



# Neural networks



Data Science, AU, Fall 2023

Ira Assent

# Where we are...

---

1. 11/10 Introduction, data preprocessing, PCA, clustering
2. 13/10 Unsupervised learning, more clustering, outlier detection
3. 23/10 Supervised learning, classical machine learning: DT, SVMs,...
4. 26/10 Neural networks, pitfalls, outlook

- ▶ We looked at supervised learning
  - ▶ Setup: known classes, training data with correct labels
    - ▶ Decision Trees: flow-chart like structure
      - Use ensemble of random trees for outlier detection in Isolation Forest
    - ▶ Naïve Bayes: probabilistic, assume attribute independence
    - ▶ Nearest neighbor: lazy, pick closest samples to determine label
    - ▶ Support Vector Machines: linear separation in high dimensions, using kernel trick
      - Support vectors (data samples on the margin) describe separation between classes
  - ▶ Evaluation: confusion matrix, derived measures like precision/recall
    - use test, not training data for evaluation!

Any  
comments?  
Questions?

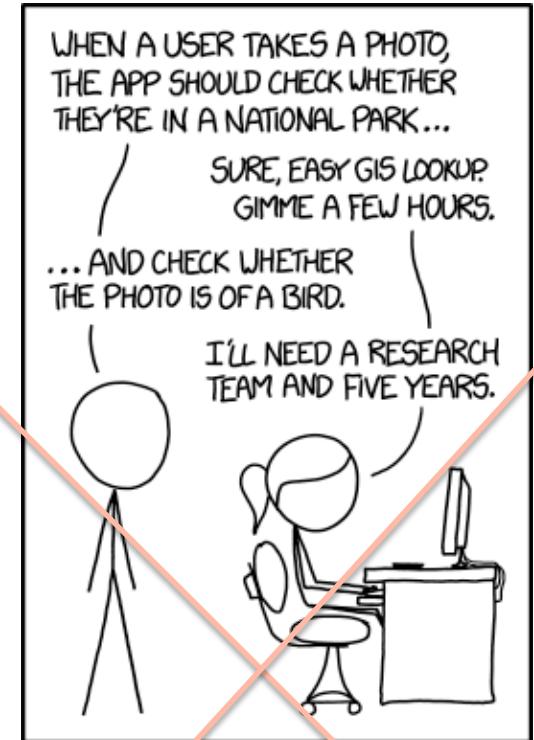
# Today's learning goals

---

- ▶ Neural networks
  - ▶ Where does the recent interest come from?
  - ▶ General architecture and learning approach
    - ▶ Activation functions
    - ▶ Loss functions
    - ▶ Overfitting + countermeasures
  - ▶ CNNs: images / spatial data
  - ▶ RNNs: text / sequence data
  - ▶ Transformers: text, complex data
- ▶ XAI: explainable AI
  - ▶ Transparency, accountability
  - ▶ Current approaches
- ▶ Golden Rules

# Image Recognition

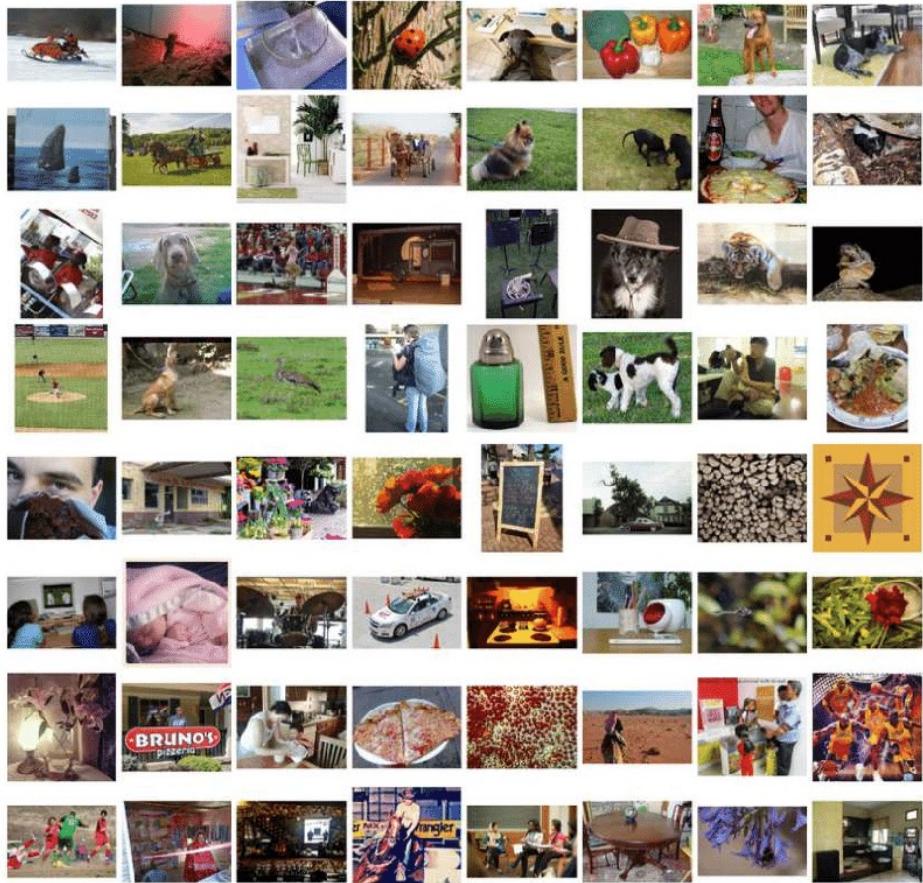
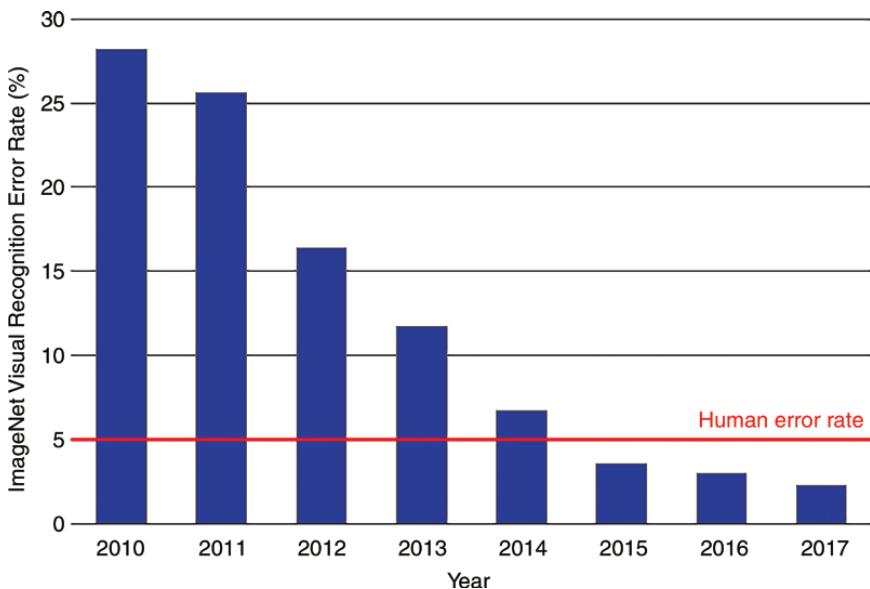
- Previously a very hard task



IN CS, IT CAN BE HARD TO EXPLAIN  
THE DIFFERENCE BETWEEN THE EASY  
AND THE VIRTUALLY IMPOSSIBLE.

# Deep Learning Breakthrough

- 2012: Started using deep neural nets. AlexNet
  - Better than humans on ImageNet data set
  - Still not better "in the wild"



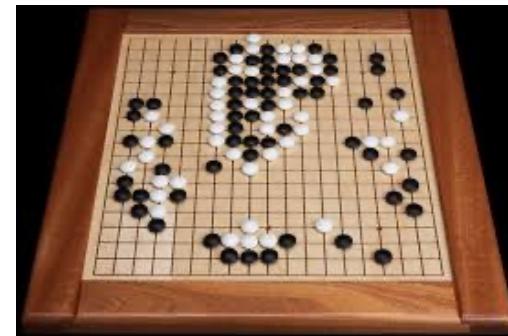
# Deep Learning Breakthrough

---

- Machine learning models based on learning multiple levels of abstraction
- Amazing improvements in object recognition, object detection, speech recognition, natural language processing and reinforcement learning



- Need lots of data
- Need computational power
  - (servers, GPUs)



# Chatgpt: generative AI for text

---

Human-like chatting ability

Developed by OpenAI



Built on top of GPT models  
(transformer neural networks),  
combined with unsupervised  
learning – hundreds of billions of  
parameters

Reinforcement learning to  
reduce misinformation and  
offensive text

The screenshot shows a dark-themed chat interface. It consists of three horizontal message cards. The first card has a green 'EK' icon and the text 'I'm giving this talk at the DDSA pre-graduate retreat'. The second card has a blue 'GPT' icon and a response about assisting with the talk. The third card has a green 'EK' icon and a request for a funny answer. The fourth card, which is partially visible, has a blue 'GPT' icon and a response about including humor in presentations, followed by a quote and a cat emoji.

EK I'm giving this talk at the DDSA pre-graduate retreat

GPT That sounds like a great opportunity to share knowledge and insights. How can I assist you with your talk for the DDSA pre-graduate retreat? Do you need help with developing your content, structuring your presentation, or something else? Please provide me with more details so I can assist you effectively.

EK Thanks, I'd like some cool funny answer that I can include in my slide

GPT Of course! Including humor in your presentation can make it more engaging and memorable. Here's a light-hearted quote you could consider using:

"Data may not be able to solve all of life's mysteries, but it sure can help us find the best cat videos on the internet!" 🐱📊

# Dall-e: generative AI for images

---

Generate realistic images  
using text prompts

Developed by OpenAI  
Built on top of GPT and CLIP  
image embeddings

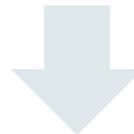
Concerns about  
misinformation and deepfakes



# Neural Nets

---

Brain Inspiration

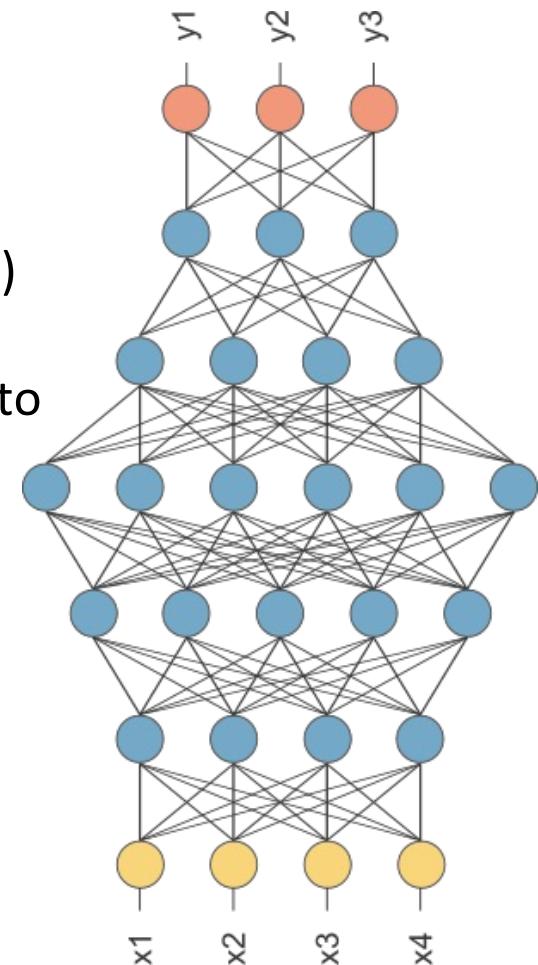


Mathematical Models for  
Machine Learning

NOT a model of the brain!

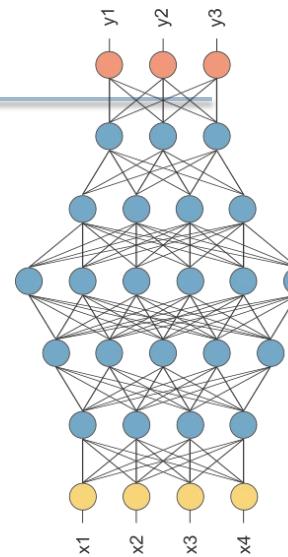
# Fully connected network

- Artificial neural network
  - many nodes (neurons), organized in layers
  - connected by edges
  - prediction / classification on data input
  - doing computations through network (details later)
  - to output layer for output(s)
- Training processes all data in many iterations (epochs) to adjust weights (values) on edges in the network
- Data scientist decides on
  - architecture (number, organization of nodes),
  - loss function,
  - initialization (how to start), and
  - parameters for training

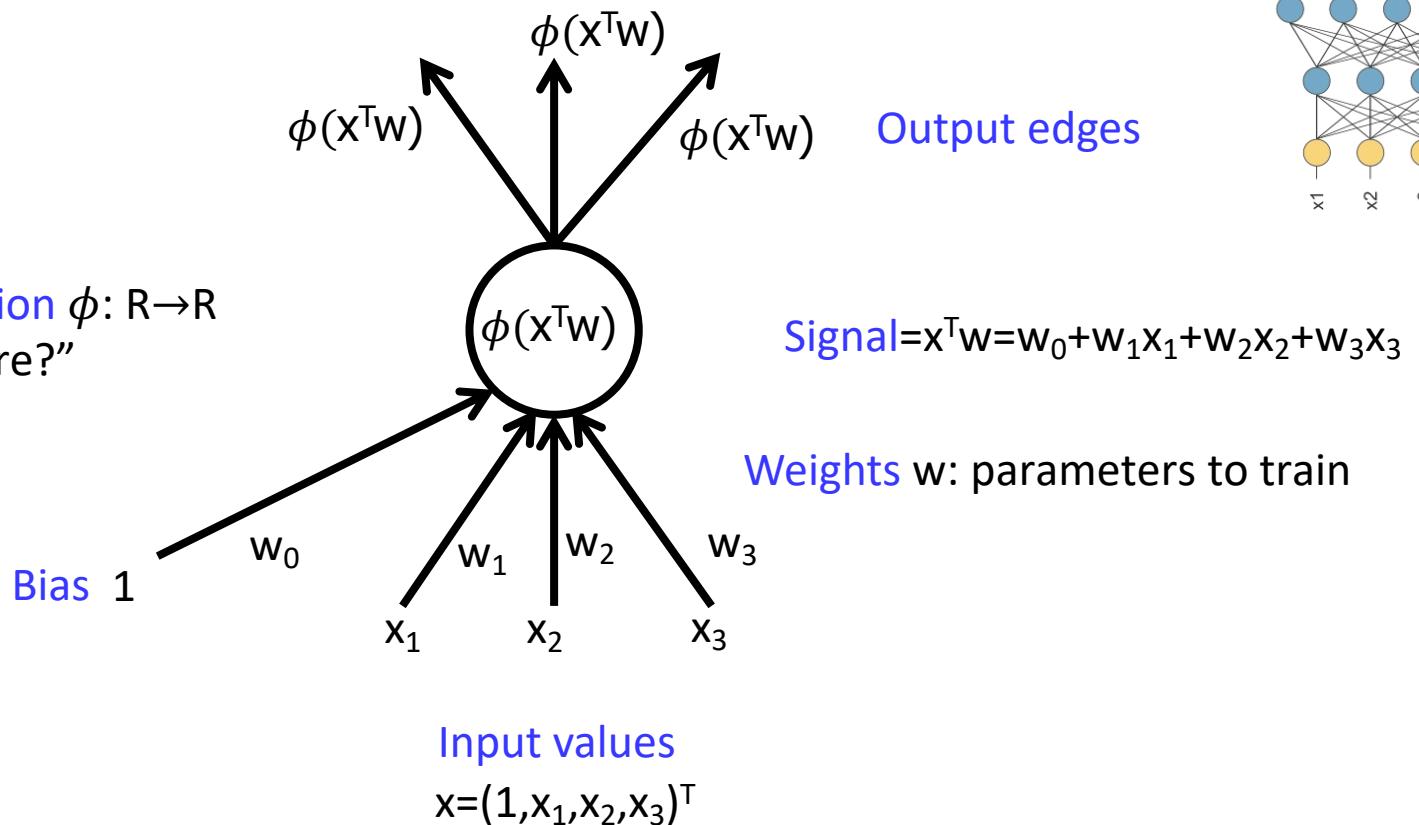


# Neurons

- Building block in neural networks
  - Input values from preceding layers

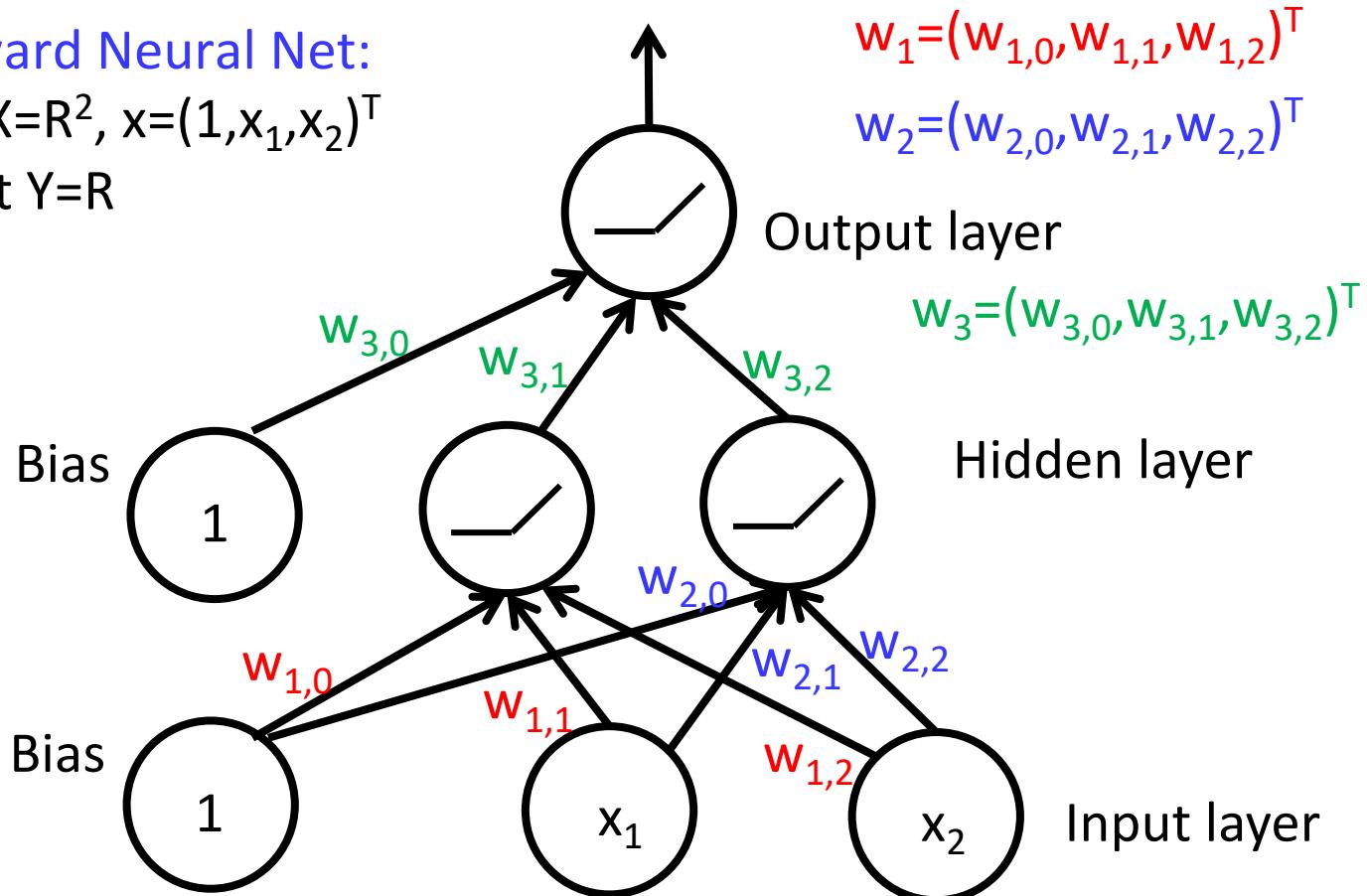


Activation function  $\phi: \mathbb{R} \rightarrow \mathbb{R}$   
"Does neuron fire?"



# Neural Nets

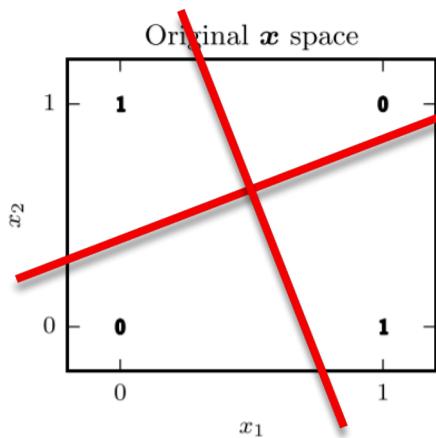
- Feed Forward Neural Net:
  - Input  $X=R^2$ ,  $x=(1, x_1, x_2)^T$
  - Output  $Y=R$



Feed Forward: All edges point "forward"

# Activation Function: linear vs. non-linear

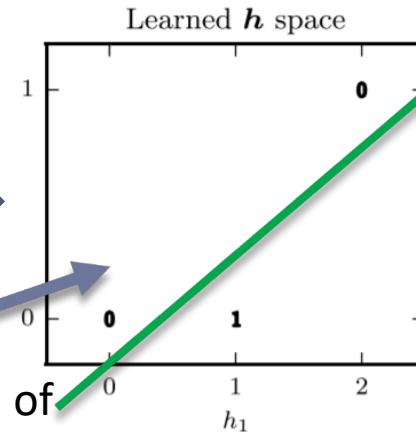
- Linear function: output is a linear transformation of input
- Example: XOR



Non-linear activation  
function

Transformation of  
Input space

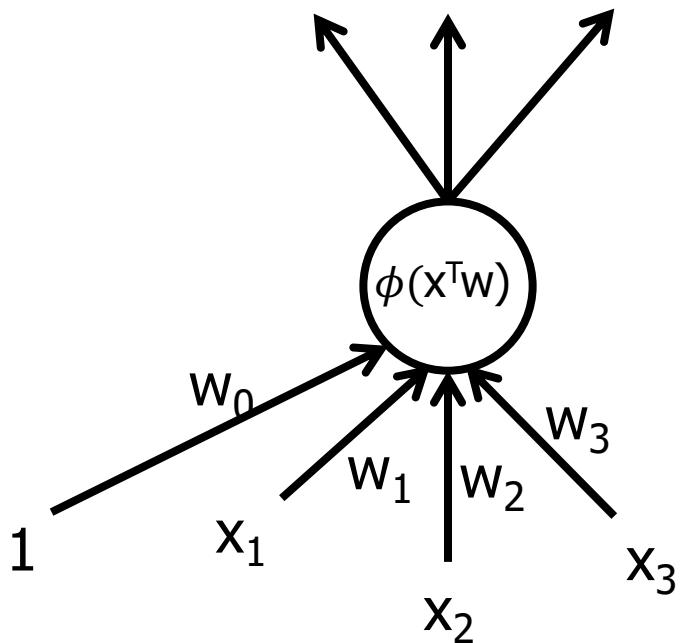
$$f(\mathbf{x}; W, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^T \max\{0, \mathbf{X}\mathbf{W} + \mathbf{c}\} + b$$



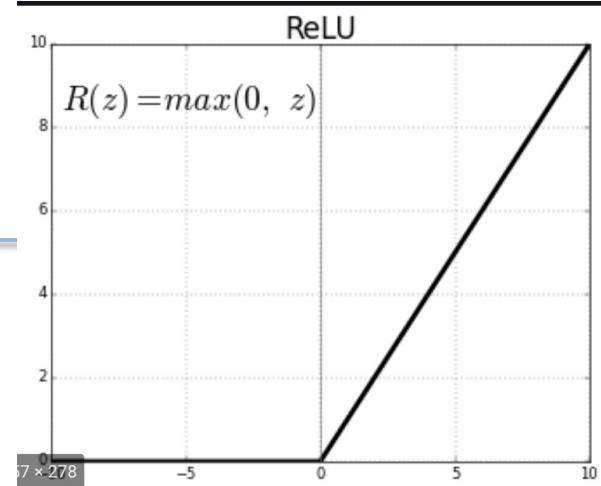
$$\mathbf{w}^T \max\{0, \mathbf{X}\mathbf{W} + \mathbf{c}\} + b = [1 \quad -2] \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix} + 0 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = Y$$

# Activation Functions

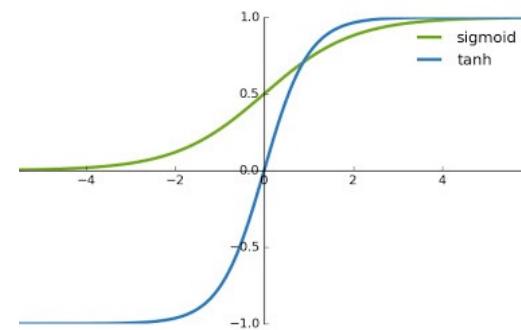
- Activations:
  - Many choices
  - Many work roughly equally well



Activation function  $\phi: \mathbb{R} \rightarrow \mathbb{R}$



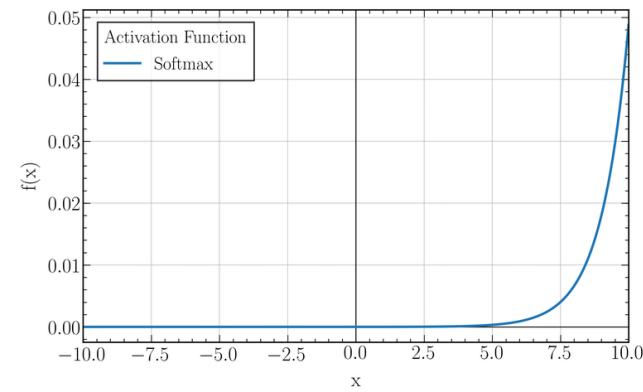
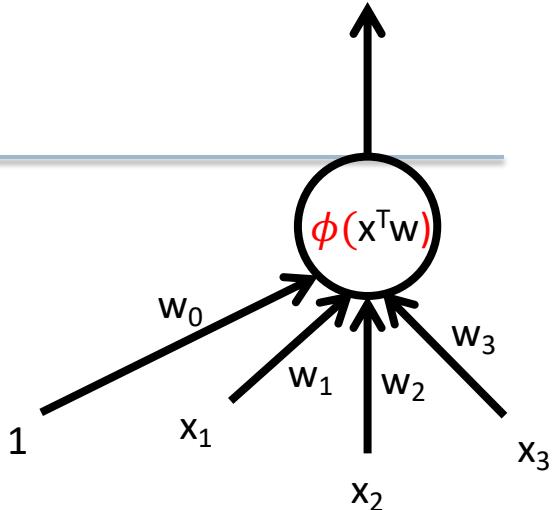
- Rectified Linear Unit
  - Linear for  $z > 0$ , constant for  $z \leq 0$
  - Non-linear function
  - "Fires" for  $z > 0$ .
- Logistic Sigmoid
  - "Fires" as  $z$  becomes big enough
- Hyperbolic Tangent



# Activation Functions

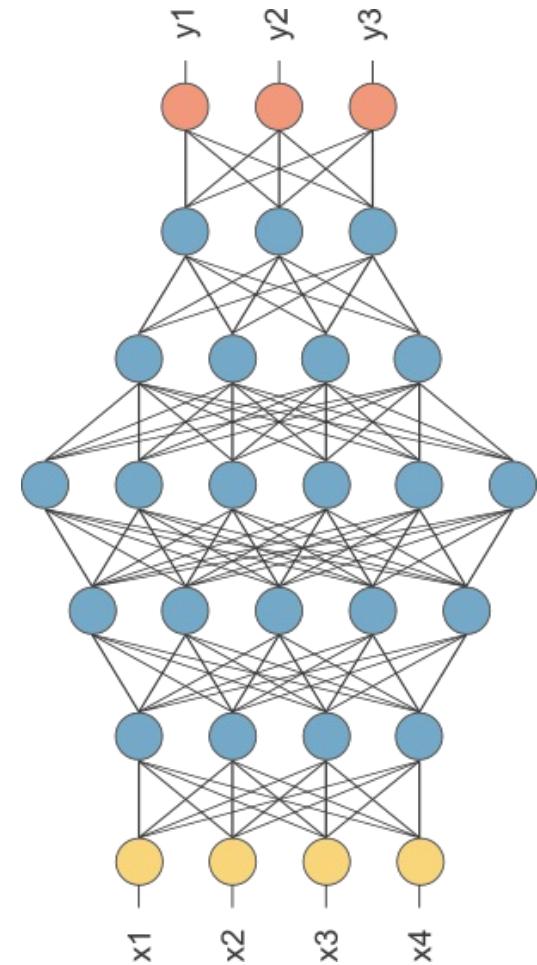
- Output Neurons
  - Often have their own activation functions
- **Regression** Output Neurons
  - For regression problems with least squares loss  $L(y', y) = (y - y')^2$ , often use **Identity**  $I(z) = z$
- **Binary Classification** Output Neurons
  - For binary classification problems with negative log-likelihood (like logistic regression), often use **Sigmoid**
- **Multiclass Classification** Output Neurons
  - For classification problems with negative log-likelihood, often use **Softmax**
    - takes in real number, outputs number between 0 and 1

$$\frac{e^{x_i}}{\sum_{j=1}^J e^{x_j}}$$



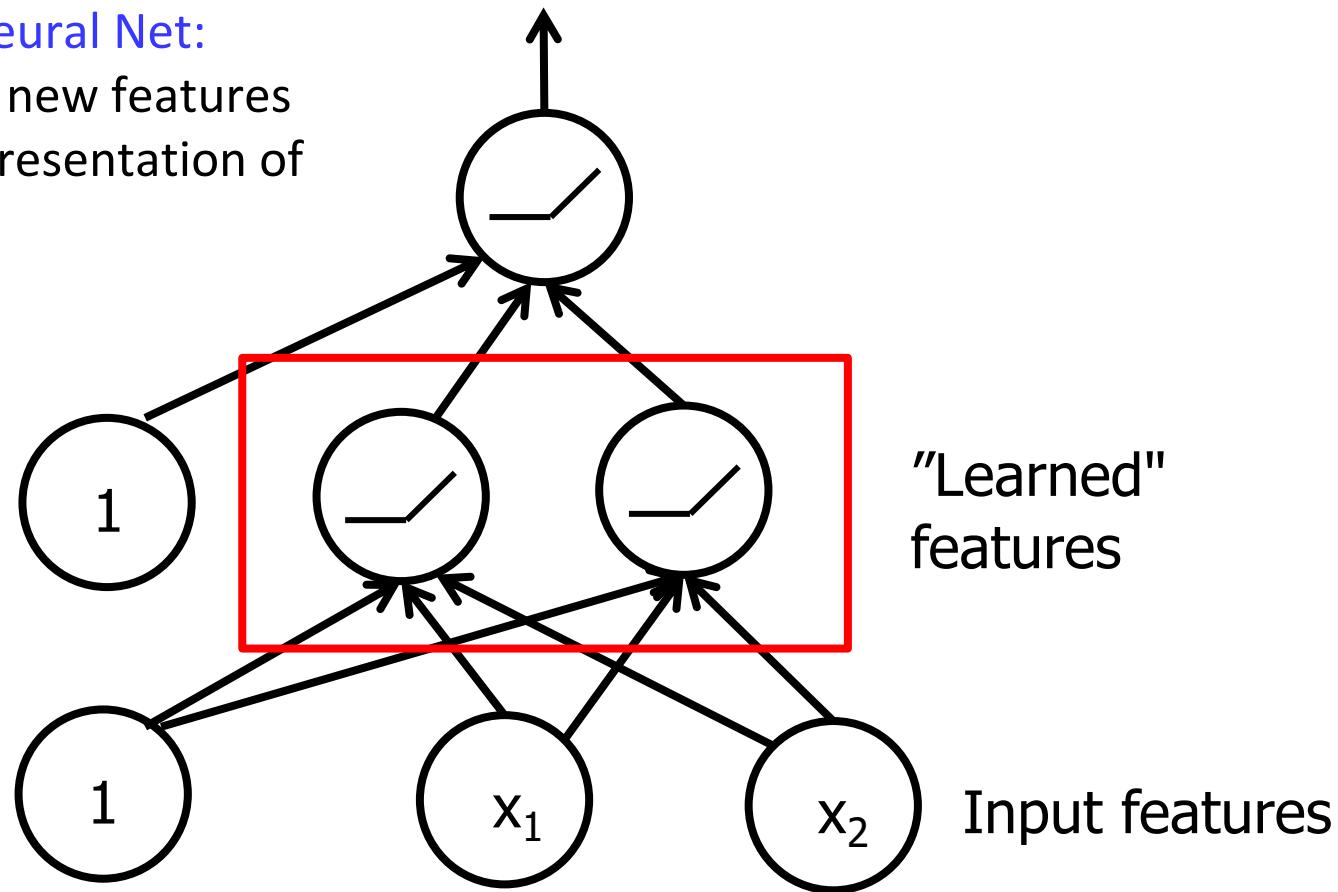
# Why so many layers?

- a) Easier to learn the weights for the input
- b) Easier to learn features from the input
- c) Easier to compute the loss function



# Feature Transform

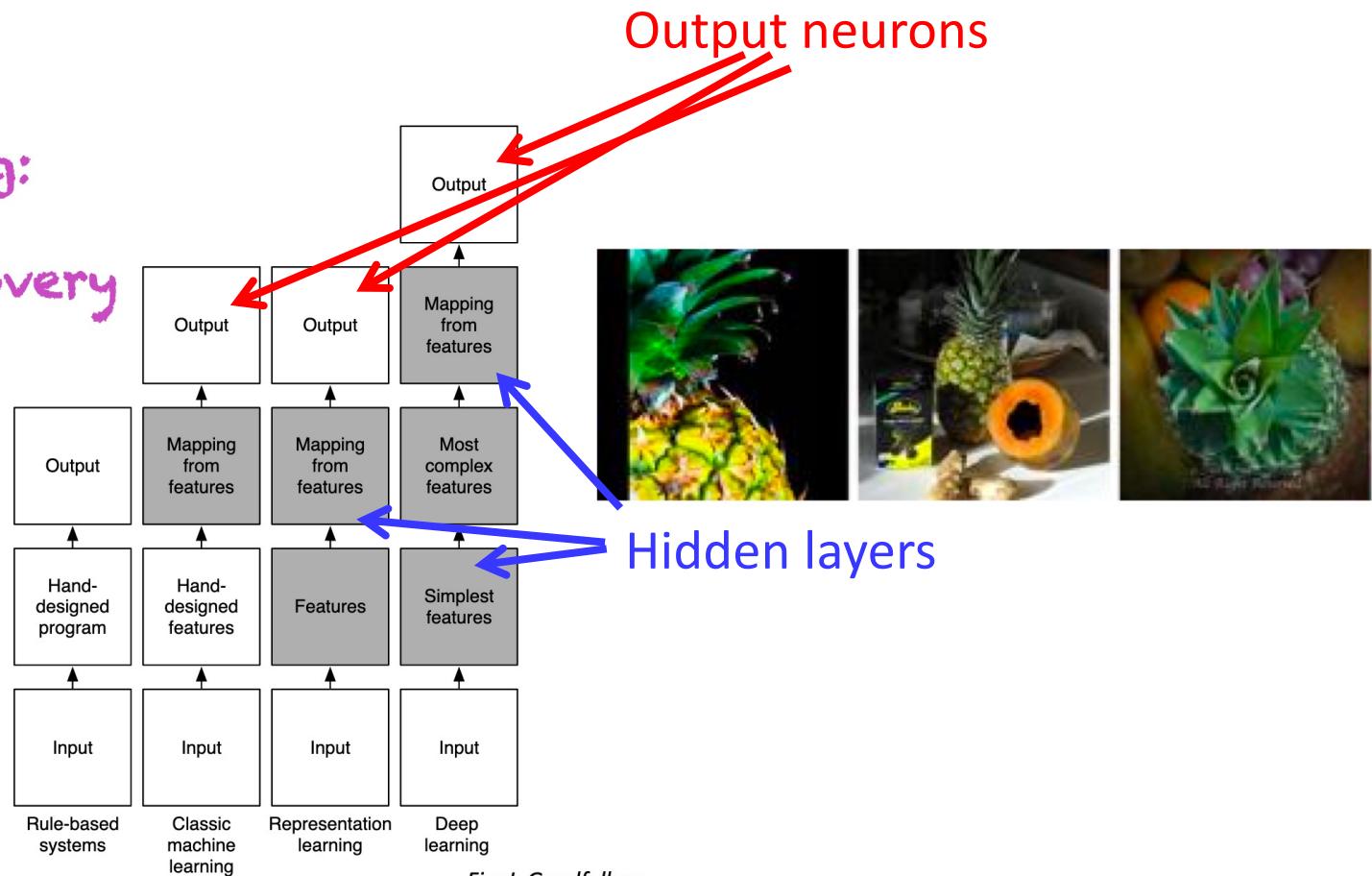
- Feed Forward Neural Net:
  - Layers learn new features
  - Another representation of input



# Deep Learning

- Machine learning models based on learning multiple levels of abstraction.

Deep Learning:  
Automating  
Feature Discovery



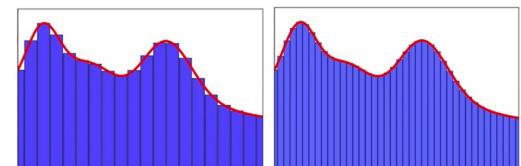
# Universal Approximation Theorem

- „A feed-forward neural network with a single hidden layer and continuous non-linear activation function can approximate any continuous function with arbitrary precision“

1. Neural Network
  2. One hidden layer
  3. Non-linear activation function
- } Approximate any function with any precision

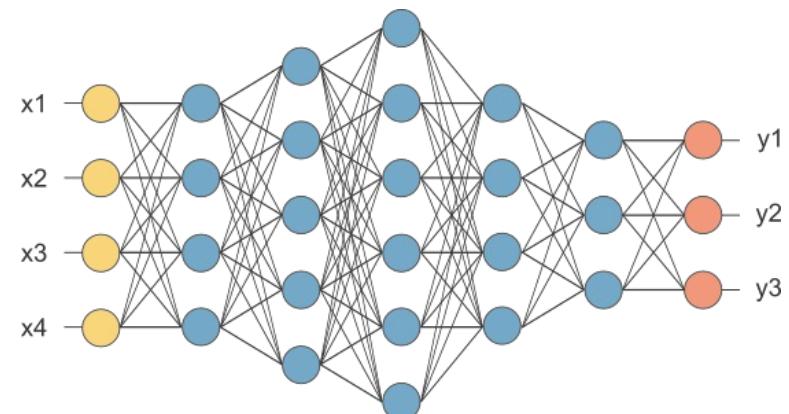
- Deep Neural Networks ><sub>in practice</sub> Wide Neural Networks

- BUT
  - Need enough data
  - Need enough training
  - ...



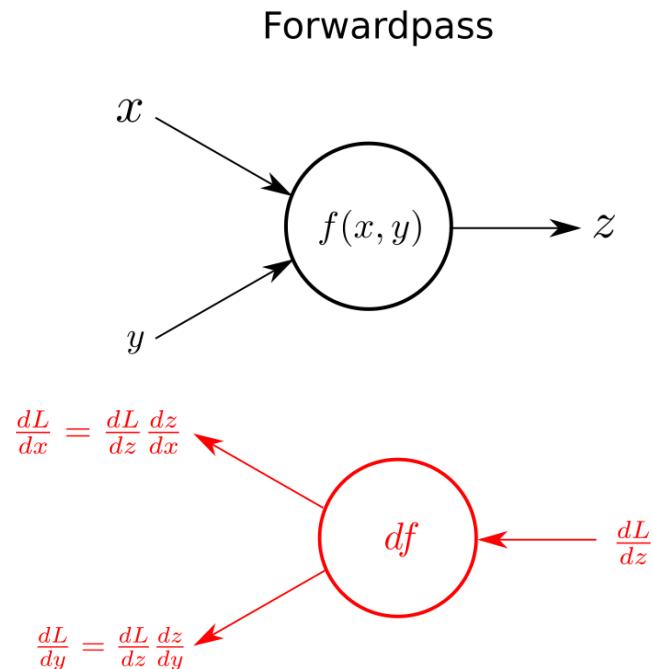
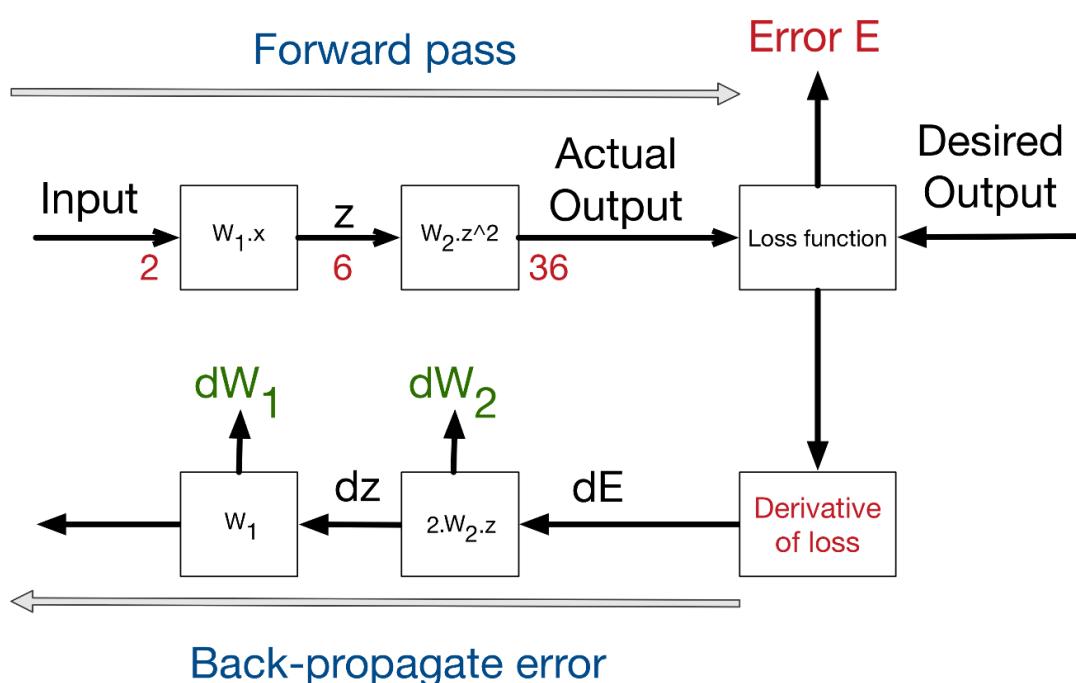
# Training

- The training processes all data in many iterations (epochs) to adjust the weights (values) on the edges in the network
- As for all classification, use training data where correct outcome known
- Use computations on training input through entire network to obtain output
- Define loss function to compare prediction with correct answer
  - So, choice of loss function important for model!
- We want to adjust the weights based on the loss (error) to improve predictions
  - Move in the opposite direction of the gradient of the loss function
    - Move towards where error is reduced
- Different training algorithms exist to increase the chances of finding a stable optimum, ideally global
  - But, not guaranteed!
  - In practice, train several times with different random initializations

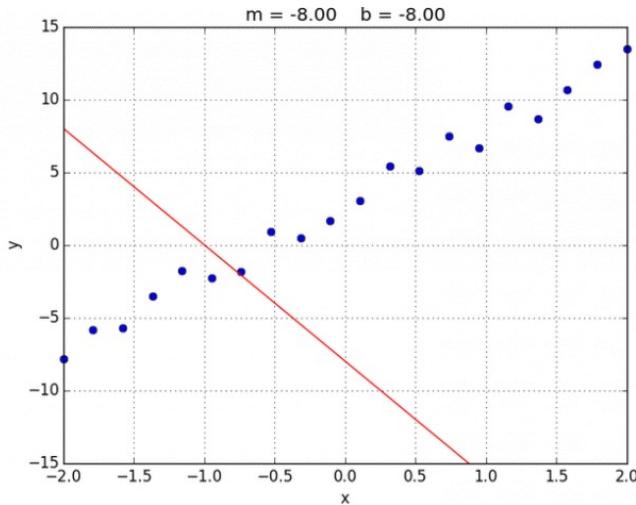
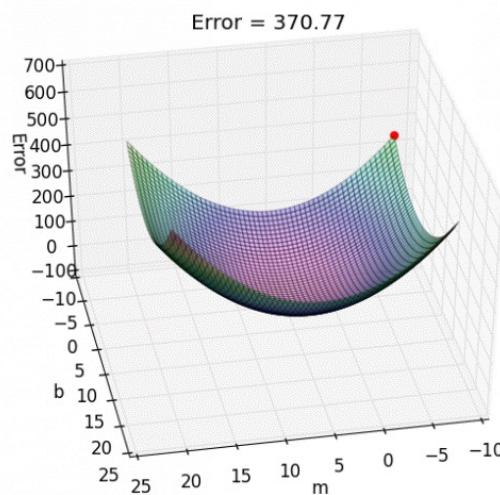


# Training Neural Networks

- 1. Forward pass: compute output based on input of training data
- 2. Backward pass: Adjust weights/biases to reduce loss



# Gradient Descent: minimizing the loss



Do this for every weight/bias

$$\theta_1^{(i+1)} = \theta_1^{(i)} + \alpha \frac{\delta}{\delta \theta_1} J(\theta^{(i)})$$

Learning Rate      Partial derivative (Gradient)      Cost function

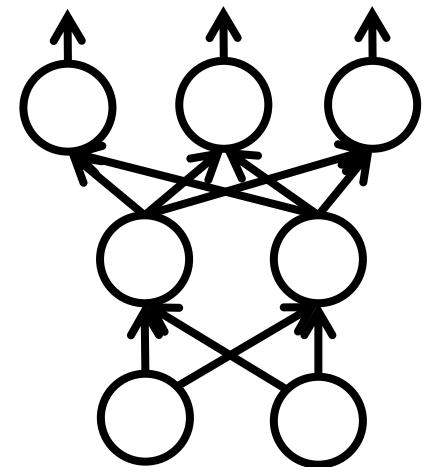
# Training

---

- How do we train such powerful models?
- Nothing more but small steps in the negative direction of the gradient (with respect to the model parameters)
- (Mini-Batch) Stochastic Gradient Descent!
  - Feed in data in chunks / partitions / batches
  - After each randomly selected batch, update weights
  - Once selected all data, one epoch /iteration is completed, start over with new random batches in next epoch
  - Add fancy control of learning rate and tricks to avoid overfitting

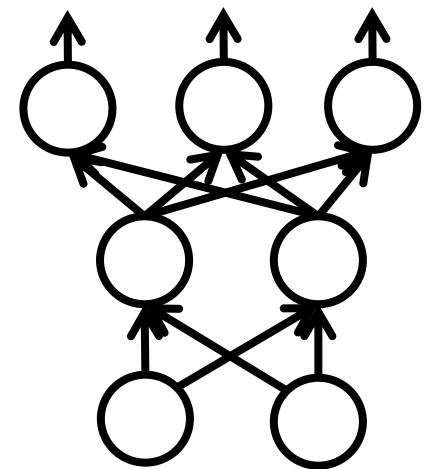
# Cross Entropy Loss

- Loss function for multiclass classification:
  - Generalizes negative log likelihood from multinomial regression
- Setup:
  - Any machine learning model  $h$  outputting  $K$  probabilities  $p_1, \dots, p_K$  summing to 1.
    - $p_j = h(x)_j$
    - Intention  $\Pr[y=j|x] = p_j$ .
- Loss:
  - $L(h(x), y) = -\sum_{j=1}^K y_j \ln(h(x)_j)$
  - $E_{in}(h) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^K y_{i,j} \ln(h(x_i)_j)$



# Cross entropy loss and regression

- Multinomial Regression:
  - $L(h(x), y) = -\ln(y^T \text{softmax}(x^T W))$
  - $E_{in}(W) = -\frac{1}{n} \sum_{i=1}^n \ln(y_i^T \text{softmax}(x_i^T W))$
- Cross Entropy Loss:
  - $L(h(x), y) = -\sum_{j=1}^K y_j \ln(h(x)_j)$
  - $E_{in}(h) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^K y_{i,j} \ln(h(x_i)_j)$
- In multinomial regression, assumed label vector  $y$  has one 1 and rest 0
  - $-\ln(y^T \text{softmax}(x^T W))$
  - $= -\sum_{j=1}^K y_j \ln(\text{softmax}(x^T W)_j)$
- Special case!



# Validation

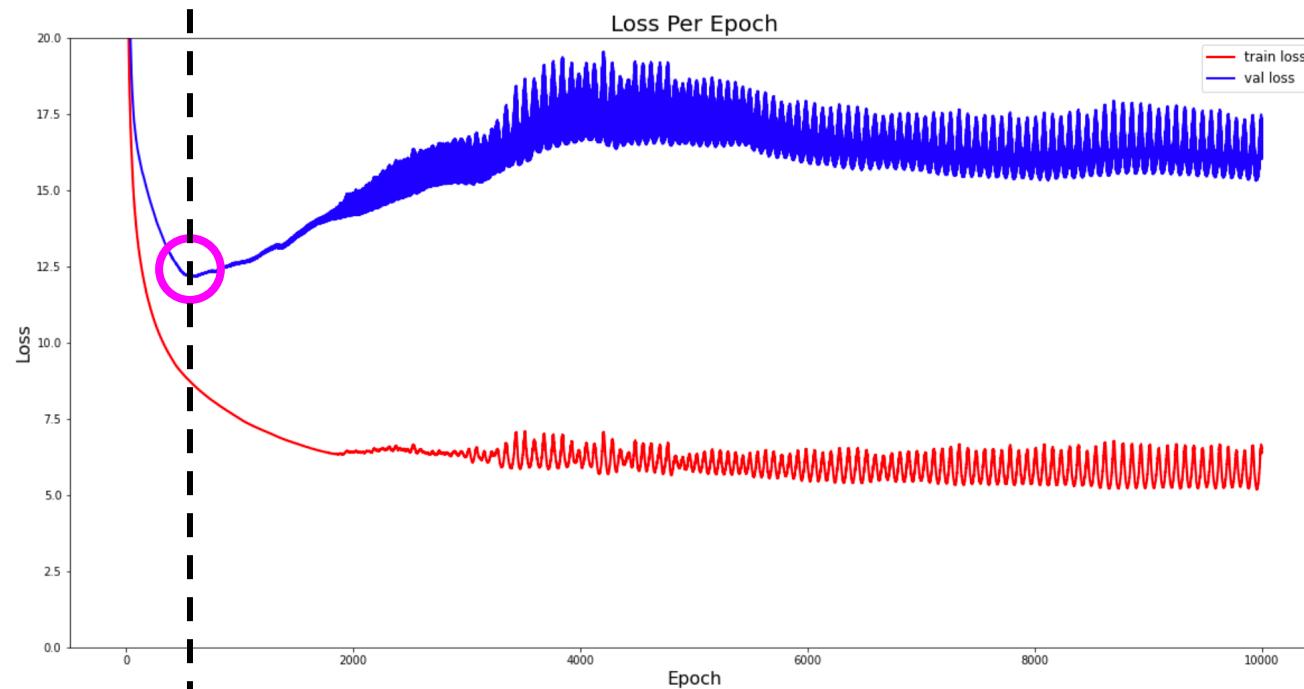
---

- ▶ In many machine learning models, we need to select hyperparameters, model types etc
  - ▶ Use validation data set to see how the model fares
- ▶ Remember
  - ▶ Training data is used for learning the model
  - ▶ Validation data used to select best learnt model
  - ▶ Test data used to assess the performance of the learnt model
- ▶ Never use test data when selecting model 
  - ▶ Gives unrealistic (overly optimistic) impression of its expected performance on unseen data
    - ▶ If you use the test data for making model decisions, it is used and no longer “unseen”
    - ▶ You would trust your model more than it deserves!

# Combat overfitting– Early Stopping

- Run 10.000 epochs of SGD:
  - $E_{in} = 6.2$
  - $E_{out} \approx E_{test} = 17.4$
- Use Validation set and keep best:
  - $E_{val} = 12.5, E_{in} = 8.5$
  - $E_{out} \approx E_{test} = 17.1$

pytorch Neural Net Regression least squares score: 17.377429962158203  
pytorch Neural Net with validation score: 17.071086883544922



# Validation – Early Stopping: approach

---

- Use Validation set and keep best
  - Set aside e.g. 20% of training data as validation set
  - After each epoch, compute loss on validation set
    - If smaller than best previous:
      - Store current model parameters
  - After completion:
    - Use best model parameters seen
- Early Stopping:
  - If no improvements to best validation score for at least k epoch, stop training and use best parameters seen

# Regularization

---

- Weight Decay:
  - Add weight decay in linear regression (ridge regression)

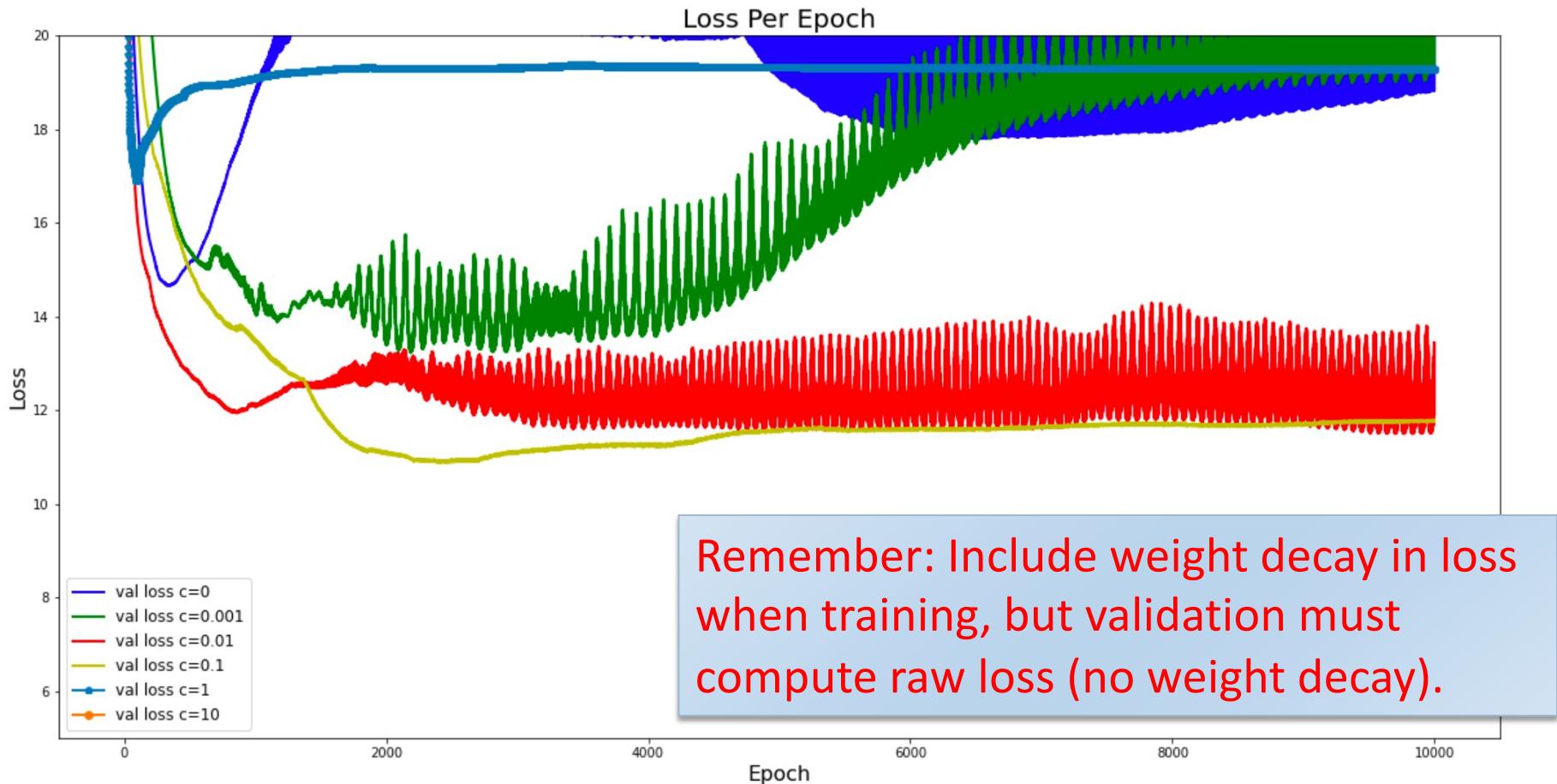
$$E_{in}(w) = \frac{1}{n} \sum_{i=1}^n (x_i^T w - y_i)^2 + \lambda \sum_{i=1}^d w_i^2$$

“Normal” linear  
regression loss

Weight decay:  
prefer small weights

- Similarly, add regularization term on the weights to loss functions in neural networks
  - Means, not an “easy” solution for the network to just increase the weights
  - Large weights tend to be a sign of overfitting

# Regularization



# Drop out

---

- ▶ Observation
  - ▶ ensembles (of neural networks) with different model configurations known to reduce overfitting
  - ▶ would require additional (training of) models: slow, expensive
- ▶ Idea
  - ▶ Simulate several models by randomly dropping out nodes during training
  - ▶ randomly disable (drop out) neurons (and their edges)
  - ▶ prevents network from relying too much on single neurons
  - ▶ forces all neurons to learn to generalize better
- ▶ remarkably effective regularization method to reduce overfitting
- ▶ use large network with more training and regularization
- ▶ important for smaller datasets (= higher risk of overfitting)

# In any case....

---

- ▶ Training neural networks starts with random initialization
  - ▶ Other randomization in training process
  - ▶ Always train, validate, test several models
  - ▶ Look at overall performance (accuracy, ....), but also at stability
- ▶ In practice, make sure to check standard deviation across cross validation folds
  - ▶ In particular when you do not have a lot of data
  - ▶ If stability (worst case performance) is important, choose model with low standard deviation
  - ▶ Otherwise, can pick model with better performance, but higher standard deviation
    - ▶ Risky, could be bad in worst case
  - ▶ Choice depends on use case

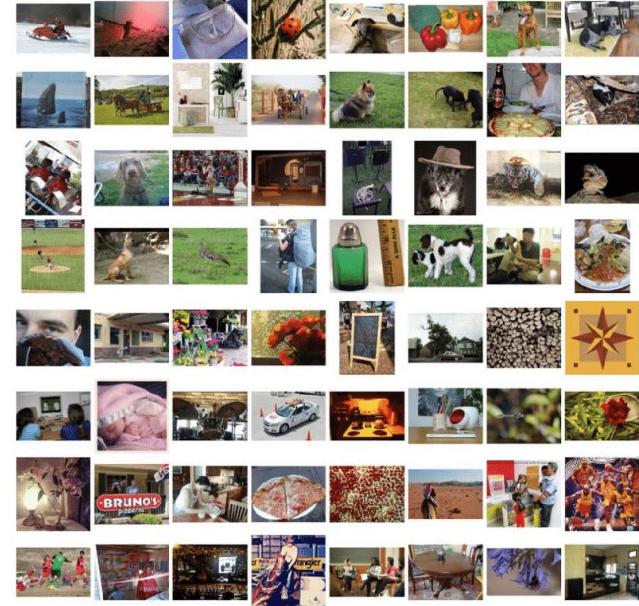
# Breathe deep

---

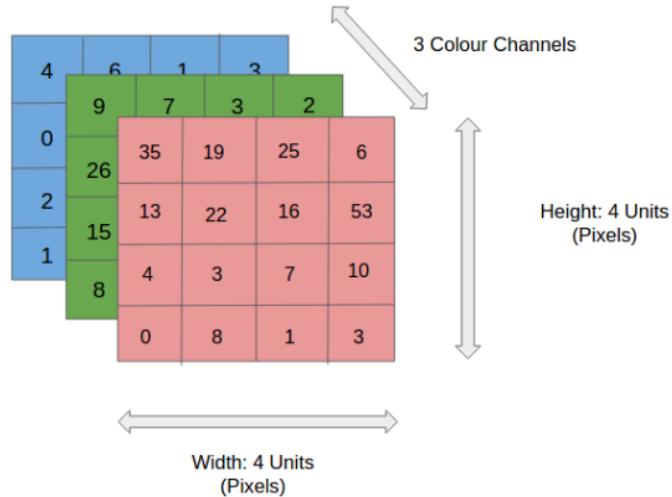


# Many more architectures

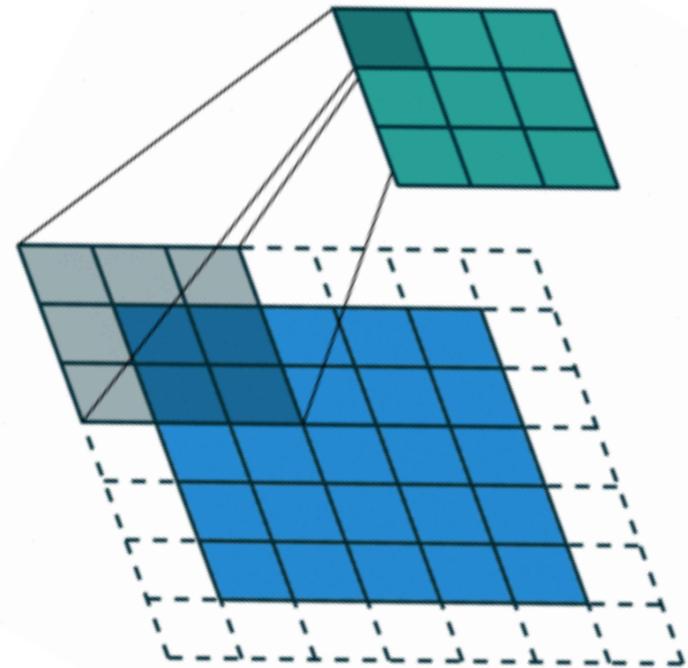
- ▶ We have seen
  - ▶ Deep neural networks powerful because
    - ▶ Complex models can be learnt
    - ▶ Much data available
    - ▶ Powerful hardware
    - ▶ Smart training algorithms
  - ▶ One more thing
    - ▶ Smart architectures
      - Many breakthroughs because of model layout
        - E.g. object recognition in images: convolutions (AlexNet)
      - An architecture is a way of connecting the neurons
        - Idea is a manual / human bias towards the kind of patterns that should be learnt



# Convolution: Intuition

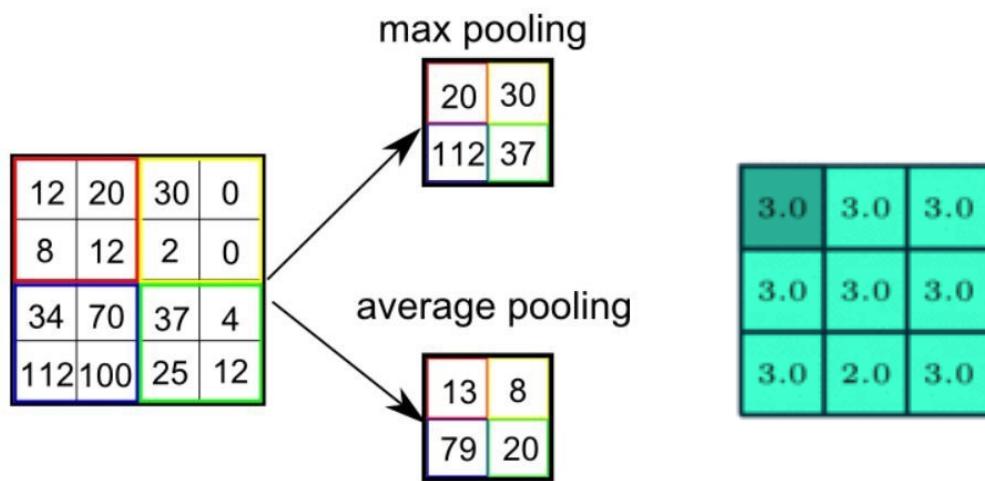


Kernel Size: 3x3x1  
Stride: 2  
Zero-Padding



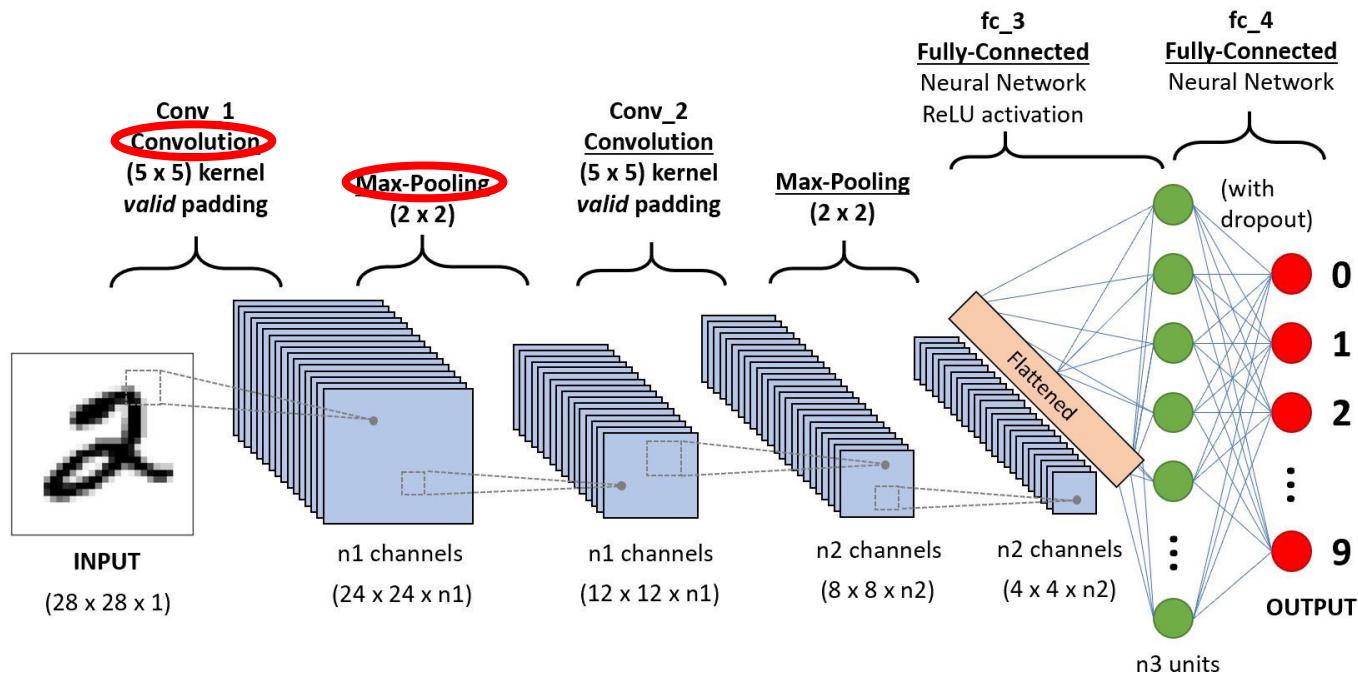
# Pooling

- Reduce Image Size
- Max/Min/Average Pooling



3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

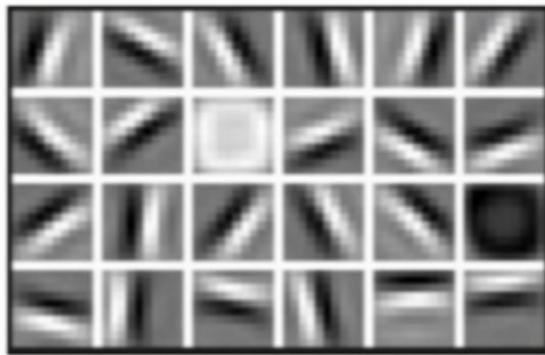
# Convolutional Neural Networks



# Feature learning

---

**Low Level Features**



Lines & Edges

**Mid Level Features**



Eyes & Nose & Ears

**High Level Features**



Facial Structure

Input---**Shallow Layers**-----**Middle Layers**-----**Deeper Layers** ----> Output

# CNNs rule?

---

- ▶ When is a CNN useful?
- ▶ When not?



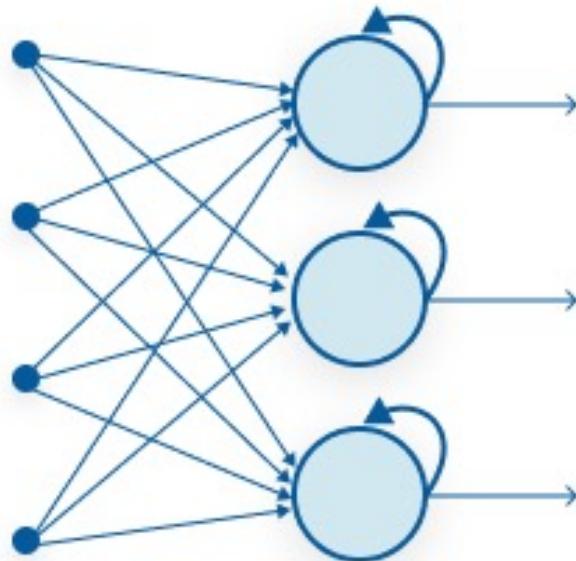
# Recurrent Neural Networks

- Used for sequence data:

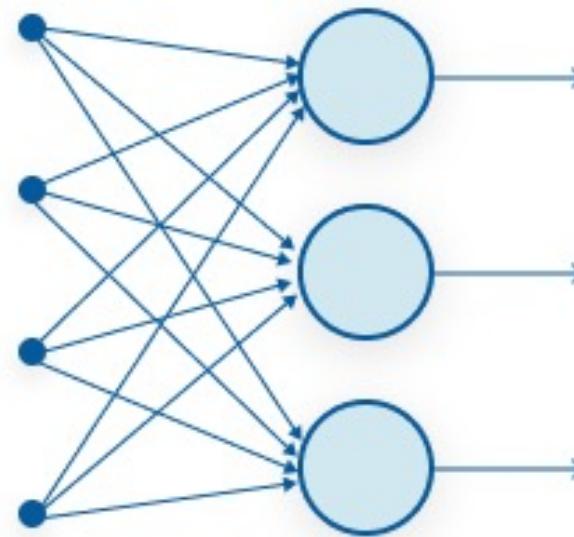


# Recurrent Neural Networks

---

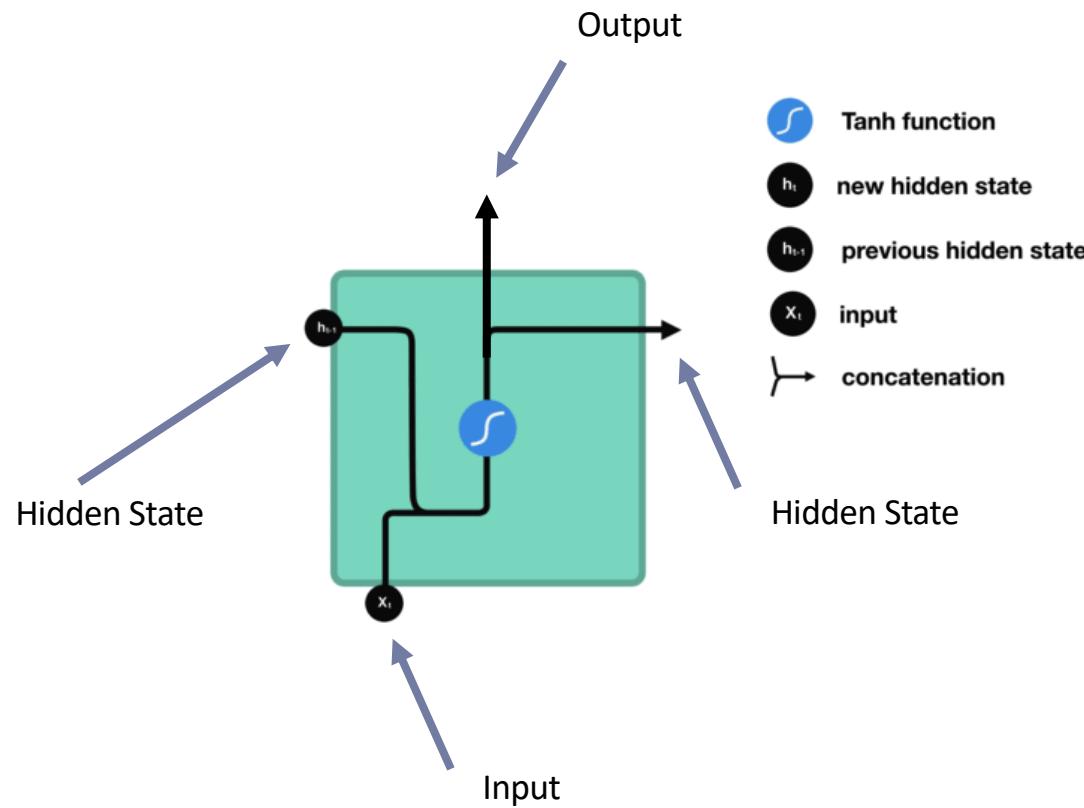


Recurrent Neural Network



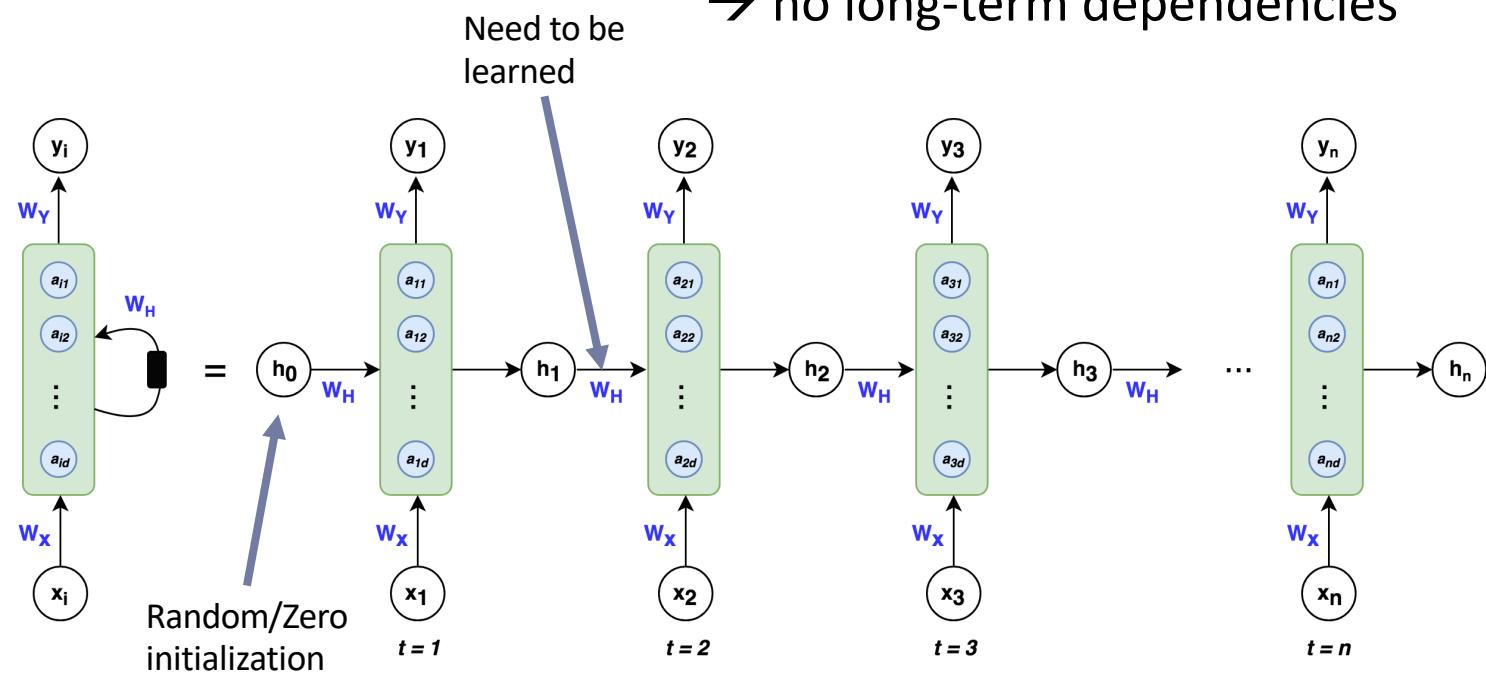
Feed-Forward Neural Network

# Recurrent Neural Networks: Intuition



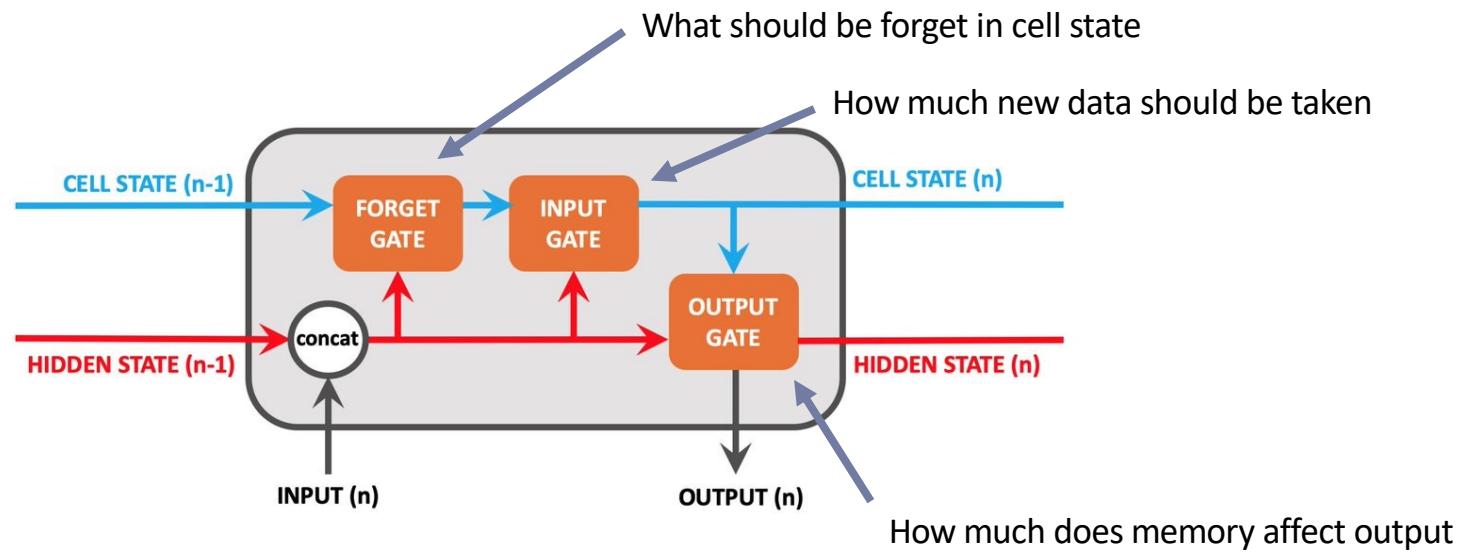
# Recurrent Neural Networks

- State is saved in hidden vector  $h \rightarrow$  last step is preserved  
 $\rightarrow$  no long-term dependencies



# RNN 2.0: Long short-term Memory (LSTM)

- Hidden State: holds previous information (Short-term memory)
- Cell State: memory of the network (Long-term memory)
- Vanishing gradient problem: the longer the dependencies the worse



# Self-Attention

- ▶ Convolution applies fixed transform weights
- ▶ Attention layers access previous states and weight them according to a learned measure of relevance
- ▶ Self-attention applies variable weights (but typically not transformations):

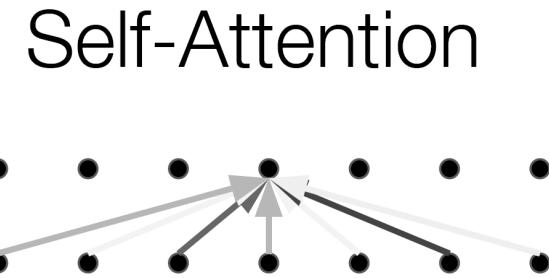
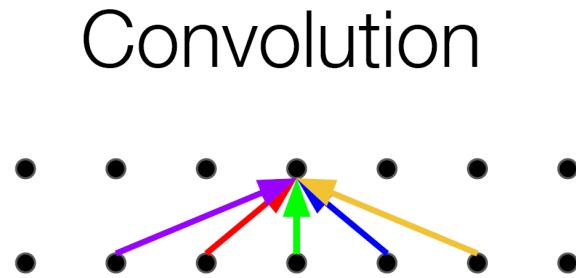
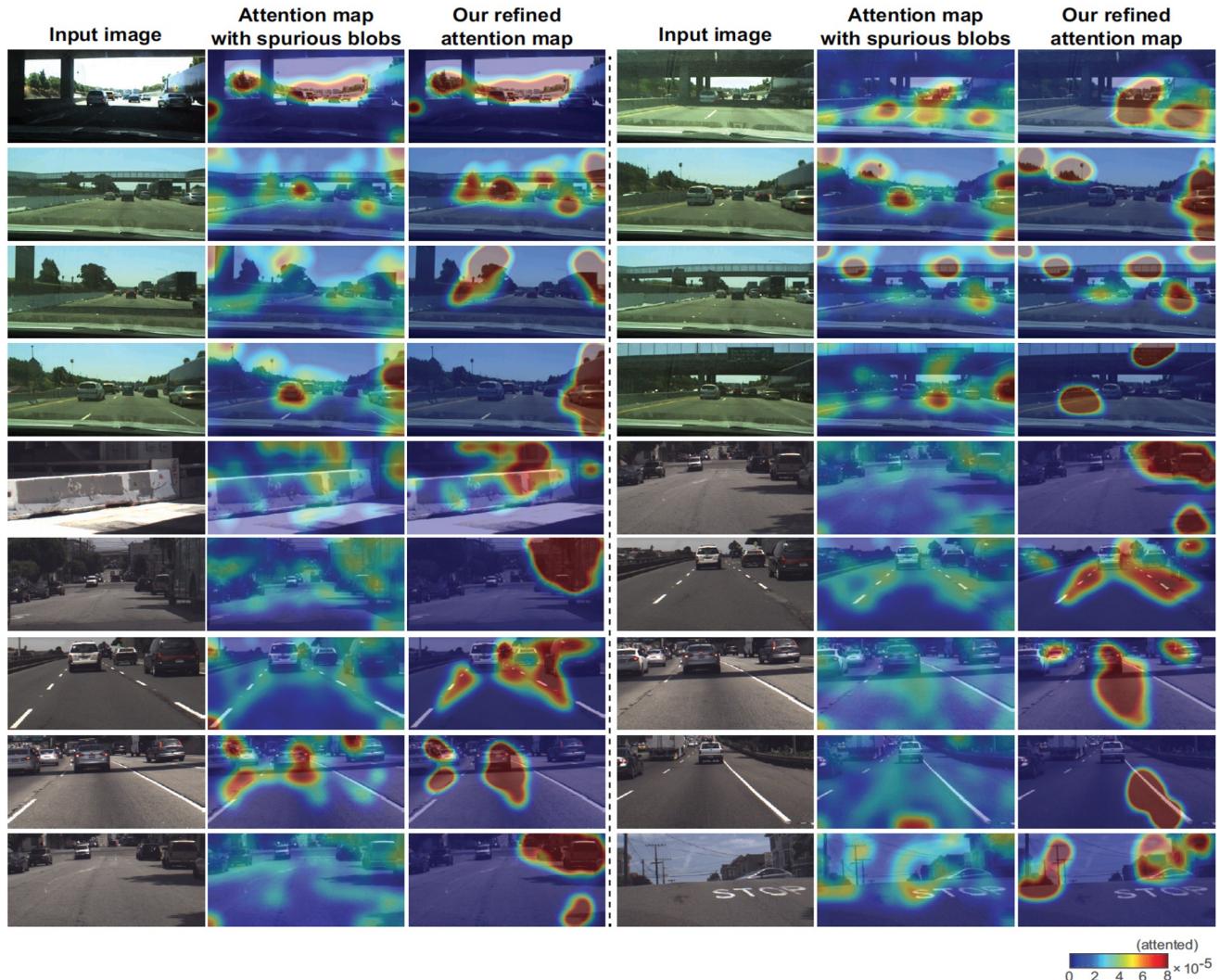


image from Lukas Kaiser, Stanford NLP seminar

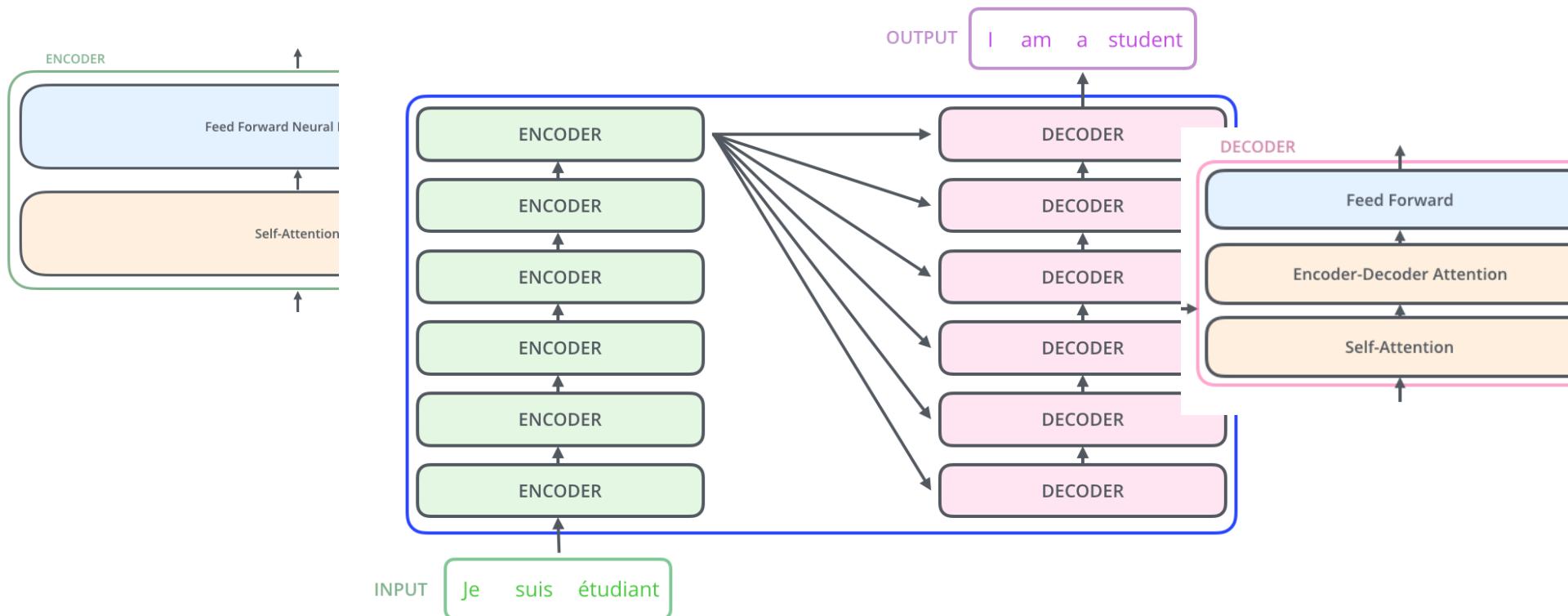
# Attention

- ▶ Attention visualizations
- ▶ Attention models learn to predict salient (important) inputs
- ▶ Not every attended region is actually important



# Transformer: Structure

- ▶ Transformer architecture
  - ▶ self-attention (weighing inputs)
  - ▶ Unlike RNNs or CNNs, process all inputs at once, not sequentially



# Self-Attention “Transformers”

---

- ▶ Variable receptive field (or the whole input sequence)
- ▶ Supports hierarchical information flow by stacking self-attention layers
- ▶ Trivial to parallelize
- ▶ Attention weighting controls information propagation
- ▶ “Attention is all you need”
  - ▶ Replaces convolutions and recurrences
    - ▶ Take with grain of salt!
- ▶ Transformers are highly complex models
  - ▶ Similar trade-off as before: powerful, but extremely challenging to get right
    - ▶ Hyperparameters, training

# Breathe deep

---



# Explainable AI (XAI)

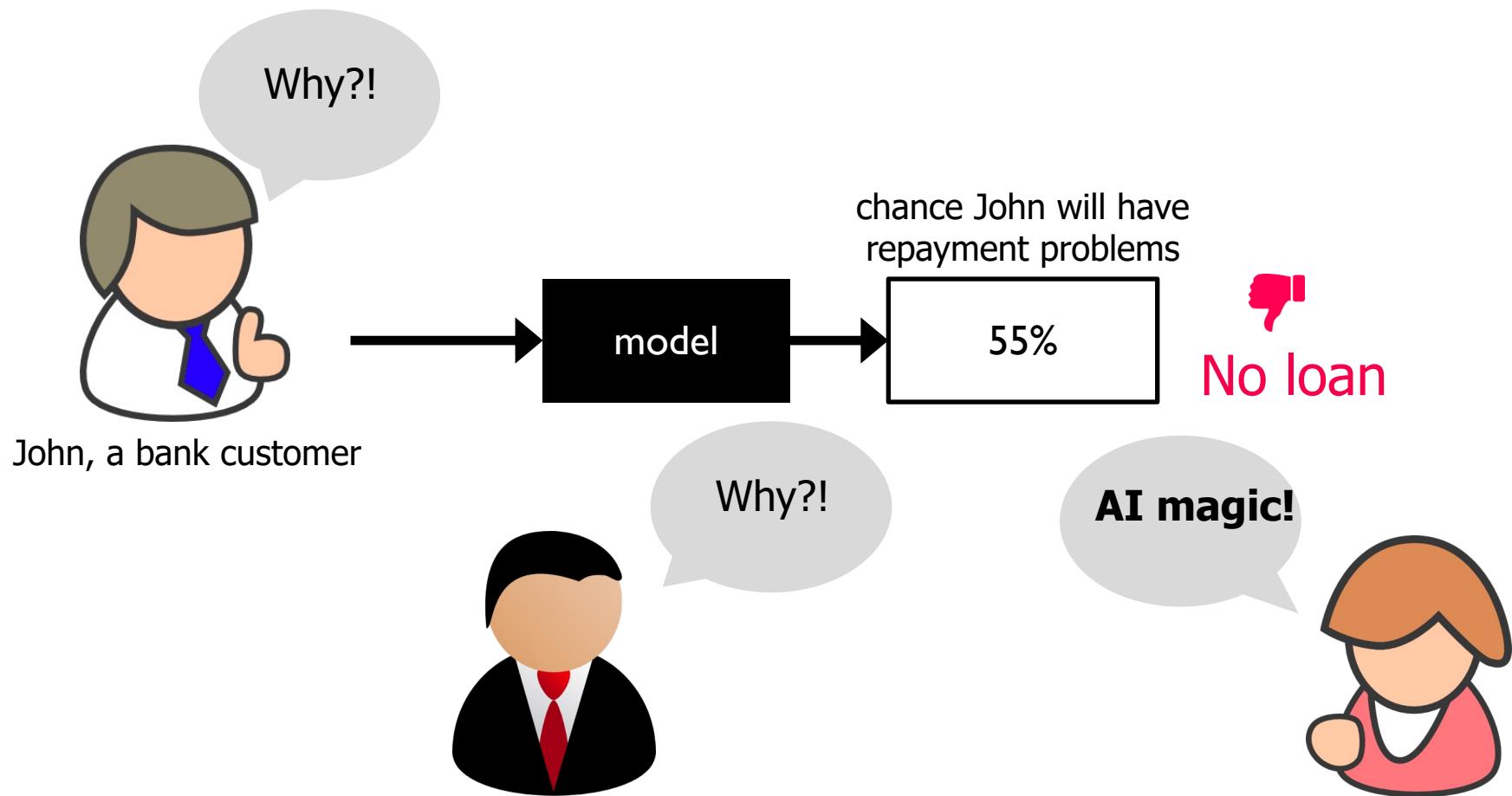
---

- What has the network actually learned?
  - We don't quite know yet...
  - Humans cannot make sense of thousands or millions of weights
- Need a way to inspect models
  - Fairness, accountability, transparency (FAccT)
  - Build trust, ensure reliability, validity
- EU “right to explanation”
  - Provide logics underlying models and algorithms
  - For example: why do I not get the bank loan?



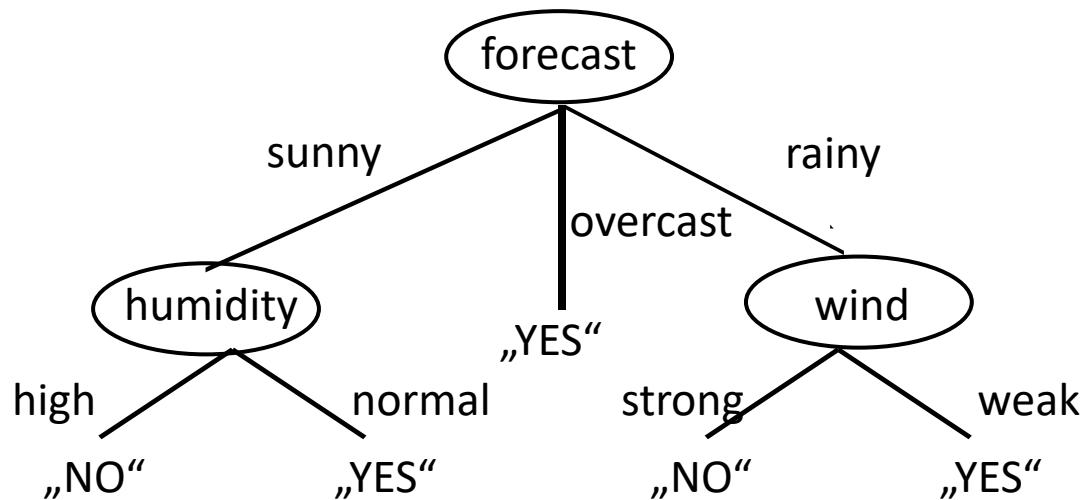
# XAI motivation

---



# Interpretable models

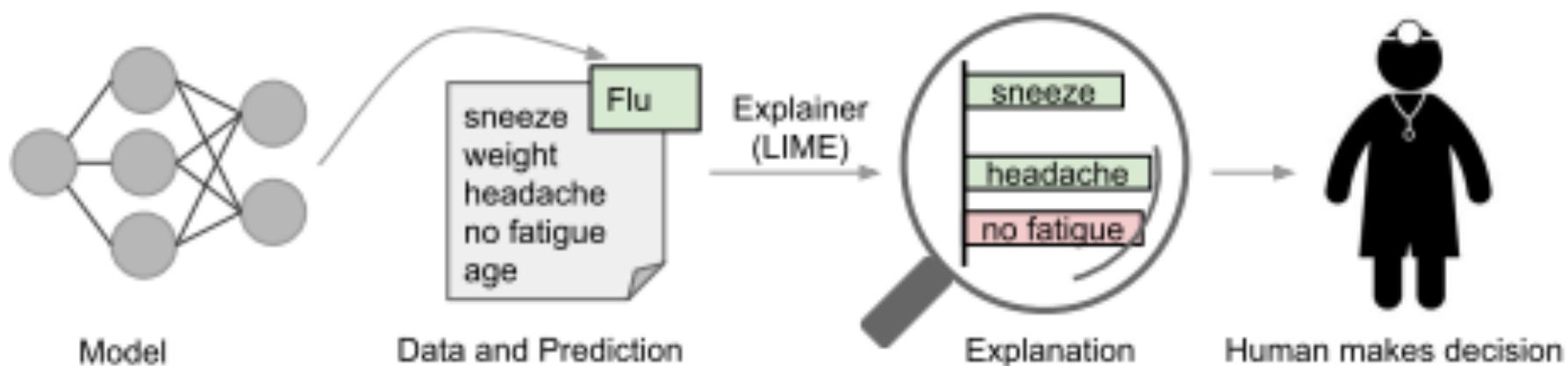
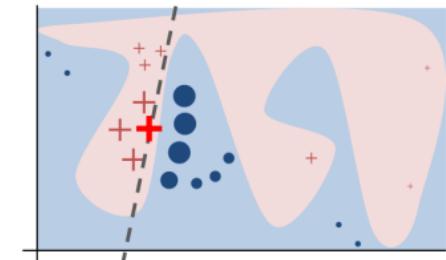
- ▶ Learn interpretable model
- ▶ E.g. Decision Tree
  - ▶ Either from scratch
  - ▶ Or, as approximation of more complex model
- ▶ Easy to inspect
- ▶ Less powerful than e.g. deep neural networks



# LIME – approximation via interpretable model

- Ribeiro, M. T., Singh, S., & Guestrin, C. Why should I trust you?: Explaining the predictions of any classifier. ACM SIGKDD 2016.

- Explanation by approximation via an interpretable model (like decision tree)
  - not for e.g. full deep neural network
  - but for a particular prediction

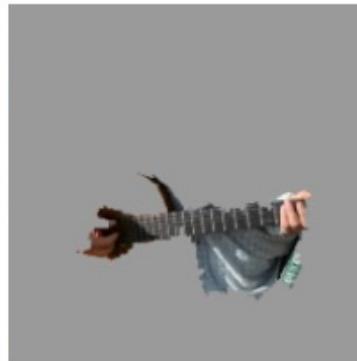


# LIME – Example

- ▶ Images: highlight parts important for a predicted class
- ▶ Text: importance given to different words



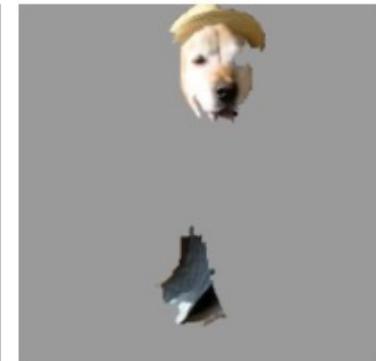
(a) Original Image



(b) Explaining *Electric guitar*



(c) Explaining *Acoustic guitar*



(d) Explaining *Labrador*

Figure 4: Explaining an image classification prediction made by Google's Inception neural network. The top 3 classes predicted are “Electric Guitar” ( $p = 0.32$ ), “Acoustic guitar” ( $p = 0.24$ ) and “Labrador” ( $p = 0.21$ )

# Any issues with the approximate model approach?

---



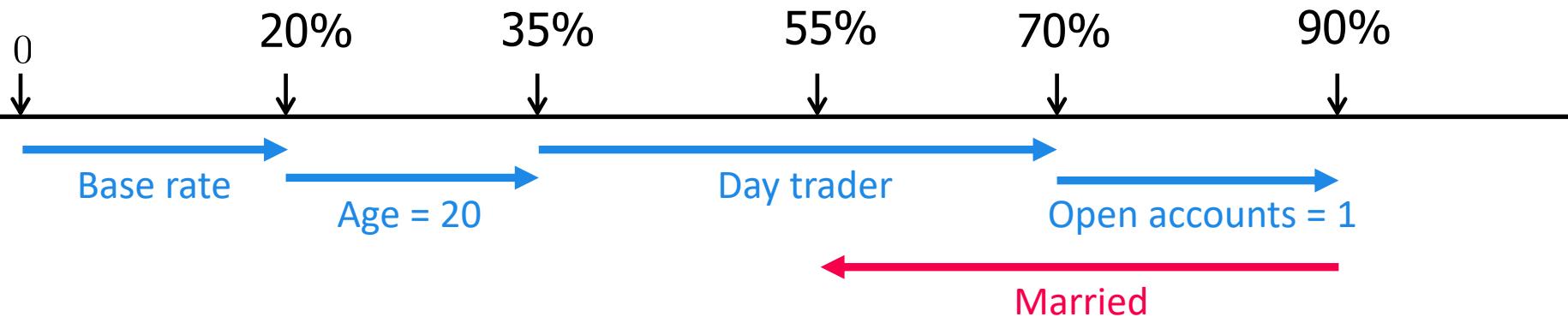
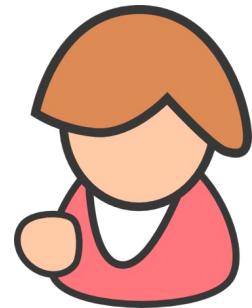
# Feature impact on a prediction

For a given prediction

- E.g. John does NOT get the loan

Compute the impact that different feature values have

(here: SHapley Additive exPlanation (SHAP) values)



# Contrastive Sample

---

- Given a prediction
  - e.g. email is classified as spam
- find a similar example that is classified as ham
- Provides a contrast to understand differences



Feature	freq_now	freq_credit	freq_!!!	freq_!	class
$x_1$	0.1	0.0	0.0	0.0	Ham
$\tilde{x}_1$	0.1	0.0	0.3	0.453	Spam

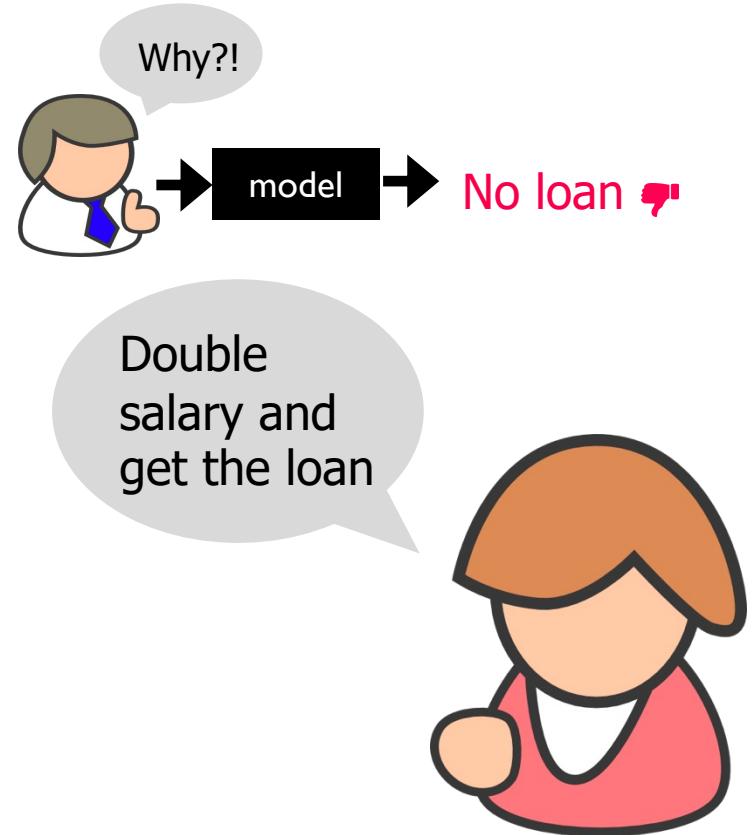
Feature	freq_you	freq_direct	avg_longest_capital	class
$x_2$	0.68	0.34	158.0	Spam
$\tilde{x}_2$	0.68	0.34	1.0	Ham



# Counterfactual Explanations

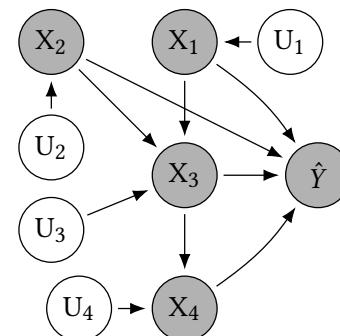
---

- Goal: help affected individuals
  - Given a prediction
  - **How to change input**
- Counterfactuals
  - How the world would have to be different for a desirable outcome
  - Ongoing research: how to ensure results are **actionable** (algorithmic recourse)



# Algorithmic Recourse

- ▶ German Credit data
- ▶ Model for actions and their impact on variables
  - ▶  $X_1$  gender(immutable),  $X_2$  age (actionable, only increase),  $X_3$  credit given by bank (actionable),  $X_4$  repayment duration of credit (non-actionable but mutable),
  - ▶  $\hat{Y}$  predicted customer risk, according to  $h$  (logistic regression or decision tree)
- ▶ Customer Male, 32, \$1938, 24 , no loan
  - ▶ Action set  $X_2 := X_2 + 1$ ,  $X_3 := X_3 - \$800$ 
    - ▶ Counterfactual Male, 33, \$1138, 22, loan
  - ▶ Action set N/A, +6, 0, 0
    - ▶ Male, 38, \$1938, 24, loan
  - ▶ Waiting 6 years to get credit more costly than 1 year with lower (-\$800) credit?



$$\left. \begin{array}{l} X_1 := U_1 \\ X_2 := U_2 \\ X_3 := f_3(X_1, X_2) + U_3 \\ X_4 := f_4(X_3) + U_4 \\ \hat{Y} = h(\{X_i\}_{i=1}^4) \end{array} \right\} M$$

# Surprising explanations

- ▶ Well-known, engineered example:
  - ▶ Train machine learning model to distinguish huskys and wolves
  - ▶ Model performs well
- ▶ Explanation shows: it learns to tell snow from no snow



(a) Husky classified as wolf



(b) Explanation

# More surprises

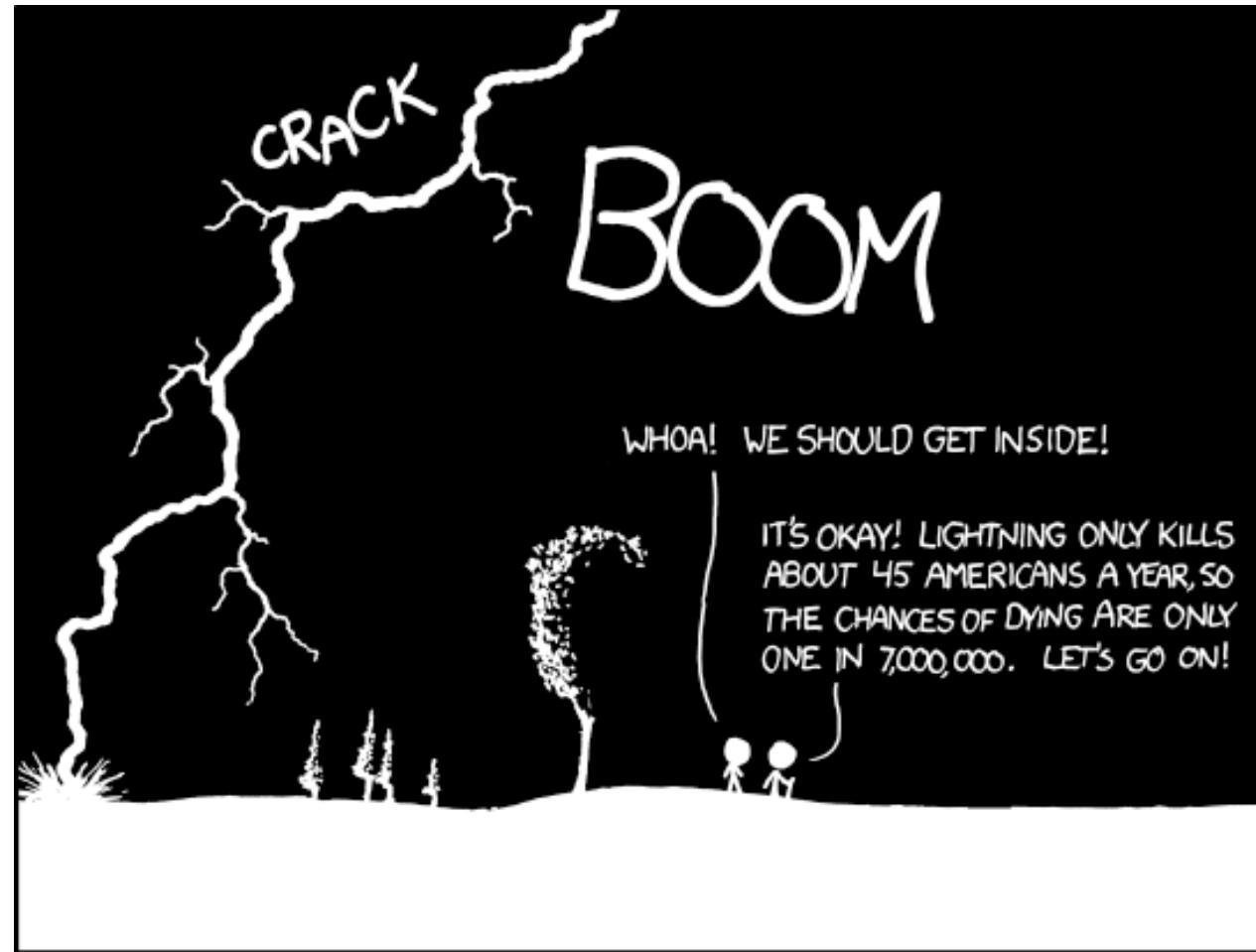
---

- ▶ 90s, US Cost Effective Health Care (CEHC)
  - ▶ neural network to predict mortality in pneumonia patients,
    - ▶ performs well
  - ▶ Rule based explanations (approximation) prior to deployment shows
    - Has Asthma → Low Risk
      - ▶ Contradictory to medical knowledge
      - ▶ Turned out to lack data that asthma patients receive intensive care
- ▶ **Explanations are important!**



# Data updates

- Assume distributional stability → often a poor assumption.
- Goodhart's Law: "any observed statistical regularity will tend to collapse once pressure is placed upon it for control purposes"
- Regularities might collapse due to concept drift, feedback loops, adversarial data, disruptive changes of the environment



Goodhart, C. A. E. (1984). Problems of Monetary Management: The UK Experience. In C. A. E. Goodhart (Ed.), *Monetary Theory and Practice: The UK Experience* (pp. 91-121). London: Macmillan Education UK.

# Golden Rules

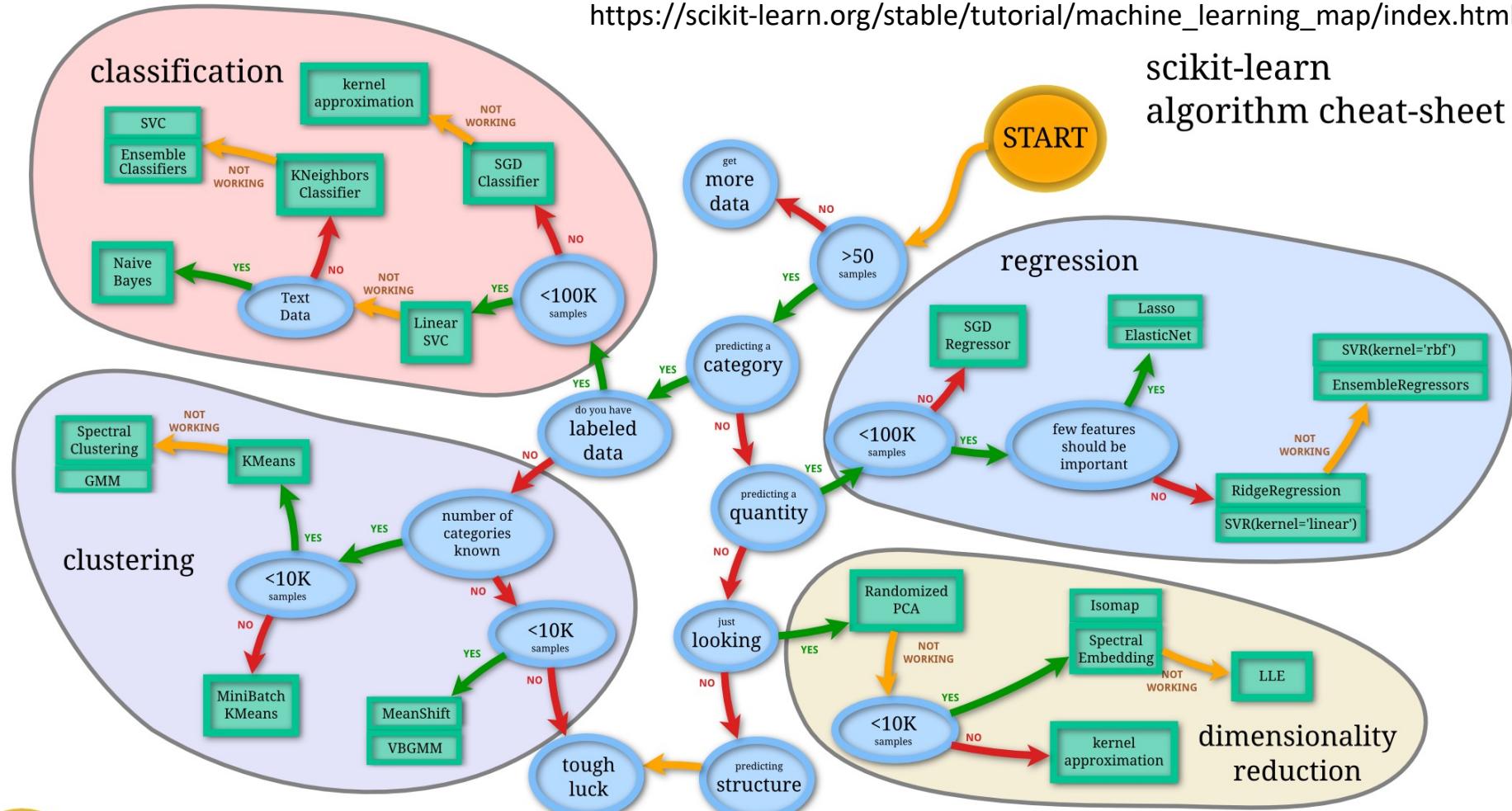
- ▶ Check your data quality / volume
  - ▶ Data pre-processing to avoid “Garbage in, garbage out”
- ▶ Choice of model
  - ▶ Why are you choosing a particular model
    - ▶ Make sure you understand strengths and limitations of your choice
- ▶ How to evaluate
  - ▶ Core to proper assessment, basis of your conclusions
  - ▶ Ensure balanced data and/or assessment
  - ▶ Use more than one measure
  - ▶ Understand what they tell you
- ▶ Any model will provide some output – but can you trust it? You would hate to build your research career on some random artefact!
  - ▶ There are many good resources online, but not everyone is equally qualified
  - ▶ Remember the human choices that go into data science
  - ▶ If unsure, consider reaching out to a data scientist



# Scikit-learn

[https://scikit-learn.org/stable/tutorial/machine\\_learning\\_map/index.html](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)

## scikit-learn algorithm cheat-sheet



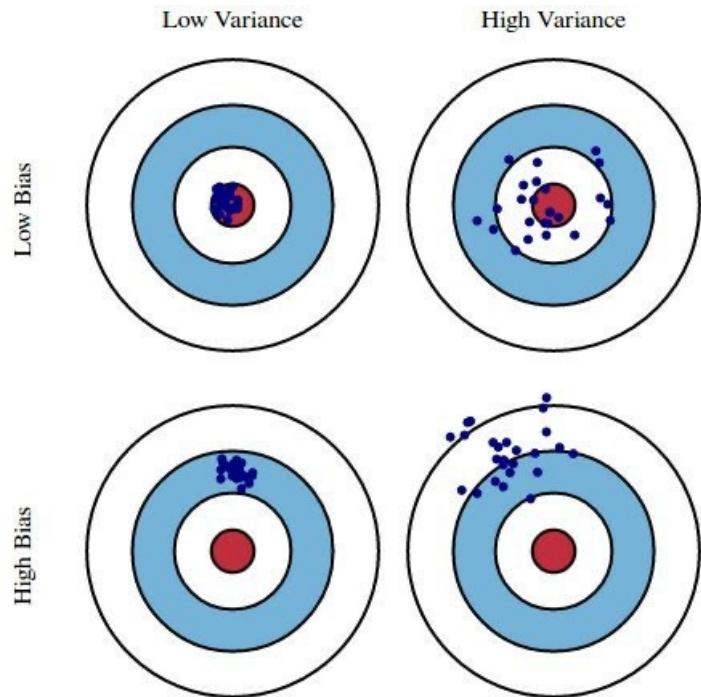
# Breathe deep

---



# A bit of background on overfitting and underfitting

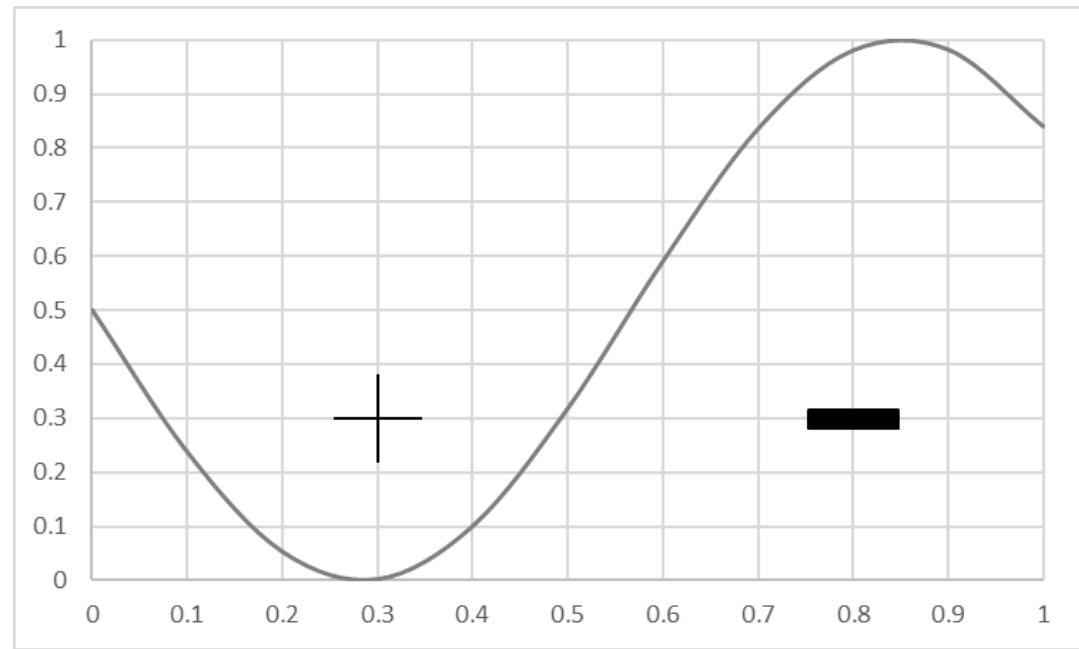
- ▶ Learning theory – a sketch with a focus on model choice and training
- ▶ The error of a machine learning model is caused by two factors:
  - ▶  $\text{TotalLoss} = \text{Bias} + \text{Variance} (+ \text{noise})$
  - ▶ Bias – error caused because the model cannot represent the concept
  - ▶ Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data



source:  
<https://pbs.twimg.com/media/CpWDWuSW8AQUuCk.jpg>

# Visualizing Bias: the learning goal setup

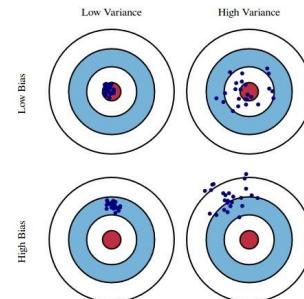
- ▶ Goal: produce a model that matches this concept



True Concept

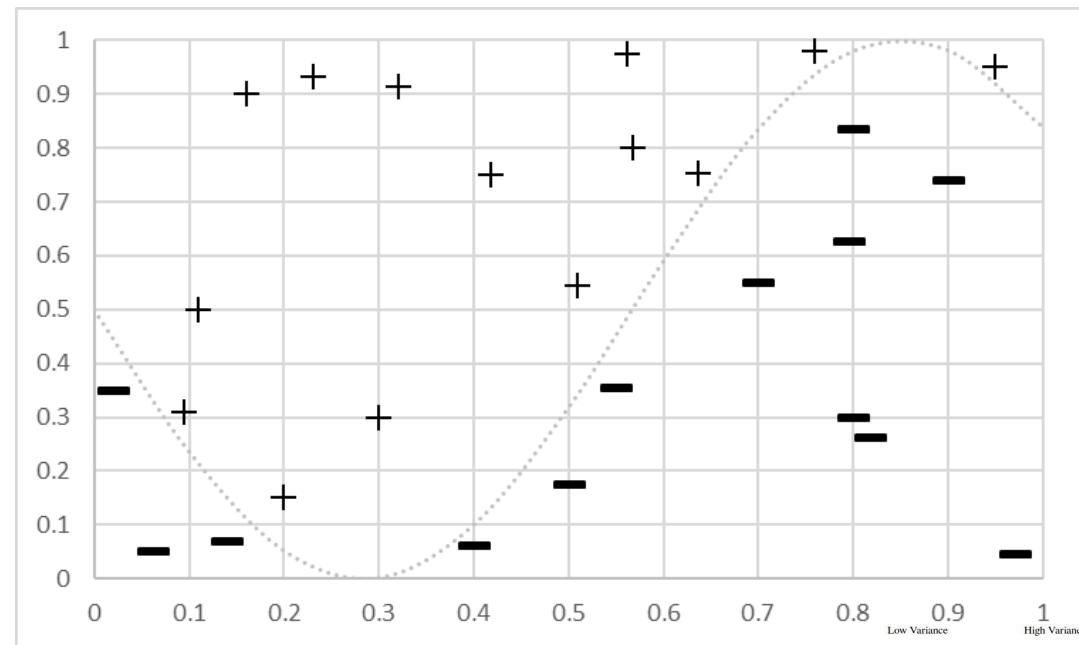
Bias – error caused because the model cannot represent the concept

Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data



# Visualizing Bias: the training data

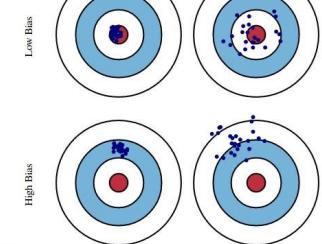
- ▶ Goal: produce a model that matches this concept
- ▶ Training Data for the concept



Bias – error caused because the model cannot represent the concept

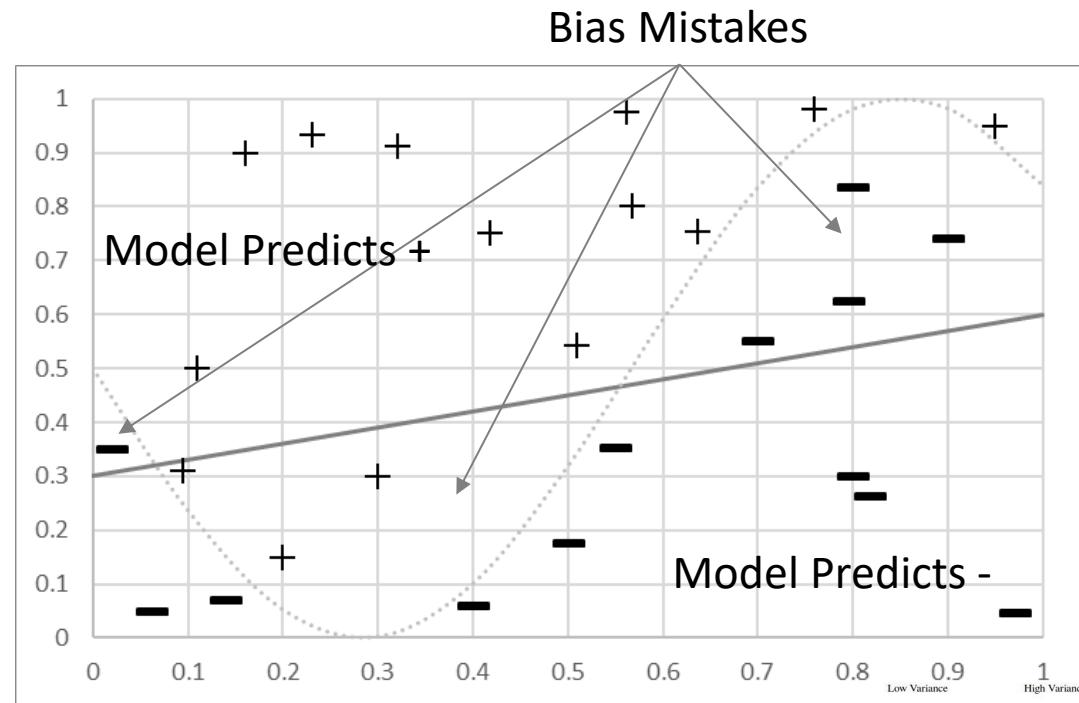
Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data

Training Data



# Visualizing Bias: learning a linear model

- ▶ Goal: produce a model that matches this concept
- ▶ Training Data for concept
- ▶ Bias: Can't represent it...

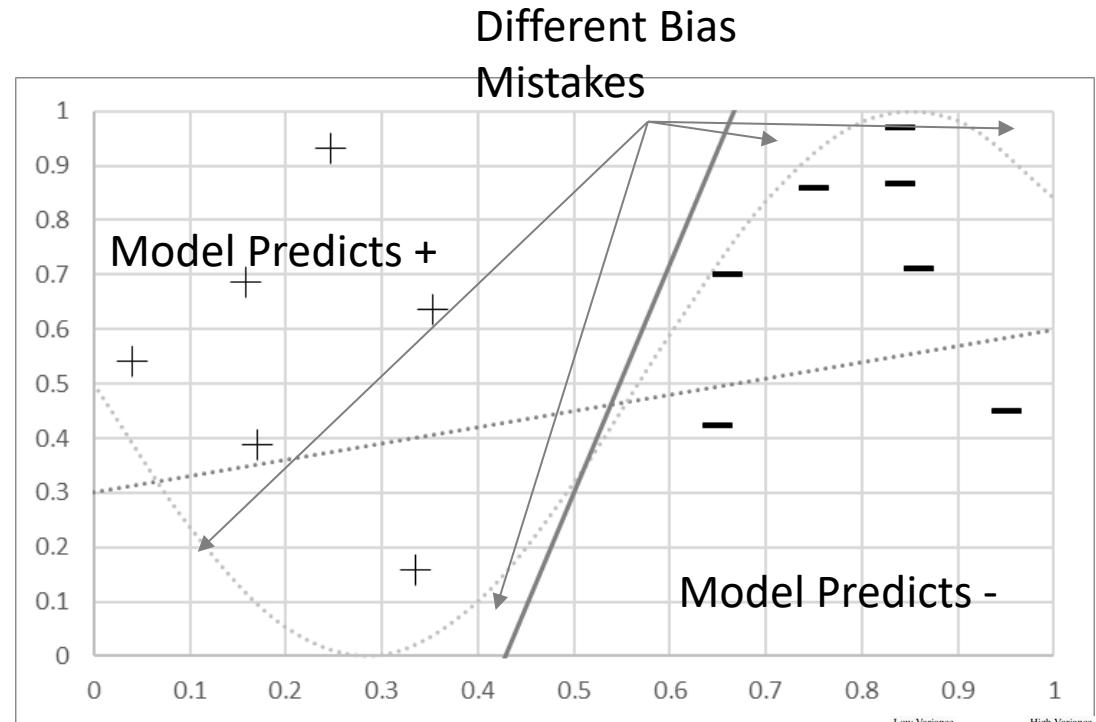


Bias – error caused because the model cannot represent the concept

Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data

# Visualizing Variance: same setup, different sample

- ▶ Goal: produce a model that matches this concept
- ▶ New data, new model



Bias – error caused because the model cannot represent the concept

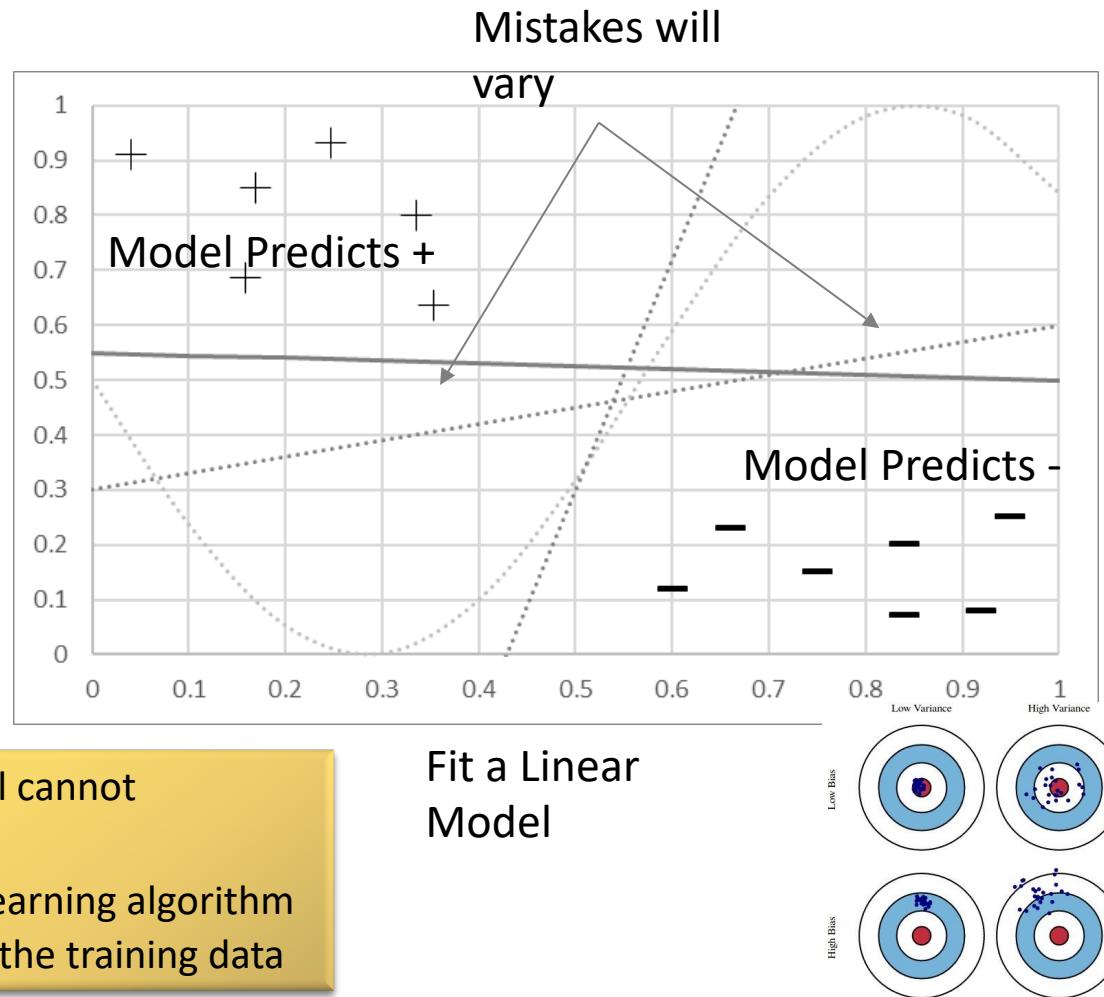
Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data

# Visualizing Variance: same setup, yet another sample

- ▶ Goal: produce a model that matches this concept
- ▶ New data, new model
- ▶ New data, new model...
- ▶ Variance: Sensitivity to changes & noise

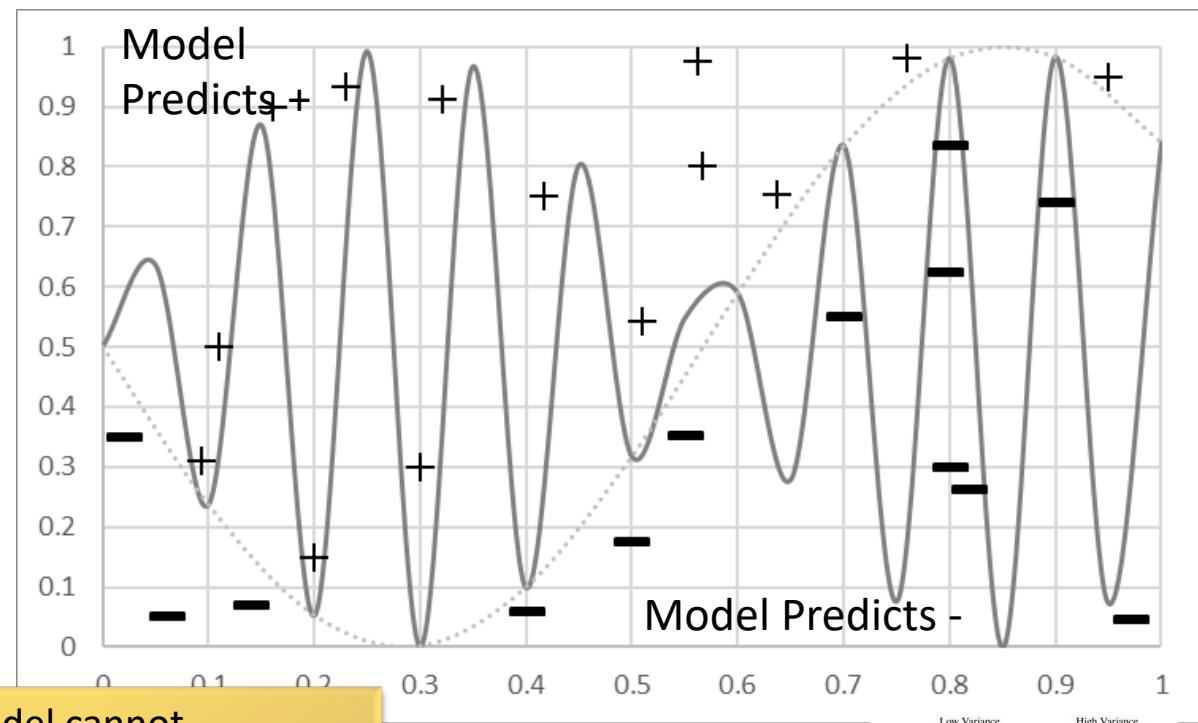
Bias – error caused because the model cannot represent the concept

Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data



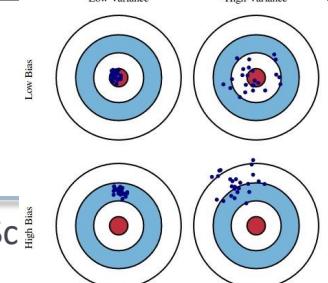
# Bias and Variance: More Powerful Model

- ▶ Powerful Models can represent complex concepts
- ▶ No Mistakes!



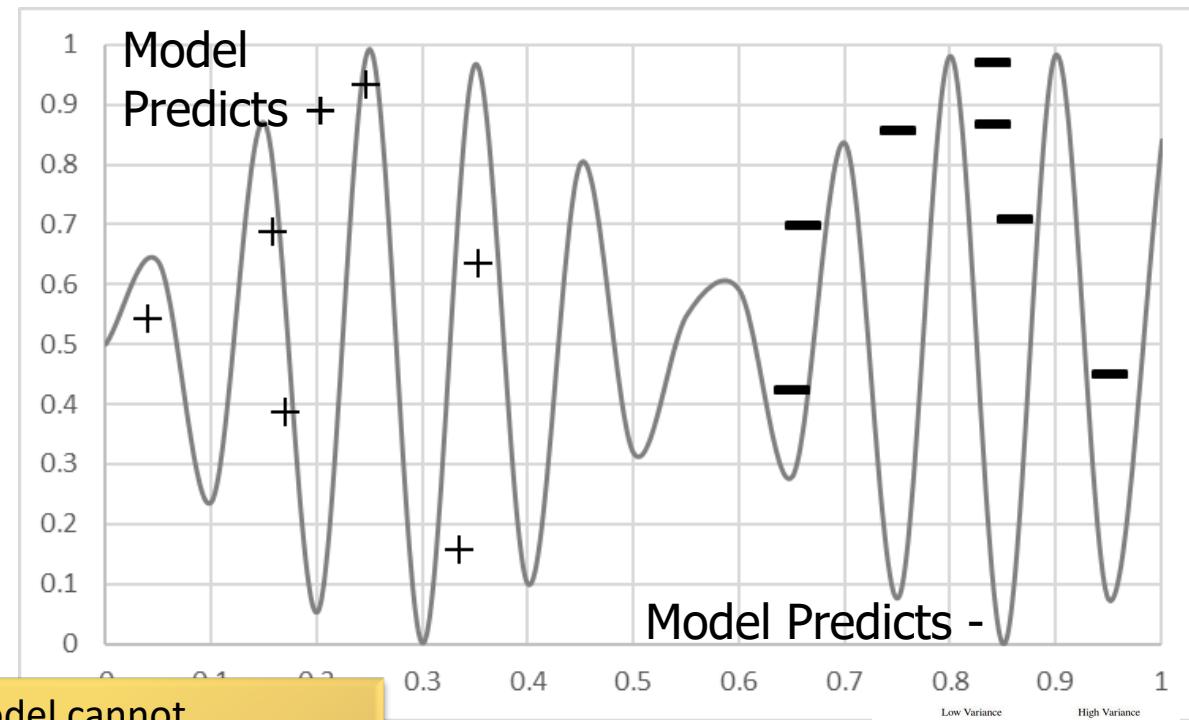
Bias – error caused because the model cannot represent the concept

Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data



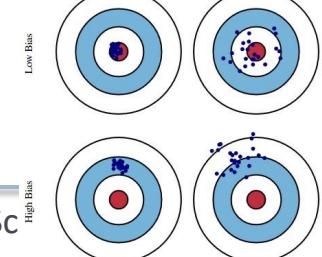
# Bias and Variance: More Powerful Model

- ▶ But get more data...
- ▶ Not good!
- ▶ Issues with generalization
  - ▶ Too specific learning



Bias – error caused because the model cannot represent the concept

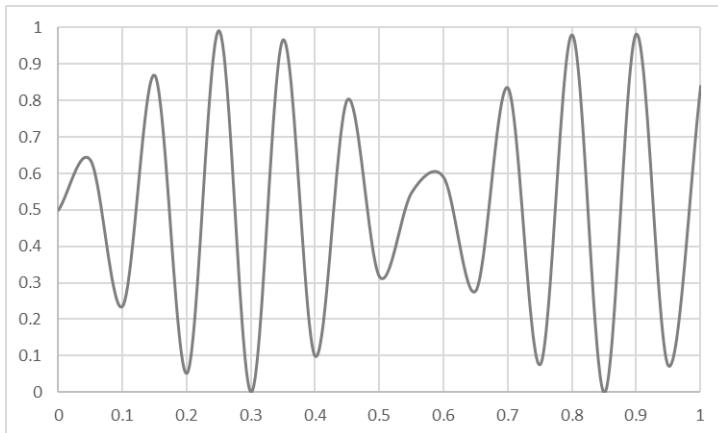
Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data



# Overfitting vs Underfitting

## Overfitting

- ▶ Fitting the data too well
  - ▶ Features are noisy / uncorrelated to concept
  - ▶ Modeling process very sensitive (powerful)
  - ▶ Too much search

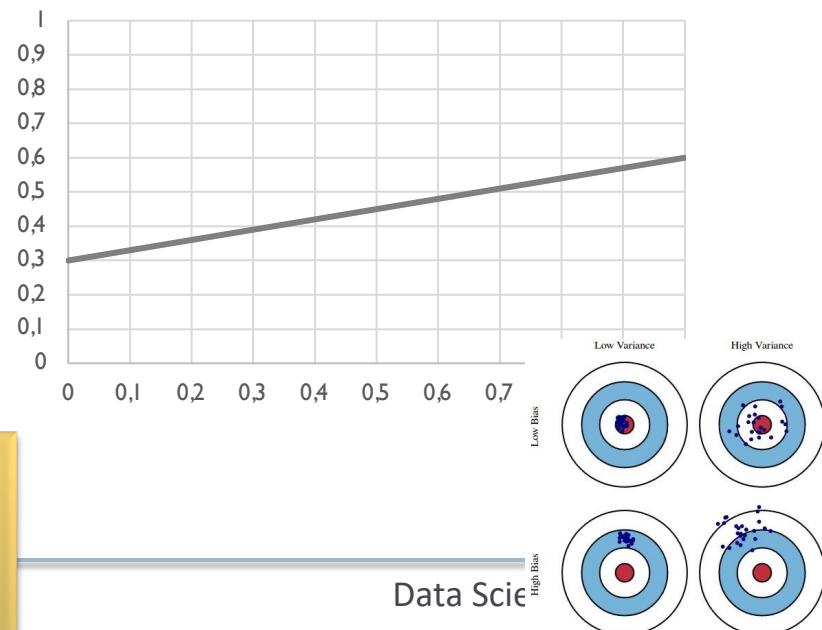


Bias – error caused because the model cannot represent the concept

Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data

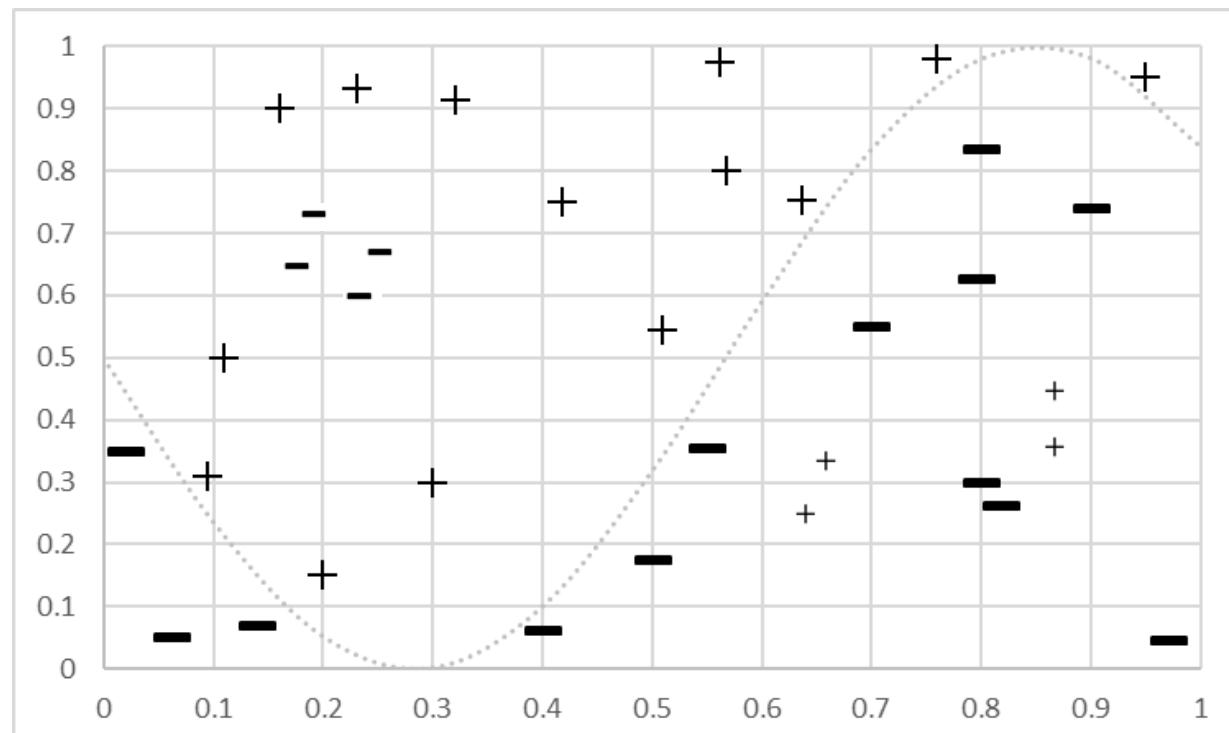
## Underfitting

- ▶ Learning too little of the true concept
  - ▶ Features don't capture concept
  - ▶ Too much bias in model
  - ▶ Too little search to fit model



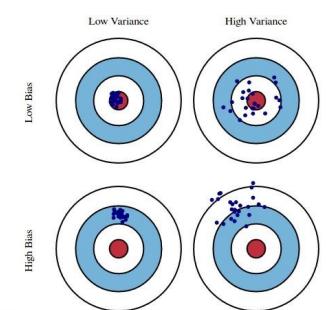
# The Effect of Noise

- ▶ “Uniform” noise usually not an issue
  - ▶ Impacts model learning more evenly
- ▶ Clustered noise can throw off the learning and impact result substantially

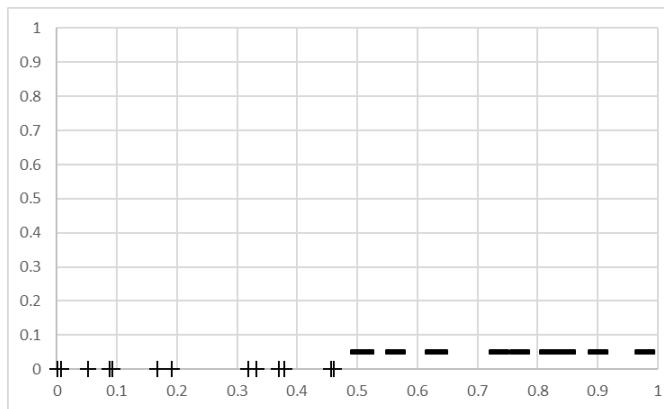
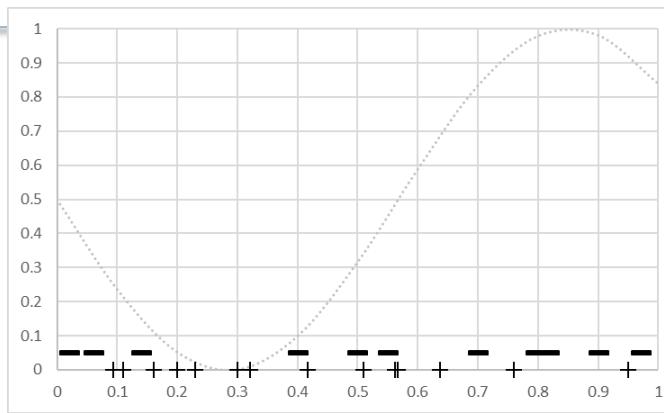
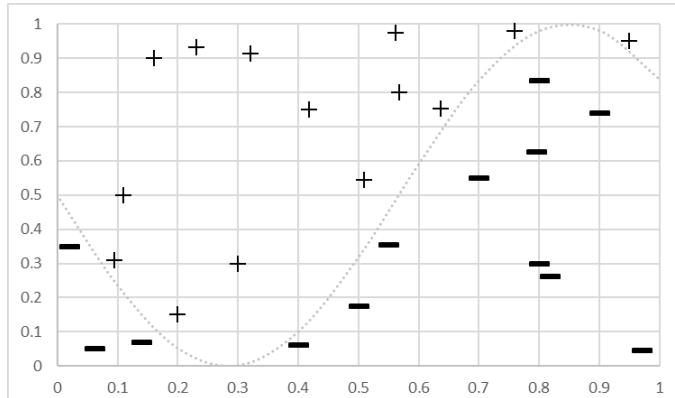


Bias – error caused because the model cannot represent the concept

Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data



# The Effect of Features

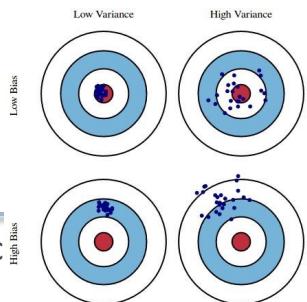


Bias – error caused because the model cannot represent the concept

Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data

- Not much info
- Won't learn well
- Powerful -> high variance

- Captures concept
- Simple model -> low bias
- Powerful -> low variance



# The Power of a Model Building Process

## Weaker Modeling Process ( higher bias )

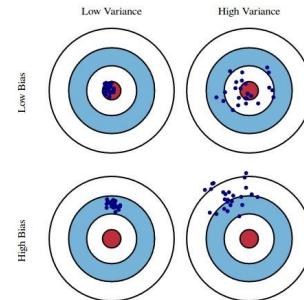
- ▶ Simple Model (e.g. linear)
- ▶ Fixed sized Model (e.g. fixed # weights)
- ▶ Small Feature Set (e.g. top 10 tokens)
- ▶ Constrained Search (e.g. few iterations of gradient descent)

Bias – error caused because the model cannot represent the concept

Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data

## More Powerful Modeling Process (higher variance)

- ▶ Complex Model (e.g. high order polynomial)
- ▶ Scalable Model (e.g. decision tree)
- ▶ Large Feature Set (e.g. every token in data)
- ▶ Unconstrained Search (e.g. exhaustive search)

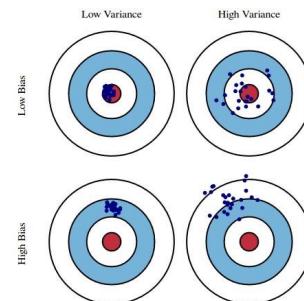


# Summary of Overfitting and Underfitting

- ▶ Bias / Variance tradeoff a primary challenge in machine learning
- ▶ Internalize: More powerful modeling is not always better
- ▶ Learn to identify overfitting and underfitting
- ▶ Tuning parameters & interpreting output correctly is key

Bias – error caused because the model cannot represent the concept

Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data



# Summary

---

- ▶ Neural networks
  - ▶ Machine learning model inspired by brain
  - ▶ If large enough model, enough data and enough training time can learn arbitrarily complex functions
    - ▶ But, hyperparameter tuning is a challenge
    - ▶ Overfitting is a constant threat
    - ▶ Choose your architectures wisely ("bias" model in the right way)
    - ▶ Train, validate, test – separately!!
- ▶ Common models
  - ▶ CNNs, RNNs, Transformers
- ▶ XAI
  - ▶ Complexity of the models is an issue for transparency, validation, trust
  - ▶ Explainability: provide insight into complex models
  - ▶ Approximate with interpretable models, find counterfactual examples, compute feature weights
  - ▶ Make sure to check if an interpretable model suffices
    - ▶ Think underfitting / overfitting – bias / variance

# References

---

- ▶ <https://jakevdp.github.io/PythonDataScienceHandbook/> Data Science in Python
- ▶ <https://wesmckinney.com/book/> Python for Data Science
- ▶ <https://scikit-learn.org> Scikit-learn
- ▶ <https://pandas.pydata.org/> Pandas
- ▶ [https://dataminingbook.info/book\\_html/](https://dataminingbook.info/book_html/) Mohammed J. Zaki, Wagner Meira, Jr., Data Mining and Machine Learning: Fundamental Concepts and Algorithms, 2nd Edition, Cambridge University Press, March 2020. ISBN: 978-1108473989.
- ▶ [https://scikit-learn.org/stable/tutorial/machine\\_learning\\_map/index.html](https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html)
- ▶ Kasper Green Larsen, Machine Learning course slides, Aarhus University
- ▶ Ming Li CS 886 Deep Learning for Biotechnology, U Waterloo, CA
- ▶ Vaswani et al. Attention is all you need. 2017.
- ▶ <https://nlp.seas.harvard.edu/2018/04/03/attention.html> (Excellent explanation of transformer model with codes)
- ▶ Jay Alammar, The illustrated transformer (source of many pictures used here)  
<http://jalammar.github.io/illustrated-transformer/>
- ▶ Machine Perception Lecture by O. Hilliges
- ▶ Advanced Machine Learning by J. Buhmann
- ▶ Jonas Bokstaller, ETHZ, Introduction to Deep Learning
- ▶ John Canny, Berkeley, Designing, Visualizing and Understanding Deep Neural Networks
- ▶ Geoff Hulten, University of Washington, Overfitting and Underfitting