Deep Learning

Chapter 11 - Practical Methodology

Cherif Jazra
@jazracherif
YouTube Part 1 Part 2

Comments section is open

Chapter Content

Introduction

How do we choose an algorithm for a particular application?
How do we improve the performance of our machine learning algorithm?

We have to make some decisions about:

- When to gather **more data**
- Whether to increase or decrease **model capacity**
- Add or remove **regularizing features**
- To improve the **optimization** of a model
- To improve **approximate inference** in a model
- To debug the **software implementation** of the model.

Knowing a wide variety of algorithms is not as important as being able to debug and optimize simple algorithms which may achieve expected goals.

Many of the chapter's recommendation adapted from Andrew Ng "ML Advice":
https://see.stanford.edu/materials/aimlcs229/ML-advice.pdf

Recommended Design Process:

1. Based on your application, determine your goal: what **error metric** to use, and its **target value**.
2. Establish ASAP a working **end-to-end pipeline**.
3. **Instrument** the system to identify bottlenecks. Diagnosis for overfitting, underfitting, or defect in data or software.
4. Iterate through incremental changes: gather new data, adjust hyperparameters, change algorithm etc…

11.1 Performance Metrics

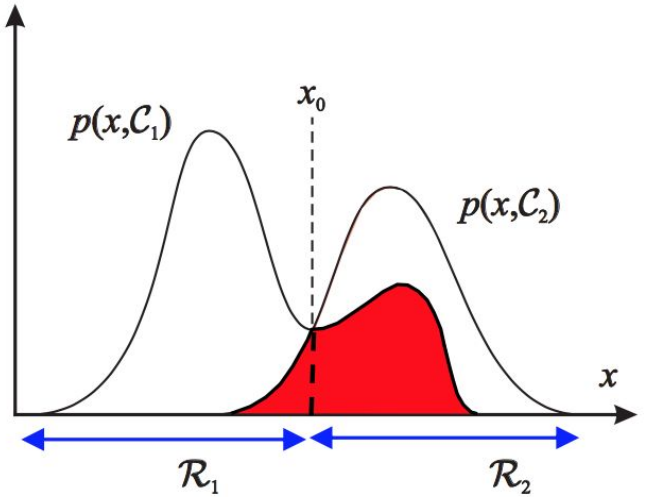
Your goal must be expressed in terms of which error metric to use and its target value.

1) Zero-error is not an option: Bayes Error defines the minimum error rate you can hope to achieve, even if you have infinite training data and can recover the true probability distribution. Why? Non-zero probability of belonging to more than 1 class: the input feature set may not contain complete information about the output variable, system may be intrinsically stochastic.

Bayes error is the misclassification error

$$
\begin{aligned}
p(\text{error}) &= \int_{-\infty}^{+\infty} p(\text{error}, x)\, dx \\
&= \int_{\mathcal{R}_1} p(x, C_2)\, dx + \int_{\mathcal{R}_2} p(x, C_1)\, dx \\
&= \int_{\mathcal{R}_1} p(C_2|x)p(x)\, dx + \int_{\mathcal{R}_2} p(C_1|x)p(x)\, dx
\end{aligned}
$$

True pdf of Classes



from p3 in http://www.robots.ox.ac.uk/~az/lectures/est/lect56.pdf

2) Data Gathering: a Tradeoff between the cost of gathering new data and the error reduction value.

Data collection requires:

- Time
- Money
- Human suffering (ex: invasive medical tests)

3) Level of Performance: How to determine what is a **reasonable performance** level?

- In academic setting, **published benchmarks** provide guidance.
- In real-world scenarios, aim for a level of performance that will
  - Ensure the application is safe.
  - It is cost-effective.
  - It is appealing to the consumer.

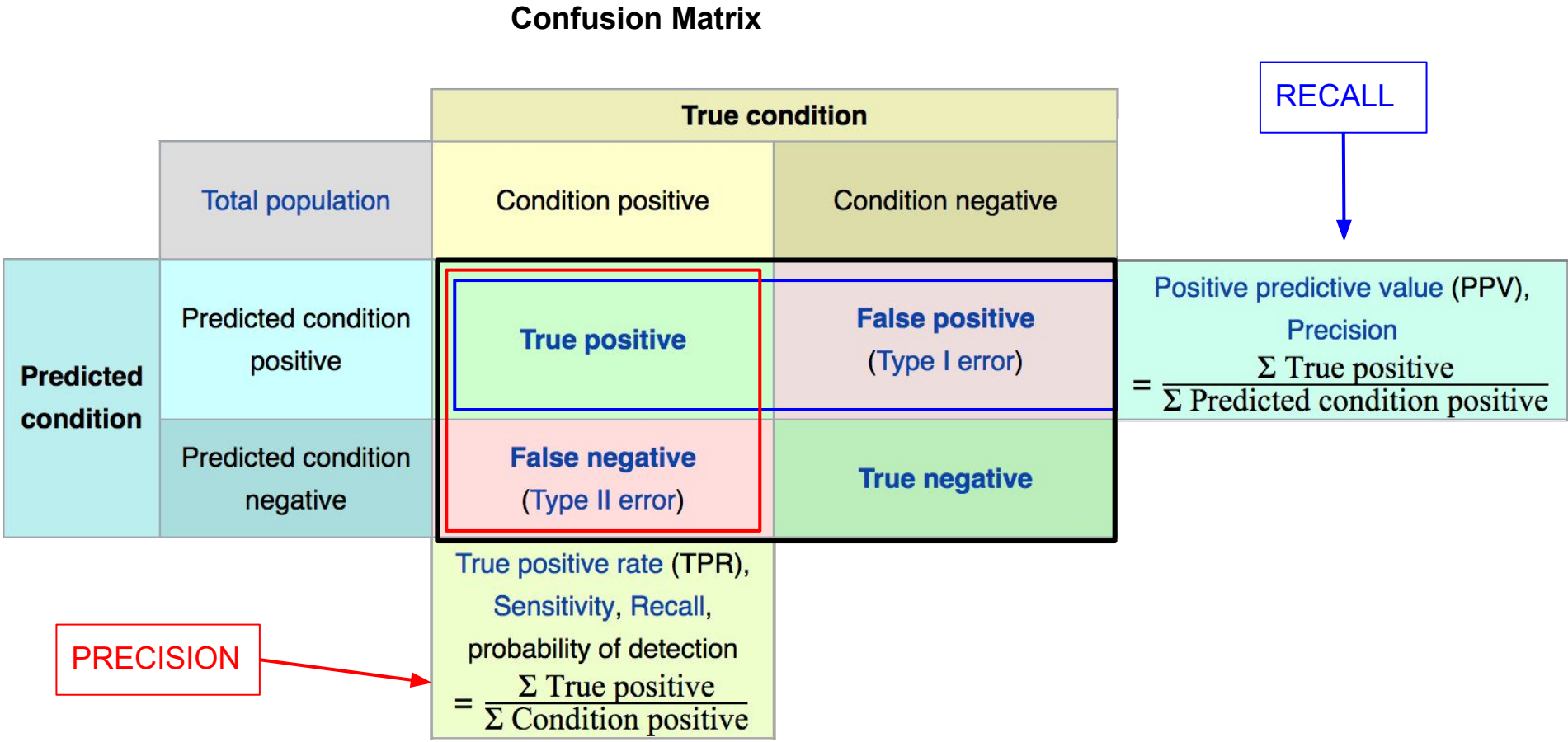
4) Choice of the metric: Commons ones such as **accuracy** and **error rates**.
BUT consider these applications:

1) Detect spam emails: It is less important to misclassify spam than misclassifying good emails.
2) Detect a rare disease: we can achieve 99.9999% accuracy by hardcoding a classifier to always report the disease is absent.

More advanced metrics:

- **Precision**: The fraction of detections reported by the model that were correct.
- **Recall**: The fraction of true events that were detected.

⇒ a **Confusion Matrix** helps calculate these metrics (https://en.wikipedia.org/wiki/Confusion_matrix)


Confusion Matrix
True condition
Total population
Condition positive
Condition negative
Predicted condition
Predicted condition positive
True positive
False positive
(Type I error)
Predicted condition negative
False negative
(Type II error)
True negative
RECALL
Positive predictive value (PPV), Precision
= Σ True positive / Σ Predicted condition positive
PRECISION
True positive rate (TPR), Sensitivity, Recall, probability of detection
= Σ True positive / Σ Condition positive

11.1 Performance Metrics

PR curve: a Plot with Precision in y-axis and Recall in x-axis.
What for? Consider δ s.t p(y =1 |x) > δ => class 1. The PR helps find the best classification threshold that improves both precision and recall.

F1 score: the harmonic mean of PR

$$F_1 = 2 * \frac{precision * recall}{(precision + recall)}$$

Others:
Area under the PR Curve
Area under the ROC curve (AUC)

Coverage: When should a Machine Learning system provide an answer?

If the machine is not confident it can answer a question better than human beings, the best course is to allow the human to perform the task.

Such models must support a measure of confidence, below which no prediction is given.

Coverage = fraction of examples for which the machine learning system is able to produce a response.

Other: Clickthrough rates, user satisfaction surveys, application specific metrics.

11.2 Default Baseline Models

Establish a working baseline model as early as possible, perhaps without deep learning, ex: logistic regression.

Is the problem **AI-Complete/AI-Hard**? Ex: object recognition, speech recognition, or machine translation, then starts with Deep Learning.

1) **General category of model**: should be based on structure of the data
    a) Supervised learning with fixed-size input vector ⇒ feedforward network with fully-connected layers
    b) Known topological structure like an image? ⇒ convolutional network
        i) Start with Relu or Leaky Relu, preLus, or maxout.
        ii) If input or output is a sequence, use GRU or LSTM

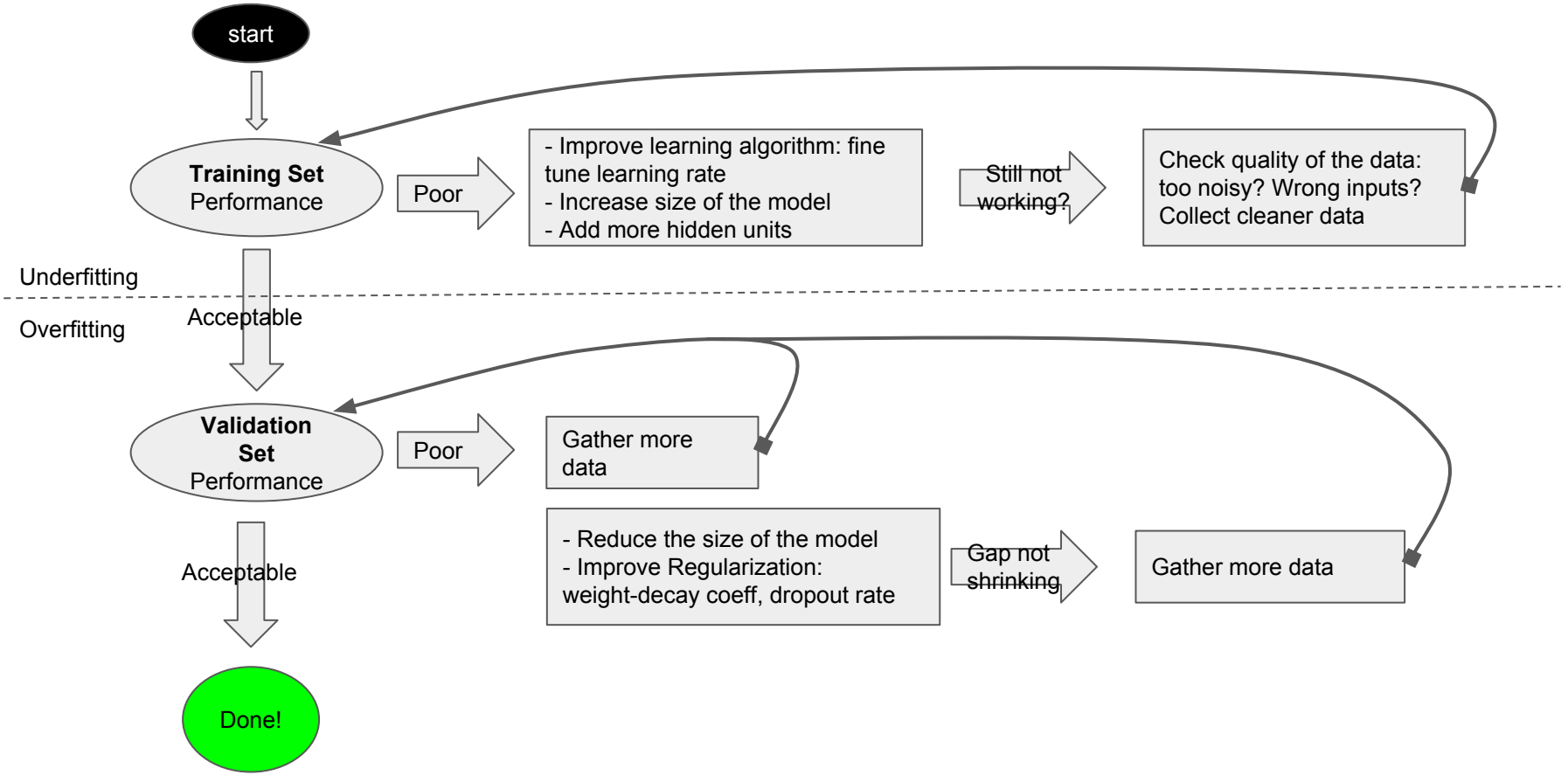
2) **Optimization Algorithm:**
   a) Starts with SGD with momentum (or Adam) and a decaying learning rate:
      i) Decay linearly until you reach a fixed minimum
      ii) Decay exponentially
      iii) Decrease the rate by factor of 2-10 when validation error plateaus
   b) **Regularization**: to avoid overfitting
      i) Always use Early Stopping (7.8)
      ii) Dropout (7.12)
      iii) Batch Normalization / Adaptative Reparametrization (8.7.1)

3) Is this a common task? Copy the model and algorithm, perhaps even copy the trained model.
Ex: use features from a ConvNet trained on ImageNet to solve other computer vision task (Girshick et al., 2015)

4) Should you begin with unsupervised learning?
Some domains benefit a lot, ex: NLP with word embeddings trained in unsupervised fashion).
Computer vision doesn't benefit much except when number of labels is small (semi-supervised)

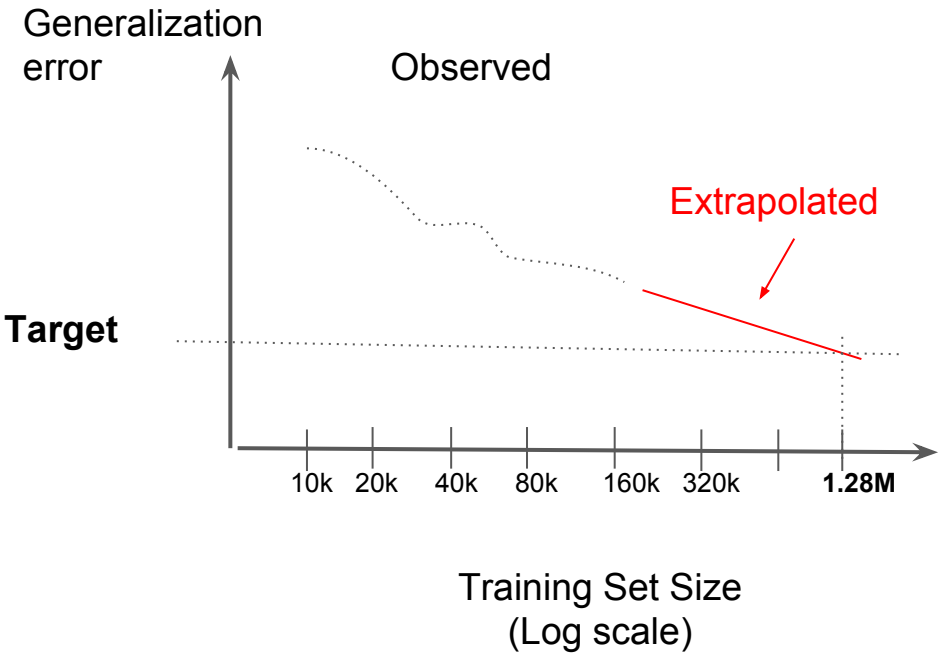
11.3 Determining Whether to Gather More Data

Start with an established End-to-End system



11.3 Determining Whether to Gather More Data

How much data to gather? Create a plot



11.4 Selecting Hyperparameters

Manual selection algorithms: requires a good understanding of what the hyperparameters do and how machine learning models achieve good generalization.

Automatic selection algorithms greatly reduce the need to understand underlying theory, but are more computationally costly.

11.4.1 Manual Hyperparameter Tuning

Goal: to adjust the **Effective Capacity** of the model to match the complexity of the task.

3 types of constraints of the effective capacity:

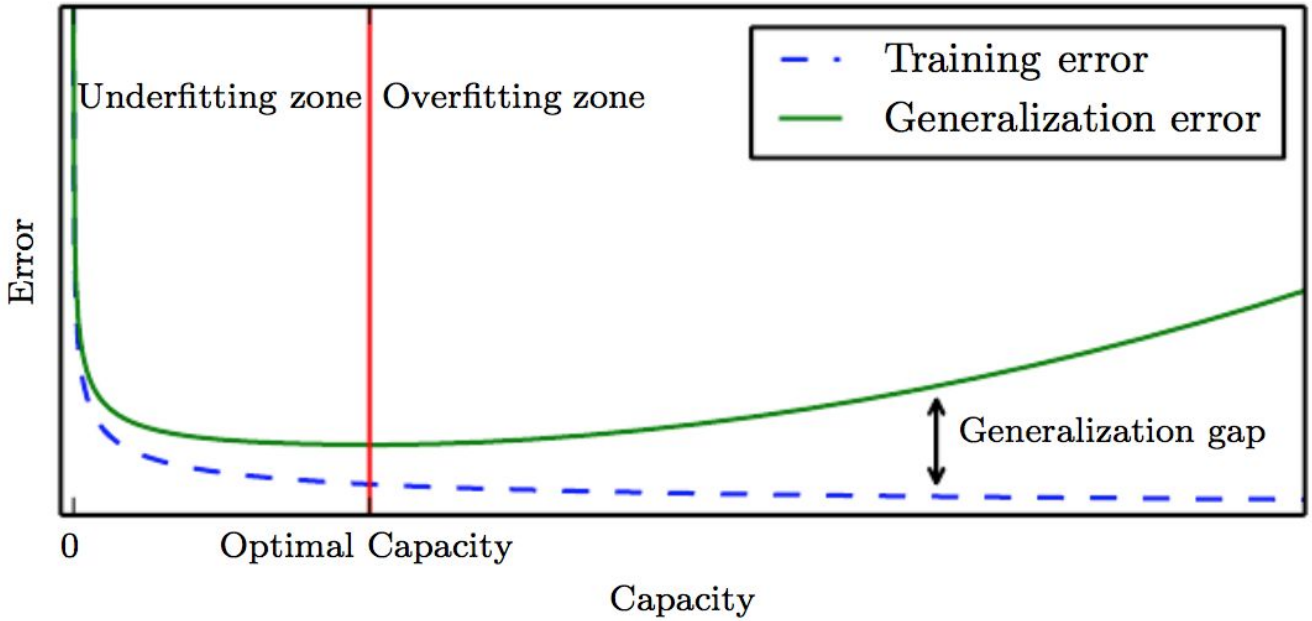
1. The **representational capacity** of the model.
2. The ability of the learning algorithm to successfully **minimize the cost function.**
3. The degree to which the cost function and training function **regularize** the model.

A non-exhaustive list of parameters:

1. Number of Layers
2. Number of hidden Units per layer
3. Learning Rate
4. Convolutional kernel Width
5. Implicit Zero Padding
6. Weight Decay coefficient
7. Dropout Rate
8. Number of linear pieces in a maxout unit
9. Preprocessing step

11.4.1 Manual Hyperparameter Tuning

Generalization error has a **U-shape** in terms of Capacity (Figure 5.3)



The goal is to explore the U-shaped curve in order to find the optimal capacity, located between underfitting and overfitting zones.

11.4.1 Manual Hyperparameter Tuning

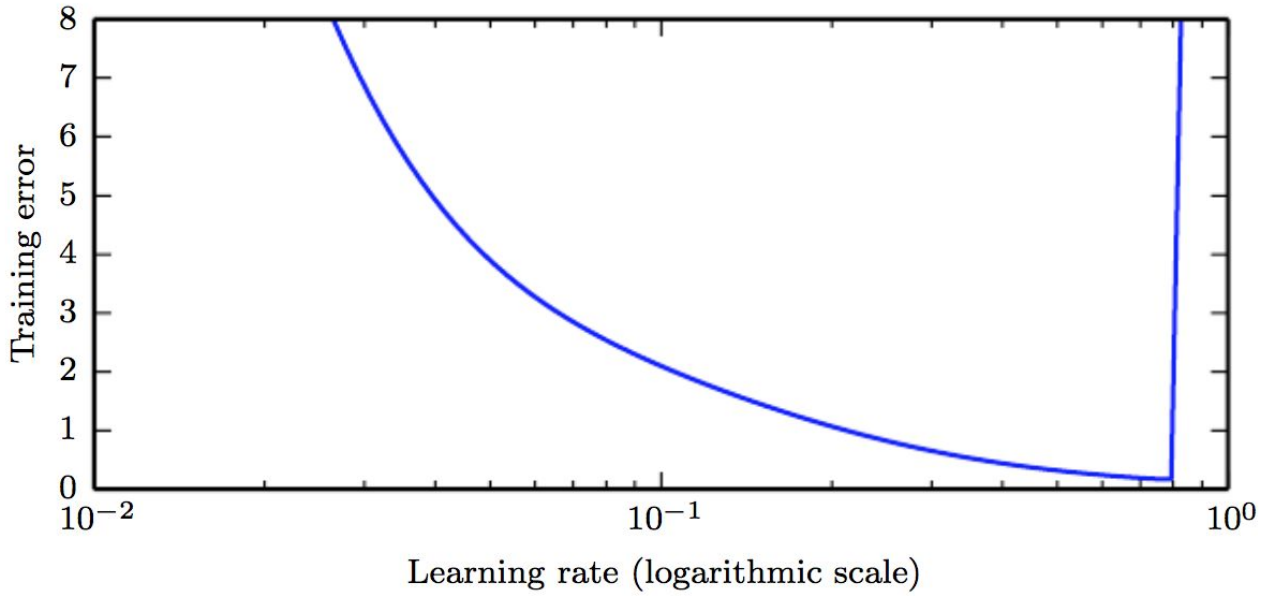
For what hyperparameter values does overfitting occurs?
Number of units : High values
Weight-decay Coefficient: Closer to zero has greater capacity

U-shaped function is not continuous:

- Some parameters are discrete: Num of units, num of linear pieces in maxout unit.
- Some are binary: switches for pre-processing feature normalization.
- Some have min-max ranges: min-value for weight decay.

11.4.1 Manual Hyperparameter Tuning

Learning rate: the most important hyperparameter.
Effective capacity is highest when the LR is correct for the optimization problem (not smaller or higher)



11.4.1 Manual Hyperparameter Tuning

Monitor both **Training** and **Validation** Error to diagnose overfitting or underfitting

Error on **Training set** too high? → increase Capacity
Not using regularization + confident Optimization algorithm is correct → add more layers or more hidden units.

Error on **Validation Set** too high? Error = Train_error + (Validation_error - Train_error)
NN work best when training error is low. Performance driven by the gap.
⇒ How to decrease Validation error faster than decreasing Training error?

To reduce gap: Change Regularization hyperparameters: add dropout or weight decay. A large model that is well regularized performs well.

If you have low training error, you can always reduce generalization error by collecting more training data.

11.4.1 Manual Hyperparameter Tuning

How to choose Hyperparameters values to **increase** capacity?

Hyperparameter	**Increases CAPACITY when...**	**Reason**	**Caveats**
Number of hidden Units	Increased ++	Increases Representational Capacity	higher time and memory cost
Learning Rate	Tuned optimally	Avoid Optimization failures	
Convolution Kernel Width	Increased ++	Increases the more parameters	Narrower output dimension unless zero padding used + increase in storage and runtime but lower memory cost
Implicit Zero Padding	Increased ++	Keeps representation size large	higher time and memory cost
Weight-Decay Coefficient	Decreased --	Lets parameter coefficients become larger	
Dropout Rate	Decreased --	More opportunities for units to "conspire" together	

11.4.2 Automatic Hyperparameter Optimization Algorithms

Usually best when a model has only small number of hyperparameters

NN however benefit significantly from tuning 40 or more parameters.

If other people worked on a similar problem, manual tuning can work well. Otherwise, use automatic as a starting point.

Manual search is an optimization problem itself. Could we develop hyperparameter optimization algorithms? Pb: they have their own hyperparameters, such as allowed range, but they are easier to choose

3 other types:

1. Grid Search
2. Random Search
3. Model-based hyperparameter Optimization

11.4.3 Grid Search

The user selects a small finite set of values to explore.

→ Cartesian product of all values tried and the one that achieves best validation set error is picked.

How to pick range? Start conservatively. Use prior experiences, pick values on a logarithmic scale.
Ex: typical **Learning Rate** range { 0.1, 0.01, 10^{-3}, 10^{-4}, 10^{-5}}
Number of hidden units {50, 100, 200, 500, 1000, 2000}

If best choice is at the edge of the range, extend that range
Ex: if for parameter in range { -1, 0, 1}, best value is for 1, then shift to {1, 2, 3}. If it is 0, then reduce the range {-0.1, 0, 0.1}

Computational cost: for m hyperparameters each with at most n values: $O(n^m)$
→ Can use parallelism but cost is still high because of exponential size of the space

11.4.4 Random Search

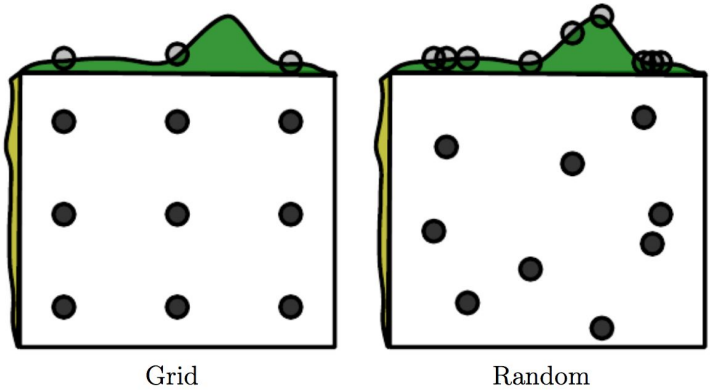
Alternative to Grid search:

- Simple to program
- More convenient to use
- Converges faster to good values

See Bergstra and Bengio 2012: http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

1) Define a marginal distribution for each parameter: Bernoulli/Multinoulli (discrete values) or Uniform on Log scale (positive real values).
2) Draw a combination of parameter from a joint distribution (typically a product of all distr.)

→ this will cover a larger hyperparameter space and be more effective in case where a small subset of parameters strongly affect the cost function (low effective dimensionality in Bergstra)



(*right*) More coverage using Random Search, therefore more chance to hit the peaks

(*left*) Grid is more likely to repeat equivalent experiments

11.4.5 Model-based Hyperparameter Optimization

The search of Hyperparameters as an optimization problem.

- The variables are the hyperparameters
- The cost function to optimize is the **validation set error**. Can we compute the gradient of this function? Usually no

We build a model of the validation set error function such as Bayesian Regression model: This model will estimate:

- The expected value of the validation set error for each hyperparameter
- The uncertainty around this expectation

Tradeoff between **exploration** and **exploitation**:

Proposing hyperparameters for which there is high uncertainty, may lead to large improvement but may also perform poorly

v.s

Proposing hyperparameters the model has high confidence they will perform well

NOTE:

- Authors cannot unambiguously recommended Bayesian hyperparameter optimization! May perform better, may fail catastrophically.
- Also may require experiments on a large set of parameters before drawing conclusions.

11.5 Debugging Strategies

Usually 2 difficulties in evaluating poor performance:

1. We don’t know a priori how the ML algorithm will behave. We can’t predict its behavior.
2. We don’t know which submodule misbehaves as another adaptable one may compensate for those errors.

Ex: suppose we manually implement SGD but we make an error for the biases updates:

$$b \leftarrow b - \alpha, \text{ where } \alpha \text{: Learning Rate}$$

→ Biases will get more negatives but weights might compensate for this effect. The bug won’t be clear from the output

11.5 Debugging Strategies

Some important debugging tests:

1. **Visualize the model in action**:
    - For object detection cases, view the objects superimposed on original image.
    - For generative speech mode, listen to the output audio
    - Don’t just rely on quantitative measures for evaluation, this also helps you confirm whether your evaluation routines properly work.

2. **Visualize the worst mistakes**:
    - Use confidence metrics such as softmax output probabilities to evaluate performance.
    - Look at the training example that performs worst. Can help find bugs with preprocessing or labeling logic. Ex: cropped images in street number labeling problem.
    - Sort the examples in order of most confident mistakes.

3. **Reason about software using training and test error**:
    - If **train_error** is low but **test_error** is high: likely that training procedure works well but it is overfitting for algorithmic purposes.
    - Check whether model is saved properly after training step for measuring test error.
    - Check whether test data is prepared differently than train data
    - If both errors are high, it is difficult to decide whether it’s a software bug or algorithmic problem

4. **Fit a tiny dataset**:
    - If train_error is high, this can help identify whether the software is buggy
    - Model should be able to fit 1 samples, either labeling it correctly if it was a classifier, or reproducing it if it was a generative model.

5. **Compare back-propagated derivatives to numerical derivatives**:
   - Implementing one's own gradient step may lead to mistakes.
   - Compare the implementation of the derivative with the method of **finite differences**:

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon} \qquad f'(x) \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

Or **centered differences**: $$f'(x) \approx \frac{f(x+\frac{1}{2}\epsilon) - f(x-\frac{1}{2}\epsilon)}{\epsilon}$$

Pick the ε large enough not to be rounded by finite-precision numerical computation.

11.5 Debugging Strategies

The method of finite difference is for single variable functions, while NN usually computes gradient of **vectors**.

Ex: If $g : R^m \rightarrow R^n$

1) One could compute m*n partial derivatives.
2) Compute the derivatives at input and output of a projection of g.
   - Ex: $f(x) = \mathbf{u}^T g(\mathbf{v}.x)$ where u, v randomly chosen vectors.
   - f has a single input and single output, so it is efficient to compute!
3) If numerical computation framework is available, use **complex number** at input and avoid cancellation effect due to subtractions (Squire and Trapp, 1998).

   By definition: $f(x + i \cdot y) = u(x,y) + i \cdot v(x,y)$ where $i=\sqrt{-1}$, $x,y \in \mathbb{R}$, u,v real functions

$$f(x + i\epsilon) = f(x) + i\epsilon f'(x) + O(\epsilon^2)$$

$$\text{real}(f(x + i\epsilon)) = f(x) + O(\epsilon^2), \quad \text{imag}(\frac{f(x + i\epsilon)}{\epsilon}) = f'(x) + O(\epsilon^2),$$

6. **Monitor histograms of activations and gradients**:
    - Look at pre-activation values of hidden units for saturation:
        - if using **ReLu,** check for the number of off-units and for how many epochs they are off
        - For **tanh**, look at the average of absolute value of pre-activation
    - For deep networks, Look at how quickly **gradients** increase or decrease.
    - Compare **magnitude** of parameters to their gradients. For a minibatch update, gradient value should be about 1% of parameter's value, not 50%, not 0.001% (Bottou 2015)
    - Are the optimization algorithm guarantees met? (approximate inference algorithm, part III)

11.5 Debugging Strategies

Summary of techniques:

1. Visualize the model in action
2. Visualize the worst mistakes
3. Reason about software using training and test error
4. Fit a tiny dataset:
5. Compare back-propagated derivatives to numerical derivatives:
6. Monitor histograms of activations and gradients

11.6 Example: Multi-Digit Number Recognition

Goodfellow's Working example: Transcription of Street view address number (2014): https://arxiv.org/abs/1312.6082

1. Begin with Data Collection: Car used to collect raw images and humans labeled them.
    - ML algorithms used for curation itself to detect the house numbers, prior to transcription
2. Choice of **performance metric** tied to business objective: maps are useful if we have high accuracy. Goal was to matching human level 98%. → this goal sacrificed **coverage**.
3. With accuracy held, coverage became target of optimization, eventually reaching 95% as ConvNets improved
4. Baseline? convNet with reLu, and use of **n different softmax** output to predict a sequence of numbers, each trained independently.
5. How to improve the coverage metric? Theoretically, the system shouldn't classify if the output sequence probability $p(y \mid x) < t$ for some threshold t. How to pick **p**?
    - Initially, p chosen to be the Product of all softmax probabilities.
    - Then developed a special output layer and cost function.

6. Then coverage reached 90%. Was problem **underfitting** or **overfitting**? **Train** and **test** error were almost identical (lots of data was available, tens of millions of labeled examples). 2 possible causes:
    - Underfitting
    - Problem with data
7. Proceeded to **visualize worst errors**: visualizing incorrect transcriptions which had high label confidence.
    - → mostly badly cropped images, ex: image of 1849 cropped to show 849
8. Solution: expand the width of the cropping region, improved coverage by 10%!
9. Remaining increase in performance came from **tuning hyperparameters**.
    - Making the **model larger** with restriction on computational cost

Overall project was successful with **hundreds of millions** of addressed were transcribed faster and at lower cost compared to human efforts.