Computational Physics (PHYS6350)

Special Lecture: Introduction to machine learning

Based on Google machine learning crash course
https://developers.google.com/machine-learning/crash-course

April 10, 2025

Instructor: Volodymyr Vovchenko (vvovchenko@uh.edu), Tripp Moss

Course materials: https://github.com/vlvovch/PHYS6350-ComputationalPhysics

Key ML terminology

ML systems learn how to combine input to produce useful predictions on never-before-seen data

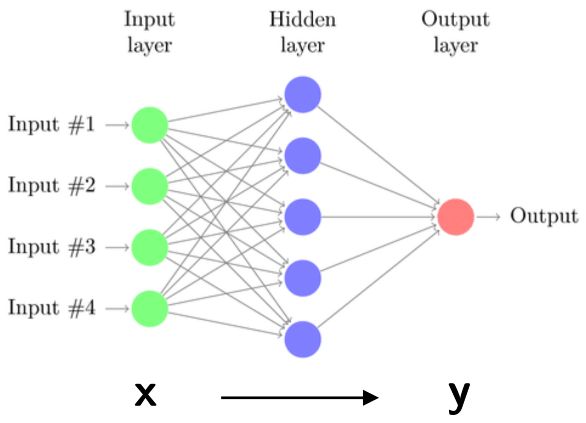
Features: input variables **x**

- e.g. coordinates and momenta of particles

Labels: a thing we're predicting, the **y** variable

- e.g. the system temperature T

ML models: produce the mapping $\mathbf{x} \longrightarrow \mathbf{y}$



Data:

- Labeled: contains both the features and the label(s)
- Unlabeled: contains only the features

Training – creating and learning the model, i.e. gradually learn the relationship between features and labels based on the labeled examples

Inference – applying the trained model to predict labels for unlabeled examples

Linear regression

Regression: predict continuous values

Linear regression – linear relationship between features and labels

$$y' = b + w_1 x_1$$

y' – the predicted label

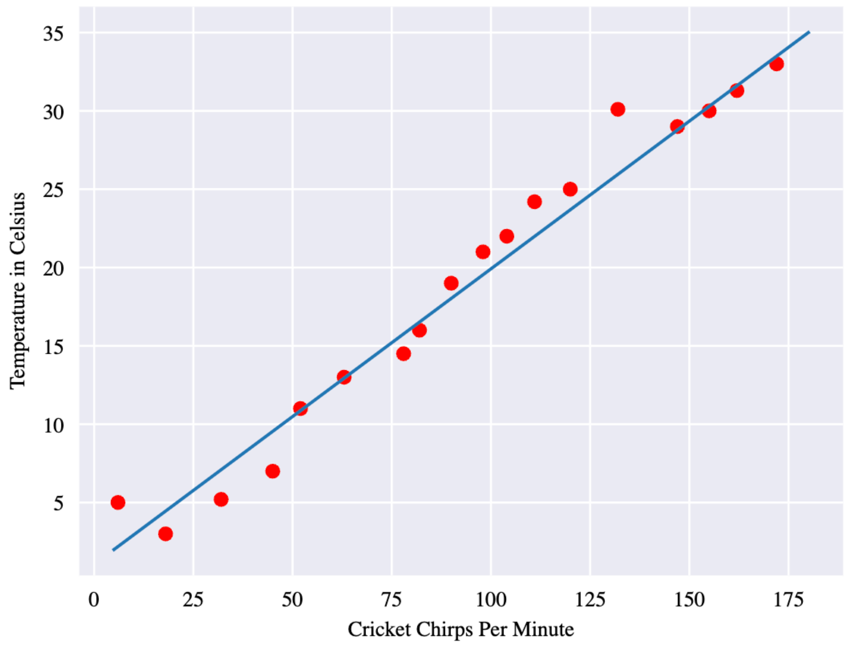
b – bias

w_1 – the weight of feature 1

x_1 – feature

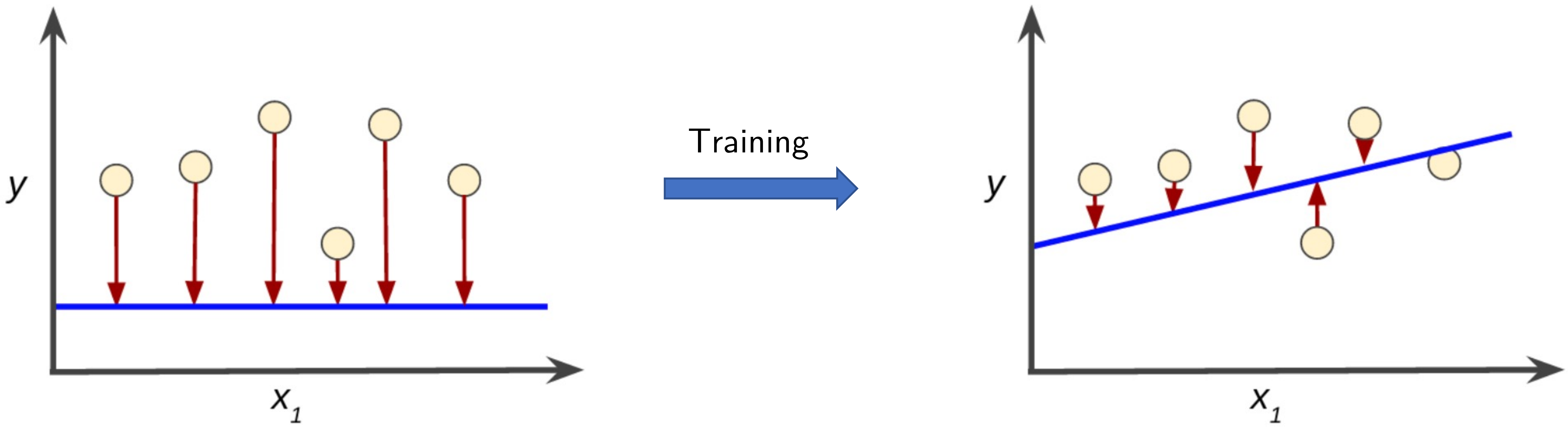
If more than one feature

$$y' = b + w_1 x_1 + w_2 x_2 + w_3 x_3$$



Training and loss

Training: learning good values for all the weights and bias from label examples



Introduce **loss function** – a measure of how bad the model prediction is, and minimize it

Common choice is the mean squared loss (**L_2 loss**)

$$MSE = \frac{1}{N} \sum_{(x,y)\in D} (y - prediction(x))^2$$

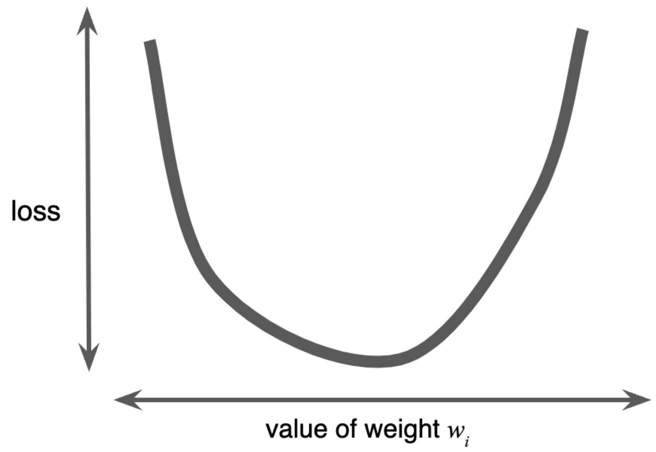
sum over labeled examples

Minimizing loss

Find weights w_i and bias b that minimize the loss function over the dataset

In principle can be achieved by solving the equations

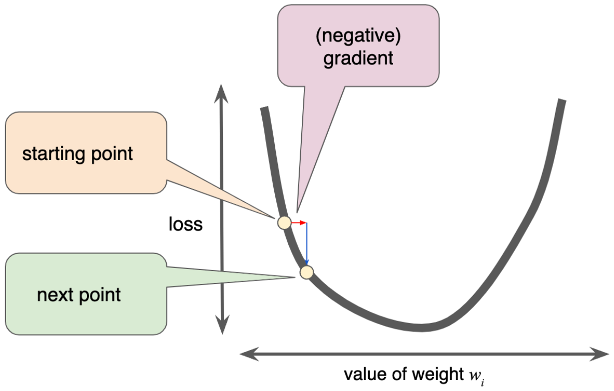
$$\frac{\partial L}{\partial w_i} = 0, \qquad \frac{\partial L}{\partial b} = 0 \; .$$



Issues:

- Becomes challenging for non-linear problems
- Does not scale well for large data sets and complex neural networks

Gradient descent: move the weights in the opposite direction of the gradient



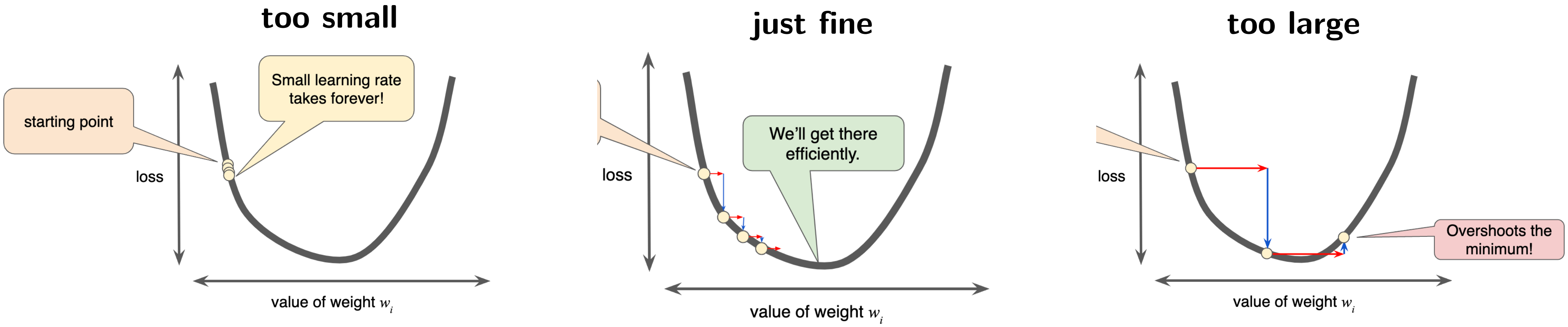
$$w_i \to w_i - k \frac{\partial L}{\partial w_i}$$

k is **learning rate**

Minimizing loss: learning rate

Learning rate is a knob that should be tweaked for ML algorithm to be efficient

$$w_i \to w_i - k\,\frac{\partial L}{\partial w_i}$$



Theoretical optimal value: $k = 1/(d^2L/dw^2)$ for 1d case or inverse Hessian (Jacobian) matrix for multi-dimensional

Equivalent to Newton's method of solving the system of (non-)linear equations $\quad \frac{\partial L}{\partial w_i} = 0, \qquad \frac{\partial L}{\partial b} = 0\ .$

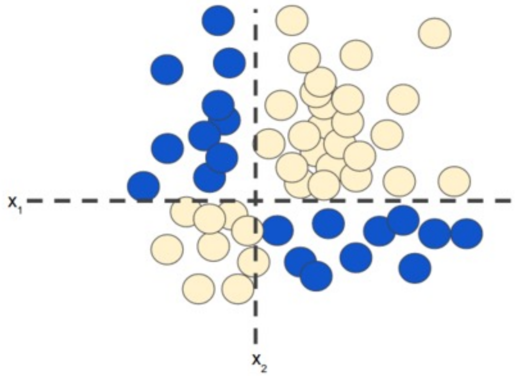
Stochastic gradient descent: randomly pick a fraction (batch) of data to estimate the loss at each step

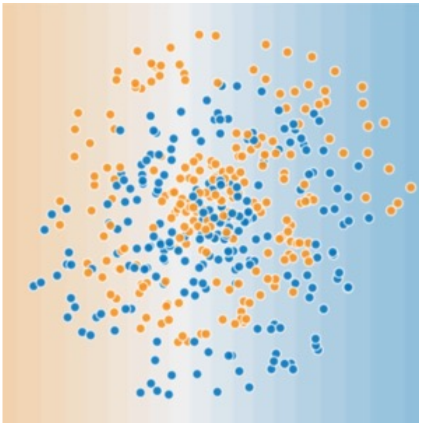
Neural networks and non-linear problems

So far we've dealt with linear regression problems

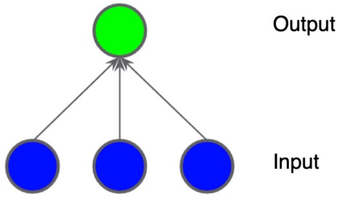
This can only get you so far. How to add non-linearity?

Non-linear problems

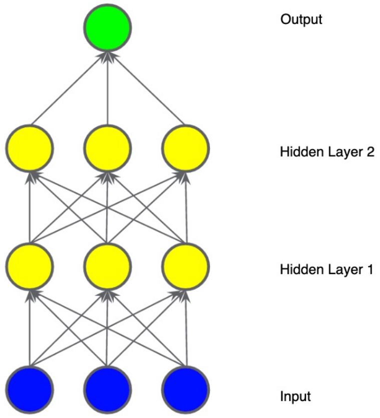




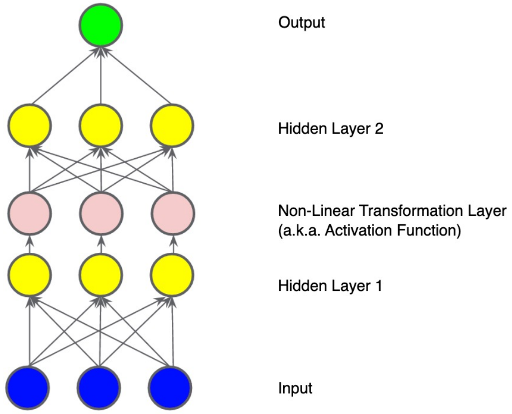
Linear



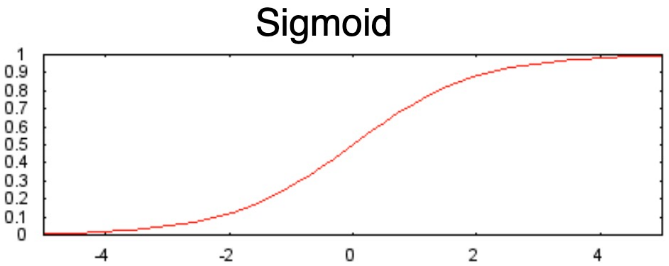
Still linear

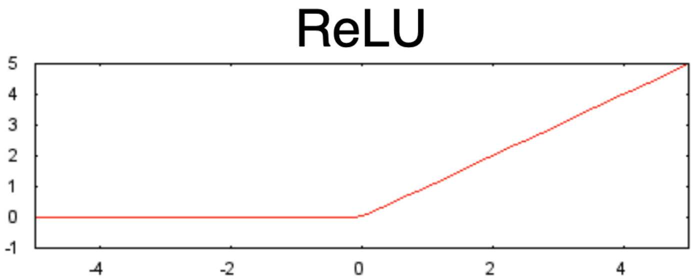


Non-linear!



Non-linearity is achieved through adding **activation functions** at intermediate layers





Train weights through **backpropagation** (chain rule for the derivatives)

Training and Test Sets

The data (labeled examples) are typically split into training and test sets

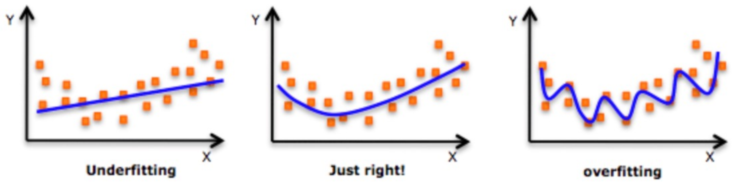
Training set: Data used to train the model
Test set: Data not used for training but for validation of the model

Entries in training and test sets should be independent (no duplication) and statistically representative of the whole data

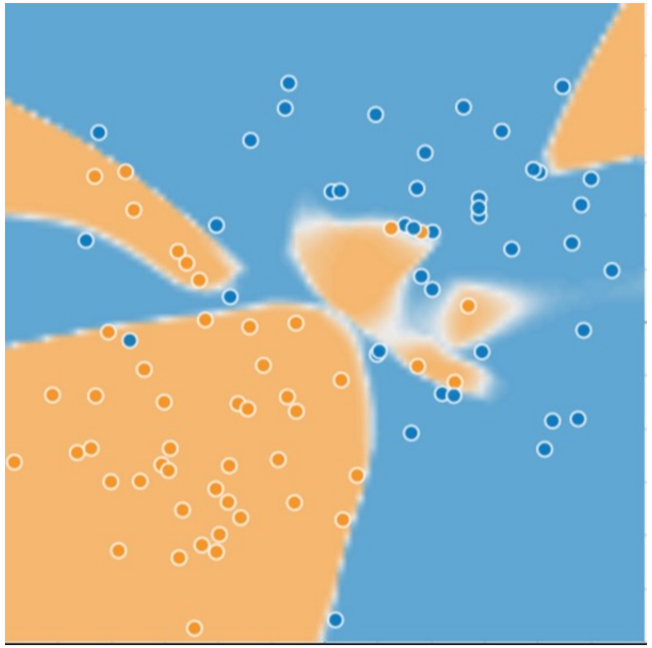
Some good practices:
- Data sanitization (remove duplicates etc.)
- Random shuffle of the order of the entries (in case they were originally sorted by some feature)
- Normalization (features should all have similar scale, e.g. numbers between 0 and 1)

Overfitting

Overfitting is a common ML problem that occurs when a model is too complicated and overfits the peculiarities of the training set



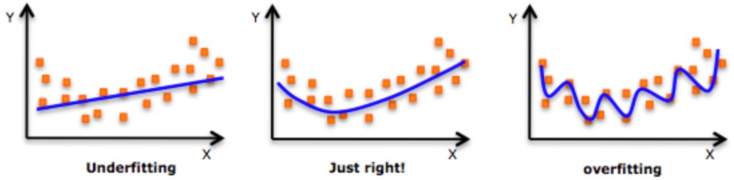
Training set



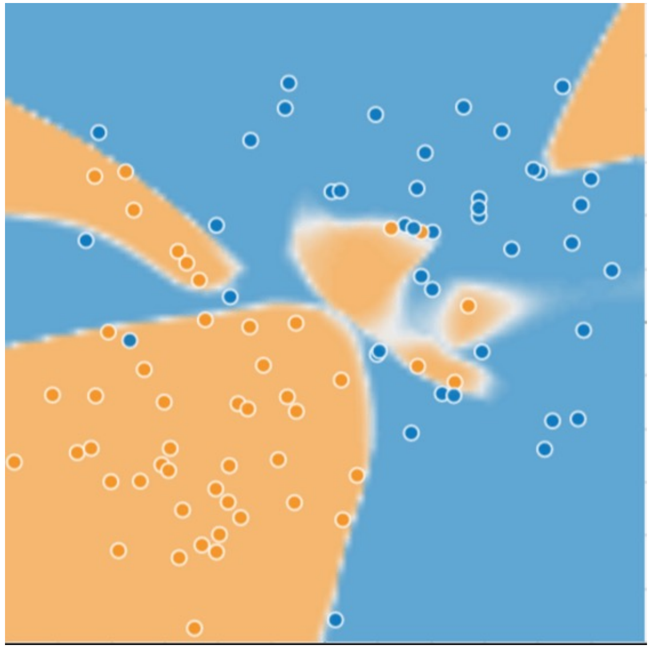
Model is complex enough to give a peculiar structure that models the training set well

Overfitting

Overfitting is a common ML problem that occurs when a model is too complicated and overfits the peculiarities of the training set

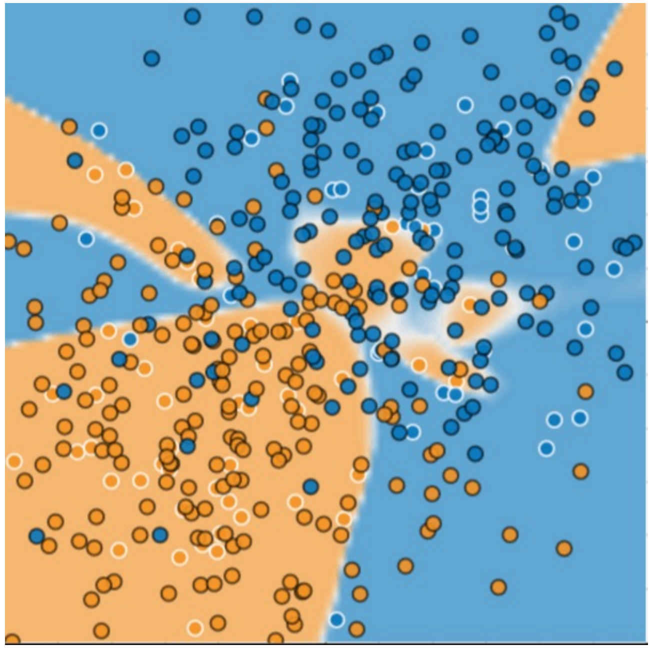


Training set



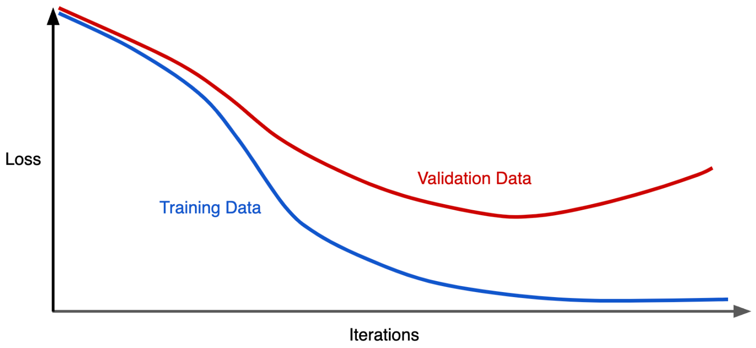
Model is complex enough to give a peculiar structure that models the training set well

Test set

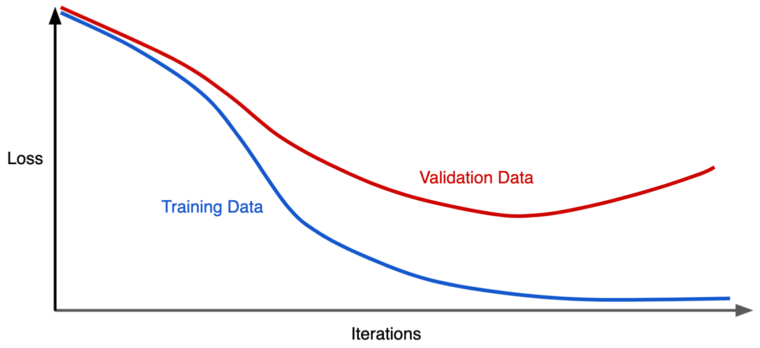


Bad performance on the test set

Generalization curve



Avoid overfitting through regularization



Overfitting occurs once the model becomes too complex (too many weights)

Occam's razor: search for the simplest possible explanation (applies here!)

Penalize complex models by introducing a *complexity term*

Instead of

$$\text{minimize}(\text{Loss}(\text{Data}|\text{Model}))$$

do the minimization of the sum

$$\text{minimize}(\text{Loss}(\text{Data}|\text{Model}) + \lambda\,\text{complexity}(\text{Model}))$$

λ – regularization rate

One possible measure of complexity is **L_2 regularization**

$$L_2 \text{ regularization term} = ||\boldsymbol{w}||_2^2 = w_1^2 + w_2^2 + \ldots + w_n^2$$

Prefers models with smaller amount of non-zero weights: **simpler models!**

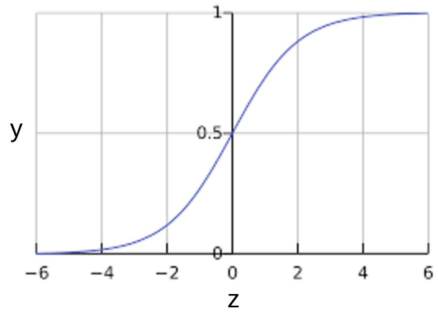
Logistic regression

Often we need the output to represent probability of some statement being true, e.g.

- The given image represents a dog
- The observed features of our system indicate that it is in a superconducting phase
- The given proton-proton collision produced the Higgs boson

To obtain a probability output, apply the sigmoid function to the output *z* of the final layer

$$y' = \frac{1}{1+e^{-z}} \qquad z = b + w_1 x_1 + w_2 x_2 + \ldots + w_N x_N$$



The loss function for logistic regression is log loss

$$\text{Log Loss} = \sum_{(x,y)\in D} -y\log(y') - (1-y)\log(1-y')$$

y – training set label (always 0 or 1)
y' – model prediction

Logistic regression models are prone to overfitting, thus regularization is important

Classification

Logistic regression returns a probability.
One can use this probability to make a binary classification:
if probability is larger than **classification threshold**, assign 1, otherwise 0

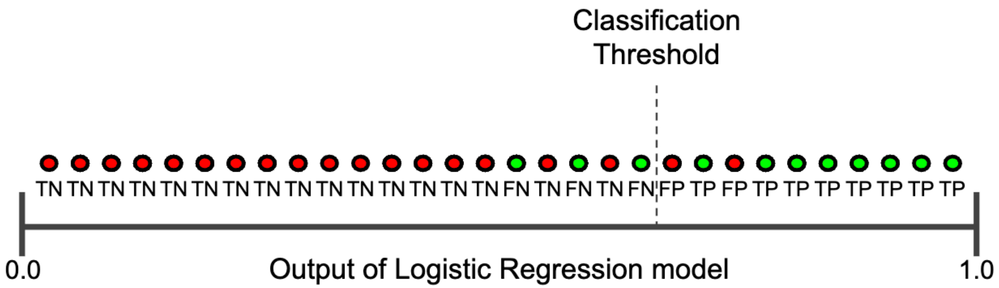
Starting choice for classification threshold can be 0.5, but it is not necessarily the optimum one

Metrics:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \qquad \text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN}$$

TP = True Positives, *TN* = True Negatives, *FP* = False Positives, and *FN* = False Negatives.

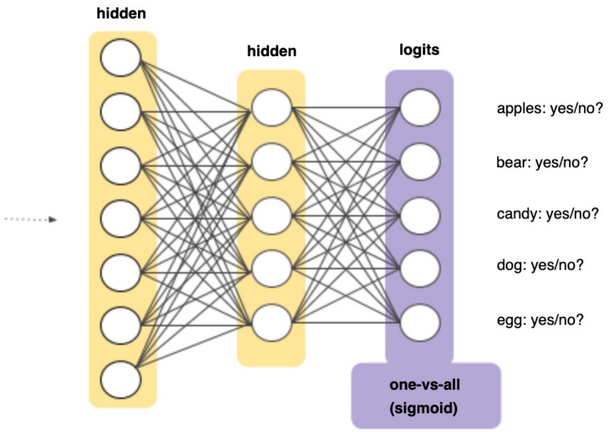
High accuracy may not be enough, also precision and recall matter (e.g. in the case where true positives are very rare but important to identify)



Multi-class classification

Sometimes we have to classify objects among multiple mutually exclusive classes.

- Digits from 0 to 9
- Animals
- Etc.



Achieved through

- Multiple output nodes (one per each class)
- Probability through **softmax equation**

$$p(y=j|\mathbf{x}) = \frac{e^{(\mathbf{w}_j^T\mathbf{x}+b_j)}}{\sum_{k\in K} e^{(\mathbf{w}_k^T\mathbf{x}+b_k)}}$$

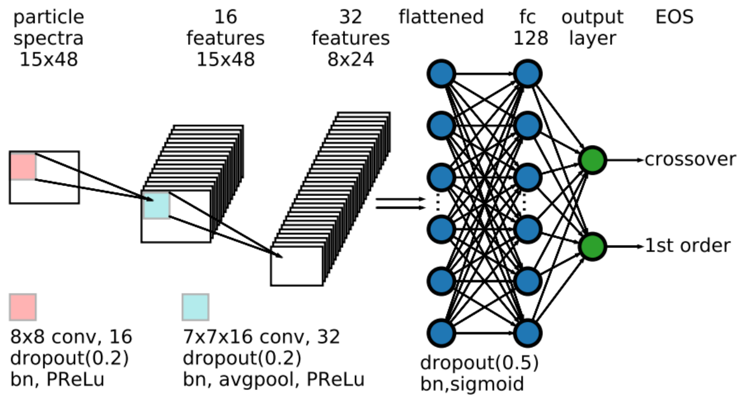
Classic example: MNIST problem (classification of hand-written digits)

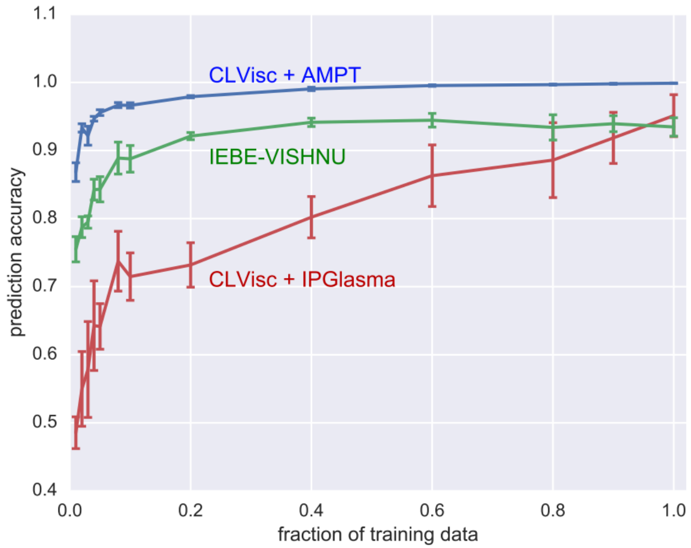
Google Colab notebook

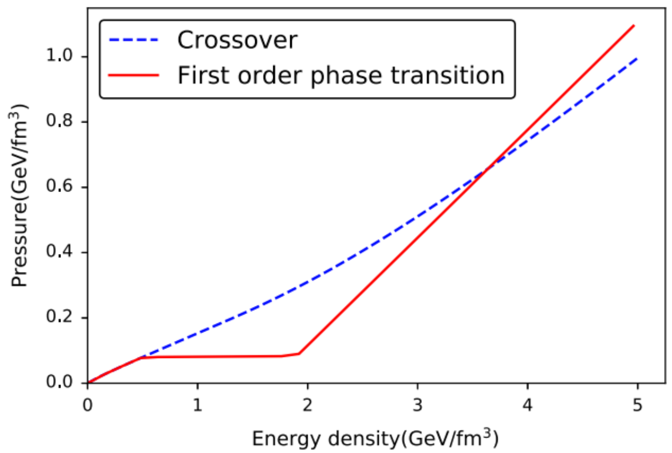
Example: Looking for the QCD phase transition

Open problem in QCD: is there a phase transition?

Models predict subtle differences in pion spectra in heavy-ion collisions







L.G. Pang, K. Zhou, N. Su, H. Petersen, H. Stoecker, X.-N. Wang, Nature Commun. 9, 210 (2018)

Neural network learns to identify the presence of phase transition in model studies

Plenty of other physics applications:

- QFT properties from lattice configurations
- Emulator of complex models/theories (e.g. hydrodynamics)
- Multi-parameter estimation