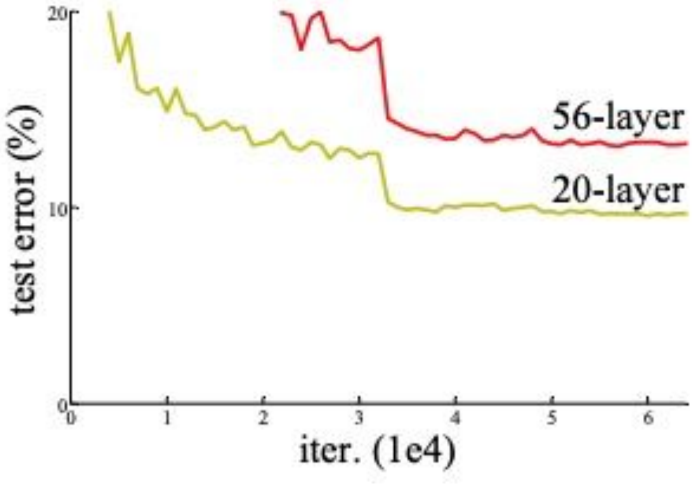
Applied Machine Learning for Business Analytics

Lecture 4: Deep Learning Practices

Lecturer: Zhao Rui

Overfitting?

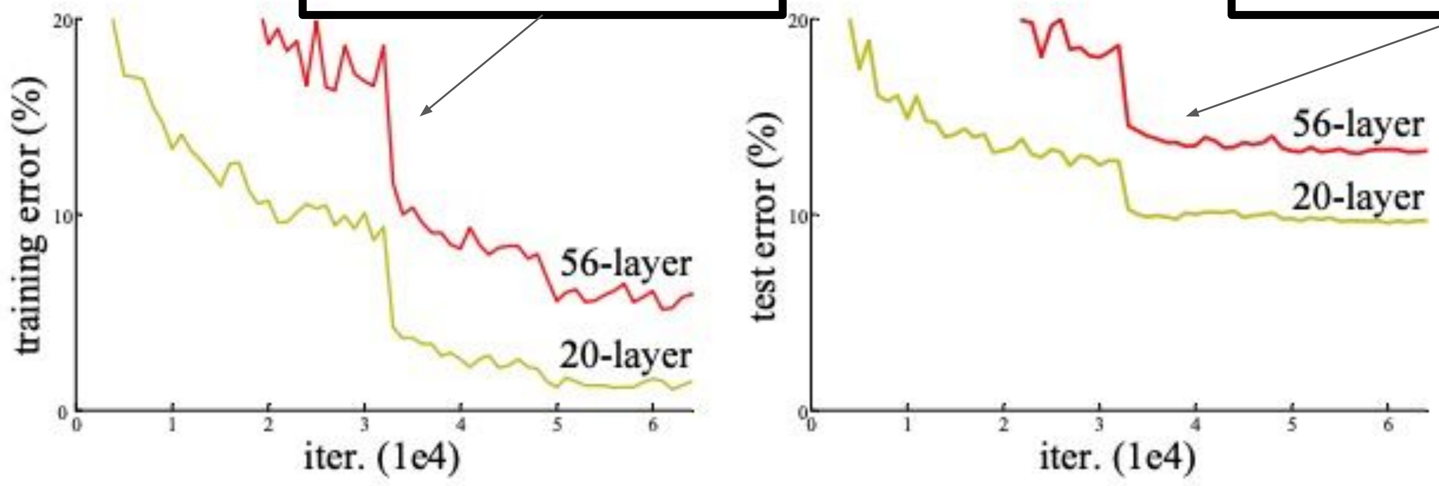


https://arxiv.org/abs/1512.03385

Training a deep model is challenging

The deeper model is not converged in training.

May not be due to overfitting



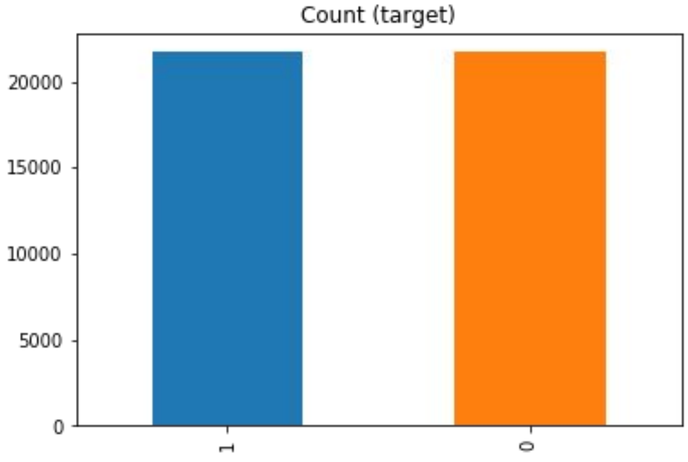
Source: https://arxiv.org/abs/1512.03385

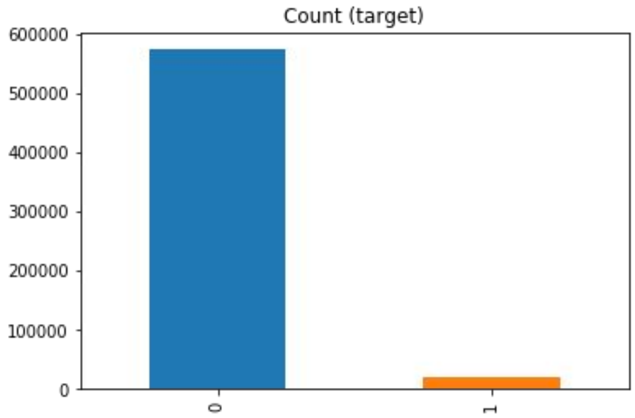
Agenda

1. Class Imbalance
2. Data Augmentation
3. Network Configuration
4. Parameters Initialization
5. Optimizers
6. Regularization Techniques
7. Do not sleep on traditional machine learning

1. Class Imbalance

Small data in some categories





Class imbalance is the norm

- Bridge Structural Fault Detection
- Fraud Detection
- Disease Diagnosis
- Spam Detection

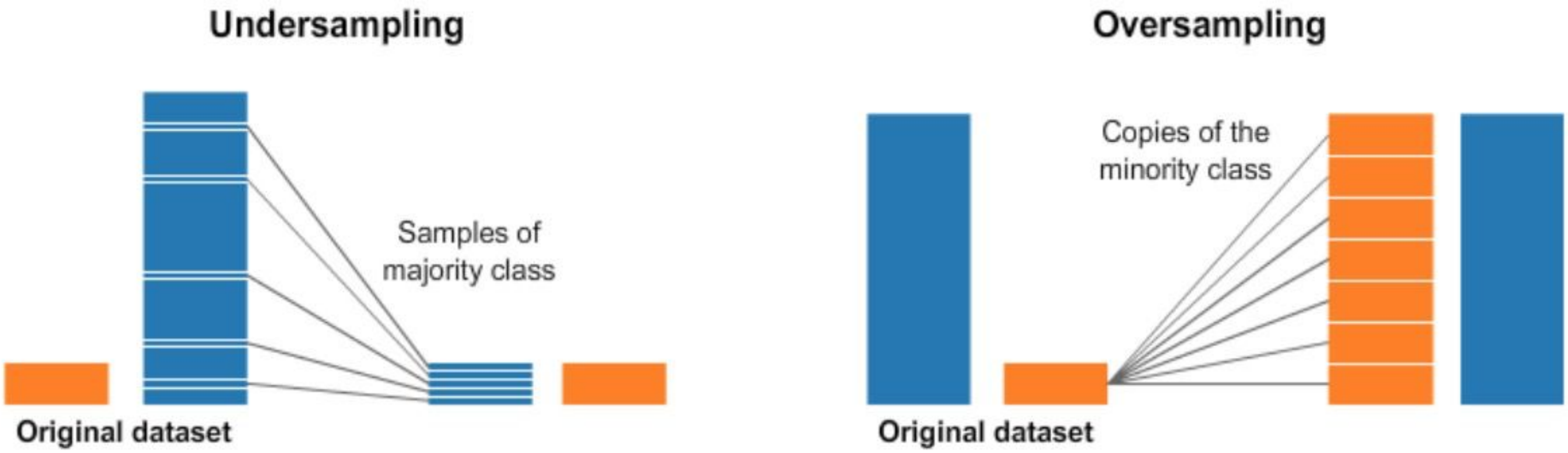
Class Imbalance is challenging

- Not enough knowledge to learn about rare classes
- Imbalanced problem: the number of fraud cases are much less than the one of normal cases.
- Rare classes are usually with high cost of wrong predictions.

How to deal with class imbalance

- Resampling
    - Add more minority samples
    - Remove majority samples
- Weights Balancing
    - Tweak the loss function
- Choose robust algorithms to class imbalance

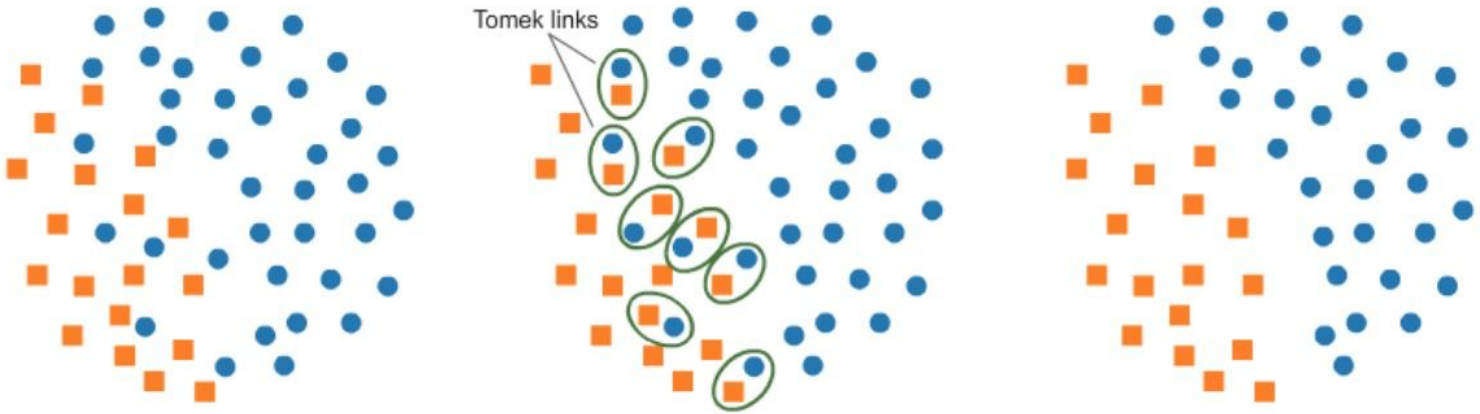
Resampling



https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets#t1

Undersampling: Tomek Links

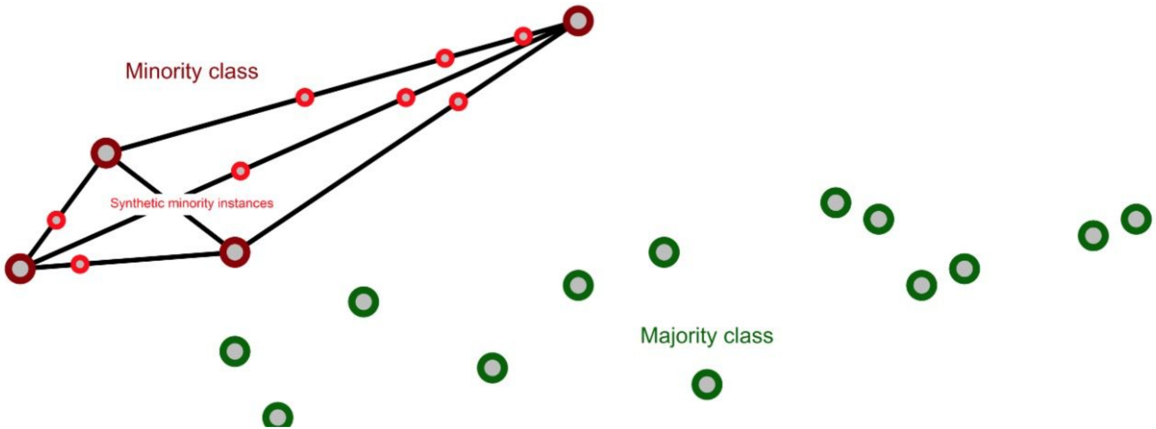
- Find pairs of close samples of opposite classes
- Remove the sample of majority class in each pair



https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets#t1

Oversampling: SMOTE

- Synthesize samples of minority class are convex(~linear) combinations of existing points and their nearest neighbors of same class.



https://rikunert.com/SMOTE_explained

Weight balancing

- Normal Loss

$$L_\theta = \sum_i L_\theta(x_i)$$

- Weighted Loss

$$L_\theta = \sum_i w_{y_i} L_\theta(x_i)$$

the number of training samples

$$w_c = \frac{N}{N_{y=c}}$$

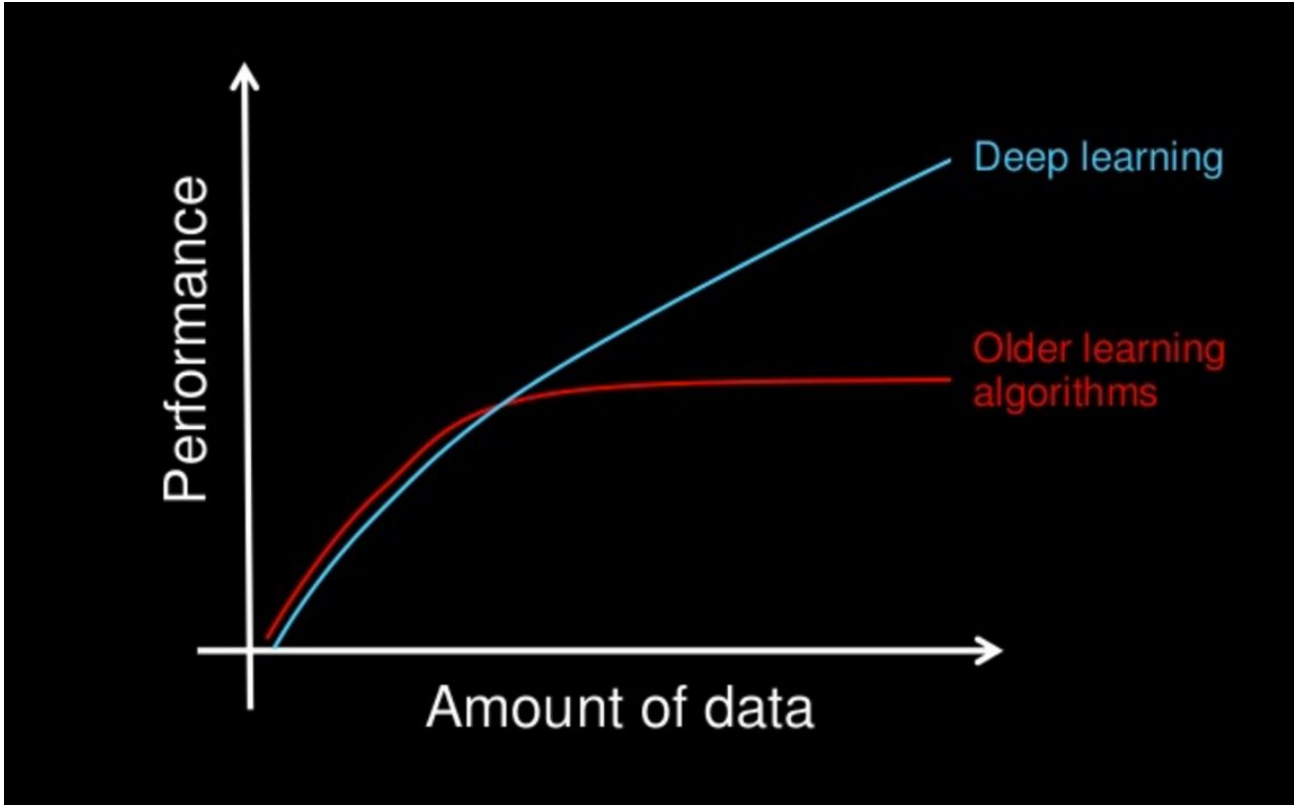
the number of training samples in the category: c

fit method

```
Model.fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose=1,
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
)
```

From Keras

2. Data Augmentation



From Andrew Ng

Data augmentation

- Deep learning models usually have billions of parameters and then require massive labeled training data
- To improve the generalization capability

Data Augmentation: create artificially labeled training datasets

Image augmentation

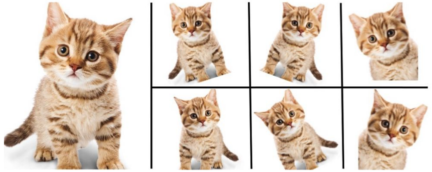
Original	Flip	Rotation	Random crop
			
• Image without any modification	• Flipped with respect to an axis for which the meaning of the image is preserved	• Rotation with a slight angle • Simulates incorrect horizon calibration	• Random focus on one part of the image • Several random crops can be done in a row

Color shift	Noise addition	Information loss	Contrast change
			
• Nuances of RGB is slightly changed • Captures noise that can occur with light exposure	• Addition of noise • More tolerance to quality variation of inputs	• Parts of image ignored • Mimics potential loss of parts of image	• Luminosity changes • Controls difference in exposition due to time of day

source : https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks

How about text data

- In computer version, data augmentation is quite common.



Enlarge your Dataset

Rotating an image a few degrees does not change its semantics

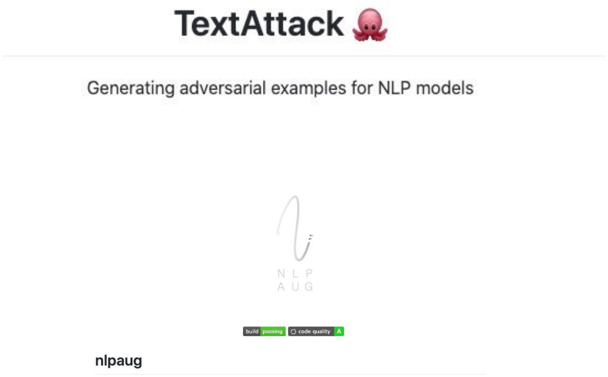
https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

- In NLP or text mining, data augmentation is challenging.

This is simple → (shuffle) Is this simple — **Semantics changed**

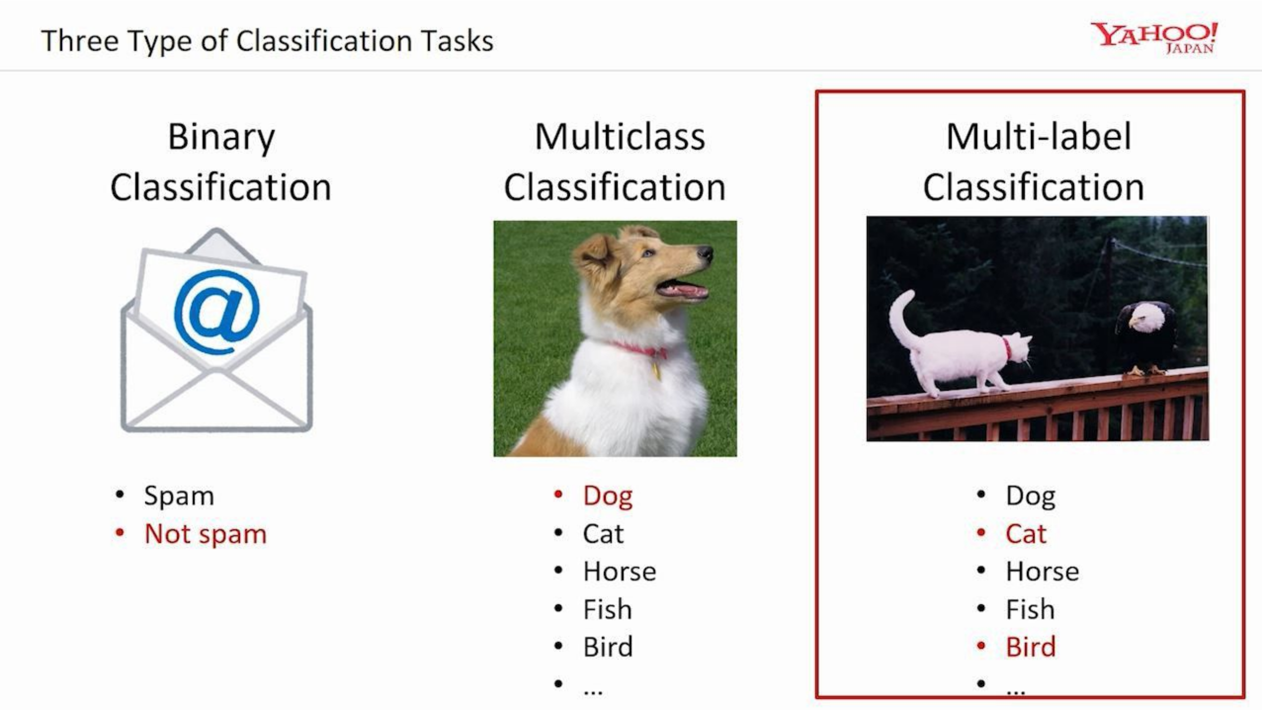
Text augmentation

- Most of methods are very task-specific
    - Lexical Replacement
    - Back Translation
    - Text Surface Transformation
    - Random Noise Injection
    - Instance Crossover Augmentation
    - Generative Methods



3. Network Configuration

Three Types of Classification Tasks



Source: https://www.microsoft.com/en-us/research/uploads/prod/2017/12/40250.jpg

Last-Layer configuration

Binary Classification

- Last-layer activation: sigmoid
- Loss function: binary_crossentropy
- Code snippets:

```
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

Multi-class Classification

- Last-layer activation: softmax
- Loss function: categorical_crossentropy
- Code snippets:

Number of unique labels in the task

```
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Last-Layer configuration

Multi-label Classification

- Last-layer activation function: sigmoid
- Loss function: binary_crossentropy
- Code snippets:

```
model.add(layers.Dense(10, activation='sigmoid'))
model.compile(loss="binary_crossentropy", optimizer='rmsprop')
```

Last-Layer configuration

Regression to arbitrary values

- Last-layer activation: Linear
- Loss function: mse
- Code snippets:

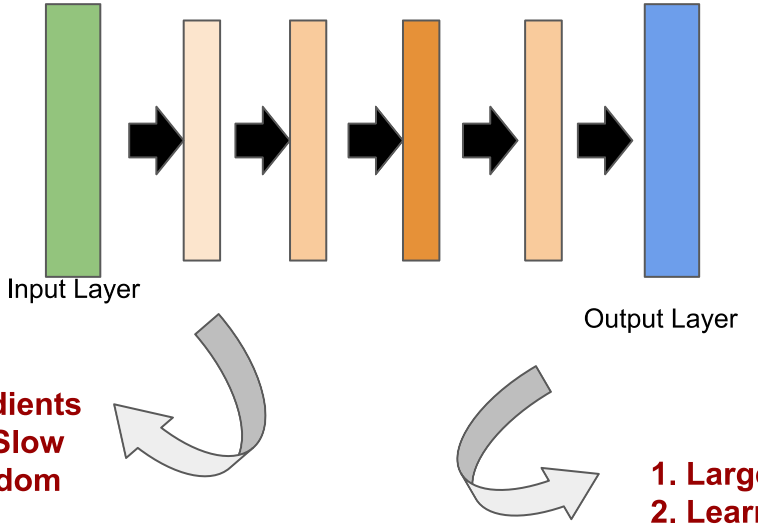
```
model.add(layers.Dense(1))
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

Regression to scaled values ranging from 0 to 1

- Last-layer activation: sigmoid
- Loss function: mse
- Code snippets:

```
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
```

Vanishing gradient problem



1.Smaller Gradients
2. Learn Very Slow
3. Almost Random

1. Larger Gradients
2. Learn Very Fast
3. Converged Already

Backpropagation (From Last Lecture)

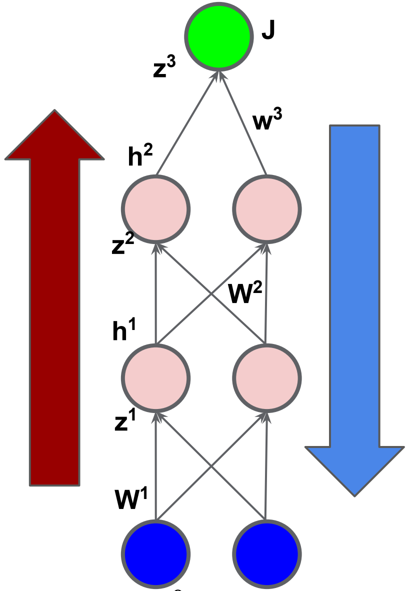
Definition（from wiki):

By computing the gradient of the loss function with respect to each weight by the **chain rule**, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule

Predictions flows forward

$$h^l = \sigma(\mathbf{z}^l)$$

$$\mathbf{z}^l = \mathbf{W}^l h^{l-1} + \mathbf{b}^l$$



$$\mathbf{x} = \mathbf{h}^0$$

Error flows backward

$$\frac{\partial J}{\partial \mathbf{w}^3} = \frac{\partial J}{\partial \mathbf{z}^3} \frac{\partial \mathbf{z}^3}{\partial \mathbf{w}^3}$$

$$\frac{\partial J}{\partial \mathbf{W}^2} = \underbrace{\frac{\partial J}{\partial \mathbf{z}^3}}_{\text{From } \mathbf{w}^3} \frac{\partial \mathbf{z}^3}{\partial \mathbf{h}^2} \frac{\partial \mathbf{h}^2}{\partial \mathbf{z}^2} \frac{\partial \mathbf{z}^2}{\partial \mathbf{W}^2}$$

$$\frac{\partial J}{\partial \mathbf{W}^1} = \underbrace{\frac{\partial J}{\partial \mathbf{z}^3}}_{\text{From } \mathbf{w}^3} \underbrace{\frac{\partial \mathbf{z}^3}{\partial \mathbf{h}^2} \frac{\partial \mathbf{h}^2}{\partial \mathbf{z}^2}}_{\text{From } \mathbf{W}^2} \frac{\partial \mathbf{z}^2}{\partial \mathbf{h}^1} \frac{\partial \mathbf{h}^1}{\partial \mathbf{z}^1} \frac{\partial \mathbf{z}^1}{\partial \mathbf{W}^1}$$

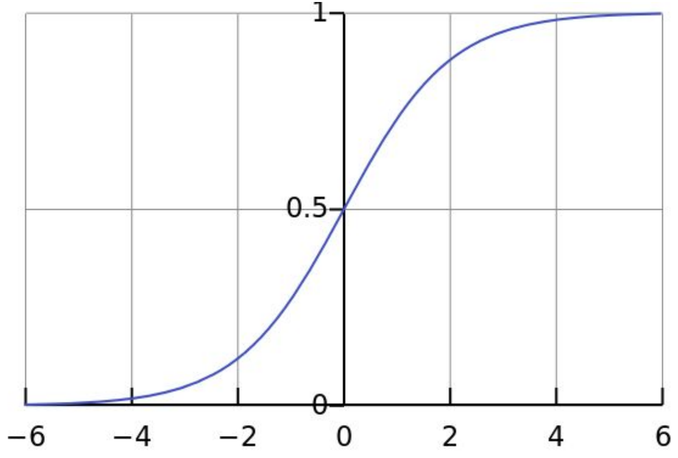
More details:
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Sigmoid function

Equation:

$$f(x) = \frac{1}{1+e^{-x}}$$

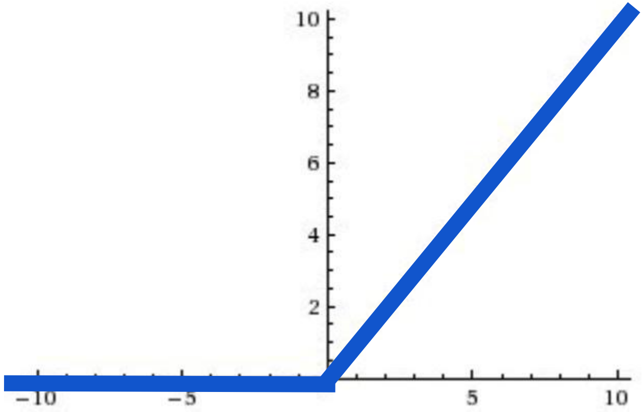
- Vanishing Gradient Problem



How about gradient curve？

ReLU function

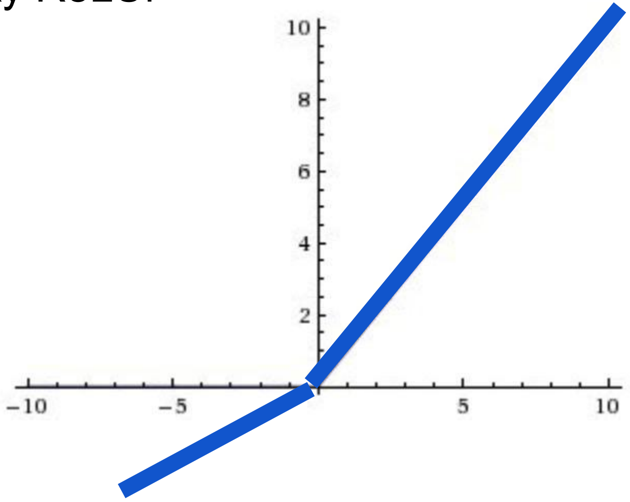
- Fast Compute
- Still have vanishing gradient problem



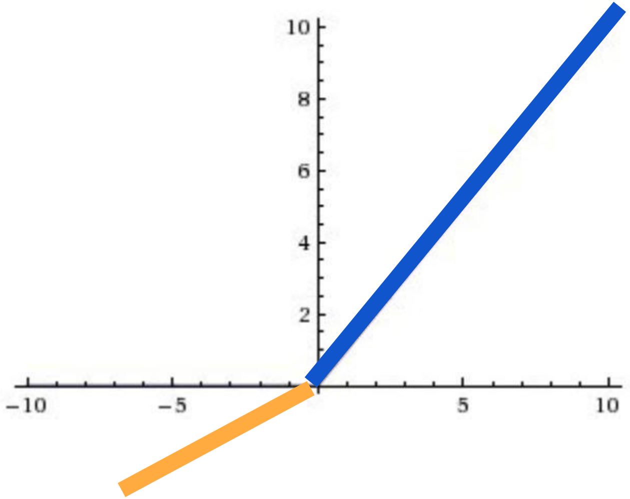
How about gradient curve？

ReLU variants

Leaky ReLU:



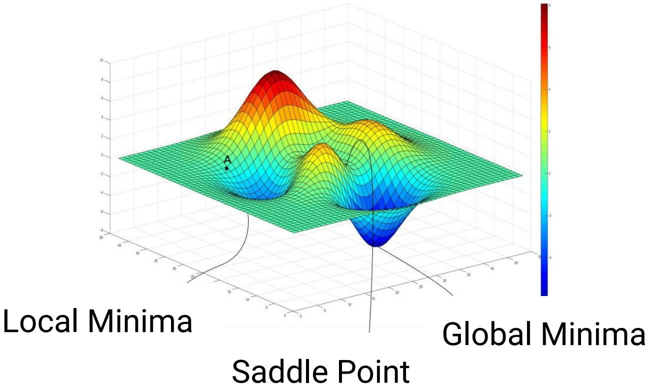
Parametric ReLU::



4. Parameters Initializations

Initialization

- Optimization for neural network in nature is a iterative method, which requires initialization



- Some general rules for initialization of model parameters:
  - Can not initialize all weights to the same value
  - Randomness should be incorporated

Normal distribution

- Initialize weights randomly, following standard normal distribution
  - The normal distribution should take into account characteristics that are unique to the architecture

For Layers with ReLu

$$\sqrt{\frac{2}{size^{[l-1]}}}$$

$W^{[l]}$ = np.random.randn(size_l, size_l-1) * np.sqrt(2/size_l-1)

https://keras.io/api/layers/initializers/

For Layers with Tanh/Sigmoid

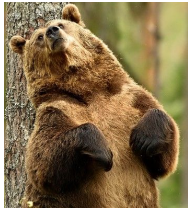
$$\sqrt{\frac{1}{size^{[l-1]}}}$$

$W^{[l]}$ = np.random.randn(size_l, size_l-1) * np.sqrt(1/size_l-1)

https://datascience.stackexchange.com/questions/17987/how-should-the-bias-be-initialized-and-regularized

Transfer learning

Task: Build a bear/cat classifier



bear

cat

Available Data: not **directly** related

dog

panda

similar domain, different tasks

bear

cat

Different domains, similar task

Applications

- Sentiment Analysis
  - Available data: IMDB reviews

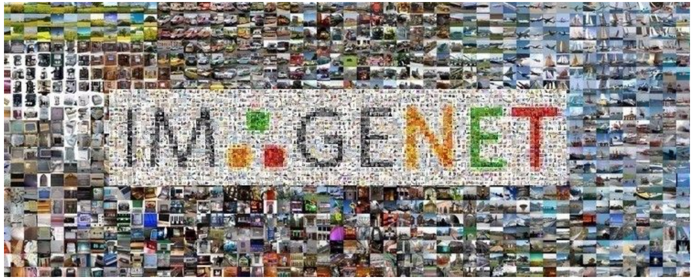

User Reviews

Sensual Italian summer
25 August 2017 | by candelach – See all my reviews

This film is pure sensuality and emotion. You can see through the character's eyes, taste through their mouths but most importantly you feel, by God how you feel. Luca Guadagnino manages to extract the very best out of his actors: Armie Hammer's performance shows unprecedented depth and Timothee Chalamet is the essence of awe inspiring acting, this film will land him an Oscar nomination at the very least.

303 of 462 people found this review helpful. Was this review helpful to you? Yes No | Report this

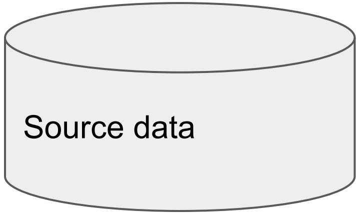
  - Target taks: Teaching feedback analysis
- Image Classification:
  - Available data: Imagenet Dataset



  - Target Task: Cancer Diagnostic (Medial Image)

How to transfer knowledge

- Task Definition:



(x^s, y^s) Large-size

(x^t, y^t) Small-size

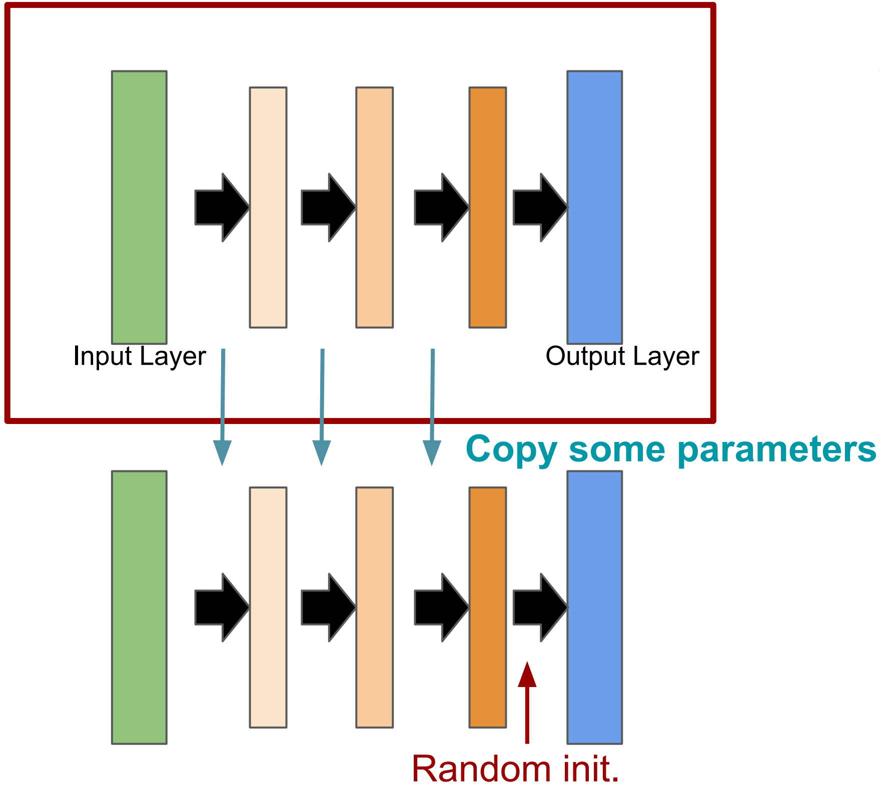
- Steps:
  - Train a model using the source data
  - Transfer layer from the model trained in source domain to the model in target domain
  - Fine-tune the model using the target data

Any concerns?

Layer transfer

Source Data

Target Data



Neural Network: Layer-wise self-contained

1. **Same Task:**
Copy all layers' parameters
2. **Different Tasks:**
Random initialize the softmax/last layer and copy the rest layers' parameters

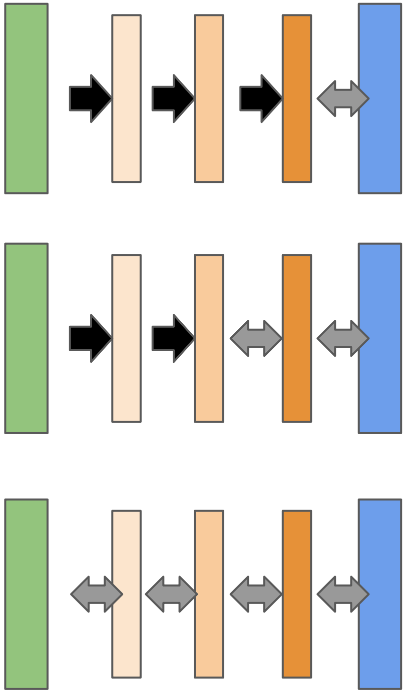
Fine-tune

Target Data Size

Small

Medium

Large



Freeze all layers, train weights on softmax/regression layer

Freeze most layers, train weights on last layers and softmax/regression layer

Fine-tune all layers

5. Optimizers for Neural Network

SGD

Gradient for loss function f over parameters, which computed by BP algorithm

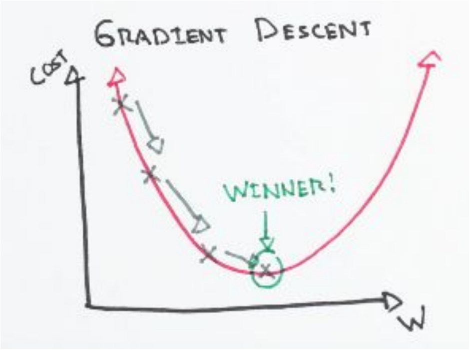
$$\theta_{t+1} = \theta_t - \alpha \boxed{\nabla J(\theta_t)}$$

New Parameters Guess

Current Parameters Guess

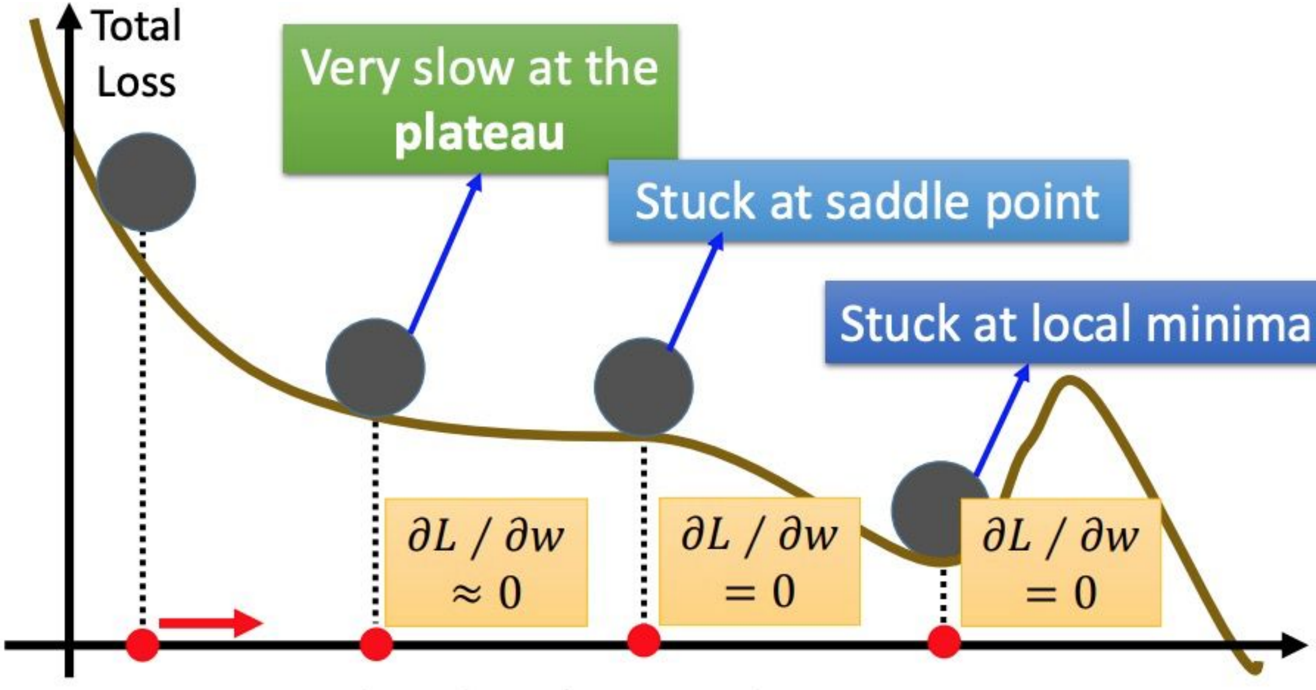
Learning Rate

Like hiking down a mountain



Credit:https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

Hard to find optimal network parameters



Source: https://speech.ee.ntu.edu.tw/~tlkagk/

Momentum

- Core idea: the current gradient computation will keep the direction as the previous gradient computation

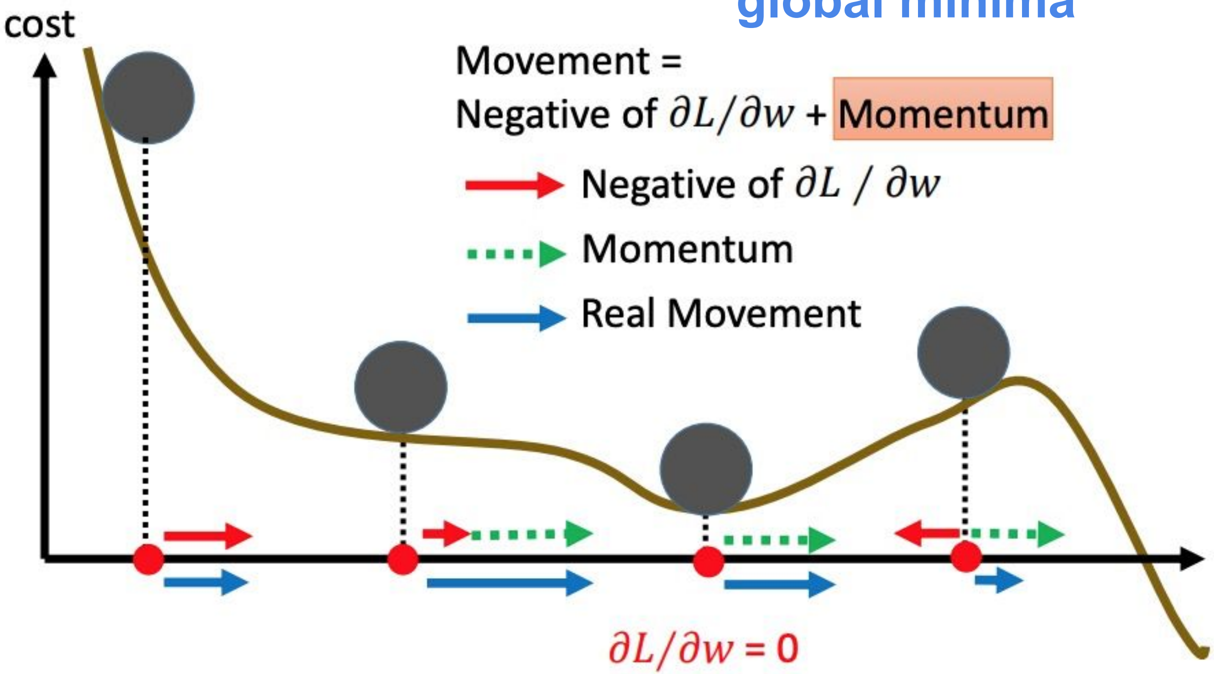
Updated vector of the
previous time step

$$v_t = \beta v_{t-1} + \alpha \bigtriangledown J(\theta_t)$$
$$\theta_{t+1} = \theta_t - v_t$$

- Accelerate SGD
- Dampens Oscillations
- Two Parameters to tune

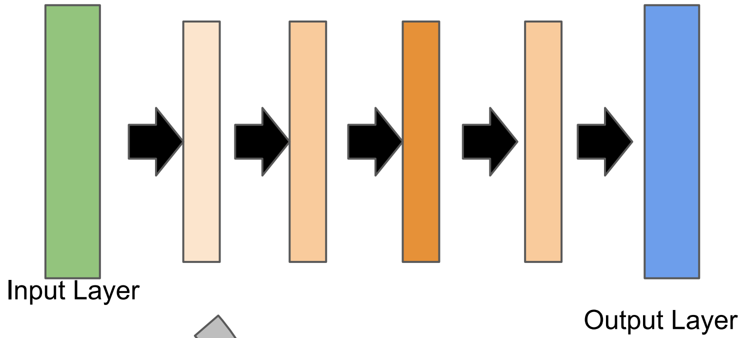
Momentum

Improve the chance to find the global minima



Source: https://speech.ee.ntu.edu.tw/~tlkagk/

Separated adaptive learning rate



Parameters Update = Gradients * Learning Rate

1.Smaller Gradients
2. Learn Very Slow
3. Almost Random

Large Learning Rate

1. Larger Gradients
2. Learn Very Fast
3. Converged Already

Small Learning Rate

Keep a moving average of the squared gradient for each parameter to change the learning rate.

How to select the optimizer

- Except SGD, Momentum, RMSprop and Adam, other popular methods include Adadelta and Adagrad.
- It is hard to find a general answer
- Adam is the most commonly used technique
- If you want to train a deep or complex neural networks with fast converge, do not just use SGD.

6. Regularization Techniques

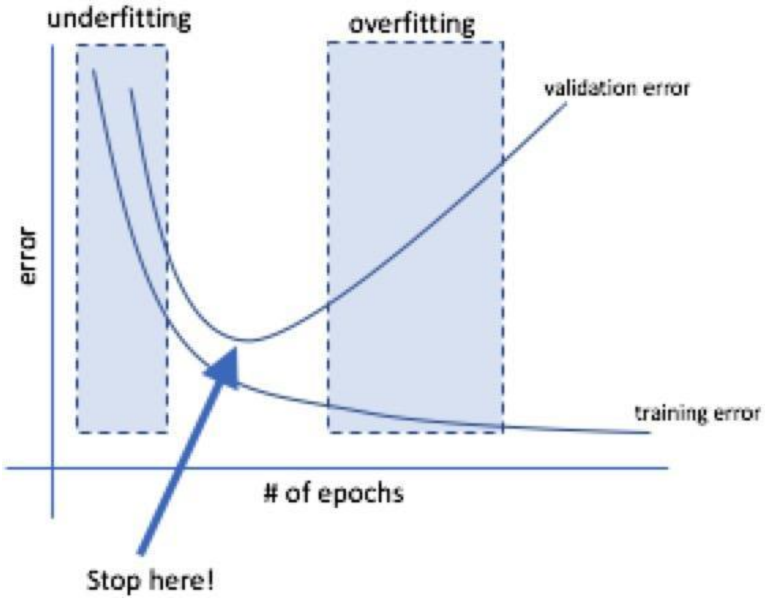
Overfitting for NN

- Neural Network with a deep structure easily get overfitted
  - Early stopping
  - Parameters Regularization
  - Dropout
  - Most effective: Train with more data.

Early stopping

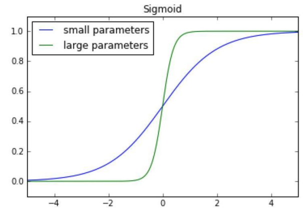
- Watch the validation curve
- Stop updating the weights once validation errors starts increasing

In Keras: https://keras.io/api/callbacks/early_stopping/



Parameter regularization

- Why large model parameters should be penalized:
  - In NN, inputs are linearly combined with parameters. Therefore, large parameters can amplify small changes in the input.
  - Large parameters may **arbitrarily** increases the confidence in our predictions.



To make sure that parameters are not too large and then the model is not overfitting
Add regularization terms to the loss function

$$\ldots\ldots + \lambda g(\theta)$$

Control the degree to which we select to penalize large parameters

Regularization terms

- L1 Regularization:

$$g(\theta) = ||\theta||_1$$

L1-norm is commonly used for feature selection as it tends to produce sparse parameter vectors where only the important features take on non-zero values

- L2 Regularization:

$$g(\theta) = ||\theta||_2^2$$

L2-Norm does not tend to push less important weights to zero and typically produces better results when training a model.

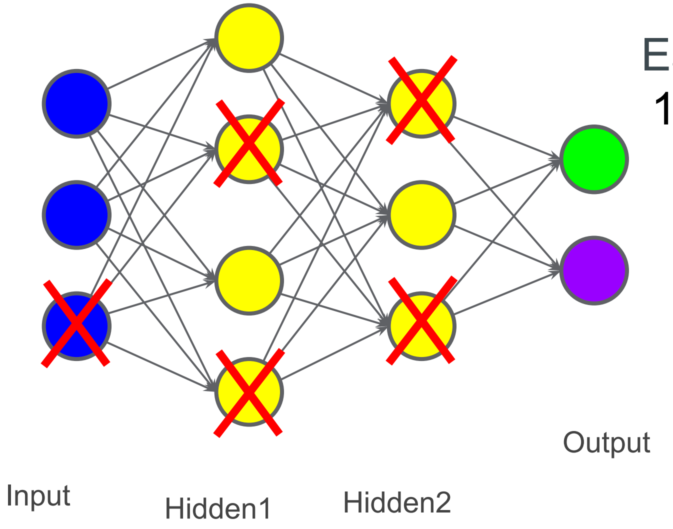
- Elastic Net:

$$g(\theta) = \alpha||\theta||_1^1 + (1-\alpha)||\theta||_2^2$$

Trade-off between L1 and L2 Regularization techniques

Dropout

Training:

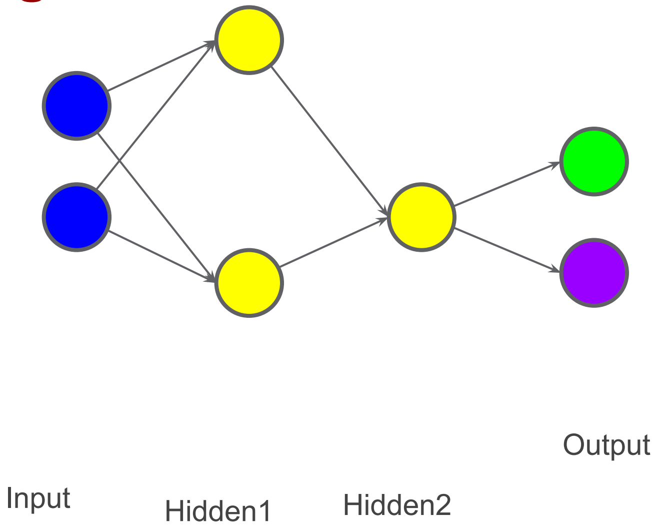


Each mini-batch before updating the parameters

1. Each neuron has **%p** to dropout(mask)

Dropout

Training:



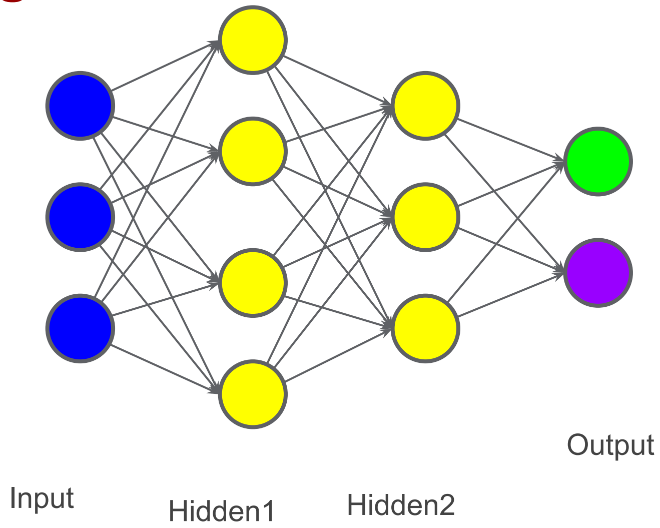
Each mini-batch before updating the parameters

1. Each neuron has **%p** to dropout(mask)
2. The network structure is changed (More Thinner!)
3. Using the updated network structure for training

For each mini-batch, we resample the dropout neurons.

Dropout

Testing:



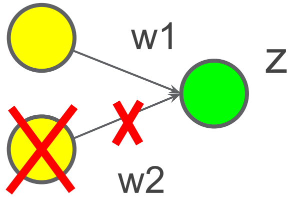
No dropout, but shrink weights following the rule:

If the dropout rate during training is p%, all the weights will time 1-p%.

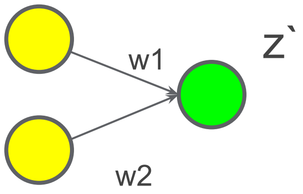
When many people work together, they usually rely on others to do more of the work and share the same results.

Dropout

Training: Assume dropout rate is 50%



Testing: No dropout



Directly Copy:

z` = 2z

Weight multiply 1-p%:

z` ~= z

Dropout effects

Experimental Studies on MNIST dataset:

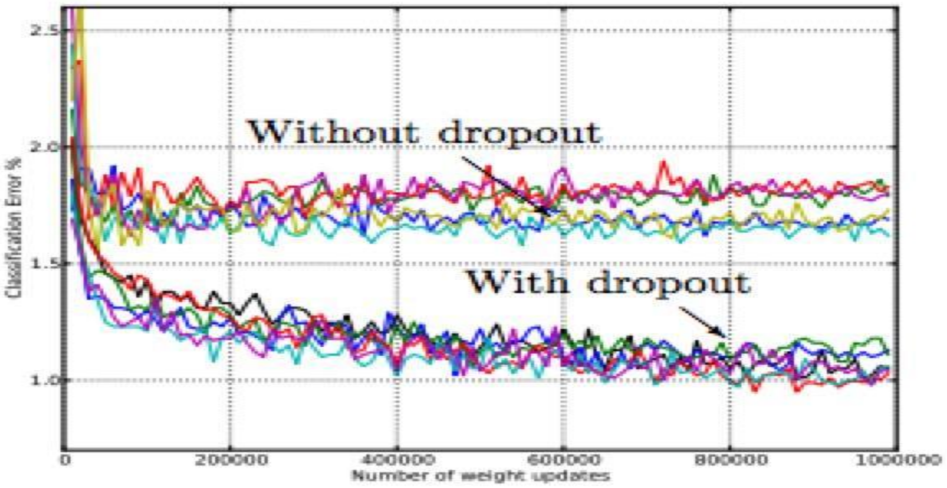


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

7. Do not sleep on traditional machine learning

Why do tree-based models still outperform deep learning on tabular data?

Léo Grinsztajn
Soda, Inria Saclay
leo.grinsztajn@inria.fr

Edouard Oyallon
ISIR, CNRS, Sorbonne University

Gaël Varoquaux
Soda, Inria Saclay

Abstract

Model comparison

Tree-based Models outperform deep learning on tabular data

Based on 45 middle-sized datasets (10, 000 samples)

From this paper, authors explain:

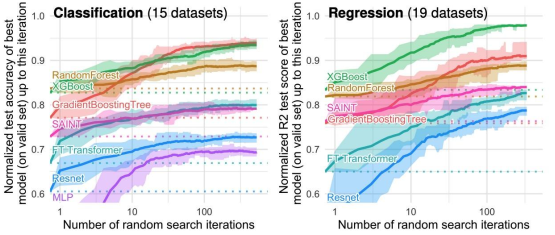


Figure 1: **Benchmark on medium-sized datasets, with only numerical features**. Dotted lines correspond to the score of the default hyperparameters, which is also the first random search iteration. Each value corresponds to the test score of the best model (on the validation set) after a specific number of random search iterations, averaged on 15 shuffles of the random search order. The ribbon corresponds to the minimum and maximum scores on these 15 shuffles.

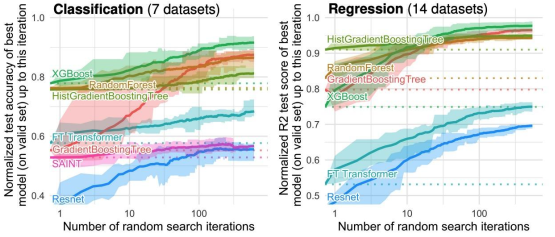


Figure 2: **Benchmark on medium-sized datasets, with both numerical and categorical features**. Dotted lines correspond to the score of the default hyperparameters, which is also the first random search iteration. Each value corresponds to the test score of the best model (on the validation set) after a specific number of random search iterations, averaged on 15 shuffles of the random search order. The ribbon corresponds to the minimum and maximum scores on these 15 shuffles.

- Deep learning bias to the overly smooth solution, while tree-based models are able to generate irregular decision boundaries
- Deep learning are very sensitive to uninformative features which could be easily spotted in tabular data, while tree-based models are more robust

Deep Learning for time series data

- “Results show that competitive performance can be achieved with a conventional machine learning pipeline consisting of **preprocessing**, **feature extraction**, and a **simple machine learning model**. In particular, we analyze the performance of a linear model and a non-linear (gradient boosting) model”

Table 3
Comparison between the proposed classical machine learning pipeline and other (deep learning) solutions using macro F1-score (MF1), overall accuracy (ACC), and Cohen’s Kappa coefficient (κ). The approaches are sorted according to macro F1. The scores in bold represent the best score for each dataset (that are comparable to our approach).

Dataset	Year	System	Technique	LP	MF1	ACC	κ	Signals
	2021	RobustSleepNet [28]	RNN	FT	0,817	–	–	EEG + EOG
	2022	*This work*	Catboost	DT	**0,810**	**0,866**	**0,816**	EEG + EOG + EMG
	2021	XSleepnet2 [46]	CNN & RNN	LFS	0,809	0,864	0,813	EEG + EOG
	2022	*This work*	Logistic regr.	LFS	0,809	0,857	0,806	EEG + EOG + EMG
	2022	*This work*	Logistic regr.	DT	0,805	0,863	0,813	EEG + EOG + EMG
	2020	TinySleepNet [47]	CNN & RNN	LFS	0,805	0,854	0,800	EEG
	2020	SimpleSleepNet [48]	RNN	LFS	0,805	–	–	EEG + EOG
	2022	*This work*	Logistic regr.	LFS	0,803	0,853	0,800	EEG + EOG
	2022	*This work*	Catboost	LFS	0,802	0,864	0,812	EEG + EOG + EMG
Sleep-EDF-SC-20	2020	XSleepnet1 [46]	CNN & RNN	LFS	0,798	0,852	0,798	EEG + EOG
	2022	*This work*	Catboost	LFS	0,797	0,860	0,807	EEG + EOG
	2019	SleepEEGNet [44]	CNN & RNN	LFS	0,797	0,843	0,790	EEG
	2020	SeqSleepNet+ [45]	RNN	FT	0,796	0,852	0,789	EEG
	2021	RobustSleepNet [28]	RNN	LFS	0,791	–	–	EEG + EOG
	2021	RobustSleepNet [28]	RNN	DT	0,791	–	–	EEG + EOG
	2020	DeepSleepNet+ [45]	CNN	FT	0,790	0,846	0,782	EEG + EOG
	2021	DeepSleepNet-Lite [15]	CNN	LFS	0,780	0,840	0,780	EEG
	2019	IITNet [43]	CNN & RNN	LFS	0,776	0,839	0,780	EEG
	2017	DeepSleepNet [41]	CNN & RNN	FT	0,769	0,820	0,760	EEG
	2022	SleepTransformer [40]	transformer	FT	0,788	0,849	0,789	EEG
	2021	XSleepnet2 [46]	CNN & RNN	LFS	**0,787**	**0,840**	**0,778**	EEG + EOG
	2020	XSleepnet1 [46]	CNN & RNN	LFS	0,784	0,840	0,777	EEG
	2020	TinySleepNet [47]	CNN & RNN	LFS	0,781	0,831	0,770	EEG
	2021	RobustSleepNet [28]	RNN	FT	0,779	–	–	EEG + EOG
	2022	*This work*	Catboost	LFS	0,775	0,831	0,766	EEG + EOG + EMG
Sleep-EDF-SC-78	2022	*This work*	Catboost	LFS	0,772	0,830	0,763	EEG + EOG
	2022	*This work*	Logistic regr.	LFS	0,771	0,821	0,756	EEG + EOG + EMG
	2022	*This work*	Logistic regr.	LFS	0,768	0,820	0,753	EEG + EOG
	2021	RobustSleepNet [28]	RNN	LFS	0,763	–	–	EEG + EOG
	2021	DeepSleepNet-Lite [15]	CNN	LFS	0,752	0,803	0,730	EEG
	2022	SleepTransformer [40]	transformer	LFS	0,743	0,814	0,743	EEG
	2021	RobustSleepNet [28]	RNN	DT	0,738	–	–	EEG + EOG
	2019	SleepEEGNet [44]	CNN & RNN	LFS	0,736	0,800	0,730	EEG
	2021	RobustSleepNet [28]	RNN	FT	0,810	–	–	EEG + EOG
	2022	*This work*	Catboost	LFS	**0,795**	**0,836**	**0,765**	EEG + EOG + EMG
	2022	*This work*	Logistic regr.	LFS	0,792	0,829	0,759	EEG + EOG + EMG
	2021	RobustSleepNet [28]	RNN	DT	0,791	–	–	EEG + EOG
Sleep-EDF-ST	2022	*This work*	Catboost	LFS	0,789	0,832	0,758	EEG + EOG
	2022	*This work*	Logistic regr.	LFS	0,788	0,825	0,754	EEG + EOG
	2021	RobustSleepNet [28]	RNN	LFS	0,786	–	–	EEG + EOG
	2020	DeepSleepNet+ [45]	CNN	FT	0,775	0,815	0,738	EEG
	2020	SeqSleepNet+ [45]	RNN	FT	0,775	0,810	0,734	EEG
	2020	SimpleSleepNet [48]	RNN	LFS	**0,847**	–	–	EEG + EOG
	2021	RobustSleepNet [28]	RNN	FT	0,840	–	–	EEG + EOG
	2020	TinySleepNet [47]	CNN & RNN	LFS	0,832	**0,875**	**0,820**	EEG
	2021	RobustSleepNet [28]	RNN	LFS	0,822	–	–	EEG + EOG
	2022	*This work*	Catboost	LFS	0,817	0,867	0,803	EEG + EOG + EMG
MASS SS3	2017	DeepSleepNet [41]	CNN & RNN	FT	0,817	0,862	0,800	EEG
	2022	*This work*	Catboost	LFS	0,809	0,863	0,797	EEG + EOG
	2021	RobustSleepNet [28]	RNN	DT	0,808	–	–	EEG + EOG
	2022	*This work*	Logistic regr.	LFS	0,807	0,853	0,786	EEG + EOG + EMG
	2019	IITNet [43]	CNN & RNN	LFS	0,805	0,863	0,790	EEG
	2021	U-Sleep [29]	CNN	DT	0,800	–	–	EEG + EOG
	2022	*This work*	Logistic regr.	LFS	0,794	0,845	0,775	EEG + EOG

Do not sleep on traditional machine learning: Simple and interpretable techniques are competitive to deep learning for sleep scoring ☆

Jeroen Van Der Donckt [1], Jonas Van Der Donckt [1], Emiel Deprost, Nicolas Vandenbussche, Michael Rademaker, Gilles Vandewiele, Sofie Van Hoecke

Show more

Source: https://www.sciencedirect.com/science/article/abs/pii/S1746809422008837

Deep Learning for unstructured data

- Deep learning are good at capturing high dimensional and spatial patterns/interactions among data
- Therefore, in those domains such as image, video, and text, deep learning is able to achieve huge success especially enough data are present

Next Class: Auto-encoders