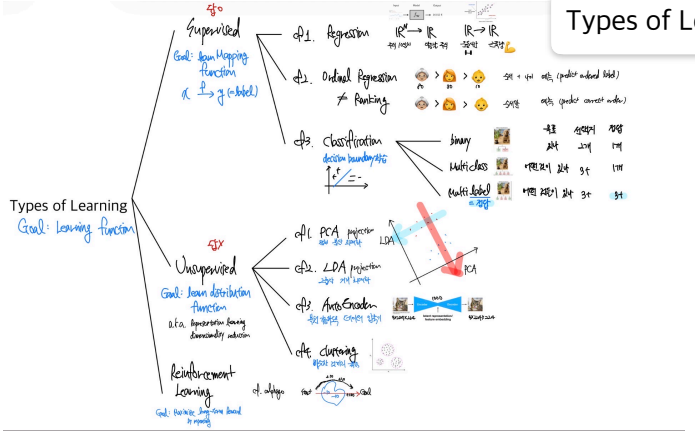
Lec1. Preliminary

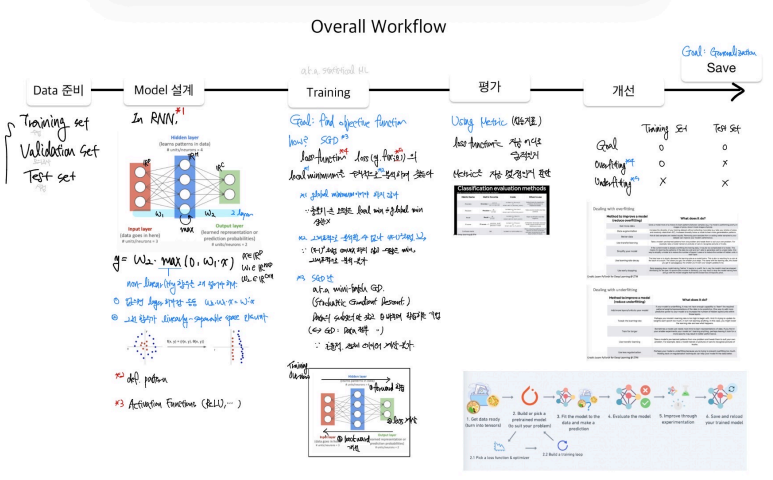
Types of Learning

Linear Algebra and Numpy

Google Colab Implementation

Overall Workflow





Types of Learning

Goal: Learning function

있O

Supervised

Goal: learn Mapping function

$x \xrightarrow{f} y$ (=label)

- cf1. Regression

$\mathbb{R}^N \rightarrow \mathbb{R}$ 주식 10일치 → 다음날 주식

$\mathbb{R} \rightarrow \mathbb{R}$ 공부시간 → 근육량💪

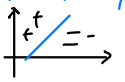
- cf2. Ordinal Regression ≠ Ranking

순서 + 차이 예측 (predict ordered label)

순서만 예측 (predict correct order)

- cf3. Classification

decision boundary찾음



	특징	선택지	정답
binary	하나	2개	1개
Multi class	여러개이 있다	3개	1개
Multi label = 정답	여러 정답이 있다	3개	3개

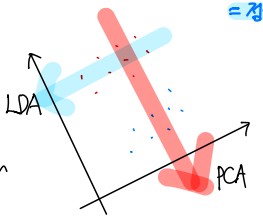
없X

Unsupervised

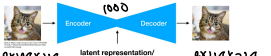
Goal: learn distribution function

a.k.a. Representation learning
dimensionality reduction

- cf1. PCA projection 정보 손실 최대화



- cf2. LDA projection 구분선 거리 최대화
- cf3. AutoEncoder 복원 가능한 데이터 압축기

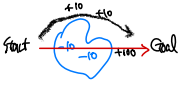


- cf4. clustering 비슷한 것끼리 묶음

Reinforcement Learning

Goal: Maximize long-term reward by reporting

cf. alphago



Overall Workflow

Goal: Generalization

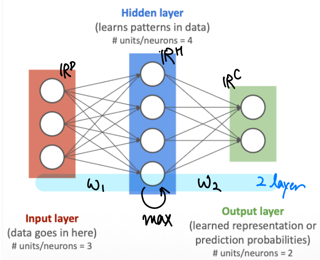
Data 준비 → Model 설계 → Training (a.k.a. statistical ML) → 평가 → 개선 → Save

Data 준비

- Training set (수업)
- Validation set (모의시사)
- Test set (시험)

Model 설계

In RNN,*1

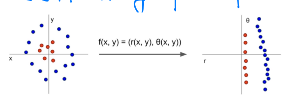


$y = W_2 \cdot \max(0, W_1 \cdot x)$ $\quad x \in \mathbb{R}^P$, $W_1 \in \mathbb{R}^{H \times P}$, $W_2 \in \mathbb{R}^{C \times H}$

non-linear (키) 함수는 꼭 넣어야 한다.

① 없으면 layer 하나랑 동등 $W_2 \cdot W_1 \cdot x = W' x$

② 그런 함수가 linearly-separable space 만든다



*2 def. pattern

*3 Activation Functions (ReLU, ⋯)

Training

Goal: find objective function

how? SGD*3

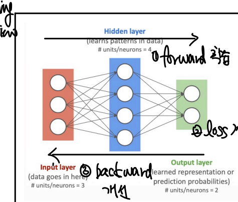
loss function*4 loss(y, f(x; θ))*5 의 local minimum*1을 수치적으로*2 분석하여 찾는다

*1 global minimum이어야 하지 않나?
∵ 충분히 큰 모델은 local min ≒ global min 상관X

*2 그래프적으로 분석할 수 없나 (x−1)²처럼 ↳
∵ (x−1)²처럼 convex 하지 않고 수많은 min, 그래프적으로 분석 불가

*3 SGD 란
a.k.a mini-batch GD.
(Stochastic Gradient Descent.)
Data의 subset만 갖고 θ 바꾸며 학습하는 기법
(⇔ GD: Data 전부 〃)
∵ 효율적. 전체 데이터 계산 불가

Training Overview



평가

Using Metric (성능지표)

loss function은 자동 이다로 손실인지

Metric은 자동 몇점인지 판단

Classification evaluation methods

Key: tp = True Positive, tn = True Negative, fp = False Positive, fn = False Negative

Metric Name	Metric Forumla	Code	When to use
Accuracy	Accuracy $= \frac{tp + tn}{tp + tn + fp + fn}$	torchmetrics.Accuracy() or sklearn.metrics.accuracy_score()	Default metric for classification problems. Not the best for imbalanced classes.
Precision	Precision $= \frac{tp}{tp + fp}$	torchmetrics.Precision() or sklearn.metrics.precision_score()	Higher precision leads to less false positives.
Recall	Recall $= \frac{tp}{tp + fn}$	torchmetrics.Recall() or sklearn.metrics.recall_score()	Higher recall leads to less false negatives.
F1-score	F1-score $= 2 \cdot \frac{precision \cdot recall}{precision + recall}$	torchmetrics.F1Score() or sklearn.metrics.f1_score()	Combination of precision and recall, usually a good overall metric for a classification model.
Confusion matrix	NA	torchmetrics.ConfusionMatrix()	When comparing predictions to truth labels to see where model gets confused. Can be hard to use with large numbers of classes.

...for Deep Learning @ ZTM

개선

	Training Set	Test Set
Goal	O	O
Overfitting*4	O	X
Underfitting*5	X	X

Dealing with overfitting

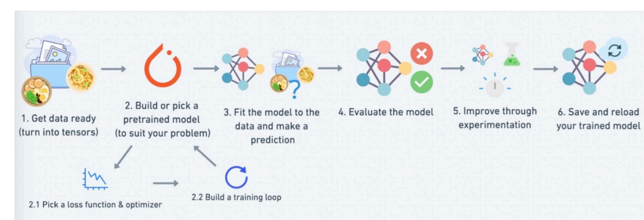
Method to improve a model (reduce overfitting)	What does it do?
Get more data	Gives a model more of a chance to learn patterns between samples (e.g. if a model is performing poorly on images of pizza, show it more images of pizza).
Data augmentation	Increase the diversity of your training dataset without collecting more data (e.g. take your photos of pizza and randomly rotate them 30°). Increased diversity forces a model to learn more generalisation patterns.
Better data	Not all data samples are created equally. Removing poor samples from or adding better samples to your dataset can improve your model's performance.
Use transfer learning	Take a model's pre-learned patterns from one problem and tweak them to suit your own problem. For example, take a model trained on pictures of cars to recognise pictures of trucks.
Simplify your model	If the current model is already overfitting the training data, it may be too complicated of a model. This means it's learning the patterns of the data too well and isn't able to generalize well to unseen data. One way to simplify a model is to reduce the number of layers it uses or to reduce the number of hidden units in each layer.
Use learning rate decay	The idea here is to slowly decrease the learning rate as a model trains. This is akin to reaching for a coin at the back of a couch. The closer you get, the smaller your steps. The same with the learning rate, the closer you get to convergence, the smaller you'll want your weight updates to be.
Use early stopping	Early stopping stops model training *before* it begins to overfit. As in, say the model's loss has stopped decreasing for the past 10 epochs (this number is arbitrary), you may want to stop the model training here and go with the model weights that had the lowest loss (10 epochs prior).

Credit: Learn PyTorch for Deep Learning @ ZTM

Dealing with underfitting

Method to improve a model (reduce underfitting)	What does it do?
Add more layers/units to your model	If your model is underfitting, it may not have enough capability to *learn* the required patterns/weights/representations of the data to be predictive. One way to add more predictive power to your model is to increase the number of hidden layers/units within those layers.
Tweak the learning rate	Perhaps your model's learning rate is too high to begin with. And it's trying to update its weights each epoch too much, in turn not learning anything. In this case, you might lower the learning rate and see what happens.
Train for longer	Sometimes a model just needs more time to learn representations of data. If you find in your smaller experiments your model isn't learning anything, perhaps leaving it train for a more epochs may result in better performance.
Use transfer learning	Take a model's pre-learned patterns from one problem and tweak them to suit your own problem. For example, take a model trained on pictures of cars to recognise pictures of trucks.
Use less regularization	Perhaps your model is underfitting because you're trying to prevent overfitting too much. Holding back on regularization techniques can help your model fit the data better.

Credit: Learn PyTorch for Deep Learning @ ZTM



부록

*1

p3 overall workflow > model

RNN이 항상 정답은 아님.
상황에 따라 더 유리한 모델 있음.
cf. RandomForest, NaiveBayes, SVM

*2

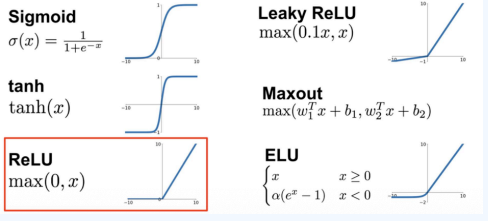
p3 overall workflow > model

아래 용어는 모두 비슷한 의미다.
pattern, embedding, weights,
feature represenation,
feature vectors

*3

p3 overall workflow > model

다음과 같은 activation function이 있다



*4

p3 overall workflow > Training

아래는 다양한 loss function이다.
MSE, MAE, CrossEntropy,
Negative log likelihood,
Connectionist temporal classification

*5

p3 overall workflow > Training

θ는 parameter이다
parameter는 함수에서 우리가 정할 수있는 값
optimization은 parameter를 바꾸는 작업

cf. $f(x;a,b,c) = ax^2+bx+c$

$\Rightarrow f(x;\theta) \quad \theta = [a,b,c]$