14620323
DEEP LEARNING

Practical Methodology

PENGAMPU

Dr. Fajar Astuti Hermawati, S.Kom.,M.Kom.

Bagus Hardiansyah, S.Kom.,M.Si

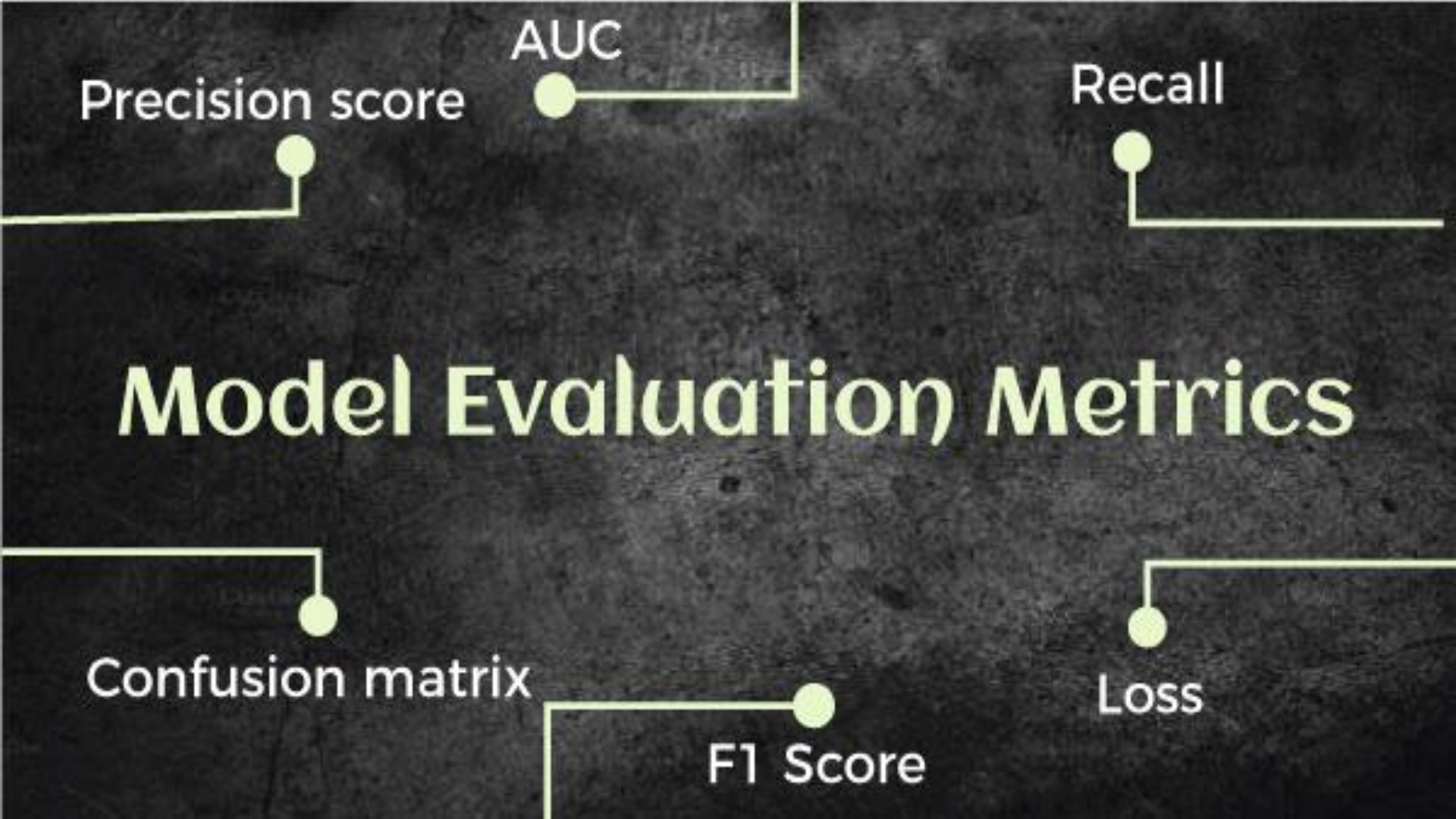
Andrey Kartika Widhy H., S.Kom., M.Kom.

Capaian Pembelajaran

- **Sub-CPMK-6**: Mampu merancang, menganalisis, dan menerapkan algoritma serta solusi perangkat lunak berbasis kecerdasan artifisial dalam pemelajaran mendalam untuk menyelesaikan permasalahan organisasi dan/atau masyarakat secara optimal [C6, A3, P3]

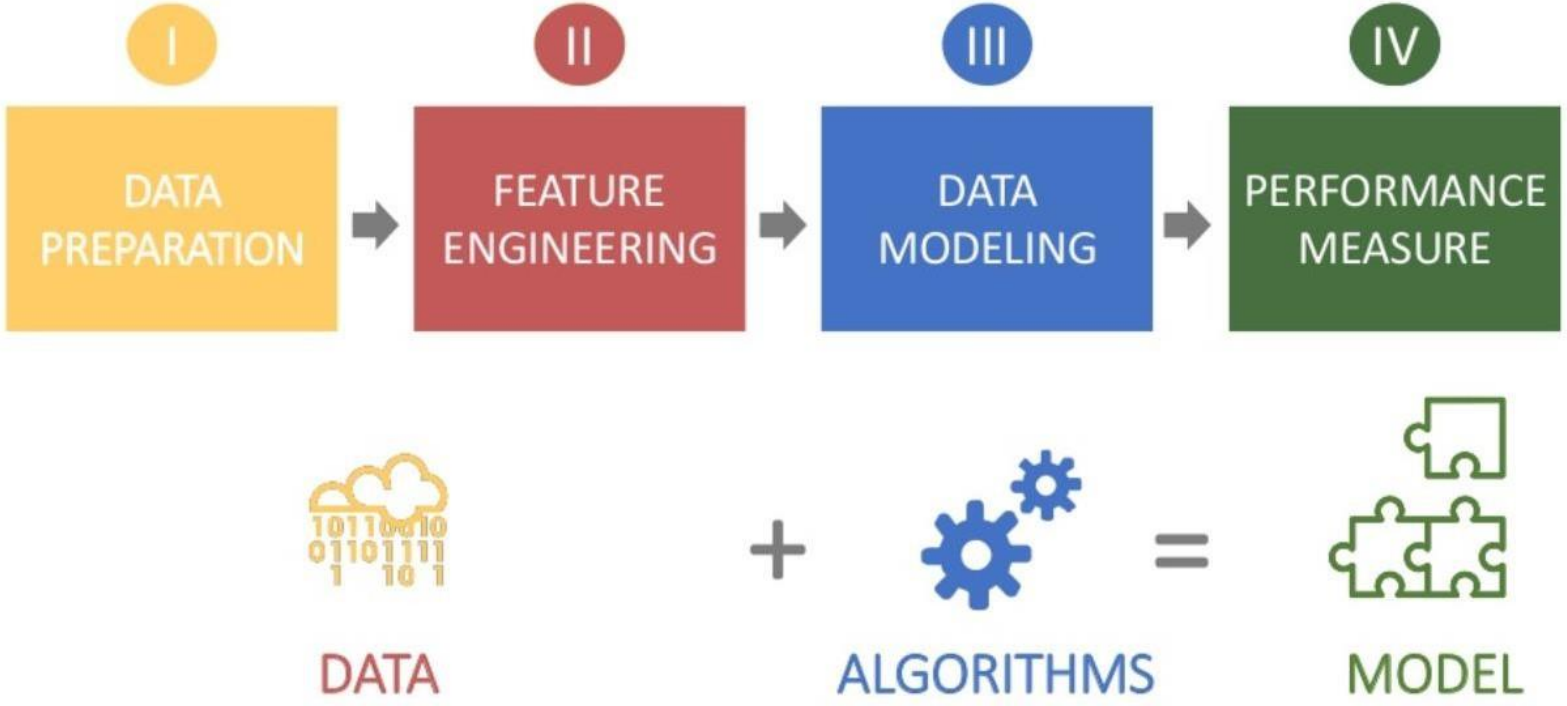
Bahan Kajian

- Model Evaluation Metrics:
  - Confusion Matrix
  - Accuracy
  - Precision
  - Recall or Sensitivity
  - Specificity
  - F1 Score
  - Log Loss
  - Area under the curve (AUC)
  - MAE – Mean Absolute Error
  - MSE – Mean Squared Error
- Parameter & Hyperparameter


AUC
Precision score
Recall
Model Evaluation Metrics
Confusion matrix
Loss
F1 Score

Performance Metrics

There are **4 steps** to build a machine learning model...



Performance Metrics

After doing the usual Feature Engineering, Selection, and of course, implementing a model and getting some output in forms of a probability or a class, the next step is to find out how effective is the model based on some metric using test datasets.

The metrics that you choose to evaluate your machine learning model is very important. Choice of metrics influences how the performance of machine learning algorithms is measured and compared.

Different metrics used:

- **Confusion Matrix**
- Accuracy
- Precision
- Recall or Sensitivity
- Specificity
- F1 Score
- Log Loss
- Area under the curve (AUC)
- MAE – Mean Absolute Error
- MSE – Mean Squared Error

Confusion Matrix

Just opposite to what the name suggests, confusion matrix is one of the most intuitive and easiest metrics used for finding the correctness and accuracy of the model. It is used for Classification problem where the output can be of two or more types of classes.

Let's say we are solving a classification problem where we are predicting whether a person is having cancer or not.

Let's give a label of to our target variable:

***1**: When a person is having cancer **0:** When a person is NOT having cancer.*

Alright! Now that we have identified the problem, the confusion matrix, is a table with two dimensions ("Actual" and "Predicted"), and sets of "classes" in both dimensions. Our Actual classifications are columns and Predicted ones are Rows.

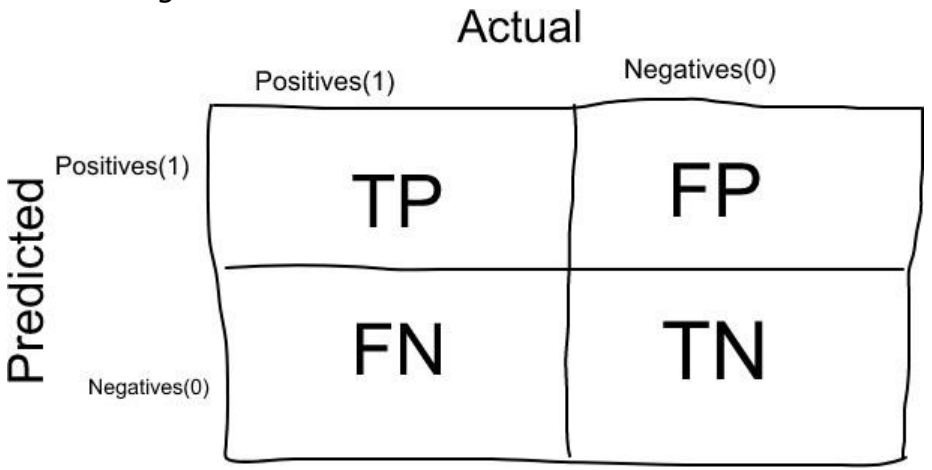
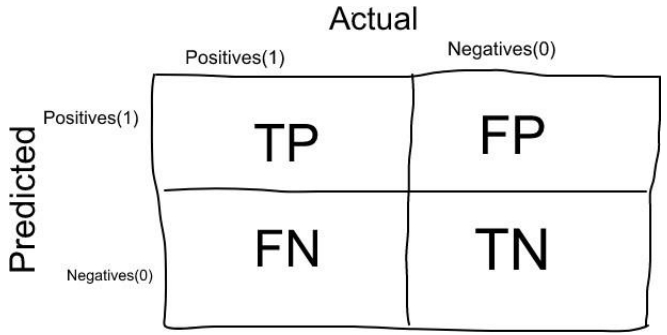


Fig. 1: Confusion Matrix

Confusion Matrix

The Confusion matrix in itself is not a performance measure as such, a lot of the performance metrics are based on Confusion Matrix and the numbers inside it.



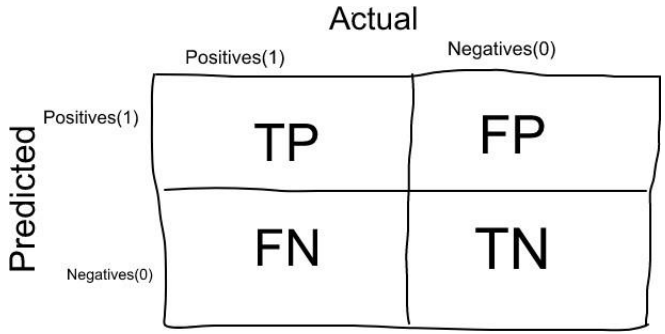
True Positives (TP) - True positives are the cases when the actual class of the data point was 1(True) and the predicted is also 1(True).

Ex: The case where a person is actually having cancer(1) and the model classifying his case as cancer(1) comes under True positive

True Negatives (TN) - True negatives are the cases when the actual class of the data point was 0(False) and the predicted is also 0(False)

Ex: The case where a person NOT having cancer and the model classifying his case as Not cancer comes under True Negatives.

Confusion Matrix



False Positives (FP) - False positives are the cases when the actual class of the data point was 0(False) and the predicted is 1(True). False is because the model has predicted incorrectly and positive because the class predicted was a positive one. (1)

Ex: A person NOT having cancer and the model classifying his case as cancer comes under False Positives.

False Negatives (FN) - False negatives are the cases when the actual class of the data point was 1(True) and the predicted is 0(False). False is because the model has predicted incorrectly and negative because the class predicted was a negative one. (0)

Ex: A person having cancer and the model classifying his case as No-cancer comes under False Negatives.

The ideal scenario that we all want is that the model should give 0 False Positives and 0 False Negatives.
But that's not the case in real life as **any model will NOT** be 100% accurate most of the times.

Confusion Matrix

When to minimize what?

We know that there will be some error associated with every model that we use for predicting the true class of the target variable. This will result in False Positives and False Negatives

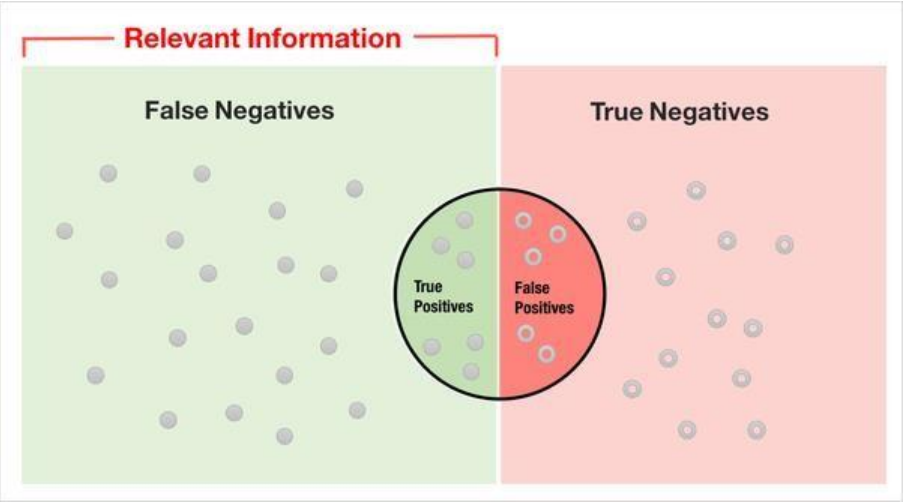
There's no hard rule that says what should be minimised in all the situations. It purely depends on the business needs and the context of the problem you are trying to solve. Based on that, we might want to minimise either False Positives or False negatives.

Confusion Matrix

When to minimize what?

Minimizing False Negatives

Let's say in our cancer detection problem example, out of 100 people, only 5 people have cancer. In this case, we want to correctly classify all the cancerous patients as even a very BAD model(Predicting everyone as NON-Cancerous) will give us a 95% accuracy. But, in order to capture all cancer cases, we might end up making a classification when the person actually NOT having cancer is classified as Cancerous. This might be okay as it is less dangerous than NOT identifying/capturing a cancerous patient since we will anyway send the cancer cases for further examination and reports. But missing a cancer patient will be a huge mistake as no further examination will be done on them.



Confusion Matrix

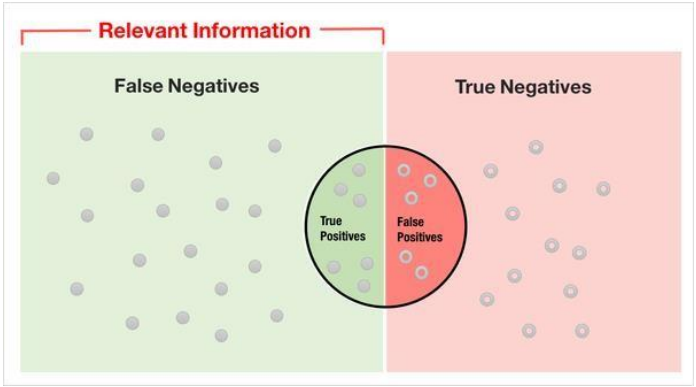
When to minimize what?

Minimizing False Positives:

For better understanding of False Positives, let's use a different example where the model classifies whether an email is spam or not.

Let's say that you are expecting an important email like hearing back from a recruiter or awaiting an admit letter from a university. Let's assign a label to the target variable and say,**1:** "Email is a spam" and **0:**"Email is not a spam".

Suppose the Model classifies that important email that you are desperately waiting for, as Spam(case of False positive). Now, in this situation, this is pretty bad than classifying a spam email as important or not spam since in that case, we can still go ahead and manually delete it and it's not a pain if it happens once a while. So in case of Spam email classification, minimising False positives is more important than False Negatives.



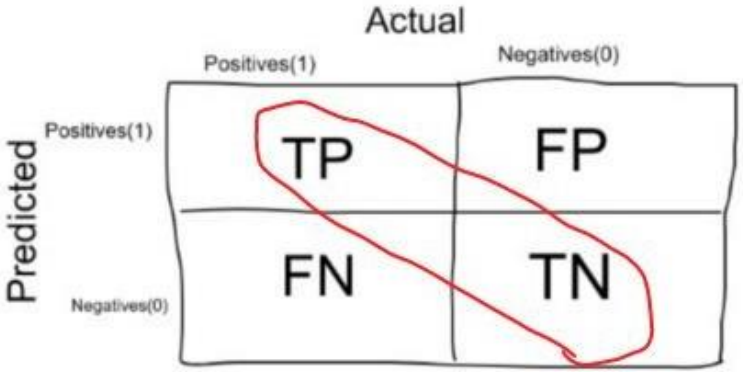
Performance Metrics

Different metrics used:

- Confusion Matrix
- **Accuracy**
- Precision
- Recall or Sensitivity
- Specificity
- F1 Score
- Log Loss
- Area under the curve (AUC)
- MAE – Mean Absolute Error
- MSE – Mean Squared Error

Accuracy

Accuracy in classification problems is the number of correct predictions made by the model over all kinds predictions made.



$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

In the Numerator, are our correct predictions (True positives and True Negatives) (marked as red in the fig above) and in the denominator, are the kind of all predictions made by the algorithm (right as well as wrong ones).

Accuracy

When to use Accuracy:

Accuracy is a good measure when the target variable classes in the data are nearly balanced.
Eg: 60% classes in our fruits images data are apple and 40% are oranges.

A model which predicts whether a new image is Apple or an Orange, 97% of times correctly is a very good measure in this example.

When not to use Accuracy:

Accuracy should never be used as a measure when the target variable classes in the data are a majority of one class.
Eg: In our cancer detection example with 100 people, only 5 people has cancer.

Let's say our model is very bad and predicts every case as No Cancer. In doing so, it has classified those 95 non-cancer patients correctly and 5 cancerous patients as Non-cancerous. Now even though the model is terrible at predicting cancer, The accuracy of such a bad model is also 95%.

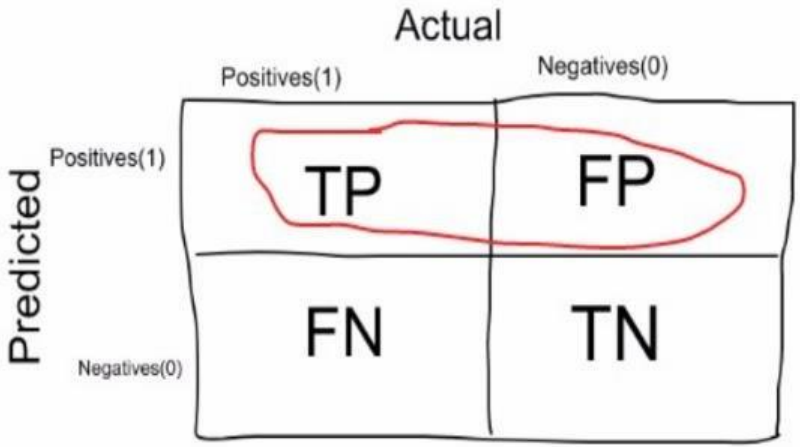
Performance Metrics

Different metrics used:

- Confusion Matrix
- Accuracy
- **Precision**
- Recall or Sensitivity
- Specificity
- F1 Score
- Log Loss
- Area under the curve (AUC)
- MAE – Mean Absolute Error
- MSE – Mean Squared Error

Precision

Precision is a measure that tells us what proportion of patients that we diagnosed as having cancer, actually had cancer. The predicted positives (People predicted as cancerous are TP and FP) and the people actually having a cancer are TP.



$$\text{Precision} = \frac{TP}{TP + FP}$$

In our cancer example with 100 people, only 5 people have cancer. Let's say our model is very bad and predicts every case as **Cancer**. Since we are predicting everyone as having cancer, our denominator (True positives and False Positives) is 100 and the numerator, person having cancer and the model predicting his case as cancer is 5. So in this example, we can say that **Precision** of such model is 5%.

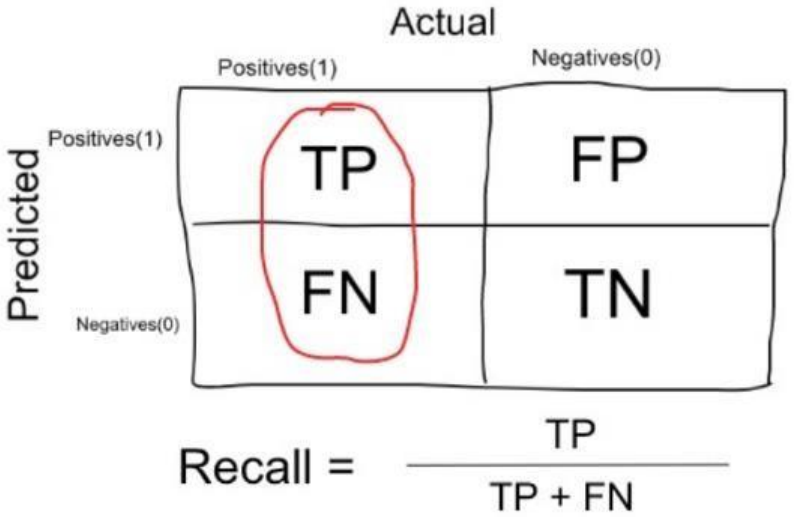
Performance Metrics

Different metrics used:

- Confusion Matrix
- Accuracy
- Precision
- **Recall or Sensitivity**
- Specificity
- F1 Score
- Log Loss
- Area under the curve (AUC)
- MAE – Mean Absolute Error
- MSE – Mean Squared Error

Recall or Sensitivity

Recall is a measure that tells us what proportion of patients that actually had cancer was diagnosed by the algorithm as having cancer. The actual positives (People having cancer are TP and FN) and the people diagnosed by the model having a cancer are TP. (Note: FN is included because the Person actually had a cancer even though the model predicted otherwise).



$$\text{Recall} = \frac{TP}{TP + FN}$$

Ex: In our cancer example with 100 people, 5 people actually have cancer. Let's say that the model predicts every case as cancer. So our denominator(True positives and False Negatives) is 5 and the numerator, person having cancer and the model predicting his case as cancer is also 5(Since we predicted 5 cancer cases correctly). So in this example, we can say that the **Recall** of such model is 100%. And Precision of such a model(As we saw above) is 5%

Accuracy

When to use Precision and when to you recall:

It is clear that recall gives us information about a classifier's performance with respect to false negatives (how many did we miss), while precision gives us information about its performance with respect to false positives(how many did we caught)

Precision is about being precise. So even if there is only one cancer case, and we captured it correctly, then we are 100% precise.

Recall is not so much about capturing cases correctly but more about capturing all cases that have "cancer" with the answer as "cancer". So if we simply always say every case as "cancer", we have 100% recall.

So basically if we want to focus more on minimising False Negatives, we would want our Recall to be as close to 100% as possible without precision being too bad and if we want to focus on minimising False positives, then our focus should be to make Precision as close to 100% as possible.

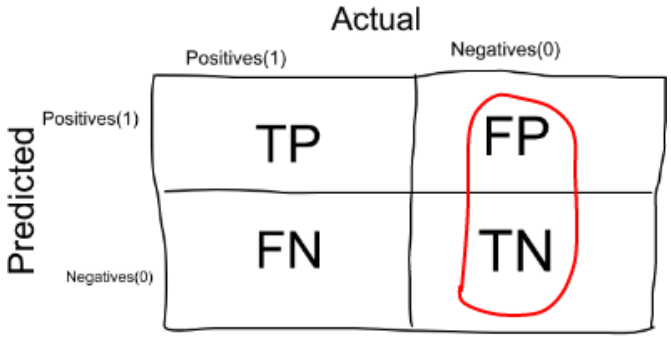
Performance Metrics

Different metrics used:

- Confusion Matrix
- Accuracy
- Precision
- Recall or Sensitivity
- **Specificity**
- F1 Score
- Log Loss
- Area under the curve (AUC)
- MAE – Mean Absolute Error
- MSE – Mean Squared Error

Specificity

Specificity is a measure that tells us what proportion of patients that did NOT have cancer, were predicted by the model as non-cancerous. The actual negatives (People actually NOT having cancer are FP and TN) and the people diagnosed by us not having cancer are TN.



$$\text{Specificity} = \frac{TN}{TN + FP}$$

Specificity is the exact opposite of Recall.

Ex: In our cancer example with 100 people, 5 people actually have cancer. Let's say that the model predicts every case as cancer.

So our denominator(False positives and True Negatives) is 95 and the numerator, person not having cancer and the model predicting his case as no cancer is 0 (Since we predicted every case as cancer). So in this example, we can that that **Specificity** of such model is 0%.

Performance Metrics

Different metrics used:

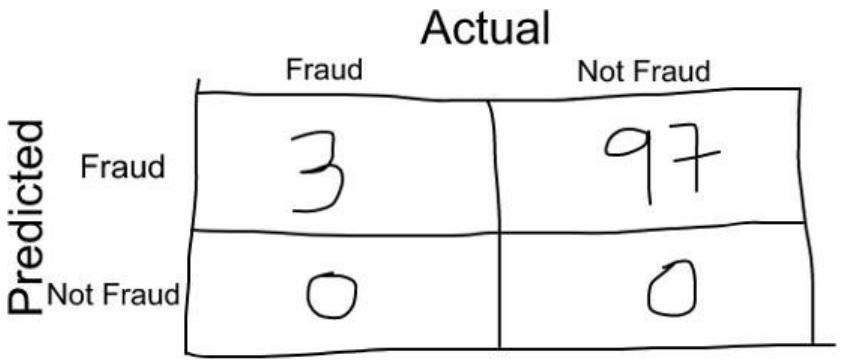
- Confusion Matrix
- Accuracy
- Precision
- Recall or Sensitivity
- Specificity
- **F1 Score**
- Log Loss
- Area under the curve (AUC)
- MAE – Mean Absolute Error
- MSE – Mean Squared Error

F1 Score

We don’t really want to carry both Precision and Recall in our pockets every time we make a model for solving a classification problem. So it’s best if we can get a single score that kind of represents both Precision(P) and Recall(R).

One way to do that is simply taking their arithmetic mean. i.e (P + R) / 2 where P is Precision and R is Recall. But that’s pretty bad in some situations.

Why? - Suppose we have 100 credit card transactions, of which 97 are legit and 3 are fraud and let’s say we came up a model that predicts everything as fraud. Precision and Recall for the example is shown in the fig below.



$$\text{Precision} = \frac{3}{100} = 3\%$$

$$\text{Recall} = \frac{3}{3} = 100\%$$

F1 Score

Now, if we simply take arithmetic mean of both, then it comes out to be nearly 51%. We shouldn't be giving such a moderate score to a terrible model since it's just predicting every transaction as fraud.

So, we need something more balanced than the arithmetic mean and that is harmonic mean.

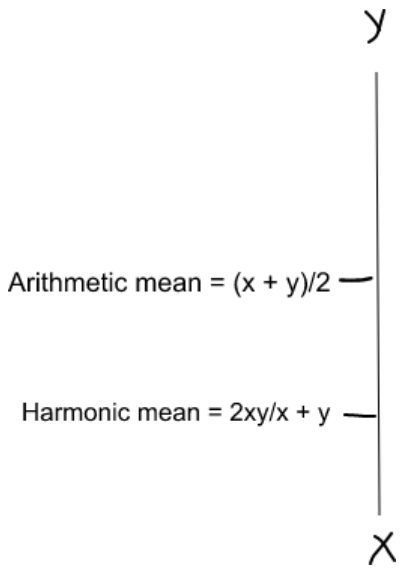
The Harmonic mean is given by the formula shown in the figure on the bottom right.

Harmonic mean is kind of an average when x and y are equal. But when x and y are different, then it's closer to the smaller number as compared to the larger number.

For our previous example, F1 Score = Harmonic Mean(Precision, Recall)

F1 Score = 2 * Precision * Recall / (Precision + Recall) = 2*3*100/103 = 5%

So if one number is really small between precision and recall, the F1 Score kind of raises a flag and is more closer to the smaller number than the bigger one, giving the model an appropriate score rather than just an arithmetic mean.



Performance Metrics

Different metrics used:

- Confusion Matrix
- Accuracy
- Precision
- Recall or Sensitivity
- Specificity
- F1 Score
- **Log Loss**
- Area under the curve (AUC)
- MAE – Mean Absolute Error
- MSE – Mean Squared Error

Log Loss

Logarithmic Loss or Log Loss, works by penalising the false classifications.

It works well for multi-class classification. When working with Log Loss, the classifier must assign probability to each class for all the samples.

Suppose, there are N samples belonging to M classes, then the Log Loss is calculated as below :

$$LogarithmicLoss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} * \log(p_{ij})$$

where,

y_ij, indicates whether sample i belongs to class j or not
p_ij, indicates the probability of sample i belonging to class j

Log Loss has no upper bound and it exists on the range [0, ∞). Log Loss nearer to 0 indicates higher accuracy, whereas if the Log Loss is away from 0 then it indicates lower accuracy.

In general, minimising Log Loss gives greater accuracy for the classifier.

Performance Metrics

Different metrics used:

- Confusion Matrix
- Accuracy
- Precision
- Recall or Sensitivity
- Specificity
- F1 Score
- Log Loss
- **Area under the curve (AUC)**
- MAE – Mean Absolute Error
- MSE – Mean Squared Error

AUC – Area Under the ROC Curve

Idea of Thresholding:

Logistic regression returns a probability. You can use the returned probability "as is" (for example, the probability that the user will click on this ad is 0.00023) or convert the returned probability to a binary value (for example, this email is spam)

A logistic regression model that returns 0.9995 for a particular email message is predicting that it is very likely to be spam. Conversely, another email message with a prediction score of 0.0003 on that same logistic regression model is very likely not spam.
However, what about an email message with a prediction score of 0.6?

In order to map a logistic regression value to a binary category, you must define a **classification threshold** (also called the **decision threshold**).

A value above that threshold indicates "spam"; a value below indicates "not spam." It is tempting to assume that the classification threshold should always be 0.5, but thresholds are problem-dependent, and are therefore values that you must tune.

AUC – Area Under the ROC Curve

ROC Curve

An **ROC curve** (**receiver operating characteristic curve**) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

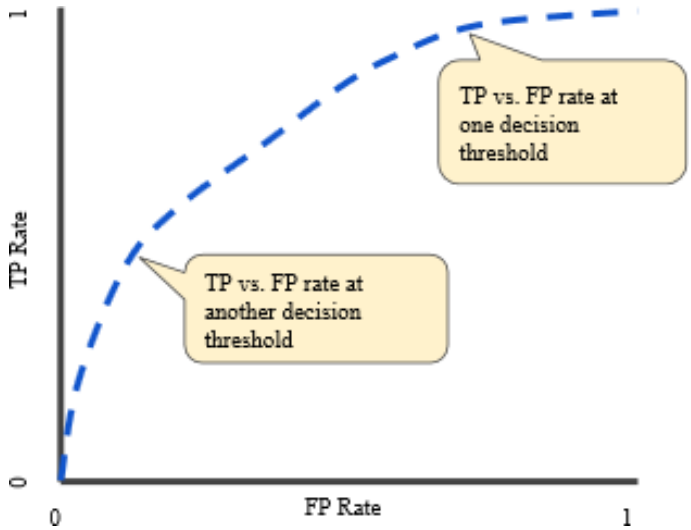
True Positive Rate (**TPR**) is a synonym for recall and is therefore defined as follows:

$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (**FPR**) is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

An ROC curve plots TPR vs. FPR at different classification thresholds. (as shown in the right fig)



AUC – Area Under the ROC Curve

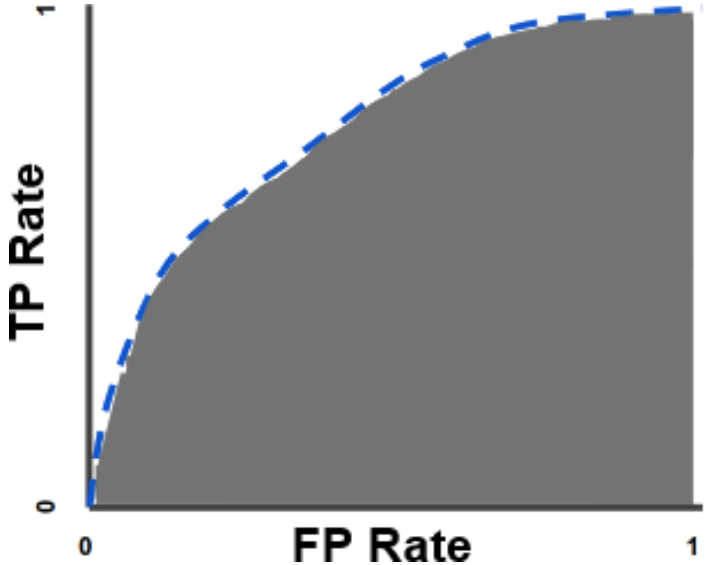
AUC Value

Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.

To compute the points in an ROC curve, we could evaluate a logistic regression model many times with different classification thresholds, but this would be inefficient. Fortunately, there's an efficient, sorting-based algorithm that can provide this information for us, called **AUC**.

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1)

AUC provides an aggregate measure of performance across all possible classification thresholds.



AUC – Area Under the ROC Curve

AUC Value

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

AUC is desirable for the following two reasons:

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

However, both these reasons come with caveats, which may limit the usefulness of AUC in certain use cases:

- **Scale invariance is not always desirable.** For example, sometimes we really do need well calibrated probability outputs, and AUC won't tell us about that.
- **Classification-threshold invariance is not always desirable.** In cases where there are wide disparities in the cost of false negatives vs. false positives, it may be critical to minimize one type of classification error. For example, when doing email spam detection, you likely want to prioritize minimizing false positives (even if that results in a significant increase of false negatives). AUC isn't a useful metric for this type of optimization.

Performance Metrics

Different metrics used:

- Confusion Matrix
- Accuracy
- Precision
- Recall or Sensitivity
- Specificity
- F1 Score
- Log Loss
- Area under the curve (AUC)
- **MAE – Mean Absolute Error**
- MSE – Mean Squared Error

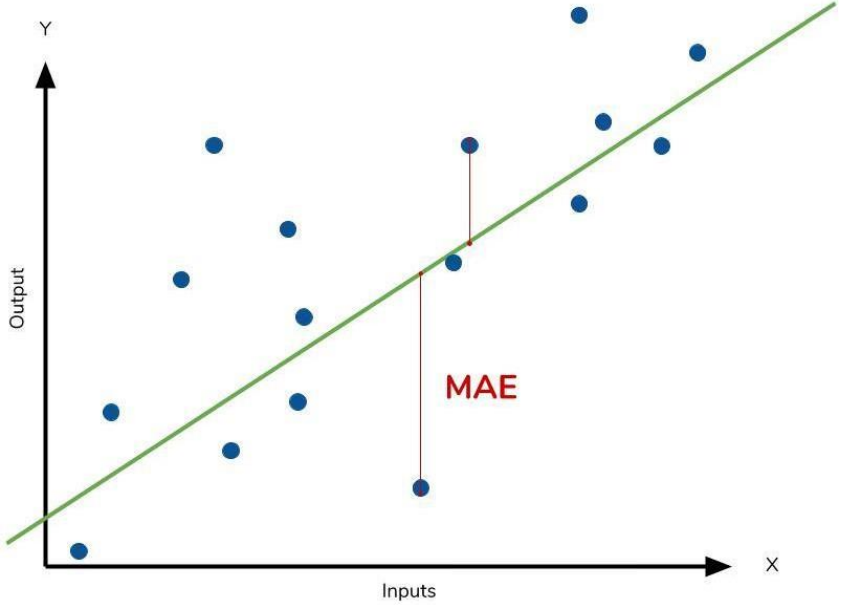
Mean Absolute Error

Mean Absolute Error is the average of the difference between the original values and the predicted values.

It gives us the measure of how far the predictions were from the actual output. However, they don't gives us any idea of the direction of the error i.e. whether we are under predicting the data or over predicting the data.

Mathematically, it is represented as :

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$



Performance Metrics

Different metrics used:

- Confusion Matrix
- Accuracy
- Precision
- Recall or Sensitivity
- Specificity
- F1 Score
- Log Loss
- Area under the curve (AUC)
- MAE – Mean Absolute Error
- **MSE – Mean Squared Error**

Mean Squared Error

Mean Squared Error(MSE) is quite similar to Mean Absolute Error, the only difference being that MSE takes the average of the **square** of the difference between the original values and the predicted values.

As, we take square of the error, the effect of larger errors become more pronounced then smaller error, hence the model can now focus more on the larger errors.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Instead of MSE, we generally use RMSE, which is equal to the square root of MSE.

Taking the square root of the average squared errors has some interesting implications for RMSE. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors.

This means the RMSE should be more useful when large errors are particularly undesirable.

Mean Squared Error

The three tables below show examples where MAE is steady and RMSE increases as the variance associated with the frequency distribution of error magnitudes also increases.

CASE 1: Evenly distributed errors

ID	Error	|Error|	Error^2
1	2	2	4
2	2	2	4
3	2	2	4
4	2	2	4
5	2	2	4
6	2	2	4
7	2	2	4
8	2	2	4
9	2	2	4
10	2	2	4

MAE	RMSE
2.000	2.000

CASE 2: Small variance in errors

ID	Error	|Error|	Error^2
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	3	3	9
7	3	3	9
8	3	3	9
9	3	3	9
10	3	3	9

MAE	RMSE
2.000	2.236

CASE 3: Large error outlier

ID	Error	|Error|	Error^2
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0
10	20	20	400

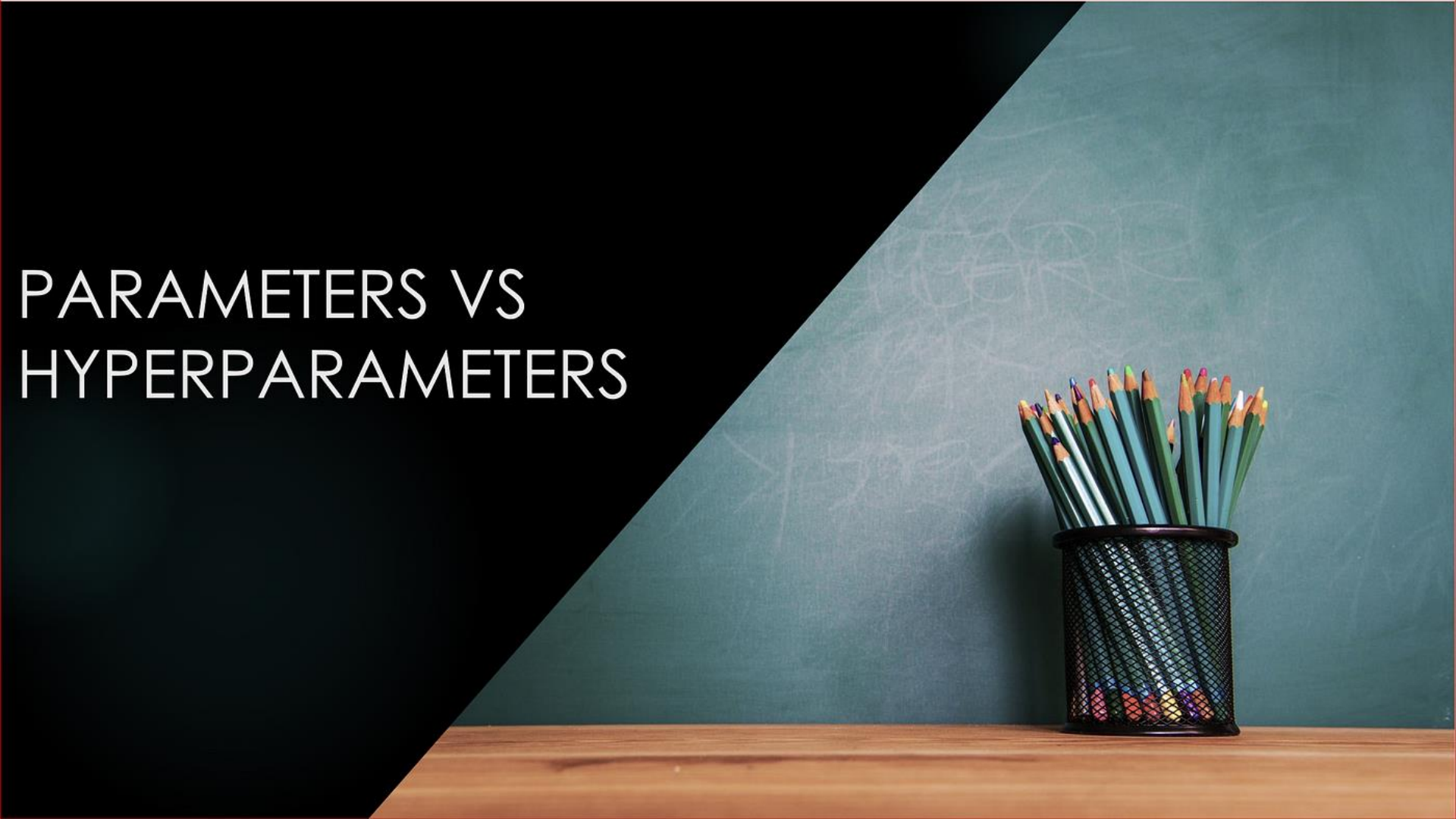
MAE	RMSE
2.000	6.325

Mean Squared Error

Conclusion:

1. RMSE has the benefit of penalizing large errors more so can be more appropriate in some cases, for example, if being off by 10 is more than twice as bad as being off by 5. But if being off by 10 is just twice as bad as being off by 5, then MAE is more appropriate.
2. From an interpretation standpoint, MAE is clearly the winner. RMSE does not describe average error alone and has other implications that are more difficult to tease out and understand.
3. One distinct advantage of RMSE over MAE is that RMSE avoids the use of taking the absolute value, which is undesirable in many mathematical calculations

PARAMETERS VS HYPERPARAMETERS



Parameters and Hyperparameters

- Model Parameters
  - These are the entities learned via training from the training data. They are not set manually by the designer.
  - With respect to deep neural networks, the model parameters are:
    - Weights
    - Biases
- Model Hyperparameters
  - These are parameters that govern the determination of the model parameters during training
    - They are typically set manually via heuristics
    - They are tuned during a cross-validation phase (discussed later)
  - Examples:
    - Learning rate, number of layers, number of units in each layer, many others to be discussed . . .

Machine Learning Models

- What is a model?
  - For purposes of this discuss, the Model comprises the hyperparameters characterizing the neural network. Because hyperparameters govern the parameters of the underlying network, implicitly the model comprises:
    - The topology of the deep neural network (i.e., layers and units and their interconnection)
    - The learned parameters (i.e., the learned weights and biases)
- The model is dependent upon the hyperparameters because the hyperparameters determine the learned parameters (weights and biases).
- Hyperparameters include:
  - Learning Rate
  - Number of Layers
  - Number of Units in each Layer
  - Activation Functions
  - Capacity – e.g., polynomial degree
  - Etc.

Model Selection

- To optimize the inference time behavior (the goal of training), a process known as model selection is performed
  - Model selection amounts to selecting an optimal set hyperparameters that yield the best performance of the neural network
  - The hyperparameters are tuned using an iterative process of either:
    - Validation
    - Cross-Validation
  - Many models may be evaluated during the validation/cross-validation phase and the optimal model is selected
    - The optimal model is then evaluated on the test dataset to determine how well it performs on data never seen before

Training, Validation and Test Sets

- Training Set – Data set used to learn the optimal model parameters (weights, biases)
- Validation ("Dev") Set – Data set used to perform model selection (tuning of hyperparameters)
  - Used to estimate the generalization error of the training allowing for the hyperparameters to be updated accordingly
  - Cross-validation set is a variant on validation set (discussed later)
- Test Set – Data set used to assess the fully trained model
  - A fully trained model is the model that has been selected via hyperparameter tuning and has been subsequently been trained to determine the optimal weights and biases (e.g., using backpropagation)
  - The test set is not used to perform further training
- Why separate test and validation sets?
  - The error rate estimate of the final model on validation data will be biased (smaller than the true error rate) since the validation set is used to select the model

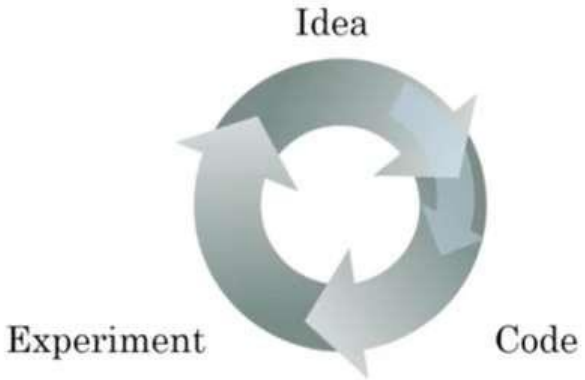
Train, Validation ("Dev") and Test Sets

Workflow:

1) Train algorithms on training set
2) Use dev set to see which of many different trained models performs
3)Once final model has been found, evaluate it on the test set to get an unbiased estimate on ho algorithm performs

The Design Process of Deep Learning

- Iteration
  - Tools do not exist to determine the optimal hyperparameters (e.g., learning rate, # of layers, # of units in each layer, etc.) a priori
    - Instead, the optimal choices are determined by experimentation and iteration



From Andrew Ng – Coursera Deep Learning Course

Train, Validation ("Dev") and Test Sets Split

- Because we live in era of big data (data is much more prevalent), the trend is to apportion a m percentage of data to the dev and test sets (e.g., may have 1*10^6 examples or even more)
- In the past the split was typically: 60%/20%/20%
- Trend: Now a typical example may be 98%/1%/1%

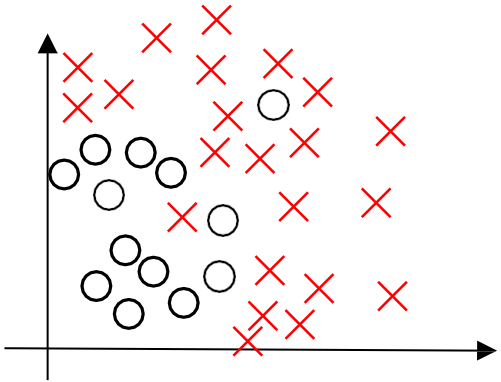
Mismatch

- Dev and Test sets should come from same distribution

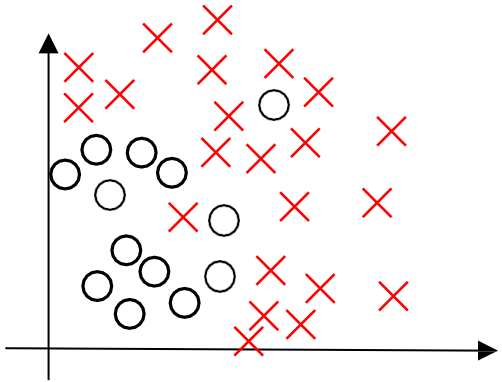
Bias and Variance

- Bias – Error from erroneous assumptions in the learning algorithm. High bias can cause al algorithm to miss the relevant relations between features and target outputs (underfitting).
- Variance – Error from the sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).
- Tradeoff – Goal is to choose a model that accurately captures the regularities in the training data but also generalizes well to unseen data. Difficult to do both simultaneously.
  - Models with low bias are typically more complex (e.g., higher order regression polynomials) enabling them to represent the training set more accurately. However, in doing so, these models may in fact capture the noise inherent in the training set making their predictions less accurate on the training set (unseen data).
  - Models with high bias (low-order polynomials) many not be able to capture the higher order (non-linear) behavior of the daa.

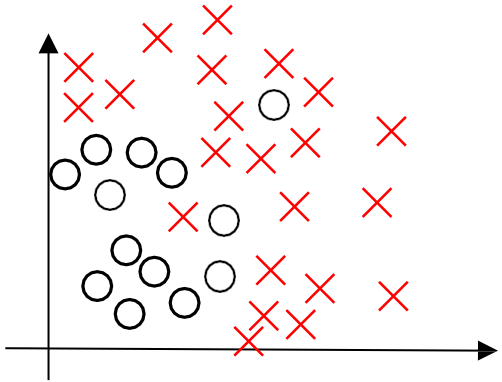
Bias and Variance Pictures



high bias



“just right”



high variance

From Coursera Deep Learning – Andrew

Capacity

- A model's capacity is its ability to fit a wide variety of functions
  - Models with low capacity may fail to fit the training set (underfitting)
  - Models with high capacity can overfit by learning properties of the training set that do not serve well o on the test set such as the noise

Bias Variance Decomposition

- Training set of points: $x^{(1)}, x^{(2)}..x^{(m)}$
- Assume function $y = f(x) + \varepsilon$ with noise ε with 0 mean and variance σ^2
- Goal: Find function $\hat{f}(x)$ that approximates the true function $f(x)$ so as to minimize $(y - \hat{f}(x))^2$
- Note: $Var[X] = E[X^2] - E[X]^2$
- Note: $E[\varepsilon]$=0, $E[y]$=$E[f + \varepsilon]$=$E[f] = f$
- Note: $Var[y] = E[(y - E[y])^2] = E[(y - f)^2]$= $E[(f + \varepsilon - f)^2]$=$\sigma^2$

Bias Variance Decomposition

Average test error over large ensemble of training sets

$$E\left[\left(y-\hat{f}\right)^2\right] = E\left[y^2+\hat{f}^2-2y\hat{f}\right]$$

$$= E[y^2] + E[\hat{f}^2] - 2E[y\hat{f}]$$

$$= Var[y] + E[y]^2 + Var[\hat{f}] + E[\hat{f}]^2 - 2fE[\hat{f}]$$

$$= Var[y] + Var[\hat{f}] + (f^2 - 2fE[\hat{f}] + E[\hat{f}^2])$$

$$= Var[y] + Var[\hat{f}] + (f - E[\hat{f}])^2$$

$$E\left[\left(y-\hat{f}\right)^2\right] = \sigma^2 + Var[\hat{f}] + Bias[\hat{f}]^2$$

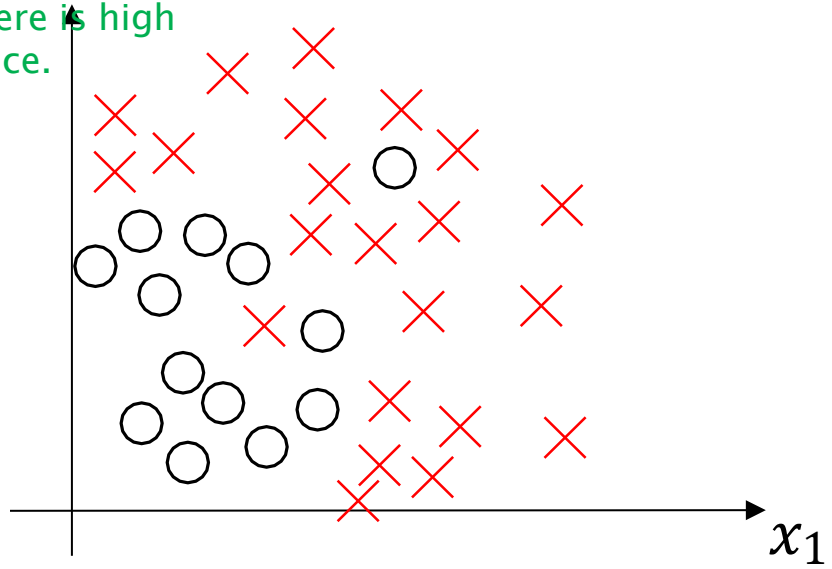
Analysis Of Bias-Variance Decomposition

$$E[(y - \hat{f})^2] = Var[\hat{f}] + Bias[\hat{f}]^2 + Var(\epsilon)$$

- What is variance?
  - Amount that $\hat{f}$ would change if estimated it with a different training set
  - Ideally, $\hat{f}$ should not vary much between training sets
  - With high variances, small perturbations in training set result in large changes in $\hat{f}$
- What is bias?
  - Bias is the error introduced by approximating real-life problems, which may be very complex.
    - For example, the world is highly non-linear and choosing a linear model will result in high bias.
- In order to minimize the expected test error, need to minimize both bias and variance

High Bias and High Variance

This means in some regions there is high bias while in others high variance.



From Coursera Deep Learning – Andrew

Bias-Variance Analysis

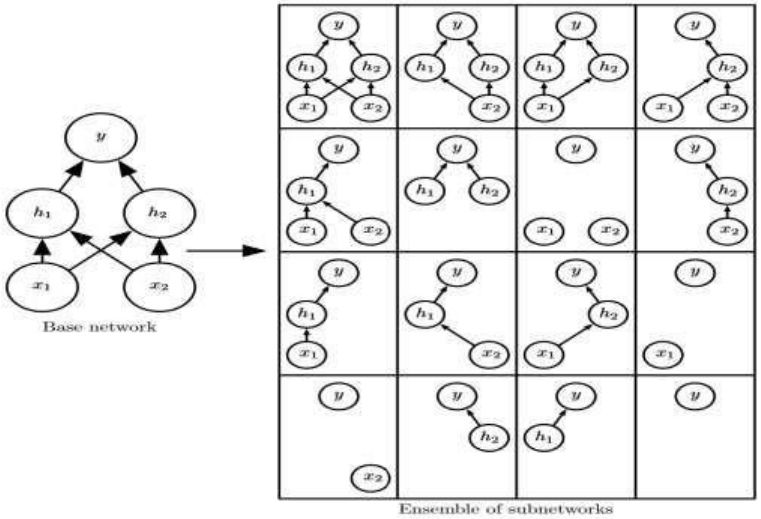
1. Analyze Training Set Performance (potential underfit)
   - If low accuracy on training set data, may have a bias problem
2. Analyze Development/Validation Set Performance
   - If low accuracy on development set, may have a variance problem
3. Bias/Variance Tradeoff Less Of An Issue In Big Data Era
   4. Bias can be driven down by introducing more capacity (larger network)
      1. Training a larger network almost never hurts so long as regularization is employed.
   2. Variance can be driven down by obtaining more training data

Potential Solutions

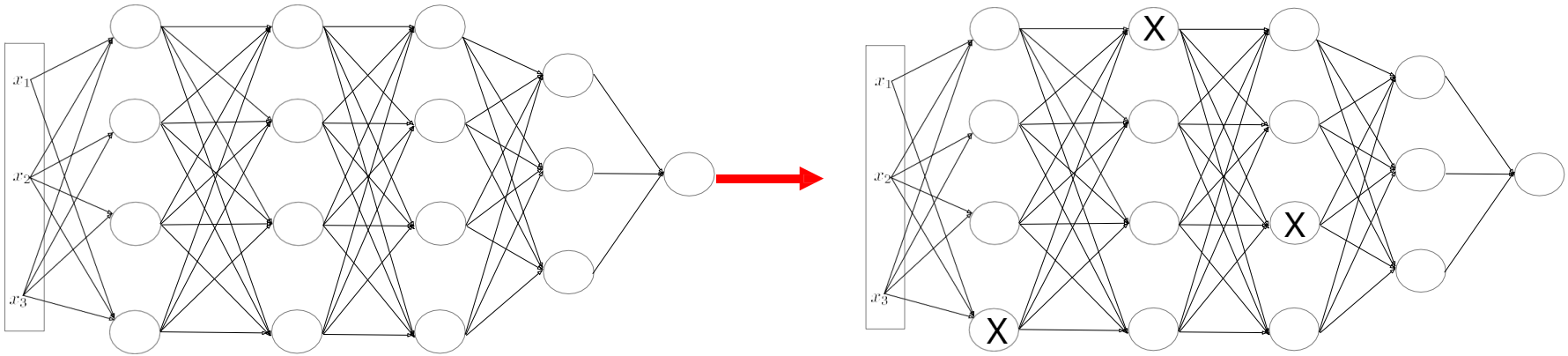
High Bias	High Variance
Try network with more capacity (e.g., more hidden units per layer)	Obtain more training data
Train longer	Regularization (to be discussed)
Try different architecture	Try different architecture

Dropout Regularization

Dropout trains an ensemble consisting of all subnetworks that can be constructed by removing nonoutput units from an underlying base network.

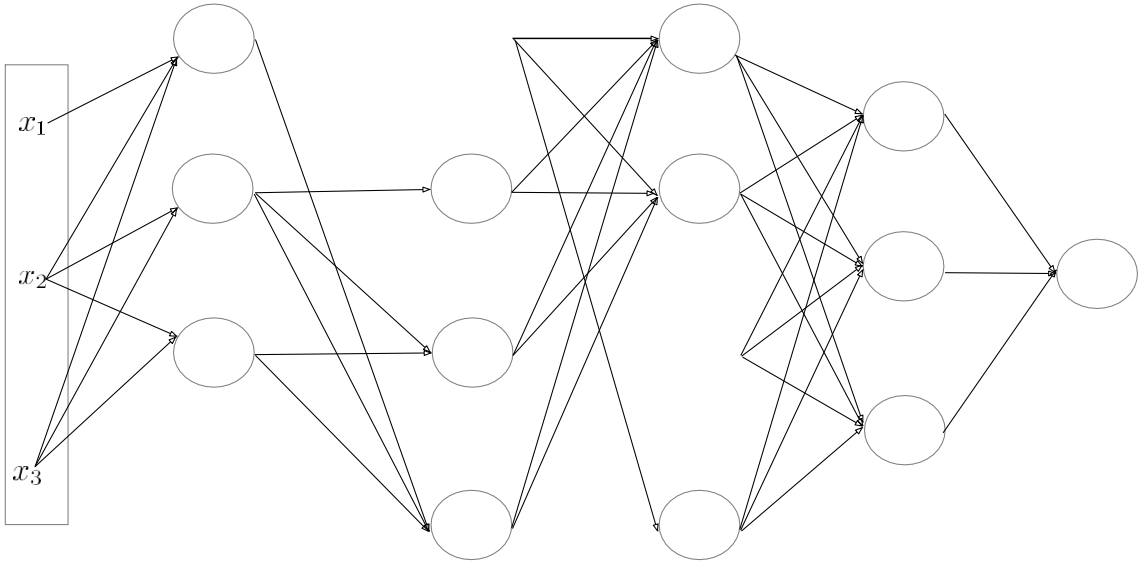


Dropout



For each training example, randomly kill hidden units at training time only

Dropout Effect Example



Dropout As Regularization

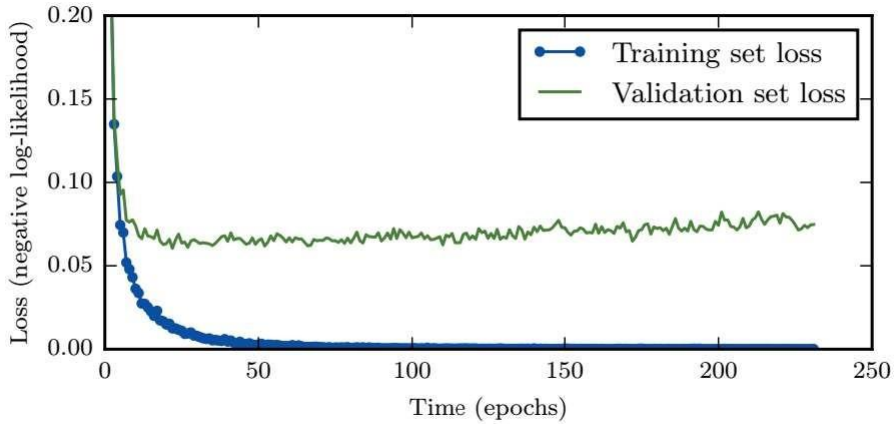
- How does dropout have a regularizing effect?
  - By ”killing” units, it reduces the capacity of the network, reducing potential for overfitting
  - By ”killing” units, it makes any neuron unable to “rely” on any one input feature. So, rather than betting heavily on any one input feature, weights are spread out among input neurons
    - That is, it reduces the weights proportionally among input nodes
  - Dropout used heavily in computer vision b/c almost never have enough data
  - Key point: Cost function is not well defined b/c killing off nodes on each iteration
    - Plotting cost function is thus not meaningful if dropout is employed
      - Solution: Turn off dropout and plot cost function to make sure it is working. Then, turn dropout on.

Data Augmentation As Regularization

- Data Augmentation
  - Synthetically transform or distort data to generate fake training examples

Early Stopping

- When training large models with sufficient representational capacity to overfit, training error steadily decreases over time but validation error falls but eventually begins to rise



Early Stopping

- Idea: Stop training when validation error is at a minimum (although the cost function is not)
  - Every time the validation set error improves, store the latest set of model parameters
    - When training terminates, return the model parameters with the lowest validation set error and the hyperparameter indicating the number of iterations
  - View the number of training iterations as a hyperparameter r to be tuned
    - Problem: This technique breaks orthogonality of training and validation (hyperparameter selection) phases
      - Complicates optimization

Normalization Of Inputs

- Subtract Mean
  - $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$
  - $x = x - \mu$
- Normalize Variance
  - $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} x^2$
  - $x = \frac{x}{\sigma^2}$

RMSProp

- Solution: Compute exponentially weighted average of the derivatives
  - In vertical direction this will zero out the oscillations because average to close to 0
  - In horizontal direction (because no oscillations) – all derivatives in same direction
- $S_{dw} = \beta S_{dw} + (1-\beta) dW^2$
- $S_{db} = \beta V_{db} + (1-\beta) db^2$
- $w = w - \alpha \frac{dW}{\sqrt{S_{dw+\epsilon}}}$
- $b = b - \alpha \frac{d}{\sqrt{S_{d\,b+}}}$

RMS terms control damping of oscillations. Larger values cause oscillations to be damped more. Can therefore use a faster learning rate and reduce risk of oscillations. Epsilon term is a small value that insures numerical stability (i.e., no divide by 0).

Adam (Adaptive Moment Estimation) Combines Momentum with RSMProp

$$V_{dw} = 0, S_{dw} = 0, Vdb = 0, S_{db} = 0$$

On iteration t:

Compute $d\mathbf{W}, dB$ using current mini-batch

$$V_{dW} = \beta_1 V_{dw} + (1 - \beta_1) dW \qquad V_{db} = \beta_1 V_{db} + (1 - \beta_1) db$$
$$S_{dW} = \beta_2 S_{dw} + (1 - \beta_2) dW^2 \qquad S_{db} = \beta_2 S_{db} + (1 - \beta_2) db$$

Momentum
RMSProp

$$V_{dw}^{Corrected} = \frac{V_{dw}}{(1 - \beta_1^t)} \qquad V_{db}^{Corrected} = \frac{V_{db}}{(1 - \beta_1^t)}$$
$$S_{dW}^{Corrected} = \frac{S_{dW}}{(1 - \beta_2^t)} \qquad S_{db}^{Corrected} = \frac{S_{db}}{(1 - \beta_2^t)}$$

Bias Correction

$$W = W - \alpha \frac{V_{dW}^{Corrected}}{\sqrt{S_{dW}^{Corrected}} + \epsilon} \qquad b = b - \alpha \frac{V_{db}^{Corrected}}{\sqrt{S_{db}^{Corrected}} + \epsilon}$$

Parameter Update

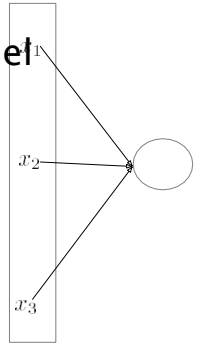
Adam Hyperparameters

- α (needs to be tuned)
- β_1 default from paper $= 0.9$
- β_2 default from paper $= 0.999$
- ϵ default from paper $= 10^{-8}$

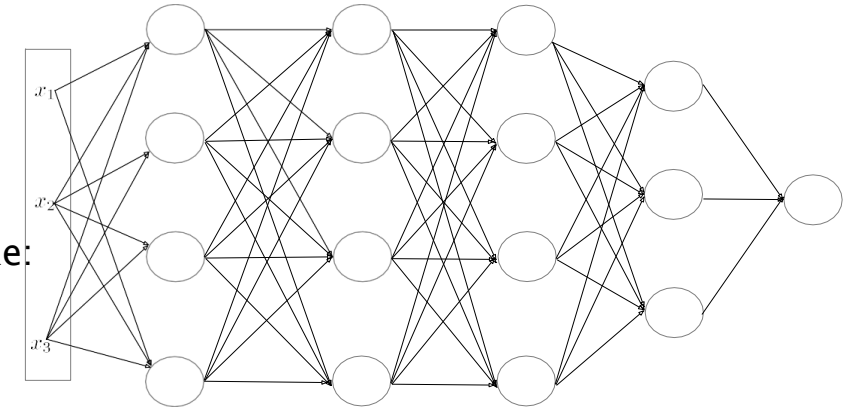
Batch Normalization Motivation

- Subtract Mean
  - $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$
  - $x = x - \mu$
- Normalize Variance
  - $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} x^2$
  - $x = \frac{x}{\sigma^2}$

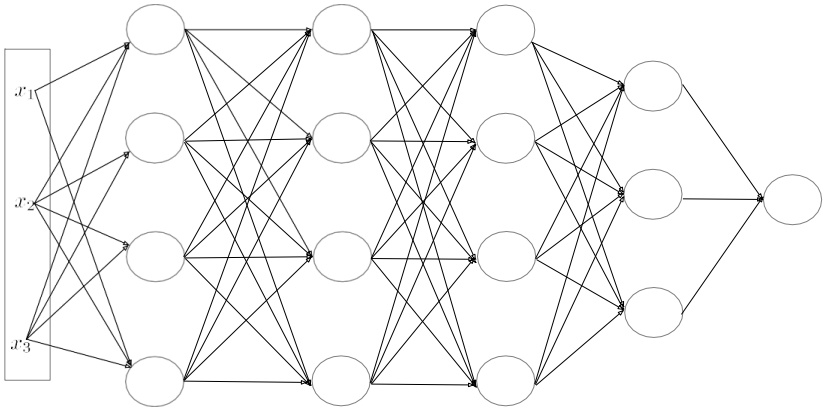
This works fine for simple model like:



But what about a deeper model like:



Batch Motivation Concept



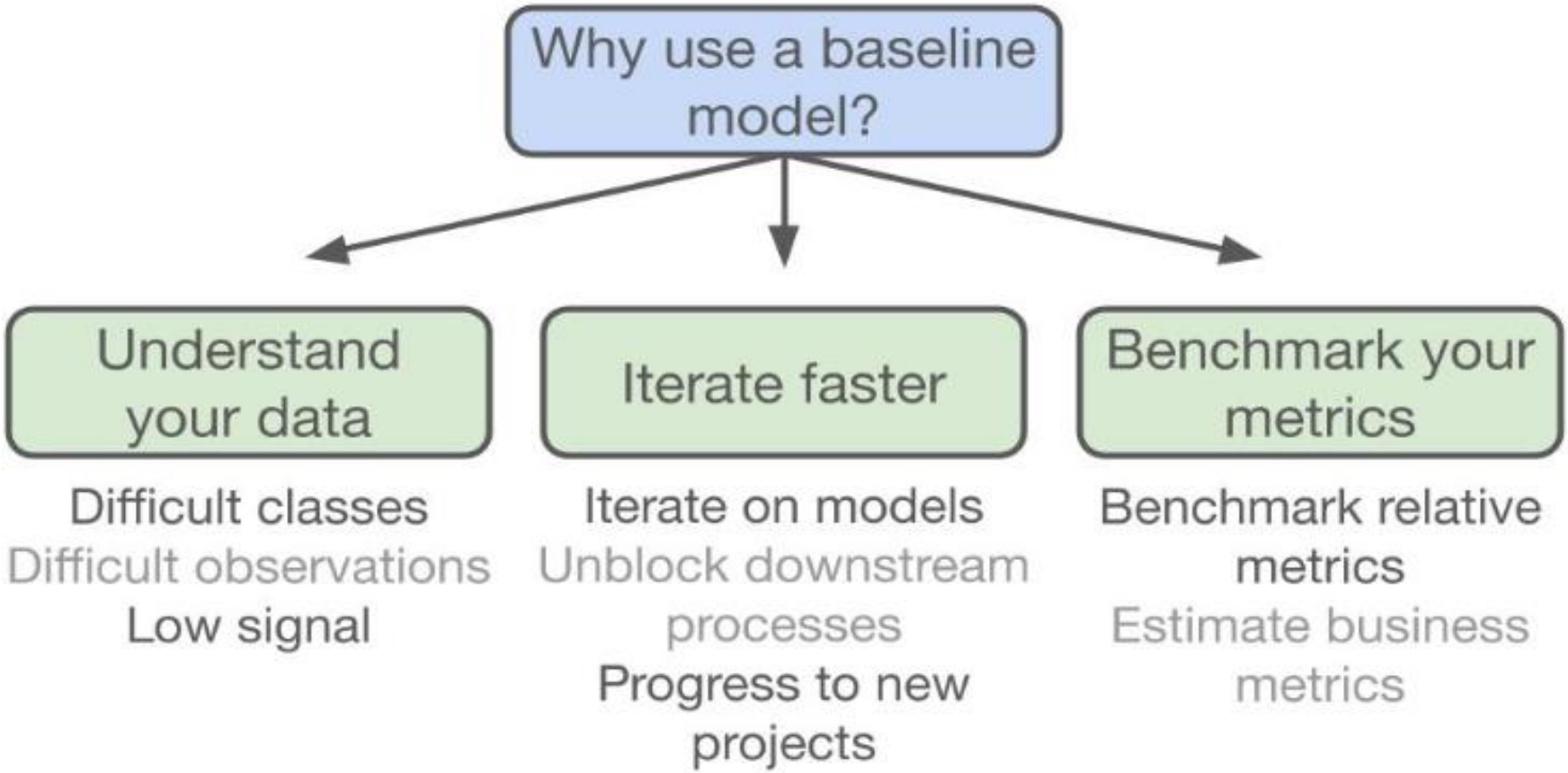
What if we could normalize the activations: $\boldsymbol{a}^{[l]}$ so that the training of $\boldsymbol{W}^{[l+1]}$ and $\boldsymbol{b}^{[l+1]}$ is more
That is, can we normalize the activations in the hidden layers too such that training of parame
layers may happen more rapidly?

In practice, $\boldsymbol{z}^{l}$ is normalized.

Why Batch Normalization Works

- Similar to input normalization, batch normalization normalizes directions of slow learning for hidden layers, which allows a higher learning rate to be used without risk oscillations
- Makes weights deeper in network more robust to weights earlier in the network

Baseline


Why use a baseline model?
Understand your data
Difficult classes
Difficult observations
Low signal
Iterate faster
Iterate on models
Unblock downstream processes
Progress to new projects
Benchmark your metrics
Benchmark relative metrics
Estimate business metrics


Practice
Is Key