UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
School of Professional & Executive Development

Soft Computing Research Group

Deep learning

Lluís A. Belanche
Computer Science Department
Universitat Politècnica de Catalunya
belanche@cs.upc.edu

Things you need to know to work in Deep Learning
Part II

Version of February 21, 2025

DEEP LEARNING

"All human knowledge begins with intuitions, proceeds from thence to concepts, and ends with ideas."

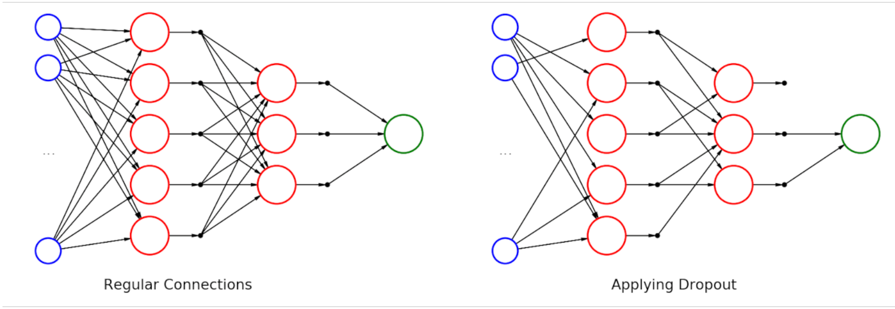
— Immanuel Kant. *Critique of Pure Reason*.

- Parameter search strategies: delicate vs permissive **hyperparameters**, nested cross-validations, Bayesian Optimization, ranges ans steps, ...
- Data partitioning and resampling (correct and wrong)
- Learning effects: **overparameterization**, **double-descent** phenomenon, grokking
- Methods for evaluating and comparing different models
- New **regularization** ideas: **dropout**, batch normalization, layer normalization as well as "old" ones (e.g., L_2)
- Effect of initialization
- Fine-tuning, transfer learning
- (Automatic and explicit) feature extraction
- **Vanishing** and Exploding **Gradients**
- Data Augmentation and Preprocessing (e.g., images)
- Dealing with **Label Noise** (sic), Imbalanced Datasets
- New optimizers and their hyperparameters
- **Computational constraints**

Parameter search ideas to bear in mind:

- classsify/sort delicate vs permissive hyperparameters
- can be make any reasonable independence assumptions?
- decide if nested cross-validations (2,3, ...) are really need
- decide ranges ans steps
- decided what is variable and what is held constant
- start with reasonable defaults
- perform logarithmic (base?) instead of linear searches
- perform nested sets of values instead of linear ones
- watch the size of the validation set!!!

Assorted hyperparameters in deep networks:
Learning Rate, Learning Rate Decay, Momentum, Nesterov Momentum, Adam β_1, Adam β_2, Adam ϵ, RMSprop Decay Rate, RMSprop ϵ, SGD Momentum, Adagrad, Adadelta or Nadam parameters, L1 Regularization, L2 Regularization, Dropout Rate, Batch Normalization Momentum, Batch Normalization ϵ, Early Stopping, Number of Layers, Number of Neurons per Layer, Activation Function, Filter Size, Number of Filters, Stride, Padding, Hidden State Size, Sequence Length, Dropout in RNN Cells, Number of Attention Heads, Number of Encoder Layers, Number of Decoder Layers, Feedforward Network Size, Dropout Rate in Attention Layers, Batch Size, Number of Epochs, Data Augmentation, Normalization Method, Shuffling Strategy, Gradient Clipping, Label Smoothing, Temperature Scaling, Knowledge Distillation Parameters, Alpha for Mixup, Alpha for CutMix, ...



On the left is a fully connected neural network with two hidden layers. On the right is the same network after applying dropout.
– from the *Wikipedia*

Setting the Dropout Rate

Depends on the type of neural network (MLP, CNN, LSTM, Transformers ,...), whether they are shallower or deeper, the sample size; all this suggests a default value or a specific range and starting value. If in doubt, consult the literature.

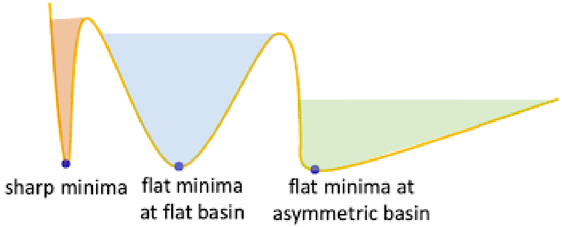
...

Overparameterization occurs when a (deep) neural network has more learnable weights than necessary (?) to learn the given task.

MNIST 10 digits, 28×28 images, 60K examples

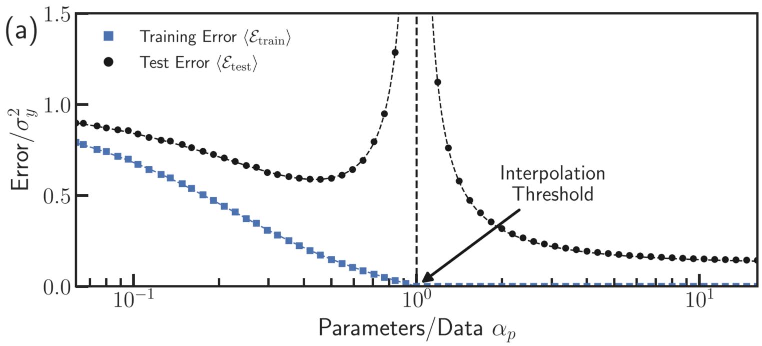
- Under-parameterized model: MLP with one hidden layer of 20 neurons ($\sim$ 16K weights)
- Well-parameterized model: A simple CNN with few layers (e.g., 100K weights)
- Overparameterized model: A complex CNN arquitecture (e.g., 100M weights).

Why does SGD find highly generalizable parameters from non-convex target functions (such as the loss function of NNs)?



Caution 1! "Overparameterization"

In **overparameterized** networks, SGD prefers flat minima (regions in the loss landscape where small changes in parameters don't affect performance much), which generalize better.



Caution 2! "Double descent phenomenon"

- As model complexity increases it would eventually start to overfit the training data.
- If you keep increasing complexity, its performance on unseen data actually starts to improve again! Kind of a "second descent" in the predictive error rate.

Vanishing gradients:

- **Gradients** tell us the direction and magnitude of the steepest ascent of the loss function.
- **Backpropagation** calculates gradients for each weight by propagating the error signal from the output layer back through the network. It relies on the chain rule of calculus.
- A problem occurs when gradients become **exponentially smaller** as they are backpropagated through many layers.

Example. Suppose we have a network with 3 layers, and we're focusing on a single path through the network

Layer 1: $z_1 = W_1 x, a_1 = \sigma(z_1)$

Layer 2: $z_2 = W_2 a_1, a_2 = \sigma(z_2)$

Layer 3: (Output) $z_3 = W_3 a_2, a_3 = \sigma(z_3)$

Let's assume, for simplicity, that all the weights are such that the input to the sigmoid function at each layer is 0 (where the derivative is maximized). This means $\sigma'(z1) = \sigma'(z2) = \sigma'(z3) = 0.25$.

As usual, the chain rule applies. If we focus on the part related to the activations:
$\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1} = \sigma'(z_3)W_3\sigma'(z_2)W_2\sigma'(z_1)$

- If the weights W_2 and W_3 are on the order of 1 (reasonable), and the derivatives are all 0.25, then the product becomes $(0.25)^3 = 0.015625$. This is already a significant reduction.
- In a much deeper network, with many more layers, this repeated multiplication can lead to extremely small gradients, causing the vanishing gradient problem.

- **Label stochasticity** is a *fundamental property* of the data itself. It signifies that, given the same input features, the true label might not be deterministic. There might be inherent uncertainty or ambiguity in the mapping from features to labels. It represents the true, underlying probability distribution of labels given the features. Even a "perfect" labeler might not be able to assign a single, absolutely correct label with 100% certainty.
- **Label noise**, on the other hand, is an *error* or *deviation* from the true label, whether that true label is deterministic or stochastic. It's a mistake in the assigned label due to some imperfection in the labeling process. This could be due to human error, data collection issues, or any other source of error.

Example. Consider predicting the weather (the "label") based on atmospheric conditions (the "features")

- **Stochasticity:** The weather is inherently stochastic. Even with perfect measurements, you can't predict it with 100% certainty. There's inherent randomness. You might say there's a 70% chance of rain, not that it *will* rain.
- **Noise:** Imagine your weather instruments are faulty, giving incorrect readings. These incorrect readings are analogous to label noise. They are errors in your *measurement* of the underlying (possibly stochastic) truth.

Computational resource constraints: Is it possible to make do with a 1,000€ laptop?

What You Can Do:

- Learning and experimentation with smaller datasets and models.
- Smaller projects (e.g., basic image classification, NLP tasks).
- Cloud collaboration for training on powerful GPUs.
- Inference (deployment) of optimized models.

What You Can't Do (not easily):

- Training large, complex models locally
- Processing massive datasets locally
- GPU-accelerated training (most laptops in this price range lack powerful dedicated GPUs)
- Competitive research (massive tuning via CVs!)

Key Considerations:

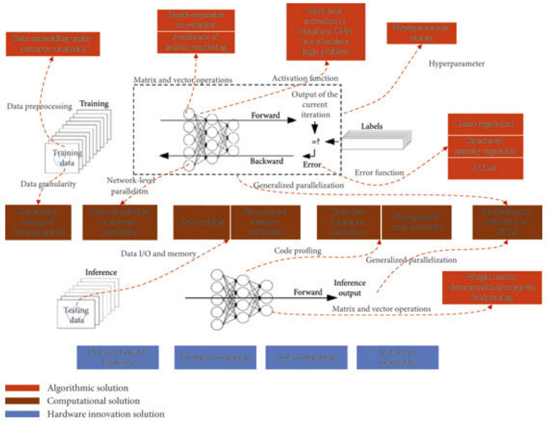
- RAM: Aim for at least 16GB, ideally 32GB
- Storage: SSD is essential
- CPU: Modern multi-core CPU (Intel Core i5, AMD Ryzen 5, ...)
- Cooling: Adequate cooling to prevent throttling.

Recommendations:

1. Leverage cloud computing platforms (Google Colab, AWSs, Kaggle)
2. Focus on learning fundamentals
3. Optimize models for CPU inference if needed
4. Consider used/refurbished laptops with dedicated GPUs

DEEP LEARNING

Introduction to the course



Categorization of existing resource-limited DL solutions: from the perspectives of training and inferencing.

– from Chen, Chunlei et al. *Deep Learning on Computational-Resource-Limited Platforms: A Survey* (2020).

Searching vs. Optimizing

Feature	Searching	Optimizing
Goal	Find an item that *satisfies criteria*	Find the *best* solution according to an *objective function*
"Best"	Meeting requirements	Maximizing/minimizing a *continuous* function
Process	Discrete exploration, often stopping after a match	Iterative refinement, guided by the objective function's gradient
Emphasis	Finding *a* solution that matches	Finding the *optimal* solution

Exercise: learning rate

"Thinking is the hardest work there is, which is probably why so few engage in it."

— Henry Ford.