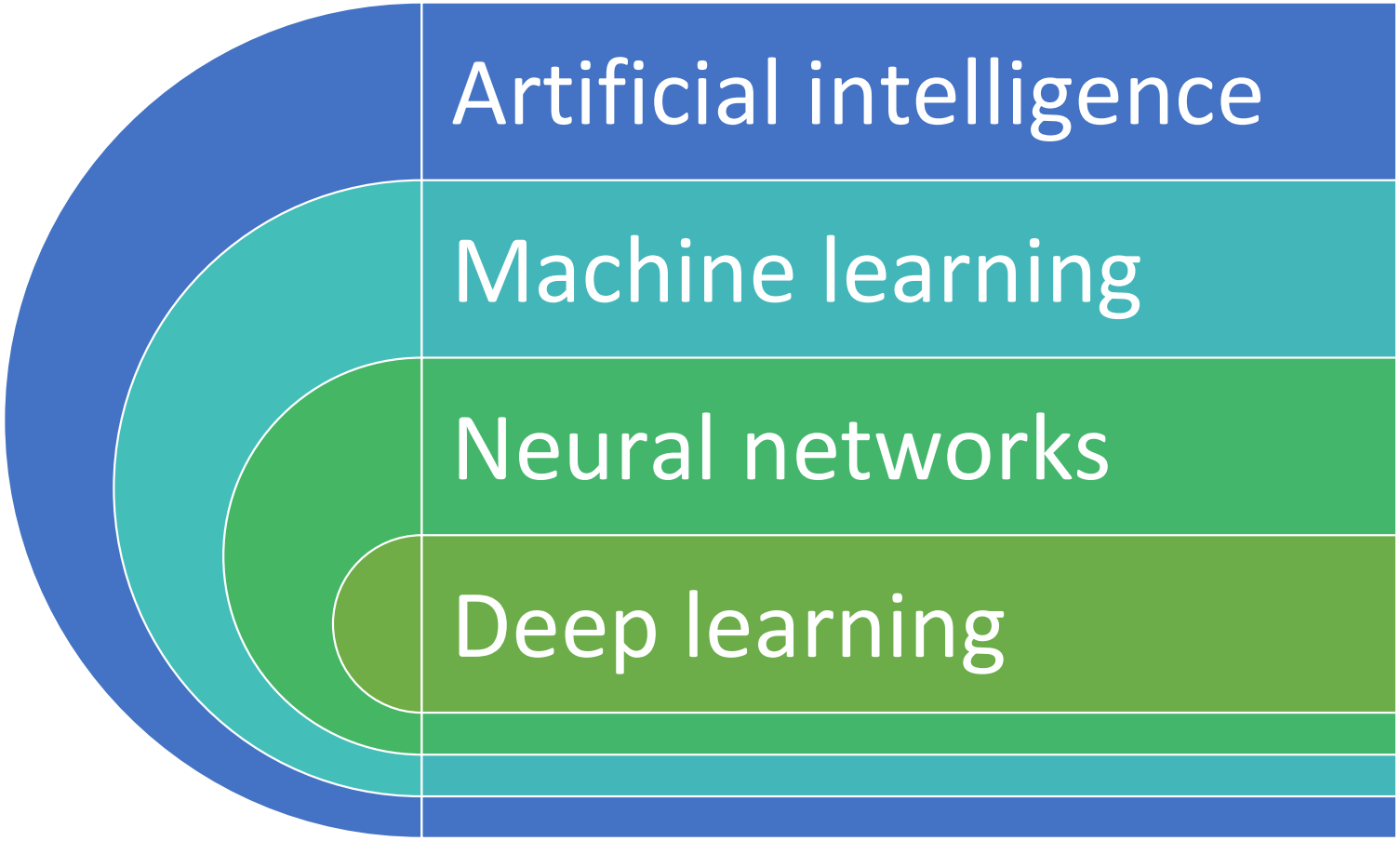
Introduction to Advanved Topics in Machine Learning

EE 782 Advanved Topics in Machine Learning
July-Nov 2025
Amit Sethi, EE, IITB
Contact: asethi@iitb, 3528, 7483

Why this course

- Rapid expansion of data and compute
  - Memory and data transfer has become cheaper
  - Multi-modal data being collected and shared
  - Compute has become cheaper
- Sudden explosion in the use of ML
  - Face recognition
  - Autonomous driving
  - Text and image generation
- Rapid pace of research and development

What is the relation between AI, ML, DL etc.?



AML will be synomyous with deep learning

- Advanced machine learning is very broad
  - Deep learning
  - Probabilistic methods
  - Theoretical ML
  - Reinforcement learning
  - ...
- Most practical and exciting applications are coming from DL

- More than enough to be covered in deep learning alone

Learning outcomes

- Formulate advanced machine learning problems
  - Inputs, outputs, labels, annotations, data quantity, data quality, prior knowledge
- Analyze and propose neural architectures
  - How are data dimensions related
  - Changes in architectures to improve outcomes
- Analyze and formulate training methods
  - Amount and quality of data
  - Using prior knowledge

What will be covered

- ML problems
  - Lack of labels
  - Mislabeled data
  - Use of prior knowledge
- Neural network architectures
  - For different types of input data
  - For different formats of output
  - For different levels of compute
- Training methods
  - Loss functions
  - Speed of convergence
  - Pre-training and fine-tuning
  - Robustness

Course material

- Textbooks:
  - “Dive into Deep Learning” by Aston Zhang, Zachary C. Lipton, Mu Li, and Alex Smola
  - “Deep Learning” by Ian Goodfellow, Yoshua Bengio, and Aaron Courville

- Papers:
  - Check out reading list on Moodle

Pre-requisites

- Introduction to ML
  - Supervised learning: regression, classification
  - Unsupervised learning: clustering, dimension reduction
- Linear algebra
  - Matrix and vector arithmetic
  - Subspaces, eigen decomposition etc.
- Probability
  - Joint distributions: marginals, conditionals
  - KL divergence etc.

Evaluation structure

- Daily notes upload (20 x 0.5 = 10 marks)
- Programming assignments (8 x 3 = 24 marks)
- Mid-sem exam (20 marks)
- End-sem exam (30 marks)
- Project proposal (2 marks)
- Project (14 marks)
  - (IEEE style self published paper on ArXiv, Github repo, video demo)
- Absolute grading (90+ AA, 80+ BB etc.)
  - Audit: 30+

Agenda

- **Overview of machine learning**
- Revision of neural networks
- Tentative list of topics in AML

ML is…

- The practice of automating the use of related data to estimate models that make useful predictions about new data, where the model is too complex for standard statistical analysis, e.g.
  - Improve accuracy of classification of images using labeled images
  - Improve win percentage on alpha-go using several simulated game move sequences and their results
  - Improve the Turing test confusion between human and machine for NLP Q&A using a large sample of text including Q&A

When not to use ML

- Possible inputs are countable and few
  - Use look up tables
- Algorithm is well-known and efficient
  - E.g. sorting, Dijkstra's shortest path
- Model is well-known and tractable
  - Use statistical estimation
- There is no notion of contiguity
  - Use dicrete variable methods or give up
- Lack of data
  - Use transfer learning or few-shot learning, or give up

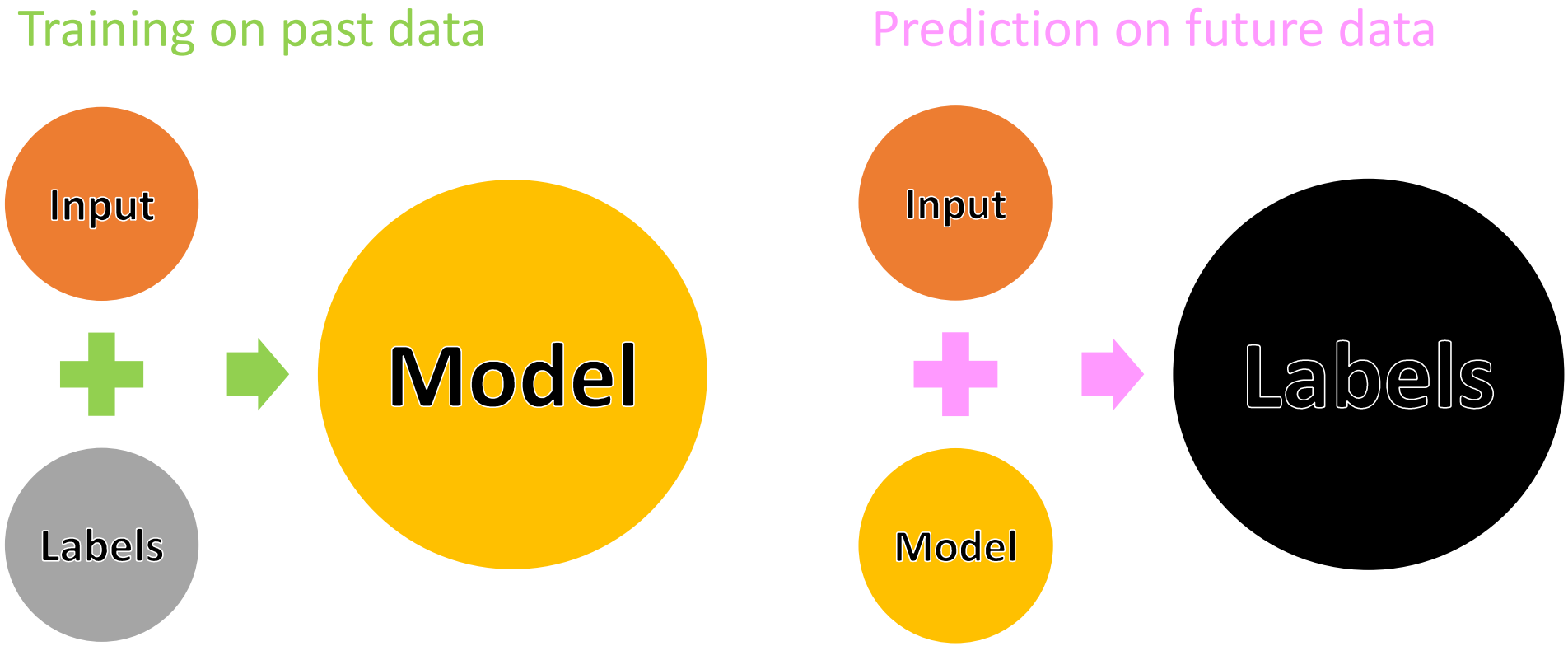
When to use ML

- Possible inputs are many or continuous
- No well-known or efficient algorithm
- Model is not well-known or tractable
- Strong notion of contiguity
- Good amount of data
- Desired output known
- Well-defined inputs

Sweet spot for ML

- Lots of structured data
- Explainability is not critical
- Prediction accuracy is the primary goal
- Underlying model is complex but stationary

ML model training and deployment



Type of ML problems

- Supervised learning: uses labeled data
  - Classification: Labels are discrete
  - Regression: Labels are continuous
  - Ranking: Labels are ordinal
- Unsupervised learning: uses unlabeled data
  - Clustering: Divide data into discrete groups
  - Dimension reduction: Represent data with fewer numbers
- Somewhere in between: fewer labels than one per example
  - Semi-supervised learning: some examples are labeled
  - Weakly supervised learning: groups of examples are labeled
  - Reinforcement learning: Label (reward) is available after a sequence of steps

Supervised Learning

- Predictor variables/features and a target variable (label)
- Aim: Predict the target variable (label), given the predictor variables
  - **Classification**: Target variable (y) consists of categories
  - **Re**

Predictor variables

Target variable **(Label)**

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

Broad types of ML problems

Output →	Categorical	Ordinal	Continuous
Supervised	Classification	Ranking	Regression
(Examples)	{Cats, dogs}	{Low, Med, High}	[-20,+10)
Unsupervised	Clustering		Dimension reduction

Some popular ML frameworks

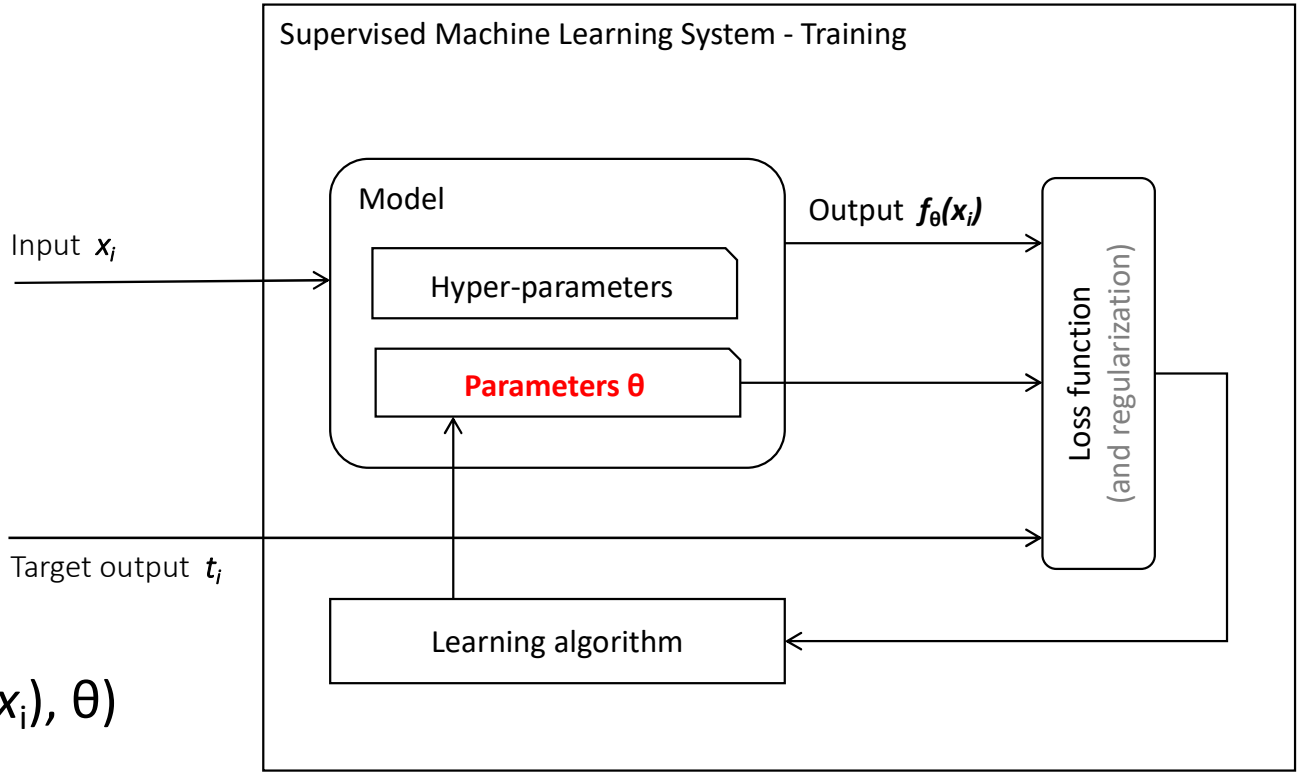
<table>
<tr><th></th><th colspan="2">Classification</th><th>Regression</th><th>Clustering</th><th>Dimension reduction</th></tr>
<tr><td rowspan="2">Vector</td><td>Logistic regression</td><td>Linear regression</td><td rowspan="2">K-means, Fuzzy C-means, DB-SCAN</td><td rowspan="2">PCA, k-PCA, LLE, ISOMAP</td></tr>
<tr><td colspan="2">SVM, RF, NN</td></tr>
<tr><td>Series, text</td><td colspan="2">RNN, LSTM, Transformer, 1-D CNN, HMM</td><td></td><td></td></tr>
<tr><td>Images</td><td colspan="2">2-D CNN, MRF</td><td></td><td></td></tr>
<tr><td>Video, MRI</td><td colspan="2">3-D CNN, CNN+LSTM, MRF</td><td></td><td></td></tr>
</table>

Recipe for ML training

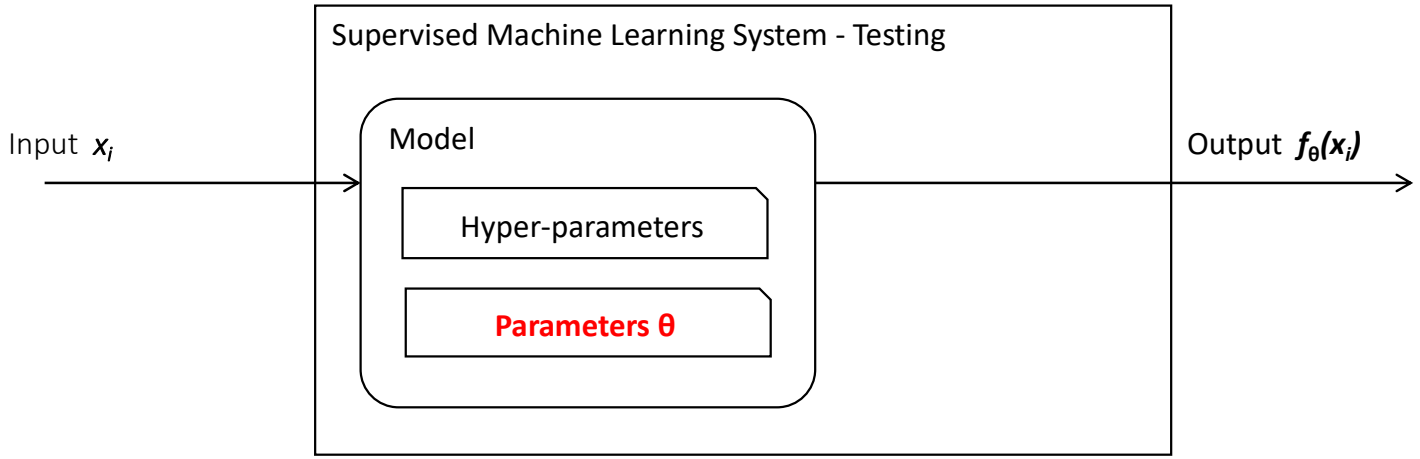
- Decide on the type of the ML problem
- Prepare data
- Shortlist ML frameworks
- Prepare training, validation, and test sets
- Train, validate, repeat
- Use test data only once

ML gives a model

- Elements of a model:
  - Input x_i
  - Function $f_\theta(x_i)$
- Utility of the model:
  - Target output t_i
  - Bring $f_\theta(x_i)$ close to t_i
  - Minimize loss $L(t_i, f_\theta(x_i), \theta)$



Components of a Trained ML System



Mathematically speaking…

- Determine f such that $t_i = f(x_i)$ and $g(T, X)$ is minimized for unseen set T and X pairs, where T is the ground truth that cannot be used
- Form of f is fixed, but some parameters can be tuned:
  - So, $y = f_\theta(x)$, where, x is observed, and y needs to be inferred
  - e.g. $y = 1$, if $mx > c$, $y = 0$ otherwise, so $\theta = (m,c)$
- Machine Learning is concerned with designing algorithms that learn "better" values of θ given "more" x (and t) for a given problem

Preparing data

- Remove useless data
  - No variance
  - Falsely assumed to be available
- Reduce redundancy
  - Correlated
    - Pearson and Spearman
- Handle missing data
  - Impute, if sporadic
  - Drop, if too frequent
- Transform variables
  - Convert discrete to one-hot-bit
  - Normalize continuous variables

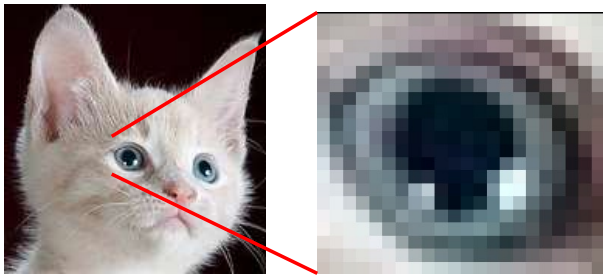
Models must exploit structure of data

- Records

Product SKU	Price	Margin	Volume
A123ajkhdf	$ 120	30%	1,000,000
B456ddsjh	$200	10%	2,000,000

- Temporal order

- Spatial order

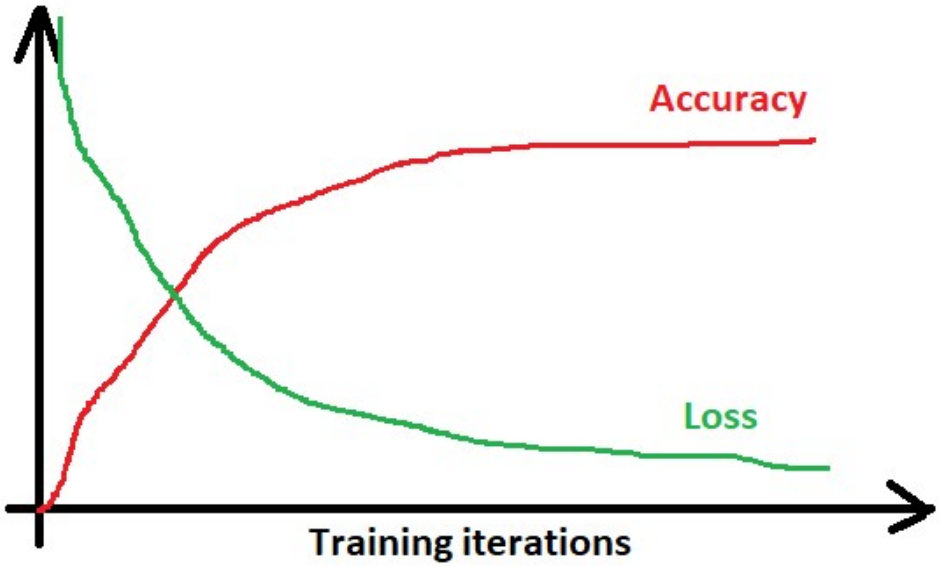


- Web of relationships

Images courtesy: Pixabay.com

Loss and accuracy

- Training accuracy saturates to a maximum
- Training loss saturates to a minimum
- Loss is a measure of error



Loss versus performance metric

- Loss is a convenient expression used for guiding the learning (optimization)
- Loss is related to performance metric, **but** it is not the same
- Loss also includes regularization
- Performance metric is what is used to judge the model
- Performance metric on only the held-out (validation or test) data makes sense

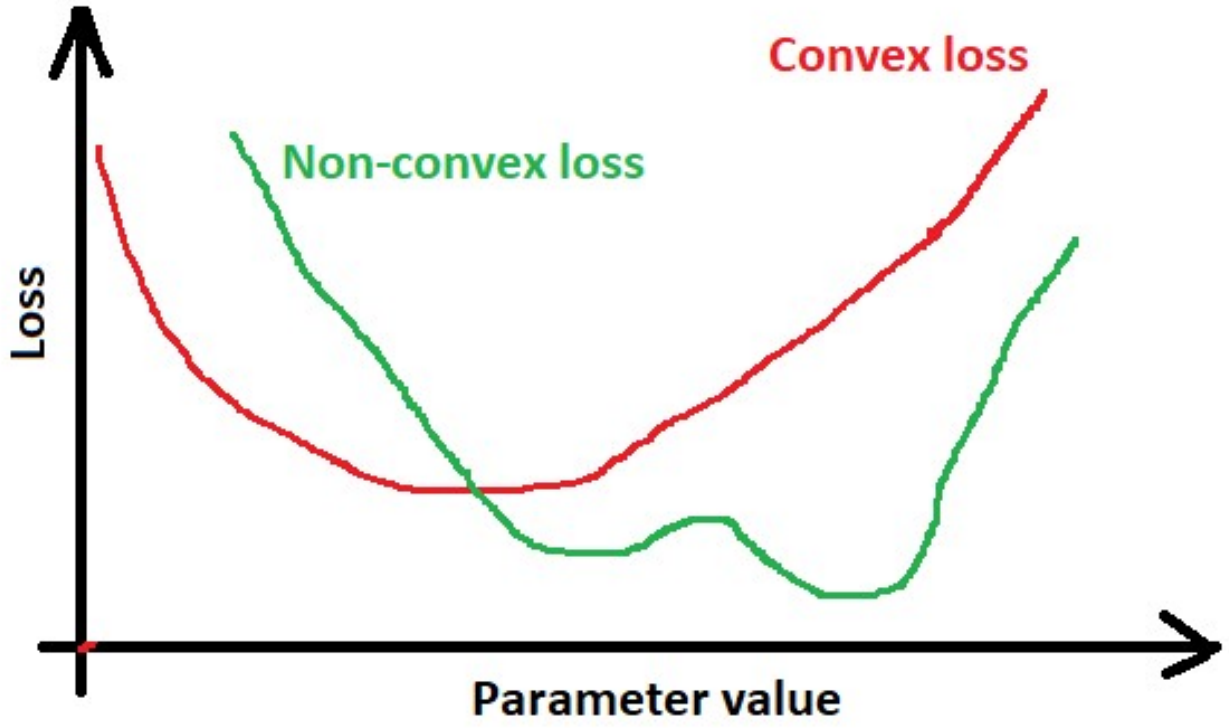
Loss function tells how bad the model is

- Loss trends opposite of accuracy
  - Loss is low when accuracy is high
  - Loss is zero for perfect accuracy (by convention)
  - Loss is high when accuracy is low
- Loss is a function of actual and desired output
- Minimizing the loss function with respect to parameters leads to good parameters

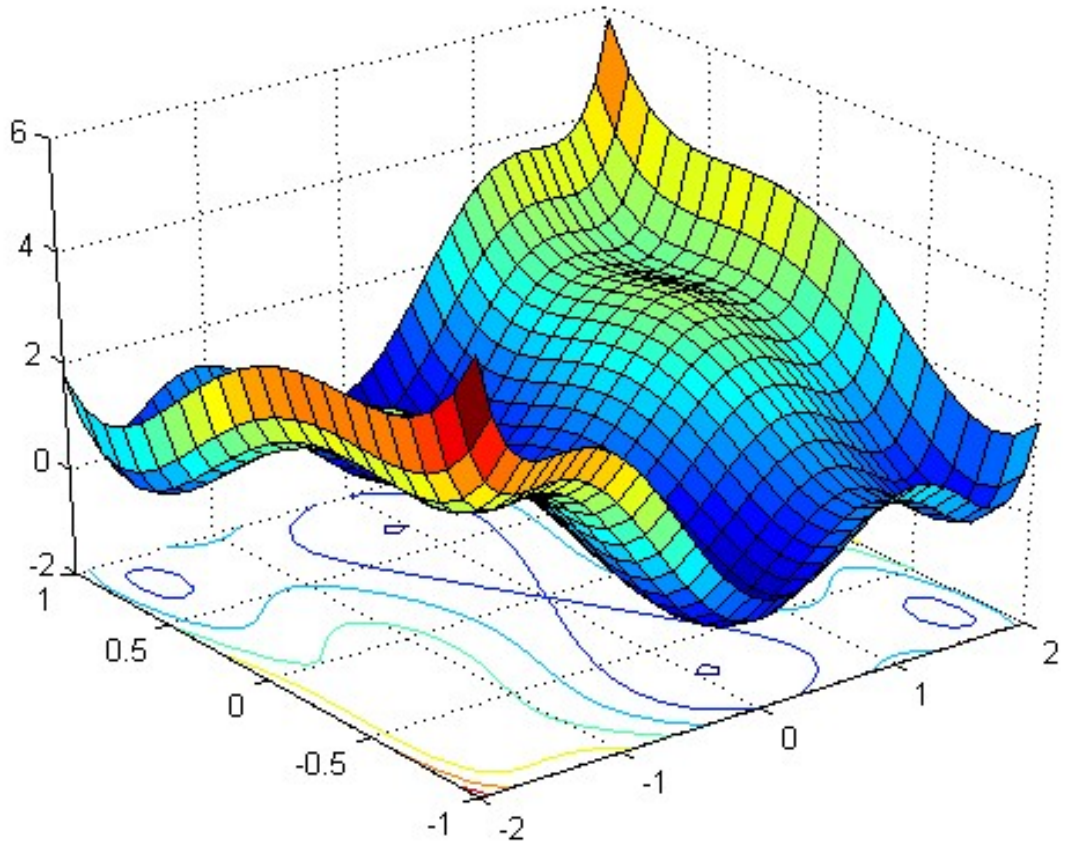
Properties of a good loss function

- Minimum value for perfect accuracy
  - Usually zero
  - *Note:* low loss on training does not guarantee low loss on validation or testing
- Varies smoothly with input
- Varies smoothly with parameters
- Good to be convex in parameters (but is usually not)
  - Like a paraboloid

Convex vs. non-convex loss



Non-convex loss can have multiple minima

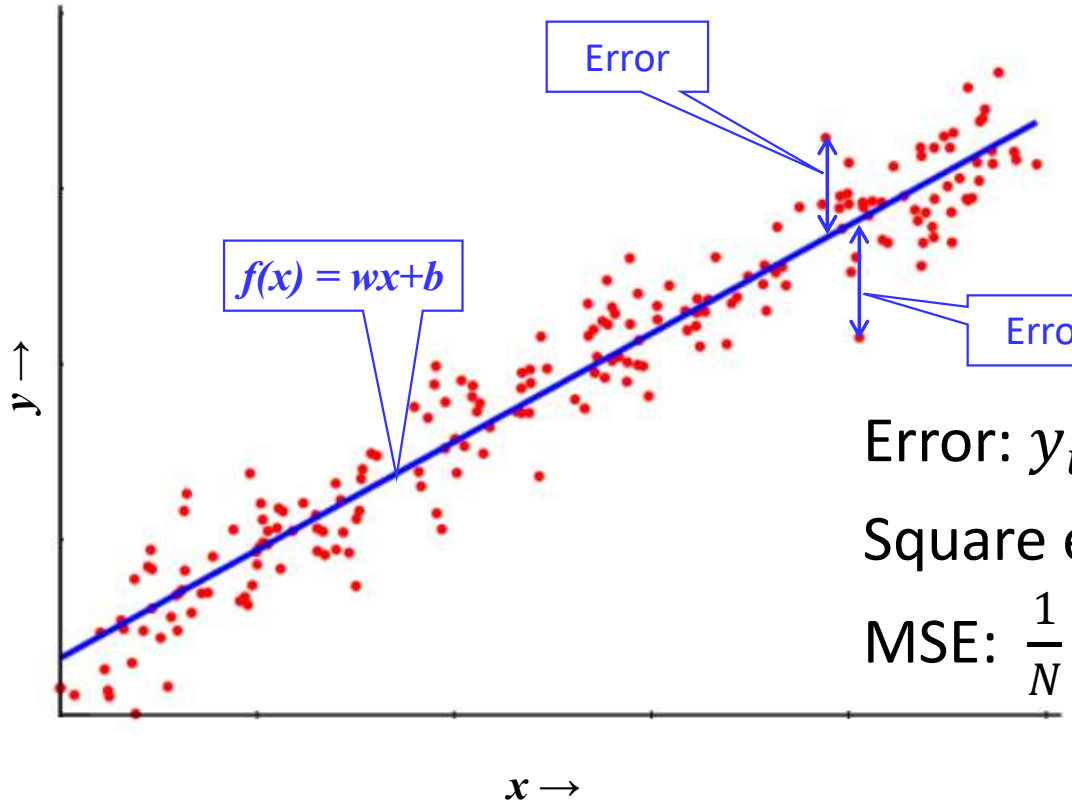


Original image source unknown

Examples of loss functions

- Regression with continuous output
  - Mean square error (MSE), log MSE, mean absolute error
- Classification with probabilistic output
  - Cross entropy (negative log likelihood), hinge loss
- Similarity between vectors or clustering
  - Euclidean distance, cosine

MSE loss for regression

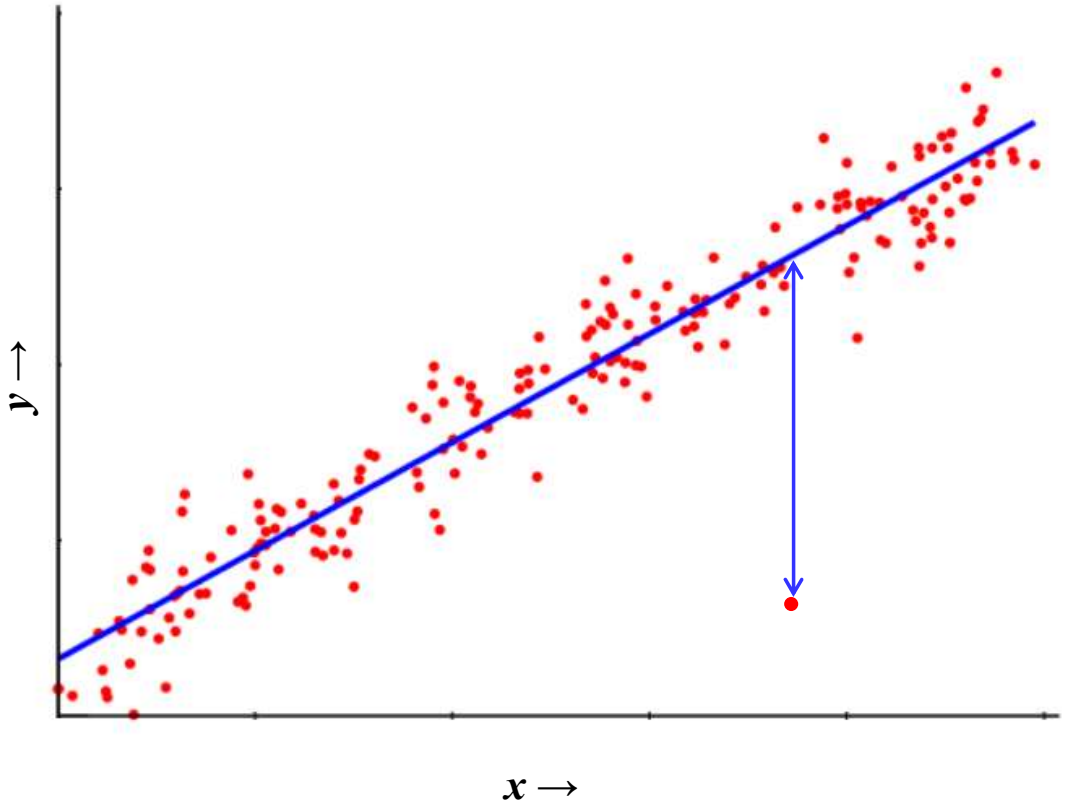


Error: $y_i - f(x_i)$

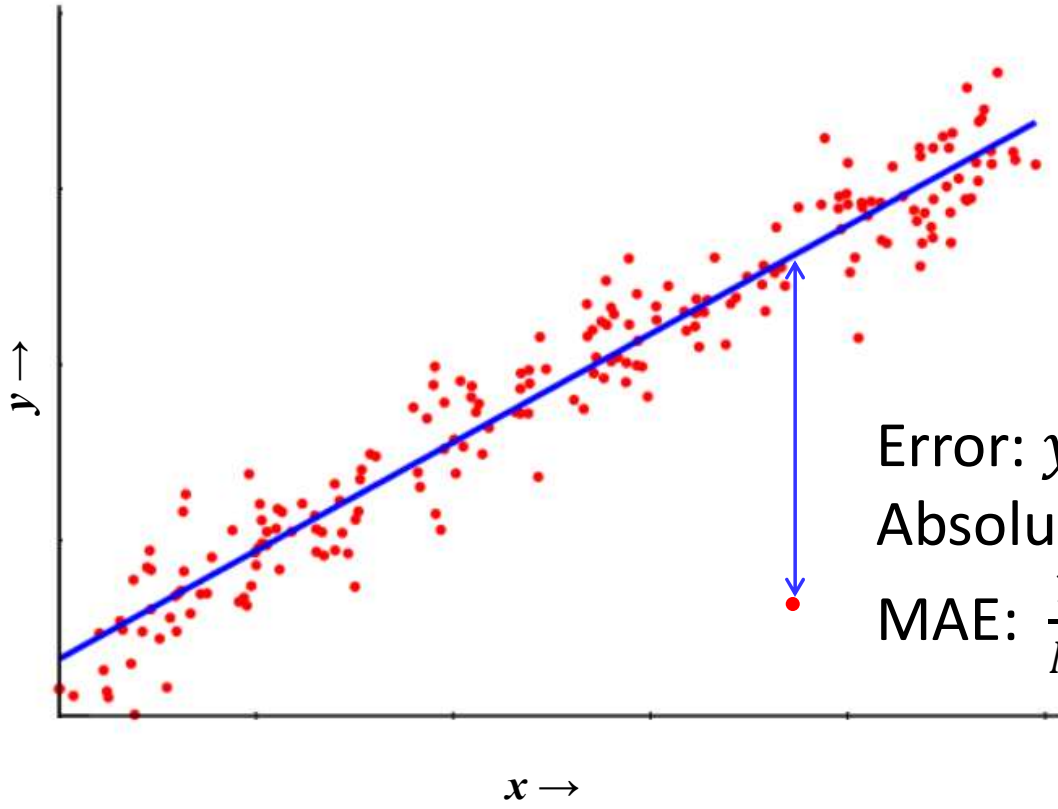
Square error: $\left(y_i - f(x_i)\right)^2$

MSE: $\frac{1}{N}\sum_{i=1}^{N}\left(y_i - f(x_i)\right)^2$

Is MSE always appropriate?



MAE loss is less affected by outliers than MSE

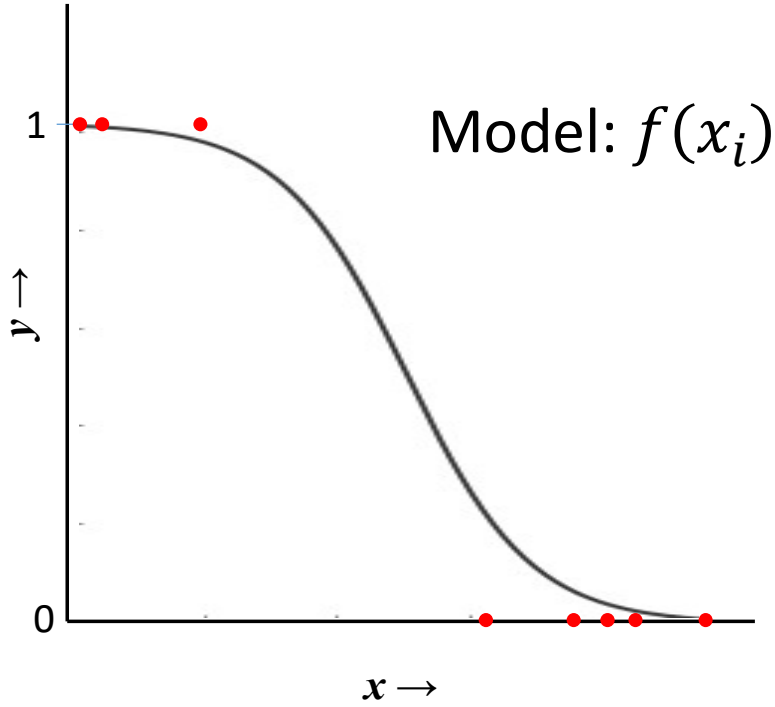


Error: $y_i - f(x_i)$

Absolute error: $|y_i - f(x_i)|$

MAE: $\frac{1}{N}\sum_{i=1}^{N} |y_i - f(x_i)|$

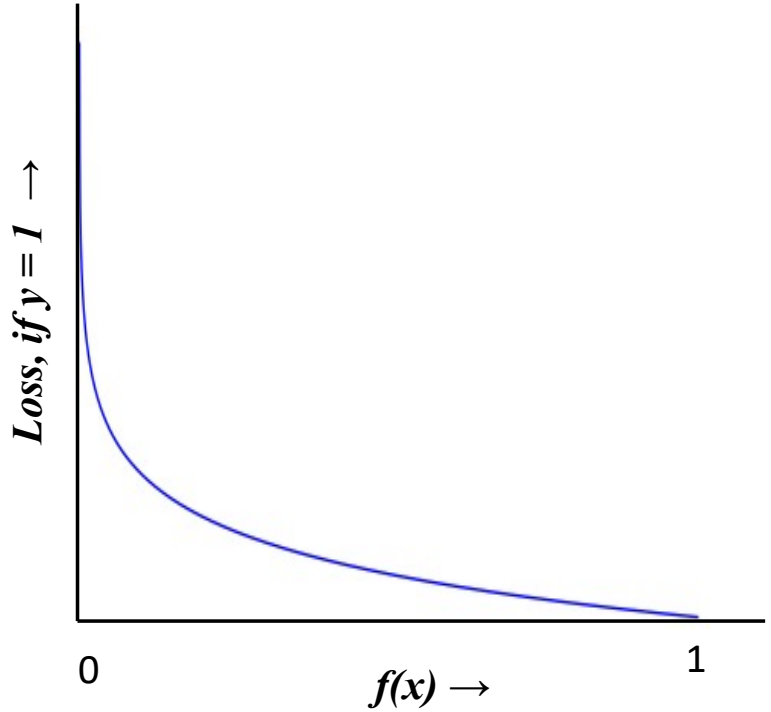
Is MSE appropriate for classification?



Model: $f(x_i) = \frac{1}{1+e^{-(wx_i+b)}}$

Original image source unknown

Cross entropy loss is preferred for classification



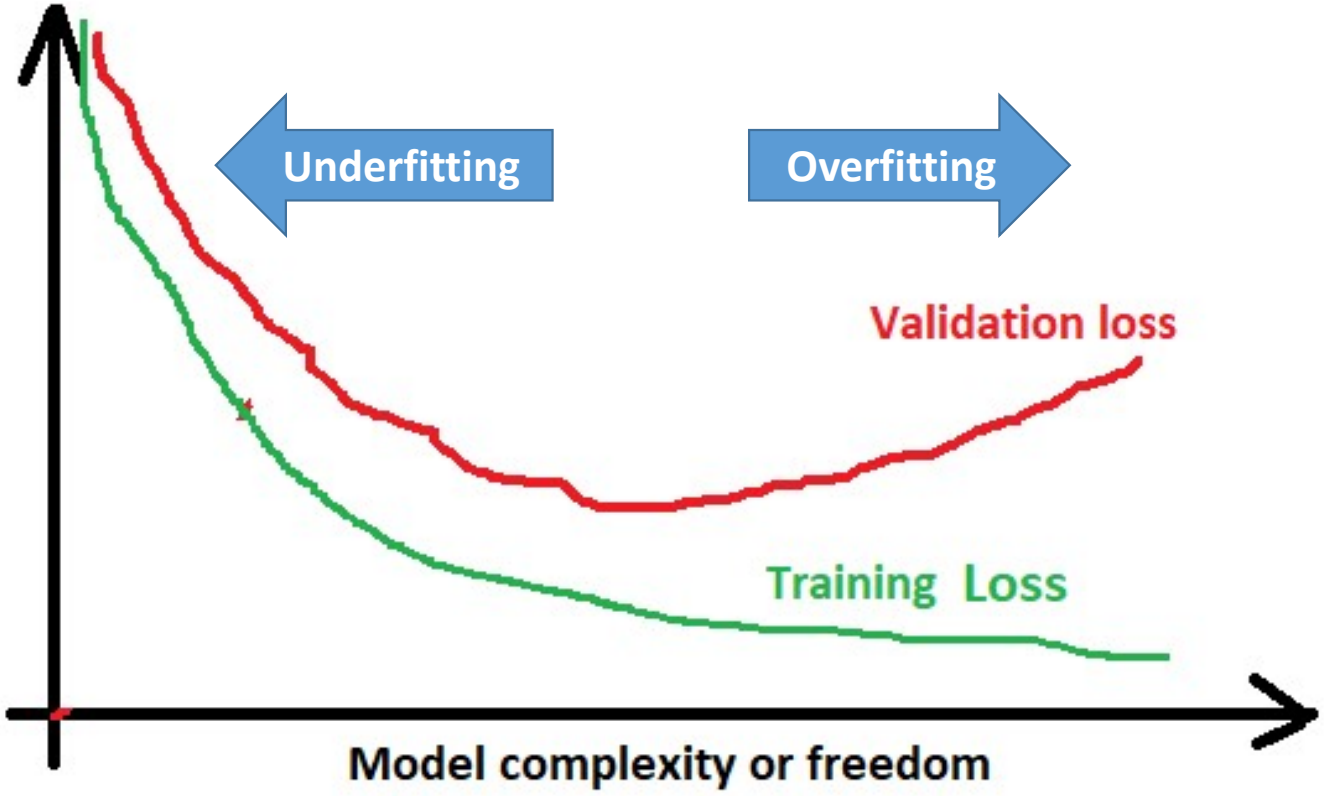
- How much does one (estimated) probability distribution *q(x)* deviates from another (real) *p(x)*
- KL-divergence of *q(x)* from *p(x)*
- For binary classification:

$$-\{y\log f(\boldsymbol{x}) + (1-y)\log\big(1-f(\boldsymbol{x})\big)\}$$

Original image source unknown

Overfitting and underfitting

- Compare training and validation loss



Regularization is a key concept in ML

- Regularization means constraining the model
- More constraints may reduce model fit on training data
- However, it may improve fit on validation and test data
- Training performance of more constrained models are more likely to reflect test performance

Parameters and Hyperparameters

- Parameters: These are the variable whose values are updated during the training process of model.
  - Feature coefficient in regression model
  - Weights of a neural network
- Hyperparameters: These are the variables/ parameter whose values are fixed by model developer before the beginning of learning process.
  - Number of variables in a tree node
  - Height of a tree
  - Number of layers of a neural network

How to regularize models

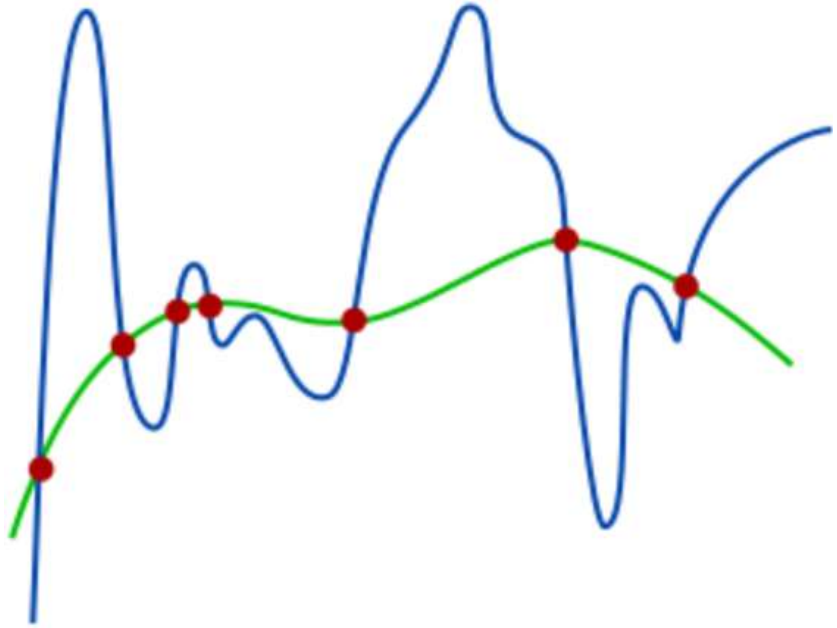


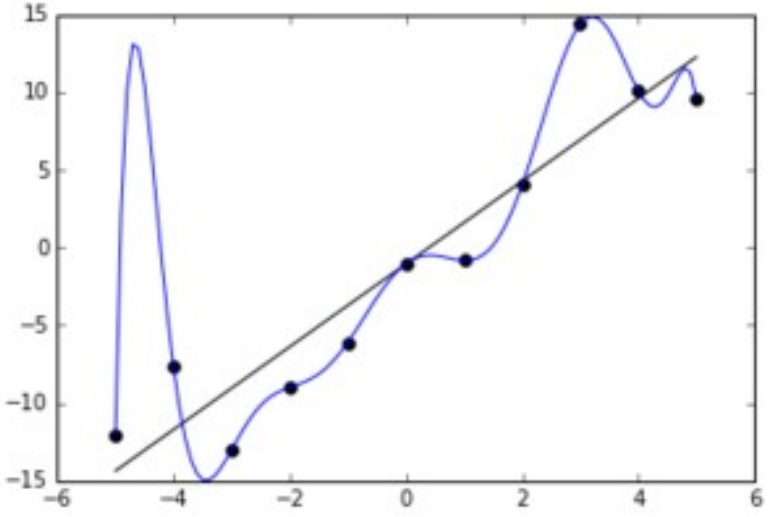
Image source: Wikipedia

Examples of hyper-parameters and parameters

- $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$
  - No hyperparameter
  - Parameters are $\boldsymbol{w}$ and b

- $f(x) = w_2 x^2 + w_1 x^1 + w_0 x^0$
  - Hyper-parameter is degree 2
  - Parameters are w_2, w_1, and w_0

Under-constrained models lead to overfitting



- An *n*-degree polynomial can fit *n* points perfectly
- But, is it overfitting?
- Is it being swayed by outliers?
- “Models should be as simple as possible, but not simplistic”
- To make model simpler:
  - Restrict number of parameters,
  - Or, restrict the set of values that they can take
- Always check validation performance

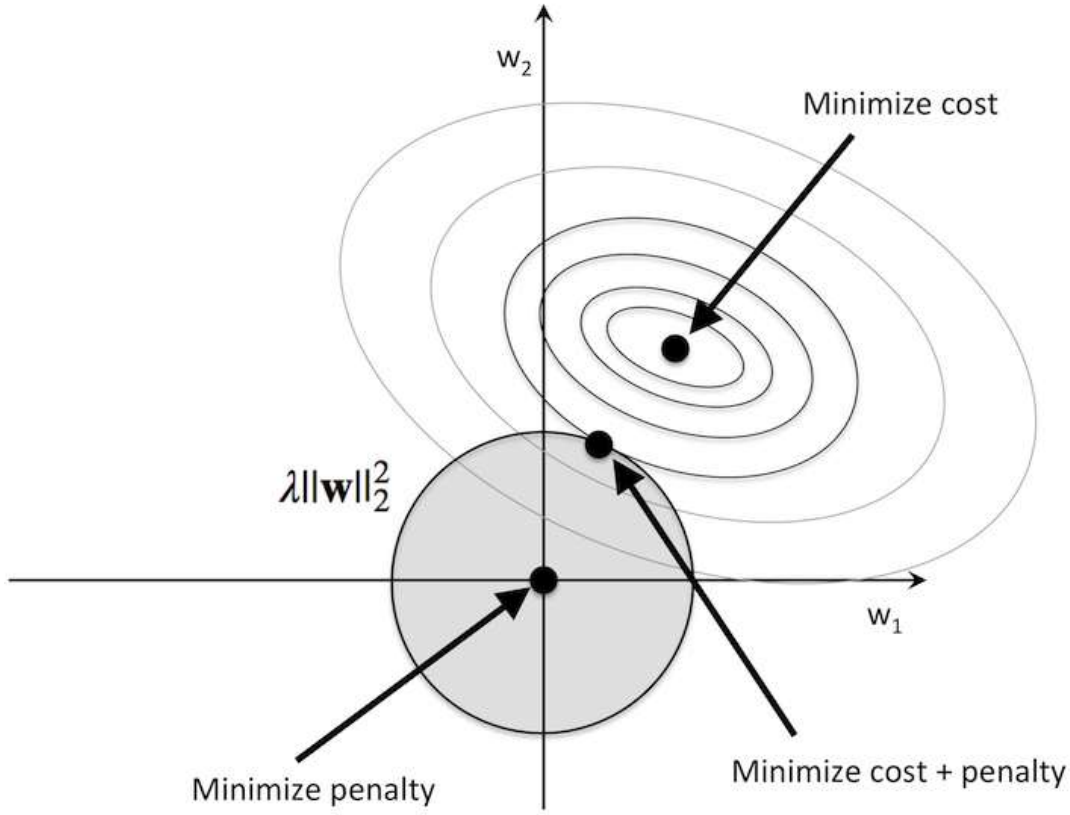
Original image source unknown

Regularization is constraining a model

- How to regularize?
  - Reduce the number of parameters
    - Share weights in structure
  - Constrain parameters to be small
  - Encourage sparsity of output in loss
- Most commonly Tikhonov (or L2, or ridge) regularization (a.k.a. weight decay)
  - Penalty on sums of squares of individual weights

$$J = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - f(x_i)\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{n} w_j{}^2 ; f(x_i) = \sum_{j=0}^{n} w_j\ x_i{}^j ;$$

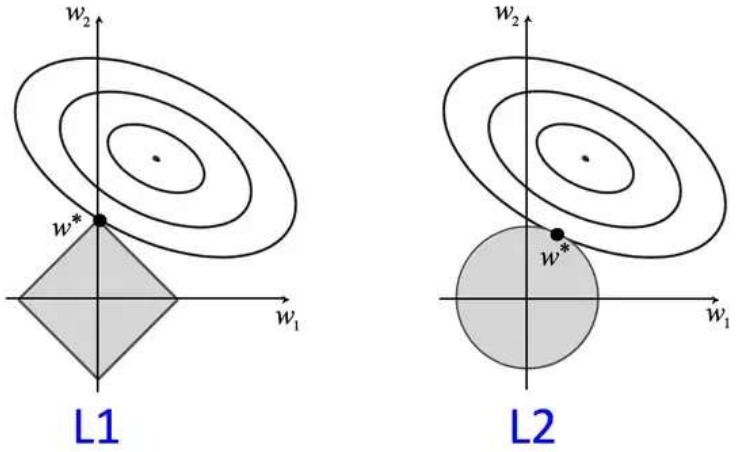
L2-regularization visualized



Original image source unknown

Other forms of regularization

- Convolutional filter structure in CNN neurons
- Max-pooling
- Dropout
- L1-regularization (sparsity inducing norm)
  - Penalty on sums of absolute values of weights



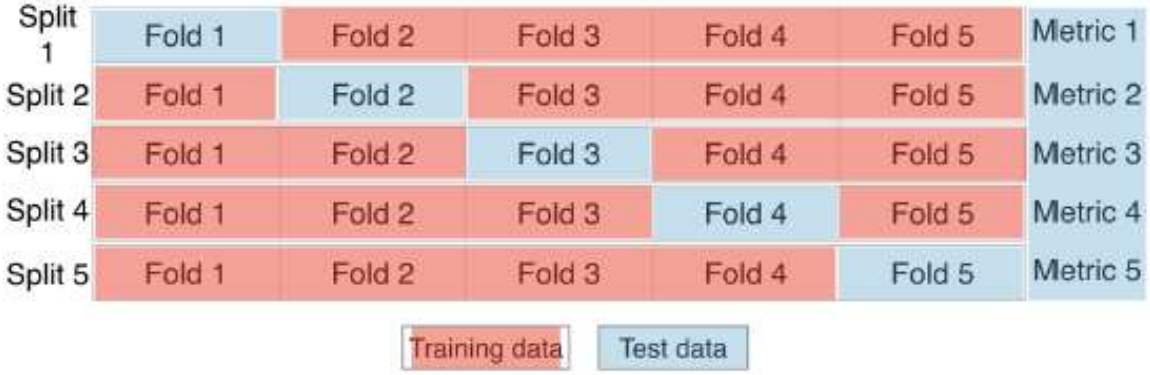
Original image source unknown

Preparing data for training and validation

- Data splits:
  - Training → Used to optimize the parameters (e.g. random 70%)
  - Validation → Used to compare models (e.g. random 15%)
  - Testing → One final check after multiple rounds of validation (e.g. random 15%)
- Cross-validation:
  - K-folds: One fold for validation, K-1 folds for training
  - Rotate folds K times
  - Select framework (hyperparameters) best average performance
  - Re-train best framework on entire data
  - Test one final time on held-out data that was not a part of any fold

Cross-validation

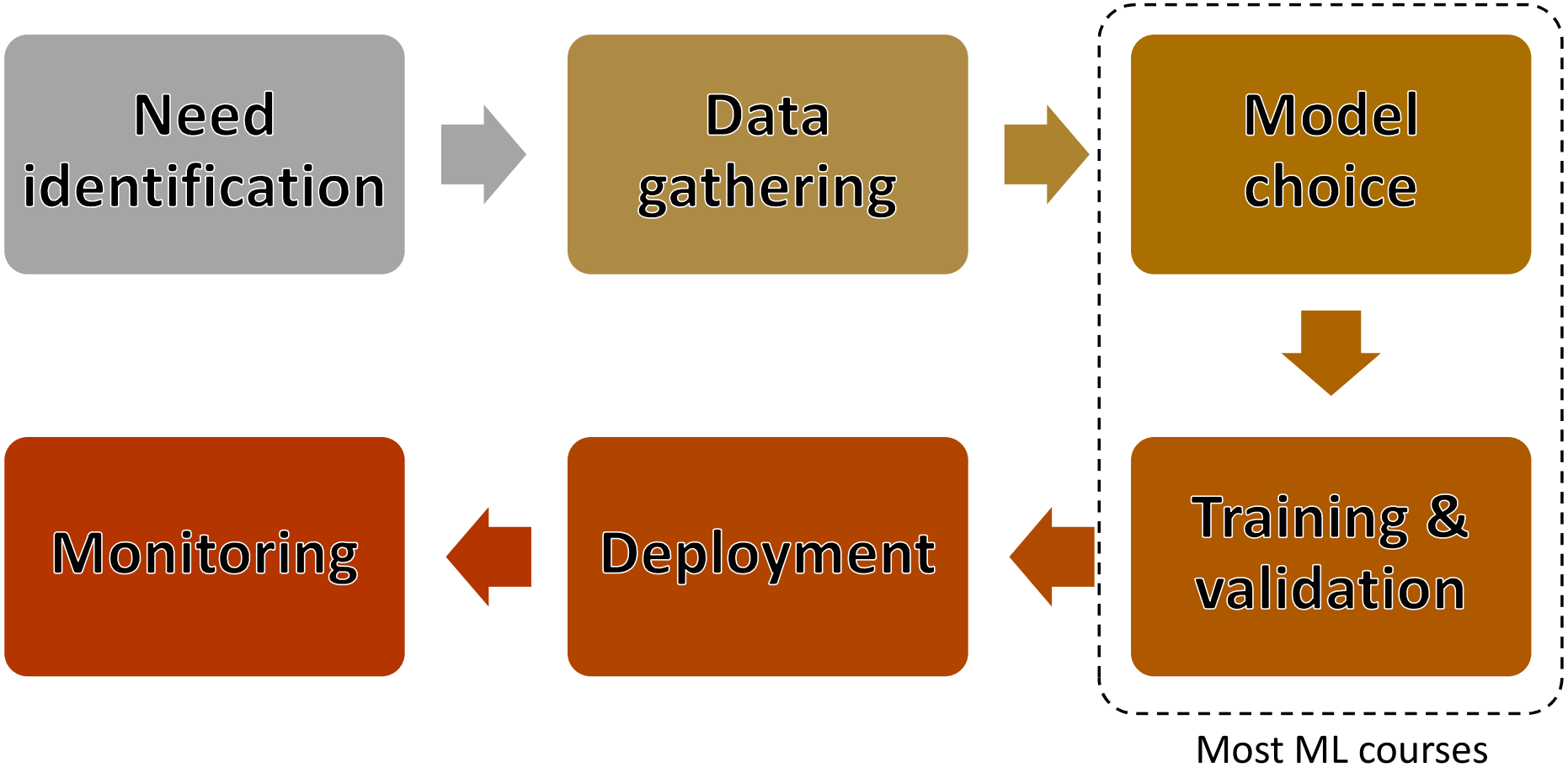
- Model performance measurement is dependent on way the data is split
- Not representative of the model's ability to generalize
- Solution: Cross-validation, especially when data is less
- Con: mor



ML can fail to perform in deployment

- Lack of training diversity: data had limited confounders
  - Single speaker, author, camera, background, accent, ethnicity, etc.
  - Data imbalance between high-value rare and more common examples
- Proxy label leak during training:
  - E.g. Only speakers A and B provide emotion “anger,” so ML confused their voice characteristics with “anger”
- Too much manual cleansing of training data
- Too little training data, and very complex models
- Concept drift: The assumptions behind training are no longer valid

ML life stages



Try to ask critical questions in response to the following statements

"We need to store every keystroke and mouse movement of use of company computers so that we can run ML on it later."

"Our product can give biometric access based on face recognition. No need for cumbersome finger, ID, or iris scans."

"We can detect pneumonia in chest x-ray with 99% accuracy."

"We can recognize people's emotions with 91% accuracy from their faces as they watch videos on our website."

"Our video call plug-in can tell when people are lying."

"Our autonomous vehicle is 20 times safer than an average driver."

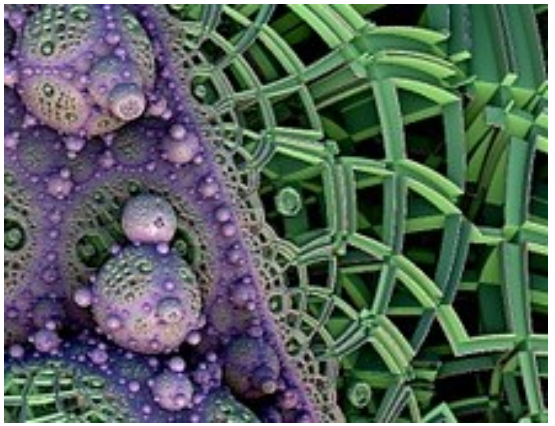
Overlooked questions

- Problem
  - Right business or societal need
- Data and provenance
  - Relevant and enough
  - Diverse and representative
  - Ethically gathered, stored
  - Meticulously recorded
- Model
  - Exploits structure in data
  - Sufficiently complex
  - Not too complex
  - Meets deployment constraints
- Validation
  - Realistic; not too optimistic
  - Covers diversity of use cases
  - Meaningful performance metrics
- User guidance
  - Declaration of intended use case
  - Description of data used
- Monitoring in-field performance
  - Data differences, concept drift
  - Unethical and unauthorized use

Relation of ML to other fields

Is AI / ML just a fad?

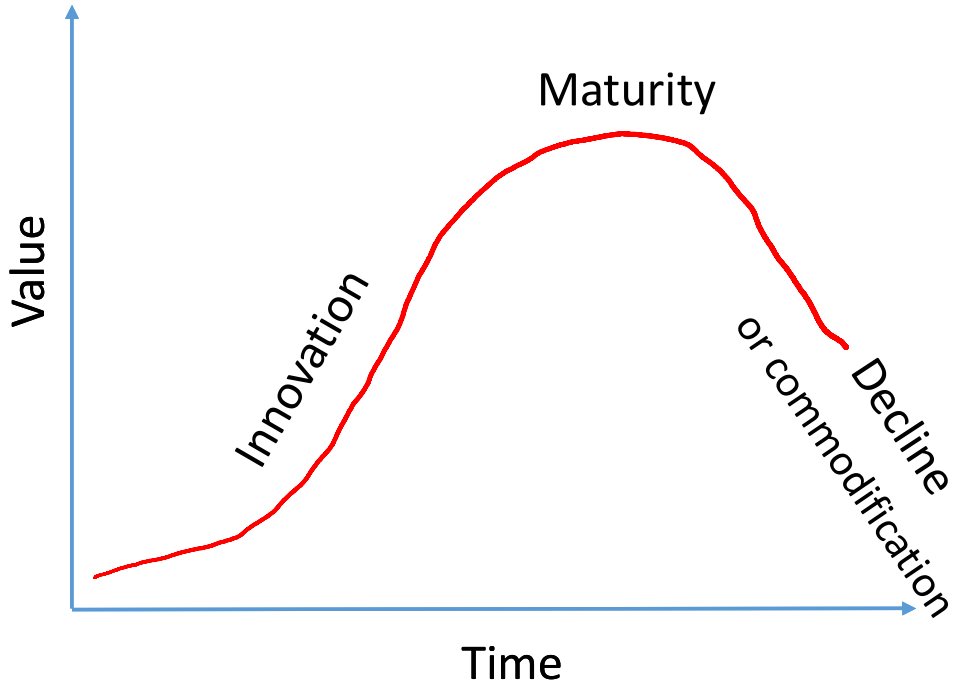
- Other hot buzzwords for VCs, businesses, and colleges over the last 30 years:
  - Programming, data structures, databases, IT
  - Computer networks, wireless communication
  - Web 2.0
  - Nano technology
  - Crypto

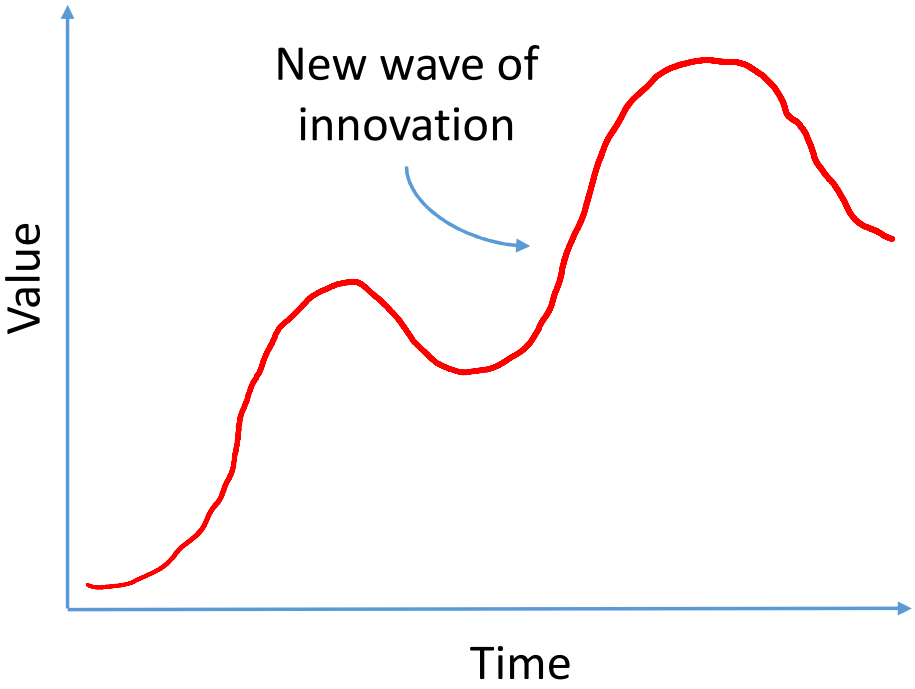


- Some of these have deeply affected our economies and society, and have reached maturity from the PoV of ongoing innovation
- Others have made a limited impact as challenges to their promises became better understood

Image courtesy: Pixabay.com

Technology life cycle has to be understood critically





ML is being deeply embedded in our economy and society

- Automated pattern recognition is already here
  - Images and videos: find people, objects, diseases …
  - Voice: convert to text, …
  - Text: queries, chat bots, translation, …
  - Time series: stocks, power consumption, …

- Automated decision-making is coming
  - Economic decisions: customer targeting, credit approval, …
  - Autonomous machines: Driverless cars, drones, robots, …
  - Critical decisions: medical diagnosis, criminal forensics …

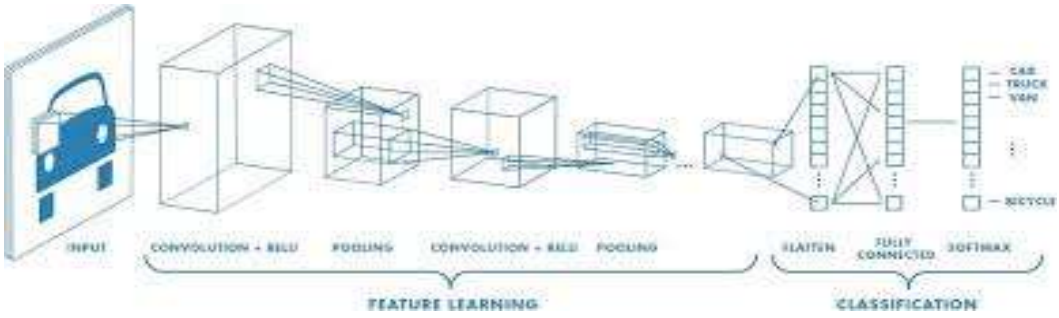
Image courtesy: Pixabay.com

Waves of AI

- Discriminative AI
  - CNN, UNet, YOLO
  - LSTM, GRU
  - Transformer
- Generative AI
  - GAN, VAE
  - Diffusion
  - State space models
- Agentic AI
  - Multi-objective
  - Multi-modal input
  - Goal oriented

Discriminative AI distinguished between inputs

- Vision
  - Image classificaiton: Does the image have a cat or a dog?
  - Image segmentation: Which pixels represent a cat?
  - Object detection: Put a box on the cat.
- NLP
  - Text classification: Is this a positive or a negative review?
  - Text annotation: Where is the positive clause in this paragraph?
- Audio
- Video



Large number of dimensions on a grid

- What features should we extract from an image?
- Should pixels be features?
- Is Euclidean distance a good metric?

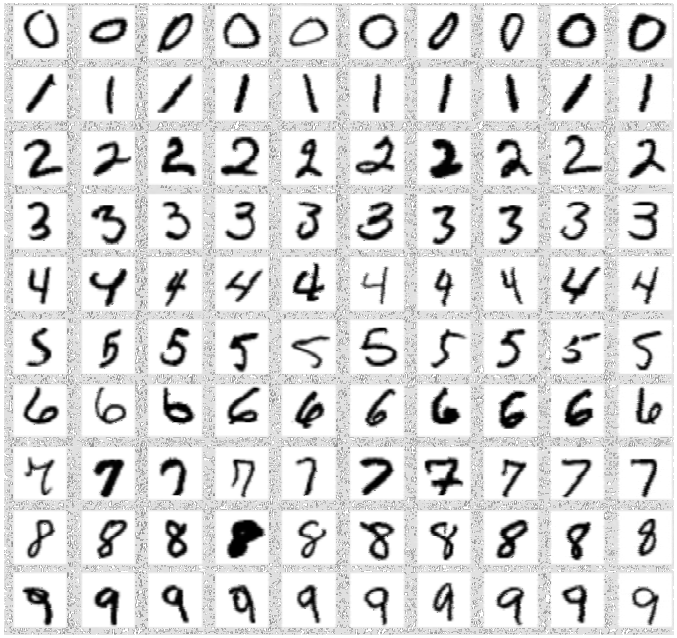


Image source: Wikipedia

Where would basic ML struggle?

Learning from similar domains

- If we have lots of labeled digits
- But only a few labeled examples of an ancient script
- Can we transfer learning?

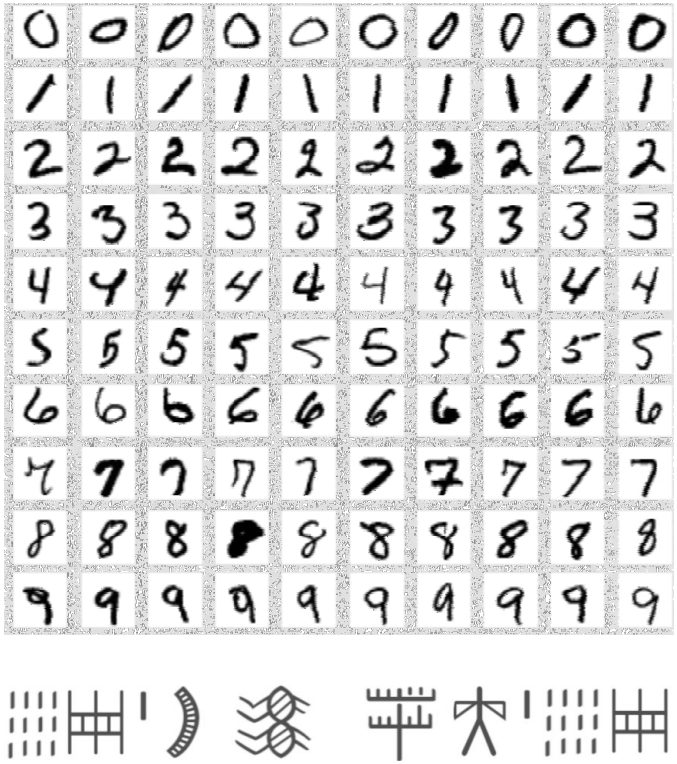
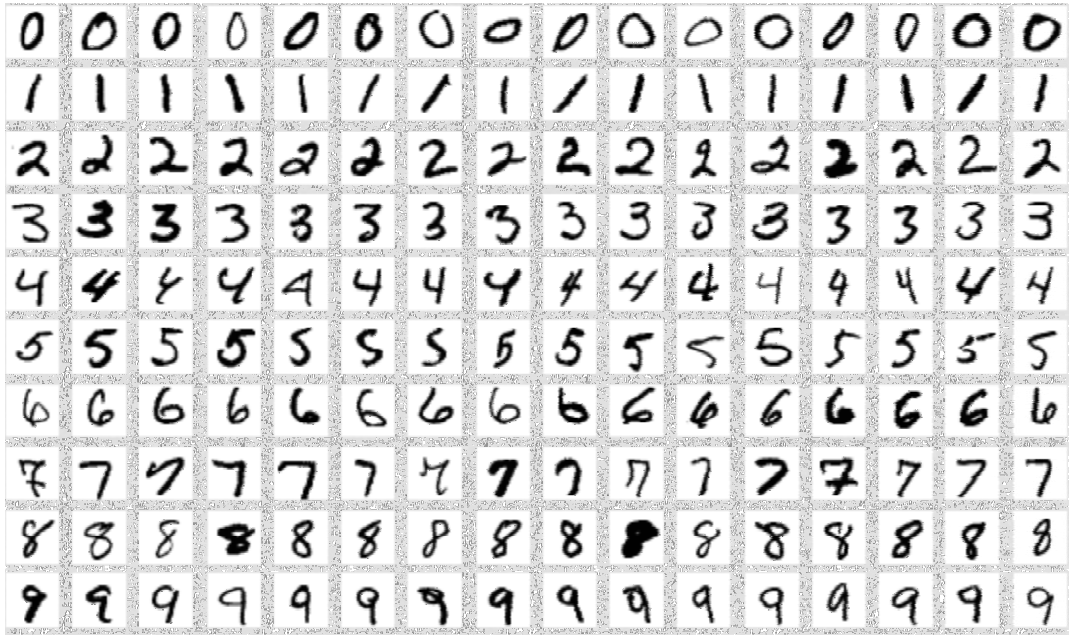


Image source: Wikipedia

Scaling feature extraction with input

- More samples

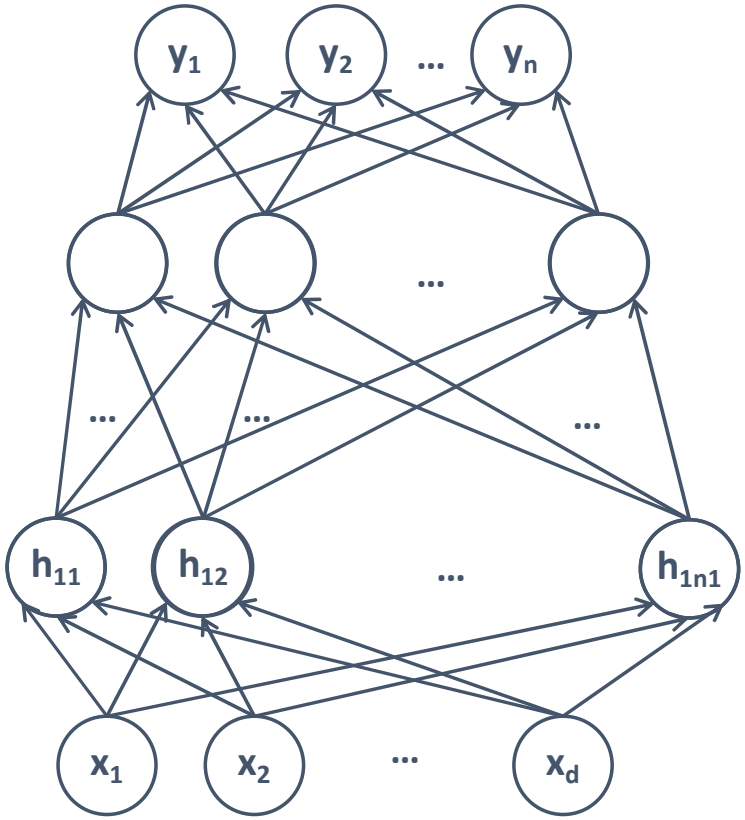


- Higher dimensional samples

Image source: Wikipedia

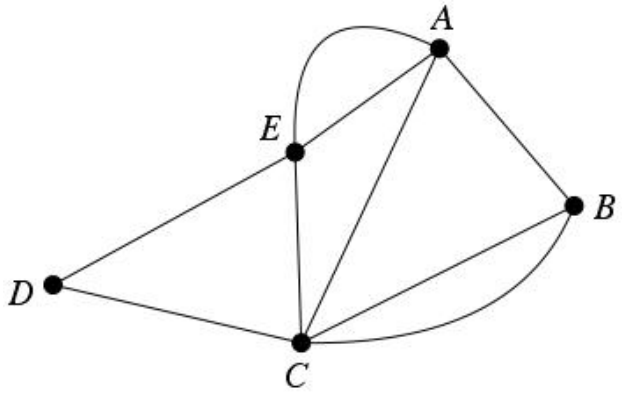
Where would basic ML struggle?

Depth vs. width

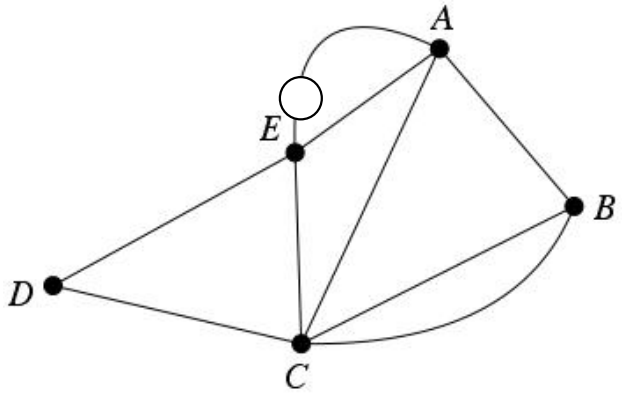


Data dimensions form a graph

- What if variables are related as graphs?

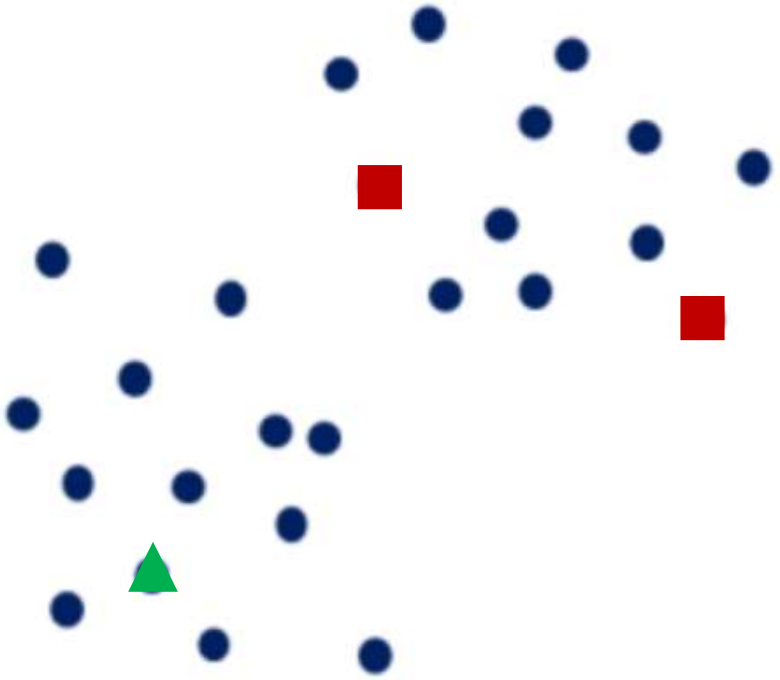


- What if different variables are missing in different samples?

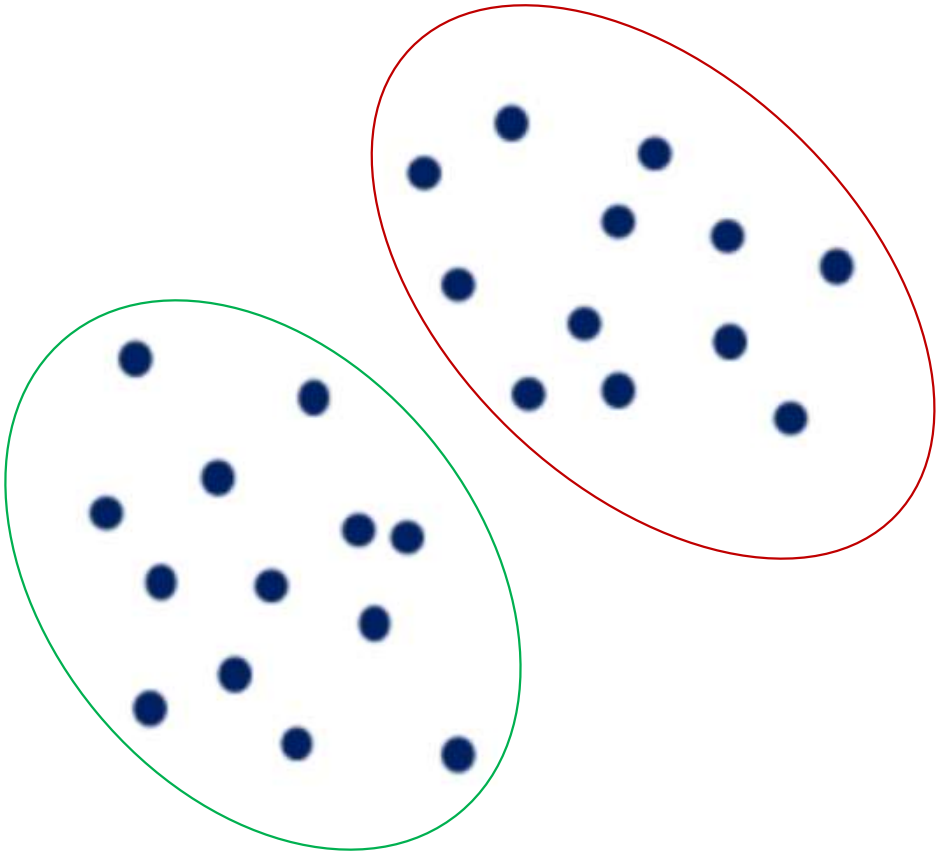


Where would basic ML struggle?

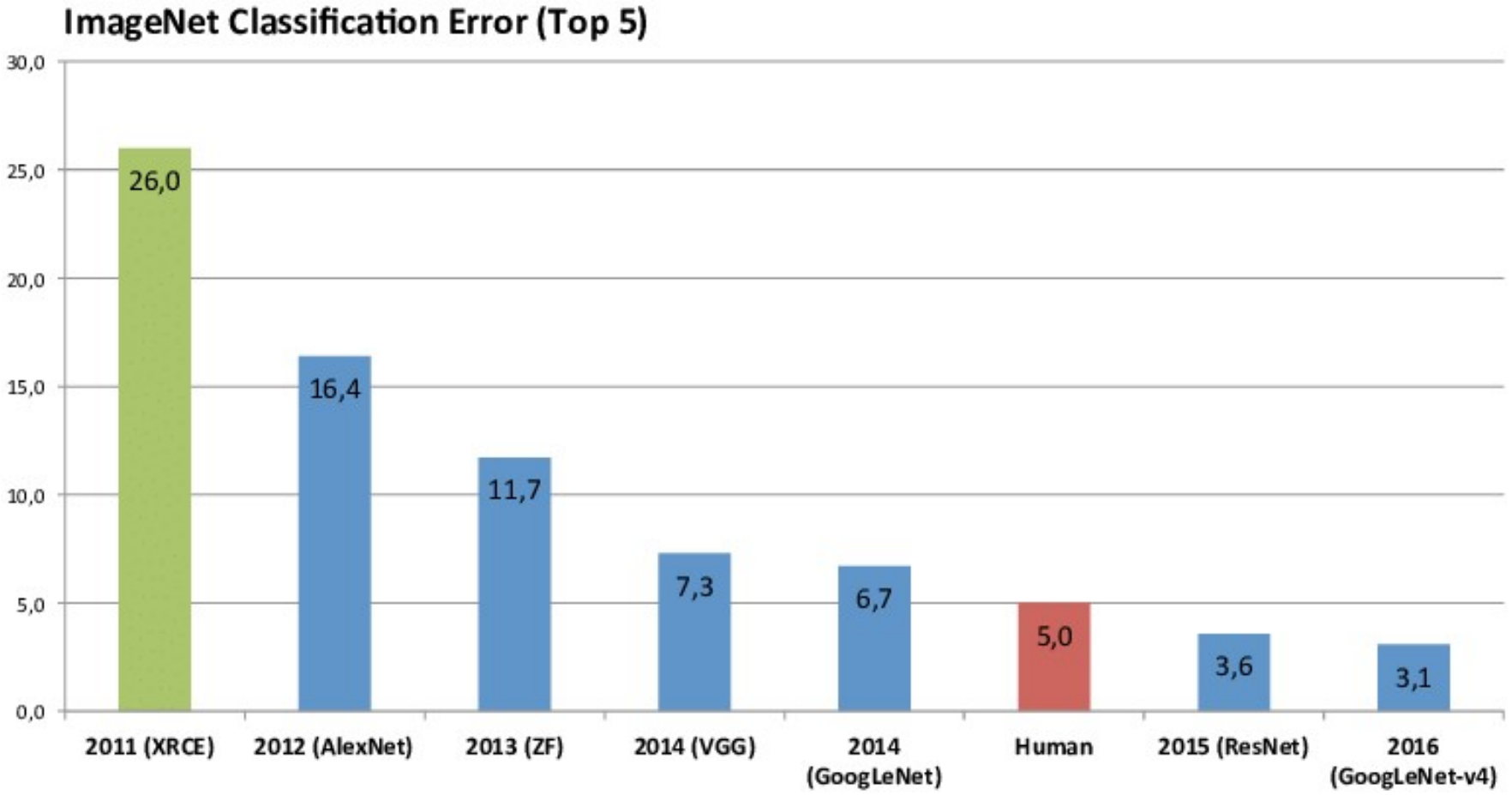
How to use unlabeled data



What if labels are given to bags of samples



Advances in AI - image recognition

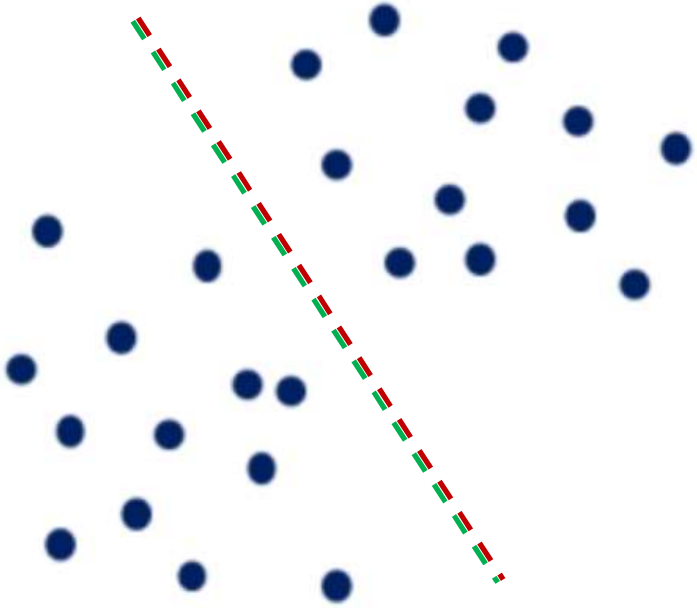


Source: devopedia.org/imagenet

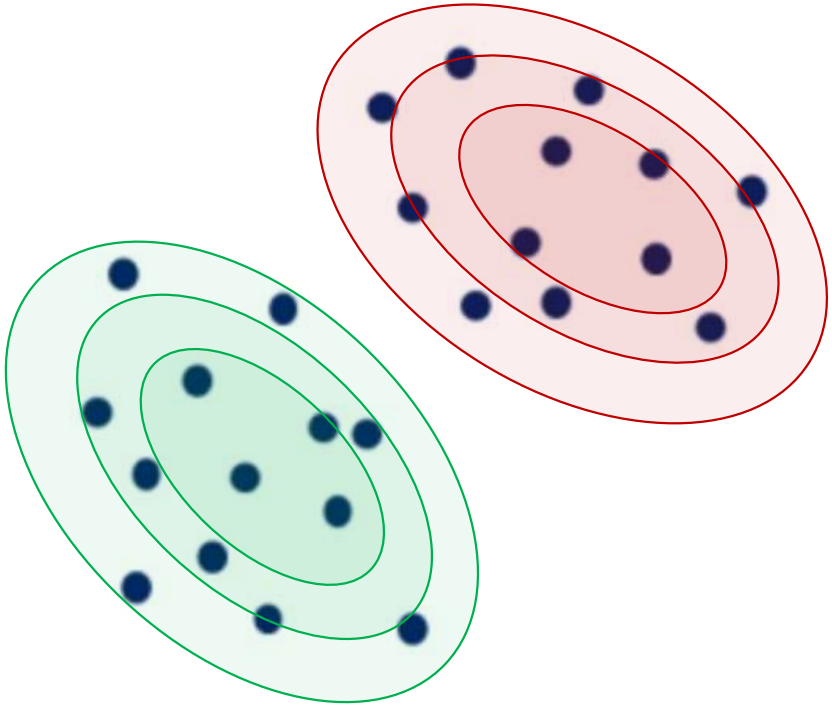
Where would basic ML struggle?

How to generate more samples

Discriminative



Generative



How does generative AI work?

We pit “counterfeiter” and “police” models to compete with each other

- Counterfeiter generates “artificial” output
- Police learns to distinguish between “artificial” and “natural”
- Goal of counterfeiter is to generate more believable “natural” outputs
- Goal of the police is to get better at spotting minute differences

Generative AI - “Gandhi in Monet style”

Source: openai.com/product/dall-e-2

Generative AI created realistic output

- Vision:
  - Image enhancement
  - Super resolution
  - Image inpainting
- Text:
  - Q&A, chatbots, RAG
  - Article and creative writing
- Multi-modal:
  - Vision-language models
  - Creating images in certain sytles, image editing

“Advantages of brushing teeth at night instead of the morning”

Brushing teeth at night has several advantages over brushing in the morning:

- **Removes plaque and bacteria:** Brushing at night removes plaque and bacteria that have accumulated throughout the day, preventing them from causing damage to your teeth and gums overnight.
- **Prevents bad breath:** Bad breath in the morning is often caused by bacteria that have been allowed to multiply in the mouth overnight. Brushing at night helps to remove these bacteria, preventing bad breath in the morning.
- **Reduces the risk of cavities:** When you brush at night, you remove food particles and sugar from your teeth, reducing the risk of cavities. If these particles are left on your teeth overnight, they can cause tooth decay.
- **Promotes better sleep:** Brushing your teeth before bed can promote a sense of cleanliness and relaxation, helping you to fall asleep more easily.
- **Gives fluoride more time to work:** If you use fluoride toothpaste, brushing at night gives the fluoride more time to work on your teeth, protecting them against decay.

Overall, brushing your teeth at night is an essential part of maintaining good oral hygiene and preventing dental problems.

Source: chat.openai.com

Literature review on domain adaptation in machine learning

ChatGPT

Domain adaptation is a subfield of machine learning that aims to improve the performance of machine learning models in a target domain, where labeled data may be scarce or nonexistent, by leveraging knowledge from a source domain. The following is a brief literature review of methods for domain adaptation in machine learning:

1. Domain Adversarial Neural Networks (DANN): DANN is a deep learning-based method for domain adaptation that learns domain-invariant features by introducing a domain classifier in the network that minimizes the discrepancy between the source and target domains.
2. Transfer Learning: Transfer learning is a popular technique for domain adaptation that leverages knowledge from a source domain to a target domain. It is commonly used when the target domain has limited labeled data.
3. Instance Weighting: Instance weighting is a simple yet effective method for domain adaptation that assigns weights to instances based on their importance in the target domain. This method can be used with various machine learning algorithms such as decision trees and neural networks.
4. Deep Adaptation Networks (DAN): DAN is a deep learning-based method that learns a mapping from the source domain to the target domain by minimizing the difference between the two domains. This method has been shown to be effective in various applications such as image classification and speech recognition.
5. Co-Training: Co-training is a method that uses multiple views of the data to train a machine learning model. In dom... hod can be used to leverage knowledge from the source don... models using different views of

Regenerate response

Google Scholar

A brief review of domain adaptation
A Farahani, S Voghoei, K Rasheed... - Advances in Data Science ..., 2021 - Springer
... To tackle the above problem, researchers proposed a new research area in **machine learning** called **domain adaptation**. In this setting, training and test sets are termed as the source ...
☆ Save 99 Cite Cited by 123 Related articles All 5 versions

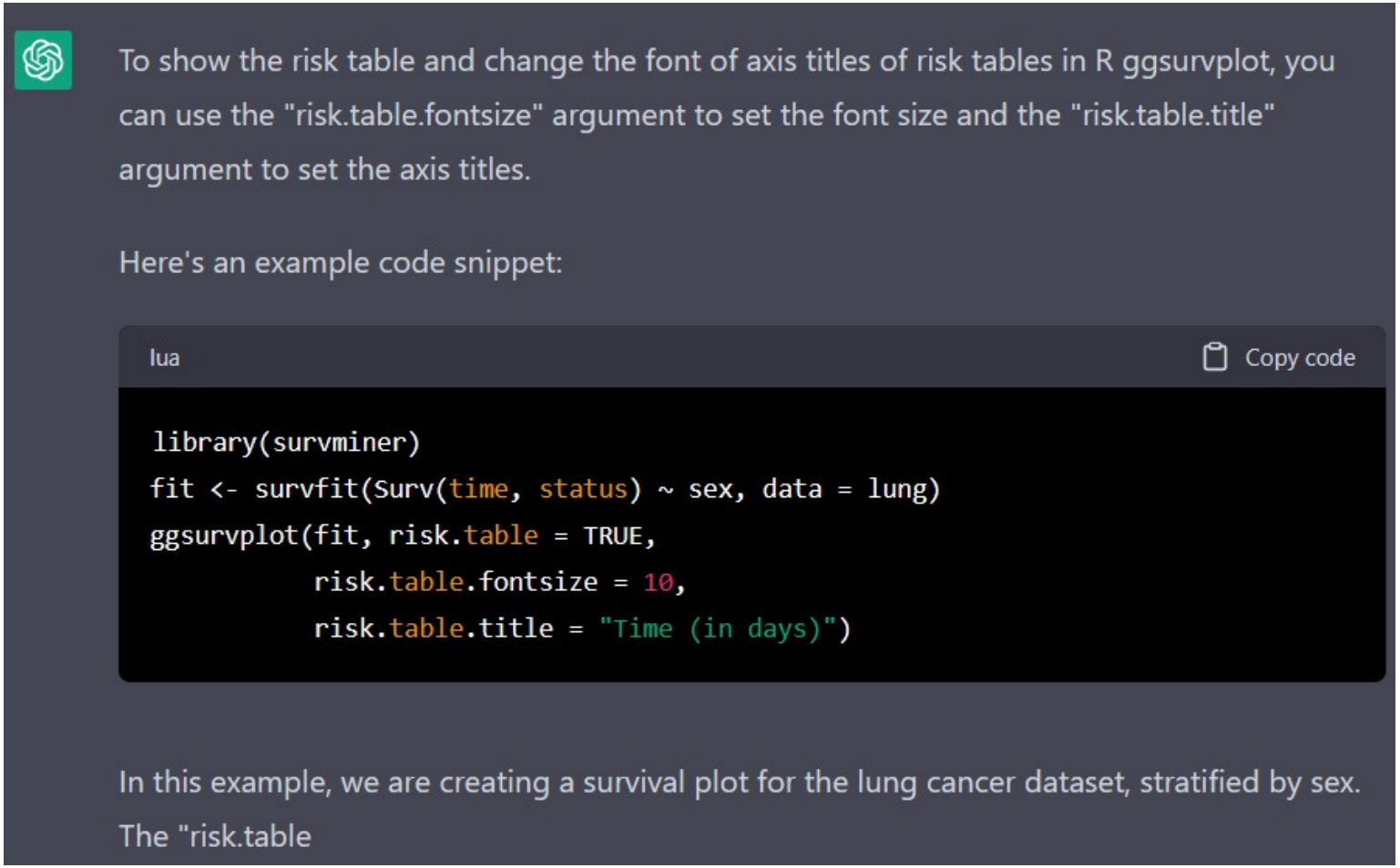
[PDF] A literature survey on domain adaptation of statistical classifiers
J Jiang - URL: http://sifaka. cs. uiuc. edu/jiang4 ..., 2008 - mysmu.edu
... **machine learning** and the natural language processing communities related to **domain adaptation**. ... First, there have been a number of methods proposed to address **domain adaptation**,
☆ Save 99 Cite Cited by 291 Related articles All 3 versions »

Return of frustratingly easy domain adaptation
B Sun, J Feng, K Saenko - Proceedings of the AAAI conference on ..., 2016 - ojs.aaai.org
... Unlike human **learning**, **machine learning** often fails to handle changes between training (... **domain adaptation** called CORrelation ALignment (CORAL). CORAL minimizes **domain** shift ...
☆ Save 99 Cite Cited by 1554 Related articles All 9 versions »

An introduction to domain adaptation and transfer learning
WM Kouw, M Loog - arXiv preprint arXiv:1812.11806, 2018 - arxiv.org
... **Domain adaptation** and transfer **learning** are sub-fields within **machine learning** that are ... risk minimization, and how transfer **learning** and **domain adaptation** expand upon this framework...
☆ Save 99 Cite Cited by 270 Related articles All 5 versions »

Domain adaptation under target and conditional shift
K Zhang, B Schölkopf, K Muandet... - ... on **machine learning**, 2013 - proceedings.mlr.press
... We consider **domain adaptation** under three possible ... transformation to find the **learning machine** that works well on ... ate distribution on the test **domain**. Thanks to kernel embedding of ...
☆ Save 99 Cite Cited by 563 Related articles All 14 versions »

How to show risk table but change the font of axis titles of risk tables in R ggsurvplot



To show the risk table and change the font of axis titles of risk tables in R ggsurvplot, you can use the "risk.table.fontsize" argument to set the font size and the "risk.table.title" argument to set the axis titles.

Here's an example code snippet:

lua Copy code

```lua
library(survminer)
fit <- survfit(Surv(time, status) ~ sex, data = lung)
ggsurvplot(fit, risk.table = TRUE,
           risk.table.fontsize = 10,
           risk.table.title = "Time (in days)")
```

In this example, we are creating a survival plot for the lung cancer dataset, stratified by sex. The "risk.table

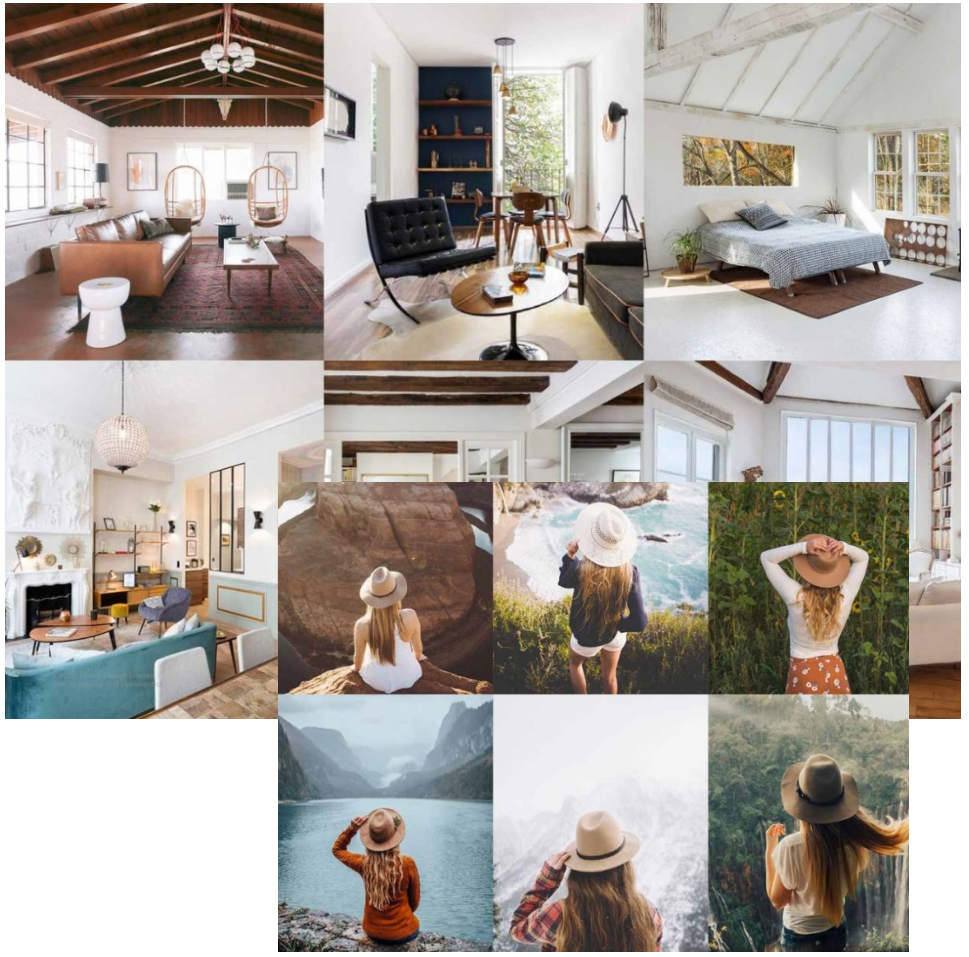
Source: chat.openai.com

“The age of average” (by Alex Murrell)

“So, this is your call to arms. Whether you’re in film or fashion, media or marketing, architecture, automotive or advertising, it doesn’t matter. Our visual culture is flatlining and the only cure is creativity.

It’s time to cast aside conformity. It’s time to exorcise the expected. It’s time to decline the indistinguishable.

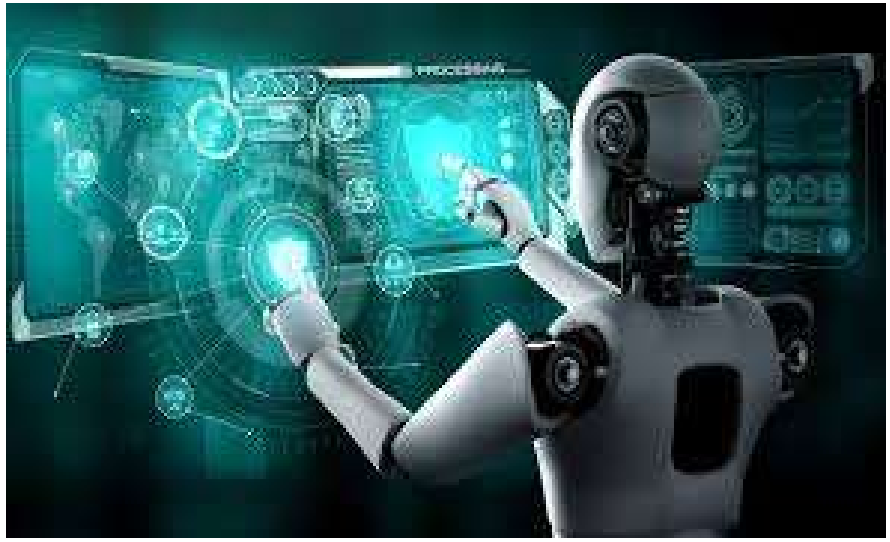
For years the world has been moving in the same stylistic direction. And it’s time we reintroduced some originality.”



Agentic AI will orchestrate various tools for specific purposes

- Agentic AI can make decisions, take autonomous actions, and continually learn from interactions to achieve specific objectives

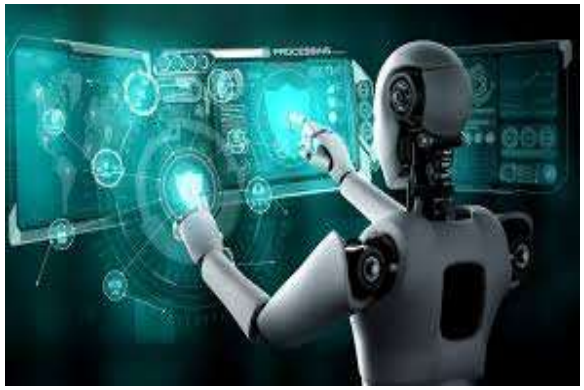
- Characteristics
  - Autonomy
  - Goal-oriented
  - Adaptability
  - Proactive

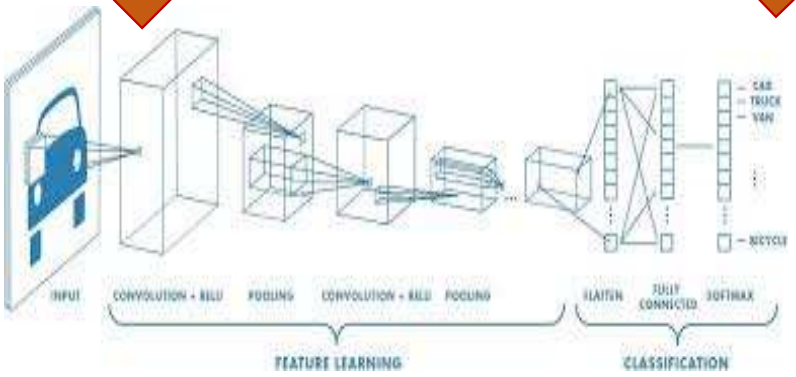


Example utilization of different AI components

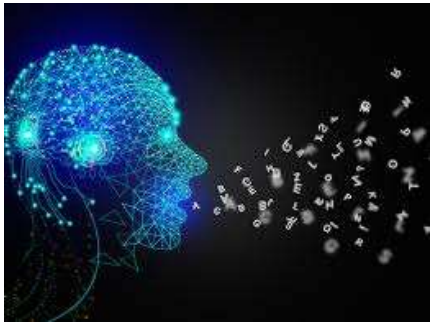
Sensors

Agentic AI





Discriminative AI



Generative AI

Actuators

Other predictions for AI in 2025

- Generative AI will plateau and hype will die down
- Realism will set in but more use cases will be discovered
- Businesses will become savvy about evaluating AI
- Agentic AI will become the buzzword
- Human feedback will make a comeback
- Debate about ethics and regulations will take pace

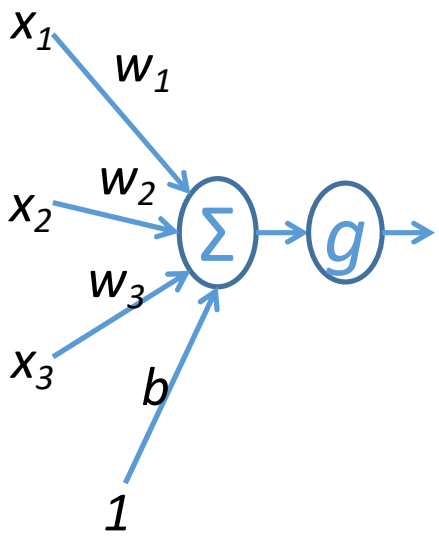
Other considerations

- Cost - dedicated hardware or cloud services and manpower
- Power and impact on environment
- Concentration of AI talent and innovations
- Conformity
- Loss of nuance
- Tailoring to in-house data and knowledge

Agenda

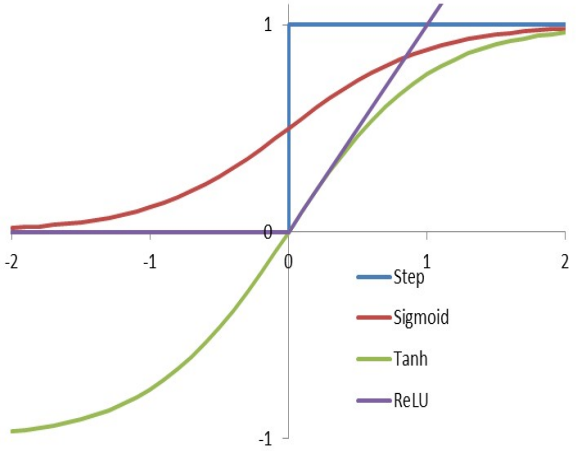
- Overview of machine learning
- **Revision of neural networks**
- Tentative list of topics in AML

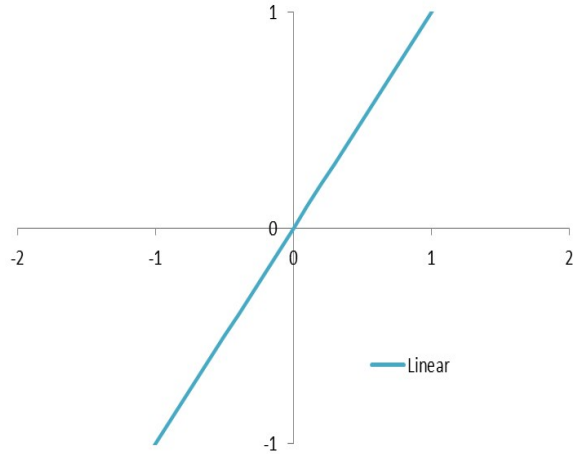
Activation function is the secret sauce of neural networks



- Neural network training is all about tuning weights and biases
- If there was no activation function *f*, the output of the entire neural network would be a linear function of the inputs
- The earliest models used a step function

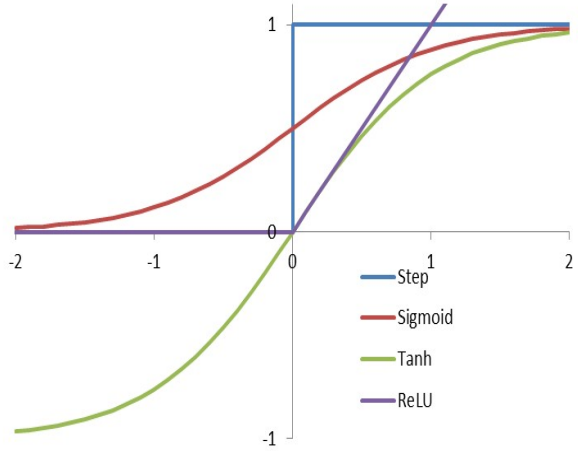
Types of activation functions

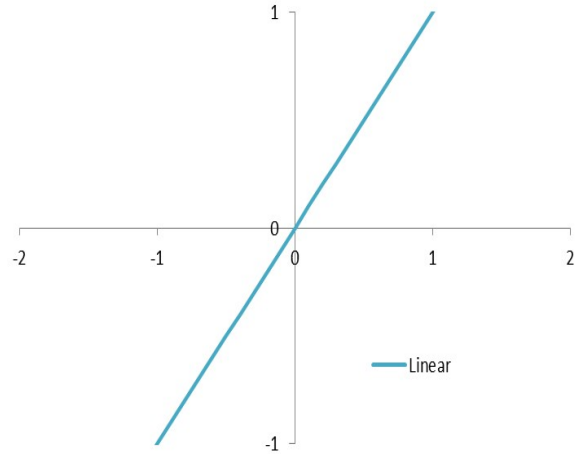




- **Step**: original concept behind classification and region bifurcation. Not used anymore
- **Sigmoid** and tanh: trainable approximations of the step-function
- **ReLU**: currently preferred due to fast convergence
- **Softmax**: currently preferred for output of a *classification net.* Generalized sigmoid
- **Linear**: good for modeling a range in the output of a *regression net*

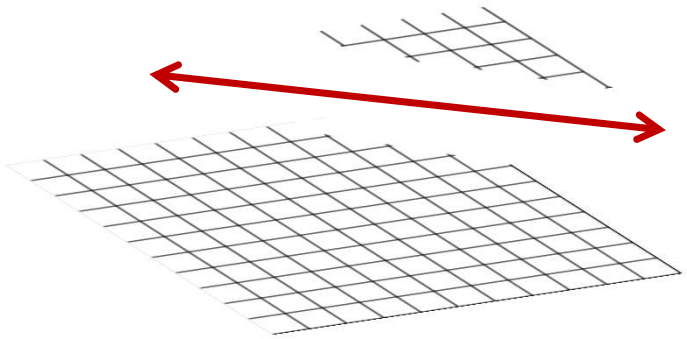
Formulas for activation functions





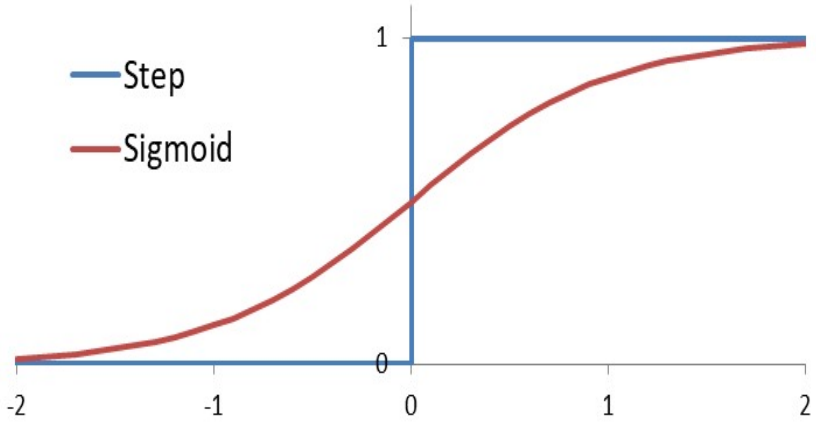
- **Step**: $g(x) = \frac{\text{sign}(x)+1}{2}$
- **Sigmoid**: $g(x) = \frac{1}{1+e^{-x}}$
- **Tanh**: $g(x) = \tanh(x)$
- **ReLU**: $g(x) = \max(0, x)$
- **Softmax**: $g(x_i) = \frac{e^{x_i}}{\sum_i e^{x_i}}$
- **Linear**: $g(x) = x$

Step function divides the input space into two halves → 0 and 1



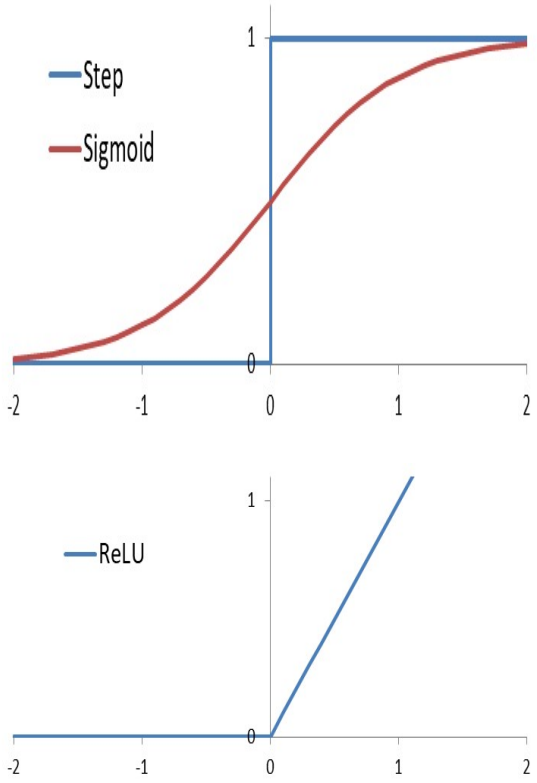
- In a single neuron, step function is a linear binary classifier
- The weights and biases determine where the step will be in n-dimensions
- But, as we shall see later, it gives little information about how to change the weights if we make a mistake
- So, we need a smoother version of a step function
- Enter: the Sigmoid function

The sigmoid function is a smoother step function



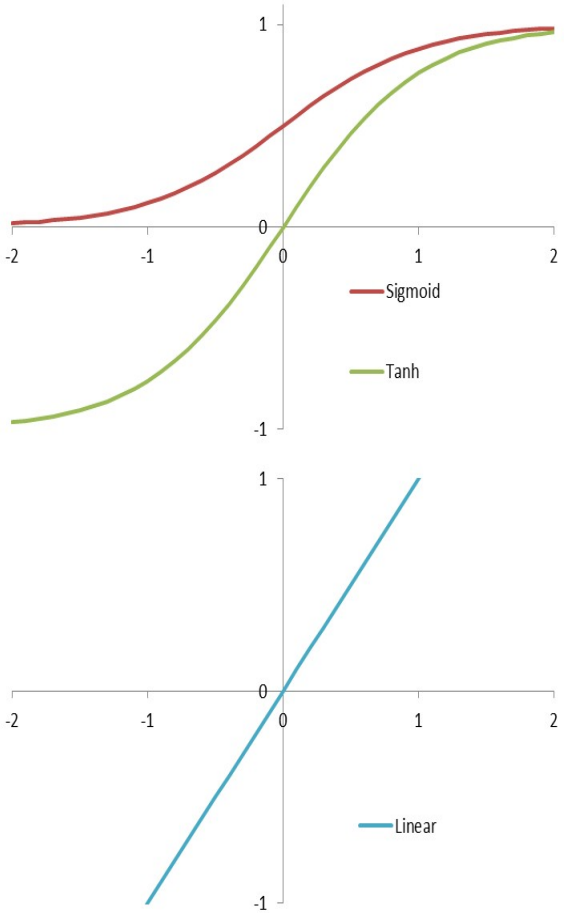
- Smoothness ensures that there is more information about the direction in which to change the weights if there are errors
- Sigmoid function is also mathematically linked to logistic regression, which is a theoretically well-backed linear classifier

The problem with sigmoid is (near) zero gradient on both extremes



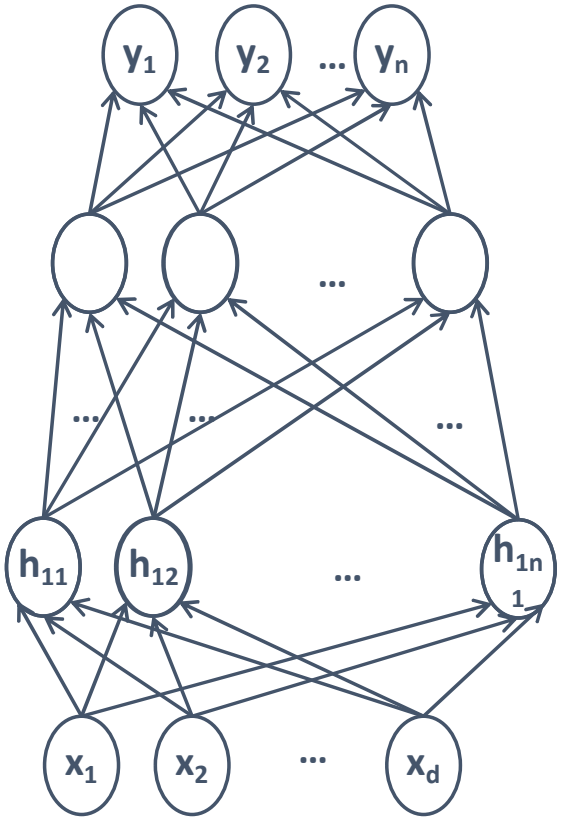
- For both large positive and negative input values, sigmoid doesn't change much with change of input
- ReLU has a constant gradient for almost half of the inputs
- But, ReLU cannot give a meaningful final output

Output activation functions can only be of the following kinds



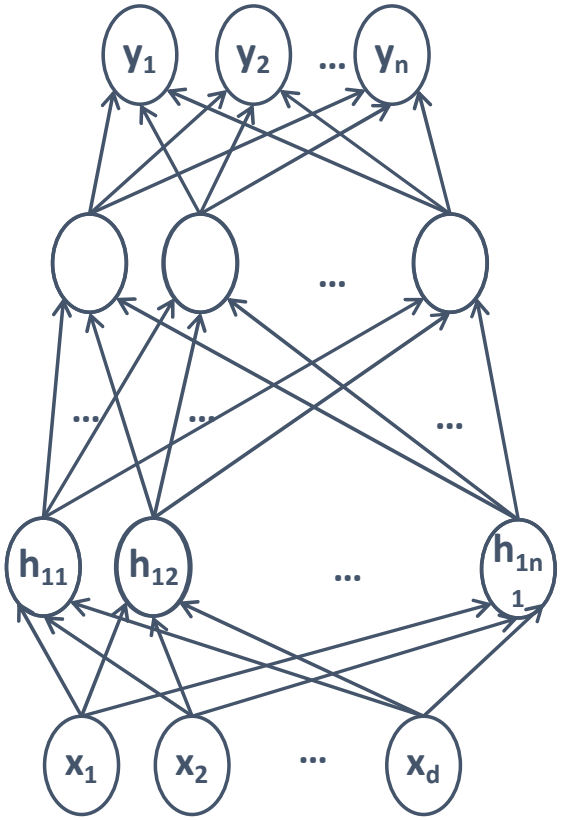
- Sigmoid gives binary classification output
- Tanh can also do that provided the desired output is in {-1, +1}
- Softmax generalizes sigmoid to n-ary classification
- Linear is used for regression
- ReLU is only used in internal nodes (non-output)

Basic structure of a neural network



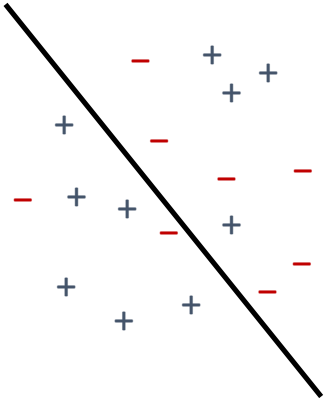
- It is feed forward
  - Connections from inputs towards outputs
  - No connection comes backwards
- It consists of layers
  - Current layer's input is previous layer's output
  - No lateral (intra-layer) connections
- That's it!

Basic structure of a neural network

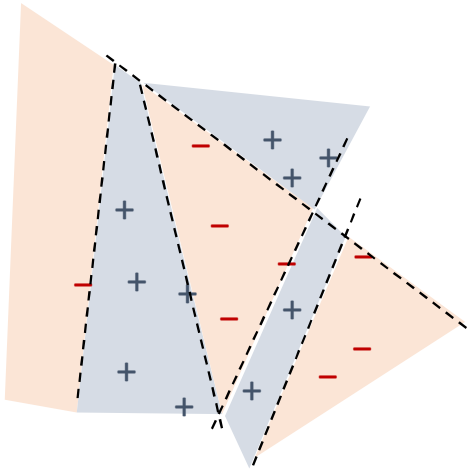


- **Output layer**
  - Represent the output of the neural network
  - For a two class problem or regression with a 1-d output, we need only one output node
- **Hidden layer(s)**
  - Represent the intermediary nodes that divide the input space into regions with (soft) boundaries
  - These usually form a *hidden layer*
  - Usually, there is only one such layer
  - Given enough hidden nodes, we can model an arbitrary input-output relation.
- **Input layer**
  - Represent dimensions of the input vector (one node for each dimension)
  - These usually form an *input layer*, and
  - Usually there is only one such layer

Importance of hidden layers



Single sigmoid



Sigmoid hidden layers and sigmoid output

- First hidden layer extracts features
- Second hidden layer extracts features of features
- …
- Output layer gives the desired output

Overall function of a neural network

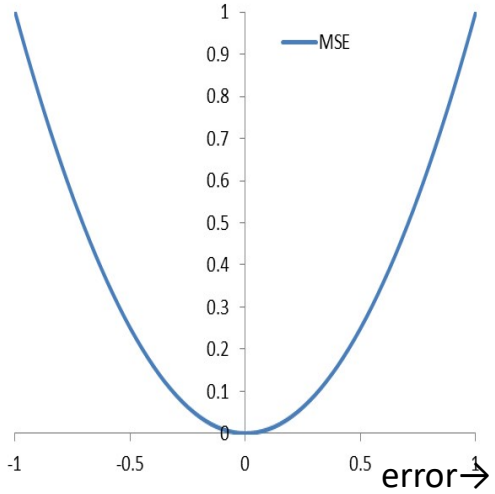
- $f(\boldsymbol{x}_i) = g_l(\boldsymbol{W}_l * g_{l-1}(\boldsymbol{W}_{l-1} \ldots g_1(\boldsymbol{W}_1 * \boldsymbol{x}_i) \ldots))$
- Weights form a matrix
- Output of the previous layer form a vector
- The activation (nonlinear) function is applied point-wise to the weight times input

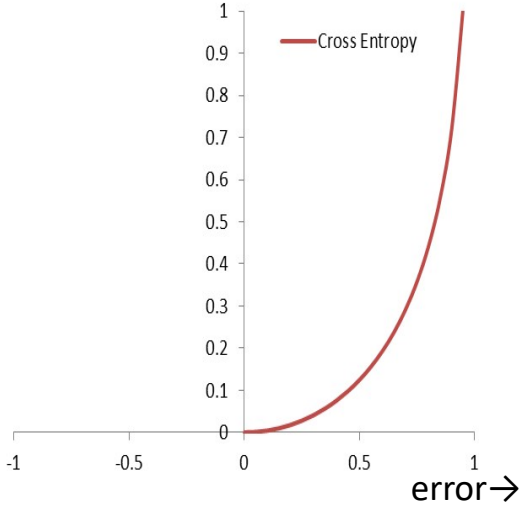
- Design questions (hyper parameters):
  - Number of layers
  - Number of neurons in each layer (rows of weight matrices)

Training the neural network

- Given $\boldsymbol{x}_i$ and y_i
- Think of what hyper-parameters and neural network design might work
- Form a neural network:
  $$f(\boldsymbol{x}_i) = g_l(\boldsymbol{W}_l * g_{l-1}(\boldsymbol{W}_{l-1} \ldots g_1(\boldsymbol{W}_1 * \boldsymbol{x}_i) \ldots))$$
- Compute $f_{\boldsymbol{w}}(\boldsymbol{x}_i)$ as an estimate of y_i for all samples
- Compute loss: $\frac{1}{N}\sum_{i=1}^{N} L(f_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i) = \frac{1}{N}\sum_{i=1}^{N} l_i(\boldsymbol{w})$
- Tweak $\boldsymbol{w}$ to reduce loss (optimization algorithm)
- Repeat last three steps

Loss function choice



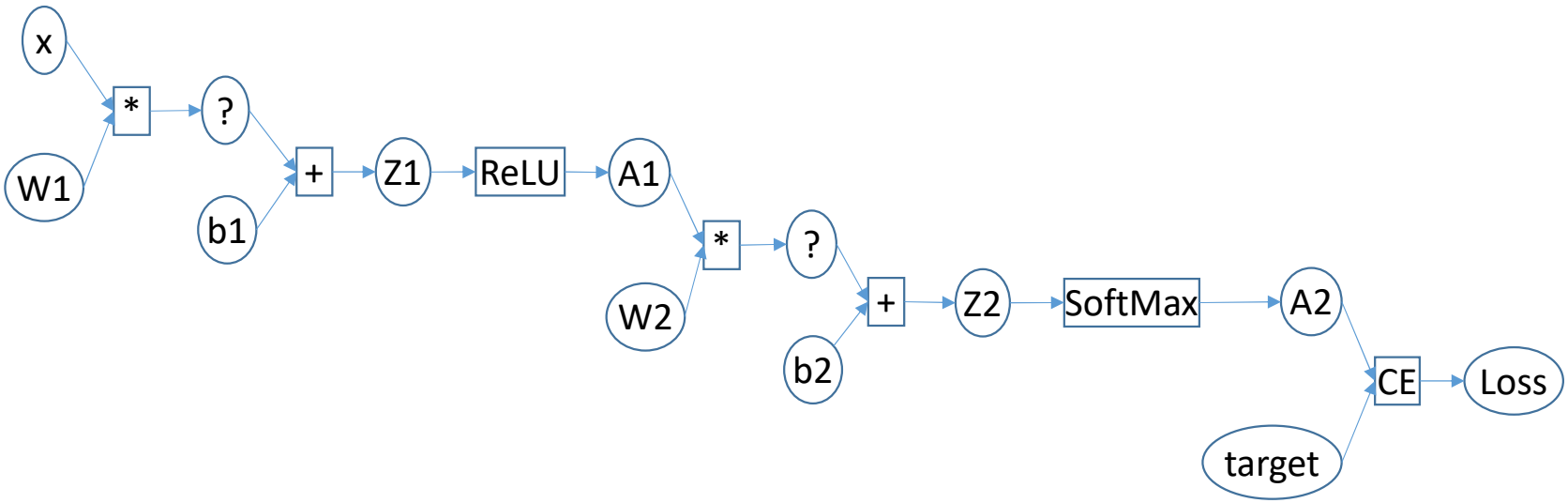


- There are positive and negative errors in classification and MSE is the most common loss function
- There is probability of correct class in classification, for which cross entropy is the most common loss function

Some loss functions and their derivatives

- Terminology
  - y is the output
  - t is the target output
- Mean square error
  - Loss: $(y - t\,)^2$
  - Derivative of the loss: $2(y - t\,)$
- Cross entropy
  - Loss: $-\sum_{c=1}^{C} t_c \log y_c$
  - Derivative of the loss: $-\frac{1}{y_c}\ \big|_{c=\omega}$

Computational graph of a single hidden layer NN



Overall function of a neural network

- $f(\boldsymbol{x}_i) = g_l(\boldsymbol{W}_l * g_{l-1}(\boldsymbol{W}_{l-1} \ldots g_1(\boldsymbol{W}_1 * \boldsymbol{x}_i) \ldots))$
- Weights form a matrix
- Output of the previous layer form a vector
- The activation (nonlinear) function is applied point-wise to the weight times input

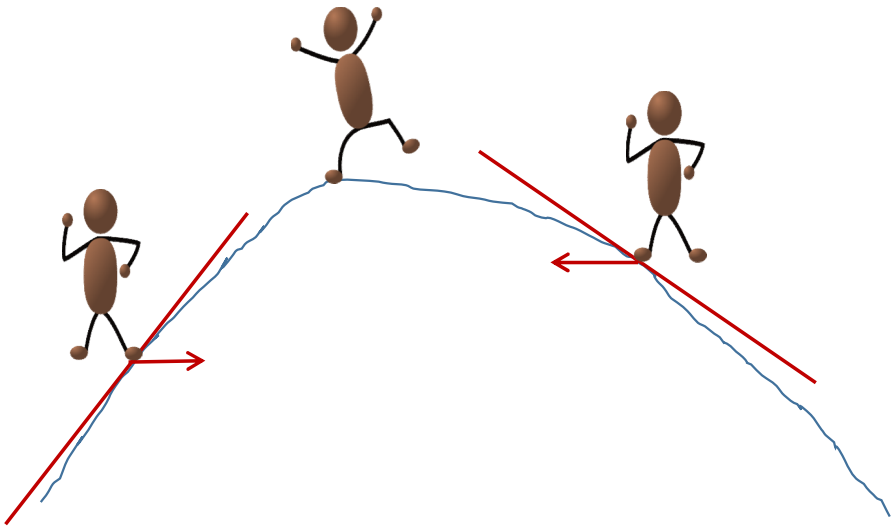
- Design questions (hyper parameters):
  - Number of layers
  - Number of neurons in each layer (rows of weight matrices)

Training the neural network

- Given $\boldsymbol{x}_i$ and y_i
- Think of what hyper-parameters and neural network design might work
- Form a neural network:
  $$f(\boldsymbol{x}_i) = g_l(\boldsymbol{W}_l * g_{l-1}(\boldsymbol{W}_{l-1} \ldots g_1(\boldsymbol{W}_1 * \boldsymbol{x}_i) \ldots))$$
- Compute $f_{\boldsymbol{w}}(\boldsymbol{x}_i)$ as an estimate of y_i for all samples
- Compute loss: $\frac{1}{N}\sum_{i=1}^{N} L(f_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i) = \frac{1}{N}\sum_{i=1}^{N} l_i(\boldsymbol{w})$
- Tweak $\boldsymbol{w}$ to reduce loss (optimization algorithm)
- Repeat last three steps

Gradient ascent

- If you didn’t know the shape of a mountain
- But at every step you knew the slope
- Can you reach the top of the mountain?



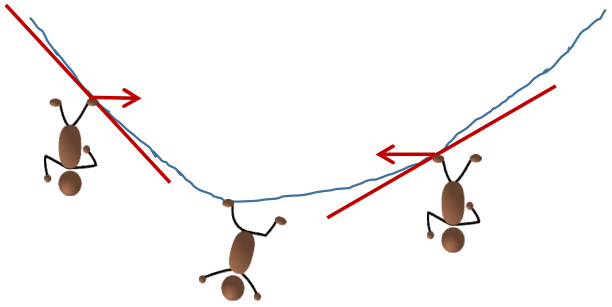
Gradient descent minimizes the loss function

- At every point, compute
  - Loss (scalar): $l_i(\boldsymbol{w})$
  - Gradient of loss with respect to weights (vector):

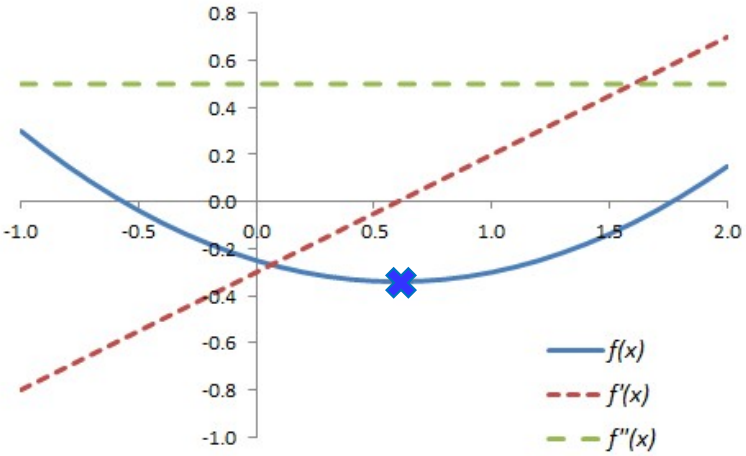
$$\nabla_{\boldsymbol{w}} l_i(\boldsymbol{w})$$

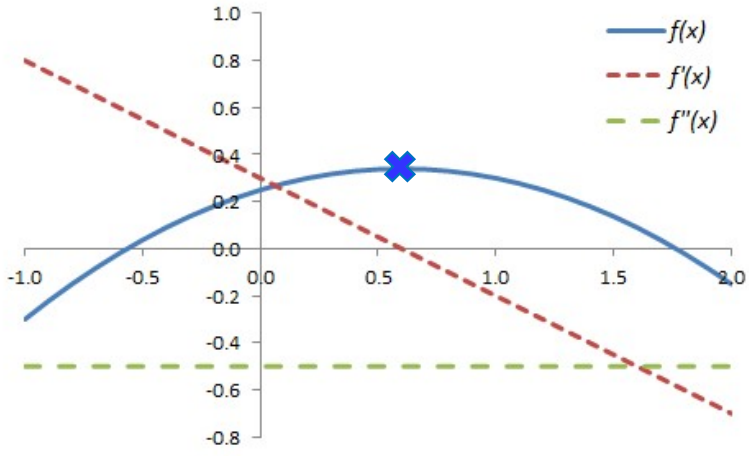
  - Take a step towards negative gradient:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla_{\boldsymbol{w}} \left(\frac{1}{N} \sum_{i=1}^{N} l_i(\boldsymbol{w}) \right)$$



Derivative of a function of a scalar

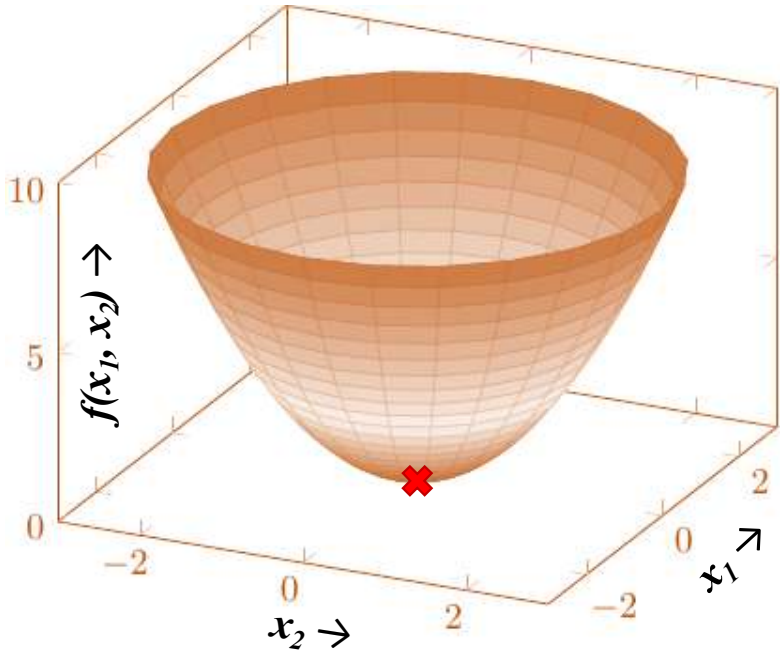




E.g. $f(x) = ax^2 + bx + c, \quad f'(x) = 2ax + b, \quad f''(x) = 2a$

- Derivative $f'(x) = \frac{d\,f(x)}{d\,x}$ is the rate of change of $f(x)$ with x
- It is zero when then function is flat (horizontal), such as at the minimum or maximum of $f(x)$
- It is positive when $f(x)$ is sloping up, and negative when $f(x)$ is sloping down
- To move towards the maxima, taking a small step in a direction of the derivative

Gradient of a function of a vector

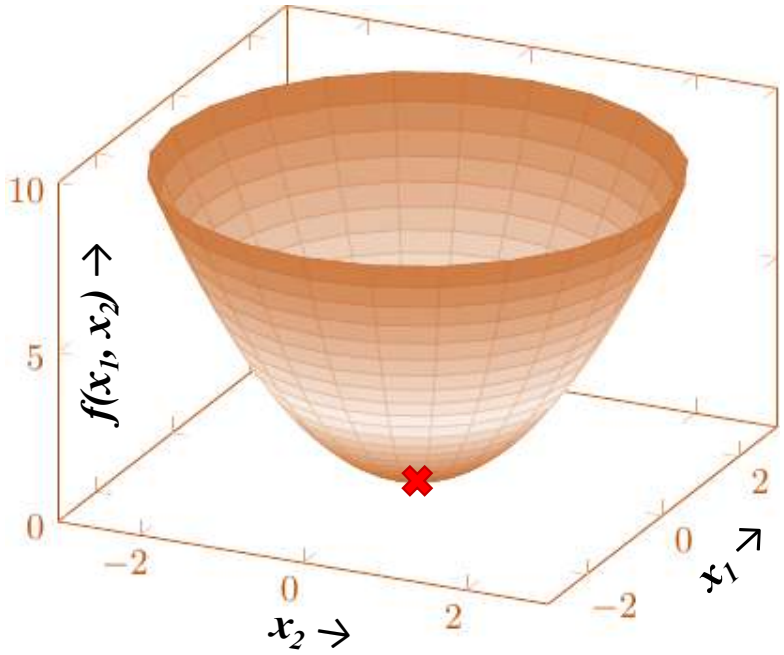


- Derivative with respect to each dimension, holding other dimensions constant
- $\nabla f(\boldsymbol{x}) = \nabla f(x_1, x_2) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$
- At a minima or a maxima the gradient is a zero vector

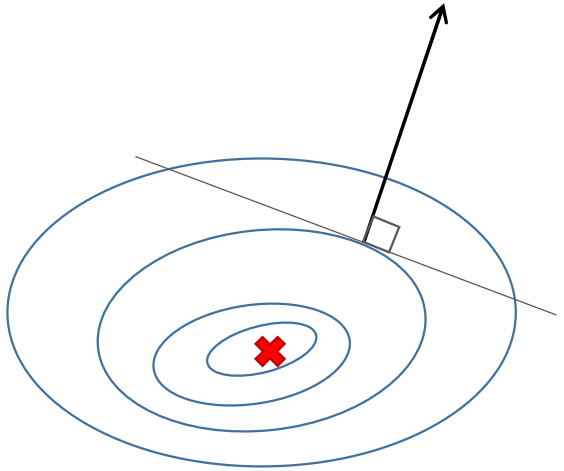
  The function is flat in every direction
- At a minima or a maxima the gradient is a zero vector

Original image source unknown

Gradient of a function of a vector



- Gradient gives a direction for moving towards the minima
- Take a small step towards negative of the gradient

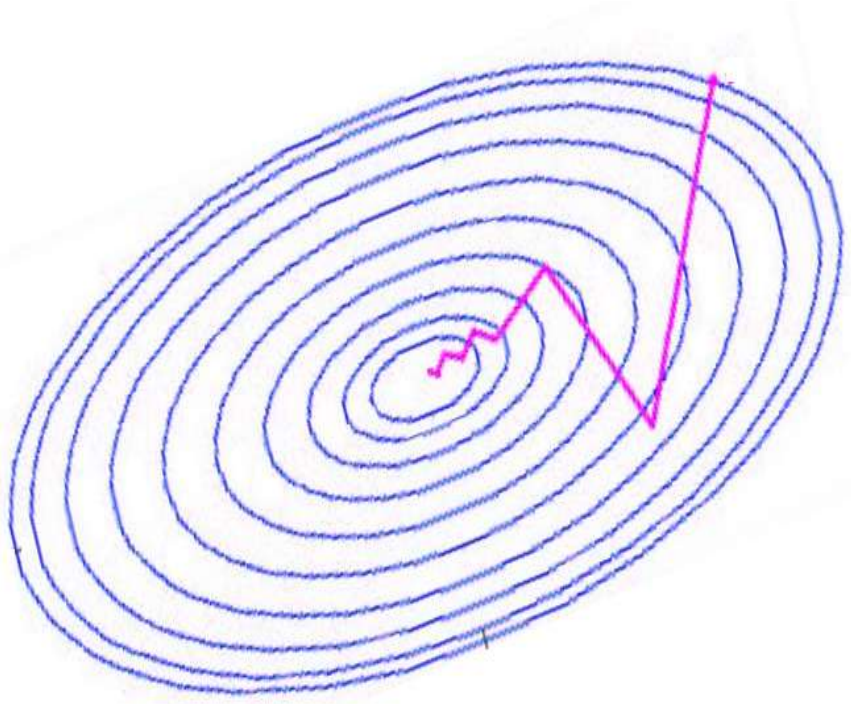


Original image source unknown

Example of gradient

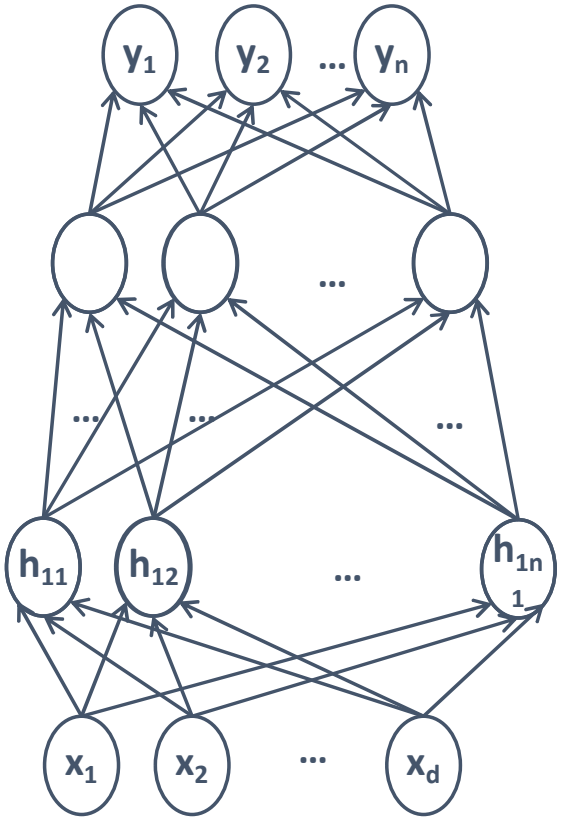
- Let $f(\boldsymbol{x}) = f(x_1, x_2) = 5{x_1}^2 + 3{x_2}^2$
- Then $\nabla f(\boldsymbol{x}) = \nabla f(x_1, x_2) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 10x_1 \\ 6x_2 \end{bmatrix}$
- At a location $(2,1)$ a step in $\begin{bmatrix} 20 \\ 6 \end{bmatrix}$ or $\begin{bmatrix} 0.958 \\ 0.287 \end{bmatrix}$ direction will lead to maximal increase in the function

This story is unfolding in multiple dimensions



Original image source unknown

Backpropagation



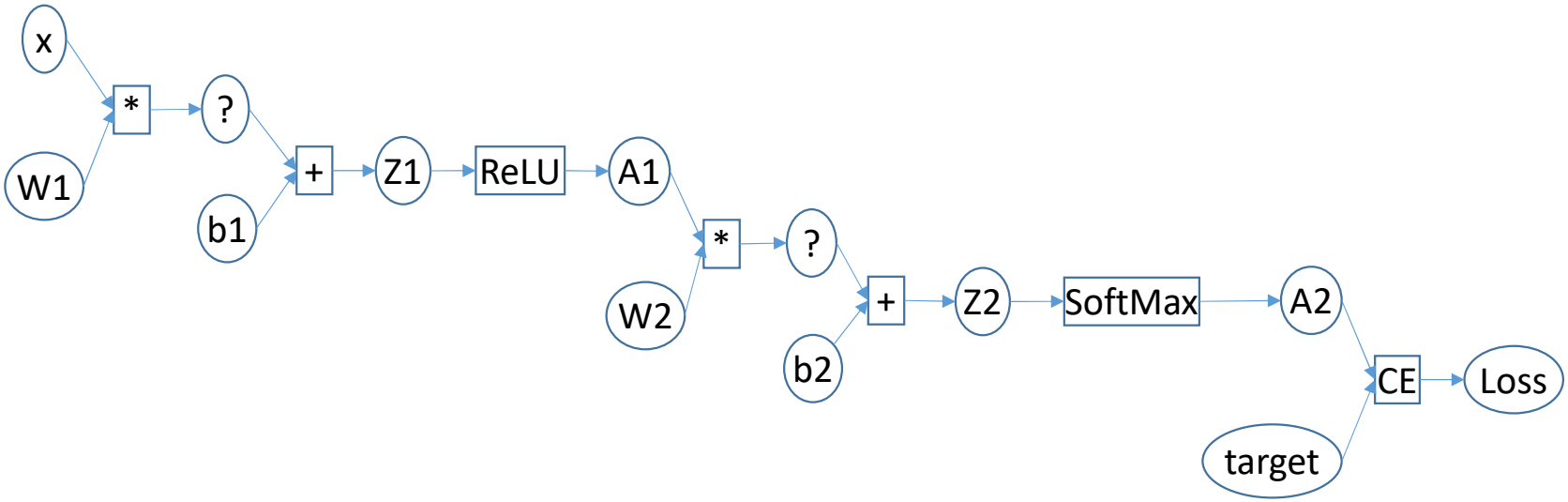
- Backpropagation is an efficient method to do gradient descent
- It saves the gradient w.r.t. the upper layer output to compute the gradient w.r.t. the weights immediately below
- It is linked to the chain rule of derivatives
- All intermediary functions must be differentiable, including the activation functions

Chain rule of differentiation

- Very handy for complicated functions
  - Especially functions of functions
  - E.g. NN outputs are functions of previous layers
- For example: Let $f(x) = g\big(h(x)\big)$
  - Let $y = h(x),\ z = g(y) = g\big(h(x)\big)$
- Then $f'(x) = \frac{d\,z}{d\,x} = \frac{d\,z}{d\,y}\frac{d\,y}{d\,x} = g'(y)h'(x)$
- For example: $\frac{d\,\sin(x^2)}{d\,x} = 2x\cos(x^2)$

Backpropagation makes use of chain rule of derivatives

- Chain rule: $\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial x}$



Vector valued functions and Jacobians

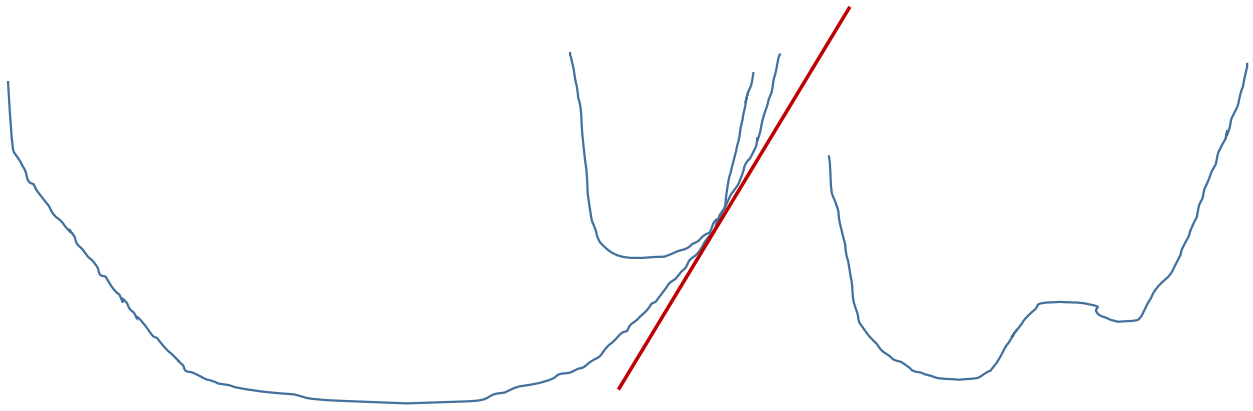
- We often deal with functions that give multiple outputs
- Let $\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} f_1(x_1, x_2, x_3) \\ f_2(x_1, x_2, x_3) \end{bmatrix}$
- Thinking in terms of vector of functions can make the representation less cumbersome and computations more efficient
- Then the Jacobian is
- $\boldsymbol{J}(\boldsymbol{f}) = \begin{bmatrix} \frac{\partial \boldsymbol{f}}{\partial x_1} & \frac{\partial \boldsymbol{f}}{\partial x_2} & \frac{\partial \boldsymbol{f}}{\partial x_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \end{bmatrix}$

Jacobian of each layer

- Compute the derivatives of a higher layer's output with respect to those of the lower layer
- What if we scale all the weights by a factor *R?*
- What happens a few layers down?

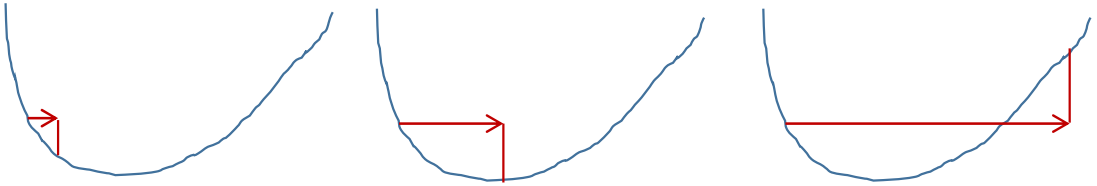
Role of step size and learning rate

- Tale of two loss functions
  - Same value, and
  - Same gradient (first derivative), but
  - Different Hessian (second derivative)
  - Different step sizes needed
- Success not guaranteed



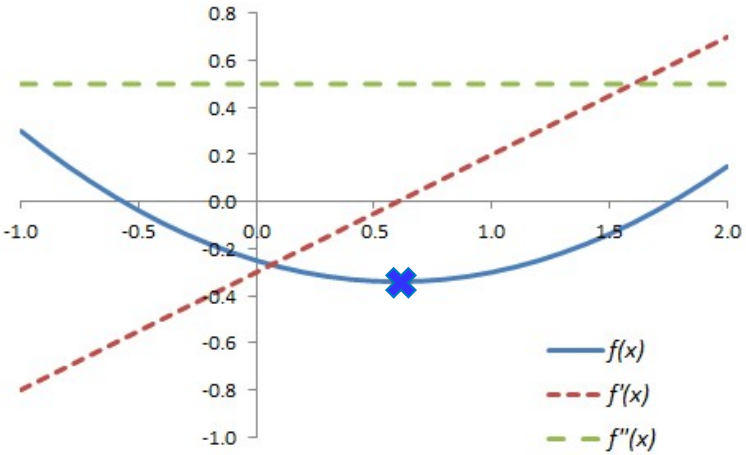
The perfect step size is impossible to guess

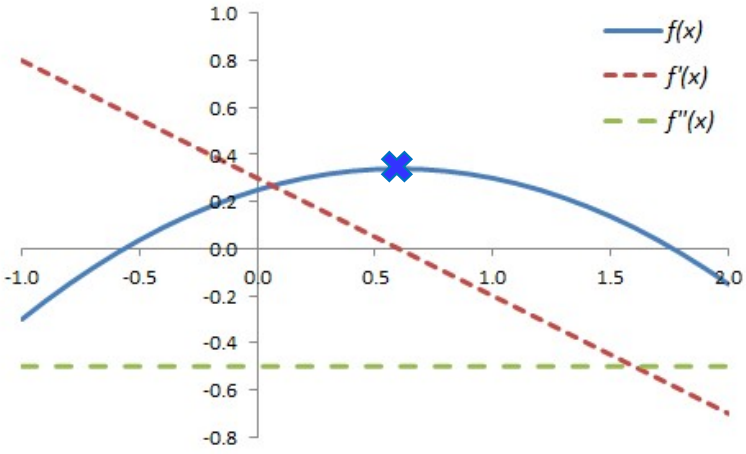
- Goldilocks finds the perfect balance only in a fairy tale



- The step size is decided by learning rate η and the gradient

Double derivative

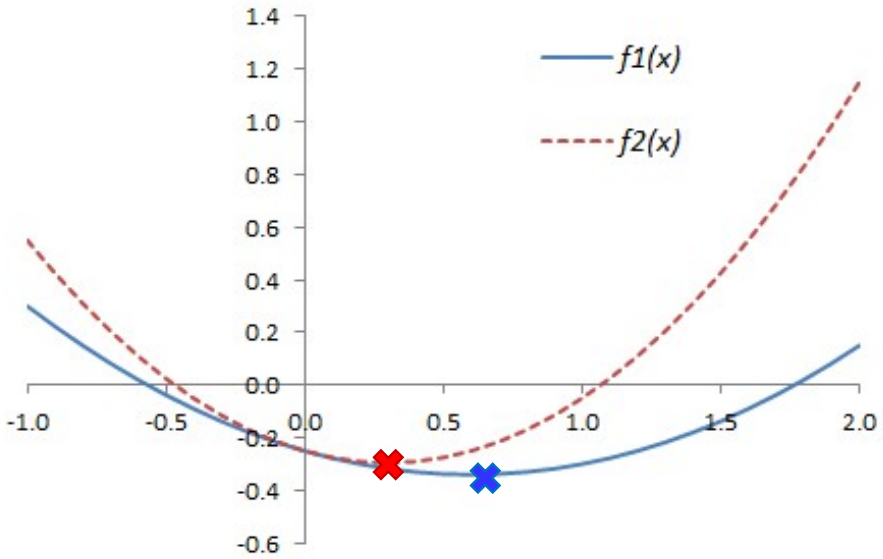




E.g. $f(x) = ax^2 + bx + c, \quad f'(x) = 2ax + b, \quad f''(x) = 2a$

- Double derivative $f''(x) = \frac{d^2 f(x)}{d x^2}$ is the derivative of derivative of $f(x)$
- Double derivative is positive for convex functions (have a single minima), and negative for concave functions (have a single maxima)

Double derivative



$$f(x) = ax^2 + bx + c,$$
$$f'(x) = 2ax + b,$$
$$f''(x) = 2a$$

- Double derivative tells how far the minima might be from a given point.
- From $x = 0$ the minima is closer for the red dashed curve than for the blue solid curve, because the former has a larger second derivative (its slope reverses faster)

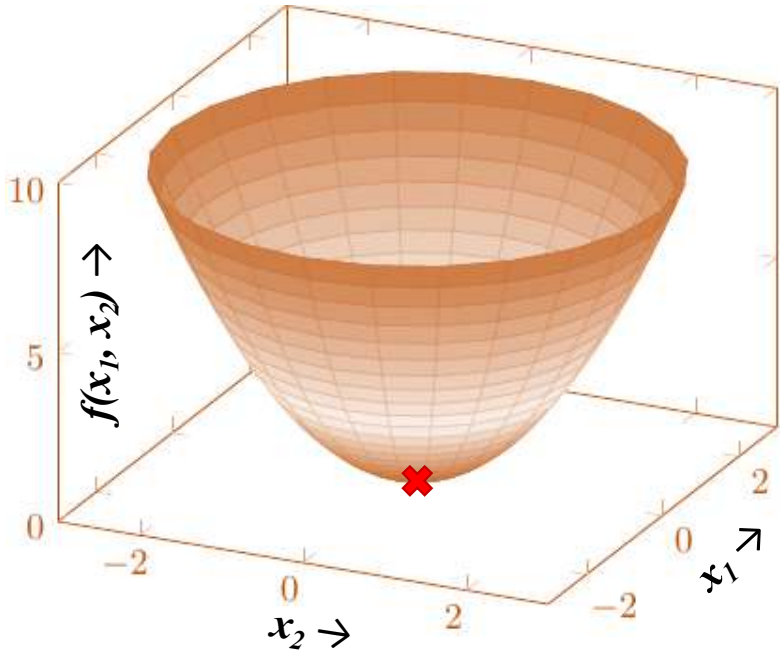
Perfect step size for a paraboloid

- Let $f(x) = ax^2 + bx + c$
- Assuming $a < 0$
- Minima is at: $x^* = -\frac{b}{2a}$
- For any x the perfect step would be:

$$-\frac{b}{2a} - x = -\frac{2ax+b}{2a} = -\frac{f'(x)}{f''(x)}$$

- So, the perfect learning rate is: $\eta^* = \frac{1}{f''(x)}$
- In multiple dimensions, $\boldsymbol{x} \leftarrow \boldsymbol{x} - H\big(f(\boldsymbol{x})\big)^{-1} \nabla(f(\boldsymbol{x}))$
- Practically, we do not want to compute the inverse of a Hessian matrix, so we approximate Hessian inverse

Hessian of a function of a vector



- Double derivative with respect to a pair of dimensions forms the Hessian matrix:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\, \partial x_n} \\ \frac{\partial^2 f}{\partial x_2\, \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\, \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\, \partial x_1} & \frac{\partial^2 f}{\partial x_n\, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- If all eigenvalues of a Hessian matrix are positive, then the function is convex

Original image source unknown

Example of Hessian

- Let $f(\boldsymbol{x}) = f(x_1, x_2) = 5{x_1}^2 + 3{x_2}^2 + 4x_1x_2$
- Then $\nabla f(\boldsymbol{x}) = \nabla f(x_1, x_2) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 10x_1 + 4x_2 \\ 6x_2 + 4x_1 \end{bmatrix}$
- And, $H(f(\boldsymbol{x})) = \begin{bmatrix} \frac{\partial^2 f}{\partial {x_1}^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial {x_2}^2} \end{bmatrix} = \begin{bmatrix} 10 & 4 \\ 4 & 6 \end{bmatrix}$

Saddle points, Hessian and long local furrows

- Some variables may have reached a local minima while others have not
- Some weights may have almost zero gradient
- At least some eigenvalues may not be negative

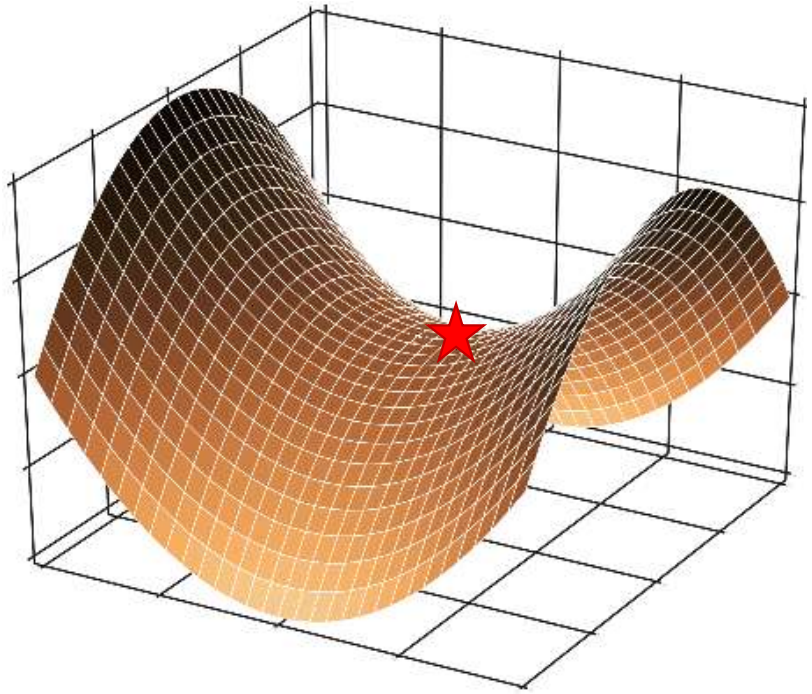
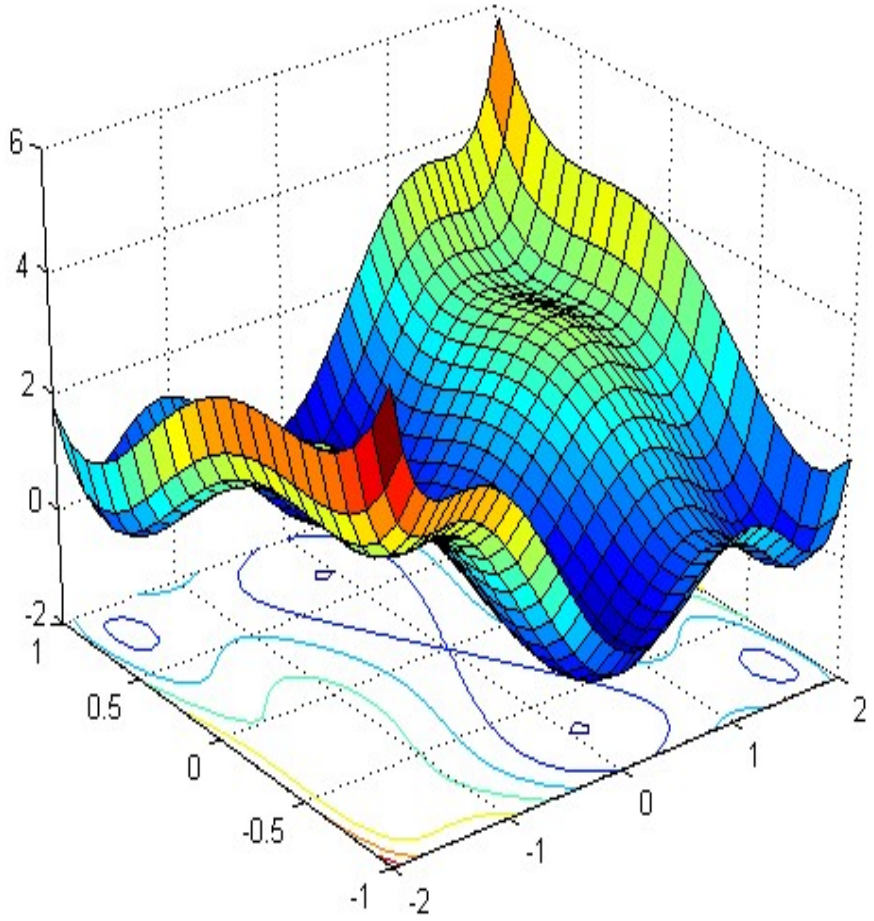


Image source: Wikipedia

Complicated loss functions



Original image source unknown

A realistic picture

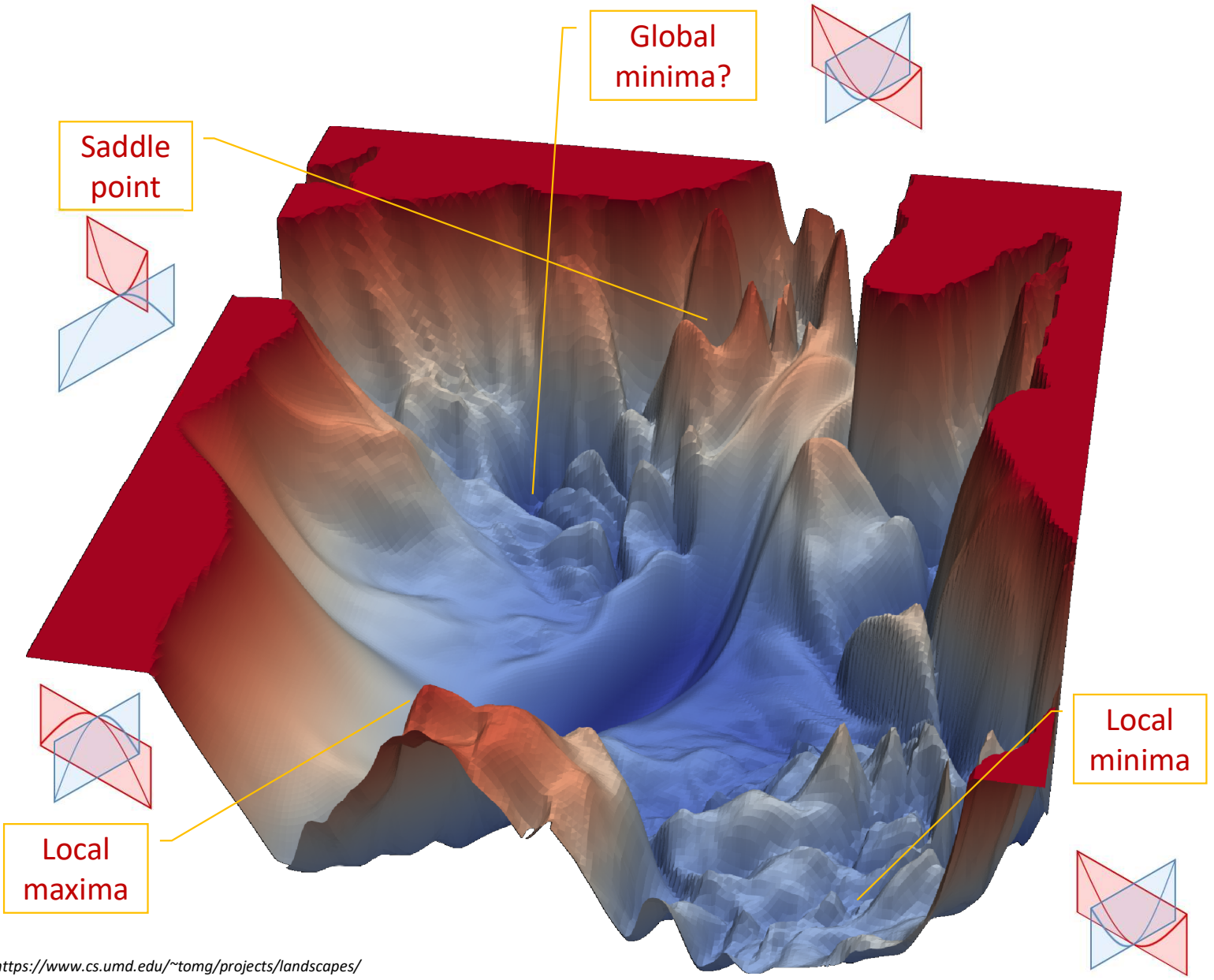


Image source: https://www.cs.umd.edu/~tomg/projects/landscapes/

Tentative list of topics

- Neural architectures - I
  - Vision
  - Audio
  - NLP
  - Graphs
- Training Methods - I
  - Convex optimization
  - Layers to help training
  - LR scheduling
  - Robustness
- Training methods - II
  - Lack of labels
  - Lack of training samples
  - Generative
  - Multi-modal
  - Pre-training and fine-tuning
  - Self-supervised learning
  - Semi and weak supervision

Next week

- Neural architectures for vision
  - LeNet
  - ResNet
  - UNet
  - YOLO

- Cover later
  - ViT
  - Swin Transformer