LEAP Summer Bootcamp
Week 2 - Day 1

Introduction to Machine Learning and Neural Networks
June 2nd, 2025

Aya Lahlou
Ph.D. Candidate
al4385@columbia.edu

Instructor

Aya Lahlou
Ph.D. Candidate, Graduate Research Assistant
Earth and Environmental Engineering
Columbia University
al4385@columbia.edu

Research Interests:

- ML applications
- Land Ecosystem Modeling
- Phenology

Today's Lecture

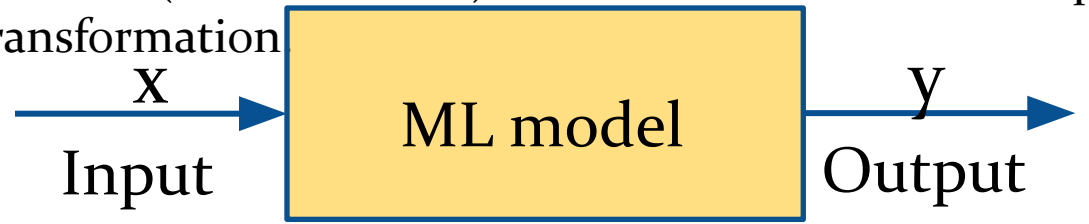
Agenda:

- ML is magic
- ML is not Magic
- Different types of ML models
- Examples of ML applications in Climate
- Intro to Neural Networks: Perceptron, Activation, Loss, Gradients
- NN in practice: Training, Regularization, L1&L2, Dropout, Early Stopping
- **Lab:** Neural Networks to predict the global temperature

What is Machine Learning?

- **The goal of classic machine learning is:**
  to extract useful information from a <u>set of data samples</u>.

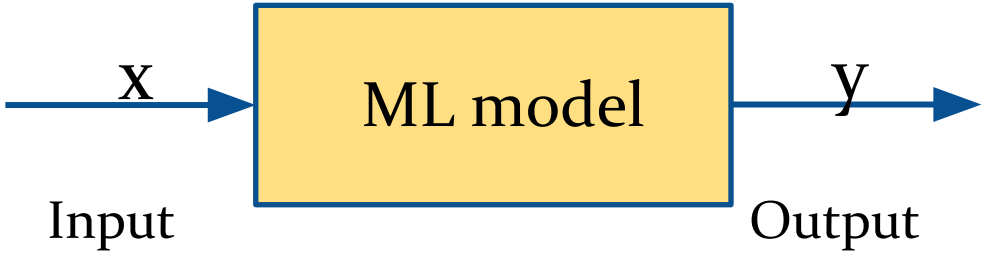
- **The result can be**
  1. A model (most common): "Model" here means an "input-output" transformation



  2. A set of rules, specific patterns, or other useful information.
     Examples:
     - Rule: "Temperatures are lower at night". Some connection between ENSO and MJO.
     - Pattern: Specific temperature *pattern* across the globe indicating climate change.

More on ML "models"

Model: "Model" here means an "input-output" transformation
ML task: Given data samples, **in form of input + output pairs**, learn a **ML model that can approximate the mapping from x to y**.



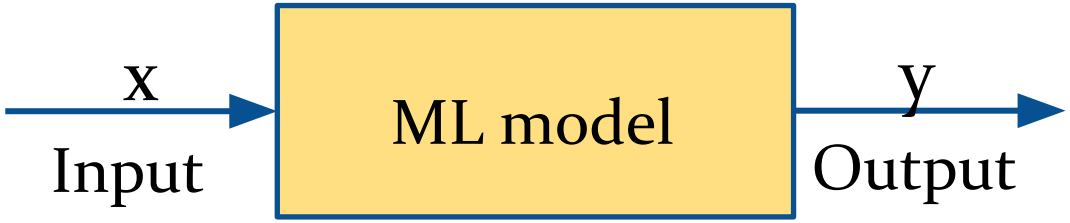
x and y can represent *any* type of data here:

- Discrete or continuous
- A scalar, a vector, an image, etc.
- Any combination of the above.

Sample ML method for this task:

Linear regression. *Yes – that's considered an ML method!*

ML vocabulary



If you have such input + output pairs, i.e. **you know what the output should be for your samples**, you call this a labeled data set, and output itself is called a label.

Learning an input-output model with labeled data, i.e. with *known output*, is called **Supervised learning**.

Two types of ML models for supervised learning:

A) Regression task - aka Prediction task: Output is **continuous**.
Examples: estimated precipitation, storm intensity, brightness temp.

B) Classification task – output is **discrete**, representing categories.
Examples: Is this a convective cloud or stratiform?
Is there a cyclone in this image?

Lots of potential for weather/climate

Potential applications include:

1) Prediction tasks

Especially for complex, non-linear, relationships with high-dimensional input space.
Ex.: Estimate cloud base height. Predict smoke, wind, humidity.

2) Classification tasks:

Which cloud type is this (stratiform, convective, etc)?

3) Object detection:

Is there a hurricane, atmospheric river, etc. in the image? If so, where is it? How big is it? How intense is it (classification)?

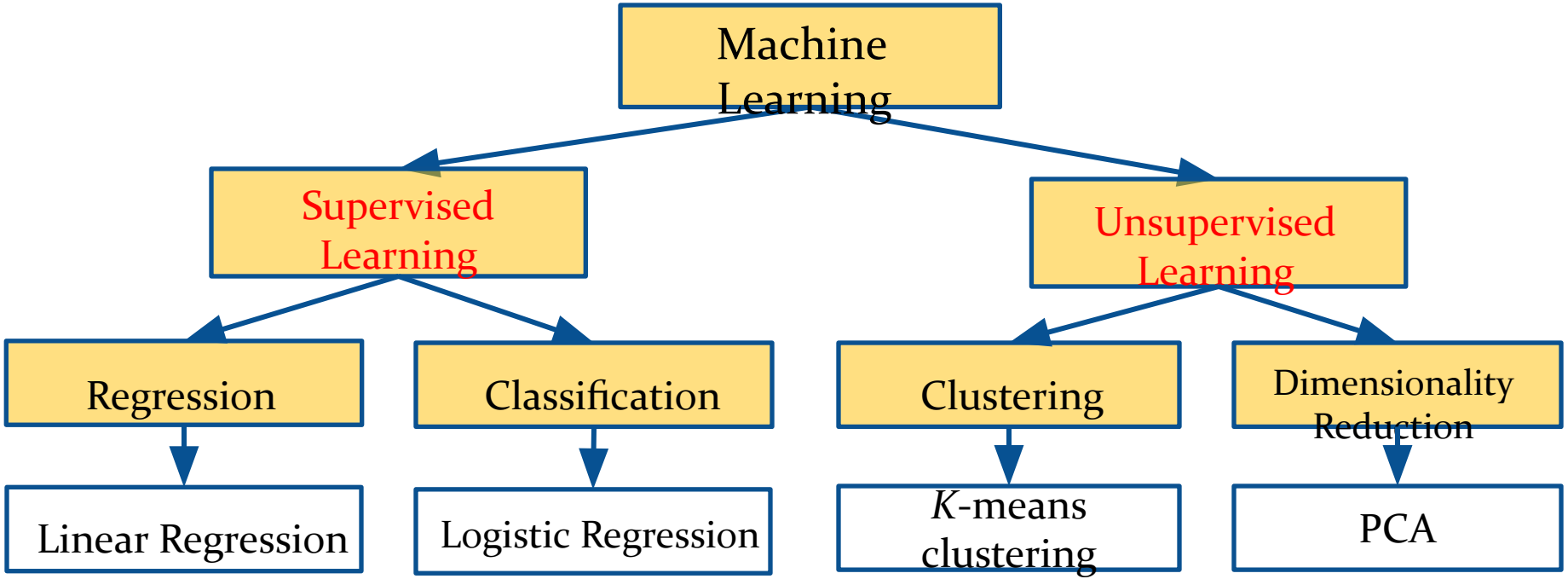
4) Anomaly detection: Is there anything "unusual" happening?

Ex.: Severe weather, heat wave, cold snap, climate change, bad data

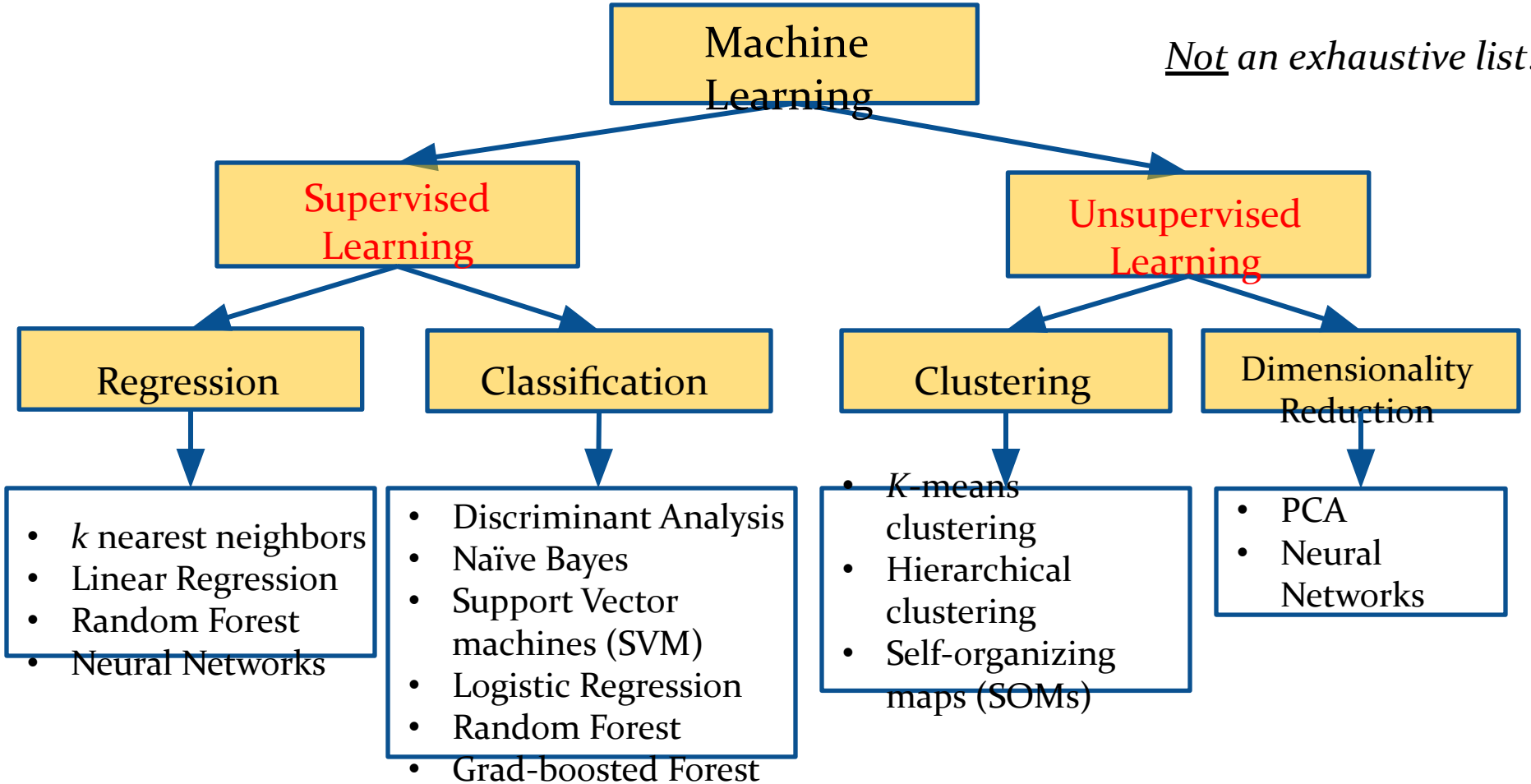
5) Subgrid-scale parametrization.

Speed up calculation and/or improve accuracy.
Ex.: Speed up radiative transfer parametrization for NWP.

Some simple ML methods you might already know



Larger selection of ML methods



ML is <u>not</u> magic!

Some costs and dangers of using ML.

“Clever Hans” Strategies

Clever Hans: German horse in 1907 that was believed to know arithmetic.

- Even the owner thought it knew arithmetic.
- It would answer questions by tapping its hoof the right number of times.
- Turns out: it read subconscious cues of the person asking the question.
- Person showed tension until the final correct tap, then relaxed.
- **So Clever Hans gave the right answer – but for the wrong reason.**

“Clever Hans” Strategies in Machine Learning

Examples from the following paper
(also source of images on the following slides):

> Lapuschkin, Sebastian, et al. “Unmasking Clever Hans Predictors and Assessing What Machines Really Learn.” Nature Communications, vol. 10, no. 1, Mar. 2019, p. 1096, https://doi.org/10.1038/s41467-019-08987-4.

Considered Task:
- Object recognition.
- ML algorithm trained to detect many different objects in images.

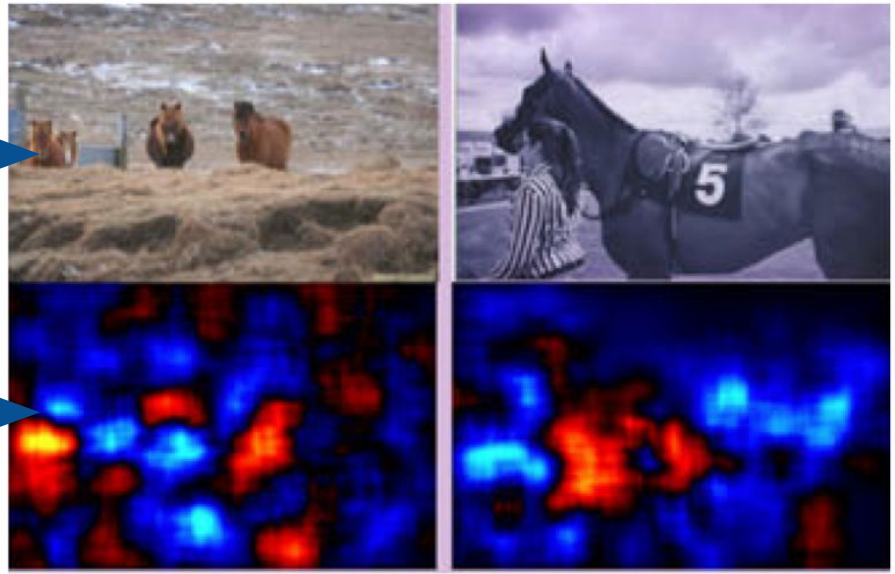
ML method used:
- Neural network (NN)

Specific task analyzed in paper:
- How does NN decide whether there is a *horse* in an image?
- Which strategies does it use to decide?

Method used for analyzing strategies:
- NN visualization technique (LRP) → constructs attribution maps.

Detecting horses – Strategy 1 of algorithm

Input Images



Attribution maps:
In red is where the NN is looking to decide whether there is a horse.

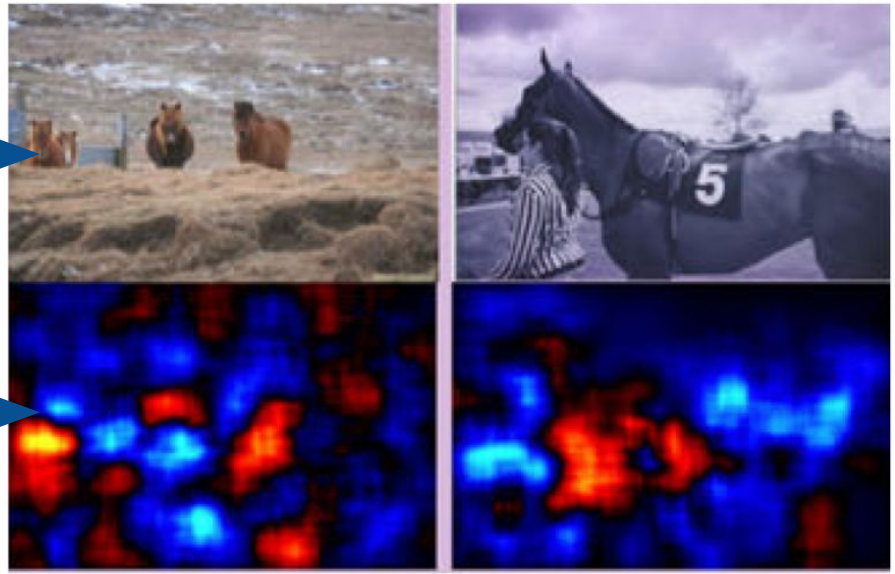
Red areas: increase confidence
Blue areas: decrease confidence
Black areas: not useful

Strategy 1: What does NN detect here?

Detecting horses – Strategy 1 of algorithm

Input Images

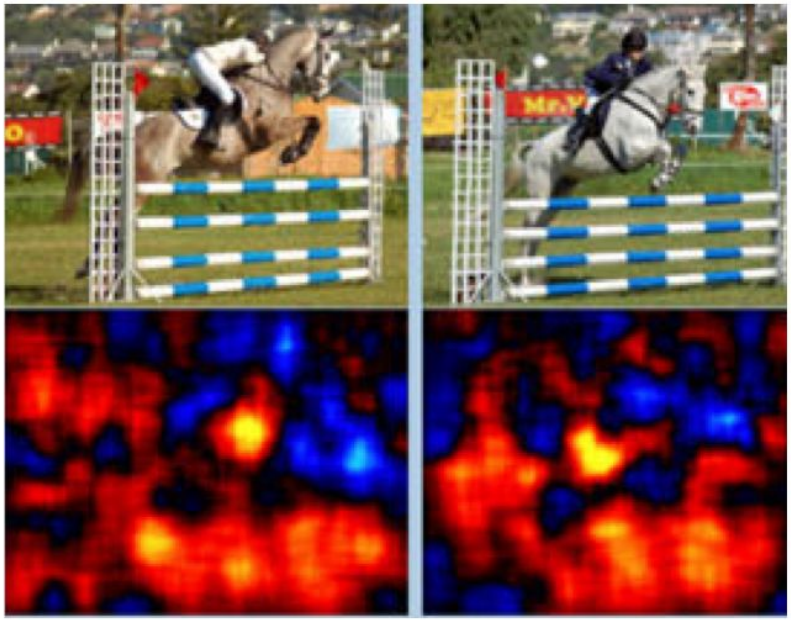
Attribution maps:
In red is where the NN is looking to decide whether there is a horse.



Strategy 1: What does NN detect here?
NN detects mainly parts of horses.
Excellent strategy!

Detecting horses – Strategy 2 of algorithm

Input Images



This is where the NN is looking to decide.

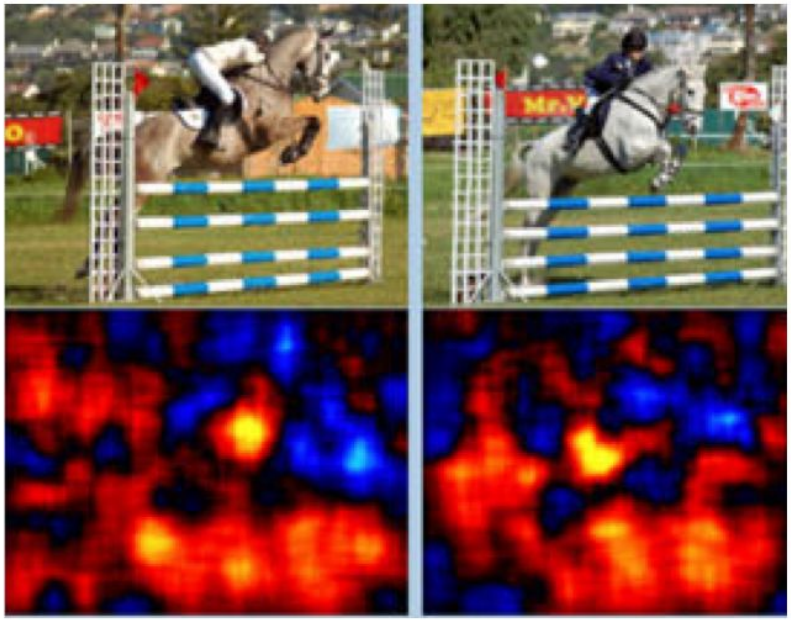
Strategy 2: What does NN detect here?

Red areas: increase confidence
Blue areas: decrease confidence
Black areas: not useful

Detecting horses – Strategy 2 of algorithm

Input Images

This is where the NN is looking to decide.



Strategy 2: What does NN detect here?

NN detects the poles – indicative of horses in provided samples.

Faulty reasoning: What if there is pole, but no horse?

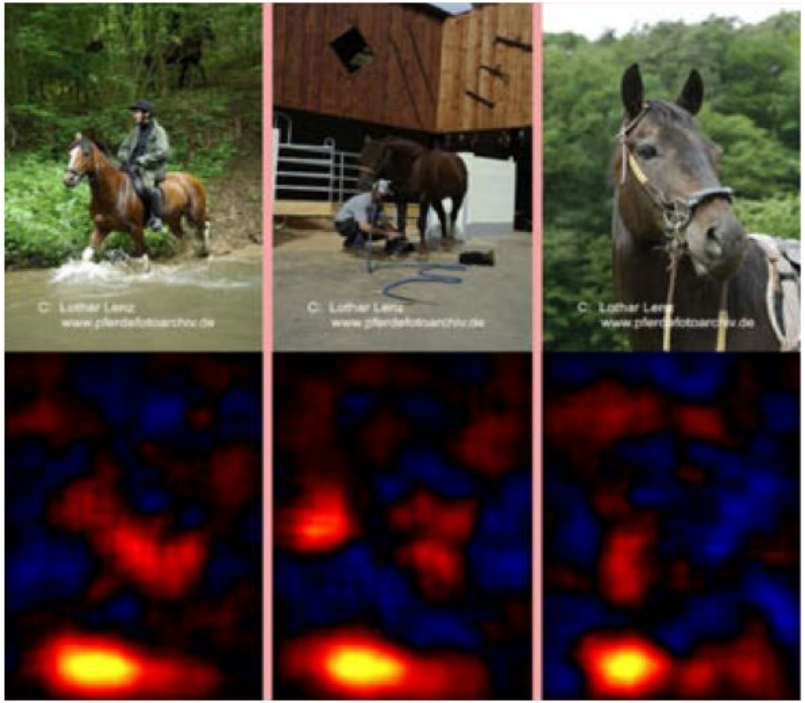
Can lead to false positives (false alarms)!

Detecting horses – Strategy 3 of algorithm

Strategy 3: What does the NN detect?

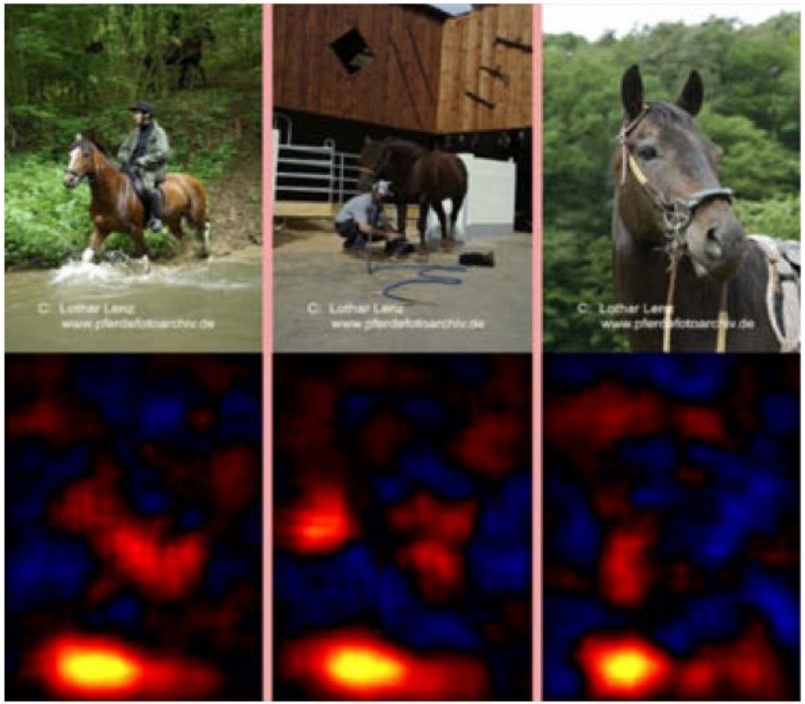
Detecting horses – Strategy 3 of algorithm

Attribution maps as hint.



Strategy 3: What does the NN detect?

Detecting horses – Strategy 3 of algorithm



Strategy 3: What does the NN detect in these images?
Html tags on the bottom of the images.
Bad strategy – would not be there in real world.
Likely to lead to false negatives (= misses).

ML may learn Clever Hans strategies, too!

ML algorithm might *also* give the right result, but for the wrong reason!

- Don't blame the algorithm.
- Algorithm did exactly what it was supposed to do, namely, to ***discover and use most helpful correlations/patterns in the data to perform its task.***

So what happened here?

- Some correlations present in data were not representative of real world:
  1. An image can contain a pole without containing a horse.
  2. Real-world images come without html tags.
- These examples might seem silly, but similar situations can happen very easily.

Example in meteorology:

- Ex.: Large hail mainly reported in highly populated areas.
- Should we conclude from such observations that large hail only occurs in high population areas?
- Of course not! It just happens to look like that *in the data*, due to observation bias.
  → Incorrect correlation in the data. → Incorrect generalization.
- Such inadvertent correlations can happen very easily!
- Ex: Sensors may only be available in certain areas, at certain times, etc.

How to avoid such traps?

Prevention:

- **Using ML methods completely as black box is generally <u>not a good idea</u>.**
- Always use the simplest and most transparent model possible.
- Find ways to merge your expert knowledge to guide/constrain the ML method
  → Results in simpler + more transparent models.

Detection:

- Rigorous testing of ML behavior for different meteorological conditions.
- Apply ML interpretation methods to learn about algorithm's reasoning.

Collaborative task:

- Environmental scientists are essential to prevent such faulty reasoning, because they know much better which meteorological conditions to test for, etc.

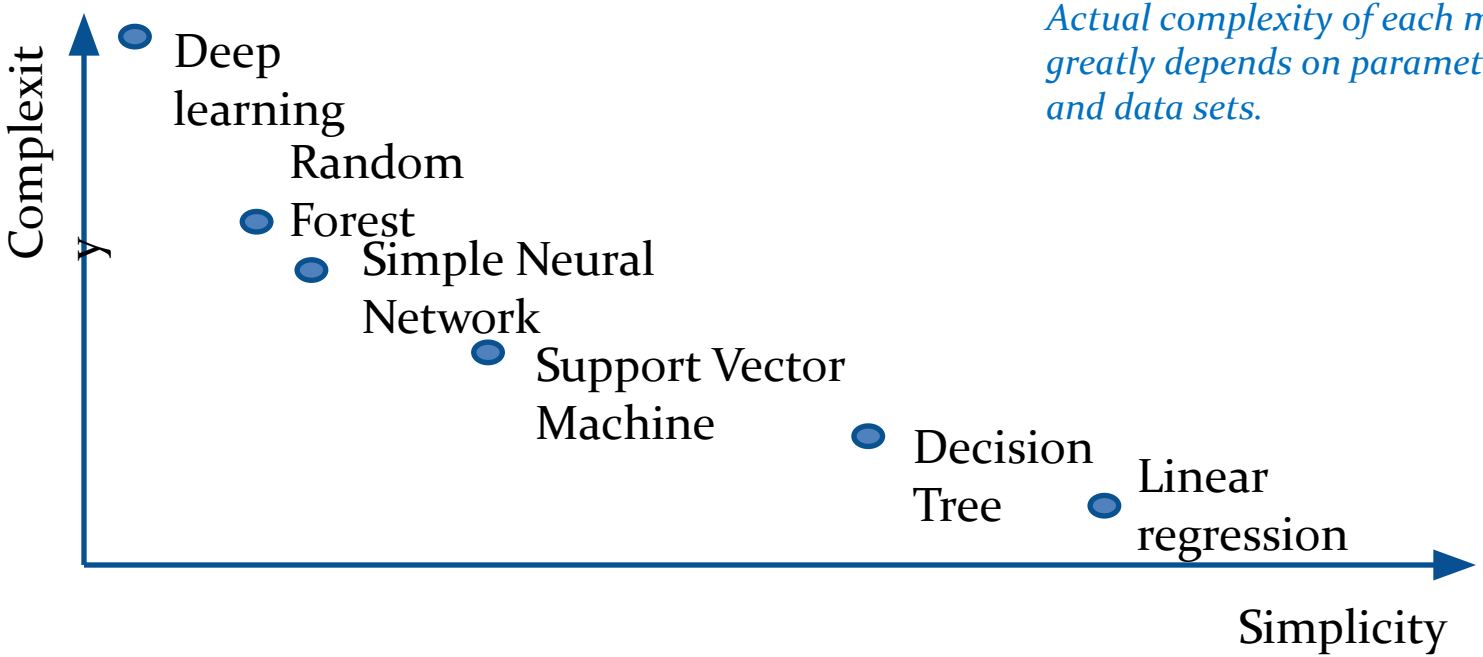
Importance of Environmental Scientist

- This is just one reason why it's **super important** to develop ML algorithms by **strong collaboration between ML expert and environmental scientist**.

- ML expert alone would *not* know:
  - Which variables to choose,
  - How to pre-process the data,
  - Which cost functions to choose,
  - Which tests to perform,
  - How to interpret the results,
  - etc, etc, etc.

- **Role of environmental scientist is crucial in all steps of ML algorithm development.**

Other costs of ML:
Trade-Offs for different methods



Graph is not to scale!
Actual complexity of each method greatly depends on parameter choices and data sets.

Trade-Offs for different methods

Higher Accuracy

Can model nonlinear + complex patterns

May need many (labeled) samples

Faulty reasoning possible

High comp. cost for learning model

Often many parameters to choose

Few parameters to choose

Simple

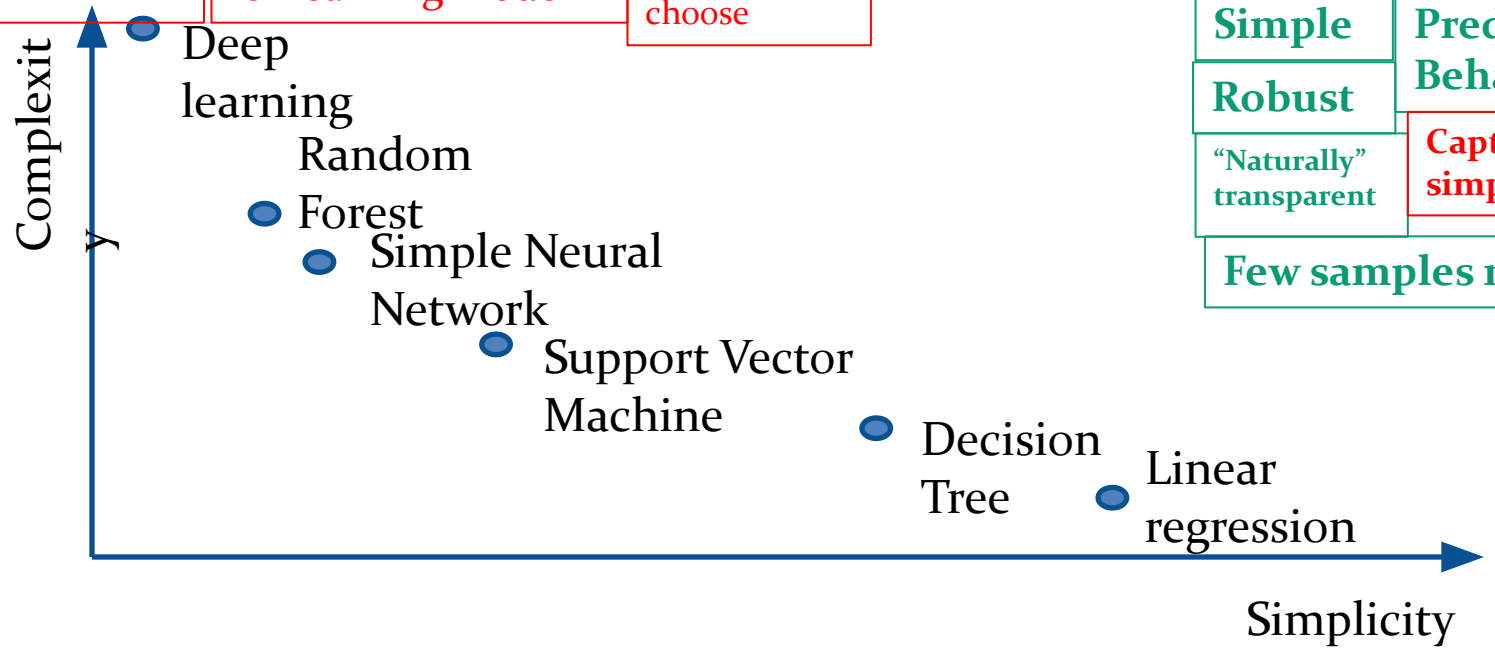
Predictable Behavior

Robust

"Naturally" transparent

Captures only simple patterns

Few samples needed



Questions to ask before using ML

1) **Why do I want to use ML for my application?**
   Speed? Accuracy? Because everyone else does it?

2) Formulating my ML task:
   Can I break up my task into subtasks, and only use ML for some sub-tasks?
   Can I just "**go the last mile with ML**"?

3) **Do I already know the strategy the algorithm should use?**
   Could I solve the task manually as a human (e.g., often the case in feature detection)?
   Or is there another algorithm used to date?
   If so, **how would I/it do it** – and which pieces of information (features, variables, etc.) are most important?
   Can I set up the ML method to use a similar strategy?

4) **Feature engineering**:
   Can I preprocess my data to maximize most important signals in the input/output? Can I make it easier for the algorithm to find the most important relationships by preprocessing the data? Making it easier □ can use simpler ML algorithm.

5) What trade-off do I want for my application?
   Ex.: how important is increased accuracy vs. lack of transparency?
   How much control do I want to have over my algorithm's behavior?
   How much freedom do I want to give to the algorithm?

6) **Available data:**
   How many samples do I have? How many are labeled? Do I trust the labels 100%?

Most common ML Methods in Weather & Climate

If we had to pick just a few ML methods

1) Linear/logistic regression – simple, powerful method.

2) Tree ensemble methods

2a) Random forest ("forest" = ensemble of decision trees)

- Few parameters to choose.
- Moderate complexity.
- Nice method, that is fairly accurate and can model non-linear relationships surprisingly well.
- For imagery: Great if you can process image information *pixel by pixel (one pixel at a time)*.

2b) Gradient-boosted forest (also an ensemble of decision trees)

- Similar to random forest
- Advantage: usually predicts better than random forest

If we had to pick just a few ML methods

3) Neural networks

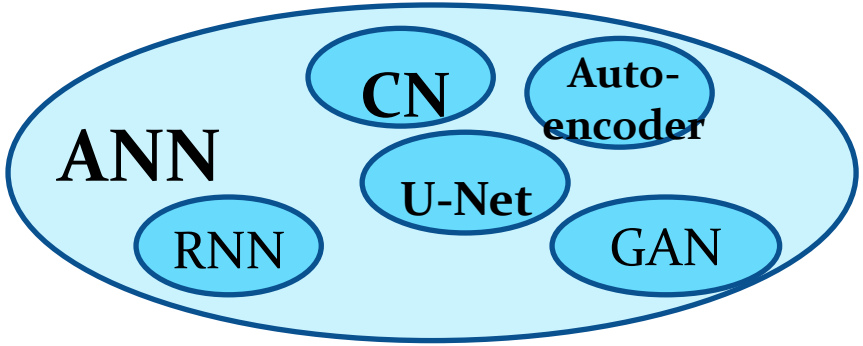
3a) Neural networks – simple architecture:

- For imagery: Great alternative if using image information only *pixel by pixel (one pixel at a time).*

3b) Neural networks – convolutional neural networks (CNNs)

- Best choice to discover/use **spatial patterns, spatial context, etc. in images.**

More AI Vocabulary



ANN/NN = Artificial Neural Network

- Very versatile and powerful ML method, can be seen as universal function generator.

DL = Deep Learning

- Any ANN with **many layers**, i.e. of very high complexity.

CNN = Convolutional Neural Network

- Type of ANN – specializes in learning ***spatial*** patterns/context.

RNN = Recurrent Neural Network

- Special time of ANN – specializes in learning ***temporal*** patterns/connections.

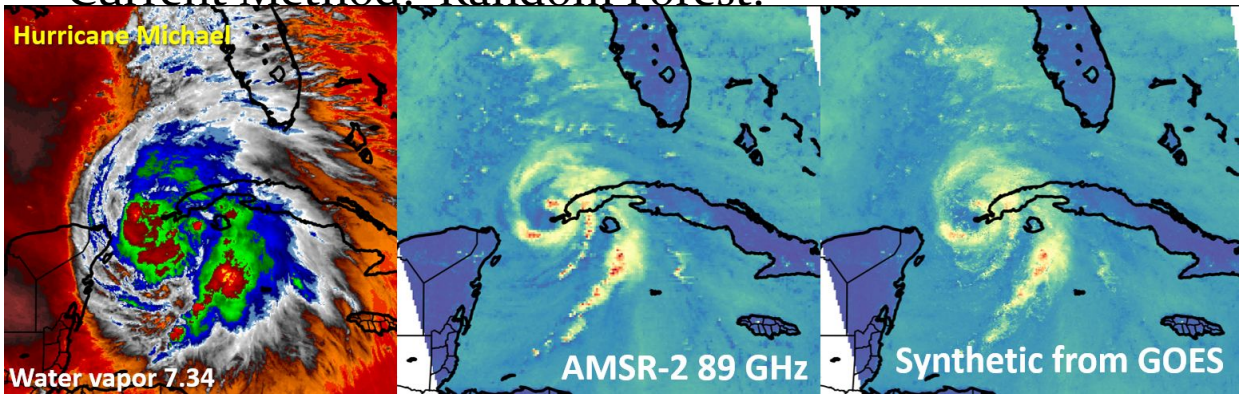
Other ANN types: **GAN**, **LSTM**, others.

Sample Application of ML at CIRA/NOAA

(not an exhaustive list)

Application 1

- **Chris Slocum (NOAA) and John Knaff (NOAA)**
- **Creating synthetic microwave imagery** using GOES-R baseline products for improved hurricane monitoring and rainfall estimation
- Current Method: Random Forest.



Input:
GOES-R imagery

Correct output:
True MW imagery

ML output:
Synthetic MW imagery

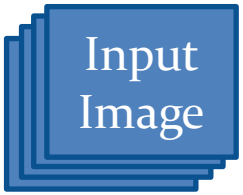
Chris Slocum

John Knaff

Application 2

- **Kyle Hilburn (CIRA) and Imme Ebert-Uphoff (CIRA)**
- **Generating synthetic radar images from GOES channels**.
- Method: Convolutional Neural Networks (CNNs).

Input:

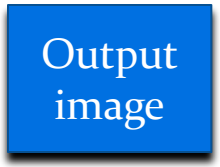


4 GOES channels

N

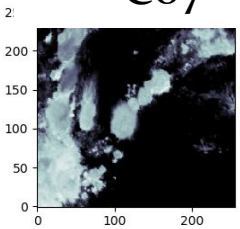
N

Output:

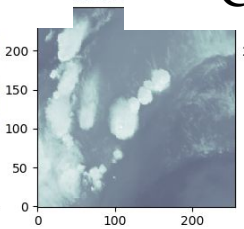


MRMS (radar) estimate

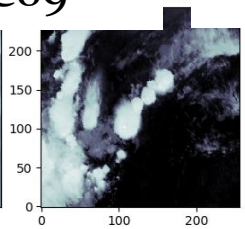
C07



C09

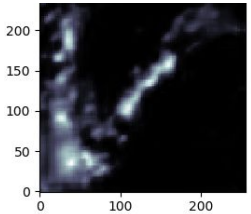


C13

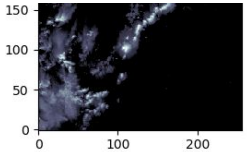


GLM (lightning)

MRMS - estimate



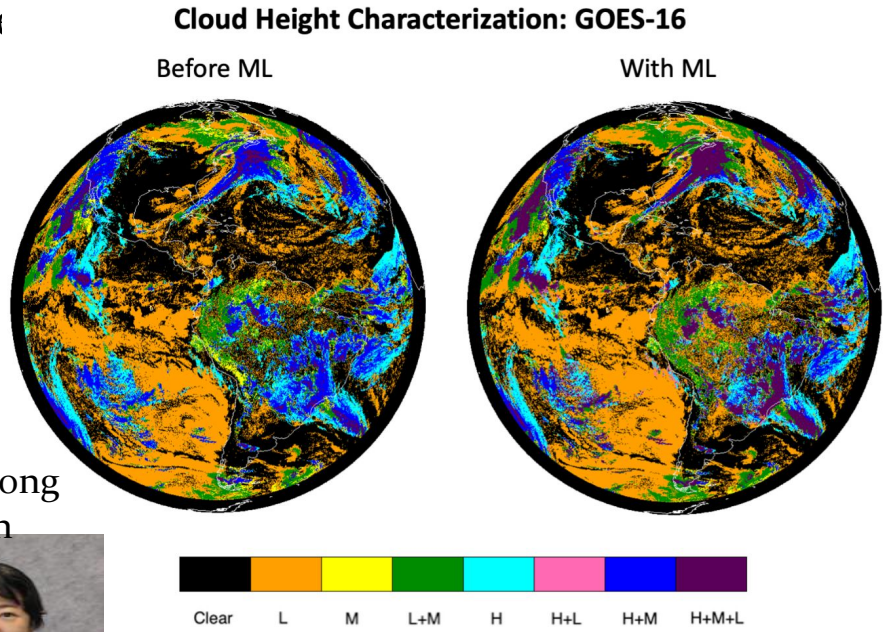
MRMS - observed



Motivation: GOES imagery is available in all of CONUS, but MRMS is not.

Application 3

- **John Haynes and Yoo-Jeong Noh (CIRA)**
- Detection of “hidden” low clouds from GOES ABI / JPSS VIIRS using training from spaceborne CloudSat radar and CALIPSO lidar
- Current M

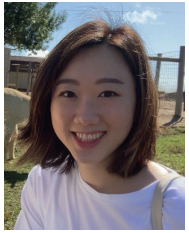


John Haynes

Yoo-Jeong Noh

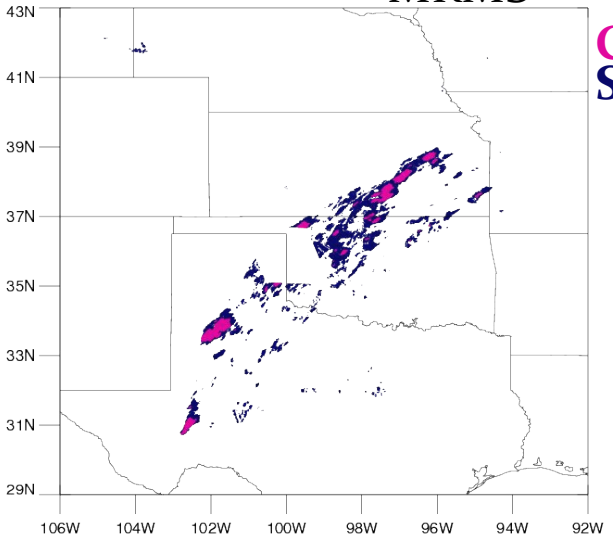
Application 4

Yoonjin Lee



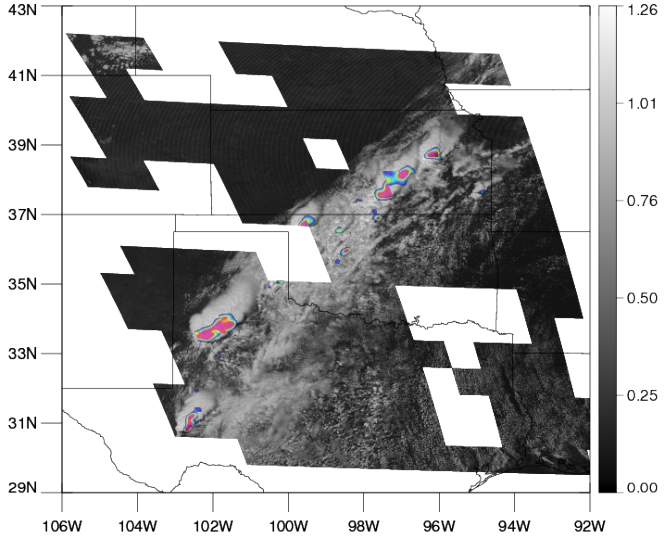
- **Yoonjin Lee (CSU), Chris Kummerow and Imme Ebert-Uphoff.**
- **Detecting convective regions from GOES-16**.
- Method: Convolutional Neural Networks (Encoder-decoder model).
- Input: Five temporal data at channel 2 (reflectance) and 14 (brightness temperature)
- Output: Multi-Radar/Multi-Sensor (MRMS)

MRMS



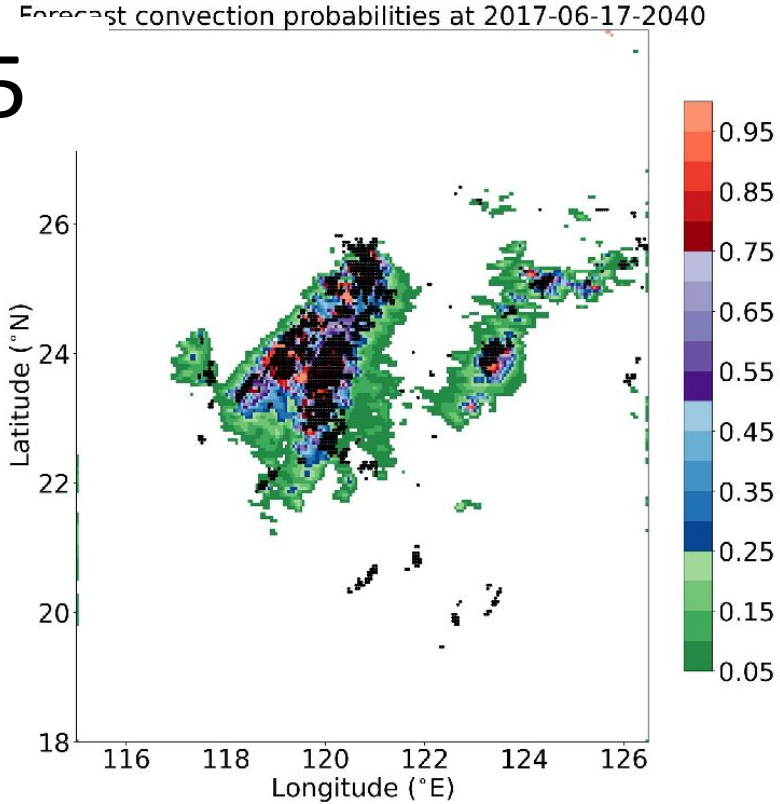
Convective
Stratiform

Predicted convective regions



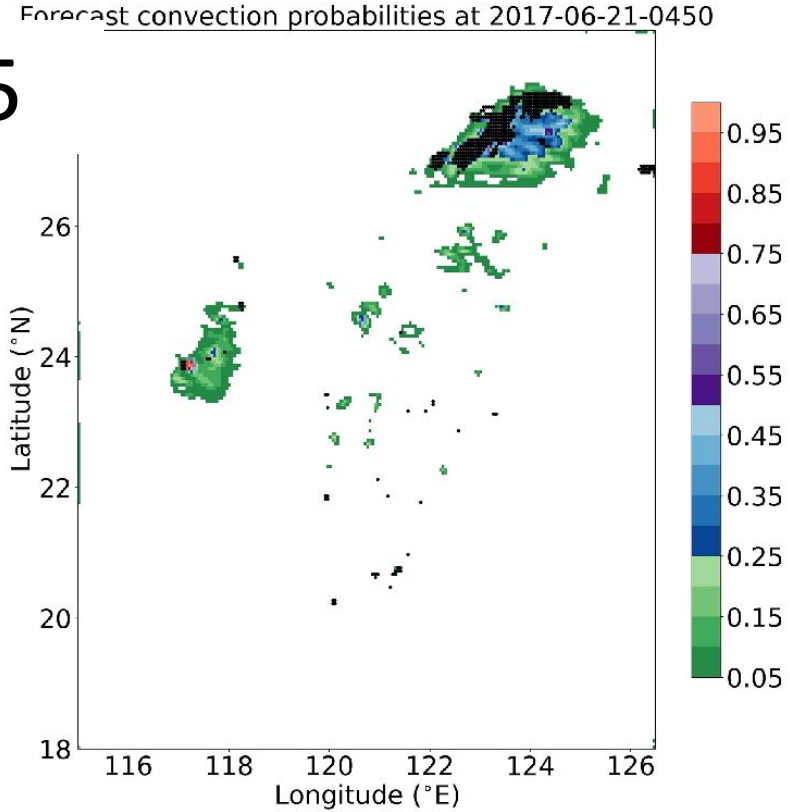
Application 5

- Ryan Lagerquist, Jebb Stewart (NOAA GSL), Christina Kumler (NOAA/CIRES)
- **Forecasting convective initiation and decay from Himawari 8 satellite data**
- **Current method: U-nets** (a type of convolutional neural network)
- Right: forecast convection probabilities (colours) and actual convection mask (black dots)



Application 5

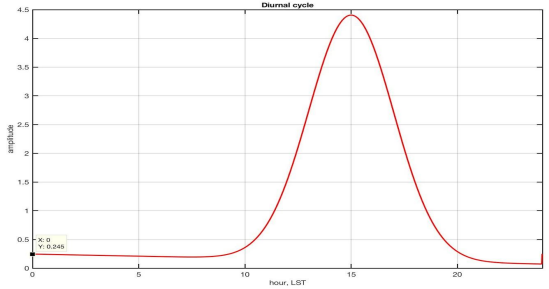
- Ryan Lagerquist, Jebb Stewart (NOAA GSL), Christina Kumler (NOAA/CIRES)
- **Forecasting convective initiation and decay from Himawari 8 satellite data**
- **Current method: U-nets** (a type of convolutional neural network)
- Right: forecast convection probabilities (colours) and actual convection mask (black dots)



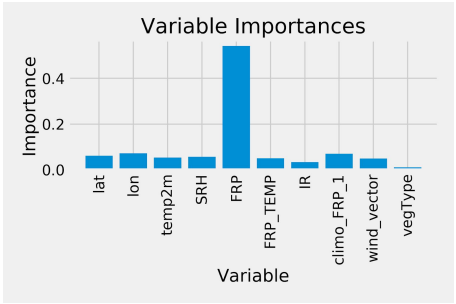
Application 6

Christina Kumler (Bonfanti)

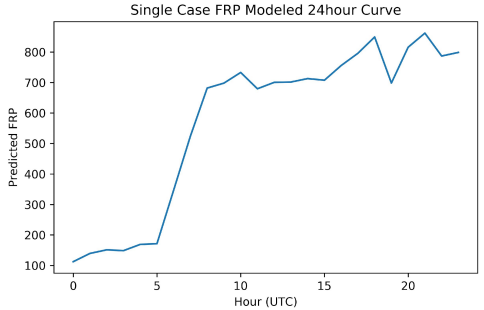
- Christina Kumler (NOAA GSL/CIRES), Ravan Ahmadov (NOAA/CIRES), and Jebb Stewart (NOAA GSL)
- **Deriving Fire Radiative Power (FRP) with Satellite and Weather Model Input Data**
- Current method: Random Forest (RF)



Old representation of all wildfire FRP values increase/decrease magnitude without meteorological



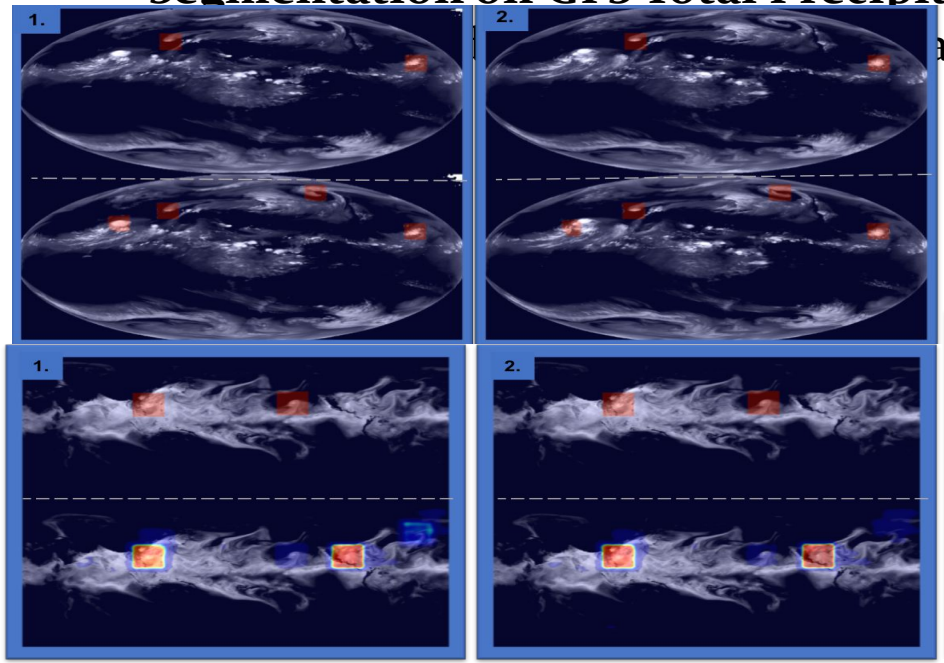
RF variable importance of the inputs to predict FRP one hour in future



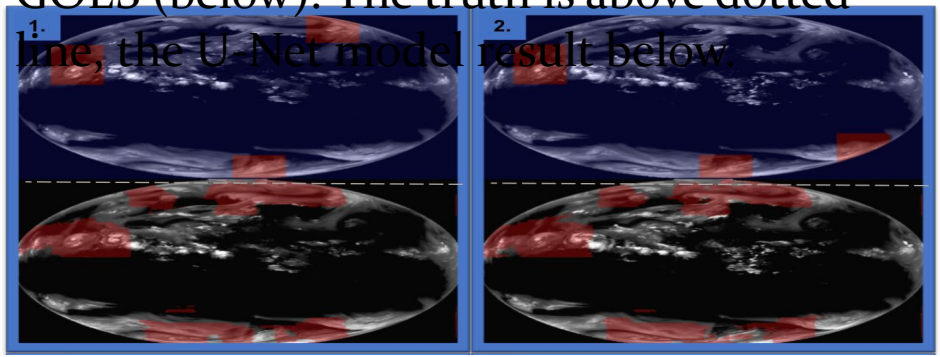
RF created 24 hour FRP "forecast" curve using meteorological data unique to individual FRP point

Application 7

- Christina Kumler (NOAA GSL /CIRES), Jebb Stewart (NOAA GSL), David Hall (NVIDA), and Mark Govett (NOAA GSL)
- **Tropical and Extratropical Cyclone Regions of Interest (ROI) Segmentation on GFS Total Precipitable Water Or GOES Water Vapor** [illegible] a U-Net

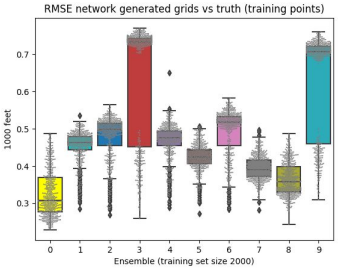


U-net results for two consecutive time steps trained on hurricanes from IBTrACS on GOES (left) and GFS IBRTrACS (lower-left), and all cyclones from Heuristic Model on GOES (below). The truth is above dotted line, the U-Net model result below.



Application 8

- Geary Layne (NOAA GSL /CIRES)
- **Tropical and Extratropical Cyclone Regions of Interest (ROI) Segmentation**
- Current method: Conditio ersarial Network (cGAN) and NVIDA's Image Painting



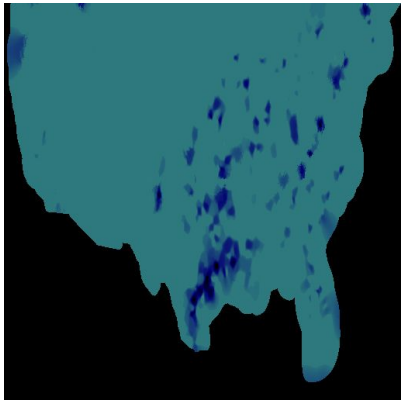
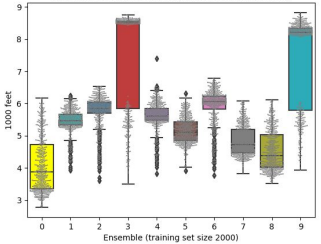
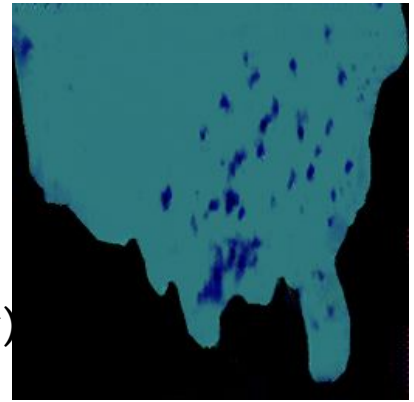


Image from Truth Set

Output evaluated only at location where truth input was provided (above) and over whole grid (below)





Results with training set 2000 samples

Last two applications:

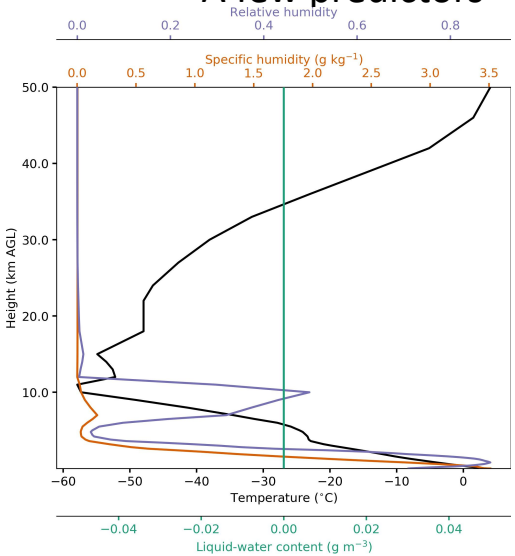
Estimating vertical profiles

Application 9

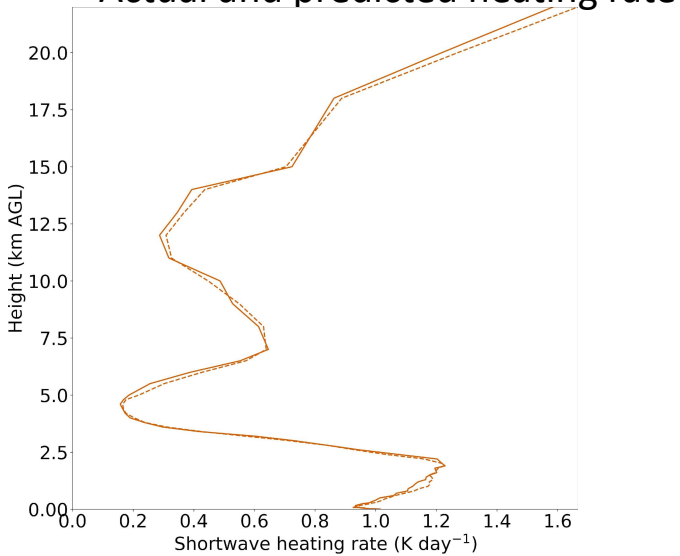
- Ryan Lagerquist, Dave Turner (NOAA GSL), Jebb Stewart, Imme Ebert-Uphoff, Christina Kumler

- **Using ML to accelerate radiative-transfer scheme in dynamical weather and climate models**
- Current method: U-nets

A few predictors



Actual and predicted heating rate

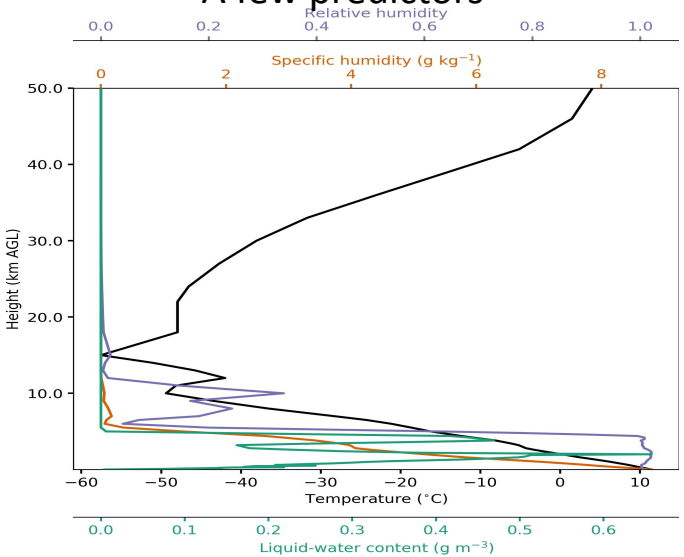


Application 9

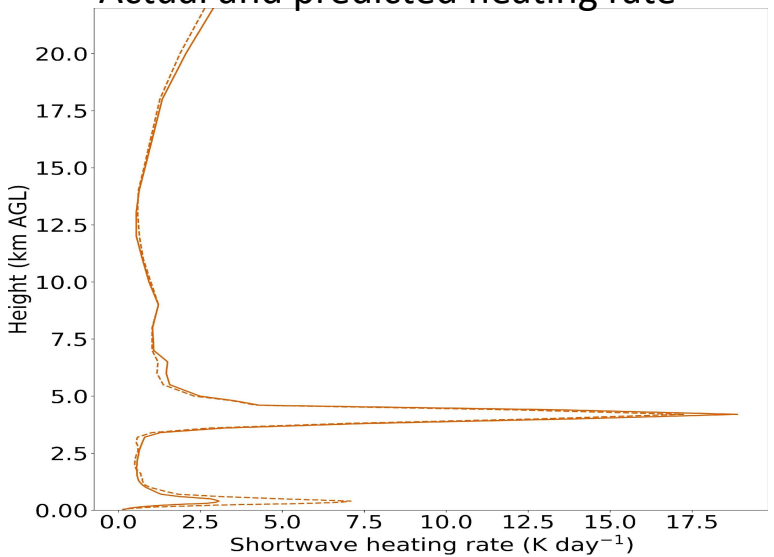
- Ryan Lagerquist, Dave Turner (NOAA GSL), Jebb Stewart, Imme Ebert-Uphoff, Christina Kumler

- **Using ML to accelerate radiative-transfer scheme in dynamical weather and climate models**
- Current method: U-nets

A few predictors



Actual and predicted heating rate

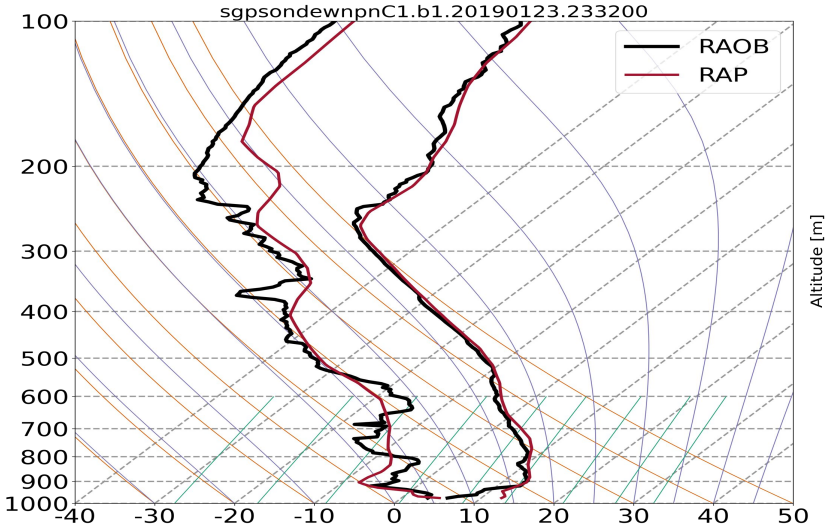


Application 10

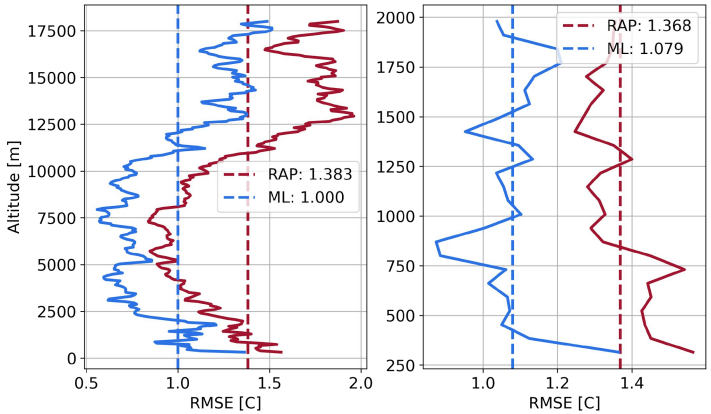
- Jason Stock, Jack Dostalek, Louie Grasso, Imme Ebert-Uphoff

- **Using machine learning to improve vertical profiles of temperature and moisture for severe weather nowcasting**
- Current method: CNN & Transfer Learning

What is being improved?



Preliminary improvements over the T profile.



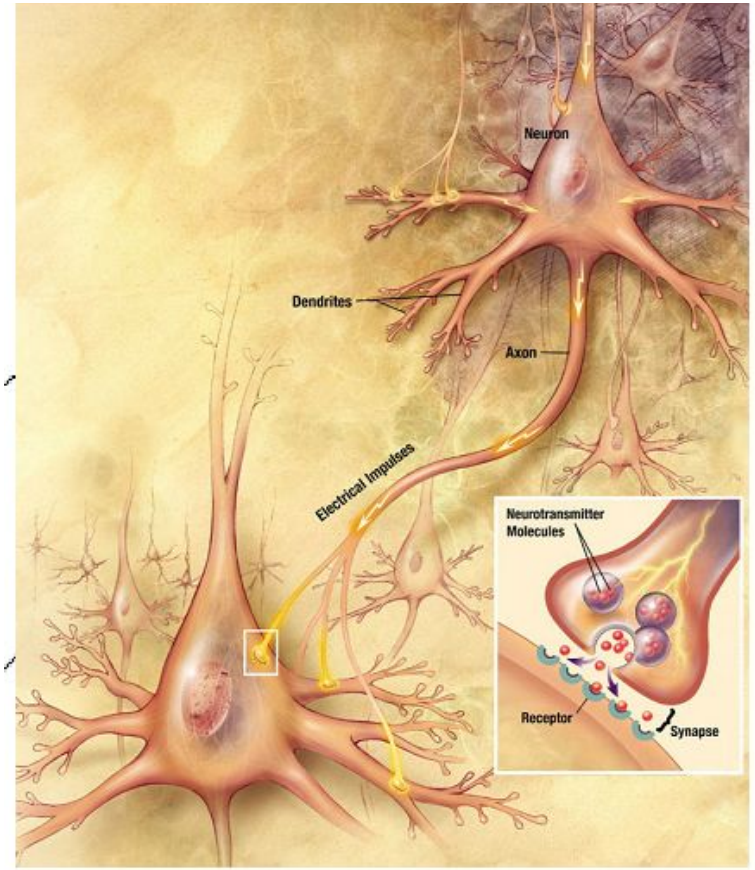
Intro to
Neural Networks

Neural networks

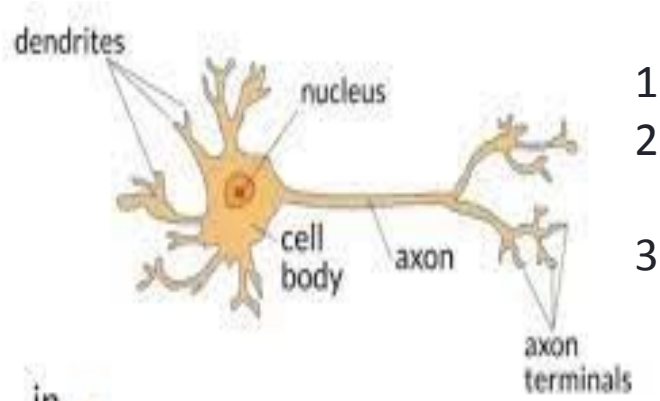
Brain function

(thought) occurs as the result of the firing of **neurons**

- Neurons connect to each other through **synapses**, which propagate **action potential** (electrical impulses) by releasing **neurotransmitters**
- Synapses can be **excitatory** (potential-increasing) or **inhibitory** (potential-decreasing), and have varying **activation thresholds**
- Learning occurs as a result of the synapses' **plasticity**: They exhibit long-term changes in connection strength
- There are about 10^{11} neurons and about 10^{14} synapses in the human brain!



Bio vs. AI



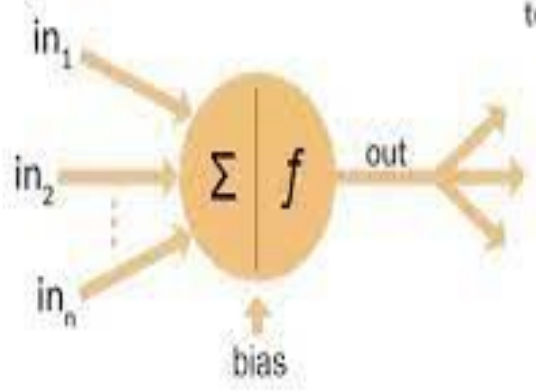


Image: Quora

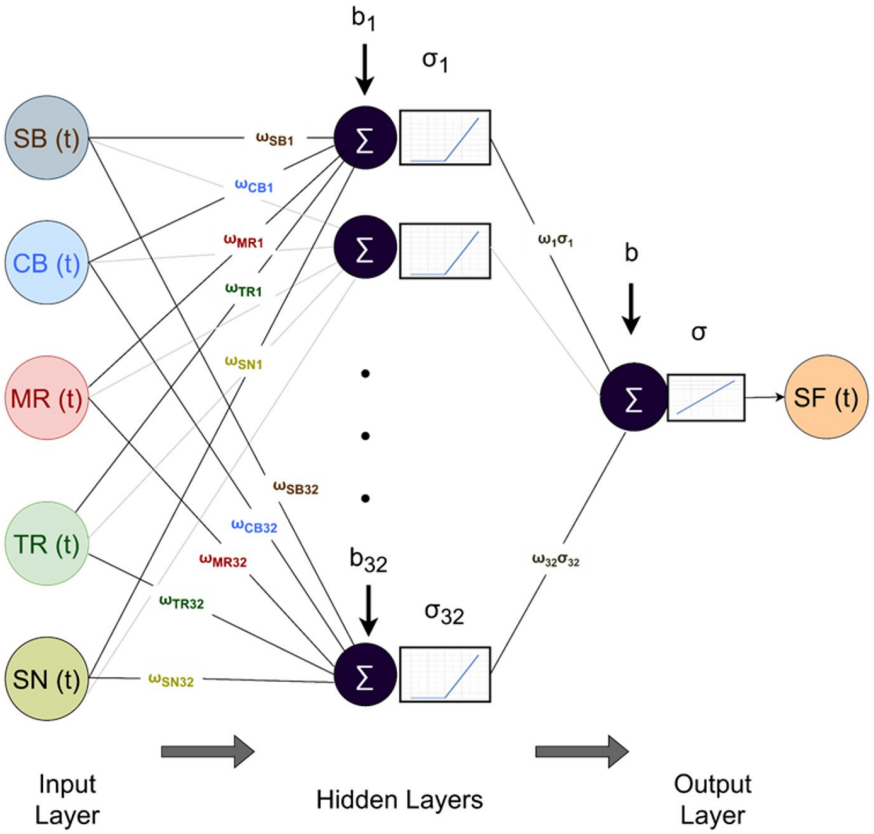
The basic functions of a neuron

1. Receive signals (information).
2. Interpret incoming signals (which pieces of info should be passed forward).
3. Communicate signals to target cells (other neurons or muscles or glands).

-Khan Academy

Similarities to AI NN:
Dendrites (input layer): Incoming information
Synapses (hidden layers): Neurons communicate to other neurons
Activation (summation of weights and activation function): which info gets propagated forward.
Axon (output layer): The final, processed output

An example of a feed forward network



For this model, at each time step, there are 32 neurons within the hidden layer, and each predictor feeds forward to all neurons. For each connection, via the Adam optimizer gradient descent method, weights, w, are initialized; moreover, for all neurons, a distinctive bias, b, is computed by random initialization. Then, for every neuron, i, there is a weighted sum calculation, as follows:

$$z_i = w_{SB,i} + w_{CB,i} + w_{MR,i} + w_{TR,i} + w_{SN,i} + b_i \quad (5)$$

The sum then enters into the ReLU activation function, σ. Activation functions aid in understanding nonlinear relationships. The ReLU was chosen, as it is known for its accuracy and is widely used in deep learning modeling (Dubey & Jain, 2019). The next step in the FFN is the forward movement of information from the last hidden layer to the output layer. Similar to the last weighted sum calculation, the weighted sum at the output neuron is computed:

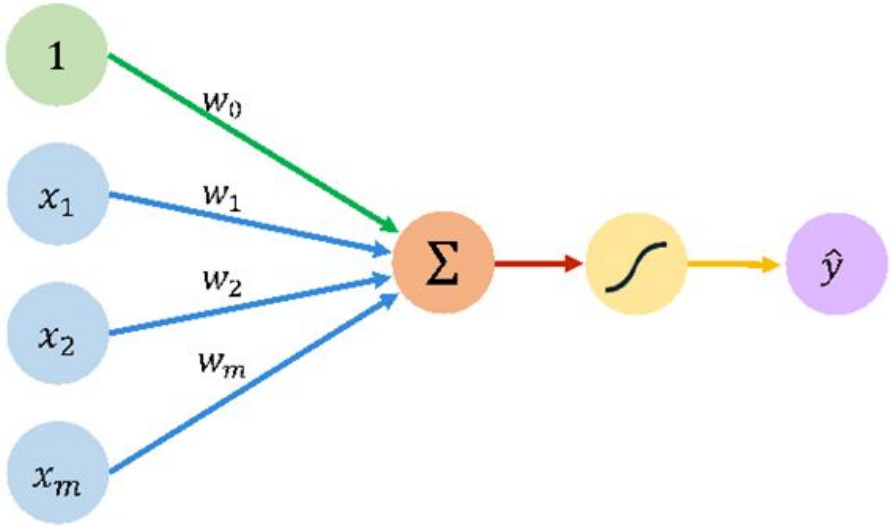
$$z_{output} = \sum \sigma_i w_i + b \quad (6)$$

From the output layer, the data enters a linear function and produces the predicted counts of SF. The model was run with a learning rate of 0.001, batch size of 32 and 100 epochs.

Neural networks: Let's break it down

Perceptron: a single-layer neural link with four main parameters: input values, weights and Bias, net sum, and an activation function

Multi-layer perceptron: more hidden layers

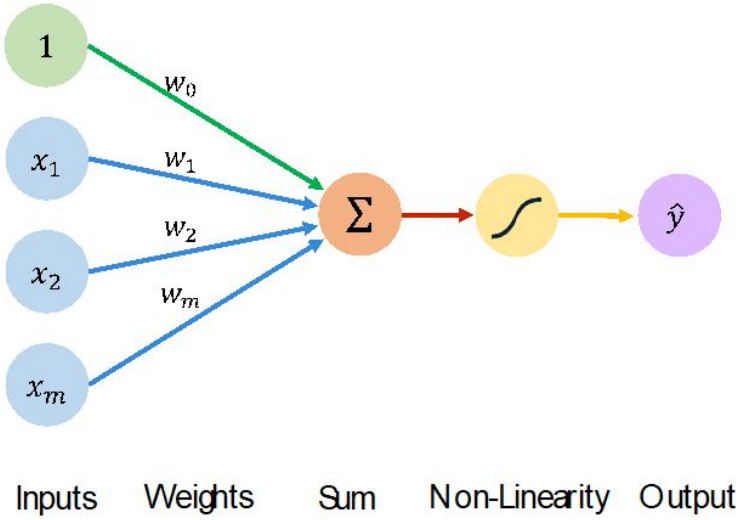


The Perceptron: Forward Propagation

1. All the inputs ***x*** are multiplied with their weights ***w***

2. Add all the multiplied values and call them Weighted Sum.

3. Apply that weighted sum to the correct Activation Function.

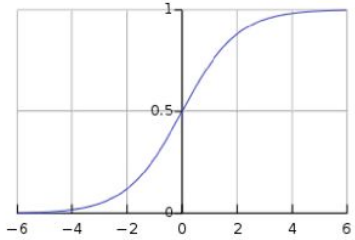


Activation Functions

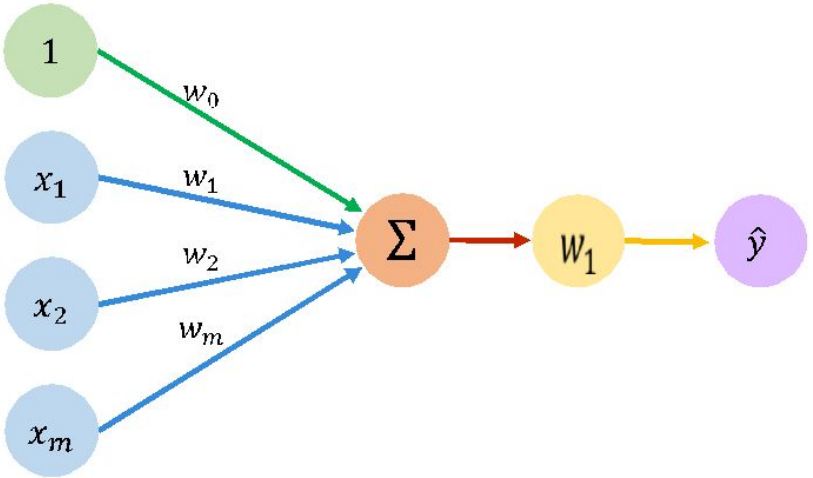
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

- Example: sigmoid function

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



The Perceptron: Forward Propagation

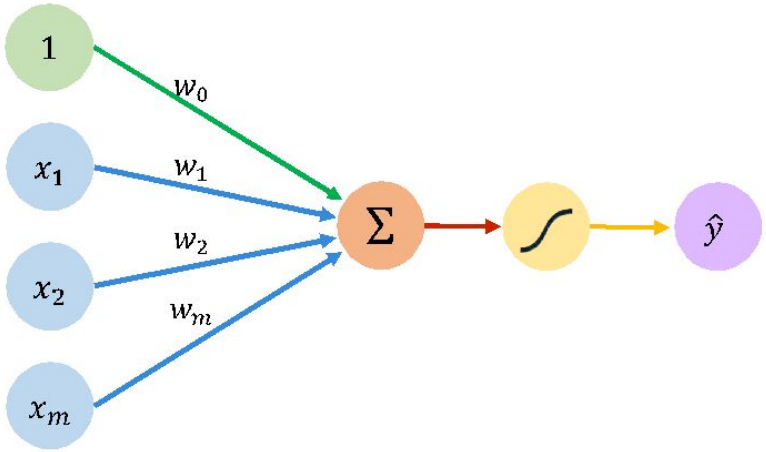


$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

The Perceptron: Forward Propagation

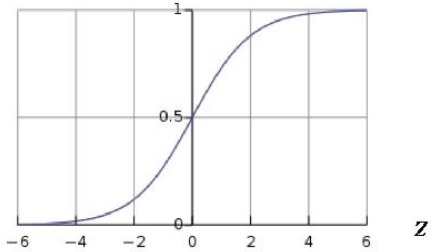


Activation Functions

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

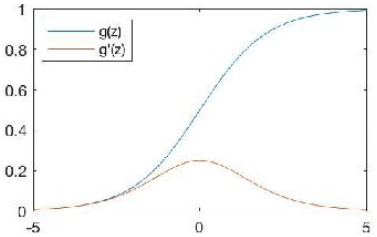
- Example: sigmoid function

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



Common Activation Functions

Sigmoid Function

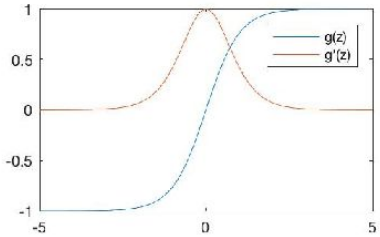


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

tf.nn.sigmoid(z)

Hyperbolic Tangent

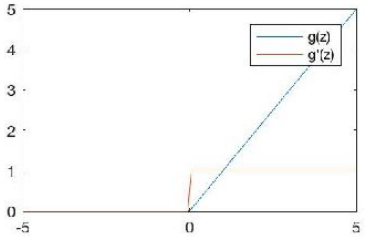


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

tf.nn.tanh(z)

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

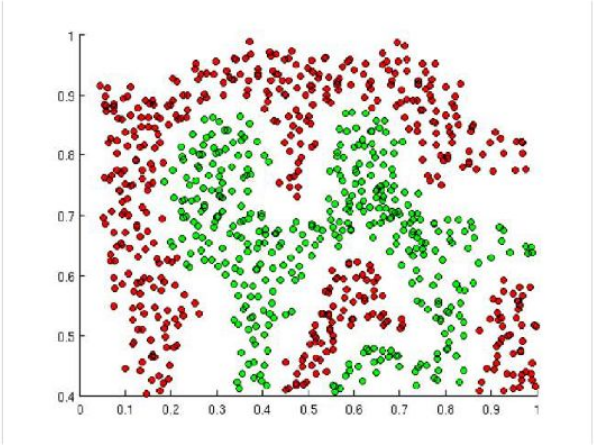
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

tf.nn.relu(z)

NOTE: All activation functions are non-linear

Importance of Activation Functions

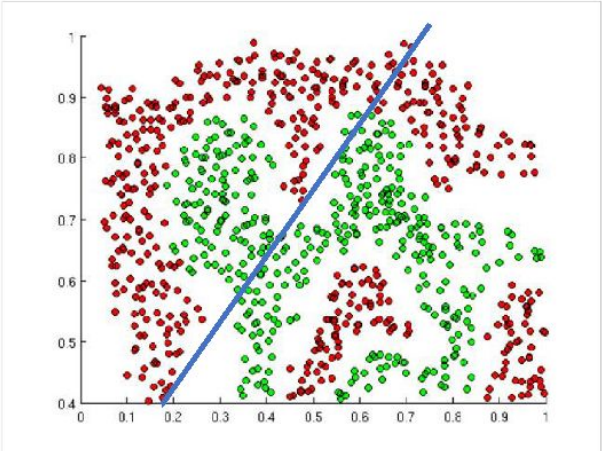
The purpose of activation functions is to introduce non-linearities into the network



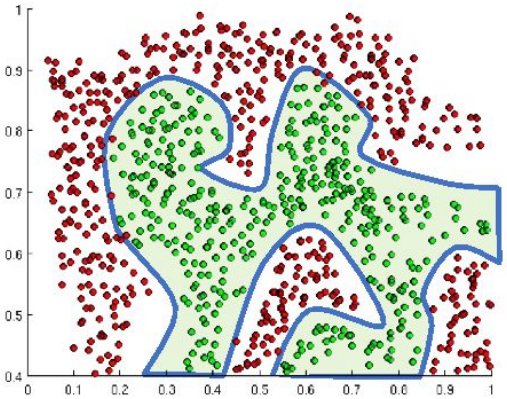
What if we wanted to build a Neural Network to distinguish green vs red points?

Importance of Activation Functions

The purpose of activation functions is to introduce non-linearities into the network

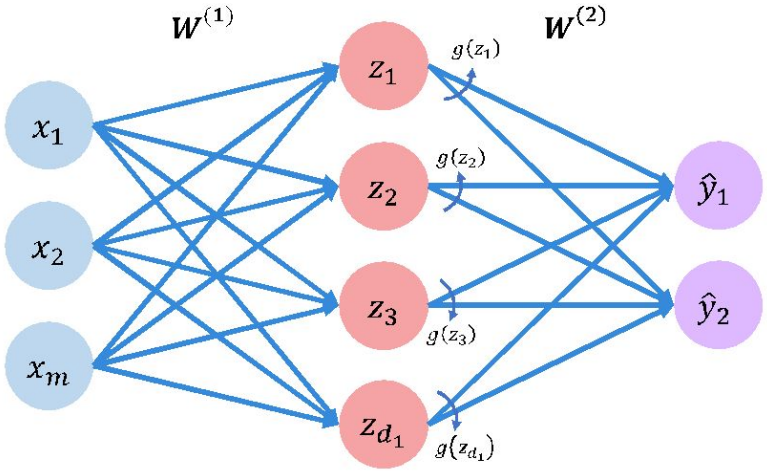


Linear Activation functions produce linear decisions no matter the network size



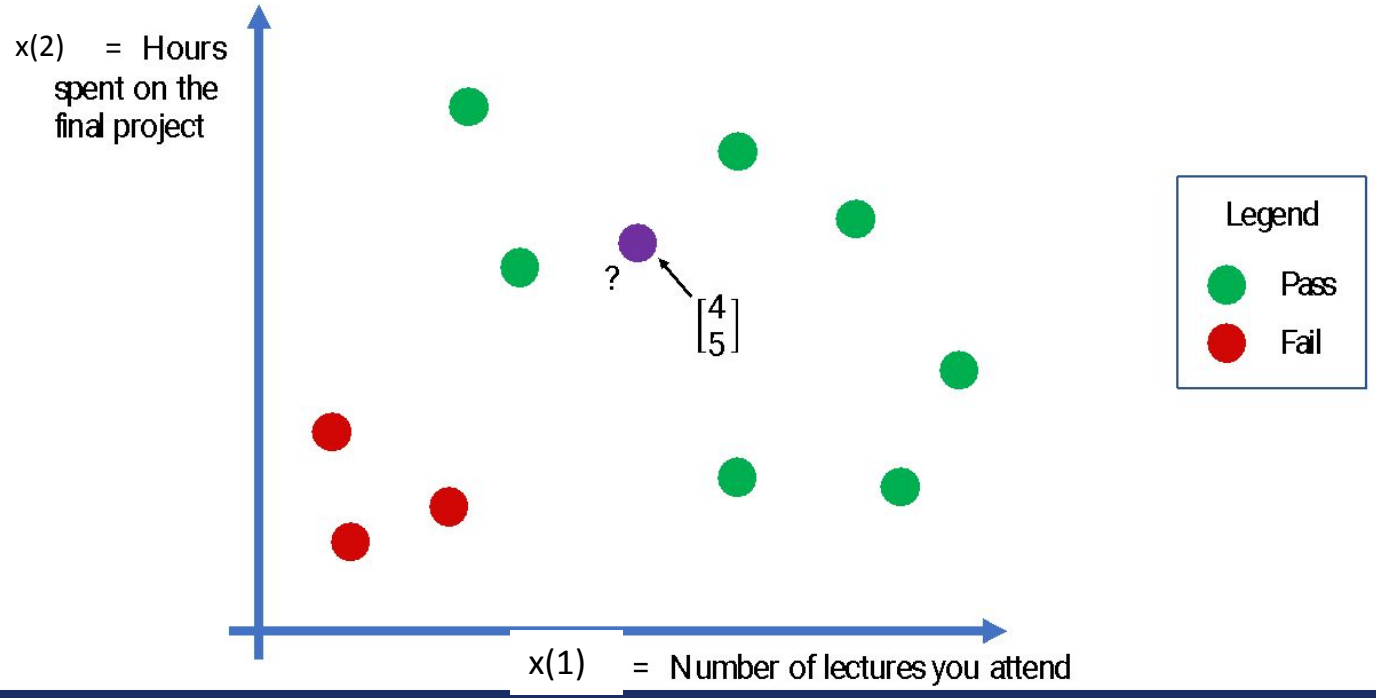
Non-linearities allow us to approximate arbitrarily complex functions

Single Layer Neural Network



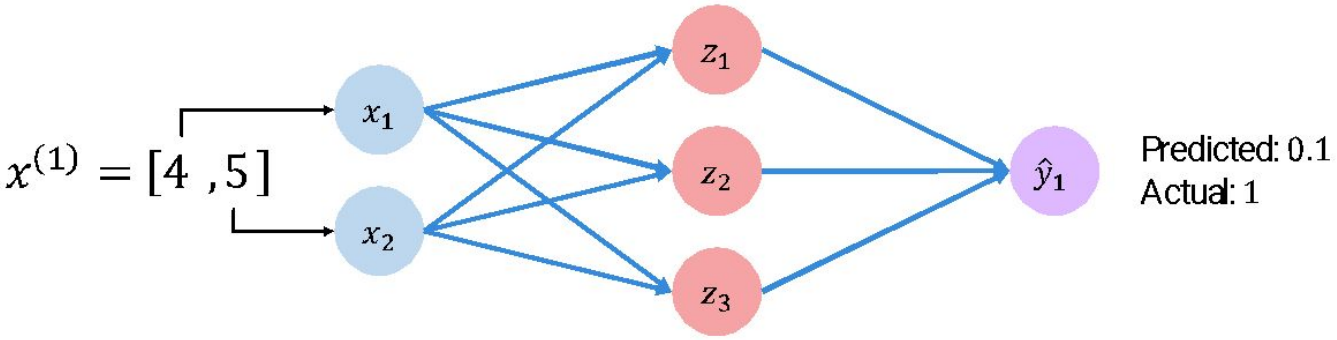
$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, w_{j,i}^{(2)}\right)$$

Example Problem: Will I pass this class?



Quantifying Loss

The loss of our network measures the cost incurred from incorrect predictions

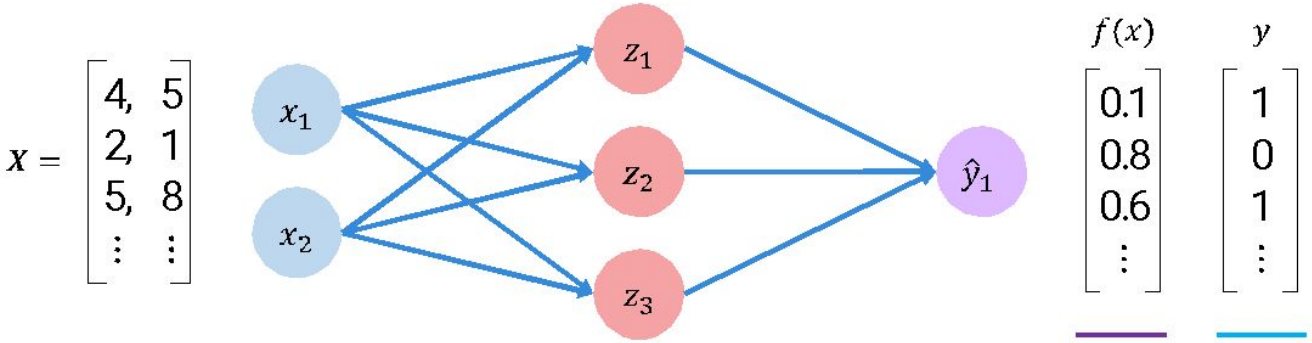


$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Empirical Loss

The empirical loss measures the total loss over our entire dataset



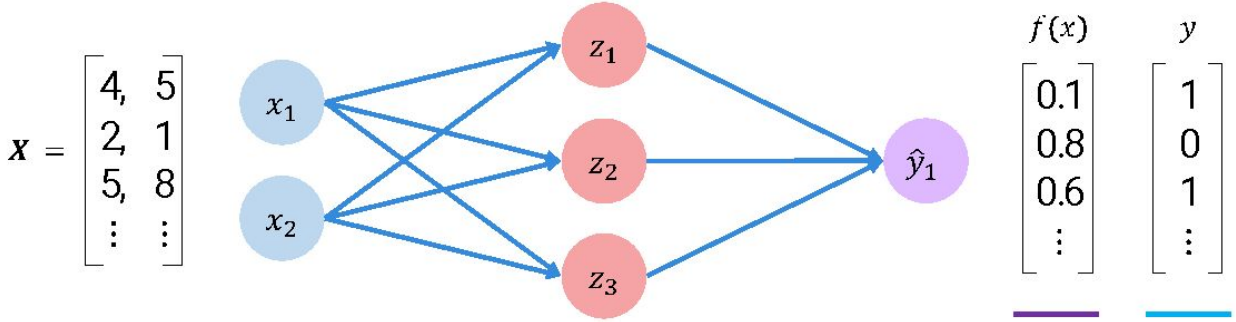
Also known as:
- Objective function
- Cost function
- Empirical Risk

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Binary Cross Entropy Loss

Cross entropy loss can be used with models that output a probability between 0 and 1



$$J(W) = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log\left(f\left(x^{(i)}; W\right)\right) + (1 - y^{(i)}) \log\left(1 - f\left(x^{(i)}; W\right)\right)$$

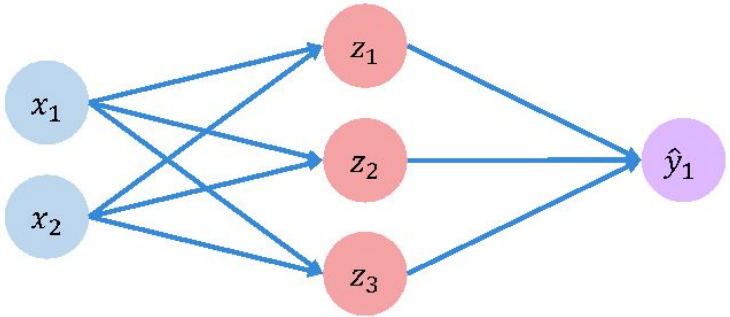
Actual Predicted Actual Predicted

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(model.y, model.pred) )
```

Mean Squared Error Loss

Mean squared error loss can be used with regression models that output continuous real numbers

$$X = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{bmatrix} f(x) \\ 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} \quad \begin{bmatrix} y \\ 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix}$$

Final Grades (percentage)

$$J(W) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; W\right)\right)^2$$

Actual Predicted

```
loss = tf.reduce_mean( tf.square(tf.subtract(model.y, model.pred) )
```

Neural networks: training

We want to find the network weights that achieve the lowest loss

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W})$$

Neural networks: training

Gradient descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$

```
weights = tf.random.normal(shape, stddev=sigma)
```

2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$

```
grads = tf.gradients(ys=loss, xs=weights)
```

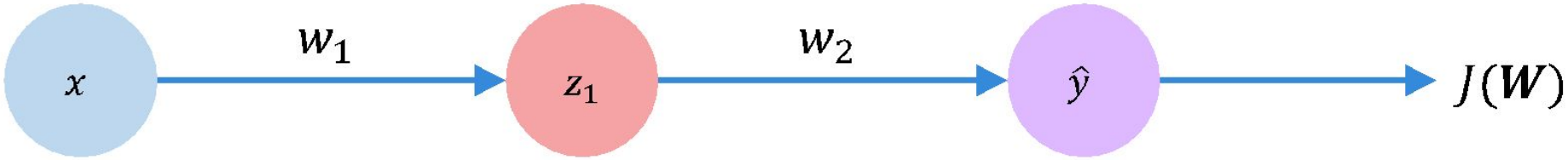
4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$

```
weights_new = weights.assign(weights - lr * grads)
```

5. Return weights

Neural networks: gradients

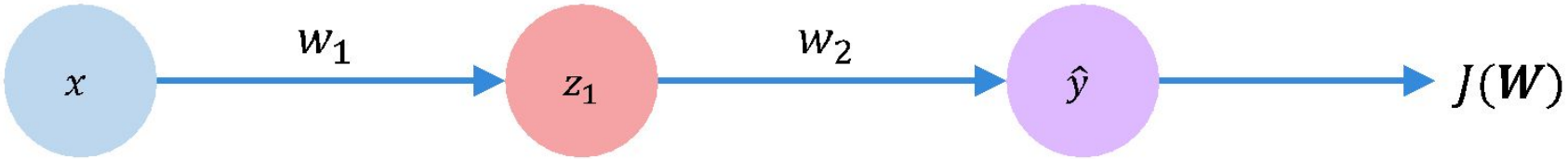
How do we compute gradients?



How does a small change in one weight (ex. w_2) affect the final loss $J(\boldsymbol{W})$?

Neural networks: gradients

How do we compute gradients?
Backpropagation

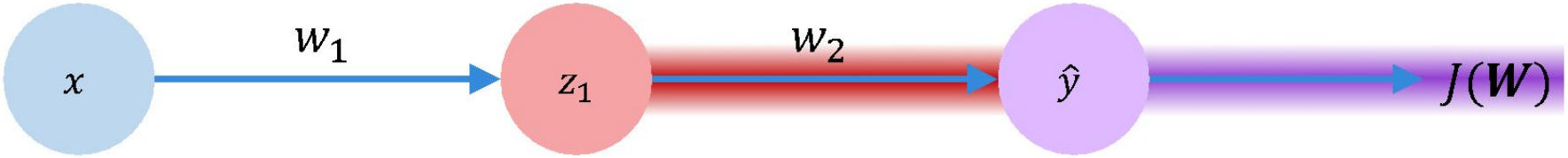


$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} =$$

Let's use the chain rule!

Neural networks: gradients

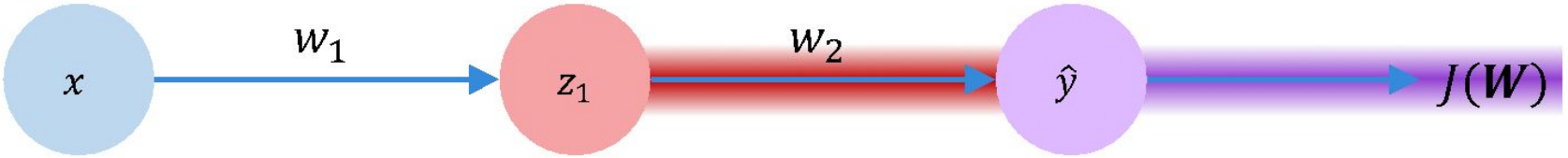
How do we compute gradients?
Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

Neural networks: gradients

How do we compute gradients?
Backpropagation

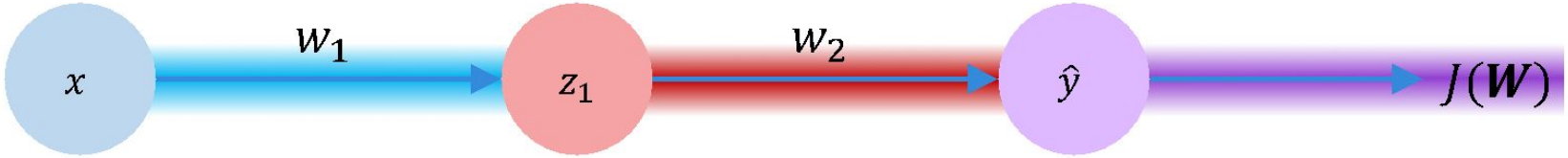


$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule! Apply chain rule!

Neural networks: gradients

How do we compute gradients?
Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for every weight in the network using gradients from later layers

Neural networks: training

Gradient descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights

```
weights = tf.random_normal(shape, stddev=sigma)
```

```
grads = tf.gradients(ys=loss, xs=weights)
```

```
weights_new = weights.assign(weights - lr * grads)
```

Learning Rate

Neural networks: training

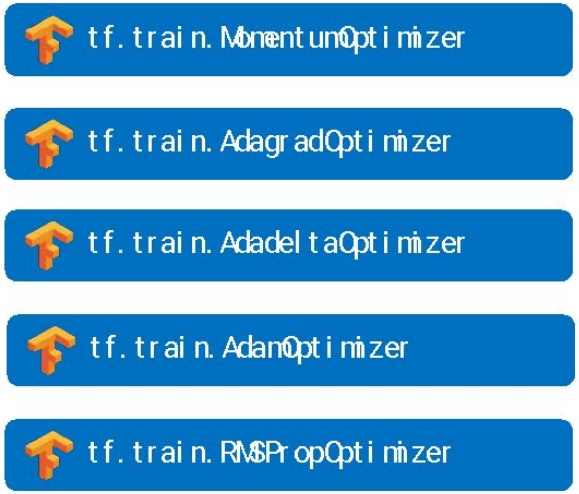
How can we handle this?

- Learning rates are no longer fixed
- Can be made larger or smaller depending on:
  - how large gradient is
  - how fast learning is happening
  - size of particular weights
  - etc...

Neural networks: training

Learning rates

- Momentum
- Adagrad
- Adadelta
- Adam
- RMSProp



Qian et al. "On the momentum term in gradient descent learning algorithms." 1999.

Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.

Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.

Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.

Neural networks: training

Learning rates

Momentum – large rate but dampens oscillations

Image 2: SGD without momentum

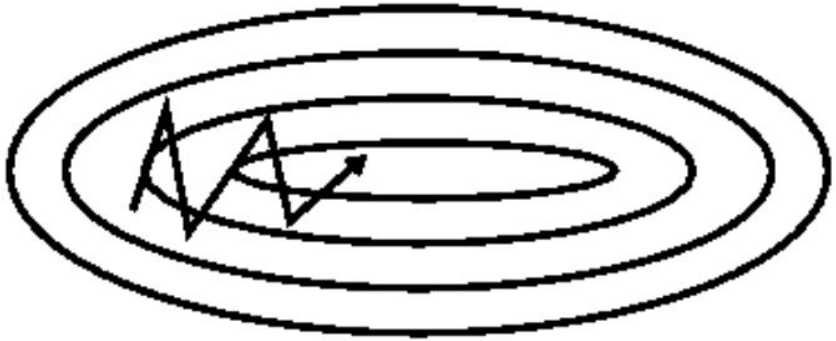


Image 3: SGD with momentum

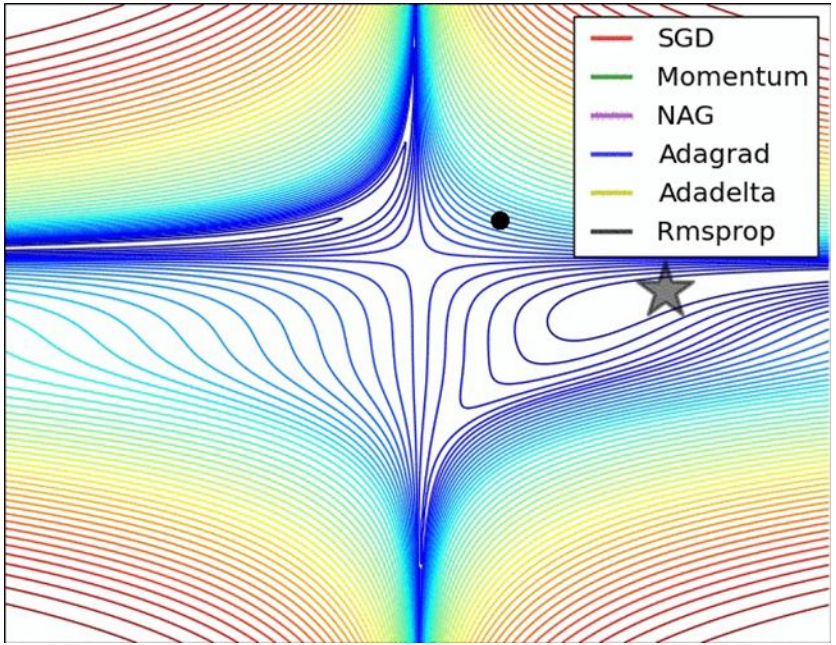
Neural networks: training

POPULAR OPTIMIZERS

Adagrad - adapts the learning rate to the parameters:
smaller updates -> frequently occurring features
larger updates -> infrequent features.

Adadelta - reduce aggressive, monotonically decreasing learning rate.

Adam - Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates. Adam also keeps an exponentially decaying average of past gradients , similar to momentum.



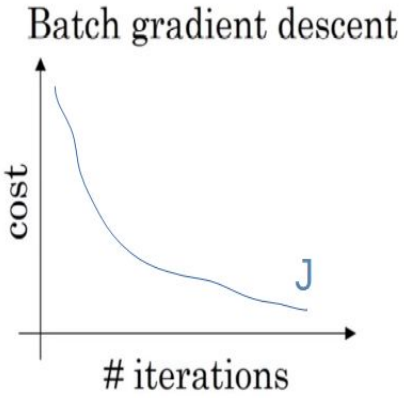
Neural networks in practices: mini batches

Upsides

- The model update frequency is higher than batch gradient descent which allows for a more robust convergence, avoiding local minima.
- The batching allows both the efficiency of not having all training data in memory and algorithm implementations.

Downsides

- Mini-batch requires the configuration of an additional “mini-batch size”
- Error information must be accumulated across mini-batches of training examples



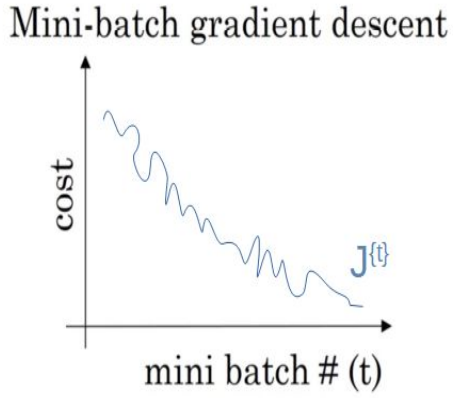
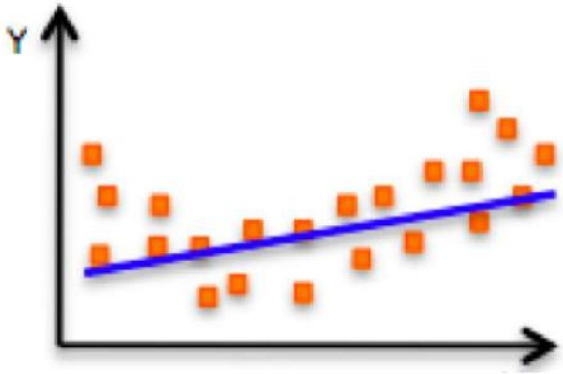
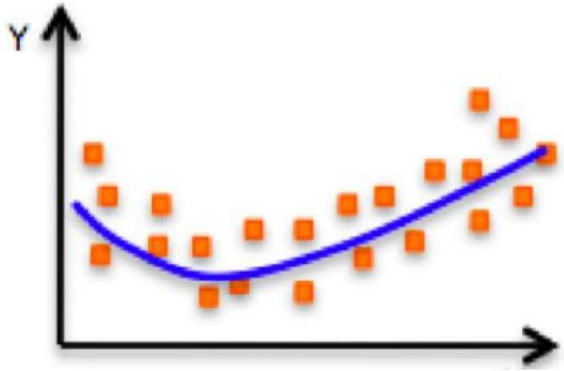
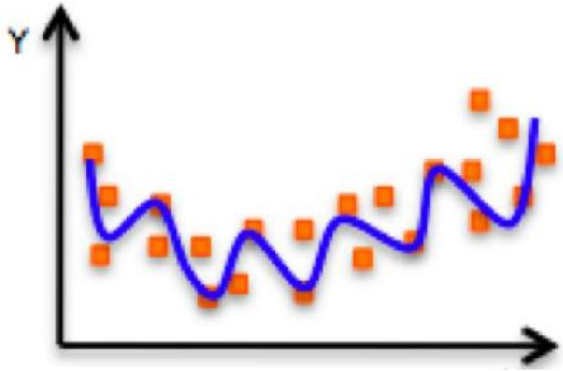


Image: Andrea Perlato

Neural networks in practices: overfitting







Underfitting
Model does not have capacity to fully learn the data

Ideal fit

Overfitting
Too complex, extra parameters, does not generalize well

Neural networks in practices: overfitting

Regularization

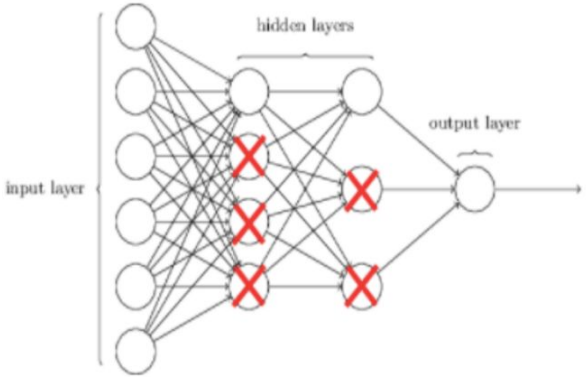
What is it?
Technique that constrains our optimization problem to ***discourage complex models***

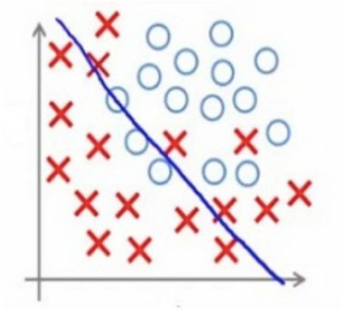
1. L1 & L2

2. Dropout

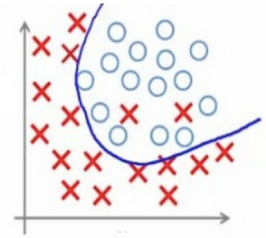
3. Early Stopping

Neural networks in practices: L1 & L2





Under-fitting



Appropriate-fitting

Images: Analytics Vidhya

Neural networks in practices: L1 & L2

L1:

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum ||w||^2$$

L2:

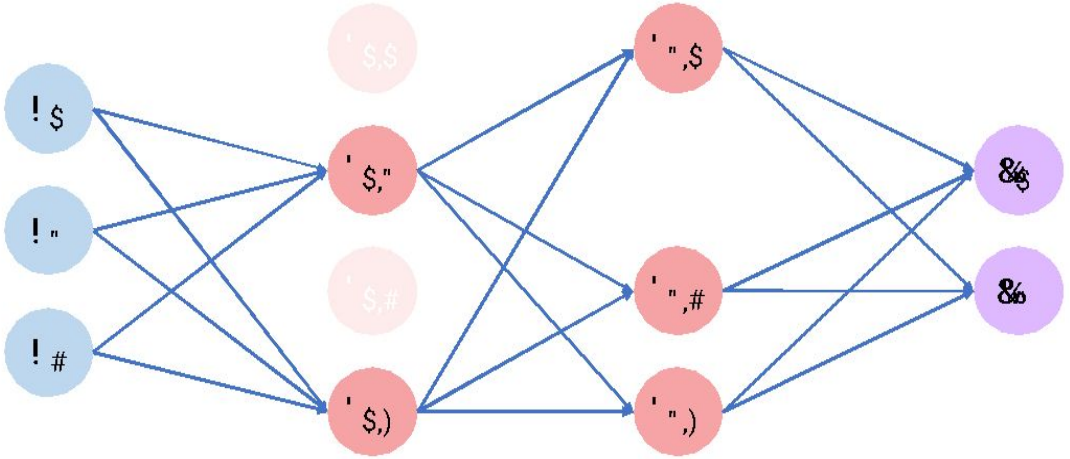
$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum ||w||$$

Images: Analytics Vidhya

Neural networks in practices: Dropout

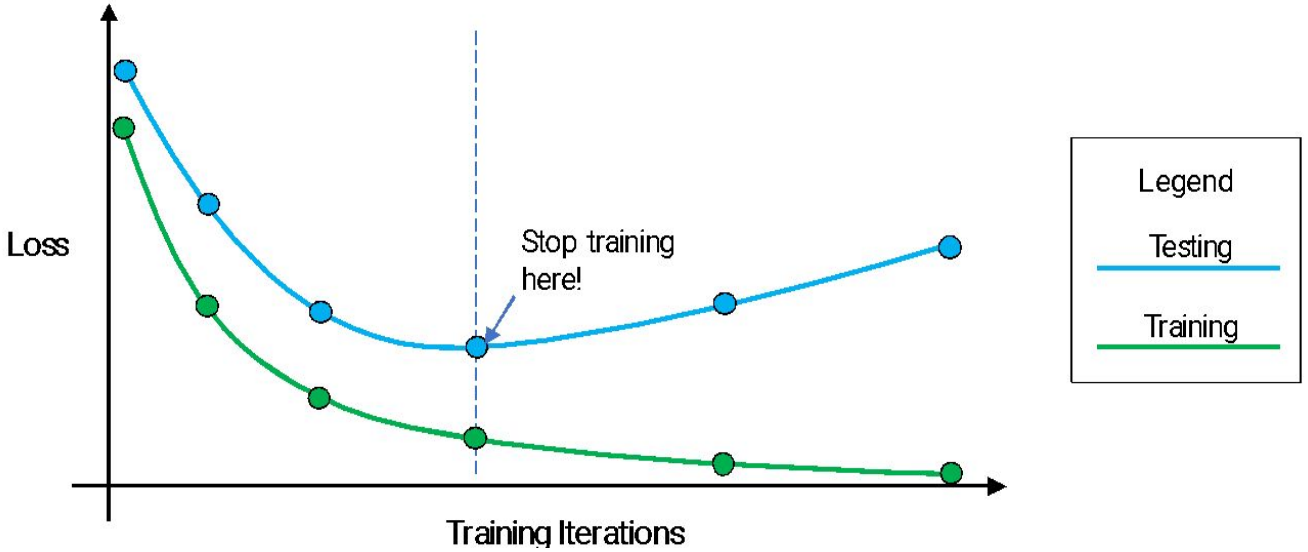
- During training, randomly set some activations to 0
  - Typically 'drop' 50% of activations in layer
  - Forces network to not rely on any 1 node

`tf.keras.layers.Dropout(p=0.5)`



Neural networks in practices: Early Stopping

- Stop training before we have a chance to overfit



Neural networks in practices

SUGGESTED STRATEGIES WITH KEY HYPERPARAMETERS:

- **Number of hidden layers:** Keep your NN as simple as possible -> Proceed with manual attempts.
- **Learning rate:** Start with a value around 0.1, and then exponentially reduce it: at some point, the value of the loss function starts decreasing in the first few iterations and that's the signal the weight took the right direction.
- **Momentum**: Start with low values and then increasing them little by little.
- **Activation Functions**: ReLU function is very quick in terms of training, while the Sigmoid is more complex and it takes more time. Start with ReLU.
- **Minibatch Size**: The typical size is 32 or higher, however you need to keep in mind that, if the size is too big, the risk is an over generalized model which won't fit new data well.

Neural networks in practices

(CONT'D) SUGGESTED STRATEGIES WITH KEY HYPERPARAMETERS:

- **Epochs:** the number of times you want your algorithm to train on your whole dataset (note that epochs are different from iterations: those latter are the number of batches needed to complete one epoch). An idea could be imposing a condition such that epochs stop when the error is close to zero. Or, more easily, you can start with a relatively low number of epochs and then increase it progressively, tracking some evaluation metrics (like accuracy).
- **Dropout**: Removing some nodes so that the NN is not too heavy. This can be implemented during the training phase. You do not want our NN to be overwhelmed by information, considering some nodes might be redundant and useless. Keep for each training stage, each node with probability p (called 'keep probability') or drop it with probability 1-p (called 'drop probability').

Lab