NEURAL NETWORKS AND DEEP LEARNING

SURG70098 - SURGICAL DATA SCIENCE AND AI

STUART BOWYER

INTENDED LEARNING OUTCOMES

1. Understand the concept behind neural networks and how they 'learn'
2. Be able to build simple neural networks to address regression and classification problems
3. Be aware of the limitations and challenges of neural networks and where to look to address them
4. Have a preliminary understanding of what deep learning is and what it can be used for
5. Be able to understand and address class imbalances and hyperparameters
6. Know the fundamental concepts behind ensemble learning models

MIMIC DATASET

The following code will load the datasets used in this lecture notes

In [ ]:

```
%pip install pandas_gbq

import pandas as pd
import pandas_gbq

project_id = 'mimic-project-439314'  # @param {type:"string"}

df_day1_vitalsign = pandas_gbq.read_gbq("""
  SELECT
    *,
    (dod IS NOT NULL) AND (dod <= dischtime) AS mortality,
    weight / POWER(height/100, 2) > 30 AS obese
  FROM `physionet-data.mimiciv_derived.first_day_vitalsign`
  LEFT JOIN (
    SELECT
      subject_id,
      stay_id,
      gender,
      race,
      dischtime,
      admission_age,
      dod
    FROM
      `physionet-data.mimiciv_derived.icustay_detail`
  )
  USING(subject_id, stay_id)
  LEFT JOIN (
    SELECT
      stay_id,
      AVG(weight) as weight
    FROM
      `physionet-data.mimiciv_derived.weight_durations`
    GROUP BY
      stay_id
  )
  USING(stay_id)
  LEFT JOIN (
    SELECT
      stay_id,
      CAST(AVG(height) AS FLOAT64) AS height
    FROM
      `physionet-data.mimiciv_derived.height`
    GROUP BY
      stay_id
  )
  USING(stay_id)
  WHERE heart_rate_mean IS NOT NULL
""", project_id=project_id)
```

RECAP SUPERVISED LEARNING

- Last week, we studied a few methods for supervised learning, all of these have some limitations
    - All methods involve some assumptions about model structure (e.g. linearity, mapped linearity)
    - KNN and SVM do not scale well to very large data sets or high dimensionality
    - Linear and logistic regression require feature engineering
    - Decision trees are prone to overfitting
    - And more...

NEURAL NETWORKS

- Neural networks are a machine learning method that aims to overcome these issues
- Particularly with respect to:
  - Large data
  - High-dimensionality data
  - Complex interdependent data
  - (Unstructured data)

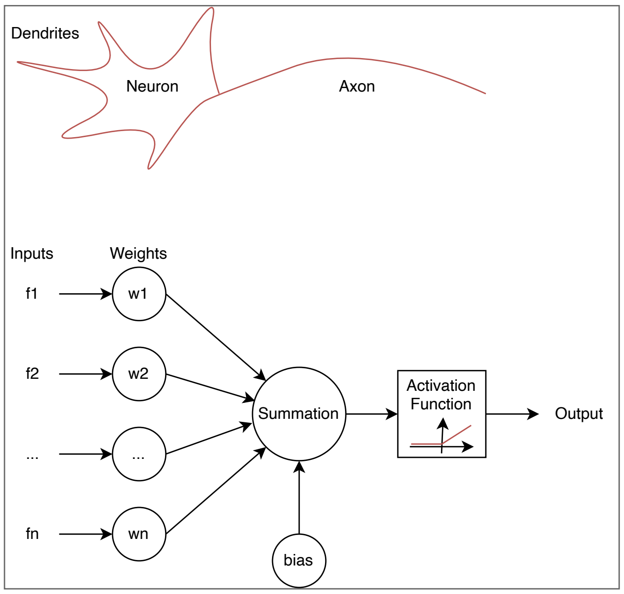
INTRODUCTION TO NEURAL NETWORKS

BIOINSPIRATION

- These models were created to mimic the function of biological neurons
- By combining many neurons they are able to
- In reality, the aritifical neural networks are quite different from biological neural networks; however, they are very effective

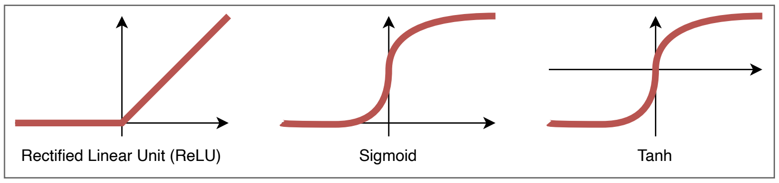
ARTIFICAL NEURONS

Neural networks are made up from multiple artificial neurons



ACTIVATION FUNCTIONS

- Activation functions are a critical component of the neural network
- They produce non-linearity
- Therefore, allow it to model complex relationships
- There are a range of functions



QUICK EXERCISE - ARTIFICIAL NEURON WORKED EXAMPLE

For a three input neuron with the following input values and parameters, can you calculate (with simple Python) the output value of the neuron for the input `A Values` and the `B Values`?

You should get a separate output value for each input set

PARAMETERS

Parameter	Value
weight 1	0.5
weight 2	1.7
weight 3	0.001
bias	-52
Activation function	ReLU

INPUTS

Input	A Values	B Values
input 1	65 kg	82 kg
input 2	1.7 m	1.85 m
input 3	8200 c/uL	9800 c/uL

POSSIBLE PYTHON SOLUTION

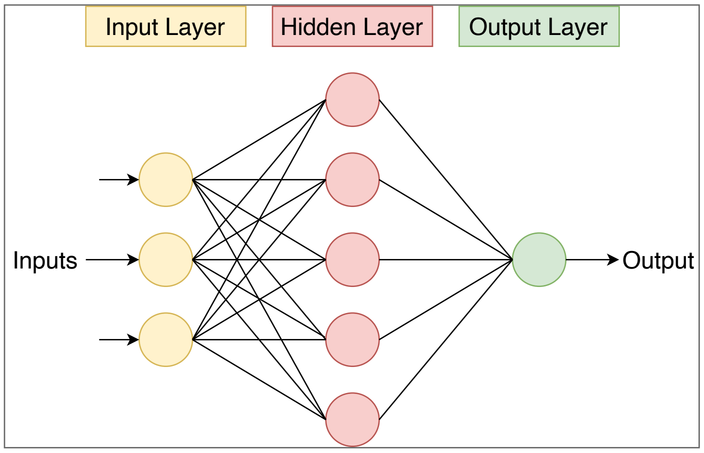
In [16]:

```
w1 = 0.5
w2 = 1.7
w3 = 0.001
bias = -52

def relu(x):
    if (x < 0):
        return(0)
    return(x)

va1 = 65
va2 = 1.7
va3 = 8200

vb1 = 82
vb2 = 1.85
vb3 = 9800

output_a = relu(w1 * va1 + w2 * va2 + w3 * va3 + bias)
output_b = relu(w1 * vb1 + w2 * vb2 + w3 * vb3 + bias)

print(f'Neuron output for A is {output_a}')
print(f'Neuron output for B is {output_b}')
```

```
Neuron output for A is 0
Neuron output for B is 1.9450000000000074
```

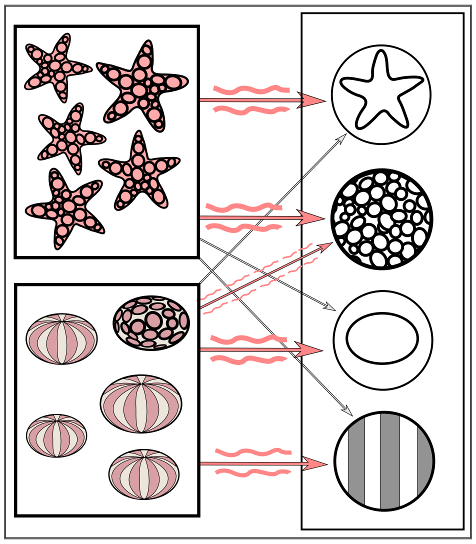
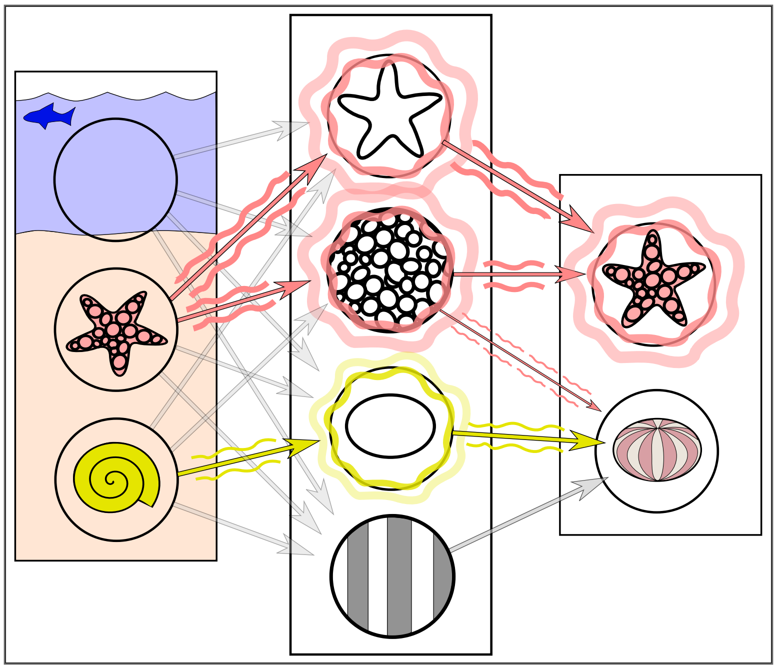
NEURAL NETWORK

- An assembly of artificial neurons in a network structure
- Neurons are grouped into layers of three different types



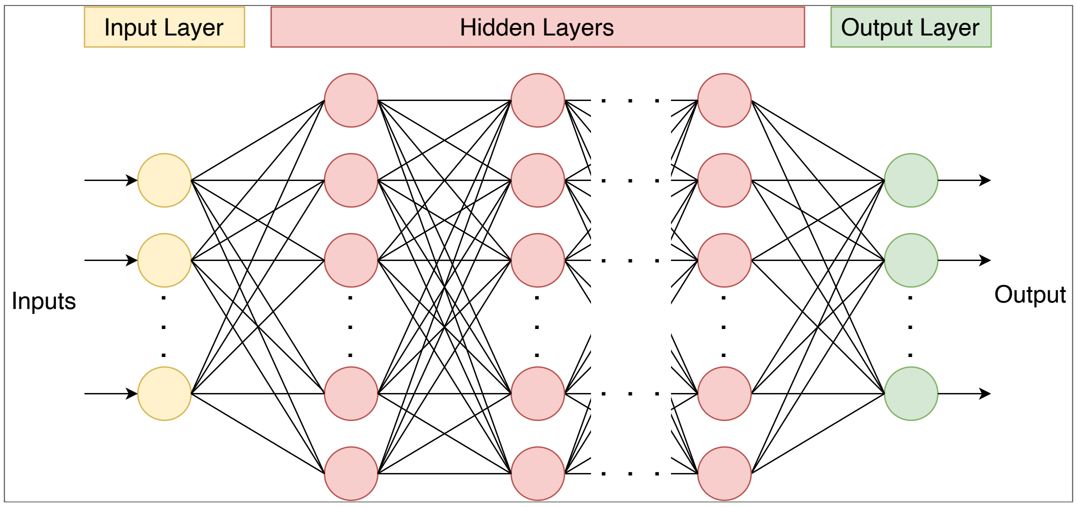
WHY NEURAL NETWORKS WORK

- Each individual neuron can model a simple activation function
- By combining them in this structure, the combination of simple activation functions can model non-linear and complex relationships

Training	Predicting
	
By Mikael Häggström, M.D. Author info- Reusing images- Conflicts of interest: NoneMikael Häggström, M.D. - Own work, CC0, Link	By Mikael Häggström, M.D. Author info- Reusing images- Conflicts of interest: NoneMikael Häggström, M.D. - Own workReference: Ferrie, C., & Kaiser, S. (2019) Neural Networks for Babies, Sourcebooks ISBN: 1492671207., CC0, Link

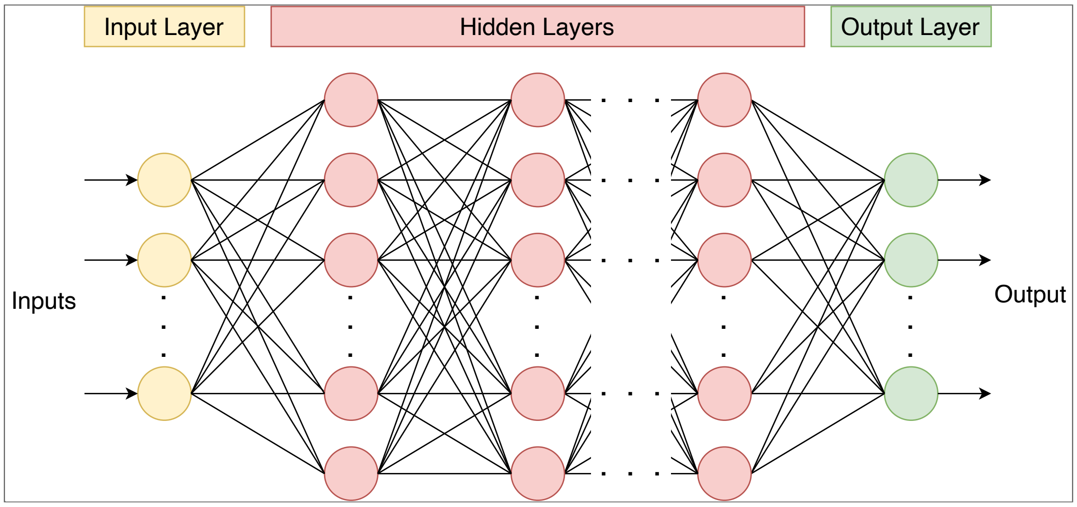
COMPLEX NEURAL NETWORK ARCHITECTURE

Networks with many neurons and many layers can model highly complex relationships



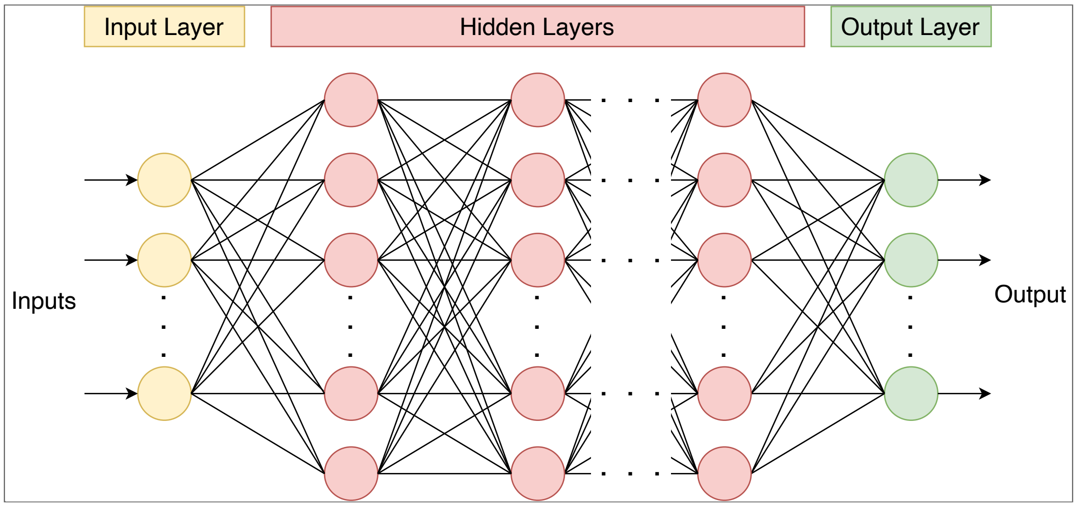
INPUT LAYER

- **Function:** Takes the raw data and sends it into the network without any transformation or activation
- **Number:** The number of neurons in the input layer is the number of features you have
- **Connection:** Each input neuron is connected to each neuron in the first hidden layer
- **Example:** Each input neuron would be one of the features you observed per patient



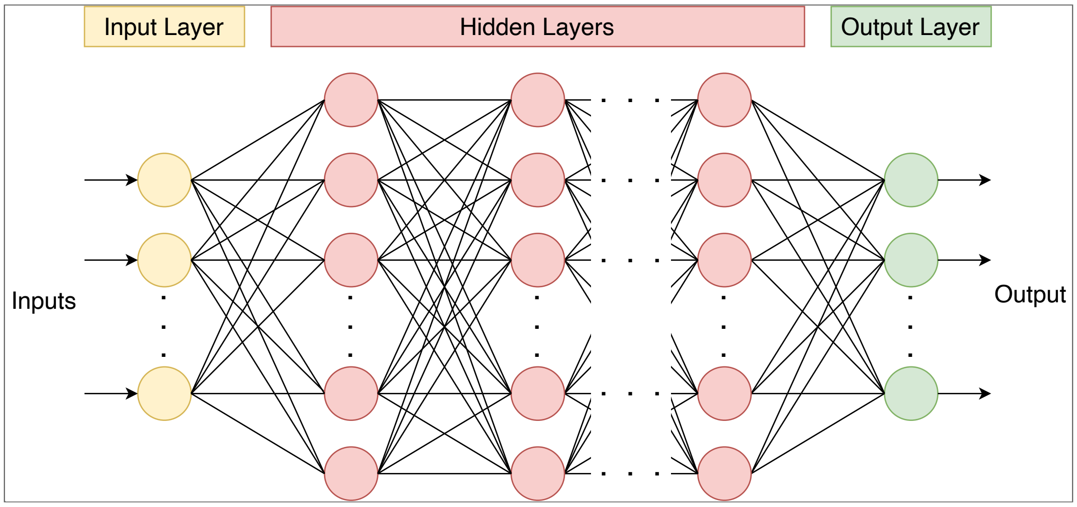
HIDDEN LAYER(S)

- **Function:** Allow for the translation of the input data into increasingly abstract representations that model them
- **Number of neurons:** The number of neurons per layer is a model parameter. More neurons can model more complex relationships.
- **Number of layers:** The number of layers is a model parameter. More layers can model deeper level features.
- **Connection:** Each neuron is (**typically**) connected to each neuron in the next layer
- **Example:** The first layer could learn to derive BMI from height and weight



OUTPUT LAYER

- **Function:** Takes the abstracted data representation from the final hidden layer and produces the final prediction/classification
- **Number (regression):** A single output neuron combines the network data into a single value
- **Number (binary classification):** A single output neuron with a value between 0 and 1
- **Number (multi-classification):** One output neuron per output class
- **Example:** For a mortality prediction the output layer would have one neuron that gives a probability of true/false



SOFTMAX

- There is a challenge with multi-classification problems where you want to have a probability value per class ...
- Consider a classifier with three possible outputs (high, medium, and low risk) and your output neurons produce `[2.2, 0.4, 0.1]`, but you want to know the probability of being in each risk category
- Softmax is a function that allows you to combine your output layer values together and produce a probability distribution across the classes such that all sum to 1 (i.e. probabilities)
  - e.g. `[2.2, 0.4, 0.1]` becomes `[0.78, 0.13, 0.10]`
- This function is computationally efficient during training (which we will consider next)

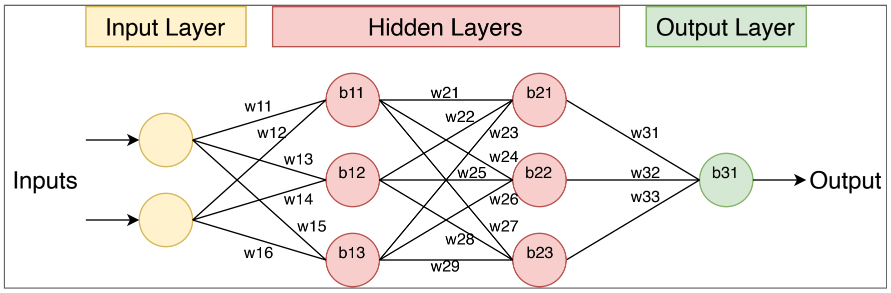
SUMMARY

Aspect	Input Layer	Hidden Layer(s)	Output Layer
Role	Receives and passes input data to the network	Processes data, learns features and patterns	Generates the final prediction or output
Data Handling	Takes raw input data (e.g., features, images)	Transforms data through weighted connections	Produces output suitable for the task (class/probability/value)
Activation Function	None	Non-linear functions (e.g., ReLU, Tanh, Sigmoid)	Depends on the problem: Softmax, Sigmoid, Linear
Number of Neurons	Equal to the number of input features	Varies based on network architecture	Depends on the output type (e.g., 1 for regression, n for n-class classification)
Complexity	Simple; no learning occurs in this layer	Complex; learns hierarchical features and patterns	Simple; maps learned features to output space
Learning Role	No learning or transformations	Learns and extracts meaningful features	Outputs the final result of the network's learning
Example	Each neuron represents a patient feature (e.g., age, heart rate, blood pressure)	Neurons learn interactions between features (e.g., BMI)	Outputs probability of mortality (e.g., risk score or binary prediction)

TRAINING NEURAL NETWORKS

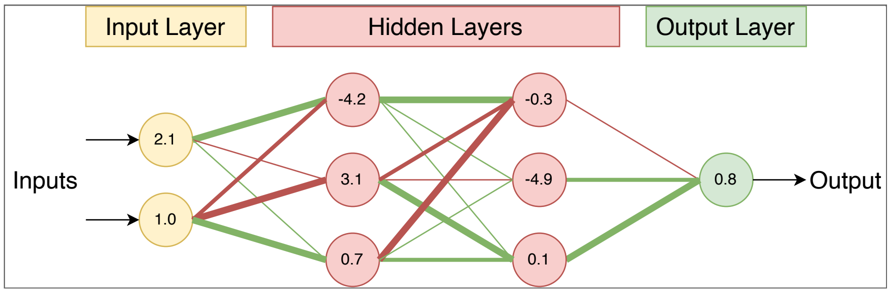
NETWORK INITIALISATION

- To start training a neural network on data, you need an initial 'guess' at the weights/biases
- You can start by assigning the values randomly; however, the distribution for should be carefully considered to avoid training instability (see later on)
- Methods such as Xavier (Glorot) or He (Kaiming) initialisation define distributions to sample from to for different activation functions



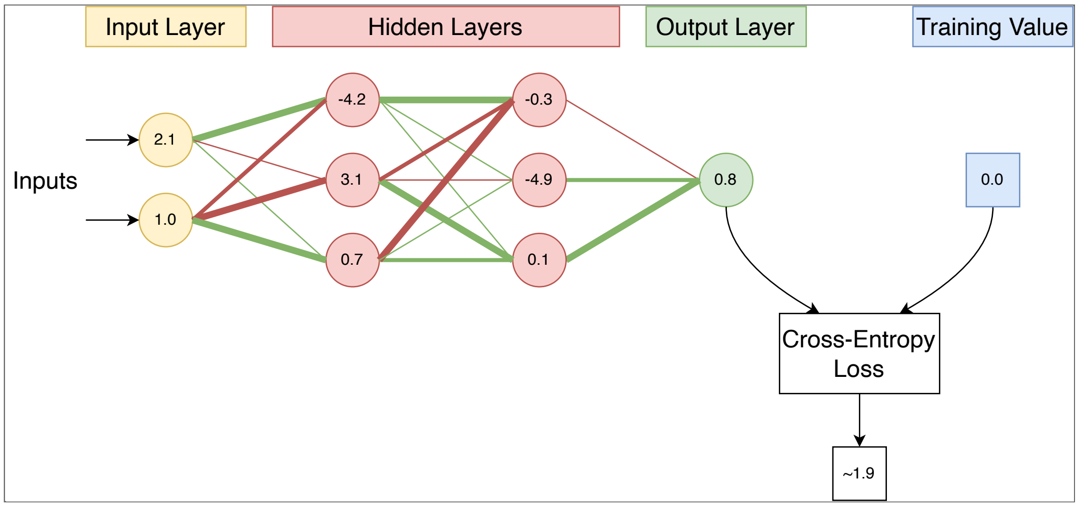
FORWARD PROPAGATION

- The next step is to simply use your (semi-randomly initialised) network to make a prediction for your training data
- For example:
  - we have a `2.1` and `1.0` as the two features of the first sample in our training data
  - the neural network output predicts a probability of `0.8`
  - however, the real value should be False (`0.0`)
  - the model is currently poorly trained



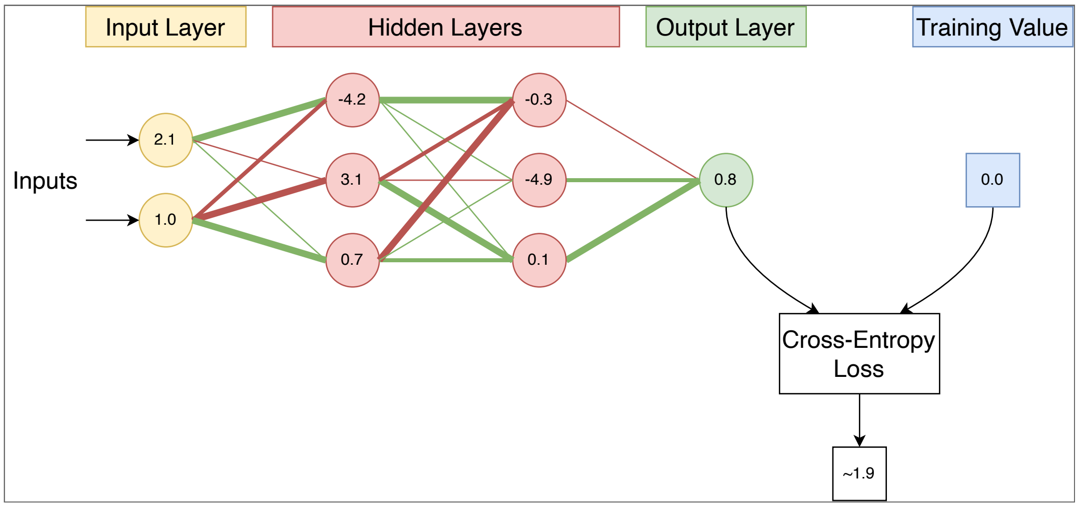
LOSS FUNCTIONS

- To train the model, we need a 'loss function'
- Simplistically, this is a measure of how 'bad' the model is at predicting
  - i.e. a loss function of 0 would be correct
- **Regression:** typically use the 'Mean Squared Error' (MSE)
- **Classification:** typically uses the 'Cross-Entropy Loss'
- There are others and you should explore these when you build a model - justify your choice



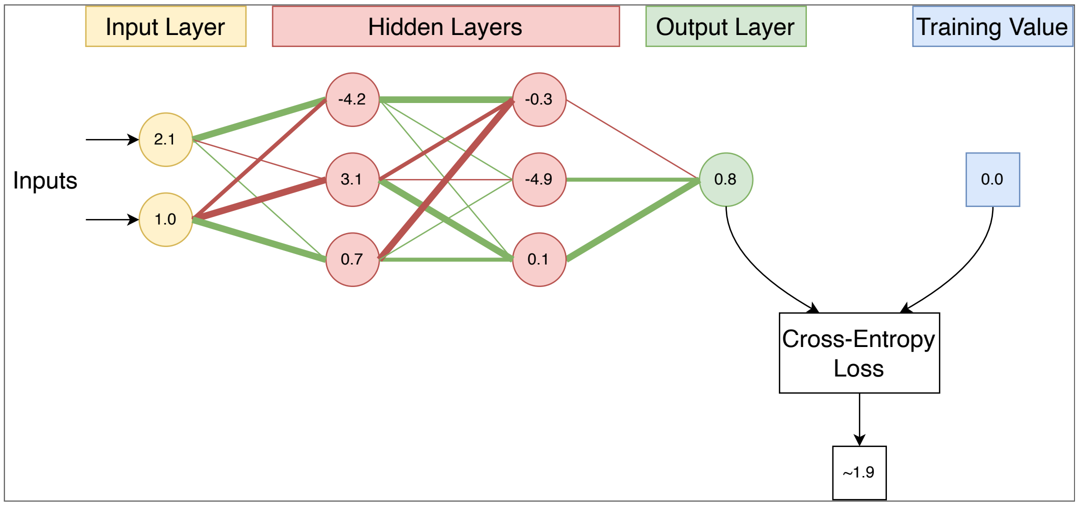
HOW DO WE UPDATE THE MODEL TO MAKE IT BETTER?

We want the cross-entropy loss to as low as possible, which means we want the output layer to predict the same as the training data ...



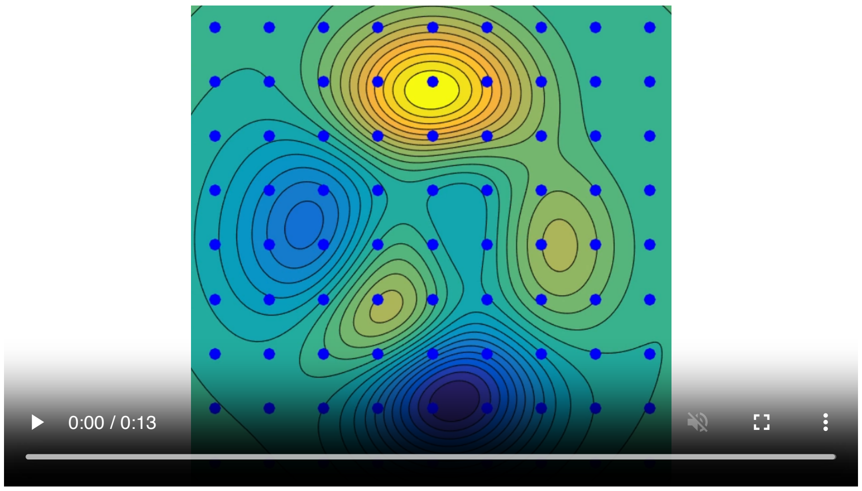
BACK PROPAGATION - CONCEPTUALLY

- The idea is to 'nudge' the weights in the network so they slightly reduce the loss at each iteration
- We step backwards through the neural network, at each layer:
  - Adjust the weights and bias so that the output moves in the direction that minimises the loss
  - Adjust the weights such that larger changes are applied to greater improvements
  - Propogate the required changes (errors) in the previous layer's outputs
- Once we reach the input layer, the neural network should be slightly better at the trained sample
- Repeating this process over and over causes the network's parameters to converge on a trained configuration



BACK PROPAGATION - COMPUTATIONALLY

- In practice, you want to minimise the loss for all your training samples at once
  - So you combine them into a single loss (or cost) function
- Python performs back propagation by computing the gradient of the cost function in the model parameters and adjusting them appropriately
- How big a 'step' the solver takes at a time is controlled by the learning rate
- This is a common computational approach called 'Gradient Descent'



By Gpeyre, CC-SA 4.0
https://commons.wikimedia.org/wiki/File:Gradient_Descent_in_2D.webm

TRAINING CHALLENGES

There are some challenges in training neural networks ...

- **Vanishing Gradients:** learning can become very slow or stop in early layers due to the very shallow gradients of the sigmoid and tanh activation functions
- **Exploding Gradients:** learning becomes unstable due to large gradients resulting from poor initialisation
- **Dead ReLU Units:** learning stops because ReLU neurons get stuck in the 0 region with no gradient to recover

NEURAL NETWORK CLASSIFIER IN `scikit-learn`

- In `scikit-learn` the typical neural network is the `MLPClassifier`
- This is a Multi-Layer Perceptron (a simple feedforward neural network)

In [ ]:

```
from sklearn.neural_network import MLPClassifier
```

EXERCISE - MORTALITY PREDICTION

Take your mortality prediction models from last week's tutorial and try to use a neural network classifier, validate and compare your performance to the linear classifier.
Explore different model parameters:

- What does increasing the number/size of layers achieve?
- Which activation functions work?
- What happens when you increase/decrease the learning rate?

What configuration gives you the best performance?
What do you think is limiting your ability to get better performance?

SIMPLE SOLUTION

In [72]:

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import cross_validate, train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score, roc_auc_score
from imblearn.over_sampling import RandomOverSampler

# Prepare our data
data = df_day1_vitalsign.dropna(subset=['admission_age', 'heart_rate_mean', 'sbp_mean', 'glucose_me
X = data[['admission_age', 'heart_rate_mean', 'sbp_mean', 'glucose_mean']]
Y = data['mortality']

# Create the Neural network with 20 neurons in the hidden layer
model = MLPClassifier(
    hidden_layer_sizes=(20),
    random_state=1
)

# Train Test Split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=1)

# Train the model
model.fit(X_train, Y_train)

# Test the model
Y_pred = model.predict(X_test)

# Metrics
print(f"Confusion Matrix:\n {confusion_matrix(Y_test, Y_pred)}")
print(f"Accuracy: {accuracy_score(Y_test, Y_pred)}")
print(f"f1-score: {f1_score(Y_test, Y_pred)}")
print(f"ROC AUC:  {roc_auc_score(Y_test, Y_pred)}")

# Cross validate to assess the balanced performance
nn_scores = cross_validate(model, X, Y, cv=5, scoring=['f1','accuracy','precision','recall','roc_au

print(f'N:                                {len(Y)}')
print(f'Cross-Validation Accuracy Mean: {nn_scores["test_accuracy"].mean()}')
print(f'Cross-Validation F1 Mean:       {nn_scores["test_f1"].mean()}')
print(f'Cross-Validation ROC AUC Mean:  {nn_scores["test_roc_auc"].mean()}')
```

```
Confusion Matrix:
 [[12550    55]
 [ 1508   109]]
Accuracy: 0.890099845310083
f1-score: 0.12240314430095452
ROC AUC:  0.5315227169083349
N:                              71110
Cross-Validation Accuracy Mean: 0.8869076079313739
Cross-Validation F1 Mean:       0.10674687476466245
Cross-Validation ROC AUC Mean:  0.7005425104687292
```

DEEP NEURAL NETWORKS

Deep neural networks are the most active area of ML research, here we will introduce the concept and give a few examples of approaches

INTRODUCTION

- Deep learning is the process of using neural networks to model highly complex relationships between data
- Simplistically, these are neural networks with many hidden layers (maybe 10s or even 100s)
- The idea is that more layers/neurons can abstract concepts to a much greater degree
- Modelling these complex relationships requires **LARGE** datasets
- Deep neural networks often involve more complicated neuron arrangements (e.g. RNN, CNN, etc.)

DEEP NEURAL NETWORK EXAMPLES

DENSENET-264

- Convolutional Neural Network architecture
- 264 layers
- 33 million parameters

ALPHAFOLD FROM DEEPMIND (ALLEGEDLY...)

- Hybrid architecture
- Dozens of layers
- ~100 million parameters
- Trained with ~1,000 TPUs for several months

GPT-4 FROM OPENAI (ALLEGEDLY...)

- Transformer architecture
- 120 layers
- 1.8 trillion parameters
- Trained with 1,000s of GPUs over 90-100 days (cost of $63M)

ADVANTAGES

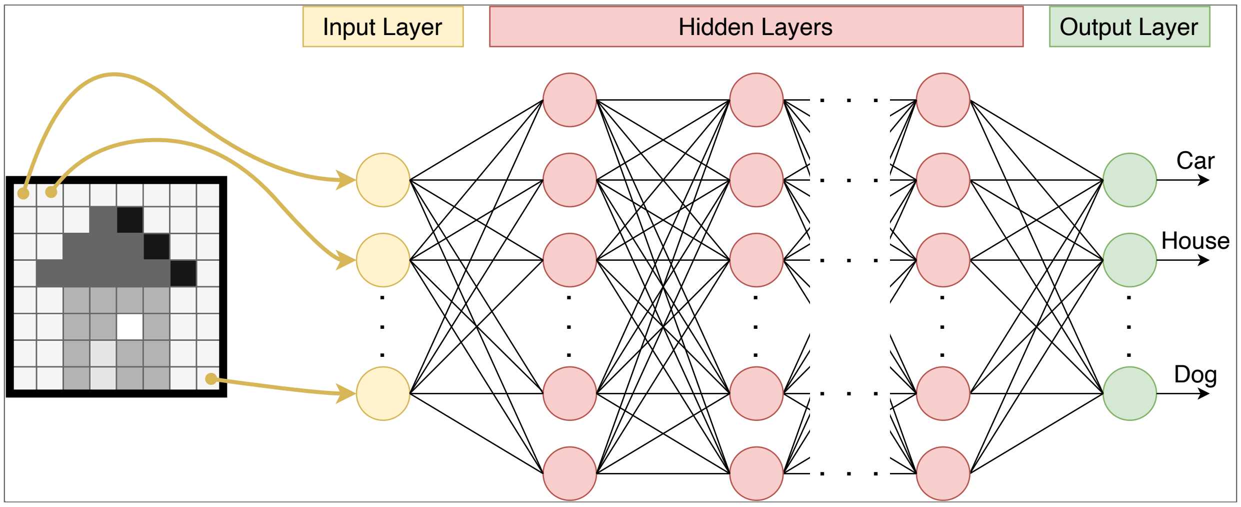
- **High Performance:** these models are repeatedly shown to perform well on complex problems
- **Model Abstraction/Translation:** the high degree of abstraction in hidden layers allows them to be applied/translated to related tasks
- **Feature Engineering:** the model's ability to abstract the input data means it can find/engineer features itself
- **Utilise Large and High-Dimensional Data:** the models are capable of learning detail from massive amounts of data with thousands of dimensions
- **Flexibility:** the architecture can be configured to specialise on several types of data/problems

DISADVANTAGES

- **Data Requirements:** massive datasets are required for training to learn complex relationships
- **Computational Cost:** training can take a long time and require expensive resources
- **High Parameterisation:** models have many parameters that need setting to perform optimally
- **Overfitting:** massive numbers of parameters can end up overfitting the data
- **Interpretability:** complex architectures and large numbers of parameters make the models almost impossible to interpret/explain

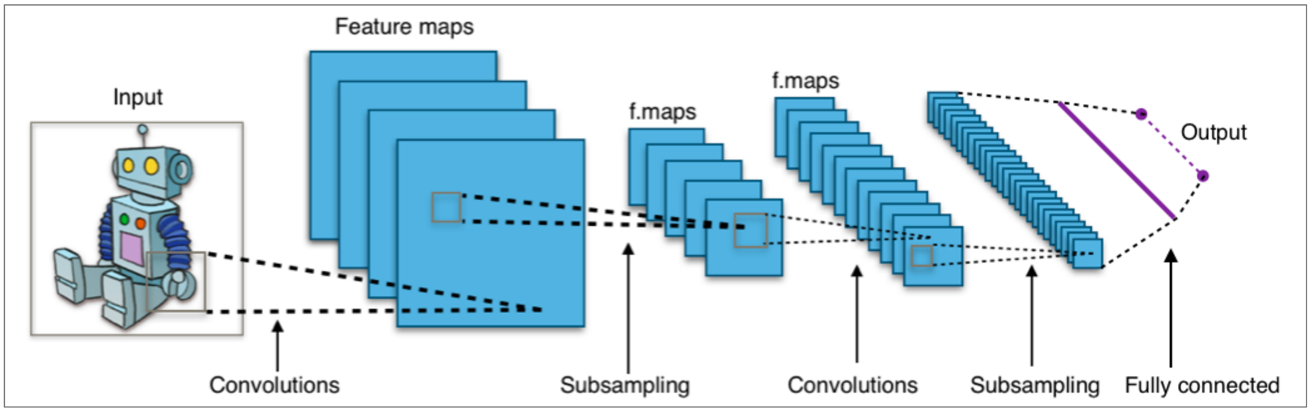
NEURAL NETWORKS FOR COMPUTER VISION

- Neural networks were crucial in the development of computer vision, e.g.:
    - Image classification
    - Object detection
    - Image generation
    - etc.
- To train a neural network on 'unstructured' data such as a medical image you need to 'flatten' the data
- Each pixel in the image becomes an individual feature, and is passed to the input layer of the neural network



CONVOLUTIONAL NEURAL NETWORKS (CNN)

- Special neurons are used in the convolution layers where a small filter is applied to the data
- These filters identify (initially) simple elements of the image (e.g. edges)
- By layering these filters, they can learn increasingly complex patterns (e.g. edges, windows, houses)
- The structure of these filters are learnt during training



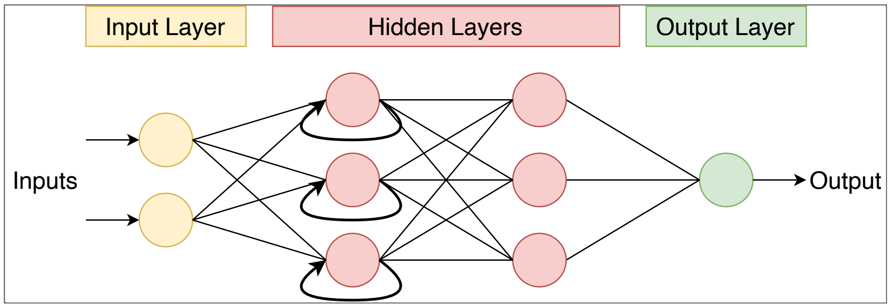
By Aphex34 - Own work, CC BY-SA 4.0, Link

DO YOU THINK THESE NEURAL NETWORKS ARE CAPABLE OF GIVING PREDICTIONS THAT REMEMBER PREVIOUS INPUTS?

For example, imagine you wanted to build a model to predict some adverse event during surgery, based on continuous monitoring data. You need your model to constantly be running on the latest observations

RECURRENT NEURAL NETWORKS

- So far, we have looked at 'feed forward' networks - i.e. the data is fed forward from the input to output layer
    - These are 'stateless' - i.e. cannot remember anything between inputs
    - It does not have to be this way
- Recurrent Neural Networks (RNN) are neural networks with memory
    - Long Short-Term Memory (LSTM) are the most popular RNN architecture that can 'remember' values over a long period and then 'forget' them when necessary



- RNNs were commonly used for timeseries data; however, have fallen out of fashion with the popularity of 'transformers'

TRANSFORMERS

- Transformers do not worry about remembering the state between inputs - they take the whole history as an input each time
- They have revolutionised NLP and other sequential (unstructured) data modelling tasks
- **Key Components**
  - **Self-attention:** allows the model to learn the importance of relationships between input tokens (i.e. surgery event -> infection event)
  - **Positional encoding:** allows the model to understand the sequential order of inputs

MODEL RELIABILITY, REPRODUCIBILITY, AND OPTIMALITY

IN THE LAST TUTORIAL, WHAT DID YOU FIND WITH THE SVM, KNN AND DECISION TREE METHODS WHEN TRYING TO GET GOOD PERFORMANCE?

HYPERPARAMETER TUNING

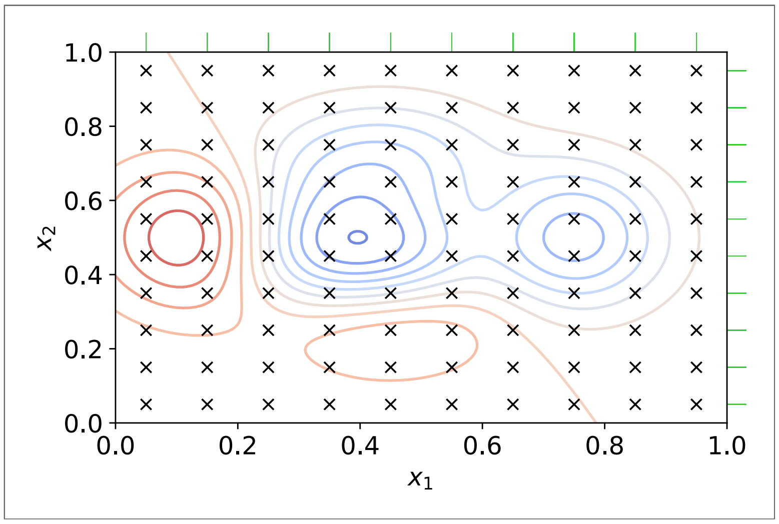
- As models become more powerful, they tend to have increasing numbers of parameters
- For example:
  - Number of neighbours in KNN
  - Depth of a decision tree
  - Layers, neurons, activation functions, etc for neural networks
- We call these **hyperparameters**
- You have seen that variation in these parameters can effect model performance enormously

How do you think we can ensure we have the best configuration for our model's hyperparameters?

GRID SEARCH

Involves systematically searching combinations of hyperparameters to find the best score

- Approach guarantees a 'reasonable' proximity to the optimal values
- `from sklearn.model_selection import GridSearchCV`

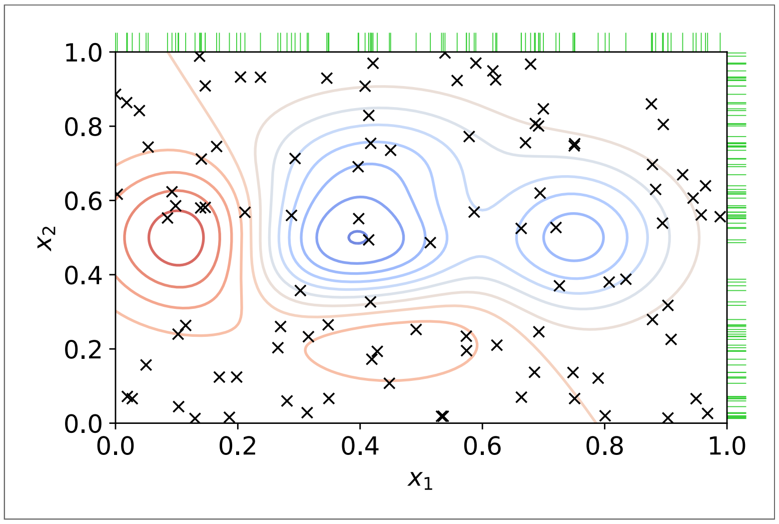


By Alexander Elvers - Own work, CC BY-SA 4.0, Link

RANDOM SEARCH

Involves randomly saerching combinations of hyperparameters to find the best score

- Explores the continuous value space more efficiently than the grid search
- `from sklearn.model_selection import RandomizedSearchCV`

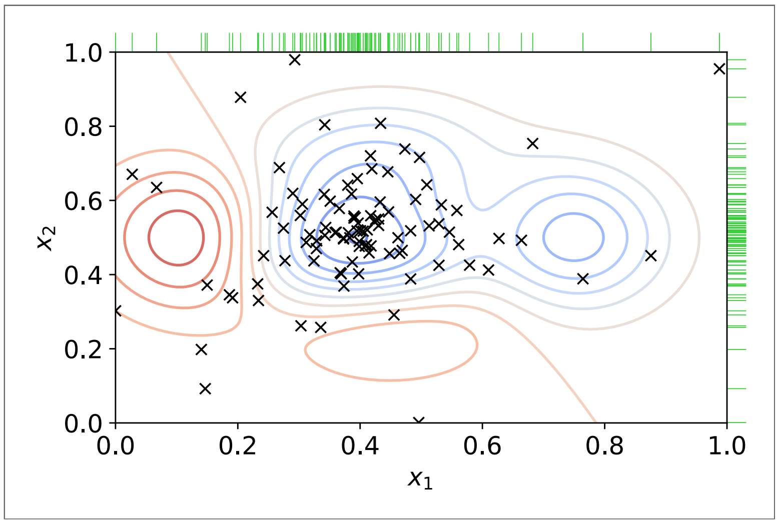


By Alexander Elvers - Own work, CC BY-SA 4.0, Link

BAYESIAN OPTIMISATION

Involves using an optimisation approach to search in areas where it expects the performance to be better

- Typically finds optima more quickly than random search
- `from skopt import BayesSearchCV`



By Alexander Elvers - Own work, CC BY-SA 4.0, Link

GRID SEARCH IN `SCIKIT-LEARN`

In [93]:

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV

data = df_day1_vitalsign.dropna(subset=['admission_age', 'heart_rate_mean', 'sbp_mean', 'glucose_me
X = data[['admission_age', 'heart_rate_mean', 'sbp_mean', 'glucose_mean']]
Y = data['mortality']

# Create the initial model
model = MLPClassifier(random_state=1)

# Define the grid search
grid_search = GridSearchCV(
    model,
    param_grid={
        'hidden_layer_sizes': range(5,45,5),
        'activation': ['relu', 'logistic', 'tanh']
    },
    scoring='roc_auc'
)

# Perform the grid search
grid_search.fit(X, Y)

# Get the best score and best parameters
print(f"Best Score:      {grid_search.best_score_}")
print(f"Best Parameters: {grid_search.best_params_}")
```

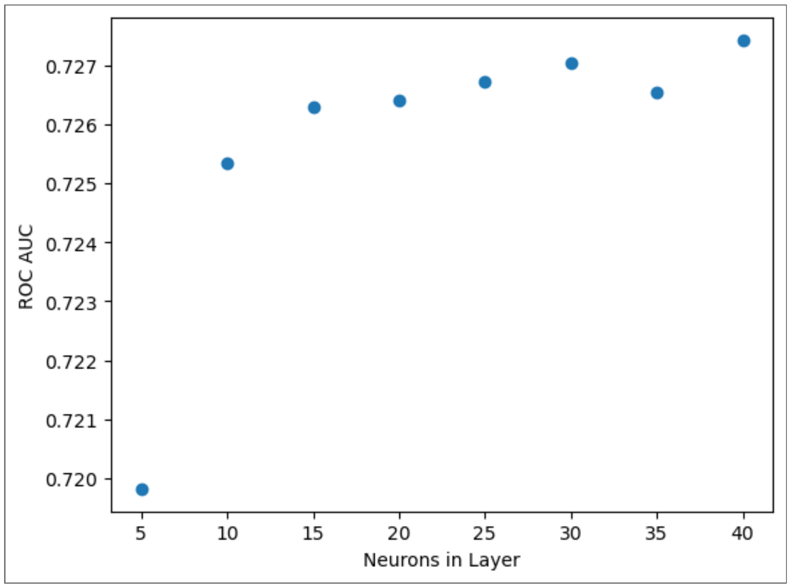
```
Best Score:      0.7274227972755576
Best Parameters: {'activation': 'logistic', 'hidden_layer_sizes': 40}
```

MODEL SENSITIVITY/STABILITY

- You can use the hyperparameter optimisation to assess the sensitivity and stability of a model
- If small changes in parameters lead to large changes in performance, the model is 'sensitive' and maybe overfit
- If large changes in parameters lead to small changes in performance, the model is either already at it's limit, underfit, or suffering from some other issues

In [98]:

```
auc_values = grid_search.cv_results_['mean_test_score'][grid_search.cv_results_['param_activation']
nneuron_values = grid_search.cv_results_['param_hidden_layer_sizes'][grid_search.cv_results_['param

import matplotlib.pyplot as plt

plt.scatter(nneuron_values, auc_values)
plt.xlabel("Neurons in Layer")
plt.ylabel("ROC AUC")
plt.show()
```



CLASS IMBALANCE

- One of the common issues you will face is class imbalances
- i.e. when one of your predicted classes is much more/less common than the others
- For example, in the MIMIC dataset ...

In [100]:

```
df_day1_vitalsign.mortality.value_counts(normalize=True)
```

Out[100]:

```
mortality
False    0.883693
True     0.116307
Name: proportion, dtype: Float64
```

- This imbalance can cause training bias and poor performance

CAN YOU SUGGEST HOW TO ADDRESS THE CLASS IMBALANCE?

OVERSAMPLING

- Involves creating new observations in the minority class by …
- **Random oversampling:** randomly duplicating entries from the minority class
- **Synthetic Minority Over-sampling Technique (SMOTE):** generating new synthetic samples in the minority class by interpolating between existing observations

UNDERSAMPLING

- Involves dropping observations in the majority class by …
- **Random undersampling:** randomly removing entries from the majority class
- **Tomek links:** removes entries from the majority class that are close to the minority class (i.e. suspected noise)
- **NearMiss:** removes entries from the majority class that are far from the minority class (i.e. easy classifications)

USING `imbalanced-learn`

- Python has a package (parallelling `scikit-learn`) for addressing imbalanced datasets
    - **Oversampling methods**
    - **Undersampling methods**

In [108]:

```
from imblearn.over_sampling import RandomOverSampler

data = df_day1_vitalsign.dropna(subset=['admission_age', 'heart_rate_mean', 'sbp_mean', 'glucose_me
X = data[['admission_age', 'heart_rate_mean', 'sbp_mean', 'glucose_mean']]
Y = data['mortality']

# Create the random oversampler
ros = RandomOverSampler(random_state=1)

# Apply it to our dataset
X_res, Y_res = ros.fit_resample(X, Y)

print("Previous value counts: ", Y.value_counts())
print("")
print("Resampled value counts:", Y_res.value_counts())
```

```
Previous value counts:  mortality
False    63076
True      8034
Name: count, dtype: Int64

Resampled value counts: mortality
False    63076
True     63076
Name: count, dtype: Int64
```

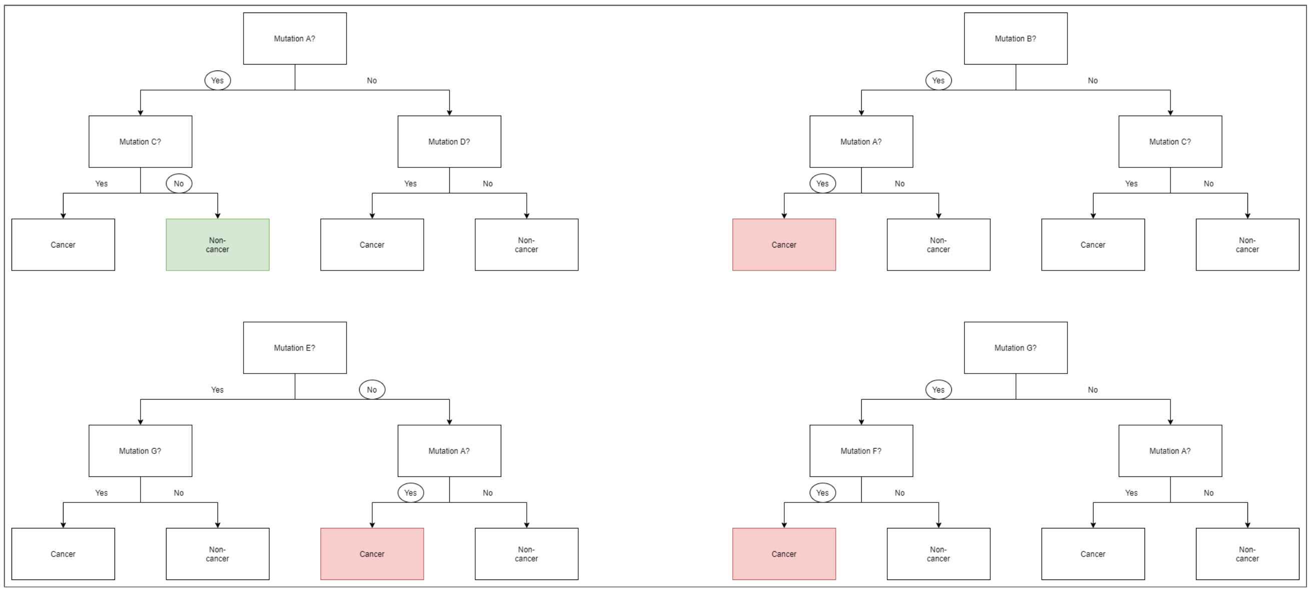
ENSEMBLE METHODS

- You probably have noticed that different models have different advantages and disadvantages
- i.e. sometimes they work well, others they do not
- Ensemble methods combine models together to improve overall performance by ...
  - Improving accuracy
  - Improving stability
  - Reducing error

How would you combine models together to optimise their group performance?

BOOTSTRAP AGGREGATING (BAGGING)

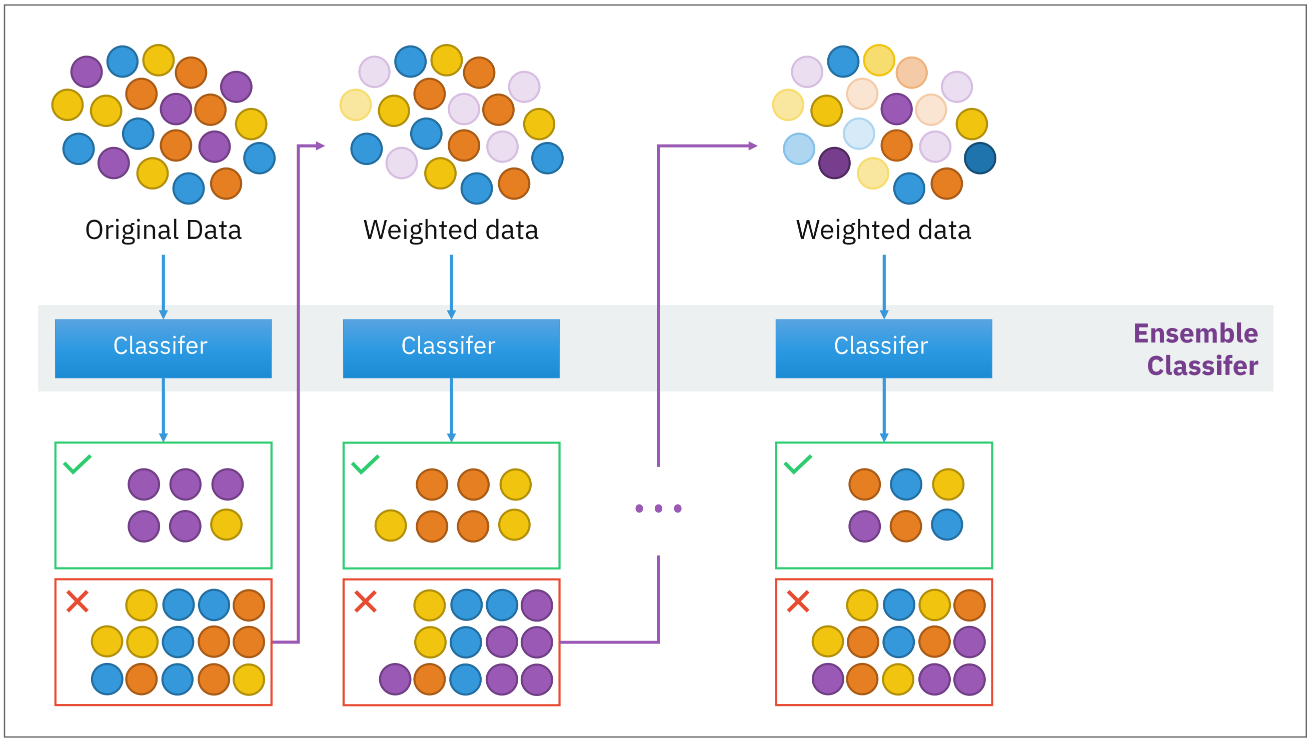
- Builds multiple parallel models independently using random (possibly overlapping) subsets of the data and combines their predictions
- Aim is to reduce overfitting by varying the patterns each model is trained on and reducing variance by combining outputs
- Very commonly used with decision trees to create **random forests**
  - `from sklearn.ensemble import RandomForestClassifier`



By CollaborativeGeneticist - Own work, CC BY-SA 4.0, Link

BOOSTING

- Builds sequential models that try to correct the errors of the predecessor
- Aim is to reduce underfitting (due to weak models) by focusing models on the errors of other models
- Very commonly used with deision trees to create **gradient boosted trees**
  - `from sklearn.ensemble import GradientBoostingClassifier`
  - popular alternative implementation is 'XGBoost'



By Sirakorn - Own work, CC BY-SA 4.0, Link

TUTORIAL EXERCISE

EXERCISE 5.1 - SUPERVISED LEARNING CHALLENGE

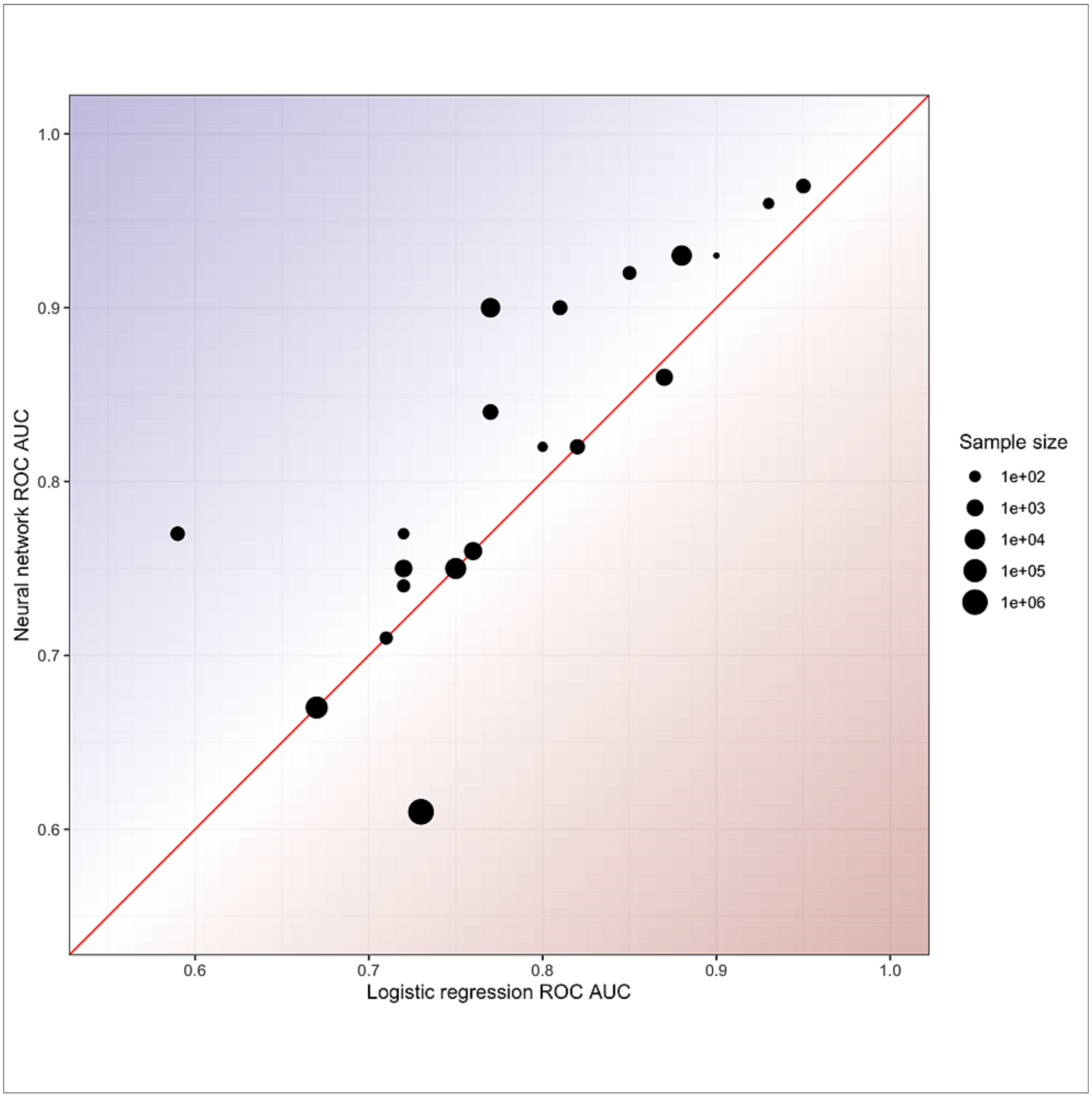
Use the **Wisconsin Breast Cancer Database** to build the most high performance classifier for predicting tumour malignancy from breast mass features.
98% accuracy is possible
There are instructions on importing the dataset to Python on the above page.
I suggest starting with logistic regression on a subset of features, but you should expect to build up the model complexity and number of features.
You will probably want to use most of the techniques you have learnt in the past two/three lectures:

- Some basic EDA of this new dataset
- Ensuring the data are fully prepared
- Exploring different classification methods
- Searching for optimal parameters
- Addressing class imbalances
- Testing other performance improvements (ensemble methods)
- Using effective model validation

Before you start, what metric/s should we use?

WRAP UP

- You now have lots of tools to address supervised machine learning problems; however, experience and reading up on the nuances of each method is the difference between using them well
- REMEMBER… if your model does not have the key features of the system/disease/procedure/etc. you are trying to predict, no matter how complex you make your model or how many samples you collect, it will never improve its performance



Issitt R W, Cortina-Borja M, Bryant W, et al. (February 21, 2022) Classification Performance of Neural Networks Versus Logistic Regression Models: Evidence From Healthcare

Practice. Cureus 14(2): e22443. doi:10.7759/cureus.2244

BEFORE NEXT SESSION

- Review background on neural networks
- Complete the tutorial exercises
- Review random forest and gradient boosting methods

NEW MATERIAL

- Random Forest - **https://williamkoehrsen.medium.com/random-forest-simple-explanation-377895a60d2d**
- xgboost Introduction - **https://xgboost.readthedocs.io/en/stable/tutorials/model.html**
- OPTIONAL Tutorial on CNN for image classification - **https://www.tensorflow.org/tutorials/images/cnn**
- OPTIONAL Very detailed description of the DeepMind AlphaFold 3, so you can understand the complexity possible in Deep Neural Networks - **https://elanapearl.github.io/blog/2024/the-illustrated-alphafold/**

CONSOLIDATION READING

- Excellent interactive Google tutorial that lets you visually explore neural networks - **https://developers.google.com/machine-learning/crash-course/neural-networks**