LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Deeplearning

Chapter 2: Regularization

Bernd Bischl

Department of Statistics – LMU Munich

Winter term 2018

REVISION OF OVERFITTING

- A model finds a pattern in the data that is actually not true in the real world. That means the model **overfits** the data.
  - Humans also overfit when they overgeneralize from an incomplete picture of the world.
  - Every powerful model can “hallucinate” patterns.
- Happens when you have too many hypotheses and not enough data to tell them apart.
  - The more data, the more “bad” hypotheses are eliminated.
  - If the hypothesis space is not constrained, there may never be enough data.
  - There is often a parameter that allows you to constrain (**regularize**) the model.

AVOIDING OVERFITTING

- You should never believe your model until you've *verified it on data that it didn't see*.
- Scientific method applied to machine learning: model must make new predictions that can be experimentally verified.
- Randomly divide the data into:
  - *Training set* $\mathcal{D}_{\text{train}}$, which we will feed the model with.
  - *Test set* $\mathcal{D}_{\text{test}}$, which we will hide to verify its predictive performance.

OVERFITTING AND NOISE

- Overfitting is seriously exacerbated by *noise* (errors in the training data).
- An unconstrained learner will model that noise.
- A popular misconception is that overfitting is always caused by noise.
- It can also arise when relevant features are missing in the data.
- In general, it's better to make some mistakes on training data ("ignore some observations") than trying to get all correct.

TRIPLE TRADE-OFF

In all learning algorithms that are trained from example data, there is a trade-off between three factors:

- the complexity of the hypothesis we fit to data
- the amount of training data
- the generalization error on new examples

The generalization error decreases with the amount of training data. As the complexity of the hypothesis space H increases, the generalization error decreases first and then starts to increase (overfitting).

- Training error and test error evolve in the opposite direction with increasing complexity:

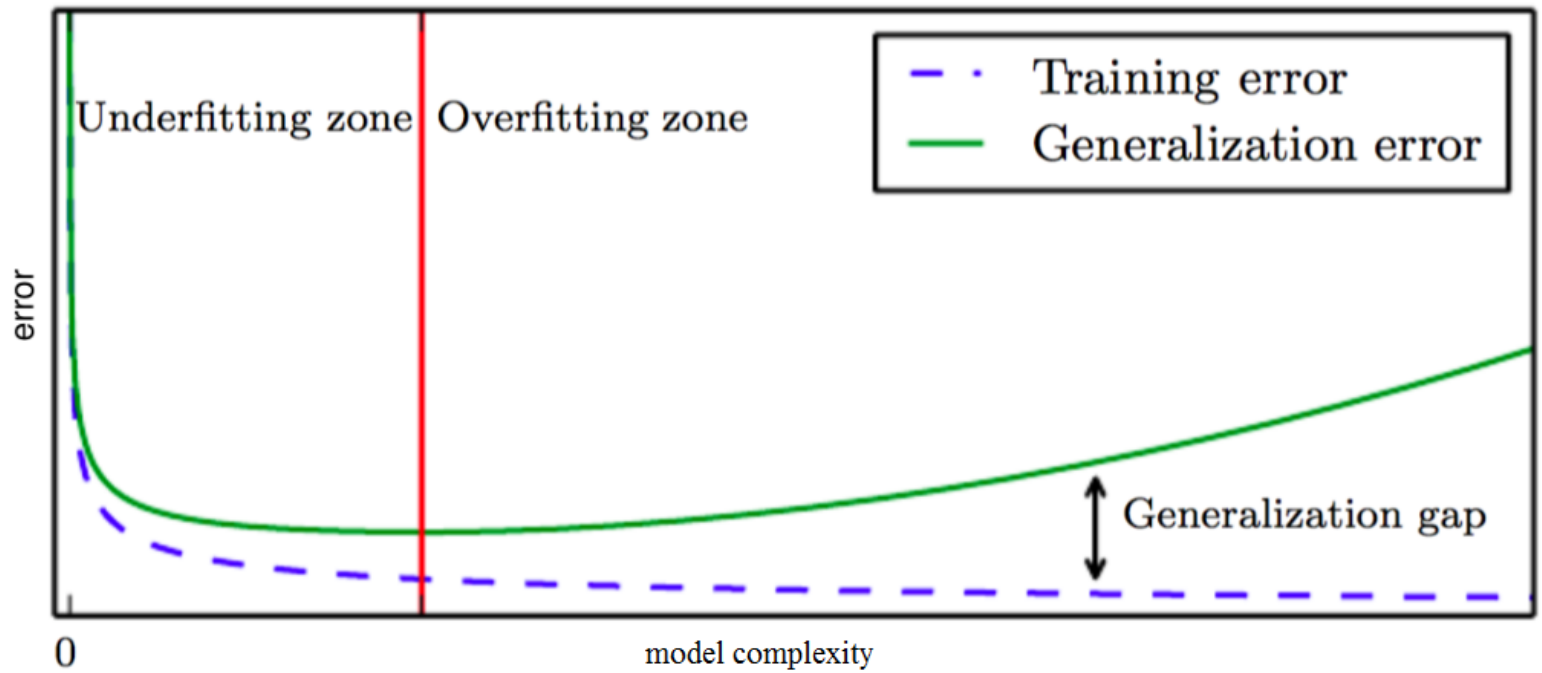


Figure: Underfitting vs. overfitting (Goodfellow et al. (2016))

$\Rightarrow$ Optimization regarding the model complexity is desirable!

GENERALIZATION ERROR

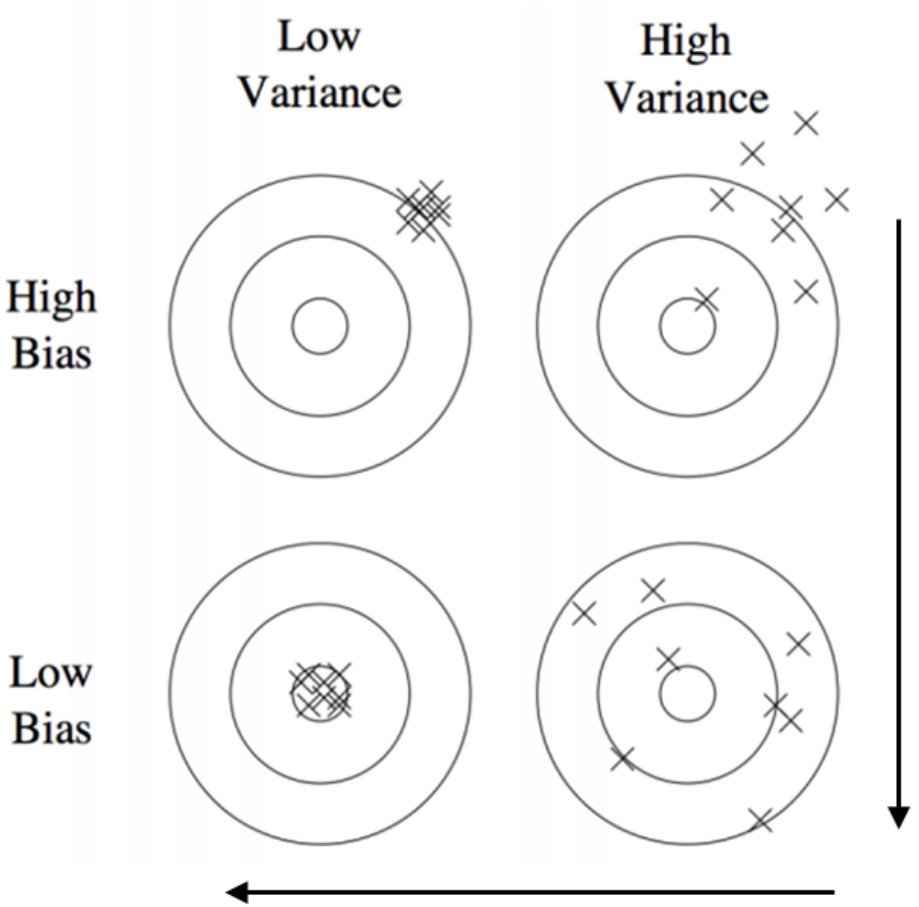
The *generalization error* is defined as the error that occurs when a model $\hat{f}_{\mathcal{D}}$ that was trained on observed data $\mathcal{D}$ is applied to (unseen) data:

$$GE(\hat{f}_{\mathcal{D}}) = \mathbb{E}(L(y, \hat{f}_{\mathcal{D}}(x))|\mathcal{D}),$$

where

- $\hat{f}_{\mathcal{D}}$ is the prediction model that was estimated using the data $\mathcal{D}$,
- the expectation is conditional on $\mathcal{D}$ that was used to build the prediction model and
- L is an *outer* loss function that tries to measure the model performance (which can be different from the *inner* loss function that was used for the empirical risk minimization).

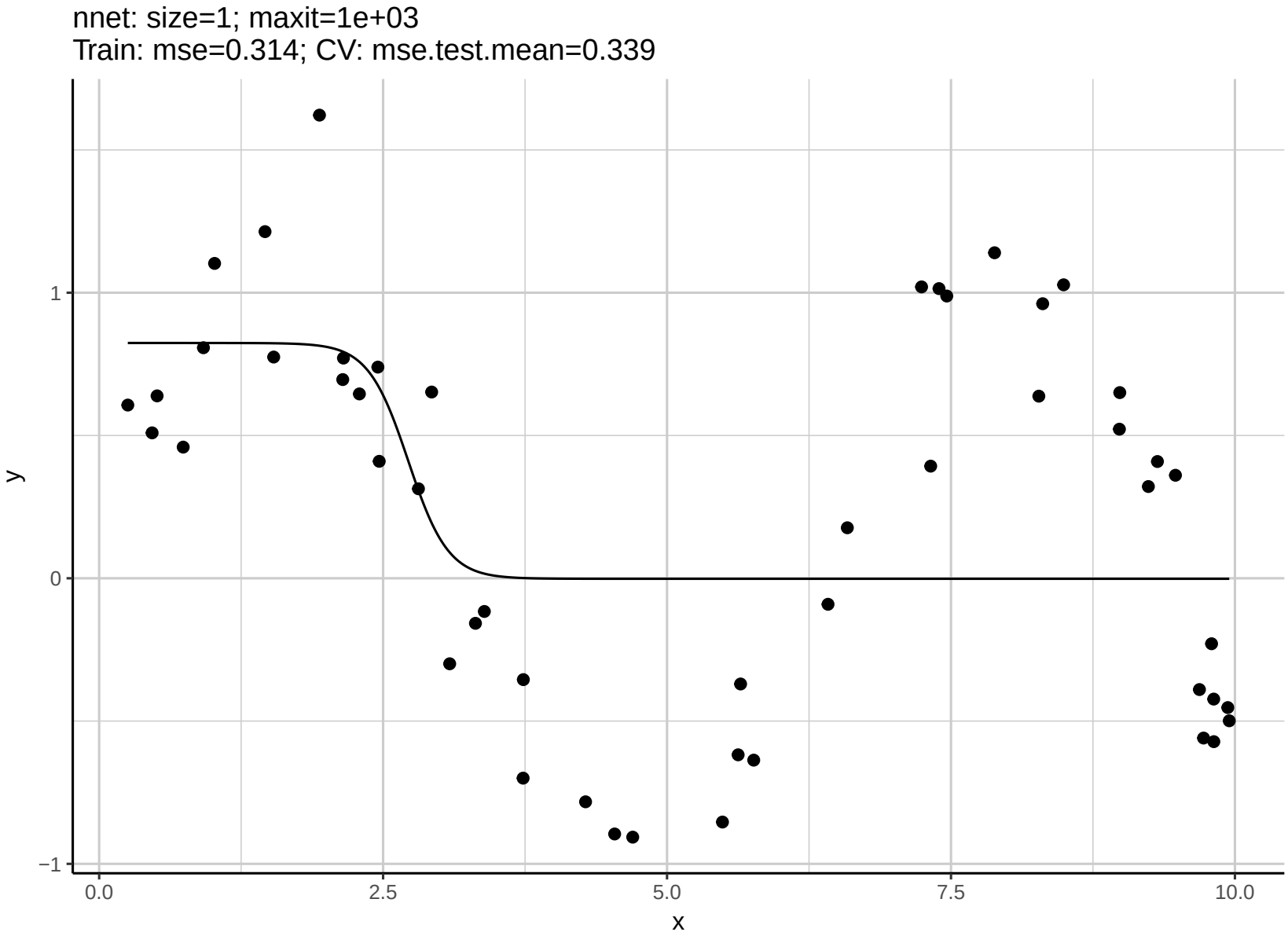
BIAS-VARIANCE DECOMPOSITION



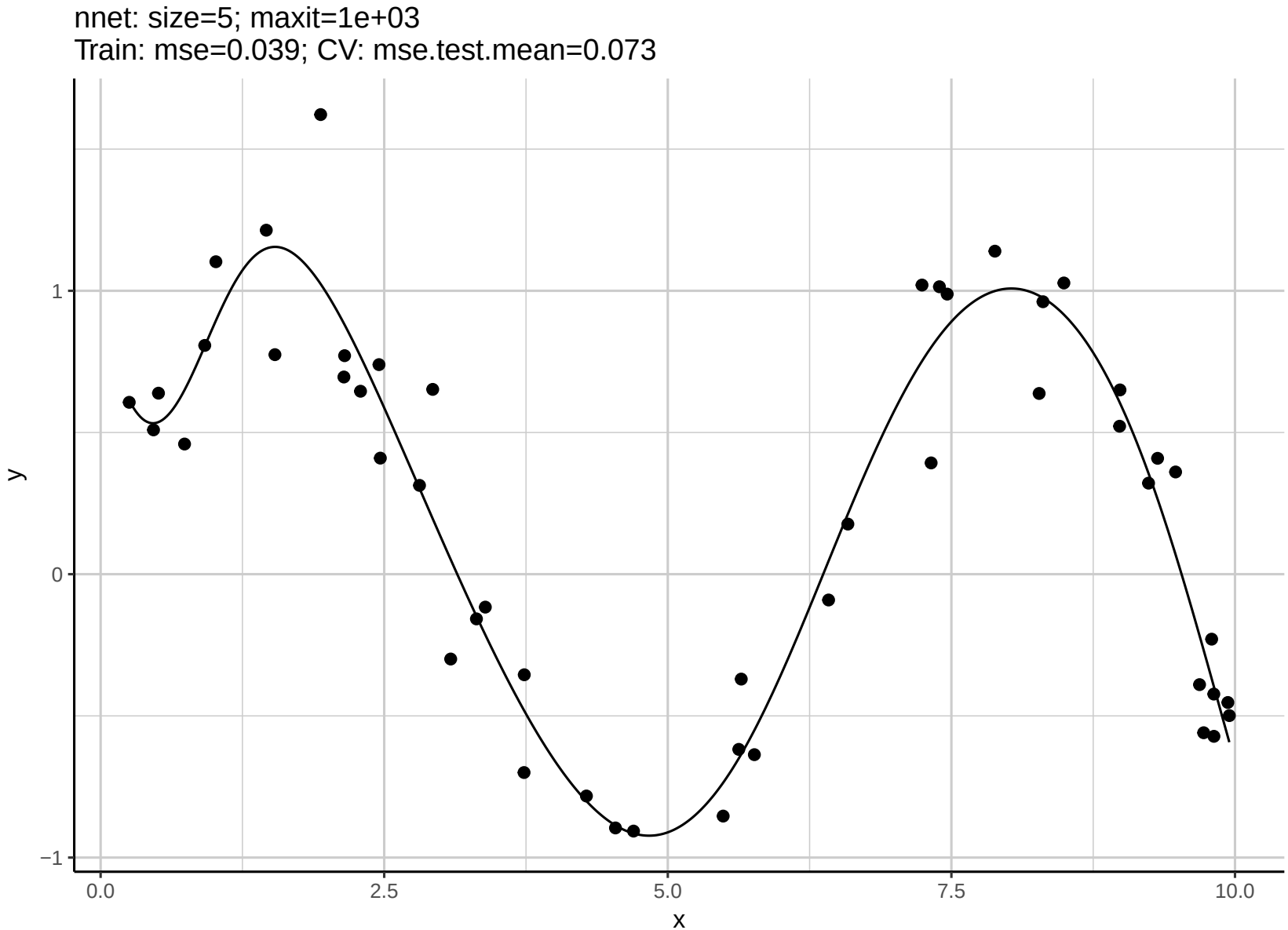
Reduce bias
$\Rightarrow$ reduce underfitting
$\Rightarrow$ make model more flexible.

Reduce variance $\Rightarrow$ Reduce overfitting
$\Rightarrow$ make model less flexible
$\Rightarrow$ **regularization**, or add more data.

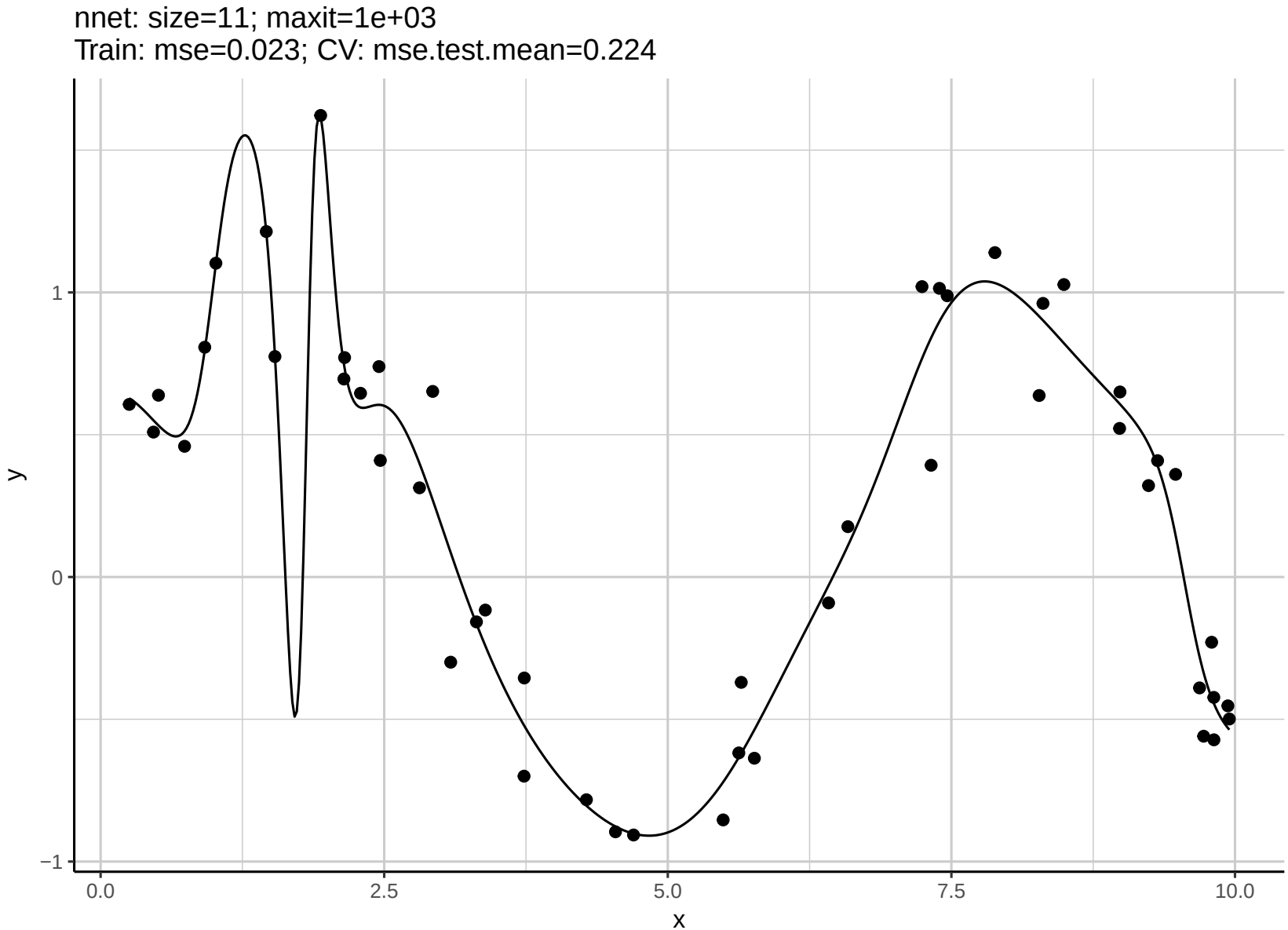
OVERFITTING IN REGRESSION



OVERFITTING IN REGRESSION

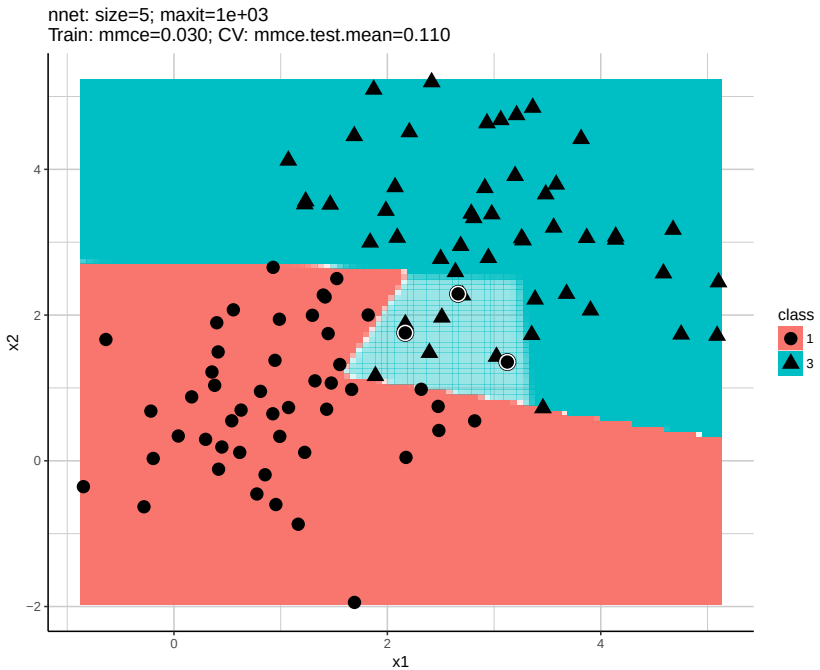


OVERFITTING IN REGRESSION



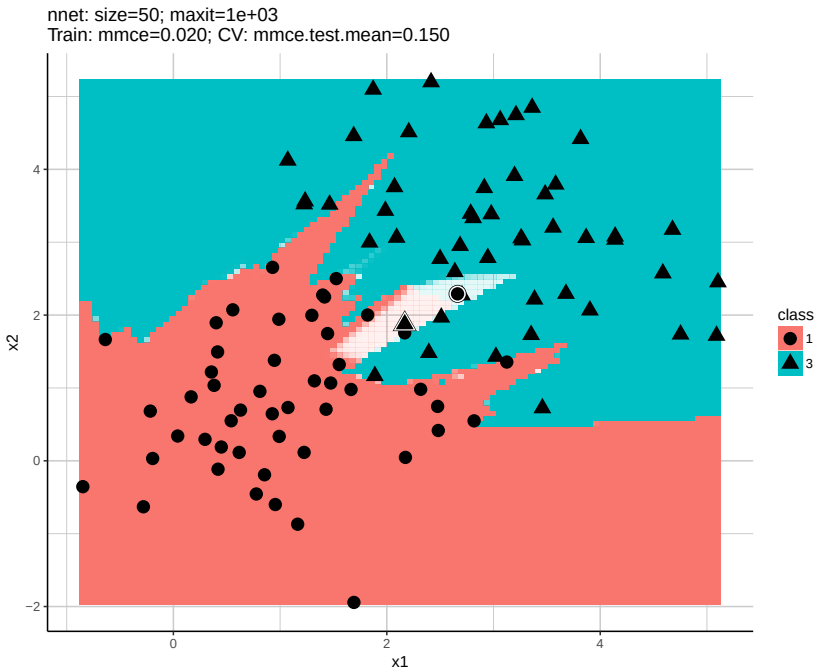
OVERFITTING IN CLASSIFICATION

- Non-overfitting model



Better test accuracy

- Overfitting model



Better training accuracy

PARAMETER NORM PENALTIES

- Norm penalization aims to limit the complexity of the model.

- Suppose we would like to regularize an objective function $R_{emp}(\theta|X, y)$, which generally is a loss function L(y, f(x)).
- By adding a parameter norm penalty term $\Omega(\theta)$, we obtain a regularized version of the objective function:

$$R_{reg}(\theta|X, y) = R_{emp}(\theta|X, y) + \lambda\Omega(\theta)$$

  with hyperparamater $\lambda \in [0, \infty)$, that weights the penalty term, relative to the unconstrained objective function $R_{emp}(\theta|X, y)$.

- Declaring $\lambda = 0$ obviously results in no penalization.
- We can choose between different parameter norm penalties $\Omega(\theta)$.

PARAMETER NORM PENALTIES

- Keep in mind, when penalizing a linear model, the parameters θ are the coefficients β.
- In neural networks, the parameters θ are the weights (e.g. w).
- In general, we do not penalize the bias.
  - Less data required for fitting the bias than for w.
  - A possible consequence of regularizing the bias is underfitting!

L2 REGULARIZATION (WEIGHT DECAY)

- Analogue to **ridge regression**: $\Omega(w) = \frac{1}{2}||w||_2^2$ such that

$$R_{reg}(w|X, y) = R_{emp}(w|X, y) + \frac{\lambda}{2} w^T w$$

with corresponding gradient:

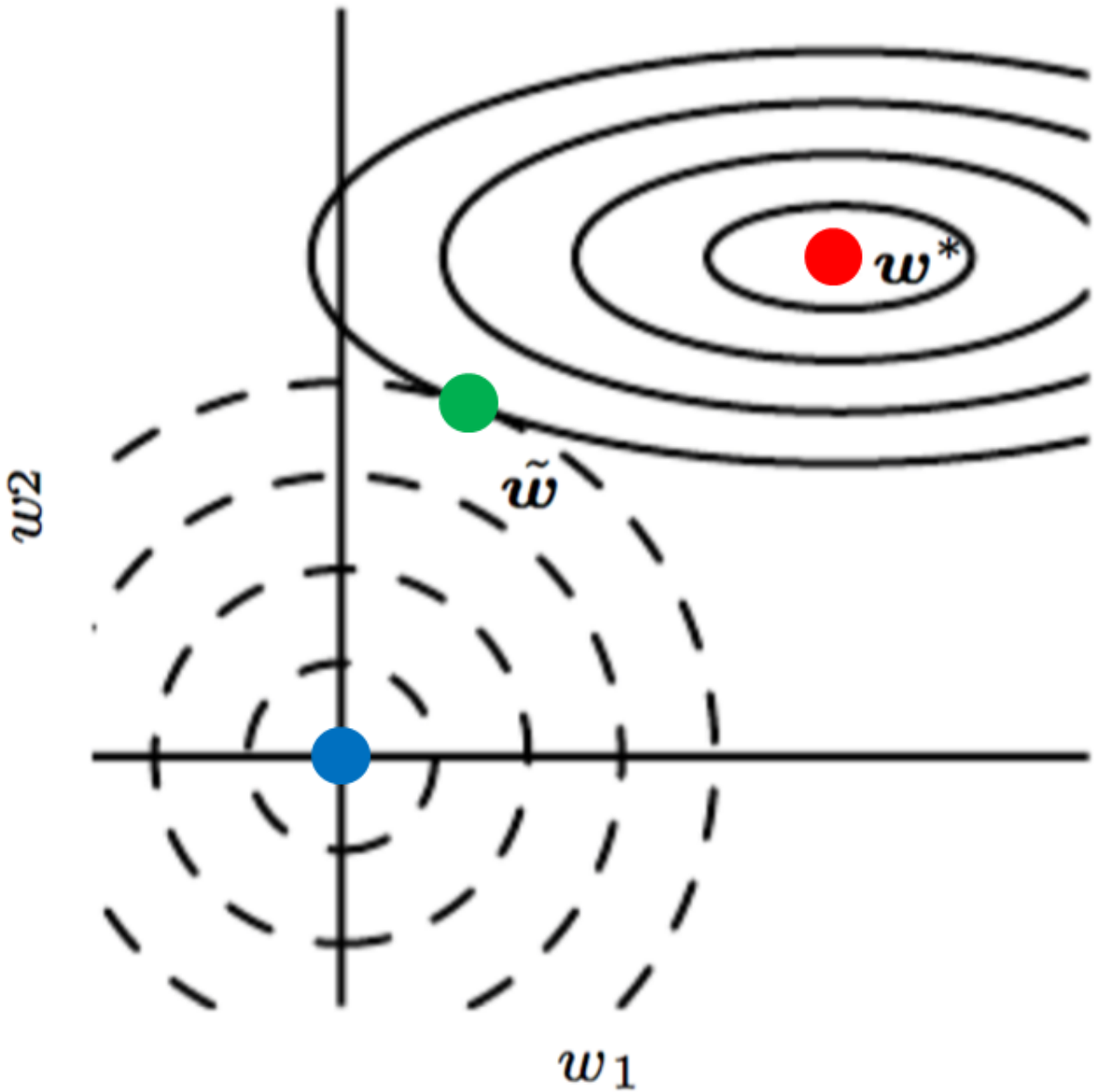
$$\nabla_w R_{reg}(w|X, y) = \nabla_w R_{emp}(w|X, y) + \lambda w$$

- One weight update using gradient descent is

$$\begin{aligned} w_{i+1} &= w_i - \alpha(\lambda w_i + \nabla_w R_{emp}(w_i|X, y)) \\ &= \underbrace{(1 - \alpha\lambda)}_{<1} w - \alpha \nabla_{w_i} R_{emp}(w_i|X, y)) \end{aligned}$$

- Therefore termed **weight decay** in neural net applications

L2 REGULARIZATION (WEIGHT DECAY)



minimize empirical risk

minimize regularized risk

minimize penalty

WEIGHT DECAY EXAMPLE

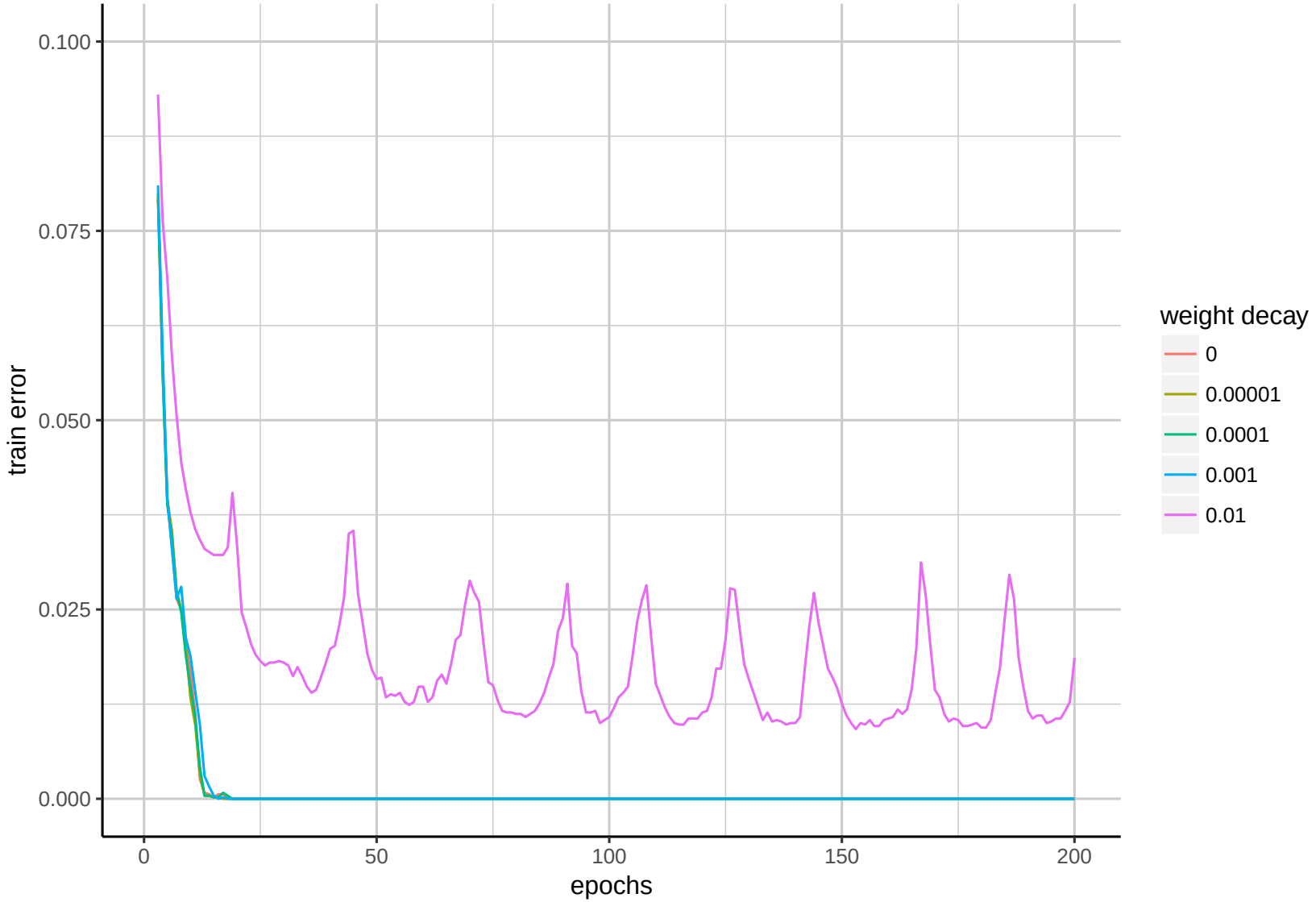
- Let's fit the following huge neural network on a smaller fraction of the mnist data:

```
layer1 = mx.symbol.FullyConnected(data, num_hidden = 512L)
activation1 = mx.symbol.Activation(layer1, act_type = "relu")
layer2 = mx.symbol.FullyConnected(activation1, num_hidden = 512L)
activation2 = mx.symbol.Activation(layer2, act_type = "relu")
layer3 = mx.symbol.FullyConnected(activation2, num_hidden = 512L)
activation3 = mx.symbol.Activation(layer3, act_type = "relu")
layer4 = mx.symbol.FullyConnected(activation3, num_hidden = 10L)
softmax = mx.symbol.SoftmaxOutput(layer4, name = "softmax")
```

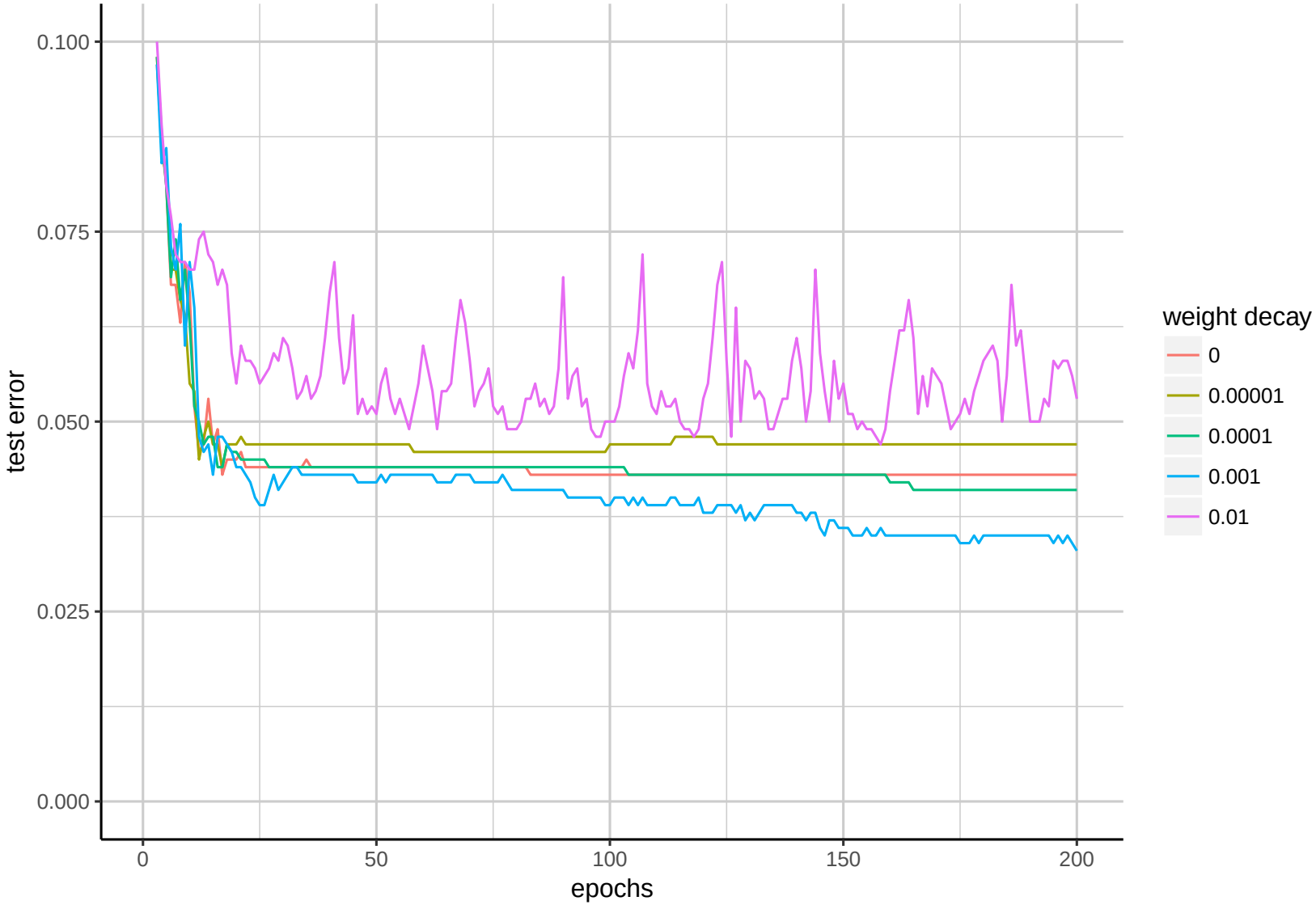
- That is to say 5000 training and 1000 testing samples with evenly distributed class labels.
- We try out different values of weight decay:

$$\alpha \in (0.01, 0.001, 0.0001, 0.00001, 0)$$

WEIGHT DECAY EXAMPLE



WEIGHT DECAY EXAMPLE



WEIGHT DECAY EXAMPLE

- The misclassification accuracy of a neural network can still improve even if we reach 100% training accuracy.
- Consider a binary classification problem with labels 0 and 1.
  - Assume a training sample with label 0 is assigned output probabilities $[0.75, 0.25]$ by the neural net.
  - Thus, the training error with respect to this point it 0, because the predicted label is 0.
  - Applying the negative log-likelihood (cross entropy) as loss function may still lead to improvements in the neural networks "confidence", even after training error reaches 0% (i.e.$[0.9, 0.1]$).
- Note that this is a theoretical concept and in practice we will almost always overfit by the time we reach 100% training accuracy.

EARLY STOPPING

- Goal: find optimal number of epochs.
- Stop algorithm early, before generalization error increases.

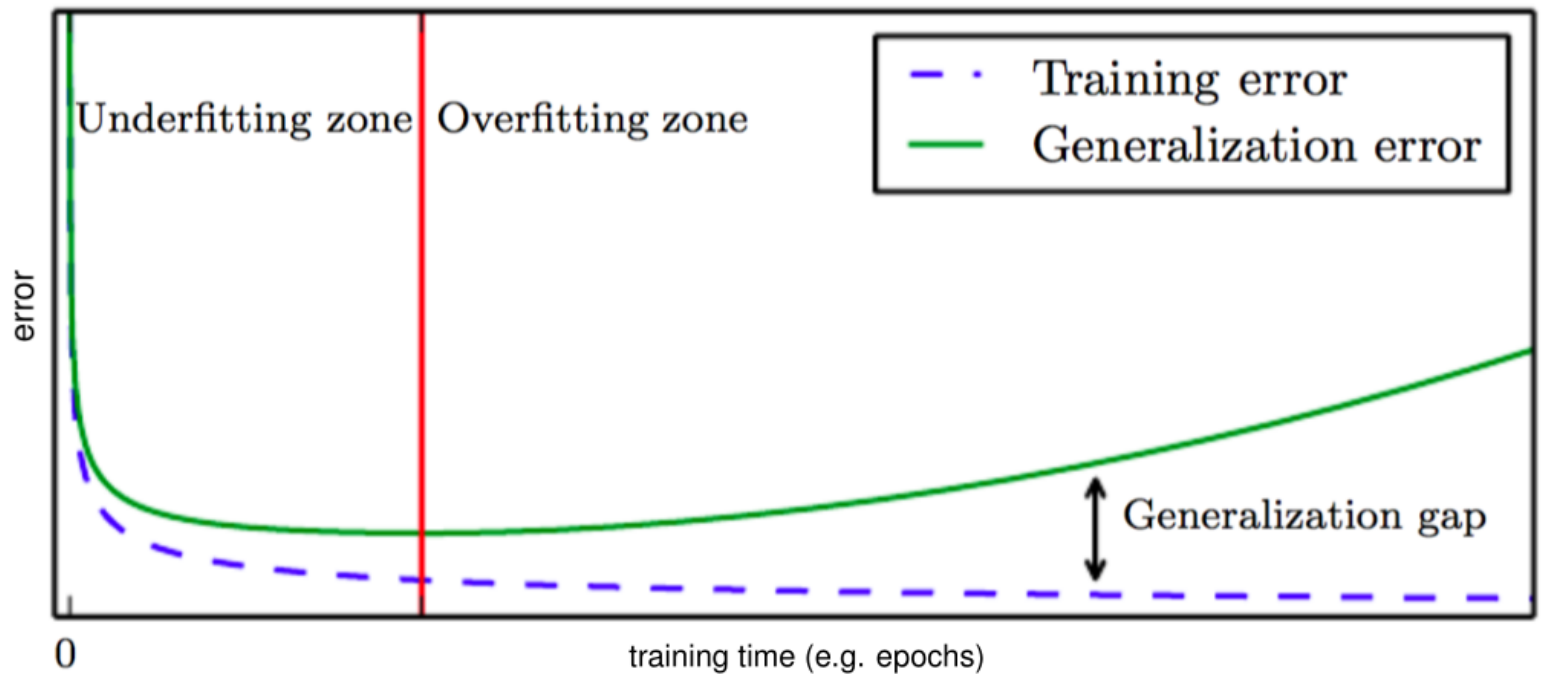


Figure: Underfitting vs. overfitting (Goodfellow et al. (2016))

EARLY STOPPING

How early stopping works:

1. Split training data $(X^{(train)}, y^{(train)})$ into $(X^{(subtrain)}, y^{(subtrain)})$ and $(X^{(validation)}, y^{(validation)})$ (e.g. with a ratio of 2:1).
2. Use $(X^{(subtrain)}, y^{(subtrain)})$ and evaluate model using the $(X^{(validation)}, y^{(validation)})$.
3. Stop training when validation error stops decreasing (after a range of “patience” steps).
4. Use parameters of the previous step for the actual model.

More sophisticated forms also apply cross-validation.

EARLY STOPPING

Strengths	Weaknesses
Effective and simple	Periodical evaluation of validation error
Applicable to almost any model without adjustment	Temporary copy of θ^* (we have to save the whole model at each iteration).
Combinable with other regularization methods	Less data for training $\rightarrow$ include $(X^{(validation)}, y^{(validation)})$ afterwards

- Relation between optimal early stopping iteration m and weight decay penalization parameter λ (see Goodfellow et al. (2016) page 251-252 for proof):

$$\begin{aligned} m &\approx \frac{1}{\alpha\lambda} \\ &\Leftrightarrow \quad \lambda \approx \frac{1}{m\alpha} \end{aligned}$$

- Small λ (low penalization) $\Rightarrow$ high m (deep model/lots of updates)

EARLY STOPPING

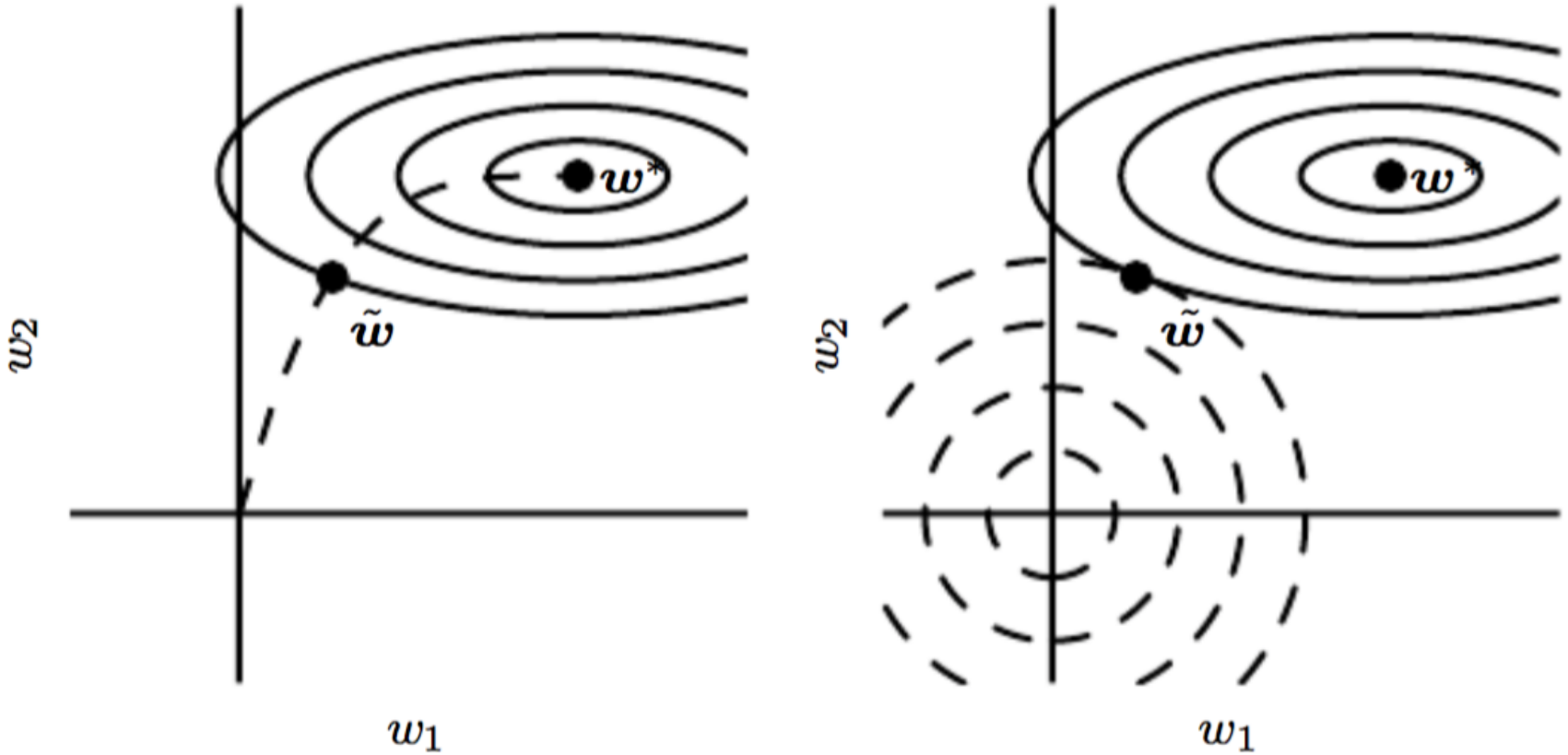
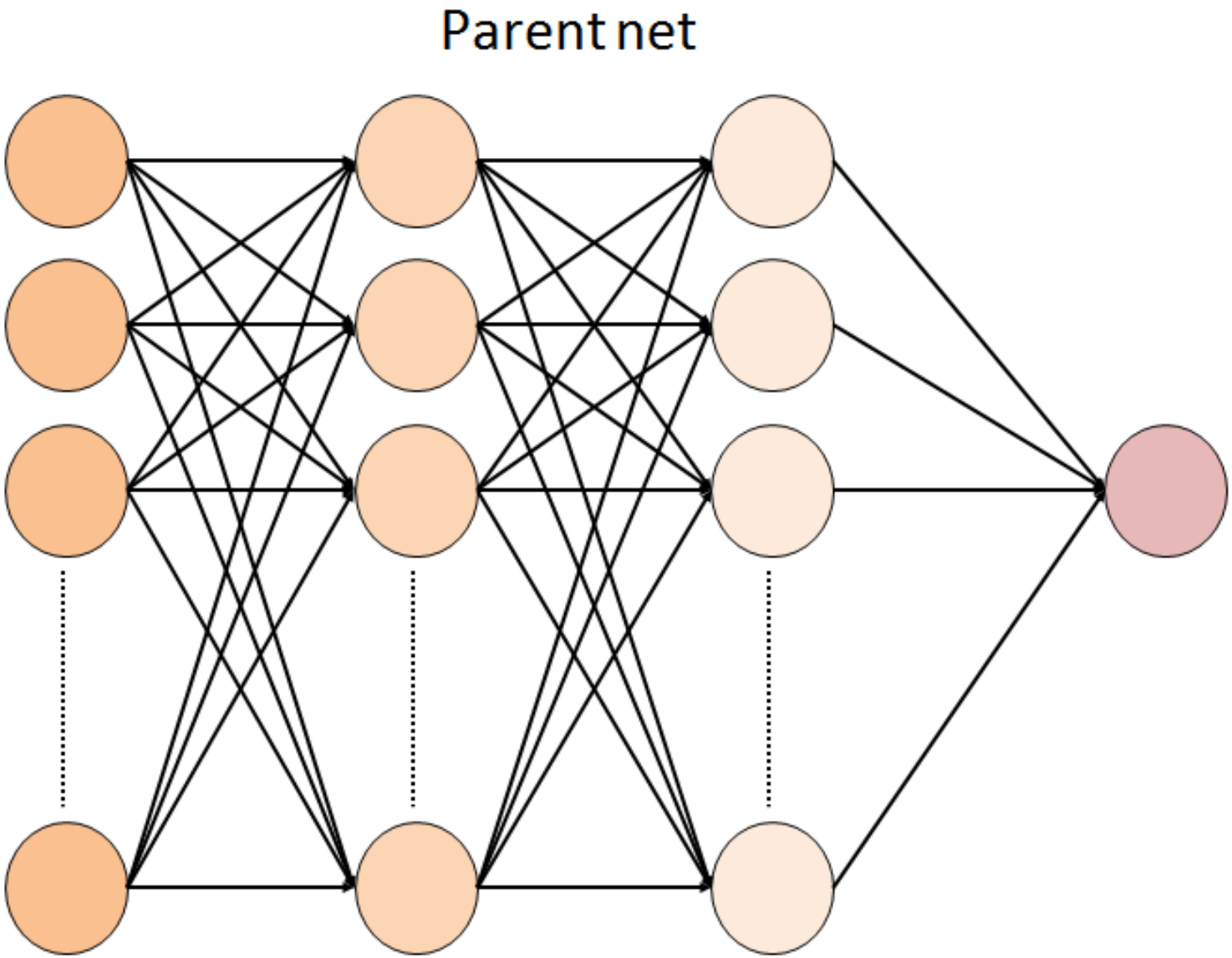


Figure: Optimization path of early stopping (left) and weight decay (right) (Goodfellow et al. (2016))

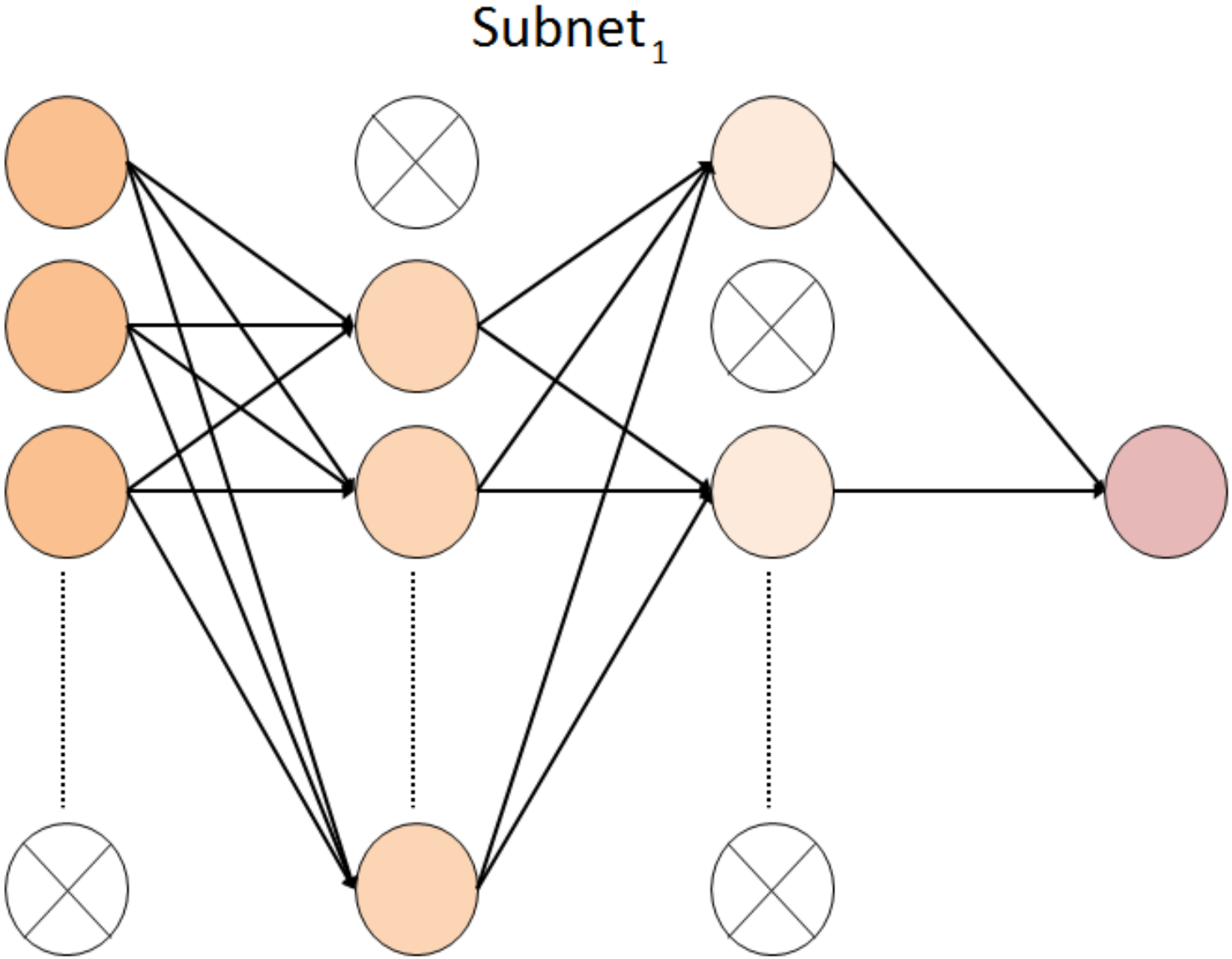
DROPOUT

- Idea: constrain the networks adaptation to the training data to avoid becoming “too smart” in learning the input data.
  - Dropout can be thought of as making bagging practical for ensembles of many large neural networks!
  - In ensemble learning we take a number of weaker classifiers, train them separately and finally average them.
  - Since each classifier has been trained independently, it has learned different “aspects” of the data.
  - Combining them helps to produce an stronger classifier, which is less prone to overfitting (e.g. random forests)
- So how does dropout actually work?

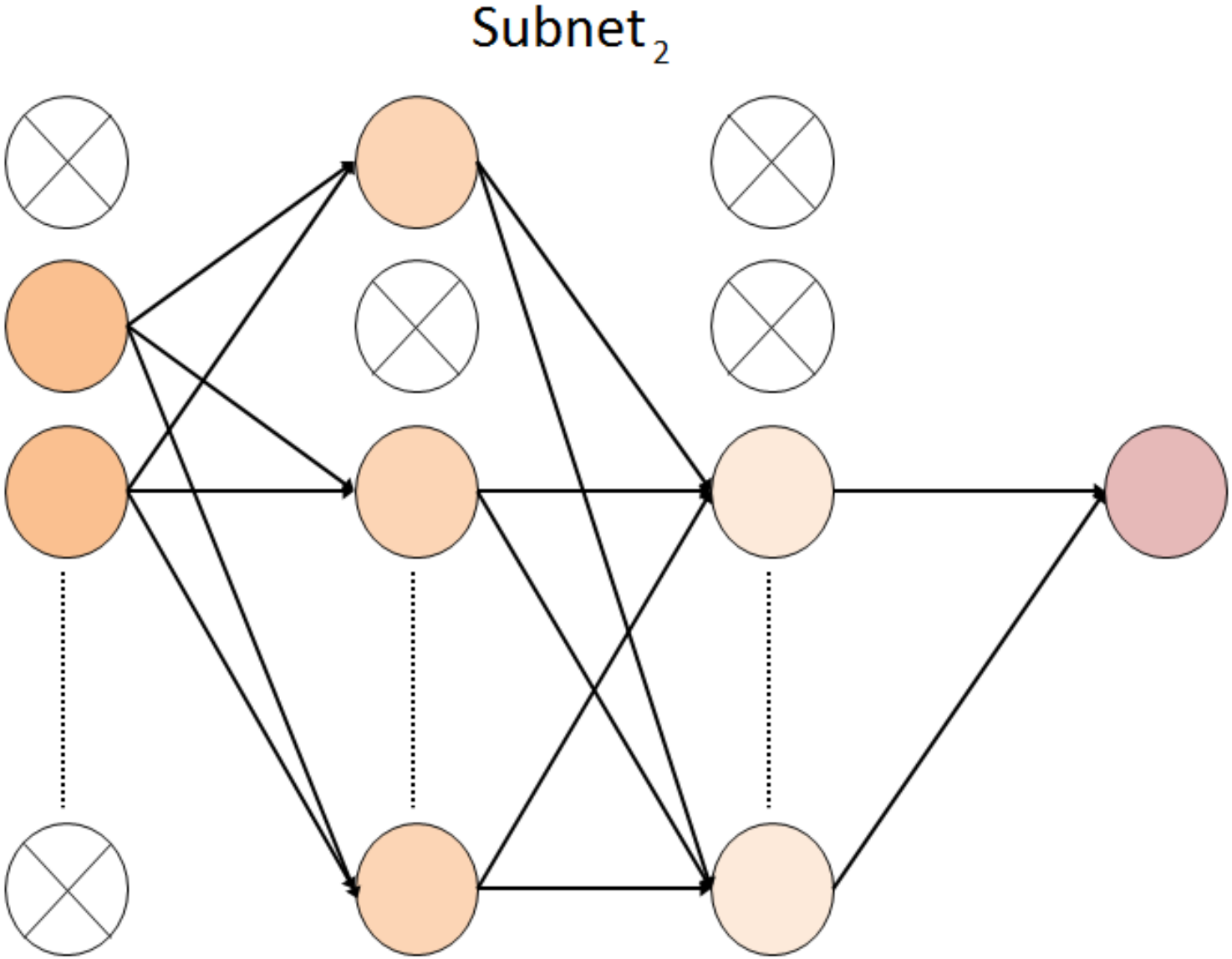
DROPOUT



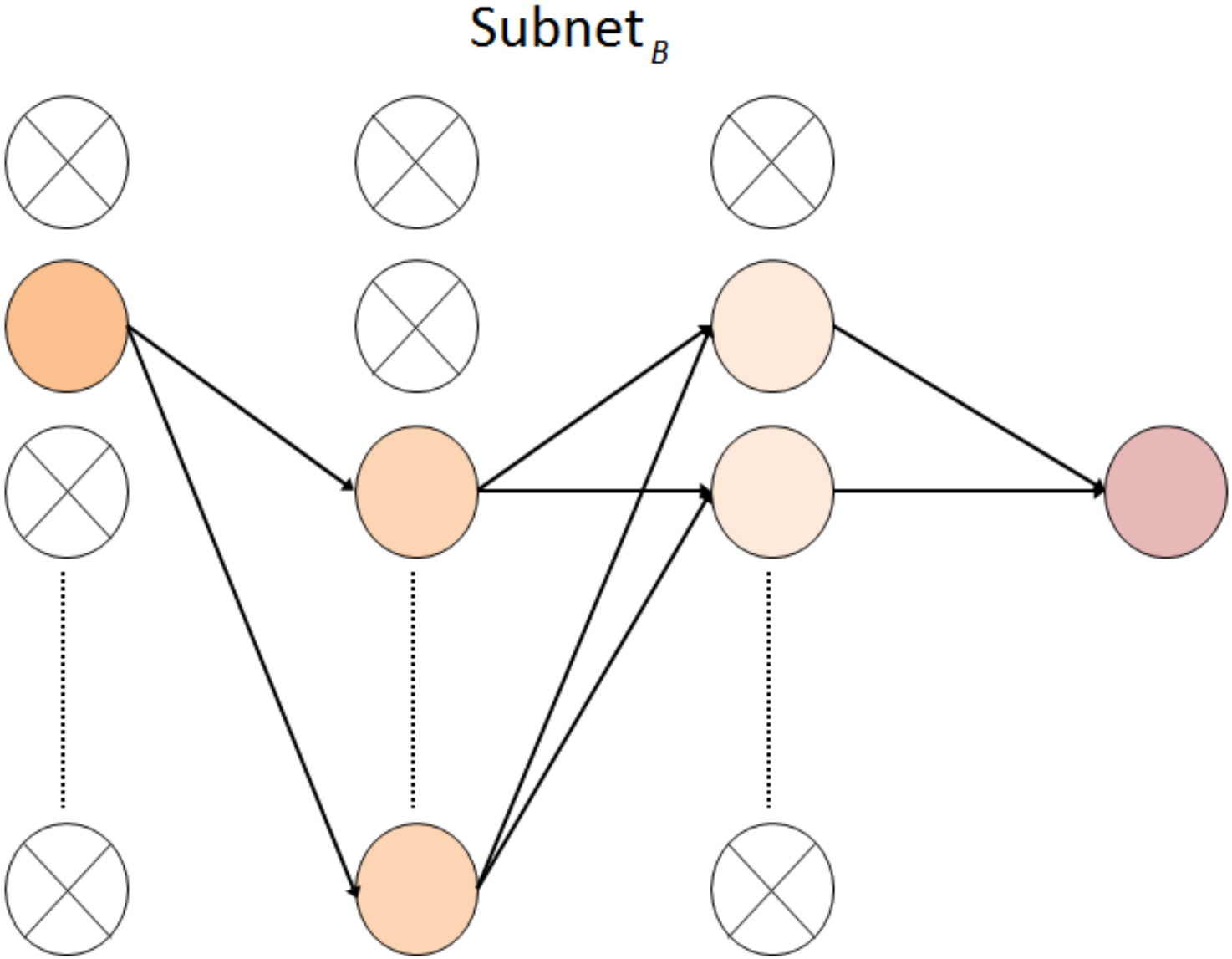
DROPOUT



DROPOUT



DROPOUT

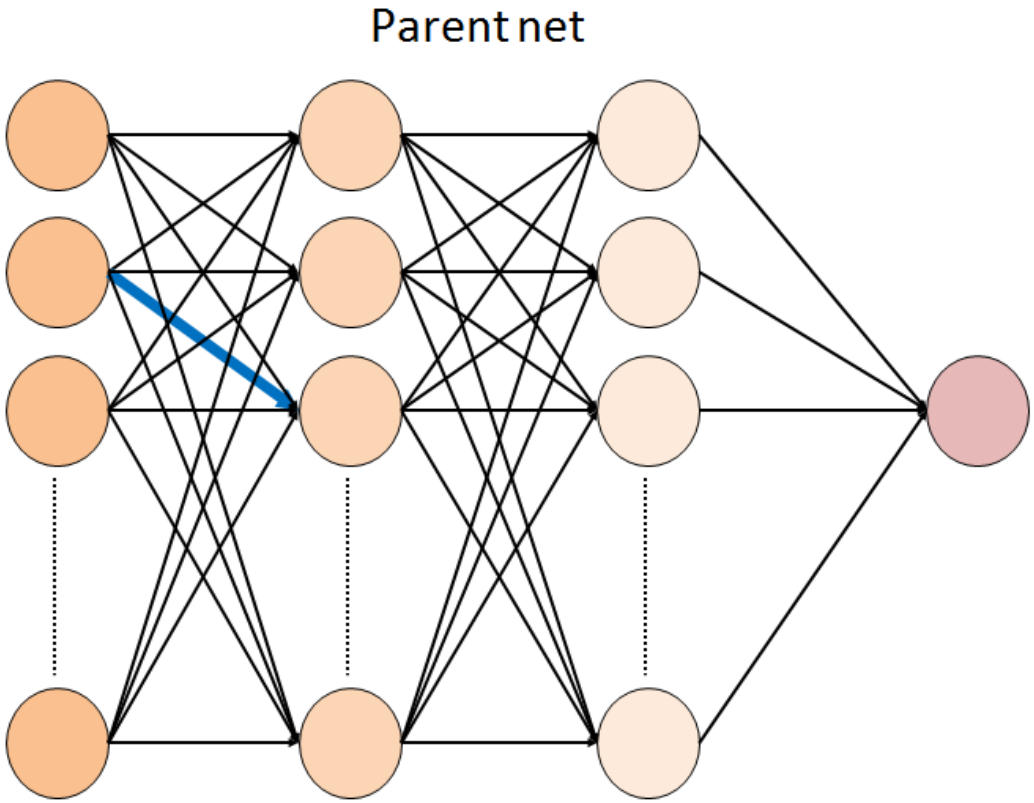


DROPOUT

- Bagging: all models are independent.
- Dropout: models not independent as they share parameters!
  - Each subnets archtitecture is defined by a "mask" μ. The mask μ randomly determines the in-or exclusion of units (neurons) and is trained on one randomly sampled training data point (x, y) (or mini batch).
  - The mask μ is a vector of length d (total number of units (neurons) in the network) with $\mu = (\mu_1, \mu_2, \ldots, \mu_d),\ \mu_i = \{0, 1\}$ and $P(\mu_i = 1) = p$.
  - Thus, each subnet inherits a different subset of parameters from the parent neural network.
  - Parameter sharing makes it possible to represent huge number of of models with particular amount of memory (hardware limitation!).

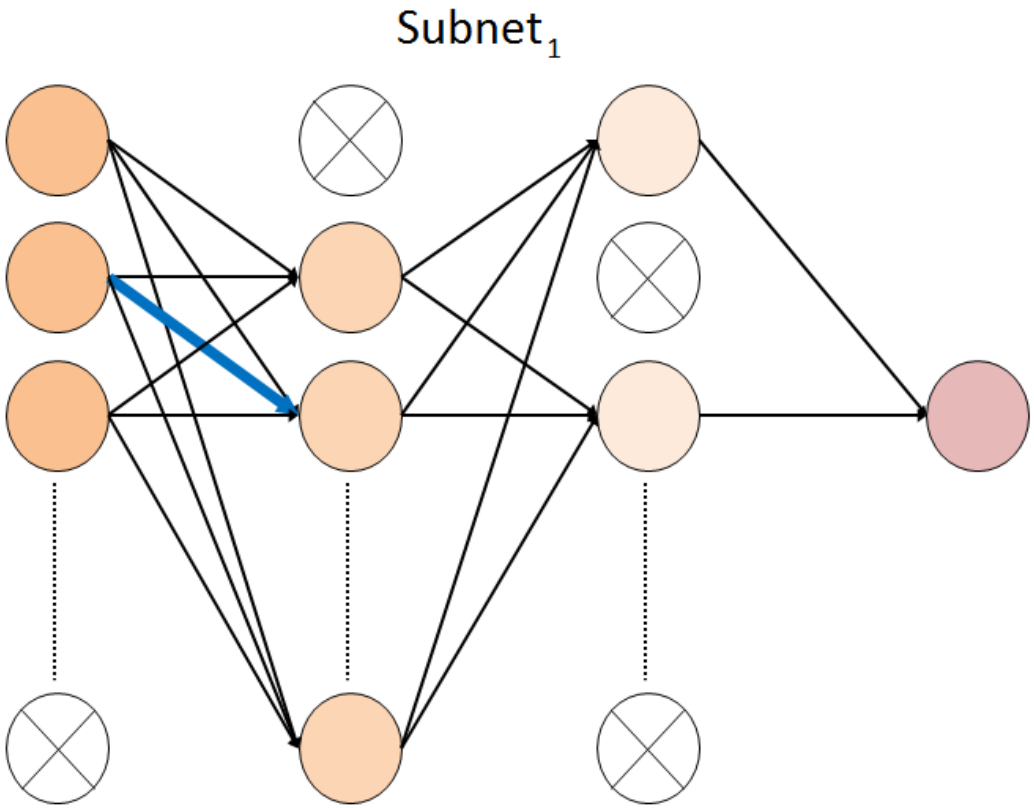
PARAMETER SHARING

- Parameter sharing:
  - In the case of bagging, the models are all independent.
  - In dropout on the other hand, all models share parameters. That means each model inherits a different subset of parameters from the parent neural network.



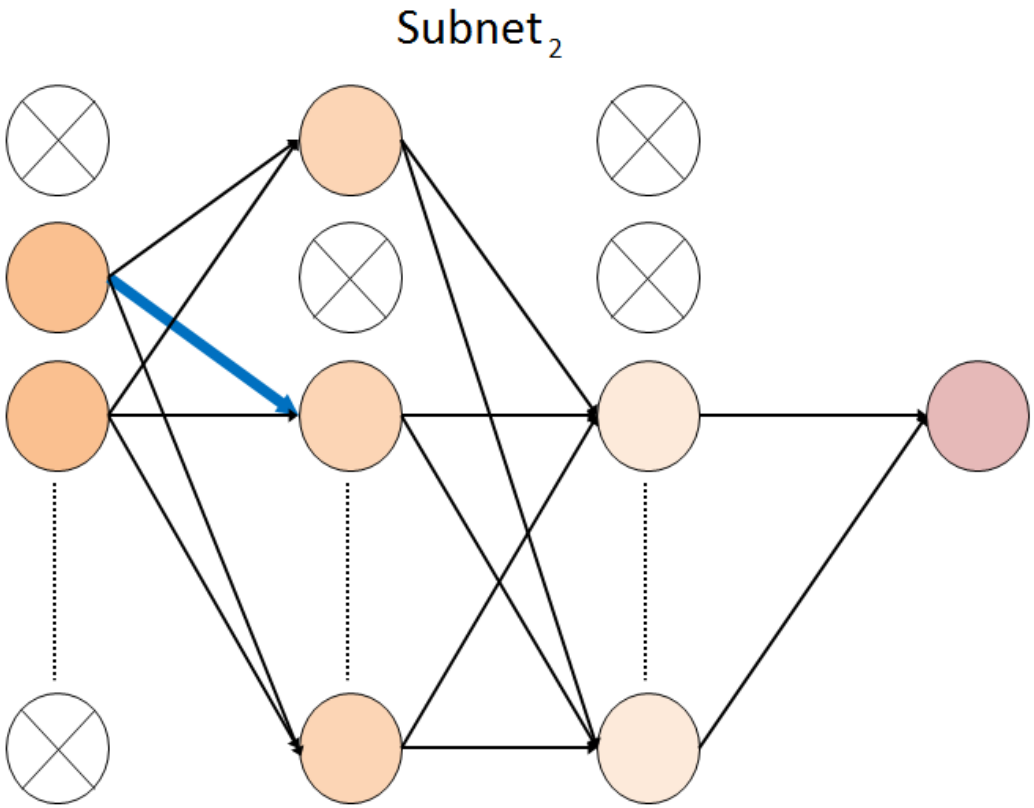
PARAMETER SHARING

- Parameter sharing:
  - In the case of bagging, the models are all independent.
  - In dropout on the other hand, all models share parameters. That means each model inherits a different subset of parameters from the parent neural network.



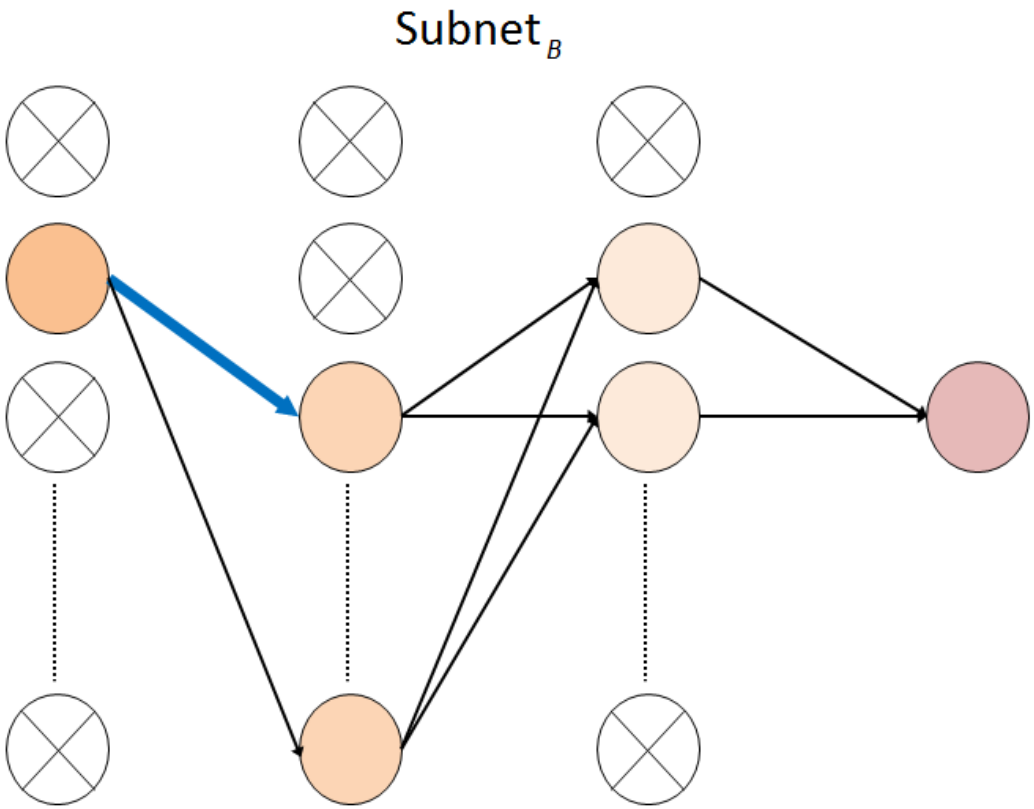
PARAMETER SHARING

- Parameter sharing:
  - In the case of bagging, the models are all independent.
  - In dropout on the other hand, all models share parameters. That means each model inherits a different subset of parameters from the parent neural network.



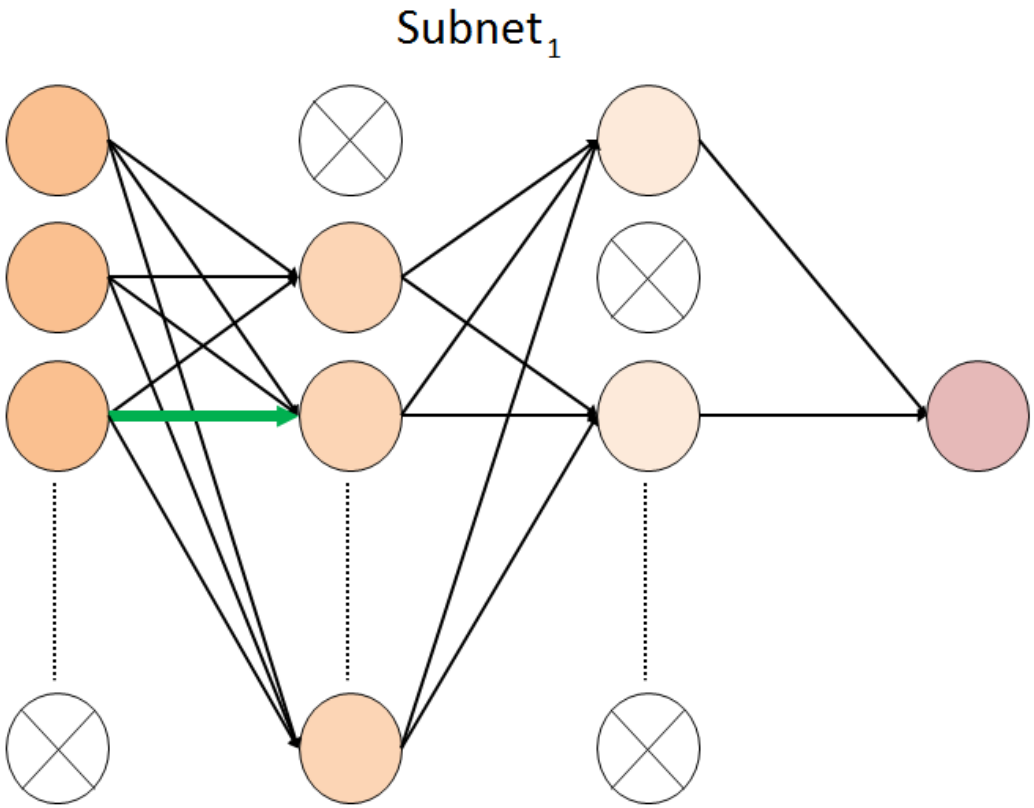
PARAMETER SHARING

- Parameter sharing:
  - In the case of bagging, the models are all independent.
  - In dropout on the other hand, all models share parameters. That means each model inherits a different subset of parameters from the parent neural network.



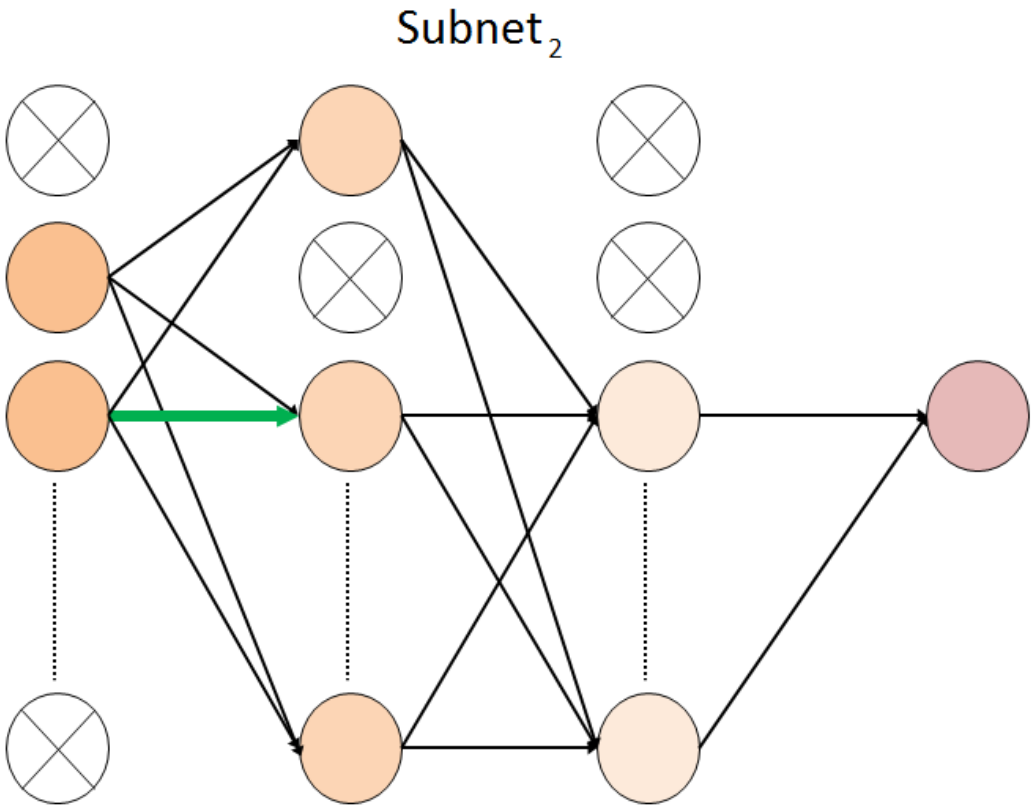
PARAMETER SHARING

- Parameter sharing:
  - In the case of bagging, the models are all independent.
  - In dropout on the other hand, all models share parameters. That means each model inherits a different subset of parameters from the parent neural network.



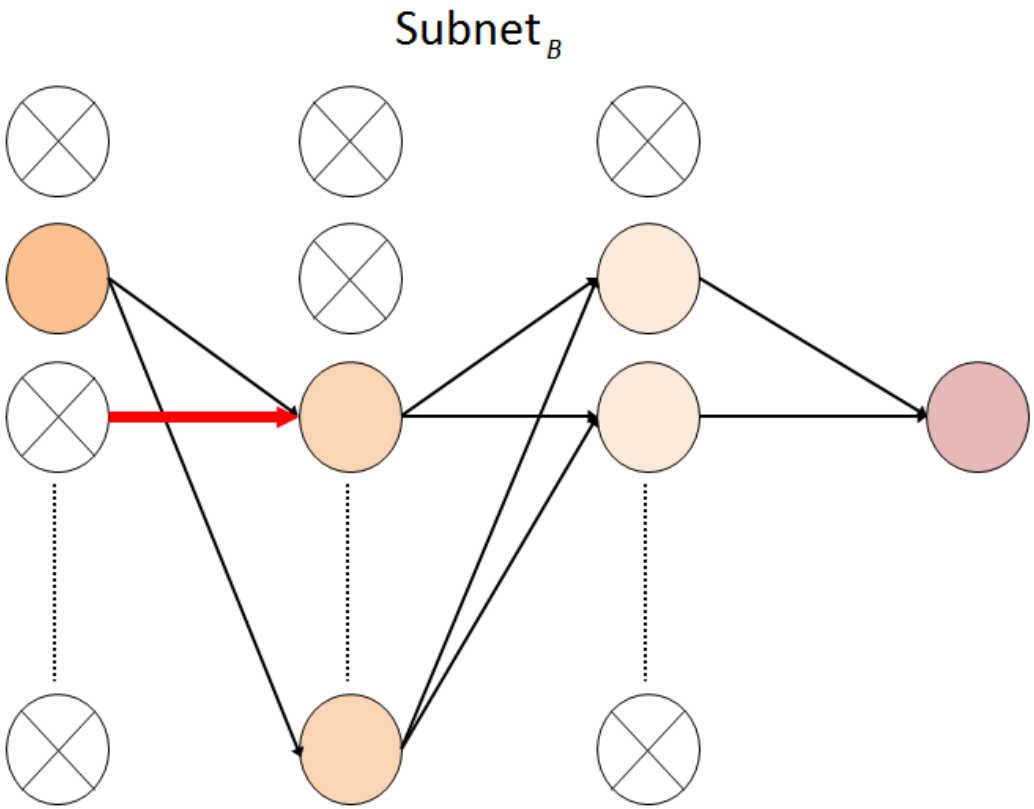
PARAMETER SHARING

- Parameter sharing:
  - In the case of bagging, the models are all independent.
  - In dropout on the other hand, all models share parameters. That means each model inherits a different subset of parameters from the parent neural network.



PARAMETER SHARING

- Parameter sharing:
  - In the case of bagging, the models are all independent.
  - In dropout on the other hand, all models share parameters. That means each model inherits a different subset of parameters from the parent neural network.



- Models output **probability distributions**: $p(y = y_j|x, \mu)$
- Bagging: arithmetic mean:

$$\tilde{p}_{ensemble}(y = y_k|x) = \frac{1}{B}\sum_{i=1}^{B} p^{(i)}(y = y_k|x)$$

- Dropout: more robust weighting via geometric mean:

$$\tilde{p}_{ensemble}(y = y_k|x) = \sqrt[2^B]{\prod_{\mu} p(y = y_k|x, \mu)}$$

and normalized for prediction:

$$p_{ensemble}(y = y_k|x) = \frac{\tilde{p}_{ensemble}(y = y_k|x)}{\sum_j \tilde{p}_{ensemble}(y = y_k|x)}$$

ILLUSTRATION GEOMETRIC MEAN IN DROPOUT

	$P(y = y_1 \mid x)$	$P(y = y_2 \mid x)$	$P(y = y_3 \mid x)$	$\sum$
Model 1	0.20	0.70	0.10	1.00
Model 2	0.10	0.80	0.10	1.00
Model 3	0.05	0.90	0.05	1.00
Model 4	0.05	0.90	0.05	1.00
Model 5	**0.80**	**0.10**	**0.10**	**1.00**
Arithmetic mean	0.24	0.68	0.08	1.00
Geometric mean	0.13	0.54	0.08	0.75
Re-normalized	0.18	0.72	0.10	1.00

$$mean_{arithmetic}(x_1, ..., x_n) = \frac{1}{n} \sum_{i=1}^{n} x_i$$

$$mean_{geometric}(x_1, ..., x_n) = \sqrt[n]{\prod_{i=1}^{n} x_i} \text{ with } x_i > 0 \forall i = 1, .., n$$

DROPOUT

	Bagging	Dropout
Basic Idea	model averaging	
# models	B	up to 2^B
Prediction	atrithmetic mean	geometric mean
Parameters	B independent models	parameter sharing
Train method	each model to convergence	each sub-net trained on mini batch restricted by μ

DROPOUT: THEORY VS PRACTICE

- Computing the complete ensemble is to expensive in practice!
- Instead we approximate it.

Algorithm 1 Training a neural network with dropout

1: Define parent network and initialize weights
2: **for** each training sample: **do**
3: Draw mask μ
4: Compute forward pass for $network_\mu$
5: Update the weights of $network_\mu$, e.g. by performing a gradient step with weight decay
6: **end for**

- For prediction: use weight scaling rule.

DROPOUT

- Weight scaling rule (Hinton et al. (2012)):
  - Approximate $p_{ensemble}$ by inspection of the **complete model**
  - Multiply shared weights of the trained model coming out of unit (neuron) *i* by *p*

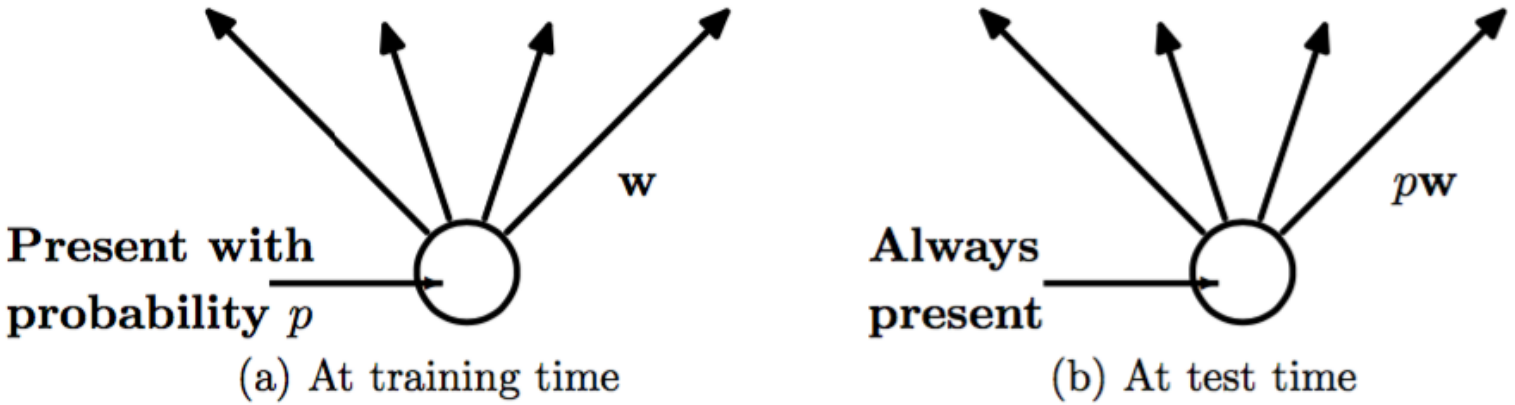


Figure: (Goodfellow et al. (2016))

DROPOUT - EXAMPLE

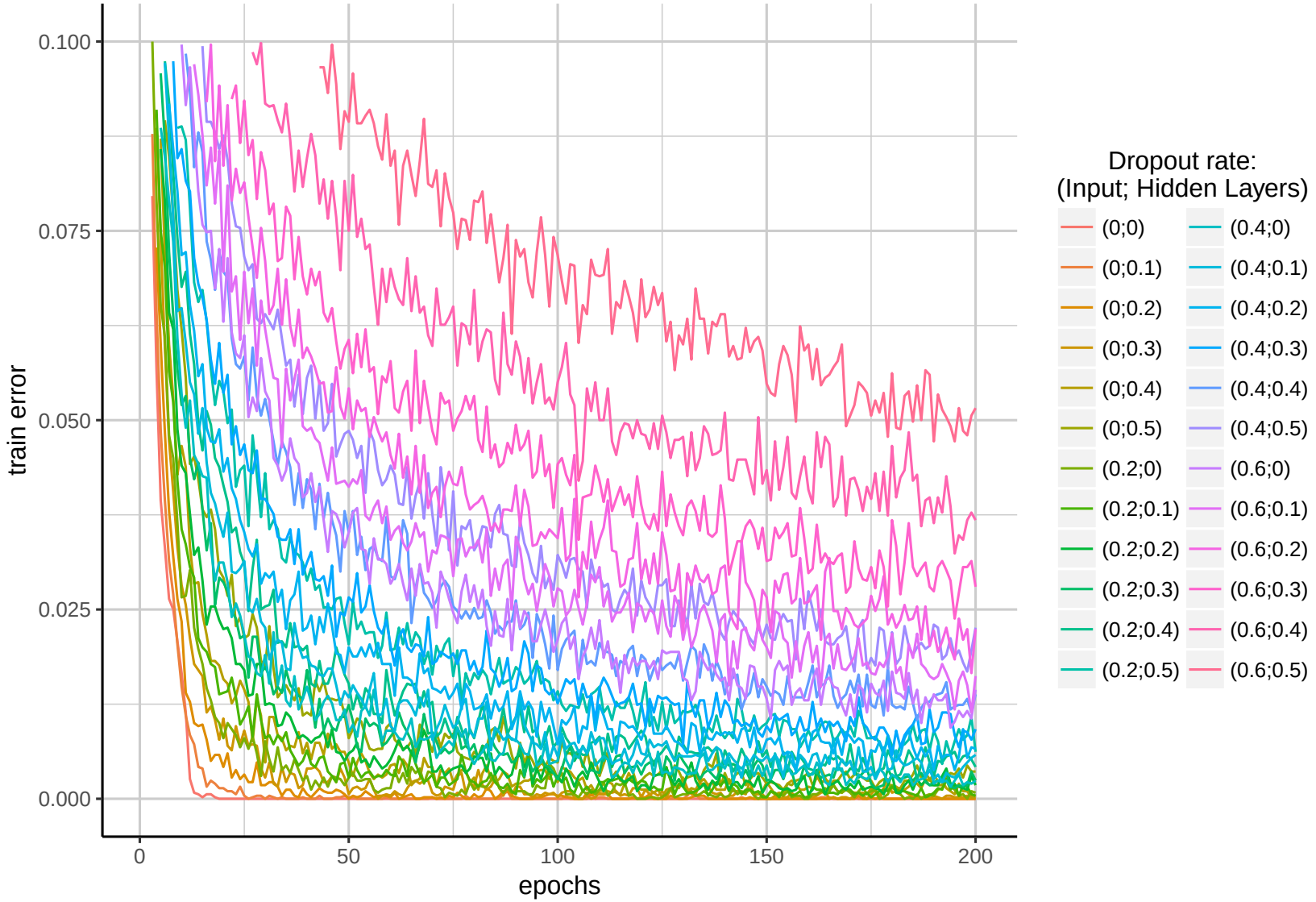
- To demonstrate how dropout can easily improve generalization we compute various models

```
drop0 = mx.symbol.Dropout(data, p = dropoutInputValues)
fc1 = mx.symbol.FullyConnected(drop0, name = "fc1", num_hidden = 512)
act1 = mx.symbol.Activation(fc1, name = "relu1", act_type = "relu")
drop1 = mx.symbol.Dropout(act1, p = dropoutLayerValues)
fc2 = mx.symbol.FullyConnected(drop1, name = "fc2", num_hidden = 512)
act2 = mx.symbol.Activation(fc2, name = "relu2", act_type = "relu")
drop2 = mx.symbol.Dropout(act2, p = dropoutLayerValues)
fc3 = mx.symbol.FullyConnected(drop2, name = "fc3", num_hidden = 512)
act3 = mx.symbol.Activation(fc3, name = "relu3", act_type = "relu")
drop3 = mx.symbol.Dropout(act3, p = dropoutLayerValues)
fc4 = mx.symbol.FullyConnected(drop3, name = "fc4", num_hidden = 10)
softmax = mx.symbol.SoftmaxOutput(fc4, name = "sm")
```

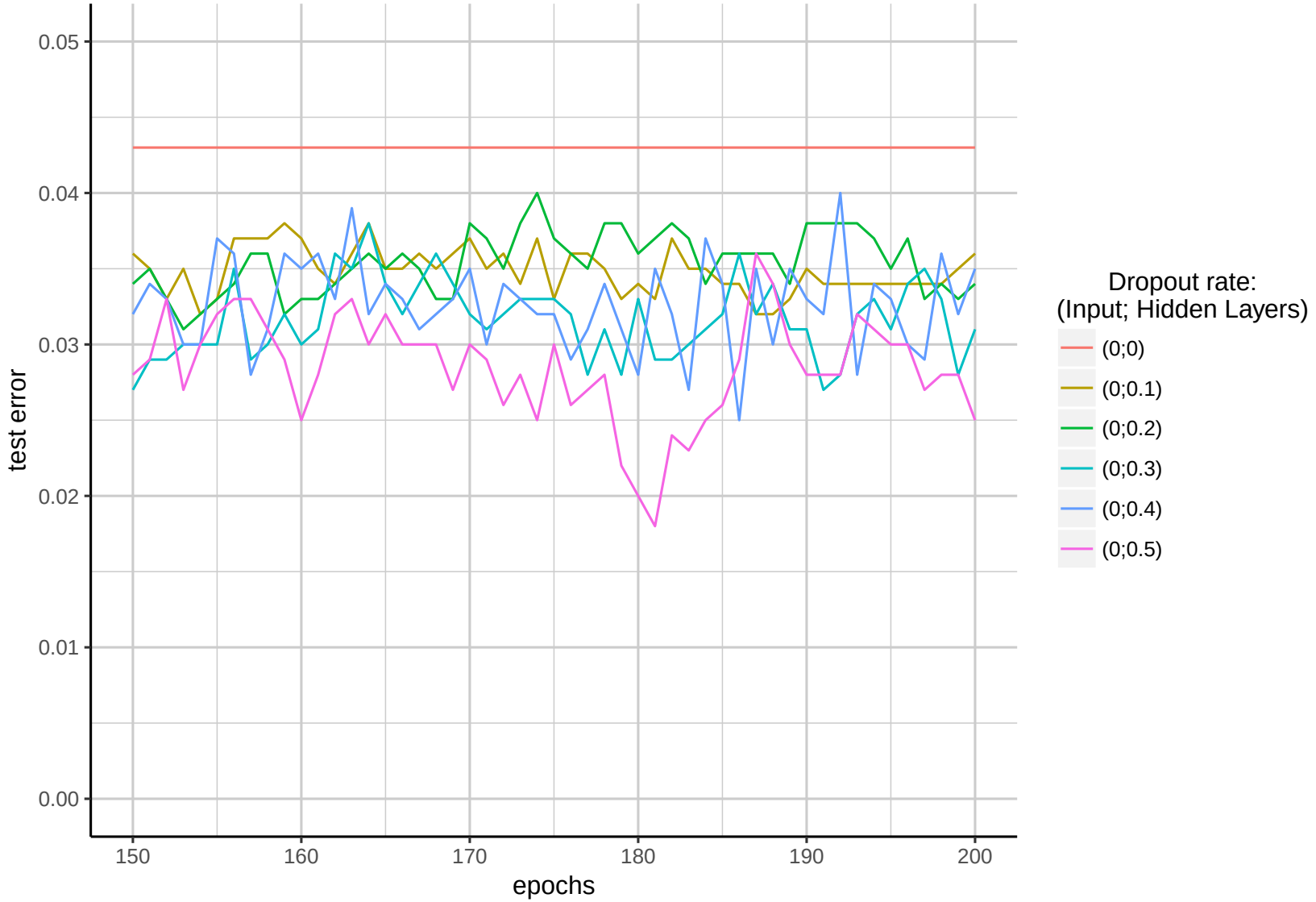
- We compute all models of the Cartesian product between the variables dropoutInputValues and dropoutLayerValues. That is

$$(0, 0.2, 0.4, 0.6) \times (0, 0.1, 0.2, 0.3, 0.4, 0.5)$$

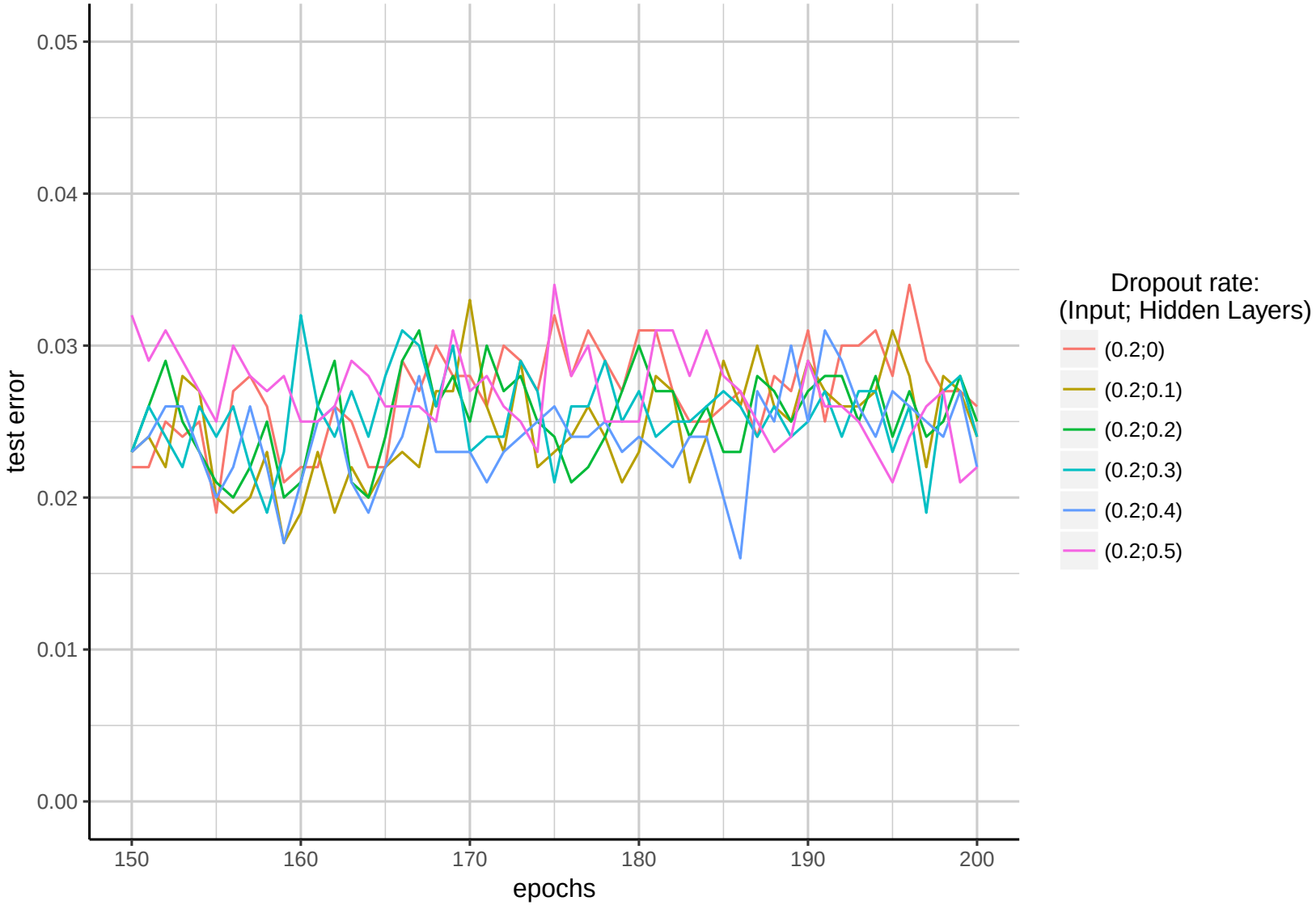
DROPOUT - EXAMPLE



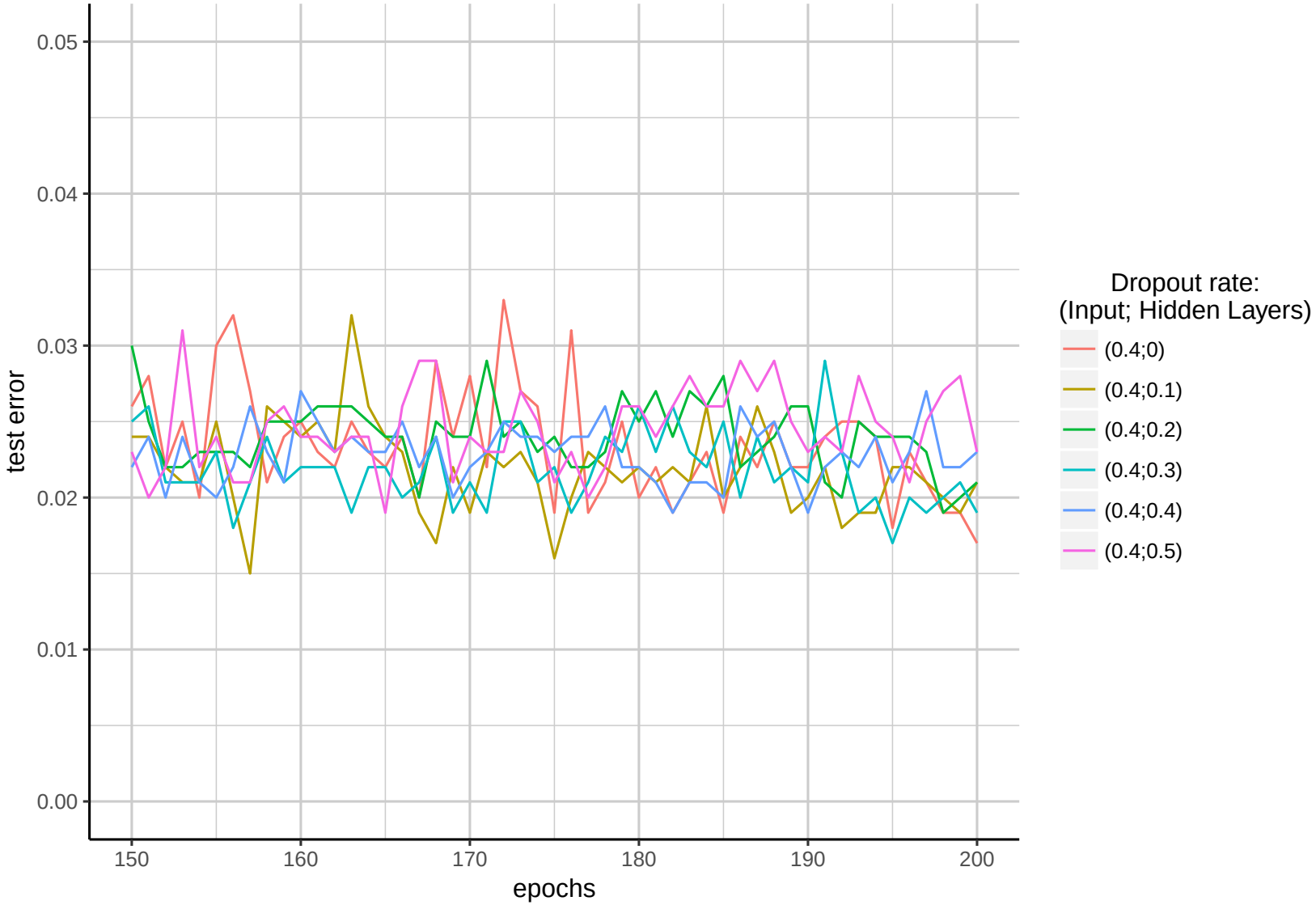
DROPOUT - EXAMPLE



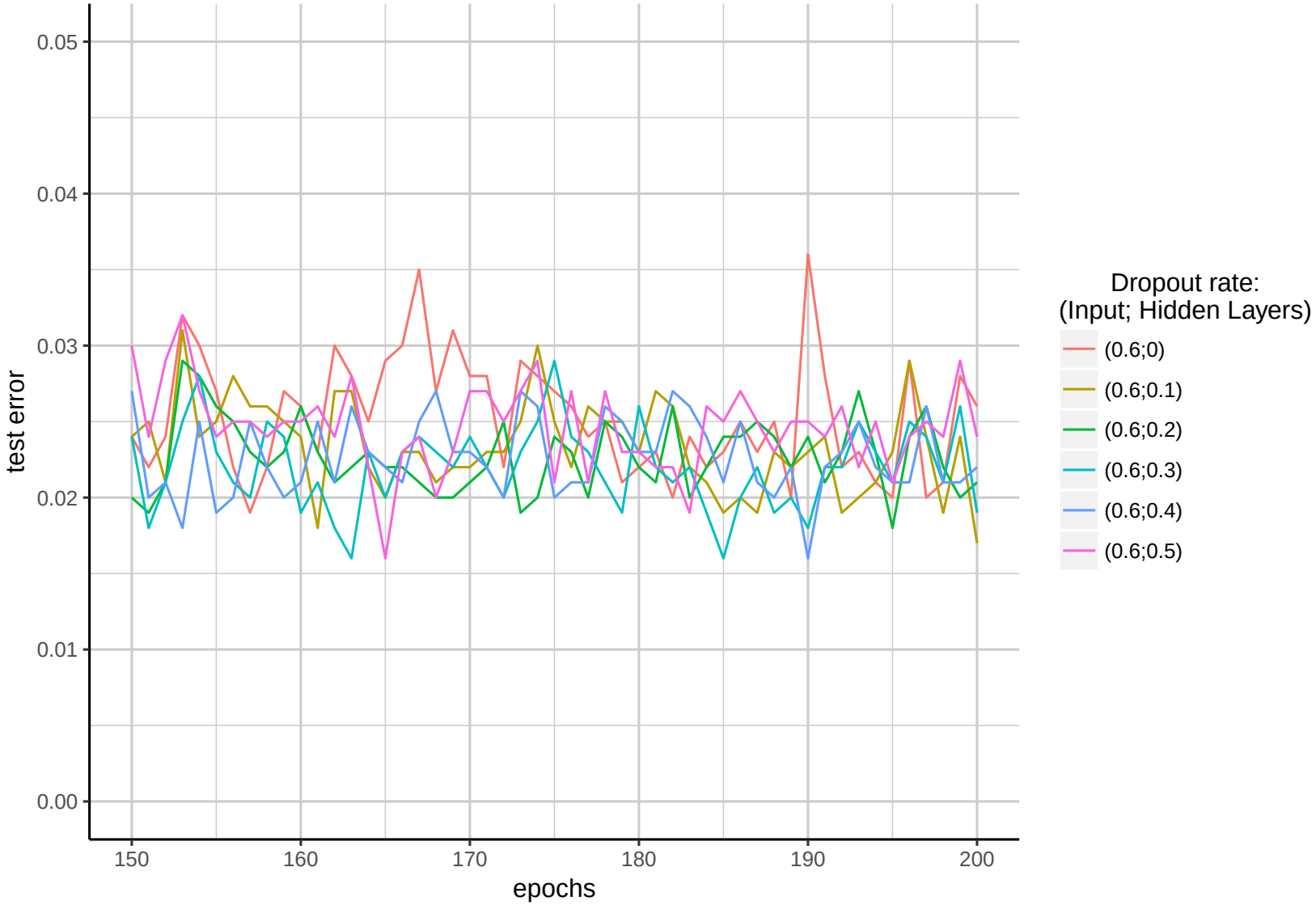
DROPOUT - EXAMPLE



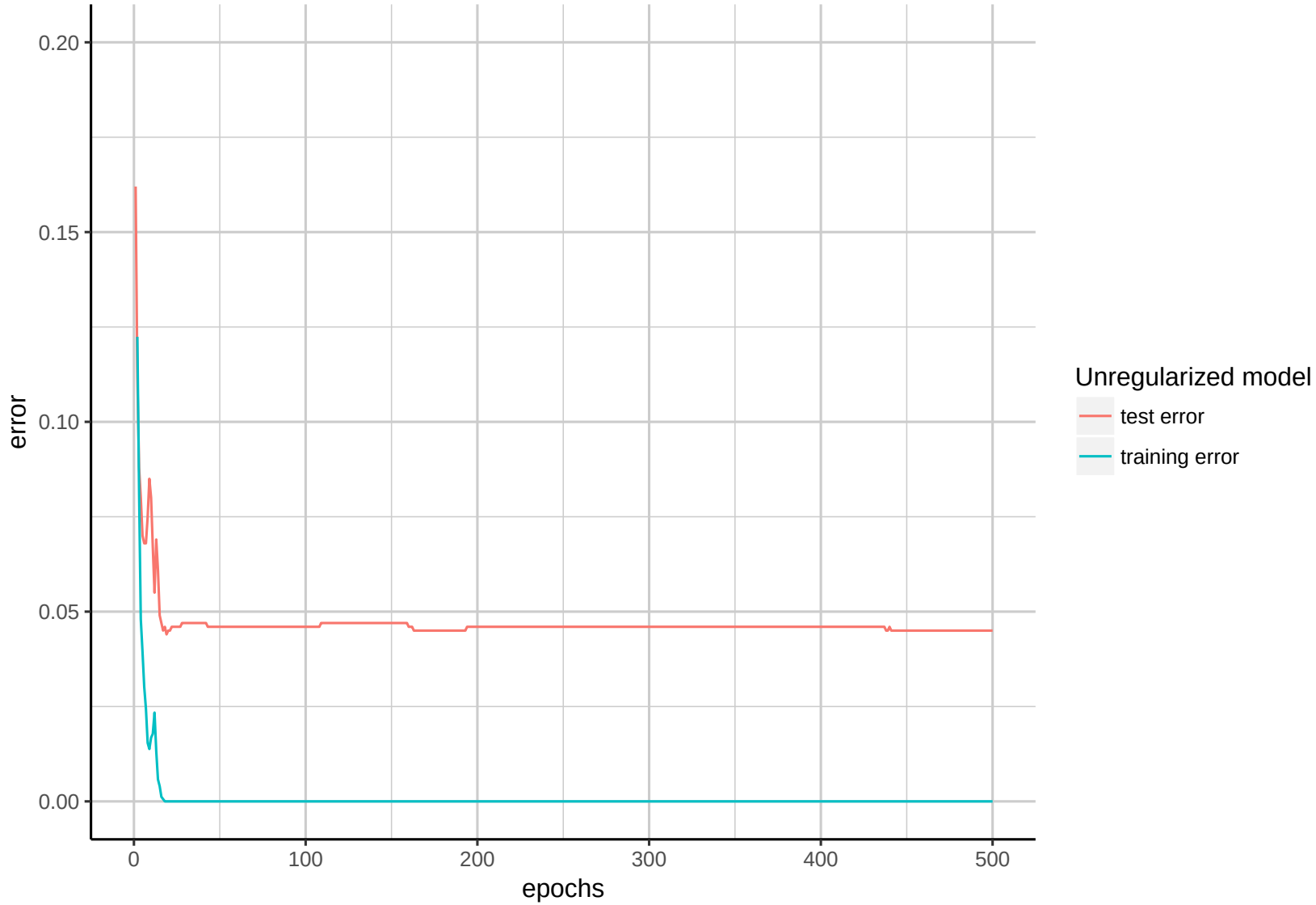
DROPOUT - EXAMPLE



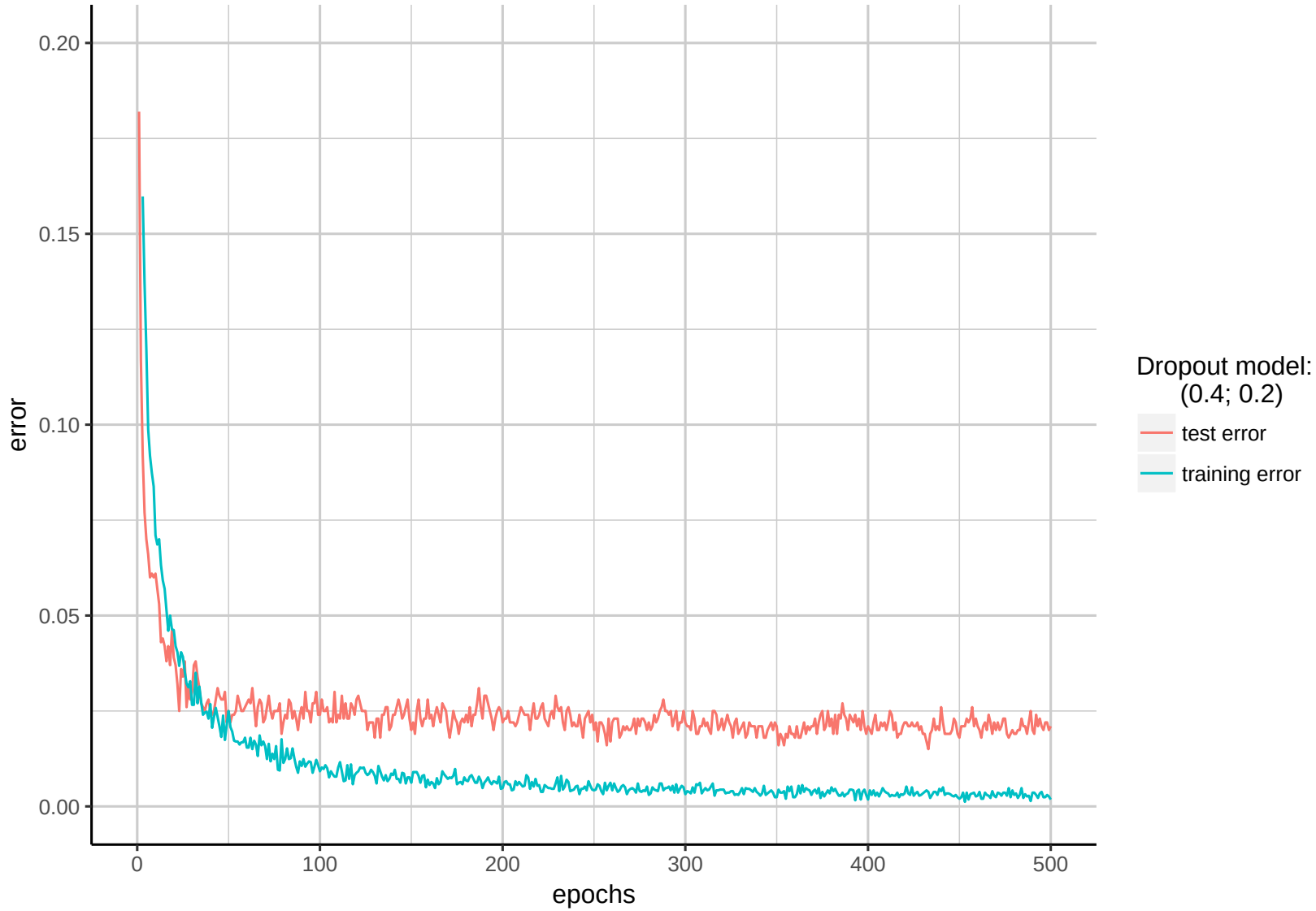
DROPOUT - EXAMPLE



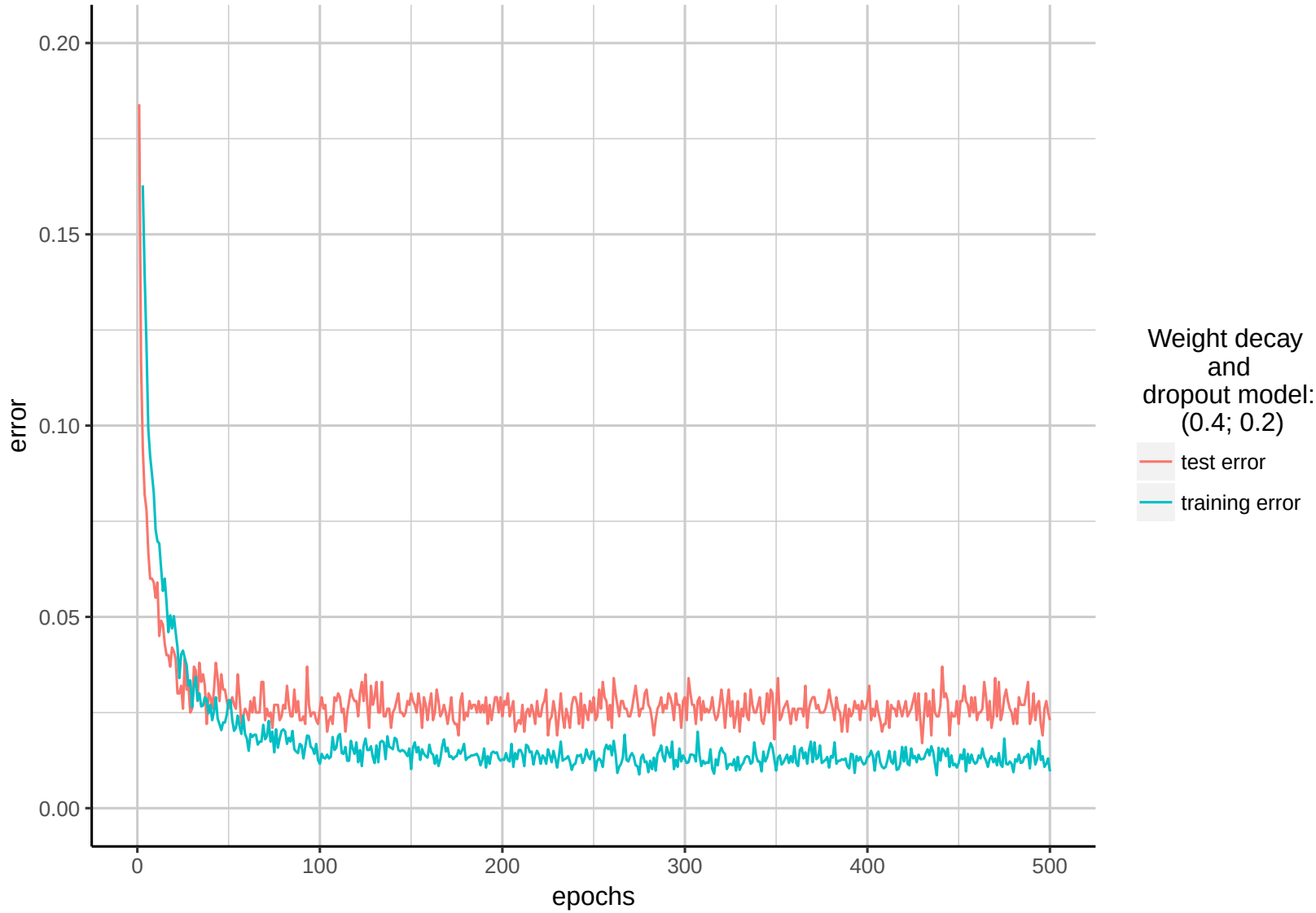
DROPOUT, WEIGHT DECAY OR BOTH?



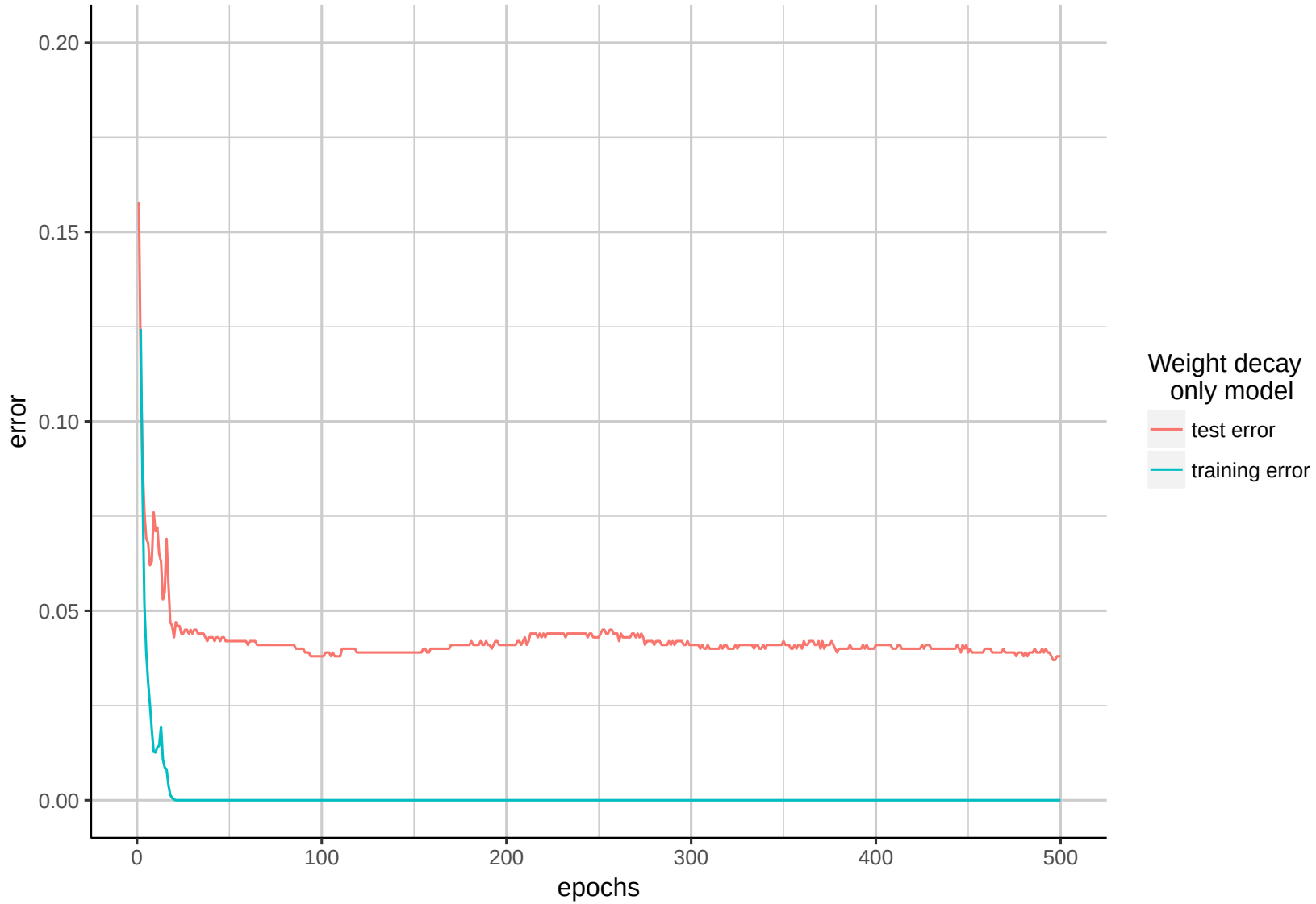
DROPOUT, WEIGHT DECAY OR BOTH?



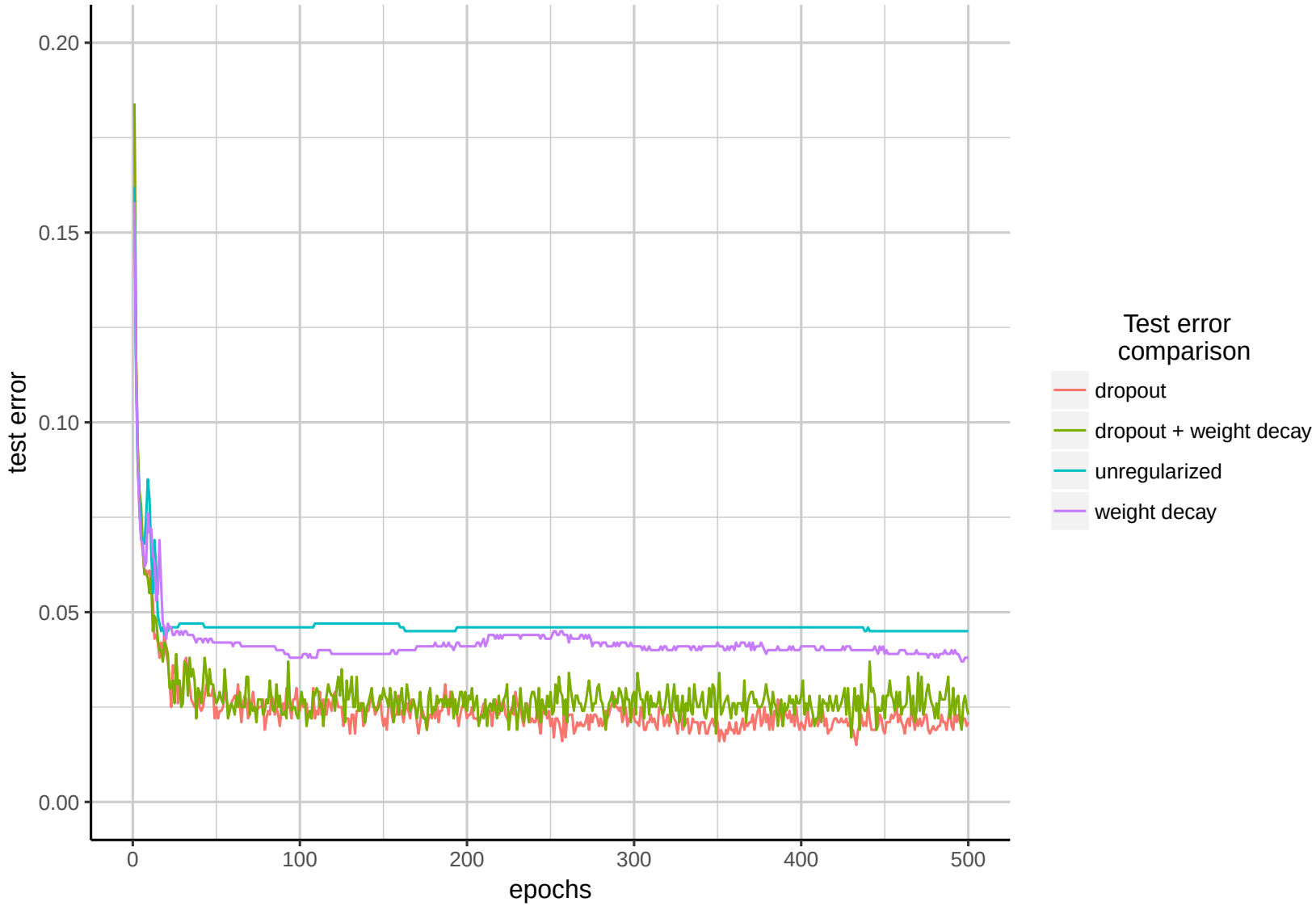
DROPOUT, WEIGHT DECAY OR BOTH?



DROPOUT, WEIGHT DECAY OR BOTH?



DROPOUT, WEIGHT DECAY OR BOTH?



DATASET AUGMENTATION

- Problem: low generalization because high ratio of

$$\frac{\text{complexity of the model}}{\text{\#train data}}$$

- Idea: artificially increase the train data.
  - Limited data supply $\rightarrow$ create “fake data”!
- Increase variation in inputs **without** changing the labels.
- Application:
  - Image and Object recognition (rotation, scaling, pixel translation, flipping, noise injection, vignetting, color casting, lens distortion, injection of random negatives)
  - Speech recognition (speed augmentation, vocal tract perturbation)

DATASET AUGMENTATION

(a) Original (b) Color

(c) Rotate (d) Horizontal Stretch

Figure: (Wu et al. (2015))

DATASET AUGMENTATION

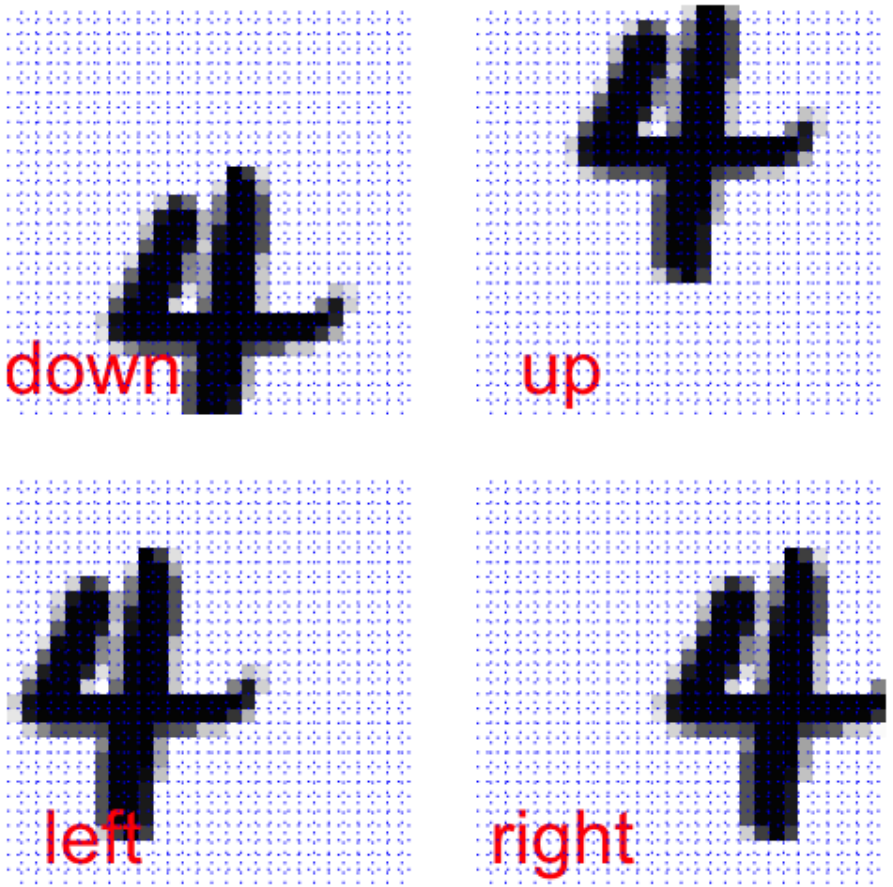


Figure: (Wu et al. (2015))

$\Rightarrow$ careful when rotating digits (6 will become 9 and vice versa)!

ARTIFICIAL NOISE

- Intentionally inject noise to the model, to make it more robust to small pertubations in the inputs.
- This method may force the model to “grow” weights in regions of flat minima.
  - Thus, the noisy model may not find perfect minima but its approximations lie in a flatter surrounding.
- Bishop (1995) shows that the injection of artifical noise has the same effect as norm penalization strategies.
- In practice, it is common to apply noise to the outputs.
  - This strategy is termed label smoothing as it incorporates a small noise term on the labels of the classification outputs. The intuition is to account for possible errors in the labeling process.

REFERENCES

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016)

Deep Learning

http://www.deeplearningbook.org/

Trevoe Hastie, Robert Tibshirani and Jerome Friedman (2009)

The Elements of Statistical Learning

https://statweb.stanford.edu/%7Etibs/ElemStatLearn/

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky Ilya Sutskever and Ruslan Salakhutdinov (2012)

Improving neural networks by preventing co-adaptation of feature detectors

http://arxiv.org/abs/1207.0580

Wu Ren, Yan Shengen, Shan Yi, Dang Qingqing and Sun Gang (2015)

Deep Image: Scaling up Image Recognition

https://arxiv.org/abs/1501.02876

REFERENCES

Bishop, Chris M. (1995)

Training with Noise is Equivalent to Tikhonov Regularization

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/bishop-tikhonov-nc-95.pdf