Department of Mathematics
and Computer Science

Advanced Machine Learning

Regularization

Lukas Galke Poech

Spring 2026

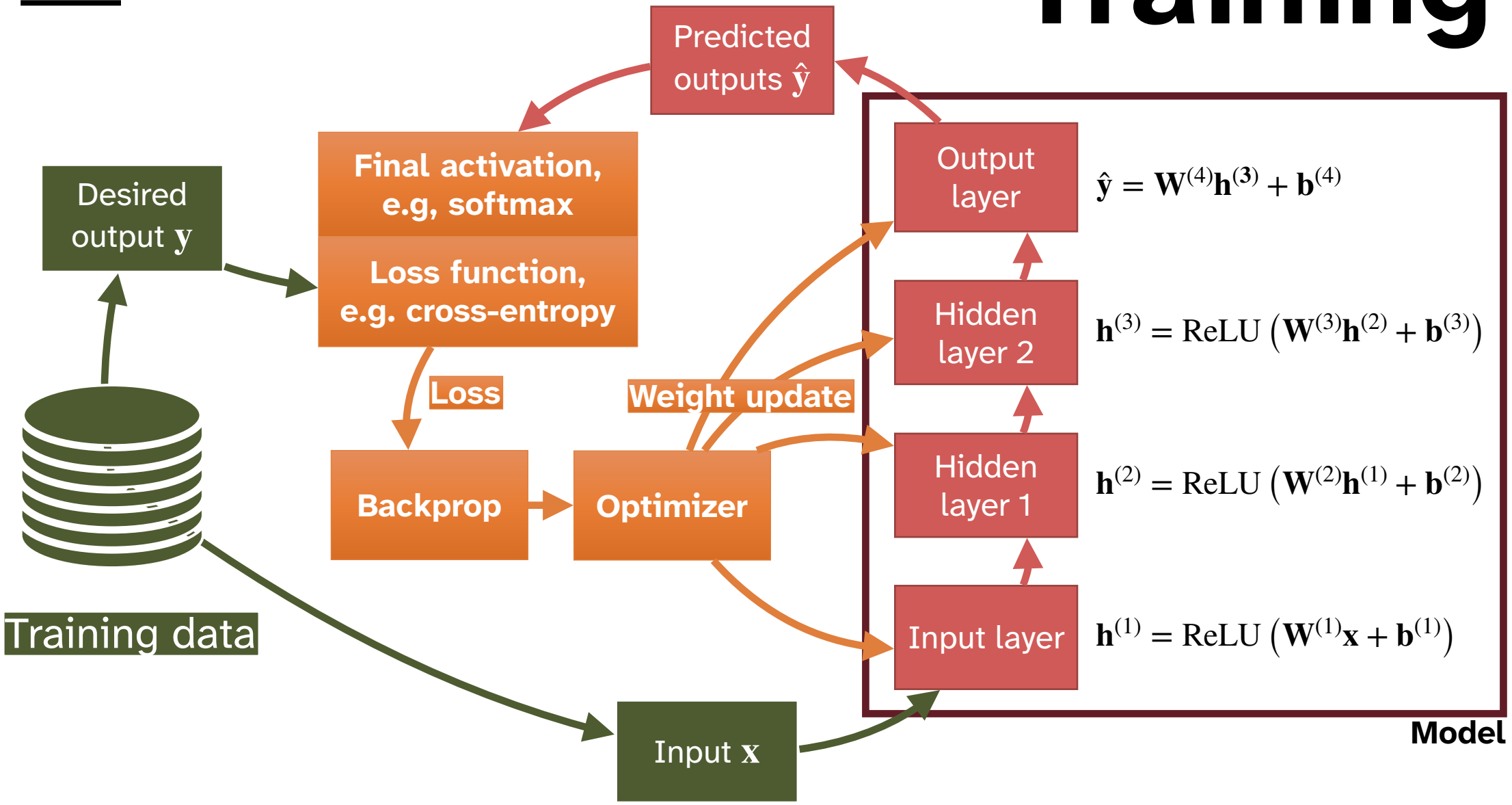
Keywords

- Bias-Variance Trade-off
- Regularization
- Model Capacity
- Invariance/equivariance
- Dropout

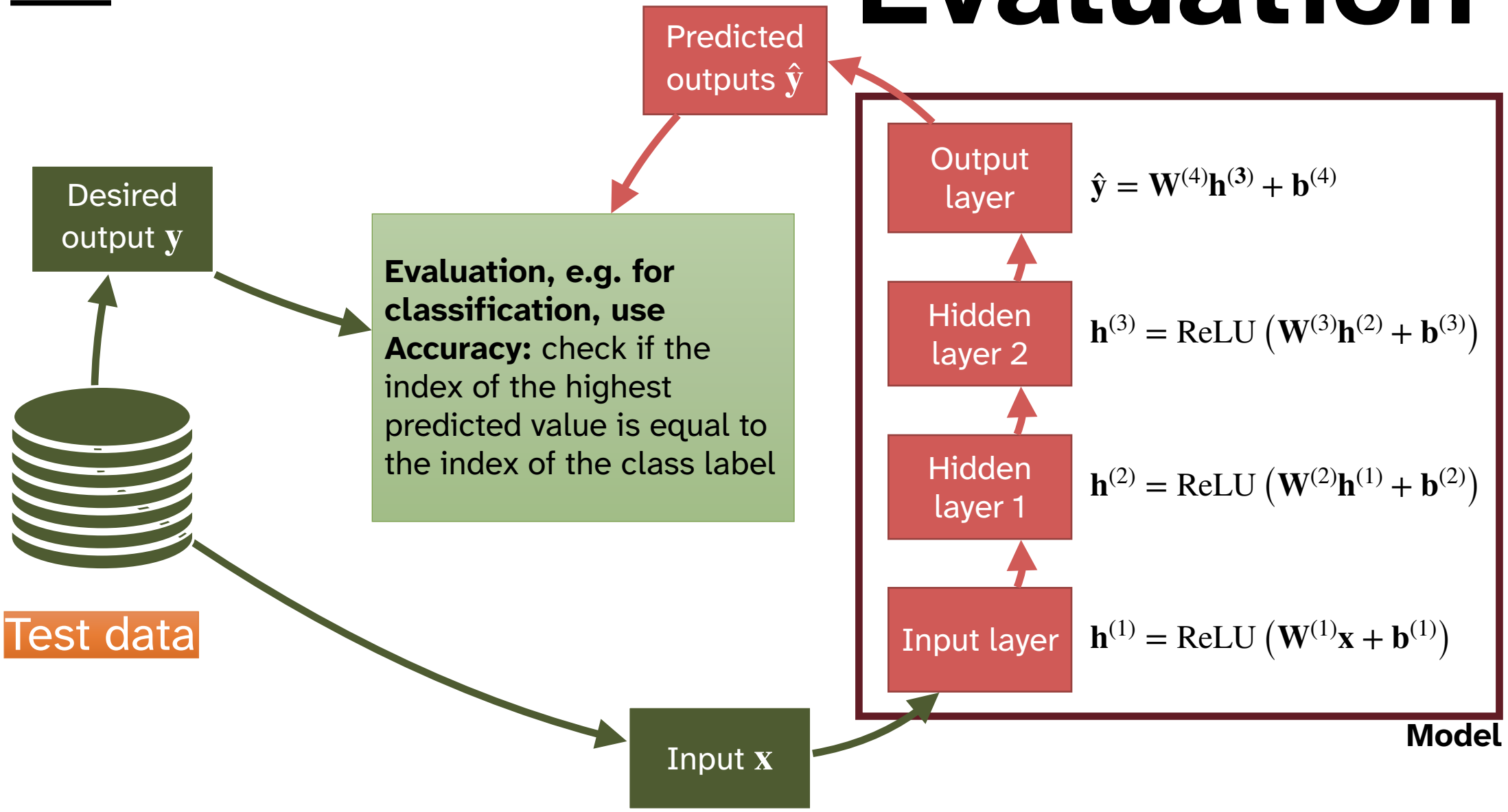
Recap

- Linear models – even multi-layered – are not enough to solve a simple XOR problem.
- **Nonlinear activation functions** allow us to model more complex functions
- **Universal Approximation**; already with a few hidden layers and nonlinearities, a neural network model can **approximate any compact function to an arbitrary degree of accuracy, depending on the width of the network.**
- So, can we just make models very large and we're good?

Training


Predicted outputs $\hat{\mathbf{y}}$
Final activation, e.g, softmax
Loss function, e.g. cross-entropy
Desired output $\mathbf{y}$
Loss
Weight update
Backprop
Optimizer
Training data
Input $\mathbf{x}$
Output layer
Hidden layer 2
Hidden layer 1
Input layer
$\hat{\mathbf{y}} = \mathbf{W}^{(4)}\mathbf{h}^{(3)} + \mathbf{b}^{(4)}$
$\mathbf{h}^{(3)} = \mathrm{ReLU}\left(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}\right)$
$\mathbf{h}^{(2)} = \mathrm{ReLU}\left(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\right)$
$\mathbf{h}^{(1)} = \mathrm{ReLU}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$
Model

Evaluation


Predicted outputs $\hat{\mathbf{y}}$
Desired output $\mathbf{y}$
Evaluation, e.g. for classification, use Accuracy: check if the index of the highest predicted value is equal to the index of the class label
Output layer
$\hat{\mathbf{y}} = \mathbf{W}^{(4)}\mathbf{h}^{(3)} + \mathbf{b}^{(4)}$
Hidden layer 2
$\mathbf{h}^{(3)} = \text{ReLU}\left(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}\right)$
Hidden layer 1
$\mathbf{h}^{(2)} = \text{ReLU}\left(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\right)$
Input layer
$\mathbf{h}^{(1)} = \text{ReLU}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$
Model
Test data
Input $\mathbf{x}$

~Recap: Cross Entropy Loss

Cross-Entropy measures how well one probability distribution approximates

$$H(p \mid q) = -\sum_{i}^{K} p(y_i) \cdot \log q(y_i)$$

Softmax normalizes an arbitrary vector into a probability distribution:

$$\operatorname{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j^K e^{z_j}}$$

Concretely: First **Softmax**, then **Cross-Entropy** against ground-truth.

Essential tool: KL Divergence

$$D_{\mathrm{KL}}(P\,||\,Q) = \sum_{x \in \mathcal{X}} P(x) \cdot \log \frac{P(x)}{Q(x)}$$

- EXPECTATION of the LOGARITHMIC DIFFERENCE between TWO PROBABILITY DISTRIBUTIONS P and Q, where EXPECTATION is taken over P
- Note the similarity to Cross-Entropy. *More on that later!*

KL-Bias and KL-Variance

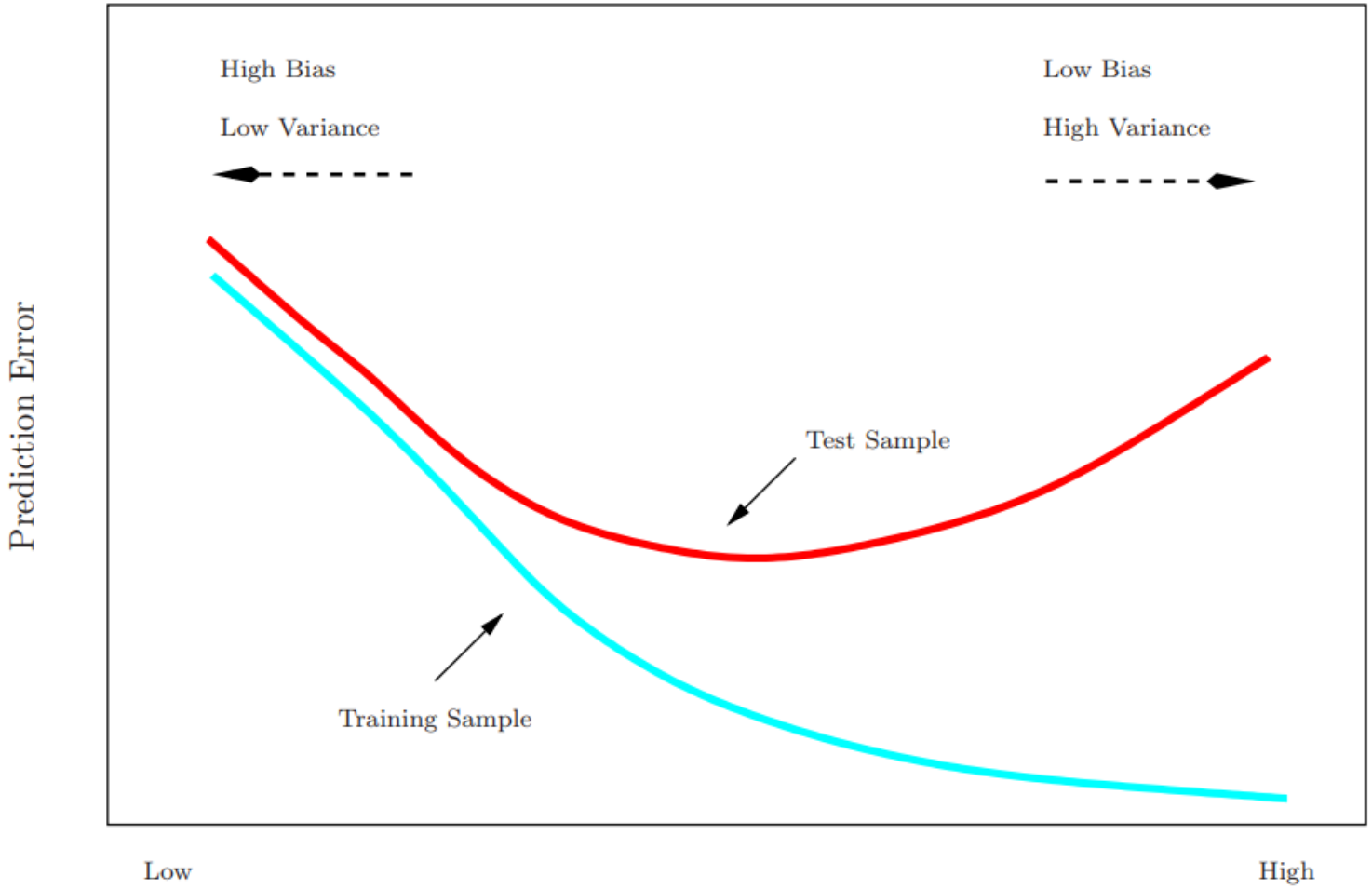
$$\underbrace{\mathbb{E}_\varepsilon\left[\mathrm{CE}(y, f_\varepsilon)\right]}_{\textsc{Error}} = \mathbb{E}_\varepsilon\left[\sum_{c=1}^{C} y[c]\log(f_\varepsilon[c])\right] = \underbrace{D_{\mathrm{KL}}\left(y\|\bar{f}\right)}_{\textsc{Bias}^2} + \underbrace{\mathbb{E}_\varepsilon\left[D_{\mathrm{KL}}(\bar{f}\|f_\varepsilon)\right]}_{\textsc{Variance}},$$

Here: f_ϵ is the model and ϵ denotes the randomness from training or from other sources of randomness at test time. $\bar{f}$ denotes the average prediction under this randomness.

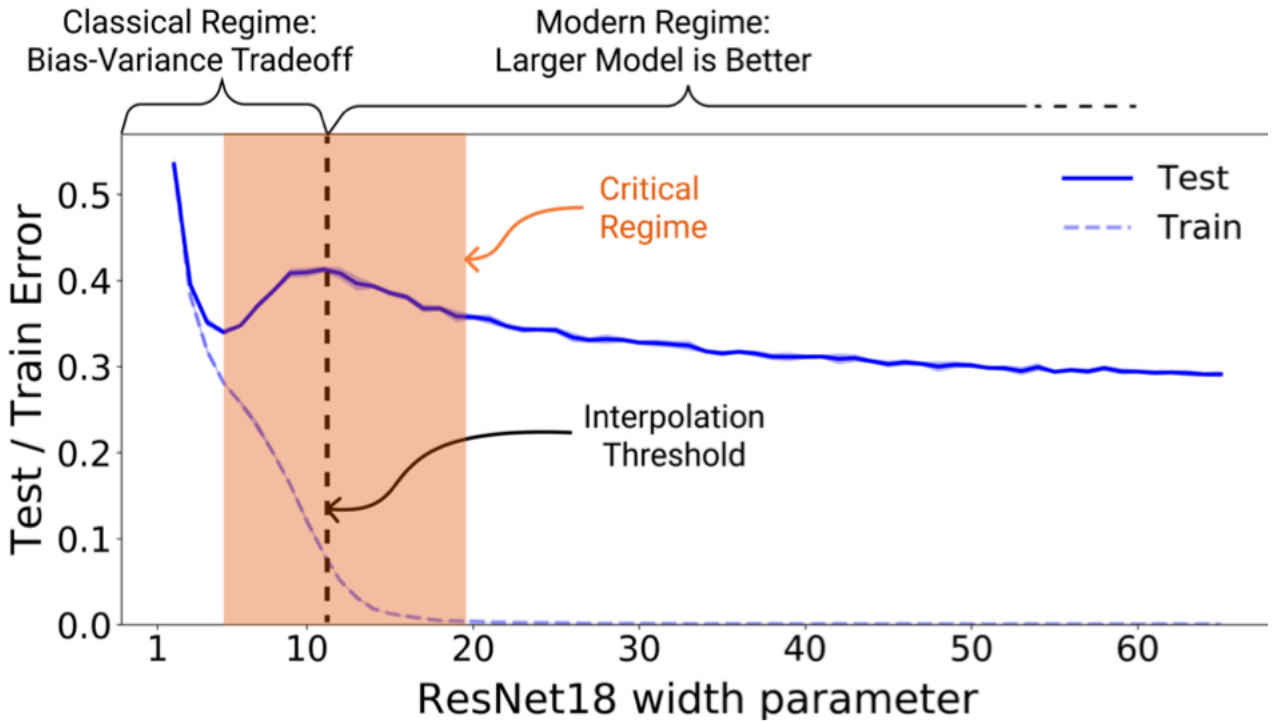
tldr; Error = Bias^2 + Variance

Hägele et al. (2026) https://arxiv.org/abs/2601.23045

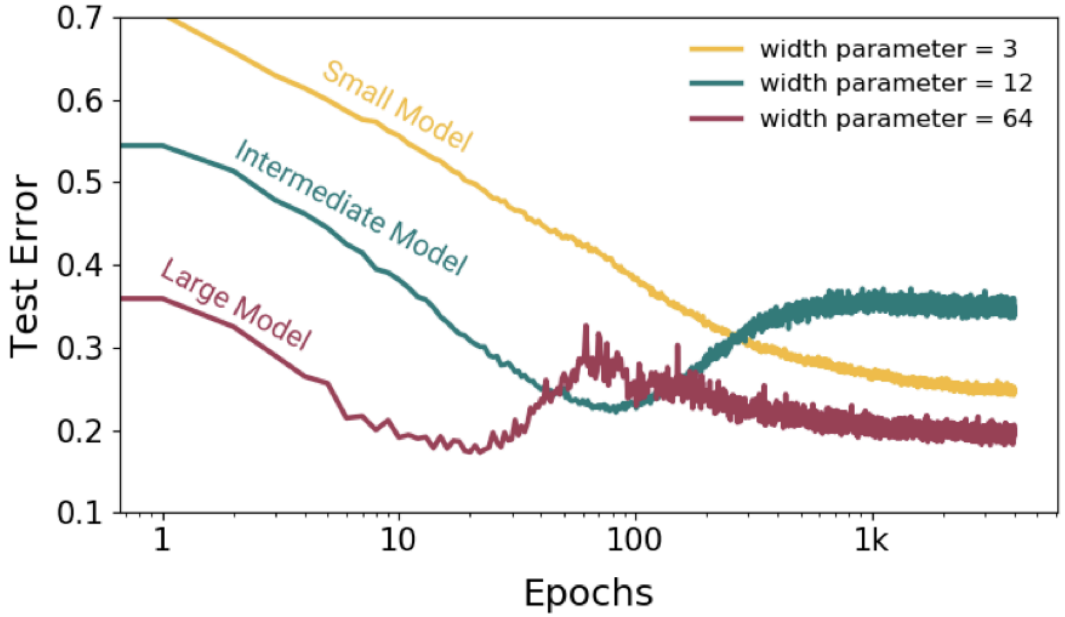
Bias Variance Trade-off



Deep Double Descent



~Model Capacity



~Training time

Source: Deep Double Descent, https://arxiv.org/abs/1912.02292

Generalization

- Central challenge of ML is that the algorithm must perform well on new, **previously unseen** inputs
- Ability to perform well on previously unobserved inputs is called **generalization**
- Normally we have a training and test dataset
- Difference to normal optimization
  - We try to minimize the error on unseen data not on the training data, e.g. in the case of a linear regression: $\hat{y} = \boldsymbol{w}^T \boldsymbol{x} + b$

$$\frac{1}{m^{(\text{test})}} \left\| X^{(\text{test})} \boldsymbol{w} - \boldsymbol{y}^{(\text{test})} \right\|_2^2$$

Estimating the generalization error

➢We optimize our parameters by minimizing the training error

$$\frac{1}{m^{(\text{train})}}\left\| X^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}\right\|_2^2$$

➢**BUT**: We are actually interested in the test error:

$$\frac{1}{m^{(\text{test})}}\left\| X^{(\text{test})}\boldsymbol{w} - \boldsymbol{y}^{(\text{test})}\right\|_2^2$$

➢**How can we affect performance when we observe only the training set?**

Under- and Over-fitting

- Factors determining how well an ML algorithm will perform are its ability to:
  1. Make the training error small
  2. Make gap between training and test errors small
- **Underfitting**
  - Inability to obtain low enough error rate on the training set
- **Overfitting**
  - Gap between training error and testing error is too large
  - We can control whether a model is more likely to overfit or underfit by altering its capacity

Capacity of a model

- Model capacity is ability to **fit variety of functions**
  - Model with low-capacity struggles to fit training set
  - A high-capacity model can overfit by memorizing properties of training set which are not useful on test set
- When a model has higher capacity, it may overfit
  - One way to control capacity of a learning algorithm is by choosing the hypothesis space
    - i.e., set of functions that the learning algorithm is allowed to select as being the solution
    - e.g., the linear regression algorithm has the set of all linear functions of its input as the hypothesis space
  - **Regularization**

Representational and Effective Capacity

Representational capacity:

➢Specifies family of functions learning algorithm can choose from

Effective capacity:

➢Imperfections in optimization algorithm can limit representational capacity

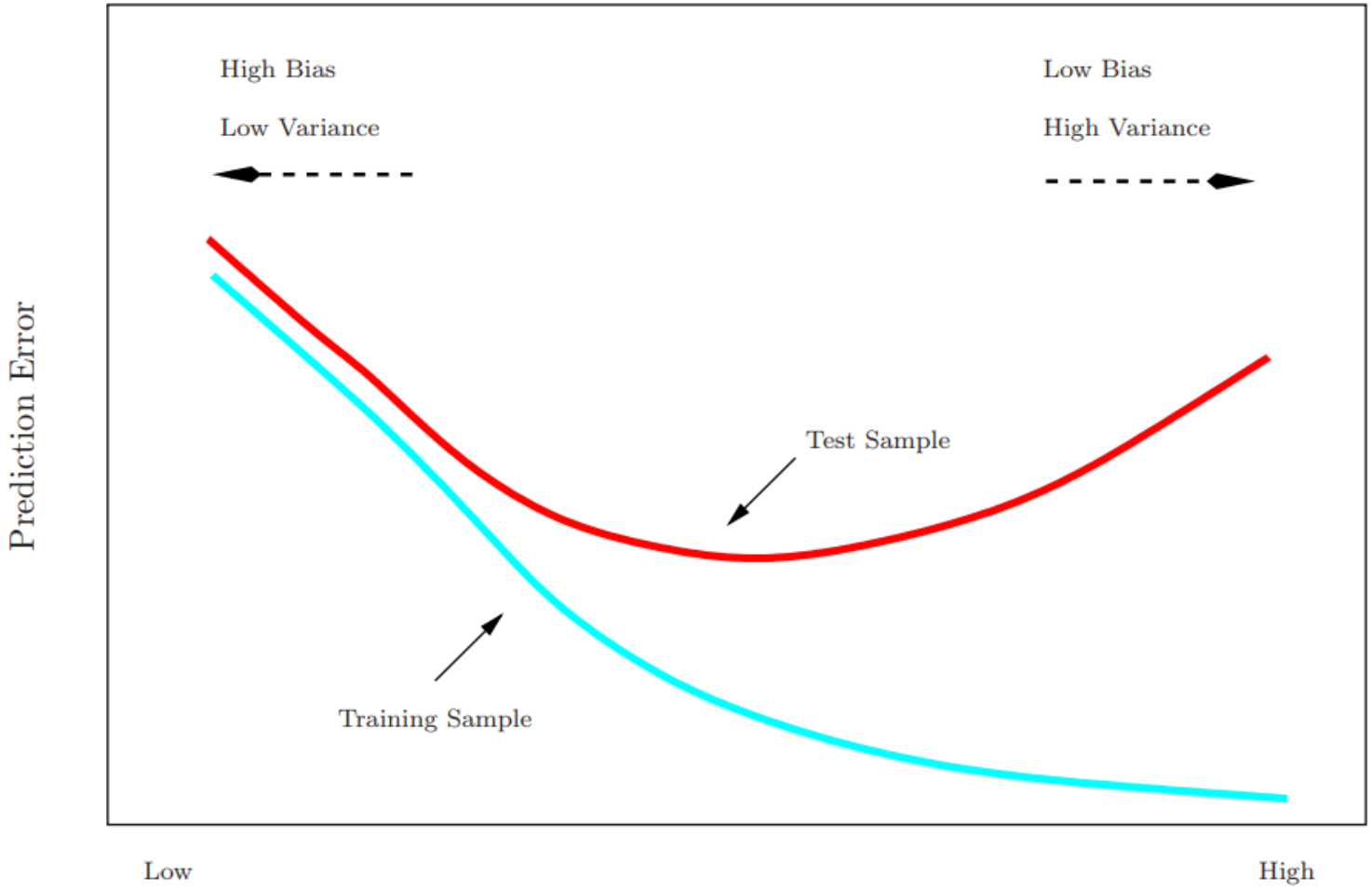
Occam's razor:

➢Among competing hypotheses that explain known observations equally well, choose the simplest one

Training Error versus Test error

- Recall the distinction between the **test error** and the **training error**:
  - The test error is the average error that results from using a statistical learning method to predict the response on a new observation, one that was not used in training the method.
  - In contrast, the training error can be easily calculated by applying the statistical learning method to the observations used in its training.
  - **But the training error rate often is quite different from the test error rate, and in particular the former can dramatically underestimate the latter.**

Training- versus Test-Set Performance



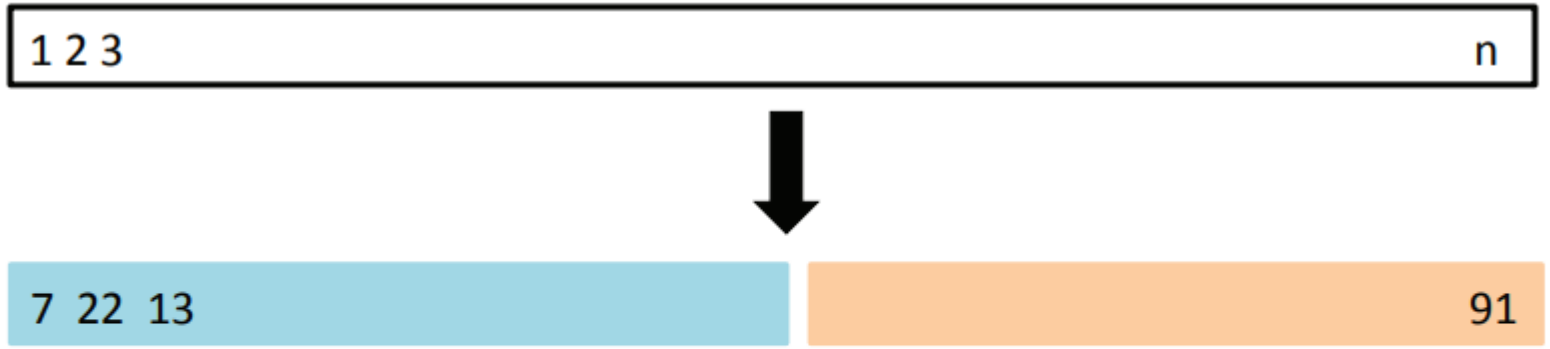
Validation-set approach

- Here we randomly divide the available set of samples into two parts: a **training set** and a **validation** or **hold-out set**.

- The model is fit on the training set, and the fitted model is used to predict the responses for the observations in the validation set.

- The resulting validation-set error provides an estimate of the test error. This is typically assessed using MSE in the case of a quantitative response and misclassification rate in the case of a qualitative (discrete) response.

Validation-set approach



A random splitting into two halves: left part is training set, right part is validation set

Drawbacks of Validation Set Approach

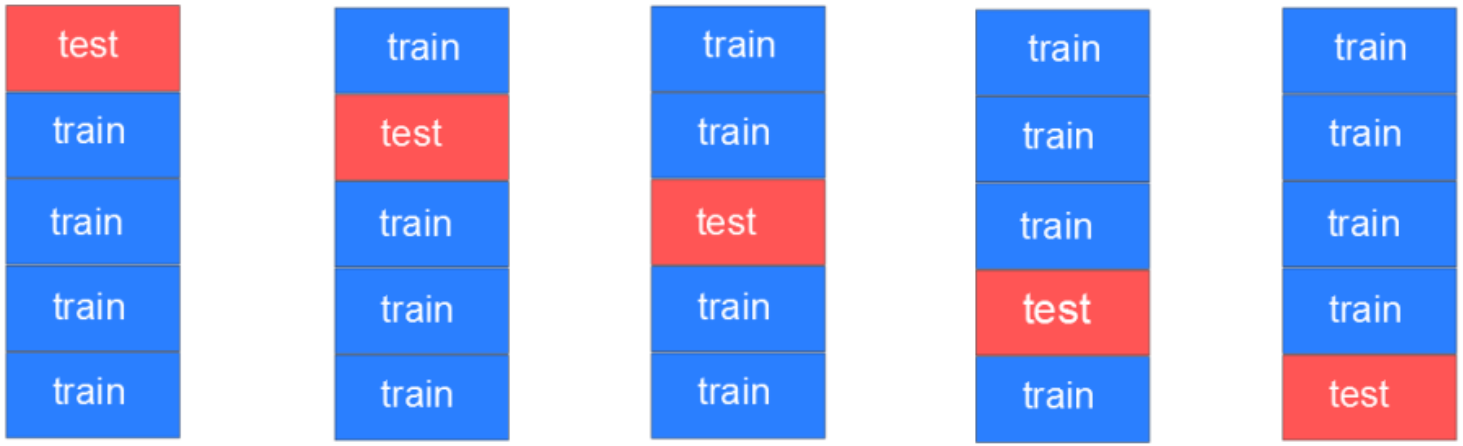
- The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.

- In the validation approach, only a subset of the observations - those that are included in the training set rather than in the validation set - are used to fit the model.

- This suggests that the validation set error may tend to overestimate the test error for the model fit on the entire data set.

Cross-Validation

- When data set is too small, a fixed test set is problematic
- k-fold cross-validation:
  - All available data is partitioned into k groups
  - $k-1$ groups are used to train and evaluated on remaining group
  - Repeat for all k choices of held-out group
  - Performance scores from k runs are averaged



Keywords

- Bias-Variance Trade-off
- **Regularization**
- Model Capacity
- Invariance/equivariance
- Dropout

What we have seen so far

- Models can overfit easily, especially when they have high capacity
  - We have seen, that we can change the capacity of the simple models
  - E.g., just fit polynomials of lower degree

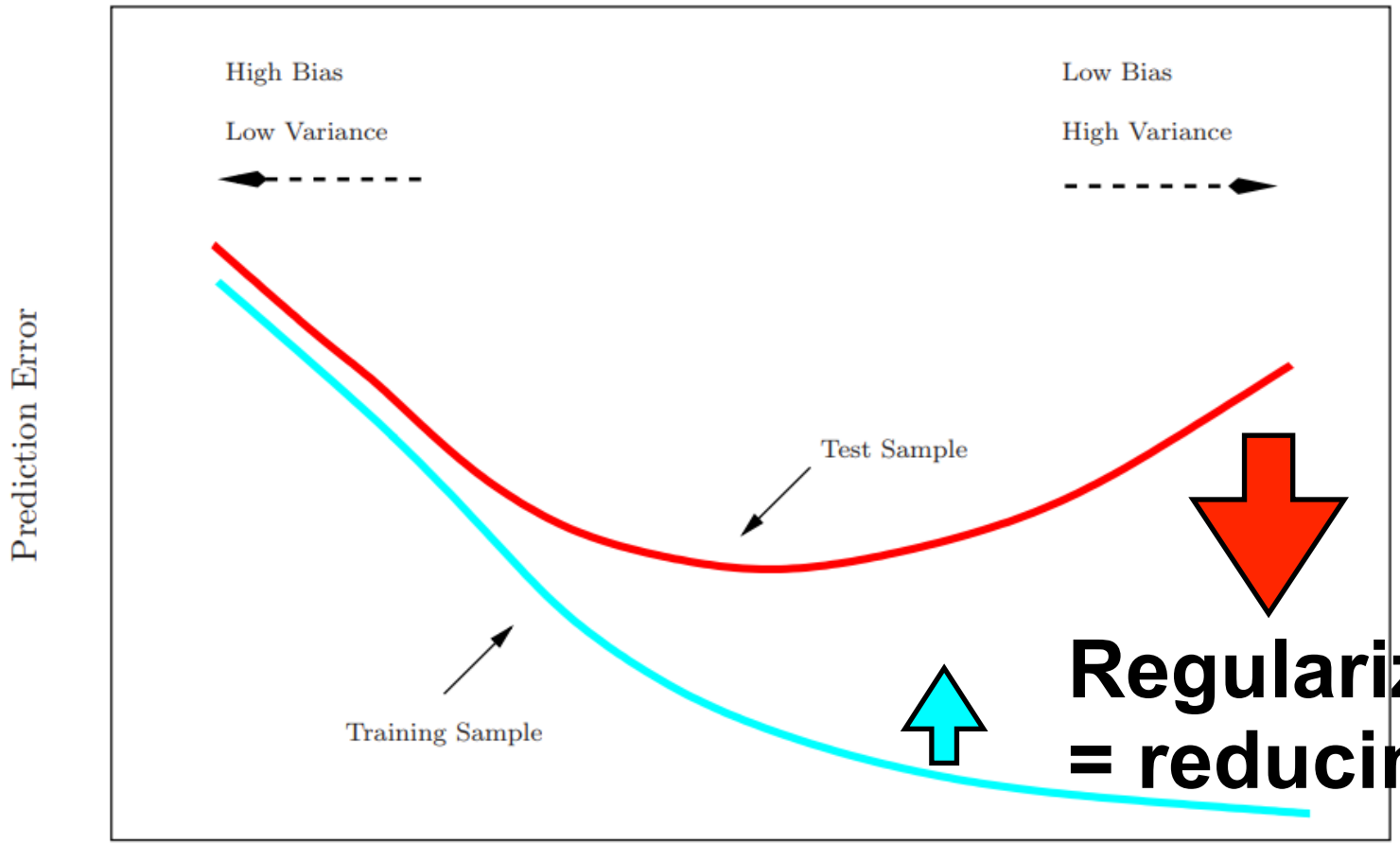
- Deep Neural Networks have very high capacity and thus we need measures to tame them → **Regularization**

What is Regularization?

- Central problem of ML is to design algorithms that will perform well not just on training data but on new inputs as well → **We want generalization!**
- **Regularization** is:
  - "any modification we make to a learning algorithm to reduce its generalization error but not its training error"
  - **Reduce test error even at the expense of (slightly) increasing training error**

- Some goals of regularization
  1. Encode prior knowledge
  2. Express preference for simpler model
  3. Needed to make underdetermined problem determined

Regularization



Regularization
= reducing overfitting

Model Types and Regularization

➢Three regimes of models:

1. Underfitting: Not capturing the data distribution well (high bias)
2. Matches the true data generating process
3. Overfitting: Capturing the data distribution too well

➢Goal of regularization is to take model from third regime to second

➢Best fitting model obtained not by finding the right number of parameters

➢Instead, **best fitting model is a large model that has been regularized appropriately**

Keywords

- Bias-Variance Trade-off
- Regularization
- **Model Capacity**
- Invariance/equivariance
- Dropout

Limiting Model Capacity

- Regularization has been used for decades prior to advent of Deep Learning
- Linear- and logistic-regression allow simple, straightforward and effective regularization strategies:
  - Adding a parameter norm penalty $\Omega(\theta)$ to the objective function J:

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

- with α is a hyperparameter that scales the relative contribution of the norm penalty term Ω

- Setting α to 0 results in no regularization. Larger values correspond to more regularization

Norm Penalty

- When our training algorithm minimizes the regularized objective function J and some measure of the size of the parameters θ

- Different choices of the parameter norm Ω can result in different solutions preferred

- Norm penalty Ω penalizes only weights at each layer and leaves biases unregularized
  - Biases require less data to fit than weights
  - Each bias controls only a single variable

- Let $\boldsymbol{w}$ indicate all weights affected by norm penalty, θ denotes both $\boldsymbol{w}$ and biases

L^2 parameter Regularization

- Simplest and most common kind
- Called Weight decay
- Drives weights closer to the origin by adding a regularization term to the objective function

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

- In other communities also known as **ridge regression** or Tikhonov regularization

A closer look

➢Objective function (with no bias parameter)

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

➢Corresponding Gradient:

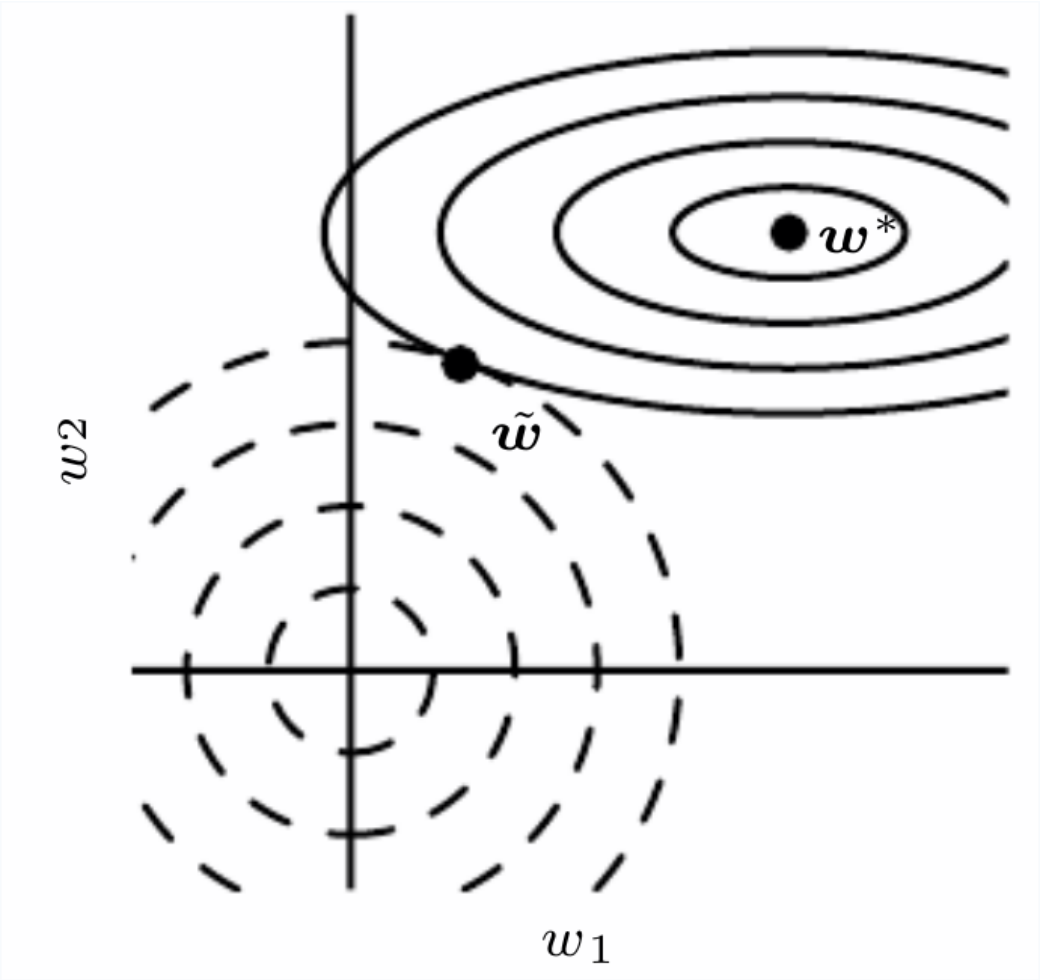
$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

➢To perform single gradient step, perform update:

$$\boldsymbol{w}' \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

➢We have modified learning rule to shrink $\boldsymbol{w}$ by constant factor $1 - \epsilon\alpha$ at each step

An illustration of the effect of weight decay



L^1 Regularization

➢While L^2 weight decay is the most common form of weight decay there are other ways to penalize the size of model parameters

➢L^1 regularization is defined as

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

➢Which is the sum of the absolute values of the individual parameters

A closer look

➢Objective function (with no bias parameter)

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

➢Corresponding Gradient:

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \text{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

➢Where $\text{sign}(\boldsymbol{w})$ is simply the element-wise sign of $\boldsymbol{w}$

- ➢Always a strong gradient unless $w_i = 0$
- ➢Leads to a sparse solution

Sparsity and Feature Selection

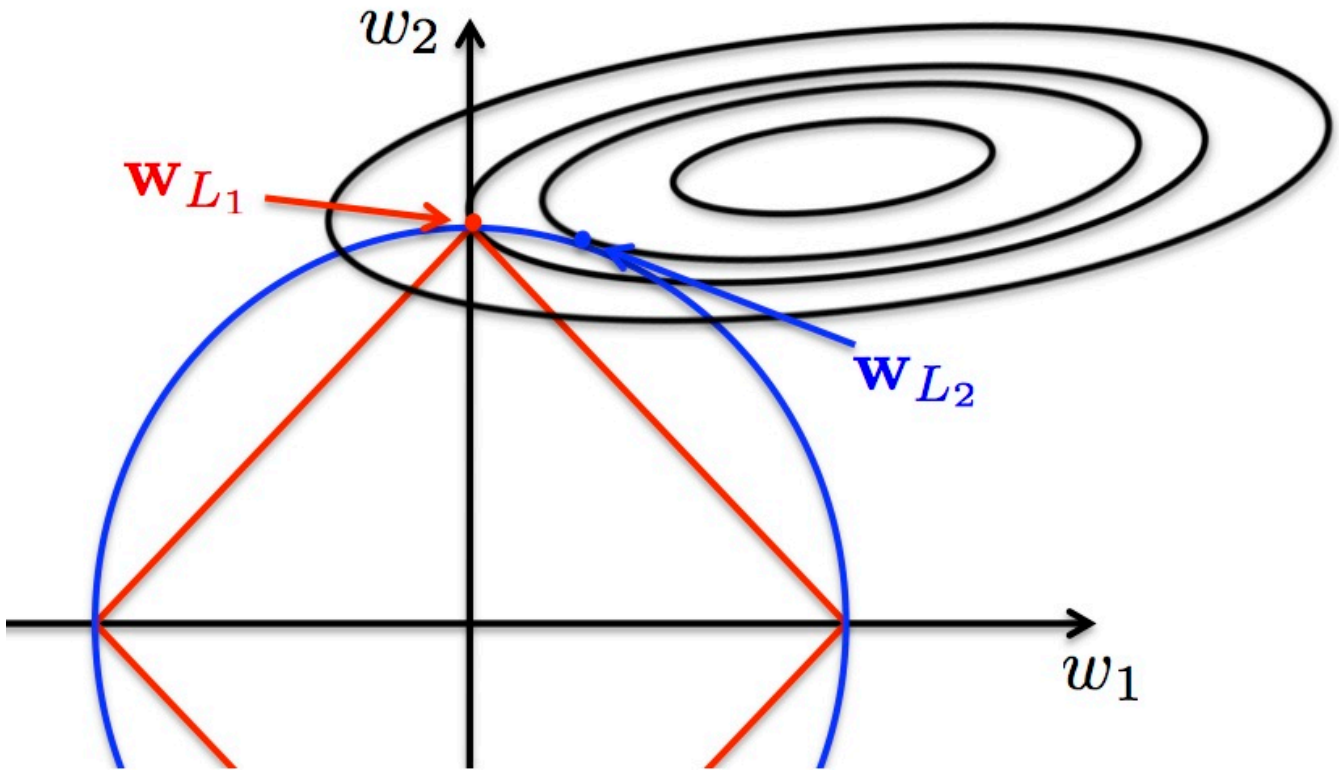
- The sparsity property induced by L^1 regularization has been used extensively as a feature selection mechanism

- Feature selection simplifies an ML problem by choosing subset of available features

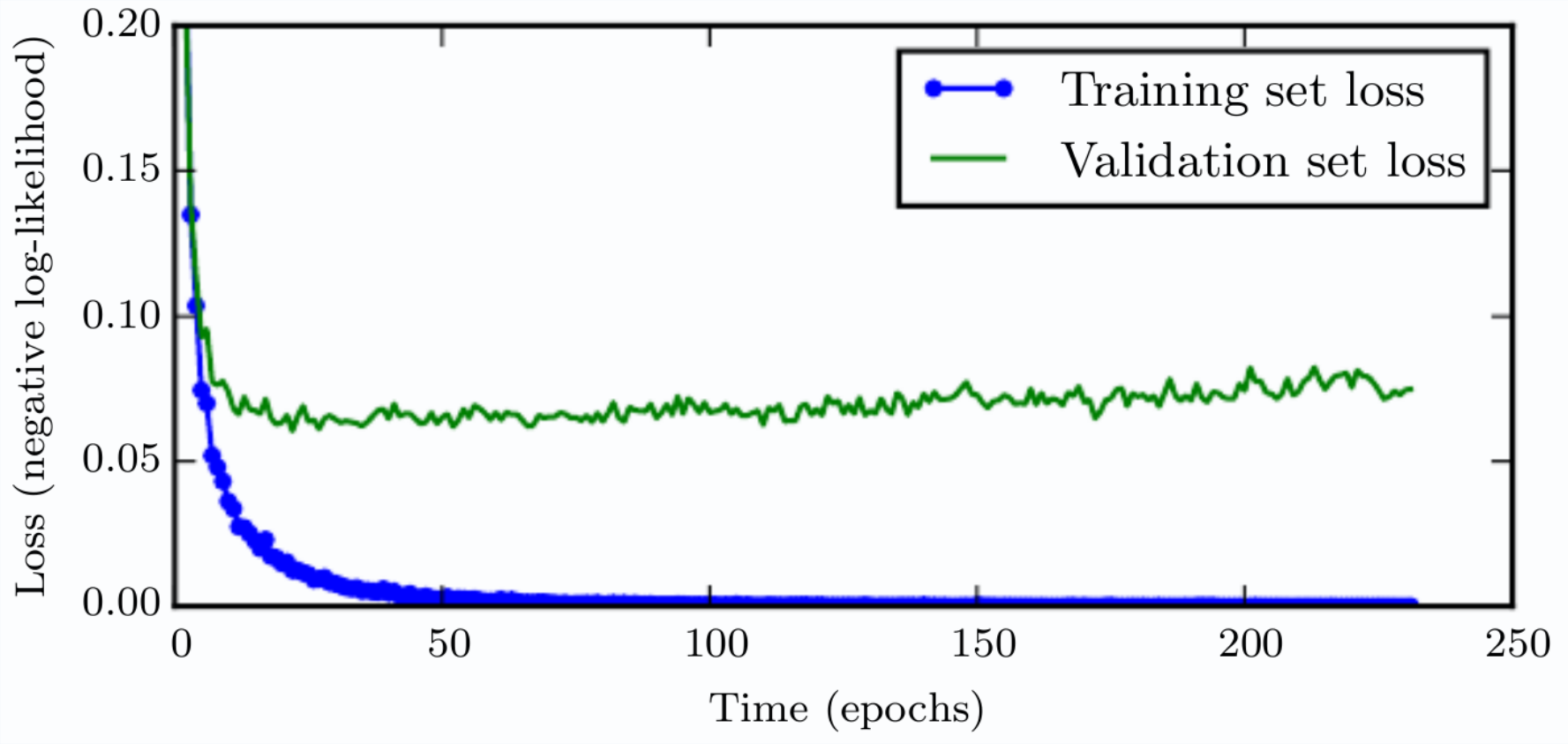
- LASSO (Least Absolute Shrinkage and Selection Operator) integrates an L^1 penalty with a linear model and least squares cost function

- The L^1 penalty causes a subset of the weights to become zero, suggesting that those features can be discarded

L^1 vs. L^2 Regularization



Typical Learning Curves



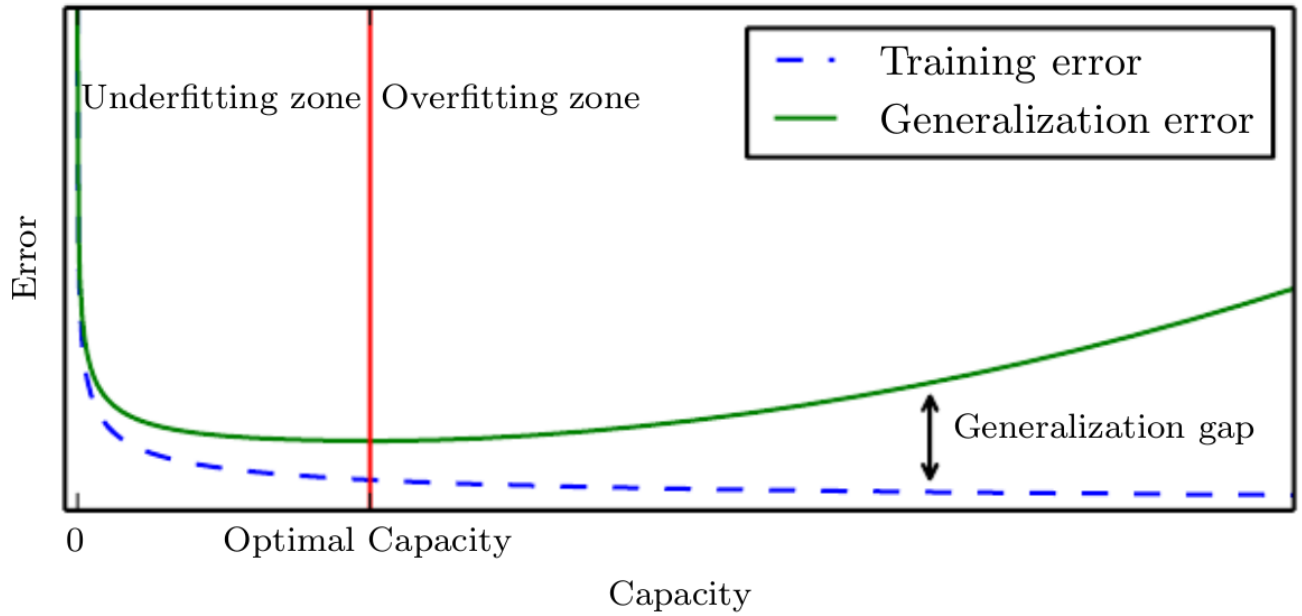
➢ In this example, we train a maxout network on MNIST. Observe that the training objective decreases consistently over time, but the validation set average loss eventually begins to increase again, forming an asymmetric U-shaped curve.

Early Stopping

- We can obtain a model with better validation set error (and thus better test error) by returning to the parameter setting at the point of time with the lowest validation set error
- Every time the error on the validation set improves, we store a copy of the model parameters.
- When the training algorithm terminates, we return these parameters, rather than the latest set
- It is the most common form of regularization in Advanced Machine Learning due to its effectiveness and its simplicity

Early Stopping as Hyperparameter Selection

- We can think of early stopping as a hyperparameter selection algorithm
  - In this view no. of training steps is just a hyperparameter
  - This hyperparameter has a U-shaped validation set performance curve
  - Most hyperparameters have such a U-shaped validation set performance curve, as seen below



Costs of Early Stopping

- Cost of this hyperparameter is running validation evaluation periodically during training
  - Ideally done in parallel to training process on a separate machine
  - Separate CPU or GPU from main training process, Or using small validation set or validating set less frequently

- Need to maintain a copy of the best parameters
  - This cost is negligible because they can be stored on a slower, larger memory
  - E.g., training in GPU, but storing the optimal parameters in host memory or on a disk drive

Early Stopping as Regularization

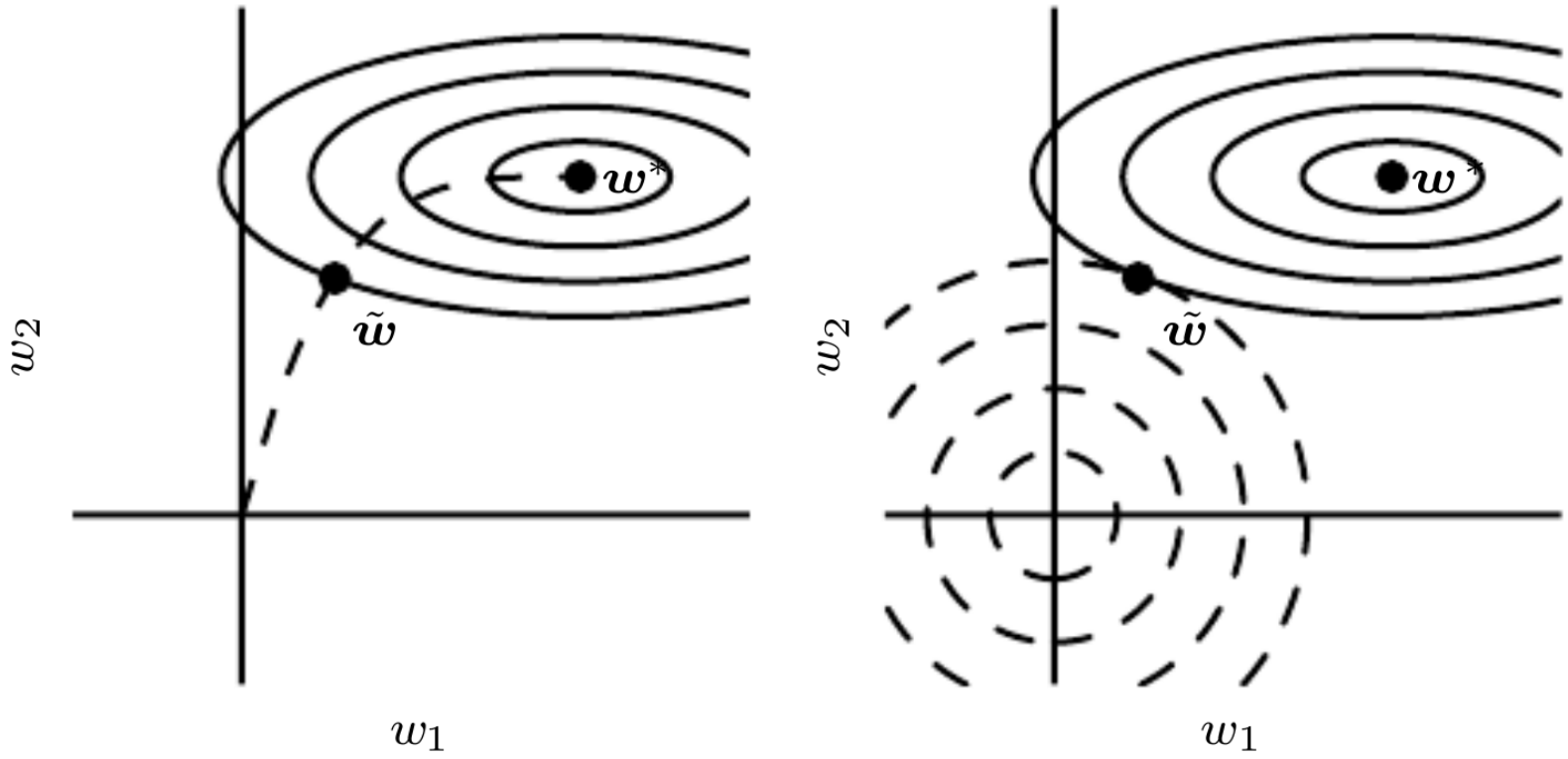
- Early stopping is an unobtrusive form of regularization
- It requires almost no change to the underlying training procedure, the objective function, or the set of allowable parameter values
- So it is easy to use early stopping without damaging the learning dynamics
- In contrast to weight decay, where we must be careful not to use too much weight decay
  - Otherwise we trap the network in a bad local minimum corresponding to pathologically small weights

Use of a second training step

- Early stopping requires a validation set
  - Thus some training data is not fed to the model
- To best exploit this extra data, one can perform extra training after the initial training with early stopping has completed.

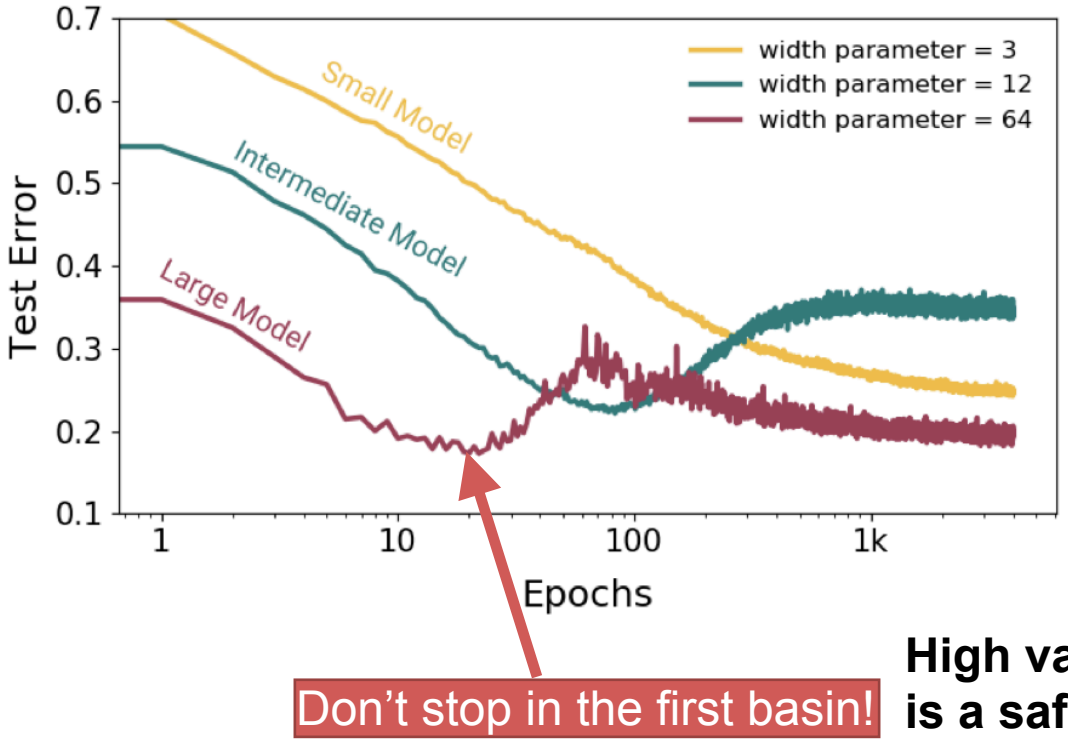
- There are two basic strategies for including the validation data
  1. Reinitialize the model to the untrained state and re-train on training AND validation sets, with the identified optimal hyperparameters – including how long to train the model (number of epochs).
  2. Continue training with the final model on the validation set (e.g., if dealing with very large models)

Early Stopping vs L^2 Regularization



- We now restrict the “distance” the weights can travel to find an optimal solution

Be patient with early stopping



High value for patience is a safe choice!

Keywords

- Bias-Variance Trade-off
- Regularization
- Model Capacity
- **Invariance/equivariance**
- Dropout

More Data is Better

- Best way to make a ML model to generalize better is to train it on more data
- In practice amount of data is limited
- **Get around the problem by creating synthesized data**
- For some ML tasks it is straightforward to synthesize data
- Especially popular for classification/object recognition

Data Augmentation for Classification

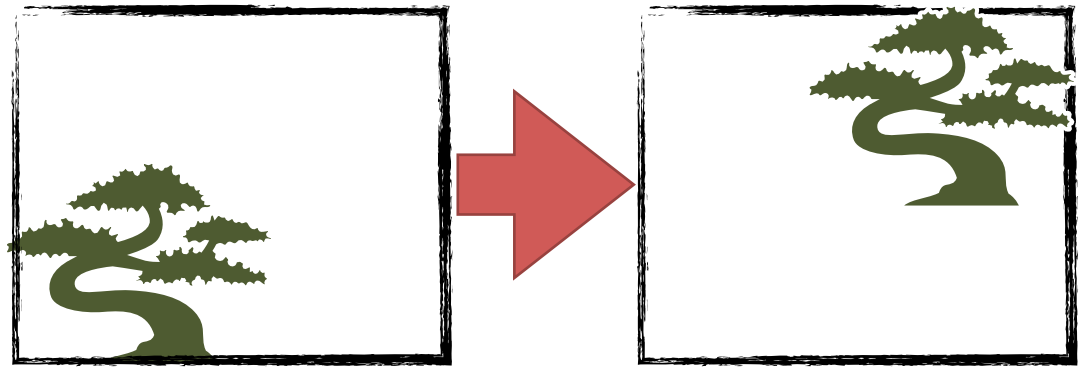
- Data augmentation is easiest for classification
  - Classifier takes high-dimensional input $\boldsymbol{x}$ and summarizes it with a single category identity y
  - Main task of classifier is to be **invariant** to a wide variety of transformations

- Generate new samples $(\boldsymbol{x}, y)$ just by transforming inputs
  - Good example: Images are high-dimensional and include a variety of variations, may easily simulated
  - Translating the images a few pixels can greatly improve performance
  - Rotating and scaling are also effective
  - It may be more difficult in other area (e.g., language)

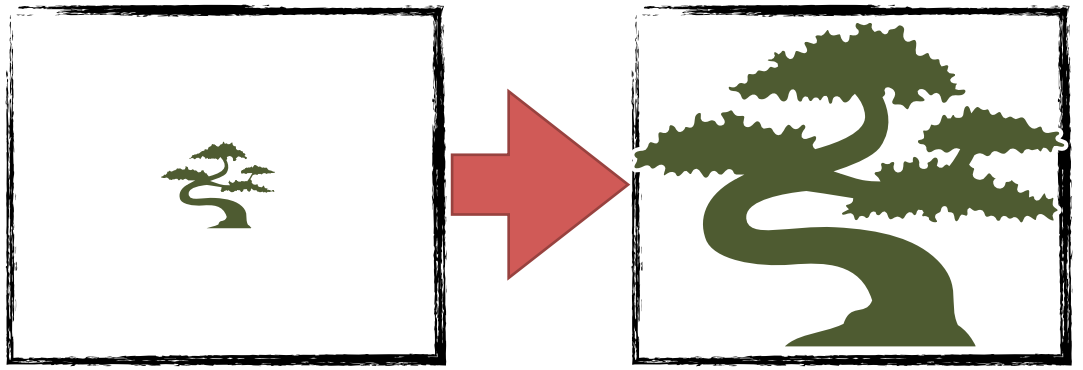
Invariance

Model output remains unchanged with respect to some transformation of the input. **S(T(I)) = S(I)**

Translation invariance



Scale invariance



Equivariance

S(T(I)) = T(S(I))

Model output changes in the same way, regardless of whether the transformation is applied before or after the model is applied.

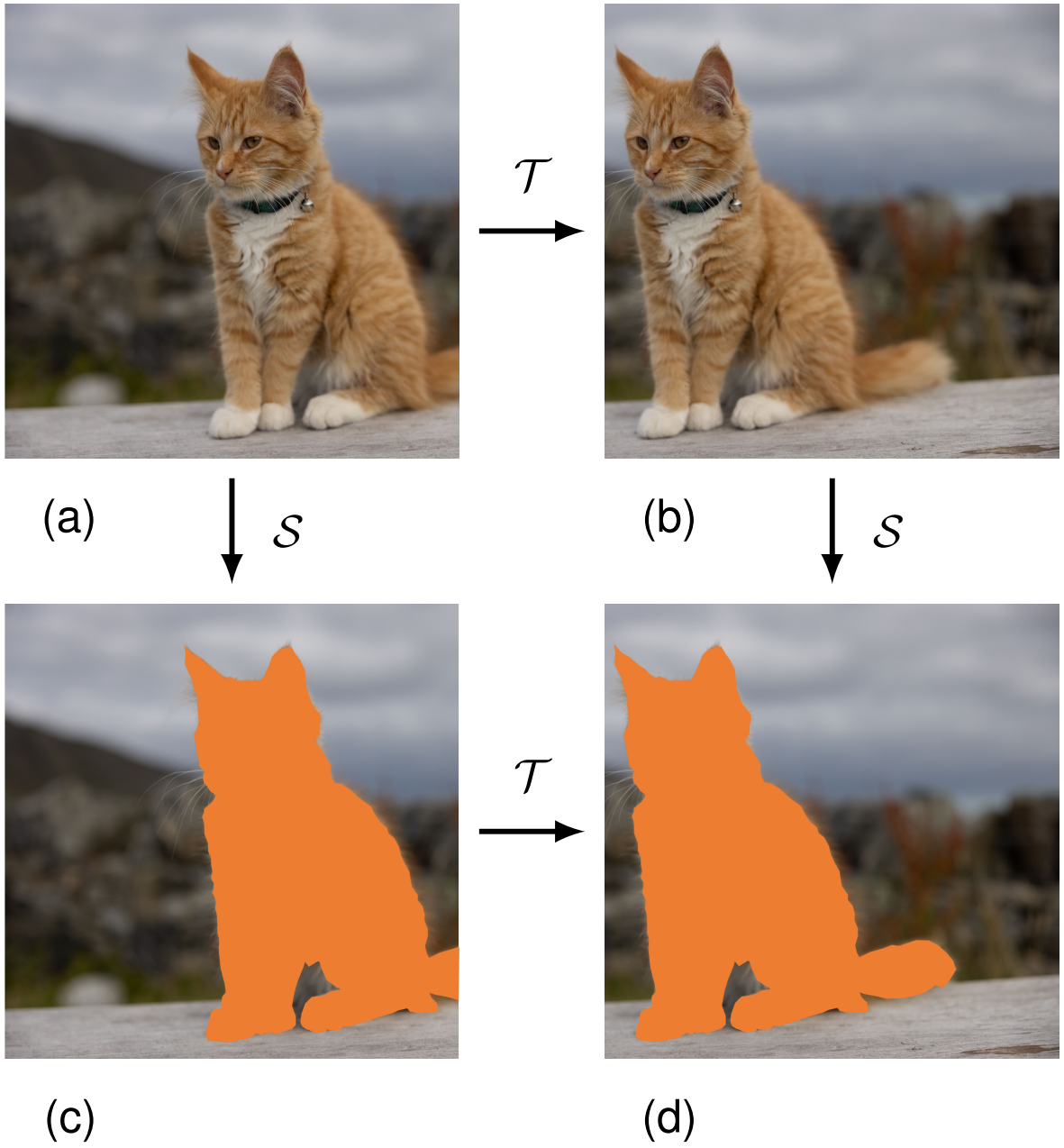
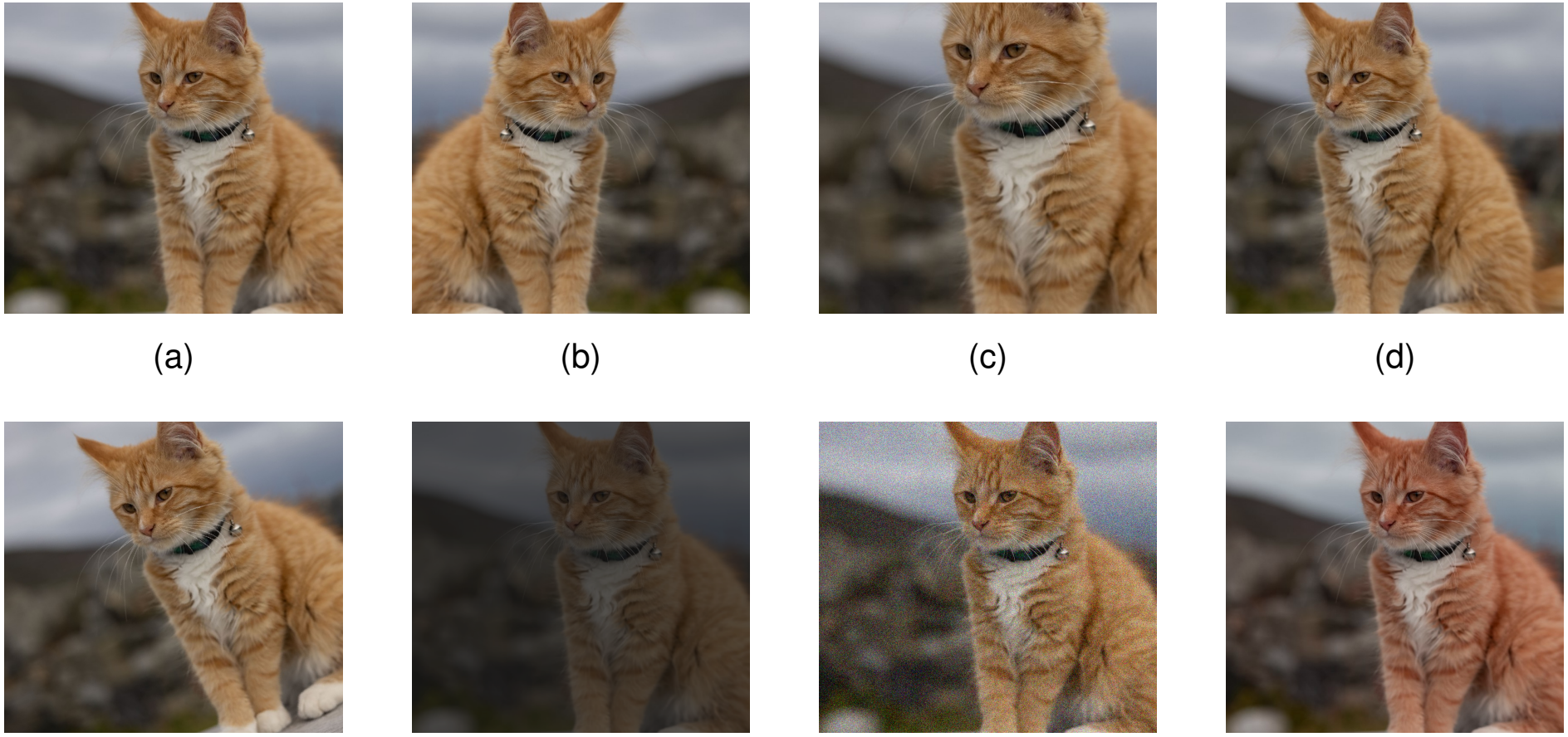


Figure from Bishop & Bishop (2024)

Injecting Noise

- Injecting noise into the input of a neural network can be seen as data augmentation
- **Neural networks are not robust to noise**
- To improve robustness, train them with random noise applied to their inputs
- Noise can also be applied to hidden units
- Dropout, a powerful regularization strategy, can be viewed as constructing new inputs by multiplying by noise (*more later*)

Commonly applied augmentation on images



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure from Bishop & Bishop (2024)

Keywords

- Bias-Variance Trade-off
- Regularization
- Model Capacity
- Invariance/equivariance
- **Dropout**

What is Bagging (Bootstrap Aggregating)

- It is a technique for reducing generalization error by combining several models
  - Idea is to train several models separately, then have all the models vote on the output for test examples
- This strategy is called model averaging
- Techniques employing this strategy are known as ensemble methods
- Model averaging works because different models will not make the same mistake

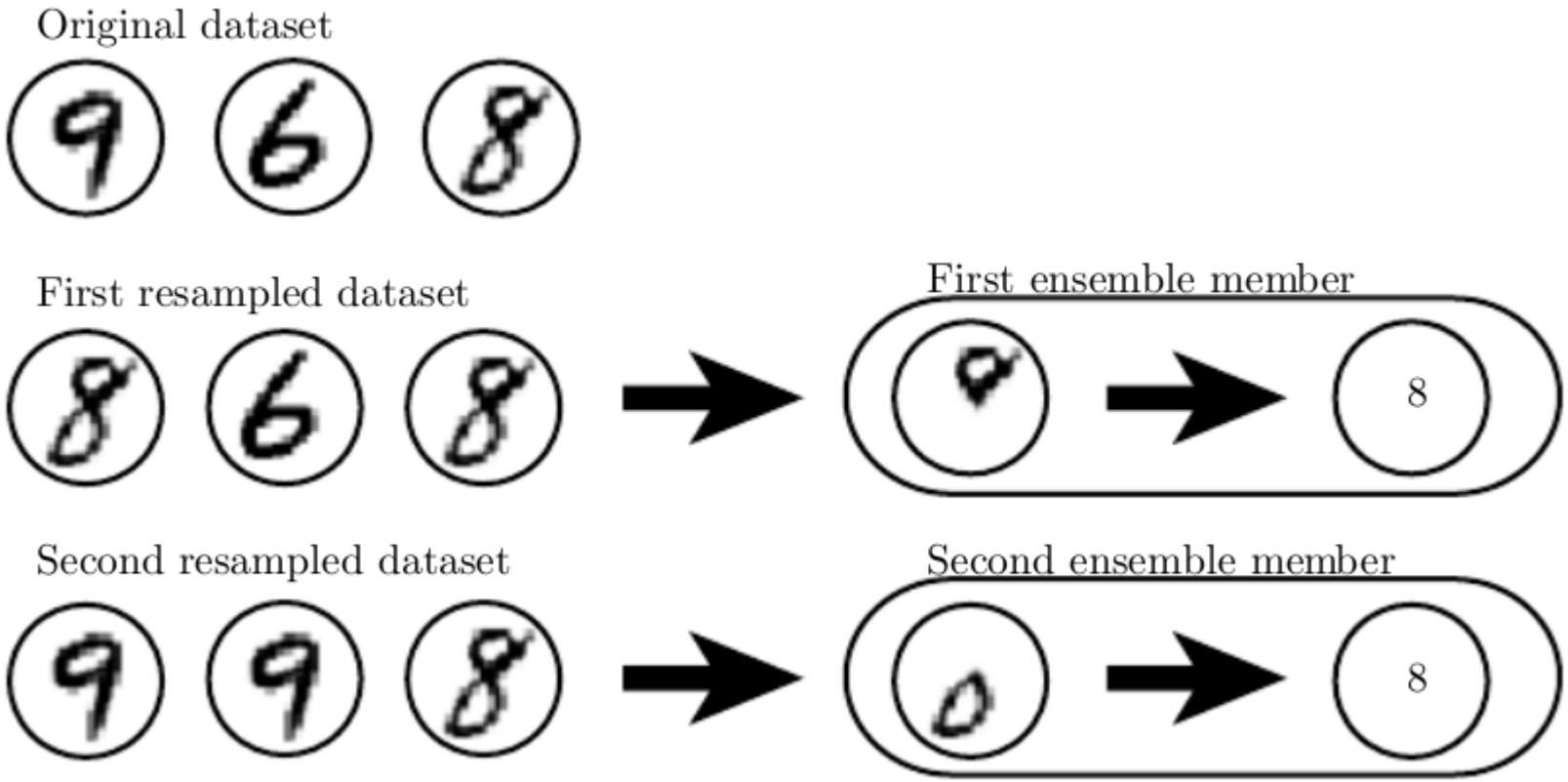
Ensemble vs Bagging

- Different ensemble methods construct the ensemble of models in different ways
- Example: each member of ensemble could be formed by training a completely different kind of model using a different algorithm or objective function
- Bagging is a method that allows the same kind of model, training algorithm and objective function to be reused several times

The Bagging Technique

- Given training set D of size N, generate k data sets of same no of examples as original by sampling with replacement
- Some observations may be repeated in D_i, some others are missing. This is known as a bootstrap sample.
- The differences in examples will result in differences between trained models
- The k models are combined by averaging the output (for regression) or voting (for classification)

Example


Original dataset
First resampled dataset
First ensemble member
8
Second resampled dataset
Second ensemble member
8

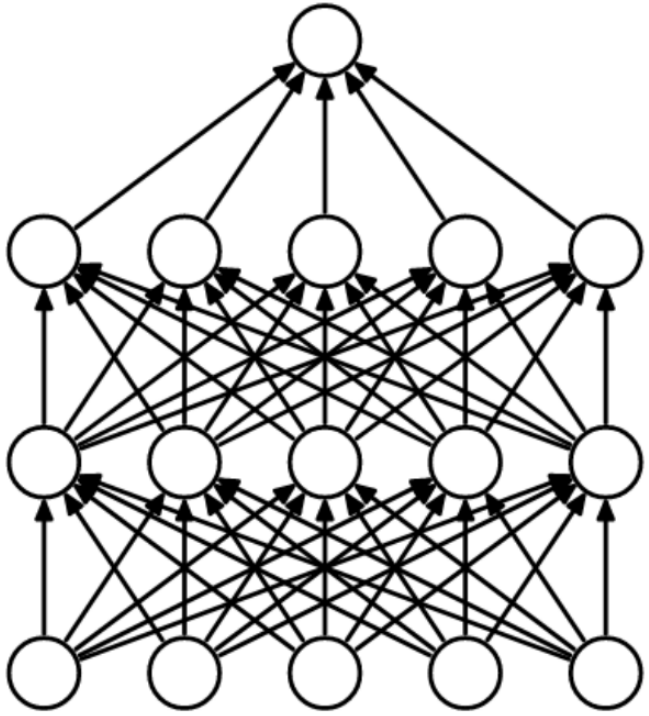
Bagging in Neural Nets

- Neural nets reach a wide variety of solution points
  - Thus, they benefit from model averaging when trained on the same dataset
  - Differences in:
    - random initializations
    - random selection of minibatches, in hyperparameters,
  - cause different members of the ensemble to make partially independent errors

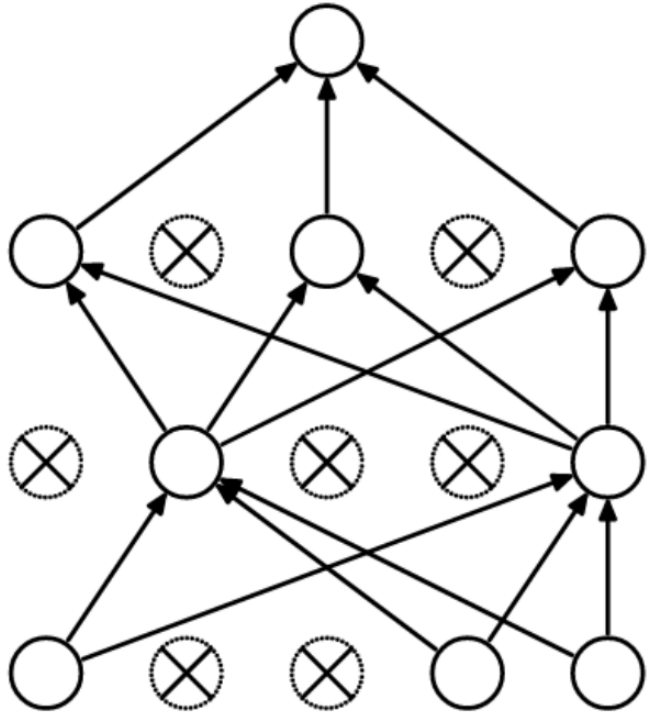
- Model averaging is a reliable method for reducing generalization error
  - Machine learning contests are usually won by model averaging over dozens of models, e.g., the Netflix grand prize
- But comes with significant computational costs

Creating new Models by Removing Units

- Dropout trains an ensemble of subnetworks
  - formed by removing non-output units from an underlying base network
- We can effectively remove units by multiplying its output value by zero



(a) Standard Neural Net



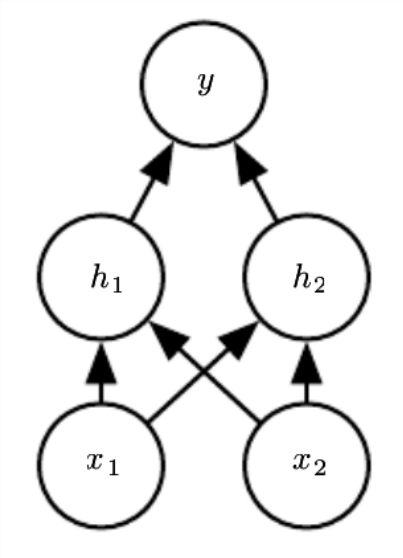
(b) After applying dropout.

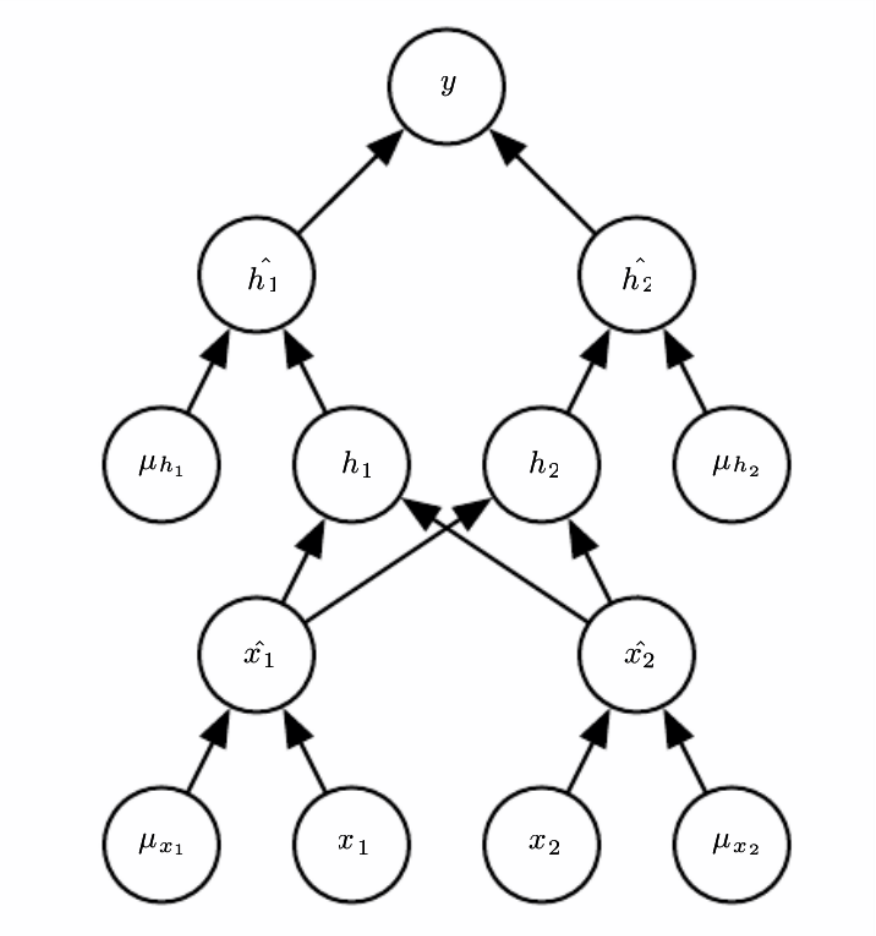
Mask for dropout training

- To train with dropout we use minibatch based learning algorithm that takes small steps such as SGD
- At each step randomly sample a binary mask
  - Probability of including a unit is a hyperparameter
  - Normally: 0.5 for hidden units and 0.8 for input units
- We run forward & backward propagation as usual
- Equivalent to randomly selecting a subnetwork of the entire network

Example

- Modified network incorporating a binary vector $\boldsymbol{\mu}$
- Training and evaluation as normal





Formal Description

➢Suppose that mask vector μ specifies which units to include

➢Cost of the model is specified by

$J(\theta, \mu)$

➢Dropout training consists of minimizing

$\mathbb{E}_{\mu}\left(J(\theta, \mu)\right)$

➢Expected value contains exponential no. of terms

➢We can get an unbiased estimate of its gradient by sampling values of μ

Dropout Prediction

- Submodel defined by mask vector $\boldsymbol{\mu}$ defines a probability distribution $p(y \mid \boldsymbol{x}, \boldsymbol{\mu})$
- At test time after training has finished, we would ideally like to find a sample average of all possible 2^n dropped-out networks
  - This is unfeasible for any realistic number n
- Approximation:
  - Each node's output scaled by a factor of p during training, so that the expected value of the output of any node is the same as in the training stages.
  - This is the biggest contribution of the dropout method:
    - **although it simulates 2^n different neural nets, at test time only a single network needs to be evaluated and tested**

Dropout

CLASS torch.nn.Dropout(*p=0.5*, *inplace=False*) [SOURCE]

During training, randomly zeroes some of the elements of the input tensor with probability `p`.

The zeroed elements are chosen independently for each forward call and are sampled from a Bernoulli distribution.

Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons as described in the paper Improving neural networks by preventing co-adaptation of feature detectors .

Furthermore, the outputs are scaled by a factor of $\frac{1}{1-p}$ during training. This means that during evaluation the module simply computes an identity function.

Parameters

- **p** (*float*) – probability of an element to be zeroed. Default: 0.5
- **inplace** (*bool*) – If set to `True`, will do this operation in-place. Default: `False`

Always check if p is p(keep) or p(drop).

Scale down during training to implement Dropout, such that no action needs to be done at test time

Make sure to properly set your model into .train() and .eval() mode!

Source: PyTorch documentation, Oct 10, 2024

Another Interpretation of Dropout

- So far we have described dropout purely as a means of performing efficient, approximate an ensemble method

- **Dropout trains an ensemble of models that share hidden units**
  - Each hidden unit must be able to perform well regardless of which other hidden units are in the model
  - Hidden units must be prepared to be swapped and interchanged between models

- **Dropout thus regularizes each hidden unit to be not merely a good feature but a feature that is good in many contexts.**

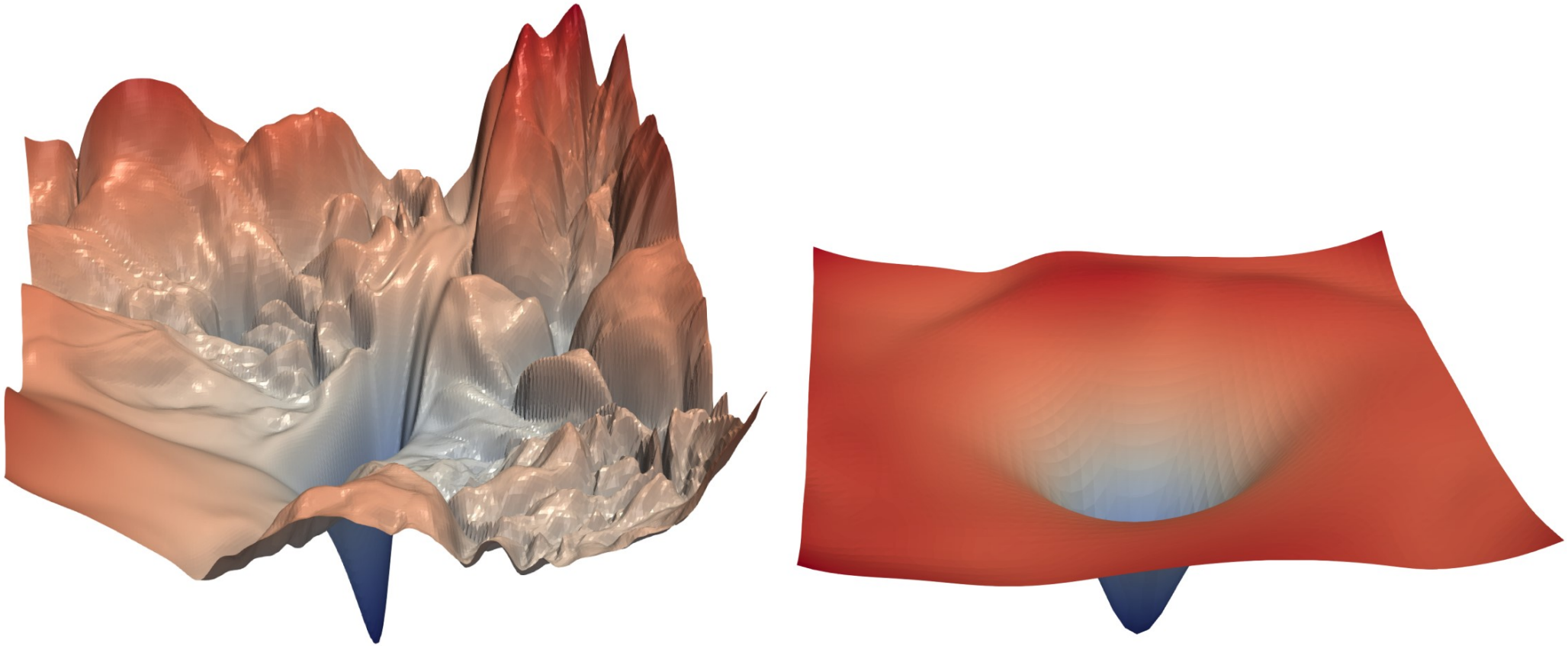
Take-home

- Best way to improve your model is to gather more data
- If that's not possible, **data augmentation** depending on domain knowledge (e.g., invariance)
- An L2 penalty (weight decay) may help to generalize better, but it shouldn't be too strong
- An L1 penalty on the weights nudges your model to be sparse (higher fraction of weights being 0)
- Early-stopping can help but be careful to not stop too early (use a very high patience to be safe)
- Ensembles/Bagging (averaging many models) usually works but is very expensive with large neural nets
- **Dropout** (randomly zeroing out activations) is an efficient way to implement bagging with neural nets

Keywords

- Bias-Variance Trade-off
- Regularization
- Model capacity
- Invariance/equivariance
- Dropout

Outlook: Residual Connections



(a) (b)

How’s my teaching?

https://www.admonymous.co/lukasgalke