The Mathemmatical Engineering of Deep Learning

Chapter 6 - Lecture 6

B. Liquet[1,2] and S. Moka[3] and Y. Nazarathy[3]

[1] Macquarie University [2] LMAP, Université de Pau et des Pays de L'Adour' [3] The University of Queensland

Outline of Lecture

- Hyper-parameters
- Bayesian Optimization
- Transfer Learning
- Data augmentation
- Imbalanced case

Hyper-parameter in Deep Learning

- Performance of Deep learning is depending on meta-parameters that have to be tuned with care
- These parameters are known as hyper-parameters or system parameters and are tuned by human experts.

Hyper-parameter in Deep Learning

Hyper-parameters in Deep Learning are crucial for defining your model and to control the success of the training process of the defined model. Two group of hyper-parameters:

- Hyper-parameters for controlling the Optimization process: **Optimizer hyper-parameters**
- Hyper-parameters for defining the model: **Model Specific hyper-parameters**

Model Hyperparameters

Parameters related to the architecture:

- Number of hidden units: related to the *capacity* to learn any function.
- Number of layers: Increasing the number of layer for shallow network improves generally the performance
- Activation functions: **ReLu** activation function is becoming the most used as is less computationally expensive than **tanh** and **sigmoid**.
  - For classification task: Question for you ?
  - For regression task: Question for you ?
  - Drawback of ReLu function : Question for you ?

Some Heuristics

"Heuristics 1: Simple first"

“A popular heuristic is to incrementally build a more complex model. This means for example to try first a model with only one or two hidden layers and expand the network if the simple model fails.”

"Heuristics 2: Increases number of neurons first"

“A second heuristic is to increase first the number of hidden neurons before trying to increase the number of hidden layers when your model perform poorly. Indeed it is less computational expensive to doubling the size of a hidden layer than doubling the number of hidden layer.”

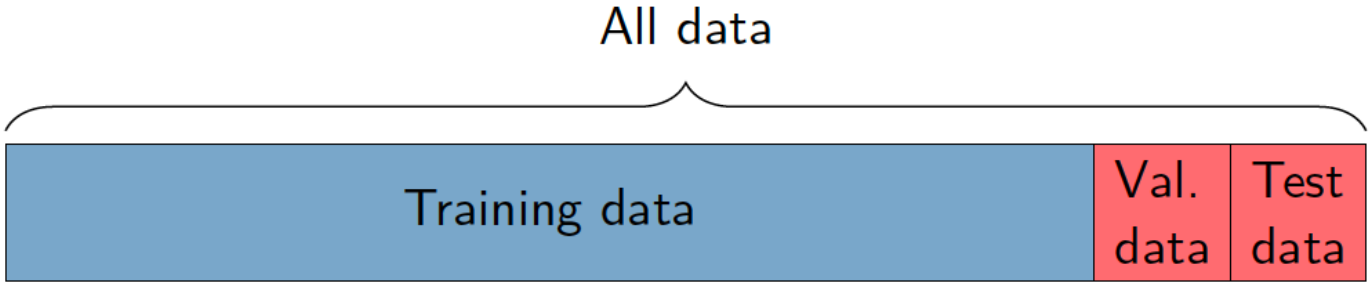
Optimizer hyper-parameters

Parameters related to the *optimization process*

- Learning rate
- Mini-Batch size
- Number of Epochs
- weight decay
- and more . . .

Reminder: Split Data for Neural Network

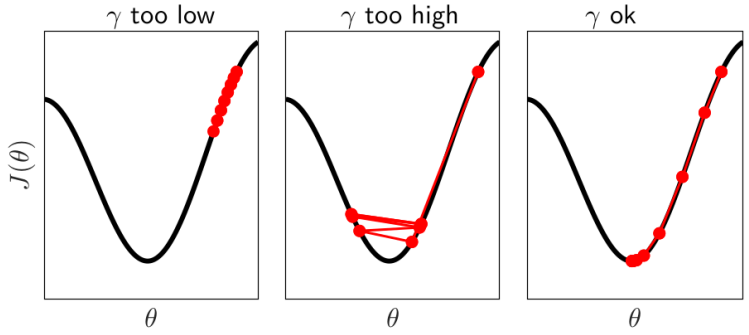
How the data are split for Neural Network ?



- **Training Data**: used for Training models
- **Validation Data**: used for **optimizing hyperparameters**, choosing between models
- **Test Data**: for evaluating the performance of the final model

IMPORTANT: Validation data and test data should come from the same distribution

Calibration of the learning rate



- not recommended to use a constant learning rate.
- learning rate to decay over time:
  - **exponential decay**: $\alpha_t = \alpha_0 \exp(-k \times t)$
  - **Inverse decay**: $\alpha_t = \frac{\alpha_0}{1+k \times t}$

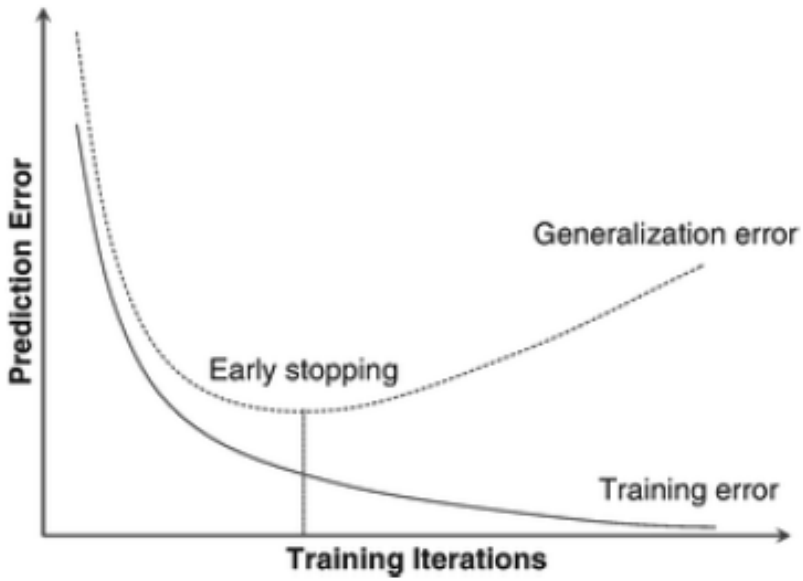
    where α_0 initial learning rate, k controls the rate of the decay which will decrease at each epoch t.
- Reduce the learning rate by some factor every few epochs: **Step decay**.
- Half the learning rate every 5 epochs, or by 0.1 every 20 epochs.

Mini-Batch Size

- Mini-Batch size of 1 sample corresponds to the **stochastic training**
- Mini Batch size of the entire data is the **batch training**.
- Popular value of the mini batch size is 32.
- It is also well recommended to try subsequent values: 1, 2, 4, 8, 16, 32, 64, 128, 256.

Number of Epochs.

- Choice of the number of epochs is driven by the result of the **Validation Error**.
- Train the model as long as the validation errors keeps decreasing.
- Early stopping: stopping training early since overfitting typically increases as training progresses.



- **keras** offers patience parameter. Interrupts training when accuracy has stopped improving for more than k epochs.

Regularized tuning parameter.

- L_2 regularization is the most exploited technique for preventing overfitting.
- What is the name of this approach ?

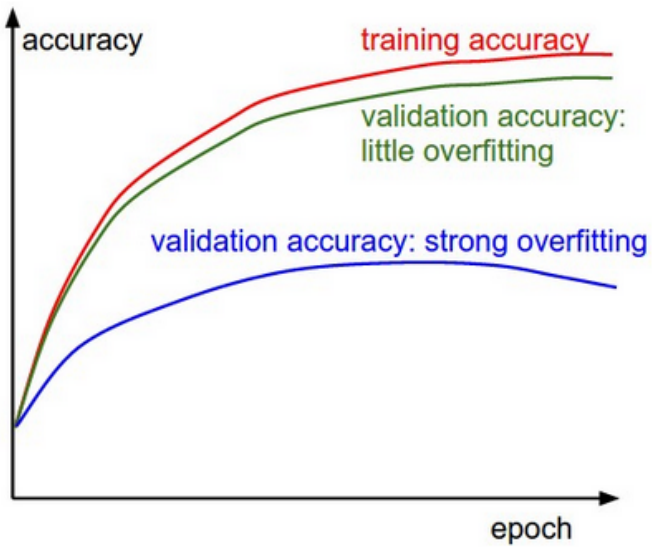
$$L(\mathbf{w}, b) + \frac{\lambda}{2}||\mathbf{w}||^2,$$

where λ is called the **weight regularization hyperparameter**.

- Small values are generally tried first for controlling the contribution of each weight to the penalty.

Regularized tuning parameter: which values to use ?

- Still reasonable to use the same weight decay at all layers.
- The best value should remain constant throughout the training.
- Consider for this hyperparameter a grid search strategy
- Possible classical choices: 10^{-3}, 10^{-4}, 10^{-5}, and 0.
- **From experiments:** smaller datasets and architectures seem to require larger values for weight decay while larger datasets and deeper architectures seem to require smaller values.



Hyper-parameters Optimization procedures

Optimizer hyper-parameters: Learning rate, Mini-Batch size, Number of Epochs, weight decay, and more . . .

The three main techniques for finding optimal values:

- Grid Search
- Random Search
- Bayesian Optimization

Two populars approaches

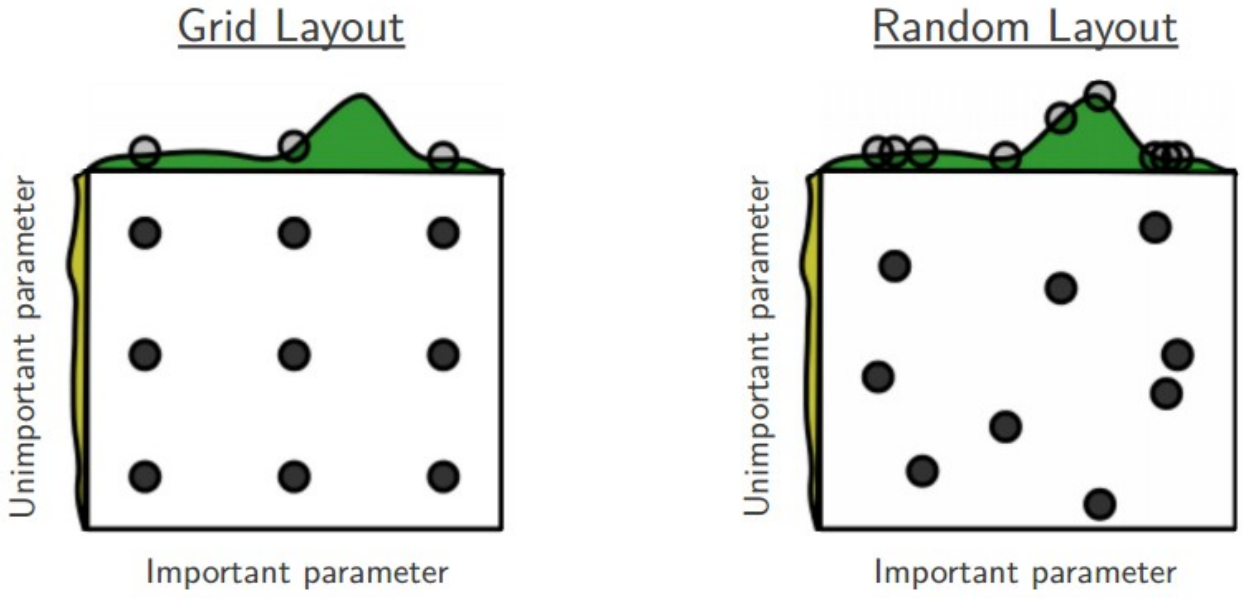


Illustration from [Bergstra and Bengio, 2012]

- Exhaustive search on a regular or random grid
- Complexity is exponential in the number of hyper-parameters
- Easy to parallelise
- Bergstra and Bengio in Random Search for Hyper-Parameter Optimization mentioned that *"randomly chosen trials are more efficient for hyper-parameter optimization than trials on a grid"*

Bayesian Optimisation

- An alternative which is becoming widely used is the Bayesian Optimization techniques.
- Main idea of using a **Bayesian approach** is to pay attention to **past results** for exploring a **better region**.
- The actual function $f(\theta)$ we are trying to optimize (function of hyper-parameters) is really complicated.
    - Cost function: $J(w, b; \theta)$ where here θ includes all the hyper-parameters

Bayesian Optimization: Methodology

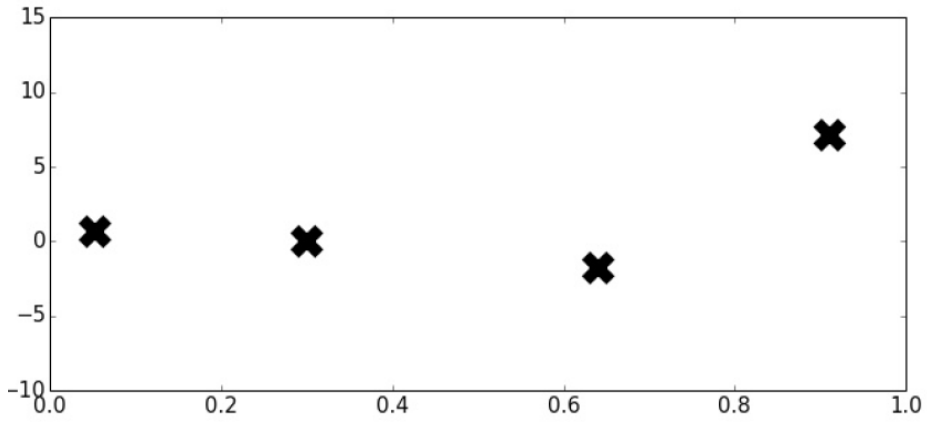
Let consider $f : \mathcal{X} \rightarrow \mathfrak{R}$ where $\mathcal{X} \subseteq \mathfrak{R}^d$ a bounded domain.

We want to solve this optimization problem

$$x^* = \arg\min_{x \in \mathcal{X}} f(x)$$

- **Input** x a configuration of hyper-parameters
- **Function value** $f(x)$: error on the validation set
- Each evaluation is **expensive**

Where would you try next?



(Image credit: Javier González)

- Builds a **probabilistic model** (called **surrogate model**) of the objective function:
  - Optimises a “proxy” instead the objective
  - Models the uncertainty **Gaussian Process**
- Combine prior and the likelihood to get a **posterior measure** given some observation
- Use the posterior to decide where to take the next evaluation according to acquisitions functions

Pobabilistic surrogate model

Popular surrogate model: Gaussian process

- allows to predict f
- quantify our uncertainty in prediction using probability distribution

A Gaussian process is a distribution over functions.

- For any set of points $\{x^{(1)}, \ldots, x^{(m)}\}$, the functions evaluation $\{y_1, \ldots, y_m\}$ are distributed

$$\begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \sim \text{Norm}\left(\begin{bmatrix} m(x^{(1)}) \\ \vdots \\ m(x^{(1)}) \end{bmatrix}, \begin{bmatrix} k(x^{(1)}, x^{(1)}) & \ldots & k(x^{(1)}, x^{(m)}) \\ \vdots & \ddots & \vdots \\ k(x^{(m)}, x^{(1)}) & \ldots & k(x^{(m)}, x^{(m)}) \end{bmatrix}\right)$$

$m(x) = E(f(x))$ is the mean function and $k(x, x^{'})$ is the covariance function.

Gaussian Process

A popular choice: exponentiated quadratic kernel (with $\ell = 1$ and $\sigma = 1$)

$$k[\theta, \theta'] = \sigma^2 \exp\left[-\frac{1}{2\ell^2}(\theta - \theta')^T(\theta - \theta')\right].$$

$\longrightarrow$ a smooth prior on functions sampled from the Gaussian process.

Prediction

Goal: make prediction about the function value at a new point x^* given some observations of f at m points $\mathbf{f} = (f[\mathbf{x}^{(1)}], f[\mathbf{x}^2], \ldots, f[\mathbf{x}^m])$.

- Property of the GP: this new function value $f^* = f(\mathbf{x}^*)$ is jointly normally distributed with the observations $\mathbf{f}$:

$$Pr\left(\begin{bmatrix}\mathbf{f}\\ f^*\end{bmatrix}\right) = \text{Norm}\left(\begin{bmatrix}m(\mathbf{X})\\ m(x^*)\end{bmatrix}, \begin{bmatrix}\mathbf{K}[\mathbf{X}, x^*] & \mathbf{K}[\mathbf{X}, x^*]\\ \mathbf{K}[x^*, \mathbf{X}] & \mathbf{K}[x^*, x^*]\end{bmatrix}\right),$$

Why is usefull ?

- Can derive the distribution of $Pr(f^*|f)$
- Get the distribution of the function at any new point x^*

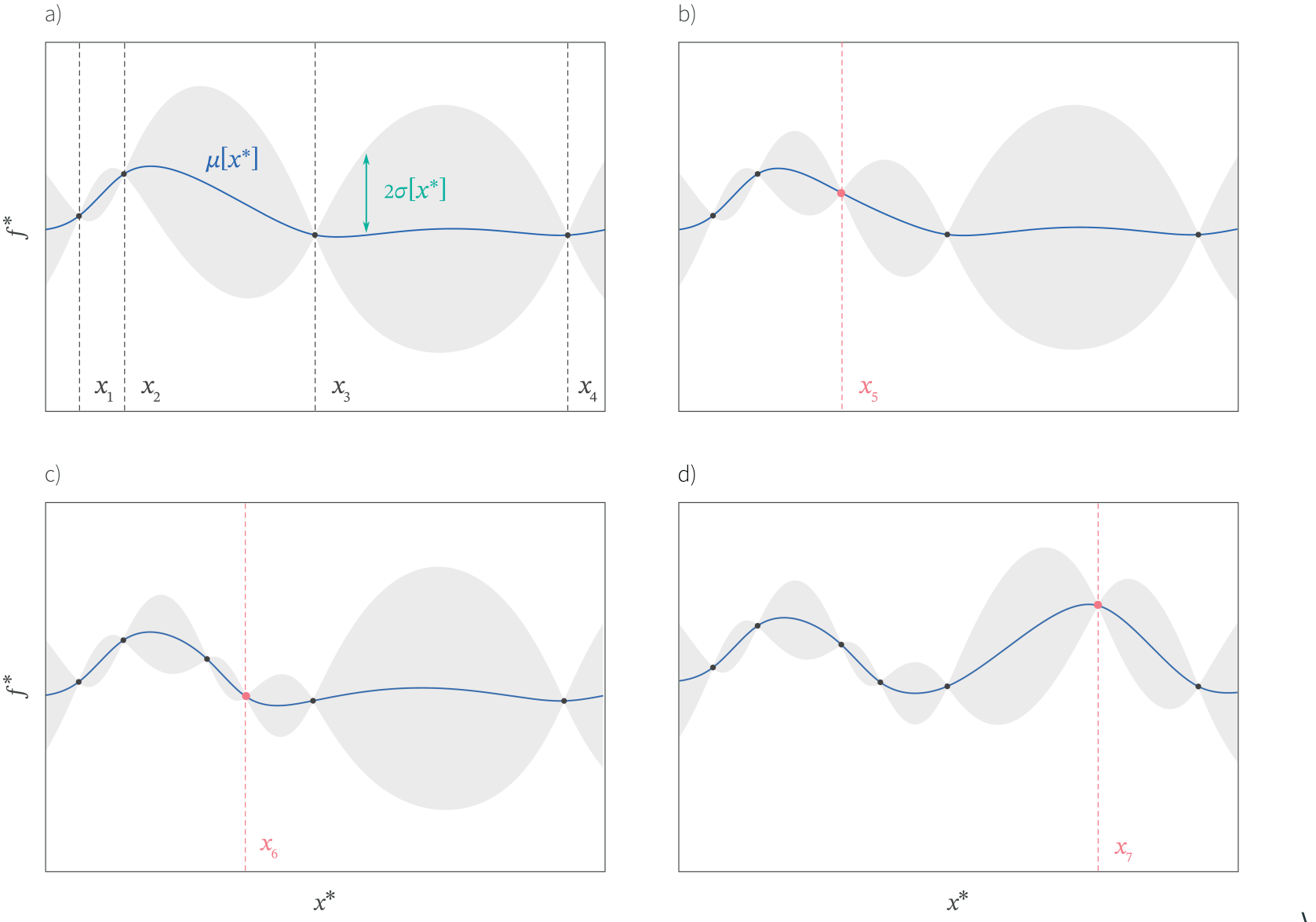


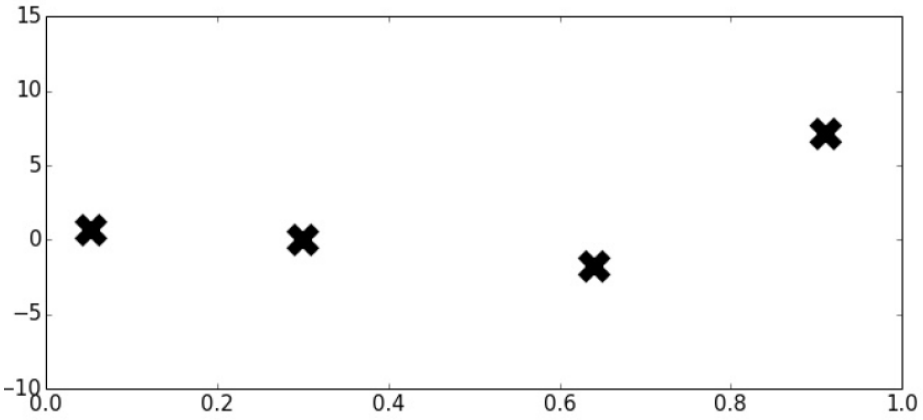
Illustration from Tutorial on Bayesian optimization

Let play with Gaussian Processes here

Features of Gaussian Process

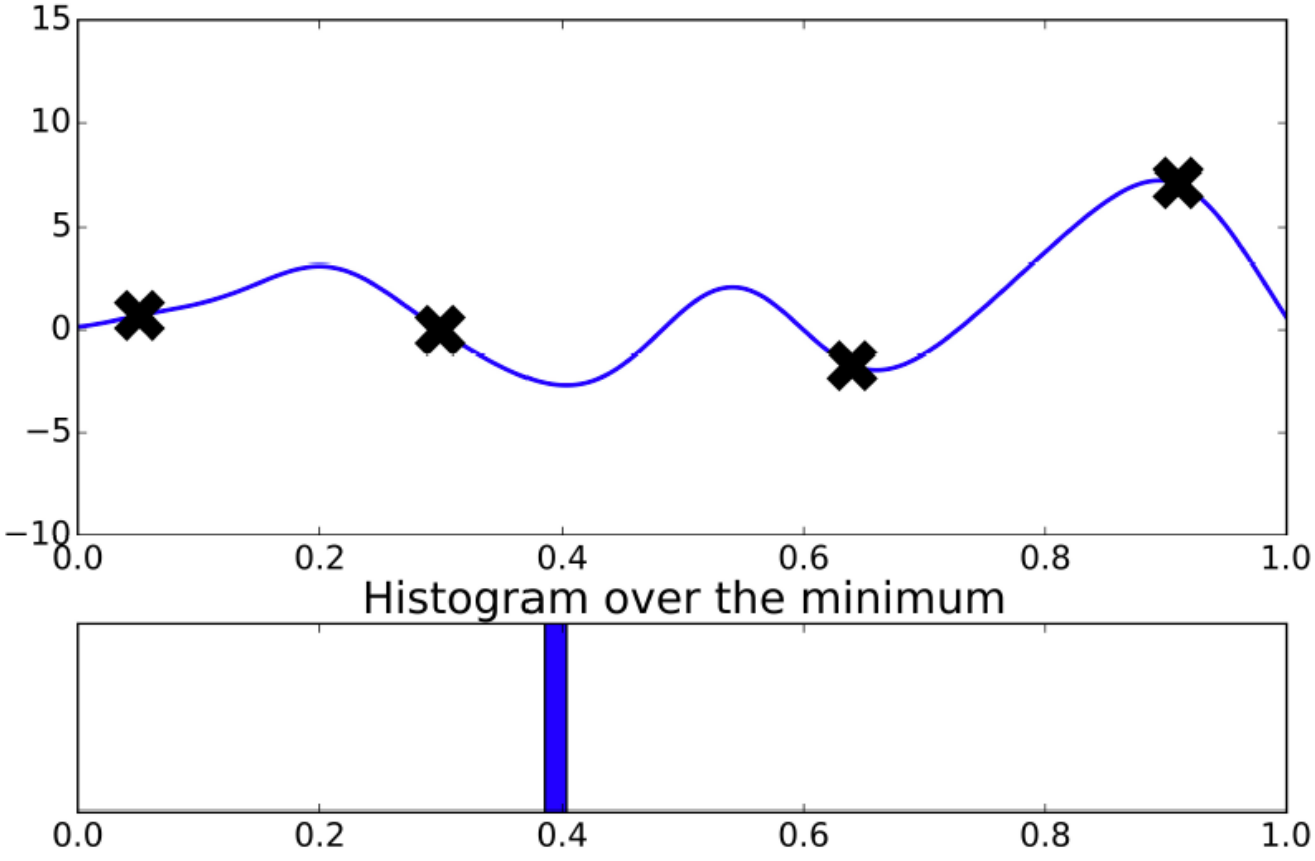
- Gaussian processes are probability distributions over **functions**.
- The choice of kernel affects the smoothness of the functions sampled from a Gaussian process.
- The **multivariate normal distribution** has analytic conditional and marginal distributions.
- We can compute the mean and standard deviation of our prediction of an objective function at a particular design point given a set of past evaluations.
- We can fit the parameters of a Gaussian process using **maximum likelihood**

Where would you try next?



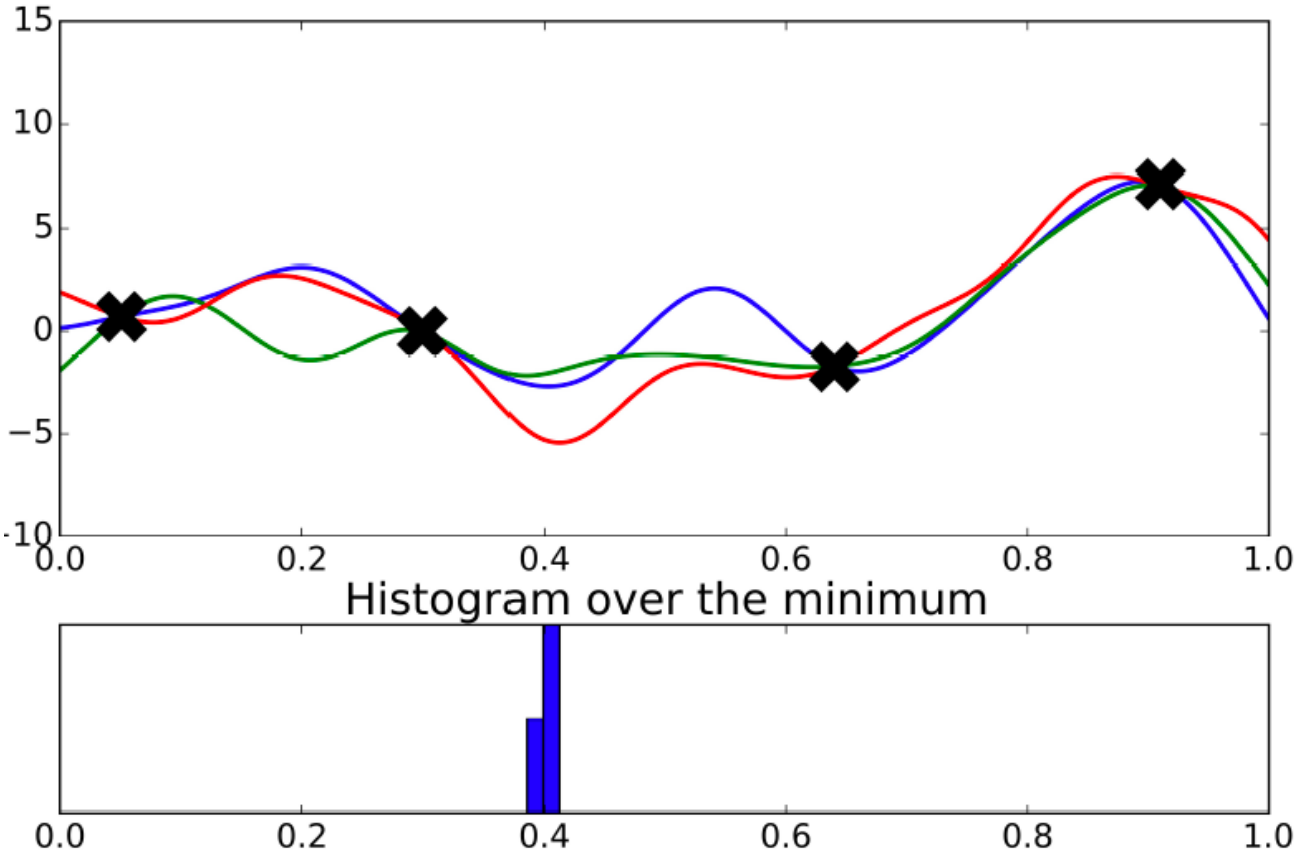
(Image credit: Javier González)

Bayesian Optimization: Intuition



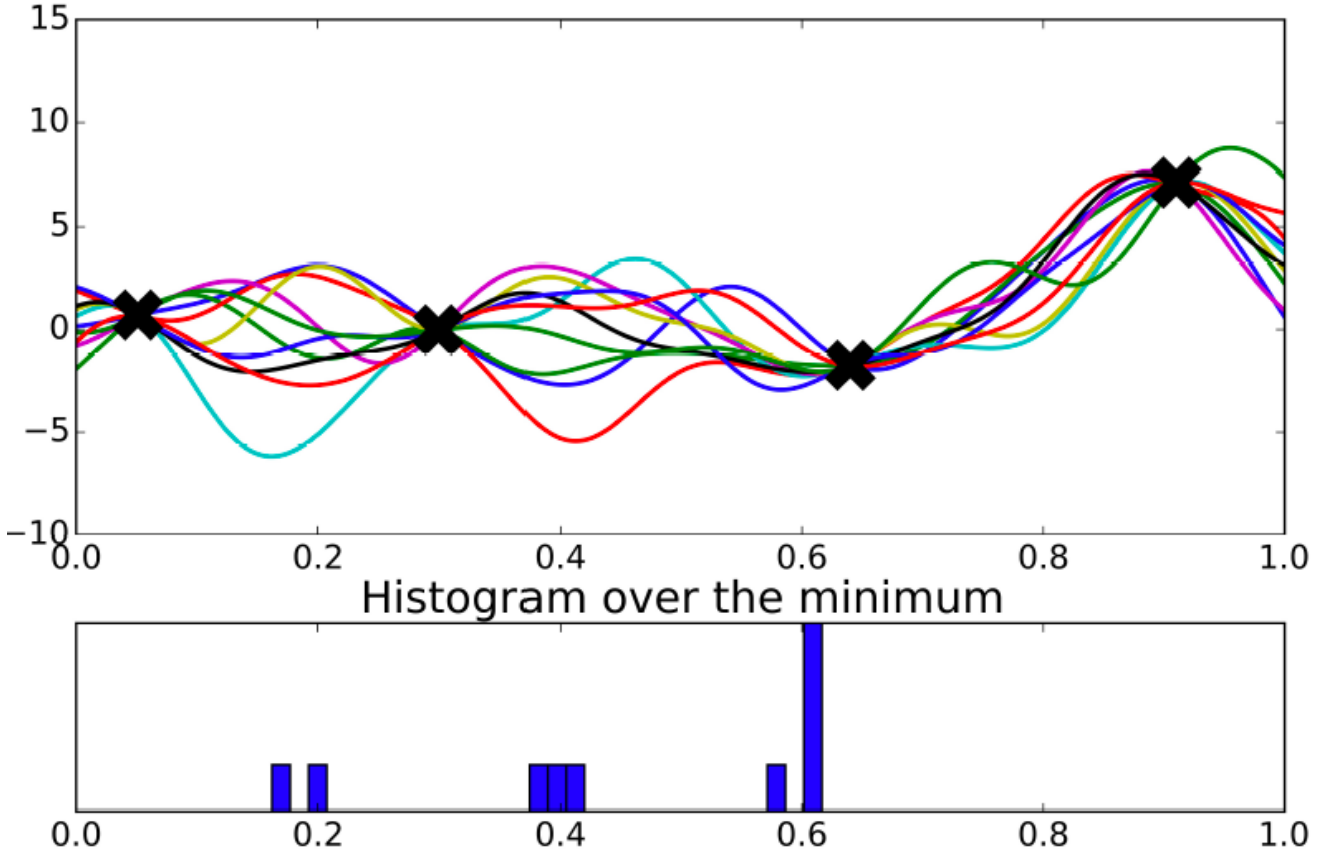
(Image credit: Javier González)

Bayesian Optimization: Intuition



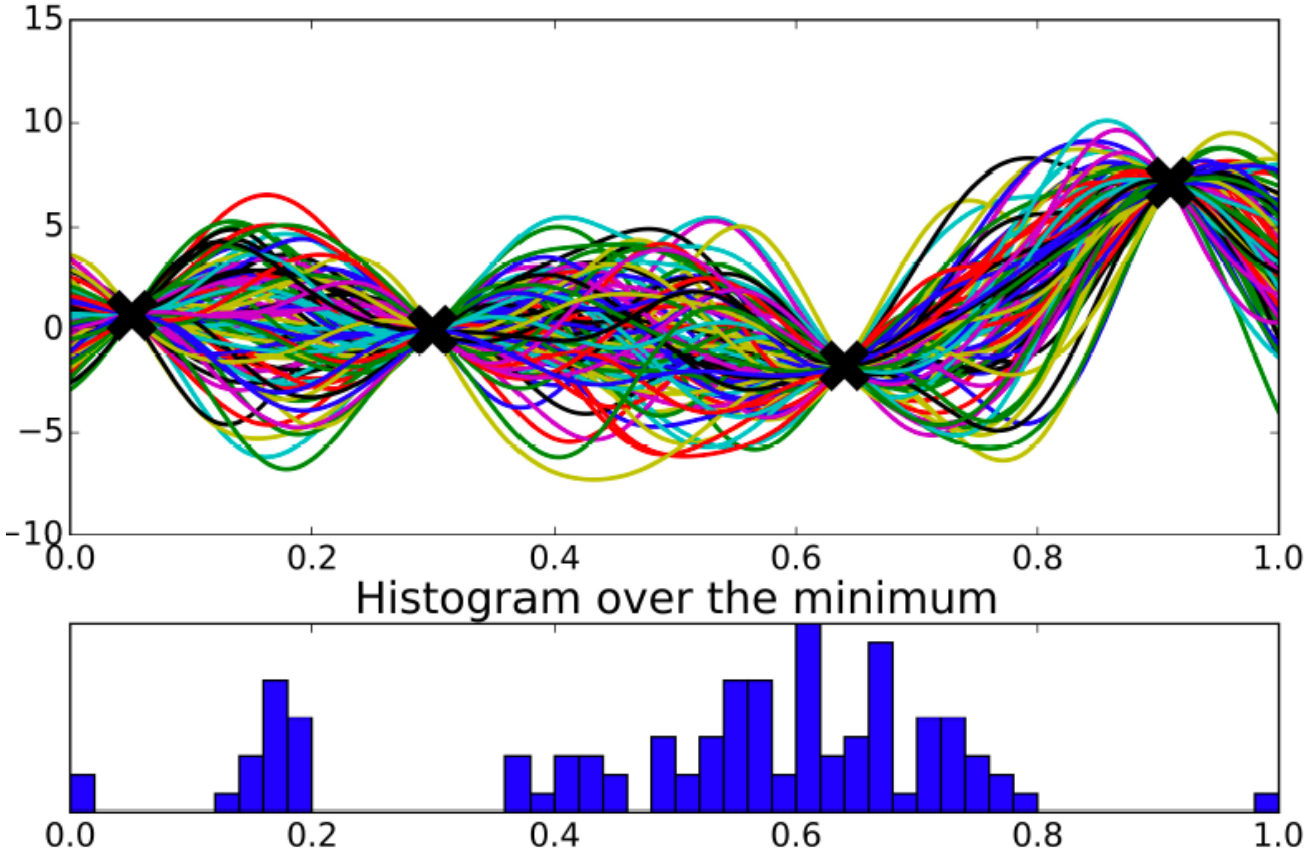
(Image credit: Javier González)

Bayesian Optimization: Intuition



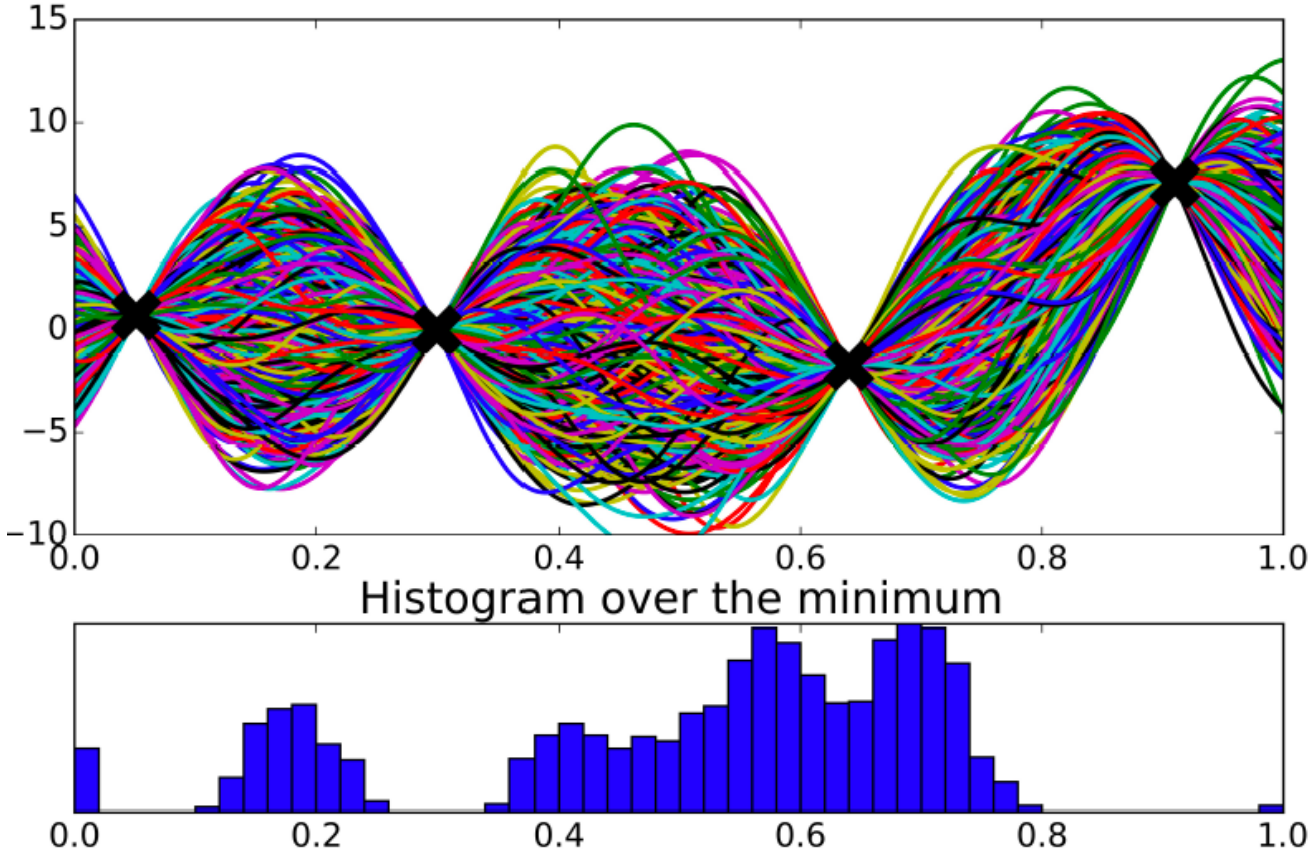
(Image credit: Javier González)

Bayesian Optimization: Intuition



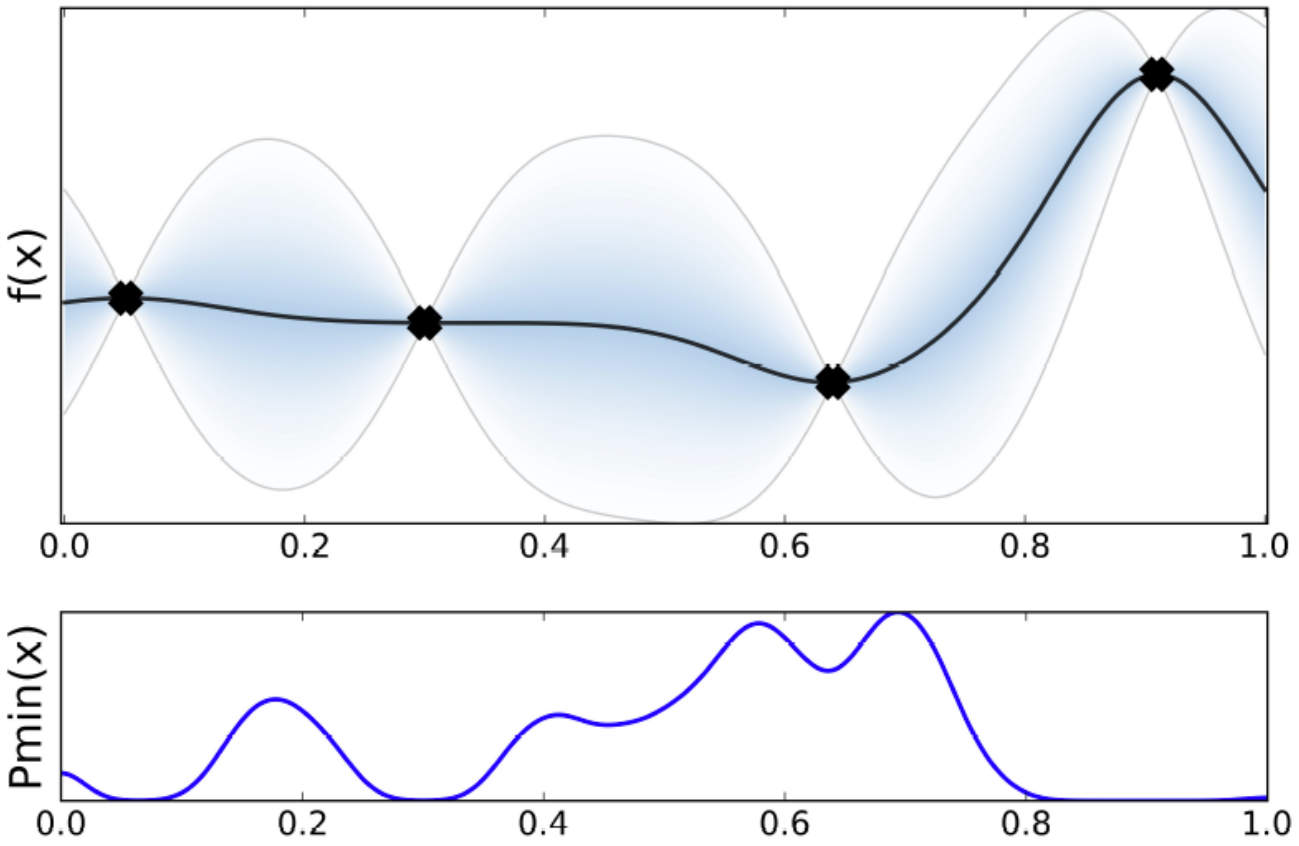
(Image credit: Javier González)

Bayesian Optimization: Intuition



(Image credit: Javier González)

Bayesian Optimization: Intuition



(Image credit: Javier González)

Bayesian Optimization: Acquisition function

How to choose the next point ?

- acquisition function indicates how promising a new candidate it is.
- acquisition function balance the Exploration and Exploitation factors to determine where to evaluate next.
- Exploration function: Evaluate in places where the variance is large
- Exploitation function: Evaluate in place where the mean is low.

Bayesian Optimization: Acquisition function

Procedure:

- Evaluate all candidates according to an **acquisition function**
- Rank them and pick the best one

Popular acquisition functions:

- Maximum Probability of Improvement (MPI)
- Expected Improvement (EI)
- Upper Confidence Bound (UCB)
- Thompson sampling, entropy search, ...

Acquisition function: Expected Improvement

Expected Improvement (EI) which is the popular one:

$$EI(\theta) = E[max\{0, f(\widehat{\theta}) - f(\theta)\}],$$

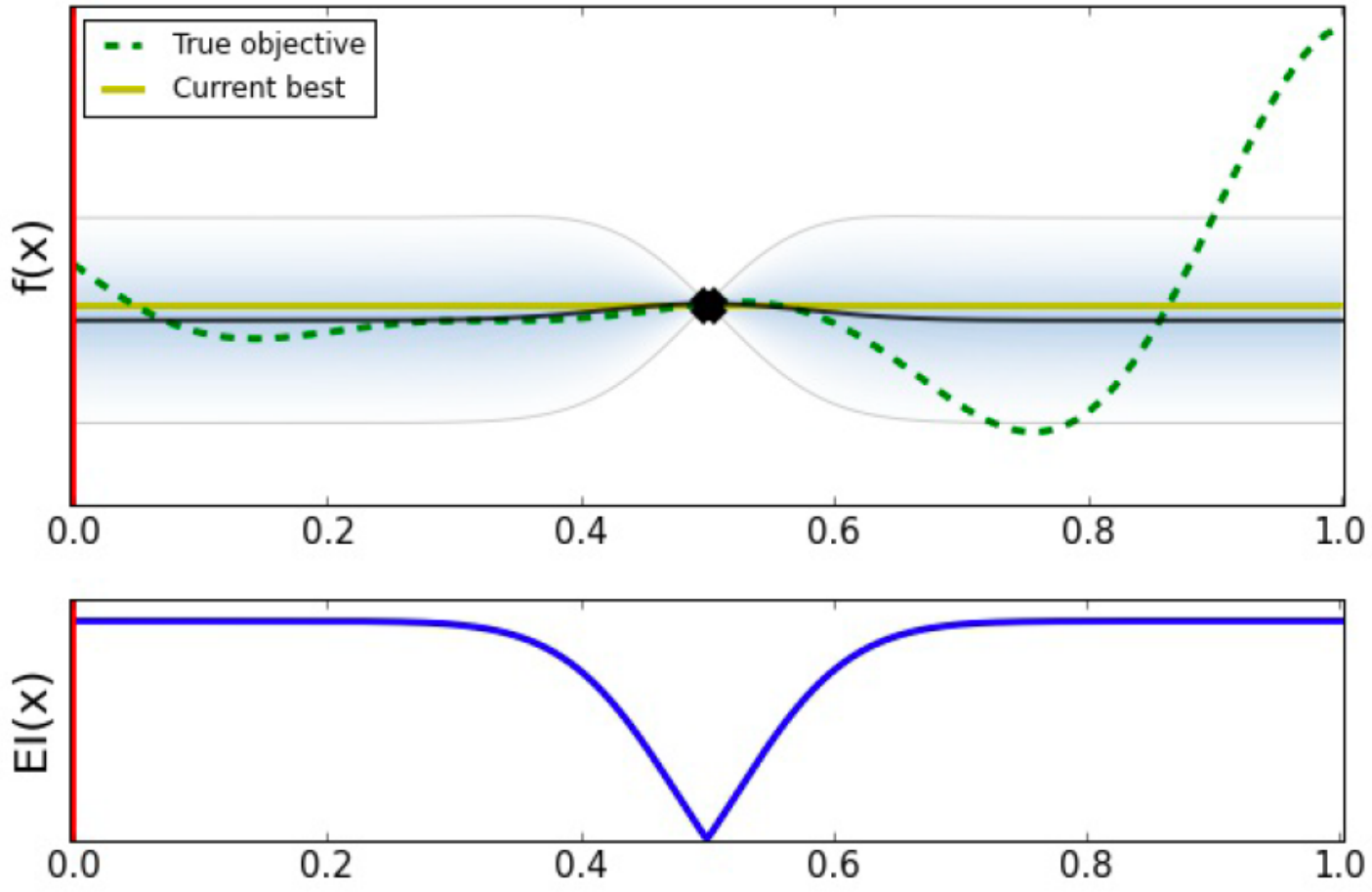
where $f(\widehat{\theta})$ is the current optimal set of hyper-parameters.

- *"if the new value is much better, we win by a lot; if it's much worse, we haven't lost anything"*

- There is an explicit formula for **Expected Improvement (EI)** under the **Gaussian Process model**.

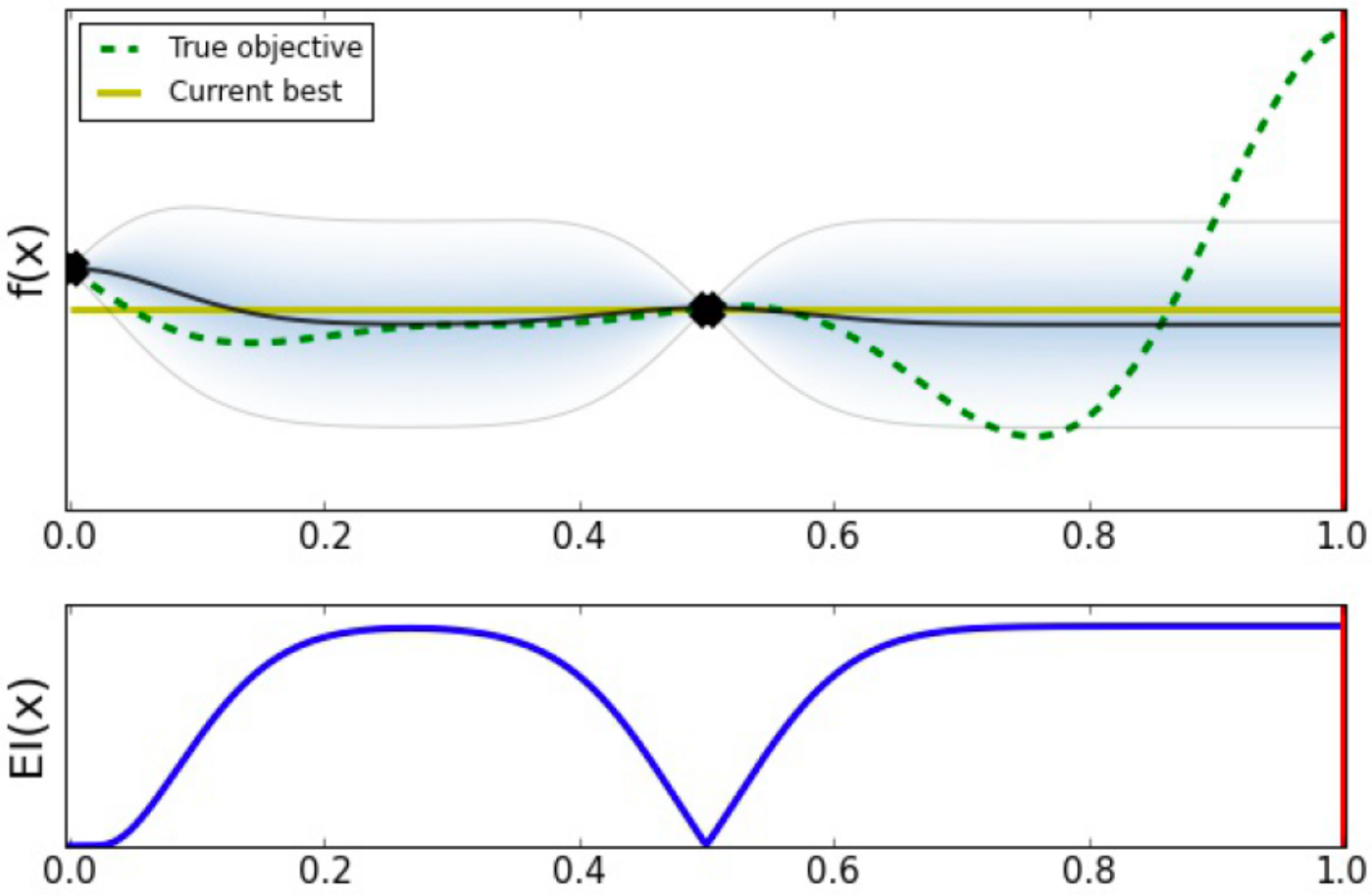
Try to derive it !!!

Bayesian Optimization: Intuition



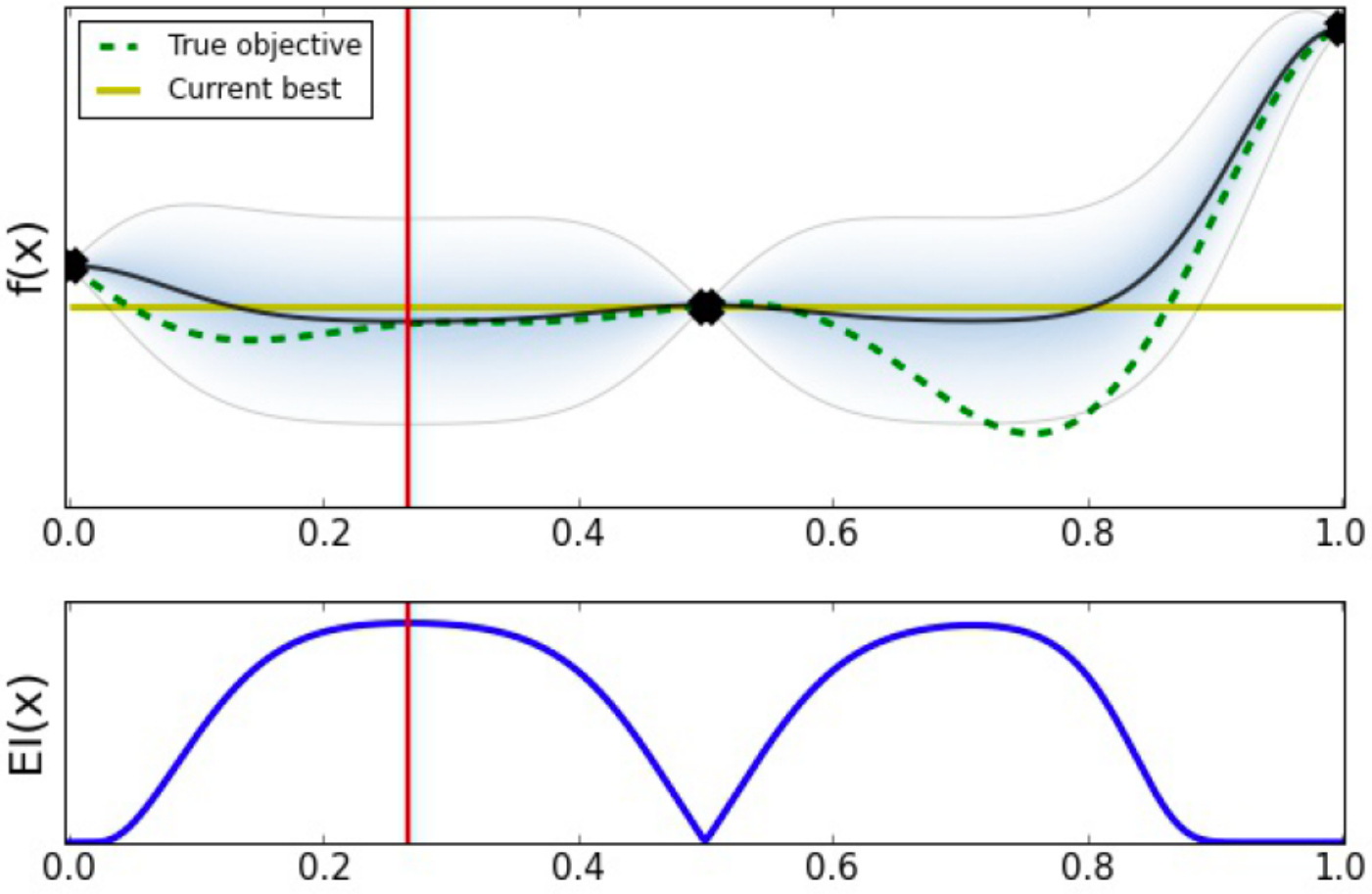
(Image credit: Javier González)

Bayesian Optimization: Intuition



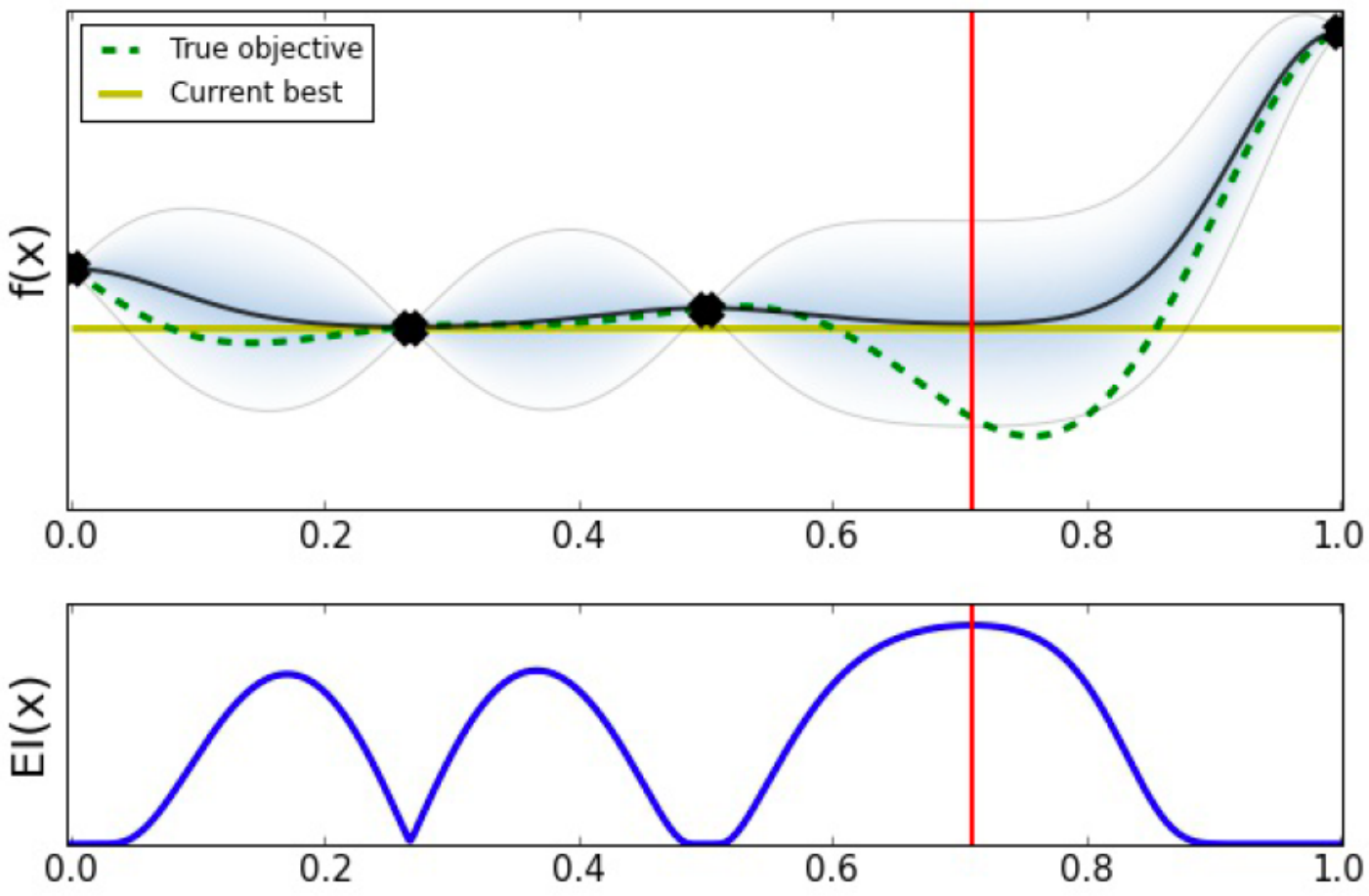
(Image credit: Javier González)

Bayesian Optimization: Intuition



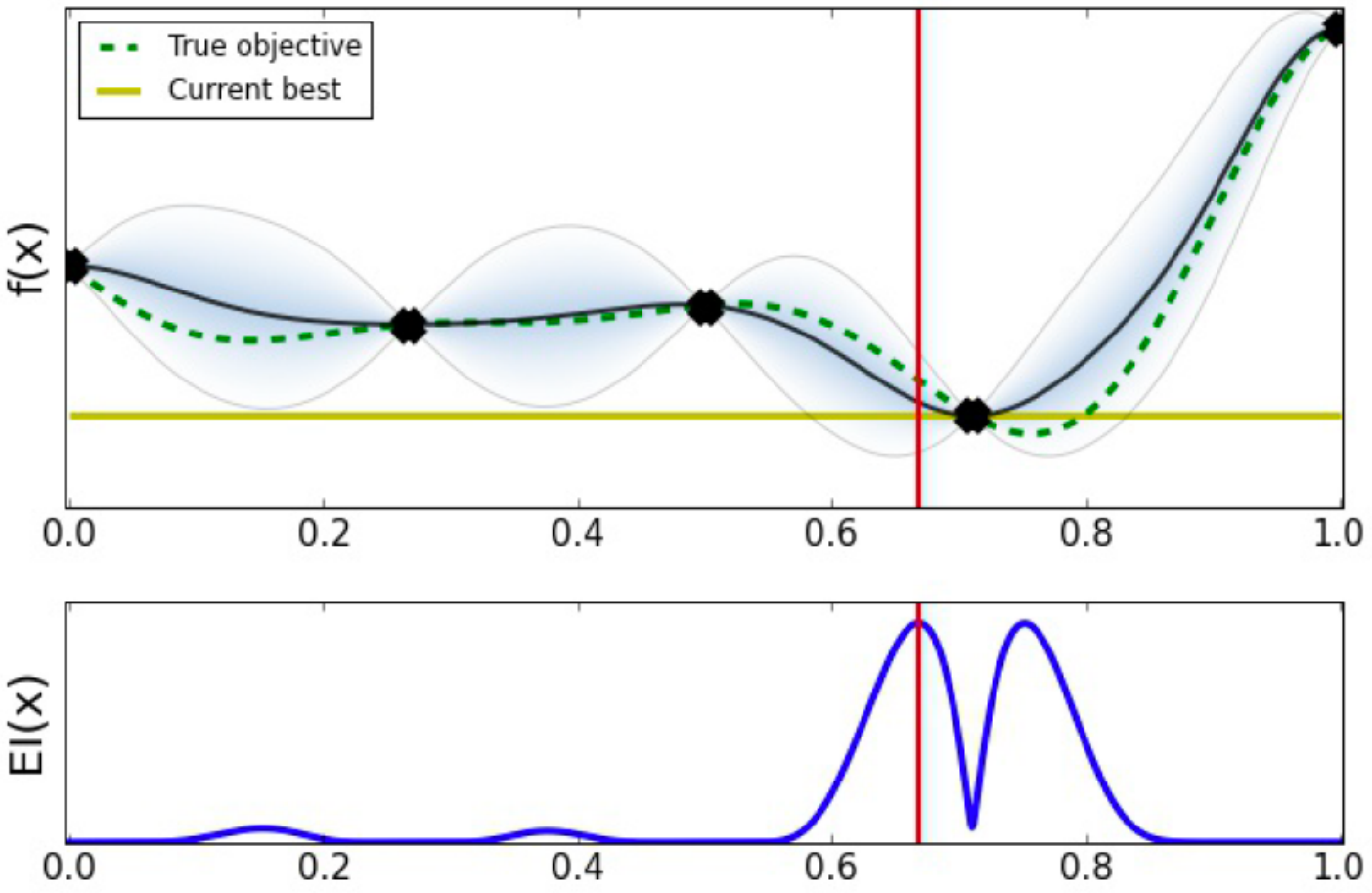
(Image credit: Javier González)

Bayesian Optimization: Intuition



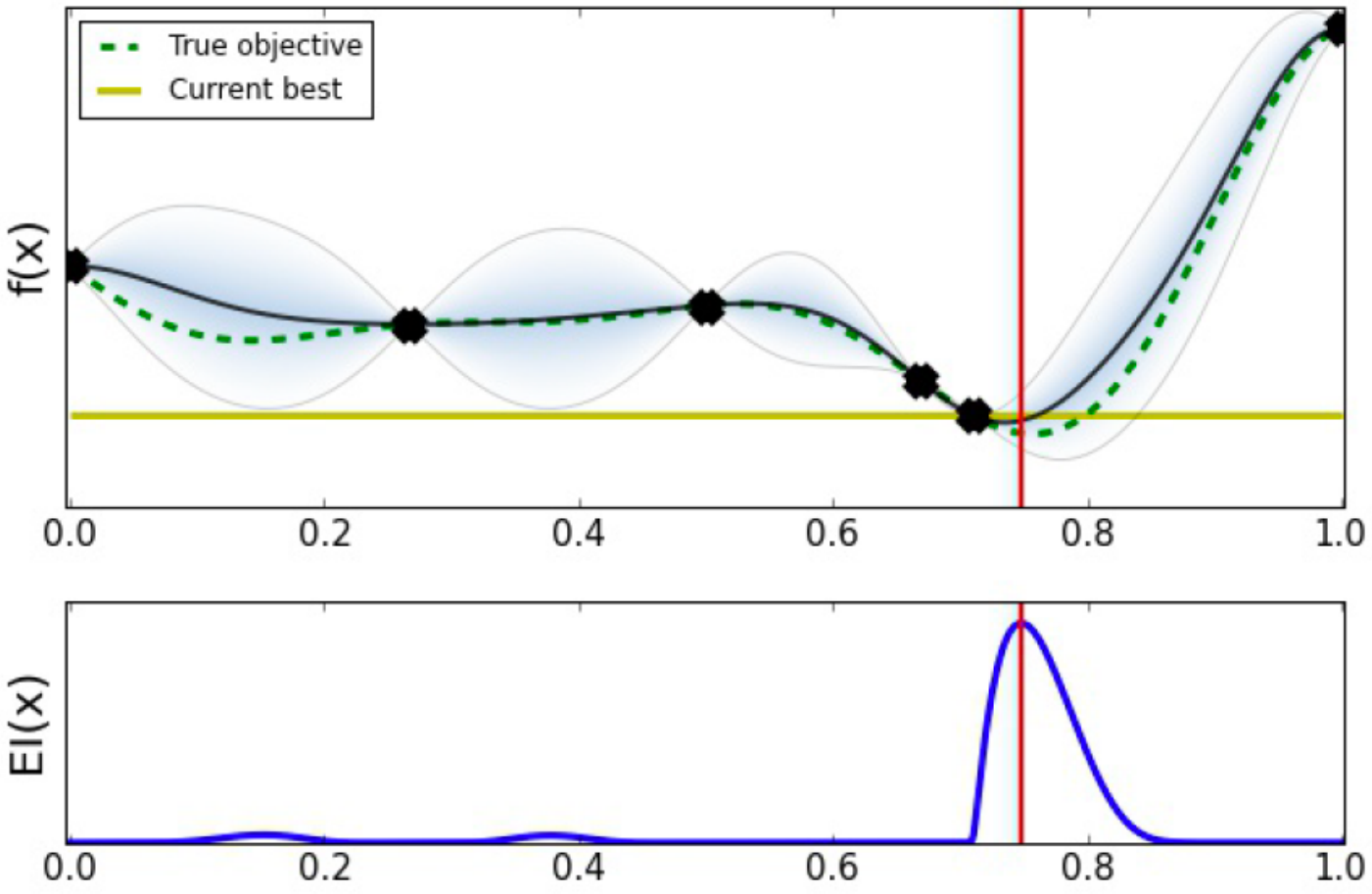
(Image credit: Javier González)

Bayesian Optimization: Intuition



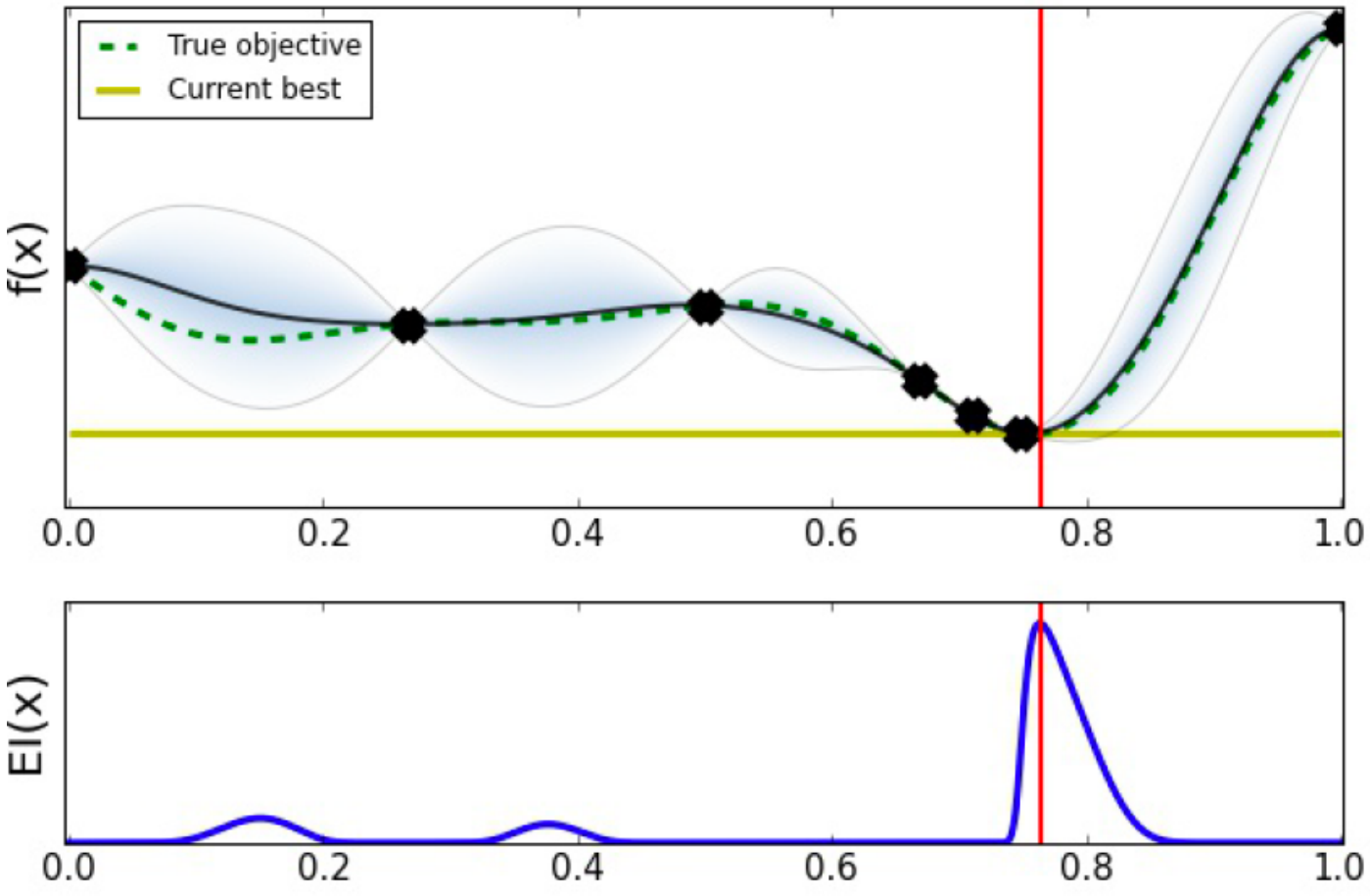
(Image credit: Javier González)

Bayesian Optimization: Intuition



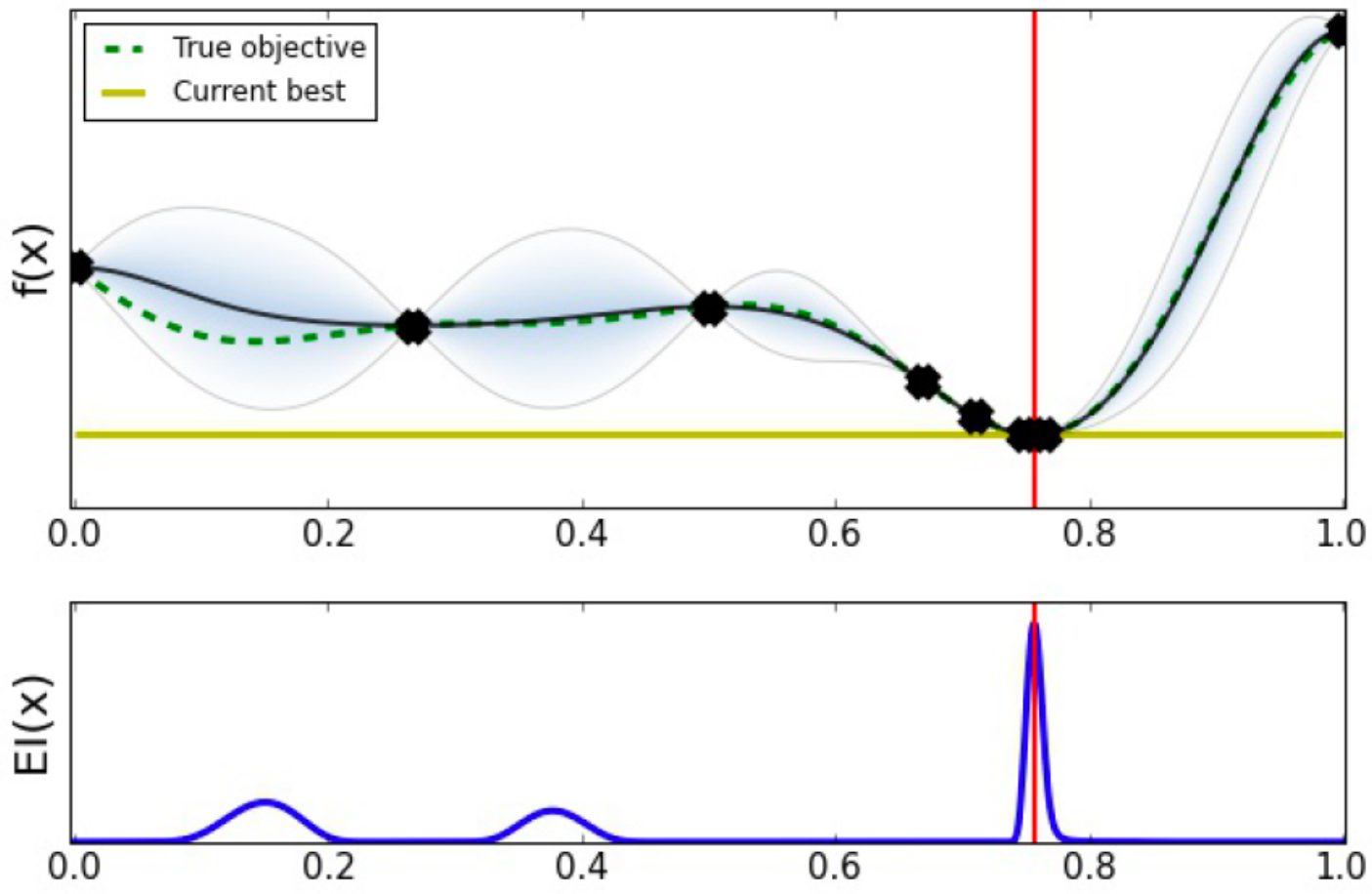
(Image credit: Javier González)

Bayesian Optimization: Intuition



(Image credit: Javier González)

Bayesian Optimization: Intuition



(Image credit: Javier González)

Bayesian Optimization: summary

The main steps of the algorithm:

- **Surrogate model** of f which is cheaper to evaluate
- Set of evaluated candidates.

For $t = 1, 2, \ldots$ repeat:

1. Find the next sampling point θ_t by optimizing the acquisition function over the GP:

   $$\theta_t = \text{argmax}_\theta \, EI(\theta | \mathcal{D}_{1:t-1}),$$

   where $\mathcal{D}_{1:t-1} = (\theta_1, y_1), \ldots, (\theta_{t-1}, y_{t-1})$

2. Compute exact loss

   $$y_t = f(\theta_t)$$

   from the objective function f.

3. Add the sample to previous samples $\mathcal{D}_{1:t} = \mathcal{D}_{1:t-1}, (\theta_t, y_t)$.

4. Update the GP.

Conclusion

- Empirically **Bayesian Optimization** has been demonstrated to get better results using in fewer experiments, compared with grid search and random search.
- You can explore:

1. tutorial on Bayesian Optimization
2. tutorial on Bayesian Optimization in R
3. video from Javier Gonzalez
4. R package **GPfit** for fitting a gaussian process (JSS 2015 paper, Vol 64)

Multitask learning

Multitask learning: train a network for solving multiple tasks.

- Different tasks benefit from having shared lower-level features
- Can be seen as a regularization (in comparison to training the task separately)

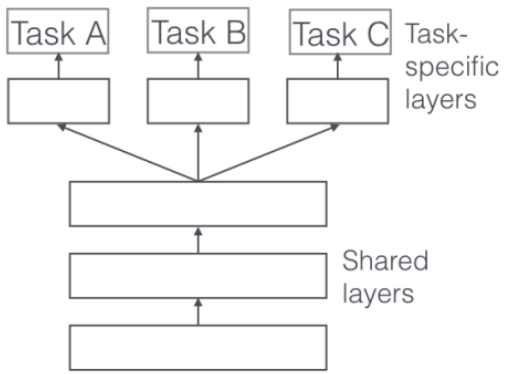
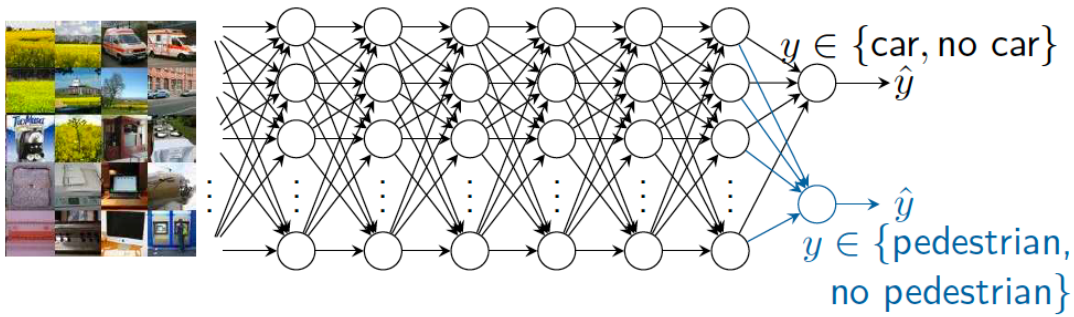


Figure from An Overview of Multi-Task Learning in Deep Neural Networks

Multitask learning

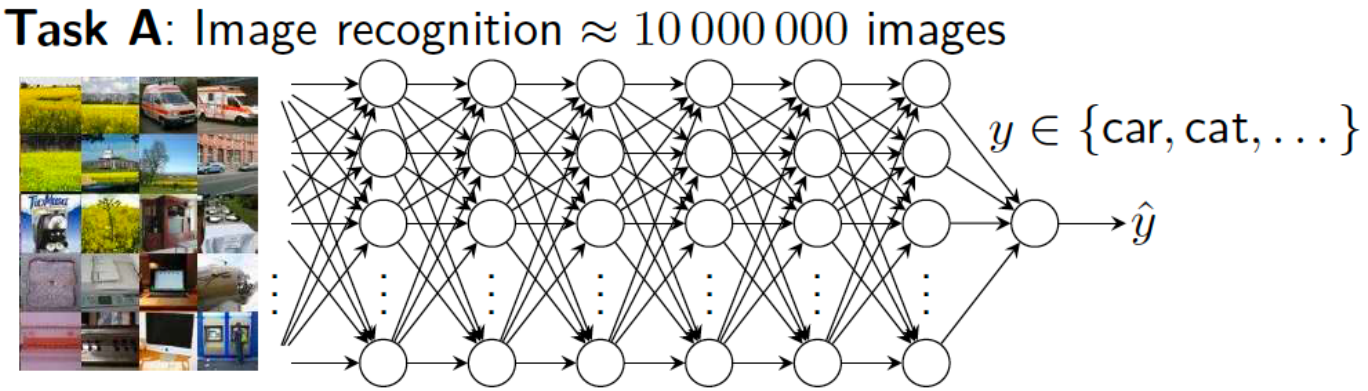
Multi-task learning: Solving multiple learning tasks at the same time (usually by defining a multitask loss), while exploiting commonalities and differences across tasks



- Example: Segmentation, object detection, ... see an overview multi-task learning

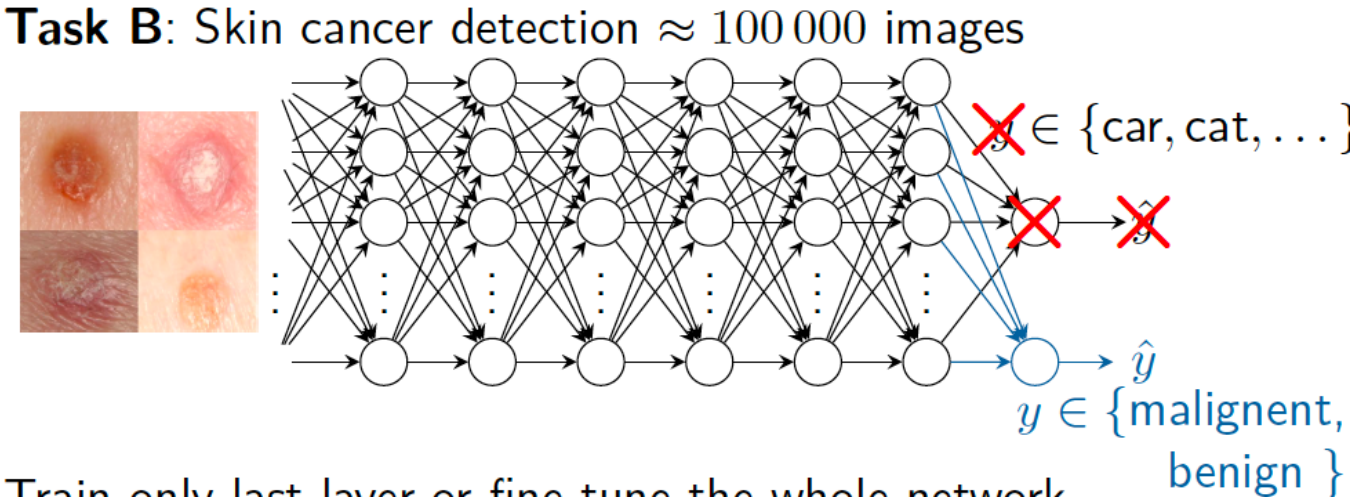
Transfer learning

transfer learning: reuse models that have been trained on a different task than the one we are interested in



Transfer learning

transfer learning: reuse models that have been trained on a different task than the one we are interested in

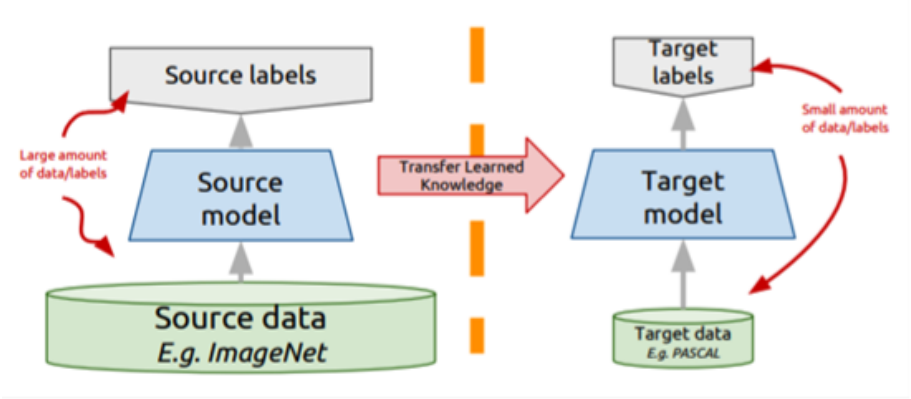


- Train only last layer or fine-tune the whole network.
- Low level features from Task A helpful for solving Task B

Transfer learning applicable when: - Task A and Task B have same input x - We have a lot more data for task A than for task B

Transfer learning: Main points

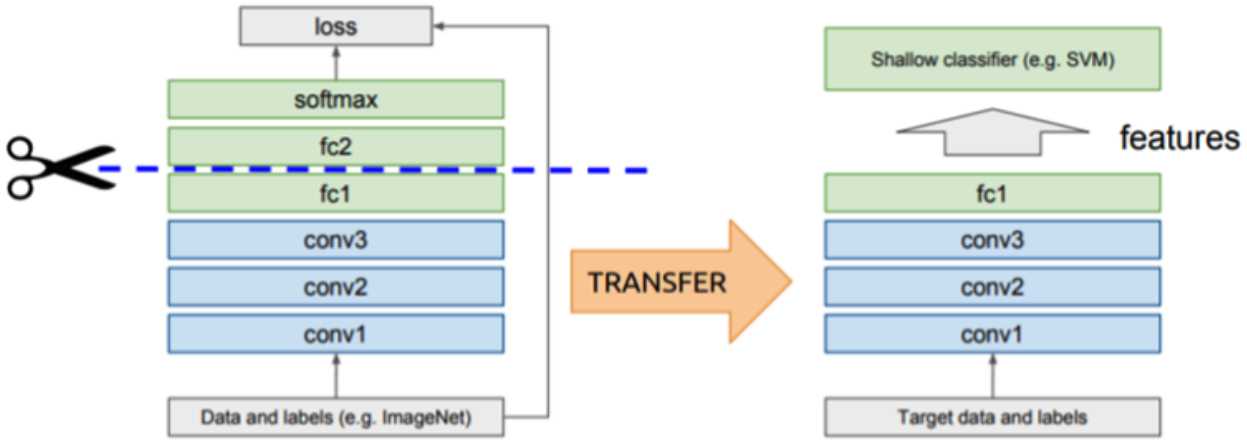
- Pick up a model “off the shelf” → adapt it to your target task.
- Take a model that has been already trained for one task (e.g., for classiying object) and then **fine-tune** it to accomplish another but relevant task.



- Largely exploited in computer vision-related tasks:
  - features learned from very large image sets, such as the **ImageNet**, are highly transferable to a variety of image recognition tasks.

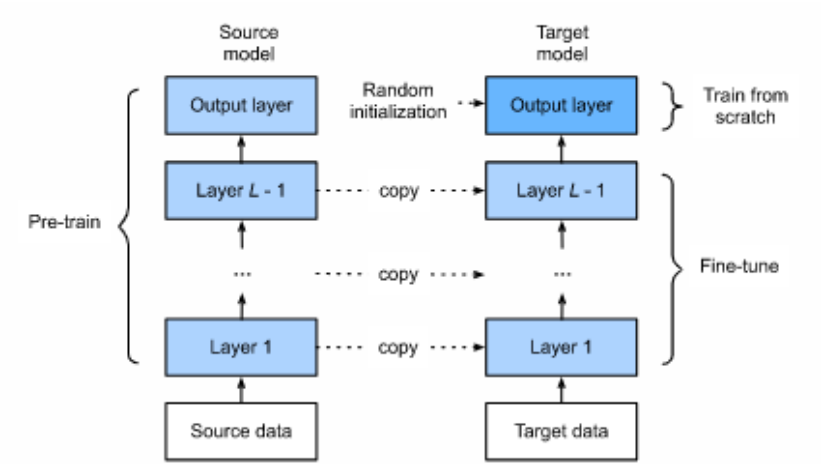
How to transfer knowledge from one model to another?.

- **Feature extractor**:
  - Chop off the top layer of the already trained model and replace it with a randomly initialized one.
  - Train the weight parameters only in the top layer for the new task, while all other weight parameters remain fixed (**frozen**).
  - Adopted when your data and the task are similar to the data and the task of the original pre-trained model.



Fine tuning

- Exploit a **“good”** source model and then construct your **target model** by replicating all architecture and parameters from the source model, except the output layer.
- All parameters are fine-tuned on the **target dataset**.
- Only the **output layer** of the target model needs to be trained from scratch
- It can be viewed as a **warm start** which will speed up the convergence.

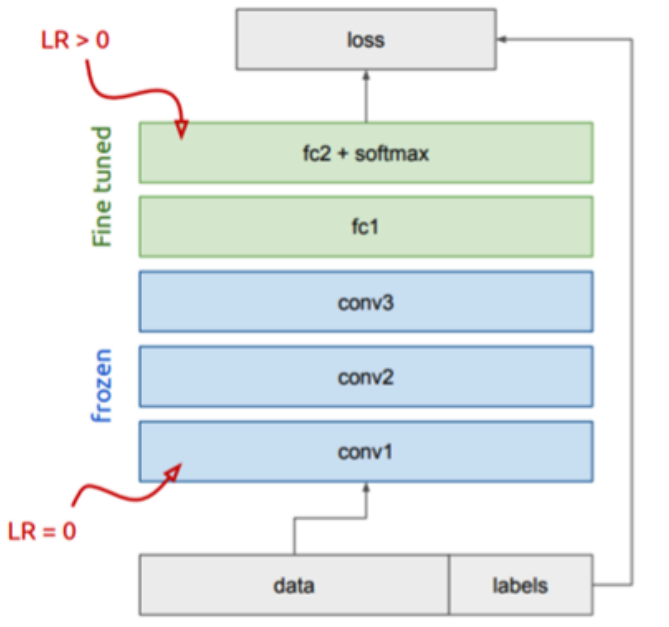


Fine-Tuning and/or Freeze

- **CNN** tend to learn edges, textures, and patterns in the first layers.
- The initial layers tend to capture generic features, while the later ones focus more on the specific task at hand.
- *Generic feature extractors* can be used in **many different types of settings**.
- Closer we get to the output, the more specific features the layers tend to learn, such as object parts and objects.

Fine-Tuning and/or Freeze

- Transfer all layers **except** the top layer or only transfer the first *n* layers.
- Only freeze the first layer of the pre-trained model and fine tune the subsequent layers.



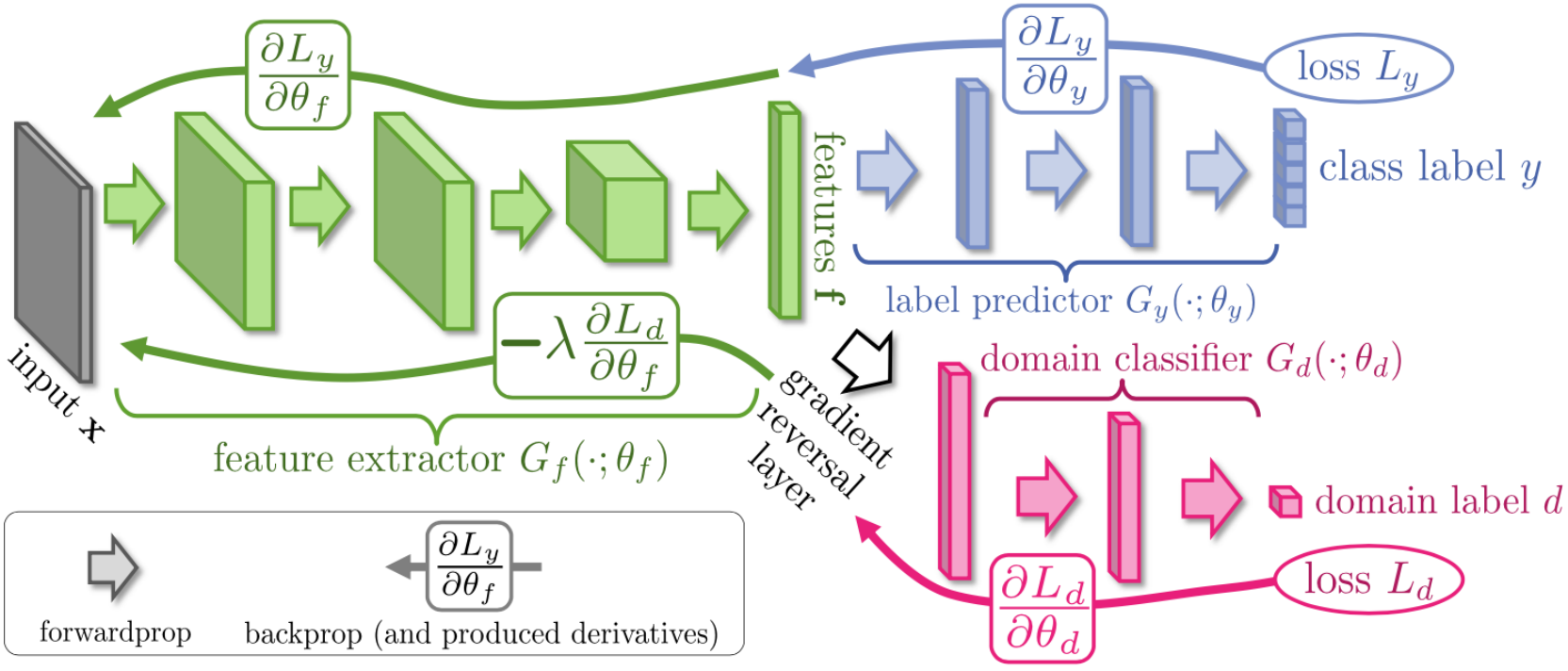
Tutorial on Transfer Learning

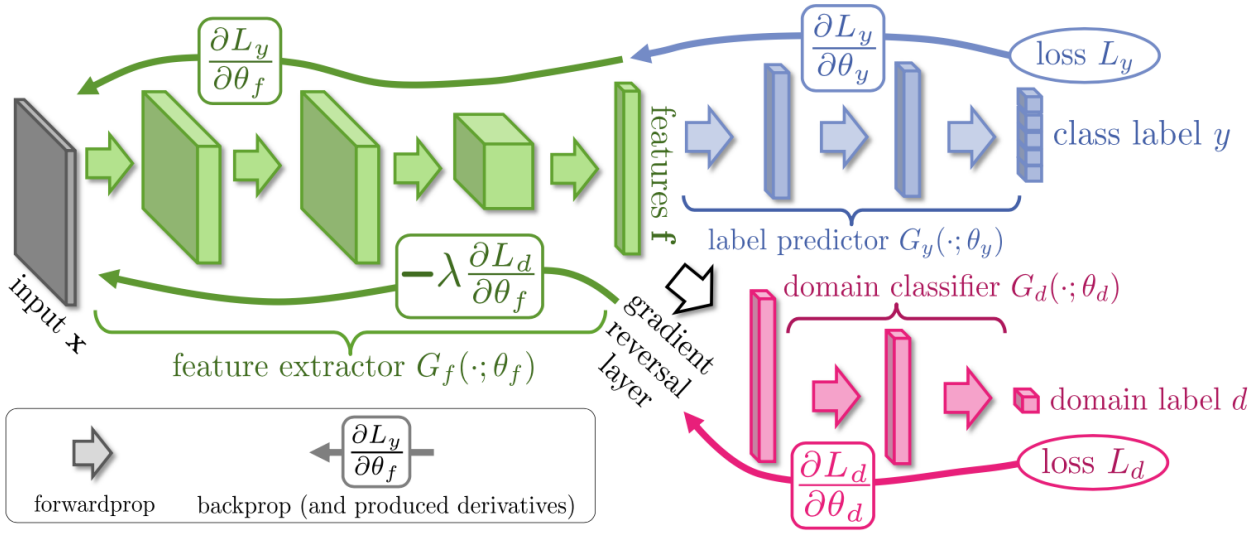
- Simple tutorial using Keras **here**

Unsupervised domain adaptation

Unsupervised Domain Adaptation is a learning framework to transfer knowledge learned from **source domains** with a large number of annotated training examples to **target domains** with unlabeled data only.

- Proposed by Yaroslav Ganin and Victor Lempitsky for Unsupervised Domain Adaptation by Backpropagation

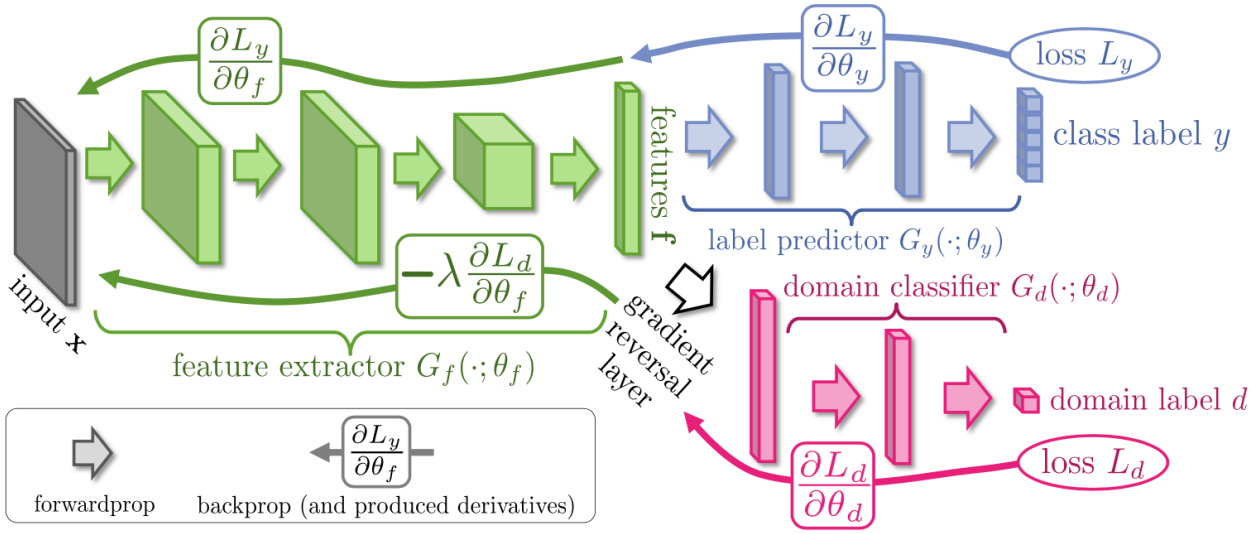




The proposed approach is based on three components:

- **Feature Extractor** component is exploited. It will learn to perform the transformation on the source and target distribution
- **Label Classifier component** will learn to perform classification on the transformed source distribution. Since, source domain is labelled
- **Label Domain Classifier** component which is a neural network that will be predicting whether the output of the Feature Extractor is from is from source distribution or the target distribution.

Intuition



- exploit the **Label Domain Classifier** to get the model to be confused about the domain meaning to maximize this loss (by reversing the gradient).
- confuse the **Feature Extractor** part in order it cannot create features that allow the domain classifier to work well but it still can create features that allow the label predictor to perform well
- details in the original paper here

Data augmentation

- Deep learning models usually need a lot of data to be properly trained.
- Collecting more data reduces variance
- Prevent overfitting
- If we cannot collect more data, we can augment the data we have

Image Augmentation for Computer Vision Applications

Simple transformations to the image include:

- **geometric transformations** such as Flipping, Rotation, Translation, Cropping, Scaling
- **color space transformations** such as color casting, Varying brightness, and noise injection.
- Example using *Pytorch* **here**

Image Augmentation for Computer Vision Applications

Original	Flip	Rotation	Random crop
			
- Image without any modification	- Flipped with respect to an axis for which the meaning of the image is preserved	- Rotation with a slight angle - Simulates incorrect horizon calibration	- Random focus on one part of the image - Several random crops can be done in a row

Figure from Afshine Amidi and Shervine Amidi

Image Augmentation for Computer Vision Applications

Color shift	Noise addition	Information loss	Contrast change
			
- Nuances of RGB is slightly changed - Captures noise that can occur with light exposure	- Addition of noise - More tolerance to quality variation of inputs	- Parts of image ignored - Mimics potential loss of parts of image	- Luminosity changes - Controls difference in exposition due to time of day

Figure from Afshine Amidi and Shervine Amidi

Image Augmentation for Computer Vision Applications

- These simple transformations have been largely exploited and shown improvement of the model for **computer vision tasks**:
  - image classification
  - object detection
  - segmentation.
- Applicable transformations are problem dependent.
- Be careful not to change correct class!
- For example, vertical flip is not appropriate if you have images of “b” and “d”
- But they are limited and might not be able to account all the possible variations.

Image Augmentation for Computer Vision Applications

An alternative is to use Deep Neural Network-based methods such as:

- **Adversarial Training**
- **Generative Adversarial Networks (GAN)**
- **Neural Style Transfer**
- See for more details the recent survey on **Image Data Augmentation** which illustrates how data augmentation can improve the performance of deep learning models.

Text Augmentation Techniques: Natural Language Processing

Easy Data Augmentation (EDA) for Natural Language Processing includes:

- **Synonym Replacement** Examples here
- **Random Insertion**
- **Random swap**
- **Random deletion**

However, such methods can struggle with preserving class labels.

- Recently **Wu et al. (2019)** have proposed a conditional BERT (CBERT) model which extends **BERT** masked language modeling (MLM) task by considering class labels to predict the masked tokens.
- BERT: **Bidirectional Encoder Representations from Transformers (BERT)** is a Transformer-based machine learning technique for NLP pre-training developed by **Google**.

Choose the good metric

TASK: Classification of skin cancer tumors (based on images)

- Decide between two models A and B to predict
  $y \in \{\text{malignant}, \text{benign}\}$

	Precision	*Recall*
Model A	93%	85%
Model B	86%	95%

- **Precision:** Of examples classified as malignant, what % are actually malignant?
- **Recall:** What % of malignant examples are actually classified as malignant?
- F1-score: **Avarage** of P and R : $\frac{2}{1/P+1/R}$

What a good evaluation metrics is depends on your problem

Changing evaluation metrics

- **Metric**: Classification error = 1- classification accuracy
  - Model A: 5%
  - Model B: 8%
- **Model A** A seems to do better but you prefer model B since it is doing better on malignant melanoma tumors which you are especially interested in.
- What to do ?
  - Change the metrics

$$\frac{1}{\sum_{i=1}^{m} w_i} \sum_{i=1}^{m} w_i \mathbf{1}_{\{\widehat{y}_i \neq y_i\}}, \quad w_i = \begin{cases} 10 & \text{if malignant melanoma} \\ 1 & \text{otherwise} \end{cases}$$

- and/or your validation/test dataset distribution (by including more malignant melanoma examples).

Changing evaluation metrics

- How do we make sure that we train our model towards this metric?
- If we have reweighed the metric, we can do the same for our training objective.

$$\text{Metric} := \frac{1}{\sum_{i=1}^{m} w_i} \sum_{i=1}^{m} w_i \mathbf{1}_{\{\widehat{y}_i \neq y_i\}}, \quad \text{Cost Function} := \frac{1}{\sum_{i=1}^{m} w_i} \sum_{i=1}^{m} w_i L(\widehat{y}_i \neq y_i)$$

Key point:

-1 Define your problem by choosing metric and validation/test dataset.

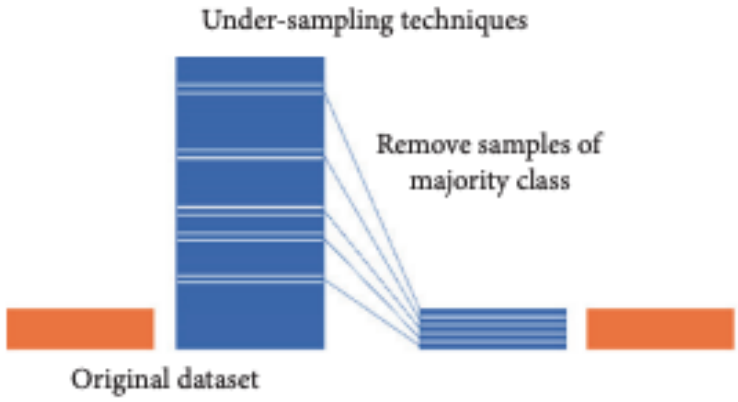
-2 Adapt your learning algorithm to do well on your metric

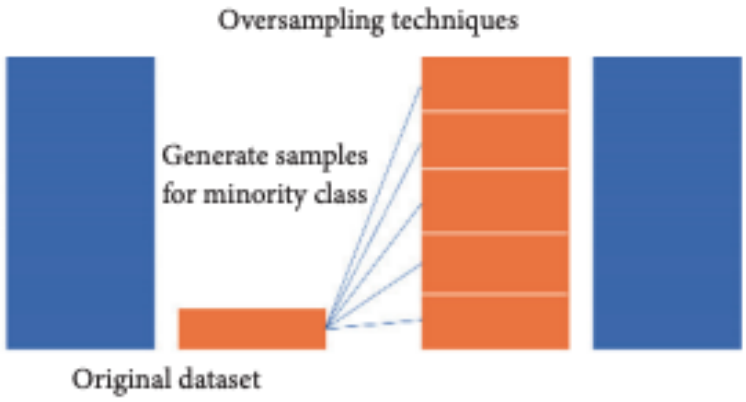
Imbalanced Dataset

Class imbalance problem in multiple areas: telecommunication managements, bioinformatics, fraud detection, medical diagnosis, . . .

Accuracy is not the good metric to look at: why ?

- A survey on Resampling approach





Drawback

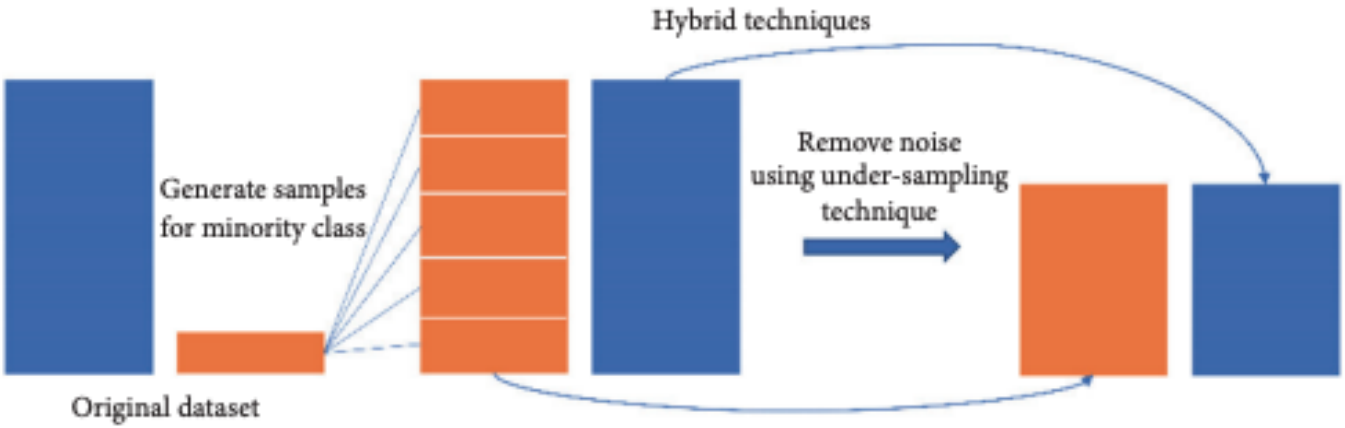
leave out important instances that provide important differences between the two classes.

Drawback

lead to model overfitting by introducing duplicate instances, drawing from a pool of instances that is already small.

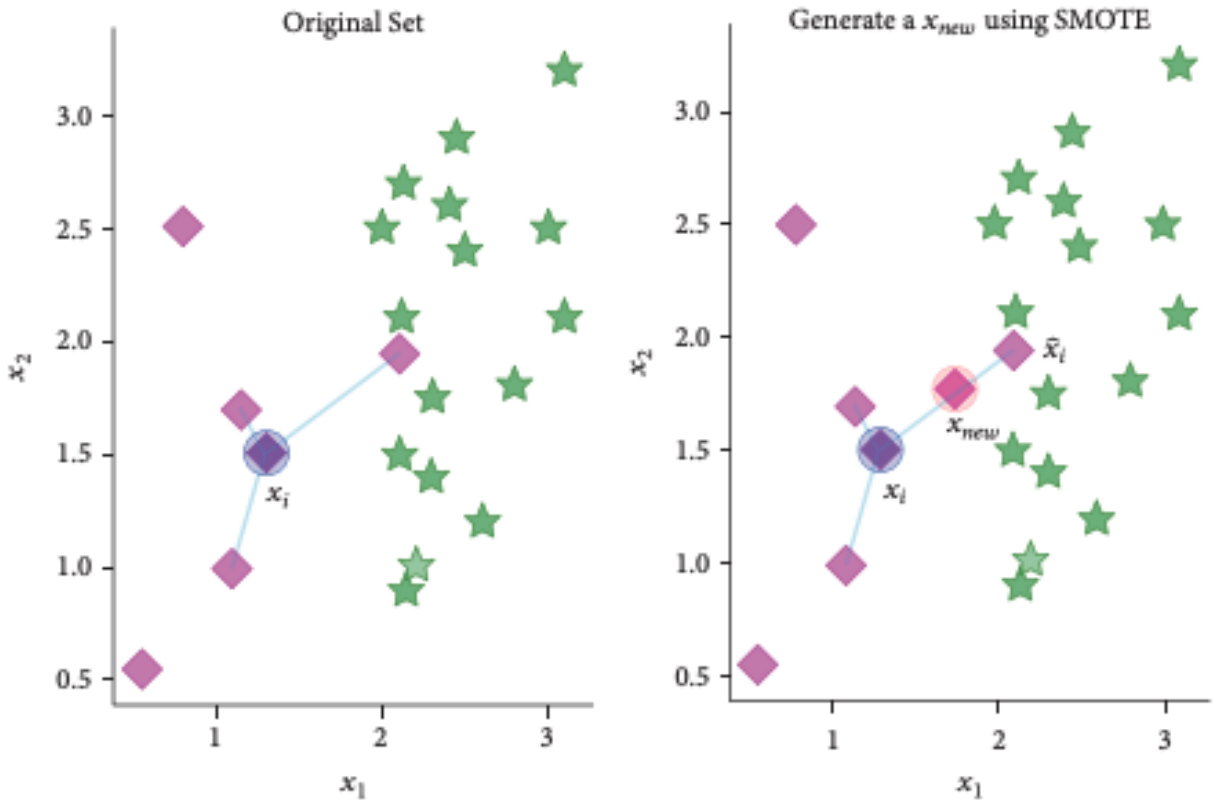
Hybrid approach

- Combining Oversampling and Undersampling



SMOTE (Synthetic Minority Oversampling Technique)

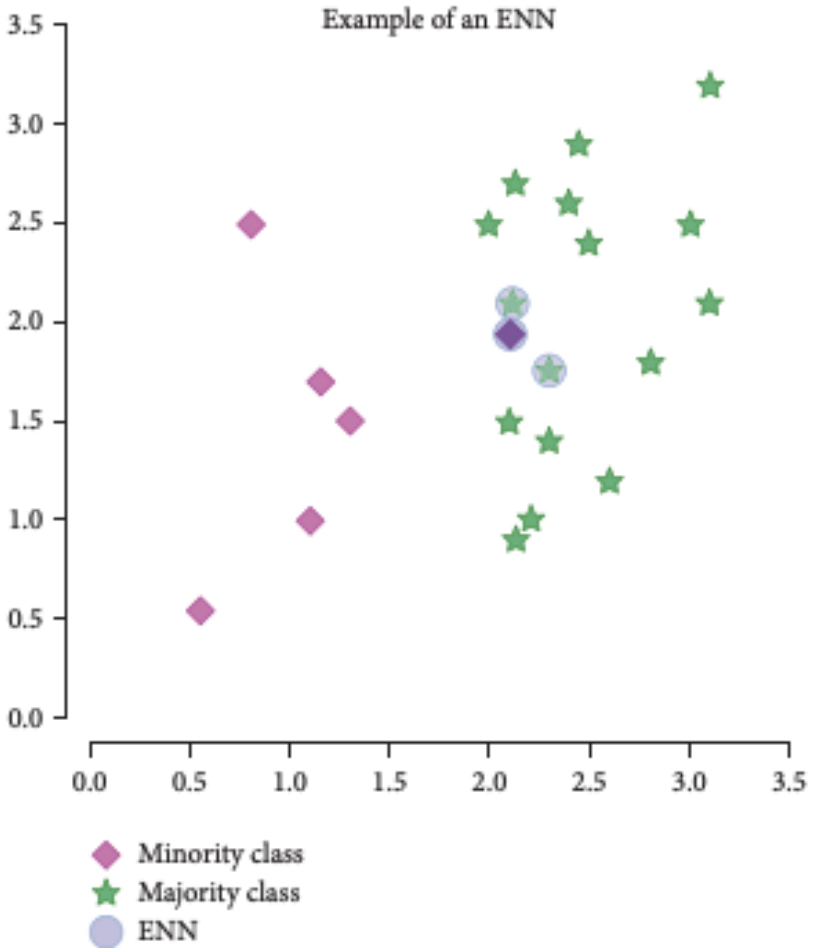
- Creates new instances of the **minority class** by creating convex combinations of neighboring instances.



Edited Nearest Neighbor (ENN)

Edited Nearest Neighbor undersampling of the majority class is done by removing points whose class label differs from a majority of its k nearest neighbor.

- Combine SMOTE and ENN: **SMOTE-ENN**



Imbalanced Dataset

- **Other Techniques**:
  - **Data augmentation**
  - **Changing the performance metric**: reweighed

Take Home Message

- Validation data
- Multitask learning
- Transfer learning
- Bayesian optimization
- Gaussian Process
- Decay step
- Early stopping
- Data augmentation