Foundations of Deep Learning, Winter Term 2021/22

Week 5: Regularization

Regularization

Frank Hutter Abhinav Valada

University of Freiburg

Overview of Week 5

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

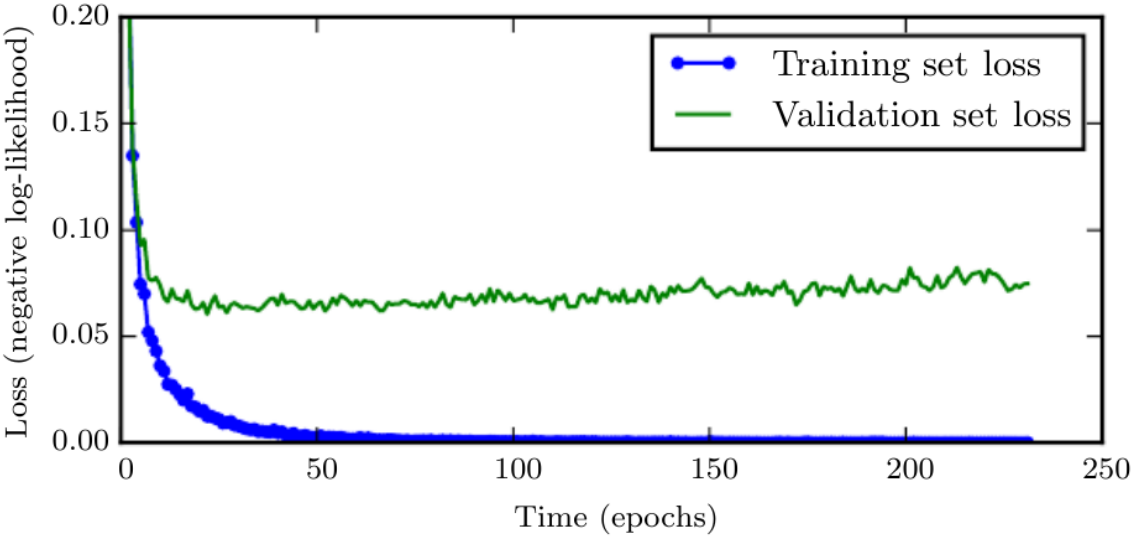
Week 5: Regularization

Motivation for Regularization in Deep Learning

Frank Hutter Abhinav Valada

University of Freiburg

Motivation



Goal of regularization: better generalization [Goodfellow et al., 2016, p. 243, fig 7.3]

Motivation

Bias vs. Variance Tradeoff of Deep Neural Networks

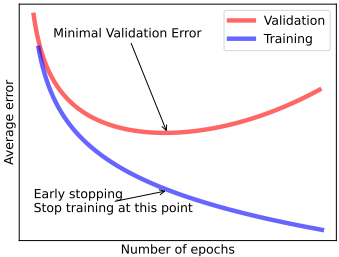
- Standard bias-variance decomposition of the generalization error (here for MSE):

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

- Deep neural networks have low bias and high variance
  - They can represent any function;
    in fact, they can perfectly fit random noise! [Zhang et al., 2017]
  - But they have high variance:
    if you change the training data a little the fitted function may change a lot
- Goal of regularization:
  - Introduce *some* bias
  - Substantially reduce variance
  - Therefore, lead to lower generalization loss

Early Stopping

- Stop training when error on the validation set has reached its minimum
- Often, training is already stopped after a few epochs
  - typically combined with small initial weights
- Simple, popular heuristic
- Needs perpetual observation of the validation error



Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  What are the goals of regularization?
- Repetition:
  Do deep neural networks typically have low or high bias? Low or high variance?
- Repetition:
  How do you implement Early Stopping?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 5: Regularization

Preference For Small Weights

Frank Hutter Abhinav Valada

University of Freiburg

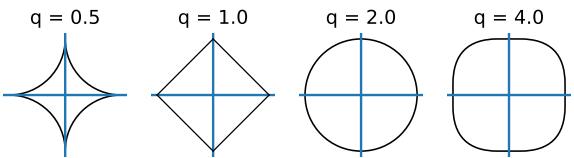
Shrinkage Methods

- Extend the loss function with an extra regularizer (penalty) term of strength λ to control overfitting:

$$L(\theta) = L_D(\theta) + \lambda L_{\text{reg}}(\theta)$$

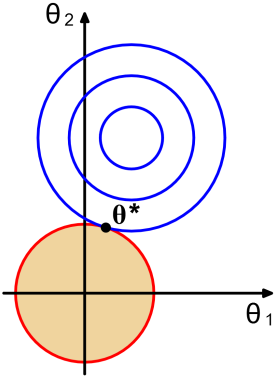
- Regularizer penalizes large weights: $L_{\text{reg}}(\theta) = \frac{1}{q} \sum_{j=1}^{M} |\theta_j|^q$
  - Case $q = 2$: L_2 regularizer, quadratic penalties
  - Case $q = 1$: L_1 regularizer (lasso in the literature)

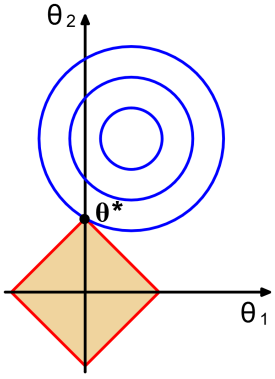
- Contour lines of equivalent penalty values



Shrinkage Methods

- L_1 regularization leads to sparser solutions $\rightarrow$ some weights are exactly zero
- Left: L_2 regularization $(q = 2)$, right: L_1 regularization $(q = 1)$





Decoupled Weight Decay

- L_2 regularization is typically implemented by directly changing the gradient to

$$\boldsymbol{g}' = \boldsymbol{g} + \frac{1}{2}\lambda\nabla_{\boldsymbol{\theta}}\sum_{j=1}^{M}|\theta_j|^2 = \boldsymbol{g} + \lambda\boldsymbol{\theta}$$

  and thus changing the SGD update to:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\boldsymbol{g}' = \boldsymbol{\theta} - \alpha(\boldsymbol{g} + \lambda\boldsymbol{\theta}) = (1 - \alpha\lambda)\boldsymbol{\theta} - \alpha\boldsymbol{g}$$

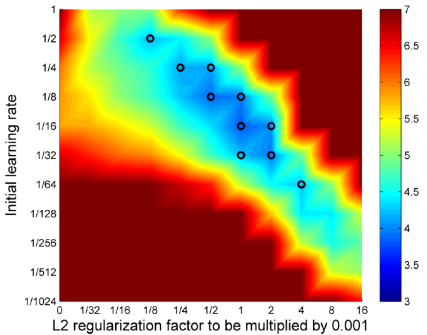
- Decoupled weight decay [Hanson & Pratt, 1988; Loshchilov & Hutter, 2017] pulls the weights towards zero at each update step of SGD:

$$\boldsymbol{\theta} \leftarrow (1 - w)\boldsymbol{\theta} - \alpha\boldsymbol{g}$$

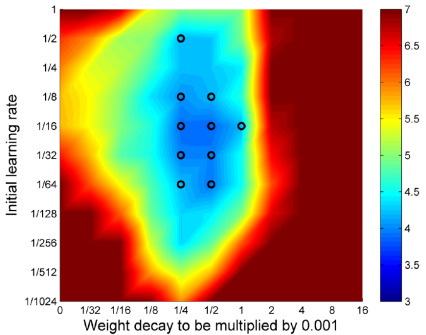
  - For SGD, this is identical to L_2 regularization with $\lambda = w/\alpha$
  - That is why L_2 regularization is often called weight decay
  - But hyperparameters λ and α are coupled in L_2 regularization: if you change α you also have to change λ to keep the same weight decay!

Decoupled Weight Decay

- Hyperparameters λ and α are coupled in L_2 regularization:
  if you change α you also have to change λ to keep the same weight decay!
  - In decoupled weight decay, the w and α hyperparameters are decoupled



L_2 regularization

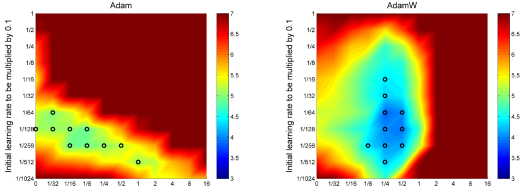


Decoupled weight decay regularization

[Image source: Loshchilov and Hutter, 2017]

Decoupled Weight Decay for Adaptive Gradient Algorithms

- For adaptive gradient algorithms, decoupled weight decay $\neq L_2$ regularization
  - With decoupled weight decay regularization: adaptive gradient algorithms only adapt gradients $\mathbf{g}$ of the loss function $\rightsquigarrow$ weight decay is decoupled from gradient adaptation
  - With L_2 regularization: adaptive gradient algorithms adapt combined gradients $\boldsymbol{g'}$ of (loss function + regularizer) $\rightsquigarrow$ weights with historically large gradients are regularized less
- Adam can generalize better with decoupled weight decay $\rightsquigarrow$ popular AdamW variant



L_2 regularization

Decoupled weight decay regularization

[Image source: Loshchilov and Hutter, 2017]

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  Which two hyperparameters have a strong interaction effect in SGD under L_2 regularization?
- Discussion:
  Why may L_2 regularization regularize some weights less than desired when used in adaptive gradient methods?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 5: Regularization

Parameter Sharing and Parameter Tying

Frank Hutter Abhinav Valada

University of Freiburg

Parameter Sharing

- If we can share parameters in the network we implicitly have more data for training them
  - E.g., convolutions: use the same filters across the input image
    - Without parameter sharing, a CNN would have to learn feature detectors (=filters) separately for each input patch!
    - With parameter sharing, feature detection is invariant to translations
  - E.g., recurrent networks: apply the same network at each time step
- Control the network's capacity, encouraging search for regular patterns
  - This introduces prior knowledge
- Limit memory cost

Example: Multitask Learning

- Use one neural network to tackle multiple tasks at once
- E.g., have an intermediate level shared representation $\boldsymbol{h}^{(shared)}$ and individual model heads $\boldsymbol{h}^{(1)}$, $\boldsymbol{h}^{(2)}$ and $\boldsymbol{h}^{(3)}$
- Use examples from the different tasks for training the shared parts of the network
- This can have a similar effect as additional data if the tasks are related

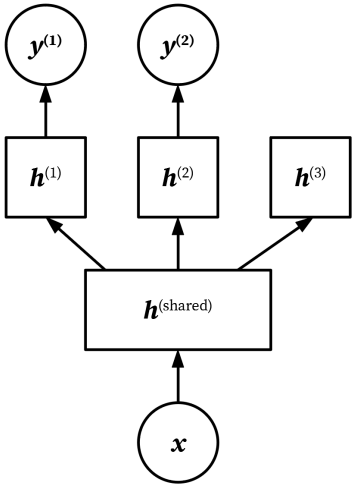


Image modified from [Goodfellow et al., 2016, p. 242, fig 7.2]

Parameter Tying

- Training a neural network to be similar to another neural network
  - E.g., perform the same task on a different input distribution
- Add a soft constraint to make parameters of the two networks similar
  - E.g., model A with parameters $\boldsymbol{\theta}^{(A)}$ and model B with parameters $\boldsymbol{\theta}^{(B)}$:

$$\hat{y}^{(A)} = f(\boldsymbol{x}, \boldsymbol{\theta}^{(A)}) \qquad \hat{y}^{(B)} = f(\boldsymbol{x}, \boldsymbol{\theta}^{(B)})$$

  - If the tasks are similar enough, we can leverage this information through regularization (e.g., L_2 norm between weights):

$$\Omega(\boldsymbol{\theta}^{(A)}, \boldsymbol{\theta}^{(B)}) = \|\boldsymbol{\theta}^{(A)} - \boldsymbol{\theta}^{(B)}\|_2^2$$

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  Describe the difference between parameter sharing and parameter tying.

- Discussion:
  What could be *disadvantages* of sharing parameters in deep networks?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 5: Regularization

Dropout

Frank Hutter Abhinav Valada

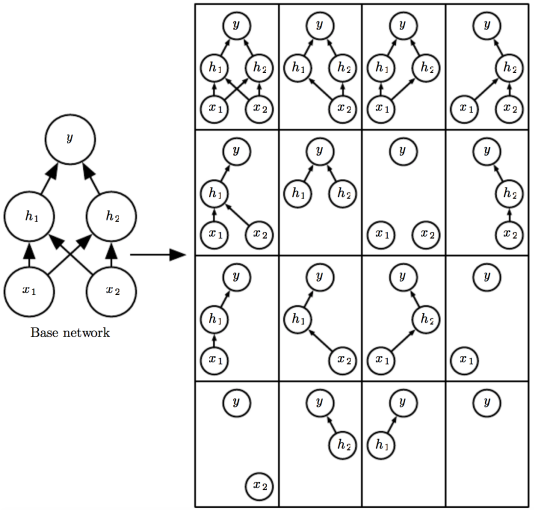
University of Freiburg

Dropout

- Randomly "drop" units from the network during training [Hinton et al., 2012]
- Dropping units: multiplying their output by zero
- For each training data point, retain each non-output unit with probability p
  - p is a hyperparameter (typically 0.5 for hidden units, 0.8 for input units)
  - For each data point, a different mask is sampled
- Forward propagation and back-propagation remain unchanged
- This avoids co-adaptation of the weights (relying on other units of the network too much)
- Note that in our exercises we use $p_{delete} = 1 - p$ as a hyperparameter to stay in line with PyTorch.

Dropout

- One view: ensemble of 2^n neural networks with shared weights (n = number of non-output units)
- A tiny fraction of the possible subnetworks are trained in each step
- Weight sharing also implicitly trains the other subnetworks
  - Updates of a weight matrix between two nodes affect all subnetworks that have this connection



[Goodfellow et al., 2016, p. 256, fig 7.6]

Dropout Inference

- To predict: optimally we would like to average predictions of all 2^n subnetworks
- Each subnetwork is defined by a binary mask vector $\boldsymbol{\mu}$
- Each subnetwork defines a probability distribution $P(y \mid \boldsymbol{x}, \boldsymbol{\mu})$.
- Correct prediction is a weighted mean over the masks:

$$\sum_{\mu} p(\boldsymbol{\mu}) P(y \mid \boldsymbol{x}, \boldsymbol{\mu}), \tag{1}$$

  where $p(\boldsymbol{\mu})$ is the distribution used to sample $\boldsymbol{\mu}$ at training time

- Unfortunately, no closed-form solution of (1) is available
  - Solution 1: Monte Carlo approximation based on 10–20 masks
  - Solution 2: Scale all weights by p; this is an approximation

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  What does dropout try to achieve?
- Discussion:
  Why is dropout more efficient than training a regular ensemble?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 5: Regularization

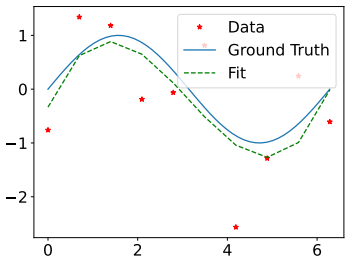
Data Augmentation

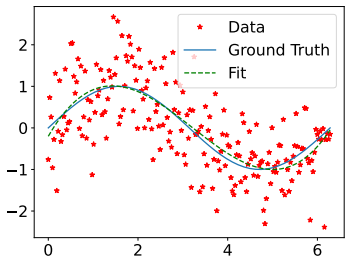
Frank Hutter Abhinav Valada

University of Freiburg

More Data

- Data analysts' fundamental slogan: *there's no data like more data!*
- Try to get more data; if not possible directly, think about related sources of similar data
- Although trivial, one of the most important techniques to improve ML models





Data Augmentation

- Data augmentation is very common in deep learning for creating additional training data
- Example: computer vision
  - Translation, Scaling, Reflection, Rotation, Stretching

- Hypothesis learns representation invariant to the perturbations
  - Often yields very substantial improvements
  - However: you need to know which perturbations you want to be invariant to
- What data do we apply these augmentations to?

All data	Only the test data
Only the validation data	Only the training data

Other Domain-Specific Data Augmentations

- E.g., speech recognition
  - adding background noise
  - changing tone
  - changing speed
- E.g., text classification
  - replacing words by synonyms
- How much of each augmentation to apply: hyperparameters!

AutoAugment [Cubuk et al., 2019]

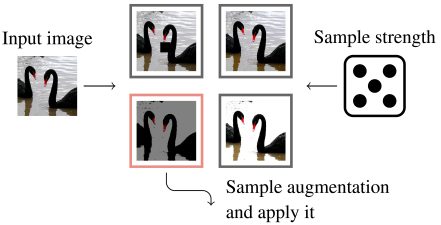
- We can easily evaluate a combination of augmentations and their respective strengths:
  - Train the model with this combination of augmentations
  - Evaluate the resulting model on a validation set
- We can search for good settings of these hyperparameters automatically, with many different methods
  - More on such methods in the lecture on hyperparameter optimization
  - In Auto-Augment, Cubuk et al. used policy gradient methods
- Results
  - Much better results than before
  - But large computational cost to search for best combination of augmentations
    - About 15 000 combinations evaluated per dataset

RandAugment [Cubuk et al., 2020]

- Rather than optimizing the choice of augmentation, sample random augmentations
- Only 2 hyperparameters remain to be tuned:
  - N: how many transformations to use each time (randomly chosen from candidate set)
  - M: strength of all the transformations
- Much cheaper and just as good performance as AutoAugment

TrivialAugment [Müller & Hutter, 2021]

- Even simpler – for each image in a batch:
  - sample a single augmentation
  - sample a strength for it
- No need for tuning
  - other than choosing the candidate set of augmentations and strengths



- Even better performance than RandAugment and AutoAugment

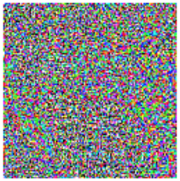
Method	Search Overhead	CIFAR-10 ShakeShake	CIFAR-100 WRN	SVHN WRN	ImageNet ResNet
AA	40 - 800×	98.0	82.9	98.9	77.6
RA	4 - 80×	98.0	83.3	**99.0**	77.6
Fast AA	1×	98.0	82.7	98.8	77.6
TA (ours)	**0×**	**98.2**	**84.3**	98.9	**78.1**

Noise Robustness

- Improve robustness by training with randomly added noise
- Noise can be added in various places:
  - Inputs
    - $\rightsquigarrow$ Data augmentation
  - Hidden units
    - $\rightsquigarrow$ Data augmentation at various levels of abstraction
  - Weights
    - $\rightsquigarrow$ Stochastic implementation of Bayesian inference
  - Output targets
    - $\rightsquigarrow$ Label smoothing

Adversarial Training

$+ .007 \times$



$=$

$\boldsymbol{x}$	$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$	$\boldsymbol{x} + \epsilon\,\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
$y =$"panda"	"nematode"	"gibbon"
w/ 57.7%	w/ 8.2%	w/ 99.3%
confidence	confidence	confidence

- Background: adversarial examples
  - Even for very strong networks we can find adversarial examples
  - By following the gradient of the cost function w.r.t the input

[Goodfellow et al., 2016, p. 265, fig 7.8]

Adversarial Training

- Procedure for adversarial training
  - Train a network on the training examples
  - Generate adversarial examples for this network
  - Add these to the training data and repeat
- This encourages local constancy and mitigates sensitivity to small input changes
- Downside: more complex than standard data augmentation
  - Requires iteration between standard training and generating adversarial examples

Questions to Answer for Yourself / Discuss with Friends

- Discussion:
  Is more data augmentation always better?
- Repetition:
  What is the difference between RandAugment and TrivialAugment?
- Repetition:
  How do you generate adversarial examples and what is their relevance?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 5: Regularization

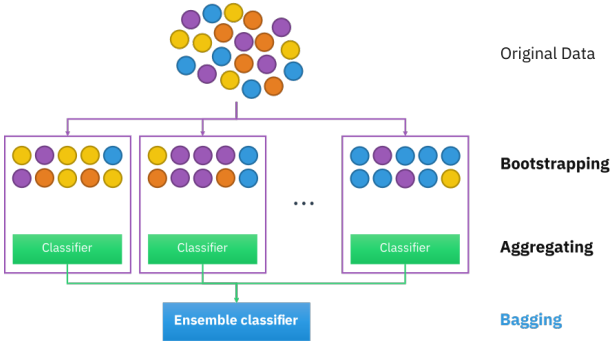
Ensemble Methods

Frank Hutter Abhinav Valada

University of Freiburg

Bagging: Bootstrapping + Aggregation [Breimann, 1996]

- Train several models on bootstrap samples of the training data
  - Data is drawn randomly with replacements ⇒ for each model, some data points may occur twice or more, others don't occur at all



[Source: Sirakorn at English Wikipedia via Wikimedia Commons]

- Aggregate the output of all trained models (e.g., via averaging)
- Introduces some bias, but reduces variance

Even Simpler: Deep Ensembles [Lakshminarayanan et al, 2017]

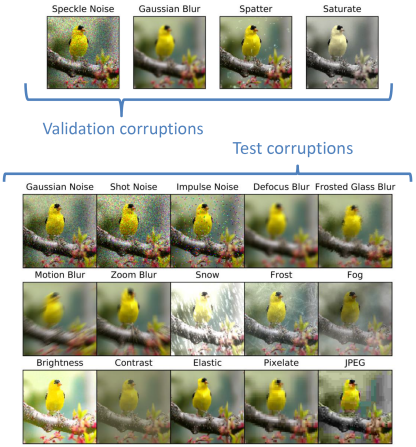
- Train several deep neural networks on the same data
  - Rely on stochastic gradient descent finding different minima
- Like in bagging: aggregate the output of all trained models (e.g., via averaging)

- Does not introduce bias, but reduces variances
- Downside of ensembling k models: k-fold increase in complexity for training time, prediction time and storage

- Surprisingly strong performance, especially for uncertainty quantification and performance under data shift

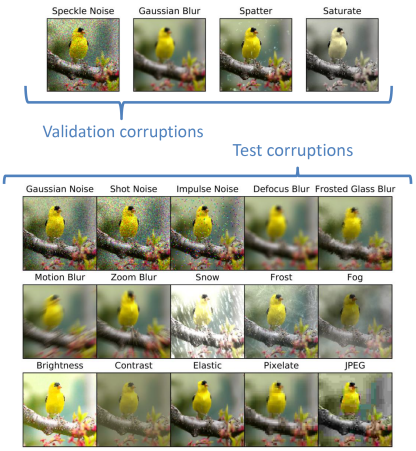
Deep Ensembles Can Handle Dataset Shift Well [Ovadia et al, 2019]

- Dataset shift: test distribution is not identical to training/validation distribution
- Example benchmark: ImageNet-C (see right) [Hendrycks & Dietterich, 2019]
- Deep ensembles yield surprisingly strong performance for uncertainty quantification and performance under data shift



Ensembles Can be Improved By Increasing the Diversity of Ensemble Members

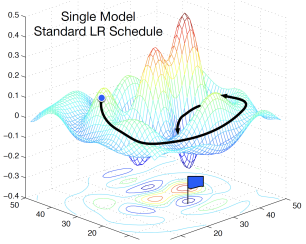
- Via different hyperparameter settings for the training pipeline
  - Hyperdeep ensembles [Wenzel et al, 2020]
  - Can directly reuse the hyperparameter evaluations from HPO [Feurer et al, 2015]
- Via different architectures
  - Different architectures often yield very different predictions
  - Neural ensemble search [Zaidi et al, 2020]

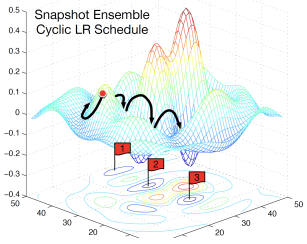


Snapshot Ensembles: Computationally Efficient Ensemble Training

[Huang et al, 2017] [Loshchilov & Hutter, 2017]

- Train using SGD with Restarts [Loshchilov & Hutter, 2017]
- Take snapshots of the weights before each restart and ensemble them [Huang et al, 2017]
- With $k = 3$ snapshots: similar performance as deep ensembles with $k = 3$ [Loshchilov & Hutter, 2017]





- Same training time complexity as a single training run
- Still k-fold overhead in memory & latency for ensembles with k members

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  What are the steps to build a deep ensemble?

- Repetition:
  What is the time complexity of creating a snapshot ensemble (compared to training a single model)?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 5: Regularization

Regularization Cocktails

Frank Hutter Abhinav Valada

University of Freiburg

Which Regularization Method Should I Use on a New Dataset?

- There are many different regularization methods
- Many of them have subsidiary hyperparameters
  - Dropout probability p, weight decay strength w, strength of parameter tying, ...
  - Augmentation hyperparameters

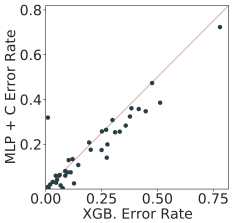
- So, which ones should be used, and with which hyperparameters?
  - Often decided based on results in the literature or personal experience
  - But in essence, this is a problem of hyperparameter optimization
    - Find the combination that optimizes validation performance

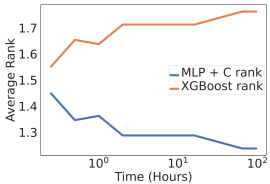
- Joint optimization yields best performance: regularization cocktails [Kadra et al, 2021]

Regularization Cocktails for Deep Learning on Tabular Data [Kadra et al, 2021]

- Tabular data is the "last unconquered castle" for deep learning
- Traditional ML methods perform best there, e.g., gradient-boosted decision trees
  - That is, they did ...
    until regularization cocktails came along.
- Well-tuned simple MLPs with 13 optimized regularizers significantly outperformed XGBoost
  - First approach to do so in a fair comparison (using a total of 31 datasets)
- Hyperparameter optimization method: BOHB [Falkner et al, 2018], to be covered in the lecture on hyperparameter optimization

Group	Regularizer	Hyperparameter	Type	Range	Conditionality
Implicit	BN	BN-active	Boolean	{True, False}	–
	SWA	SWA-active	Boolean	{True, False}	-
	LA	LA-active	Boolean	{True, False}	–
		Step size	Continuous	[0.5, 0.8]	LA-active
		Num. steps	Integer	[5, 10]	LA-active
W. Decay	WD	WD-active	Boolean	{True, False}	–
		Decay factor	Continuous	$[10^{-3}, 0.1]$	WD-active
Ensemble	DO	DO-active	Boolean	{True, False}	–
		Dropout shape	Nominal	{funnel, long funnel, diamond, hexagon, brick, triangle, stairs}	DO-active
		Drop rate	Continuous	[0.0, 0.8]	DO-active
	SE	SE-active	Boolean	{True, False}	-
Structural	SC	SC-active	Boolean	{True, False}	–
		MB choice	Nominal	{SS, SD, Standard}	SC-active
	SD	Max. probability	Continuous	[0.0, 1.0]	SC-active ∧ MB choice = SD
	SS	-	-	-	SC-active ∧ MB choice = SS
Augmentation	–	Augment	Nominal	{MU, CM, CO, AT, None}	–
	MU	Mix. magnitude	Continuous	[0.0, 1.0]	Augment = MU
	CM	Probability	Continuous	[0.0, 1.0]	Augment = CM
	CO	Probability	Continuous	[0.0, 1.0]	Augment = CO
		Patch ratio	Continuous	[0.0, 1.0]	Augment = CO
	AT	-	-	-	Augment = AT





Questions to Answer for Yourself / Discuss with Friends

- Repetition:
  Name some regularization methods and their hyperparameters

- Discussion:
  Do you have a domain of interest in which it would be interesting to study the regularization cocktail approach?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 5: Regularization

Summary, Further Reading, References

Frank Hutter Abhinav Valada

University of Freiburg

Summary by Learning Goals

Having heard this lecture, you can now ...

- Describe different general regularization techniques, including
  - early stopping
  - shrinkage methods
  - data augmentation
  - ensembling methods
- Explain some regularization techniques specific to deep learning, including
  - parameter sharing and tying
  - multitask learning
  - dropout
  - adversarial training
  - deep ensembles and snapshot ensembles
  - regularization cocktails

Further Reading

Sources for this lecture:

- Most material is based on Deep Learning book by Goodfellow, Bengio, and Courville, Chapter 7: Regularization

Further reading:

- More dropout intuition
- Dropout math and implementation

References

Goodfellow, I., Bengio, Y., Courville, A. (2016)
Deep Learning
MIT Press.
https://www.deeplearningbook.org/

Breiman., L. (1996)
Bagging predictors
Machine Learning, Vol. 24, No. 2, 1996, pp. 123–140
https://link.springer.com/content/pdf/10.1023%2FA%3A1018054314350.pdf

Loshchilov, I. and Hutter, F. (2017)
Decoupled weight decay regularization
arXiv preprint arXiv:1711.05101
https://arxiv.org/abs/1711.05101

Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. (2912)
Improving neural networks by preventing co-adaptation of feature detectors
arXiv preprint arXiv:1207.0580
https://arxiv.org/abs/1207.0580