HELMHOLTZ AI | ARTIFICIAL INTELLIGENCE COOPERATION UNIT

Multivariate methods

Machine learning in particle physics

James Kahn (james.kahn@kit.edu)

Karlsruhe Institute of Technology / 2021-03-24

Goals

1. Understand the basic concepts of learning theory and machine learning (ML)
2. See how some of the algorithms actually work
3. Gently introduce modern deep learning
4. Glance over some ML examples in Belle II
5. If we get time:
   Highlight shortcomings/future research directions

Goals

1. Understand the basic concepts of learning theory and machine learning (ML)
2. See how some of the algorithms actually work
3. Gently introduce modern deep learning
4. Glance over some ML examples in Belle II
5. If we get time:
   Highlight shortcomings/future research directions

This will **not** be a tutorial on deep learning techniques (though we will look at the principles of neural networks)[1]

[1] This will also not be presented like a rockstar (sorry Florian)

Goals

1. Understand the basic concepts of learning theory and machine learning (ML)
2. See how some of the algorithms actually work
3. Gently introduce modern deep learning
4. Glance over some ML examples in Belle II
5. If we get time:
   Highlight shortcomings/future research directions

This will **not** be a tutorial on deep learning techniques (though we will look at the principles of neural networks)[1]

Overall: Want to give you a solid foundation to be able to understand/interpret the use of ML in physics

[1] This will also not be presented like a rockstar (sorry Florian)

About Me

Experimental particle physicist by training, now an AI researcher

- Master's with Belle in Australia ($B \to K_S^0 \pi^0$)

About Me

Experimental particle physicist by training, now an AI researcher

- Master's with Belle in Australia ($B \to K_S^0 \pi^0$)
- PhD with Belle II in Munich ($B \to K\nu\bar{\nu}$ and ML for simulation)

About Me

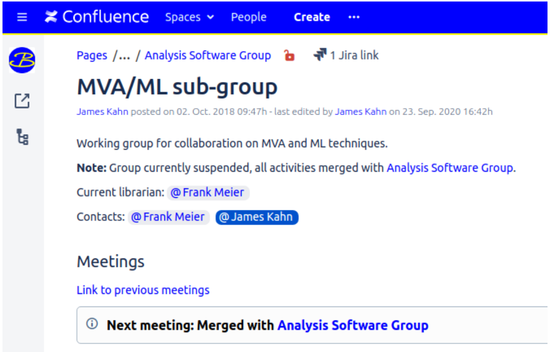
Experimental particle physicist by training, now an AI researcher

- Master's with Belle in Australia ($B \rightarrow K_S^0 \pi^0$)
- PhD with Belle II in Munich ($B \rightarrow K \nu \bar{\nu}$ and ML for simulation)
- First postdoc with Belle II/GridKa at KIT (Belle II grid computing liaison)

About Me

Experimental particle physicist by training, now an AI researcher

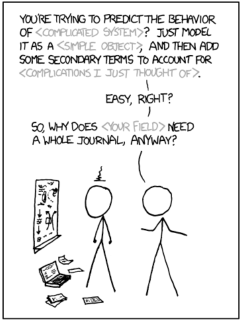
- Master's with Belle in Australia ($B \rightarrow K_S^0 \pi^0$)
- PhD with Belle II in Munich ($B \rightarrow K \nu \bar{\nu}$ and ML for simulation)
- First postdoc with Belle II/GridKa at KIT (Belle II grid computing liaison)
  - Also head of Belle II MVA group



About Me

Experimental particle physicist by training, now an AI researcher

- Master's with Belle in Australia ($B \to K_S^0 \pi^0$)
- PhD with Belle II in Munich ($B \to K\nu\bar{\nu}$ and ML for simulation)
- First postdoc with Belle II/GridKa at KIT (Belle II grid computing liaison)
    - Also head of Belle II MVA group
- Now an AI consultant for Helmholtz AI (energy focused)
  Still a technical member of Belle II



LIBERAL-ARTS MAJORS MAY BE ANNOYING SOMETIMES, BUT THERE'S *NOTHING* MORE OBNOXIOUS THAN A PHYSICIST FIRST ENCOUNTERING A NEW SUBJECT.

About Me

Experimental particle physicist by training, now an AI researcher

- Master's with Belle in Australia ($B \to K_S^0 \pi^0$)
- PhD with Belle II in Munich ($B \to K \nu \bar{\nu}$ and ML for simulation)
- First postdoc with Belle II/GridKa at KIT (Belle II grid computing liaison)
  - Also head of Belle II MVA group
- Now an AI consultant for Helmholtz AI (energy focused)
  Still a technical member of Belle II

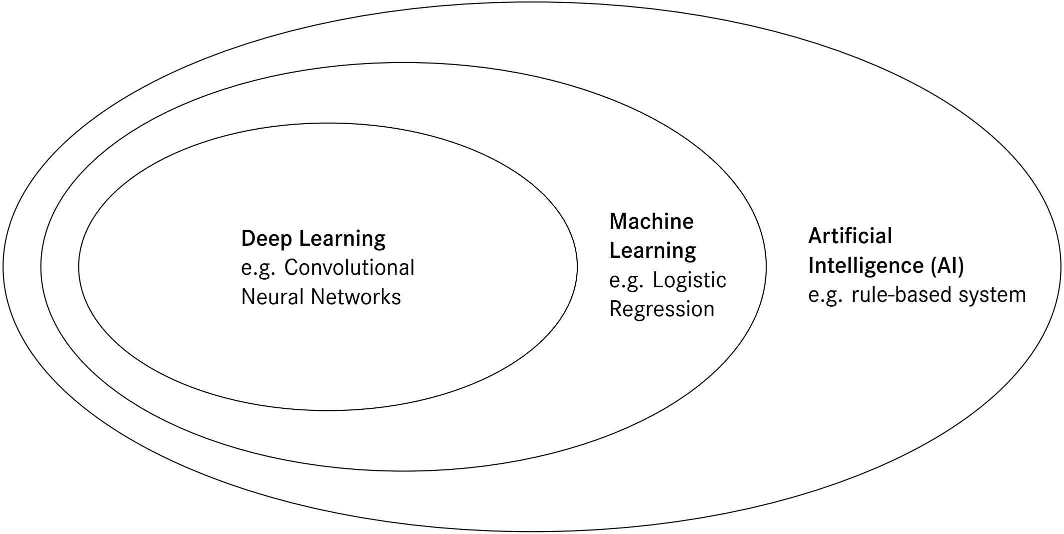
About Me

Experimental particle physicist by training, now an AI researcher

- Master's with Belle in Australia ($B \to K_S^0 \pi^0$)
- PhD with Belle II in Munich ($B \to K \nu \bar{\nu}$ and ML for simulation)
- First postdoc with Belle II/GridKa at KIT (Belle II grid computing liaison)
  - Also head of Belle II MVA group
- Now an AI consultant for Helmholtz AI (energy focused)
  Still a technical member of Belle II

If you have questions about transitioning out of physics but staying in academia don't hesitate to reach out:
Email: james.kahn@kit.edu Belle II rocket chat: `@jkahn` (General Kahnobi)

Terminology



Machine learning

From Patterns, predictions, and actions: A story about machine learning[1]:

> *This sets the stage for the subsequent chapters on what is now called machine learning: making near-optimal decisions from data alone, without probabilistic models of the environment.*

Representation learning

Use machine learning to transform input data into a new representation, learning to do so from the data alone. You gear this representation towards your needs.

In physics analysis: Decisions often amount to deciding on whether an event was signal or background*

Learning Approaches

Supervised learning: Learn by "mimicking supervisor", i.e. pattern annotations
examples: image classification, stock forecasting

Unsupervised learning: Determine patterns purely based on data
examples: customer cluster analysis, distribution estimation

Reinforcement learning: Pavlov-style learning with punishment and reward in dynamic environments
examples: game AIs, e.g. AlphaGo or Dota OpenAI

Learning Approaches

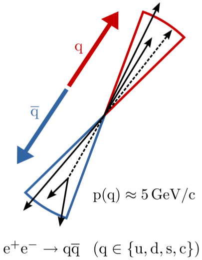
Supervised learning: Learn by “mimicking supervisor”, i.e. pattern annotations
examples: image classification, stock forecasting

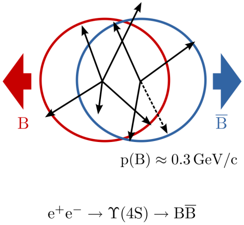
Unsupervised learning: Determine patterns purely based on data
examples: customer cluster analysis, distribution estimation

Reinforcement learning: Pavlov-style learning with punishment and reward in dynamic environments
examples: game AIs, e.g. AlphaGo or Dota OpenAI

BDTs and NNs are king
Some physics examples

BDT Continuum suppression



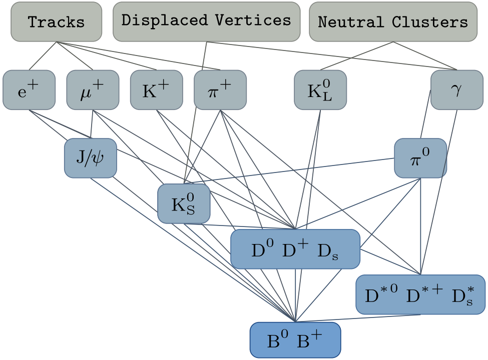


BDTs and NNs are king

Some physics examples

BDT Continuum suppression

BDT Full Event Interpretation



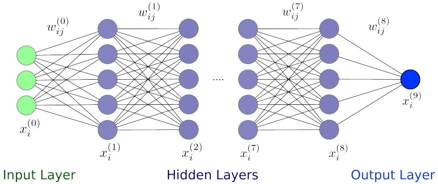
BDTs and NNs are king

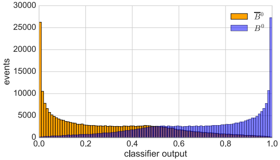
Some physics examples

BDT Continuum suppression

BDT Full Event Interpretation

NN Deep Flavor Tagger





BDTs and NNs are king

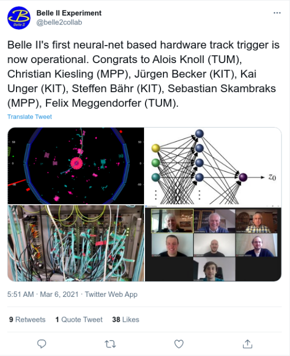
Some physics examples

BDT Continuum suppression

BDT Full Event Interpretation

NN Deep Flavor Tagger

NN Neuro-Z trigger



BDTs and NNs are king

Some physics examples

BDT Continuum suppression

BDT Full Event Interpretation

NN Deep Flavor Tagger

NN Neuro-Z trigger

- Reviews
  - Modern reviews
    - Jet Substructure at the Large Hadron Collider: A Review of Recent Adva
    - Deep Learning and its Application to LHC Physics
    - Machine Learning in High Energy Physics Community White Paper
    - Machine learning at the energy and intensity frontiers of particle physics
    - Machine learning and the physical sciences [DOI]
    - Machine and Deep Learning Applications in Particle Physics [DOI]
  - Specialized reviews
    - The Machine Learning Landscape of Top Taggers [DOI]
    - Dealing with Nuisance Parameters using Machine Learning in High Ene
    - Graph neural networks in particle physics [DOI]

For a **living review** of ML in HEP see [2].
See the *Machine Learning in High Energy Physics Community White Paper*[3] for an **LHC-focused** overview of methods and field-adoption.
Also: **IRIS-HEP** has regular meetings on ML in HEP.

Introduction to machine learning

ML basics

Introduction to ML will follow [1]:

1. Decision theory
   - Definitions
   - Likelihood ratio test
   - Neyman Pearson lemma
   - ROC curves
2. Supervised learning
   - IID assumption
   - Risk minimisation
   - The generalisation gap

Then look at:

- Fisher discriminant (a.k.a. ye olde class separator)
- Decision trees (fast and boosted)
- Neural networks (where all the cool kids are these days)

Decision theory

Use available information to
make a decision about an
unknown outcome

Decision theory

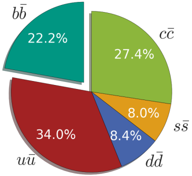
Use available information to make a decision about an unknown outcome

You will probably never use this

Decision theory

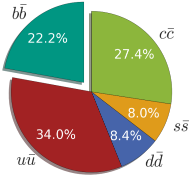
Basic definitions

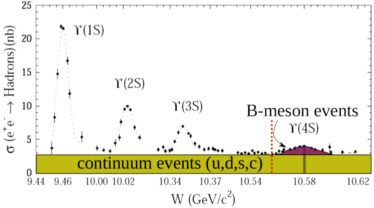
- Suppose we have two hypothesis: H_0 (background/continuum) and H_1 (signal)
- Each has some a priori probability: $p_0 = \mathbb{P}[H_0 \text{ is true}]$ $p_1 = \mathbb{P}[H_1 \text{ is true}]$



Basic definitions

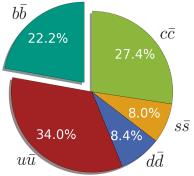
- Suppose we have two hypothesis: H_0 (background/continuum) and H_1 (signal)
- Each has some a priori probability: $p_0 = \mathbb{P}[H_0 \text{ is true}]$ $p_1 = \mathbb{P}[H_1 \text{ is true}]$
- Suppose we have some corresponding data $X \in \mathbb{R}^d$ which has a different distribution for H_0 and H_1
- That is: $p(x|H_i \text{ is true})$ forms a **likelihood function** under each scenario

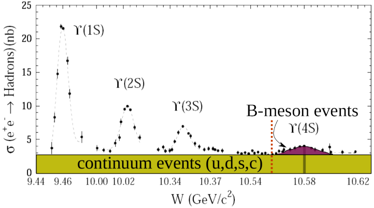




Basic definitions

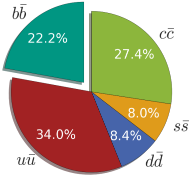
- Suppose we have two hypothesis: H_0 (background/continuum) and H_1 (signal)
- Each has some a priori probability: $p_0 = \mathbb{P}[H_0 \text{ is true}]$ $p_1 = \mathbb{P}[H_1 \text{ is true}]$
- Suppose we have some corresponding data $X \in \mathbb{R}^d$ which has a different distribution for H_0 and H_1
- That is: $p(x|H_i \text{ is true})$ forms a **likelihood function** under each scenario
- Goal: Optimise over algorithms (functions) that map data to decisions/predictions.

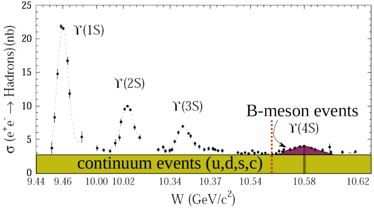




Basic definitions

- Suppose we have two hypothesis: H_0 (background/continuum) and H_1 (signal)
- Each has some a priori probability: $p_0 = \mathbb{P}[H_0 \text{ is true}]$ $p_1 = \mathbb{P}[H_1 \text{ is true}]$
- Suppose we have some corresponding data $X \in \mathbb{R}^d$ which has a different distribution for H_0 and H_1
- That is: $p(x|H_i \text{ is true})$ forms a **likelihood function** under each scenario
- Goal: Optimise over algorithms (functions) that map data to decisions/predictions.
- Do so by constructing and appropriate **cost** for each decision $\rightarrow$ minimise expected value of this cost





Labels

- In general: want simple, labelled data to use as learning examples for decision making.
- Luckily in HEP we have that thanks to simulations.

Definitions:

Labels

- In general: want simple, labelled data to use as learning examples for decision making.
- Luckily in HEP we have that thanks to simulations.

Definitions:

X The set of (multi-dimensional) input data (observables), e.g. $X \in \mathbb{R}^{H\times W\times C}$ for images

x_i Individual samples from the data, e.g. an individual image or event

Labels

- In general: want simple, labelled data to use as learning examples for decision making.
- Luckily in HEP we have that thanks to simulations.

Definitions:

X The set of (multi-dimensional) input data (observables), e.g. $X \in \mathbb{R}^{H \times W \times C}$ for images

x_i Individual samples from the data, e.g. an individual image or event

Y The set of ground-truth labels.

y_i Label corresponding to sample x_i, e.g. “signal” or “background”

Labels

- In general: want simple, labelled data to use as learning examples for decision making.
- Luckily in HEP we have that thanks to simulations.

Definitions:

X The set of (multi-dimensional) input data (observables), e.g. $X \in \mathbb{R}^{H \times W \times C}$ for images

x_i Individual samples from the data, e.g. an individual image or event

Y The set of ground-truth labels.

y_i Label corresponding to sample x_i, e.g. "signal" or "background"

$\hat{Y}$ The set of label predictions from our model/algorithm

f The algorithm that we are trying to optimise: takes in X and produces predictions $\hat{Y}$ (or what we use to make them)

Loss functions (risk)

- We construct a **loss function** which tells us the cost of declaring H_i when we have H_j as $\ell(i,j)$
- Define the risk associated with an algorithm f as $R[f] = \mathbb{E}\left[\ell(\hat{Y}(X), Y)\right]$
- Goal is to determine which f minimizes the risk

A Boolean example

Quick mafs

Say we have to decide if a given sample x belongs to H_0 or H_1, then our expected risks of each decision are:

$$\mathbb{E}[\ell(0,Y) \mid X=x] = \ell(0,0)\,\mathbb{P}[Y=0 \mid X=x] + \ell(0,1)\,\mathbb{P}[Y=1 \mid X=x]$$
$$\mathbb{E}[\ell(1,Y) \mid X=x] = \ell(1,0)\,\mathbb{P}[Y=0 \mid X=x] + \ell(1,1)\,\mathbb{P}[Y=1 \mid X=x]$$

A Boolean example
Quick mafs

Say we have to decide if a given sample x belongs to H_0 or H_1, then our expected risks of each decision are:

$$\mathbb{E}[\ell(0, Y) \mid X = x] = \ell(0,0)\,\mathbb{P}[Y = 0 \mid X = x] + \ell(0,1)\,\mathbb{P}[Y = 1 \mid X = x]$$
$$\mathbb{E}[\ell(1, Y) \mid X = x] = \ell(1,0)\,\mathbb{P}[Y = 0 \mid X = x] + \ell(1,1)\,\mathbb{P}[Y = 1 \mid X = x]$$

Optimal choice: pick whichever is smaller (costs less)

$$\hat{Y}(x) = \mathbb{1}\left\{\mathbb{E}[\ell(0, Y) \mid X = x] \geq \mathbb{E}[\ell(1, Y) \mid X = x]\right\}$$

A Boolean example

Quick mafs

Say we have to decide if a given sample x belongs to H_0 or H_1, then our expected risks of each decision are:

$$\mathbb{E}[\ell(0,Y) \mid X=x] = \ell(0,0)\,\mathbb{P}[Y=0 \mid X=x] + \ell(0,1)\,\mathbb{P}[Y=1 \mid X=x]$$
$$\mathbb{E}[\ell(1,Y) \mid X=x] = \ell(1,0)\,\mathbb{P}[Y=0 \mid X=x] + \ell(1,1)\,\mathbb{P}[Y=1 \mid X=x]$$

Optimal choice: pick whichever is smaller (costs less)

$$\hat{Y}(x) = \mathbb{1}\left\{\mathbb{E}[\ell(0,Y) \mid X=x] \geq \mathbb{E}[\ell(1,Y) \mid X=x]\right\}$$

rearrange...

$$\hat{Y}(x) = \mathbb{1}\left\{\mathbb{P}[Y=1 \mid X=x] \geq \frac{\ell(1,0)-\ell(0,0)}{\ell(0,1)-\ell(1,1)}\;\mathbb{P}[Y=0 \mid X=x]\right\}$$

A Boolean example
Quick mafs

Say we have to decide if a given sample x belongs to H_0 or H_1, then our expected risks of each decision are:

$$\mathbb{E}[\ell(0,Y) \mid X=x] = \ell(0,0)\,\mathbb{P}[Y=0 \mid X=x] + \ell(0,1)\,\mathbb{P}[Y=1 \mid X=x]$$
$$\mathbb{E}[\ell(1,Y) \mid X=x] = \ell(1,0)\,\mathbb{P}[Y=0 \mid X=x] + \ell(1,1)\,\mathbb{P}[Y=1 \mid X=x]$$

Optimal choice: pick whichever is smaller (costs less)

$$\hat{Y}(x) = \mathbb{1}\left\{\mathbb{E}[\ell(0,Y) \mid X=x] \geq \mathbb{E}[\ell(1,Y) \mid X=x]\right\}$$

rearrange...

$$\hat{Y}(x) = \mathbb{1}\left\{\mathbb{P}[Y=1 \mid X=x] \geq \frac{\ell(1,0)-\ell(0,0)}{\ell(0,1)-\ell(1,1)}\ \mathbb{P}[Y=0 \mid X=x]\right\}$$

Substitute in **Bayes rule**:

$$\mathbb{P}[Y=i \mid X=x] = \frac{p(x \mid H_i \text{ is true})\,\mathbb{P}[H_i \text{ is true}]}{p(x)}$$

and get the **likelihood ratio test**:

$$\hat{Y}(x) = \mathbb{1}\left\{\frac{p(x \mid H_1 \text{ is true})}{p(x \mid H_0 \text{ is true})} \geq \frac{p_0(\ell(1,0)-\ell(0,0))}{p_1(\ell(0,1)-\ell(1,1))}\right\}.$$

Likelihood ratio test

Likelihood ratio test:

$$\hat{Y}(x) = \mathbb{1}\left\{\frac{p(x \mid H_1 \text{ is true})}{p(x \mid H_0 \text{ is true})} \geq \frac{p_0(\ell(1,0) - \ell(0,0))}{p_1(\ell(0,1) - \ell(1,1))}\right\}.$$

Likelihood ratio:

$$\mathcal{L}(x) := \frac{p(x \mid H_1 \text{ is true})}{p(x \mid H_0 \text{ is true})}$$

And let:

$$\eta = \frac{p_0(\ell(1,0) - \ell(0,0))}{p_1(\ell(0,1) - \ell(1,1))}$$

Then the **risk-minimizing** decision rule is then:

$$\hat{Y}(x) = \mathbb{1}\{\mathcal{L}(x) \geq \eta\}$$

Likelihood ratio test

$$\hat{Y}(x) = \mathbb{1}\{\mathcal{L}(x) \geq \eta\}$$

divides any set of samples $\mathcal{X}$ into two unique partitions:

$$\begin{aligned}\mathcal{X}_0 &= \{x \in \mathcal{X} : \mathcal{L}(x) \leq \eta\} \\ \mathcal{X}_1 &= \{x \in \mathcal{X} : \mathcal{L}(x) > \eta\}\ .\end{aligned}$$

We want some function (model) $f : \mathcal{X} \to \mathbb{R}$ which produces the same partitions:

$$\hat{Y}_f(x) = \mathbb{1}\{f(\mathcal{L}(x)) \geq f(\eta)\} \approx \mathbb{1}\{\mathcal{L}(x) \geq \eta\}$$

Types of errors
Confusion matrix

Ground truth

Predicted value	H_1	H_0	**Total**
$\hat{Y}(X) = 1$	True Positive	False Positive	P′
$\hat{Y}(X) = 0$	False Negative	True Negative	N′
Total	P	N	

Types of errors
Confusion matrix

Predicted value	Ground truth: H_1	Ground truth: H_0	Total
$\hat{Y}(X) = 1$	True Positive	False Positive	P′
$\hat{Y}(X) = 0$	False Negative	True Negative	N′
Total	P	N	

True Positive Rate: (Recall)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$
$$= \mathbb{P}[\hat{Y}(X) = 1 \mid H_1 \text{ is true}]$$

False Negative Rate: $\mathrm{FNR} = 1 - \mathrm{TPR}$

False Positive Rate:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$
$$= \mathbb{P}[\hat{Y}(X) = 1 \mid H_0 \text{ is true}]$$

True Negative Rate: $\mathrm{TNR} = 1 - \mathrm{FPR}$

Precision: $\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} = \mathbb{P}[H_1 \text{ is true} \mid \hat{Y}(X) = 1]$

Types of errors
Confusion matrix

Ground truth

Predicted value	H_1	H_0	**Total**
$\hat{Y}(X) = 1$	True Positive	False Positive	P′
$\hat{Y}(X) = 0$	False Negative	True Negative	N′
Total	P	N	

Special mention

F1-score explicitly accounts for **class imbalances**

$$F_1 = \frac{2\text{TPR}}{1 + \text{TPR} + \frac{p_0}{p_1}\text{FPR}}$$

WARNING: metric must account for $N(\text{background}) >> N(\text{signal})$

Precision: $\text{PPV} = \frac{\text{TP}}{\text{TP}+\text{FP}} = \mathbb{P}[H_1 \text{ is true} \mid \hat{Y}(X) = 1]$

Types of errors
Confusion matrix

Ground truth

Predicted value	H_1	H_0	**Total**
$\hat{Y}(X) = 1$	True Positive	False Positive	P′
$\hat{Y}(X) = 0$	False Negative	True Negative	N′
Total	P	N	

Special mention

F1-score explicitly accounts for **class imbalances**

$$F_1 = \frac{2\text{TPR}}{1 + \text{TPR} + \frac{p_0}{p_1}\text{FPR}}$$

WARNING: metric must account for $N(\text{background}) >> N(\text{signal})$

Precision: $\text{PPV} = \frac{\text{TP}}{\text{TP}+\text{FP}} = \mathbb{P}[H_1 \text{ is true} \mid \hat{Y}(X) = 1]$

Honestly: Wikipedia article on *Confusion matrix* summarises the main error metrics

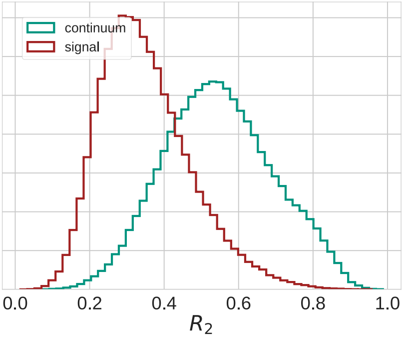
Neyman-Pearson lemma
Minimizing FP and FN errors

Neyman-Pearson lemma tells us when we've picked the best possible rejection/critical region **for a given** fixed error rate

Neyman-Pearson lemma
Minimizing FP and FN errors

Neyman-Pearson lemma tells us when we've picked the best possible rejection/critical region **for a given** fixed error rate

Example: Differentiating **signal** (positive) and **continuum** (negative)



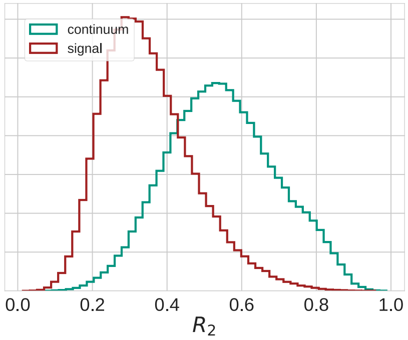
Neyman-Pearson lemma

Minimizing FP and FN errors

Neyman-Pearson lemma tells us when we've picked the best possible rejection/critical region **for a given** fixed error rate

Example: Differentiating **signal** (positive) and **continuum** (negative)

- Fix the FNR: $\mathbb{P}[\hat{Y}(X) = \text{cont} \mid H_{\text{sig}} \text{ is true}] = \alpha$



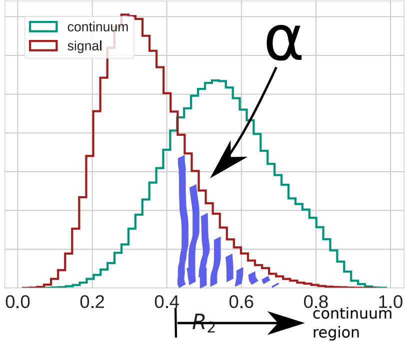
Neyman-Pearson lemma

Minimizing FP and FN errors

Neyman-Pearson lemma tells us when we've picked the best possible rejection/critical region **for a given** fixed error rate

Example: Differentiating **signal** (positive) and **continuum** (negative)

- Fix the FNR: $\mathbb{P}[\hat{Y}(X) = \text{cont} \mid H_{\text{sig}} \text{ is true}] = \alpha$
- Can we find a $k > 0$ such that:
  - $\frac{p(x|H_{\text{sig}} \text{ is true})}{p(x|H_{\text{cont}} \text{ is true})} \leq k$ for every $X \in (\hat{Y}(X) = \text{cont})$
  - $\frac{p(x|H_{\text{sig}} \text{ is true})}{p(x|H_{\text{cont}} \text{ is true})} \geq k$ for every $X \in (\hat{Y}(X) = \text{sig})$



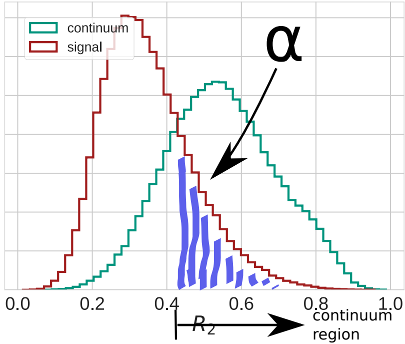
Neyman-Pearson lemma

Minimizing FP and FN errors

Neyman-Pearson lemma tells us when we've picked the best possible rejection/critical region **for a given** fixed error rate

Example: Differentiating **signal** (positive) and **continuum** (negative)

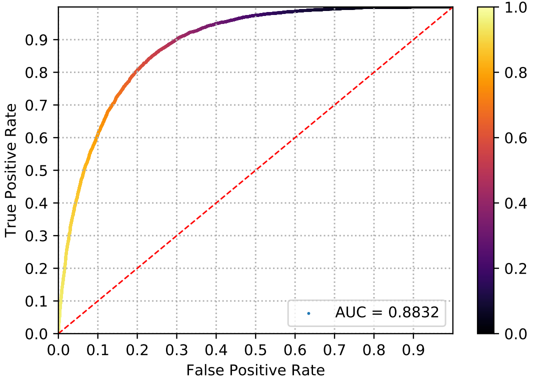
- Fix the FNR: $\mathbb{P}[\hat{Y}(X) = \text{cont} \mid H_{\text{sig}} \text{ is true}] = \alpha$
- Can we find a $k > 0$ such that:
  - $\dfrac{p(x|H_{\text{sig}} \text{ is true})}{p(x|H_{\text{cont}} \text{ is true})} \leq k$ for every $X \in (\hat{Y}(X) = \text{cont})$
  - $\dfrac{p(x|H_{\text{sig}} \text{ is true})}{p(x|H_{\text{cont}} \text{ is true})} \geq k$ for every $X \in (\hat{Y}(X) = \text{sig})$
- If so then we've found the **best** continuum region of size α



Receiver Operating Characteristic (ROC) curve

A simple way to show performance

- A way to **visualise** and **compare** model performances
- At different thresholds measure
  TPR/Recall/Sensitivity
  vs
  FPR / $1-$ **Specificity**
- Summarise performance with **area under the curve** (AUC):
  - $1.0 =$ perfect classifier
  - $0.5 =$ might as well flip a coin
  - $0.0 =$ how did you even get here?

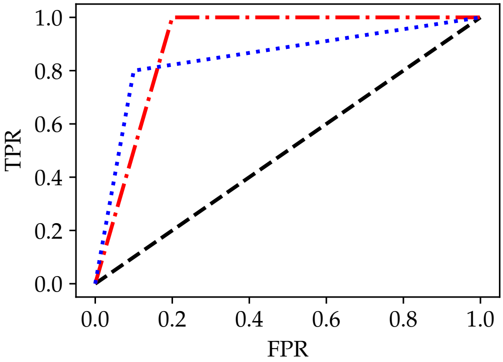


Example: binary classifier which outputs a signal probability between $[0-1]$

Receiver Operating Characteristic (ROC) curve

A simple way to show performance

- A way to **visualise** and **compare** model performances
- At different thresholds measure
  TPR/Recall/Sensitivity
  vs
  FPR / $1 -$ **Specificity**
- Summarise performance with **area under the curve** (AUC):
  - $1.0 =$ perfect classifier
  - $0.5 =$ might as well flip a coin
  - $0.0 =$ how did you even get here?
- AUC does not tell the whole story: **shape** of ROC curve is more important



Example: Two ROC curves with same AUC

Supervised learning

Independent and identically distributed

This is an **underlying** assumption in most ML you will do

If we have observed n labelled samples $(x_1, y_1), \ldots, (x_n, y_n)$

1. Assume each sample (x_i, y_i) is drawn from the **same underlying distribution** (X, Y)
2. Assume each sample is drawn **independently**

Independent and identically distributed

This is an **underlying** assumption in most ML you will do

If we have observed n labelled samples $(x_1, y_1), \ldots, (x_n, y_n)$

1. Assume each sample (x_i, y_i) is drawn from the **same underlying distribution** (X, Y)
2. Assume each sample is drawn **independently**

In physics terms:

1. Assume physics and experimental conditions stay the same
2. Assume each event has no influence over others

Independent and identically distributed

This is an **underlying** assumption in most ML you will do

If we have observed n labelled samples $(x_1, y_1), \ldots, (x_n, y_n)$

1. Assume each sample (x_i, y_i) is drawn from the **same underlying distribution** (X, Y)
2. Assume each sample is drawn **independently**

In physics terms:

1. Assume physics and experimental conditions stay the same
2. Assume each event has no influence over others

Like always in physics: pretend this is true and let uncertainties take care of the rest

Empirical risk minimisation
Optimisation in the real world

Recall we defined the **risk** of an algorithm f as $R[f] = \mathbb{E}\left[\ell(\hat{Y}(X), Y)\right]$

Empirical risk minimisation
Optimisation in the real world

Recall we defined the **risk** of an algorithm f as $R[f] = \mathbb{E}\left[\ell(\hat{Y}(X), Y)\right]$

In practice we only have access to some subset of the data $S = ((x_1, y_1), ..., (x_n, y_n))$, so our **empirical risk** is

$$R_S[f] = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$$

Empirical risk minimisation

Optimisation in the real world

Recall we defined the **risk** of an algorithm f as $R[f] = \mathbb{E}\left[\ell(\hat{Y}(X), Y)\right]$

In practice we only have access to some subset of the data $S = ((x_1, y_1), ..., (x_n, y_n))$, so our **empirical risk** is

$$R_S[f] = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$$

The learning task then comes down to finding the algorithm (e.g. a neural network) which **minimises the empirical risk:**

$$\min_{f \in \mathcal{F}} R_S[f]$$

But: this is just an approximation of the true risk of f based off the samples we have available.

Empirical risk minimisation

Optimisation in the real world

Recall we defined the **risk** of an algorithm f as $R[f] = \mathbb{E}\left[\ell(\hat{Y}(X), Y)\right]$

In practice we only have access to some subset of the data $S = ((x_1, y_1), ..., (x_n, y_n))$, so our **empirical risk** is

$$R_S[f] = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i)$$

The learning task then comes down to finding the algorithm (e.g. a neural network) which **minimises the empirical risk:**

$$\min_{f \in \mathcal{F}} R_S[f]$$

But: this is just an approximation of the true risk of f based off the samples we have available.

The difference between the **risk** and **empirical risk** is called the **generalisation gap.**

Generalisation gap

From [1]:

> *The* ***generalisation gap*** *($R[f] - R_S[f]$) ...tells us how well the performance of our classifier transfers from seen examples (the training examples) to unseen examples (a fresh example from the population) drawn from the same distribution.*

A large generalisation gap is what we call an **overfitted** model.

Key considerations

When designing an ML solution consider:

Representation What is the class of algorithms f to choose?

Optimisation How will you solve the optimisation problem?

Generalisation Will the algorithm transfer to unseen samples?

Key considerations

When designing an ML solution consider:

$\boxed{\text{Representation What is the class of algorithms } f \text{ to choose?}}$

Optimisation How will you solve the optimisation problem?

$\boxed{\text{Generalisation Will the algorithm transfer to unseen samples?}}$

Algorithms in order of ~~coolness~~ chronology

Fisher's Linear Discriminant

Dimensionality reduction to minimise class overlap

- An early form of **linear discriminant analysis**
- Old and overly simplistic, but demonstrates clearly a method of creating a new **representation** that's specific to the task
- So old it was published in the journal *Annals of Eugenics*
- Aims to project data onto a line such that classes are well separated

THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS

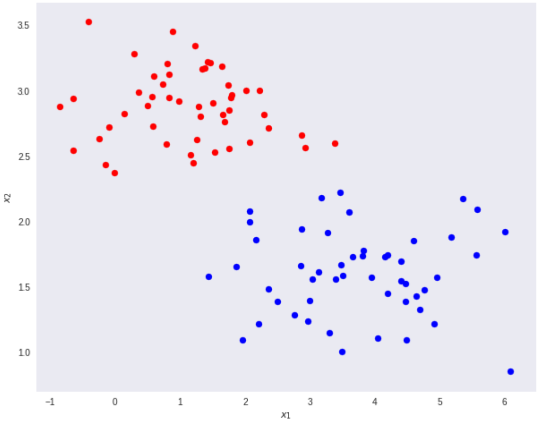
BY R. A. FISHER, Sc.D., F.R.S.

I. DISCRIMINANT FUNCTIONS

WHEN two or more populations have been measured in several characters, $x_1, \ldots, x_s$, special interest attaches to certain linear functions of the measurements by which the populations are best discriminated. At the author's suggestion use has already been made of this fact in craniometry (*a*) by Mr E. S. Martin, who has applied the principle to the sex differences in measurements of the mandible, and (*b*) by Miss Mildred Barnard, who showed how to obtain from a series of dated series the particular compound of cranial measurements showing most distinctly a progressive or secular trend. In the present paper the application of the same principle will be illustrated on a taxonomic problem; some questions connected with the precision of the processes employed will also be discussed.

Fisher's Linear Discriminant
Dimensionality reduction to minimise class overlap

Suppose we have two classes in $\mathbb{R}^2$ space[1]

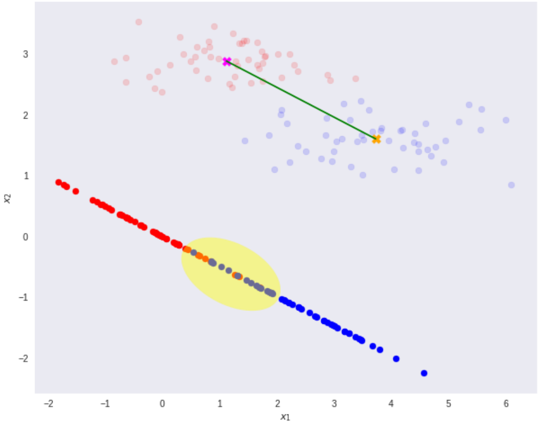


[0] Images from article An illustrative introduction to Fisher's Linear Discriminant

Fisher's Linear Discriminant
Dimensionality reduction to minimise class overlap

Suppose we have two classes in $\mathbb{R}^2$ space[1]

Naive way: Project class means into 1D



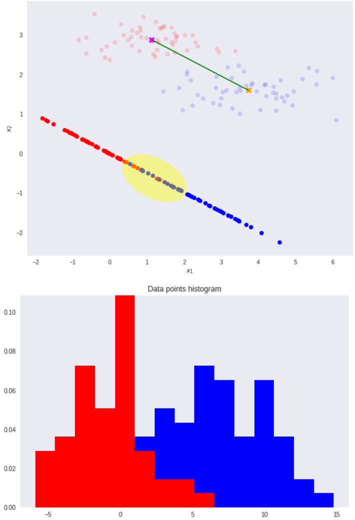
[0]Images from article An illustrative introduction to Fisher's Linear Discriminant

Fisher's Linear Discriminant

Dimensionality reduction to minimise class overlap

Suppose we have two classes in $\mathbb{R}^2$ space[1]

Naive way: Project class means into 1D



[0]Images from article An illustrative introduction to Fisher's Linear Discriminant

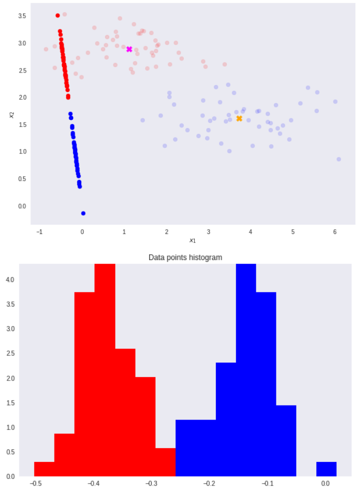
Fisher's Linear Discriminant

Dimensionality reduction to minimise class overlap

Suppose we have two classes in $\mathbb{R}^2$ space[1]

Naive way: Project class means into 1D

Ideally: Want to (consistently) get to here



[0] Images from article An illustrative introduction to Fisher's Linear Discriminant

Fisher's Linear Discriminant

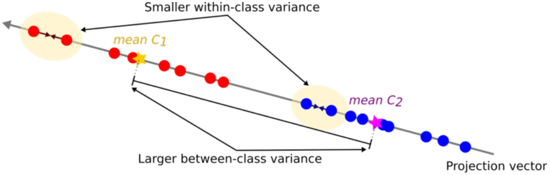
Dimensionality reduction to minimise class overlap

Suppose we have two classes in $\mathbb{R}^2$ space[1]

Naive way: Project class means into 1D

Ideally: Want to (consistently) get to here

Approach: **Maximise** inter-class variance while **minimising** intra-class variance



[0]Images from article An illustrative introduction to Fisher's Linear Discriminant

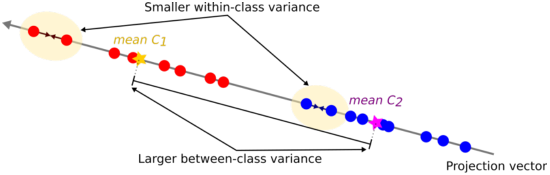
Fisher's Linear Discriminant
Dimensionality reduction to minimise class overlap

Suppose we have two classes in $\mathbb{R}^2$ space[1]

Naive way: Project class means into 1D

Ideally: Want to (consistently) get to here

Approach: **Maximise** inter-class variance while **minimising** intra-class variance



Maximise:

$$\text{Fisher's discriminent ratio} = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{\sigma}_1^2 + \tilde{\sigma}_2^2}$$

$\tilde{\mu}_i, \tilde{\sigma}_i^2 =$ projected mean, variance

[0] Images from article An illustrative introduction to Fisher's Linear Discriminant

Fisher's Linear Discriminant
The maths (kind of)

$$\text{Fisher's discriminant ratio} = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{\sigma}_1^2 + \tilde{\sigma}_2^2}$$

Let $\boldsymbol{v}$ be the **unit vector** defining the projection line we want to find.
The sample projections are now $y_i = \boldsymbol{v}^T \boldsymbol{x}_i$

Fisher's Linear Discriminant

The maths (kind of)

$$\text{Fisher's discriminant ratio} = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{\sigma}_1^2 + \tilde{\sigma}_2^2}$$

Let $\boldsymbol{v}$ be the **unit vector** defining the projection line we want to find.
The sample projections are now $y_i = \boldsymbol{v}^T \boldsymbol{x}_i$

Projected variance: $\tilde{\sigma}_i^2 = \sum_{y_i \in C_i} (y_i - \tilde{\mu}_i)^2$

$$\begin{aligned}
\tilde{\sigma}_i^2 &= \sum_{y_i \in C_i} (\boldsymbol{v}^T \boldsymbol{x}_i - \boldsymbol{v}^T \mu_i)^2 \\
&= \text{maths...} \\
&= \sum_{y_i \in C_i} \boldsymbol{v}^T (\boldsymbol{x}_i - \mu_i)(\boldsymbol{x}_i - \mu_i)^T \boldsymbol{v} \\
&= \boldsymbol{v}^T \boldsymbol{S}_i \boldsymbol{v}
\end{aligned}$$

So: $\tilde{\sigma}_1^2 + \tilde{\sigma}_2^2 = \boldsymbol{v}^T \boldsymbol{S}_1 \boldsymbol{v} + \boldsymbol{v}^T \boldsymbol{S}_2 \boldsymbol{v} = \boldsymbol{v}^T S_W \boldsymbol{v}$

Fisher's Linear Discriminant

The maths (kind of)

$$\text{Fisher's discriminant ratio} = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{\sigma}_1^2 + \tilde{\sigma}_2^2}$$

Let $\boldsymbol{v}$ be the **unit vector** defining the projection line we want to find.
The sample projections are now $y_i = \boldsymbol{v}^T \boldsymbol{x}_i$

Projected variance: $\tilde{\sigma}_i^2 = \sum_{y_i \in C_i} (y_i - \tilde{\mu}_i)^2$

$$\begin{aligned}
\tilde{\sigma}_i^2 &= \sum_{y_i \in C_i} (\boldsymbol{v}^T \boldsymbol{x}_i - \boldsymbol{v}^T \mu_i)^2 \\
&= \text{maths...} \\
&= \sum_{y_i \in C_i} \boldsymbol{v}^T (\boldsymbol{x}_i - \mu_i)(\boldsymbol{x}_i - \mu_i)^T \boldsymbol{v} \\
&= \boldsymbol{v}^T \boldsymbol{S}_i \boldsymbol{v}
\end{aligned}$$

So: $\tilde{\sigma}_1^2 + \tilde{\sigma}_2^2 = \boldsymbol{v}^T \boldsymbol{S}_1 \boldsymbol{v} + \boldsymbol{v}^T \boldsymbol{S}_2 \boldsymbol{v} = \boldsymbol{v}^T S_W \boldsymbol{v}$

Projected means: $\tilde{\mu}_i = \boldsymbol{v}^T \frac{1}{n_i} \sum_{x_i \in C_i}^{n_i} \boldsymbol{x_i} = \boldsymbol{v^T} \mu_i$

$$\begin{aligned}
(\tilde{\mu}_1 - \tilde{\mu}_2)^2 &= (\boldsymbol{v}^T \mu_1 - \boldsymbol{v}^T \mu_2)^2 \\
&= \boldsymbol{v}^T (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \boldsymbol{v} \\
&= \boldsymbol{v}^T \boldsymbol{S}_B \boldsymbol{v}
\end{aligned}$$

Fisher's Linear Discriminant

The maths (kind of)

$$\text{Fisher's discriminant ratio} = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{\sigma}_1^2 + \tilde{\sigma}_2^2} = \frac{\boldsymbol{v}^T \boldsymbol{S}_B \boldsymbol{v}}{\boldsymbol{v}^T \boldsymbol{S}_W \boldsymbol{v}}$$

Fisher's Linear Discriminant

The maths (kind of)

$$\text{Fisher's discriminant ratio} = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{\sigma}_1^2 + \tilde{\sigma}_2^2} = \frac{\boldsymbol{v}^T \boldsymbol{S}_B \boldsymbol{v}}{\boldsymbol{v}^T \boldsymbol{S}_W \boldsymbol{v}}$$

Taking derivative w.r.t $\boldsymbol{v}$, setting to zero, and more maths gives an eigenvalue problem:

$$\boldsymbol{S}_W^{-1} \boldsymbol{S}_B \boldsymbol{v} = \lambda \boldsymbol{v}, \quad \lambda = \frac{\boldsymbol{v}^T \boldsymbol{S}_B \boldsymbol{v}}{\boldsymbol{v}^T \boldsymbol{S}_W \boldsymbol{v}}$$

Fisher's Linear Discriminant

The maths (kind of)

$$\text{Fisher's discriminant ratio} = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{\sigma}_1^2 + \tilde{\sigma}_2^2} = \frac{\boldsymbol{v}^T \boldsymbol{S}_B \boldsymbol{v}}{\boldsymbol{v}^T \boldsymbol{S}_W \boldsymbol{v}}$$

Taking derivative w.r.t $\boldsymbol{v}$, setting to zero, and more maths gives an eigenvalue problem:

$$\boldsymbol{S}_W^{-1} \boldsymbol{S}_B \boldsymbol{v} = \lambda \boldsymbol{v}, \quad \lambda = \frac{\boldsymbol{v}^T \boldsymbol{S}_B \boldsymbol{v}}{\boldsymbol{v}^T \boldsymbol{S}_W \boldsymbol{v}}$$

But $\boldsymbol{S}_B \boldsymbol{v}$ points in the direction of $(\mu_1 - \mu_2)$, so:

$$\begin{aligned} \boldsymbol{S}_B \boldsymbol{v} &= \alpha(\mu_1 - \mu_2) \\ \implies \boldsymbol{S}_W^{-1} \boldsymbol{S}_B \boldsymbol{v} &= \alpha \boldsymbol{S}_W^{-1}(\mu_1 - \mu_2) = \lambda \boldsymbol{v} \\ \implies \boldsymbol{v} &= \boldsymbol{S}_W^{-1}(\mu_1 - \mu_2) \end{aligned}$$

Recalling: $\boldsymbol{S}_W = \boldsymbol{S}_1 + \boldsymbol{S}_2 =$ class variances before projection

Fisher's Linear Discriminant

The maths (kind of)

$$\text{Fisher's discriminant ratio} = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{\sigma}_1^2 + \tilde{\sigma}_2^2} = \frac{\boldsymbol{v}^T \boldsymbol{S}_B \boldsymbol{v}}{\boldsymbol{v}^T \boldsymbol{S}_W \boldsymbol{v}}$$

Taking derivative w.r.t $\boldsymbol{v}$, setting to zero, and more maths gives an eigenvalue problem:

$$\boldsymbol{S}_W^{-1} \boldsymbol{S}_B \boldsymbol{v} = \lambda \boldsymbol{v}, \quad \lambda = \frac{\boldsymbol{v}^T \boldsymbol{S}_B \boldsymbol{v}}{\boldsymbol{v}^T \boldsymbol{S}_W \boldsymbol{v}}$$

But $\boldsymbol{S}_B \boldsymbol{v}$ points in the direction of $(\mu_1 - \mu_2)$, so:

$$\begin{aligned} \boldsymbol{S}_B \boldsymbol{v} &= \alpha(\mu_1 - \mu_2) \\ \implies \boldsymbol{S}_W^{-1} \boldsymbol{S}_B \boldsymbol{v} &= \alpha \boldsymbol{S}_W^{-1}(\mu_1 - \mu_2) = \lambda \boldsymbol{v} \\ \implies \boldsymbol{v} &= \boldsymbol{S}_W^{-1}(\mu_1 - \mu_2) \end{aligned}$$

Recalling: $\boldsymbol{S}_W = \boldsymbol{S}_1 + \boldsymbol{S}_2 =$ class variances before projection

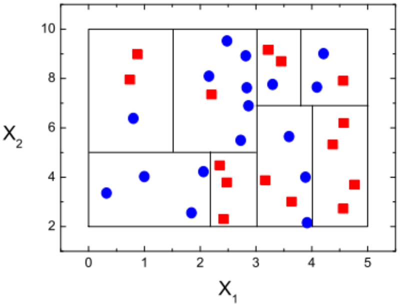
The point: You can calculate the ideal projection from the original class means and variances alone

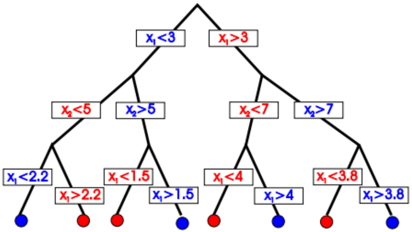
Decision trees

Decision trees are the current workhorse in Belle II

$\Longrightarrow$ FEI, continuum suppression

- Classify examples by sorting them from root down to a leaf
- Each node applies a decision to one variable
- Discrete targets: classification trees
  Continuous targets: regression trees
- Branches can be binary or more

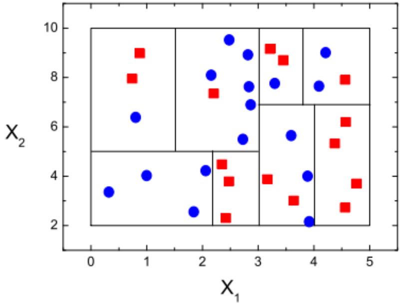


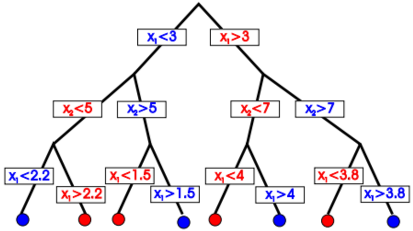


Decision trees

Advantages

- Captures **interactions** between features
- Simple **interpretation** of sample groupings (**explainable** results)
- Trivial to find **feature importance**
- No need to **transform** input features





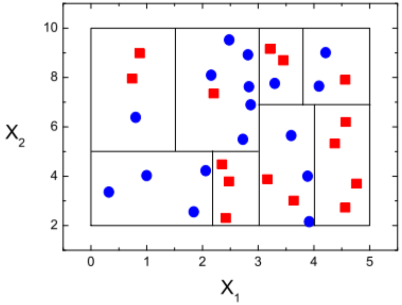
Decision trees

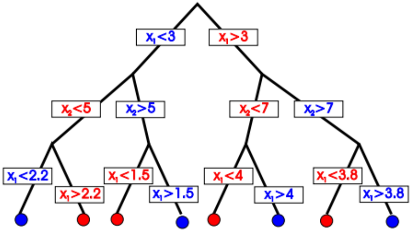
Advantages

- Captures **interactions** between features
- Simple **interpretation** of sample groupings (**explainable** results)
- Trivial to find **feature importance**
- No need to **transform** input features

Disadvantages

- Fails to effectively handle **linear relationships**
- **Lack of smoothness:** small changes to inputs can have big impact on predicted outcomes
- **Unstable** to train: small changes to dataset = big changes to tree
- No. of leaves can **grow exponentially** with depth – kills interpretability





Decision trees
How to construct them?

Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node
Several common costs exist:

- Entropy
- Information gain
- Gini index
- Gain ratio
- Reduction in Variance
- Chi-square

Decision trees

How to construct them?

Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node

Several common costs exist:

- Entropy
- Information gain
- Gini index
- Gain ratio
- Reduction in Variance
- Chi-square

Decision trees

How to construct them?

Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node

$$\text{Gini} = 1 - \sum_{i=1}^{C} p_i^2 = \text{chance of incorrectness}$$

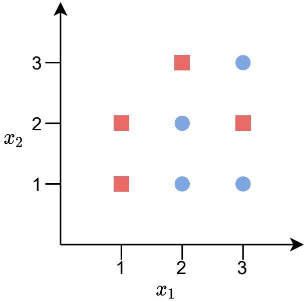
Decision trees

How to construct them?

Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node

$$\text{Gini} = 1 - \sum_{i=1}^{C} p_i^2 = \text{chance of incorrectness}$$

To decide ideal feature/cut for node, evaluate Gini for each: $x_1 \in [1.5, 2.5]$ and $x_2 \in [1.5, 2.5]$



Decision trees

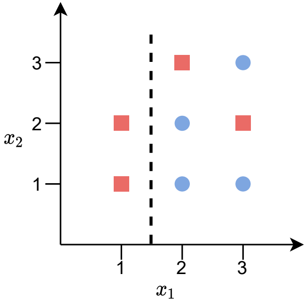
How to construct them?

Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node

$$\text{Gini} = 1 - \sum_{i=1}^{C} p_i^2 = \text{chance of incorrectness}$$

To decide ideal feature/cut for node, evaluate Gini for each: $x_1 \in [1.5, 2.5]$ and $x_2 \in [1.5, 2.5]$

$$\mathbb{P}(x > 1.5) = \frac{6}{8}, \quad \mathbb{P}(x < 1.5) = \frac{2}{8}$$



Decision trees

How to construct them?

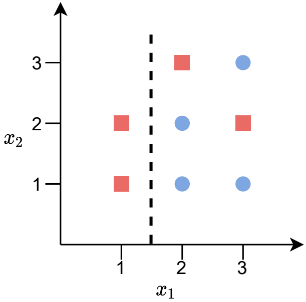
Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node

$$\text{Gini} = 1 - \sum_{i=1}^{C} p_i^2 = \text{chance of incorrectness}$$

To decide ideal feature/cut for node, evaluate Gini for each: $x_1 \in [1.5, 2.5]$ and $x_2 \in [1.5, 2.5]$

$$\mathbb{P}(x > 1.5) = \frac{6}{8}, \quad \mathbb{P}(x < 1.5) = \frac{2}{8}$$

$$\mathbb{P}(r \mid x_1 > 1.5) = \frac{2}{6}, \quad \mathbb{P}(b \mid x_1 > 1.5) = \frac{4}{6}$$



Decision trees
How to construct them?

Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node

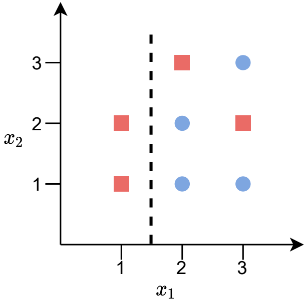
$$\text{Gini} = 1 - \sum_{i=1}^{C} p_i^2 = \text{chance of incorrectness}$$

To decide ideal feature/cut for node, evaluate Gini for each: $x_1 \in [1.5, 2.5]$ and $x_2 \in [1.5, 2.5]$

$$\mathbb{P}(x > 1.5) = \frac{6}{8}, \quad \mathbb{P}(x < 1.5) = \frac{2}{8}$$

$$\mathbb{P}(r \mid x_1 > 1.5) = \frac{2}{6}, \quad \mathbb{P}(b \mid x_1 > 1.5) = \frac{4}{6}$$

$$\text{Gini}(x_1 > 1.5) = 1 - \left(\left(\tfrac{2}{6}\right)^2 + \left(\tfrac{4}{6}\right)^2\right) = \frac{4}{9}$$



Decision trees

How to construct them?

Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node

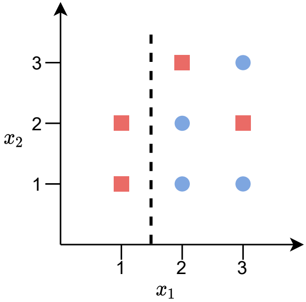
$$\text{Gini} = 1 - \sum_{i=1}^{C} p_i^2 = \text{chance of incorrectness}$$

To decide ideal feature/cut for node, evaluate Gini for each: $x_1 \in [1.5, 2.5]$ and $x_2 \in [1.5, 2.5]$

$$\mathbb{P}(x > 1.5) = \tfrac{6}{8}, \quad \mathbb{P}(x < 1.5) = \tfrac{2}{8}$$

$$\mathbb{P}(r \mid x_1 > 1.5) = \tfrac{2}{6}, \quad \mathbb{P}(b \mid x_1 > 1.5) = \tfrac{4}{6}$$

$$\text{Gini}(x_1 > 1.5) = 1 - \left((\tfrac{2}{6})^2 + (\tfrac{4}{6})^2\right) = \tfrac{4}{9}$$
$$\text{Gini}(x_1 < 1.5) = 1 - \left((\tfrac{2}{2})^2 + (\tfrac{0}{2})^2\right) = 0$$



Decision trees

How to construct them?

Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node

$$\text{Gini} = 1 - \sum_{i=1}^{C} p_i^2 = \text{chance of incorrectness}$$

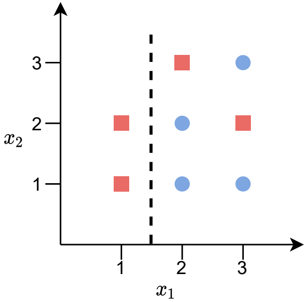
To decide ideal feature/cut for node, evaluate Gini for each: $x_1 \in [1.5, 2.5]$ and $x_2 \in [1.5, 2.5]$

$$\mathbb{P}(x > 1.5) = \tfrac{6}{8}, \quad \mathbb{P}(x < 1.5) = \tfrac{2}{8}$$

$$\mathbb{P}(r \mid x_1 > 1.5) = \tfrac{2}{6}, \quad \mathbb{P}(b \mid x_1 > 1.5) = \tfrac{4}{6}$$

$$\text{Gini}(x_1 > 1.5) = 1 - \left((\tfrac{2}{6})^2 + (\tfrac{4}{6})^2\right) = \tfrac{4}{9}$$
$$\text{Gini}(x_1 < 1.5) = 1 - \left((\tfrac{2}{2})^2 + (\tfrac{0}{2})^2\right) = 0$$

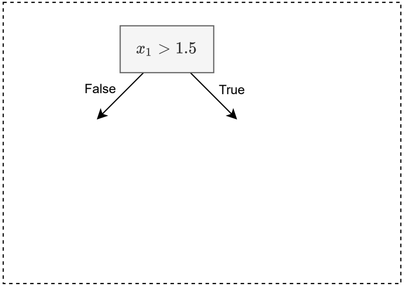
Weighted sum of Gini indices:
$$\text{Gini}(x_1 : 1.5) = (\tfrac{6}{8})(\tfrac{4}{9}) + (\tfrac{2}{8})(0) = \tfrac{1}{3}$$

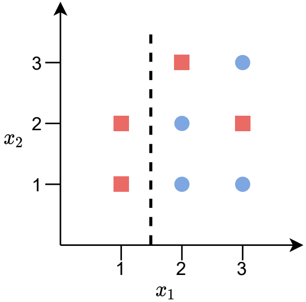


Decision trees
How to construct them?

Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node

- Find feature cut with lowest Gini index (e.g. $x_1 > 1.5$)
- This becomes your tree root



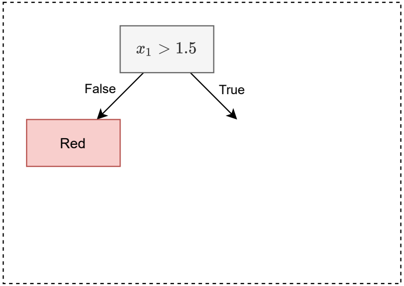


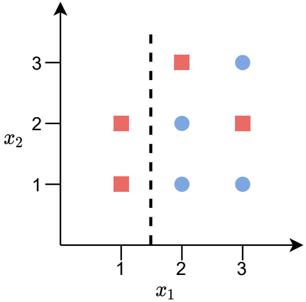
Decision trees

How to construct them?

Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node

- Find feature cut with lowest Gini index (e.g. $x_1 > 1.5$)
- This becomes your tree root
- Repeat for both branches



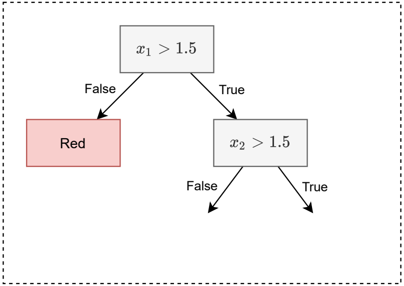


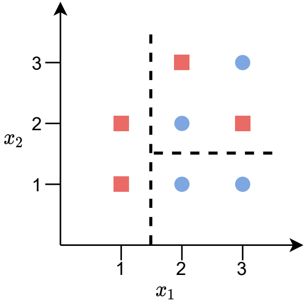
Decision trees

How to construct them?

Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node

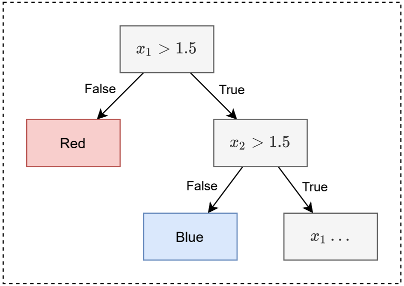
- Find feature cut with lowest Gini index (e.g. $x_1 > 1.5$)
- This becomes your tree root
- Repeat for both branches

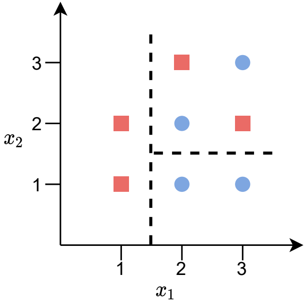




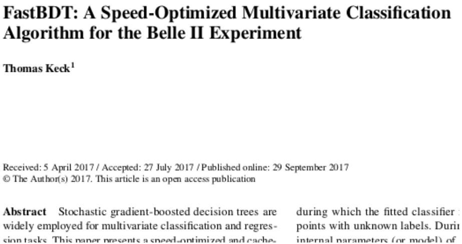
Basic principle: Work from **root** down, using some **cost** to decide which **feature** to use at each node

- Find feature cut with lowest Gini index (e.g. $x_1 > 1.5$)
- This becomes your tree root
- Repeat for both branches
- Stop when you reach pre-defined conditions (e.g. max depth)





FastBDT
Fast and boosted


FastBDT: A Speed-Optimized Multivariate Classification Algorithm for the Belle II Experiment

Thomas Keck[1]

Received: 5 April 2017 / Accepted: 27 July 2017 / Published online: 29 September 2017

Abstract Stochastic gradient-boosted decision trees are widely employed for multivariate classification and regres-

during which the fitted classifier
points with unknown labels. Durin

- Current basf2 implementation is **FastBDT**
- B = boosted
  - Train many **weak learners** (small trees)
  - Each find a “rule of thumb”
  - Combine into a single strong learner
- Fast = speed optimised implementation written by Thomas Keck[4]

Figure: Keck-sama

Scikit learn

Doing the thinking for you

If you want to have a go at using these and more (classical) machine learning algorithms:

The supervised learning page from **scikit learn** (sklearn) has many easy to use implementations

1. Supervised learning

1.1. Linear Models

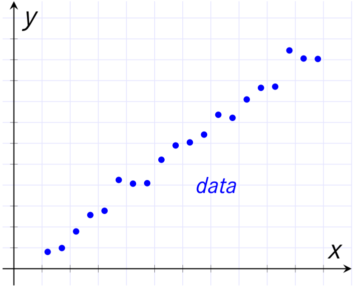
- 1.1.1. Ordinary Least Squares
- 1.1.2. Ridge regression and classification
- 1.1.3. Lasso
- 1.1.4. Multi-task Lasso
- 1.1.5. Elastic-Net
- 1.1.6. Multi-task Elastic-Net
- 1.1.7. Least Angle Regression
- 1.1.8. LARS Lasso
- 1.1.9. Orthogonal Matching Pursuit (OMP)
- 1.1.10. Bayesian Regression
- 1.1.11. Logistic regression
- 1.1.12. Generalized Linear Regression
- 1.1.13. Stochastic Gradient Descent - SGD
- 1.1.14. Perceptron
- 1.1.15. Passive Aggressive Algorithms
- 1.1.16. Robustness regression: outliers and modeling errors
- 1.1.17. Polynomial regression: extending linear models with basis functions

1.2. Linear and Quadratic Discriminant Analysis

- 1.2.1. Dimensionality reduction using Linear Discriminant Analysis
- 1.2.2. Mathematical formulation of the LDA and QDA classifiers
- 1.2.3. Mathematical formulation of LDA dimensionality reduction

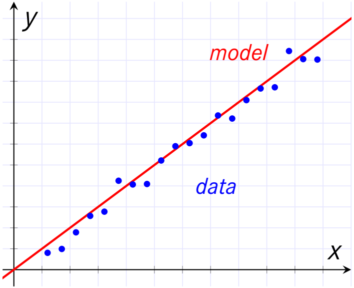
Neural Networks

Linear Regression



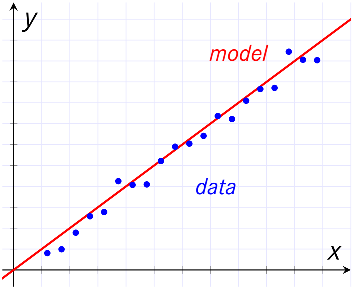
- **Data set:** $\{samples, labels\} = \{x, y\}$

Linear Regression



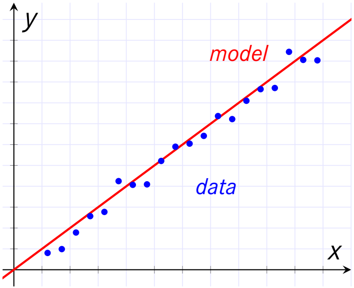
- **Data set:** $\{samples, labels\} = \{x, y\}$
- **Model:** definition $\hat{y} = wx + b$
  with w and b trainable parameters

Linear Regression



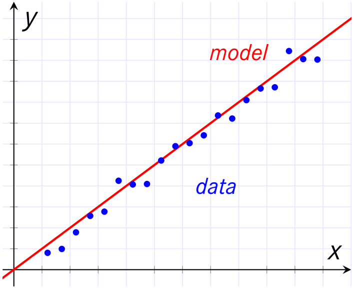
- **Data set:** $\{samples, labels\} = \{x, y\}$
- **Model:** definition $\hat{y} = wx + b$
  with w and b trainable parameters
- **Loss function:** or cost/error/objective
  $\ell(w, b) = MSE(w, b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$

Linear Regression



- **Data set:** $\{samples, labels\} = \{x, y\}$
- **Model:** definition $\hat{y} = wx + b$
  with w and b trainable parameters
- **Loss function:** or cost/error/objective
  $\ell(w, b) = MSE(w, b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
- **Train:** the model, e.g. optimization
  $\hat{w}, \hat{b} = \arg\min \ell(w, b)$

Linear Regression



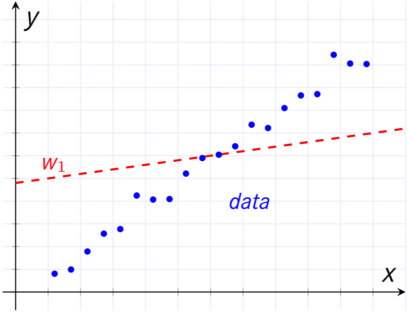
- **Data set:** $\{samples, labels\} = \{x, y\}$
- **Model:** definition $\hat{y} = wx + b$
  with w and b trainable parameters
- **Loss function:** or cost/error/objective
  $\ell(w, b) = MSE(w, b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$
- **Train:** the model, e.g. optimization
  $\hat{w}, \hat{b} = \arg\min \ell(w, b)$

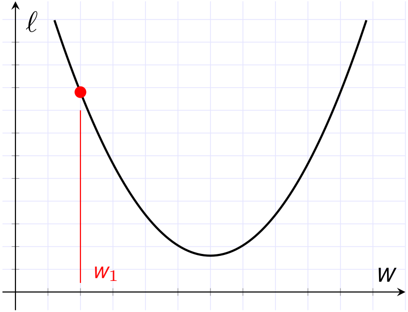
- **Basic recipe for most machine learning algorithms**

Optimization: Gradient Descent

- Iterative optimization technique, weight update in direction of negative gradient

$$w_{i+1} = w_i - \eta \nabla_{w_i} \ell(w_i)$$

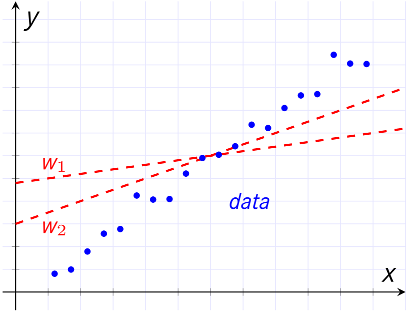


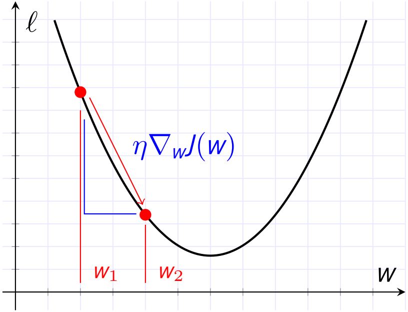


Optimization: Gradient Descent

- Iterative optimization technique, weight update in direction of negative gradient

$$w_{i+1} = w_i - \eta \nabla_{w_i} \ell(w_i)$$

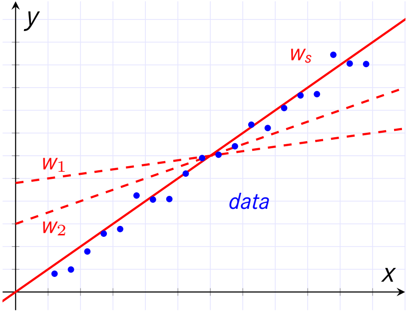


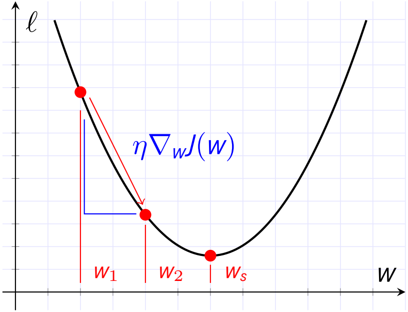


Optimization: Gradient Descent

- Iterative optimization technique, weight update in direction of negative gradient

$$w_{i+1} = w_i - \eta \nabla_{w_i} \ell(w_i)$$

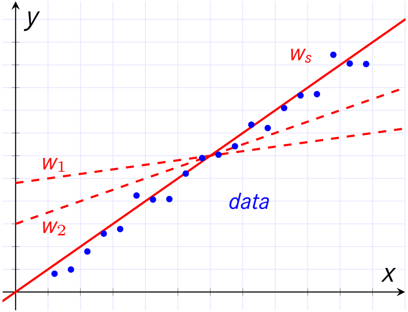


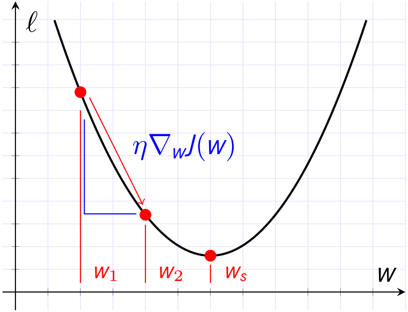


Optimization: Gradient Descent

- Iterative optimization technique, weight update in direction of negative gradient

$$w_{i+1} = w_i - \eta \nabla_{w_i} \ell(w_i)$$



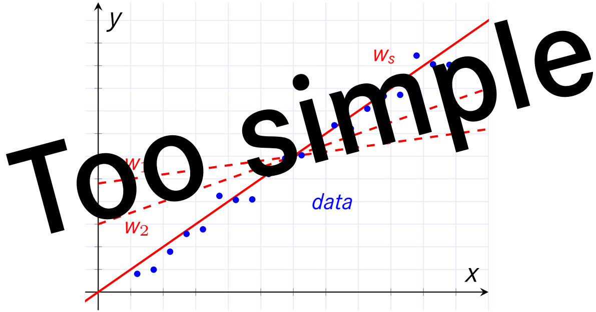


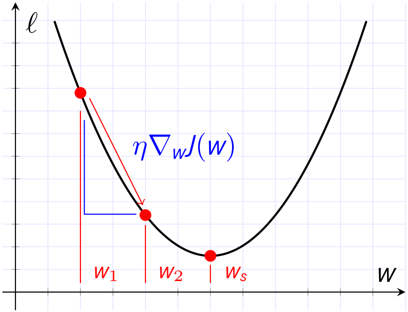
- η is learning rate, gradient update factor
- **Stochastic gradient descent (SGD)**, sample subset (**batch**) updates

Optimization: Gradient Descent

- Iterative optimization technique, weight update in direction of negative gradient

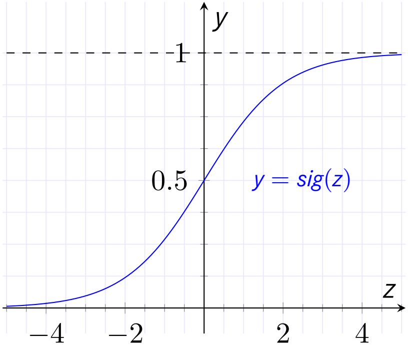
$$w_{i+1} = w_i - \eta \nabla_{w_i} \ell(w_i)$$





- η is learning rate, gradient update factor
- **Stochastic gradient descent (SGD)**, sample subset (**batch**) updates

Logistic Regression



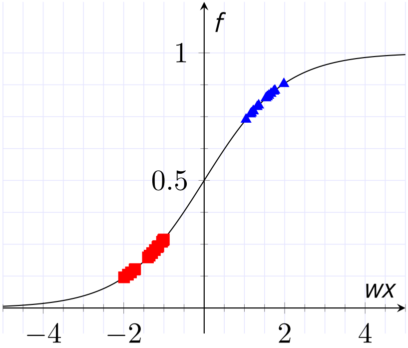
- Squash linear regression output into fixed interval, e.g. $y \in [0, 1]$
- Interpretation: **probability** of sample belonging to a binary class
- **sigmoid-/logistic** function: $sig(z) = \frac{1}{1+e^{-z}}$
- **Model:** $f = sig(wx) = \frac{1}{1+e^{-wx}}$
- **Prediction:** $\hat{y} = 1$ if $f \geq 0.5$
  $\hat{y} = 0$ if $f < 0.5$

Logistic Regression

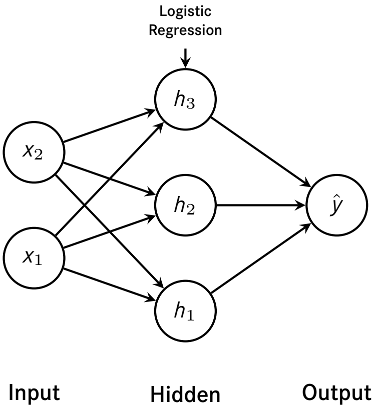
- **Data set** must be mapped
  - $\blacksquare \rightarrow 0$
  - $\blacktriangle \rightarrow 1$
- **Model:** $f = sig(wx) = \frac{1}{1+e^{-wx}}$
- **Loss function:**
  $\ell(w) = MSE(w) = \frac{1}{n}\sum_{i=1}^{n}(y - f)^2$

  $\nabla_w \ell(w) = (f - y) \times f^2 \times e^{-wx} \times x$

  (Hint: chain rule)
- **Train:** gradient descent optimization



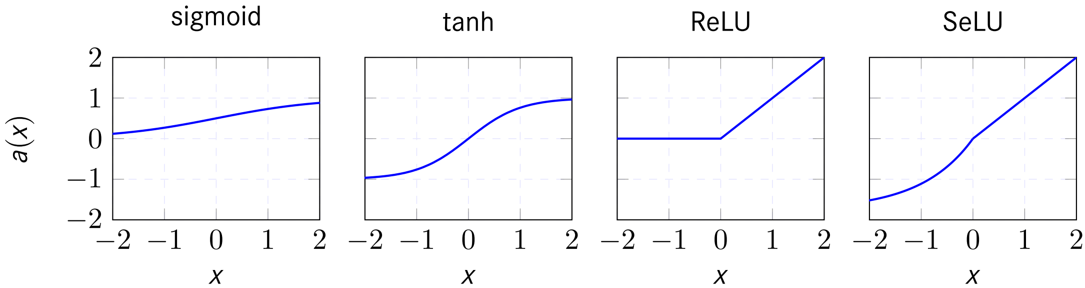
Fully-connected Neural Network



- Inspired by biological neural network
- A **neuron** is a logistic regression
- Neurons are arranged in **layers**
  $\hat{y} = \mathrm{sig}(\sum_i w_i h_i)$
- Layers are **fully-connected** with subsequent layer, also called **Dense**
- **Width:** neuron count
- **Depth:** layer count

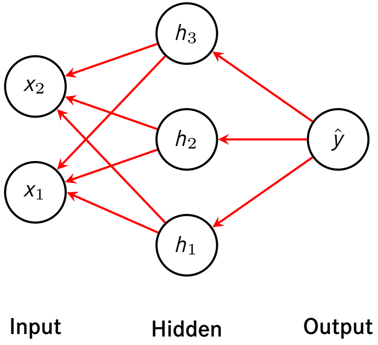
Activation Functions

- Activation functions $a(x)$ introduce **non-linearity**, e.g. sigmoid function
- Other non-linear choices, e.g. $tanh(x)$, $relu(x) = max(0, x)$, etc.
- Better computational properties, e.g. avoid **vanishing gradient**



Backpropagation

- Alternate forward and backward pass
- **Hidden layer** are nested functions
  - Requires **chain rule** for gradient
  - Every component must have a gradient **defined**
  - $h'(x) = f'(g(x)) * g'(x)$
  - **Neurons store forward result**
- Weight initialization in network small random numbers
- Iterations across dataset called **epochs**

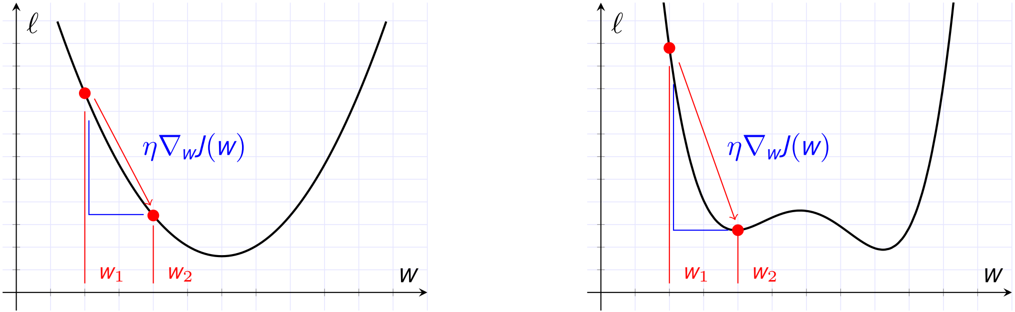


Autograd Frameworks: TensorFlow & Co

- **Numerical** and **autograd** libraries
- **Eager** and **flow graph** computation
- Multiple supported devices
  CPU, **GPU**, TPU, smartphone
- TensorFlow (Google), MXNet (Amazon), PyTorch (Facebook)
- **Keras**—neural network wrapper for TensorFlow and MXNet backends

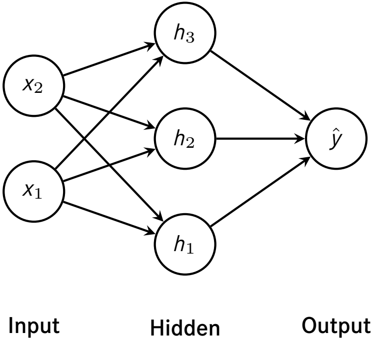
A comment on convexity

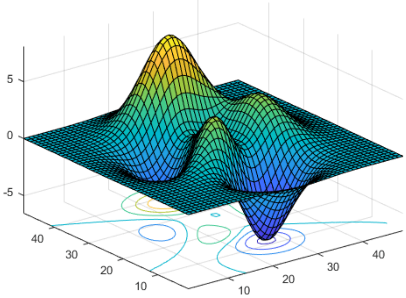
- Gradient descent guarantees a global minimum for a **(quasi)convex loss**
- The loss of a neural network is in general **not** convex



A comment on convexity

- Gradient descent guarantees a global minimum for a **(quasi)convex loss**
- The loss of a neural network is in general **not** convex
- Why? Permuting the weights of any two neurons will produce the **same** loss value
  And: many non-linear activations end up producing a complex **loss landscape**





Universal Approximation Theorem

A feed-forward neural network with a linear output and at least one hidden layer can approximate any reasonable function to arbitrary precision with a finite number of nodes.

Universal Approximation Theorem

A feed-forward neural network with a linear output and at least one hidden layer can approximate any reasonable function to arbitrary precision with a finite number of nodes.

- Good News
  - Networks can perform highly complex tasks
  - All necessary ingredients available

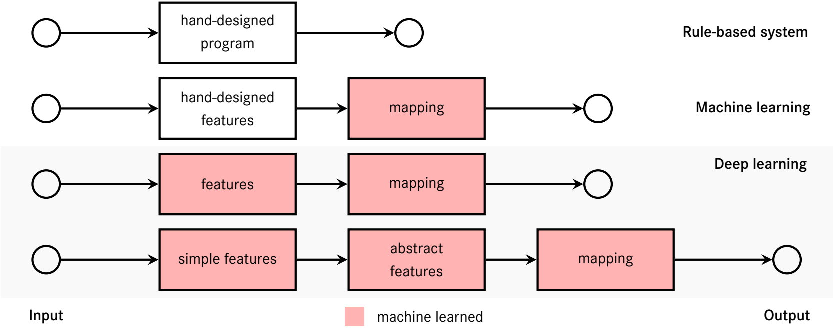
Universal Approximation Theorem

A feed-forward neural network with a linear output and at least one hidden layer can approximate any reasonable function to arbitrary precision with a finite number of nodes.

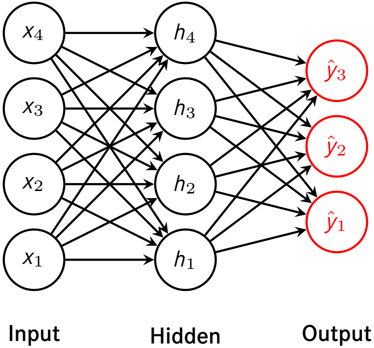
- Good News
  - Networks can perform highly complex tasks
  - All necessary ingredients available
- Bad News
  - Does not specify number of necessary nodes
  - No remarks on neuron connectivity

Deep Learning

- In practice: **stacking layers** works better
- **Deep learning:** more than one stage of non-linearities, e.g. layers



Multi-class Classification



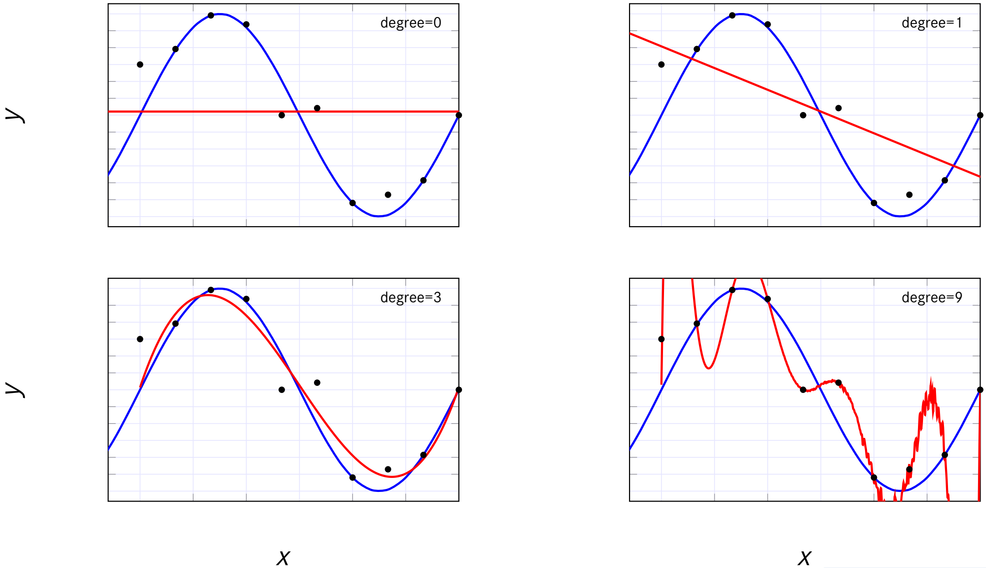
- Extension of binary classification concept
- **One-versus-all classification**
  - Build c binary classifiers
  - Pick class with highest confidence/probability
- In neural networks
  - Create multiple networks
  - **Add output neurons**

Multi-class Classification

Multi-class classification recipe:

- **One-hot class encoding:** encode classes as sparse vectors $y = (y_1, y_2, ..., y_c)$, only one is active, e.g. class $2 \rightarrow (0, 1, ..., 0)$
- **Softmax output activation:** $\hat{y} = softmax(z) = \frac{e^{z_j}}{\sum_j e^{z_j}}$ for $j = 1...c$
  achieve joint-probability of 1, normalize across model outputs z
- **Cross-entropy loss:** convex-function $J(w) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j}^{c} y_{i,j} \log \hat{y}_{i,j}$
  maximum likelihood principle

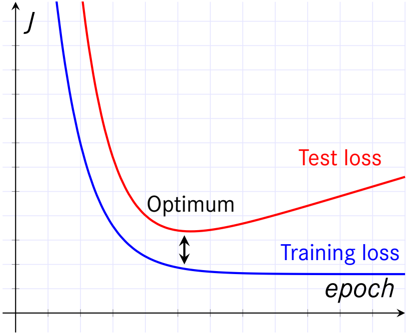
Over- and Underfitting



Over- and Underfitting

- How do we know a network is not over- or underfitting?
- **Idea:** simulate “unseen” data
- Split data artificially into disjoint subsets
  - **Training set** for training the model (usually $60\% - 80\%$)
  - **Validation set** for fine tunine the model (usually $10\% - 20\%$)
  - **Test set** to test validation (usually $20\% - 40\%$)

Over- and Underfitting



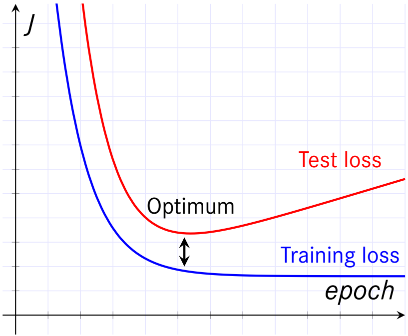
- Separate monitoring of training and validation loss during training
- Training loss will decrease indefinitely $J \to 0$, **memorization effect**
- **Validation loss minimum** is optimal
- **Stop** training when train/val losses diverge*

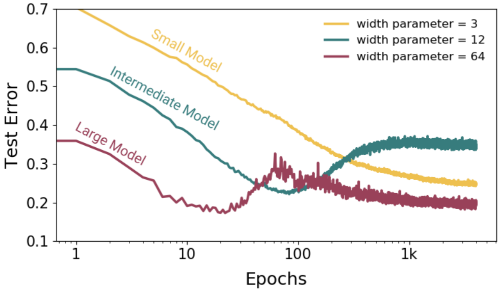
*Up until very recently this was believed to true...

Deep double descent[5]

A small aside

- Traditional belief was that once validation/test loss diverges it diverges forever.
- Recent work shows this is not always the case

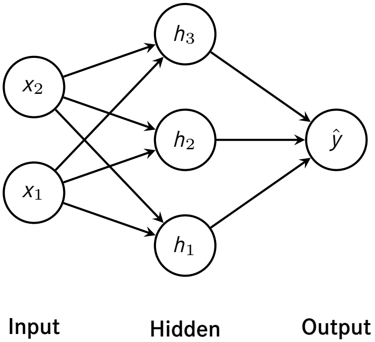




Types of neural network layers

Why do variants even exist?

- Look again at the fully-connected network
- What if x_i represent e.g. the absolute momentum of two detected particles?
- Which particle should be x_1? Which should be x_2?
- If no clear ordering exists: A fully-connected network needs to learn every possible permutation

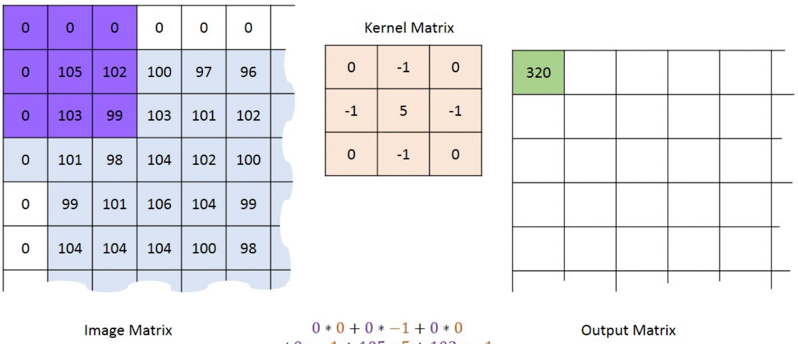


Convolutional neural networks

- Element-wise weighted sum of input and filter
- $(f * g)[n] = \sum_{m=-K}^{K} f[m]g[n-m]$
- **Filter size K:** window size of convolution kernel
- **Stride:** pixel distance for slide
- 2D input: volume of *width* $\times$ *height*$(\times$*channels*$)$
- Models effects on images, e.g. edge detection
- In CNN: model "eye", sparse weight sharing

Figure: Belle II software developer

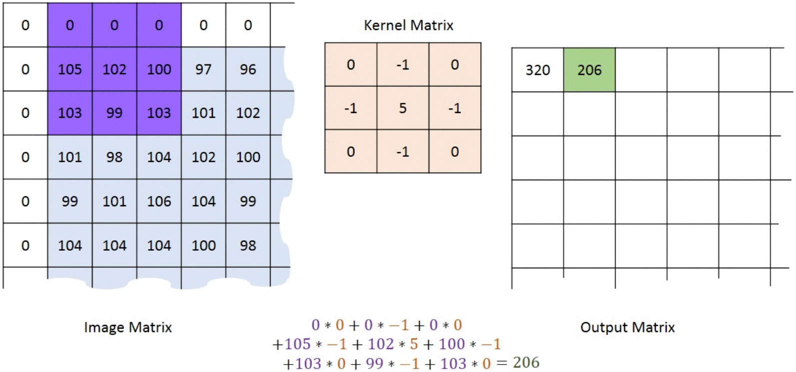
Convolutional neural networks



$$
\begin{aligned}
&0*0+0*-1+0*0\\
&+0*-1+105*5+102*-1\\
&+0*0+103*-1+99*0 \quad = 320
\end{aligned}
$$

Convolution with horizontal and vertical strides = 1

Convolutional neural networks



Convolution with horizontal and vertical strides = 1

Convolutional neural networks

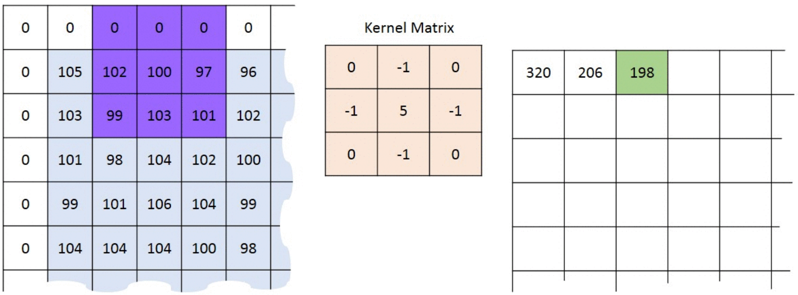


Image Matrix

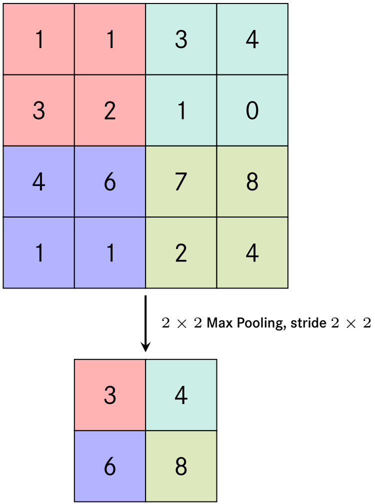
$$0*0+0*-1+0*0$$
$$+102*-1+100*5+97*-1$$
$$+99*0+103*-1+101*0=198$$

Output Matrix

Convolution with horizontal and vertical strides = 1

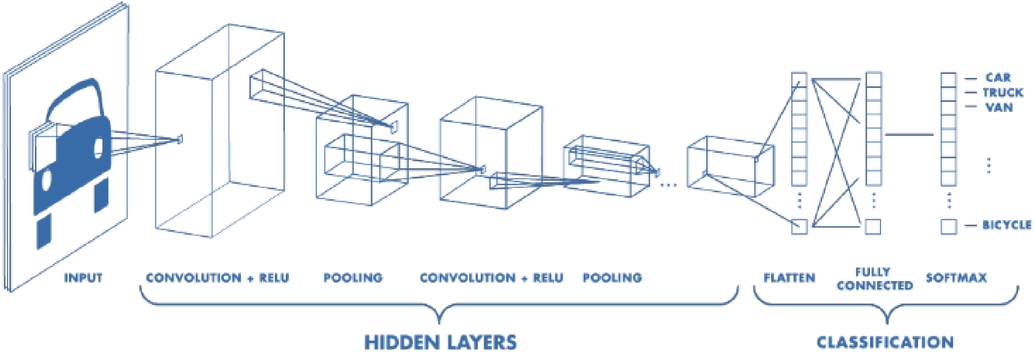
Pooling

- **Pooling** reduces input sizes, abstract downsampled copy
- **Pool size:** kernel height/width
- **Strides:** step width
- Typical pooling layers
  - Max Pooling
  - Average Pooling



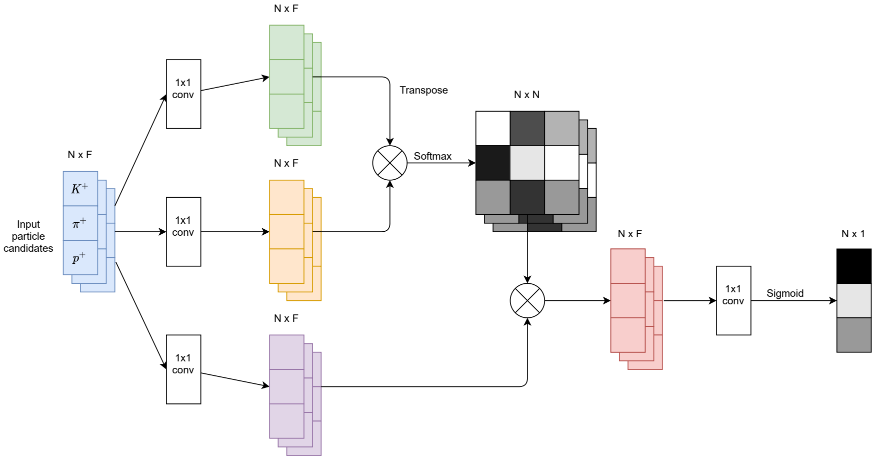
Convolutional Neural Network Pyramid

Convolutions allow **large-scale objects** to be anywhere in the image, but the **small-scale** structure is rigid $\rightarrow$ a step up from fully-connected networks



Attention

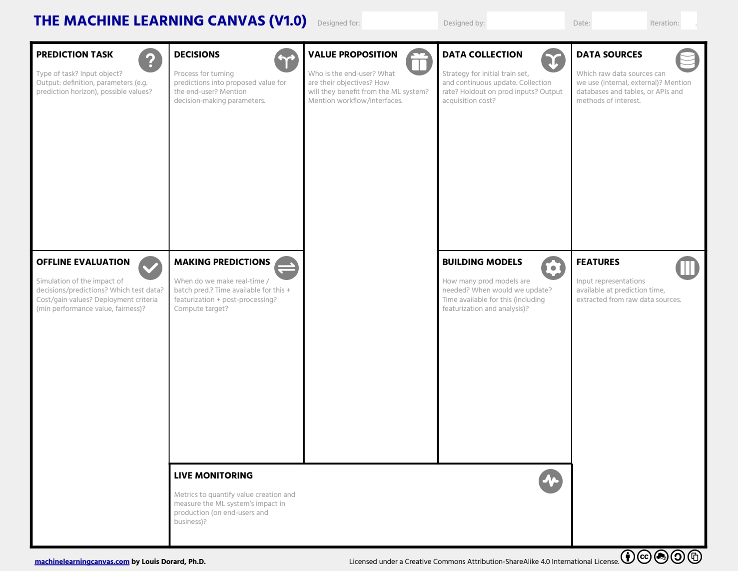
- Attention compares **pairs** of inputs $\implies$ highlights interesting pairs
- Most current state-of-the-art architectures (e.g. Transformers, GPT-3) are based on this



Final remarks: make use of existing processes to plan projects

Example: The machine learning canvas

Source: machinelearningcanvas.com



Roadshow: ML in Belle II

Broad categories of uses

In Belle II we divide into three main categories[1]:

Simulation	Simulate detector responses Goal: Speed up simulation time
Reconstruction	Identify particle candidates from detector responses Goal: Improve the accuracy of particle finding
Analysis	Use reconstructed particles to measure something Goal: Improve signal/background separation.

[1] I have simplified the goals here a lot, please don't be mad :(

Broad categories of uses

In Belle II we divide into three main categories[1]:

Simulation Simulate detector responses
Goal: Speed up simulation time

Reconstruction Identify particle candidates from detector responses
Goal: Improve the accuracy of particle finding

Analysis Use reconstructed particles to measure something
Goal: Improve signal/background separation.

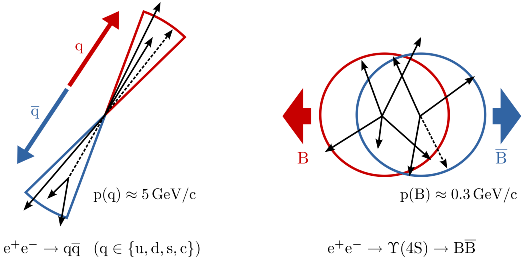
We've seen how ML works, let's look at some examples of its use in Belle II...

[1] I have simplified the goals here a lot, please don't be mad :(

Focus: separate $b\bar{b}$ events from $e^+e^{\rightarrow}q\bar{q}$

Motivation: $q\bar{q}$ are lighter $\Longrightarrow$ more kinetic energy $\Longrightarrow$ jet-like decays

Solution: Use kinematics to separate based on decay shape



Analysis use case
Continuum suppression

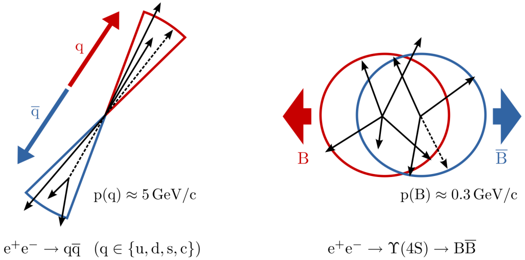
Focus: separate $b\bar{b}$ events from $e^+e^- \to q\bar{q}$

Motivation: $q\bar{q}$ are lighter $\implies$ more kinetic energy $\implies$ jet-like decays

Solution: Use kinematics to separate based on decay shape

Current approach: **FastBDT**

- Requires: Fixed input features
- Inputs: High-level, engineered variables
- Limitations: Compresses all particles' kinematics into fixed number of features



$$\text{e.g.: Thrust} = \max_{\vec{n}} \frac{\sum_j |\vec{p}_j \cdot \vec{n}|}{\sum_j |\vec{p}_j|}$$

Analysis use case
Continuum suppression

Focus: separate $b\bar{b}$ events from $e^{+}e^{\rightarrow}q\bar{q}$

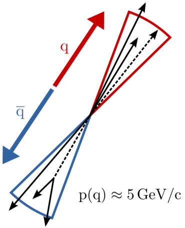
Motivation: $q\bar{q}$ are lighter $\implies$ more kinetic energy $\implies$ jet-like decays

Solution: Use kinematics to separate based on decay shape

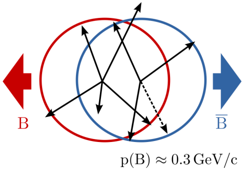
Current approach: **FastBDT**

- Requires: Fixed input features
- Inputs: High-level, engineered variables
- Limitations: Compresses all particles' kinematics into fixed number of features

Ideal approach: use the kinematics of individual particles



$\mathrm{e^+e^-} \to \mathrm{q\overline{q}} \quad (\mathrm{q} \in \{\mathrm{u, d, s, c}\})$



$\mathrm{e^+e^-} \to \Upsilon(4\mathrm{S}) \to \mathrm{B\overline{B}}$

$$\text{e.g.: Thrust} = \max_{\vec{n}} \frac{\sum_j |\vec{p}_j \cdot \vec{n}|}{\sum_j |\vec{p}_j|}$$

Analysis use case

Continuum suppression

Focus: separate $b\bar{b}$ events from $e^+e^\rightarrow q\bar{q}$

Motivation: $q\bar{q}$ are lighter $\Longrightarrow$ more kinetic energy $\Longrightarrow$ jet-like decays

Solution: Use kinematics to separate based on decay shape

Current approach: **FastBDT**

Requires: Fixed input features

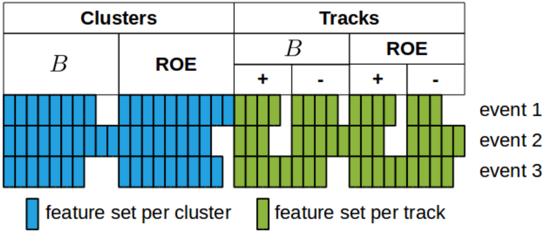
Inputs: High-level, engineered variables

Limitations: Compresses all particles' kinematics into fixed number of features

Ideal approach: use the kinematics of individual particles

First attempt by D. Weyland[6]:

- Deep continuum suppression
- Use a fully-connected network
- Take at most top N momentum particles
- Still not ideal...



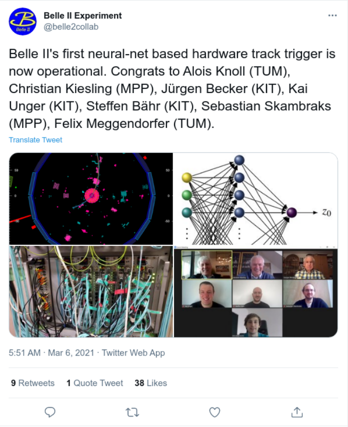
Current work looking into **attention-based** neural networks

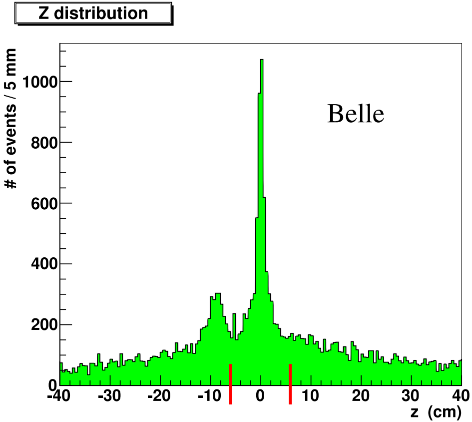
Reconstruction use case
Neuro-z trigger

Focus: Use z vertex of tracks to filter out background events

Approach: Use CDC (2D) hits to estimate z-vertex (and θ) of each track

Requirement: Inference performed fast ($\sim$ 2 μs)





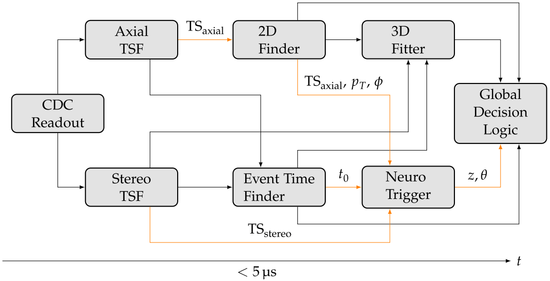
Reconstruction use case
Neuro-z trigger

Focus: Use z vertex of tracks to filter out background events

Approach: Use CDC (2D) hits to estimate z-vertex (and θ) of each track

Requirement: Inference performed fast ($\sim 2\,\mu$s)

- Insert a fully-connected network into the trigger pipeline
- Embed trained network onto FPGA hardware
- A good example of when simplicity is priority
  $\Longrightarrow$ single hidden layer



Reconstruction use case

Neuro-z trigger

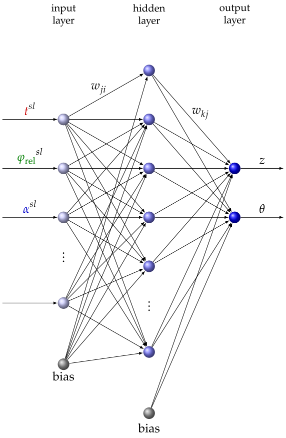
Focus: Use z vertex of tracks to filter out background events

Approach: Use CDC (2D) hits to estimate z-vertex (and θ) of each track

Requirement: Inference performed fast ($\sim 2\ \mu$s)

- Insert a fully-connected network into the trigger pipeline
- Embed trained network onto FPGA hardware
- A good example of when simplicity is priority

  $\Longrightarrow$ single hidden layer



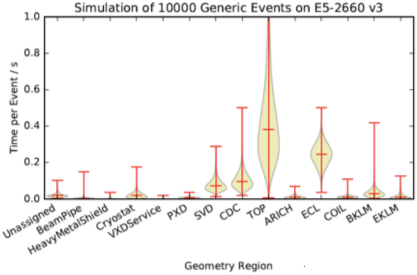
Simulation use case (in development)

Fast TOP simulation

Focus: Speed up the slowest part of event simulation

Approach: Learn how to propagate photons through the TOP bars

Requirement: Distribution of photons hits for particle types (e.g. K^+, π^+) preserved



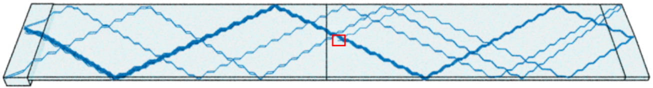


Figure: Tracks of 100 photons starting at (0,0,0) with $\phi, \theta, \psi = 45°$

Simulation use case (in development)

Fast TOP simulation

Focus: Speed up the slowest part of event simulation

Approach: Learn how to propagate photons through the TOP bars

Requirement: Distribution of photons hits for particle types (e.g. K^+, π^+) preserved

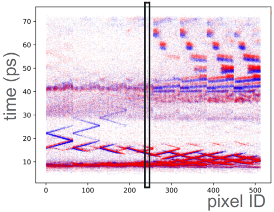


Figure: Pixel hits for 10k Pions (blue) and Kaons (red)

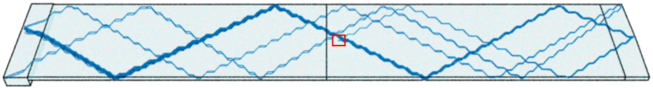


Figure: Tracks of 100 photons starting at (0,0,0) with $\phi, \theta, \psi = 45°$

Simulation use case (in development)

Fast TOP simulation

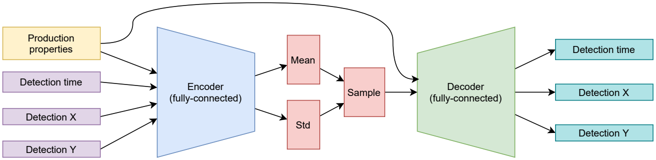
Focus: Speed up the slowest part of event simulation

Approach: Learn how to propagate photons through the TOP bars

Requirement: Distribution of photons hits for particle types (e.g. K^+, π^+) preserved

- A fully-connected network is **deterministic** but photon transport is **stochastic**
- Solution: conditionalise with photon initial conditions
- Conditional variational autoencdoer (C-VAE)

  $\Longrightarrow$ use **decoder** for inference



Are Neural Networks the future of particle physics?

Are Neural Networks the future of particle physics?

Yes

Are Neural Networks the future of particle physics?

Yes...for now

Are Neural Networks the future of particle physics?

Yes...for now

If they're so great, why aren't they everywhere?

Are Neural Networks the future of particle physics?

Yes...for now

If they're so great, why aren't they everywhere?

Three main problems (that I see):

1. Lack of reliable uncertainties
2. Decorrelation to prevent biasing measurements
3. Lack of expertise

Discussion points

If we have spare time

- Dealing with uncertainties
- Graph neural networks
- Implementation in `basf2`
- Other points...?

References

References I

[1] M. Hardt and B. Recht, "Patterns, predictions, and actions: A story about machine learning", arXiv e-prints, arXiv:2102.05242, arXiv:2102.05242 (2021).

[2] HEP ML Community, *A Living Review of Machine Learning for Particle Physics*,

[3] K. Albertsson et al., "Machine Learning in High Energy Physics Community White Paper", in Journal of physics conference series, Vol. 1085, Journal of Physics Conference Series (Sept. 2018), p. 022008.

[4] T. Keck, "FastBDT: A Speed-Optimized Multivariate Classification Algorithm for the Belle II Experiment", Comput. Softw. Big Sci. **1**, 2 (2017).

[5] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, "Deep Double Descent: Where Bigger Models and More Data Hurt", arXiv e-prints, arXiv:1912.02292, arXiv:1912.02292 (2019).

[6] D. Weyland, "Continuum Suppression with Deep Learning techniques for the Belle II Experiment", MA thesis (KIT, Karlsruhe, ETP, Nov. 2017).