Lecture 7: Practical Considerations For Training Deep Models

Ali Harakeh

University of Waterloo

WAVE Lab
ali.harakeh@uwaterloo.ca

July 18, 2017

Overview

Section 1

Introduction

Introduction

- Successfully applying deep learning techniques requires more than just a good knowledge of what algorithms exist and the principles that explain how they work.
- A good machine learning practitioner also needs to know how to choose an algorithm for a particular application and how to monitor and respond to feedback obtained from experiments in order to improve a machine learning system.

Introduction

- During our day to day development effort, we need to know when to **collect more data**, **increase or decrease the capacity of our model**, **add or remove regularization**, **improve the optimization algorithm**, or simply just **debug the software implementation of the model**.
- Each one of the above operations is extremely time consuming and it is very important to us that we know exactly where we went wrong.

Advice From The Masters

- In practice, one can usually do much better with a correct application of a commonplace algorithm than by sloppily applying an obscure algorithm.

Methodology From The Masters

- **Determine your goals** : what error metric to use, and your target value for this error metric. These goals and error metrics should be driven by the problem that the application is intended to solve.
- **Establish a working end-to-end pipeline**: This should be done as soon as possible, and should include the estimation of the appropriate performance metrics.
- **Instrument the system well to determine bottlenecks in performance**: Diagnose which components are performing worse than expected and whether it is due to overfitting, underfitting, or a defect in the data or software.
- **Repeatedly make incremental changes**: This includes gathering new data, adjusting hyperparameters, or changing algorithms, based on specific findings from your instrumentation.

Section 2

Determining The Goals Of The System

Performance Metrics

- Determining which **error metric** to use is the most important first step for designing a deep learning system. This is because your error metric will guide all your future actions.
- The second step is determining the value of the error metric to **beat**.
- Keep in mind that for most applications, it is impossible to achieve **zero error**.

Performance Metrics

- The **most common reasons** for this phenomenon are:
  - **Imperfect optimization** procedure.
  - Not enough **training data**.
  - The real data distribution is not part of the **model distribution** family.
- Even with infinite training data and the ability to recover the true data distribution, there is a minimum error bound that a system can achieve.
- The less known reasons for this are:
  - The **input features** may not contain complete information about the output variable.
  - The process to be estimated might be **inherently stochastic**.

Performance Metrics

- The question is, how can we determine what minimum performance measure is required ?

Performance Metrics

- **Academic Setting**: *Standard benchmarks* for almost all applications already exist.
- The result to beat is the best performing published algorithm on those benchmarks.
- Keep in mind to not compare apples to oranges. Do not compare two algorithms that have been trained with different training data, and do not compare ensembles of networks to a single network.

Performance Metrics

- **Applications Setting**: We usually have some idea of the error rate that is necessary for an application to be safe,cost-effective, or appealing to consumers.
- In this setting, every method to get a performance boost should be used. This includes collecting more data, model ensembles, empirically determined thresholds, and dataset augmentation.

Section 3

Default Baseline Models

Default Baselines

- After choosing performance metrics and goals, the next step in any practical application is to establish a reasonable end-to-end system **as soon as possible**.
- I will provide some recommendations for default baselines based on my modest experience with deep networks.
- Keep in mind that deep learning research progresses quickly, so better default algorithms are likely to become available soon after this course ends.

Default Baselines

- You should **never use a cannon to hunt a rabbit.**
- Depending on the complexity of your problem, you might not want to use deep learning at all.
- If your problem does not fall in the **AI-Complete** category, then you will likely do well with a simple statistical model such as linear SVMs or logistic regression.

Default Baselines

- If your problem falls in the AI-Complete category, choose the base architecture based on the structure of the problem.
- Supervised learning problems with fixed small size input vectors will most likely use **feed forward Fully Connected Networks**.
- Supervised learning problems with with input variables that has known topological structure will most likely use **feed forward Convolutional Networks**.
- Supervised learning problems with input or output that are sequences, trees, or graphs should use **gated recurrent networks such as LSTMs or GRUs**.
- Unsupervised learning problems should consider **VAEs or GANs**.

Default Baselines

- **Image Feature Extractors**: VGG-16, ResNet-101, or Inception-V3.
- **Object Detection Baseline Models (2D)**: Faster-RCNN, YOLO-9000, RFCNs.
- **Semantic Segmentation**: Segnet, fully convolutional network.

Default Baselines

- **Optimizers**: ADAM, RMS-Prop. (Both with a decaying learning rate). Apply batch norm when possible, you will thank me later.
- **Regularization**: Early stopping should universally be used. Dropout is an easy regularizer to implement. Be careful to try batch norm first, since it might remove the necessity to use dropout.

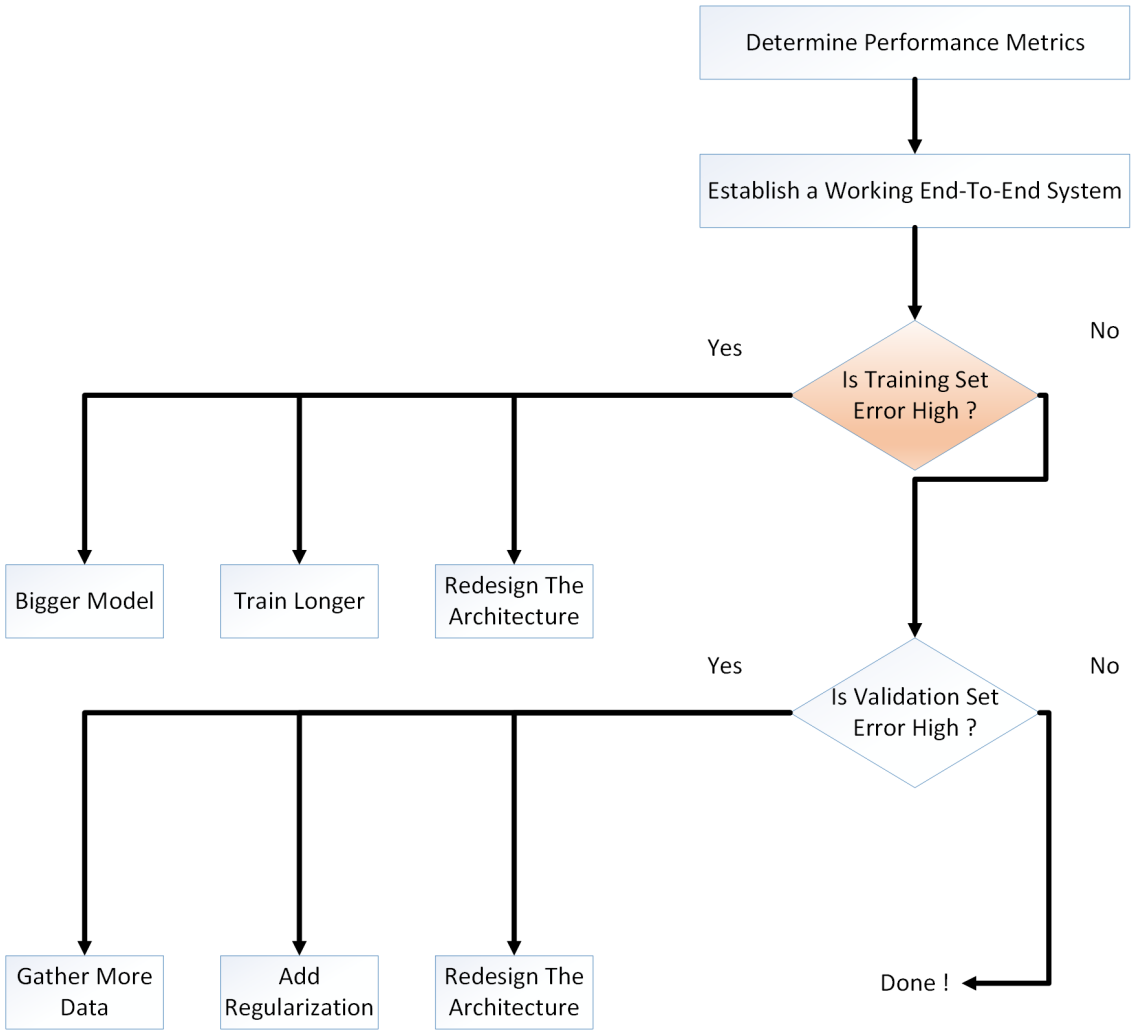
Section 4

The Design Process

The Design Process

- After establishing an end-to-end baseline, train and test on the dataset at hand. Always plot the training and validation errors.
- There are three scenarios that machine learning practitioners usually face:
  - The training set error remains high.
  - The training set error decreases, but the validation set error remains high.
  - Both training and validation set errors are low.

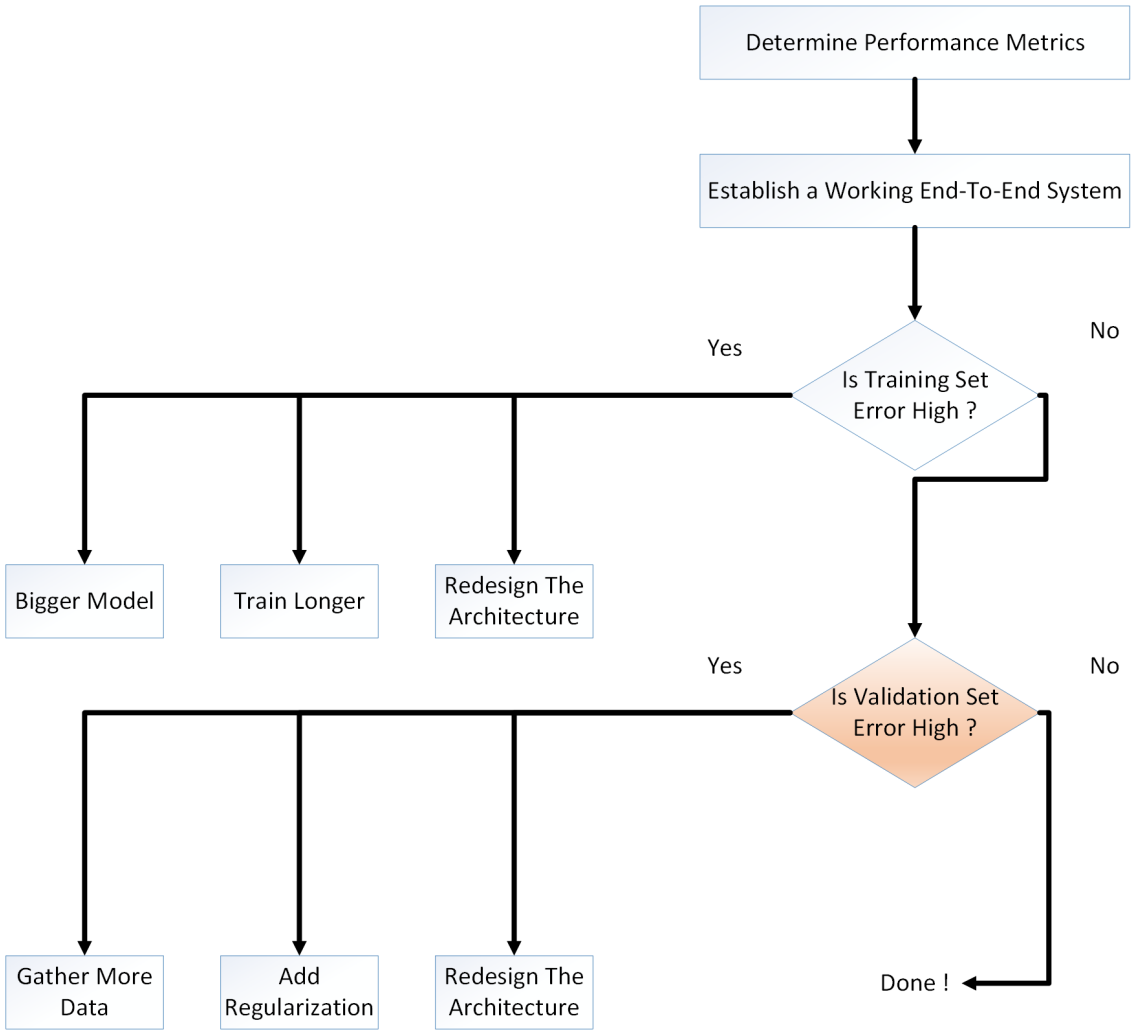
The Design Process



High Training Error: Debugging

- **Is the optimizer code running correctly?** *If you implemented the gradient functions yourself, make sure the gradient is correct via gradient checks. Is the learning rate correct?*
- **Does the model have high enough capacity?** *Make sure the model is able to overfit. a small portion of the dataset.*
- **Did you train for a reasonable amount of epochs ?**
- If all else fails, you need to rethink your architecture.

The Design Process



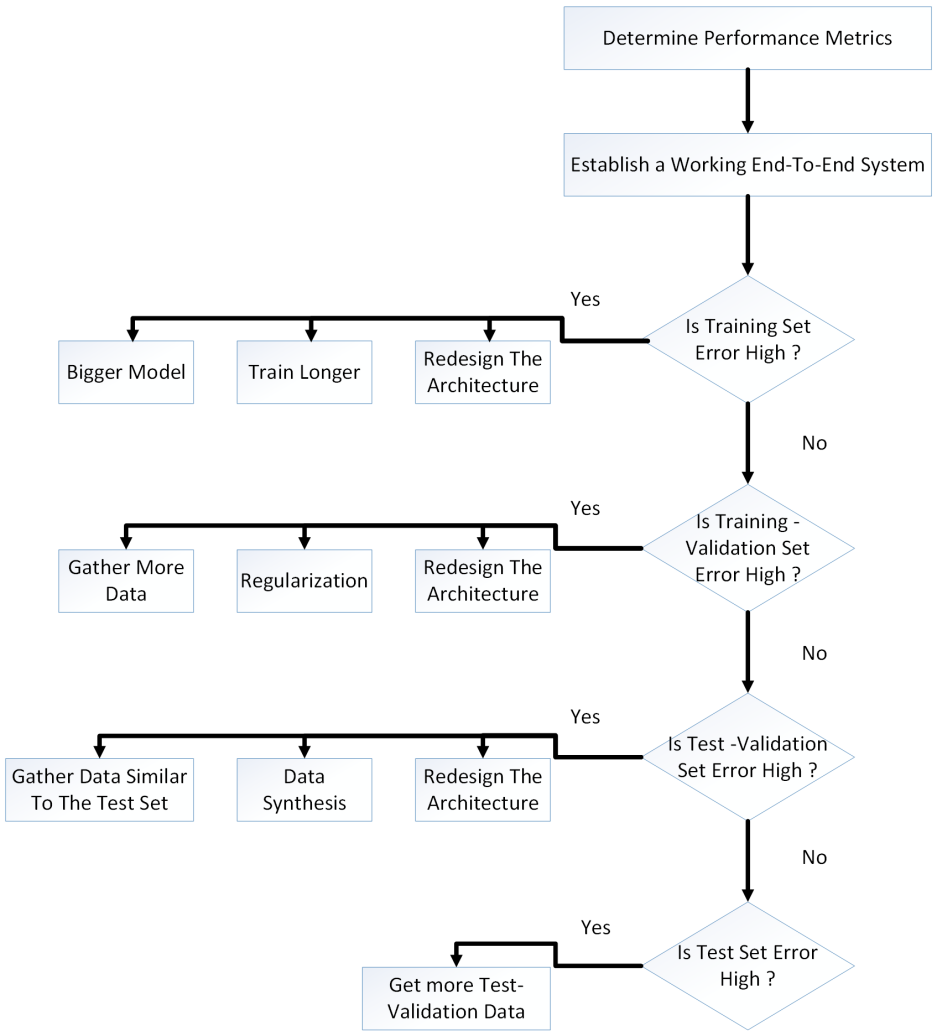
High Validation Error: Debugging

- **Do you have enough training data?** *Collect more training data, use dataset augmentation methods.*
- **Are you overfitting?** *Employ regularization strategies.*
- If all else fails, you will need to rethink your architecture.

What If Nothing Helps ?

- There is a very pathological case that is basically unknown to deep learning practitioners in academia.
- This case is illustrated as the validation set having a different distribution than the actual test set.
- This case motivates us to rethink the design process shown above.

The Design Process



Section 5

Hyperparameter Selection

Hyperparameter Selection: Manual Tuning

- Choosing hyperparameters manually requires understanding what the hyperparameters do the exact relationship between the hyperparameters, training error, generalization error, and computational resources.
- The primary goal of manual hyperparameter search is to adjust the effective capacity of the model to match the complexity of the task.

Effective Capacity

- **Effective Capacity** is governed by three factors:
  - The **representational capacity** of the model.
  - The ability of the learning algorithm to **successfully optimize** the cost function.
  - The degree to which the cost function and the training procedure regularize the model.
- Next, I will show some common hyperparameters and their effect on the effective capacity.

Number Of Hidden Units

- **Number of hidden units:** Increases representational capacity when increased and hence increase effective capacity.
- Increasing the number of hidden units increases both time and memory required for essentially every operation on the model.

Convolutional Kernel Width

- **Convolutional kernel width:** Increases the number of parameters in the model when increased and hence increase effective capacity.
- A wider kernel results in a narrower output dimension, reducing model capacity unless you use implicit zero padding to reduce this effect.
- Wider kernels require more memory for parameter storage and increase runtime, but a narrower output reduces memory cost.

Implicit Padding

- **Implicit padding:** Adding implicit zeros before convolution keeps the representation size large. Increasing the size of the padding increases the effective capacity of the model.
- A wider kernel results in a narrower output dimension, reducing model capacity unless you use implicit zero padding to reduce this effect.
- Implicit padding increases the size of the input and hence increases the time and memory cost of most operations.

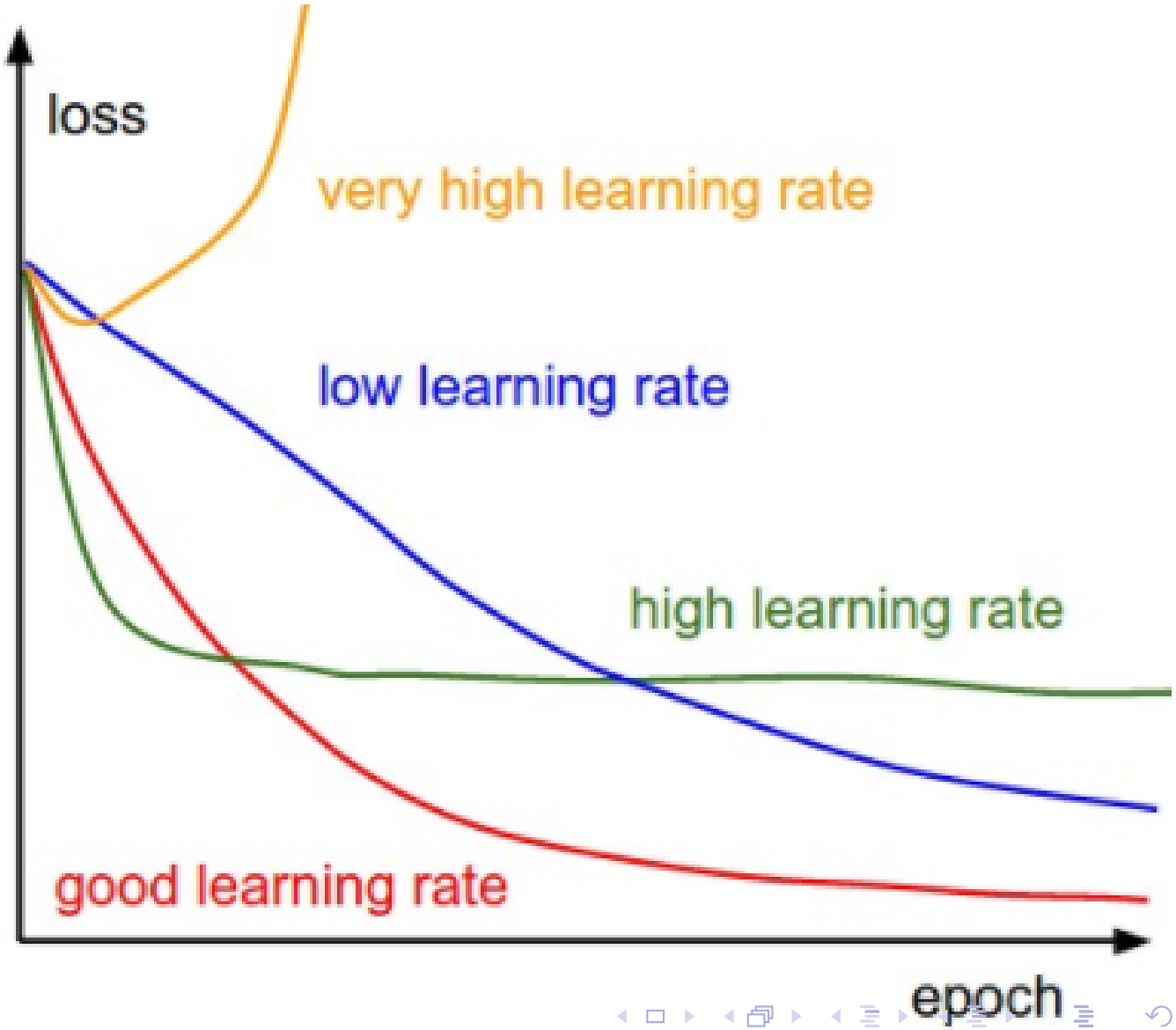
Weight Decay Coefficient

- **Weight decay coefficient:** Decreasing the weight decay coefficient frees the model parameters to become larger hence increasing the effective capacity of the model.

Dropout Rate

- **Dropout rate:** Dropping units less often gives the units more opportunities to *conspire* with each other to fit the training set.

Learning Rate



Learning Rate

- **The Learning Rate** is perhaps the most important hyperparameter. If you have time to tune only one hyperparameter, do that for the learning rate.
- The learning rate controls the effective capacity of the model in a complex way.
- The effective capacity is the highest when the learning rate is **correct**.

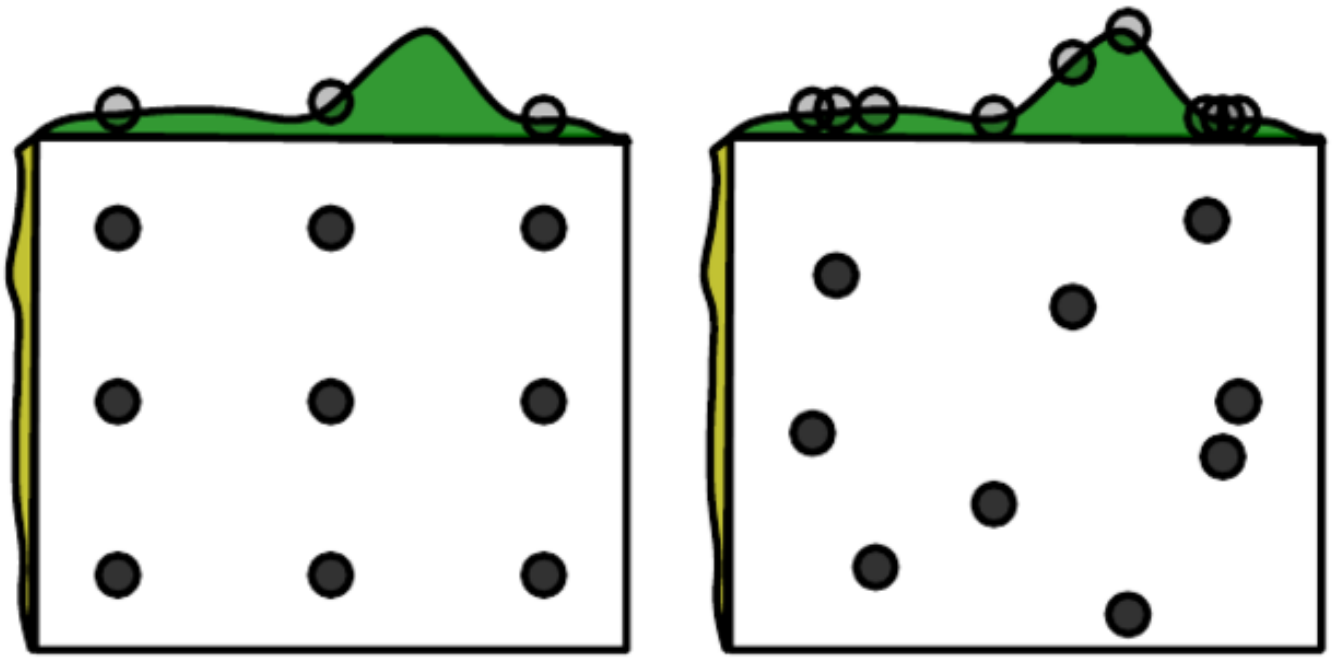
Hyperparameter Selection: Automatic Tuning

- The ideal learning algorithm just takes a dataset and outputs a function, without requiring hand-tuning of hyperparameters.
- Manual hyperparameter tuning can work very well when the user has a good starting point, such as one determined by others having worked on the same type of application and architecture, or when the user has months or years of experience in exploring hyperparameter values for neural networks applied to similar tasks.

Hyperparameter Selection: Automatic Tuning

- **Automatic Hyperparameter Selection** algorithms wrap around the learning algorithm and choose its hyperparameters.
- Hyperparameter optimization algorithms often have their own hyperparameters, such as the range of values that should be explored for each of the learning algorithm's hyperparameters. These however are much easier to determinate.

Grid VS Random Search



Grid Random

Section 6

Course Conclusion

What Can Deep Learning Do ?

- **Deep Learning Is A Black Box That Can Solve Everything !**

What Can Deep Learning Do ?

- **NO**!
- The current state of deep learning allows us to tackle **inference tasks** that can be done by humans in less than **one second**.
- It also allows us to tackle a set of tasks that is very hard for human beings to perform. This set is the set of **prediction tasks**.

What Can Deep Learning Do ?

- Deep learning is a set of mathematical tools, and as with any other tool, it can be misused.
- This course served as a mere **introduction** to deep learning.
- Many very important concepts such as unsupervised learning models, few shot, one shot, and zero shot learning, domain adaptation, adversarial examples, and Generative models have not been covered in this course.
- Finally, you should know that there is a high probability that whatever was discussed in this course will become obsolete in **6 months**.