CHAPTER 4: FUNDAMENTALS OF MACHINE LEARNING

This chapter covers

* Forms of machine learning beyond classification and regression
* Formal evaluation procedures for machine- learning models
* Preparing data for deep learning
* Feature engineering
* Tackling overfitting
* The universal workflow for approaching machine-learning problems

1. Annotations: known targets (from input data to output)

2. **Supervised learning:** mapping input to output (most of problem is this type)

Ex: optical character recognition, speech recognition, image classification, and language translation

* **Sequence generation**—Given a picture, predict a caption describing it. Sequence generation can sometimes be reformulated as a series of classification problems (such as repeatedly predicting a word or token in a sequence).
* **Syntax tree prediction**—Given a sentence, predict its decomposition into a syntax tree.
* **Object detection**—Given a picture, draw a bounding box around certain objects inside the picture. This can also be expressed as a classification problem (given many candidate bounding boxes, classify the contents of each one) or as a joint classification and regression problem, where the bounding-box coordinates are predicted via vector regression.
* **Image segmentation**—Given a picture, draw a pixel-level mask on a specific object.

3. **Unsupervised learning:** finding transformations of the input data without the help of any targets (for data visualization, data compression, data denoising, or correlation of data…)

**Dimensionality reduction** and **clustering** are categories of UL

4. **Self-supervised learning:** instance of SL. Leaning without human-annotated labels. The labels are generated from the input data, using heuristic algorithm

Ex: autoencoders**,** trying to pre-dict the next frame in a video, given past frames, or the next word in a text, given previ-ous words, are instances of self-supervised learning

5. **Reinforcement learning:** an agent receives information about its environment and learns to choose actions that will maximize some reward.

Future: self-driving cars, robotics, resource management, education, and so on. It’s an idea whose time has come, or will come soon.

6. **Vector regression**—A task where the target is a set of continuous values: for

example, a continuous vector. If you’re doing regression against multiple val-

ues (such as the coordinates of a bounding box in an image), then you’re

doing vector regression.

7. **Mini-batch or batch**—A small set of samples (typically between 8 and 128)

that are processed simultaneously by the model. The number of samples is

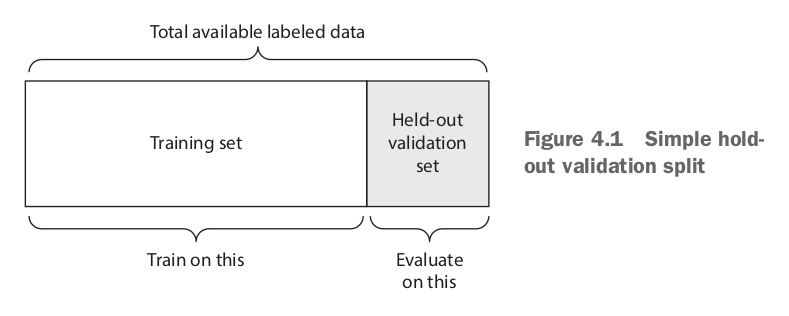
often a power of 2, to facilitate memory allocation on GPU. When training, a

mini-batch is used to compute a single gradient-descent update applied to

the weights of the model.

8. **Hyper-parameter:** number of layers or the size of the layers (a search for good configuration of the model based on its performance on the validation set)

**9. Information leaks:** every time tuning the hyperparamater based on validation set, some information about the validation data leaks into model.

**10. Simple hold-out validation:**

Issue with small data set, different measures of model each time.

**11. K-fold validation:** calculate the average of K time split.

12. **Iterated K-fold validation with shuffling**: relatively little data available

Applying K-fold multiple times, shuffling the data before splitting K ways.

=> More time and resource to train, but more effective

13. **Things to look for when choosing an evaluation protocol:**

- Data representativeness: shuffle data first

- The arrow of time if predict future time, should not randomly shuffle data before splitting, because will create **temporal leak**

- Redundancy in your data data appear twice, make sure they are all disjoint

14. **Data preprocessing for neural networks:** Data preprocessing aims at making the raw data at hand more amenable to neural networks. This includes vectorization, normalization, handling missing values, and feature extraction.

15. **Vectorization:** all inputs are tensor (sounds, images, text…).

16. **Value normalization:** change floating point value to float32 range 0-1…

Normalize each feature independently, so mean = 0, STD = 1

**Take small value:** 0-1

**Be homogenous:** all feature is roughly the same range

*x -= x.mean(axis=0)*

*x /= x.std(axis=0)*

17. **MISSING VALUES:** create some training data that have missing value

18. **Feature engineering:** apply your knowledge into processing data in order to reduce training time and amount of training data => faster and better

Ex: read the clock face

19. **Overfitting and underfitting:** The fundamental issue in machine learning is the tension between optimization and generalization

20. **Regularization:** module the quantity of information that your model is allowed to store or to add constraints on what information it’s allowed to store.

21. **Reducing the network size (learnable parameter or capacity):** simplest way to reduce overfitting is reducing the size of model (num of layers and unit per layers)

More parameter means more **memorization capacity** => perfect mapping on TRAINING SET**,** therefore will be bad on test set.

Should use model have just enough parameters so they don’t underfit or overfit. **Compromise** between **too much capacity** and **not enough capacity.**

22. **Reasonable workflow:** start with **few** layers and parameters, and increase the size of layers or add new layers until you see diminishing returns with regard to validation loss.

23. **Remember** the consistency of validation loss and training loss score.

24. **Weight regularizaton:** put constraint on the complexity of a network by forcing its weights to take only small values, which makes the distribution of weight values more regular.

It’s done by adding to the loss function of the network a **cost** associated with having large weights. **2 flavors:**

**L1 regularization:** the cost added is proportional to the **absolute value of the weight coefficients** (the L1 norm of the weights)

**L2 regularization (**or **weight decay):** the cost added is proportional to the **square of the value of weight coefficients (**the L2 norm of the weights)

*from keras import regularizers*

*model = models.Sequential()*

*model.add(layers.Dense(16, kernel\_regularizer=regularizers.l2(0.001),*

*activation='relu', input\_shape=(10000,)))*

*model.add(layers.Dense(16, kernel\_regularizer=regularizers.l2(0.001),*

*activation='relu'))*

*model.add(layers.Dense(1, activation='sigmoid'))*

The 0.001 means every coefficient in the weight matrix of the layer will ad 0.001\***weight\_coefficient\_value** to the total cost loss of the network. (Only at training time)

25. **Drop out: (**most commonly used regularization techniques) randomly drop out (set to zero) a number of output features of layers during training. Fraction of the features that are zeroed out (usually 0.2 to 0.5)

At test time, no dropout, but the output values will be scaled down to dropout rate

*layer\_output \*= np.random.randint(0, high=2, size=layer\_output.shape)*

*layer\_output /= 0.5*

*model.add(layers.Dropout(0.5))*

***WORKFLOW***

26. **Defining the problem and assembling the dataset**

What will your input data be? What are you trying to predict? You can only learn to predict something if you have available training data: for example, you can only learn to classify the sentiment of movie reviews if you have both movie

reviews and sentiment annotations available. As such, data availability is usually

the limiting factor at this stage (unless you have the means to pay people to col-

lect data for you).

What type of problem are you facing? Is it binary classification? Multiclass classi-fication? Scalar regression? Vector regression? Multiclass, multilabel classifica-tion? Something else, like clustering, generation, or reinforcement learning? Identifying the problem type will guide your choice of model architecture, loss function, and so on.

 You **hypothesize** that your outputs can be predicted given your inputs.

 You **hypothesize** that your available data is sufficiently informative to learn the

relationship between inputs and outputs.

**Nonstationary problems:** when problem is not stable (seasons in years will be different for clothing recommendations…)

27. **Choosing metrics:**

Classification**: accuracy** or **area under the receiver operating characteristic curve (ROC AUC)**

Ranking problem or multilable classification: **mean average precision**

Class-imbalanced problems, **precision** and **recall**

28. **Decide evaluation protocol**

**Maintaining a hold-out validation set**

**Doing k-fold cross-validation**

**Doing iterated K-fold validation**

29. **Preparing data**

**-** Data should be formatted as tensors

- The value inside the tensor should usually be scaled to small values

Ex: [-1,1] or [0,1]

- If different features values in different ranges (heterogeneous data), the data should normalized

- Feature engineering (small-data problems)

30. **Developing a model that does better than a baseline**

- Achieve **statistical power:** mean creating a model that beat the baseline (over 0.5 for binary classification…)

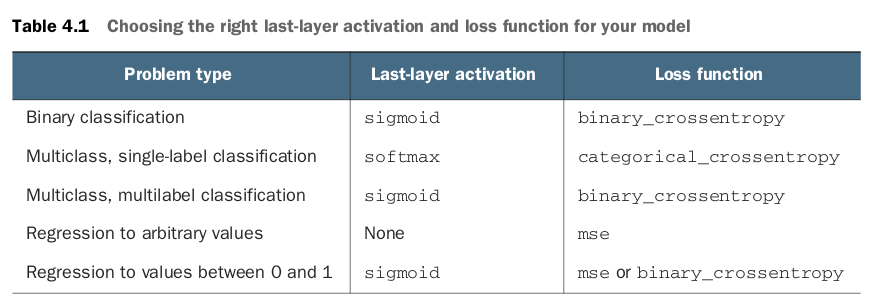
- Testing if your **hypothesis** is right or wrong after trying various architecture.

**-** If thing goes well, make these 3 choices:

- **Last-layer activation**: Establish useful constraints on the network’s output. Ex: sigmoid for binary classification, nothing in regression

- **Loss function:** Match the problem. Ex: binary\_crossentropy for binary classification, mse for regression problem.

- **Optimization configuration:** rmsprop for most cases. Later on will learn.



31. **Scaling up: developing a model that overfits**

**-** Build a model that stands right at the boarder of underfitting and overfitting, between undercapacity and overcapacity.

- To find out where is the boarder, you need to cross it.

1. Add layers

2. Make the layers bigger.

3. Train for more apochs.

- Monitor the training loss and validation loss, as well as the training and validation values for any metrics you care about. When seeing performance on validation data degrade => start overfitting.

32. **Regularizing the model and tuning hyperparameters.**

**-** Take most of the time. Repeatedly modify model,t rain, evaluate on validation data (not test). Over and over again.

**- Add dropout.**

**- Try different architectures: add or remove layers.**

**- Add L1 and/or L2 regularization.**

**- Try different hyperparameters (such as the number of units per layer or the**

**learning rate of the optimizer) to find the optimal configuration.**

**- Optionally, iterate on feature engineering: add new features, or remove fea-tures that don’t seem to be informative.**

**Careful:** each time you do the configuration, you leak information about the validation process into the model.

**Chapter summary**

* Define the problem at hand and the data on which you’ll train. Collect this data, or annotate it with labels if need be.
* Choose how you’ll measure success on your problem. Which metrics will you monitor on your validation data?
* Determine your evaluation protocol: hold-out validation? K-fold validation? Which portion of the data should you use for validation?
* Develop a first model that does better than a basic baseline: a model with statistical power.
* Develop a model that overfits.
* Regularize your model and tune its hyperparameters, based on performance on the validation data. A lot of machine-learning research tends to focus only on this step—but keep the big picture in mind.