Module 2 :

Deep Networks

Deep Forward Networks

Géraldine Conti, August 2020

Bibliography

- Deep Learning book (Goodfellow, Bengio, Courville)
- Machine Learning @ Stanford (Prof Andrew Ng)
- Hands-On Machine Learning with Scikit-Learn & Tensorflow (Aurélien Géron)

Learning Objectives

- What are Deep Forward Networks ?
- Regularization for deep learning
- Training and optimization for deep models

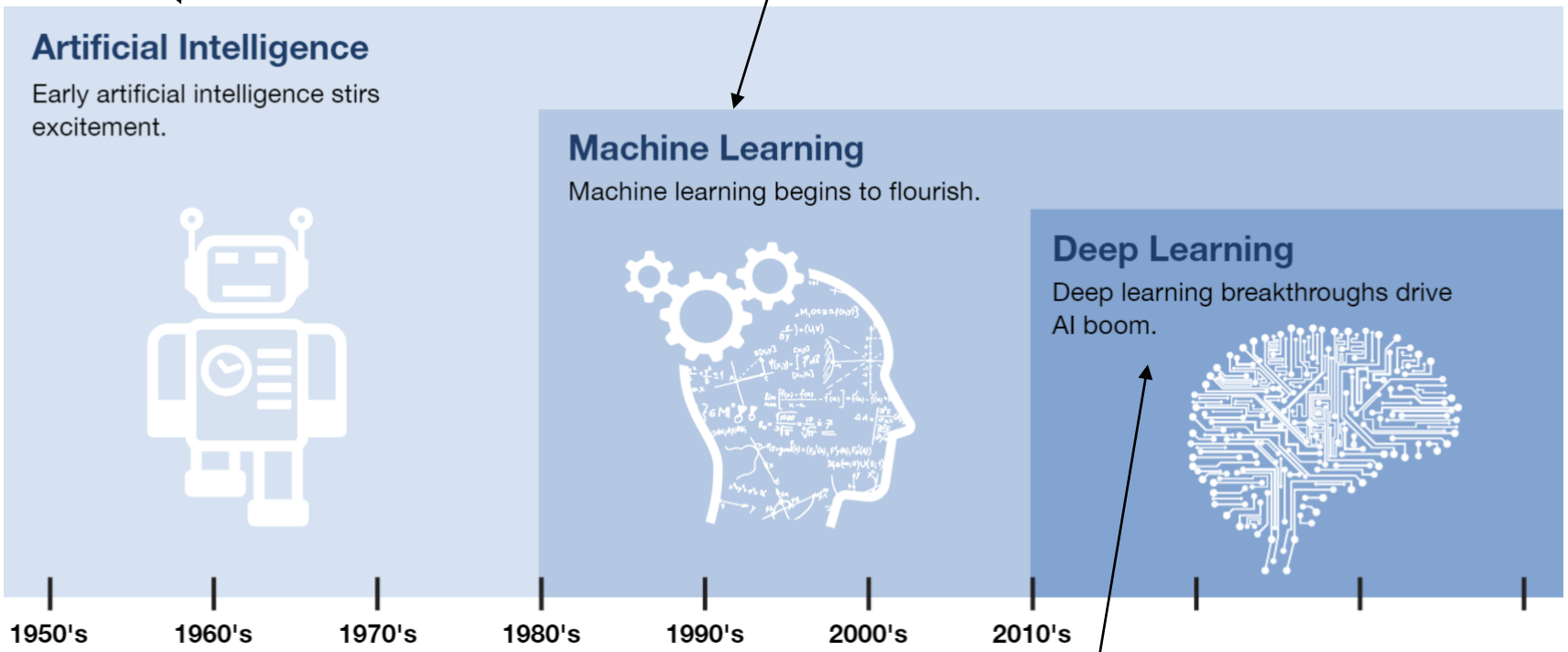
Deep Learning

- Definition
- Motivations
- Comparison with Machine Learning

Deep Learning: Apparition

Ability of a machine to act in a way that imitates intelligent human behaviour

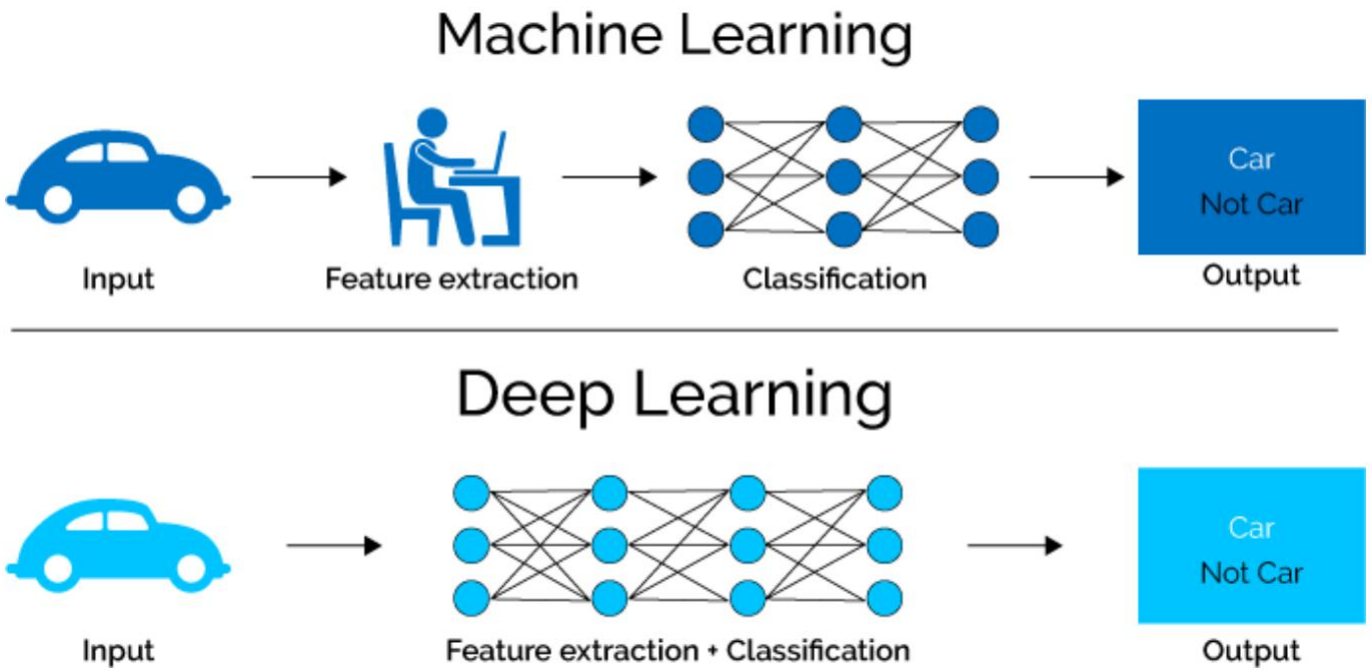
Study of the algorithms and methods that enable computers to solve specific tasks without being explicitly instructed how to solve these tasks



Subset of ML algorithms that make use of large arrays of Artificial Neural Networks

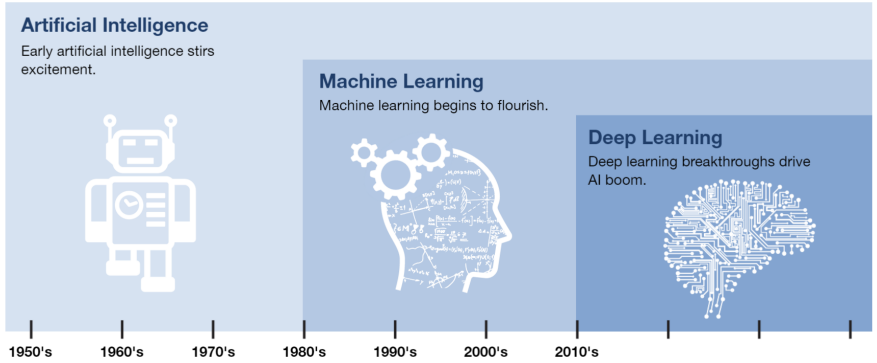
Deep Learning: Definition

- Multiple levels of features represented by multiple layers
- Each level represents abstract features that are discovered from the features represented in the previous level.
  - level of abstraction increases with each level



Deep Learning: Motivations

- How do we get AI ?
  - Knowledge → Learning
  - Generalization
  - Fight the curse of dimensionality

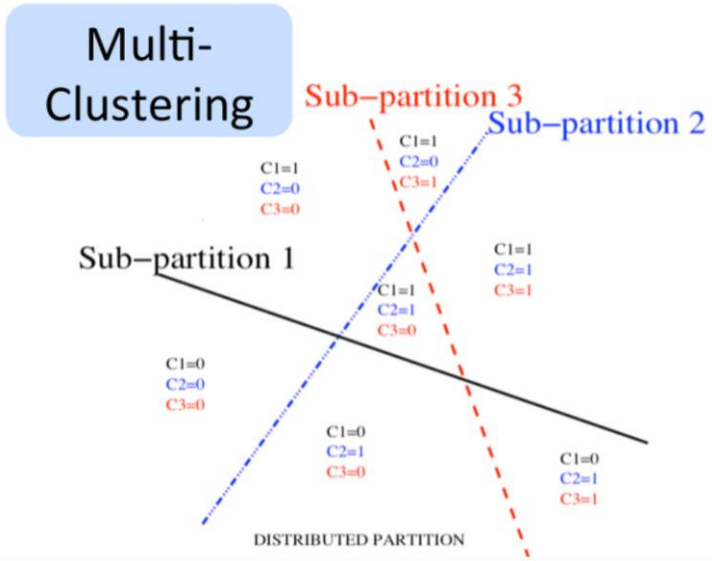


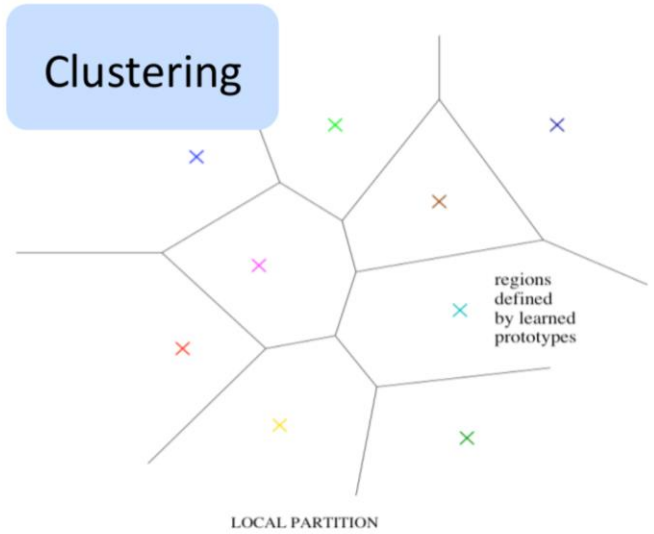
- 3 key ingredients for ML towards AI :
  - Lots of data
  - Very flexible models (as we get more data)
    - *we change the number of hidden units*
  - Powerful priors to defeat curse of dimensionality
    1) distributed representations
    2) deep architecture

1. Distributed Representations

- Possible to represent an exponential number of regions with a linear number of parameters
- Can learn a very complicated function with a low number of examples
- Features are individually meaningful

Algorithms using non-distributed representations ?

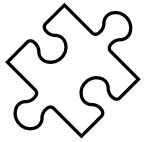




2. Deep Architecture

- Universal approximation property (shallow NN sufficient to represent any function with required degree of accuracy)
- Deep network allows to represent the same function with fewer hidden units.
- Number of units needed in a shallow network can be *exponentially larger* than in deep enough network

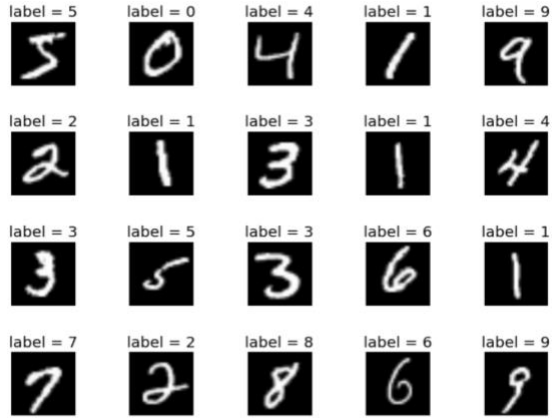
Does depth imply to have a flexible family of functions ?

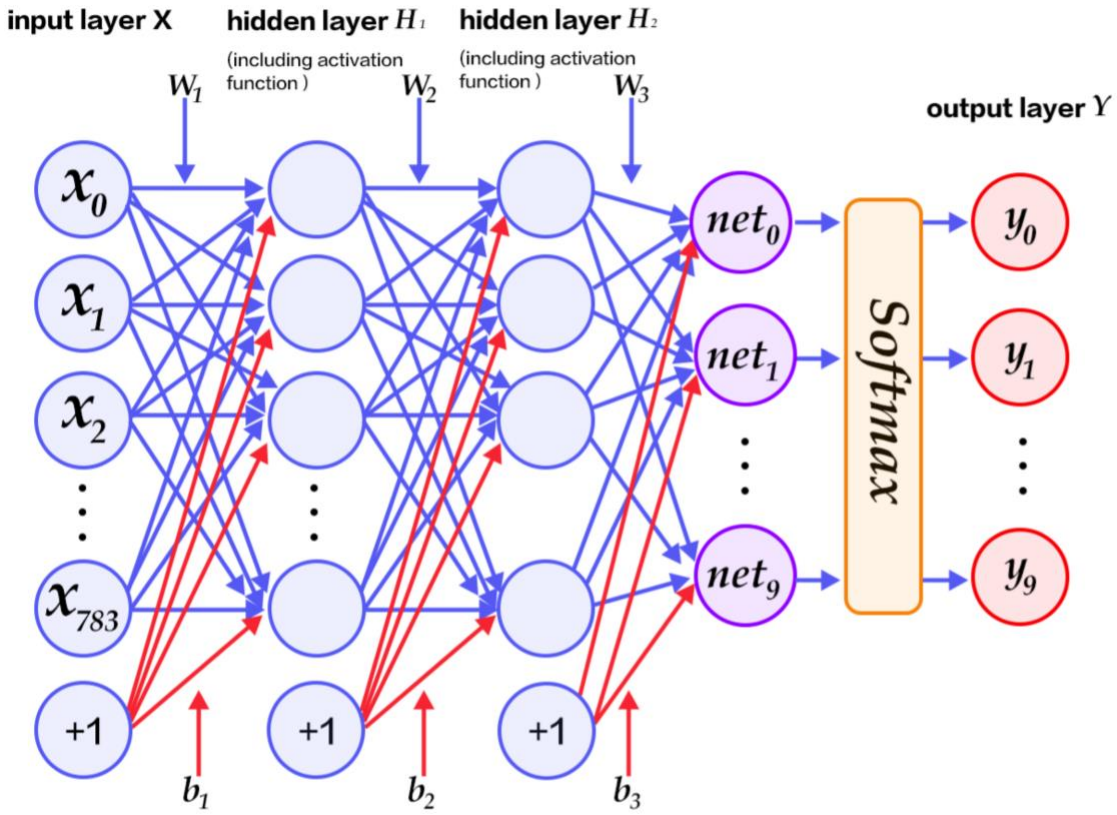


Do deeper networks correspond to a higher capacity ?

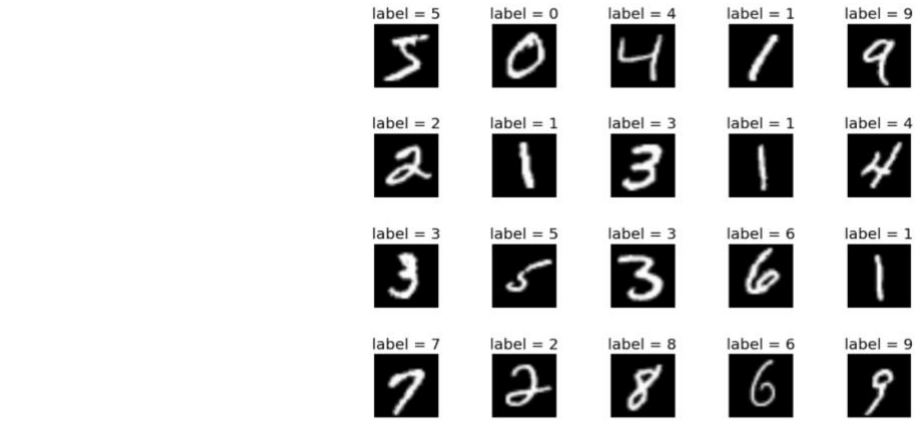
Does deeper mean that we can represent more functions ?

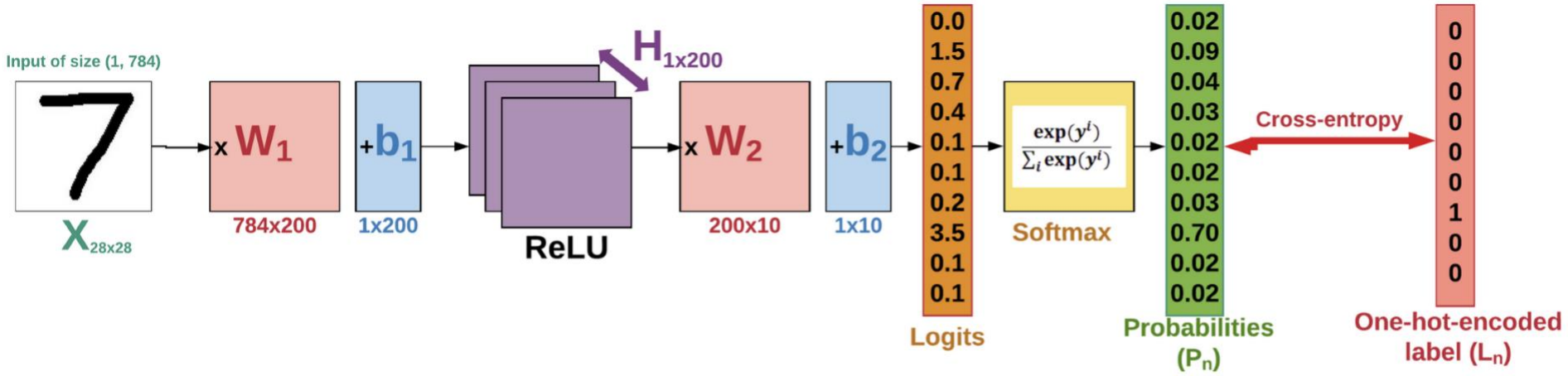
Example : MNIST





Example : MNIST



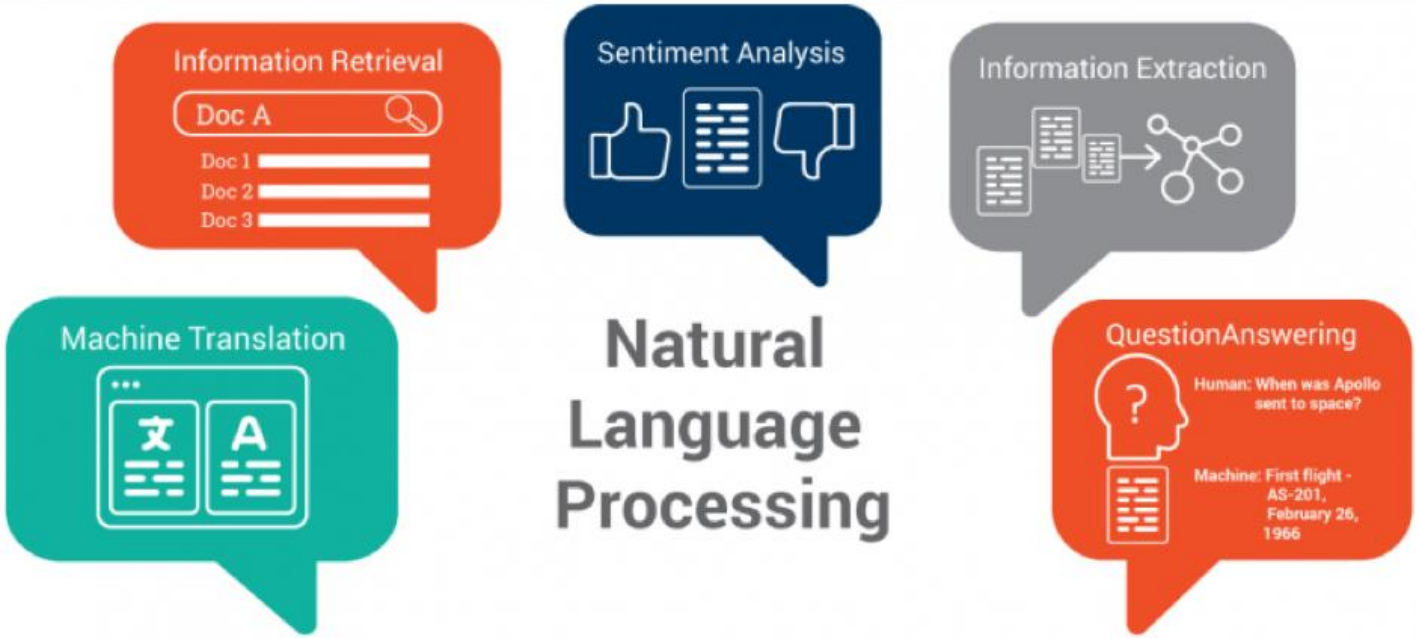


Importance of Deep Nets

- Form a basis for many commercial applications

1) CNNs are a special kind of FNN
   - Used to recognize objects from photos,…

2) They are a conceptual stepping stone to RNNs
   - Power many Natural Language Processing (NLP) applications



DL versus ML

Criteria	Machine Learning	Deep Learning
Data size		
Computer infrastructure		
Feature engineering		
Nature of problem		

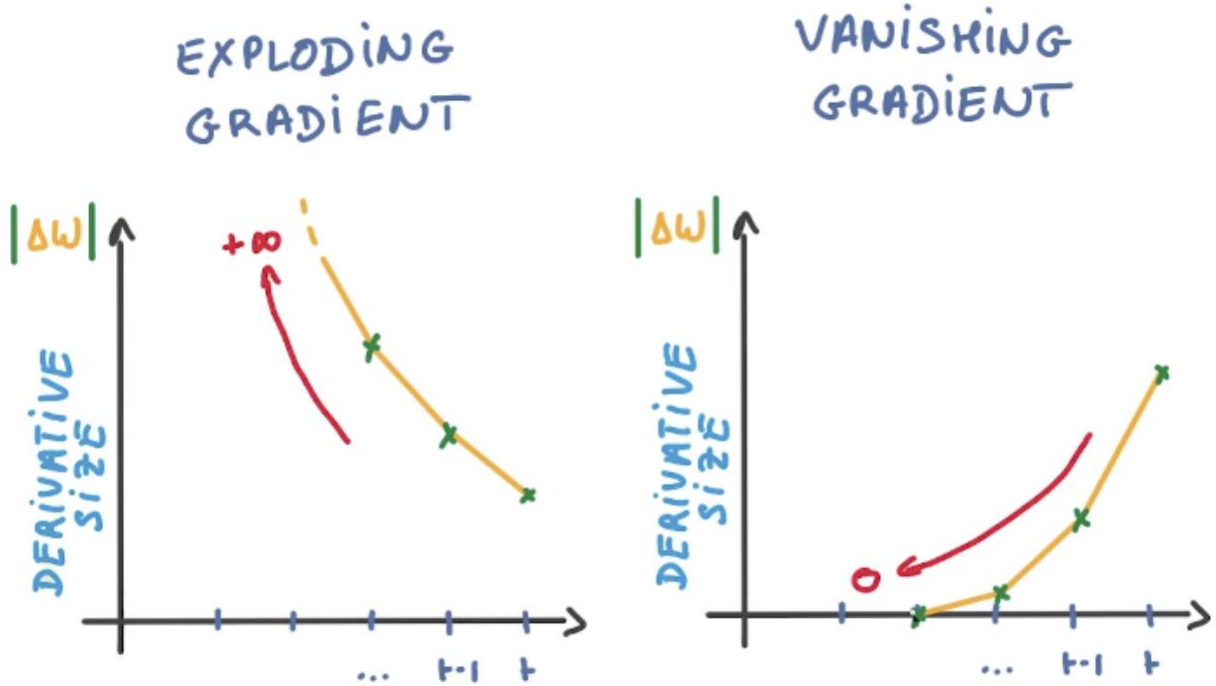
Design Decisions

- Similar to linear model
  - Basics of gradient descent :
    - Optimizer
    - Cost function
    - Form of output units
- Unique to DFNN
  - Concept of hidden layer
    - Activation functions
  - Architecture of network
    - Number of layers
    - How they are connected to each other
    - Number of units in each layer
  - Learning requires gradients of complicated functions
    - Backpropagation

Training Challenges

- Example : 10 layers with 100s of neurons each, connected by hundreds of thousands of connections
- Vanishing/exploding gradient problem
  - Problem affecting deep neural networks
  - Makes lower layers very hard to train
- Training would be extremely slow with such a large network
- Model with millions of parameters would severely risk overfitting the training set

Vanishing/ Exploding gradient problem



Exploding gradient problem (mostly in RNNs) :

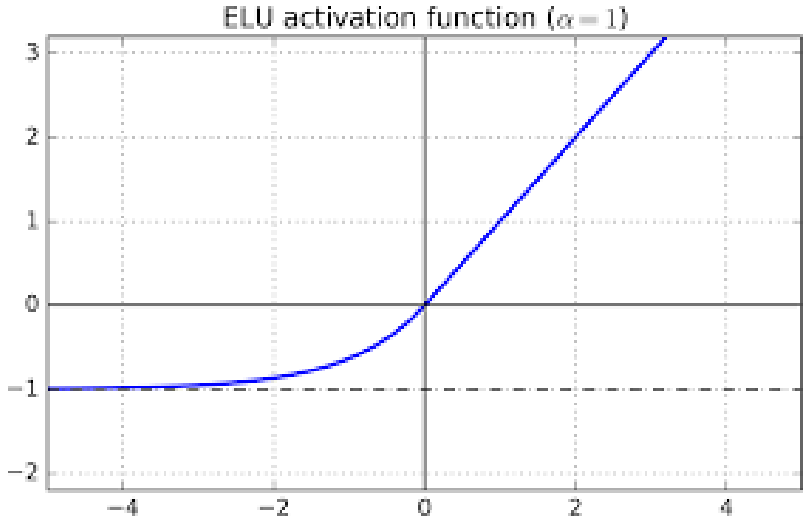
- Gradients grow bigger and bigger
- Many layers get insanely large weight updates
- The algorithm diverges

Vanishing gradient problem :

- Gradients often get smaller and smaller as the algorithm progresses down to the lower layers
- Gradient Descent update leaves the lower layer connection weights virtually unchanged (training never converges to a good solution)

Solution to the Gradient Problem

- Problem found to be a combination of logistic sigmoid activation functions with random initialization of the weights
  - Prefer ELU or any variant of ReLU activation functions



  - Use another weight initialization (see next slide)

Weight Initialization

- The initial parameters need to break the symmetry between different units
- Use Xavier weights, the best ones to prevent gradient problems from happening: random draws from truncated normal distribution with :

$$\mu = 0 \text{ and } \sigma = \sqrt{\frac{2}{a+b}}$$

- Another strategy is to initialize weights by transfering weights learnt via an unsupervised learning method (method also called fine-tuning)

Batch Normalization

- Tackles the vanishing/exploding gradient problems

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Add an operation in the model just before the activation function of each layer to zero-center and normalize inputs:mean/std evaluation using the current mini-batch

Scale and shift the result using two parameters per layer

Networks much less sensitive to weight initialization

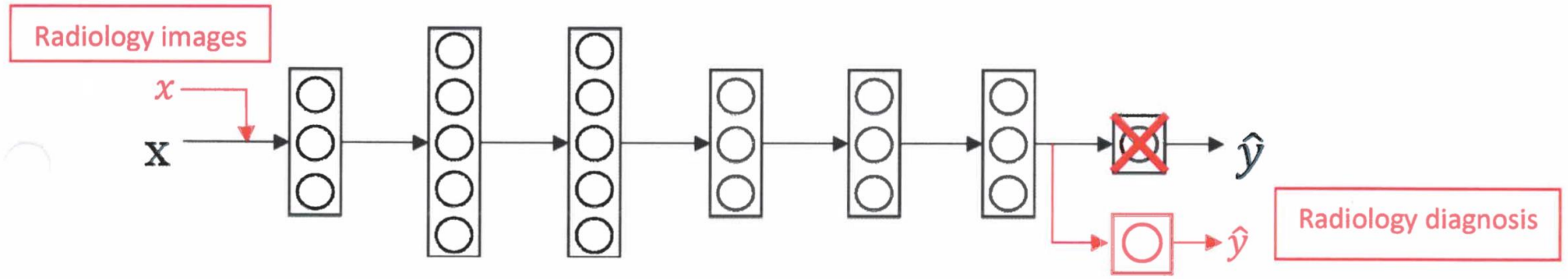
Possible to use much larger learning rates

Gradient Clipping

- Tackles the exploding gradient problem
- Clip the gradient during backpropagation so that they never exceed some threshold
- Mostly used in RNNs
- Nowadays, people prefer batch normalization
  - gradient clipping not ideal because of loss of information

Reusing Pretrained Layers

- Always try to find an existing neural network that accomplishes a similar task to the one you are trying to tackle
- Transfer Learning : reuse the lower layers of this network
  - Output layer should usually be replaced
- Speeds up training and requires much fewer training data



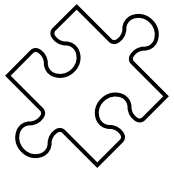
Pre-training : training on cat images
Fine-tuning : update the weights for radiology

Summary of techniques to speed up training

- Good initialization strategy for the connection weights (Xavier)
- Good activation function (ReLU, ELU)
- Batch Normalization
- Reusing parts of a pretrained network (transfer learning)
- Faster optimizer (momentum, Adam)

Exercise

Initialization	He initialization
Activation function	ELU
Normalization	Batch normalization
Regularization	Dropout
Optimizer	Gradient Descent
Learning rate schedule	None



- What can you try if :
  - You can’t find a good learning rate ?
  - Your training set is too small ?
  - You need a lightning-fast model at runtime ?

Two-Minute Papers

Optimization

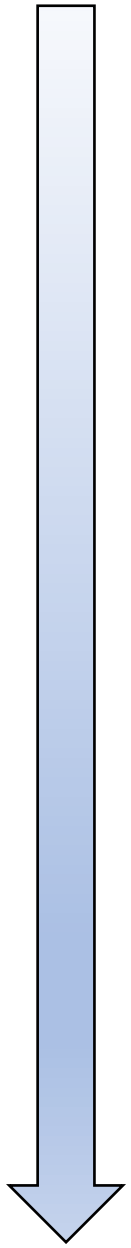
In terms of performance and time

1. Performance

Bias reduction techniques

Variance reduction techniques

List of Errors



- Bayes optimal error
- Human-level error
- Training error
- Dev error
- Test error
- Real world

Avoidable bias

Variance

Depending on where you get the *largest discrepancy*, you will use a different technique to tackle the problem

List of Errors

Medical image classification

	Classification error (%)		
	Scenario A	Scenario B	Scenario C
Human (proxy for Bayes error)	1	1	0.5
	0.7	0.7	
	0.5	0.5	
Training error	5	1	0.7
Development error	6	5	0.8

Bias reduction technique

Variance reduction technique

List of Errors

- Bayes optimal error
- Human-level error
- Training error
- Dev error
- Test error
- Real world

Bias reduction techniques :

- Hyperparameter tuning
- Model tuning
- Optimization algorithm
- decrease regularization

Variance reduction techniques :

- Bigger training set
- Regularization
- Hyperparam. and model tuning

- bigger development test

- change development set
- change cost function

Bias Reduction techniques

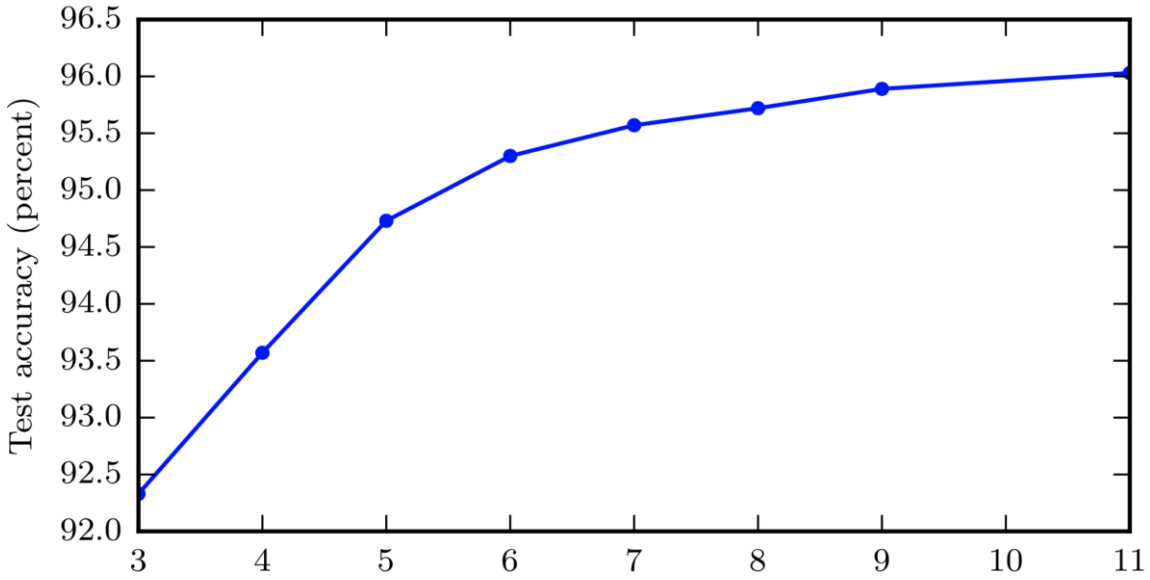
Hyperparameter tuning

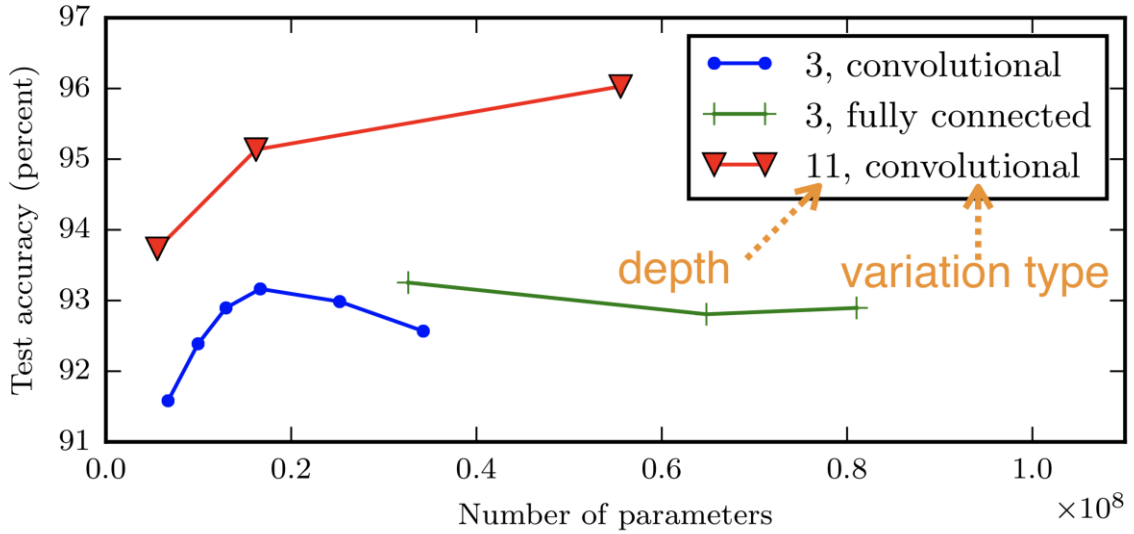
Model tuning

Optimization algorithm

Hyperparameters:
number of hidden layers

- To go deeper helps generalization





better to have many simple layers than few highly complex ones

Hyperparameters:
number of neurons in hidden layers

- Input and output layer number of neurons is determined by the type of input/output of the task
- For hidden layers, common practice was historically to size them to form a funnel, with fewer neurons at each layer
- Now, simply use the same size for all hidden layers
  - *Only one hyperparameter left*

Hyperparameters:
activation function

A generalisation of ReLU is

$$g(z, \alpha) = \max\{0, z\} + \alpha \min\{0, z\}$$

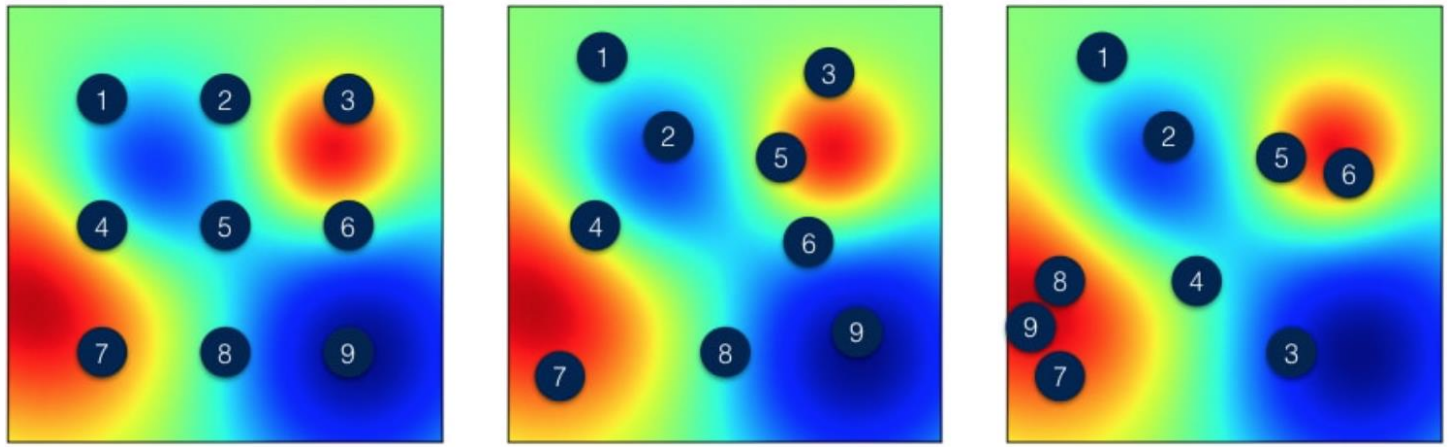
To avoid a null gradient the following are in use

1. Absolute value rectification $\alpha = -1$
2. Leaky ReLU $\alpha = 0.01$
3. Parametric ReLU α learnable
4. Maxout Units $g(z)_i = \max_{j \in S_i} z_j$

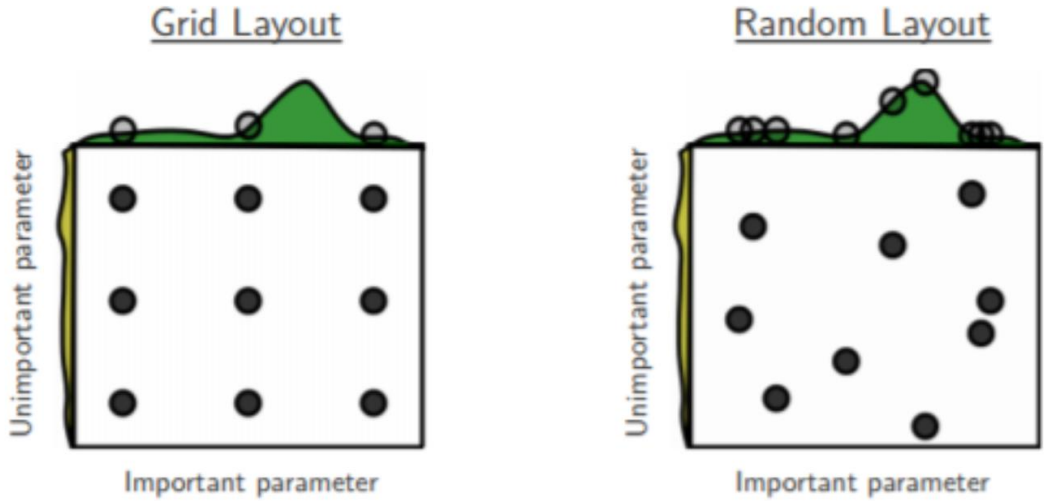
$$\cup_i S_i = [1, \ldots, m]$$

$$S_i \cap S_j = \varnothing \quad i \neq j$$

Hyperparameters:
global search

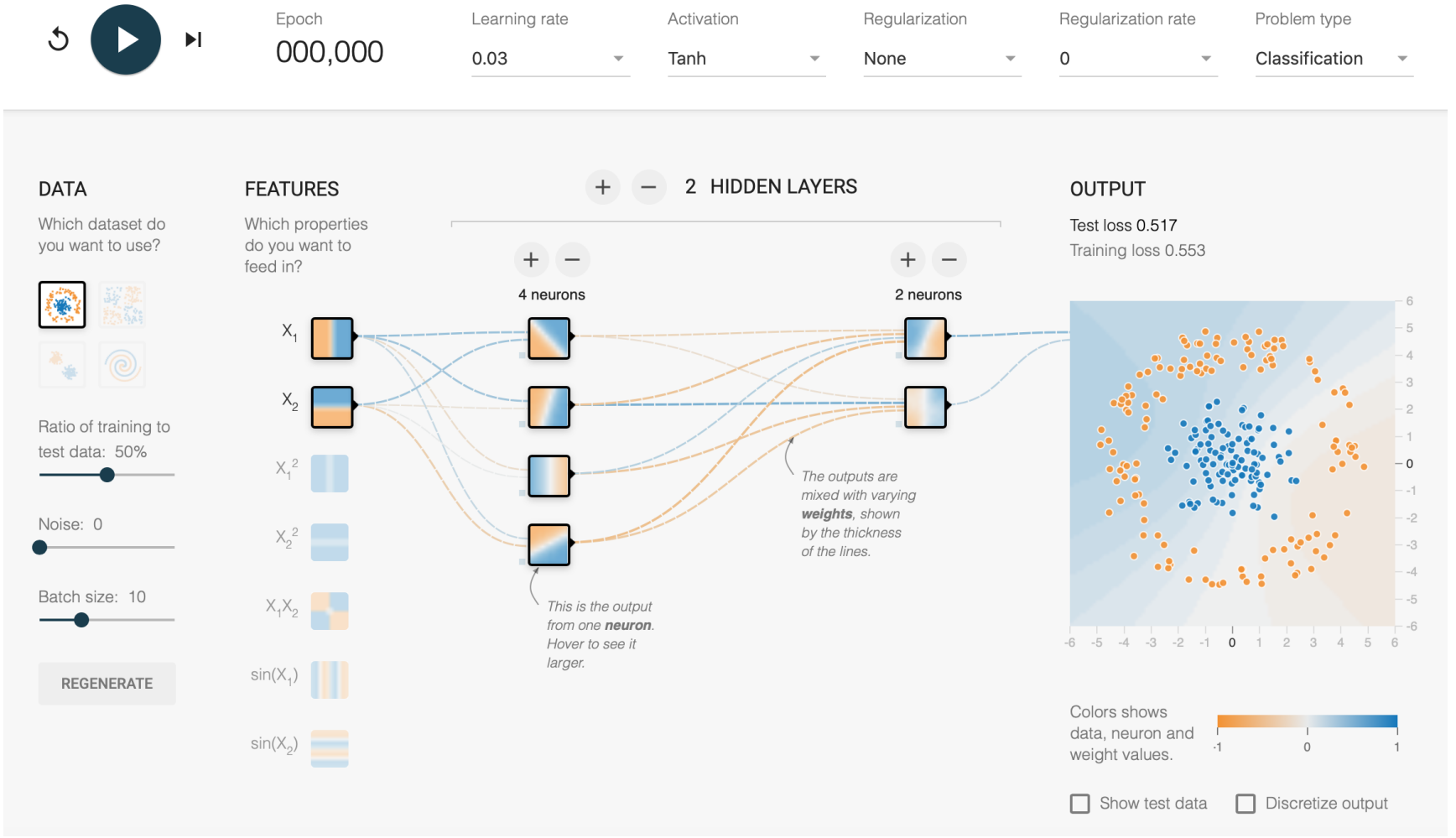


Which one is best ?



Demo

Hyperparameters:
global search

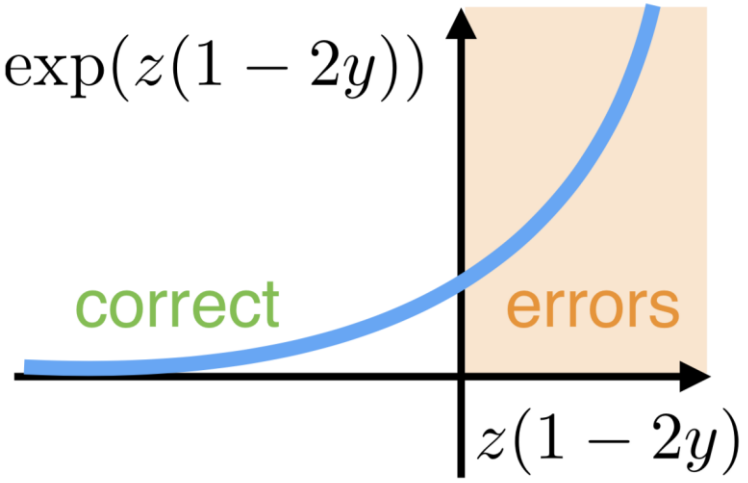


Model: loss function

- chosen such that they have a non-flat region when the answer is incorrect

- *Exponential* or *logarithm* functions help

- Other functions :
  - $L_1(\hat{y}, y) = \sum_{i=0}^{m} \left| y^i - \hat{y}^i \right|$ (L1 loss)
  - $L_2(\hat{y}, y) = \sum_{i=0}^{m} \left(y^i - \hat{y}^i \right)^2$ (L2 loss)
  - …



Model: global architecture

DARTS: Differentiable Architecture Search

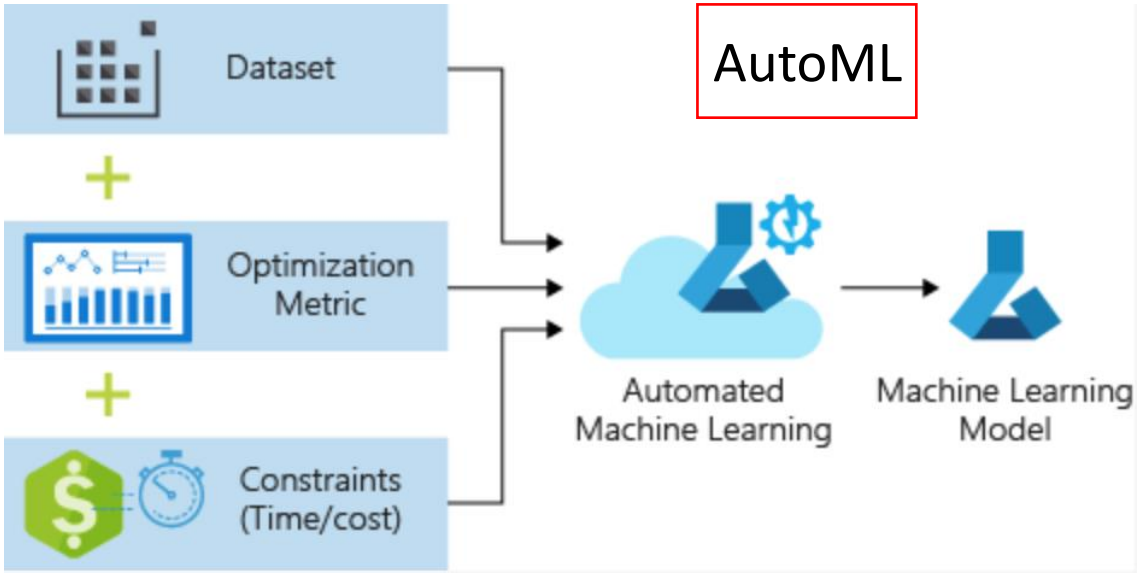
Hanxiao Liu*
CMU
hanxiaol@cs.cmu.com

Karen Simonyan
DeepMind
simonyan@google.com

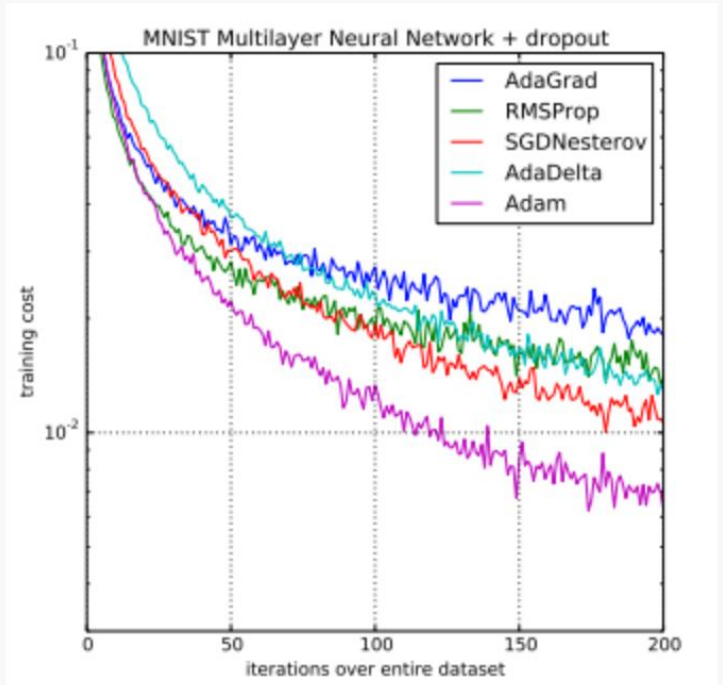
Yiming Yang
CMU
yiming@cs.cmu.edu

Abstract

This paper addresses the scalability challenge of architecture search by formulating the task in a differentiable manner. Unlike conventional approaches of applying evolution or reinforcement learning over a discrete and non-differentiable search space, our method is based on the continuous relaxation of the architecture representation, allowing efficient search of the architecture using gradient descent. Extensive experiments on CIFAR-10, ImageNet, Penn Treebank and WikiText-2 show that our algorithm excels in discovering high-performance convolutional architectures for image classification and recurrent architectures for language modeling, while being orders of magnitude faster than state-of-the-art non-differentiable techniques. Our implementation has been made publicly available to facilitate further research on efficient architecture search algorithms.

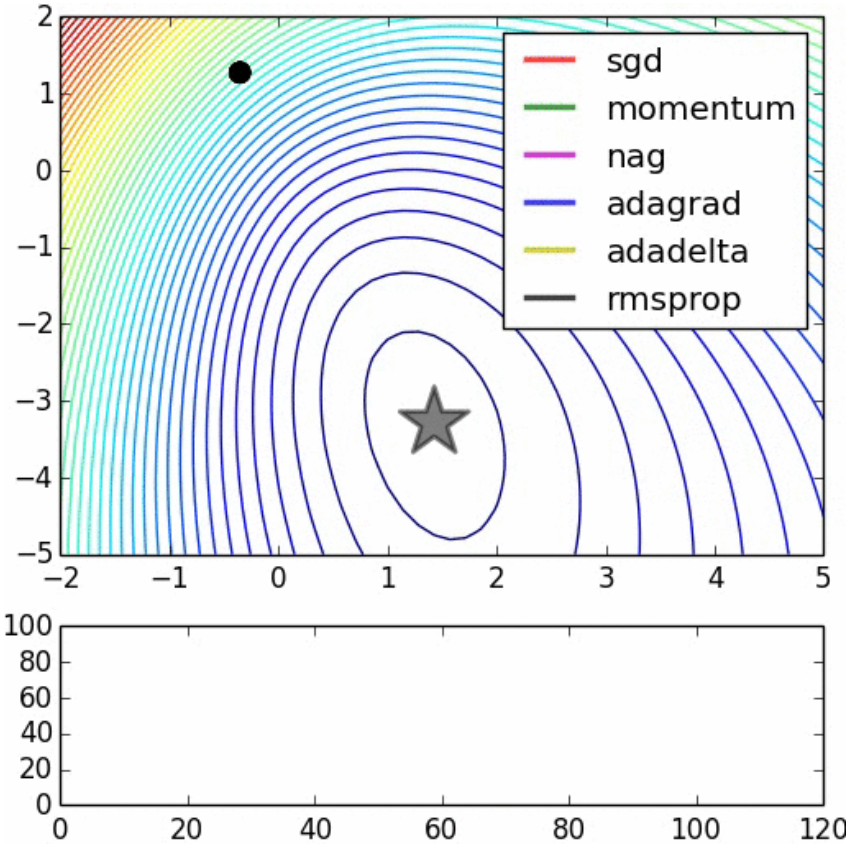


Optimization Algorithm (*Reminder*)



Comparison of Adam to Other Optimization Algorithms Training a Multilayer Perceptron

Taken from Adam: A Method for Stochastic Optimization, 2015.



Two-Minute Papers

Variance Reduction techniques

Bigger training set

Regularization

Regularization

- Different strategies :

  1) Dataset (division, augmentation,…)

  2) Model (dropout, L2 regularization,…)

  3) Training (early stopping)

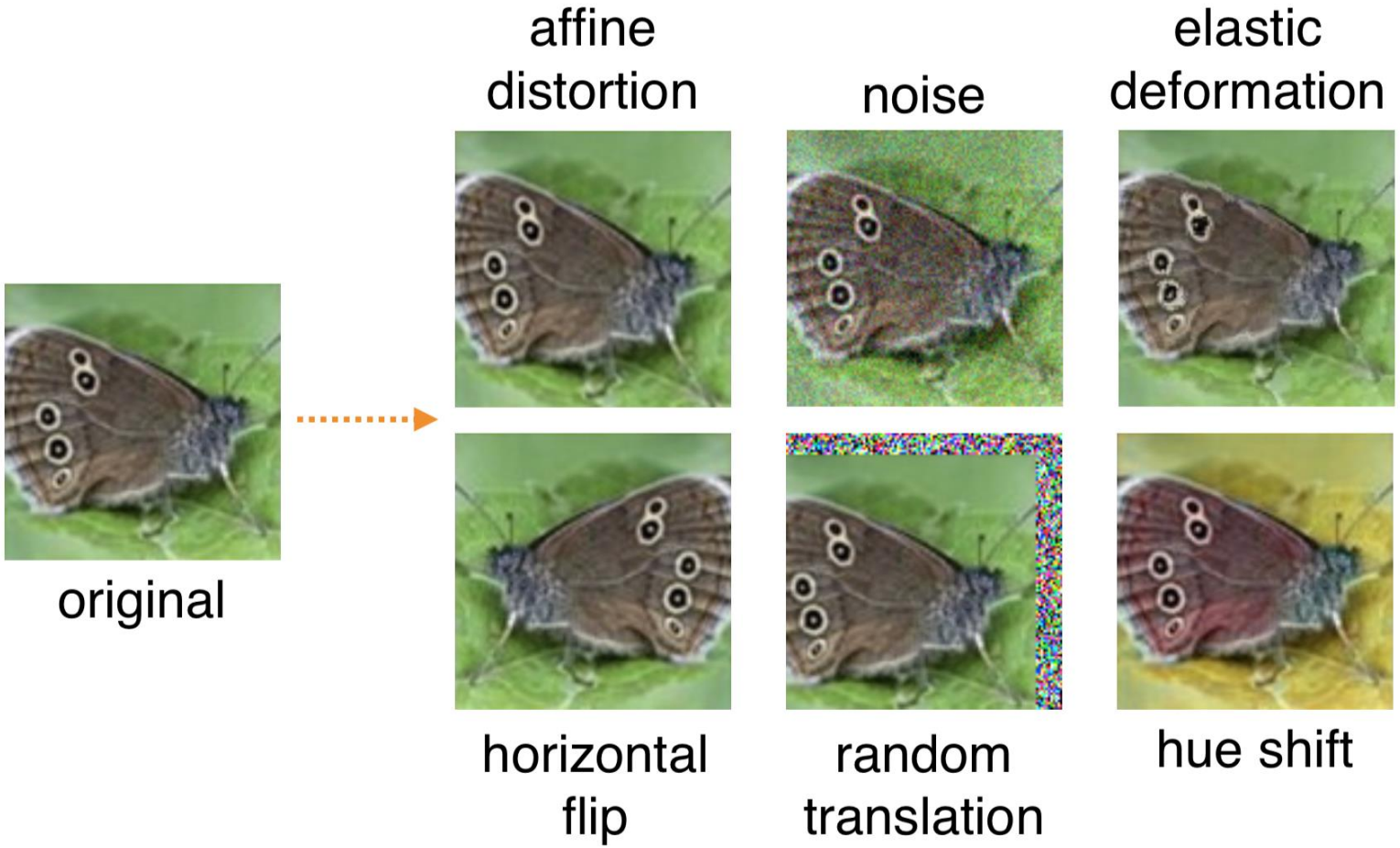
- Use cases : if few data or if model has more than 50 layers (CNN)

1. Regularization (dataset): Division

- Divide the data into a training, validation and test sets
  - Training set to define the optimal predictor
  - Validation set to choose the capacity
  - Test set to evaluate the performance

1. Regularization (dataset): Augmentation

- Apply realistic transformations to data to create new synthetic samples, with same label



- Process also called jittering

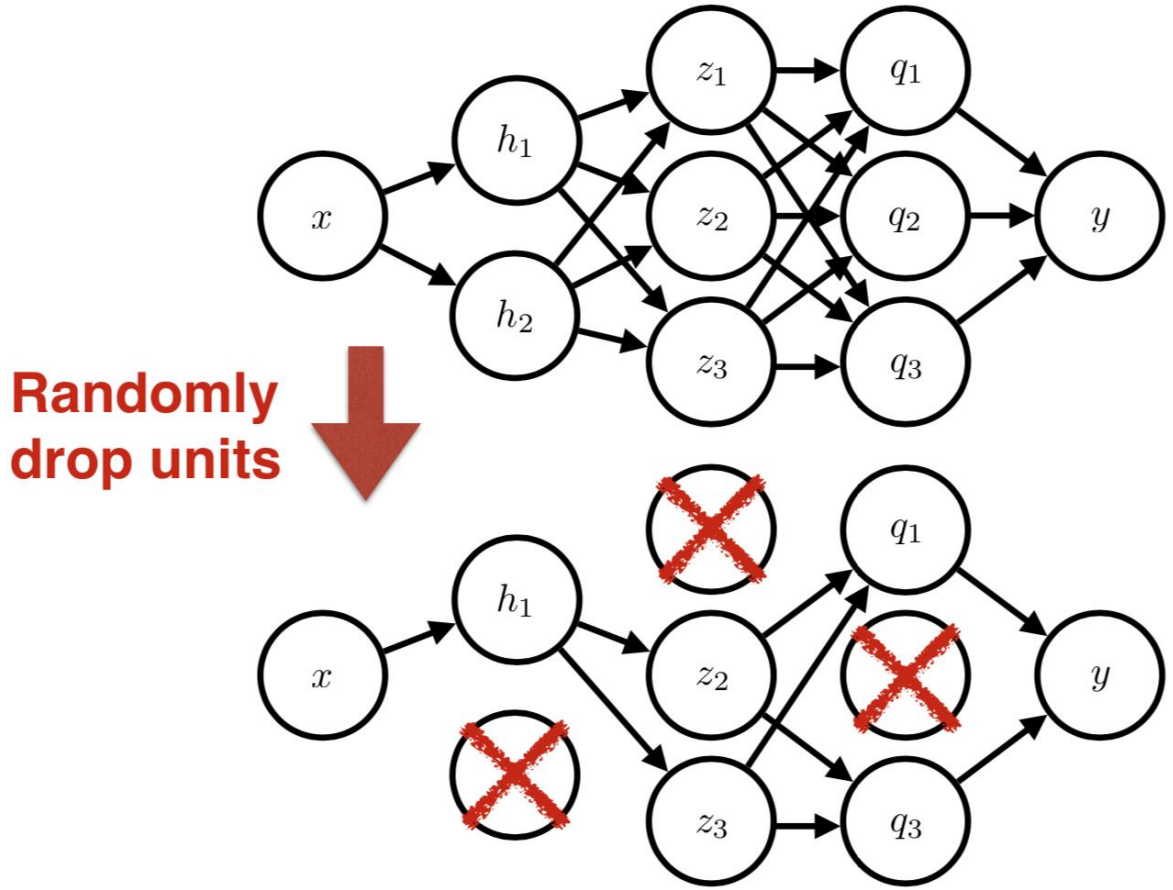
1. Regularization (dataset): Label Smoothing

- Labels might be wrong (human annotation)
- Instead of asking the model to predict 1 for the right class, we ask it to predict 1-ε for the correct class and ε for all the others

$$\text{cross-entropy loss} = (1-\epsilon)ce(i) + \epsilon \sum \frac{ce(j)}{N}$$

2. Regularization (model): Dropout

- Apply it both in forward and backward propagations
- *BUT* use it only in the *training phase !*



Two-Minute Papers

Regularization (model): L2

- Modify the cost function (add soft constraint) :

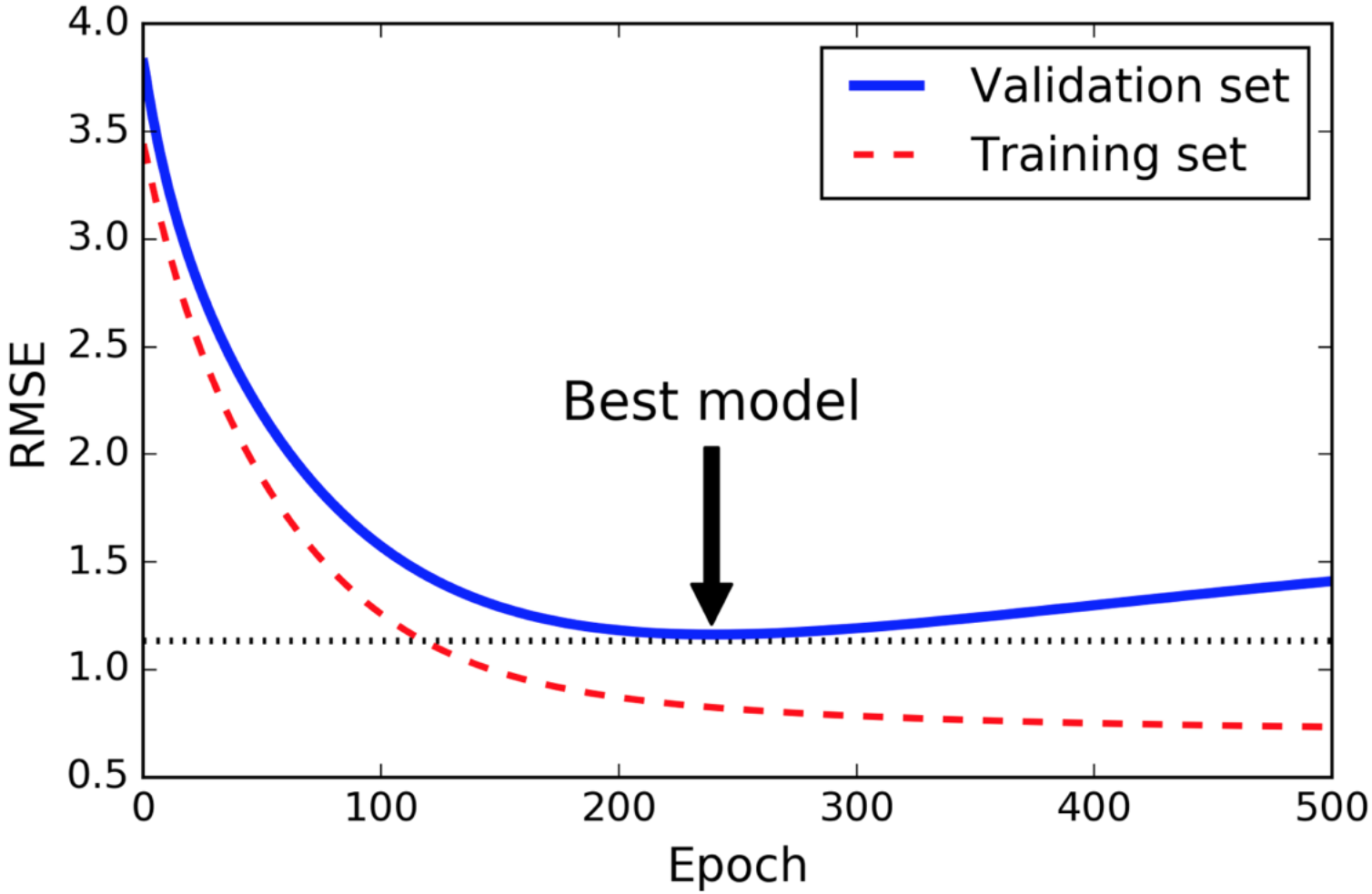
$$J = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)$$

$$J_{regularized} = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$

- λ regulates the complexity/capacity of the predictors. It smoothens the decision boundary → weights pushed to smaller values.

- Typical values : logarithmic scale between 0 and 0.1, such as *0.1, 0.001, 0.0001* (tuned using dev set)
  - If λ is too large, it can "oversmooth", resulting in a model with high bias

3. Regularization (training) : early stopping

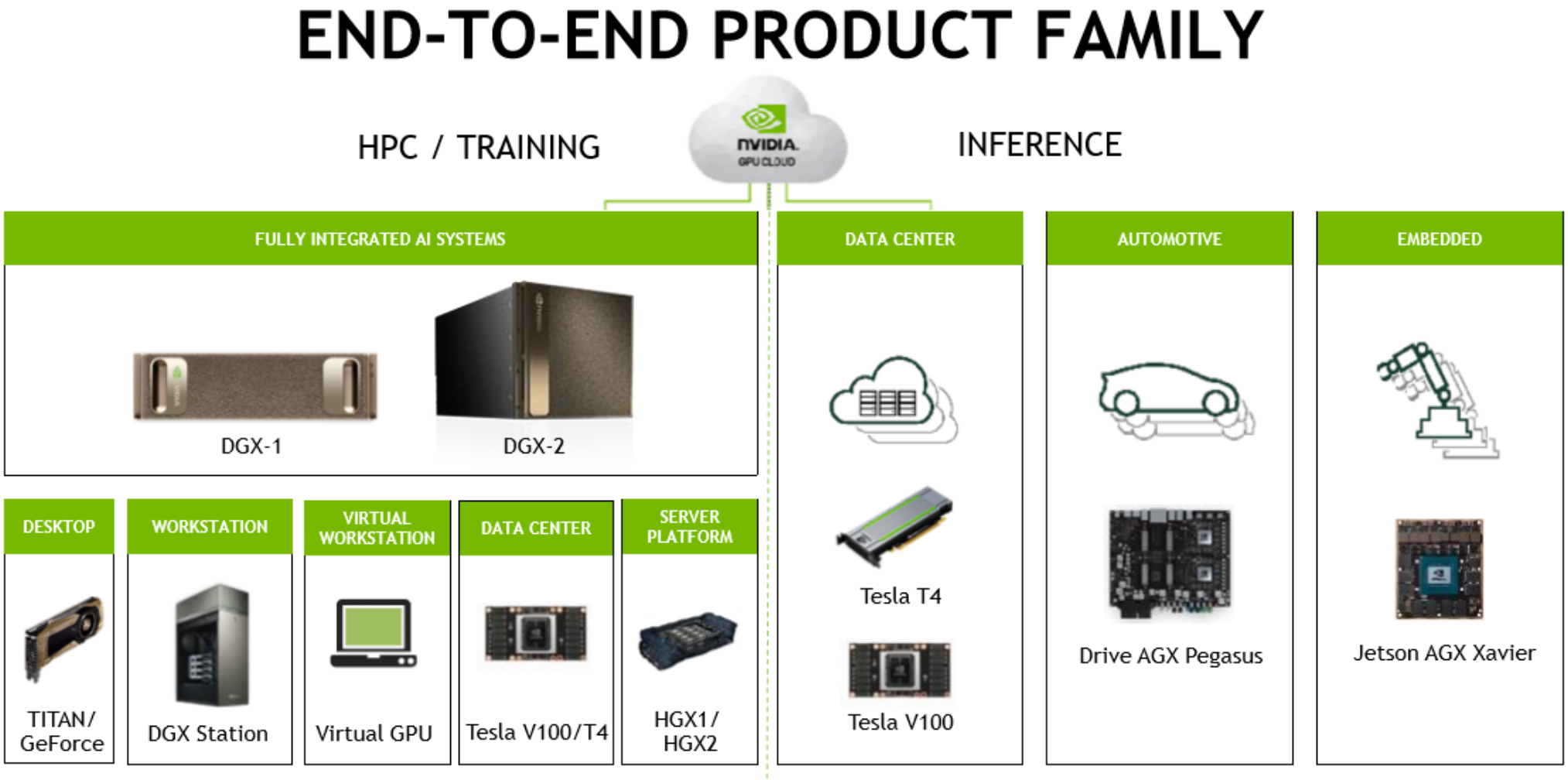


- Stop training before the validation set error start growing

2. Time

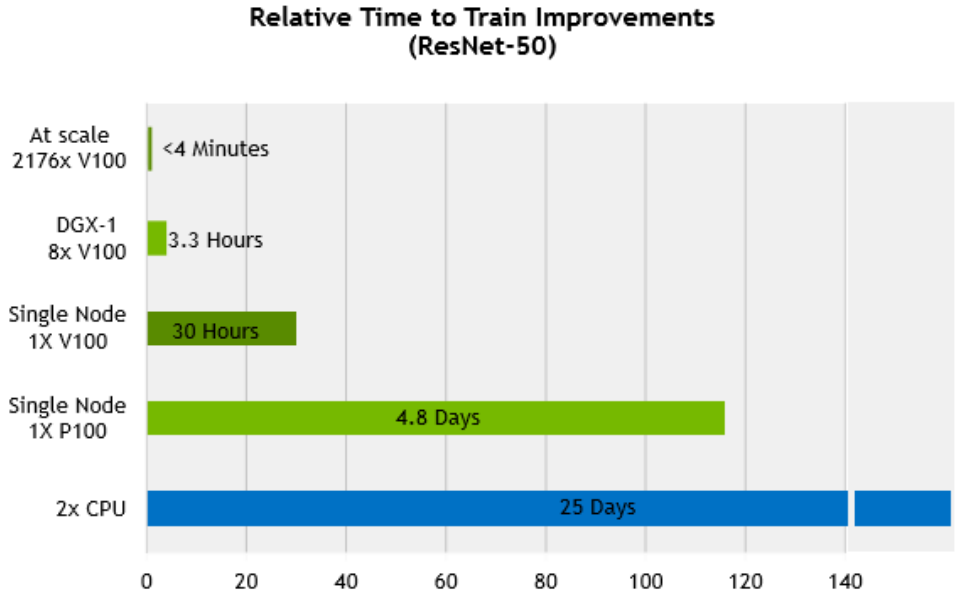
How to improve time consumption when critical to get results

Material Acceleration (GPUs)



Material Acceleration : Example

TESLA PLATFORM ENABLES DRAMATIC REDUCTION IN TIME TO TRAIN



ResNet-50, 90 epochs to solution | CPU Server: dual socket Intel Xeon Gold 6140
Sony 2176x V100 record on https://nnabla.org/paper/imagenet_in_224sec.pdf

Quiz

https://b.socrative.com/login/student/

Room : CONTI6128