ECBM E4040
Neural Networks and Deep Learning

Practical Methodology for Building Models

Zoran Kostić
Columbia University
Electrical Engineering Department
& Data Sciences Institute

COLUMBIA ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Outline

- Introduction
- Best Design Practices
- Performance Metrics
- Establishing a Working Architecture
- Data Preprocessing and Weight Initialization/Normalization

References and Acknowledgments

- **Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville, http://www.deeplearningbook.org/ , chapter 11.**
- **Lecture material by bionet group / Prof. Aurel Lazar (http://www.bionet.ee.columbia.edu/).**

DL Practical Methodology - Introduction

Applying deep learning techniques requires

- a good knowledge of existing algorithms in the literature and,
- a deep understanding of the principles underlying these algorithms.

DL Practical Methodology - Introduction

A DL 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.

DL Practical Methodology - Introduction

Practitioners need to decide whether to

SOME ARE MUTUALLY RELATED

- gather more data,
- increase or decrease model capacity,
- add or remove regularizing features,
- improve the optimization of a model,
- improve approximate inference in a model, or
- debug the software implementation.

DL Practical Methodology
Best Design Practices

(1) Determine your performance goals:

- specify the error (performance)metric, and
- estimate the target value for the error metric.

(2) Establish a working end-to-end pipeline of the architecture and asses the likelihood for achieving the targeted error metrics.

DL Practical Methodology
Best Design Practices

(3) Instrument the system to monitor and determine bottlenecks in performance.

(4) Diagnose which components are under-performing and why: overfitting, underfitting, or a defect in data or software.

DL Practical Methodology
Best Design Practices

(5) Repeatedly make incremental changes such as gathering new data, adjusting hyperparameters, or changing algorithms, based on specific findings from your instrumentation.

DL Practical Methodology
Performance Metrics - Error vs. Data

Error metric will guide all (re)modeling actions.

In deep learning, having A LOT of data is key:

- For most applications, it is impossible to achieve absolute zero error even if there is infinite training data and if one can recover the true probability distribution.

DL Practical Methodology
Performance Metrics - Error vs. Data

In reality - there is finite amount of training data.

One could possibly collect more data

- but must determine the value of reducing the error against the cost of collecting more data.

DL Practical Methodology
Performance Metrics - Choice of Error Target

Choosing a reasonable performance target:

- from previously published benchmark results,
- for new problems/applications, (bravely) establish the error rate that is necessary for an application to be safe, cost-effective, or appealing to consumers.

DL Practical Methodology
Performance Metric - What Should it be?

Sometimes it is much more costly to make one kind of a mistake than another.

For example, an e-mail spam detection system can make two kinds of mistakes:

- **incorrectly classifying a legitimate message as spam, vs.**
- **incorrectly allowing a spam message to appear in the inbox.**

DL Practical Methodology
Performance Metrics - Cost vs. Error

It is much worse to block a legitimate message than to allow a questionable message to pass through

- **similar to sending an innocent person to a prison.**

Rather than measuring the error rate of a spam classier, we may wish to measure some form of total cost, where the cost of blocking legitimate messages is higher than the cost of allowing spam messages.

DL Practical Methodology
Performance Metrics - Rare Events

At times we wish to train a binary classifier that is intended to detect some rare event. Consider a medical test for a rare disease.

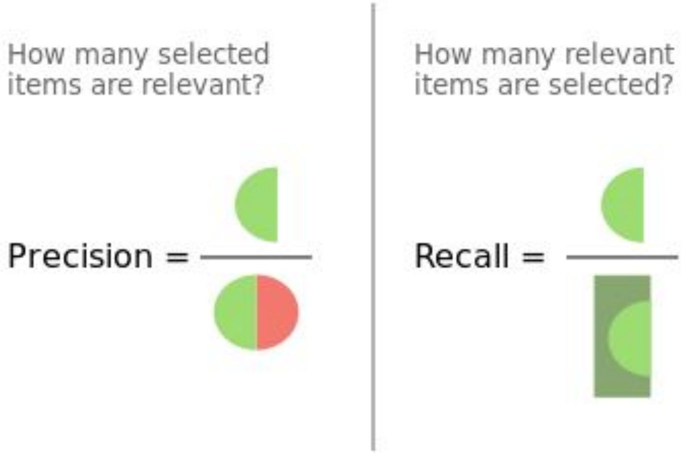
One can solve this problem by measuring the

- precision: the fraction of detections reported by the model that were correct; or
- recall: the fraction of true events that were detected.

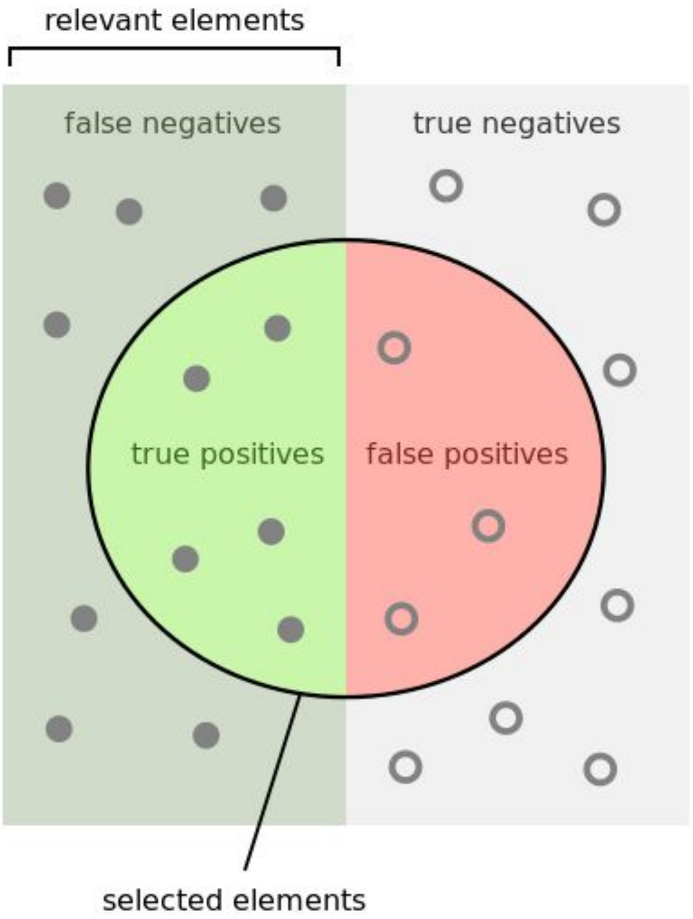
A detector that says no one has the disease would achieve perfect precision, but zero recall.

A detector that says everyone has the disease would achieve perfect recall, but precision equal to percentage of people with the disease.

Error: Precision vs. Recall


How many selected items are relevant?
Precision =
How many relevant items are selected?
Recall =

By Walber - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=36926283



Error: Precision vs. Recall

In a classification task:

a precision score of 1.0 for a class C means that every item labeled as belonging to class C does indeed belong to class C (but says nothing about the number of items from class C that were not labeled correctly) whereas

a recall of 1.0 means that every item from class C was labeled as belonging to class C (but says nothing about how many other items were incorrectly also labeled as belonging to class C).

per https://en.wikipedia.org/wiki/Precision_and_recall

Error: Precision vs. Recall

Usually, precision and recall scores are not discussed in isolation.

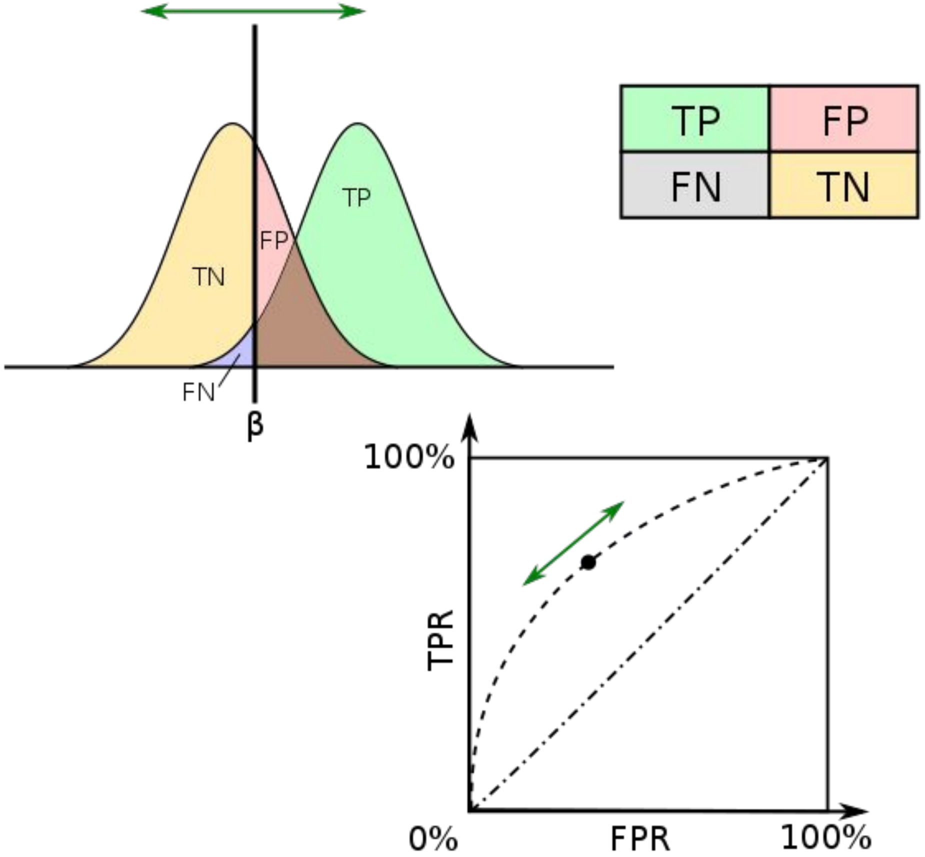
Instead, either values for one measure are compared for a fixed level at the other measure (e.g. precision at a recall level of 0.75) or both are combined into a single measure.

Several measures combine precision and recall: F-measure, Matthews correlation coefficient, Accuracy (a weighted arithmetic mean of Precision and Inverse Precision).

ROC curves: Recall vs. Inverse Recall (true positive rate vs. false positive rate), plotted against each other - provide a principled mechanism to explore operating point tradeoffs.

Error: ROC Curves Recall vs. Inverse Recall

Obtaining the ROC curve.

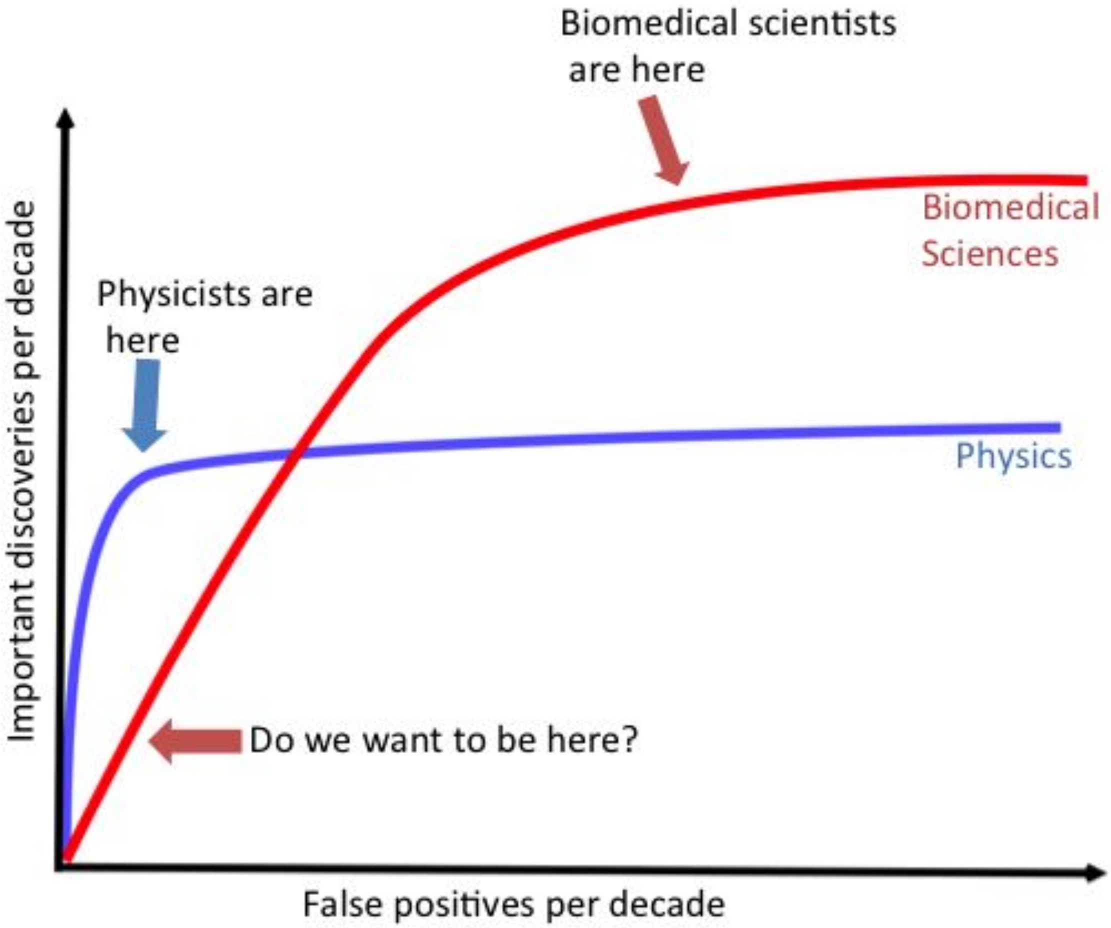


TP	FP
FN	TN

https://commons.wikimedia.org/wiki/File:ROC_curves_colors.svg

Error: ROC Curves Recall vs. Inverse Recall

Some realities of false positives.



Rafael Irizarry
https://simplystatistics.org/2013/08/01/the-roc-curves-of-science/

DL Practical Methodology
Performance Metrics - Refuse to Decide

In some applications, a machine learning system should refuse to make a decision.

This is useful when the machine learning algorithm can estimate that it is not confident in a decision, especially if a wrong decision can be harmful.

Establishing a Working Architecture

Starting Points

More Data

More Capacity

More Regularization

DL Practical Methodology
Establish a (Reasonable) Working Architecture

Starting points (but not exclusive):

- For supervised learning with fixed-size input vectors, use fully connected feedforward ANN.
- For input with known topological structure (e.g. image), use a convolutional network with ReLUs, Leaky ReLUs, PreLus, maxout.

DL Practical Methodology
Establish a (Reasonable) Working Architecture

Starting points (but not exclusive):

- If input or output is a sequence, use a gated recurrent net (LSTM or GRU).

DL Practical Methodology
Establish a (Reasonable) Working Architecture

If training set < tens of millions of examples:

- include mild regularization from the start.
- early stopping should be used almost always
- dropout is an excellents simple regularizer
- (batch) normalization
  - can reduce generalization errors and allow omission of dropout, due to the noise in the estimate of the statistics used to normalize each variable.

Should One Gather More Data?

It is often much better to gather more data than to improve the learning algorithm. How does one decide whether to gather more data?

(1) If the performance on the training set is not acceptable:

- **the learning algorithm is not using the training data that is already available, so there is no reason to gather more data.**
- **try increasing the size of the model by adding more layers or adding more hidden units to each layer.**
- **try improving the learning algorithm, for example by tuning the learning rate hyperparameter.**

Should One Gather More Data?

It is often much better to gather more data than to improve the learning algorithm. How does one decide whether to gather more data?

(2) If large models and carefully tuned optimization algorithms do not work well, then the problem might be the quality of the training data. The data may be too noisy or may not include the right inputs needed to predict the desired outputs. This suggests starting over, collecting cleaner data or collecting a richer set of features.

Should One Gather More Data?

How does one decide whether to gather more data?

(3) If the performance on the training set is acceptable, then measure the performance on a test set:

- **If the performance on the test set is also acceptable, then there is nothing left to be done.**
- **If test set performance is much worse than training set performance, then gathering more data is one of the most effective solutions.**

Should One Gather More Data?

How does one decide whether to gather more data?

The key considerations are:

- **the cost and feasibility of gathering more data,**
- **the cost and feasibility of reducing the test error by other means, and**
- **the amount of data that is expected to be necessary to improve test set performance significantly.**

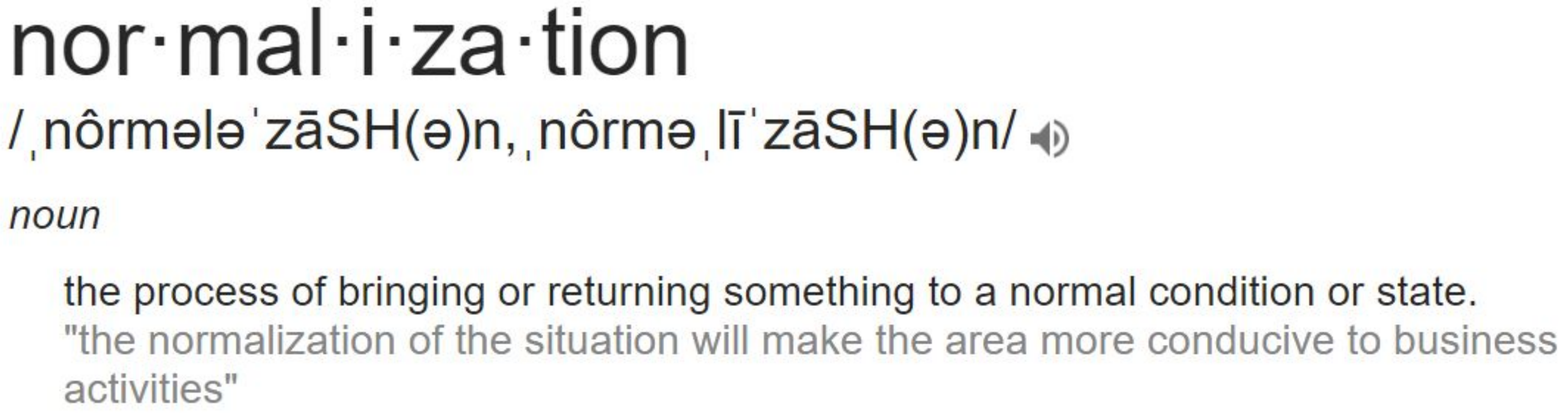
Data Preprocessing and Weight Init/Normalization

Normalization

Weight Scaling

Batch Normalization

Normalization



nor·mal·i·za·tion

/ˌnôrmələˈzāSH(ə)n, ˌnôrməˌlīˈzāSH(ə)n/

noun

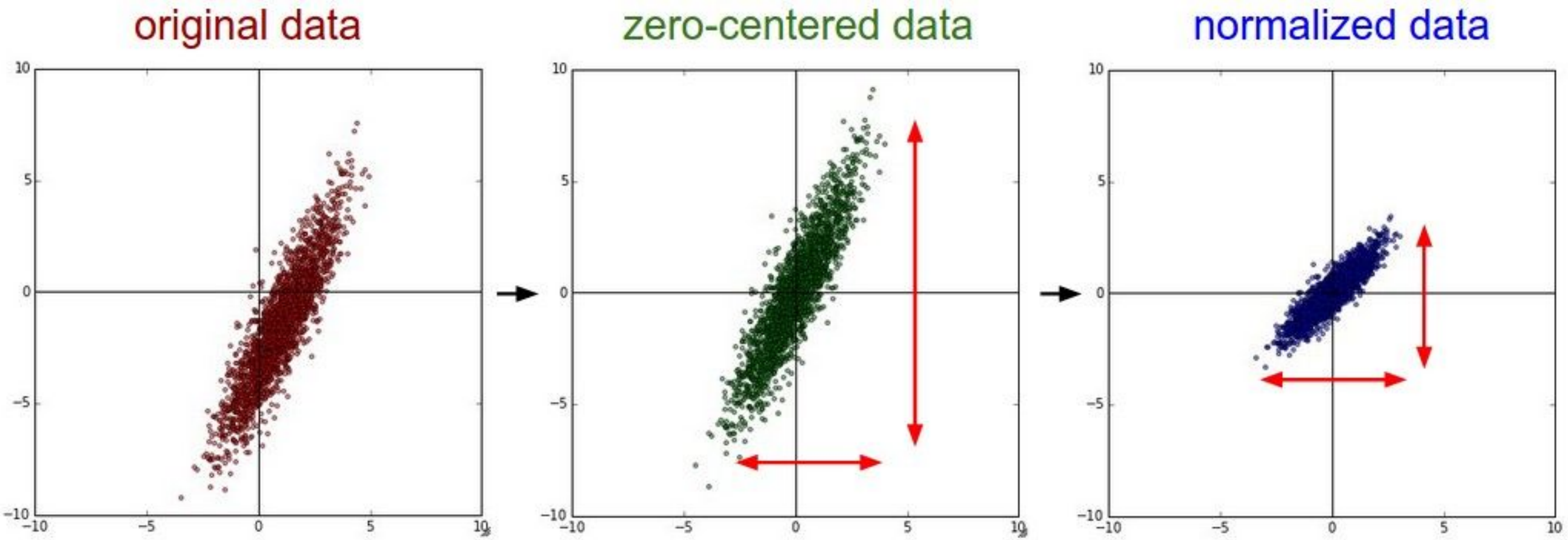
the process of bringing or returning something to a normal condition or state.
"the normalization of the situation will make the area more conducive to business activities"

Normalization

Casting the data to (approximately) the same (specific) range/scale.

- divide each dimension by its standard deviation, once it has been zero-centered.
- normalize each dimension so that the min and max along the dimension is -1 and 1 respectively.

Normalization



http://cs231n.github.io/neural-networks-2/

Normalization

- (Illegitimate) data outliers can be a problem - data scrubbing

Motivation for Normalization in ANNs

- A large number of coefficients: coefficient values can diverge wildly
- To give features “equal” chances (with caveat)
- Least significant bits of coefficient values do not change.
- Dynamic range of variables gets busted.
- SGD convergence can be improved in stability

Weight Initialization and Vanishing (or Exploding) Gradient Problem

Start with randomly initialized weights and biases in the network.

- **One should not immediately expect “proper” contribution**

In at least some deep neural networks, the gradient tends to get smaller as one moves backward through the hidden layers (vanishing gradient)

- **This means that neurons in the earlier layers learn much more slowly than neurons in later layers.**

More generally, it turns out that the gradient in deep neural networks is unstable, tending to either explode or vanish in earlier layers.

http://neuralnetworksanddeeplearning.com/chap5.html

Weight - Impact on Scaling the Inputs

If each layer, not properly initialized, scales input by k, the final scale would be k^L, where L is a number of layers.

- **Values of k > 1 lead to extremely large values of output layers**
- **k < 1 leads to a diminishing signal and gradient.**

Large weights lead to divergence via updates larger than the initial values, small initial weights do not allow the network to learn since the updates are of the order of 0.0001% per iteration.

Weights can be scaled/normalized.

Weight Initialization + Scaling Effect

Various formulas exist for scaling of weights as a function of the number of input and output channels, layers.

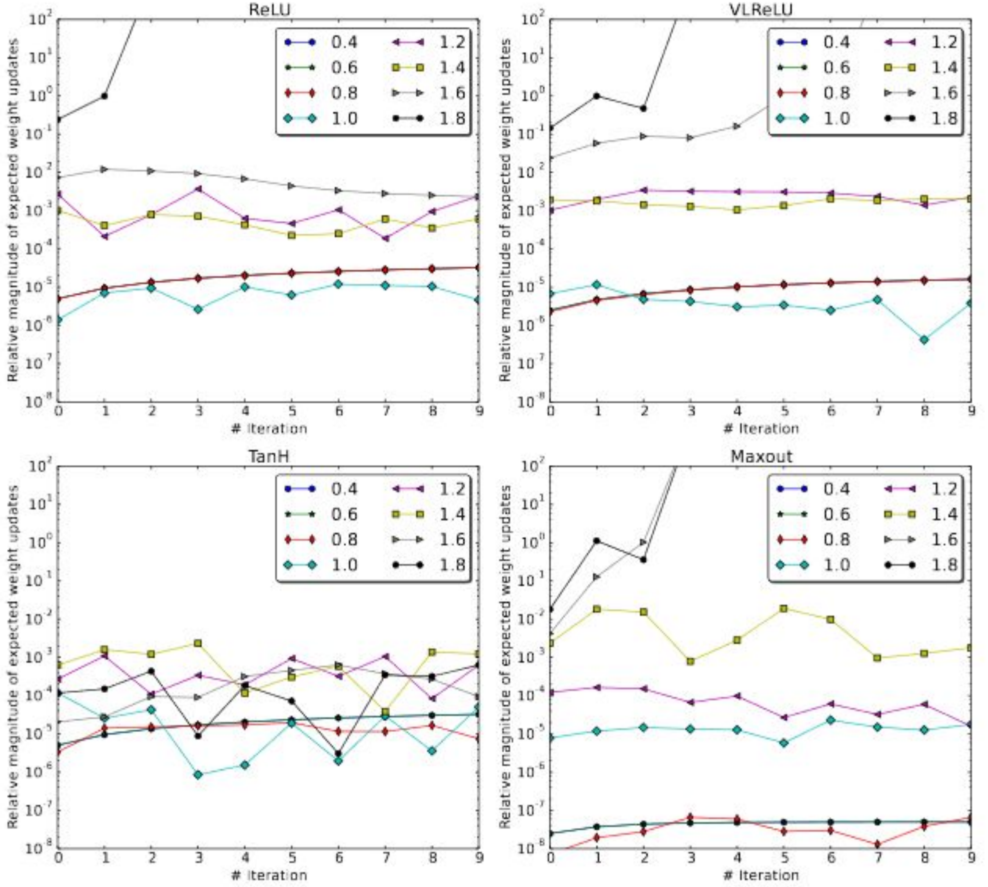


Figure 1: Relative magnitude of weight updates as a function of the training iteration for different weight initialization scaling after ortho-normalization. The values in the range 0.1% .. 1% lead to convergence, larger to divergence, for smaller, the network can hardly leave the initial state. Subgraphs show results for different non-linearities – ReLU (top left), VLReLU (top right), hyperbolic tangent (bottom left) and Maxout (bottom right).

Summary paper: Dmytro Mishkin, Jiri Matas,"All you need is a good init," https://arxiv.org/pdf/1511.06422.pdf

Batch Normalization in ANNs

Whitening the inputs to each layer

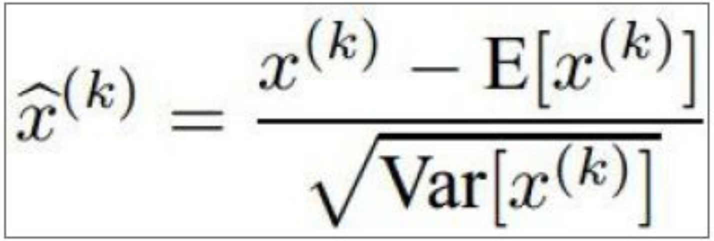
- Normalize each batch at each unit of each layer so that it has mean = 0 and variance = 1.
- After convolutional (or FC layer), and before nonlinearity, within a layer.
- If needed, scaling and bias can be added back as learned parameters

Sergey Ioffe, Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," https://arxiv.org/pdf/1502.03167.pdf

Batch Normalization in ANNs

Compute the empirical mean and variance independently for each dimension

Normalize



$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization in ANNs

But normalization disturbs the operation of nonlinearities

So provide placeholders to reintroduce scaling with

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

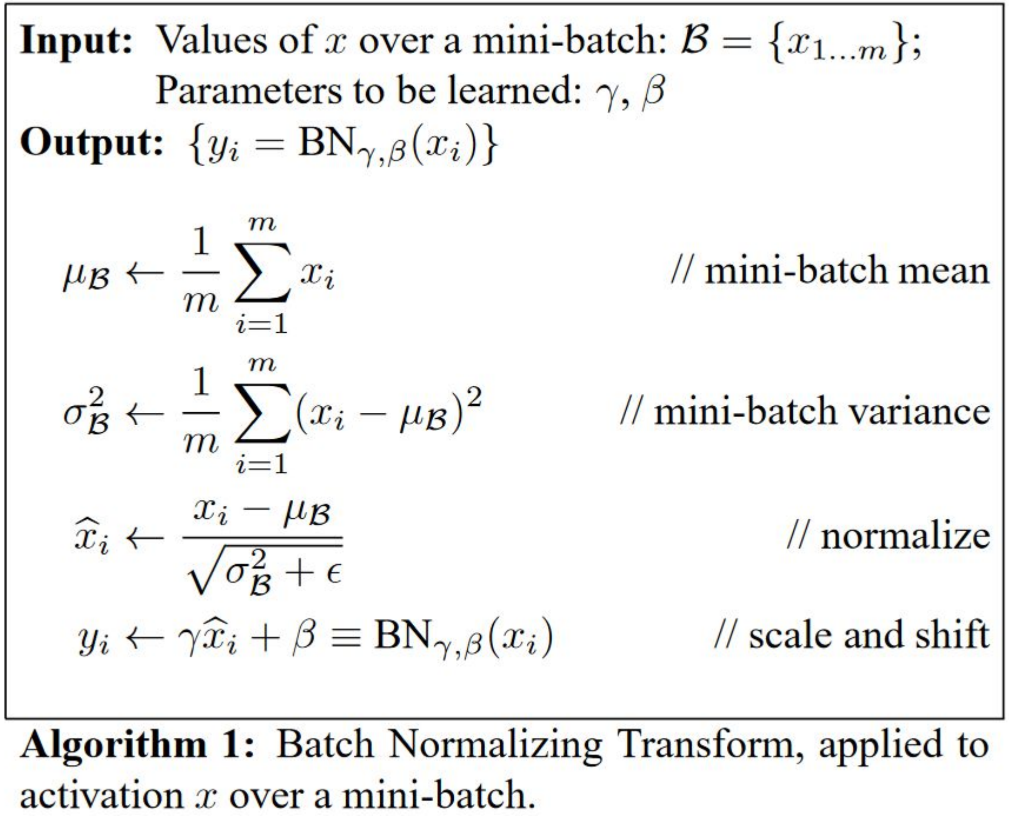
By

$$\boxed{y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}}$$

Batch Normalization

Training

- **Improves gradient flow**
- **Facilitates faster learning rates**
- **Makes correct initialization less relevant**



```
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
       Parameters to be learned: $\gamma$, $\beta$
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                              // mini-batch mean
    $\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$           // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$                     // normalize
    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$              // scale and shift
```

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch Normalization

- Improves gradient flow
- Facilitates faster learning rates
- Makes correct initialization less relevant

Test

At test time Batch Normalization layer functions differently:

- The mean/std are not computed based on the batch.
- Instead, a fixed empirical mean of activations during training is used.
- this mean can be computed during training using running averages

Batch Normalization in ANNs

Batch normalization becomes another layer of the ANN model

- The layer is subject to both forward and back propagation

Hyperparameters

Selection

Search

Manual

Automated

Hyperparameter Selection

Model hyperparameters affect

- the time and memory cost of running the algorithm;
- the quality of the model learned by the training process and its ability to infer correct results when deployed on new inputs.

Hyperparameter Search

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

Automated: reduces the need to understand the ideas above, but at an increased computational cost.

Hyperparameter Search - Manual

Key Hyperparameters

- number of hidden units
- learning rate
- convolution kernel width
- implicit zero padding
- weight decay coefficient
- dropout rate

Hyperparameter Tuning- Manual

One must understand the relationship between

- hyperparameters,
- training error,
- generalization error and,
- computational resources - memory & runtime

Hyperparameter Tuning- Manual

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 is constrained by 3 factors:

- the representational capacity of the model,
- the ability of the learning algorithm to successfully minimize the cost function used to train the model, and
- the degree to which the cost function and training procedure regularize the model.

Hyperparameter Tuning- Manual

A model with more layers and more hidden units per layer has higher representational capacity - it is capable of representing more complicated functions.

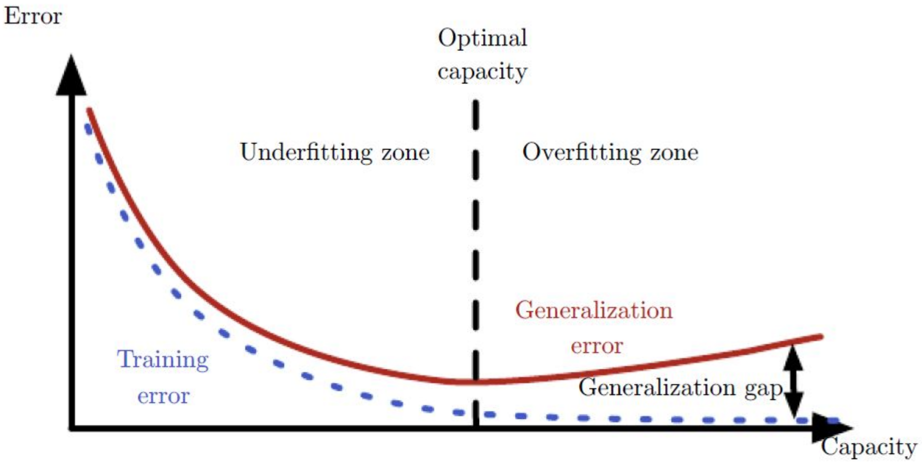
It can not necessarily actually learn all of these functions though, if the training algorithm cannot discover that certain functions do a good job of minimizing the training cost, or if regularization terms such as weight decay forbid some of these functions.

Hyperparameter Tuning- Manual

At one extreme, the hyperparameter value corresponds to low capacity, and generalization error is high because training error is high. This is the underfitting regime.

Overfitting: At the other extreme, the hyperparameter value corresponds to high capacity, and the generalization error is high because the gap between training and test error is high.

In the middle lies the optimal model capacity.



Typical relationship between capacity (horizontal axis) and both training (bottom curve, dotted) and generalization (or test) error (top curve, bold).

© ZK

Hyperparameter Tuning- Manual

Overfitting can occur when the value of a hyperparameter is:

- large: the number of hidden units in a layer, because increasing the number of hidden units increases the capacity of the model.
- small: the smallest allowable weight decay coefficient of zero corresponds to the greatest effective capacity of the learning algorithm.

Hyperparameter Tuning- The Learning Rate

The learning rate is perhaps the most important hyperparameter:

- **It controls the effective capacity of the model in a more complex ways than other hyperparameters.**
- **The effective capacity of the model is highest when the learning rate is correct for the optimization problem, not when the learning rate is especially large or especially small.**

Getting the learning rate to be “good” is a worthwhile effort.

Hyperparameter Tuning- The Learning Rate

The learning rate can be

- **too large: gradient descent can inadvertently increase rather than decrease the training error. In the idealized quadratic case, this occurs if the learning rate is at least twice as large as its optimal value.**
- **too small: training is not only slower, but may become permanently stuck with a high training error. This effect is poorly understood (it would not happen for a convex loss function).**

Manual Hyperparameter Tuning - Summary

If the error on the training set is higher than the target error rate, one must increase model capacity:

- If you are not using regularization and you are confident that the optimization algorithm is performing correctly, then to increase capacity, you must
  - add more layers to your network or
  - add more hidden units.
  - Unfortunately, this increases the computational costs associated with the model.

Manual Hyperparameter Tuning - Summary

If the error on the test set is higher than the target error rate, one can take two kinds of actions. The test error is the sum of the training error and the gap between training and test error. The optimal test error is found by trading off these quantities.

- Increase capacity: Neural networks typically perform best when the training error is very low (and thus, when capacity is high).
- Reduce the gap *without increasing training error faster than the gap decreases*. To reduce the gap, change regularization hyperparameters to reduce effective model capacity, such as by adding dropout or weight decay.

Manual Hyperparameter Tuning - Summary

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 tuning

Hyperparameter	Number of hidden units	Learning rate	Convolution kernel width	Implicit zero padding	Weight decay coefficient	Dropout rate
Increases capacity when …	increased	tuned optimally	increased	increased	decreased	decreased
Reason	Increasing the number of hidden units increases the representational capacity of the model.	An improper learning rate, whether too high or too low, results in a model with low effective capacity due to optimization failure	Increasing the kernel width increases the number of parameters in the model	Adding implicit zeros before convolution keeps the representation size large	Decreasing the weight decay coefficient frees the model parameters to become larger	Dropping units less often gives the units more opportunities to "conspire" with each other to fit the training set
Caveats	Increasing the number of hidden units increases both the time and memory cost of essentially every operation on the model.		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.	Increased time and memory cost of most operations.		

Hyperparameter tuning - links

https://cloud.google.com/ml-engine/docs/tensorflow/hyperparameter-tuning-overview

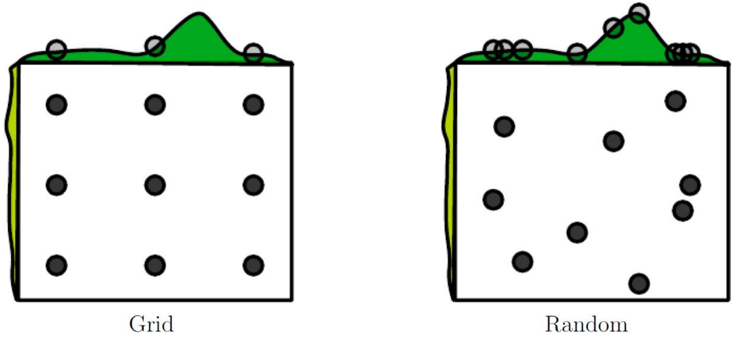
Automatic Hyperparameter Optimization Algorithms

Develop hyperparameter optimization algorithms that wrap a learning algorithm and choose its hyperparameters, thus hiding the hyperparameters of the learning algorithm from the user.

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.

However, these secondary hyperparameters are usually easier to choose, in the sense that acceptable performance may be achieved on a wide range of tasks using the same secondary hyperparameters for all tasks.

Hyperparemeters
Grid and Random Search



Comparison of grid search and random search. For illustration purposes two hyperparameters are displayed.

Observe that “vertical” parameter exhibits little variability -> low value.

Random search is much more informative.

Model-Based Hyperparameter Optimization

The search for good hyperparameters can be cast as an optimization problem, where the decision variables are the hyperparameters.

The cost to be optimized is the validation set error that results from training using these hyperparameters.

In simplified settings where it is feasible to compute the gradient of some differentiable error measure on the validation set with respect to the hyperparameters, one can simply follow this gradient.

Model-Based Hyperparameter Optimization

The search for good hyperparameters can be cast as an optimization problem, where the decision variables are the hyperparameters.

In most practical settings, the gradient is unavailable, either due to its high computation and memory cost, or due to hyperparameters having intrinsically non-differentiable interactions with the validation set error, as in the case of discrete-valued hyperparameters.

Currently, one can not unambiguously recommend Bayesian hyperparameter optimization as an established tool for achieving better deep learning results or for obtaining those results with less effort.

Example of Model Construction: Multi-Digit Number Recognition

Designing the deep learning components in Street View:

- **data collection: the cars collected the raw data and human operators provided labels;**
- **dataset curation, including using other machine learning techniques to detect the house numbers prior to transcribing them;**
- **transcription task: choice of performance metrics and desired values for these metrics; human-level, 98% accuracy at the cost of coverage.**
- **establish a baseline architecture consisting of a convolutional network with rectified linear units.**

Example of Model Construction: Multi-Digit Number Recognition

Iteratively refined the baseline, and tested each change:

- For coverage below 90%, instrumented the train and test set performance to determine whether the problem is underfitting or overfitting. Because train and test set error were similar, the problem was either due to underfitting or to a problem with the training data.
- Expanded the width of the crop region to be systematically wider than the address number detection system predicted. This single change added ten percentage points to the transcription system's coverage.
- The last few percentage points of performance came from adjusting hyperparameters. This mostly consisted of making the model larger while maintaining some restrictions on its computational cost

Backup Slides

Various