Structural Learning of Neural Networks

PhD Defense

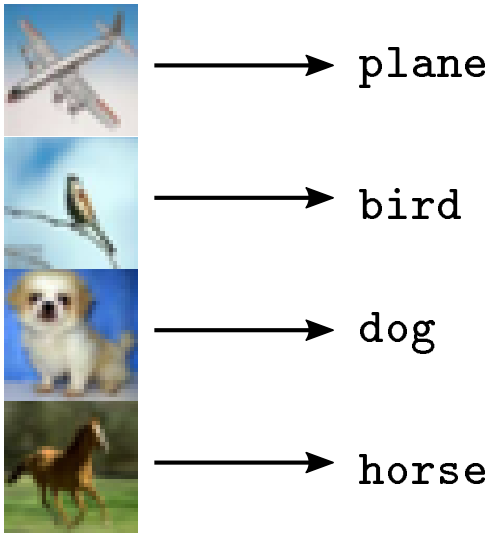
Pierre Wolinski

TAU Team, LRI & Inria

March 6, 2020

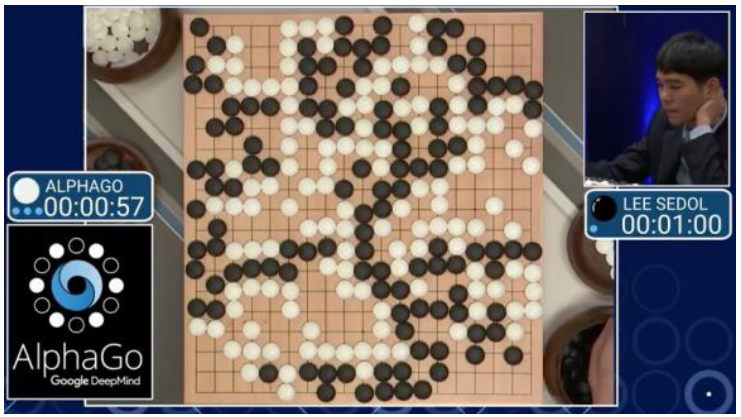
Artificial Intelligence

Artificial Intelligence: solve a given task, like image recognition/classification or winning at the go game...



Classification task

Credits: Alex Krizhevsky



AI "AlphaGo"

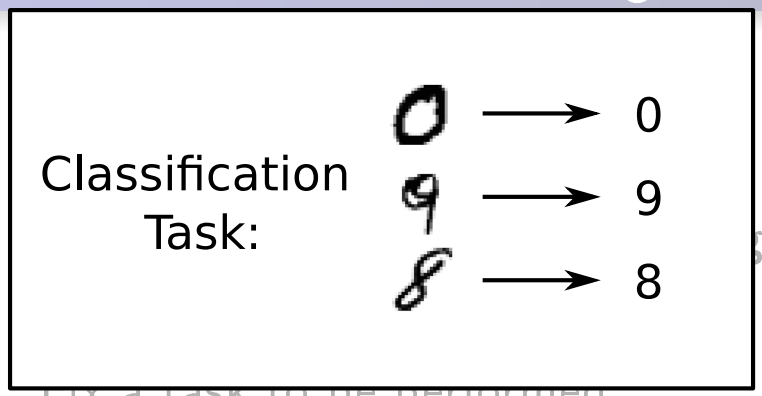
Credits: Google DeepMind

Machine Learning: instead of implementing "human methods" into programs, let programs find the best method to solve given tasks.

How to Do Machine Learning

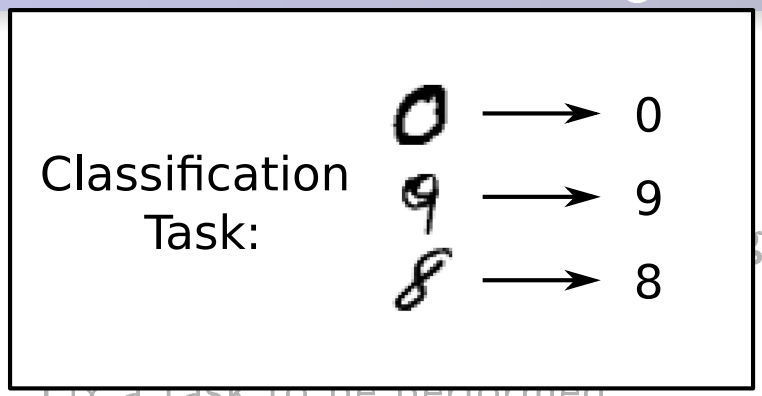
Recipe of Machine Learning	Example
Fix a task to be performed	Task: recognize handwritten digits.
Build a training dataset	Dataset: database of labeled images of handwritten digits.
Build a model to perform the task (usually a program with initially undetermined parameters)	Model: Convolutional Neural Network
Design an algorithm to train the model to perform the task (i.e., which modifies its parameters to improve it)	Training Algorithm: Stochastic Gradient Descent with backpropagation of the gradient

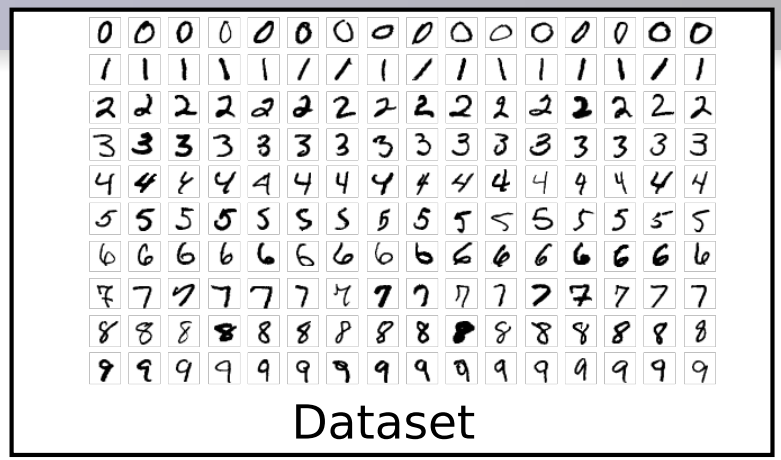
How to Do Machine Learning



	Example
Fix a task to be performed	Task: recognize handwritten digits.
Build a training dataset	Dataset: database of labeled images of handwritten digits.
Build a model to perform the task (usually a program with initially undetermined parameters)	Model: Convolutional Neural Network
Design an algorithm to train the model to perform the task (i.e., which modifies its parameters to improve it)	Training Algorithm: Stochastic Gradient Descent with backpropagation of the gradient

How to Do Machine Learning





Fix a task to be performed

Build a training dataset

Dataset: database of labeled images of handwritten digits.

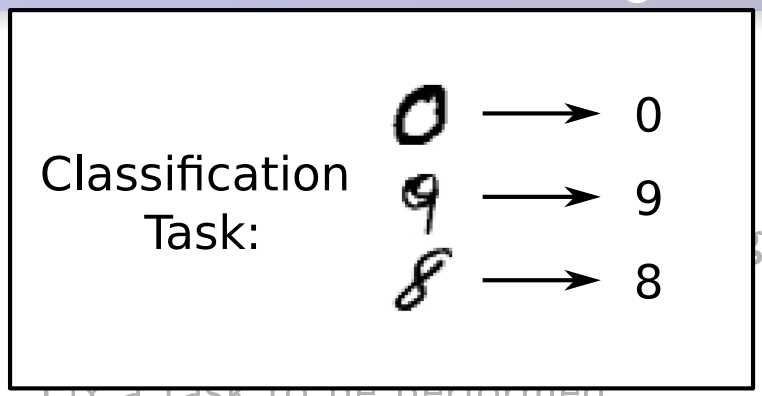
Build a model to perform the task (usually a program with initially undetermined parameters)

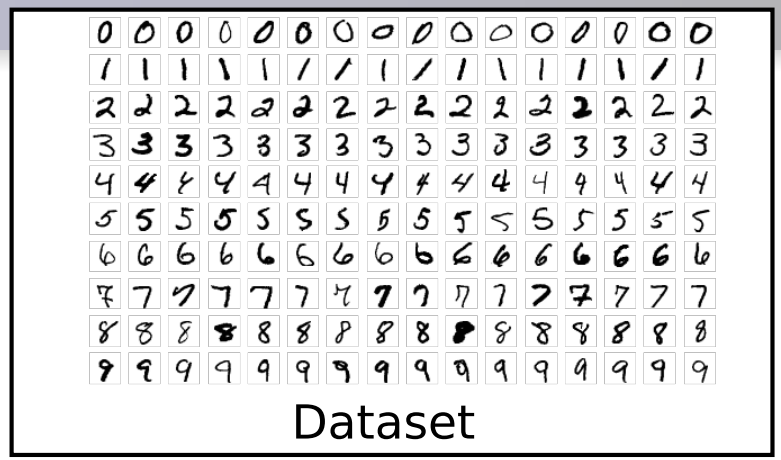
Model: Convolutional Neural Network

Design an algorithm to train the model to perform the task (i.e., which modifies its parameters to improve it)

Training Algorithm: Stochastic Gradient Descent with backpropagation of the gradient

How to Do Machine Learning





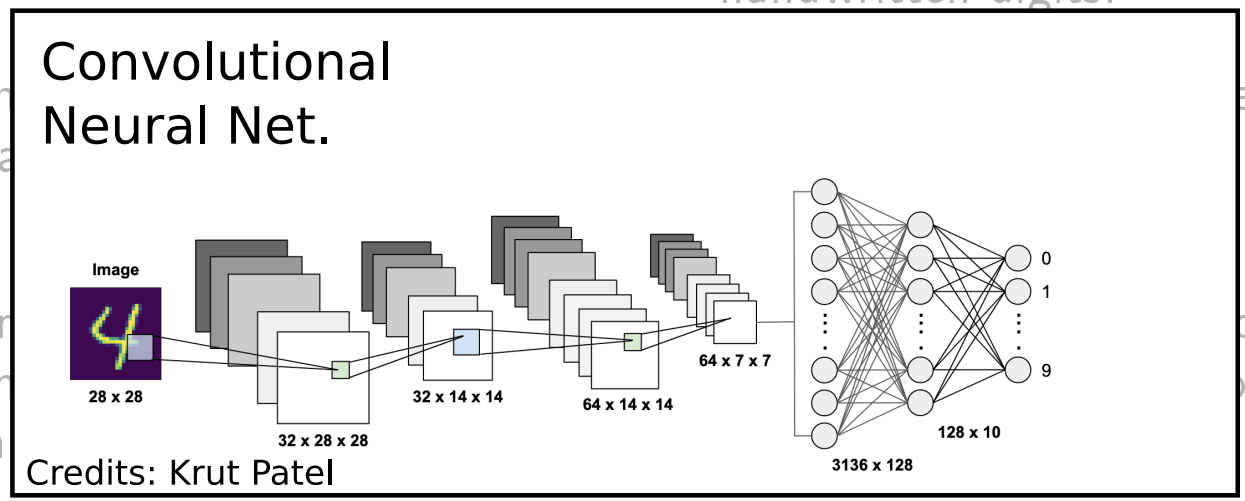
Fix a task to be performed

Build a training dataset

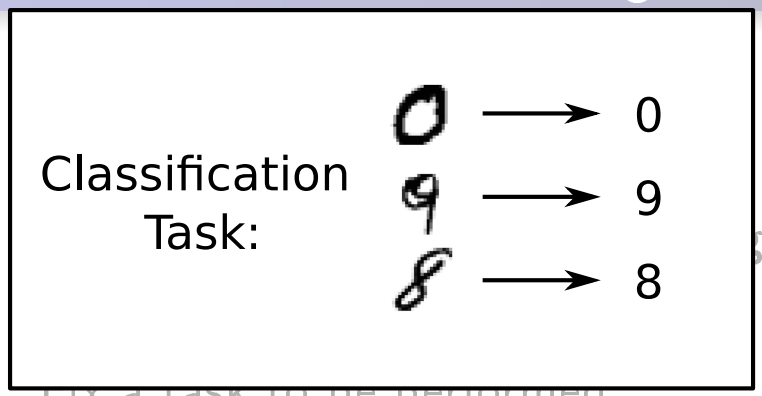
Dataset: database of labeled images of handwritten digits.

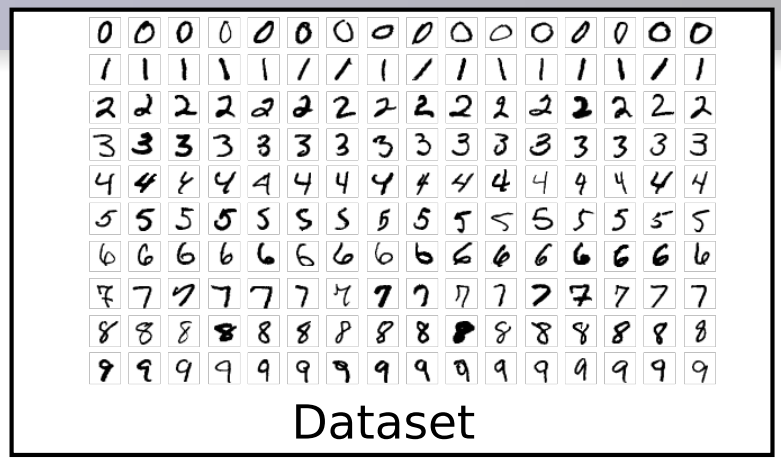
Build a m eural Network (usually a termined

Design ar ochastic Gradi- to perforn pagation of the its param



How to Do Machine Learning





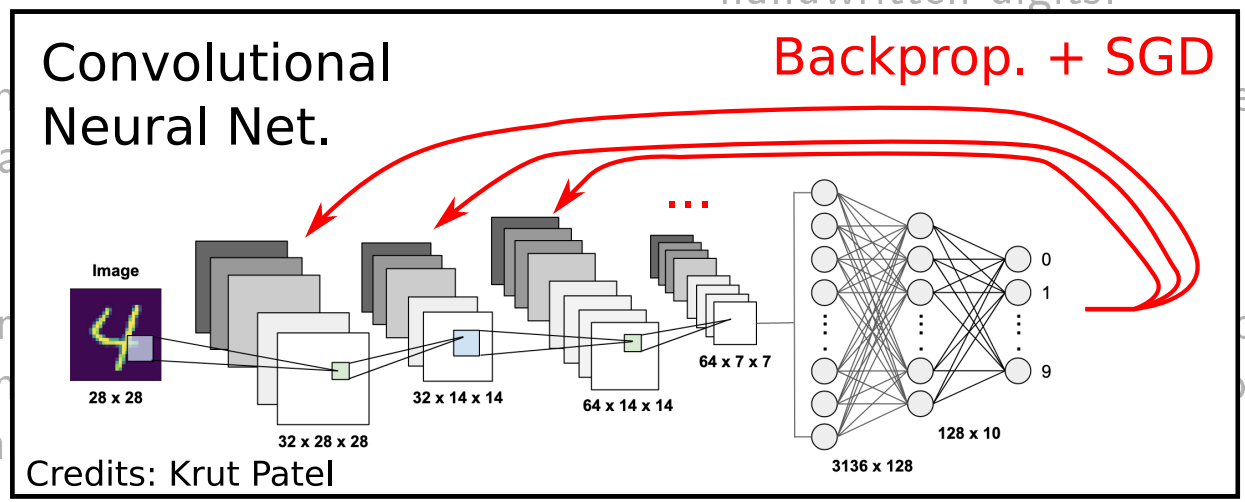
Fix a task to be performed

Build a training dataset

Dataset: database of labeled images of handwritten digits.

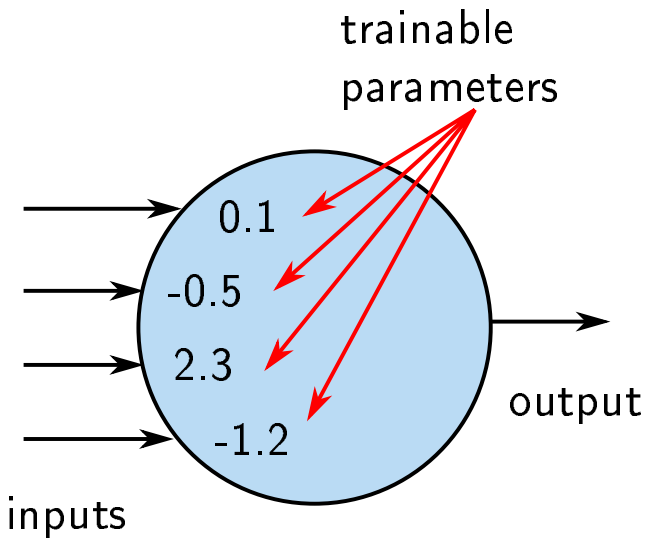
Build a m… eural Network (usually a… termined

Design an… ochastic Gradient to perform… pagation of the its param…

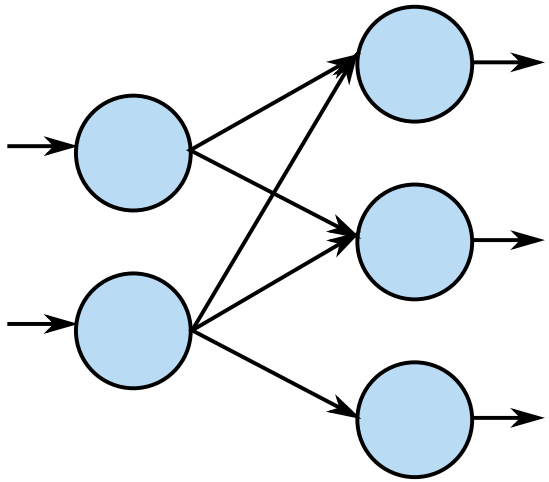


Neural Networks

Approach of Machine Learning: Neural Networks.
Very flexible class of models, trainable with a generic algorithm.



neuron



neural network

The *artificial neuron* is the elementary brick of every neural network, infinite ways to combine them $\Rightarrow$ choice of the model, training hyperparameters.

Problem: Hyperparameters

Hyperparameter search:

- architecture of the neural network;
- learning rate η: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} L$;
- penalty λ, r: $L(\mathbf{w}, \mathcal{D}) = \sum_{i=1}^{n} \|y_i - \mathcal{M}_{\mathbf{w}}(x_i)\|_2^2 + \lambda\, r(\mathbf{w})$.

Main issues:

- efficient architectures are usually very large:
  how to reduce their size? $\Rightarrow$ pruning.
- optimal η and λ depend strongly on the neural network architecture:
  find rules to get rid of η or find default values for η and λ.

Summary

Goal:
Build theoretically well-founded methods to help fixing the hyperparameters η and λ, and the architecture.

1. *Learning with Random Learning Rates*,
   get rid of the learning rate η;
2. *Interpreting a Penalty as the Influence of a Bayesian Prior*,
   bridge between empirical penalties and the Bayesian framework,
   default value for λ;
3. *Asymmetrical Scaling Layer for Stable Network Pruning*,
   reparameterization of neural networks to find a default value for η,
   pruning technique.

Robustness of Training Methods

When training a neural network, we want robustness:

- during training,
- between runs,
- between architectures and datasets,
- robust to hyperparameter change.

Sensitive hyperparameter: learning rate.

Robustness of Training Methods

When training a neural network, we want robustness:

- during training,
- between runs,
- between architectures and datasets,
- robust to hyperparameter change.

Sensitive hyperparameter: learning rate.

Idea of Alrao: instead of searching the *optimal* learning rate, add *diversity* in the learning rates:

- we attribute randomly a learning rate to each neuron, sampled from a distribution spanning many orders of magnitude;
- neurons with a wrong learning rates will be useless and ignored;
- emergence of an efficient sub-network.

Alrao: in the Internal Layers

Setting: classification task performed by a feedforward network:

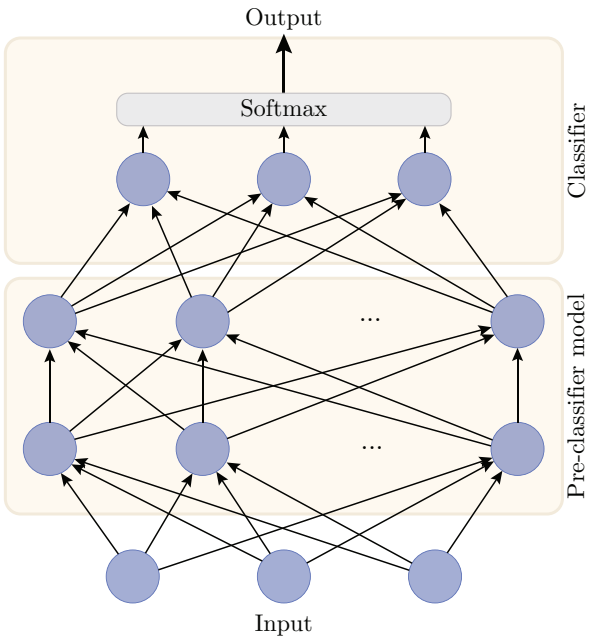
$$\text{Network} = \textit{internal layers} + \text{classification layer}$$

Learning rates sampling: we sample a learning rate η_{neuron} per neuron from the log-uniform distribution on the interval $(\eta_{\min}, \eta_{\max})$:

$$\log \eta \sim \mathcal{U}(\eta_{\min}, \eta_{\max}).$$

Update rule: for every neuron, the update of its parameters θ_{neuron} is:

$$\theta_{\text{neuron}} \leftarrow \theta_{\text{neuron}} - \eta_{\text{neuron}} \nabla_{\theta_{\text{neuron}}} L(\theta)$$



Alrao: in the Output Layer

Issue: in the classification layer, each neuron corresponds to a class. All classes should be trained equivalently.

How to use Alrao in the output layer?

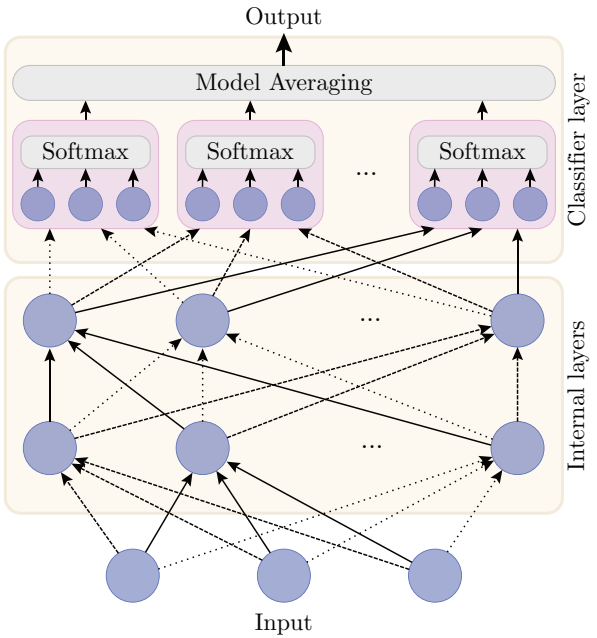
Modified architecture: we replace the single classification layer by a convex combination C^{Mix} of classifier layers $(C_i)_i$:

$$C^{\text{Mix}} = \sum a_i C_i,$$

each one with a learning rate sampled in $(\eta_{\text{min}}, \eta_{\text{max}})$.

Training method:

- each classifier is trained separately;
- averaging weights $(a_i)_i$ are updated with the *Switch* Bayesian averaging method.



How Does Alrao Compare to SGD and Adam?

Experiments: comparison between Alrao, SGD with optimized η, and Adam.

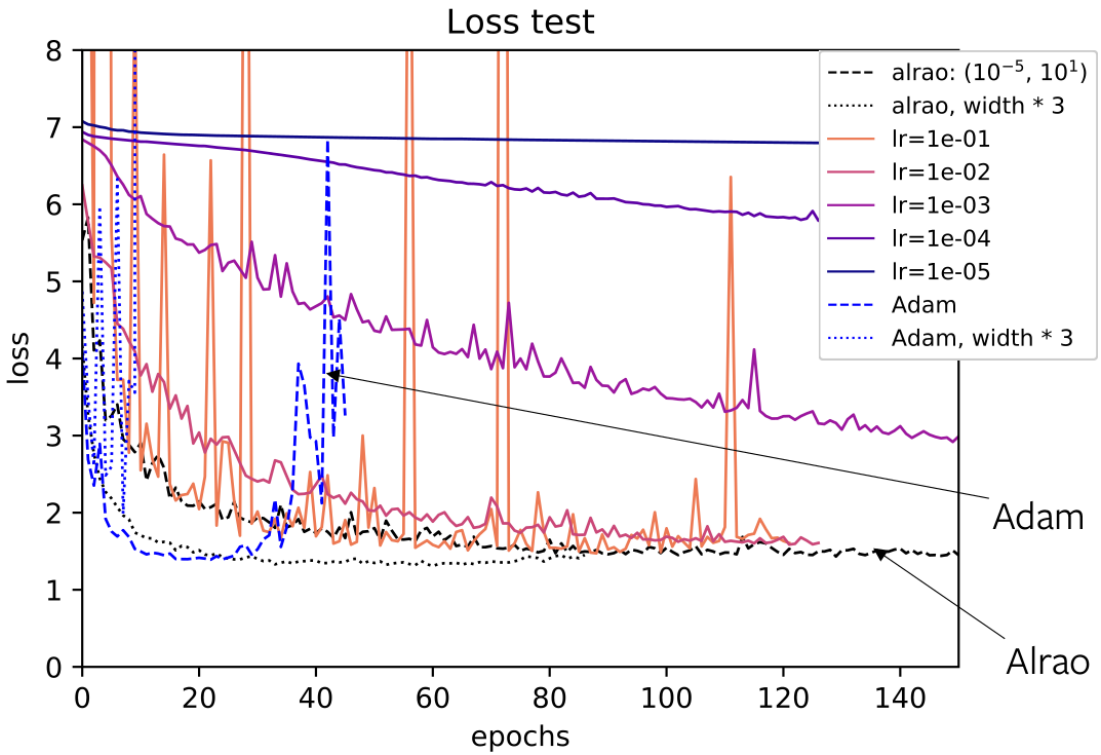
Tasks and architectures:

- CIFAR-10: MobileNet, GoogLeNet, VGG19;
- ImageNet: AlexNet, DenseNet121, ResNet50;
- PennTreeBank: LSTM;
- Reinforcement Learning: Pendulum, Lunar Lander.

Observations:

- Alrao performs almost as well as SGD with its best learning rate, *without grid search*;
- not a single run of Alrao has failed.

How Does Alrao Compare to SGD and Adam?



ResNet50 trained on ImageNet.

Setups:

- *Alrao* with a large interval $(10^{-5}, 10^{1})$;
- *SGD* with $\eta \in \{10^{-5}, \cdots, 10^{1}\}$;
- *Adam* with its standard hyperparams.

Is the Learning Rate Interval of Alrao Important?

We claim to remove the learning rate η hyperparameter... but we replace it by two hyperparameters: η_{min} and η_{max}!

Is the Learning Rate Interval of Alrao Important?

We claim to remove the learning rate η hyperparameter... but we replace it by two hyperparameters: η_{min} and η_{max}!

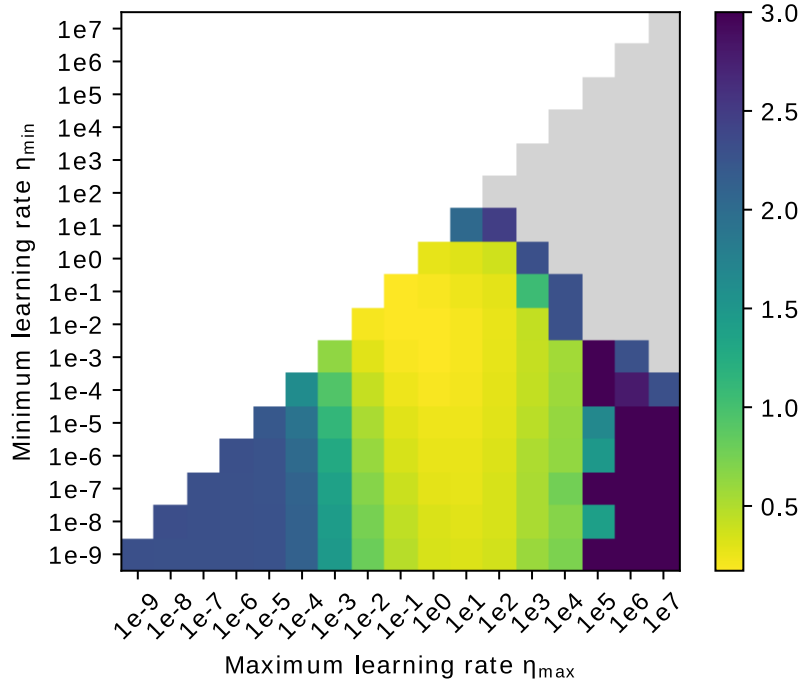
Typically, the interval of all learning rates you would have used in a grid search will work.

Is the Learning Rate Interval of Alrao Important?

We claim to remove the learning rate η hyperparameter... but we replace it by two hyperparameters: η_{min} and η_{max}!

Typically, the interval of all learning rates you would have used in a grid search will work.

If the interval contains the optimal learning rate and is not absurdly large, Alrao will perform roughly as well as the optimal learning rate.



GoogLeNet model trained with Alrao on CIFAR10, for η_{min} and η_{max} in $(10^{-9}, \cdots, 10^{7})$.

Back to the Hypothesis: Are There Dead Neurons?

First intuition: neurons with too large or too small learning rate will be ignored. Is that true?

More generally:

Which are the learning rates of the least useful neurons?

Back to the Hypothesis: Are There Dead Neurons?

First intuition: neurons with too large or too small learning rate will be ignored. Is that true?

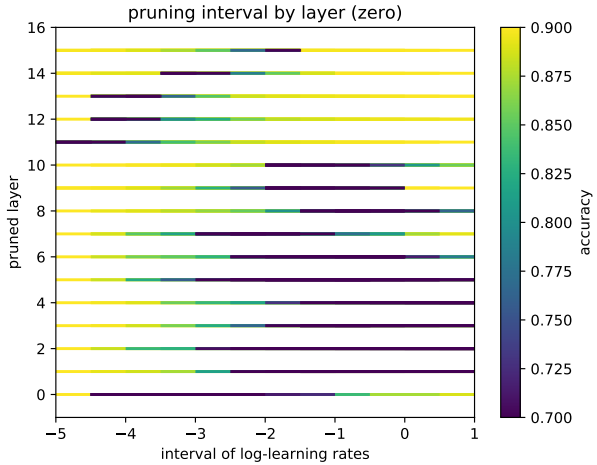
More generally:

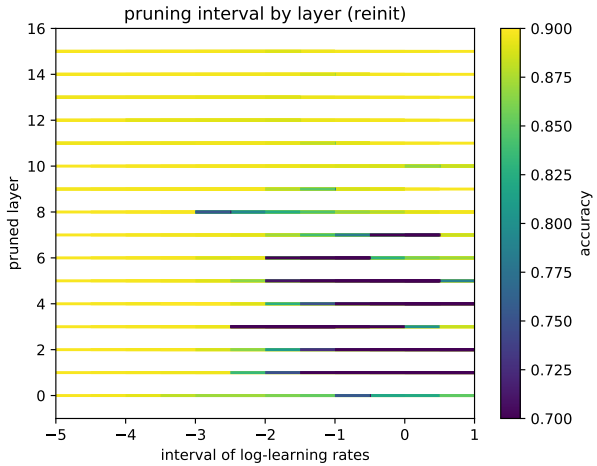
Which are the learning rates of the least useful neurons?

Results:

- the "wrong" learning rates depend on the layer;
- in some layers, many neurons can be reinitialized with little accuracy loss.

In a deep neural network as VGG, there is no "wrong" learning rate. However, neurons with a small learning rate can possibly be reinitialized.





Limitations

Alrao increases the number of parameters in the model.
On classification tasks with many classes, the last layer might represent a large part of the network's weights. Duplicating the last layer can be computationally expensive:

- on CIFAR10: Only +5% on the number of weights;
- on ImageNet: Between +50% and +100%.

Alrao would not be efficient for word-level NLP.

Performance.
In our experiments, Alrao always performed close to SGD with its optimal learning rate.
Still, its performance is sometimes slightly below.

Conclusion

Summary:

- Alrao is a method for robust optimization without grid-search on η;
- not a single run of Alrao has failed;
- it is designed especially for deep learning models (and would not make sense in other settings);
- instead of searching optimal hyperparameters, adding diversity in the network.

Perspectives:

- Lottery Ticket Hypothesis (Frankle, 2018): towards more diversity in neural networks;
- implement usual tricks: learning rate schedule, momentum, regularization;
- application to AutoML: testing new architectures easily.

Introduction

Context:

- train a model of parameters $\mathbf{w} \in \mathbb{R}^N$ on dataset $(\mathbf{x}, \mathbf{y})$;
- inputs $\mathbf{x} \Rightarrow$ distribution $p_{\mathbf{w}}(\cdot|\mathbf{x})$ on the targets $\mathbf{y}$;
- fit the data $\Rightarrow$ minimize the negative log-likelihood.

$$\text{usual:} \quad L_{\text{usual}}(\mathbf{w}) = -\ln p_{\mathbf{w}}(\mathbf{y}|\mathbf{x}) + r(\mathbf{w})$$

Introduction

Context:

- train a model of parameters $\mathbf{w} \in \mathbb{R}^N$ on dataset $(\mathbf{x}, \mathbf{y})$;
- inputs $\mathbf{x} \Rightarrow$ distribution $p_{\mathbf{w}}(\cdot|\mathbf{x})$ on the targets $\mathbf{y}$;
- fit the data $\Rightarrow$ minimize the negative log-likelihood.

$$\begin{array}{rrcccc} \text{usual:} & L_{\text{usual}}(\mathbf{w}) & = & -\ln p_{\mathbf{w}}(\mathbf{y}|\mathbf{x}) & + & r(\mathbf{w}) \\ \text{Variational Inference:} & L_{\text{VI}}(\beta) & = & \mathbb{E}_{\mathbf{w}\sim\beta}\Big[-\ln p_{\mathbf{w}}(\mathbf{y}|\mathbf{x})\Big] & + & \mathrm{KL}(\beta\|\alpha) \end{array}$$

Variational Inference:

- Bayesian method: optimize β instead of $\mathbf{w}$, prior α;
- β contains more information than $\mathbf{w}$: uncertainty...
- case $\beta = \delta_{\mathbf{w}}$: $L_{\text{VI}}(\beta) = -\ln p_{\mathbf{w}}(\mathbf{y}|\mathbf{x}) + \ln(\alpha(\mathbf{w})) \Rightarrow \alpha \propto \exp(r)$.

Link between a Penalty and a Bayesian Prior

We assume that $\mathbf{w} \sim \beta$. We define, for a penalty r and a prior α:

$$\begin{aligned} L(\beta) &= -\mathbb{E}_{\mathbf{w}\sim\beta} \ln p_{\mathbf{w}}(\mathbf{y}|\mathbf{x}) + r(\beta) \\ L_{\mathrm{VI}}(\beta) &= -\mathbb{E}_{\mathbf{w}\sim\beta} \ln p_{\mathbf{w}}(\mathbf{y}|\mathbf{x}) + \mathrm{KL}(\beta\|\alpha). \end{aligned}$$

Questions

Given r, does there exist a prior α such that for all β, $r(\beta) = \mathrm{KL}(\beta\|\alpha)$?

If so, is there a systematic way to compute α from r?

Link between a Penalty and a Bayesian Prior

We assume that $\mathbf{w} \sim \beta$. We define, for a penalty r and a prior α:

$$\begin{aligned} L(\beta) &= -\mathbb{E}_{\mathbf{w}\sim\beta} \ln p_{\mathbf{w}}(\mathbf{y}|\mathbf{x}) + r(\beta) \\ L_{\mathrm{VI}}(\beta) &= -\mathbb{E}_{\mathbf{w}\sim\beta} \ln p_{\mathbf{w}}(\mathbf{y}|\mathbf{x}) + \mathrm{KL}(\beta\|\alpha). \end{aligned}$$

Questions

Given r, does there exist a prior α such that for all β, $r(\beta) = \mathrm{KL}(\beta\|\alpha)$?

If so, is there a systematic way to compute α from r?

Theorem 1 (informal, c.f. Theorem 1)

We provide an explicit condition $(\star)$ on r such that:

r fulfills $(\star) \Leftrightarrow$ there exists a unique prior α corresponding to r

Moreover, we have a formula for α.

Main Theorem Applied to a Particular Case

Family $\mathcal{B}$ of variational posteriors. We look for: $\beta^* = \arg\min_{\beta \in \mathcal{B}} L_{\mathrm{VI}}(\beta)$.

Example: posteriors parameterized by their mean: $\mathcal{B} = \{\beta_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in \mathbb{R}^N\}$.

Goal: given a function r, find a probability distribution α such that:

$$\exists K \in \mathbb{R} : \forall \beta \in \mathcal{B}, \qquad r(\beta) = \mathrm{KL}(\beta \| \alpha) + K. \tag{1}$$

Main Theorem Applied to a Particular Case

Family $\mathcal{B}$ of variational posteriors. We look for: $\beta^* = \arg\min_{\beta \in \mathcal{B}} L_{\mathrm{VI}}(\beta)$.

Example: posteriors parameterized by their mean: $\mathcal{B} = \{\beta_{\boldsymbol{\mu}} : \boldsymbol{\mu} \in \mathbb{R}^N\}$.

Goal: given a function r, find a probability distribution α such that:

$$\exists K \in \mathbb{R} : \forall \beta \in \mathcal{B}, \qquad r(\beta) = \mathrm{KL}(\beta \| \alpha) + K. \tag{1}$$

Corollary 2 (c.f. Corollary 6)

Under technical conditions over β_0 and r, there exists a unique prior α solution of Equation (1):

$$\alpha(\mathbf{w}) = \frac{1}{\kappa} \exp\left(-\mathrm{Ent}(\beta_0) - \mathcal{F}^{-1}\left[\frac{\mathcal{F}r}{\mathcal{F}\check{\beta}_0}\right](\mathbf{w})\right),$$

where $\kappa > 0$ is a normalization constant, $\mathrm{Ent}(\beta_0)$ is the entropy of β_0, $\check{\beta}_0(\mathbf{w}) = \beta_0(-\mathbf{w})$ and $\mathcal{F}$ is the Fourier transform.

Applications

Gaussian distributions with $\mathcal{L}^2$ penalty:

- $\beta = \beta_{\mu,\sigma^2} = \mathcal{N}(\mu, \sigma^2)$;
- $r_{f_1,f_2}(\beta_{\mu,\sigma^2}) = f_1(\sigma^2) + f_2(\sigma^2)\mu^2$.

Corollary 3 (informal, c.f. Corollary 7)

If the penalty r_{f_1,f_2} above corresponds to a prior α, then $\alpha \sim \mathcal{N}(0, \sigma_0^2)$ and $r_{f_1,f_2}(\mu, \sigma^2) = r_{\sigma_0^2}(\mu, \sigma^2)$.

Deterministic distributions:

- $\beta = \beta_{\mu} = \delta_{\mu}$;
- we recover the L_{MAP} loss, i.e., $\alpha \propto \exp(r)$.

How to Fix the Penalty Factor λ?

We can interpret penalties in the same way: probability distributions.

Goal: fix the *penalty factor* λ in the penalized loss: $L(\beta) = \ell(\beta) + \lambda r(\beta)$.

How to Fix the Penalty Factor λ?

We can interpret penalties in the same way: probability distributions.

Goal: fix the *penalty factor* λ in the penalized loss: $L(\beta) = \ell(\beta) + \lambda r(\beta)$.

Idea:

- use Theorem 1 on λr to get the corresponding prior α_λ;
- the prior α_λ should be a reasonable initialization rule;
- then, apply Glorot's condition on α_λ: $\begin{cases} \mathbb{E}_{w\sim\alpha_\lambda}[w] &= 0 \\ \mathbb{E}_{w\sim\alpha_\lambda}[w^2] &= 1/P \end{cases}$, where P is the number of parameters in the considered neuron;
- finally, reflect this condition on λ, which fixes its value.

Gaussian distributions with $\mathcal{L}^2$ penalty: we get $\alpha \sim \mathcal{N}(0, 1/P)$.

Conclusion

Summary of the theoretical part:

- Theorem 1: link between *empirical penalties* and *Bayesian priors* when learning a distribution over the parameters $\mathbf{w}$;
- Bayesian interpretation: unified interpretation of the penalty.

Experimental results:

- our theoretical penalty factor is overestimated by a factor 10–100;
- this overestimation is stable across tested architectures and penalties.

Perspectives:

- is the Bayesian posterior overcautious?
- investigate the issues with the combination Bayes/Glorot;
- taking into account the global structure of the considered network.

Pruning Arbitrarily Large Neural Networks

Efficient neural networks usually very large: how to reduce their size?
$\Rightarrow$ Train and prune a very large neural network.

Pruning Arbitrarily Large Neural Networks

Efficient neural networks usually very large: how to reduce their size?
$\Rightarrow$ Train and prune a very large neural network.

Problems to solve:

- training: is the training algorithm robust to width change? $(\rightarrow \infty)$
- pruning: is the resulting network stable between runs, provided it was initially wide enough?

Pruning Arbitrarily Large Neural Networks

Efficient neural networks usually very large: how to reduce their size?
$\Rightarrow$ Train and prune a very large neural network.

Problems to solve:

- training: is the training algorithm robust to width change? $(\to \infty)$
- pruning: is the resulting network stable between runs, provided it was initially wide enough?

Let $\mathbf{w}$ be the tensor of weights of a layer:

- **ScaLa**: training procedure.
  Instead of training directly $\mathbf{w}$, we train $\tilde{\mathbf{w}}$, defined by $\mathbf{w} = S\tilde{\mathbf{w}}$, where S is typically $\mathrm{Id}/\sqrt{n_{\mathrm{in}}}$, changes the dynamics of SGD;
- **ScaLP**: training and pruning procedure.
  Combination of ScaLa, e.g., with S typically proportional to $\mathrm{Diag}(1^{-1}, 2^{-1}, 3^{-1}, \cdots)$, and pruning.

Results

Discussion:

- ScaLa and ScaLP worked with a learning rate $\eta \approx 1$;
- as the width of a layer tends to infinity, the behavior of ScaLa does not change, while the standard SGD tends to diverge after one gradient step;
- ScaLP leads to the same network structure, regardless of the initial width of the layers (provided that they are wide enough).

Perspectives:

- link with the Neural Tangent Kernels (NTK);
- adapt ScaLP to remove connections between entire layers in inception-like or U-Net-like networks.

Conclusion

Developing theoretical considerations:

- diversity in neural networks;
- Glorot's heuristic for weight initialization;
- Bayesian interpretations.

Further works:

- application of Alrao to AutoML;
- asymmetrical scaling layers $\Rightarrow$ link with Neural Tangent Kernels (NTK);
- stronger theorem for the penalty–prior equivalence.

Questions:

- limitations of Glorot's heuristic: how to take into account the architecture?
- random weights: in neural networks, what is the role of diversity?

Penalty–Bayes – Variational Inference: Framework

We train a model $\mathcal{M}_{\mathbf{w}}$ parameterized with $\mathbf{w} \in \mathbb{R}^N$ on a dataset $\mathcal{D} = \{(x_i, y_i)_i\}$.

User-defined objects:

- family of variational posteriors on $\mathbf{w}$: $\mathcal{B} = \{\beta_{\mathbf{u}} : \mathbf{u} \in \mathcal{U}\}$;
- Bayesian prior on $\mathbf{w}$: distribution α.

Details:

- given an input x, $\mathcal{M}_{\mathbf{w}}$ outputs $p_{\mathbf{w}}(\cdot|x)$;
- vector $\mathbf{w} \in \mathbb{R}^N$ randomly drawn from a distribution $\beta_{\mathbf{u}} \in \mathcal{B}$;
- instead of learning $\mathbf{w}$, we learn $\beta_{\mathbf{u}}$, i.e. $\mathbf{u}$;
- loss: $L(\mathbf{u}) = -\mathbb{E}_{\mathbf{w}\sim\beta_{\mathbf{u}}} \ln p_{\mathbf{w}}((y_i)_i|(x_i)_i) + \mathrm{KL}(\beta_{\mathbf{u}}\|\alpha)$;
- we call $\beta_{\mathbf{u}^*} \in \mathcal{B}$ the *variational posterior*, where $\mathbf{u}^* = \arg\min_{\mathbf{v}\in\mathcal{U}} L(\mathbf{v})$.

$\beta_{\mathbf{u}^*}$ is the best approximation in $\mathcal{B}$ of the Bayesian posterior given $\mathcal{D}$ and α.

Penalty–Bayes – Variational Inference: Examples

Gaussian posteriors:

- family of products of Gaussian distributions on $\mathbf{w} \in \mathbb{R}^N$:

$$\beta_{\mathbf{u}} = \beta_{(\mu_1, \sigma_1^2, \cdots, \mu_N, \sigma_N^2)} = \mathcal{N}(\mu_1, \sigma_1^2) \otimes \cdots \otimes \mathcal{N}(\mu_N, \sigma_N^2);$$

- prior: $\alpha = \mathcal{N}(0, \sigma^2)$, where σ is fixed.

Deterministic posteriors:

- family of products of Diracs on $\mathbf{w} \in \mathbb{R}^N$:

$$\beta_{\mathbf{u}} = \beta_{(\mu_1, \cdots, \mu_N)} = \delta_{\mu_1} \otimes \cdots \otimes \delta_{\mu_N};$$

- prior: $\alpha = \mathcal{N}(0, \sigma^2)$, where σ is fixed.

Penalty–Bayes – Assumptions and Notation

- we parameterize the variational posteriors $\beta_{\mathbf{u}}$ on $\mathbf{w} \in \mathbb{R}^N$ by their mean $\boldsymbol{\mu} \in \mathbb{R}^N$ and another parameter $\boldsymbol{\nu}$: $\beta_{\mathbf{u}} = \beta_{\boldsymbol{\mu},\boldsymbol{\nu}}$.
  Example: if $\beta_{\mathbf{u}} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then $\beta_{\mathbf{u}} = \beta_{\boldsymbol{\mu},\boldsymbol{\Sigma}}$;
- moreover, we assume that the family $\mathcal{B}$ of variational posteriors is translation-invariant:

$$\forall \mathbf{w} \in \mathbb{R}^N, \qquad \beta_{\boldsymbol{\mu},\boldsymbol{\nu}}(\mathbf{w}) = \beta_{\mathbf{0},\boldsymbol{\nu}}(\mathbf{w} - \boldsymbol{\mu});$$

- we denote indistinctly: $r(\mathbf{u}) = r(\boldsymbol{\mu}, \boldsymbol{\nu}) = r_{\boldsymbol{\nu}}(\boldsymbol{\mu})$;
- entropy of a distribution: $\mathrm{Ent}(\beta_{\mathbf{0},\boldsymbol{\nu}}) = -\mathbb{E}_{\mathbf{w}\sim\beta_{\mathbf{0},\boldsymbol{\nu}}}[\ln \beta_{\mathbf{0},\boldsymbol{\nu}}(\mathbf{w})]$;
- let $\check{\beta}_{\mathbf{0},\boldsymbol{\nu}}(\mathbf{w}) = \beta_{\mathbf{0},\boldsymbol{\nu}}(-\mathbf{w})$.

Penalty–Bayes – Main Theorem

Goal: given a function r, find a probability distribution α such that:

$$\exists K \in \mathbb{R} : \forall \mathbf{u} \in \mathcal{U}, \qquad r(\mathbf{u}) = \mathrm{KL}(\beta_{\mathbf{u}} \| \alpha) + K. \tag{2}$$

Definition 4 (c.f. Definition 1)

Let $A_{\nu} = -\mathrm{Ent}(\beta_{0,\nu})\mathbb{1} - \mathcal{F}^{-1}\left[\frac{\mathcal{F} r_{\nu}}{\mathcal{F}\check{\beta}_{\mathbf{0},\nu}}\right]$.

r fulfills condition $(\star) \Leftrightarrow \begin{cases} A_{\nu} \text{ does not depend on } \nu \\ A \text{ is a function s.t. } \exp(A) \text{ integrates to } \kappa > 0 \end{cases}$.

Theorem 5 (informal, c.f. Theorem 1)

Equation (1) *has a solution* $\alpha \in \mathcal{T} \Leftrightarrow$ *r fulfills* $(\star)$ *and* $\alpha = \frac{1}{\kappa}\exp(A)$.

Penalty–Bayes – Fixing λ: Examples

Gaussian distributions with $\mathcal{L}^2$ penalty: in Corollary 3, we have proven that $\alpha = \alpha_{\sigma_0^2} \sim \mathcal{N}(0, \sigma_0^2)$.

$$\alpha \text{ fulfills Glorot} \Leftrightarrow \sigma_0^2 = \frac{1}{P_l}.$$

Deterministic distributions: for a f.c. layer $\mathbf{w}$, we consider the penalty $\lambda \tilde{r}(\mathbf{w})$. We propose to use $\lambda_{\text{Bayesian}}$ given below, instead of λ_{usual}.

$\tilde{r}(\mathbf{w})$	$\|\mathbf{w}\|_2^2$	$\|\mathbf{w}\|_1$	$\|\mathbf{w}\|_{2,1}$	$\|\mathbf{w}^T\|_{2,1}$
$\lambda_{\text{Bayesian}}$	$P_l/2$	$\sqrt{2P_l}$	$\sqrt{P_l(P_l+1)}$	$\sqrt{P_l(n_l+1)}$
λ_{usual}	1	1	$\sqrt{P_l}$	$\sqrt{n_l}$

where $\|\mathbf{w}\|_{2,1} = \sum_i \|\mathbf{w}_i\|_2$ is the group-Lasso penalty, $\mathbf{w}_i$ being the i-th row of $\mathbf{w}$.

Penalty–Bayes – Experimental Results

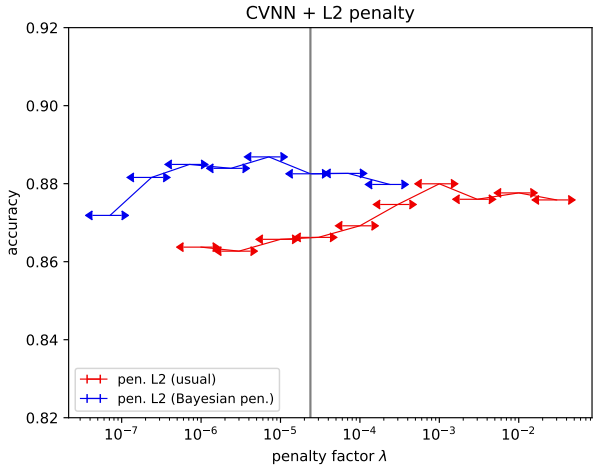
Tested architectures: simple convolutional NN (CVNN) and VGG19.
Complete penalty: $\lambda \sum_l \lambda_l r(\mathbf{w}_l)$, where $\mathbf{w}_l$ is the tensor of the l-th layer.

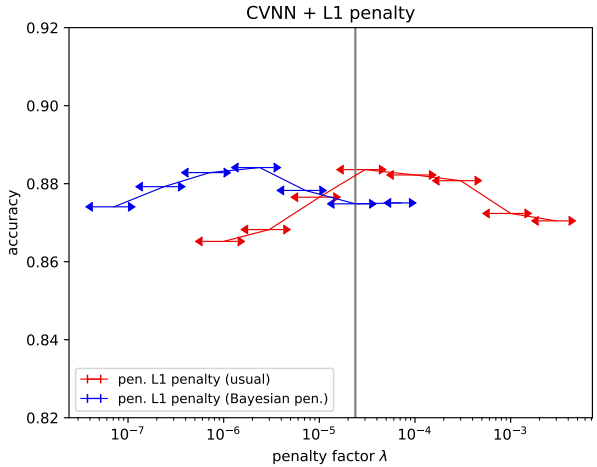
Experiments:

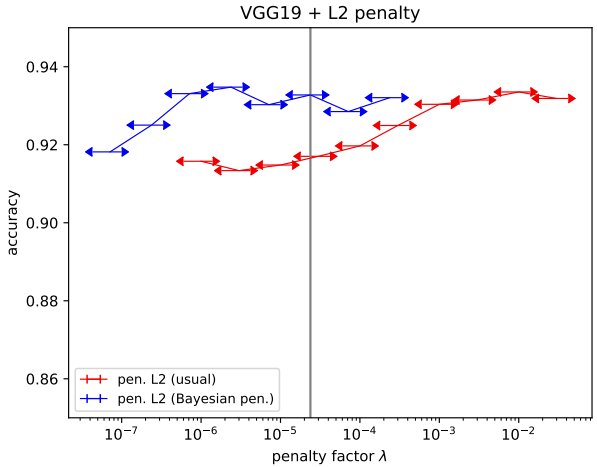
- "usual setup": λ_l is set to λ_{usual} (see above);
- "Bayesian setup": λ_l is set to $\lambda_{\text{Bayesian}}$ (see above);
- in both setups: grid search over $\lambda \Rightarrow (\lambda^*, \text{acc}^*)$;
- in the Bayesian setup: λ should be theoretically equal to $\lambda_{\text{Th}} = 1/\#[\text{training set}]$.

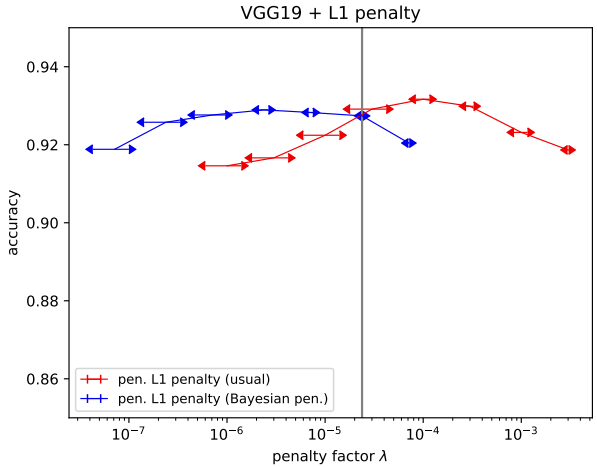
	$\|w\|_2^2$		$\|w\|_1$		$\|w\|_{2,1}$		$\|w^T\|_{2,1}$	
	CVNN	VGG	CVNN	VGG	CVNN	VGG	CVNN	VGG
$\text{acc}^*_{\text{usual}}$ (%)	$88.00 \pm .4$	$93.35 \pm .15$	$88.36 \pm .3$	$93.17 \pm .3$	$88.43 \pm .14$	$92.78 \pm .19$	$88.04 \pm .4$	$93.37 \pm .09$
$\text{acc}^*_{\text{Bayesian}}$	$88.69 \pm .12$	$93.48 \pm .09$	$88.41 \pm .3$	$92.89 \pm .2$	$88.67 \pm .09$	$92.35 \pm .18$	$88.32 \pm .16$	$93.03 \pm .15$
$\text{acc}_{\text{Bayesian}}$	$88.25 \pm .3$	$93.28 \pm .17$	$87.48 \pm .08$	$92.74 \pm .19$	$87.45 \pm .17$	$92.24 \pm .14$	$85.49 \pm .3$	$92.85 \pm .06$
$\lambda_{\text{Th}}/\lambda^*$	$10^{0.5}$	10^1	10^1	10^1	10^2	10^2	$10^{1.5}$	10^1

Penalty–Bayes – Graphs (1)

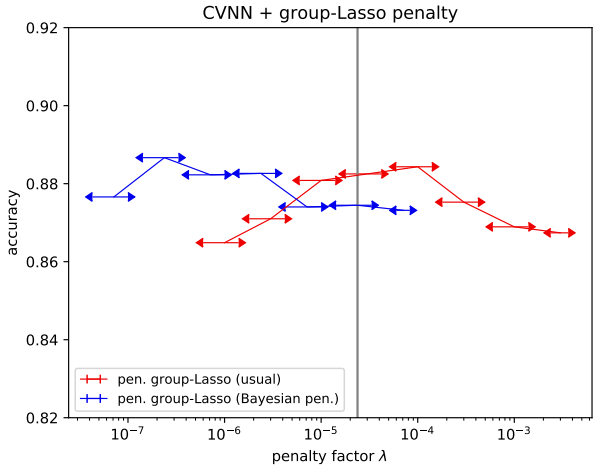


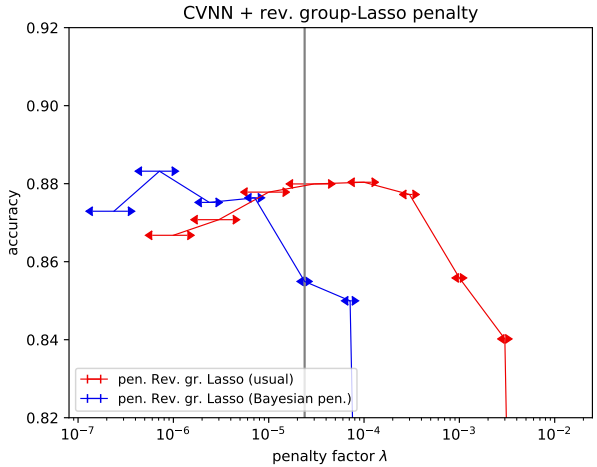


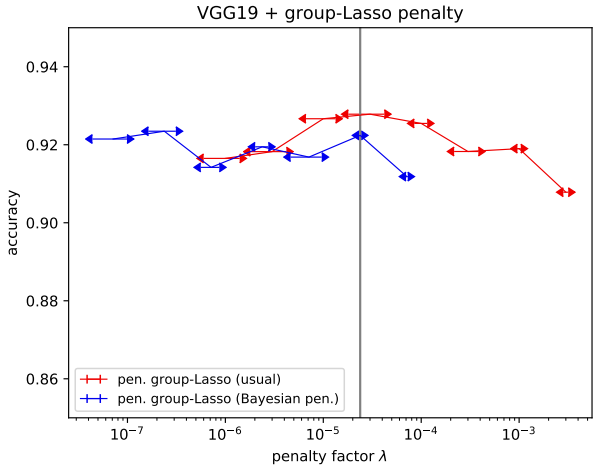


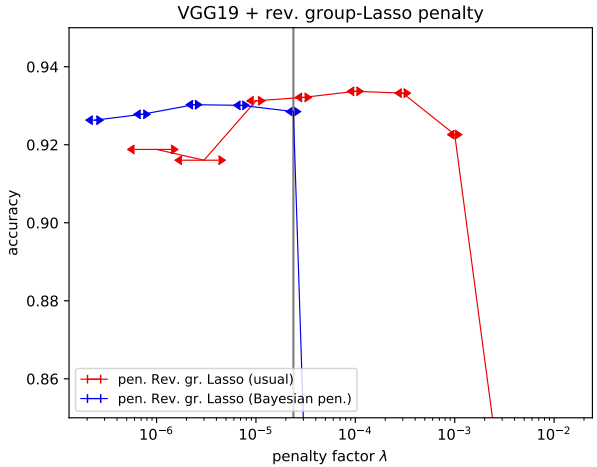


Penalty–Bayes – Graphs (2)







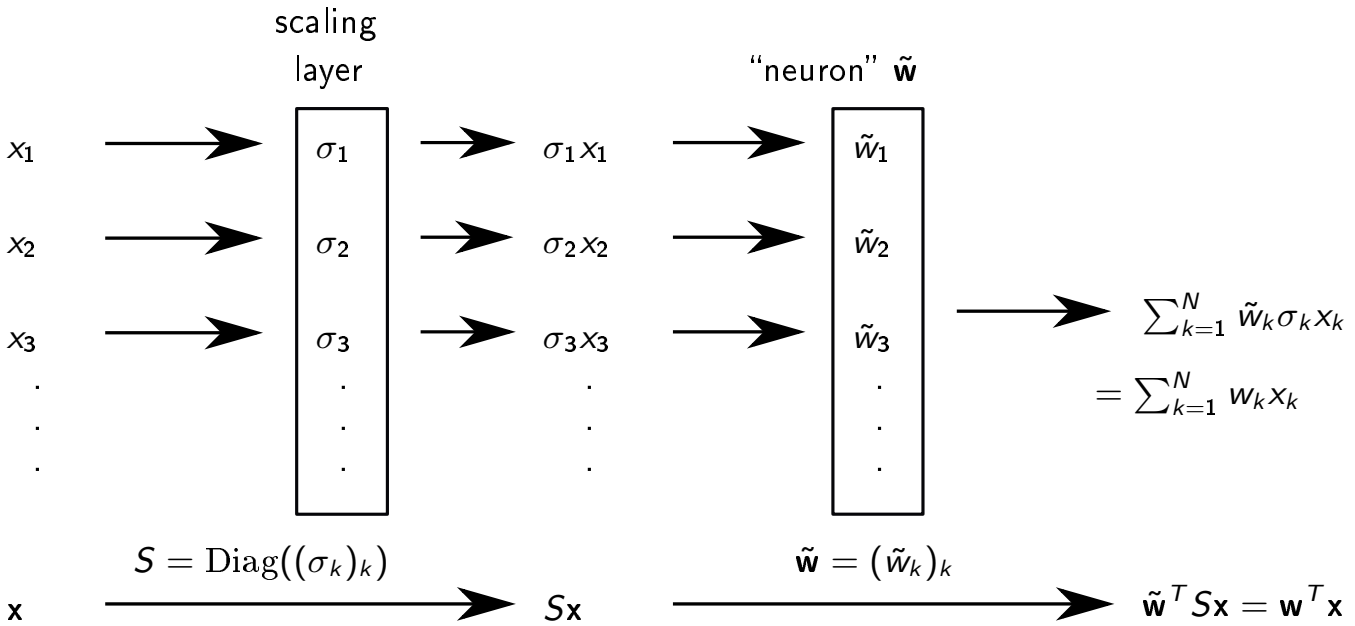


Scaling Layers – ScaLa: Definition of the Scaling Layer

We consider: a loss L, one neuron $\mathbf{w} \in \mathbb{R}^N$ and one data point $(\mathbf{x}, y) \in \mathbb{R}^N \times \mathbb{R}$.

SGD on $\mathbf{w} \Rightarrow$ SGD on $\tilde{\mathbf{w}}$ (where $\mathbf{w} = S\tilde{\mathbf{w}}$, with $S = \mathrm{Diag}(\sigma_1, \cdots, \sigma_N)$).

Equivalent to a new update rule: $w_k \leftarrow w_k - \eta \nabla_{w_k} L \;\Rightarrow\; w_k \leftarrow w_k - \eta \sigma_k^2 \nabla_{w_k} L$.



Scaling Layers – ScaLa: Forward Pass and Backward Pass

$\mathbf{w}$ is randomly initialized, and $\mathbf{x}$ and $\nabla_y L$ are supposed to be random.
Properties to check:

- **forward pass:** we want a bounded variance for activation y as $N \to \infty$;
- **backward pass:** we want a bounded variance for the difference $(y' - y)$ of activations y', obtained after one update, and y, as $N \to \infty$.

We check it for *one neuron* $\mathbf{w}$.

Proposition 6 (informal, c.f. Corollary 2)

We have:

$$\sum_{k=1}^{\infty} \sigma_k^2 < \infty \Leftrightarrow \begin{cases} \lim_{N\to\infty} \mathrm{Var}(y_{(N)}) & < \infty \\ \lim_{N\to\infty} \mathrm{Var}(y'_{(N)} - y_{(N)}) & < \infty \end{cases} .$$

Examples: $\sigma_k \propto 1/(k^{1/2} \ln k)$ or $\sigma_k \propto 1/k$.

Scaling Layers – ScaLa: Insensitivity to Width Change

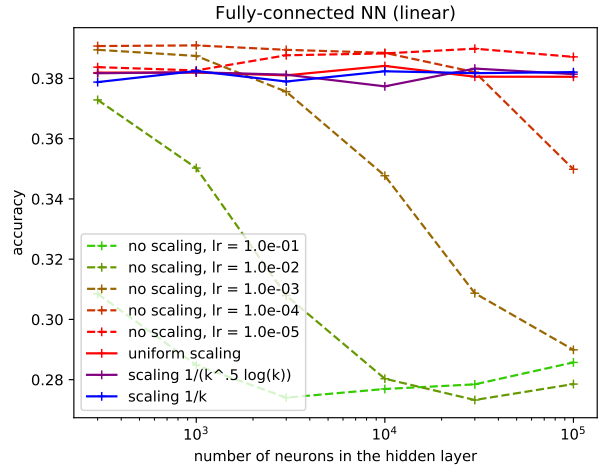
Practical scalings: $\sigma_k \propto \begin{cases} 1/\sqrt{N} \\ 1/(k^{1/2} \ln k) \\ 1/k \end{cases}$

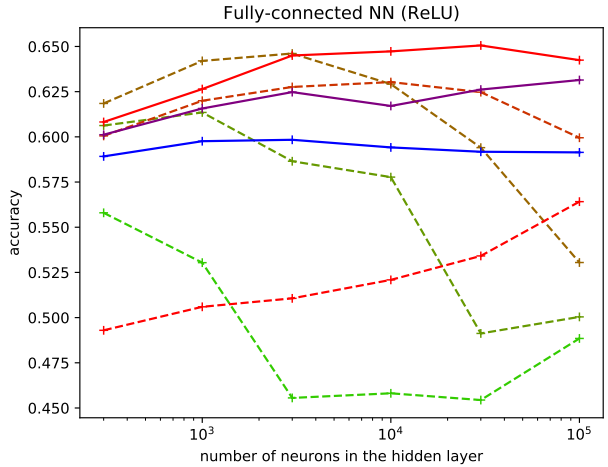
Experiments: CIFAR-10, NN with one hidden layer, Id or ReLU activation function.
Test of various N: from 300 to 100 000

Standard SGD: test of many learning rates, from 10^{-5} to 10^{-1}.

SGD with ScaLa: one only learning rate ($\eta = 1$), and test of various scaling layers.

SGD with ScaLa is more resilient than standard SGD to width change.





Scaling Layers – ScaLP: Asymmetrical Penalization

ScaLP: use ScaLa with asymmetrical scaling layers to prune neurons.

Example: $\mathcal{L}^2$ penalty.

$$\text{pen}(\mathbf{w}) := \sum_{l=1}^{L} \sum_{k=1}^{n_l} \sum_{i} (\tilde{w}_{ki}^l)^2 = \sum_{l=0}^{L-1} \sum_{k=1}^{n_l} \left[\frac{1}{\sigma_{l+1,k}^2} \underbrace{\sum_{w \in \mathbf{w}_{k\rightarrow}^l} w^2}_{m(\mathbf{w}_k^l)^2, \text{ usefulness of neuron } \mathbf{w}_k^l \text{ from the point of view of layer } l+1} \right],$$

where $\mathbf{w}_k^l$ is the k-th neuron in the l-th layer, and $\mathbf{w}_{k\rightarrow}^l$ is the set of its *output weights*, i.e., weights in layer $l+1$ that are linked to $\mathbf{w}_k^l$.

Interpretation.
The usefulness $m(\mathbf{w}_k^l)^2$ of each neuron $\mathbf{w}_k^l$ is pushed towards 0 with a scaling factor $1/\sigma_{l+1,k}^2$.

Asymmetrical scaling layer $S \Rightarrow$ Asymmetrical penalization

Scaling Layers – ScaLP: Consequences on Pruning

Summary:

- we choose a (decreasing) sequence $(\sigma_{l+1,k})_k$ with $\sum_{k=1}^{\infty} \sigma_{l+1,k}^2 < \infty$;
- we build an asymmetrical scaling layer $\mathrm{Diag}(\sigma_{l+1,1}, \cdots, \sigma_{l+1,n_l})$;
- its output weights $\mathbf{w}_{k\rightarrow}^l$ are pushed towards 0 with force $1/\sigma_{l+1,k}^2$.

Should we prune neuron $\mathbf{w}_k^l$?

- we measure its usefulness: $m(\mathbf{w}_k^l) = \sqrt{\sum_{w \in \mathbf{w}_{k\rightarrow}^l} w^2}$;
- given a threshold ϵ, if $\frac{1}{\#[\mathbf{w}_{k\rightarrow}^l]} m(\mathbf{w}_k^l) < \epsilon$, then $\mathbf{w}_k^l$ is pruned.

The more the index k of a neuron is high, the more its outputs weights are pushed toward 0, the less it is useful, the more likely it will be pruned.

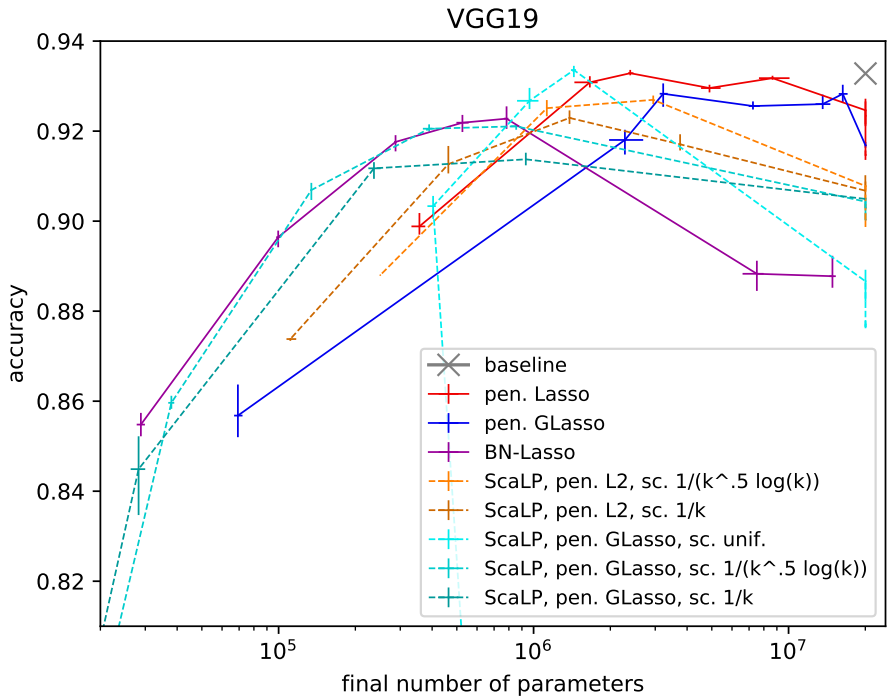
Scaling Layers – ScaLP: Accuracy and Pruning Results

Experiments: VGG19 trained on CIFAR-10, for various penalty factors λ:

$$L(\mathbf{w}) = \ell(\mathbf{w}) + \lambda \mathrm{pen}(\mathbf{w}).$$

Accuracy: best accuracy obtained with ScaLP (pen. GLasso and uniform scaling).

Pruning: ScaLP competitive with BN-Lasso (pen. GLasso and scaling $1/(k^{1/2} \log k)$)



Final accuracy in function of final number of parameters.

Dotted lines: with ScaLP
Continuous lines: without ScaLP

Scaling Layers – ScaLP: Stability of the Final Architecture

With asymmetrical scaling layers, we hope that the final width of a layer is independent from its initial width.

Experiment: train a neural neural network with one hidden layer of size N for different N (MNIST).
Test with BN-Lasso and ScaLa with group-Lasso.

Unlike BN-Lasso, ScaLP with asymmetrical scaling returns the same network architecture.

Over the runs: the returned architecture is stable between runs with ScaLP, unlike BN-Lasso.