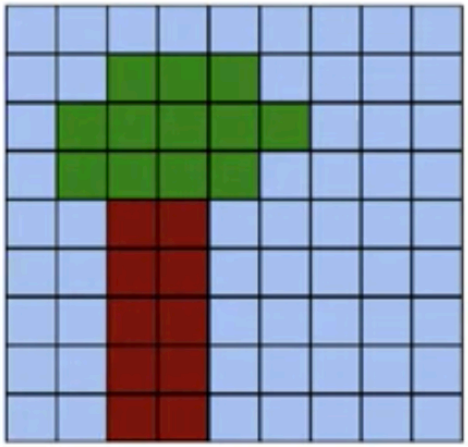
$$(512 \times 512)^2 = 7e10$$

BUT

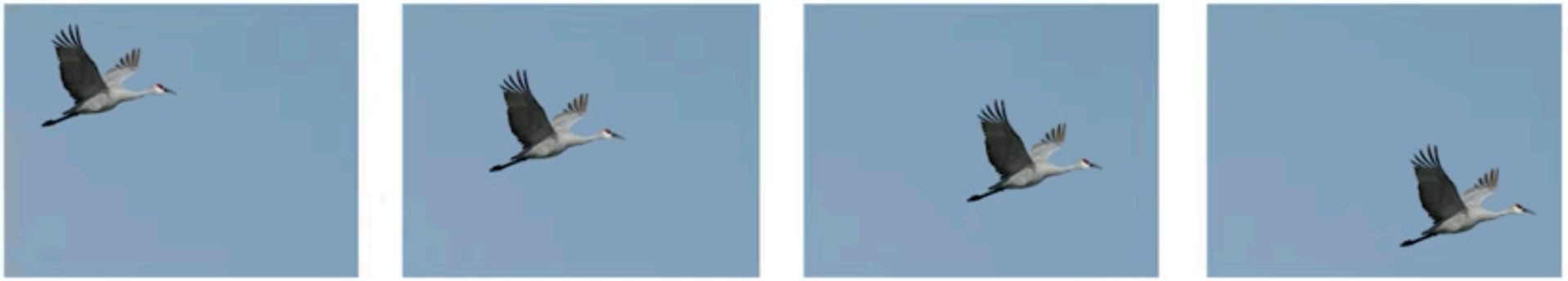
FEEDING INDIVIDUAL RESOLUTION ELEMENTS IS NOT VERY EFFICIENT SINCE IT LOOSES ALL INVARIANCE TO TRANSLATION AND IGNORES CORRELATION IN THE DATA AT ALL SCALES







(Dielemann@Deepmind)



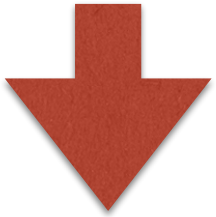
TWO BASIC PROPERTIES OF IMAGING DATA (BUT ALSO SPECTROSCOPY IN SOME SENSE) ARE **LOCALITY**
TRANSLATION INVARIANCE

locality: nearby pixels are more strongly correlated

translational invariance: meaningful patterns can appear anywhere in the image

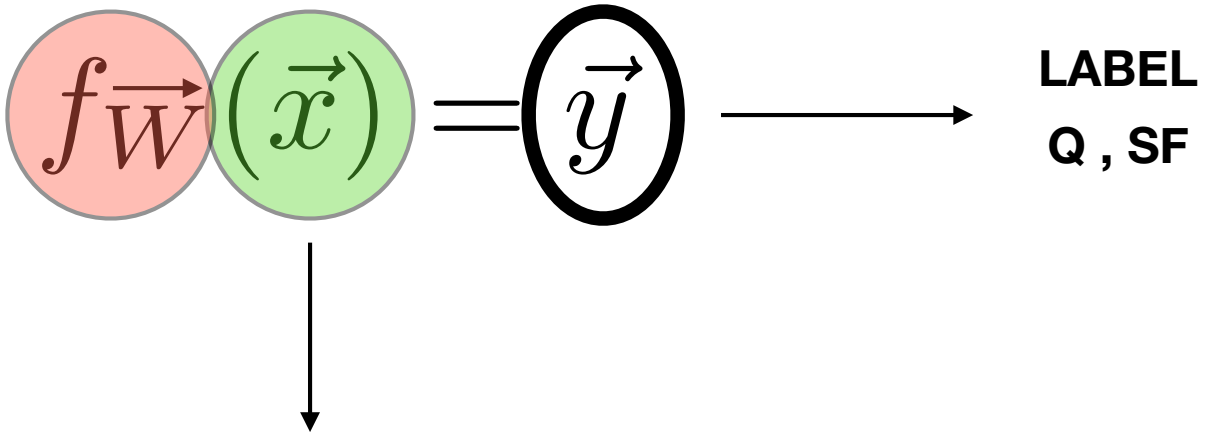
(Dielemann@Deepmind)

FEEDING INDIVIDUAL RESOLUTION ELEMENTS IS NOT VERY EFFICIENT SINCE IT LOOSES ALL INVARIANCE TO TRANSLATION



SO?

DEEP LEARNING



LET THE MACHINE FIGURE THIS OUT (“unsupervised feature extraction”)

LET’S GO A STEP FORWARD INTO LOOSING CONTROL….