

Machine Learning for APAT & ASAT

2024. 11

Yongjin Jeong, KwangWoon University

[참고] 본 자료에는 인터넷에서 다운받아 사용한 그림이나 수식들이 일부 있으니 다른 용도로 사용하거나 외부로 유출을 금해 주시기 바랍니다.

Contents

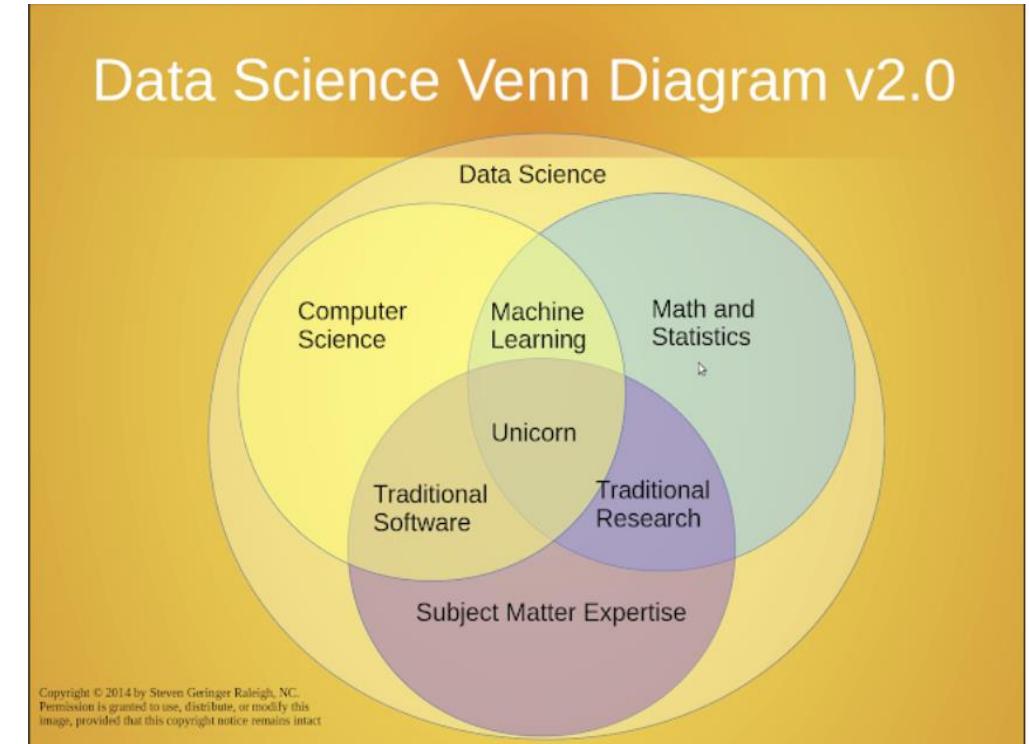
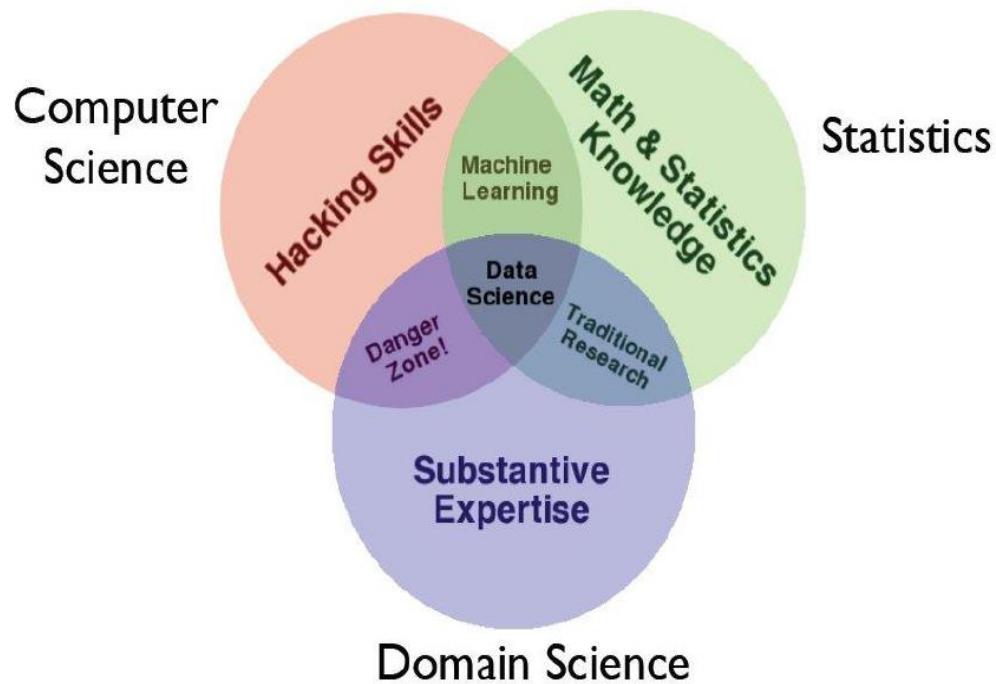
- Data science workflow
- Data preprocessing
- Machine learning models
- Linear Models
- Performance
- Overfitting
- Imbalance problem
- Sklearn library convention

What is Data Science?

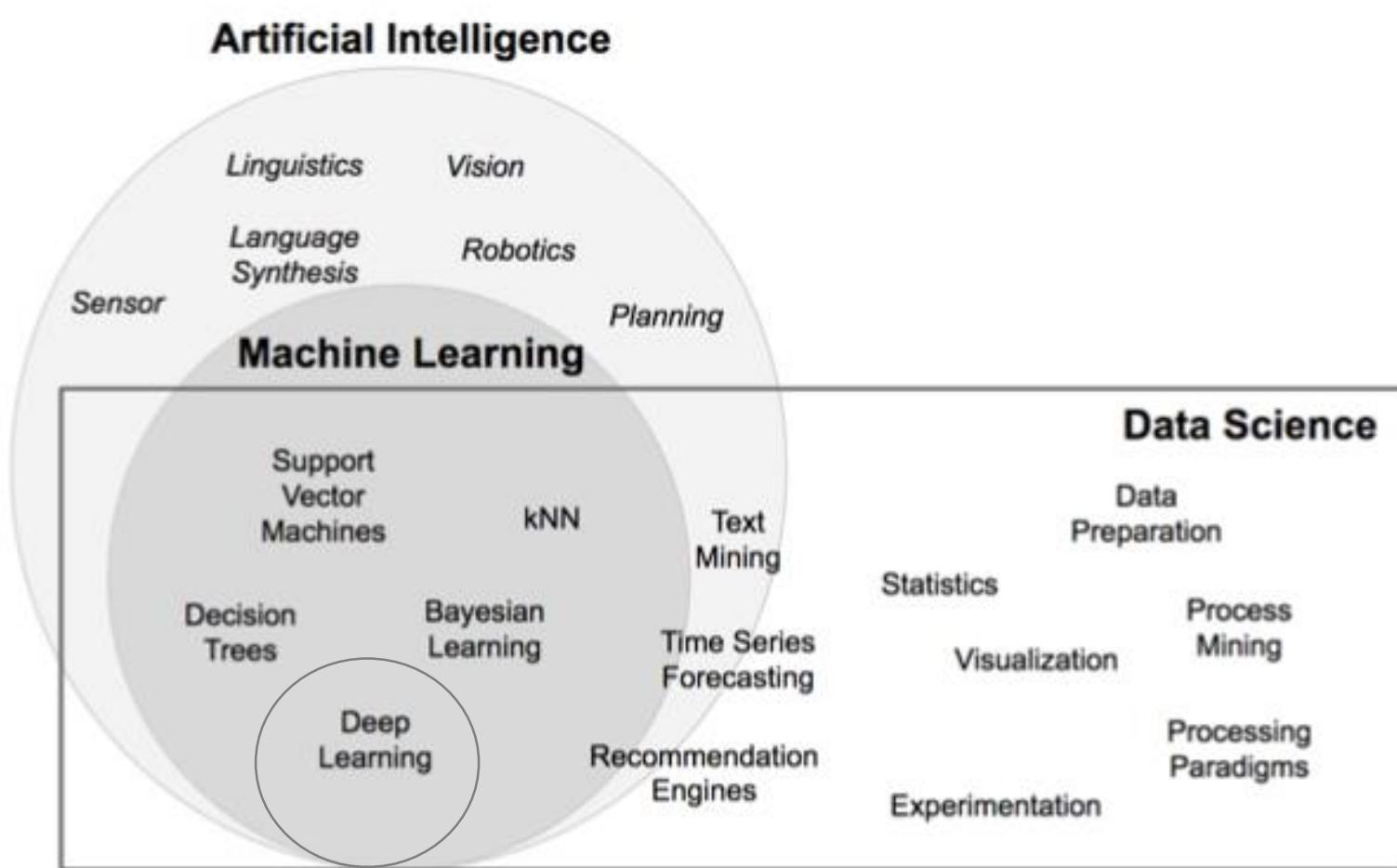
- **Definition (from Wikipedia)**
 - ✓ concept to unify [statistics](#), [data analysis](#), [machine learning](#), [domain knowledge](#) and their related methods in order to understand and analyze actual phenomena with data
 - ✓ It uses techniques and theories drawn from many fields within the context of [mathematics](#), [statistics](#), [computer science](#), [domain knowledge](#), and [information science](#)
- **Components of Data Science**
 - ✓ **Software Programming** -> Data mining, Database
 - ✓ **Statistics/mathematical modeling** -> Machine Learning, Scientific Computing
 - ✓ **Domain Knowledge** -> Data driven business analytics
- **Main applications**
 - ✓ E-commerce, social media, IoT, biometrics, financial, management, health, pharmacy
 - ✓ Autonomous vehicles, smart energy, medical, ships, logistic, robots, etc.
 - ✓ Almost all areas

What is Data Science? – One definition

- Venn Diagrams (Drew Conway 2010, Steven Geringer 2014)



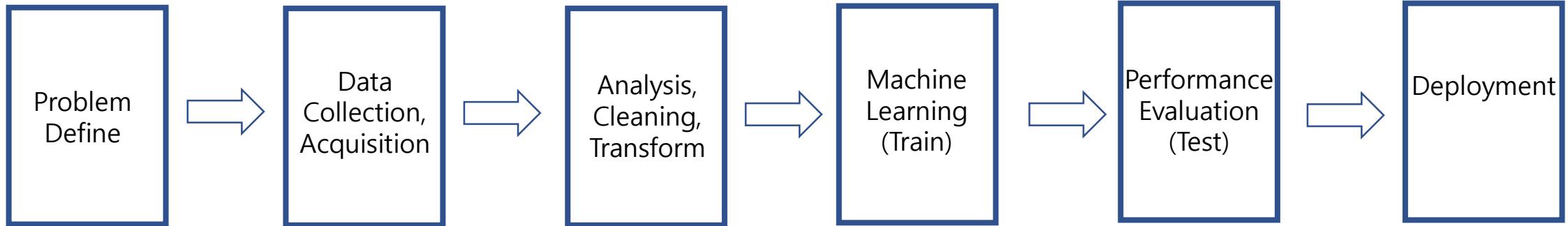
What is Data Science?



Statistics vs. Data Science

	Statistics	Data science
Common	aim to extract knowledge from data <ul style="list-style-type: none">- exploratory data analysis & Visualization- characterization & prediction- use sample data to make conclusions about either future data (machine learning) or the population (statistics)	
Main goal	<ul style="list-style-type: none">- Inference and <u>explanation</u>- emphasis on <u>understanding</u> relationships and estimating population parameters	<ul style="list-style-type: none">- prediction accuracy and complex datasets- focus on accurate <u>predictions</u> often without detailed understanding or interpretability
Language	estimating (inferencing) data point/observation independent variable dependent variable dummy variable	learning (training) & prediction example/instance/ <u>sample</u> <u>feature</u> <u>label</u> or <u>target</u> one-hot encoding
Data	small or medium sized mostly structured more manual data collection (or surveys) <u>In general, no web scraping or data processing</u>	huge (big data) structured or unstructured more data collection/acquisition (from web and SNS)
Processing	query (past)	predict (future)
Tools	Mathematics <u>prefer R</u> SAS (statistics package)	<u>programming</u> (<u>prefer Python</u>) ML libraries (sklearn, tensorflow, etc.)

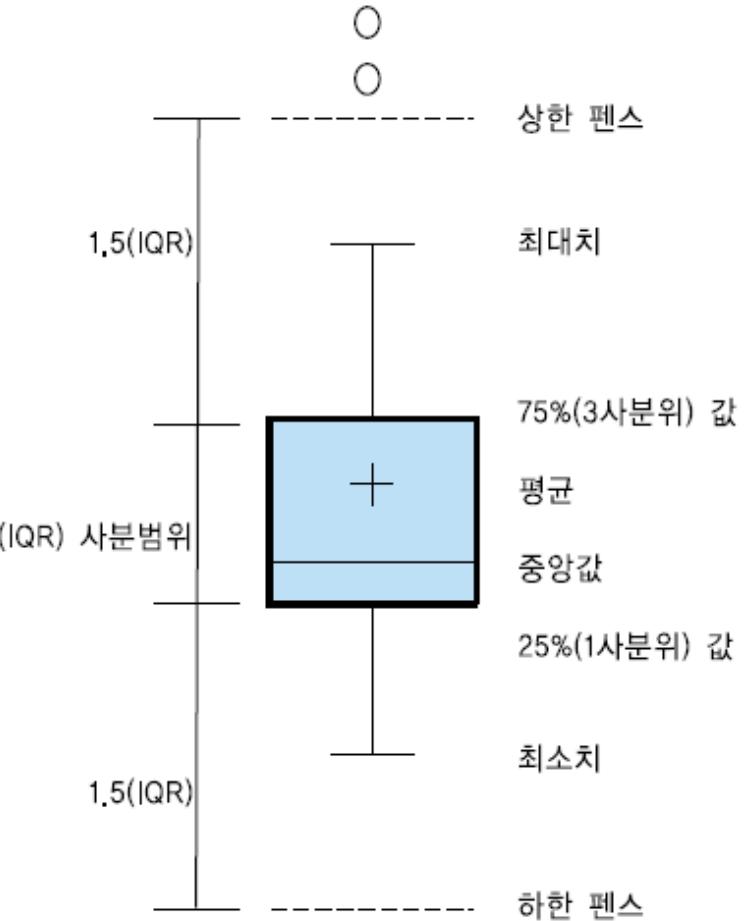
Data Science Work Flow



- | | | | | | |
|---------------------|----------------------|------------------------|--------------------------|--------------------|----------|
| • Domain knowledge | • CSV/Excel | • Visualization | • Supervised | • R-square | • Server |
| • Business strategy | • JSON | • Missing values | • Unsupervised | • Accuracy | • Mobile |
| | • HTML/XML | • Invalid values | • Loss (or Error) | • Precision/recall | |
| | • SNS | • Outliers | • Bias and Variance | • F-1 score | |
| | • String(structured) | • Categorical encoding | • Overfitting | • ROC/AUC | |
| | • Text(unstructured) | • Scaling | • Regularization | • mAP | |
| | • Image, Voice | • Transform | • MLP/CNN/RNN | • IoU | |
| | • Language | • Feature engineering | • Generative model | | |
| | • Multi-modal | | • Reinforcement learning | | |
| | | | • Transformer | | |

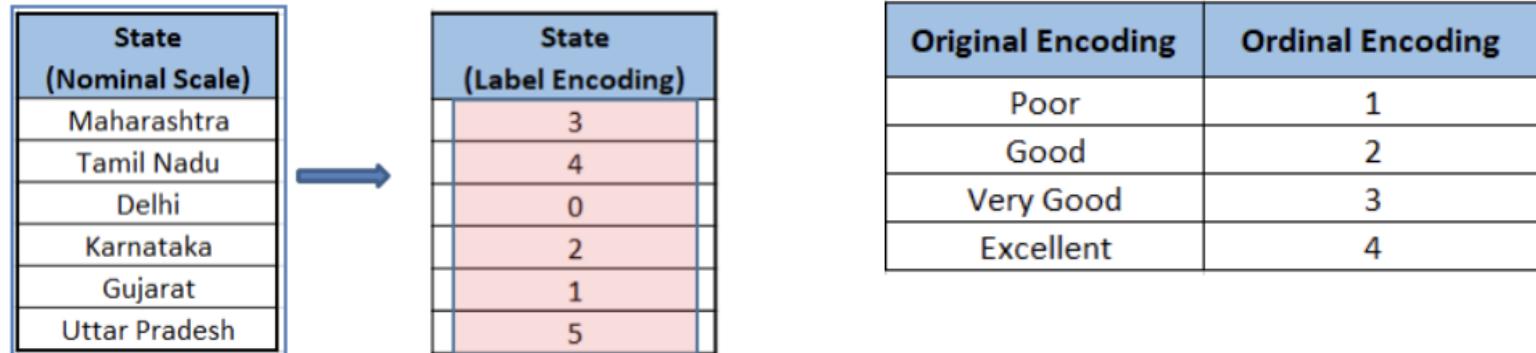
Data Preprocessing - Scaling

- Why scaling?
 - What if two columns have different ranges for the values? Hard to compare.
- Normalization (min-max scaling):
 - $x = (x - x_{\min}) / (x_{\max} - x_{\min})$
- Standardization (standard scaling):
 - $z = (x - \mu) / \sigma$
- Robust scaling:
 - $z = (x - \text{median}) / \text{IQR}$
 - use median and IQR (instead of μ and σ), robust to outliers
- Which one to use?
 - depends on your data



Categorical Encoding

- Label encoding
- Ordinal encoding
- One-hot encoding
- Binary encoding
- Target encoding
- many more...

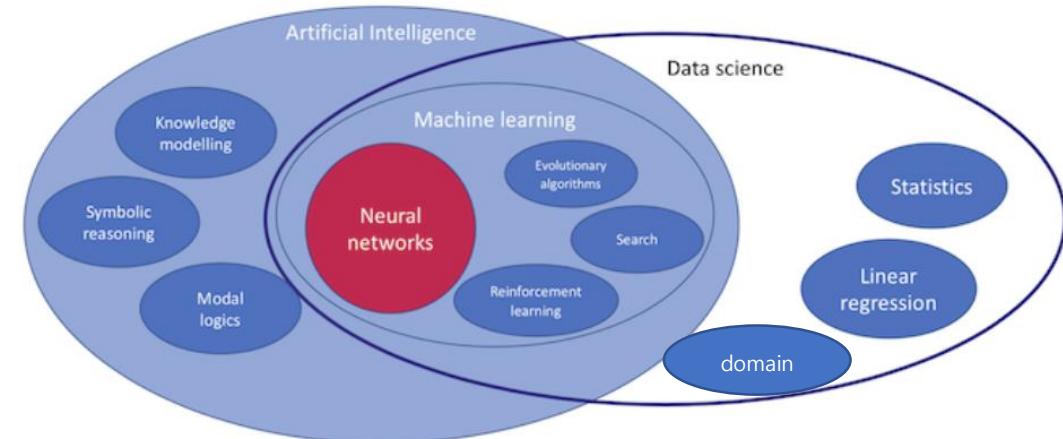
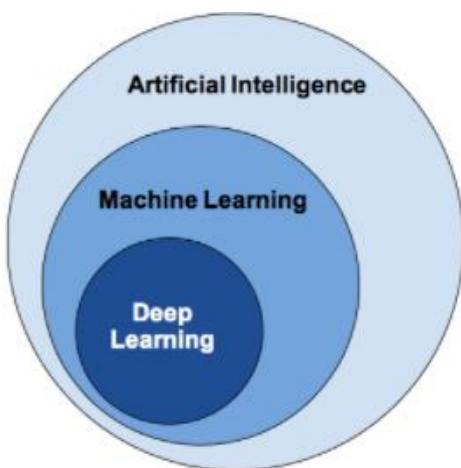


State	State_Maharashtra	State_Tamil Nadu	State_Delhi	State_Karnataka	State_Gujarat	State_Uttar Pradesh
Maharashtra	1	0	0	0	0	0
Tamil Nadu	0	1	0	0	0	0
Delhi	0	0	1	0	0	0
Karnataka	0	0	0	1	0	0
Gujarat	0	0	0	0	1	0
Uttar Pradesh	0	0	0	0	0	1

Machine Learning

- **What is ML**

- the study of computer algorithms that improve automatically through experience and by the use of data. It is seen as a part of artificial intelligence. [wikipedia]
- ML algorithms build a model based on sample data (training data) in order to make predictions or decisions without being explicitly programmed to do so.



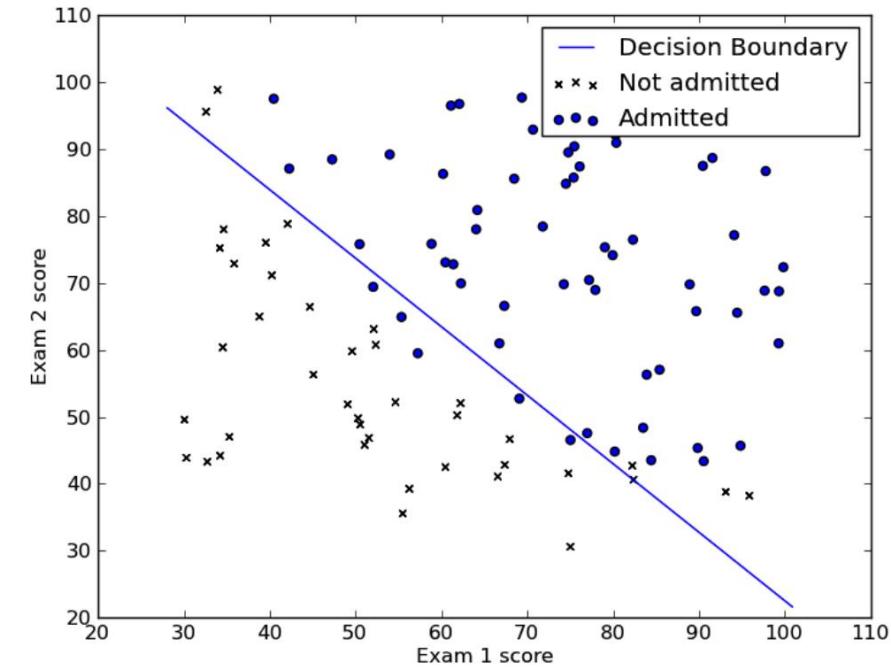
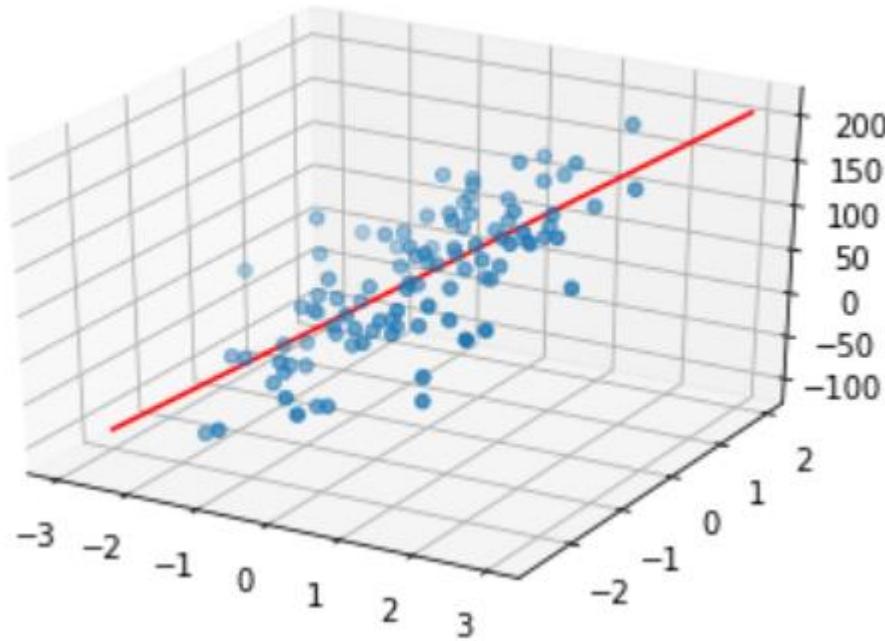
[ref] <https://ictinstitute.nl/ai-machine-learning-and-neural-networks-explained/>

Machine Learning

- **Machine Learning approaches**
 - **Supervised learning**: inputs(features) and outputs(labels) are both given (by a teacher) to learn a general rule to map features to labels
 - **Regression** : labels are continuous values
 - **Classification**: labels are categorical values
 - **Unsupervised learning**: no labels are given, and to discover hidden patterns or features in data
 - **Dimension reduction**: PCA(Principal Component Analysis), tSNE, Autoencoder
 - **Clustering** (or grouping)
 - **Semi-supervised learning**: large unlabeled data with small labeled data
 - **Reinforcement learning**: interacts with the environment by producing actions and discovers errors or rewards (trial and error search, delayed reward)
 - Model-based RL (like control theory)
 - Model-free RL: Policy-iteration (Policy gradient, A3C) and value-iterations (Q-learning)

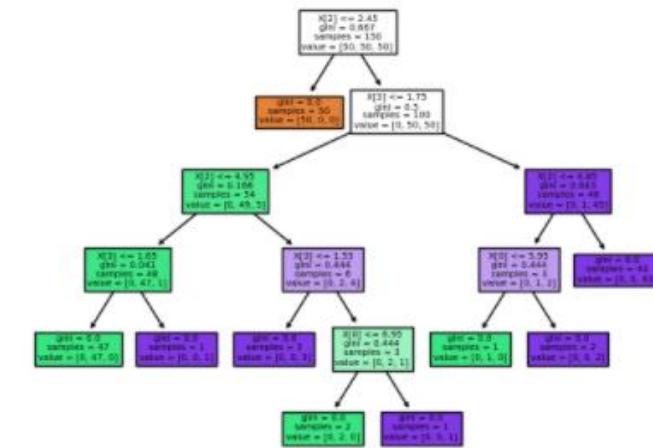
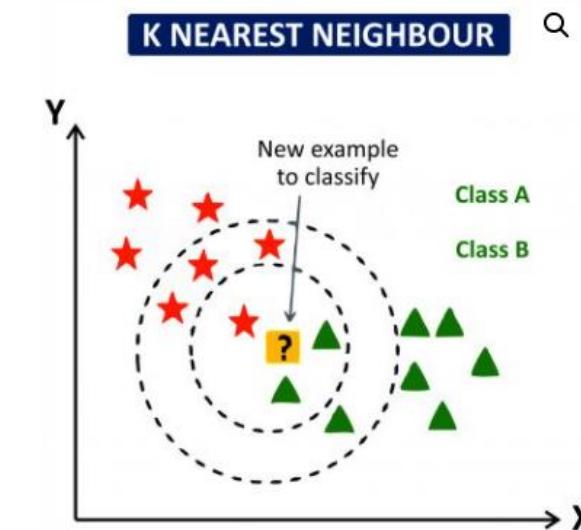
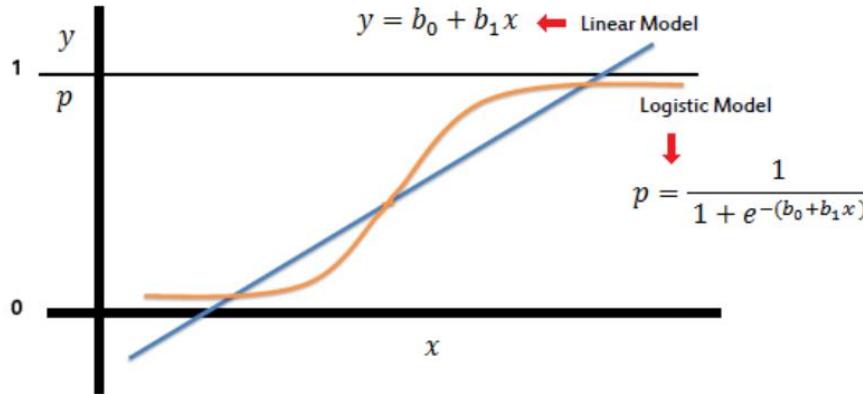
Supervised Learning

- **Linear Regression and Linear Classification**
 - (regression) $y = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
 - (classification) $w_1 \cdot x_1 + w_2 \cdot x_2 + b = 0$



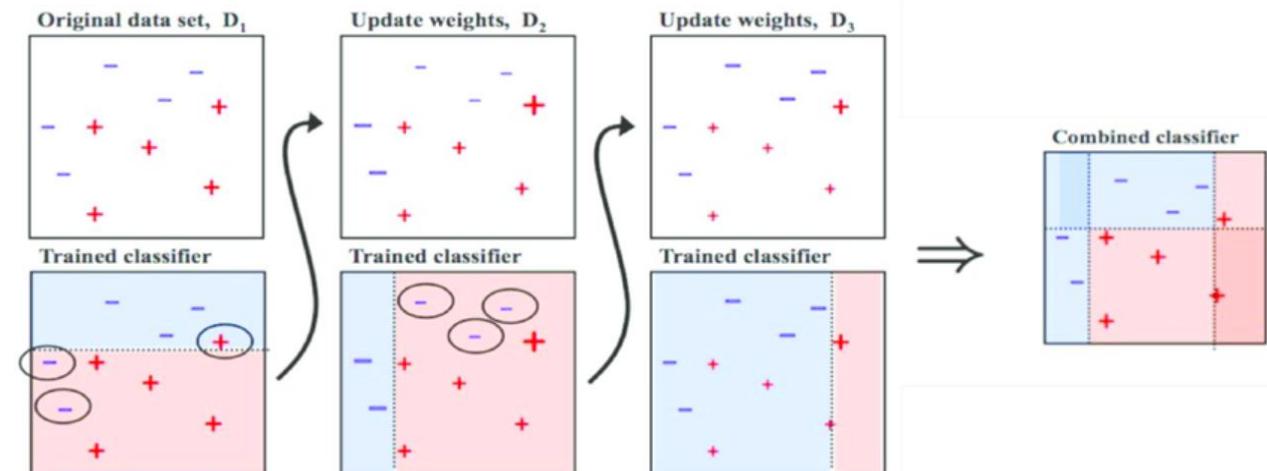
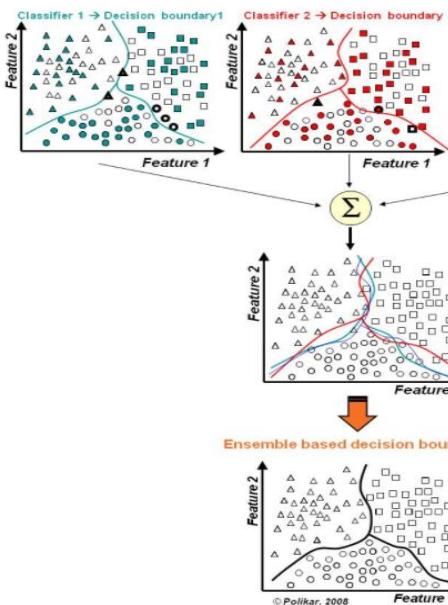
Supervised Learning

- Other algorithms (linear and nonlinear)
 - Logistic Regression Classifier
 - Knn (k-nearest neighbor)
 - Decision Tree
 - SVM (Support Vector Machine)



Supervised Learning

- Ensemble method
 - Combine many weak learners
 - Bagging: learns them independently in **parallel** and averages them (ex: Random Forest)
 - Boosting: learns them **sequentially** in an adaptive way and combines them (ex: Gradient Boost, Adaboost)

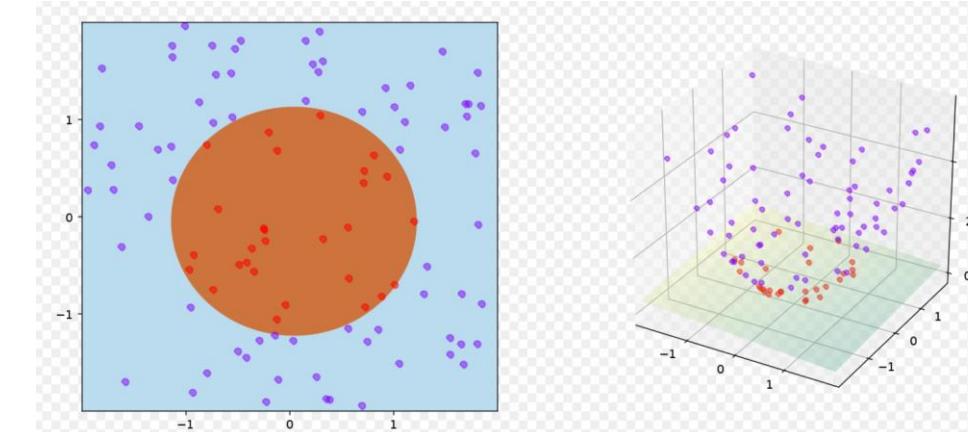
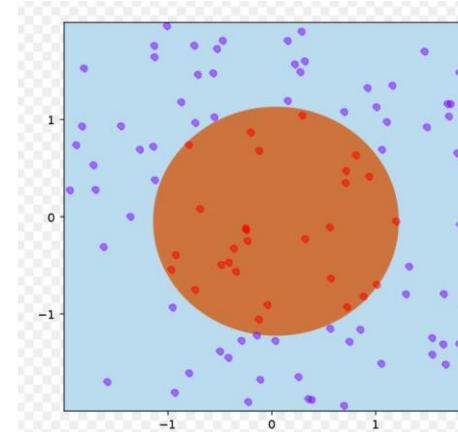
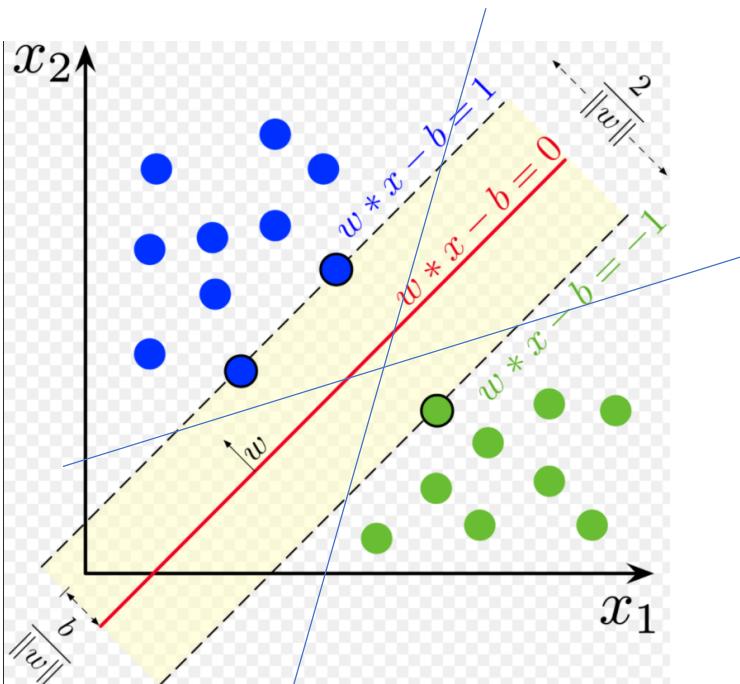


출처: Medium (Boosting and Bagging explained with examples)

Supervised Learning

- **SVM (Support Vector Machine)**

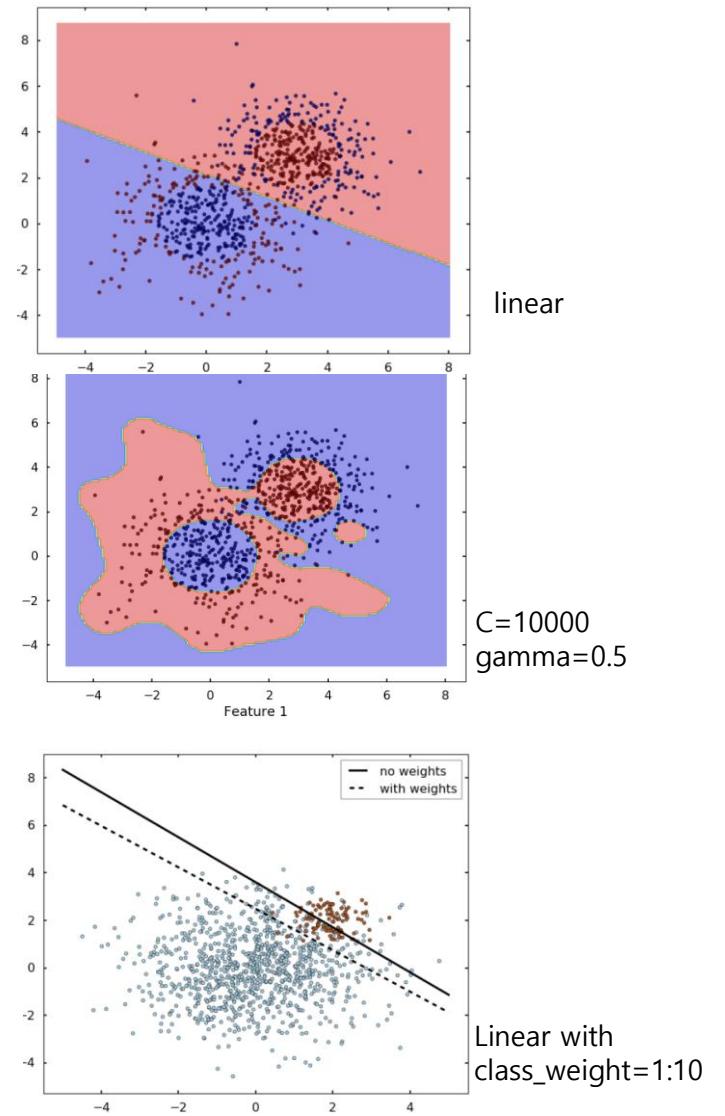
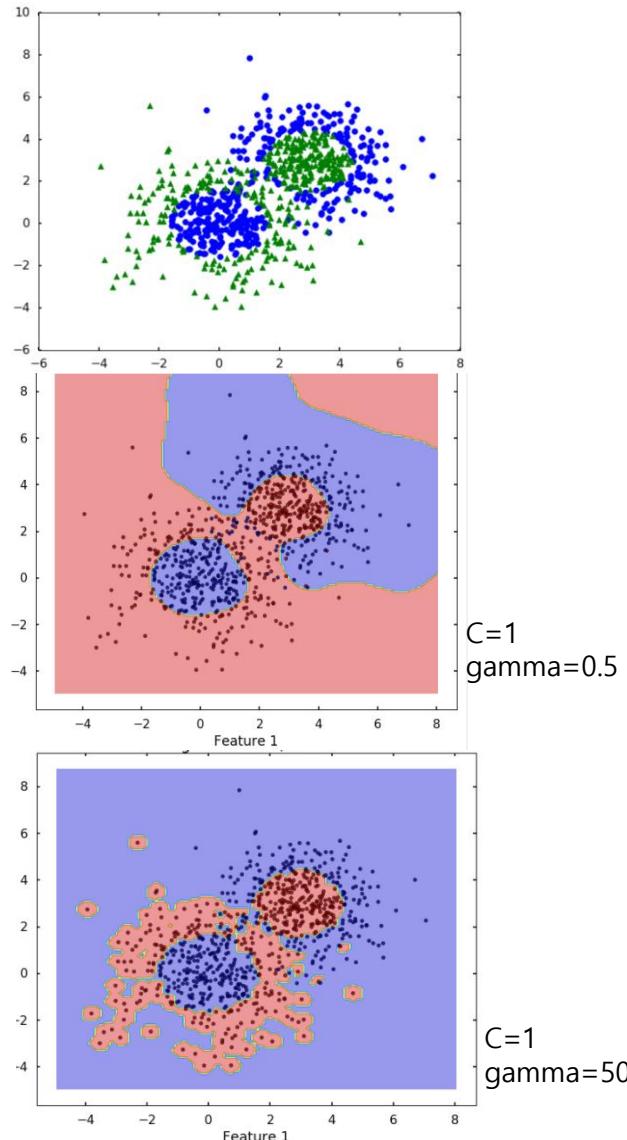
- Finds a maximum-margin hyperplane
- Linear SVM
- Nonlinear SVM: use kernel (polynomial, sigmoid, rbf)



SVM with kernel given by $\varphi((a, b)) = (a, b, a^2 + b^2)$

Supervised Learning

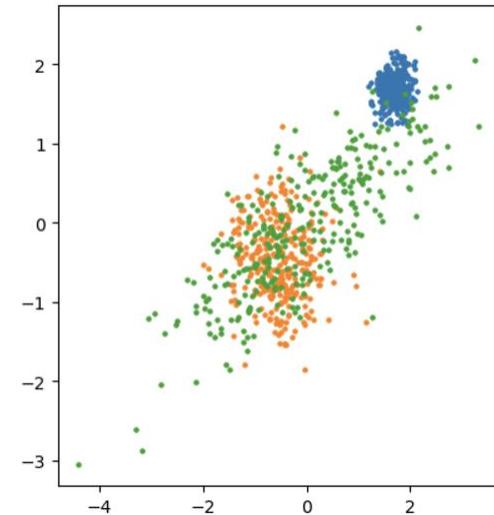
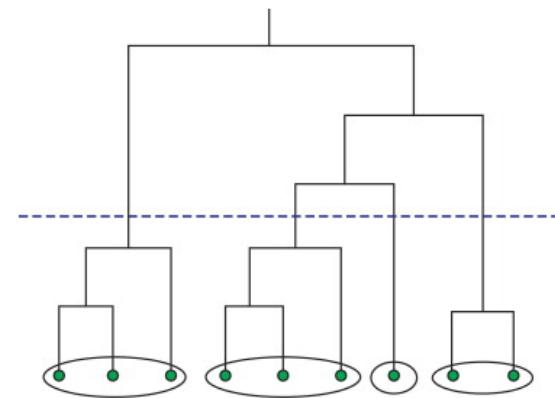
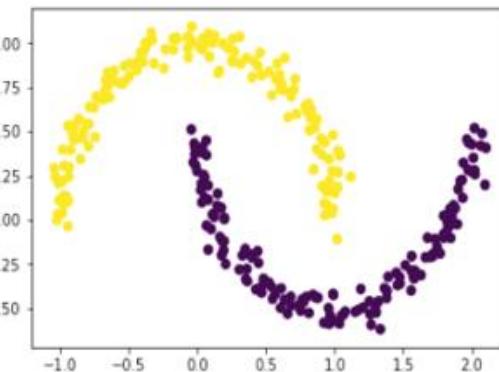
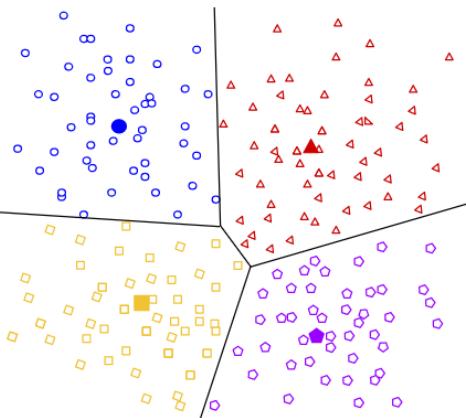
- **SVM (continued)**
 - 'rbf' kernel (hyperparameters: C and gamma)
 - **C**: tradeoff between classification error and simplicity of the boundary
 - **gamma**: defines how far the influence of a single example reaches (high value is 'close')
 - `class_weight`: imbalance cases



Unsupervised Learning

- **Clustering (or Grouping)**

- Divide the data points into several clusters based on similarity (유사도)
- Need scaling as a preprocessing step
- Applications: detect hackers and criminal activity, identifying fake news, etc.
- Centroid-based ([K-means](#))
- Density-based ([DBSCAN](#))
- Hierarchical (ex: [dendrogram](#))
- GMM (Gaussian Mixture Model)

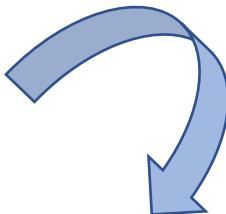


Unsupervised Learning

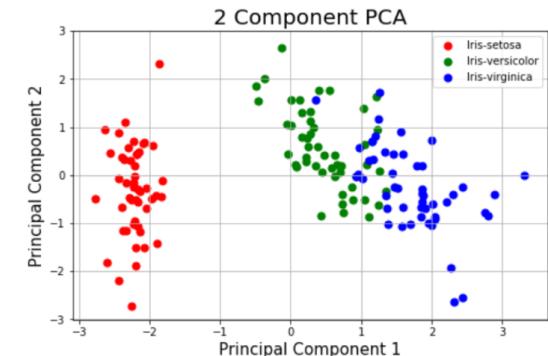
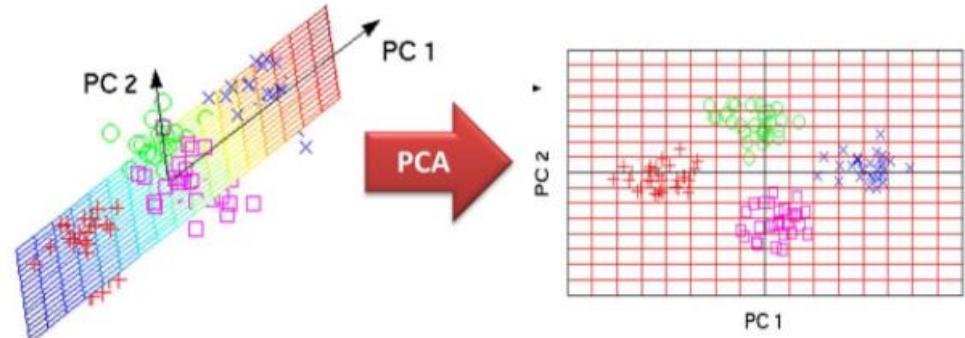
- Dimension reduction by **PCA**

- Standard scaling
- Calculate covariance (or correlation) matrix
- Eigen-decomposition ($A = P \Lambda P^{-1}$)
- Select k eigenvectors
- `pca_result = PCA(n_components=2).fit_transform(X_all)`
- Also, [tSNE\(\)](#), [Autoencoder](#)
- [SelectPercentile\(\)](#)

	sepal_len	sepal_wid	petal_len	petal_wid	class
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica



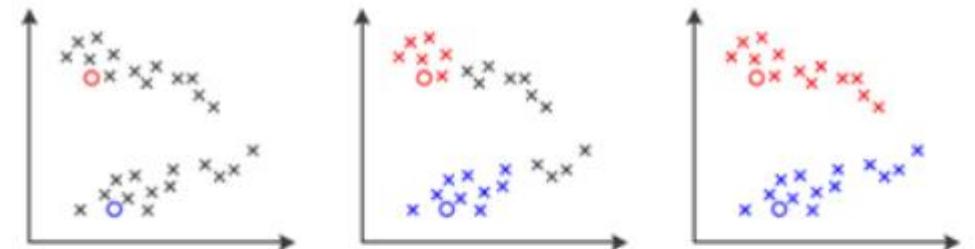
	principal component 1	principal component 2	species
0	-2.264542	-0.505704	Iris-setosa
1	-2.086426	0.655405	Iris-setosa
2	-2.367950	0.318477	Iris-setosa
3	-2.304197	0.575368	Iris-setosa
4	-2.388777	-0.674767	Iris-setosa



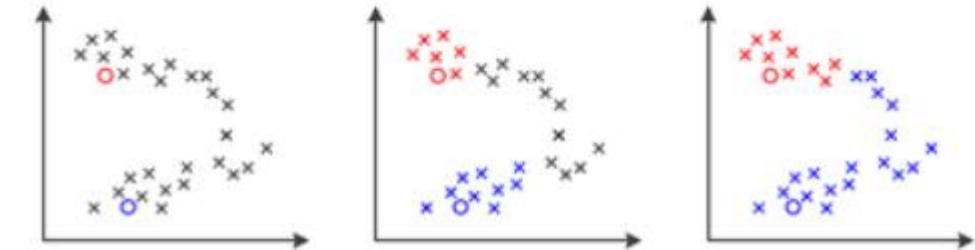
Semi-supervised Learning

- **Semi-supervised**

- large unlabeled data with small labeled data
- Will unlabeled data be helpful? Yes or No
- Self-training:
 - Train with labeled data
 - Classify unlabeled data
 - Select samples with **high confidence** and include them in labeled data
 - Retrain the classifier
 - Repeat
- Using GAN
- Many others

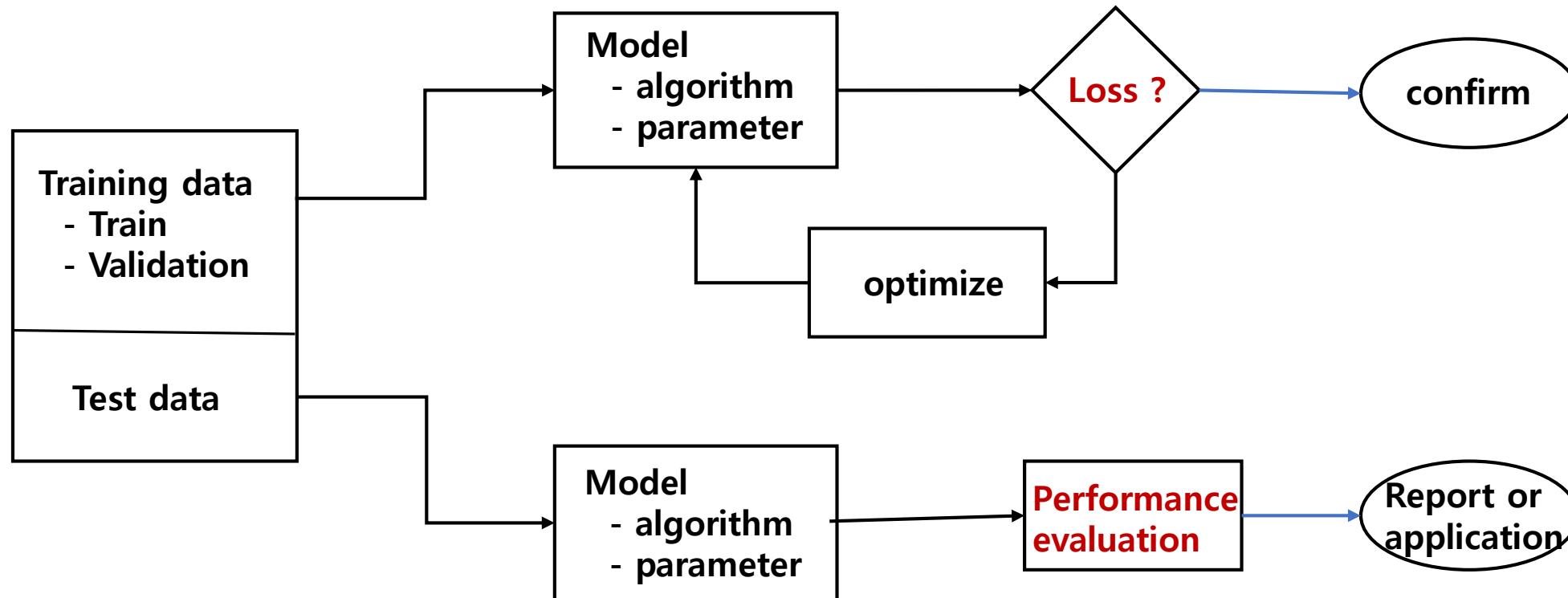


(a) 잘 작동하는 상황



(b) 소속이 애매한 샘플에 민감한 상황

Machine Learning Model (supervised)

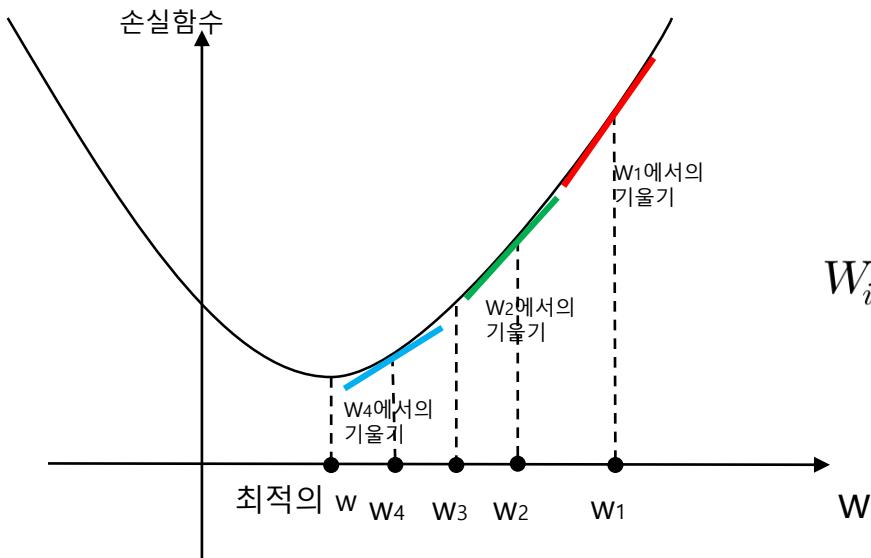


- (model) Parameter: estimated from the dataset (learned during training from the historical data sets)
- Hyper-parameter: external to the model (defined manually before the model training by trial-error)

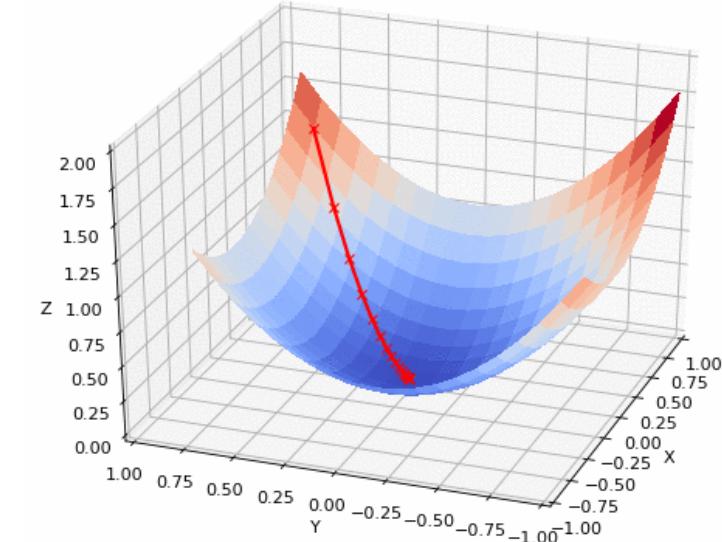
Gradient Descent (GD) algorithm

- **Gradient Descent** (경사하강법)

- General optimization algorithm
- take repeated steps in the opposite direction of the **gradient** (or approximate **gradient**) of the function at the current point



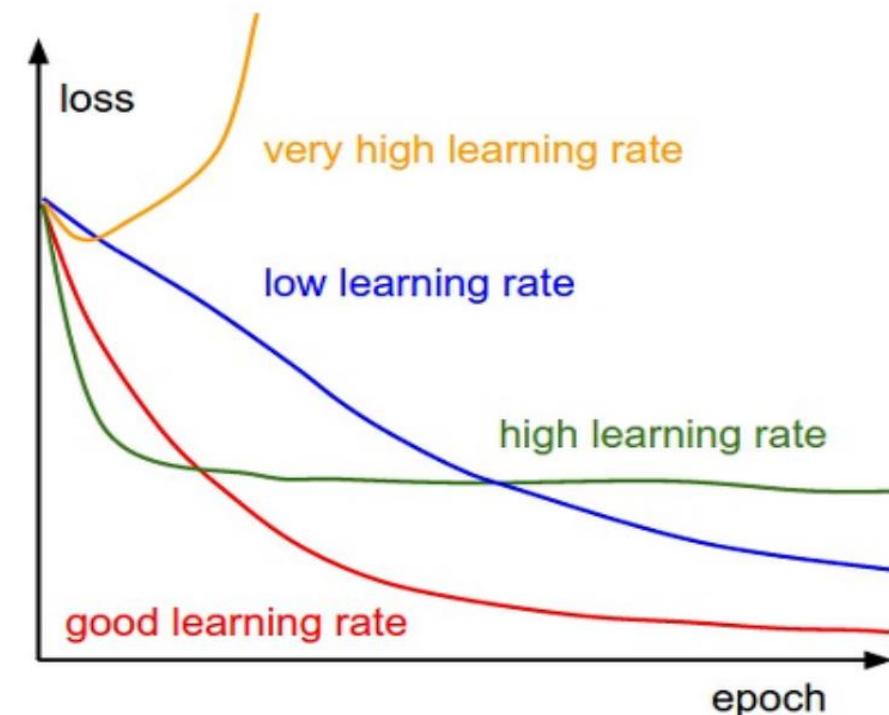
$$W_i = W_{i-1} - \eta \text{Grad}(i)$$



Gradient Descent (GD) algorithm

- **Learning rate: η (eta)**

- low: takes time to converge, and may get stuck in an undesirable local minimum
- high: may jump over minima
- too high: may diverge
- Need adaptive adjustment



Loss Function

- What to reduce? (**Loss or Error or Cost**: 손실함수)

- Regression (회귀): **MSE** (Mean Square Error)

$$MSE = \sum_{k=1}^N (y - \hat{y})^2$$

- Classification (분류): **Cross Entropy (CE)**, **Gini Coefficient**

$$CE = \sum_i p_i \log\left(\frac{1}{p_i}\right) \quad Gini = 1 - \sum_{k=1}^m p_k^2$$

- Binary case:

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

Linear Regression

- **Gradient Descent Algorithm**

- Loss (or **cost**) function: MSE (mean square error)

$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

$$w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j}$$

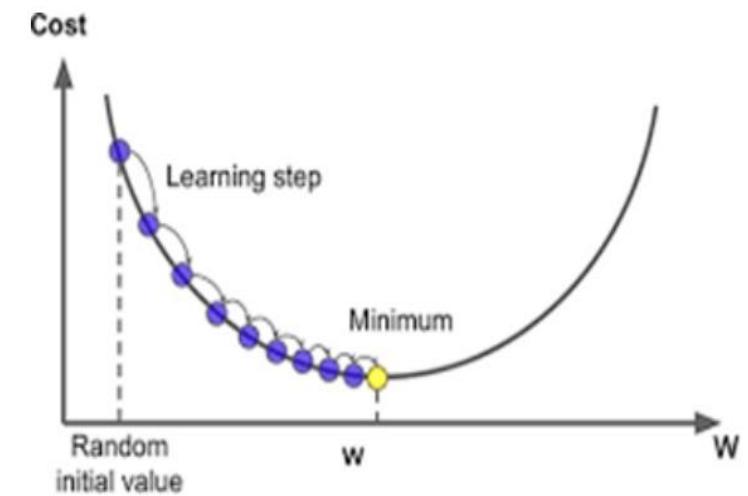
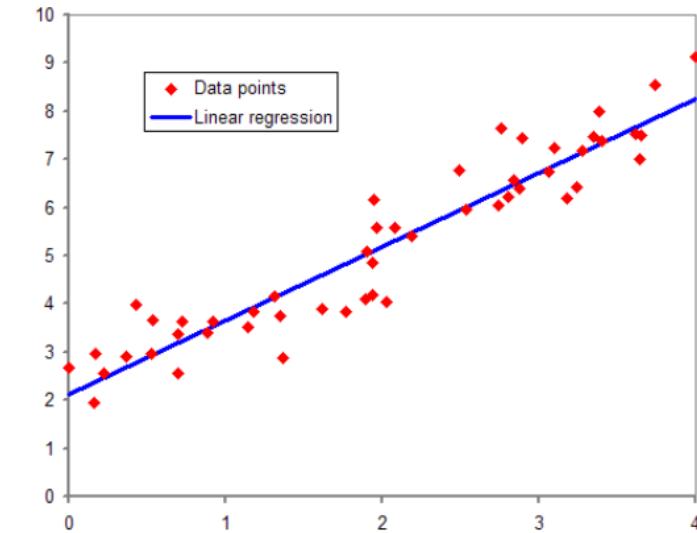
$$= w_j - \frac{\alpha}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) x_j^{(i)}$$

- In vector form

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}}$$
$$= \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^N (y^{(i)} - t^{(i)}) \mathbf{x}^{(i)}$$

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix}$$

$$y = wx + b$$



Linear Regression

- Normal equation (OLS: Ordinary Least Squares)
 - LinearRegression() in sklearn library
 - Don't require learning rate (no iterative steps)
 - High computational complexity (inverse requires $O(n^3)$ complexity)

$$u^T u = [u_1 \ u_2 \ u_3]^T [u_1 \ u_2 \ u_3] = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}^T \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$$

$$\frac{\partial(u^T u)}{\partial x} = 2u_1 \frac{\partial u_1}{\partial x} + 2u_2 \frac{\partial u_2}{\partial x} + 2u_3 \frac{\partial u_3}{\partial x} = 2[u_1 \ u_2 \ u_3] \begin{bmatrix} \frac{\partial u_1}{\partial x} \\ \frac{\partial u_2}{\partial x} \\ \frac{\partial u_3}{\partial x} \end{bmatrix} = 2u^T \frac{\partial u}{\partial x}$$

$$J = \sum_{i=1}^n (y_i - \theta^T x_i)^2$$

$$\rightarrow J = \|Y - X\theta\|^2 = (Y - X\theta)^T (Y - X\theta)$$

$$\frac{\partial J}{\partial \theta} = 2(Y - X\theta)^T \frac{\partial(Y - X\theta)}{\partial \theta} = 2(Y - X\theta)^T (-X) = 0$$

$$Y^T X - (X\theta)^T X = 0$$

$$Y^T X = (X\theta)^T X \rightarrow X^T Y = X^T X \theta$$

$$\Rightarrow \theta_* = (X^T X)^{-1} X^T Y$$

$$\hat{\theta} = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$

The diagram illustrates the linear regression model $y = Xw + b$. It shows a box labeled x_0, x_1 with an arrow pointing to a column vector y . Another box contains $x_{00}, x_{01}, x_{10}, x_{11}, x_{20}, x_{21}$, and below it are three values y_0, y_1, y_2 . A third box contains w_0, w_1, b , with an arrow pointing to the same y .

$$y_0 = x_{00}w_0 + x_{01}w_1 + b$$

$$y_1 = x_{10}w_0 + x_{11}w_1 + b$$

$$y_2 = x_{20}w_0 + x_{21}w_1 + b$$

$$y = Xw + b$$

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \\ x_{20} & x_{21} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} + b$$

$$= \begin{bmatrix} x_{00} & x_{01} & 1 \\ x_{10} & x_{11} & 1 \\ x_{20} & x_{21} & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ b \end{bmatrix}$$

$$\Rightarrow Y = XW$$

$$\Rightarrow X^T Y = X^T X W$$

$$\Rightarrow \underline{W_{best}} = (X^T X)^{-1} X^T Y$$

Linear Regression

- OLS regression analysis

```
sm.OLS(y_train, X_train_ols).fit().summary()
```

OLS Regression Results						
Dep. Variable:	SystemProduction	R-squared:	0.641			
Model:	OLS	Adj. R-squared:	0.641			
Method:	Least Squares	F-statistic:	2084.			
Date:	Tue, 26 Nov 2024	Prob (F-statistic):	0.00			
Time:	09:14:26	Log-Likelihood:	-57635.			
No. Observations:	7008	AIC:	1.153e+05			
Df Residuals:	7001	BIC:	1.153e+05			
Df Model:	6					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	697.0526	10.787	64.618	0.000	675.906	718.199
x1	19.7436	11.578	1.705	0.088	-2.953	42.440
x2	-214.2258	17.908	-11.963	0.000	-249.331	-179.121
x3	-66.7511	10.909	-6.119	0.000	-88.136	-45.366
x4	1196.0869	19.381	61.714	0.000	1158.094	1234.080
x5	86.6055	12.972	6.676	0.000	61.177	112.034
x6	-177.8948	15.225	-11.685	0.000	-207.739	-148.050
Omnibus:	1086.969	Durbin-Watson:	2.017			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	19237.530			
Skew:	0.040	Prob(JB):	0.00			
Kurtosis:	11.116	Cond. No.	3.74			

결정계수 R^2 (R-Squared):

모델이 종속 변수의 변동을 얼마나 잘 설명하는지 나타냄.

- $R^2 = 0$: 모델이 아무것도 설명하지 못함.
- $R^2 = 1$: 모든 변동을 완벽히 설명.

F-통계량 (F-test)

- 회귀모델 전체의 유의성을 평가
- H_0 : "모든 회귀 계수가 0이다" (독립 변수들이 종속 변수에 영향을 미치지 않는다)
- $P\text{-value} < 0.05$ (reject $H_0 \rightarrow$ 모델이 유의미하다)

개별독립 변수의 유의성

- 각 독립 변수가 종속 변수에 유의미한 영향을 미치는지를 평가 (t-검정)
- H_0 : "해당 독립 변수의 회귀 계수 $\beta_i=0$, 즉 종속 변수에 영향을 미치지 않는다"

Omnibus: 잔차(residuals)가 정규성을 따르는지를 검정 (잔차의 왜도와 첨도를 함께 분석)

- H_0 : "잔차가 정규분포를 따른다"
- prob(omnibus): $p\text{-value} < 0.05$ (reject H_0 , 정규성을 벗어남)

Linear Regression

- (결과 해석) F-test, t-test 는 모두 유의미하지만 잔차는 정규성을 벗어나고 있다.

(1) 비선형성 존재:

- 독립 변수와 종속 변수의 관계가 선형이 아닌 경우, 회귀 모델은 오차를 정규적으로 처리하지 못한다.
- 하지만, 모델은 여전히 전체 데이터의 일부 패턴을 학습하여 유의미한 결과를 낼 수 있다.

(2) 이상치(Outliers):

- 데이터셋에 이상치가 포함되면 잔차의 분포가 왜곡될 수 있다.
- 이상치는 잔차의 정규성을 심각하게 훼손하지만, t-test나 F-test에는 상대적으로 덜 민감할 수 있다.

(3) 데이터 변환의 필요성:

- 독립 변수나 종속 변수가 특정 분포(예: 비정규 분포)를 가진다면, 데이터 변환(예: 로그 변환, Box-Cox 변환)이 필요할 수 있다.
- 변환을 하지 않으면 잔차의 정규성이 깨질 가능성이 크다.

(4) 다중공선성(Multicollinearity):

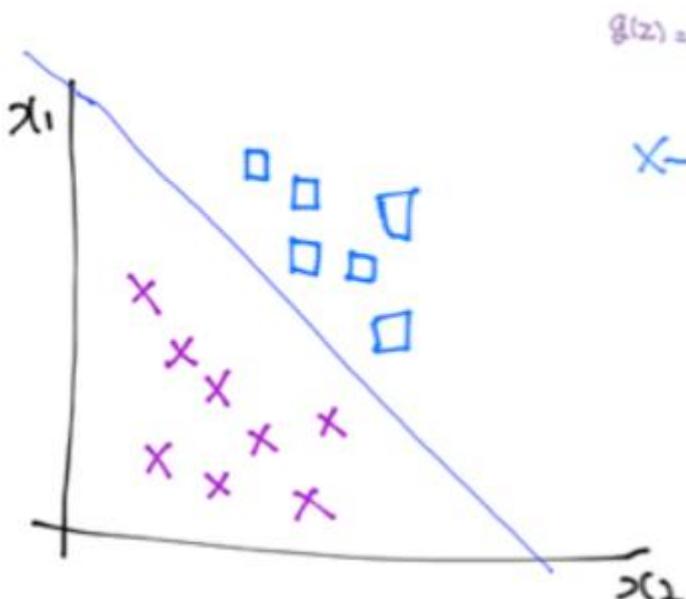
- 독립 변수들 간 강한 상관관계가 있는 경우, 잔차가 정규성을 벗어나면서도 모델 자체는 유의미한 결과를 보여줄 수 있다.

(5) 해결방안:

- 비선형 모델 사용, 이상치 제거, 데이터 변환 시도, 높은 상관관계 변수 제거, 등

Linear Classification

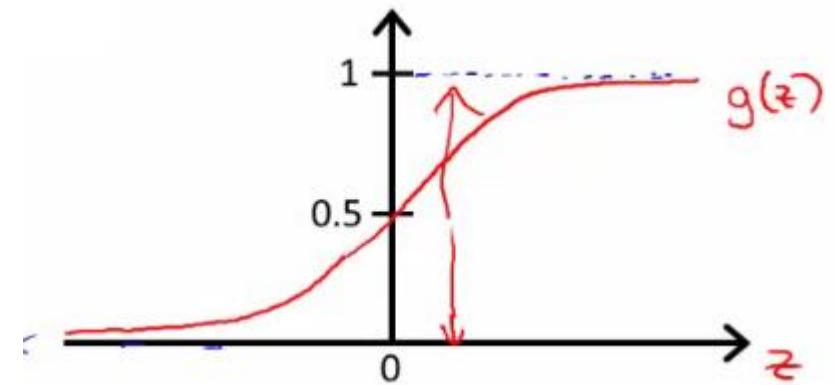
- Logistic Regression Classifier (**Binary**)



$$g(z) = \frac{1}{1 + e^{-z}} \quad H_{\theta}(x) = g(H_{\theta}(x))$$

$x \rightarrow \boxed{w} \rightarrow z \rightarrow \boxed{g} \rightarrow \tilde{Y}$

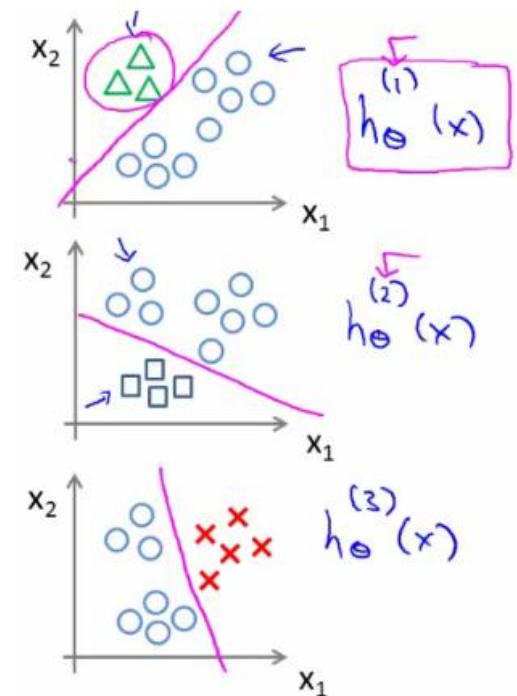
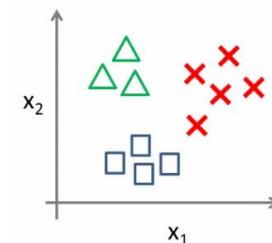
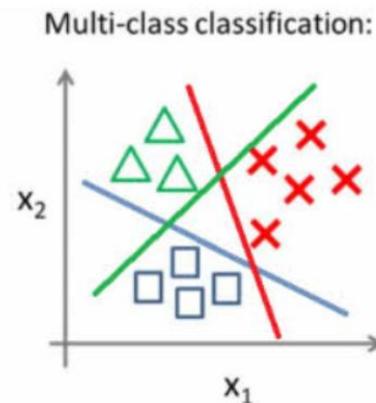
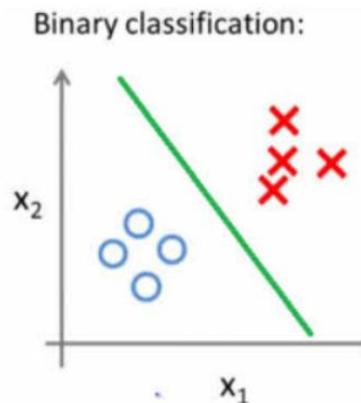
The diagram illustrates the logistic regression model architecture. An input vector x is multiplied by a weight vector w to produce a weighted sum z . This sum z is then passed through a sigmoid function g to produce the final output \tilde{Y} .



- Since this is a binary classification task we know $y = 0$ or 1
 - So the following must be true
 - $P(y=1|x ; \theta) + P(y=0|x ; \theta) = 1$
 - $P(y=0|x ; \theta) = 1 - P(y=1|x ; \theta)$

Linear Classification

- **Binary classification vs. Multi-class classification**



- Multi-class classification (with n -classes)
 - One vs. All: n binary classifier models (preferred)
but, prone to creating an imbalance
 - One vs. One: ${}_n C_2 = n*(n-1)$ binary classifier models

Performance Metrics - Regression

- **MSE and MAE**

- MSE(Mean squared error)
- MAE(Mean absolute error)
- MAE 가 이상치(outlier)에 대해서는 MSE 보다 robust 하다고 알려짐.

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

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

- **R Squared (R^2)**

- 실제 출력 값에 대한 예측 출력 값 세트의 우수성 또는 적합성 표시
- Numerator(MSE), denominator(variance)
- What if all Y_i is correctly predicted?
- What if all Y_i is predicted as the mean?

SS: sum of squares
around mean, fit

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} = \frac{SS(\text{mean}) - SS(\text{fit})}{SS(\text{mean})}$$

- $R^2=0$ means that the model predicts the expected value of y disregarding the input features.

Performance Metrics - classification

- Classification – static, dynamic
- Static: **confusion matrix** (a.k.a. Error matrix)
 - Tabular visualization of the model prediction vs. ground-truth labels
 - Row: the instances in a predicted class
 - Column: the instances in an actual class

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

- True-positive (100), false-negative (5)
- true-negative (50), false-positive (10)

Performance Metrics - classification

- **Accuracy**
 - Number of correct predictions divided by the total number of predictions
 - (ex) accuracy = $(100+50)/165$
- **Precision**
 - In some cases, accuracy is not a good indicator of your model performance, for example, when your class distribution is imbalanced.
 - Precision_yes = $100/110$, Precision_noncat = $50/55$
 - Says “**How reliable your prediction is...**” or “how much I can trust...”
- **Recall**
 - Recall_yes = $100/105$
 - Recall_no = $50/60$
 - Says “**how many actual class samples are correctly predicted...**”
- **F1-Score**
 - Combine the above two metrics (precision and recall) as harmonic mean:
$$\text{F1-score} = 2 * \text{Precision} * \text{Recall} / (\text{Precision} + \text{Recall})$$

Performance Metrics - classification

- **Dynamic: ROC Curve (Receiver Operating Characteristic curve)**
 - Shows the performance of a binary classifier as function of its cut-off threshold
 - It essentially shows the true positive rate(tpr) against the false positive rate(fpr) for various threshold values.
 - **AUC** (Area Under Curve)
- **As an example,**
 - Suppose your model predicts 4 sample images with probabilities [0.45, 0.6, 0.7, 0.3].
 - Then, depending on the threshold, you will get different labels:

cut_off=0.5: predicted_value=[0,1,1,0] (default threshold)

cut_off=0.2: predicted_value=[1,1,1,1]

cut_off=0.8: predicted_value=[0,0,0,0]

- **Making different confusion matrix depending on the threshold.**

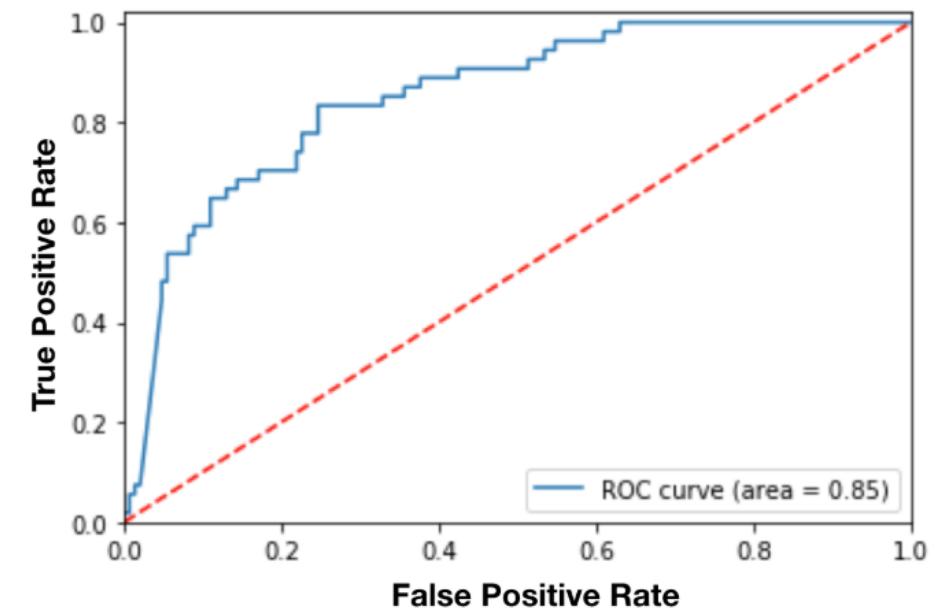
ROC/AUC (Example)

↑
↑
↑
↑
↑

환자번호	성별	점수	순위	실제 값	TPr	FPr
7	F	0.98	1	N(0)	0	$1/m$
125	M	0.96	2	C(1)	$1/n$	$2/m$
4	F	0.95	3	N	$2/n$	$3/m$
199	M	0.86	4	C	$3/n$	$4/m$
2	F	0.84	5	N	$4/n$	$4/m$
200	M	0.82	6	C	$5/n$	$5/m$
176	M	0.81	7	C	$6/n$	$6/m$
73	M	0.80	8	N
82	M	0.79	9	C
3	F	0.77	10	N	1	1
123	F	0.76	11	N		
		...		C		
43	F	0.48	198	N		
93	M	0.42	199	N		
120	F	0.40	200	N		

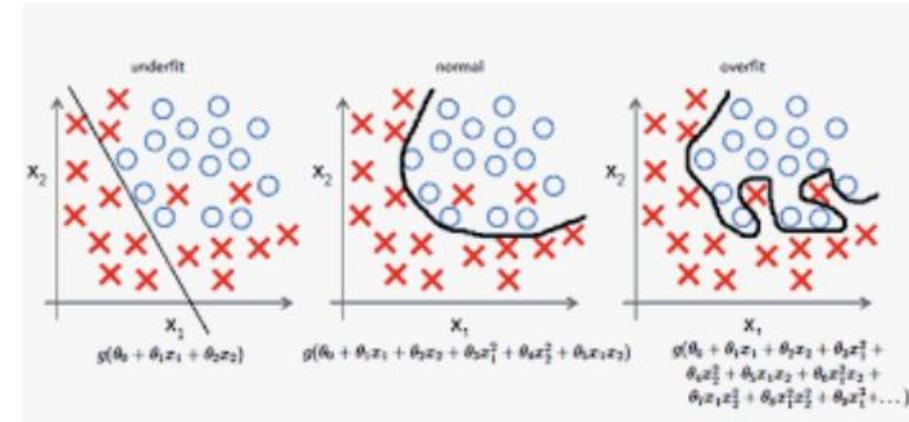
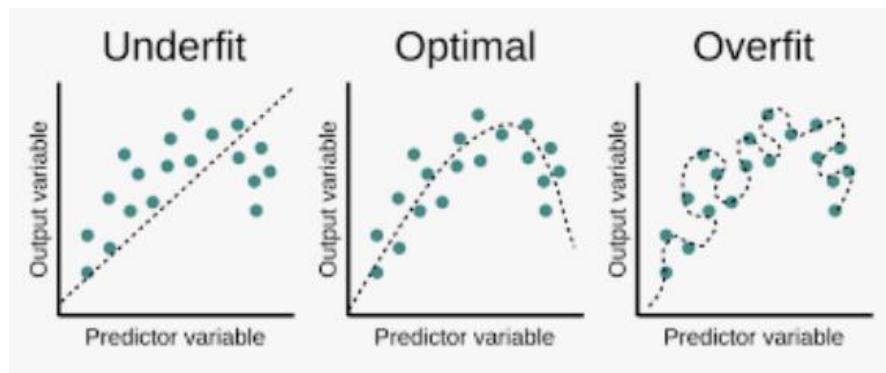
(*) score : probability of cancer

(*) depending on the threshold, you will get different confusion matrix



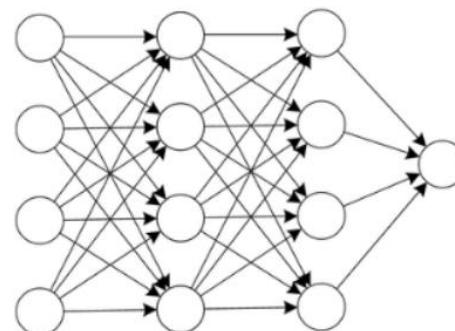
Overfitting and Underfitting

- Overfitting and Underfitting

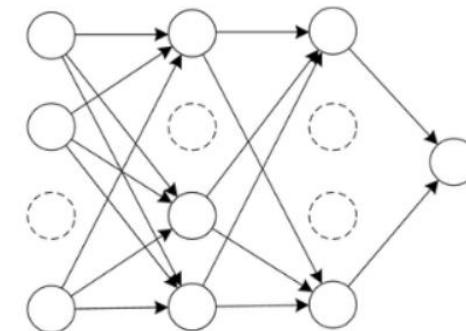


- How to reduce overfitting?

- More data
- Data augmentation
- Simplify the model
- Feature selection (or reduction)
- Regularization
- Hyperparameter tuning
- Early stopping
- Dropouts



(a) Standard Neural Network

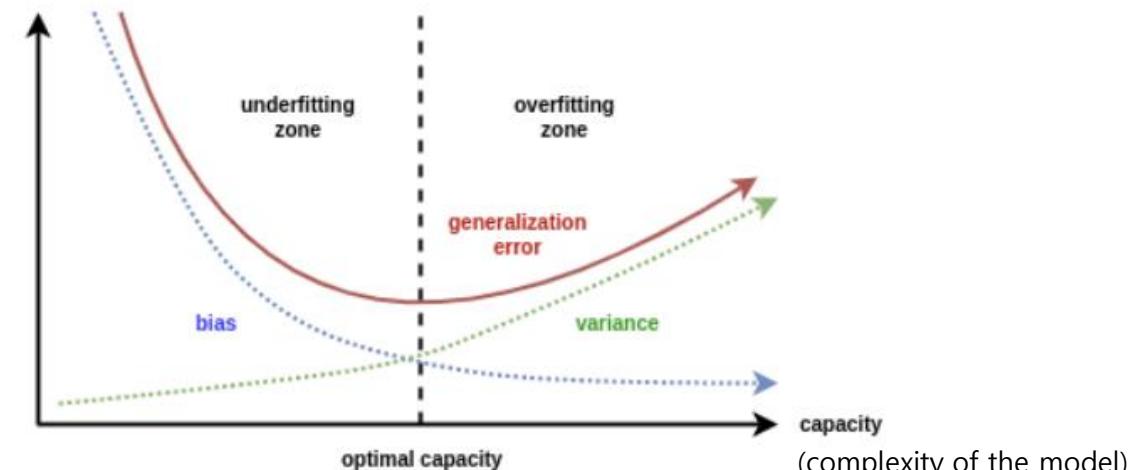
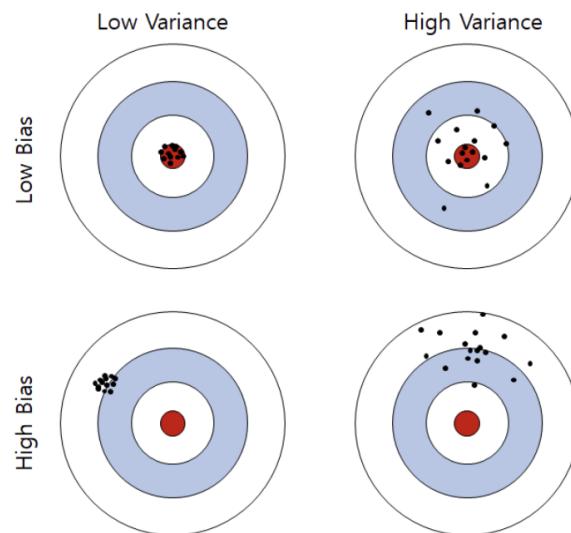


(b) Network after Dropout

Bias and Variance

- **Bias-Variance tradeoff (편향과 분산)**

- Bias: difference between the average prediction of the model and the correct value (wrong model -> **high bias** -> **underfitting**)
- Variance: variability of model prediction (noisy dataset -> **high variance** -> **overfitting**)
- **total error = Bias + Variance + noise**



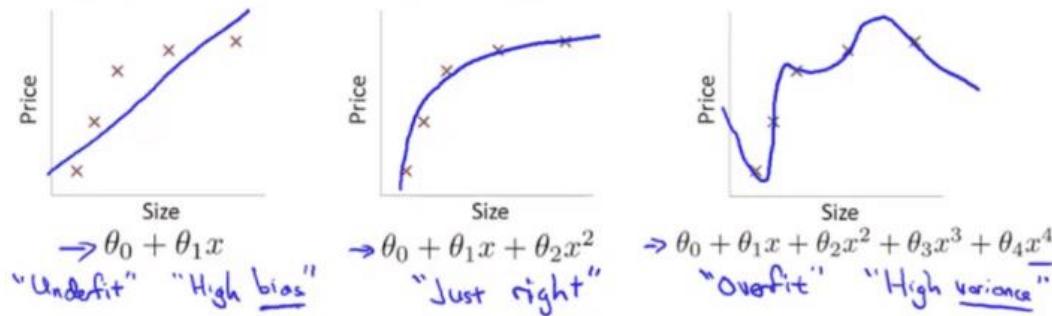
Bias-Variance tradeoff

Regularization

• Regularization (규제화)

- Add information in objective function to reduce model complexity for reducing overfitting (regression and classification)
- **Ridge** (L2): give more penalties on large-valued coefficients
- **Lasso** (L1) (Least Absolute Shrinkage and Selection Operator): shrinks the less important features' coefficient to zero (good for **feature selection**)
- **Elastic net**: use the both

Example: Linear regression (housing prices)

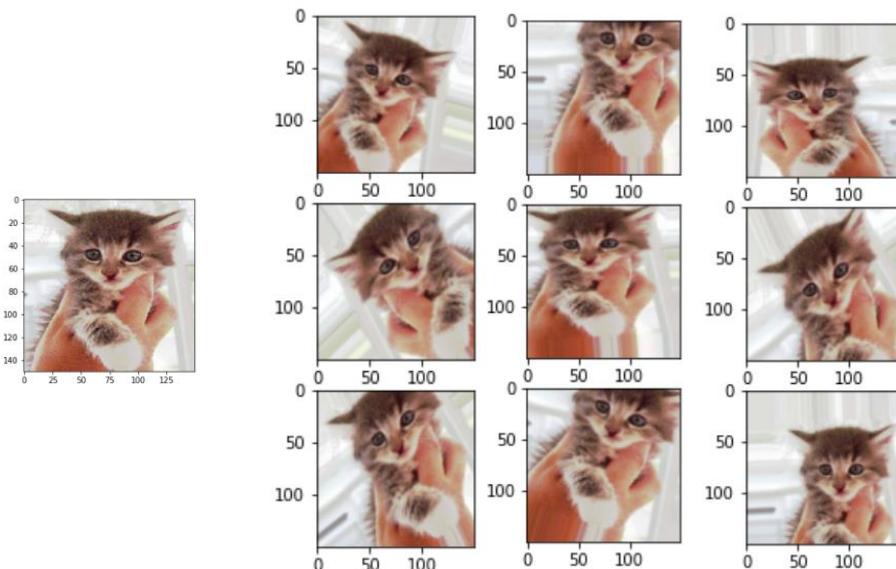


$$J(W) = MSE(W) + \alpha \frac{1}{2} \sum_{i=1}^n W_i^2$$
$$J(W) = MSE(W) + \alpha \sum_{i=1}^n |W_i|$$
$$J(\theta) = MSE(\theta) + \gamma \alpha \sum_{i=1}^n |\theta_i| + \frac{1-\gamma}{2} \alpha \sum_{i=1}^n \theta_i^2$$

Augmentation

- **Augmentation** (증강 or 확장)

- Increase the amount of data by adding **slightly modified copies** of existing data or **newly created synthetic data** from existing data
- Introducing new synthetic images: transformation, GAN, image synthesis
- Data augmentation for speech recognition based on RNN



```
1 train_datagen = ImageDataGenerator(  
2     rescale= 1./255,  
3     rotation_range = 40,  
4     width_shift_range = 0.2,  
5     height_shift_range = 0.2,  
6     shear_range=0.2,  
7     zoom_range=0.2,  
8     horizontal_flip = True)
```

Augmentation

- **Audio Data Augmentation**

- Noise injection: add Gaussian or random noise to the audio dataset
- Shifting: shift audio left (fast forward) or right with random seconds.
- Changing the speed: stretches times series by a fixed rate.
- Changing the pitch: randomly change the pitch of the audio.

- **Text Data Augmentation**

- Word or sentence shuffling: randomly changing the position of a word or sentence.
- Word replacement: replace words with synonyms.
- Syntax-tree manipulation: paraphrase the sentence using the same word
- Random word insertion: inserts words at random.
- Random word deletion: deletes words at random

- **Image Augmentation**

- Geometric transformations: randomly flip, crop, rotate, stretch, and zoom images
- Color space transformations: randomly change RGB color channels, contrast, and brightness
- Kernel filters: randomly change the sharpness or blurring of the image
- Random erasing: delete some part of the initial image
- Mixing images: blending and mixing multiple images
- Autoencoders (add random noise to the feature vector)

Imbalance Problem

- **Class imbalance problem**

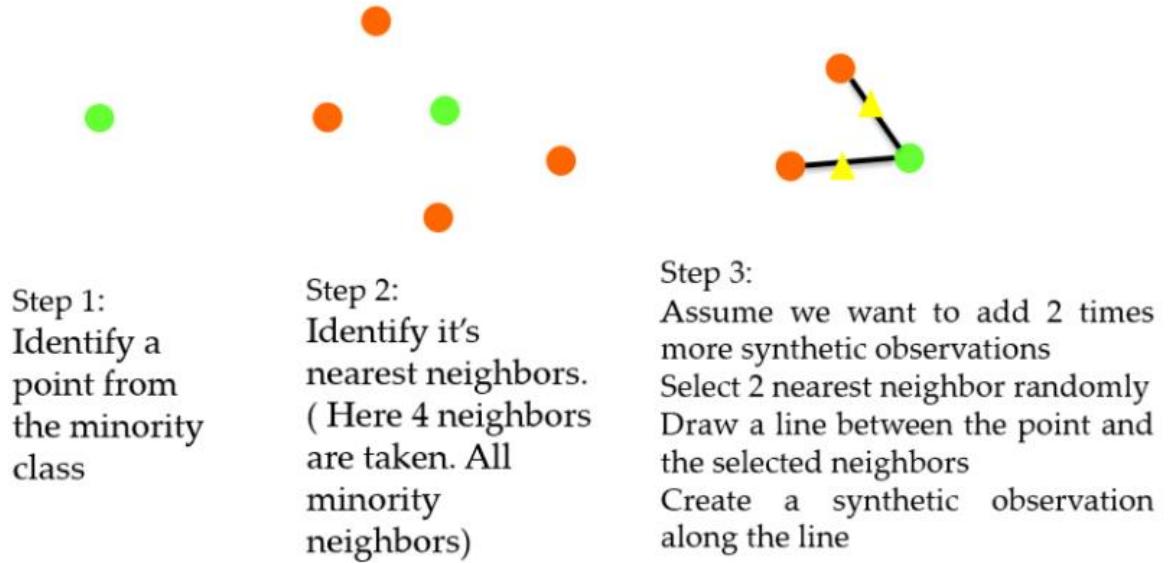
- Class distributions are highly imbalanced in the training dataset
- Tend to have low prediction accuracy for the infrequent (minority) class
- Exists in many real word classification problems, such as fraud detection, spam detection, threat-object detection, anomaly detection, etc.
- Causes: properties of the domain, biased sampling, measurement error

- **How to reduce the imbalance problem?**

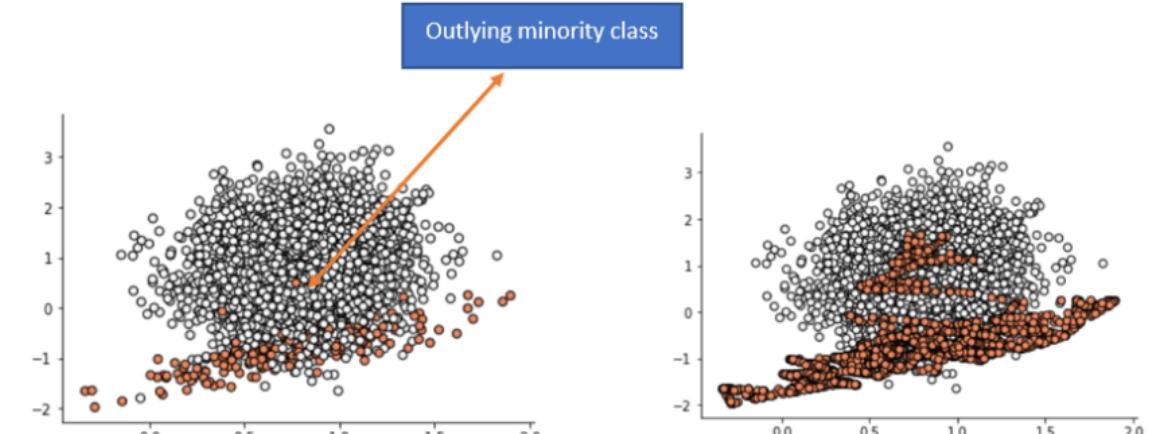
- Artificial resampling: over-sampling (replicating minority class), under-sampling of majority class
- **SMOTE**(Synthetic Minority Oversampling TEchnique): make synthetic data points by finding the nearest neighbors to each minority sample
- **Augmentation** or Use generative model for synthetic data
- More weights on minority samples
- Majority sample selection based on RL
- Resampling is to be done **only on Train dataset (not Test dataset !)**
- Still a hot research topic

Imbalance Problem

- **SMOTE** (Synthetic Minority Oversampling Technique)



SMOTE, Synthetic Minority Observation Generation Process (Source: Author)



- If there are outlying minority classes and appear in the majority class, it creates a line bridge with the majority class. (problem!)

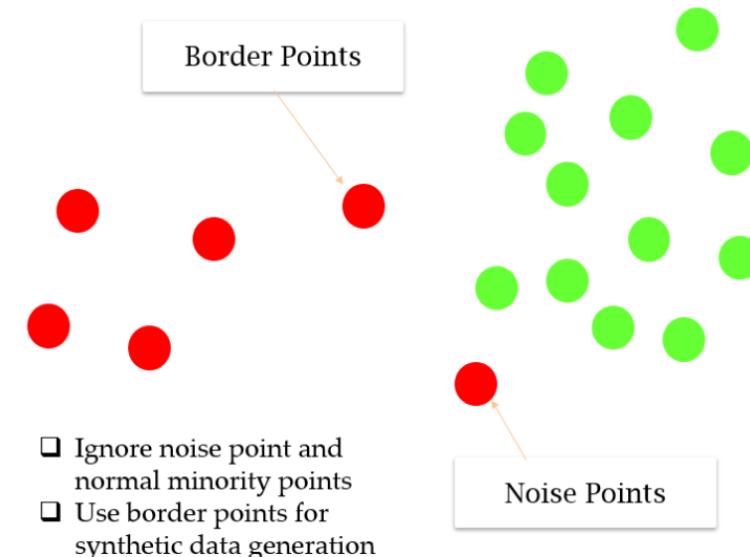
Imbalance Problem

- **Borderline SMOTE**

- Classify minority classes as **noise point** if all neighbors are the majority class – ignored.
- Classifies a few points as **border points** that have both majority and minority class.
- Resample only from border points.
- **End up giving more attention to extreme observations.**

- More variants

- ADASYN: generating more synthetic samples in areas where classification is difficult, such as where the density of minority examples is low.
- and more ...



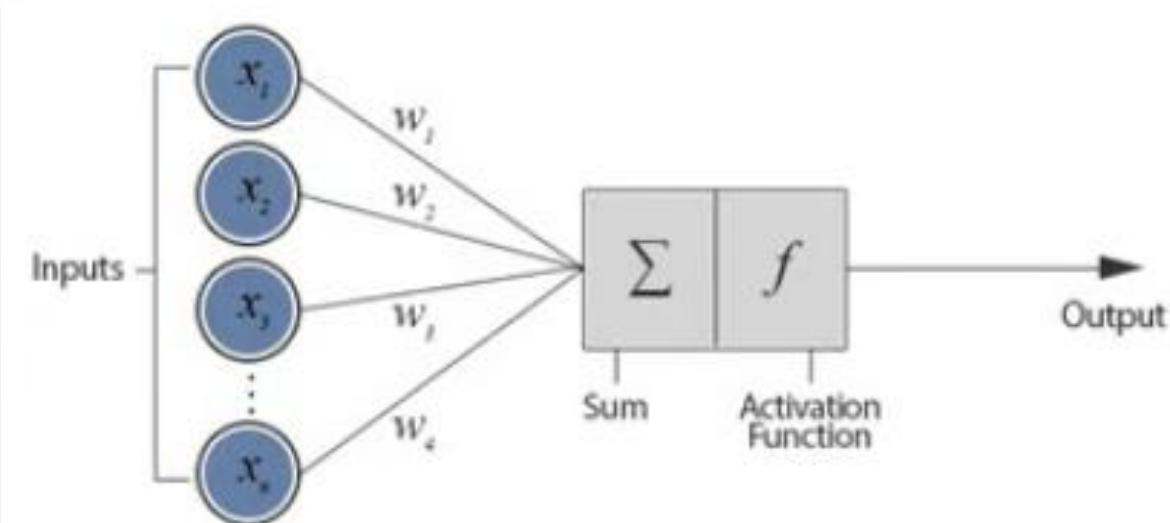
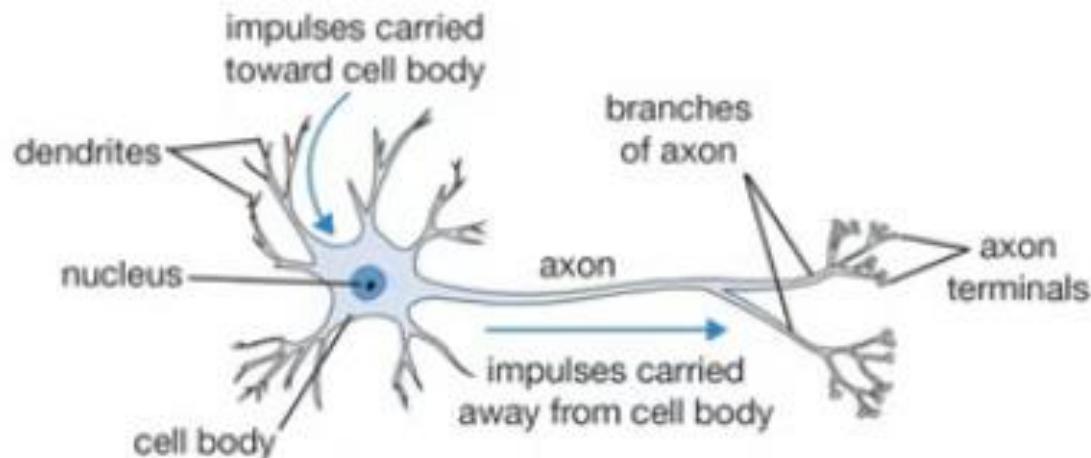
Border Line SMOTE : (Image Source Author)

Scikit-Learn design principle

- **Consistency**
 - **Estimators**: estimate some parameters based on dataset
 - `fit(X [,y])` method with some hyper-parameters
 - **Transformers**: some estimators can transform a dataset
 - `transform(X)` method
 - `fit_transform(X [,y])`: equivalent to calling `fit()` and `transform()`
 - **Predictors**: making predictions
 - `predict()` method : returns a dataset of corresponding predictions
 - `score()` method : measure the quality of the predictions
- **Inspection**
 - Hyper-parameters are accessed via instance variable (e.g. `imputer.strategy`)
 - Estimator's learned parameters are accessed via instance variable with an underscore suffix (e.g. `imputer.statistics_`)

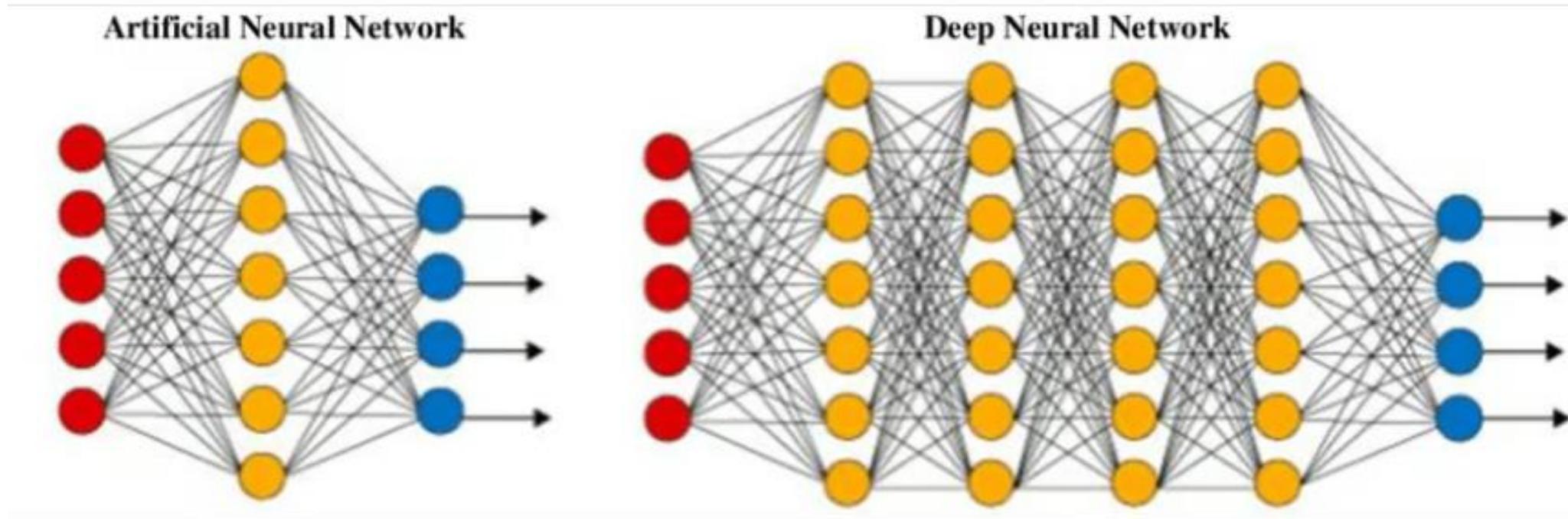
Deep Learning

Biological Neuron versus Artificial Neural Network



Biological Neuron vs. Artificial Neural Network Source: [ResearchGate](#)

Multi-Layer Perceptron (MLP)



MLP for Regression

Table 10-1. Typical Regression MLP Architecture

Hyperparameter	Typical Value
# input neurons	One per input feature (e.g., $28 \times 28 = 784$ for MNIST)
# hidden layers	Depends on the problem. Typically 1 to 5.
# neurons per hidden layer	Depends on the problem. Typically 10 to 100.
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None or ReLU/Softplus (if positive outputs) or Logistic/Tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

MLP for Classification

MLP (including ReLU and Softmax) for classification

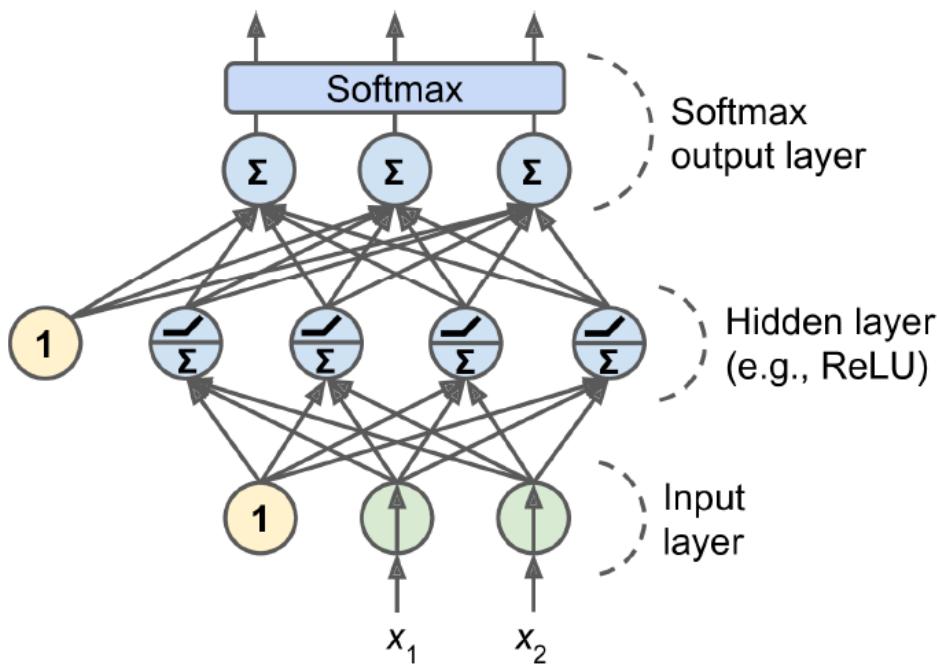
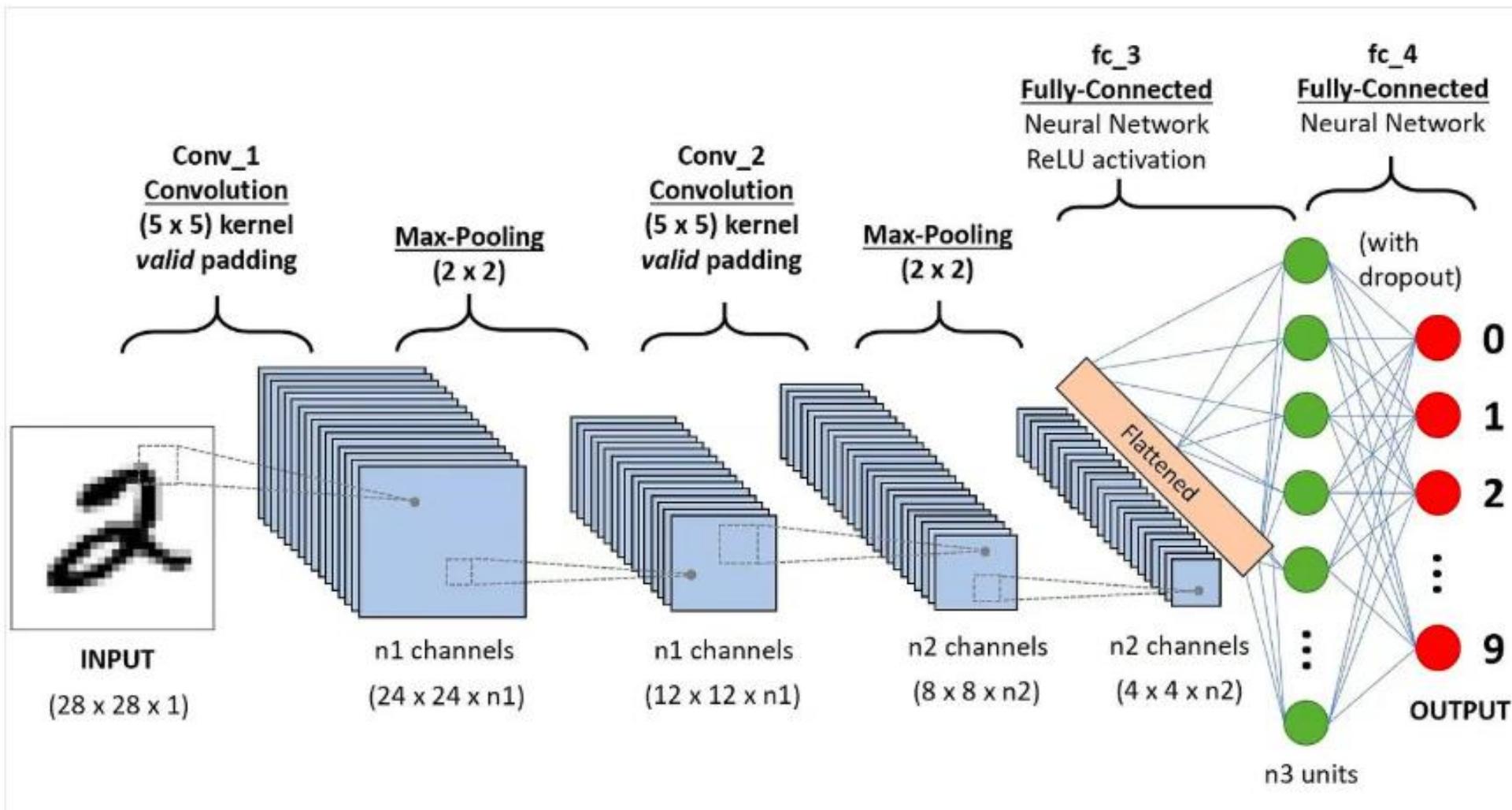


Table 10-2. Typical Classification MLP Architecture

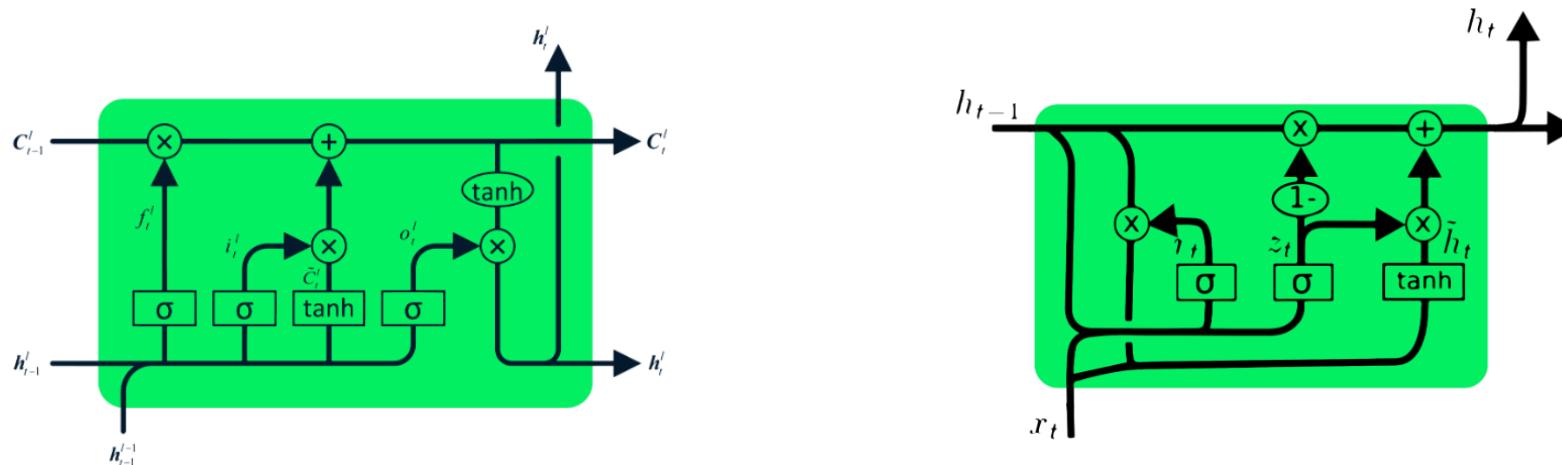
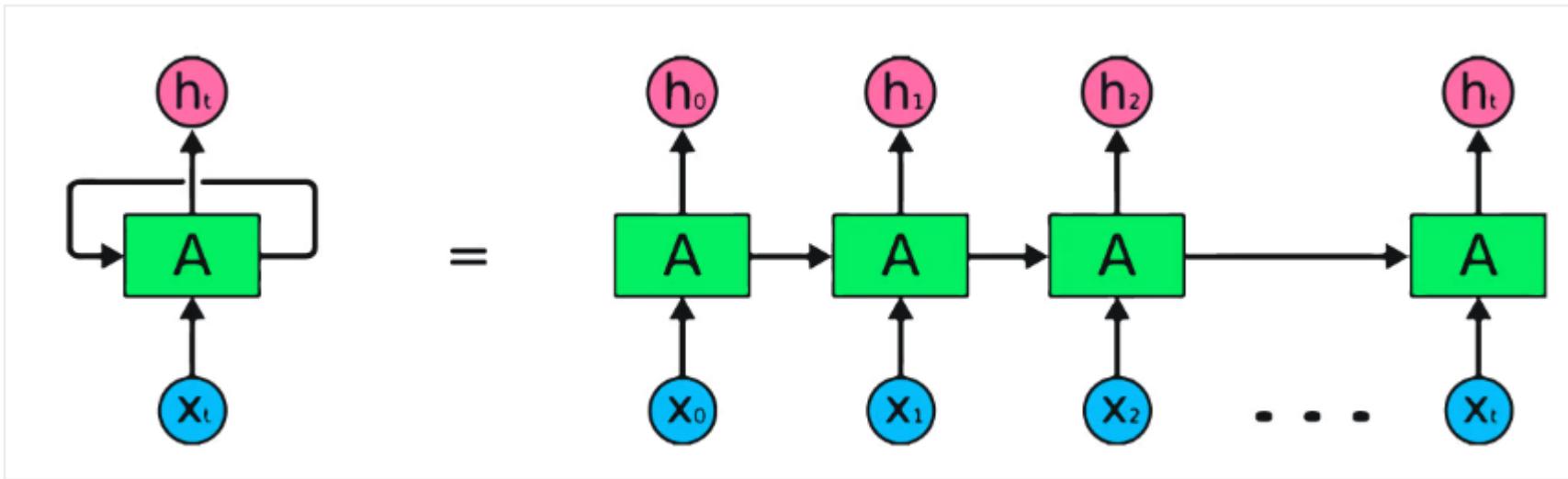
Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross-Entropy	Cross-Entropy	Cross-Entropy

multi-class: single label multi-class, (e.g. [0] or [1] or [2])
multi-label: may have more than one class label
(e.g. [1,0,1], [1,1,0])

Convolutional Neural Nets (CNN)



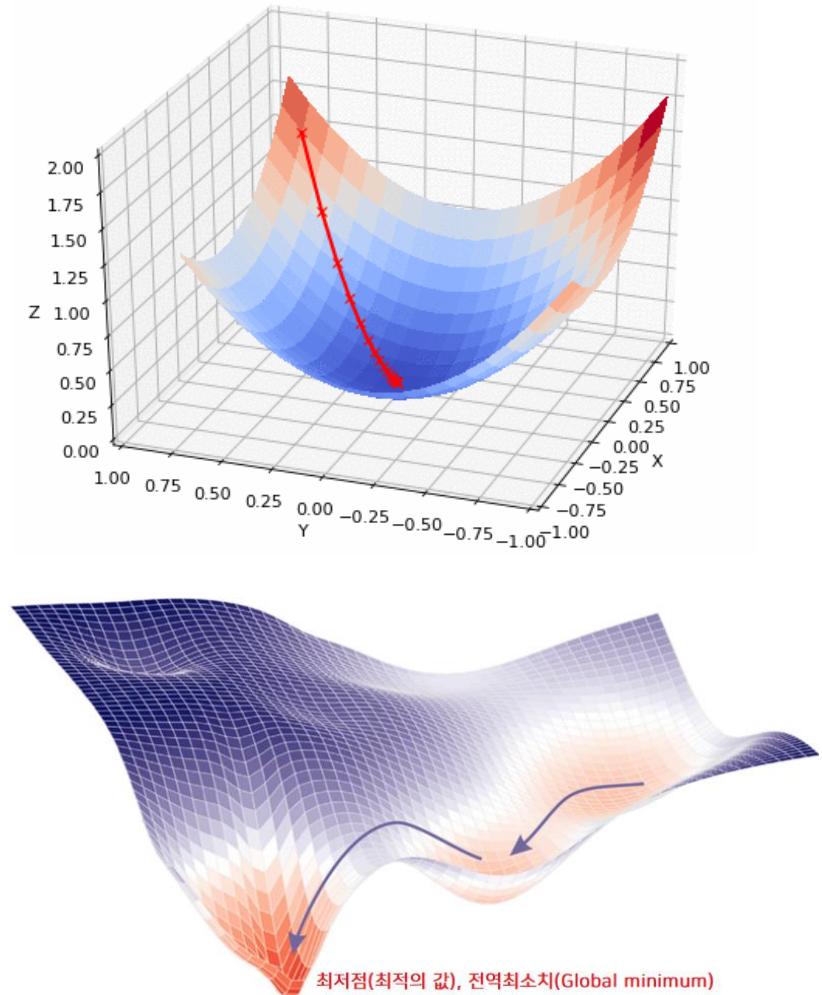
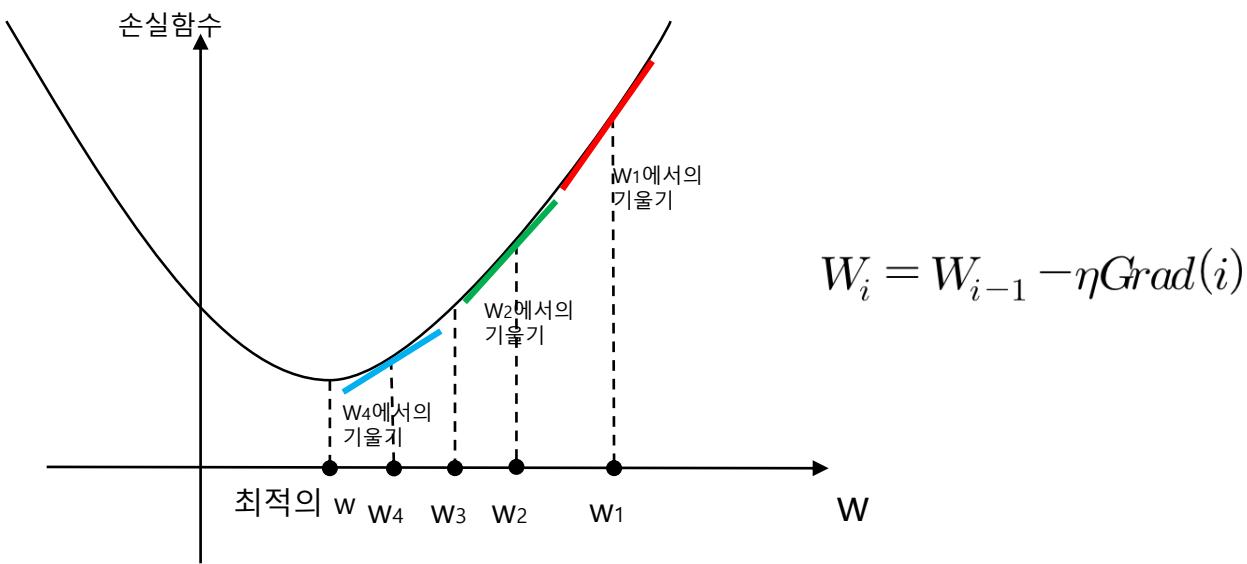
Recurrent Neural Nets (RNN), LSTM, GRU



Optimization algorithms

- **Gradient Descent** (경사하강법)

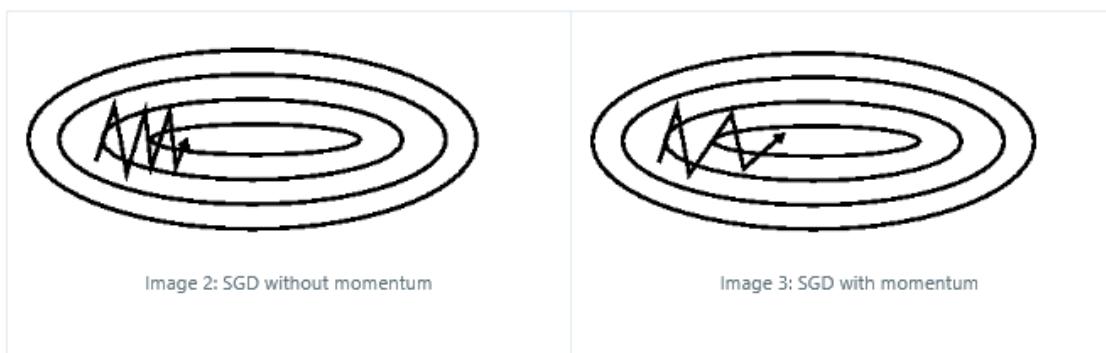
- General optimization algorithm
- take repeated steps in the opposite direction of the **gradient** (or approximate **gradient**) of the function at the current point



Optimization algorithms

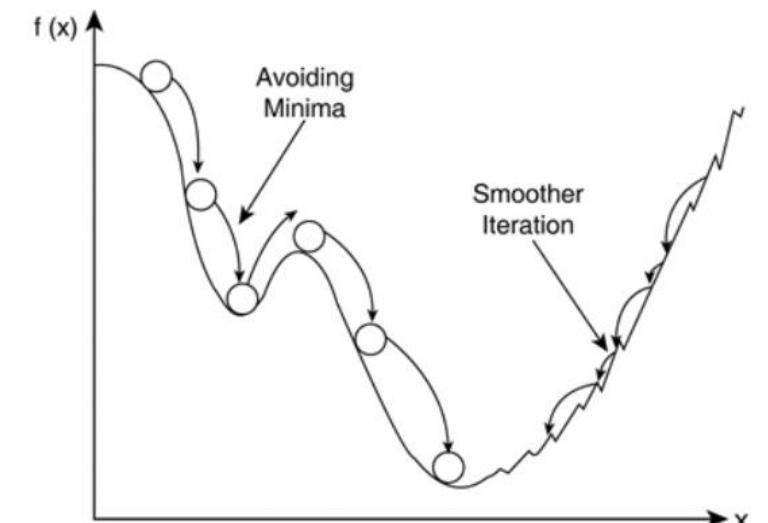
- Momentum (batch GD with Momentum)

- movement(update) vector (v) introduced
- Also use the results from previous batch (momentum term m is close to 0.9)
- Converge faster, reduced vibration, and helps to escape from local minimums.



$$V_t = m \times V_{t-1} - \eta \nabla_{\omega} J(\omega_t)$$

$$\omega_{t+1} = \omega_t + V_t$$



Avoiding Local Minima. Picture from <http://www.yaldex.com>.

Optimization algorithms

- Separate adaptive learning rates: 학습률을 Weight에 따라 다르게 함.
- **Adagrad** (Adaptive gradient)
 - 파라미터 별 update (different learning rates for ω_i 's)
 - 과거에 많이 변경되지 않은 매개 변수에 더 큰 learning rate 적용 -> step-size 감소
 - 아래 식에서 squaring 과 dot(.) 연산은 element-wise 연산

$$G_t = G_{t-1} + (\nabla_{\omega} J(\omega_t))^2 = \sum_{i=1}^k (\nabla_{\omega_i} J(\omega_i))^2$$

$$\omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_{\omega} J(\omega_t)$$

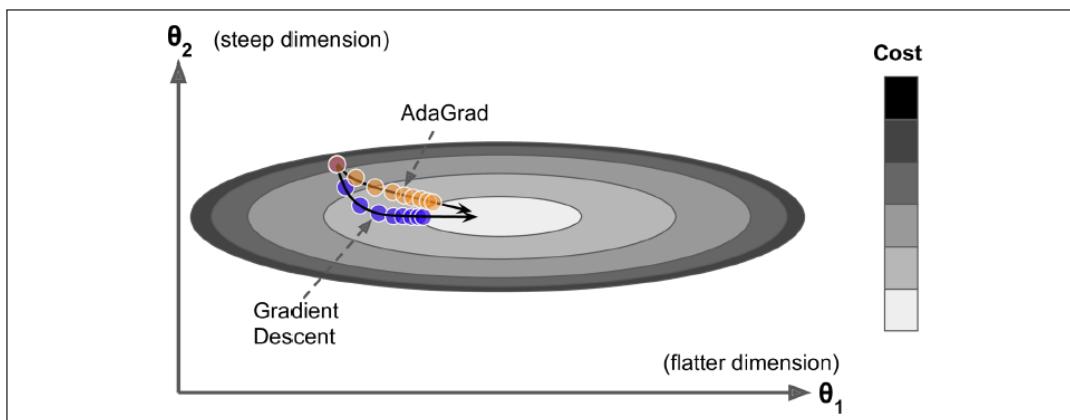


Figure 11-7. AdaGrad versus Gradient Descent

파이썬 소스 코드

```
1 | g += gradient**2  
2 | weight[i] += - learning_rate ( gradient / (np.sqrt(g) + e) )
```

Tensorflow 소스 코드

```
1 | optimizer = tf.train.AdagradOptimizer(learning_rate=0.01).minimize(loss)
```

Keras 소스 코드

```
1 | keras.optimizers.Adagrad(lr=0.01, epsilon=1e-6)
```

Ref: Hands-on-Machine Learning (2nd)

Optimization algorithms

- **RMSProp** (root mean square propagation)

- Adagrad 알고리즘은 너무 급격히 감소하여 global optimum에 도달하지 못하는 경우 발생
- 처음부터 모든 gradient G_t 를 합산하는 대신 **지수 평균** (exponential moving average) 사용하여 최근 것 사용 (more weights on the recent gradient): typical decay rate $\gamma = 0.9 \sim 0.999$
- **Always performs much better than Adagrad, and most preferred until Adam came around.**

$$G_t = \gamma G_{t-1} + (1 - \gamma)(\nabla_{\omega} J(\omega_t))^2$$

$$\omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_{\omega} J(\omega_t)$$

```
optimizer = keras.optimizers.RMSprop(lr=0.001, rho=0.9)
```



gamma

Optimization algorithms

- **Adam** (Adaptive Moment Estimation)

- RMSProp + Momentum
- Momentum 과 유사하게 기울기의 지수 평균 반영 (과거 기울기 (past gradient) 의 지수적 으로 감소하는 평균을 유지)
- RMSProp 과 유사하게 기울기 제곱 값의 지수 평균 반영 (각 매개 변수에 대한 적응형 학습 률 (adaptive learning rate))
- Recommended values: $\epsilon=10^{-8}$, $\beta_1=0.9$, $\beta_2=0.999$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_w J(\omega_t) \quad \text{first moment}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_w J(\omega_t))^2 \quad \text{second moment}$$

$$\omega_{t+1} = \omega_t - m_t \frac{\eta}{\sqrt{v_t + \epsilon}}$$

- bias correction terms (exponential decay)
- 일반적으로 m, v 가 0 으로 초기화되어 있기 때문에 훈련 초기에 크게 bias 되는데 이를 교정하기 위해 bias correction term 이 필요함.
- 훈련 iteration 이 진행 되면서 bias correction term 은 1 로 접근하게 됨.

$$\begin{aligned} M(t) &= \beta_1 M(t-1) + (1 - \beta_1) \frac{\partial}{\partial w(t)} \text{Cost}(w(t)) \\ V(t) &= \beta_2 V(t-1) + (1 - \beta_2) \left(\frac{\partial}{\partial w(i)} \text{Cost}(w(i)) \right)^2 \\ \hat{M}(t) &= \frac{M(t)}{1 - \beta_1^t} \quad \hat{V}(t) = \frac{V(t)}{1 - \beta_2^t} \\ W(t+1) &= W(t) - \alpha * \frac{\hat{M}(t)}{\sqrt{\hat{V}(t) + \epsilon}} \end{aligned}$$

Optimization algorithms

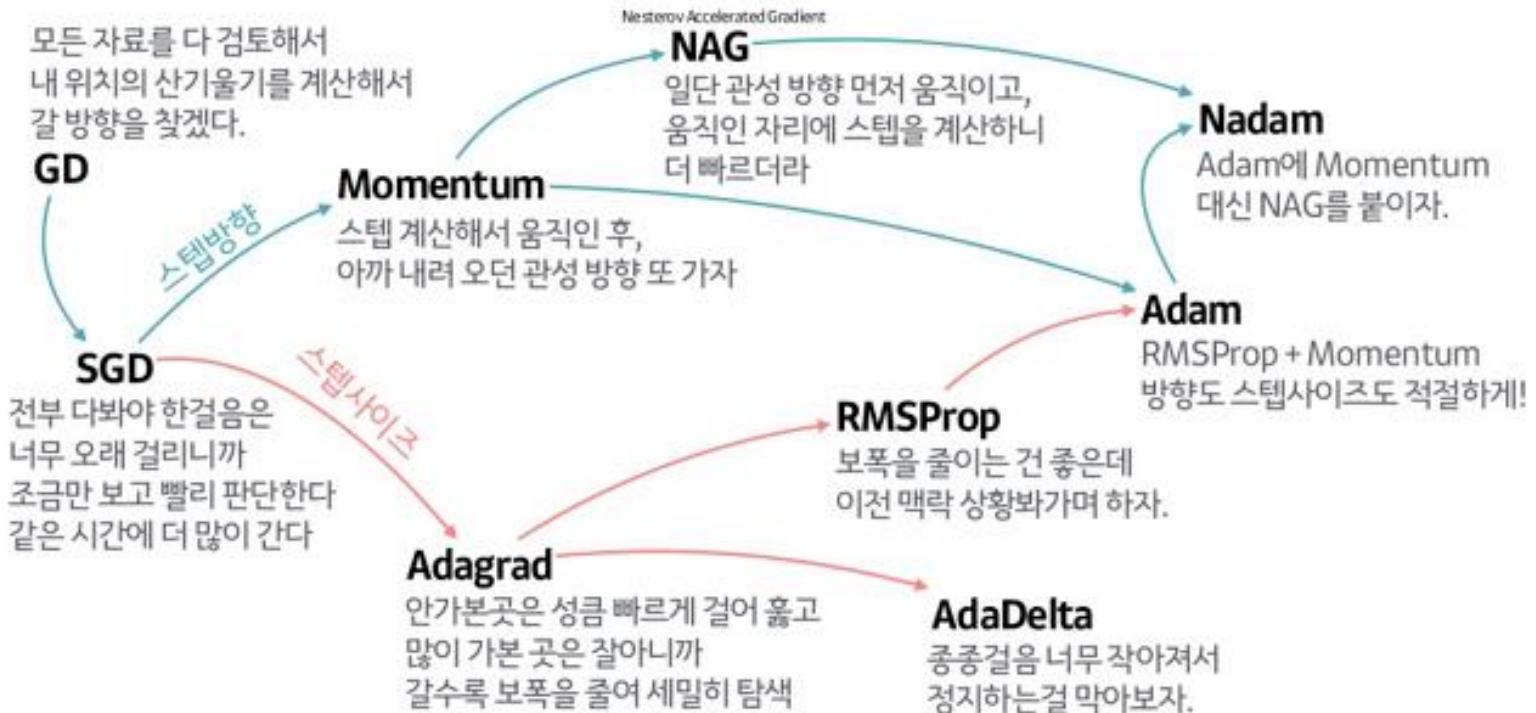
- Summary

- More:

- Adamax
- Nadam
- AMSGrad
- more ...

- Good animation

- <https://towardsdatascience.com/complete-guide-to-adam-optimization-1e5f29532c3d>



Building a Model with Keras

- **Sequential API**

- Create the model

```
model = keras.models.Sequential()  
model.add(keras.layers.Flatten(input_shape=[28, 28]))  
model.add(keras.layers.Dense(300, activation="relu"))  
model.add(keras.layers.Dense(100, activation="relu"))  
model.add(keras.layers.Dense(10, activation="softmax"))
```

```
model = keras.models.Sequential([  
    keras.layers.Flatten(input_shape=[28, 28]),  
    keras.layers.Dense(300, activation="relu"),  
    keras.layers.Dense(100, activation="relu"),  
    keras.layers.Dense(10, activation="softmax")  
])
```

- Compile the Model (specify loss function and optimizer, [extra metrics])
- Training and evaluating
- Make prediction

Building a Model with Keras

- Sequential API

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])

>>> history = model.fit(X_train, y_train, epochs=30,
...                      validation_data=(X_valid, y_valid))

>>> model.evaluate(X_test, y_test)
8832/10000 [=====] - ETA: 0s - loss: 0.4074 - acc: 0.8540
[0.40738476498126985, 0.854]

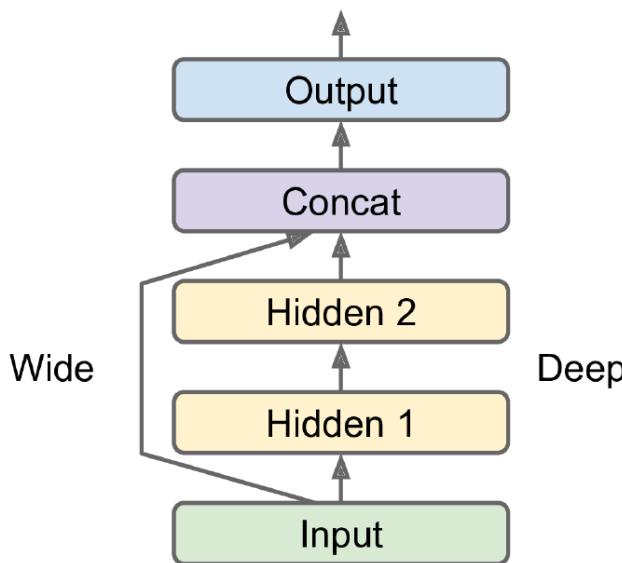
>>> X_new = X_test[:3]
>>> y_proba = model.predict(X_new)
>>> y_proba.round(2)
array([[0.   , 0.   , 0.   , 0.   , 0.   , 0.09, 0.   , 0.12, 0.   , 0.79],
       [0.   , 0.   , 0.94, 0.   , 0.02, 0.   , 0.04, 0.   , 0.   , 0.   ],
       [0.   , 1.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   ]],
      dtype=float32)
```

(*) sparse_categorical_crossentropy: when the target is integer (sparse) labeled (not one-hot encoded)

Building a Model with Keras

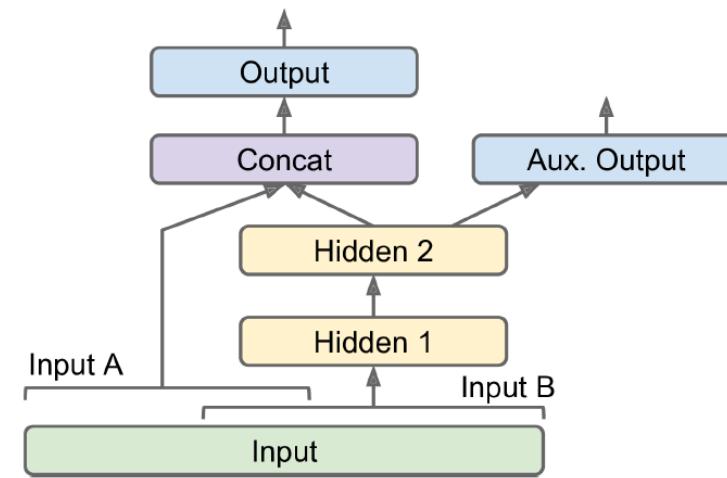
- Building complex models using the **Functional API**

Wide and deep Neural network



```
input = keras.layers.Input(shape=X_train.shape[1:])
hidden1 = keras.layers.Dense(30, activation="relu")(input)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.concatenate([input, hidden2])
output = keras.layers.Dense(1)(concat)
model = keras.models.Model(inputs=[input], outputs=[output])
```

Handling multiple outputs

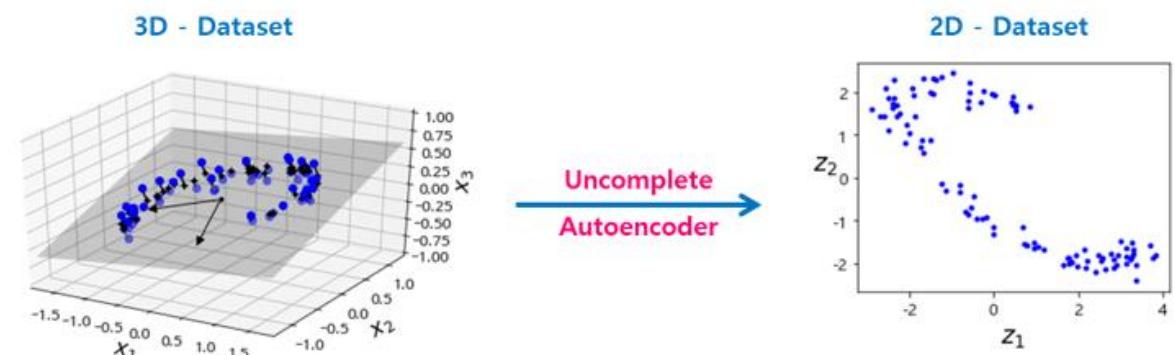
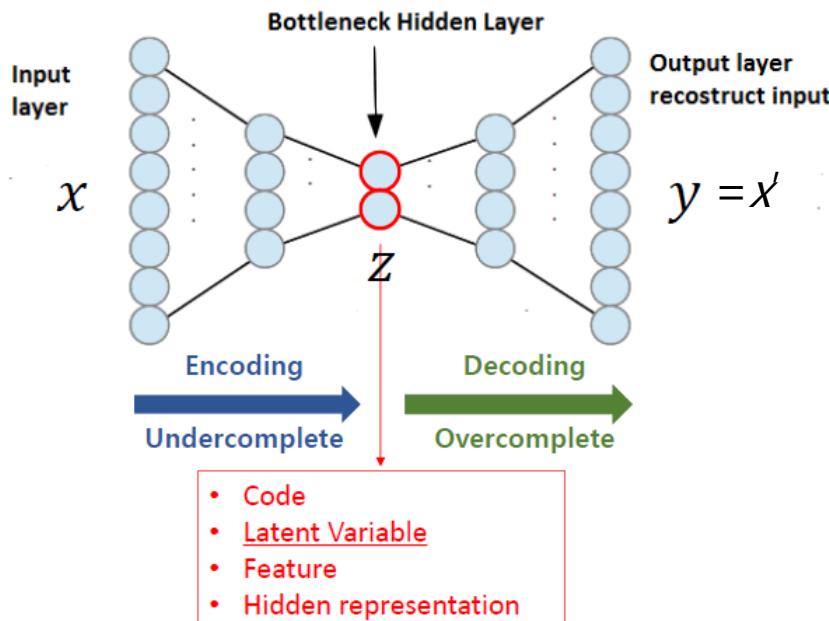


Autoencoders

- **Autoencoder**

- To learn a representation (encoding) for training data set (**dimensionality reduction**) by training the network to ignore signal “noise”, in other words, **feature extraction**.

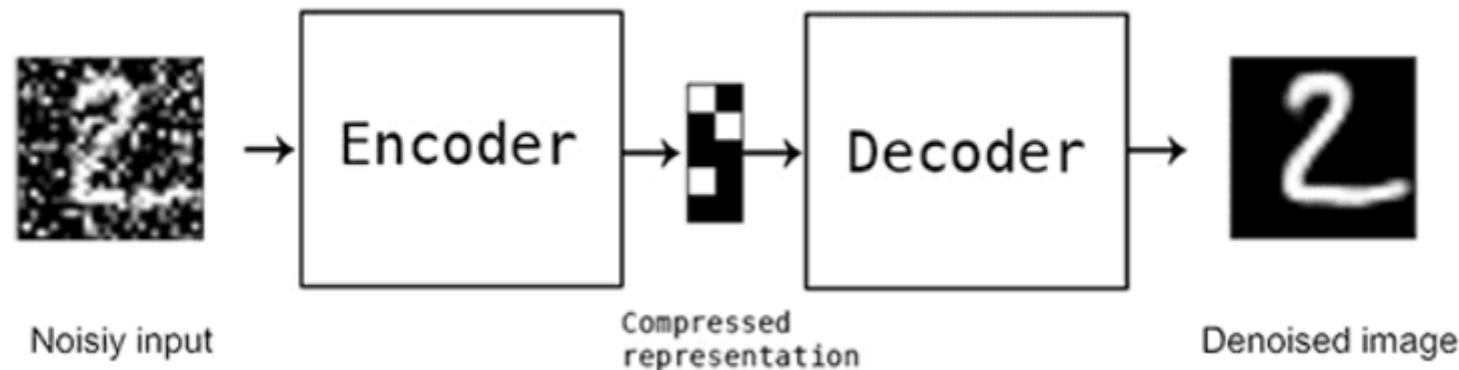
$$\mathcal{L}(x, D(E(x))) = \sum_{i=1}^m \|x^{(i)} - x'^{(i)}\|^2$$



Denoising Autoencoder

- **Denoising Autoencoder**

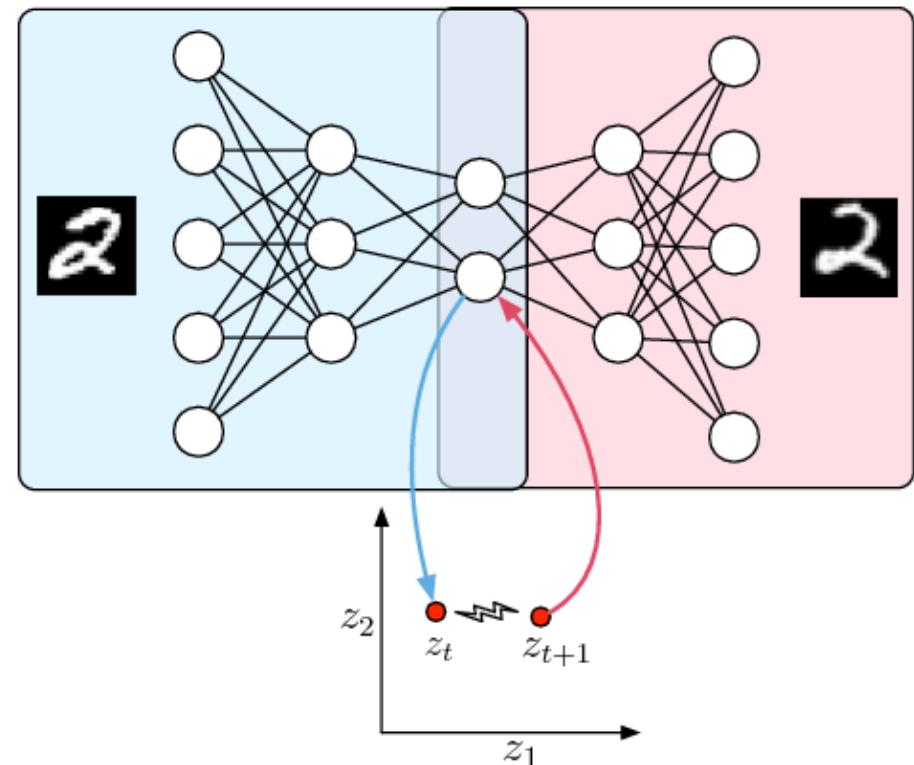
- Add noise to the input and learn to reconstruct the original input without noise
- Noise can be generated in random input units using Gaussian Noise or Dropout.



Autoencoders for Augmentation

- **Autoencoder(or VAE) for Augmentation**

- The encoder encodes the image into the latent space, providing a vector z_t .
- Then, performing some perturbation over this vector, we get the altered version z_{t+1} .
- Finally, we decode the last vector into a new image, a modified version of the original input.
- Possible perturbation: geometric transformation, pixel level alteration, occlusion or cutouts, distortions.



GPT (Generative Pre-trained Transformer)

- G (Generative):
 - The model can generate content—text, code, or ideas.
- P (Pre-trained):
 - The model has already been trained on a huge dataset before you interact with it.
- T (Transformer):
 - A "Transformer" is the core architecture or technology behind GPT, allowing it to process and understand language effectively.
 - Transformers break down text into smaller pieces (like words or phrases), understand relationships between them, and build a coherent response.

GPT

- GPT's responses come from a combination of built-in capabilities and external interactions, depending on the type of task or query.
- Built-in Capabilities:
 - General knowledge: explaining concepts, theories, and ideas
 - Language tasks: translation, summarization, paraphrasing, and text analysis
 - Programming support: writing, explaining, and debugging code in languages like Python, C++, Java, and others
 - Math and logic: Performing calculations, solving equations, or reasoning about problems
 - Hardware and software design: explaining principles, generating codes, and guiding through workflows.
- External interactions:
 - Python interpreter: for numerical computation, simulations, and data visualization.
 - Web Search: for access to recent or location-specific information (e.g., latest news, local weather)
 - Image generation (DALL-E): creates images or visual representations
 - File handling: reads and processes user-uploaded files, such as CSV, Excel, or JSON, for data analysis or visualization

Hardware Design with GPT

- **4-bit Signed Integer Adder**
- **4x4 Matrix Multiplier**
- **16-bit Floating Point Arithmetic Units (Adder, Multiplier)**
- **Pipelined Adder**
- **Finite State Machine (FSM)**
- **AXI-Lite Slave Interface (in the text)**

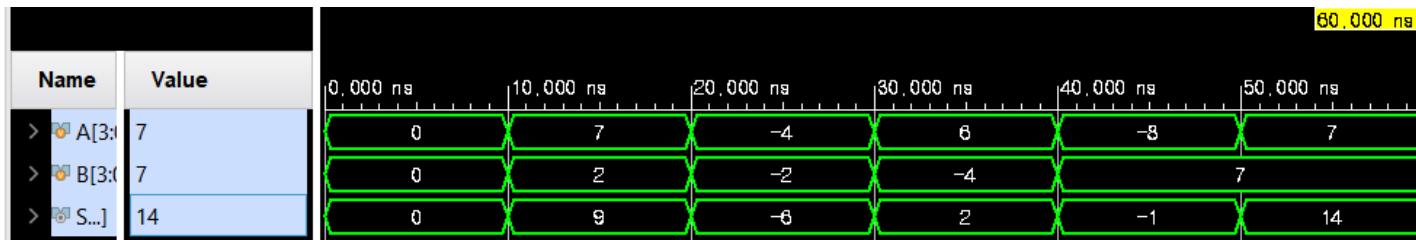
1. 4-bit Signed Integer Adder

"Design a signed 4-bit binary adder in SystemVerilog. It should have two 4-bit inputs, A and B, and one 5-bit output, SUM. Include a simple testbench that tests the addition of multiple input cases. The test case should include boundary conditions, i.e. zeros or maximum and minimum values."

systemverilog

 Copy code

```
// Signed 4-bit Binary Adder
module signed_binary_adder_4bit (
    input signed [3:0] A, // 4-bit signed input A
    input signed [3:0] B, // 4-bit signed input B
    output signed [4:0] SUM // 5-bit signed output SUM
);
    assign SUM = A + B; // Perform signed addition
endmodule
```



```
// Testbench for Signed 4-bit Binary Adder
module tb_signed_binary_adder_4bit;
    reg signed [3:0] A;      // Test signed input A
    reg signed [3:0] B;      // Test signed input B
    wire signed [4:0] SUM;   // Test signed output SUM

    // Instantiate the adder module
    signed_binary_adder_4bit uut (
        .A(A),
        .B(B),
        .SUM(SUM)
    );

    initial begin
        $display("A      B      | SUM");
        $display("-----");
        // Test case 1: Both inputs are zero
        A = 4'sb0000; B = 4'sb0000; #10;
        $display("%d + %d = %d", A, B, SUM);

        // Test case 2: Positive values
        A = 4'sb0111; B = 4'sb0010; #10; // 7 + 2
        $display("%d + %d = %d", A, B, SUM);

        // Test case 3: Negative values
        A = 4'sb1100; B = 4'sb1110; #10; // -4 + -2
        $display("%d + %d = %d", A, B, SUM);

        // Test case 4: Positive and negative values
        A = 4'sb0110; B = 4'sb1100; #10; // 6 + (-4)
        $display("%d + %d = %d", A, B, SUM);
    end
endmodule
```

2. 4x4 Matrix Multiplier

"Design a 4x4 matrix multiplier in SystemVerilog. It should have two inputs, matrix A and matrix B (each 4x4 with 8-bit elements), and output the resulting matrix C (4x4 with 16-bit elements).
Include a testbench with at least two test cases."

Why do you not use 2-D arrays to store input and output values?

Print the output matrix in matrix form, not a flattened format, for easier debugging.

Expressing the matrix indices in incremental order will make it easier to read and debug

Again, I want the matrices to be displayed in matrix form.

Based on your SystemVerilog description, how many integer multipliers are going to be generated after the synthesis?

As you mentioned, your original SystemVerilog description will be synthesized to $4 \times 4 \times 4 = 64$ integer multipliers, which is quite large. By sharing them, I want to use fewer multipliers. Can you revise it to use a single multiplier? However, it will require 64 clocks to finish the calculation.

Name	Value	0.000 ns	5.000 ns	10.000 ns	15.000 ns
> A[3:0][3:0][7:0]	{01,01,01,01}	{10,0f,0e,0d}, {0e,0b,0a,09}, ...	{01,01,01,01}, {01,01,01,01}, ...		
> B[3:0][3:0][7:0]	{01,01,01,01}	{10,0e,08,04}, {0e,09,06,03}, ...	{01,01,01,01}, {01,01,01,01}, ...		
✓ C[3:0][3:0][15:0]	{4,4,4,4},{4,4,4,4}	{600,450,300,150}, {440,330,220,110}, ...	{4,4,4,4},{4,4,4,4},{4,4,4,4}		
> [3][3:0][15:0]	4,4,4,4	600,450,300,150		4,4,4,4	
> [2][3:0][15:0]	4,4,4,4	440,330,220,110		4,4,4,4	
> [1][3:0][15:0]	4,4,4,4	280,210,140,70		4,4,4,4	
> [0][3:0][15:0]	4,4,4,4	120,90,60,30		4,4,4,4	

Matrix C (Result):

[30	60	90	120]
[70	140	210	280]
[110	220	330	440]
[150	300	450	600]

Matrix C (Result):

[4	4	4	4]
[4	4	4	4]
[4	4	4	4]
[4	4	4	4]

To reduce the number of multipliers to **1**, we can implement resource sharing by reusing a single multiplier across all the required calculations. This approach will require a state machine to sequentially compute each element of the result matrix *C*, taking 64 clock cycles to complete the full 4x4 matrix multiplication.

Here's the revised **SystemVerilog code**:

-> The state machine didn't work! Need more discussion.

IEEE 754 Standard Format

half: 5 bits

single: 8 bits

double: 11 bits

half: 10 bits

single: 23 bits

double: 52 bits

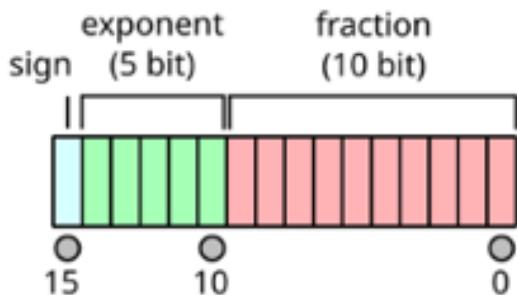
S	Exponent	Fraction
---	----------	----------

$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

- S: sign bit (0 ⇒ non-negative, 1 ⇒ negative)
- Normalize significand: $1.0 \leq |\text{significand}| < 2.0$
 - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
 - Significand is Fraction with the “1.” restored
- Exponent: excess representation: actual exponent + Bias
 - Ensures exponent is unsigned
 - Single: Bias = 127; Double: Bias = 1203; half: Bias = 15

Floating Point Numbers

- IEEE 754 16-bit (Half Precision) Floating Point Specification



- Sign bit: 1 bit
- Exponent width: 5 bits
- Significand precision: 11 bits (10 explicitly stored)

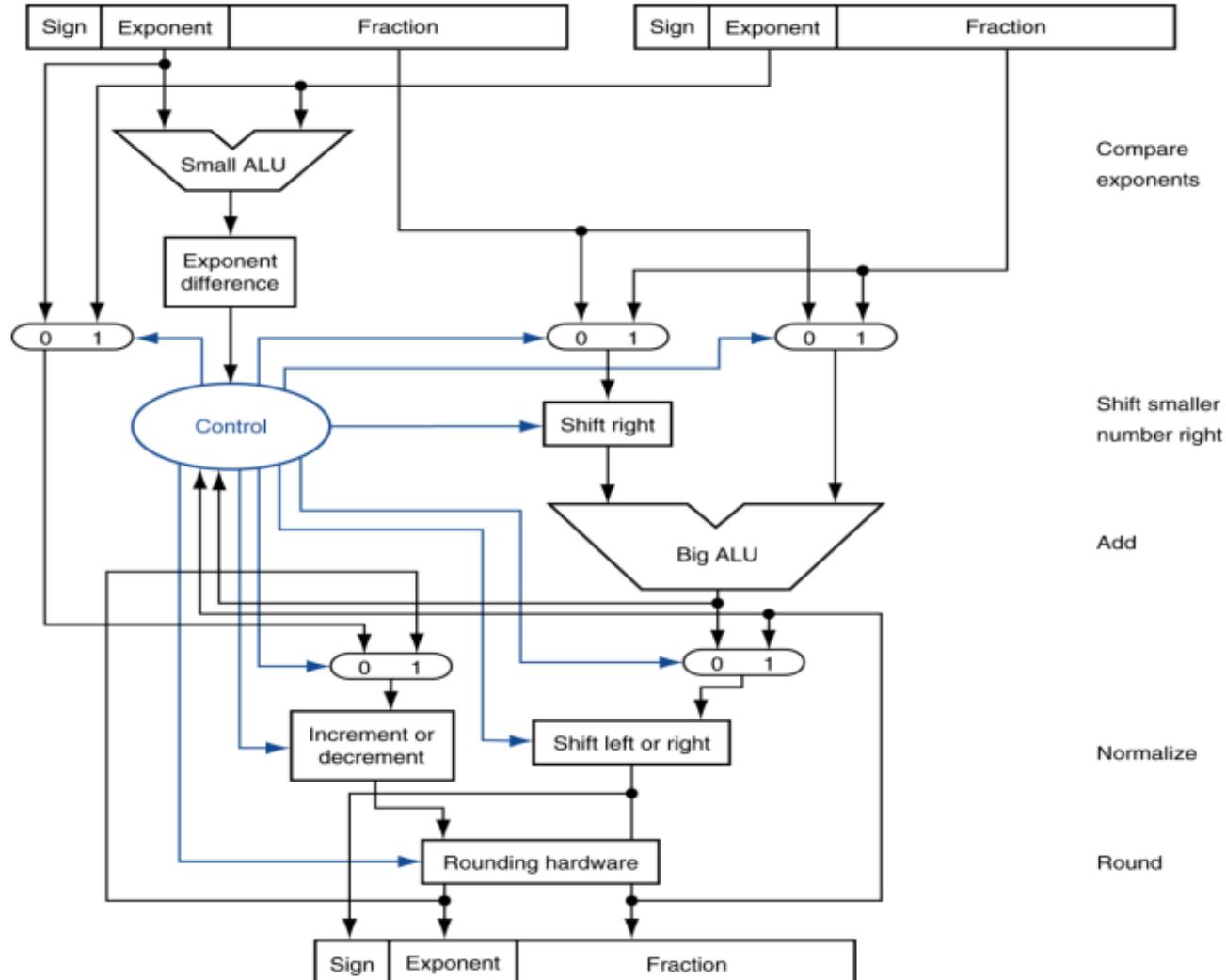
- Special cases

Number	Sign	Exponent	Fraction
0	X	00000	0000000000
∞	0	11111	0000000000
$-\infty$	1	11111	0000000000
NaN	X	11111	non-zero

Denormalized numbers: Exp = 00000000, Fraction(F) = Non-zero fractions

Floating Point Adder Hardware

- Floating Point Adder



3. FP16 Floating Point Adder/Multiplier

I want to design a 16-bit floating point adder that complies with the IEEE 754 standard using SystemVerilog HDL. I do not need to consider denormalized numbers, but I need to consider zero and infinite numbers. Start with the input F1 and F2, and the result is F3. I also need a testbench file that includes normal numbers and extreme numbers like zero and infinity. It should be synthesizable, but the testbench code may have non-synthesizable statements like 'task'. We will have to define 16-bit FP first if it is not defined in IEEE 754.

I want to have a task in the testbench that translates floating numbers into decimal fractional numbers so that I can easily debug the design. I also want to have more test cases that include normal numbers as well as extreme boundary case numbers.

The first test case gave the wrong result. It should be 3.0, which is 16'h4200, but your code gave me the result 16'h3e00.

what was wrong in your previous code and what change did you make?

can you add more test cases in the testbench. Especially I want to have more cases of normal numbers with different signs.

add these numbers: -2.3 + 1.3, 5.6 + 2.7, 0.11 + 2.11, 0.2 + (-1.4).

your code still generates the incorrect result, especially when the normal numbers are being added with infinites.

I will give show you the cases that your code generates incorrect result: F1: c1cc (-2.898438), F2: 3a66 (inf), F3: c032 (-2.097656)
F1: 455c (5.359375), F2: 42d9 (3.423828), F3: 4864 (8.781250)
F1: 2d70 (inf), F2: 4008 (2.015625), F3: 4033 (2.099609)
F1: 33d7 (inf), F2: b999 (-inf), F3: b747 (-inf). It seems that your code gives incorrect results even in the cases of finite numbers. Add a few more cases in two numbers that are all finite in your testbench.

3. FP16 Floating Point Adder/Multiplier

can you see the cases that generate incorrect result from the following text: F1: c1cc (-2.898438), F2: 3a66 (inf), F3: c032 (-2.097656)

F1: 455c (5.359375), F2: 42d9 (3.423828), F3: 4864 (8.781250)

F1: 2d70 (inf), F2: 4008 (2.015625), F3: 4033 (2.099609)

F1: 33d7 (inf), F2: b999 (-inf), F3: b747 (-inf)

F1: 3a00 (inf), F2: 3a80 (inf), F3: 3e40 (1.562500)

F1: 4200 (3.000000), F2: 4240 (3.125000), F3: 4620 (6.125000)

F1: 3e80 (1.625000), F2: be80 (-1.625000), F3: 0000 (0.000000)

F1: 3c40 (1.062500), F2: bc80 (-1.125000), F3: ac00 (-inf)?

Your code still generates incorrect results. Let's redesign the adder from the very beginning. Now, you focus only on the finite numbers. The edge cases will be considered later after the finite cases are confirmed. You got it?

The cases of the incorrect results are: F1: 3e00 (1.500000), F2: bc00 (-1.000000), F3: 3800 (inf)

F1: 3c40 (1.062500), F2: bc80 (-1.125000), F3: ac00 (-inf). can you trace them?

I think you haven't counted the hidden bit for the mantissa, have you?

can you trace the cases: F1: 3e00 (1.500000), F2: bc00 (-1.000000), F3: 3800 (inf)

F1: 3c40 (1.062500), F2: bc80 (-1.125000), F3: ac00 (-inf)?

i think 0.0625 is .0001 and .125 is .001. please check it again.

change the test cases as follows: +3.1 + 5.1, 2.3 + 4.1, 3.1 + (-5.1), 2.3 + (-4.1), -3.1 + 5.1, -2.3 + 4.1, -3.1 + (-5.1), -2.3 + (-4.1).

let's change the test cases: A=1.625, B=4.75, and the the test cases are A+B, A+(-B), (-A)+B, -A+(-B). So there will be only 4 test cases. Correct the testbench file.

Sorry. The numbers I gave are all decimal.

'h4900 is 10 in decimal, not 4.75.

3. FP16 Floating Point Adder/Multiplier

I guess you are translating the numbers incorrectly. 4.75 is 100.11 in binary, and its matissa with the hidden bit should be 1.0011 and the expomemt should be $2+15=17$, which is 10001 in binary. Please check it.

are you sure your task to translate f16 numbers into real is correct?
Please check it once again.

Now it seems to work fine. Please add a few more test cases before we consider edge numbers.

Ok. it looks good. Now, we will add functions that can handle special cases like zeroes and infinities. Please revise the design and testbench codes so that they can handle special numbers including NaN.

add a few more test cases including many combinations of finite and special numbers.

You are focusing heavily on edge cases now, which is good for now. But I want to have a few more finite-number cases.

Great. It works fine. You are such a smart AI!

We have been discussing the design of a floating point adder so far. Now you should be much more familiar with floating point numbers with a lot of test cases.

Now, I want to extend it to the design of a floating point multiplier. can you do it?

you are still giving me the incorrect result as follows: F1: 4000 (2.000000), F2: 3c00 (1.000000), F3: 4200 (3.000000)

F1: 4500 (5.000000), F2: c400 (-4.000000), F3: ce80 (-26.000000)

F1: 4600 (6.000000), F2: c500 (-5.000000), F3: cfc0 (-31.000000)

F1: 3e00 (1.500000), F2: 3d00 (1.250000), F3: 3fc0 (1.937500)

F1: 3e00 (1.500000), F2: be00 (-1.500000), F3: c240 (-3.125000).

Please trace the first one.

3. FP16 Floating Point Adder/Multiplier

Great. You have done it with just two feedbacks. You became much brighter than before.

FP16-Adder

```
// IEEE 754 16-bit Floating Point Representation Adder with Special Cases
module FP16_Adder (
    input [15:0] F1, // First floating-point operand
    input [15:0] F2, // Second floating-point operand
    output reg [15:0] F3 // Result of addition
);

// Constants for sign, exponent, and fraction bit widths
parameter SIGN_BIT = 1;
parameter EXP_BITS = 5;
parameter FRAC_BITS = 10;
parameter BIAS = 15;

// Special number detection
wire is_nan1 = (F1[14:10] == 5'b11111) && (F1[9:0] != 0);
wire is_nan2 = (F2[14:10] == 5'b11111) && (F2[9:0] != 0);
wire is_inf1 = (F1[14:10] == 5'b11111) && (F1[9:0] == 0);
wire is_inf2 = (F2[14:10] == 5'b11111) && (F2[9:0] == 0);
wire is_zero1 = (F1[14:0] == 0);
wire is_zero2 = (F2[14:0] == 0);

// Fields extraction
wire sign1 = F1[15];
wire sign2 = F2[15];
wire [EXP_BITS-1:0] exp1 = F1[14:10];
wire [EXP_BITS-1:0] exp2 = F2[14:10];
wire [FRAC_BITS:0] frac1 = (exp1 == 0) ? {1'b0, F1[9:0]} : {1'b1, F1[9:0]}; // Add implicit lead
wire [FRAC_BITS:0] frac2 = (exp2 == 0) ? {1'b0, F2[9:0]} : {1'b1, F2[9:0]}; // Add implicit lead

// Intermediate signals
reg [EXP_BITS-1:0] exp_diff;
reg [FRAC_BITS+1:0] aligned_frac1, aligned_frac2;
reg [FRAC_BITS+2:0] sum_frac;
reg [EXP_BITS:0] result_exp;
reg result_sign;
```

```
always @(*) begin
    // Handle special cases
    if (is_nan1 || is_nan2) begin
        F3 = 16'h7FFF; // NaN
    end else if (is_inf1 && is_inf2) begin
        if (sign1 == sign2) begin
            F3 = {sign1, 5'b11111, 10'b0}; // +Inf or -Inf
        end else begin
            F3 = 16'h7FFF; // NaN (inf - inf)
        end
    end else if (is_inf1) begin
        F3 = F1; // Return the infinity
    end else if (is_inf2) begin
        F3 = F2; // Return the infinity
    end else if (is_zero1) begin
        F3 = F2; // 0 + anything = anything
    end else if (is_zero2) begin
        F3 = F1; // anything + 0 = anything
    end else begin
        // Align exponents by shifting fractions
        if (exp1 > exp2) begin
            exp_diff = exp1 - exp2;
            aligned_frac1 = {frac1, 1'b0};
            aligned_frac2 = {frac2, 1'b0} >> exp_diff;
            result_exp = exp1;
        end else begin
            exp_diff = exp2 - exp1;
            aligned_frac1 = {frac1, 1'b0} >> exp_diff;
            aligned_frac2 = {frac2, 1'b0};
            result_exp = exp2;
        end
    end
    // Add or subtract fractions based on signs
    if (sign1 == sign2) begin
        sum_frac = aligned_frac1 + aligned_frac2;
        result_sign = sign1;
    end else begin
        if (aligned_frac1 >= aligned_frac2) begin
            sum_frac = aligned_frac1 - aligned_frac2;
            result_sign = sign1;
        end else begin
            sum_frac = aligned_frac2 - aligned_frac1;
            result_sign = sign2;
        end
    end
end
```

FP16-Adder

```
// Normalize result
if (sum_frac[FRAC_BITS+2]) begin
    result_exp = result_exp + 1;
    sum_frac = sum_frac >> 1;
end
while (!sum_frac[FRAC_BITS+1] && result_exp > 0) begin
    sum_frac = sum_frac << 1;
    result_exp = result_exp - 1;
end

// Ensure proper underflow handling for small results
if (result_exp == 0 && sum_frac[FRAC_BITS+1] == 0) begin
    F3 = 16'b0; // Zero result
end else if (result_exp > 0) begin
    // Preserve the implicit bit in the final result
    F3 = {result_sign, result_exp[4:0], sum_frac[FRAC_BITS:1]};
end else begin
    F3 = 16'b0; // Proper underflow to zero
end
end
end
endmodule
```

```

// Testbench for the FP16_Adder with Special Cases
module FP16_Adder_tb;
    reg [15:0] F1, F2;
    wire [15:0] F3;

    FP16_Adder uut (
        .F1(F1),
        .F2(F2),
        .F3(F3)
    );

    // Task to display results in floating-point decimal format
    task display_result;
        input [15:0] in1, in2, out;
        real val1, val2, val3;
        begin
            val1 = convert_to_real(in1);
            val2 = convert_to_real(in2);
            val3 = convert_to_real(out);
            $display("F1: %h (%f), F2: %h (%f), F3: %h (%f)", in1, val1, in2, val2, out, val3);
        end
    endtask

    // Task to convert 16-bit floating-point to real
    function real convert_to_real;
        input [15:0] fp;
        reg sign;
        reg [4:0] exponent;
        reg [10:0] fraction;
        real value;
        begin
            sign = fp[15];
            exponent = fp[14:10];
            fraction = fp[9:0];
            if (exponent == 0) begin
                value = 0.0;
            end else if (exponent == 5'b11111) begin
                value = (fraction == 0) ? ((sign) ? -1.0/0.0 : 1.0/0.0) : 0.0/0.0; // Inf or NaN
            end else begin
                value = (1.0 + (fraction / 1024.0)) * (2.0 ** (exponent - 15));
                value = (sign) ? -value : value;
            end
            convert_to_real = value;
        end
    endfunction
endmodule

initial begin
    // Test special cases
    F1 = 16'h7C00; // +Inf
    F2 = 16'h7C00; // +Inf
    #10 display_result(F1, F2, F3);

    F1 = 16'h7C00; // +Inf
    F2 = 16'hFC00; // -Inf
    #10 display_result(F1, F2, F3);

    F1 = 16'hFFFF; // NaN
    F2 = 16'h7C00; // +Inf
    #10 display_result(F1, F2, F3);

    F1 = 16'h0000; // +0
    F2 = 16'h8000; // -0
    #10 display_result(F1, F2, F3);

    F1 = 16'h3D00; // 1.625
    F2 = 16'h0000; // +0
    #10 display_result(F1, F2, F3);

    F1 = 16'h8000; // -0
    F2 = 16'h4680; // 4.75
    #10 display_result(F1, F2, F3);

    // Additional finite-number combinations
    F1 = 16'h3C00; // 1.5
    F2 = 16'h4200; // 2.5
    #10 display_result(F1, F2, F3);

    F1 = 16'h4000; // 2.0
    F2 = 16'h3C00; // 1.5
    #10 display_result(F1, F2, F3);

    F1 = 16'h4500; // 5.0
    F2 = 16'hC400; // -4.0
    #10 display_result(F1, F2, F3);

    F1 = 16'h4600; // 7.0
    F2 = 16'hC500; // -5.0
    #10 display_result(F1, F2, F3);

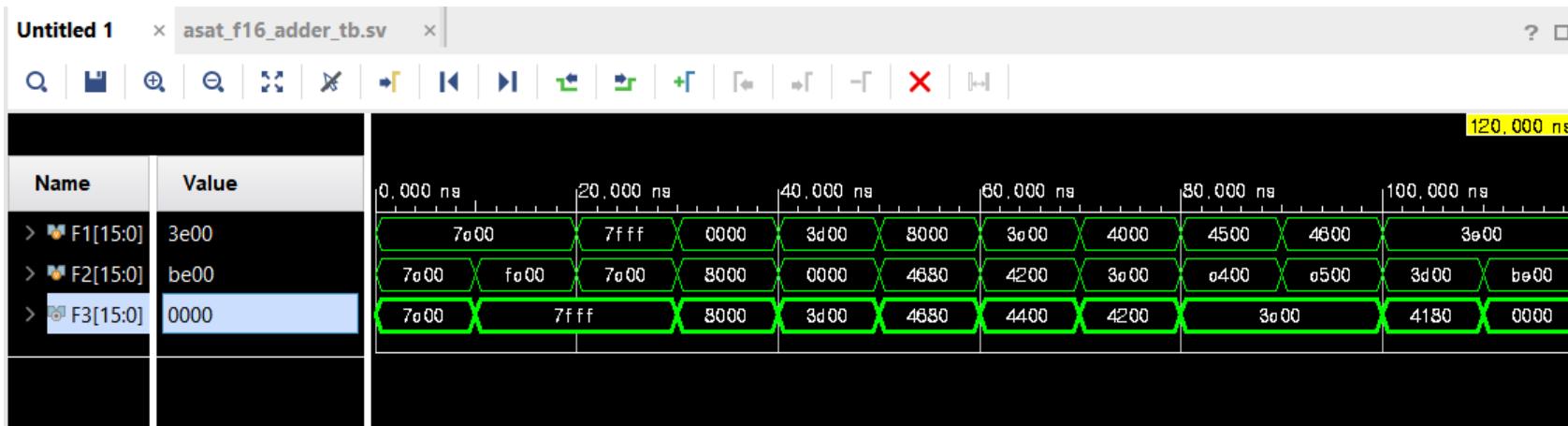
    F1 = 16'h3E00; // 1.75
    F2 = 16'h3D00; // 1.625
    #10 display_result(F1, F2, F3);

    F1 = 16'h3E00; // 1.75
    F2 = 16'hBE00; // -1.75
    #10 display_result(F1, F2, F3);

    // Finish simulation
    $stop;
end

```

FP16-Adder



Tcl Console x Messages Log

```
# run 1000ns
F1: 7c00 (inf), F2: 7c00 (inf), F3: 7c00 (inf)
F1: 7c00 (inf), F2: fc00 (-inf), F3: 7fff (nan)
F1: 7fff (nan), F2: 7c00 (inf), F3: 7fff (nan)
F1: 0000 (0.000000), F2: 8000 (0.000000), F3: 8000 (0.000000)
F1: 3d00 (1.250000), F2: 0000 (0.000000), F3: 3d00 (1.250000)
F1: 8000 (0.000000), F2: 4680 (6.500000), F3: 4680 (6.500000)
F1: 3c00 (1.000000), F2: 4200 (3.000000), F3: 4400 (4.000000)
F1: 4000 (2.000000), F2: 3c00 (1.000000), F3: 4200 (3.000000)
F1: 4500 (5.000000), F2: c400 (-4.000000), F3: 3c00 (1.000000)
F1: 4600 (6.000000), F2: c500 (-5.000000), F3: 3c00 (1.000000)
F1: 3e00 (1.500000), F2: 3d00 (1.250000), F3: 4180 (2.750000)
F1: 3e00 (1.500000), F2: be00 (-1.500000), F3: 0000 (0.000000)
$stop called at time : 120 ns : File "C:/2024_Asat_Xilinx/asat_f16_adder/asat_f16_adder.srcs/sim_1/new/asat_f16_adder_tb.sv" Line 189
INFO: [USF-XSim-96] XSim completed. Design snapshot 'FP16_Adder_tb_behav' loaded.
```

FP16-Multiplier

```
// IEEE 754 16-bit Floating Point Representation Multiplier with Special Cases
module FP16_Multiplier (
    input [15:0] F1, // First floating-point operand
    input [15:0] F2, // Second floating-point operand
    output reg [15:0] F3 // Result of multiplication
);

// Constants for sign, exponent, and fraction bit widths
parameter SIGN_BIT = 1;
parameter EXP_BITS = 5;
parameter FRAC_BITS = 10;
parameter BIAS = 15;

// Special number detection
wire is_nan1 = (F1[14:10] == 5'b11111) && (F1[9:0] != 0);
wire is_nan2 = (F2[14:10] == 5'b11111) && (F2[9:0] != 0);
wire is_inf1 = (F1[14:10] == 5'b11111) && (F1[9:0] == 0);
wire is_inf2 = (F2[14:10] == 5'b11111) && (F2[9:0] == 0);
wire is_zero1 = (F1[14:0] == 0);
wire is_zero2 = (F2[14:0] == 0);

// Fields extraction
wire sign1 = F1[15];
wire sign2 = F2[15];
wire [EXP_BITS-1:0] exp1 = F1[14:10];
wire [EXP_BITS-1:0] exp2 = F2[14:10];
wire [FRAC_BITS:0] frac1 = (exp1 == 0) ? {1'b0, F1[9:0]} : {1'b1, F1[9:0]}; // Add implicit leading 1
wire [FRAC_BITS:0] frac2 = (exp2 == 0) ? {1'b0, F2[9:0]} : {1'b1, F2[9:0]}; // Add implicit leading 1

// Intermediate signals
reg [EXP_BITS:0] result_exp;
reg [FRAC_BITS*2+1:0] product_frac;
reg result_sign;

always @(*) begin
    // Handle special cases
    if (is_nan1 || is_nan2) begin
        F3 = 16'h7FFF; // NaN
    end else if (is_inf1 || is_inf2) begin
        if (is_zero1 || is_zero2) begin
            F3 = 16'h7FFF; // NaN (inf * 0)
        end else begin
            F3 = {sign1 ^ sign2, 5'b11111, 10'b0}; // Inf with correct sign
        end
    end else if (is_zero1 || is_zero2) begin
        F3 = {sign1 ^ sign2, 15'b0}; // Zero with correct sign
    end else begin
        // Calculate result sign
        result_sign = sign1 ^ sign2;

        // Calculate result exponent
        result_exp = exp1 + exp2 - BIAS;

        // Multiply fractions
        product_frac = frac1 * frac2;

        // Normalize the result
        if (product_frac[FRAC_BITS*2+1]) begin
            product_frac = product_frac >> 1;
            result_exp = result_exp + 1;
        end

        // Check for overflow or underflow
        if (result_exp >= 31) begin
            F3 = {result_sign, 5'b11111, 10'b0}; // Overflow to infinity
        end else if (result_exp <= 0) begin
            F3 = 16'b0; // Underflow to zero
        end else begin
            // Construct the result
            F3 = {result_sign, result_exp[4:0], product_frac[FRAC_BITS+9:FRAC_BITS]};
        end
    end
end
endmodule
```

```

// Testbench for the FP16_Multiplier with Special Cases
module FP16_Multiplier_tb;
    reg [15:0] F1, F2;
    wire [15:0] F3;

    FP16_Multiplier uut (
        .F1(F1),
        .F2(F2),
        .F3(F3)
    );

    // Task to display results in floating-point decimal format
    task display_result;
        input [15:0] in1, in2, out;
        real val1, val2, val3;
        begin
            val1 = convert_to_real(in1);
            val2 = convert_to_real(in2);
            val3 = convert_to_real(out);
            $display("F1: %h (%f), F2: %h (%f), F3: %h (%f)", in1, val1, in2, val2, out, val3);
        end
    endtask

    // Task to convert 16-bit floating-point to real
    function real convert_to_real;
        input [15:0] fp;
        reg sign;
        reg [4:0] exponent;
        reg [10:0] fraction;
        real value;
        begin
            sign = fp[15];
            exponent = fp[14:10];
            fraction = fp[9:0];
            if (exponent == 0) begin
                value = 0.0;
            end else if (exponent == 5'b11111) begin
                value = (fraction == 0) ? ((sign) ? -1.0/0.0 : 1.0/0.0) : 0.0/0.0; // Inf or NaN
            end else begin
                value = (1.0 + (fraction / 1024.0)) * (2.0 ** (exponent - 15));
                value = (sign) ? -value : value;
            end
            convert_to_real = value;
        end
    endfunction
endmodule

initial begin
    // Test special cases
    F1 = 16'h7C00; // +Inf
    F2 = 16'h7C00; // +Inf
    #10 display_result(F1, F2, F3);

    F1 = 16'h7C00; // +Inf
    F2 = 16'hFC00; // -Inf
    #10 display_result(F1, F2, F3);

    F1 = 16'h7FFF; // NaN
    F2 = 16'h7C00; // +Inf
    #10 display_result(F1, F2, F3);

    F1 = 16'h0000; // +0
    F2 = 16'h8000; // -0
    #10 display_result(F1, F2, F3);

    F1 = 16'h3D00; // 1.625
    F2 = 16'h0000; // +0
    #10 display_result(F1, F2, F3);

    F1 = 16'h8000; // -0
    F2 = 16'h4680; // 4.75
    #10 display_result(F1, F2, F3);

    // Additional finite-number combinations
    F1 = 16'h3C00; // 1.5
    F2 = 16'h4200; // 2.5
    #10 display_result(F1, F2, F3);

    F1 = 16'h4000; // 2.0
    F2 = 16'h3C00; // 1.5
    #10 display_result(F1, F2, F3);

    F1 = 16'h4500; // 5.0
    F2 = 16'hC400; // -4.0
    #10 display_result(F1, F2, F3);

    F1 = 16'h4600; // 7.0
    F2 = 16'hC500; // -5.0
    #10 display_result(F1, F2, F3);

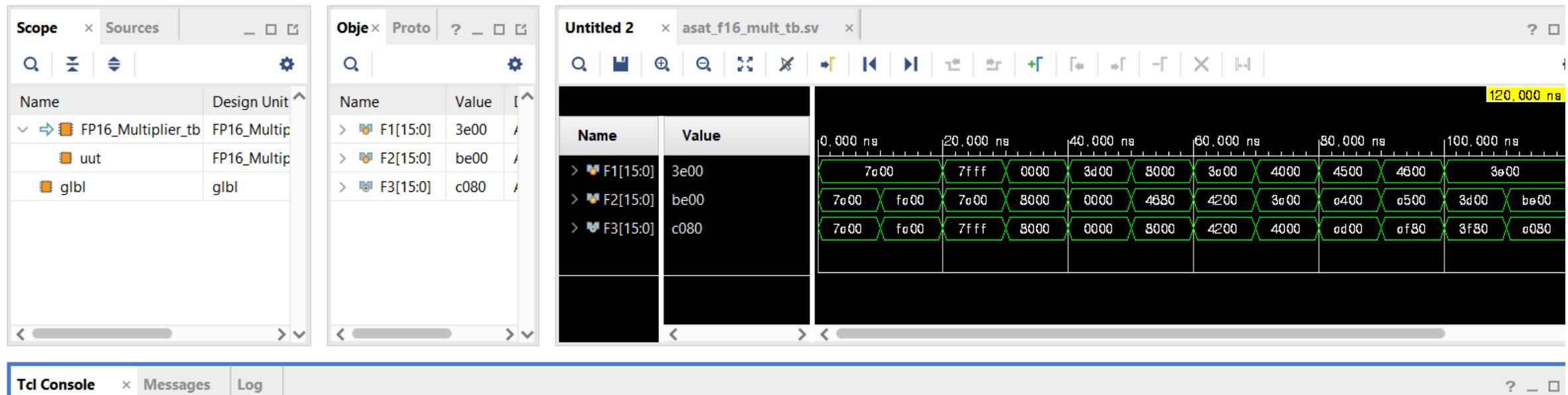
    F1 = 16'h3E00; // 1.75
    F2 = 16'h3D00; // 1.625
    #10 display_result(F1, F2, F3);

    F1 = 16'h3E00; // 1.75
    F2 = 16'hBE00; // -1.75
    #10 display_result(F1, F2, F3);

    // Finish simulation
    $stop;
end

```

FP16-Multiplier



```
# run 1000ns
F1: 7c00 (inf), F2: 7c00 (inf), F3: 7c00 (inf)
F1: 7c00 (inf), F2: fc00 (-inf), F3: fc00 (-inf)
F1: 7fff (nan), F2: 7c00 (inf), F3: 7fff (nan)
F1: 0000 (0.000000), F2: 8000 (0.000000), F3: 8000 (0.000000)
F1: 3d00 (1.250000), F2: 0000 (0.000000), F3: 0000 (0.000000)
F1: 8000 (0.000000), F2: 4680 (6.500000), F3: 8000 (0.000000)
F1: 3c00 (1.000000), F2: 4200 (3.000000), F3: 4200 (3.000000)
F1: 4000 (2.000000), F2: 3c00 (1.000000), F3: 4000 (2.000000)
F1: 4500 (5.000000), F2: c400 (-4.000000), F3: cd00 (-20.000000)
F1: 4600 (6.000000), F2: c500 (-5.000000), F3: cf80 (-30.000000)
F1: 3e00 (1.500000), F2: 3d00 (1.250000), F3: 3f80 (1.875000)
F1: 3e00 (1.500000), F2: be00 (-1.500000), F3: c080 (-2.250000)
$stop called at time : 120 ns : File "C:/2024_Asat_Xilinx/asat_f16_mult/asat_f16_mult.srcs/sim_1/new/asat_f16_mult_tb.sv" Line 103
INFO: [XISE-XSIM-961] XSim completed. Design snapshot 'FP16 Multiplier tb behav' loaded
```

4. Pipelined Adder

There are two typical sequential logics: pipelines and FSMs. Why don't we start with pipeline design first? Let's assume we are going to calculate $A+B+C$ using two signed adders. Since we have a lot of input data coming from the buffer, we want to design it using a pipeline in which each stage has one adder. The input data will be latched to the input registers A and B at the first clock. At the second clock, they will be added together and latched to a pipeline register. Furthermore, they will be added again with the C input and latched to the final output register S during the third clock cycle. I hope you will understand the operation without any confusion. Please give me the SystemVerilog code for this operation with the testbench. The design should be synthesizable, but any non-synthesizable statements may be used in the testbench.

We assume $A[i]$, $B[i]$, and $C[i]$ are coming together, and we want the intermediate pipeline register to be observable for debugging.

Your code doesn't seem to give the correct result. I know why: input C is not synchronized with inputs A and B. You should use another pipeline register to latch C before adding to the result of $A+B$.

Make the number of input data sets 10.

Great, look fine. Now, we move on to the FSM design.

```
// Pipeline design for A + B + C using two pipeline stages
module PipelineAdder (
    input logic clk,
    input logic rst,
    input logic signed [31:0] A,
    input logic signed [31:0] B,
    input logic signed [31:0] C,
    output logic signed [31:0] S,
    output logic signed [31:0] debug_sum_stage1,
    output logic signed [31:0] debug_C_stage1
);
    // Pipeline registers
    logic signed [31:0] sum_stage1;
    logic signed [31:0] C_stage1;

    always_ff @(posedge clk or posedge rst) begin
        if (rst) begin
            sum_stage1 <= 32'sd0;
            C_stage1 <= 32'sd0;
            S <= 32'sd0;
        end else begin
            // Stage 1: A + B, latch C
            sum_stage1 <= A + B;
            C_stage1 <= C;
            // Stage 2: sum_stage1 + C_stage1
            S <= sum_stage1 + C_stage1;
        end
    end

    // Debug output for intermediate pipeline registers
    assign debug_sum_stage1 = sum_stage1;
    assign debug_C_stage1 = C_stage1;
endmodule
```

```

// Testbench for PipelineAdder
module Testbench;
    logic clk;
    logic rst;
    logic signed [31:0] A, B, C;
    logic signed [31:0] S;
    logic signed [31:0] debug_sum_stage1;
    logic signed [31:0] debug_C_stage1;

    // Instantiate the PipelineAdder
    PipelineAdder uut (
        .clk(clk),
        .rst(rst),
        .A(A),
        .B(B),
        .C(C),
        .S(S),
        .debug_sum_stage1(debug_sum_stage1),
        .debug_C_stage1(debug_C_stage1)
    );

    // Clock generation
    initial clk = 0;
    always #5 clk = ~clk;

    // Test procedure
    initial begin
        // Initialize inputs
        rst = 1;
        A = 0;
        B = 0;
        C = 0;

        // Apply reset
        #10 rst = 0;

        // Apply test inputs
        repeat (10) begin
            #10 A = $urandom_range(-100, 100);
            B = $urandom_range(-100, 100);
            C = $urandom_range(-100, 100);
        end

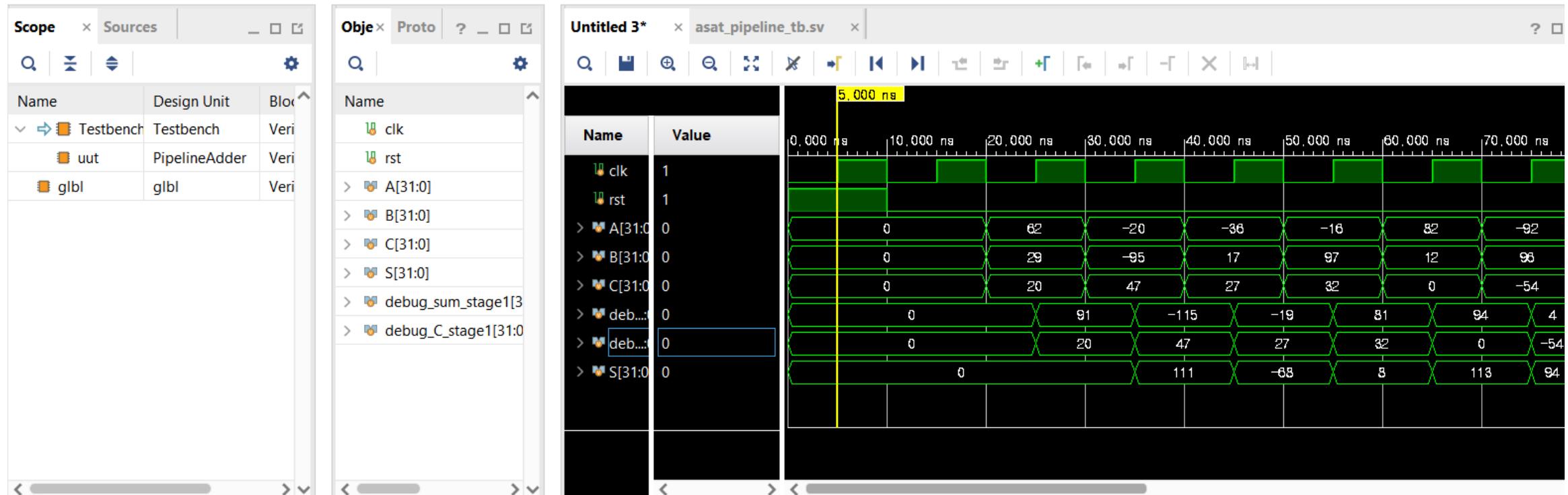
        // Wait for output to stabilize
        #50;

        // Finish simulation
        $stop;
    end

    // Monitor signals
    initial begin
        $monitor("Time: %0t | A: %0d, B: %0d, C: %0d | debug_sum_stage1: %0d | debug_C_stag
    end
endmodule

```

4. Pipelined Adder



Tcl Console window output:

```
Time: 55000 | A: -16, B: 97, C: 32 | debug_sum_stage1: 81 | debug_C_stage1: 32 | S: 8
Time: 60000 | A: 82, B: 12, C: 0 | debug_sum_stage1: 81 | debug_C_stage1: 32 | S: 8
Time: 65000 | A: 82, B: 12, C: 0 | debug_sum_stage1: 94 | debug_C_stage1: 0 | S: 113
Time: 70000 | A: -92, B: 96, C: -54 | debug_sum_stage1: 94 | debug_C_stage1: 0 | S: 113
Time: 75000 | A: -92, B: 96, C: -54 | debug_sum_stage1: 4 | debug_C_stage1: -54 | S: 94
Time: 80000 | A: 40, B: -23, C: 58 | debug_sum_stage1: 4 | debug_C_stage1: -54 | S: 94
Time: 85000 | A: 40, B: -23, C: 58 | debug_sum_stage1: 17 | debug_C_stage1: 58 | S: -50
Time: 90000 | A: -61, B: -15, C: -96 | debug_sum_stage1: 17 | debug_C_stage1: 58 | S: -50
```

5. Finite State Machine (FSM)



Great! Let's move on to designing an FSM (Finite State Machine). FSMs are fundamental in sequential circuit design and are commonly used for control logic.

yes, please.

FSM Task Example

Let's design a simple FSM that has three states:

1. **IDLE**: Waits for a `start` signal.
2. **PROCESS**: Counts from 0 to a specified value (e.g., `N`) using an input clock signal.
3. **DONE**: Indicates the completion of the count.

The FSM will take the following inputs:

- `c1k` : Clock signal.
- `rst` : Reset signal.
- `start` : Signal to begin counting.
- `N` : Target count value.

The FSM will produce the following outputs:

- `done` : High when the count reaches `N`.
- `count` : The current count value.

Would you like to use this task as a starting point, or do you have a specific FSM design in mind?



When we design FSMs, we usually specify the state changes in the sequential 'always' block and two combinational functions (next state logic and output logic) in two separate combinational 'always' blocks. How does your code differ from the usual cases?

It seems that your design counts one more cycle to generate the output. I want you to refactor the design so that we can easily debug it.

I think the output should become 1 after the counter reaches the N.

The counter should be returned to zero after it reaches the target, and I want to see the state changes in the wave file.

I want the counter to be reset right after it reaches the target. Also, show me a few more cycles in the simulation.

You did a great job. You are getting more thoughtful and more intelligent. Thank you.

```

// FSM Design: Counter FSM
module FSM_Counter (
    input logic clk,
    input logic rst,
    input logic start,
    input logic [31:0] N,
    output logic done,
    output logic [31:0] count,
    output logic [1:0] state_out
);
    // Define state encoding
    typedef enum logic [1:0] {
        IDLE = 2'b00,
        PROCESS = 2'b01,
        DONE = 2'b10
    } state_t;
    state_t current_state, next_state;

    // Registers for count and done signal
    logic [31:0] count_reg;

    // Sequential logic: State transition
    always_ff @(posedge clk or posedge rst) begin
        if (rst) begin
            current_state <= IDLE;
            count_reg <= 0;
        end else begin
            current_state <= next_state;

            if (current_state == PROCESS) begin
                if (count_reg < N) begin
                    count_reg <= count_reg + 1;
                end else begin
                    count_reg <= 0; // Reset counter after reaching target
                end
            end else begin
                count_reg <= 0;
            end
        end
    end
end

// Combinational logic: Next state logic
always_comb begin
    case (current_state)
        IDLE: begin
            if (start) begin
                next_state = PROCESS;
            end else begin
                next_state = IDLE;
            end
        end
        PROCESS: begin
            if (count_reg == N) begin
                next_state = DONE;
            end else begin
                next_state = PROCESS;
            end
        end
        DONE: begin
            if (!start) begin
                next_state = IDLE;
            end else begin
                next_state = DONE;
            end
        end
        default: next_state = IDLE;
    endcase
end

// Combinational logic: Output logic
always_comb begin
    done = (current_state == DONE);
    count = count_reg;
    state_out = current_state; // Expose current state for debugging
end

endmodule

```

```

// Testbench for FSM_Counter
module Testbench_FSM_Counter;
    logic clk;
    logic rst;
    logic start;
    logic [31:0] N;
    logic done;
    logic [31:0] count;
    logic [1:0] state_out;

    // Instantiate FSM_Counter
    FSM_Counter uut (
        .clk(clk),
        .rst(rst),
        .start(start),
        .N(N),
        .done(done),
        .count(count),
        .state_out(state_out)
    );

    // Clock generation
    initial clk = 0;
    always #5 clk = ~clk;

    // Test procedure
    initial begin
        rst = 1;
        start = 0;
        N = 0;

        // Apply reset
        #10 rst = 0;

        // Test case 1: Count to 5
        #10 N = 5; start = 1;
        #10 start = 0; // Deactivate start

        // Wait for done signal
        wait(done);
        #10;

        // Test case 2: Count to 10
        #10 N = 10; start = 1;
        #10 start = 0; // Deactivate start

        // Wait for done signal
        wait(done);
        #10;

        // Additional cycles for observation
        #50;

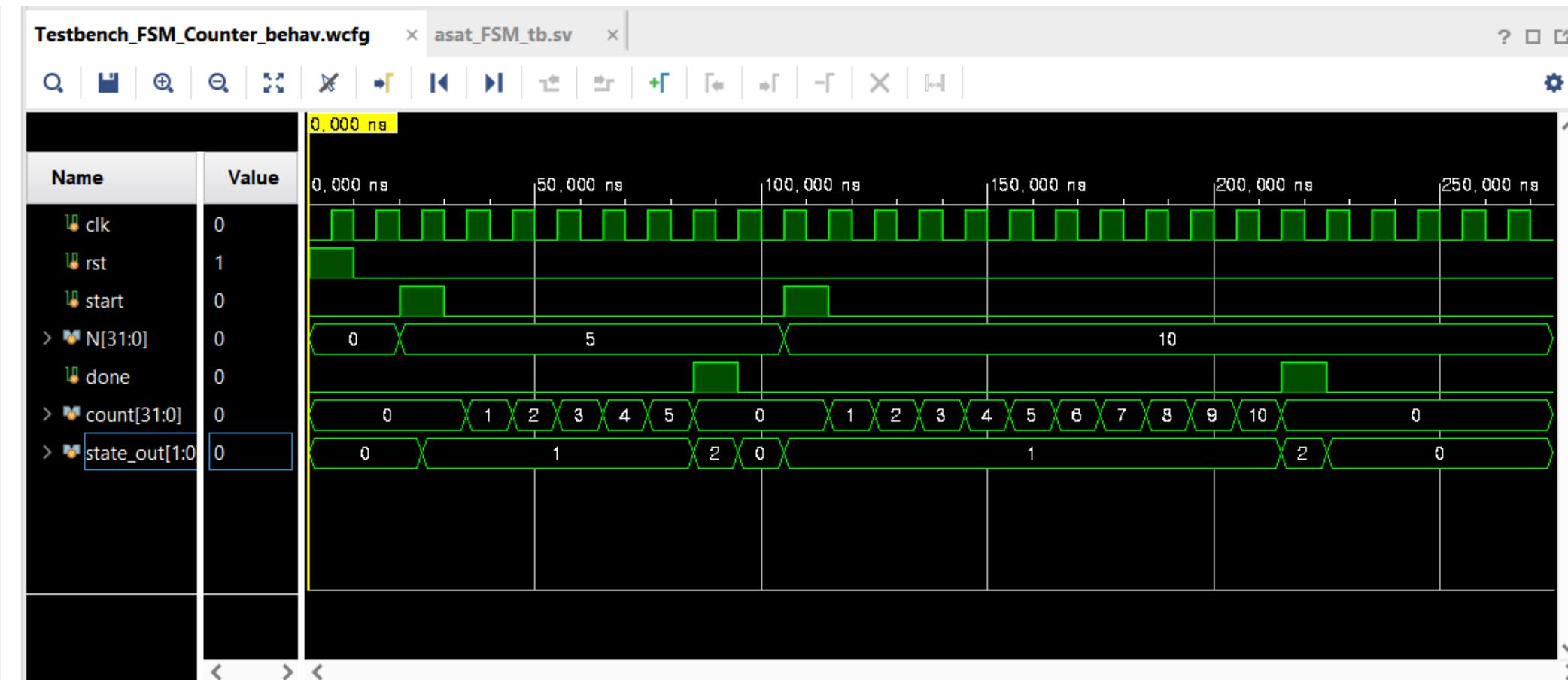
        // Finish simulation
        $stop;
    end

    // Monitor signals
    initial begin
        $monitor("Time: %0t | State: %0b | N: %0d | Count: %0d | Done: %0b", $time, state_out, N, count, done);
    end

endmodule

```

5. Finite State Machine (FSM)



Hardware Design Project Summary

- **General Observations:**

- Unlike software coding tasks, where GPT provides near-complete solutions for Python or C programming, hardware design requires additional attention, particularly **concurrency** and **timing** logic.
- The complexities of hardware design stem from its inherent parallelism and the need for precise control of timing and resources, which are less deterministic than sequential software logic.

- **Challenges Identified:**

- Concurrency for Parallel Processing: Designing logic for parallel data handling (e.g., matrix multiplication with shared processing units) requires careful coordination, which has led to incomplete or non-optimal designs.
- Timing Control: Generating control logic for sequential and time-shared operations (e.g., pipeline control or time-multiplexed buses) often results in unreliable implementations.
- Testbenches and Simulation: Testbench generation is generally functional but may lack the depth needed for corner-case testing in real-world applications. Designer guides are required.

Hardware Design Project Summary

- **Specific Examples Explored:**

- AXI-Lite GPIO Design: Initial attempts were functional but required refinement for proper integration and testing.
- Matrix Multiplication Module: Implementation with structural design (instantiating multiple multipliers) showed promise but highlighted the need for improved timing and control logic.
- Floating-Point Units, Pipelined Logic, and FSM: Simpler designs for combinational and sequential logic worked reasonably well, serving as a foundation for more complex designs.

- **Approach Taken:**

- Designs were **iteratively improved based on user feedback**, focusing on generating SystemVerilog code with clear and testable logic.
- Emphasis on including testbenches and driver software to verify functionality.
- Integration with tools like Vivado for synthesis and simulation to validate designs in practical scenarios.

Hardware Design Project Summary

- **Lessons Learned:**
 - Hardware design requires explicit details about concurrency, timing, and resource utilization, which GPT-generated code might not fully capture without iterative refinement.
 - A structured, step-by-step approach (starting from simple examples and progressing to complex designs) is crucial for success.
 - User expertise in reviewing, testing, and modifying designs is critical for transforming GPT outputs into production-ready hardware.
- **Future Directions:**
 - Focus on modular designs to enhance reusability and maintainability.
 - Explore advanced concurrency and timing logic through iterative refinement and simulation.
 - Develop a library of verified hardware modules to streamline future projects.
 - Just as GPT has access to a Python interpreter for mathematical computations, data processing, and script execution, it could be equipped with an HDL synthesizer and simulator in the future to provide enhanced support for hardware design.