Deep Learning 1

2025-2026 – Pascal Mettes

Lecture 4

Deep learning optimization II

Previous lecture

Lecture	Title
1	Intro and history of deep learning
3	Deep learning optimization I
5	Convolutional deep learning
7	Graph deep learning
9	Multi-modal deep learning
11	What doesn't work in deep learning
13	Q&A

Lecture	Title
2	AutoDiff
4	Deep learning optimization II
6	Attention-based deep learning
8	From supervised to unsupervised deep learning
10	Generative deep learning
12	Non-Euclidean deep learning
14	Deep learning for videos

This lecture

Overfitting and regularization

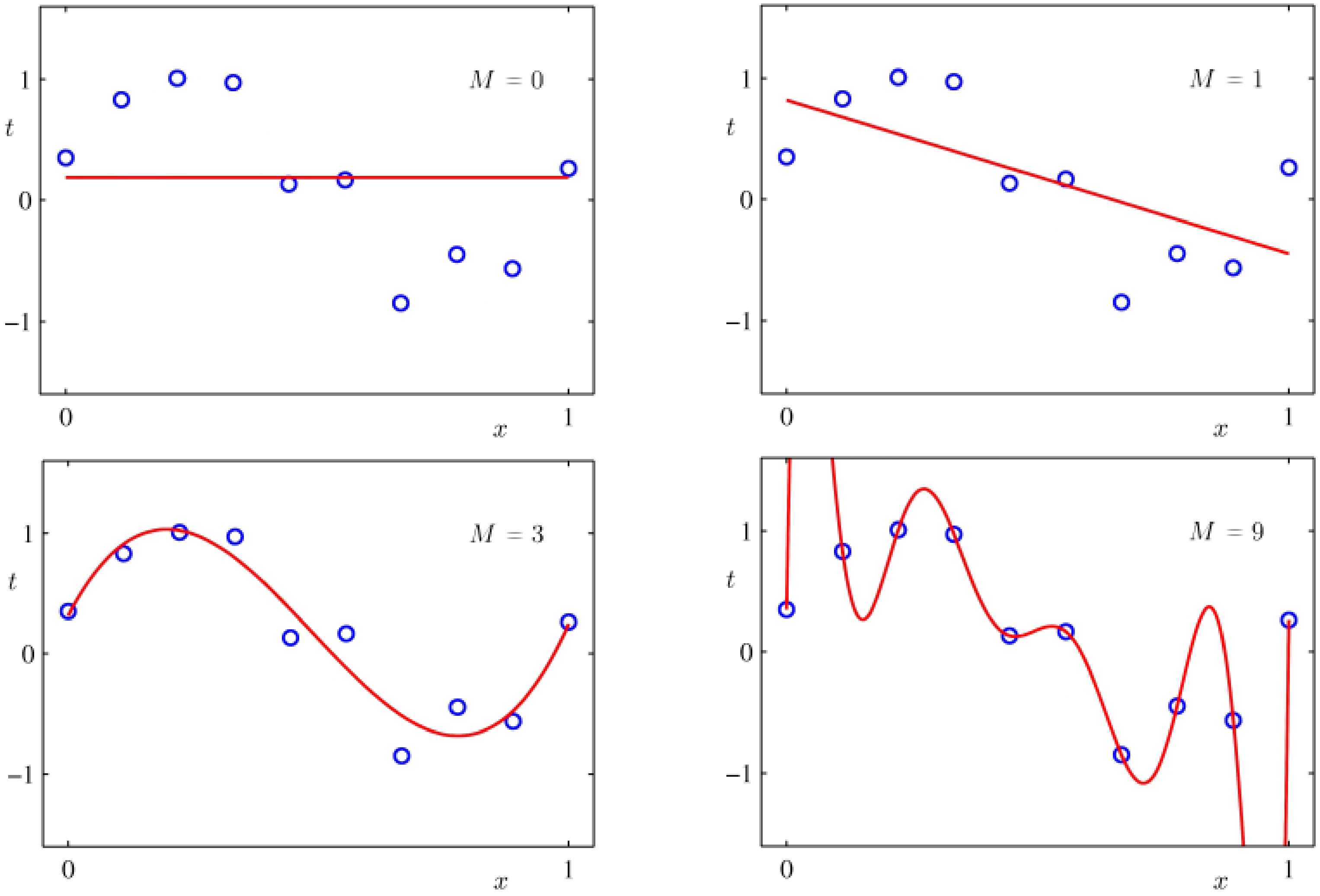
Initialization

Augmentation

Normalization

Hyperparameters

Which fit is best?



Bias-variance tradeoff

Bias

“The difference between an estimator’s expected value and the true value of the parameter being estimated”.

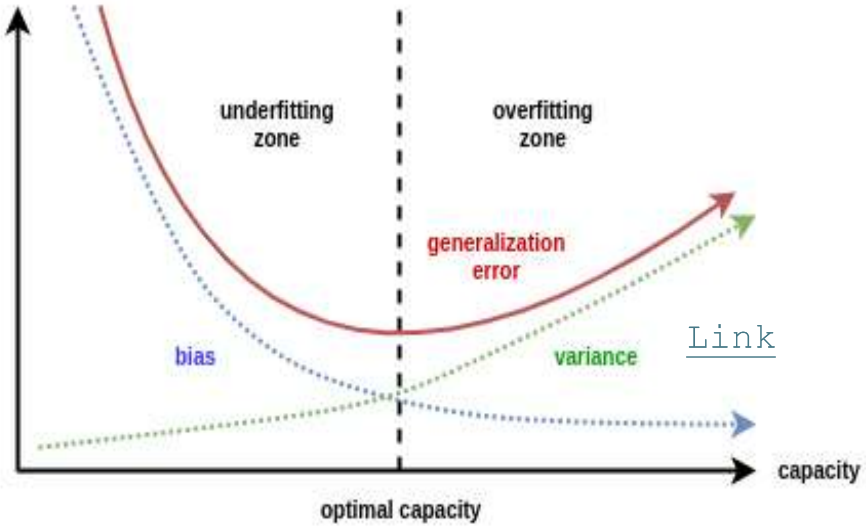
The bias error is an error from erroneous assumptions in the learning algorithm.

Variance

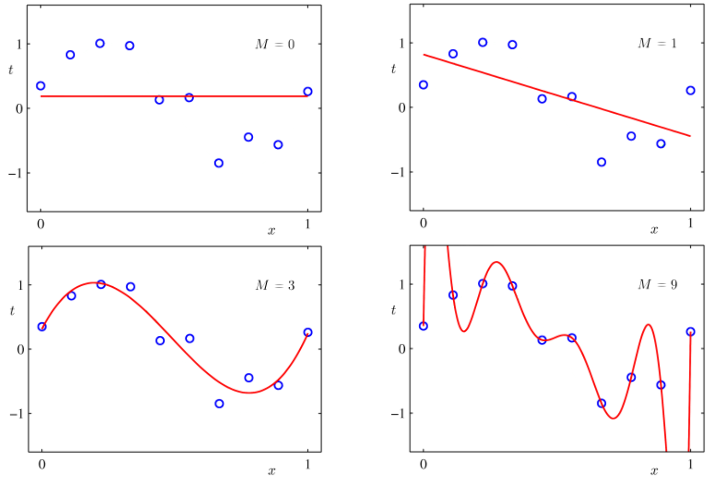
The amount that the estimate of the target function will change if different training data was used.

The variance is an error from sensitivity to small fluctuations in the training set.

Bias-variance tradeoff



Link



High bias: algorithm misses relevant relations between features and targets.

Relates to underfitting, high bias is common in linear models.

High variance: algorithm uses random noise in training data

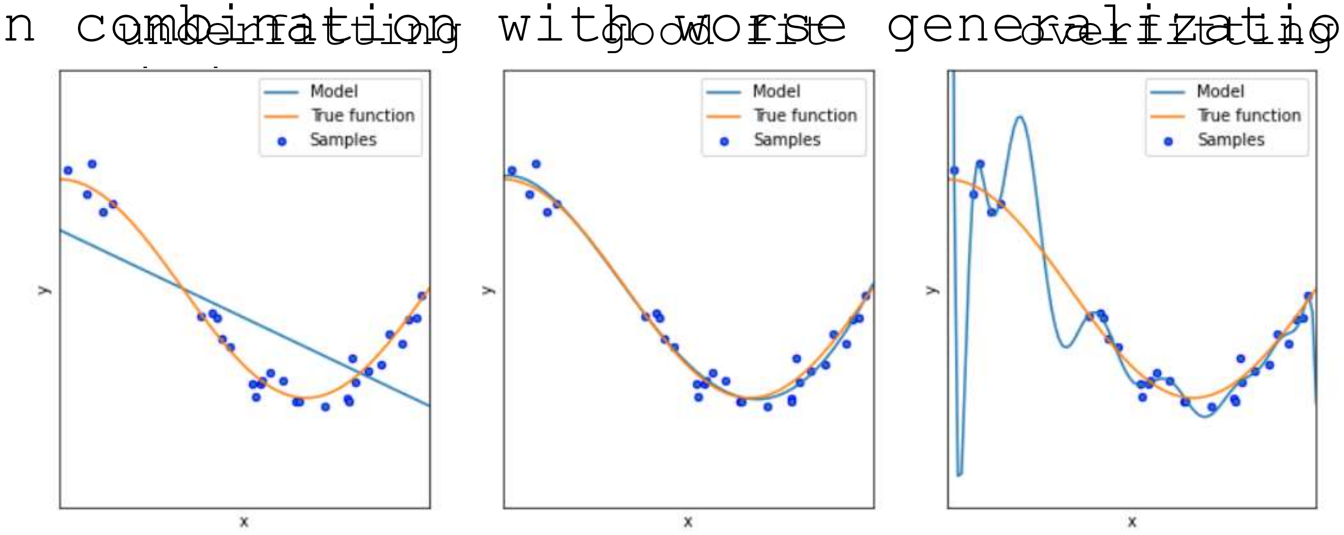
Overfitting

One of the key aspects to deal with when training neural networks.

Overfitted models perform poorly on new data from the same domain.

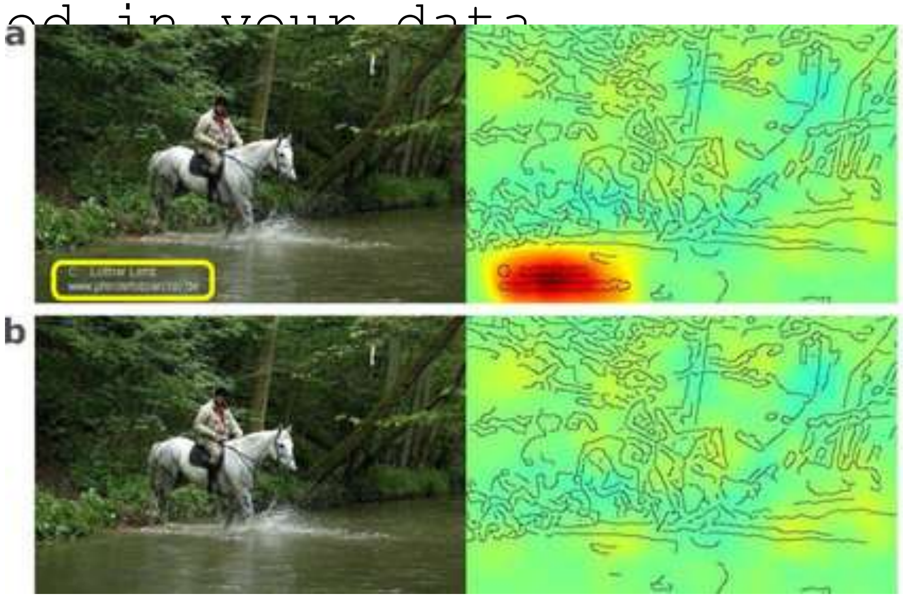
Low/zero training error is not automatically overfitting!

Only in combination with worse generalization as a function of

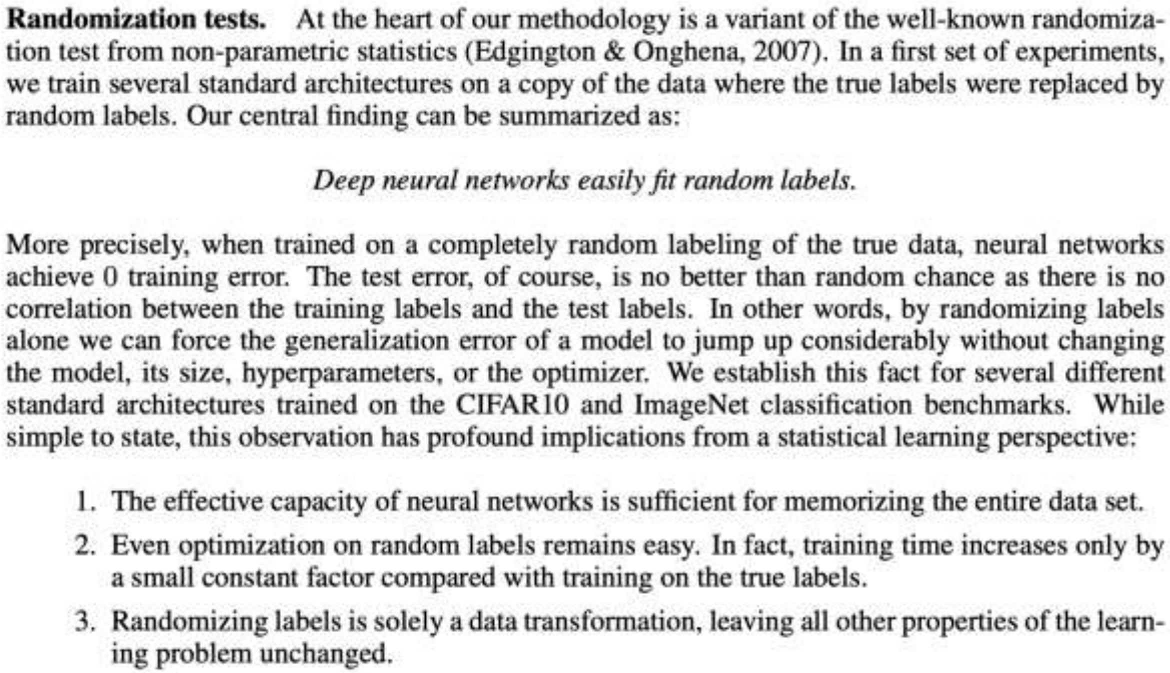


Why would overfitting even happen?

1. Complexity / parameter count >> problem / data.
2. Overfitting especially common when dealing with co-occurrences.
3. Memorization (i.e., learning individual samples instead of their distribution).
4. Silly things you might have missed in your data.



The severity of the overfitting problem

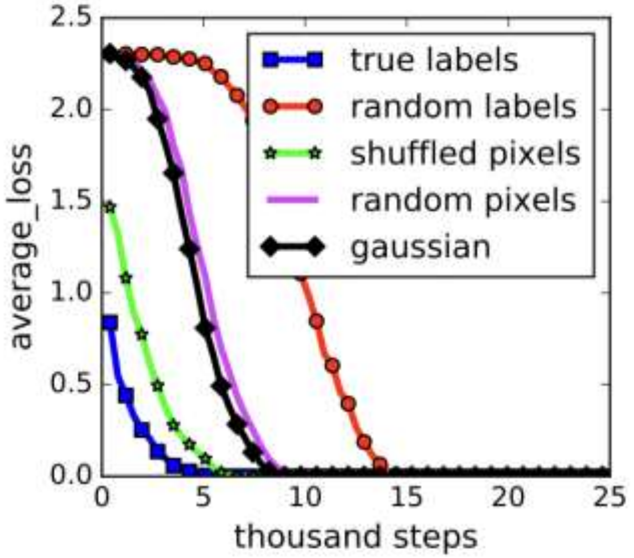


Randomization tests. At the heart of our methodology is a variant of the well-known randomization test from non-parametric statistics (Edgington & Onghena, 2007). In a first set of experiments, we train several standard architectures on a copy of the data where the true labels were replaced by random labels. Our central finding can be summarized as:

Deep neural networks easily fit random labels.

More precisely, when trained on a completely random labeling of the true data, neural networks achieve 0 training error. The test error, of course, is no better than random chance as there is no correlation between the training labels and the test labels. In other words, by randomizing labels alone we can force the generalization error of a model to jump up considerably without changing the model, its size, hyperparameters, or the optimizer. We establish this fact for several different standard architectures trained on the CIFAR10 and ImageNet classification benchmarks. While simple to state, this observation has profound implications from a statistical learning perspective:

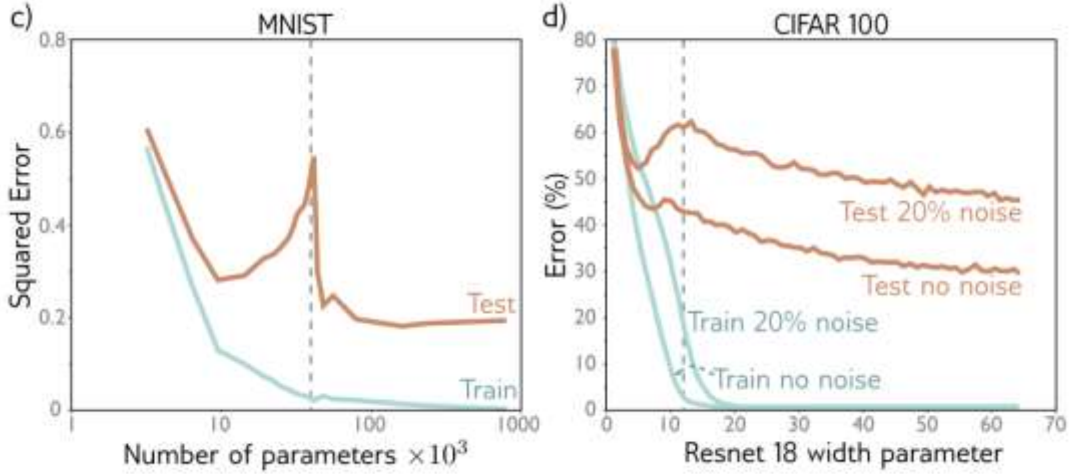
1. The effective capacity of neural networks is sufficient for memorizing the entire data set.
2. Even optimization on random labels remains easy. In fact, training time increases only by a small constant factor compared with training on the true labels.
3. Randomizing labels is solely a data transformation, leaving all other properties of the learning problem unchanged.



(a) learning curves

Understanding deep learning requires rethinking generalization. Zhang et al.

Bias-variance tradeoff – the sequel



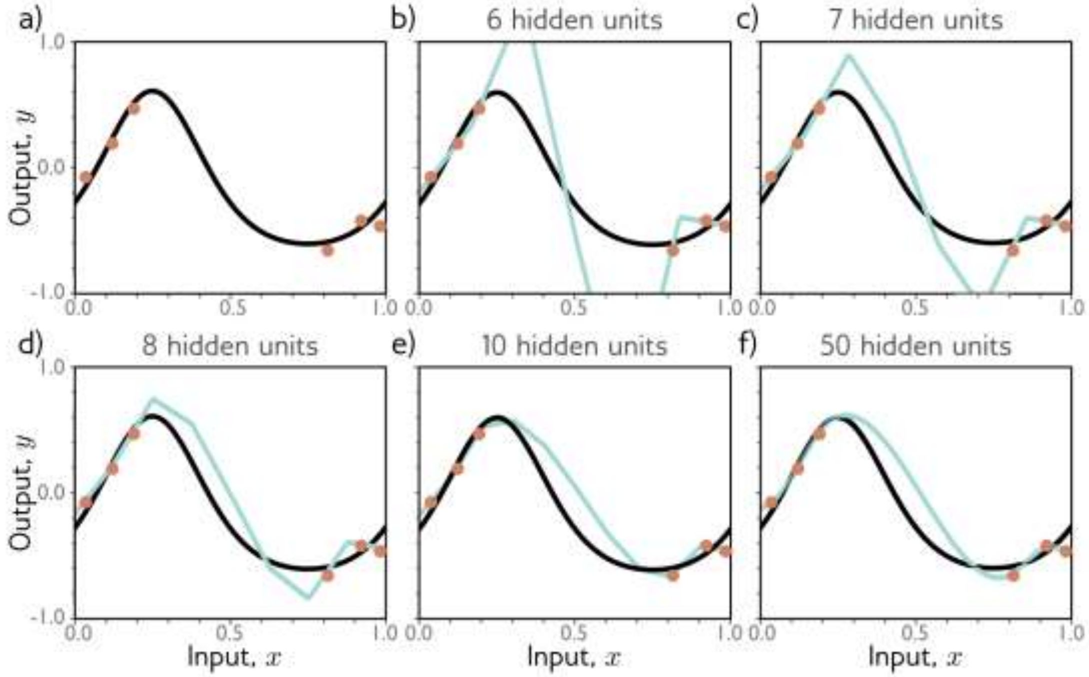
A counter-intuitive finding: when model size > dataset size, error goes down again.

This is double-descent.

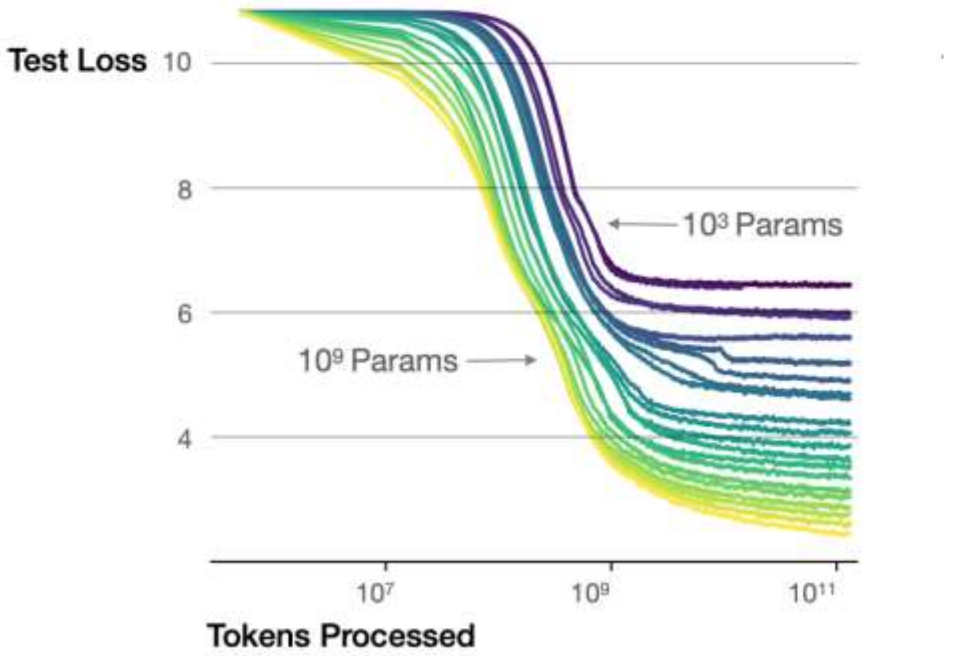
In short: one form of regularization can come from simply using bigger models, in direct conflict with the bias-variance tradeoff.

Belkin, M., Hsu, D., Ma, S., & Mandal, S. (2019) Reconciling modern machine-learning practice and the classical bias–variance trade- off. Proceedings of the National Academy of Sciences

Double descent: smoothness from bigger models



Larger models require **fewer samples** to reach the same performance



Scaling Laws for Neural Language Models.
Kaplan et al. 2020

Double descent: no golden ticket

In practice: simply increasing parameter count does not magically solve all problems.

Double descent only happens under some conditions, not universal.

Solution: fall back on regularization (ubiquitous in all of deep learning, including LLMs).

ℓ_2-regularization

The ℓ_2 regularization is the most common type of all regularization techniques

- Commonly known as *weight decay* or *ridge regression* (in the linear case).

The regularization term Ω is defined as the Euclidean Norm of the weight matrices.

- I.e., simply the sum over all squared weight values of a weight matrix.

$$\frac{1}{2}\sum_l \|w_l\|_2^2$$

ℓ_2 regularization encourages the weight values towards zero.

ℓ_2-regularization

The loss function with ℓ_2-regularization:

$$\mathrm{w}^* \leftarrow \arg\min_w \sum_{(x,y)\subseteq(X,Y)} \mathcal{L}(y, a_L(x; w_{1,\dots,\mathrm{L}})) + \frac{\lambda}{2}\sum_l \|w_l\|_2^2$$

The ℓ_2-regularization is added to the gradient descent update rule

$$w_{t+1} = w_t - \eta_t(\nabla_\theta \mathcal{L} + \lambda w_l) \Longrightarrow$$
$$w_{t+1} = (1 - \lambda\eta_t)w^{(t)} - \eta_t \nabla_\theta \mathcal{L}$$

"Weight decay", because weights get smaller

λ is usually about $10^{-1}, 10^{-2}$

Why would a force to zero values help prevent overfitting?

ℓ_1-regularization

ℓ_1-regularization is one of the most important regularization techniques

$$\mathrm{w}^* \leftarrow \arg\min_w \sum_{(x,y)\subseteq(X,Y)} \mathcal{L}(y, a_L(x; w_{1,\dots,\mathrm{L}})) + \frac{\lambda}{2}\sum\nolimits_l |w_l|$$

Also ℓ_1-regularization is added to the gradient descent update rule

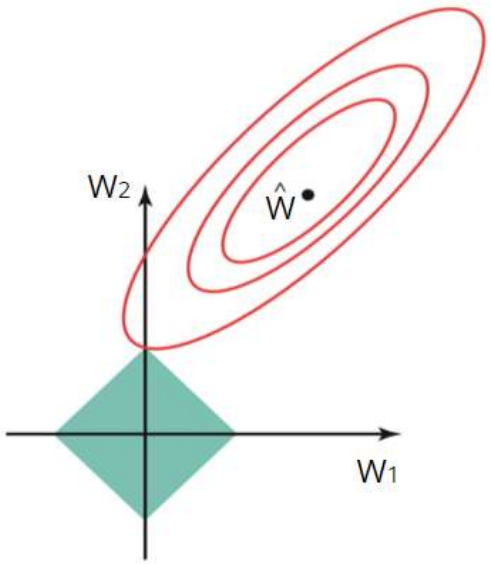
$$w_{t+1} = w_t - \eta_t \left(\nabla_\theta \mathcal{L} + \lambda \frac{w^{(t)}}{\mathrm{sgn}(w^{(t)})} \right)$$

Sign function

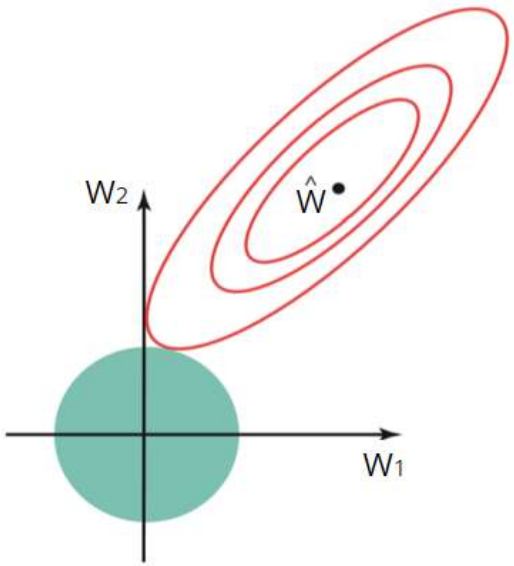
ℓ_1-regularization → sparse weights

- $\lambda \nearrow$ → more weights become 0

Visualizing weight decay



ℓ_1-regularization

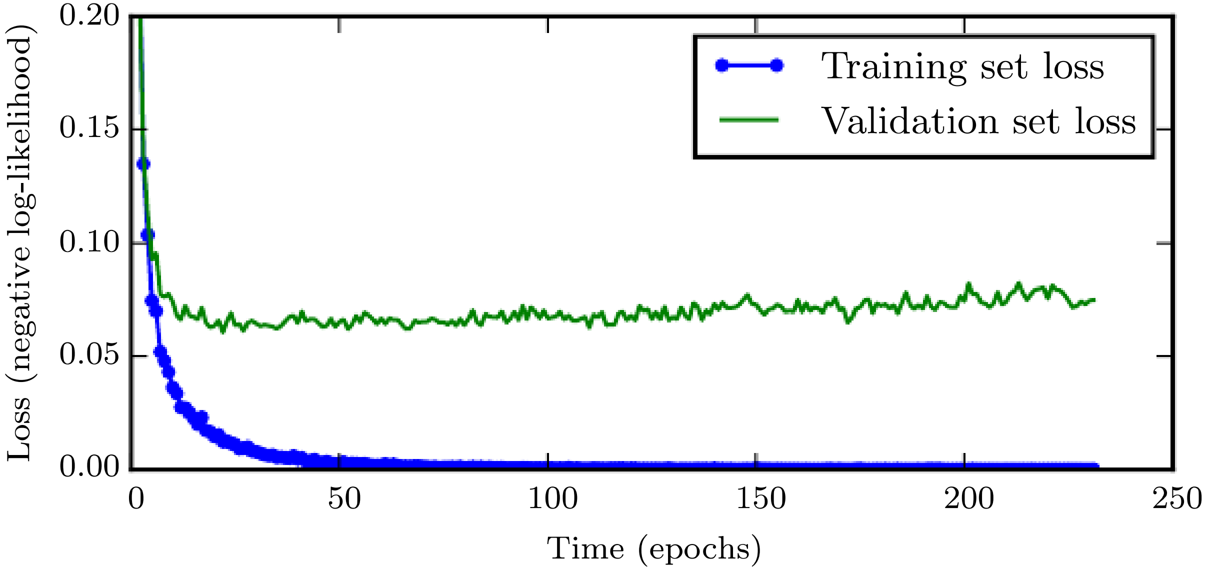


ℓ_2-regularization

Early stopping

Even simpler solution: just stop training.

Test losses tend to increase gradually, avoid by checking with validation set.

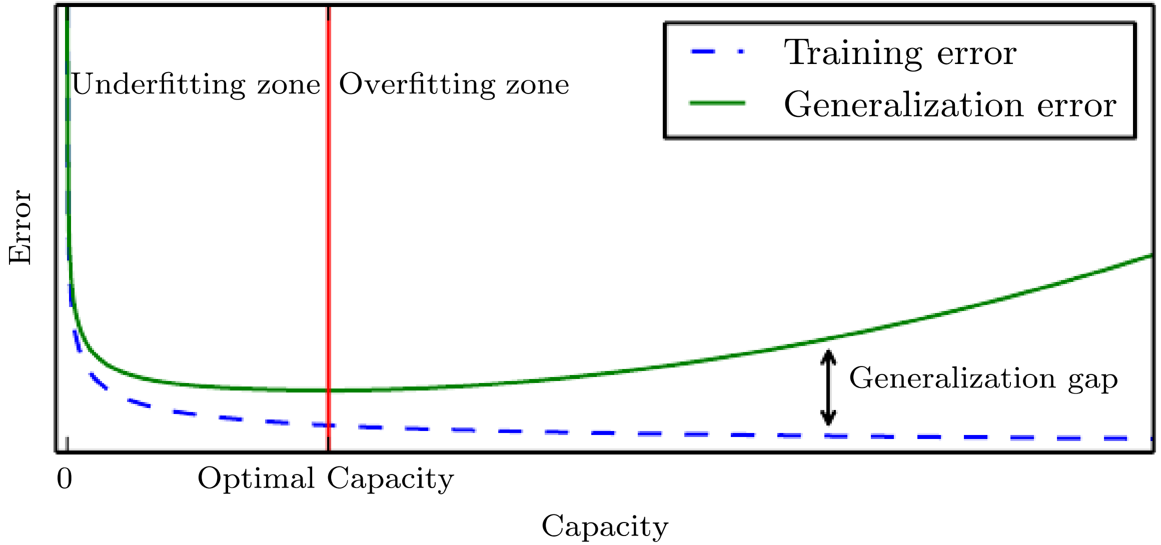


Early stopping as a hyperparameter

All we need to do if find the right epoch to stop training.

Underlying idea: bias-variance progression gradually occurs as a function of #epochs.

Naturally not possible to monitor using the training set, only with inde



DropOut

The co-adaptation observation

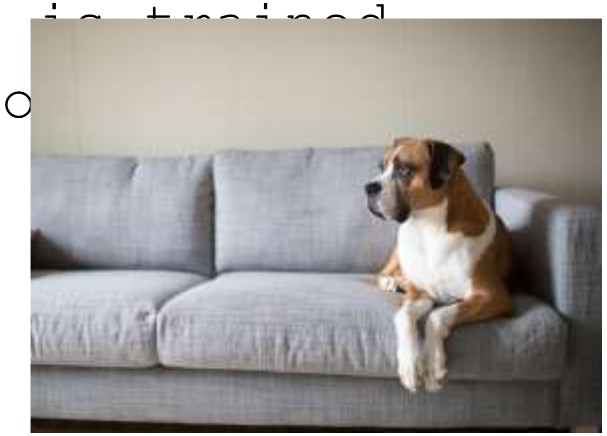
When different hidden units in a neural networks have highly correlated behaviour.

Some of the connections will have more predictive capability than the others.

These powerful connections are learned more while the weaker ones are ignored.

Over many iterations, only a fraction of the node connections is trained.

The rest sto . prevent this.



Implementation of DropOut

During training randomly set activations to 0

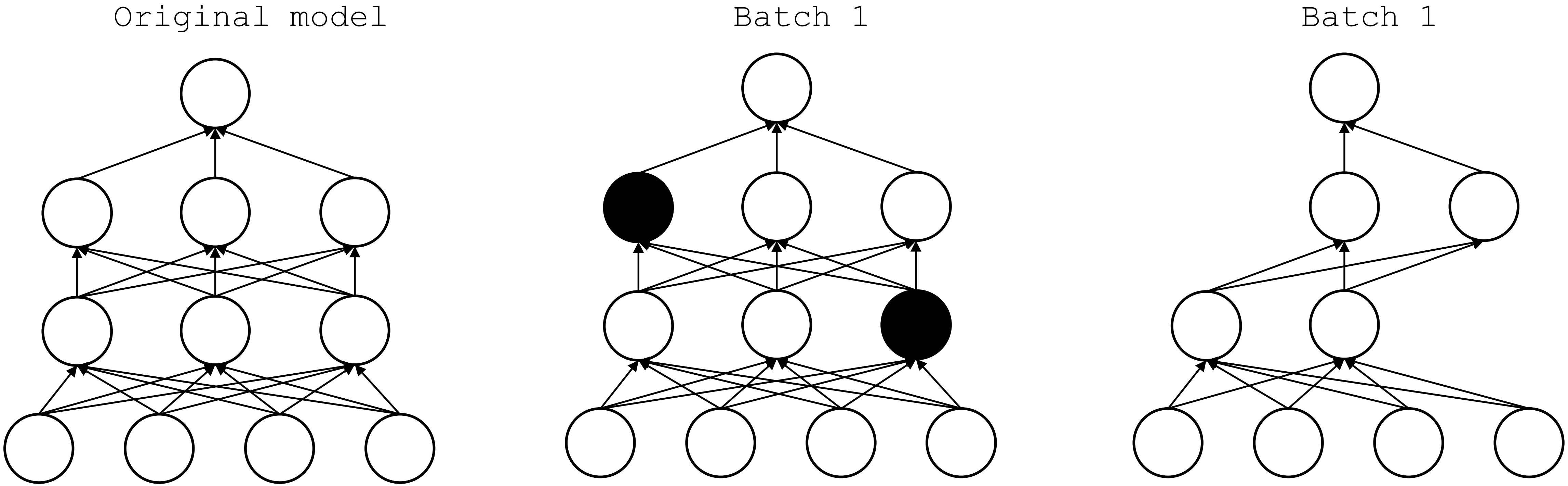
- Neurons sampled at random from a Bernoulli distribution with p (eg, $p = 0.5$)
- Neuron activations reweighted by 1/p

During testing all neurons are used

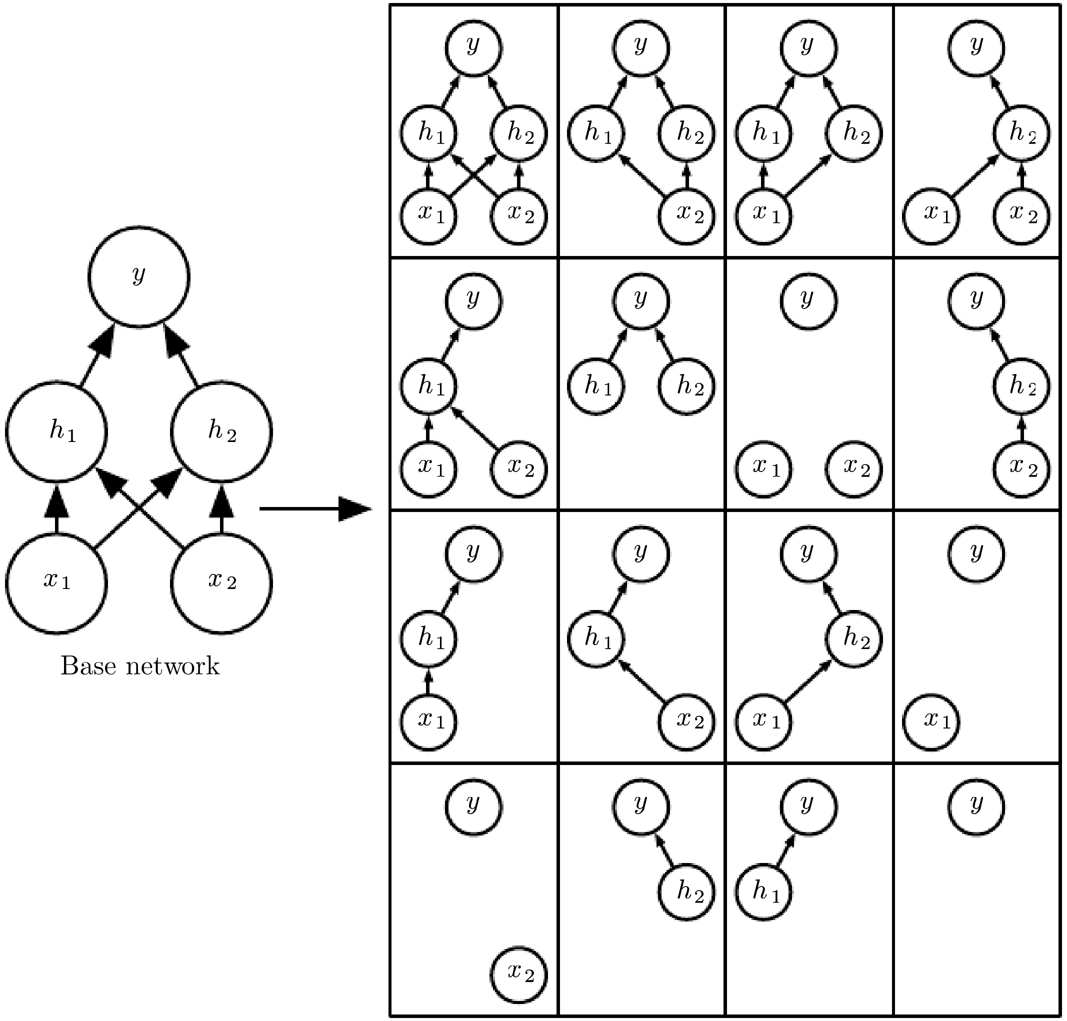
Benefits

- Reduces complex co-adaptations between neurons
- Every neuron becomes more robust
- Decreases overfitting

DropOut: effectively 2^n architectures



DropOut: effectively 2^n architectures



Base network

Ensemble of subnetworks

DropOut: effectively 2^n architectures

Ensembling is a well-known way to improve/regularize machine learning models.

In DropOut, each combination of selected neurons forms its on submodel.

The parameters between the submodels share weights.

During testing, no need to run data on multiple models (unlike assembling).

Setting neurons to zero also breaks co-adaption/co-occurrrences, i.e.:

Network initialization

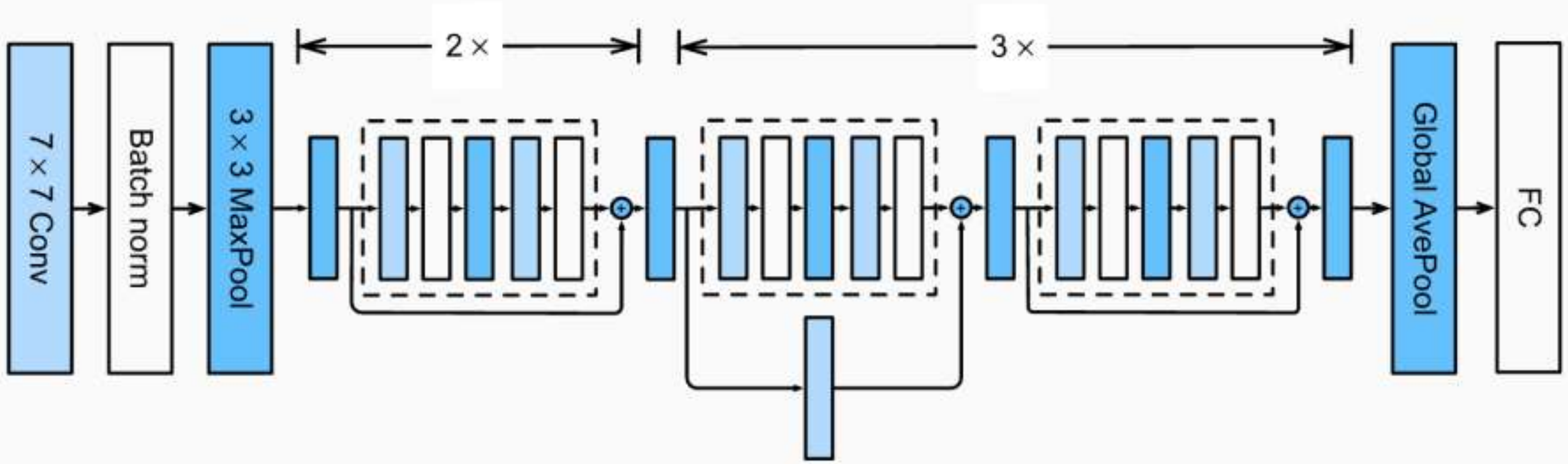


Fig. 8.6.4 The ResNet-18 architecture.

Weight initialization

To prevent layer activation outputs from exploding or vanishing gradients.

Initialize weights correctly and our objective will be achieved in the least time.

Zero Initialization

- Leads to symmetric hidden layers.
- Makes your network no better than a linear model.
- Setting biases to 0 will not create any problems.

Random Initialization

- Breaks symmetry.
- Prevents neurons from learning the same features.

Random how?

Weights initialized **to preserve the variance** of the activations

- During the forward and backward computations.
- We want similar input and output variance because of modularity.

Weights must be initialized to be different from one another

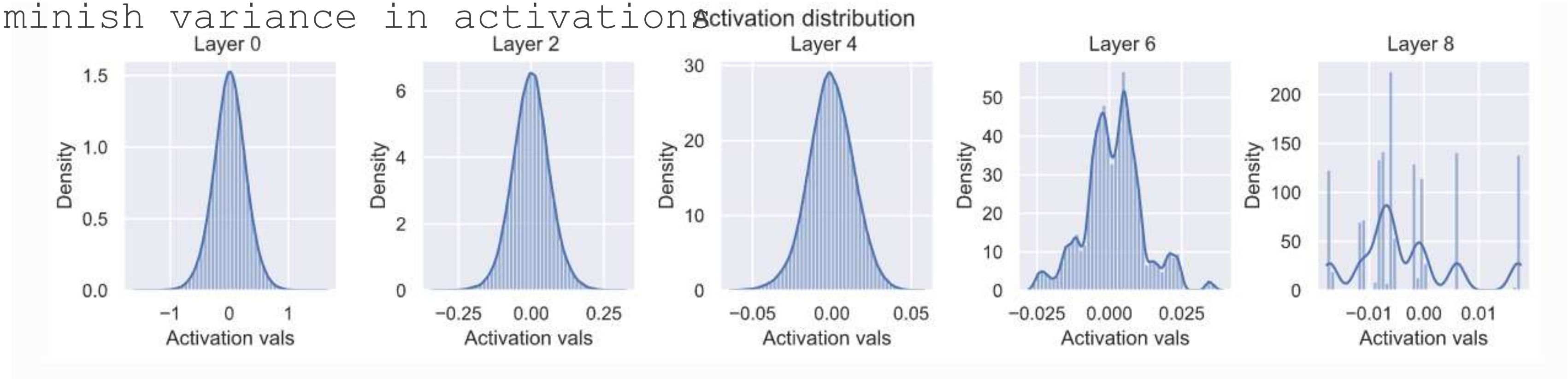
- Don’t give same values to all weights (like all 0).
- All neurons (in one layer) generate same gradient → no learning.

Initialization depends on:

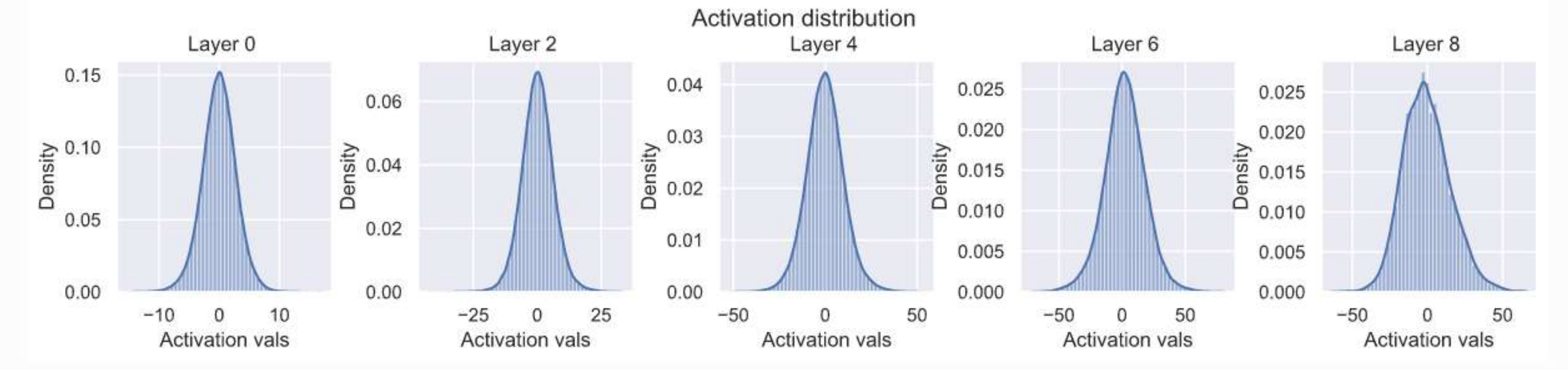
- non-linearities.
- data normalization

Bad initialization will come back to haunt you

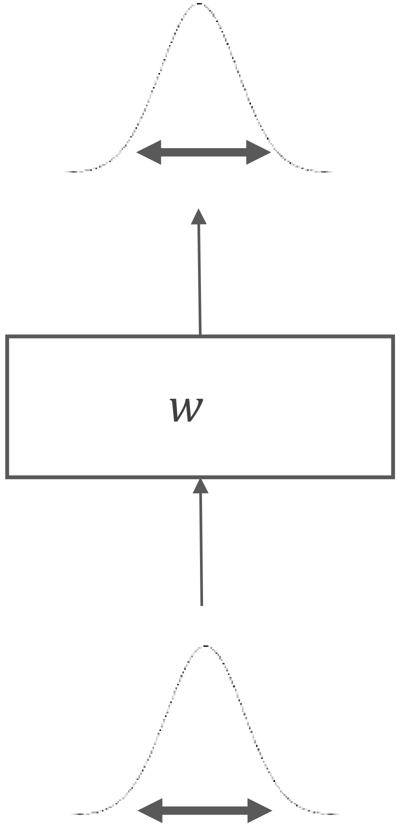
Initializing weights in every layer with same constant variance → can diminish variance in activations



Initializing weights in every layer with increasing variance → can explode the variance in activations



Preserving variance



For x and y independent

$$var(xy) = \mathbb{E}[x]^2 var(y) + \mathbb{E}[y]^2 var(x) + var(x)var(y)$$

For $a = wx \Rightarrow var(a) = var(\sum_i w_i x_i) = \sum_i var(w_i x_i) \approx d \cdot var(w_i x_i)$

$$var(w_i x_i) = \mathbb{E}[x_i]^2 var(w_i) + \mathbb{E}[w_i]^2 var(x_i) + var(x_i)var(w_i)$$

$$= var(x_i)var(w_i)$$

Because we assume that x_i, w_i are unit Gaussians $\rightarrow \mathbb{E}[x_i] = \mathbb{E}[w_i] = 0$

So, the variance in our activation $var(a) \approx d \cdot var(x_i)var(w_i)$

Understanding the difficulty of training deep feedforward neural

Preserving variance

Since we want the same input and output variance

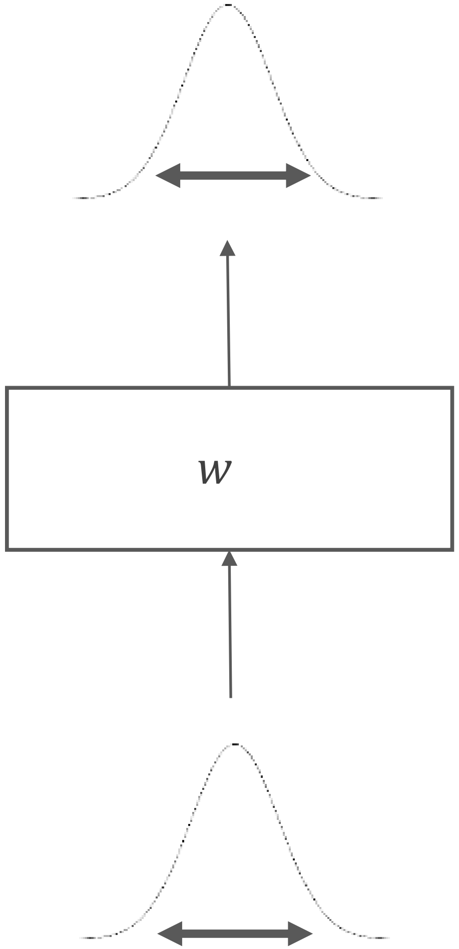
$$var(a) = d \cdot var(x_i)var(w_i) \Rightarrow var(w_i) = \frac{1}{d}$$

Draw random weights from

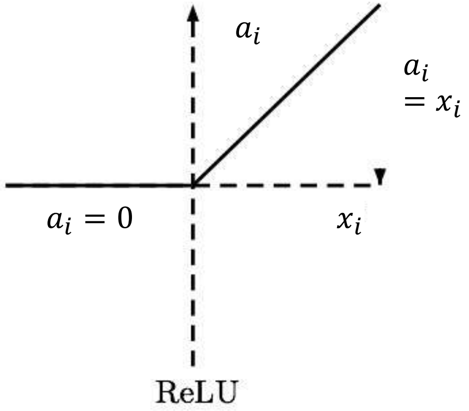
$$w \sim N(0, 1/d)$$

where $\boldsymbol{d}$ is the number of input

variables to the layer



Kaiming Initialization



ReLU's return 0 half of the time: $\mathbb{E}[w_i] = 0$ but $\mathbb{E}[x_i] \neq$

$$var(w_i x_i) = var(w_i)(\mathbb{E}[x_i]^2 + var(x_i))$$

$$= var(w_i)\mathbb{E}\left[x_i^2\right] \quad (var(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2)$$

$$\mathbb{E}\left[x_i^2\right] = \int_{-\infty}^{\infty} x_i^2 p(x_i) dx_i = \int_{-\infty}^{\infty} \max(0, a_i)^2\, p(a_i) da_i = \int_{0}^{\infty} a_i^2 p(a_i) da_i$$

$$= 0.5 \int_{-\infty}^{\infty} a_i^2 p(a_i) da_i = 0.5 \cdot \mathbb{E}\left[a_i^2\right] = 0.5 \cdot var(a_i)$$

Draw random weights from $w \sim N(0, 2/d)$ – **Kaiming Initialization**

Xavier initialization

For tanh: initialize weights from $\mathrm{U}\left[-\sqrt{\frac{6}{d_{l-1}+d_l}}, \sqrt{\frac{6}{d_{l-1}+d_l}}\right]$

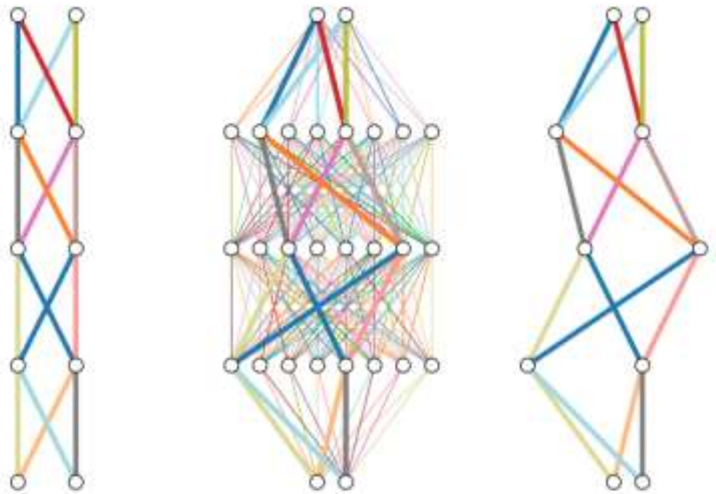
d_{l-1} is the number of input variables to the tanh layer and d_l is the number of the output variables

For a sigmoid $\mathrm{U}\left[-4\cdot\sqrt{\frac{6}{d_{l-1}+d_l}}, 4\cdot\sqrt{\frac{6}{d_{l-1}+d_l}}\right]$

Random networks are already great deep learners

What's Hidden in a Randomly Weighted Neural Network?
Ramanujan et al. 2019

Hidden in a ***randomly weighted Wide Re***
we find a subnetwork (with random wei
that is smaller than, but matches the
performance of a ResNet-34 trained or
ImageNet *[4]. Not only do these "untr*
subnetworks" exist, but we provide ar
to effectively find them.



A neural network τ which achieves good performance

Randomly initialized neural network N

A subnetwork τ' of N

Figure 1. If a neural network with random weights (center) is sufficiently overparameterized, it will contain a subnetwork (right) that perform as well as a trained neural network (left) with the same number of parameters.

Break

Data augmentation

The best way to make a machine learning model generalize better is to train it on more data. (see: ”The unreasonable effectiveness of data”)

- Data* is limited in practice
- One way is to create fake data – *Data Augmentation***

Your neural network is only as good as the data you feed it.

By performing augmentation, we can prevent neural networks from learning or memorizing irrelevant patterns, essentially boosting overall performance.

* Labeled data
** Not that trivial. Augmentations are more than just fake data!

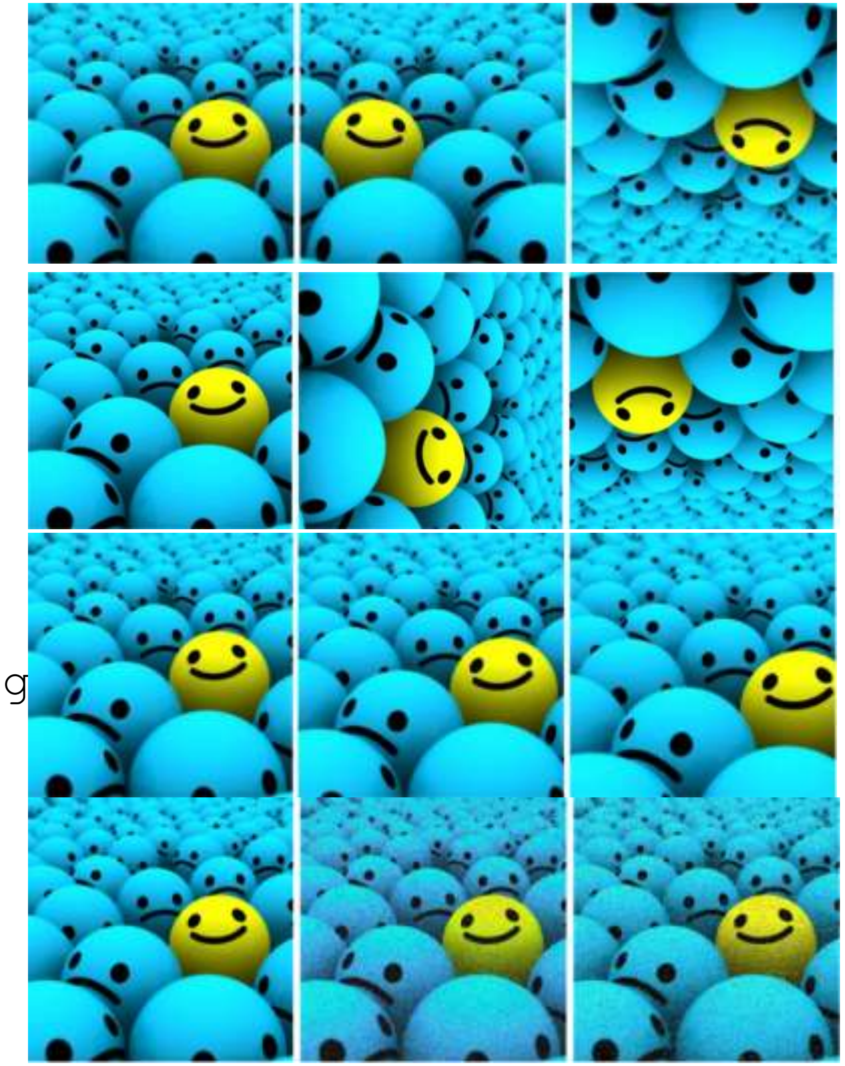
Examples of data augmentation

In computer vision

- Flip
- Rotation
- Scale
- Crop
- Translation
- Gaussian noise

Be aware of label chang

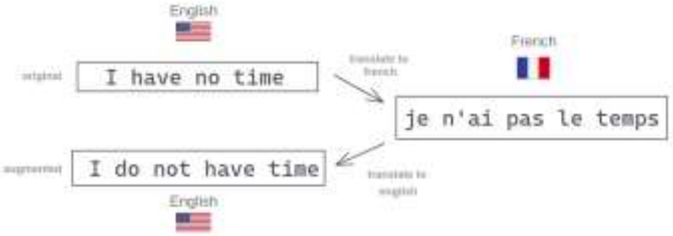
- "b" and "d"
- "6" and "9"



[Link]

In NLP

Backtranslation



I have no time

je n'ai pas le temps

I do not have time

Synonym replacement

Random insertion

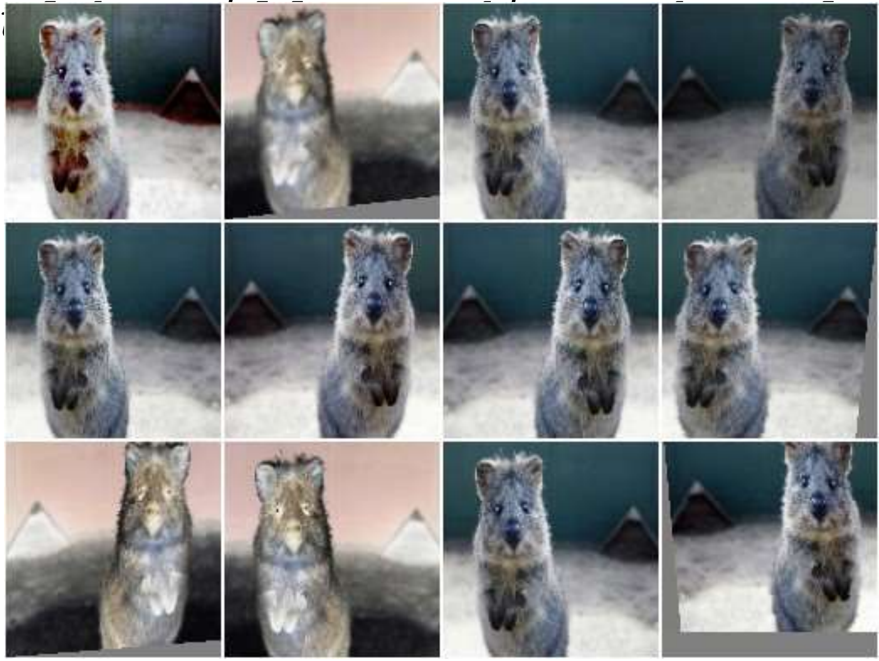
Random deletion

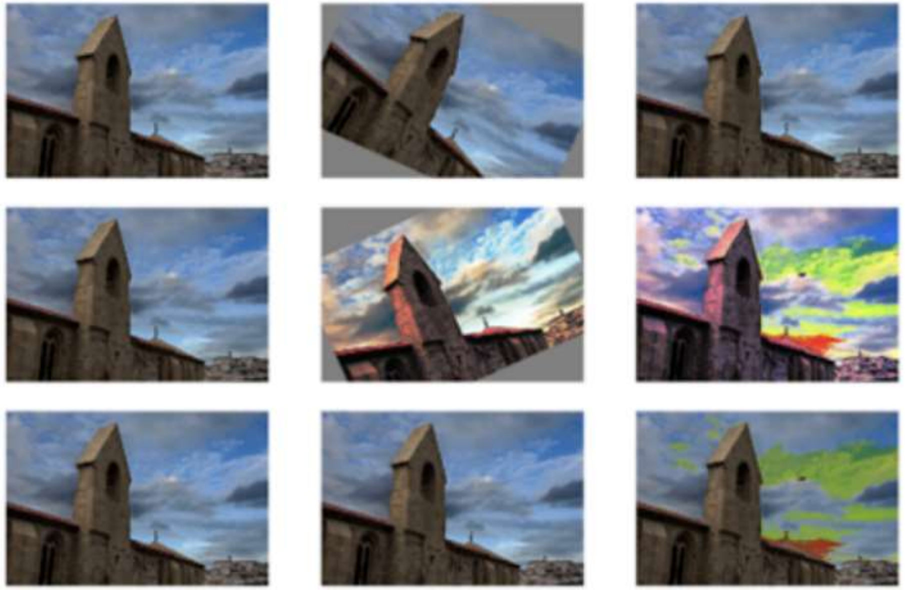
Random swapping

Data augmentation = pre-defined invariance

Essentially a form of injecting prior knowledge to instill **invariance.**

A dog flipped vertically is still a dog, so a network sho bel





Other data augmentations

Noise robustness

- Adding noise to weights – uncertaint
- Adding noise to outputs - label smoc

Semi or self-supervised learnir

- Introducing a particular form of prior belief about the solution.

Multi-task learning

- Shared input and parameters – improv statistical strength.
- Requires statistical relationship between tasks.

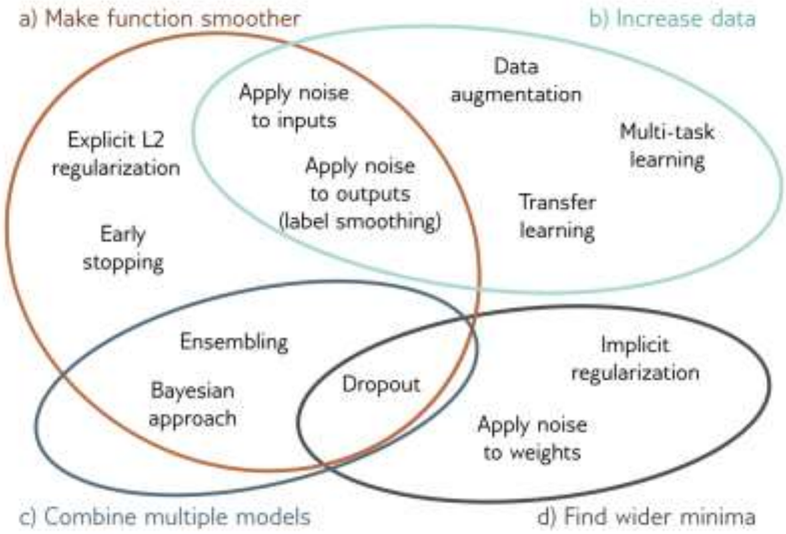
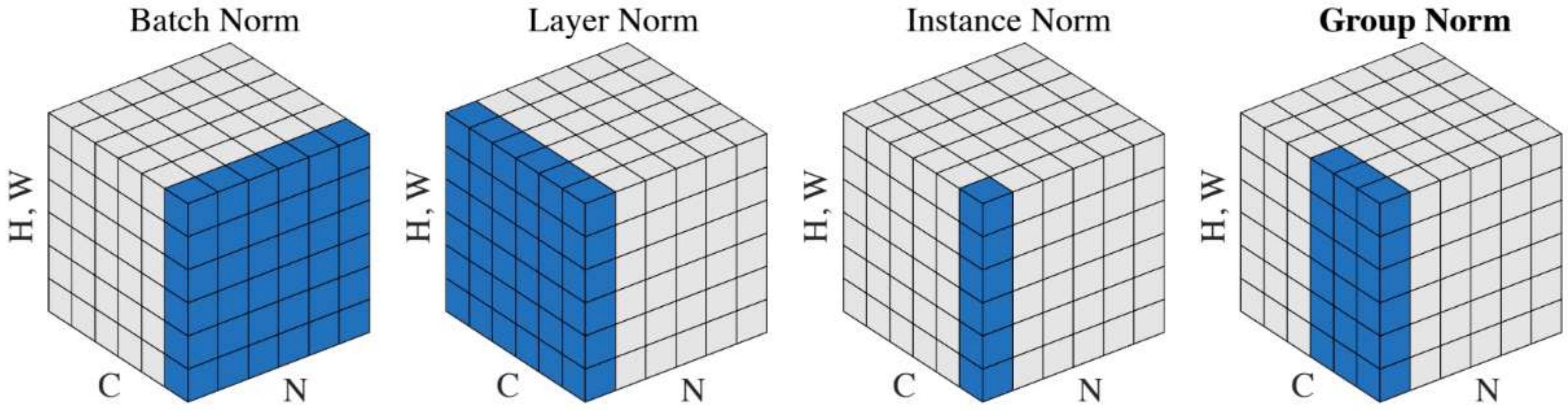


Figure 9.14 Regularization methods. The regularization methods discussed in this chapter aim to improve generalization by one of four mechanisms. a) Some methods aim to make the modeled function smoother. b) Other methods increase the effective amount of data. c) The third group of methods combine multiple models and hence mitigate against uncertainty in the fitting process. d) Finally, the fourth group of methods encourages the training process to converge to a wide minimum where small errors in the estimated parameters are less important.

UDL – Chapter 9

Normalization

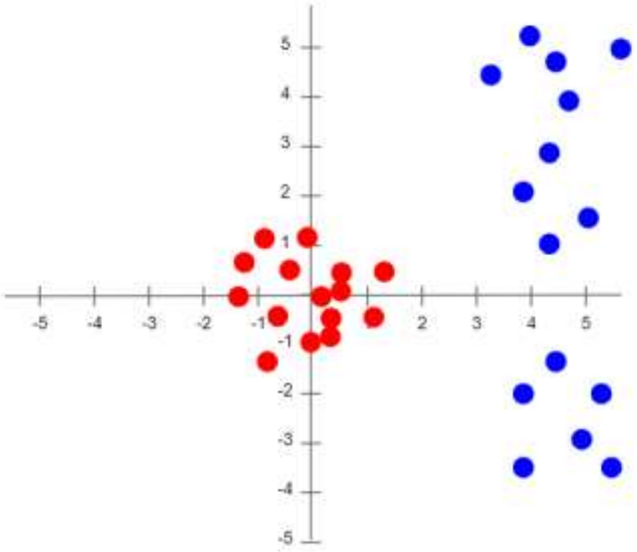


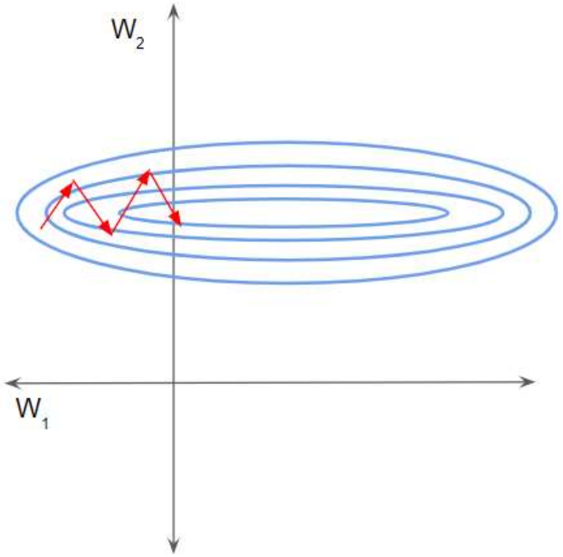
Normalization as data preprocessing

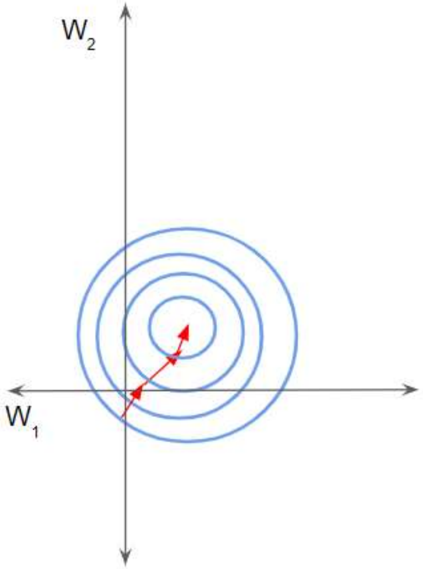
Data pre-processing brings numerical data to a common scale without distorting shape.

The reason is partly to ensure that our model can generalize appropriately.

This ensures that all the feature values are now on the same



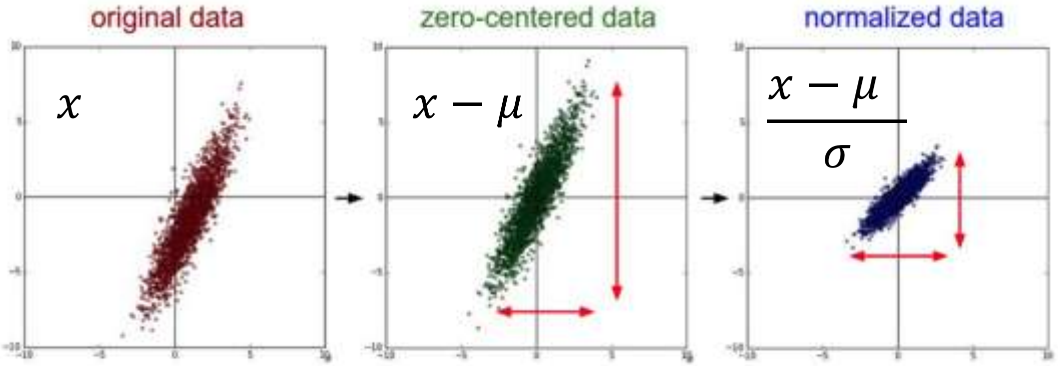




Must in deep learning: normalizing input data

Transforming the input to zero-mean, unit variance

- Assume: Input variables follow a Gaussian distribution (roughly)
- Subtract input by the mean
- Optionally, divide by the standard deviation $N(\mu,\sigma^2) \rightarrow N(0,1)$



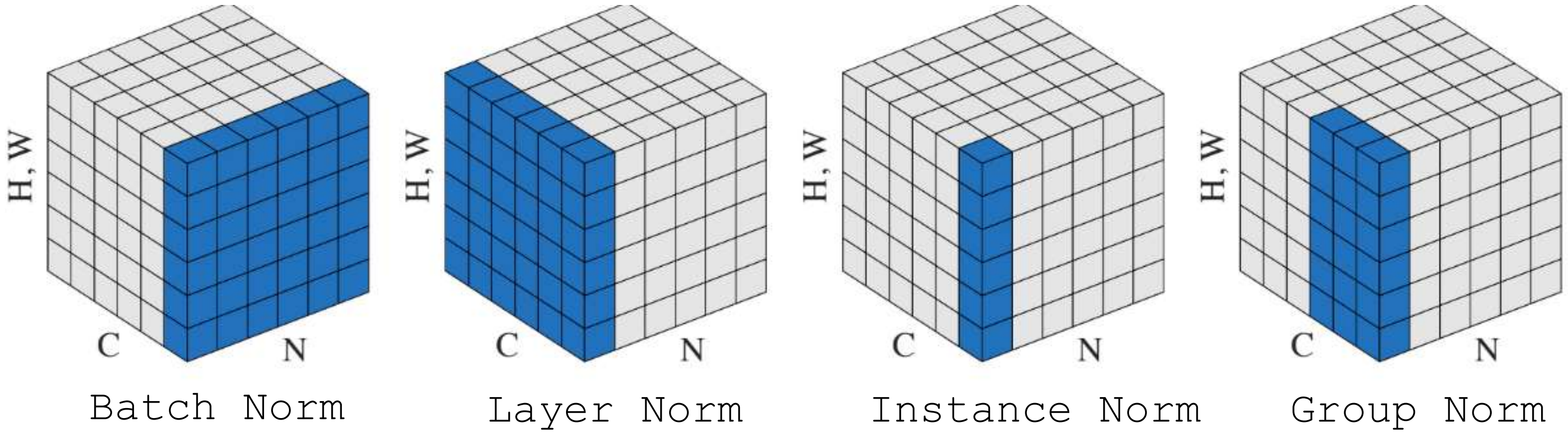
Known by all practitioners:

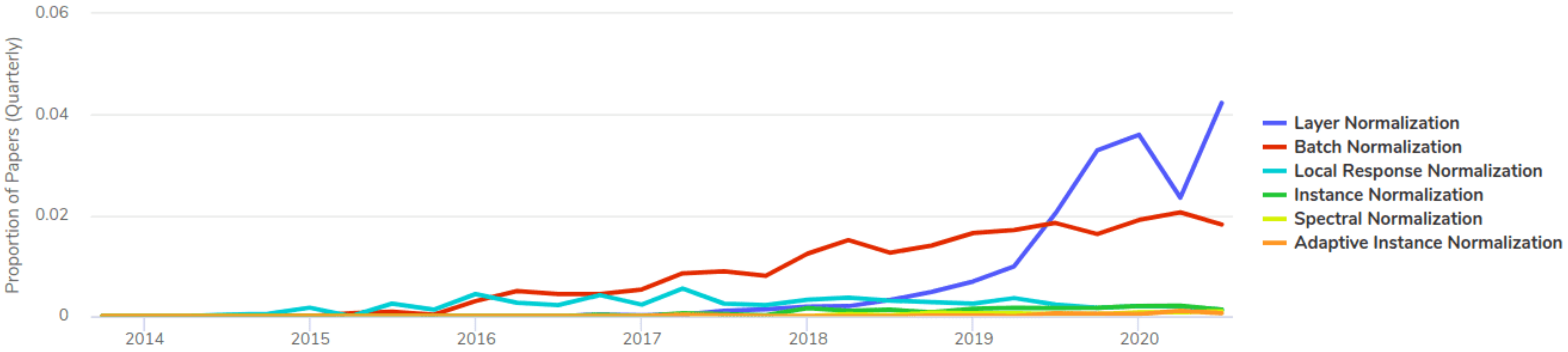
ImageNet: mean = [0.485, 0.456, 0.406] and std = [0.229,

Picture credit: Stanford

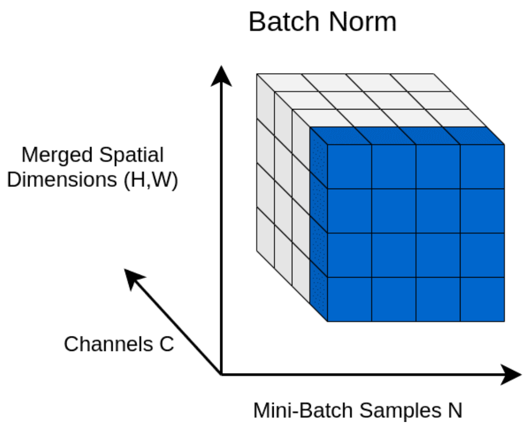
Normalizing intermediate layers

- Batch normalizatio
- Layer normalizatio
- Instance normaliza
- Group normalizatio
- Weight normalizati





Batch normalization



Normalize the layer inputs with batch normalization

Normalize $a_l \sim N(0,1)$

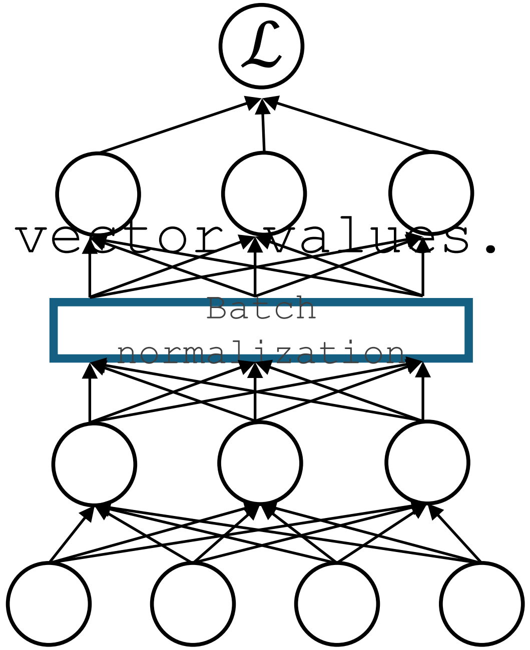
Followed by affine transformation $a_l \leftarrow \gamma a_l + \beta$

The parameters γ and β are trainable

Used for re-scaling (γ) and shifting (β) of the vector values.

Ensure the optimal values of γ and β are used.

Enable the accurate normalization of each batch.



Batchnorm algorithm

$\mu_j \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_{ij}$ [compute mini-batch mean]

$\sigma_j^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}\left(x_{ij} - \mu_j\right)^2$ [compute mini-batch variance]

$\hat{x}_{ij} \leftarrow \frac{x_{ij}-\mu_j}{\sqrt{\sigma_j^2+\varepsilon}}$ [normalize input]

$\hat{x}_{ij} \leftarrow \gamma\hat{x}_{ij} + \beta$ [scale and shift input]

Trainable parameters

i runs over mini-batch samples, j over the feature dimensions

Batchnorm at test time

How do we ship the Batch Norm layer after training?

- We might not have batches at test time
- Batches are random? -> not reproducible

Usually: keep a moving average of the mean and variance during training

- Plug them in at test time
- In the limit, the moving average of mini-batch statistics approaches the batch statistics

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \varepsilon}}$$

$$\widehat{y}_i \leftarrow \gamma \widehat{x}_i + \beta$$

Benefits of batchnorm

Networks train faster

Allows higher learning rates

Makes more activation functions viable

Reduces overfitting

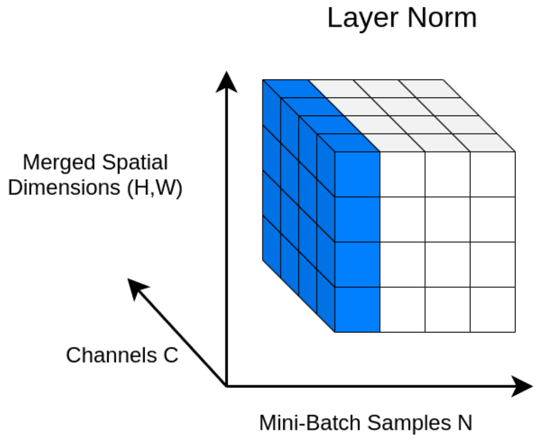
Simplifies the creation of deeper networks

May give better results overall

Drawbacks of batchnorm

- Requires large mini-batches
  - Cannot work with mini-batch of size 1 ($\sigma = 0$)
  - Performance is sensitive to the batch size
  - Memory intense, all the batch statistics must be stored in the layer.
  - Discrepancy between training and test data
  - Breaks the independence between training examples in the minibatch
  - Not applicable to online learning
- Awkward to use with recurrent neural networks
  - Must interleave it between recurrent layers
  - Also, store statistics per time step
- Often the reason for bugs

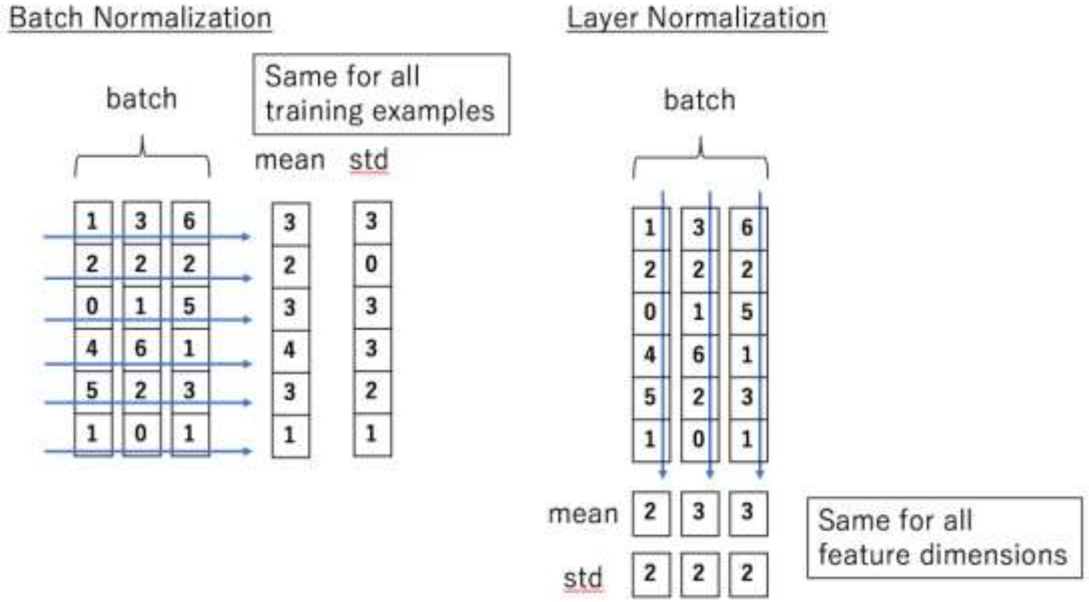
Layer normalization



The statistics (mean and variance) are computed across all channels and spatial dimensions.

The statistics are independent of the batch.

This layer was initially introduced to handle vectors (mostly the RNN outputs).


batch
Same for all training examples
batch
Same for all feature dimensions

Batch Normalization

batch			mean	std
1	3	6	3	3
2	2	2	2	0
0	1	5	3	3
4	6	1	4	3
5	2	3	3	2
1	0	1	1	1

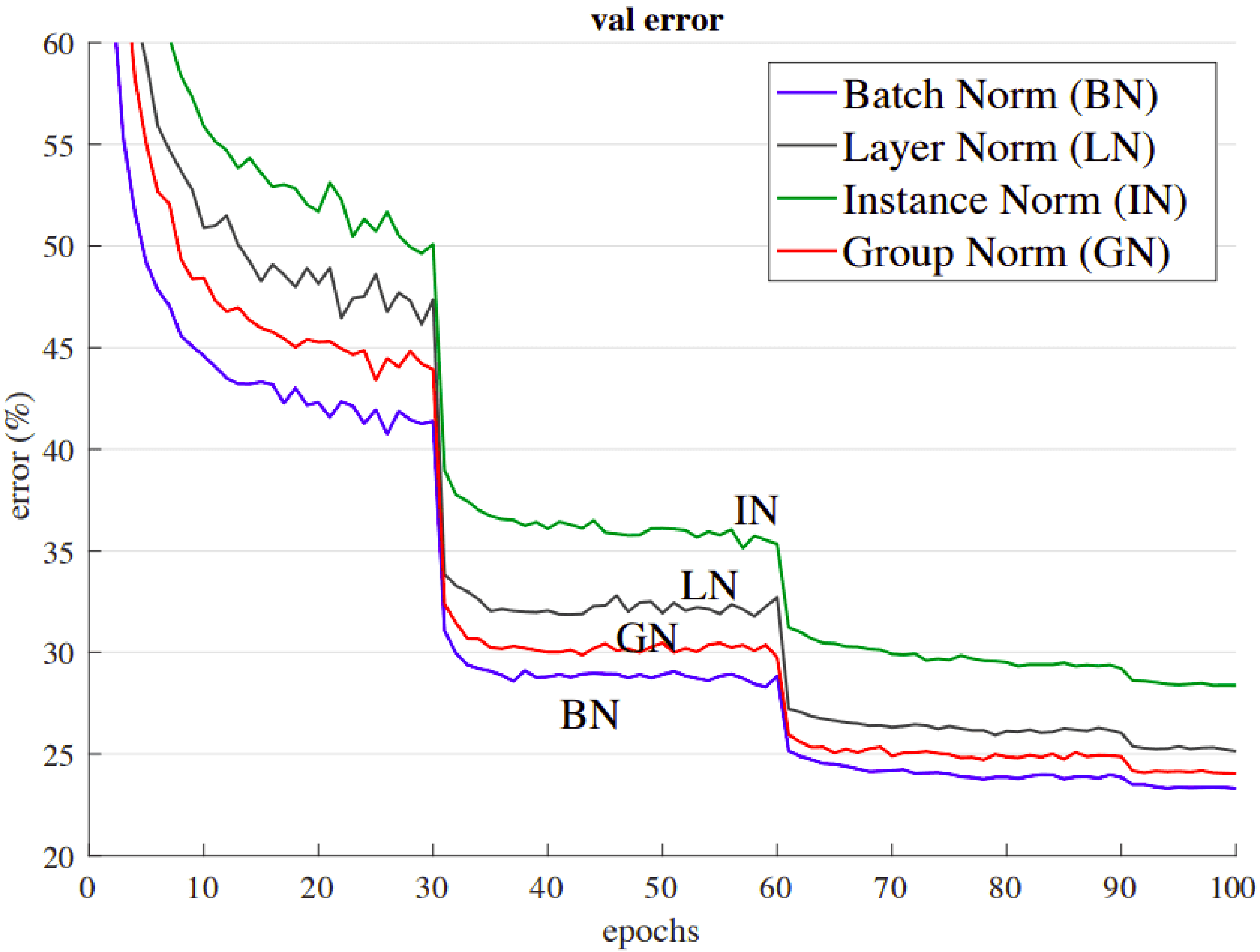
Layer Normalization

	batch		
	1	3	6
	2	2	2
	0	1	5
	4	6	1
	5	2	3
	1	0	1
mean	2	3	3
std	2	2	2

Layer normalization performs the same computation at training and test times

Layer Normalization, Ba, Kiros, Hinton, 2016

Comparing different normalizations



Paper list for the interested reader

Ioffe, S., & Szegedy, C. (2015).Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

Salimans, T., & Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In Advances in neural information processing systems (pp. 901-909).

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv preprint arXiv:1607.06450.

Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2016). Instance normalization: The missing ingredient for fast stylization. arXiv preprint arXiv:1607.08022.

Wu, Y., & He, K. (2018). Group normalization. In Proceedings of the European conference on computer vision (ECCV) (pp. 3-19).

Zhang, H., Dana, K., Shi, J., Zhang, Z., Wang, X., Tyagi, A., & Agrawal, A. (2018). Context encoding for semantic segmentation. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (pp. 7151-7160).

Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization?. In Advances in Neural Information Processing Systems (pp. 2483-2493).

Dumoulin, V., Shlens, J., & Kudlur, M. (2016). A learned representation for artistic style. arXiv preprint arXiv:1610.07629.

Park, T., Liu, M. Y., Wang, T. C., & Zhu, J. Y. (2019). Semantic image synthesis with spatially-adaptive normalization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 2337-2346).

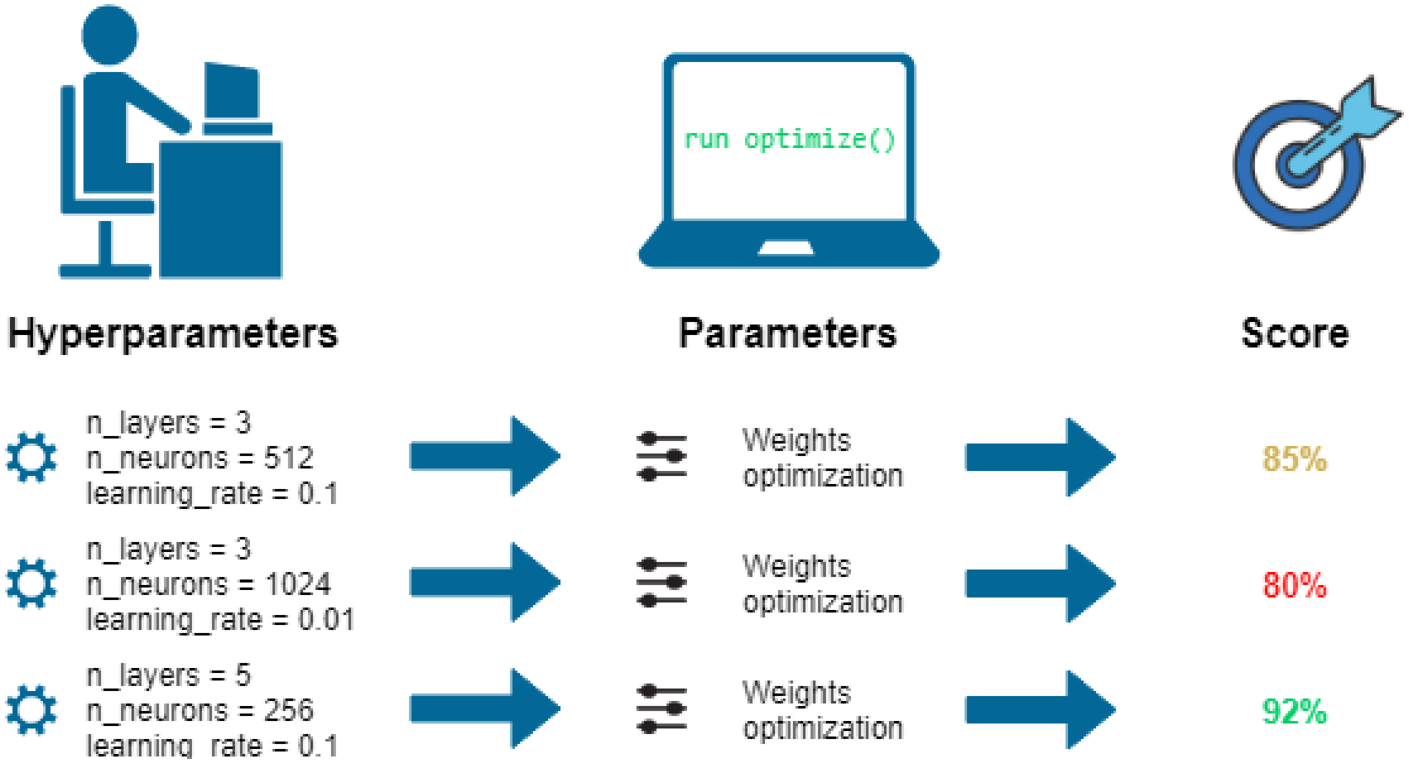
Huang, X., & Belongie, S. (2017). Arbitrary style transfer in real-time with adaptive instance normalization. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1501-1510).

Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., & Houlsby, N. (2019). Big transfer (BiT): General visual representation learning. arXiv preprint arXiv:1912.11370.

Qiao, S., Wang, H., Liu, C., Shen, W., & Yuille, A. (2019). Weight standardization. arXiv preprint arXiv:1903.10520.

Hyperparameters

All values that cannot be tuned through backprop. Hyperparameter tuning most important part of daily deep learning life.



Hyperparameters	Parameters	Score

n_layers = 3
n_neurons = 512
learning_rate = 0.1
	Weights optimization	85%

n_layers = 3
n_neurons = 1024
learning_rate = 0.01
	Weights optimization	80%

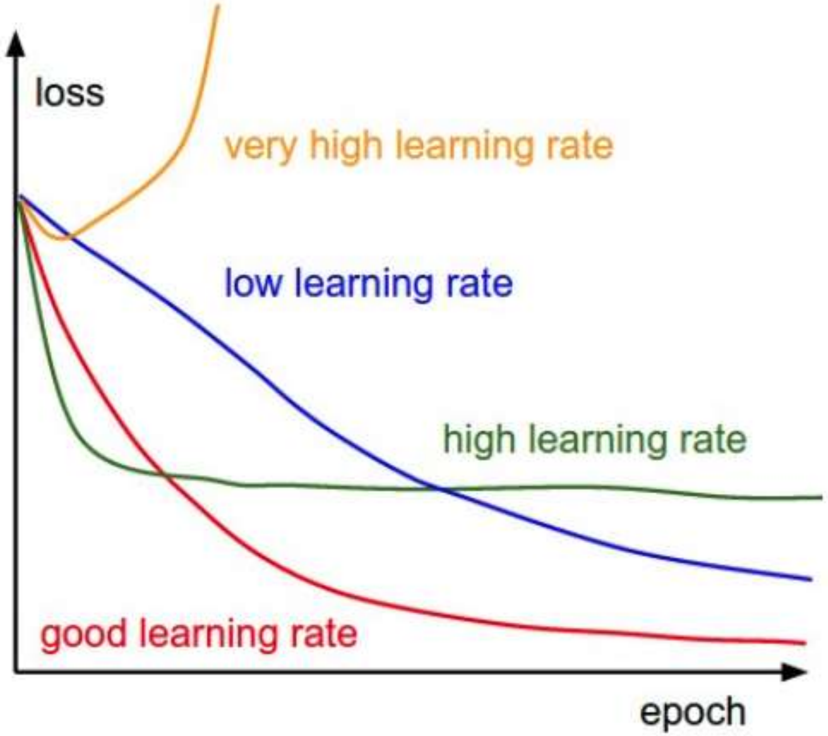
n_layers = 5
n_neurons = 256
learning_rate = 0.1
	Weights optimization	92%

Learning rates in practice

Try several log-spaced values $10^{-1}, 10^{-2}, 10^{-3}, \ldots$ on a smaller set.

Then, you narrow it down from there around where you get the lowest **validation** error.

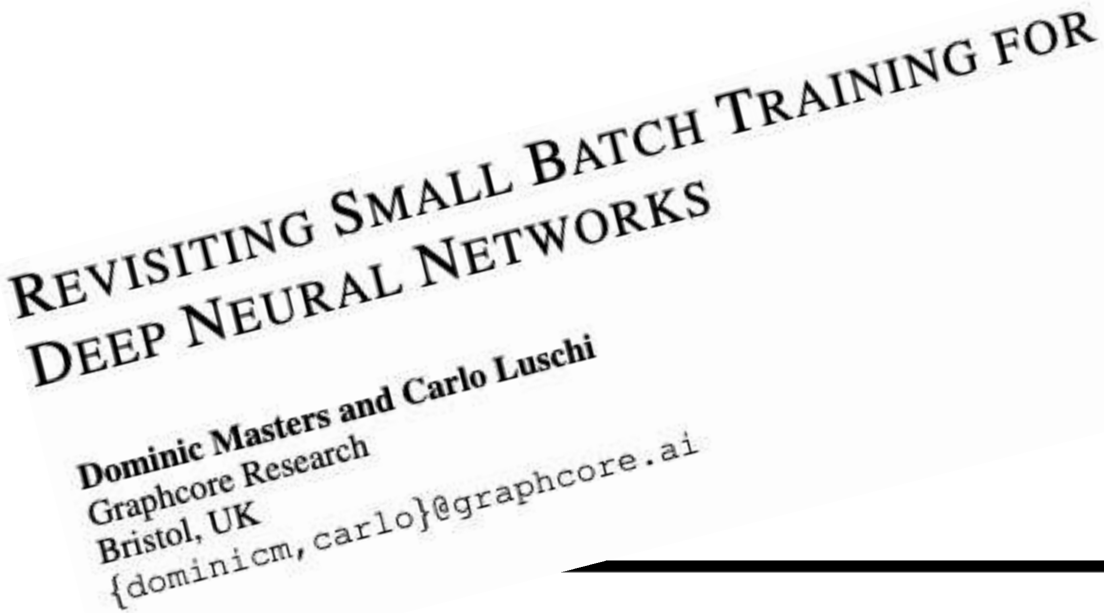
You will learn to become and expert in learning rate pattern recognition over t



Picture credi

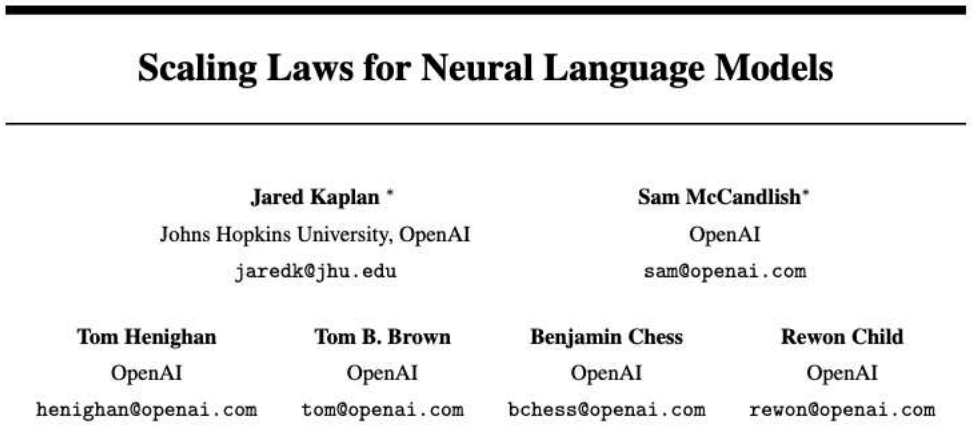
Stanford Cou

Batch siz



Revisiting Small Batch Training for Deep Neural Networks

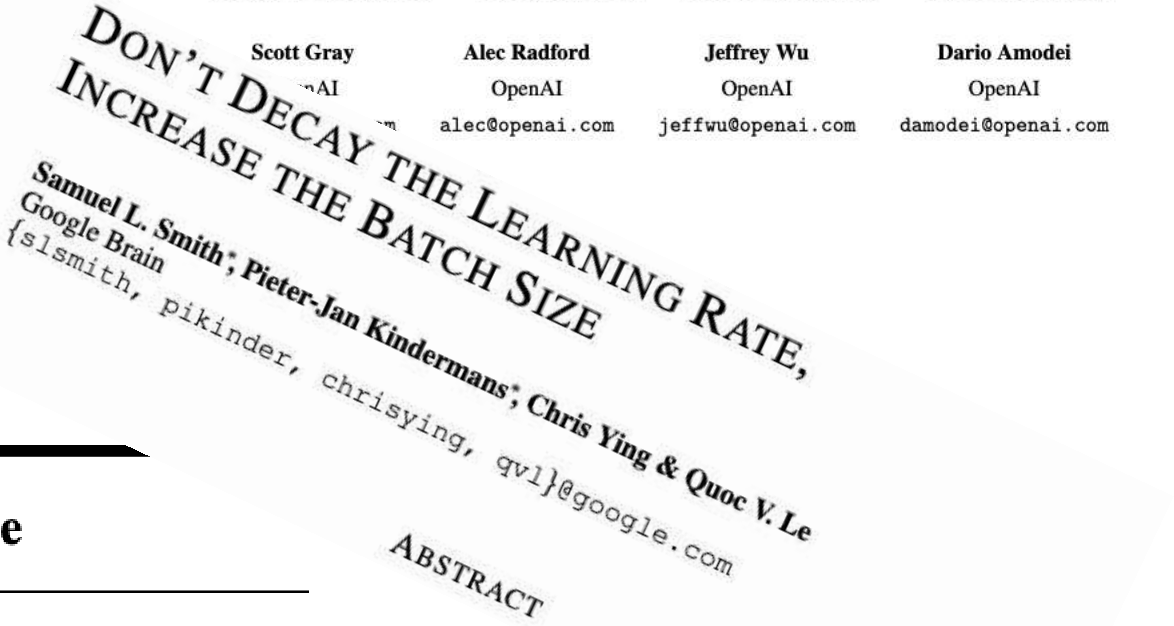
Dominic Masters and Carlo Luschi
Graphcore Research
Bristol, UK
{dominicm, carlo}@graphcore.ai



Scaling Laws for Neural Language Models

Jared Kaplan *	Sam McCandlish*
Johns Hopkins University, OpenAI	OpenAI
jaredk@jhu.edu	sam@openai.com

Tom Henighan	Tom B. Brown	Benjamin Chess	Rewon Child
OpenAI	OpenAI	OpenAI	OpenAI
henighan@openai.com	tom@openai.com	bchess@openai.com	rewon@openai.com
Scott Gray	**Alec Radford**	**Jeffrey Wu**	**Dario Amodei**
nAI	OpenAI	OpenAI	OpenAI
m	alec@openai.com	jeffwu@openai.com	damodei@openai.com

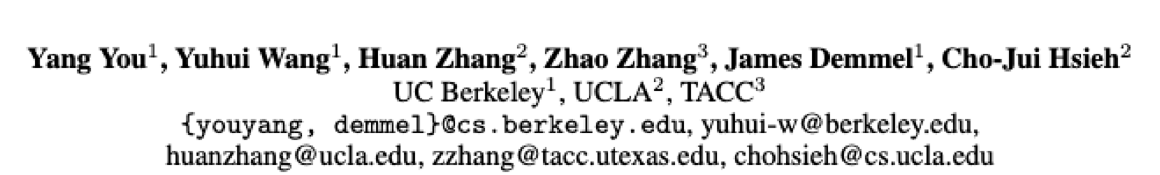


Don't Decay the Learning Rate, Increase the Batch Size

Samuel L. Smith*, Pieter-Jan Kindermans*, Chris Ying & Quoc V. Le
Google Brain
{slsmith, pikinder, chrisying, qvl}@google.com

Abstract

The Limit of the Batch Size



Yang You[1], Yuhui Wang[1], Huan Zhang[2], Zhao Zhang[3], James Demmel[1], Cho-Jui Hsieh[2]
UC Berkeley[1], UCLA[2], TACC[3]
{youyang, demmel}@cs.berkeley.edu, yuhui-w@berkeley.edu,
huanzhang@ucla.edu, zzhang@tacc.utexas.edu, chohsieh@cs.ucla.edu

Architecture hyperparameter tuning

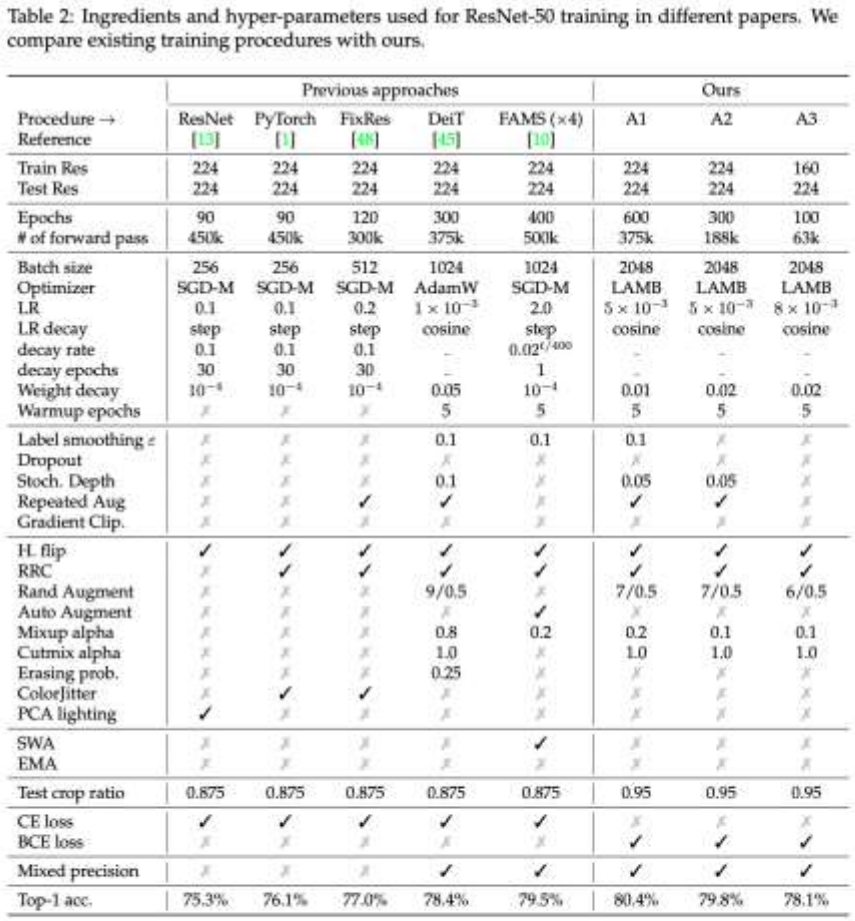
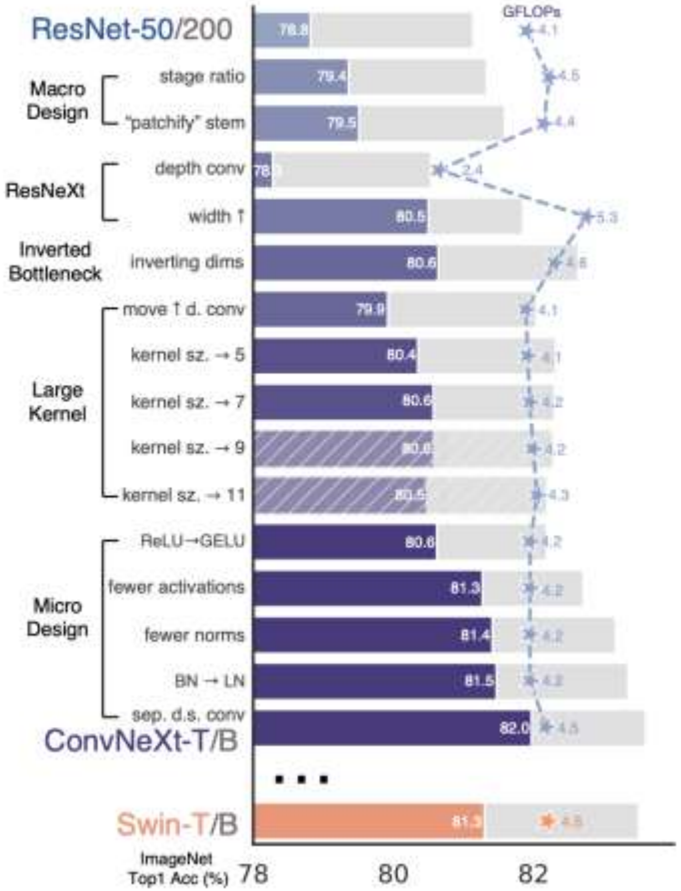


Table 2: Ingredients and hyper-parameters used for ResNet-50 training in different papers. We compare existing training procedures with ours.

	Previous approaches					Ours		
Procedure → Reference	ResNet [13]	PyTorch [1]	FixRes [48]	DeiT [45]	FAMS (×4) [10]	A1	A2	A3
Train Res	224	224	224	224	224	224	224	160
Test Res	224	224	224	224	224	224	224	224
Epochs	90	90	120	300	400	600	300	100
# of forward pass	450k	450k	300k	375k	500k	375k	188k	63k
Batch size	256	256	512	1024	1024	2048	2048	2048
Optimizer	SGD-M	SGD-M	SGD-M	AdamW	SGD-M	LAMB	LAMB	LAMB
LR	0.1	0.1	0.2	1×10^{-3}	2.0	5×10^{-3}	5×10^{-3}	8×10^{-3}
LR decay	step	step	step	cosine	step	cosine	cosine	cosine
decay rate	0.1	0.1	0.1	-	$0.02^{t/400}$	-	-	-
decay epochs	30	30	30	-	1	-	-	-
Weight decay	10^{-4}	10^{-4}	10^{-4}	0.05	10^{-4}	0.01	0.02	0.02
Warmup epochs	✗	✗	✗	5	5	5	5	5
Label smoothing ε	✗	✗	✗	0.1	0.1	0.1	✗	✗
Dropout	✗	✗	✗	✗	✗	✗	✗	✗
Stoch. Depth	✗	✗	✗	0.1	✗	0.05	0.05	✗
Repeated Aug	✗	✗	✓	✓	✗	✓	✓	✗
Gradient Clip.	✗	✗	✗	✗	✗	✗	✗	✗
H. flip	✓	✓	✓	✓	✓	✓	✓	✓
RRC	✗	✓	✓	✓	✓	✓	✓	✓
Rand Augment	✗	✗	✗	9/0.5	✗	7/0.5	7/0.5	6/0.5
Auto Augment	✗	✗	✗	✗	✓	✗	✗	✗
Mixup alpha	✗	✗	✗	0.8	0.2	0.2	0.1	0.1
Cutmix alpha	✗	✗	✗	1.0	✗	1.0	1.0	1.0
Erasing prob.	✗	✗	✗	0.25	✗	✗	✗	✗
ColorJitter	✗	✓	✓	✗	✗	✗	✗	✗
PCA lighting	✓	✗	✗	✗	✗	✗	✗	✗
SWA	✗	✗	✗	✗	✓	✗	✗	✗
EMA	✗	✗	✗	✗	✗	✗	✗	✗
Test crop ratio	0.875	0.875	0.875	0.875	0.875	0.95	0.95	0.95
CE loss	✓	✓	✓	✓	✓	✗	✗	✗
BCE loss	✗	✗	✗	✗	✗	✓	✓	✓
Mixed precision	✗	✗	✗	✓	✓	✓	✓	✓
Top-1 acc.	75.3%	76.1%	77.0%	78.4%	79.5%	80.4%	79.8%	78.1%

ResNet strikes back: An improved training procedure in timm. Wightman et al. 2021



A ConvNet for the 2020s. Liu et al. CVPR 2022

Some magic starting numbers I’m aware of

Learning rate: 0.01 with a nice scheduler (my goto is standard multi-step)

Weight decay: 1e-4 or 5e-4

Batch size: Biggest power of 2 that fits in memory

DropOut: 20-50% drop out rate

And make sure to compute the mean and variance of the training samples!

Babysitting deep networks

Establish baselines

Check that in the first round you get loss that corresponds to random guess

Check network with few samples

- Turn off regularization. You should predictably overfit and get a loss of 0
- Turn on regularization. The loss should be higher than before

Always a separate validation set for hyper-parameter tuning

- Compare the training and validation losses - there should be a gap, not too large

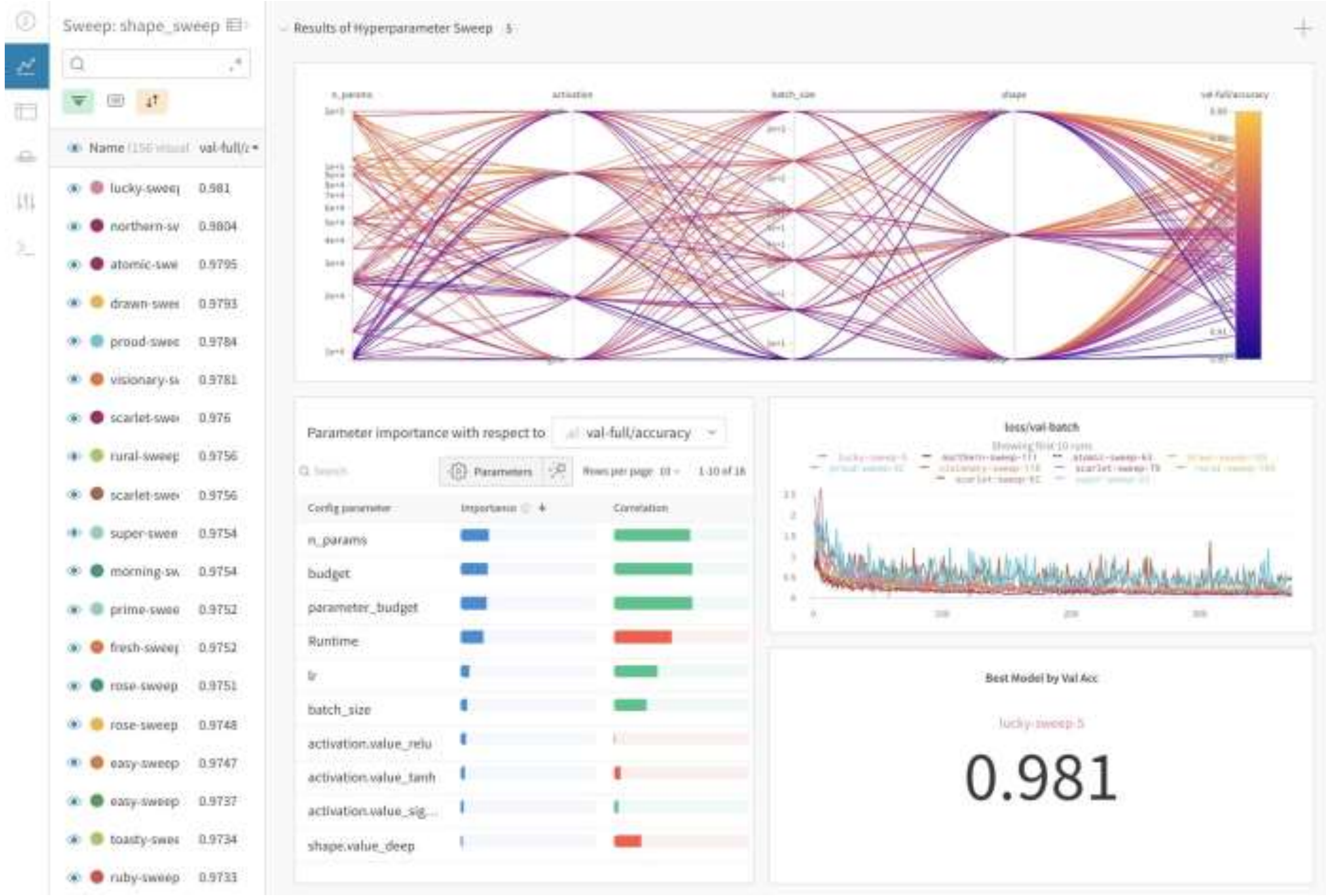
Preprocess the data (at least to have 0 mean)

Initialize weights based on activations functions Xavier or Kaiming initialization

Use regularization (ℓ_2-regularization, dropout, ...)batch normalization

Prefer residual connections, they make a difference

Logging tools



Sweep: shape_sweep

Results of Hyperparameter Sweep 5

Name (156 visual)	val-full/a
lucky-sweep	0.981
northern-sw	0.9804
atomic-swe	0.9795
drawn-swee	0.9793
proud-swee	0.9784
visionary-s	0.9781
scarlet-swe	0.976
rural-sweep	0.9756
scarlet-swe	0.9756
super-swee	0.9754
morning-sw	0.9754
prime-swee	0.9752
fresh-sweep	0.9752
rose-sweep	0.9751
rose-sweep	0.9748
easy-sweep	0.9747
easy-sweep	0.9737
toasty-swee	0.9734
ruby-sweep	0.9733

Parameter importance with respect to val-full/accuracy

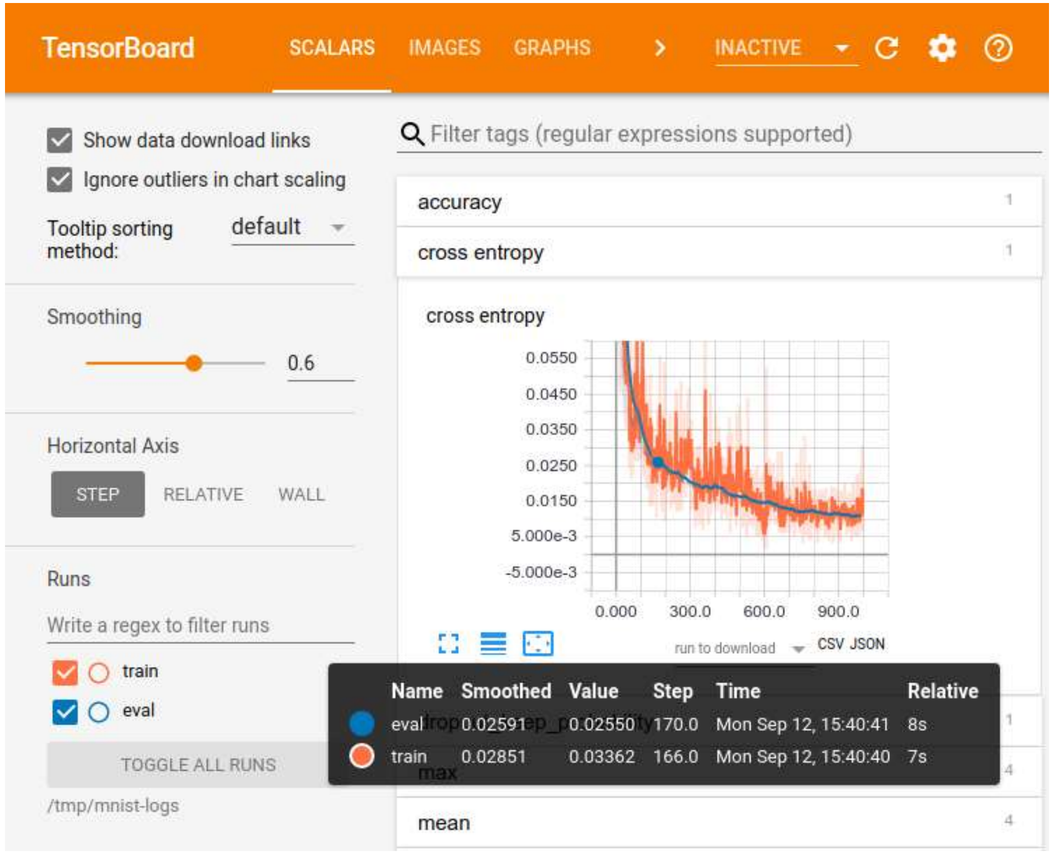
Parameters Rows per page 10 1-10 of 18

Config parameter	Importance	Correlation
n_params		
budget		
parameter_budget		
Runtime		
lr		
batch_size		
activation.value_relu		
activation.value_tanh		
activation.value_sig...		
shape.value_deep		

Best Model by Val Acc

lucky-sweep-5

0.981



TensorBoard

☑ Show data download links

☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0.6

Horizontal Axis: STEP RELATIVE WALL

Runs

Write a regex to filter runs

☑ ○ train

☑ ○ eval

TOGGLE ALL RUNS

/tmp/mnist-logs

Filter tags (regular expressions supported)

accuracy 1

cross entropy 1

cross entropy

	Name	Smoothed	Value	Step	Time	Relative
●	eval	0.02591	0.02550	170.0	Mon Sep 12, 15:40:41	8s
●	train	0.02851	0.03362	166.0	Mon Sep 12, 15:40:40	7s

max

mean 4

Summary

Tackling overfitting is key in deep learning.

Initialization: *een goed begin is het halve werk*.

Regularization, DropOut, early stopping, normalization: all dampening factors.

Your lives will be dominated by hyperparameters.

Learning and reflection

Understanding Deep Learning: Chapter 8

Understanding Deep Learning: Chapter 9

Next lecture

Lecture	Title
1	Intro and history of deep learning
3	Deep learning optimization I
5	Convolutional deep learning
7	Graph deep learning
9	Multi-modal deep learning
11	What doesn’t work in deep learning
13	Q&A

Lecture	Title
2	AutoDiff
4	Deep learning optimization II
6	Attention-based deep learning
8	From supervised to unsupervised deep learning
10	Generative deep learning
12	Non-Euclidean deep learning
14	Deep learning for videos