Regularization

Tripp Deep Learning F23

With figures from Goodfellow et al., Deep Learning, and Bishop, Pattern Recognition & Machine Learning

TODAY'S GOAL

By the end of the class, you should understand various regularization approaches and other ways to reduce overfitting and how to apply them.

Summary

1. Overfitting can be reduced by simplifying the model, using more training data, or regularization
2. Training data can often be augmented to increase its apparent size
3. Overfitting can be reduced by training on multiple datasets
4. Overfitting tends to develop during training, and early stopping can reduce it
5. Parameter norm penalties shrink weights selectively
6. Dropout is a practical approximation of bagging for large networks
7. Batch normalization is also a regularizer

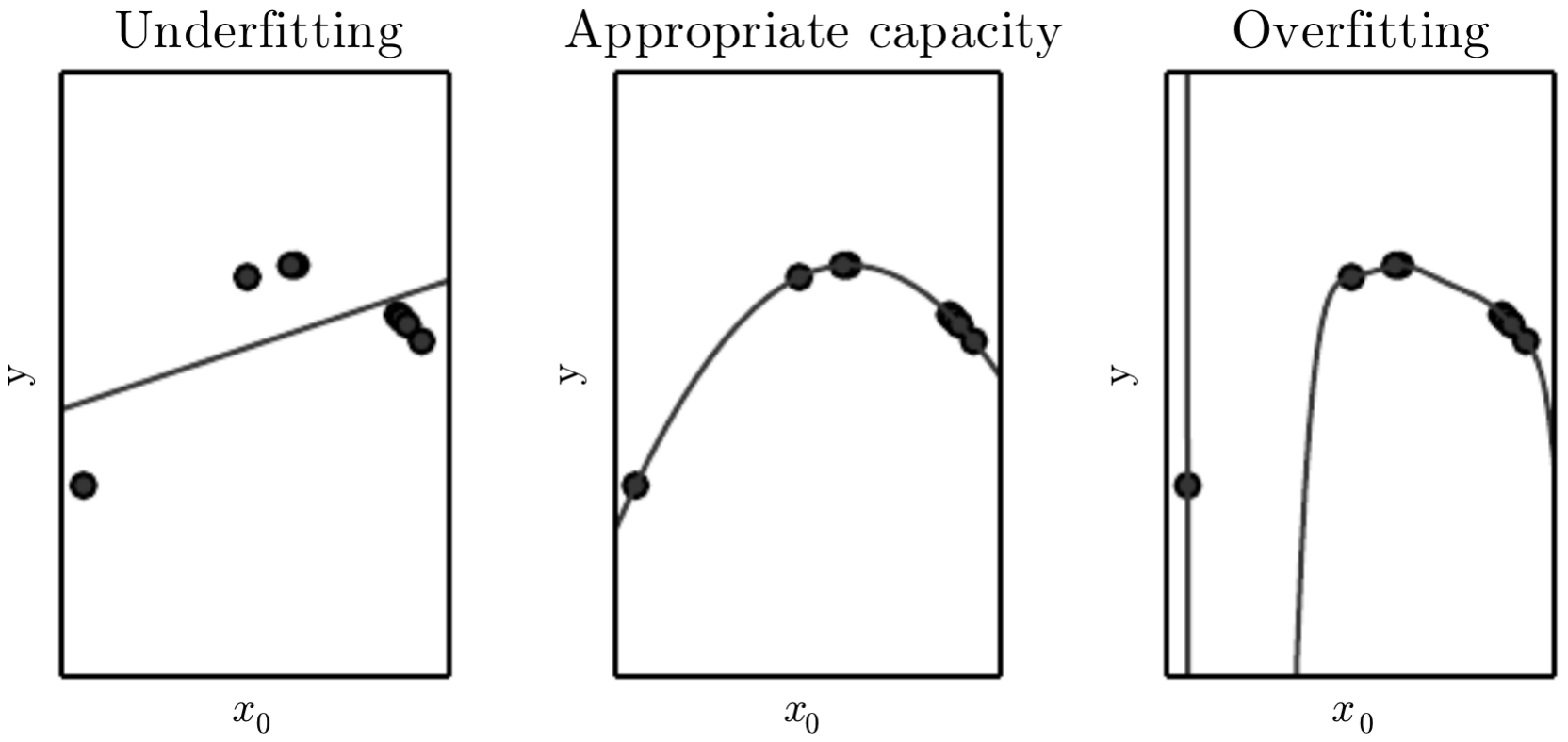
Recall

- In supervised learning we fit a model to labelled data, but the important thing is performance on new data; good performance on data we already have labels for is of no use
- Good performance on training data doesn't imply good performance on new data
- If there is a substantial difference between performance on training data and validation data, the model is said to overfit the training data

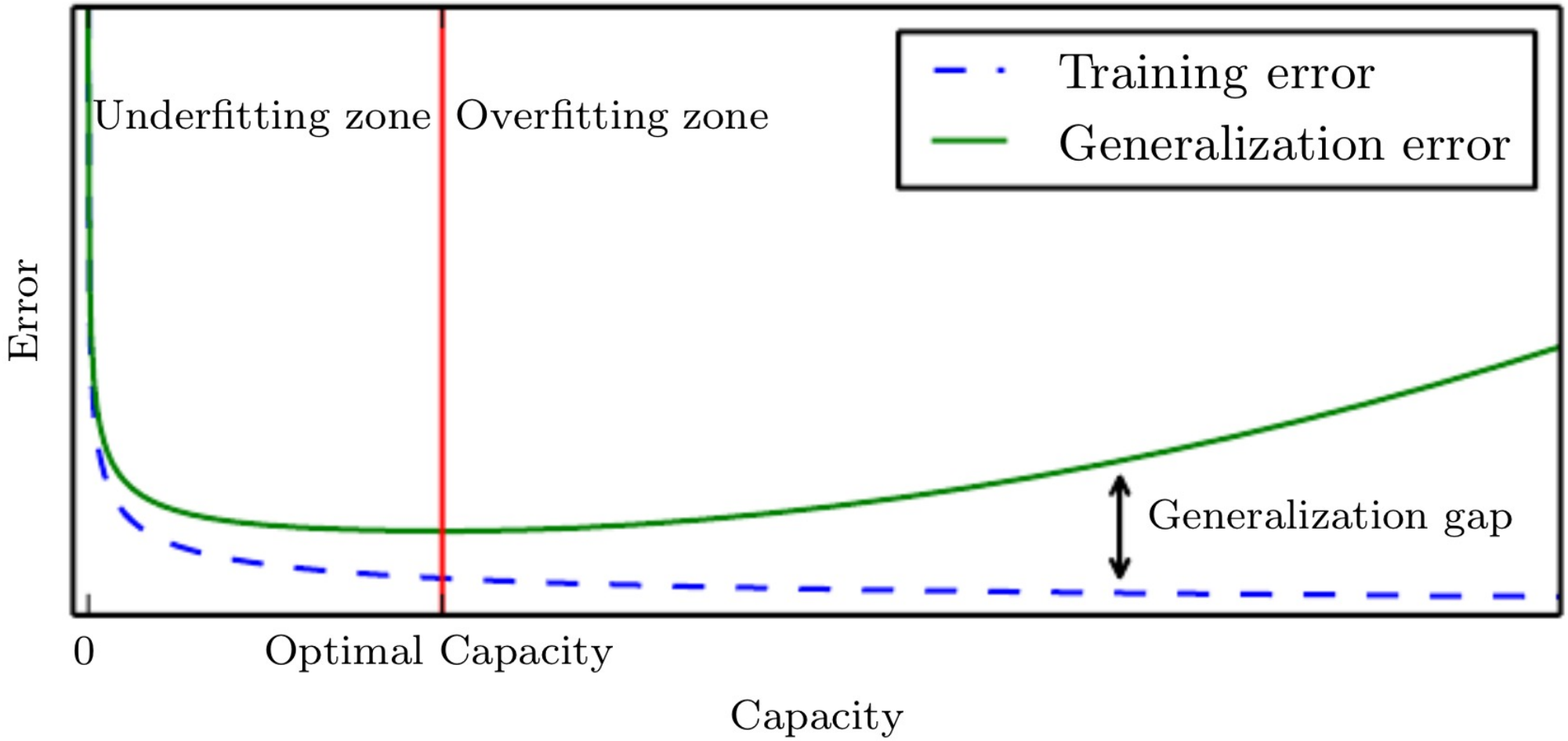
OVERFITTING CAN BE REDUCED BY SIMPLIFYING THE MODEL, USING MORE TRAINING DATA, OR REGULARIZING

Large model capacity can lead to overfitting

- Capacity is the variety of functions that a model can implement

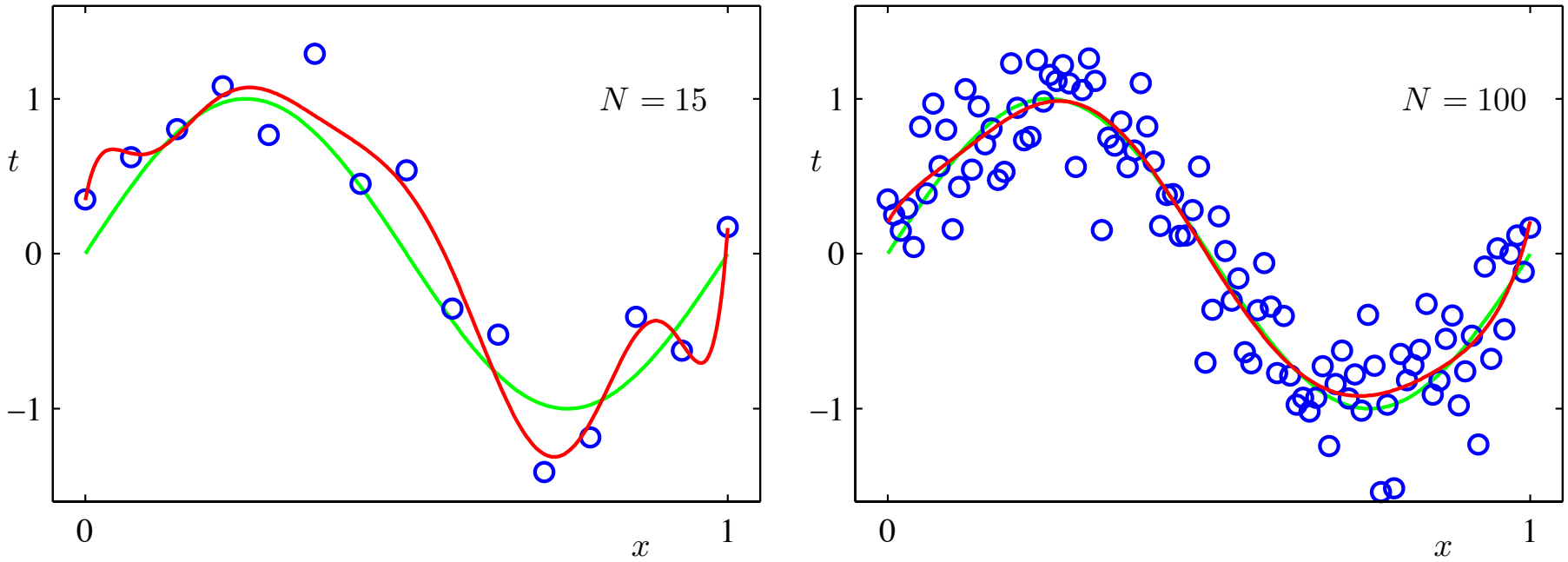


Large model capacity can lead to overfitting

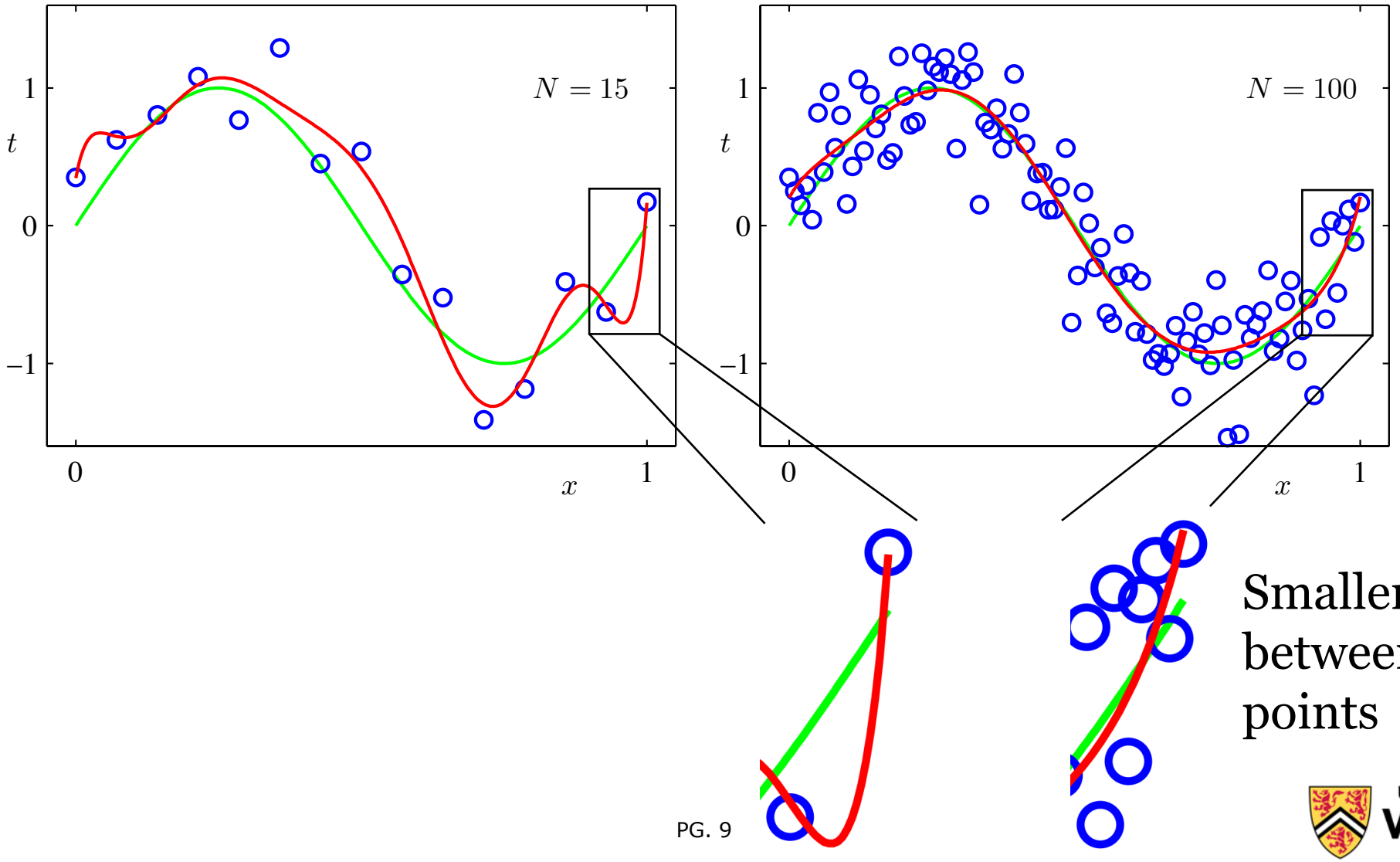


More training data is better

Example: 9th-order polynomial fit in with few data points vs. many data points



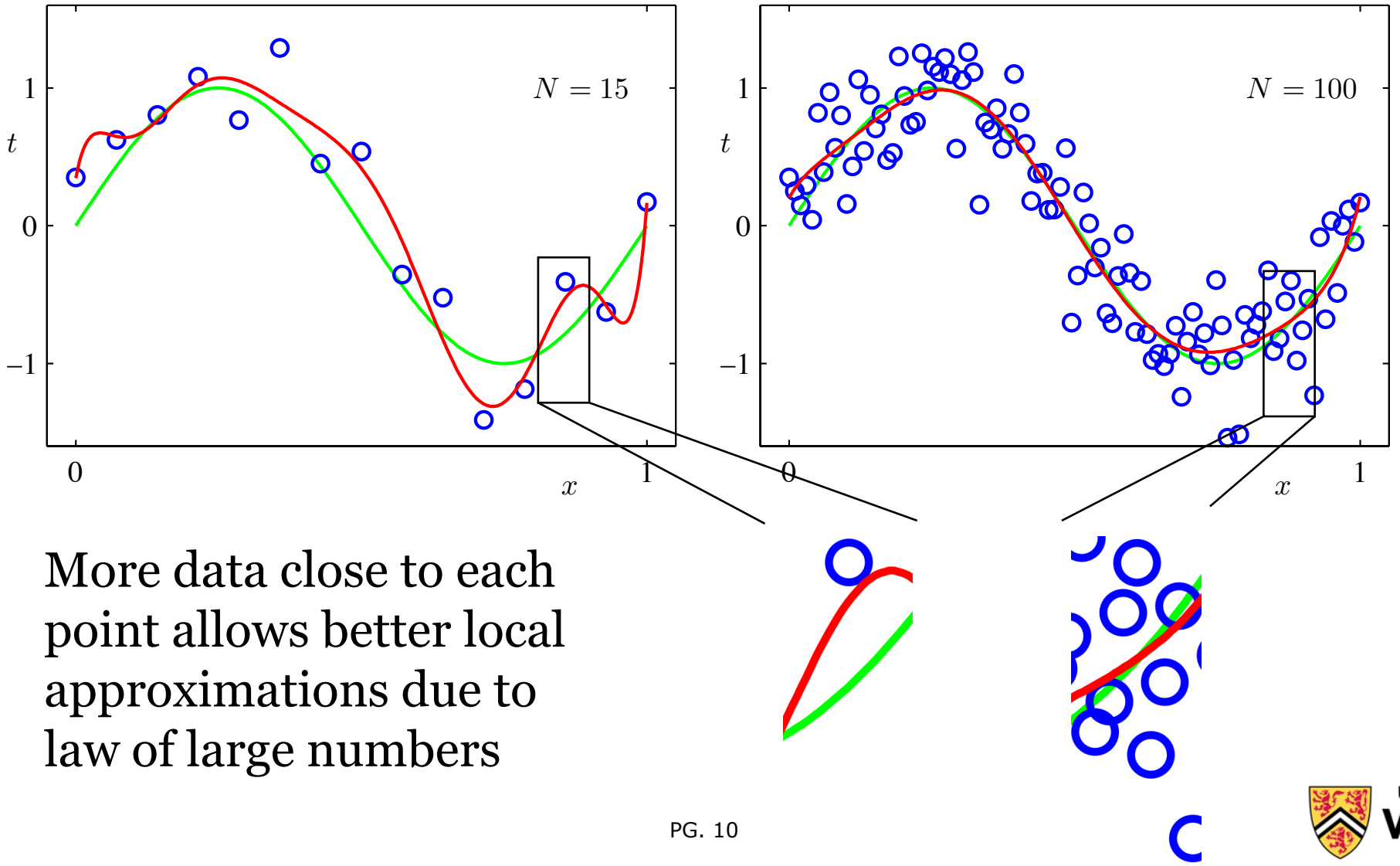
More training data is better



Smaller gaps between data points

UNIVERSITY OF WATERLOO

More training data is better



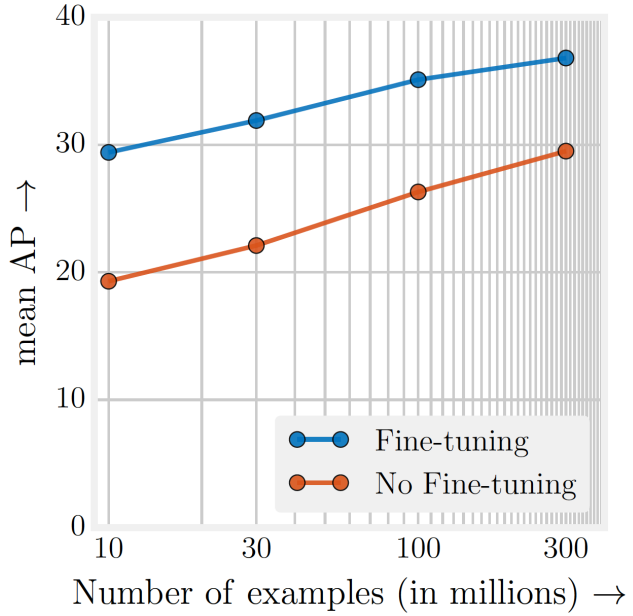
More data close to each point allows better local approximations due to law of large numbers

UNIVERSITY OF WATERLOO

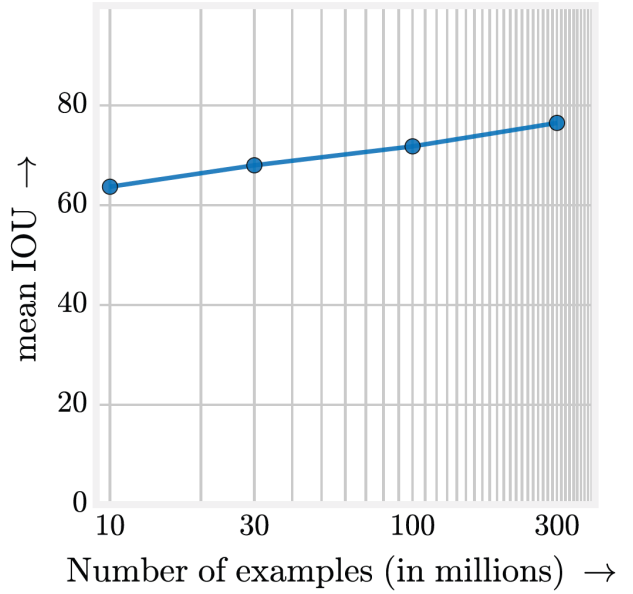
More training data is better

- For vision tasks, networks are often trained from scratch on thousands to hundreds of thousands of examples, or pre-trained on ImageNet 1K (1000 categories and ~1.2M images)
- Sun et al. (2017) pretrained a ResNet-101 network on the JFT-300M dataset (~300M images with noisy labels in ~18K categories) for object recognition, object detection, semantic segmentation, and pose detection
- In each task performance improved with more data (logarithmically with dataset size)
- This did not work with a less complex network

Object detection (COCO minival)



Semantic segmentation (Pascal VOC 2012)



More training data is better

- Typically, a multiplicative increase in good-quality data from a suitable distribution can lead to an additive increase in performance, if the model is sufficiently complex

Regularization

- *"... any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."*
  - Goodfellow et al., *Deep Learning*
- When deep networks are used for complex problems, typically regularizing a high-capacity model gives better results than using a low-capacity unregularized model

Regularization methods

- Common regularization methods include:
  - Early stopping
  - Parameter norm penalties
  - Dropout
  - Batch normalization

TRAINING DATA CAN OFTEN BE AUGMENTED TO INCREASE ITS APPARENT SIZE

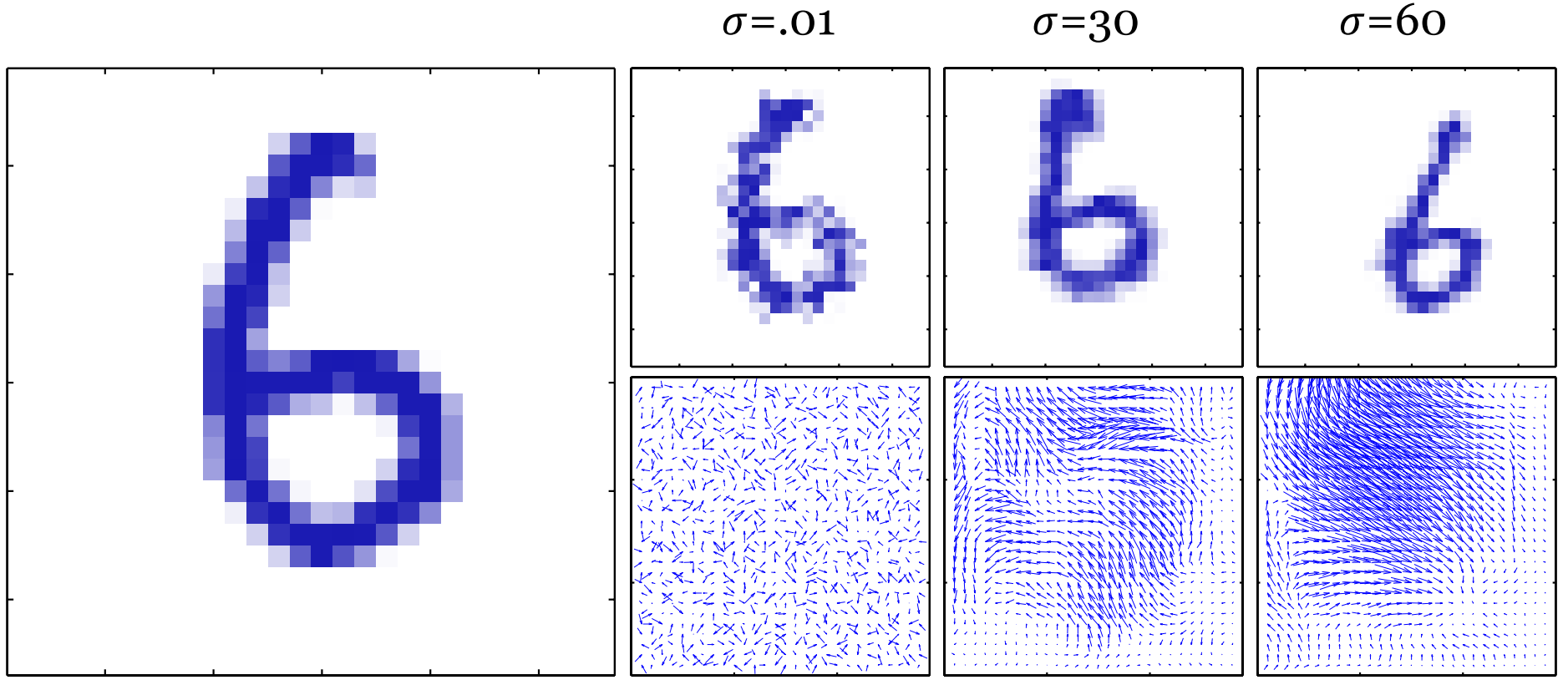
Augmentation creates varied copies of inputs

- Example: Standard ways to augment ImageNet (about one million labelled images of things, widely used for object recognition)
  - Flip image horizontally
  - Randomly perturb colours
  - Random crops
  - Small random rotations
- This creates a million groups of similar images, which is somewhat better than a million images
- Other tasks may allow other kinds of augmentation

Example: Augmentation of handwritten digits

Pixels of an image are moved at random.

This can create realistic-looking digits if perturbations of nearby pixels are correlated.



Bishop, *PRML*

random perturbations

random perturbations smoothed with Gaussian kernels

UNIVERSITY OF WATERLOO

Example: Augmentation of 3D shapes

Here a mesh was stretched symmetrically from the centre by a random polynomial function of height.

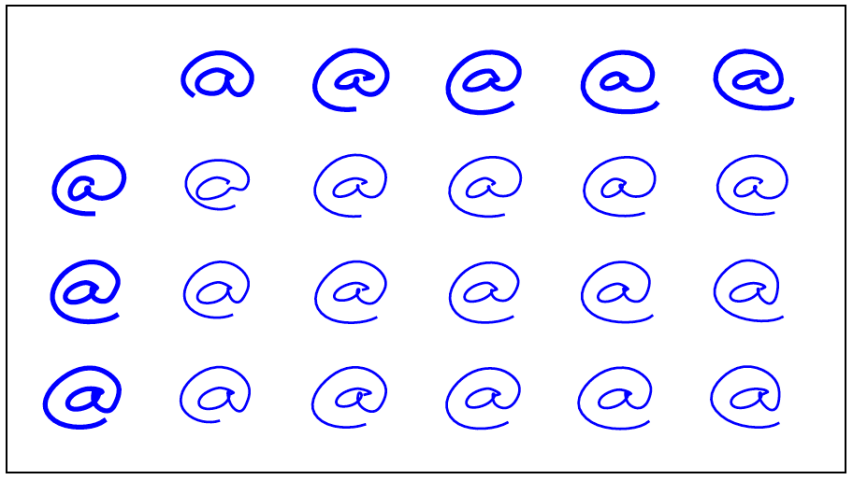
Augmentation hyperparameters

- The types of augmentations, numbers of augmentations to apply, and degree to which input is changed are all hyperparameters that can affect performance
- It is expensive to explore this hyperparameter space
- Uniform sampling of several types of augmentation and using a uniform strength for all augmentations (relative to a manually defined maximum) simplifies the search space and seems to work well (Cubuk et al., 2020)

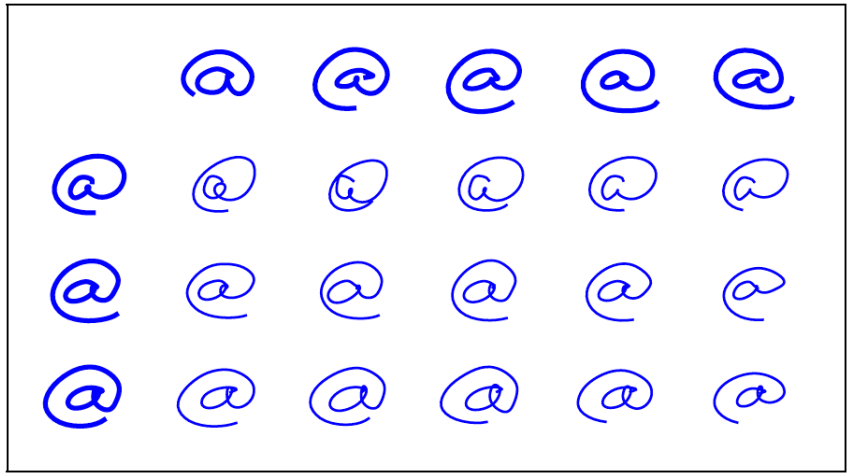
Augmentation in feature space

- Augmentations are domain-specific and arbitrary augmentations don't always work
  - E.g., for singing detection, pitch shifting and application of random frequency filters are useful augmentations, time stretching is not, and mixing different samples is harmful (Schlüter & Grill, 2015)
- DeVries & Taylor (ICLR 2017) suggested extrapolating between examples in feature space (not input space)
  - This method is network-agnostic and task-agnostic
  - It may produce better examples since features tend to vary along relevant stimulus dimensions
  - It was found to be effective in a variety of tasks

Interpolation



Extrapolation



OVERFITTING CAN BE REDUCED BY TRAINING ON MULTIPLE DATASETS

Transfer learning

Given a source domain DS and learning task TS, a target domain DT and learning task TT, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in DT using the knowledge in DS and TS, where $DS \neq DT$, or $TS \neq TT$.

– Pan & Yang, 2010, *IEEE Trans Knowledge & Data Engineering*

Transfer learning

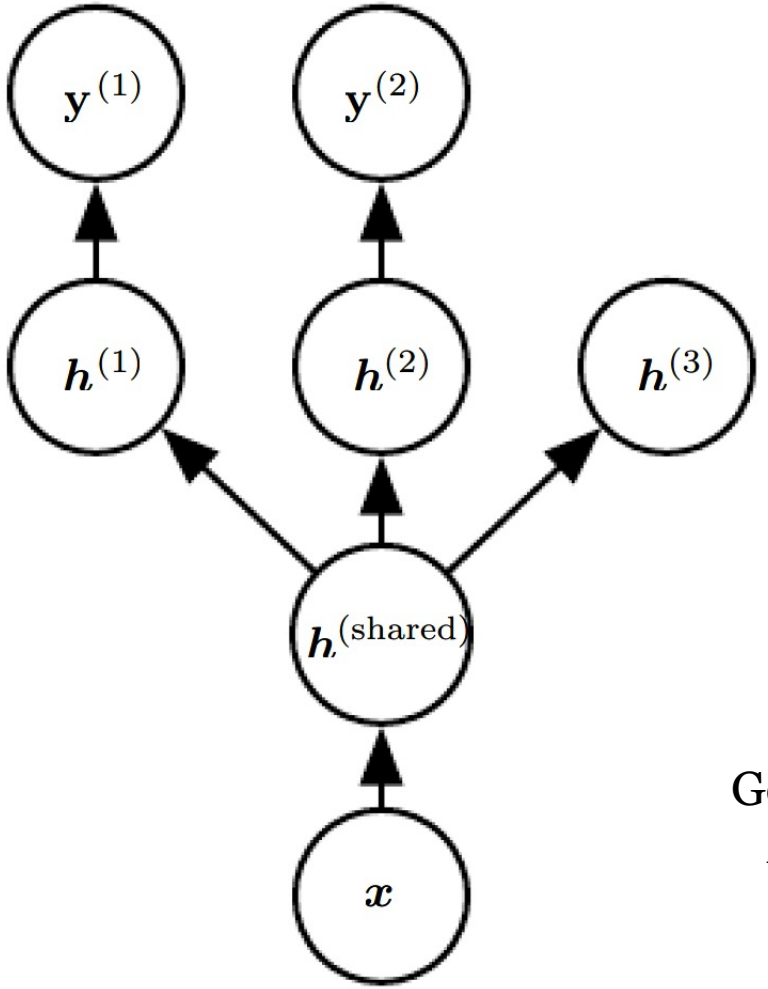
- A common kind of transfer learning consists of:
  - Supervised learning on a dataset that is not the target dataset (pre-training)
  - Supervised learning on the target dataset, beginning with the parameters learned on the non-target dataset
- The non-target dataset should ideally be:
  - Similar to the target in some way
  - Large
- Example: Deep learning in computer vision often involves transfer learning with the ImageNet 1K dataset
- This can improve generalization by constraining parameters more than is possible with the target dataset (particularly if the target dataset is small)

Self-supervised pre-training

- A network can also be initialized with parameters from self-supervised learning
- Self-supervised learning consists of predicting labels that can be calculated automatically from the inputs, so that they do not require manual labelling
- This is important because most data is unlabelled
- Examples:
  - Predicting masked words in a sentence
  - Predicting masked parts of an image
  - Predicting the next frame of video

Multi-task learning

- Training a model to perform multiple tasks provides additional data to constrain shared parts of the model
- This can improve generalization if some factors that explain variations in the data are shared across tasks

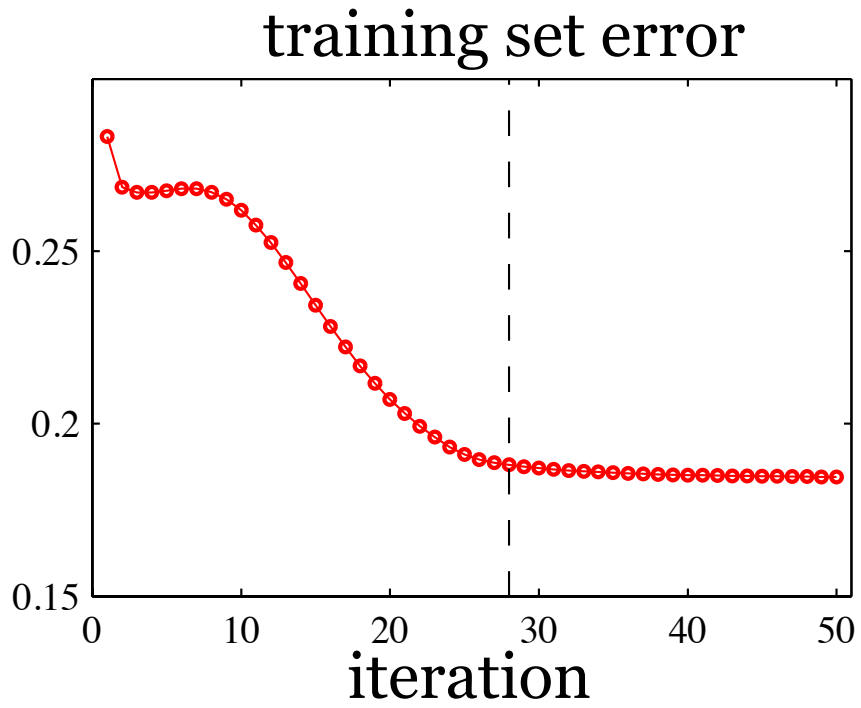


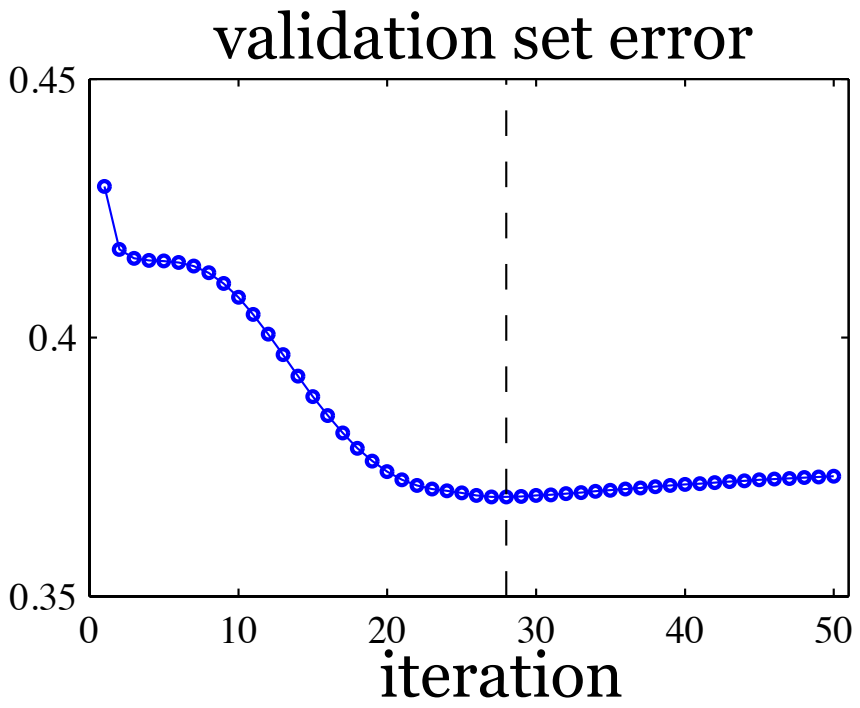
Goodfellow et al.
Deep Learning

OVERFITTING TENDS TO DEVELOP DURING TRAINING, AND EARLY STOPPING CAN REDUCE IT

Early stopping motivation

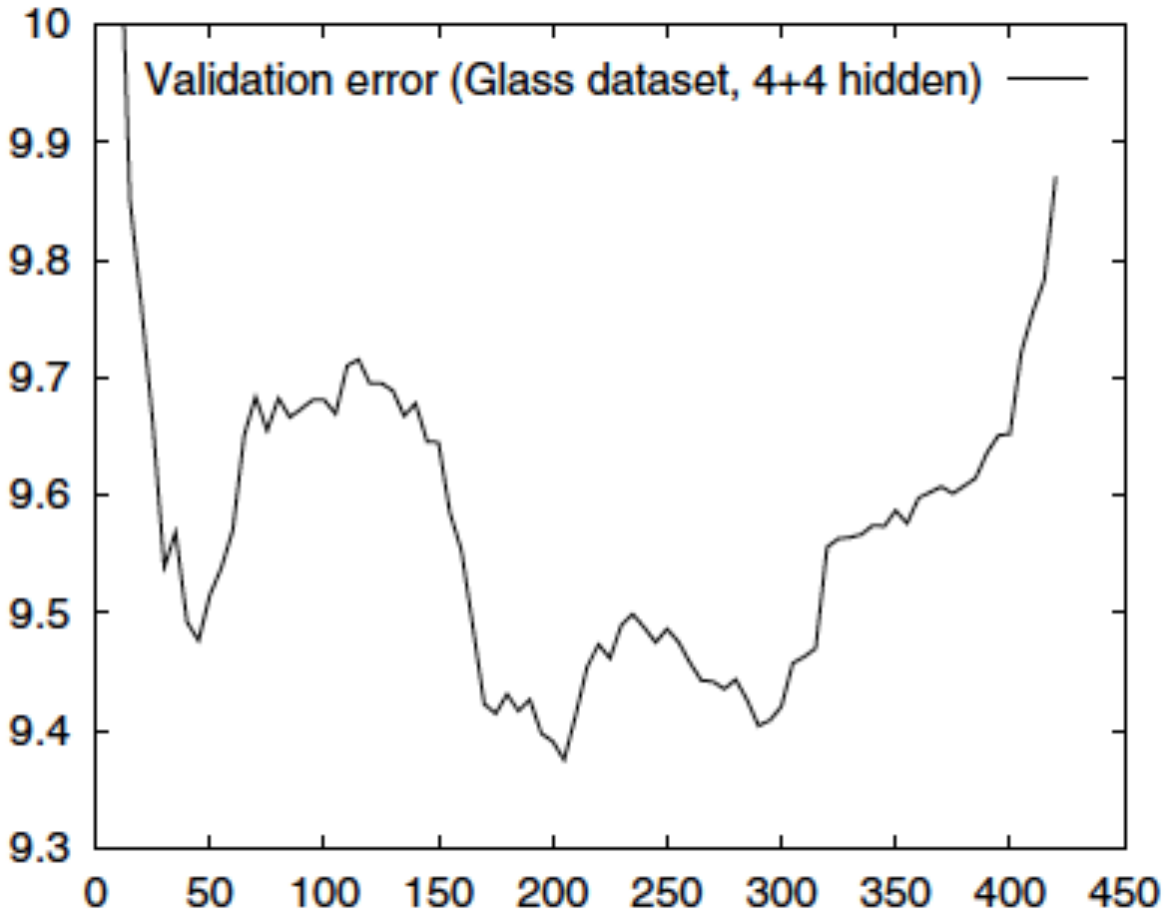
- Training error often decreases almost monotonically, while validation error often decreases then increases, as the network starts to overfit the training data
- A simple way to reduce overfitting is to stop training when the validation error starts to rise (before convergence of training loss)





UNIVERSITY OF WATERLOO

Typical validation error curve is not guaranteed



Prechelt (2012) "Early stopping-but when?" *Neural Networks: Tricks of the trade. 2nd Ed.*

UNIVERSITY OF WATERLOO

Algorithm

- θ_0 are the initial parameters
- n is the number of steps between evaluations
- p is the "patience," the number of times to observe worse validation set error before giving up

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*

Efficiency

- Early stopping is a particularly efficient regularization method because many levels of early-stopping regularization can be tested without retraining
- In contrast, with other methods, it is necessary to guess regularization parameter values in advance and test each value with a separate training run

PARAMETER NORM PENALTIES SHRINK WEIGHTS SELECTIVELY

General idea of parameter norm penalties

- So far, we have considered task-related loss functions, $L(\theta, X, Y)$, where X and Y are inputs and outputs and θ are network parameters (weights and biases)
- To perform parameter-norm regularization, we add a term to the loss that penalizes larger parameters,

$$\tilde{L}(\theta, X, y) = L(\theta, X, y) + \alpha\Omega(\theta)$$

where Ω is the parameter penalty function and $\alpha \geq 0$ sets the amount of regularization.

General idea of parameter norm penalties

- $\Omega(\theta)$ is usually a vector p-norm,

$$\|\theta\|_p = \left(\sum_{i=1}^{n} |\theta_i|^p\right)^{1/p}$$

- The most common norms are the 1-norm (L^1 regularization) and the 2-norm (L^2 regularization)
- Typically, the weights are regularized but not the biases, as biases don't usually cause problems

Example: L² penalty in linear regression (ridge regression)

No regularization:

$$L = (X\boldsymbol{w} - \boldsymbol{y})^T(X\boldsymbol{w} - \boldsymbol{y})$$

$$\boldsymbol{w}^* = (X^TX)^{-1}X^T\boldsymbol{y}$$

This product usually has some small eigenvalues, which lead to large weights, which can lead to large excursions of the prediction between samples

L² regularization:

$$\tilde{L} = (X\boldsymbol{w} - \boldsymbol{y})^T(X\boldsymbol{w} - \boldsymbol{y}) + \alpha\boldsymbol{w}^T\boldsymbol{w}$$

$$\boldsymbol{w}^* = (X^TX + \alpha I)^{-1}X^T\boldsymbol{y}$$

This sum does not have small eigenvalues

L^2 regularization is like weight decay

The loss with L^2 regularization and weights w can be written,

$$\tilde{L}(\boldsymbol{w}, X, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w} + L(\boldsymbol{w}, X, y)$$

The gradient of the loss with respect to $\boldsymbol{w}$ is,

$$\nabla_w \tilde{L}(\boldsymbol{w}, X, y) = \alpha\boldsymbol{w} + \nabla_w L(\boldsymbol{w}, X, y)$$

After a gradient step with learning rate ϵ, the weight vector changes to,

$$\boldsymbol{w}_{new} = \boldsymbol{w} - \epsilon\big(\alpha\boldsymbol{w} + \nabla_w L(\boldsymbol{w}, X, y)\big) = (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_w L(\boldsymbol{w}, X, y)$$

Like shrinking weights slightly before a normal update

L^2 regularization scales weights selectively

To see how weight decay affects the optimized weights, we use a quadratic approximation of the loss around the non-regularized optimal weights $\boldsymbol{w}^*$,

$$\hat{L}(\boldsymbol{w}) = L(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T H(\boldsymbol{w} - \boldsymbol{w}^*)$$

where H is the Hessian matrix of 2^{nd}-order partial derivatives of L at $\boldsymbol{w}^*$. The gradient of this approximate loss is,

$$\nabla_w \hat{L}(\boldsymbol{w}) = H(\boldsymbol{w} - \boldsymbol{w}^*)$$

which is zero at $\boldsymbol{w}^*$. Regularization results in different optimal weights, $\tilde{\boldsymbol{w}}$. Adding the weight decay gradient,

$$\alpha\tilde{\boldsymbol{w}} + H(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0$$

L^2 regularization scales weights selectively

$$\alpha\tilde{\boldsymbol{w}} + H(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0$$

$$(H + \alpha I)\tilde{\boldsymbol{w}} = H\boldsymbol{w}^*$$

$$\tilde{\boldsymbol{w}} = (H + \alpha I)^{-1} H\boldsymbol{w}^*$$

← Note $\tilde{w}$ approaches w^* as α approaches 0

H is real and symmetric, so it can be decomposed as $H = Q\Lambda Q^T$, a product of a diagonal matrix Λ and a matrix of orthonormal eigenvectors Q. So,

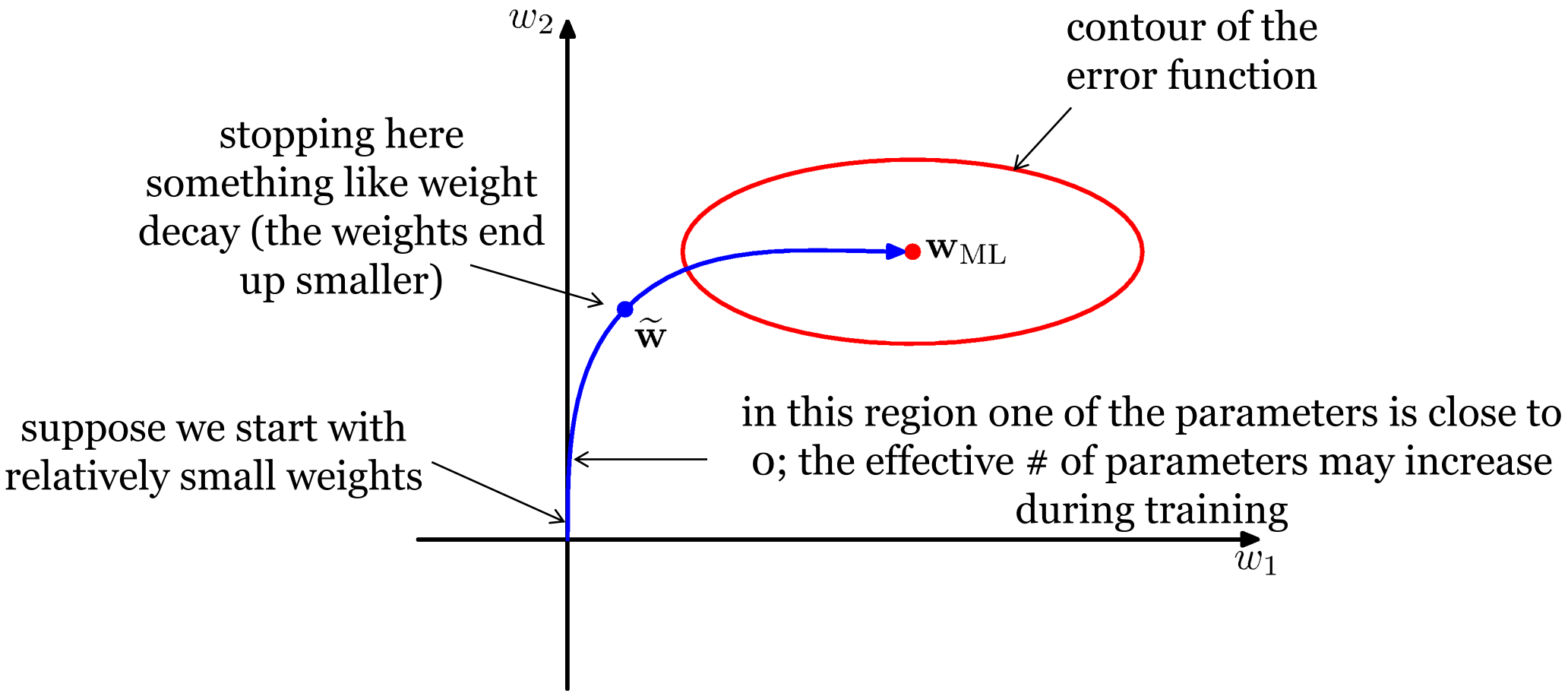
$$\tilde{\boldsymbol{w}} = (Q\Lambda Q^T + \alpha I)^{-1} Q\Lambda Q^T \boldsymbol{w}^*$$

$$\tilde{\boldsymbol{w}} = (Q(\Lambda + \alpha I)Q^T)^{-1} Q\Lambda Q^T \boldsymbol{w}^*$$

$$\tilde{\boldsymbol{w}} = Q(\Lambda + \alpha I)^{-1} \Lambda Q^T \boldsymbol{w}^*$$

← Weights are rescaled by factors $\lambda_i/(\lambda_i + \alpha)$; not much for large λ_i (directions of high curvature in the cost)

L^2 regularization can resemble early stopping



L^2 regularization can resemble early stopping

- Early stopping limits weight size because gradient, learning rate, and # steps are bounded
- These methods can be equivalent if:
  - The task-based loss is quadratic
  - Weights are initialized at origin

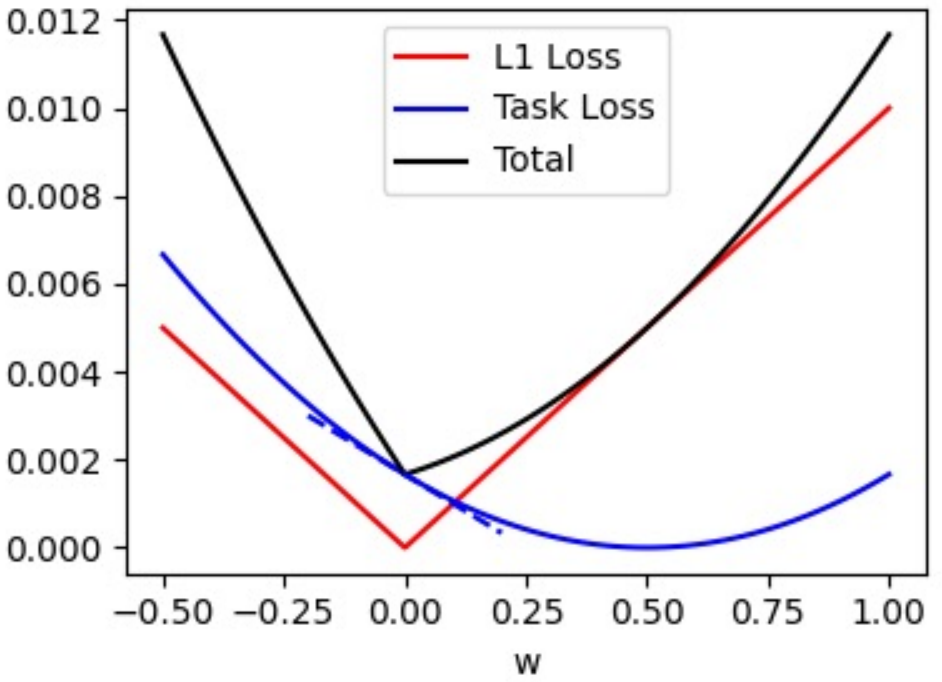
L^1 regularization makes weights sparse

- Sets some weights to zero
- This can be useful at the input for feature selection, and possibly for more efficient computation, depending on hardware

$$\Omega(\theta) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

$$\tilde{L}(\boldsymbol{w}, X, y) = \alpha\|\boldsymbol{w}\|_1 + L(\boldsymbol{w}, X, y)$$

$$\nabla_w \tilde{L}(\boldsymbol{w}, X, y) = \alpha\, \mathrm{sgn}(\boldsymbol{w}) + \nabla_w L(\boldsymbol{w}, X, y)$$



← w_i gets pushed to 0 unless moving it away from zero affects L with slope < -α

UNIVERSITY OF WATERLOO

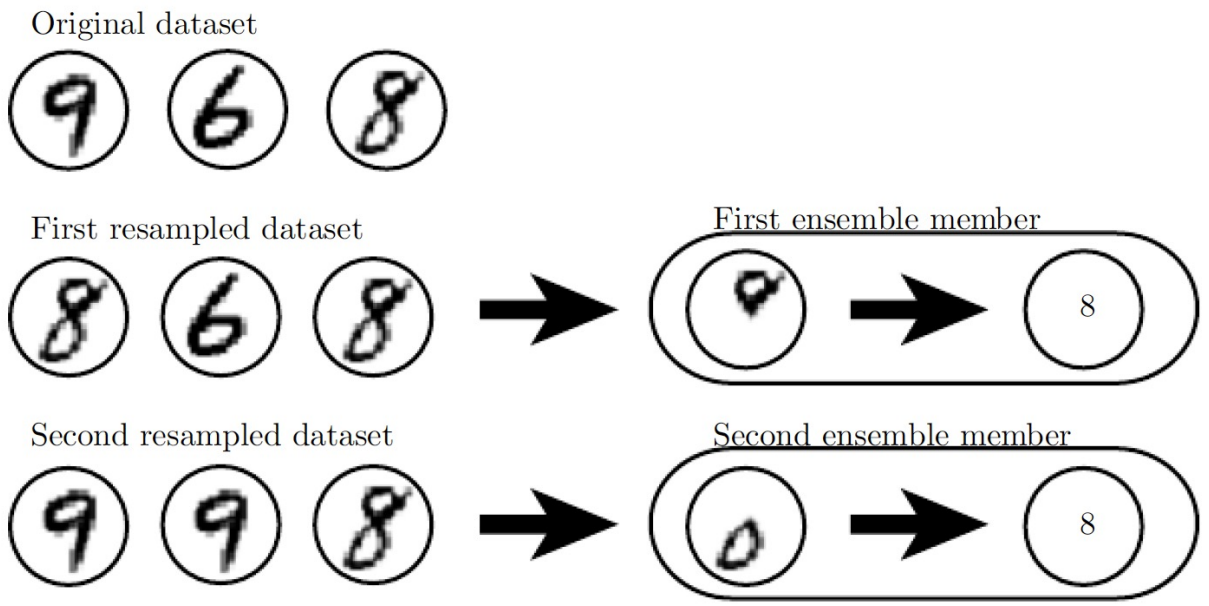
DROPOUT IS A PRACTICAL APPROXIMATION OF BAGGING FOR LARGE NETWORKS

Background: Bootstrap aggregating (bagging)

- Create multiple replicates of the training dataset, each the same size as the dataset but sampled with replacement; train a model on each replicate
- The predictions should be different even if the models are initialized identically, due to exposure to different examples
- The average prediction (for regression tasks) or plurality (for classification tasks) of all the models can be more accurate than prediction of single model
- If the networks' errors aren't perfectly correlated, net performance improves
- This works with deep networks but isn't done as routinely as with smaller models because deep networks are expensive to train

Learning complementary distinctions with bagging

- Here is a simple example: Suppose a dataset consists of handwritten 9s, 6s, and 8s.
- If one replicate has mainly 8s and 6s, the corresponding model should learn that a loop at the **top** is an important distinguishing feature
- If another replicate has mainly 9s and 8s, the corresponding model should learn that a loop at the **bottom** is an important distinguishing feature



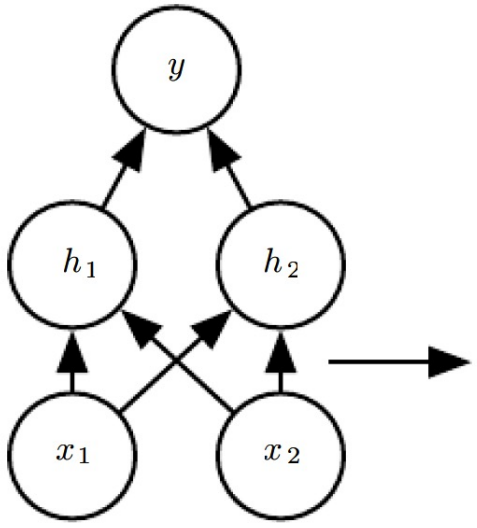
Dropout

Training:

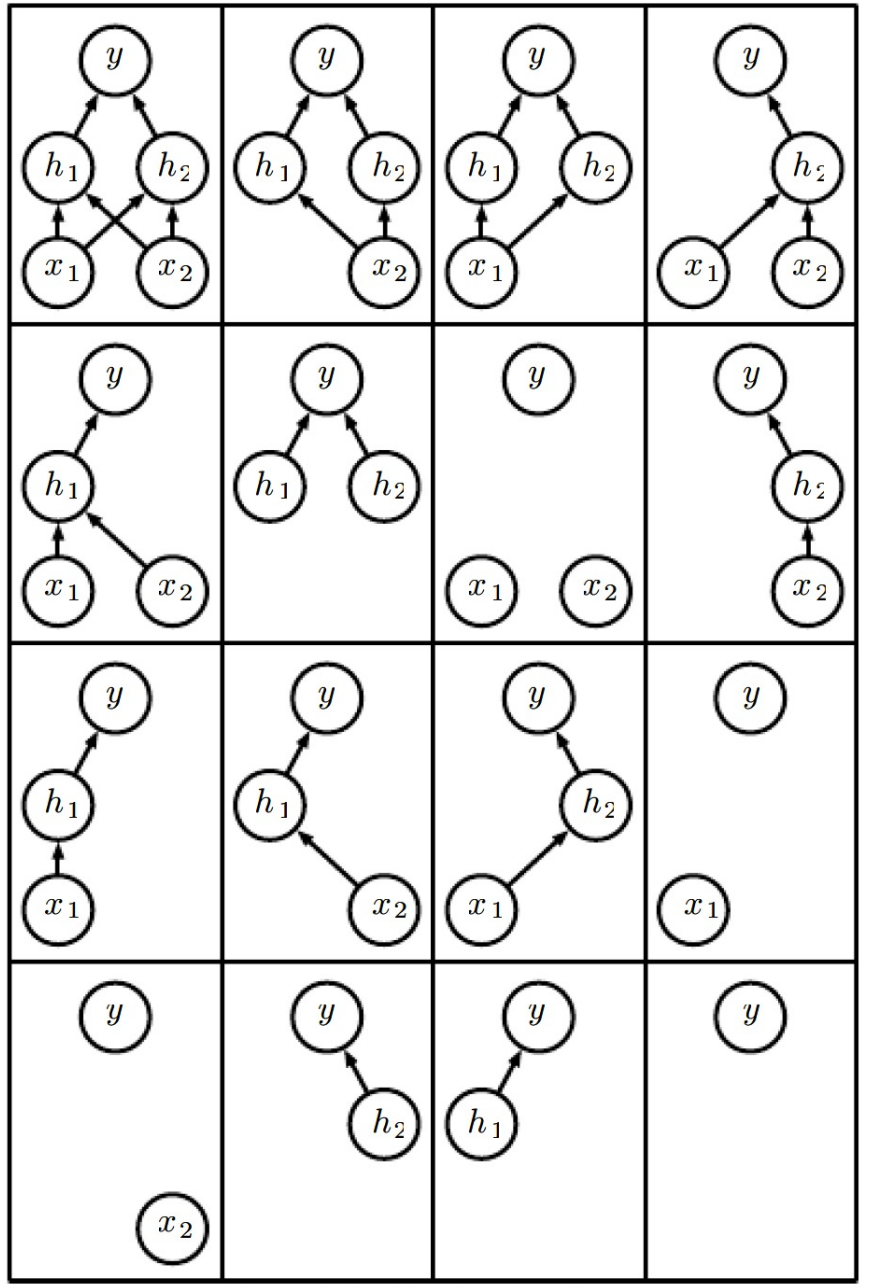
- In each forward/ backward training pass, remove each unit from the network with probability $1 - p$
- Rescale outbound weights of other units by $1/p$

Inference:

- Include all units and undo weight rescaling



Base network

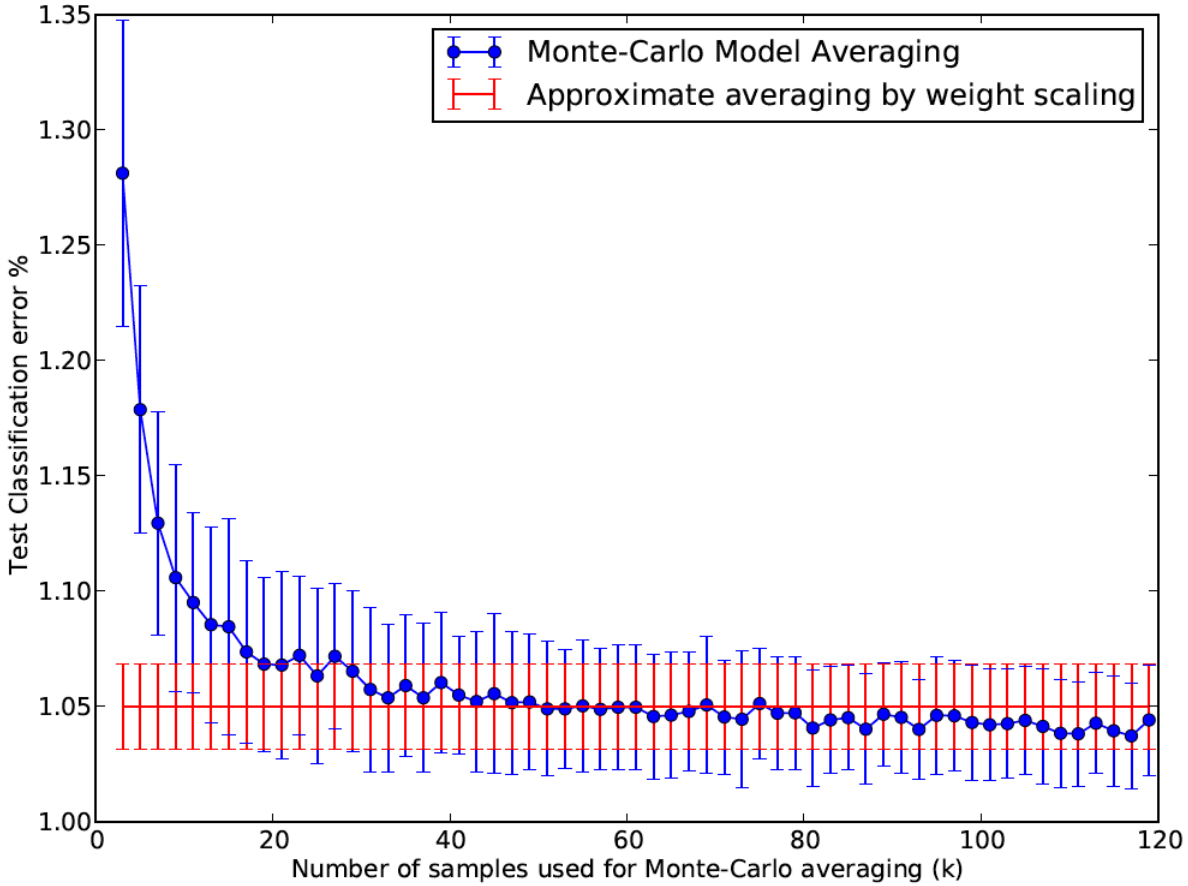


Ensemble of subnetworks

Dropout is related to bagging

- This approximates bagging in a way that is practical for large models
- In contrast with bagging:
  - There are as many models as training passes (sampled from $2^{\#units}$ possibilities)
  - Parameters are shared between models
  - Each model sees at most one training example (or batch)
- Inference method approximates averaging over many random network samples

Srivastava et al. (2014) JMLR

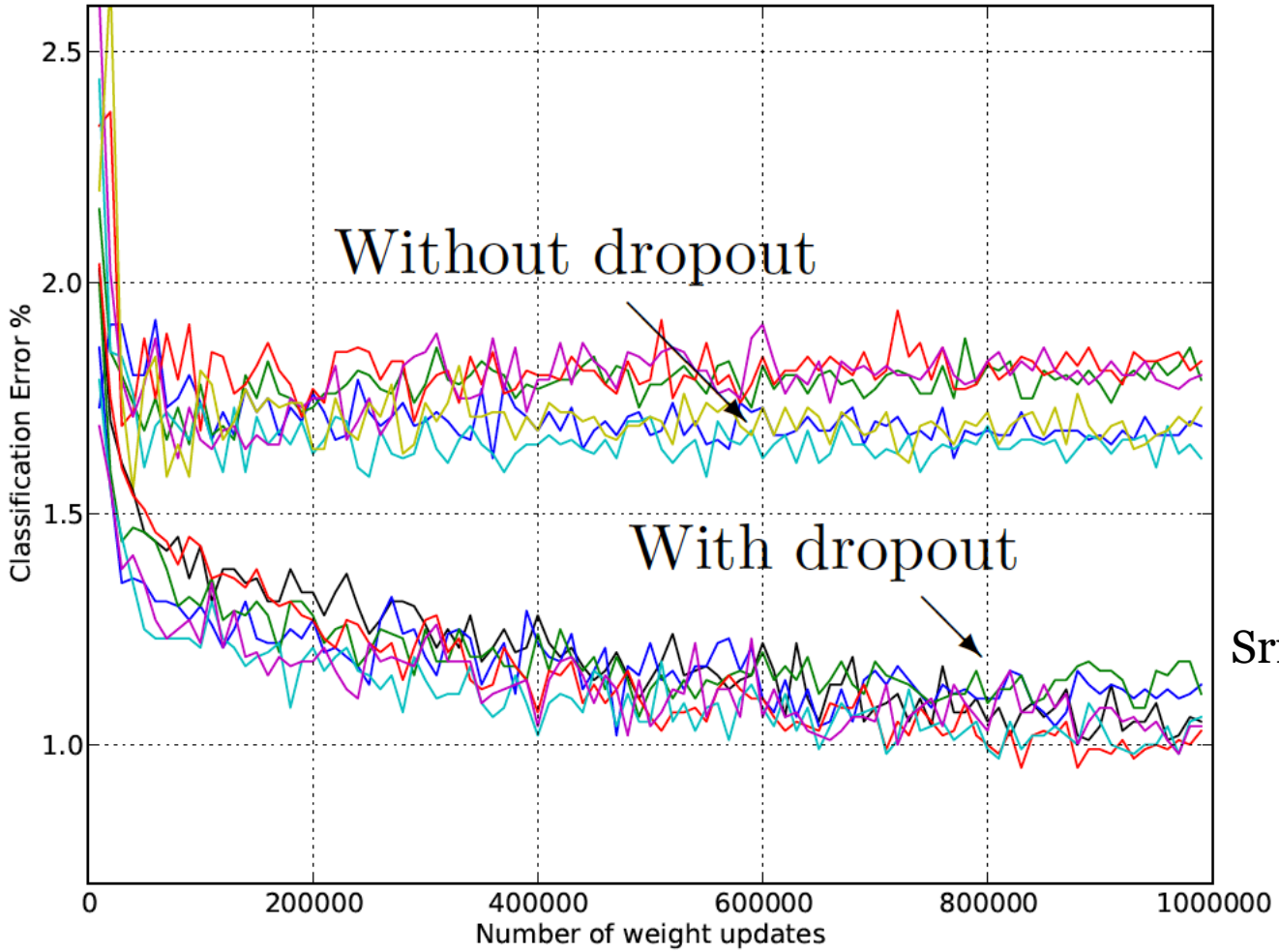


UNIVERSITY OF WATERLOO

Dropout regularizes

- Dropout typically reduces overfitting with a given architecture
- Dropout may allow increase of model size with little overfitting, further improving performance at a greater computational cost

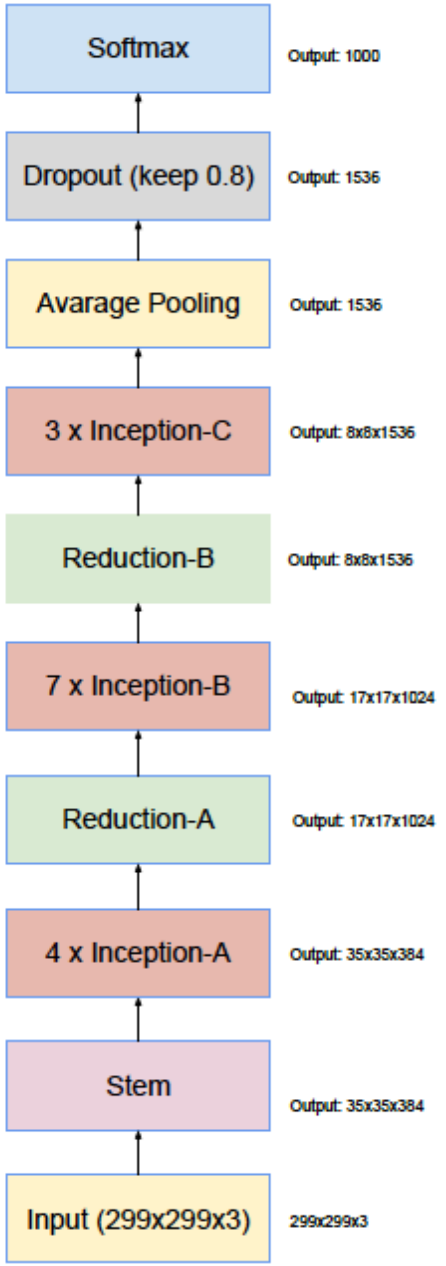
Test error on MNIST with various network architectures



Srivastava et al. (2014) JMLR

Libraries usually treat dropout as a layer

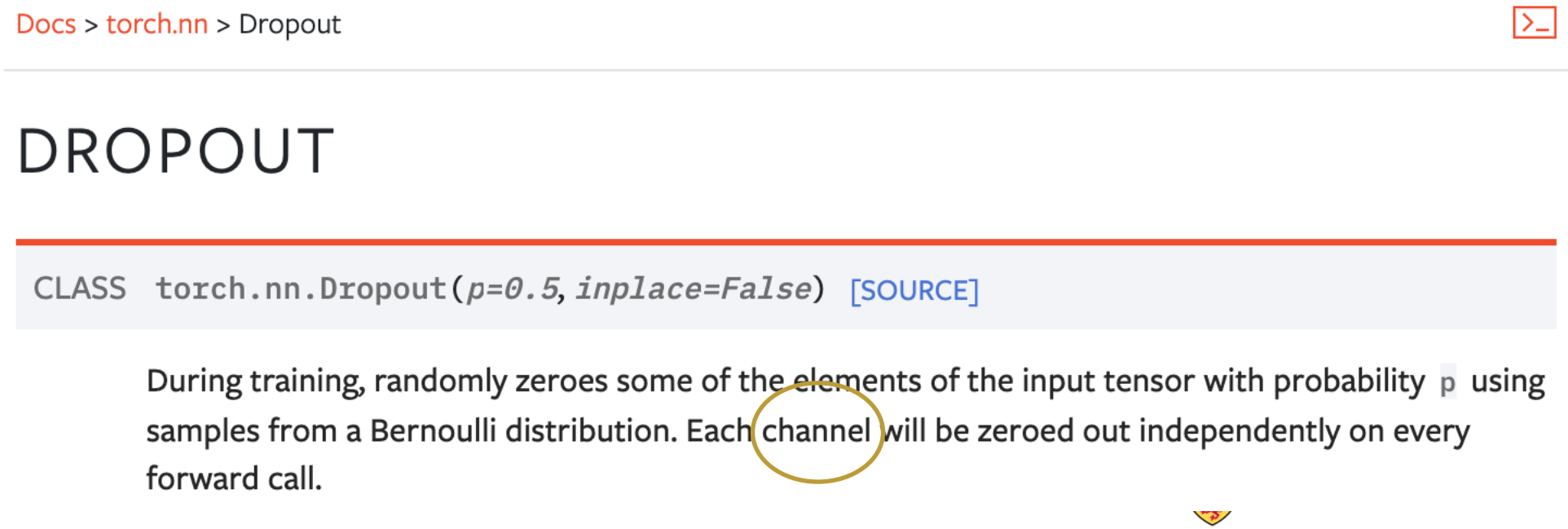
- Implemented in modern deep learning libraries as a separate layer that zeros out its inputs randomly, e.g.
  - torch.nn.Dropout
  - tf.keras.layers.Dropout
- Usually applied selectively to a small number of layers



Szegedy et al. (2017)

Dropout can be applied channel-wise in CNNs

- In convolutional neural networks, independent dropout of individual units may be less effective because neighbouring units' activities are correlated
- An alternative is to drop entire feature maps (this is the default in PyTorch; in TensorFlow it's available as SpatialDropout)



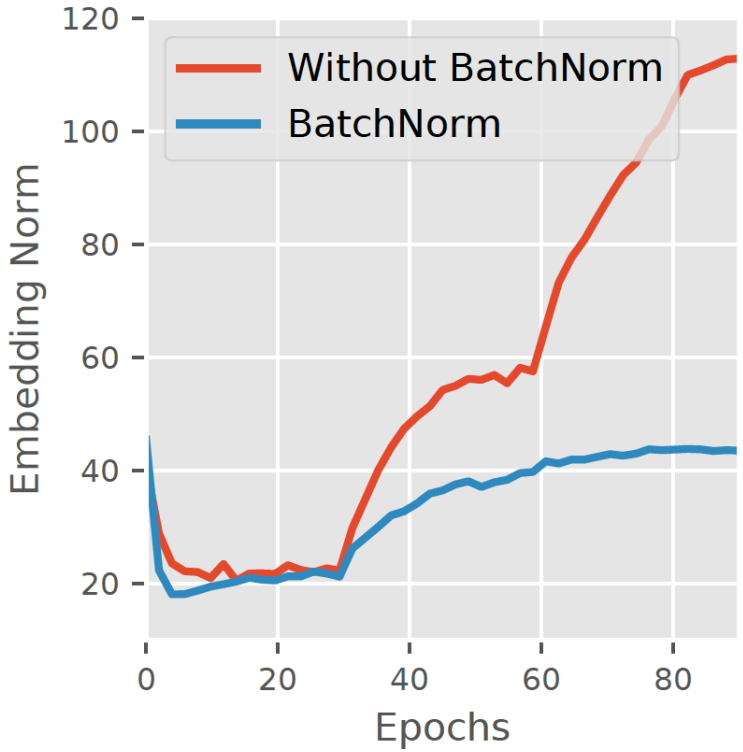
BATCH NORMALIZATION IS ALSO A REGULARIZER

Batch norm as a regularizer

- Although batch norm was intended to simplify optimization, it has also been found to improve generalization
- Batch norm can reduce or eliminate the need for dropout
- Recall that batch norm prevents updates that increase the activation variance
  - This can reduce growth of weights
  - But it is not the same thing as an L^2 penalty, e.g., large positive and negative weights from similar inputs could partly cancel each other, leading to moderate activation variance

Batch norm as a regularizer

- Dauphin & Cubuk (2020) showed that:
  - One effect of batch norm is to reduce the norm of the features produced by the network, just before the output layer
  - Applying a norm penalty on the features directly recovers much of the benefit of batch norm
  - Dropout can have a similar effect, so it may be partly redundant with batch norm



Summary

1. Overfitting can be reduced by simplifying the model, using more training data, or regularization
2. Training data can often be augmented to increase its apparent size
3. Overfitting can be reduced by training on multiple datasets
4. Overfitting tends to develop during training, and early stopping can reduce it
5. Parameter norm penalties shrink weights selectively
6. Dropout is a practical approximation of bagging for large networks
7. Batch normalization is also a regularizer

References

- Bishop (2006) *Pattern Recognition & Machine Learning*, Springer
- Cubuk, E. D., Zoph, B., Shlens, J., & Le, Q. V. (2020). Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops* (pp. 702-703).
- Dauphin, Y., & Cubuk, E. D. (2020, October). Deconstructing the regularization of batchnorm. In *International Conference on Learning Representations*.
- DeVries, T., & Taylor, G. W. (2017). Dataset augmentation in feature space. arXiv preprint arXiv:1702.05538.
- Goodfellow, Bengio & Courville (2016) *Deep Learning*, MIT Press
- Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 1345-1359.

References

- Prechelt, L. (2002). Early stopping-but when?. In Neural Networks: Tricks of the trade (pp. 55-69). Berlin, Heidelberg: Springer Berlin Heidelberg.
- Schlüter, J., & Grill, T. (2015, October). Exploring data augmentation for improved singing voice detection with neural networks. In ISMIR (pp. 121-126).
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958.
- Sun, C., Shrivastava, A., Singh, S., & Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 843-852).
- Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. (2017, February). Inception-v4, inception-resnet and the impact of residual connections on learning. In Proceedings of the AAAI conference on artificial intelligence (Vol. 31, No. 1).