Regularizations

L1 Regularization – Lasso Regression

$$\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- Works well for *feature selection* in case we have a huge number of features. L1 penalty tends to pick one variable at random when predictor variables are correlated.

L2 Regularization – Ridge Regression

$$\sum_{i=1}^{n}\left(y_i - \sum_{j=1}^{p} x_{ij}\beta_j\right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

- If λ is very large then it will add too much weight and it will lead to under-fitting
- This technique works very well *to avoid over-fitting issue*

Variance-Bias Trade-off

http://scott.fortmann-roe.com/docs/BiasVariance.html

Conceptual Definitions

- **Error due to Bias**: The error due to bias is taken as the difference between the expected (or average) prediction of our model and the correct value which we are trying to predict. Of course you only have one model so talking about expected or average prediction values might seem a little strange. However, imagine you could repeat the whole model building process more than once: each time you gather new data and run a new analysis creating a new model. Due to randomness in the underlying data sets, the resulting models will have a range of predictions. Bias measures how far off in general these models' predictions are from the correct value.
- **Error due to Variance**: The error due to variance is taken as the variability of a model prediction for a given data point. Again, imagine you can repeat the entire model building process multiple times. The variance is how much the predictions for a given point vary between different realizations of the model.

Mathematical Definition

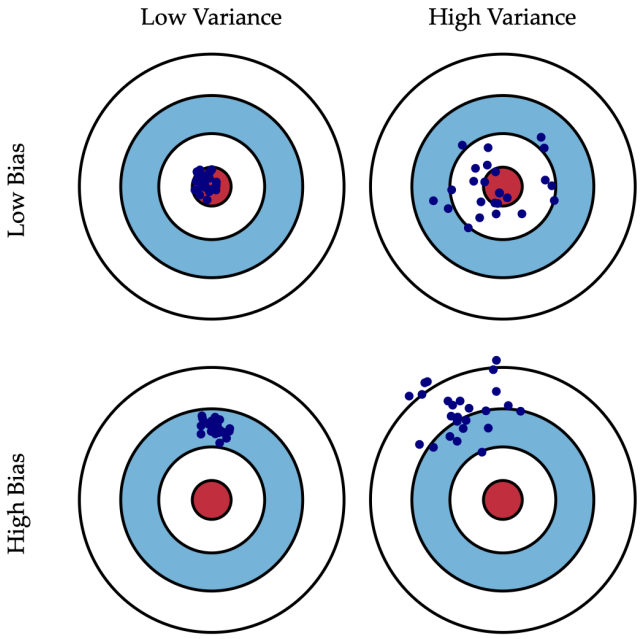
We denote the variable we want to predict as Y and the covariates as X. we may assume that there is a relationship relating one to the other such as $Y = f(X) + \epsilon$ and the error term is normally distributed as $\epsilon \sim N(0, \sigma_e^2)$.

We might estimate a model $\hat{f}(x)$ of $f(x)$ using linear regression or another modelling technique.

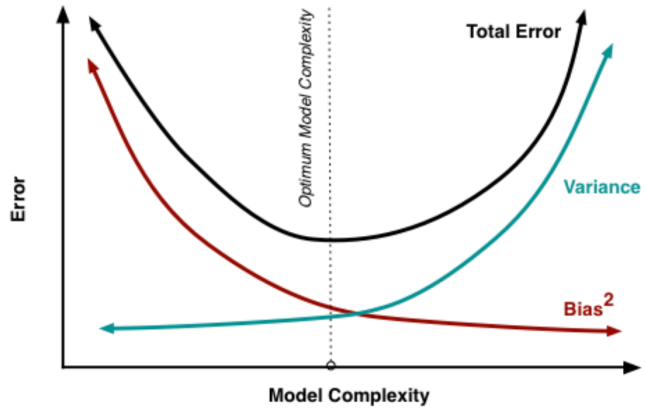
$$Err(x) = E\left[\left(Y - \hat{f}(x)\right)^2\right] = \left(E\left[\hat{f}(x)\right] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E\left[\hat{f}(x)\right]\right)^2\right] + \sigma_e^2$$

$$Err(x) = Bias^2 + Variance + Irreducible\ Error$$

Graphical Definition



At its root, dealing with bias and variance is really about dealing with over- and under-fitting. *Bias is reduced and variance is increased in relation to model complexity*. As more and more parameters are added to a model, the complexity of the model rises and variance becomes our primary concern while bias steadily falls. For example, as more polynomial terms are added to a linear regression, the greater the resulting model's complexity will be.



Another Example

Squiggly line (overfitted)

Low bias
High variance – it results in vastly different sums of squares for different datasets

Straight line (underfitted)
High bias – can't capture the relations
Low variance – it makes consistently good predictions

Neural Network Embeddings Explained
An embedding is a mapping of a discrete — categorical — variable to a vector of continuous numbers.
https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526

Embedding versus one-hot encoding
The one-hot encoding technique has two main drawbacks:

1. For high-cardinality variables — those with many unique categories — the dimensionality of the transformed vector becomes unmanageable.
2. The mapping is completely uninformed: "similar" categories are not placed closer to each other in embedding space.

Generative Model vs. Discriminative Model
A Generative Model learns the joint probability distribution $P(X, Y)$. It predicts the conditional probability with the help of Bayes Theorem. *A Discriminative Model* learns the conditional probability distribution $P(Y|X)$.

Generative models use probability estimates and likelihood to model data points and distinguish between different class labels in a dataset. These models *are capable of generating new data instances*. However, they also have a major drawback. The presence of outliers affects these models to a significant extent.

Discriminative models, also called *conditional models*, tend to learn *the boundary between classes/labels* in a dataset. Discriminative models are *not capable of generating new data instances*.

In Math, training classifiers involve estimating f: X -> Y or P(Y|X)

Generative classifiers

- Assume some functional form for $P(Y)$, $P(X|Y)$
- Estimate parameters of $P(X|Y)$ and $P(Y)$ directly from training data
- Use Bayes rule to calculate $P(Y|X) = P(Y)P(X|Y)/P(X) = P(X, Y)/P(X)$

Discriminative Classifiers

- Assume some functional form for $P(Y|X)$
- Estimate parameters of $P(Y|X)$ directly from training data

Generative Classifiers

- Naïve Bayes (Bayesian networks)
- Hidden Markov Models (HMM)

Discriminative Classifiers

- Logistic regression
- Support Vector Machine
- Decision Tree
- Random Forest

Dense & Sparse Features

Dense features incorporate information from users/items pairs, historical statistics, predictions from upstream models [1] and etc. Typically *sparse features* include IDs of users/items, demographics, keywords and etc. Many recommendation models rely on both dense and sparse features in conjunction in order to achieve the best results. Dense features cannot completely replace sparse features and vice versa.

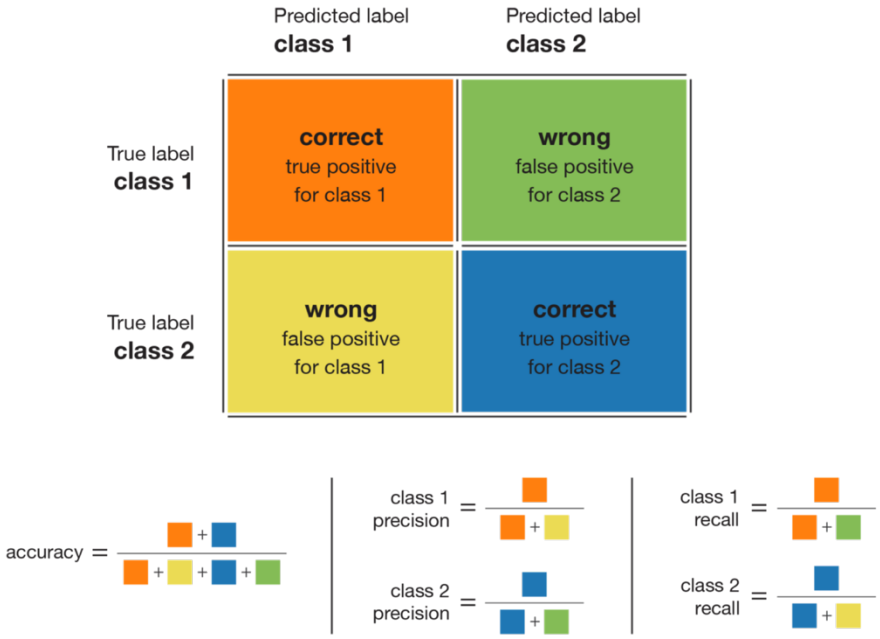
Dense and sparse features have different properties. For example, *the relative ordering of dense feature values are meaningful but this is rarer for sparse features*. In an online production system values for *sparse features are often restricted within a predefined dictionary while ranges of dense features can be undetermined*. Due to the inherent differences of dense and sparse features, they are processed in different ways.

Bagging/Boosting Variance and Bias

Bagging decreases ensemble variance by averaging, but has little effect on the ensemble bias, and solves the overfitting issues in a model.

Boosting decreases bias, not variance. I think it is because these mislabeled samples are with more weights, then the model would have less ensemble bias.

Confusion Matrix & ROC & AUC



- The **precision** of a class define *how trustable* is the result when the model answer that a point belongs to that class. **Pr**ecision: TP/**Pr**edicted Positive
- The **recall** of a class expresses how well the model is able to detect that class/what proportion of samples are correctly classified. Low recall value means our model is not doing well for this class. **Re**call: TP/**Re**al Positive
- The **accuracy** of the model is basically the total number of correct predictions divided by total number of predictions.
- The **F1** score of a class is given by the harmonic mean of precision and recall (2×precision×recall / (precision + recall)), it combines precision and recall of a class in one metric.

For a given class, the different combinations of recall and precision have the following meanings :

- high recall + high precision : the class is perfectly handled by the model
- low recall + high precision : the model can't detect the class well but is highly trustable when it does
- high recall + low precision : the class is well detected but the model also include points of other classes in it
- low recall + low precision : the class is poorly handled by the model

ROC & AUC

Each value of the threshold T generates a point (false positive, true positive) and, then, the ROC curve is the curve described by the ensemble of points generated when the threshold T varies from 1 to 0.

y-axis True Positive Rate = Recall

x-axis False Positive Rate = 1 – Precision

ROC curves make it easy to identify the best threshold for making a decision; while the **AUC** helps you decide which categorization method is better.

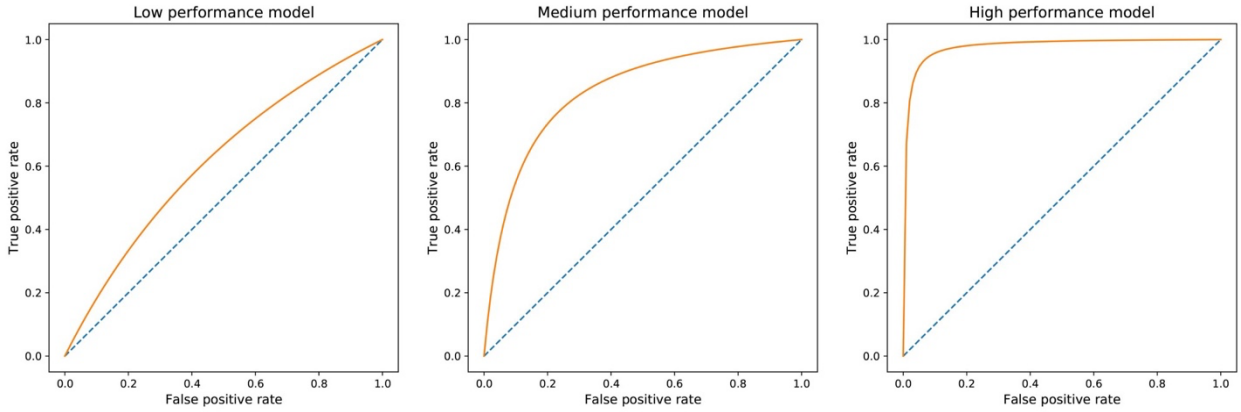


Illustration of possible ROC curves depending on the effectiveness of the model. On the left, the model has to sacrifice a lot of precision to get a high recall. On the right, the model is highly effective: it can reach a high recall while keeping a high precision.

Linear Regression Outliers

How to identify outliers?

- Standard residuals

How to deal with outliers?

- A measurement error or data entry error, correct the error if possible. If you can't fix it, remove that observation because you know it's incorrect.
- Not a part of the population you are studying (i.e., unusual properties or conditions), you can legitimately remove the outlier.
- A natural part of the population you are studying, you should not remove it.
    - What you should do in that case is two separate analyses, one with the outlier in the analysis and one with the outlier removed from the analysis. Then report both analyses and let the reader make the decision as to which one should be used.

How to make outliers less influence to the linear regression?

- Robust regression uses a method called *iteratively reweighted least squares to assign a weight to each data point*. This method is less sensitive to large changes in small parts of

the data. As a result, robust linear regression is less sensitive to outliers than standard linear regression.

Linear Regression Multicollinearity

How to detect multicollinearity?

The Variance Inflation Factor (VIF) quantifies how much the variance is inflated. The VIF for the j^{th} predictor is:

$$VIF_j = \frac{1}{1 - R_j^2}$$

where R_j^2 is the R^2-value obtained by regressing the j^{th} predictor on the remaining predictors (considering the x_j as the dependent variable and all others are independent variables).

VIF is 1 means that there is no correlation among the j^{th} predictor and the remaining predictor variables, and hence the variance of b_j is not inflated at all. The general rule of thumb is that VIFs exceeding 4 warrant further investigation, while VIFs exceeding 10 are signs of serious multicollinearity requiring correction.

How to deal with multicollinearity?

One solution to dealing with multicollinearity is to remove some of the violating predictors from the model.

Linear Regression High Cardinality

What is the high cardinality?

A categorical feature is said to possess *high cardinality* when there are too many of these unique values. One-hot Encoding becomes a big problem in such a case since we have a separate column for each unique value in the categorical variable. This leads to two problems (1) storage space consumption; (2) *the curse of dimensionality*.

How to deal the feature with a high cardinality?

A simple aggregation method. Leave instances belonging to a value with high frequency as they are and replace the other instances with a new category which we will call *other*.
Say our column color has 100 values and our threshold is 90% (that is 90). We have 5 different categories of colors: Red (50), Blue(40), Yellow (5), Green (3) and Orange (2). The numbers within the bracket indicate how many instances of that category are present in the column.
We see that Red (50) + Blue (40) reaches our threshold of 90. In that case, we retain only 2 categories (Red, Blue) and mark all other instances of other colors as "*Other*". *Thus we have reduced cardinality from 5 to 3 (Red, Blue, Other).*

Boosting Algorithms

What's the differences between AdaBoost and Gradient Boosting?

See Ace the Data Science Interview page 100.

Optimizations

Gradient Descent (also called batch gradient descent)

$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

Repeat {

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

(for every $j = 0, \dots, n$)

}

Notes: when m (the sample size) is large, computing is very expensive.

Stochastic Gradient Descent

$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

1. Randomly shuffle dataset
2. Repeat {

   For $i = 1, \dots, m$ { // GD on each observation, so the convergence path is the zigzag shape

   $$\theta_j := \theta_j - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

   (for every $j = 0, \dots, n$)

   }

   }

Mini-Batch Gradient Descent

$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

Say $b = 10$ (batch size) and $m = 1000$

Repeat {

For $i = 1,11,21, \dots,991$ {

$$\theta_j := \theta_j - \alpha\frac{1}{10}\sum_{k=i}^{i+9}\left(h_\theta\left(x^{(k)}\right) - y^{(k)}\right)x_j^{(k)}$$

(for every $j = 0, \dots, n$)

}

}

Note: It's between batch GD and stochastic GD. The vectorization can be parallelized.

RMSprop (Root Mean Square Propagation)

On iteration t:

Compute dW, db on current *mini-batch*

$S_{dW} = \beta_2 S_{dW} + (1-\beta_2)dW^2$

$S_{db} = \beta_2 S_{db} + (1-\beta_2)db^2$

$w := w - \alpha \frac{dW}{\sqrt{S_{dW}}+\epsilon}$

$b := b - \alpha \frac{db}{\sqrt{S_{db}}+\epsilon}$

Notes:

The algorithm damps out the oscillations in the gradient descent, so a larger learning rate can be applied to speed up the learning algorithm.

Adam (ADAptive Moment estimation)

$V_{dW} = 0, S_{dW} = 0, V_{db} = 0, S_{db} = 0$

On iteration t:

Compute

$V_{dW} = \beta_1 V_{dW} + (1-\beta_1)dW, V_{db} = \beta_1 V_{db} + (1-\beta_1)db$ // the "momentum" β_1

$S_{dW} = \beta_2 S_{dW} + (1-\beta_2)dW^2, S_{db} = \beta_2 S_{db} + (1-\beta_2)db^2$ // the "RMSProp" β_2

$V_{dW}^{corrected} = V_{dW}/(1-\beta_1^t), V_{db}^{corrected} = V_{db}/(1-\beta_1^t)$ // t is the iteration number

$S_{dW}^{corrected} = S_{dW}/(1-\beta_2^t), S_{db}^{corrected} = S_{db}/(1-\beta_2^t)$

$w := w - \alpha \frac{V_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected}}+\epsilon}$

$b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}}+\epsilon}$

Configuration parameters:

α: needs to be tuned

β_1: 0.9 // the first moment

β_2: 0.999 // the second moment

ϵ: 10E-8

A/B Testing

What's the A/B testing?

A/B testing is to compare 2 or more versions.

What's the null hypothesis and the alternative hypothesis?

The null hypothesis is that the change in the design made for the test group **would results in no change in the conversion rate.** The alternative hypothesis is the opposing that the change in the design for the test group **would results in an improvement (or reduction) in the conversion rate.**

How to determine the minimum sample size?

You will need *the baseline conversion rate* and *the minimum detectable effect*, which is the minimum difference between the control and test group that you or your team will determine to be worth the investment of making the design change in the first place. The equation for minimum sample size in each group (assumes equal sized groups) is

Error! Filename not specified.

Error! Filename not specified.: z-score represents the desired Power (typically 0.84 for 80% power – see below for explanation of the Power)

Error! Filename not specified.: z-score represents the desired level of statistical significance (typically 1.96)

Error! Filename not specified.: pooled probability or average of **Error! Filename not specified.** and **Error! Filename not specified.**

Error! Filename not specified.: success rate of control group

Error! Filename not specified.: success rate of test group

What is the Power?

Power is also known as **Error! Filename not specified.**, it can be explained as the strength of your test to detect an actual difference in your variant.

Conversely, **Error! Filename not specified.** is the probability that your test does not reject the null hypothesis when it should actually be rejecting the null hypothesis. *The higher the power, the lower the probability of a Type II error.* Experiments are usually set at a power level of 80%, or 20% **Error! Filename not specified.**. Another way of putting it is: if there is a difference in the test, you're willing to make Type II error 20% of the time.

How to deal with small sample sizes?

We can lower the required sample size by:

- Reducing the confidence threshold / increasing the significance threshold – accepting higher risk of false positives
- Increasing the minimum effect of interest (MEI/MDE)
- By accepting the test will have lower power against a particular MEI

Can you run an A/B test with unequal sample sizes? (Hyperlink)

TL;DR: Yes, but you wouldn't want to.

You can run an experiment with an unequal allocation (e.g. 10–90) as long as you don't modify the allocation while the experiment is running. However it will be less efficient than a 50–50 allocation – either your test will have less power, or you will need to run it longer to achieve a comparable result.

Imbalanced Dataset

All these approaches aim at rebalancing (partially or fully) the dataset.

- Undersampling consists in sampling from the majority class in order to keep only a part of these points

- Oversampling consists in replicating some points from the minority class in order to increase its cardinality
- Generating synthetic data consists in creating new synthetic points from the minority class (see SMOTE method for example) to increase its cardinality

SMOTE

SMOTE (synthetic minority oversampling technique) is one of the most commonly used oversampling methods to solve the imbalance problem. It aims to balance class distribution by randomly increasing minority class examples by replicating them. SMOTE synthesises new minority instances between existing minority instances. It generates the **virtual training records by linear interpolation** for the minority class.

- **Step 1:** Setting the minority class set **A**, for each $\boldsymbol{x} \in \boldsymbol{A}$**,** the **k-nearest neighbors of** x are obtained by calculating the **Euclidean distance** between $\boldsymbol{x}$ and every other sample in set $\boldsymbol{A}$.
- **Step 2:** The sampling rate $\boldsymbol{N}$ is set according to the imbalanced proportion. For each $x \in A$, $\boldsymbol{N}$ examples (i.e $x_1, x_2, \dots, x_n$) are randomly selected from its k-nearest neighbors, and they construct the set A_1.
- **Step 3:** For each example $x_k \in A_1$ (k=1, 2, 3…N), the following formula is used to generate a new example
  $x' = x + rand(0,1) \cdot |x - x_k|$
  in which $rand(0,1)$ represents the random number between 0 and 1.

Machine Learning System Design

- Search Ranking
- Feed Based System
- Recommendation System
- Self-Driving Car: Image Segmentation
- Entity Linking System
- Ad Prediction System

CNN

The pooling operation involves sliding a two-dimensional filter over each channel of feature map and summarizing the features lying within the region covered by the filter. For a feature map having dimensions $n_h \times n_w \times n_c$, the dimensions of output obtained after a pooling layer is

$$(n_h - f + 1)/s \times (n_w - f + 1)/s \times n_c$$

- n_h - height of feature map
- n_w - width of feature map
- n_c - number of channels in the feature map
- f - size of filter
- s - stride length

A common CNN model architecture is to have **a number of convolution and pooling layers stacked one after the other**.

Why to use Pooling Layers?

- Pooling layers are used to **reduce the dimensions of the feature maps**. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.
- The pooling layer **summarizes the features present in a region of the feature map generated by a convolution layer.** So, further operations are performed on summarized features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

Type of pooling layers:

- Max pooling
- Average pooling
- Global pooling – global max pooling or global average pooling
  Global pooling reduces each channel in the feature map to a single value. Thus, an $n_h \times n_w \times n_c$ feature map is reduced to $1 \times 1 \times n_c$ feature map. This is equivalent to using a filter of dimensions $n_h \times n_w$ i.e. the dimensions of the feature map. Further, it can be either global max pooling or global average pooling.

The motivation of padding

- For a gray scale $(n \times n)$ image and $(f \times f)$ filter/kernel, the dimensions of the image resulting from a convolution operation is $(n - f + 1) \times (n - f + 1)$. **Thus, the image shrinks every time a convolution operation is performed.** This places an upper limit to the number of times such an operation could be performed before the image reduces to nothing thereby **precluding (阻止 v.) us from building deeper networks**.
- Also, the pixels on the corners and the edges are used much less than those in the middle.

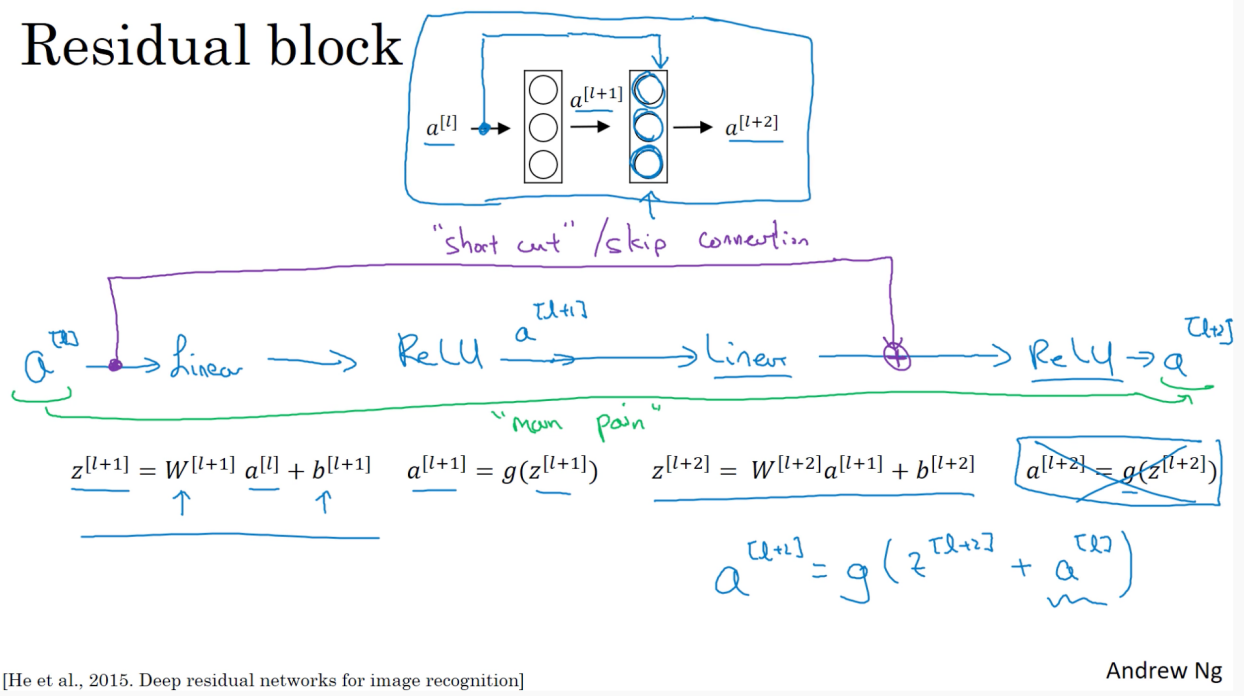
The same padding:

Same Padding: In this case, we add 'p' padding layers (p layers of zeros) such that the output image has the same dimensions as the input image.

$$[(n + 2p) \times (n + 2p) \text{ image}] * [(f \times f) \text{ filter}] \text{ -> } [(n \times n) \text{ image}]$$

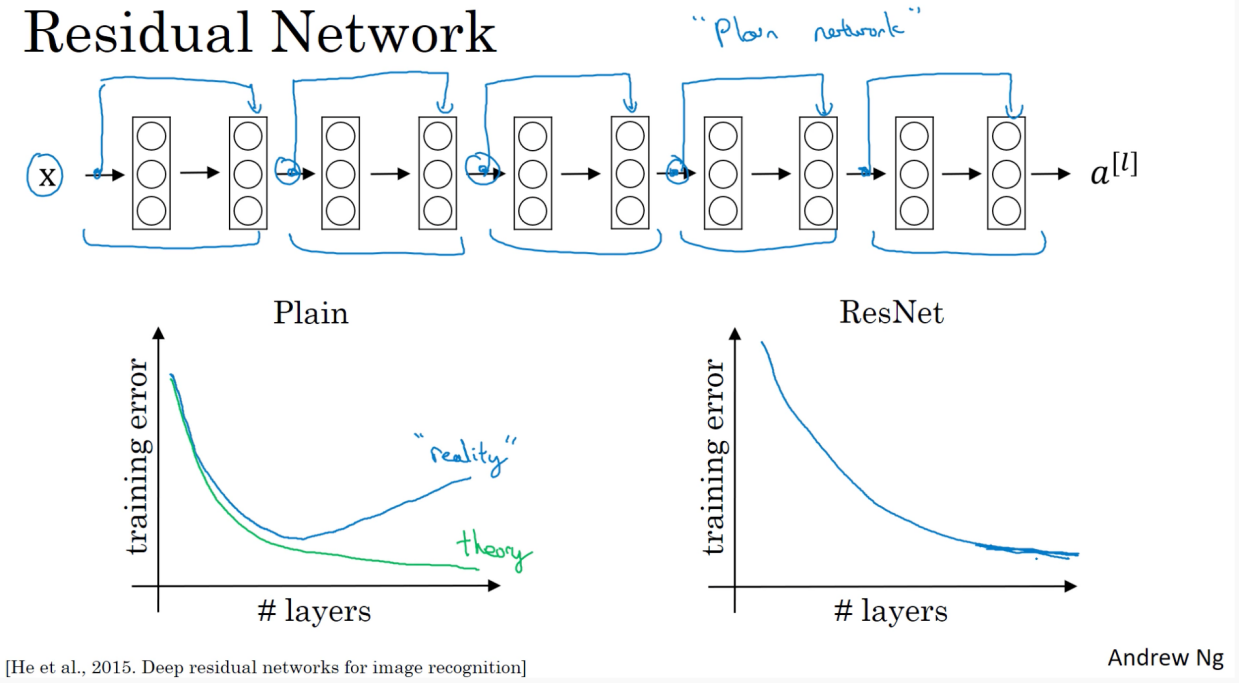
where * represents a convolution operation
which gives p = (f – 1) / 2 (because n + 2p – f + 1 = n).

Residual Networks (ResNets):



[He et al., 2015. Deep residual networks for image recognition]

Andrew Ng



[He et al., 2015. Deep residual networks for image recognition]

Andrew Ng

Why ResNets work?

- **The identity function** is easy for residual block to learn. It won't hurt the overall performance as the depth increases. Deep networks hard to learn even the identity function.

A bag has 18 black balls and 9 white balls. On each turn you randomly select two balls and set them aside. If both balls are the same color you then add a black ball, otherwise you add a white ball. What is the probability that the last ball left in the bag is white? What if we instead start with 18 black balls and 36 white balls?

B(t) and W(t) are number of black balls and white balls at t
So we have B(0) = 18 and W(0) = 9 and also each time number of balls in the bag decreased by 1
So B(t+1) + W(t+1) = B(t) + W(t) – 1 and at the 26th step, B(26) + W(26) = 1
If we have BB, then B(t+1) = B(t) – 1 and W(t+1) = W(t)
If we have BW, then B(t+1) = B(t) – 1 and W(t+1) = W(t)
If we have WW, then B(t+1) = B(t) + 1 and W(t+1) = W(t) – 2
Notice that W is *always* odd because W(0) = 9 and it is decreased by 2 and B(26) + W(26) = 1. So the last one must be a white ball.

If there are 18 blacks balls and 36 white balls.
Because W(0) = 36, so W is *always* even and B(53) + W(53) = 1. So the last one must be a black ball.

You know that the probability it rains on any given day is 0.4. You are wondering if you should bring your umbrella with you today. You ask your friend if it's raining right now and he says yes. Knowing that your friend may be lying to you with probability 0.2, what is the true probability that it is raining now?

R standards for prob of raining and Y stands for friends says yes.
So the question is asking for $P(R|Y)$ and we know $P(R) = 0.4$ and $P(Y|NR) = 0.2$.

$$P(R|Y) = \frac{P(Y|R)P(R)}{P(Y|R)P(R) + P(Y|NR)P(NR)} = \frac{0.8 \times 0.4}{0.8 \times 0.4 + 0.2 \times 0.6} = \frac{8}{11}$$