Deep Learning

Chapter 7

Regularization for Deep Learning

- In ML we want to generalize beyond the training data
- Many strategies designed to reduce test error (at the expense of increased training error)
- These strategies are known as **regularization**

Regularization for Deep Learning

- Example of regularization strategies
  - Add restrictions on parameter values
  - Add extra terms to the objective function
- Constraints and penalties designed to:
  - Encode prior knowledge
  - Prefer a simpler model class
  - Make underdetermined problem determined

- Combine hypotheses to explain training data (ensemble methods)

Parameter Norm Penalties

- Limit the **capacity** of the model by adding a parameter norm penalty
  - Capacity → the model's hypothesis space

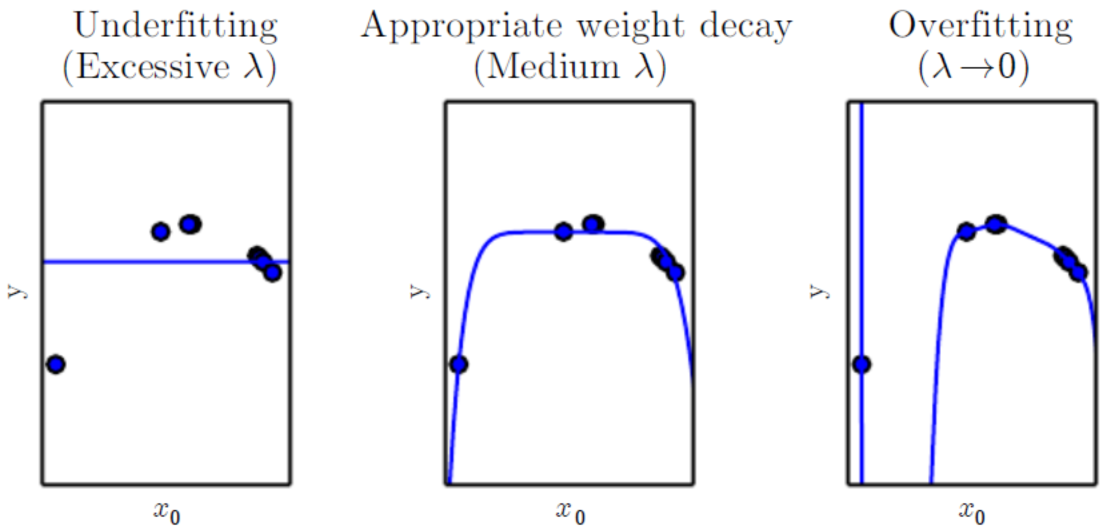
- Normal to only penalize the weights.
  - Leave the biases unregularized

- Weight Decay is the most common parameter norm penalty
  - Will drive the weights closer to the origin (0) → reducing the norm
  - Also known as L^2 parameter norm penalty

Weight Decay Example

$$J(\boldsymbol{w}) = \text{MSE}_{\text{train}} + \lambda \boldsymbol{w}^\top \boldsymbol{w}$$

- Where
  - MSE (mean-squared error)
  - **w** is vector of weights
  - λ – value deciding our preference for small weights



$L^2 and\ L^1 comparison$

- L^1regularization term: $\sum_i |w_i$
- $L^2 regularization\ term\ \frac{1}{2}||w||_2^2$

- L^1results in a solution that is more sparse
  - Sparse → parameters with optimal value of zero
  - Extensively used as a feature selector mechanism
  - Weights than end up being zero suggest the corresponding features can be removed safely

Dataset Augmentation

- More examples → Generalizes better
- We can get more (fake) data by augmenting our dataset
- Examples:
  - For images: rotate, scale, translate by a few pixels
  - Injecting noise to inputs
  - Dropout(more later) – construct new inputs by multiplying noise

- Proven to be a particularly effective technique for object recognition

Noise Robustness

- Add (infinitesimal) noise to input → impose penalty on the norm of the weights
- Add noise to hidden units (7.12)
- Add noise to the weights (primarily recurrent NN)

- Common to have mistakes in the y labels
  - Prevent by explicitly model noise at output layer

Semi-Supervised Learning

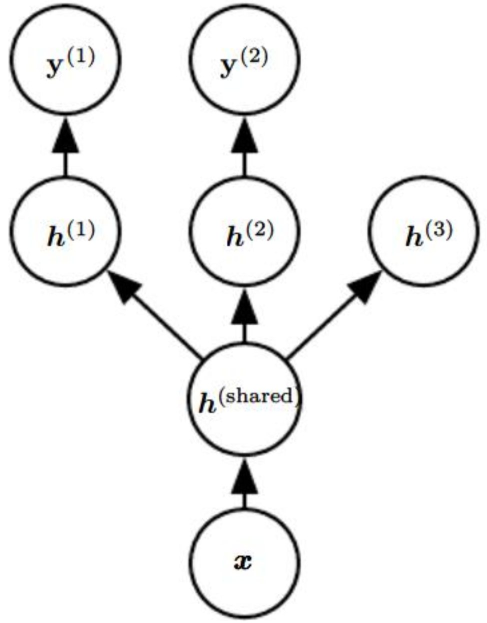
- Combination of labeled and unlabeled training examples
- Estimate P (y | x) using:
  - Labeled examples P(x,y)
  - Unlabeled examples P(x)

- Goal – learn a representation such that examples from same class have similar representations

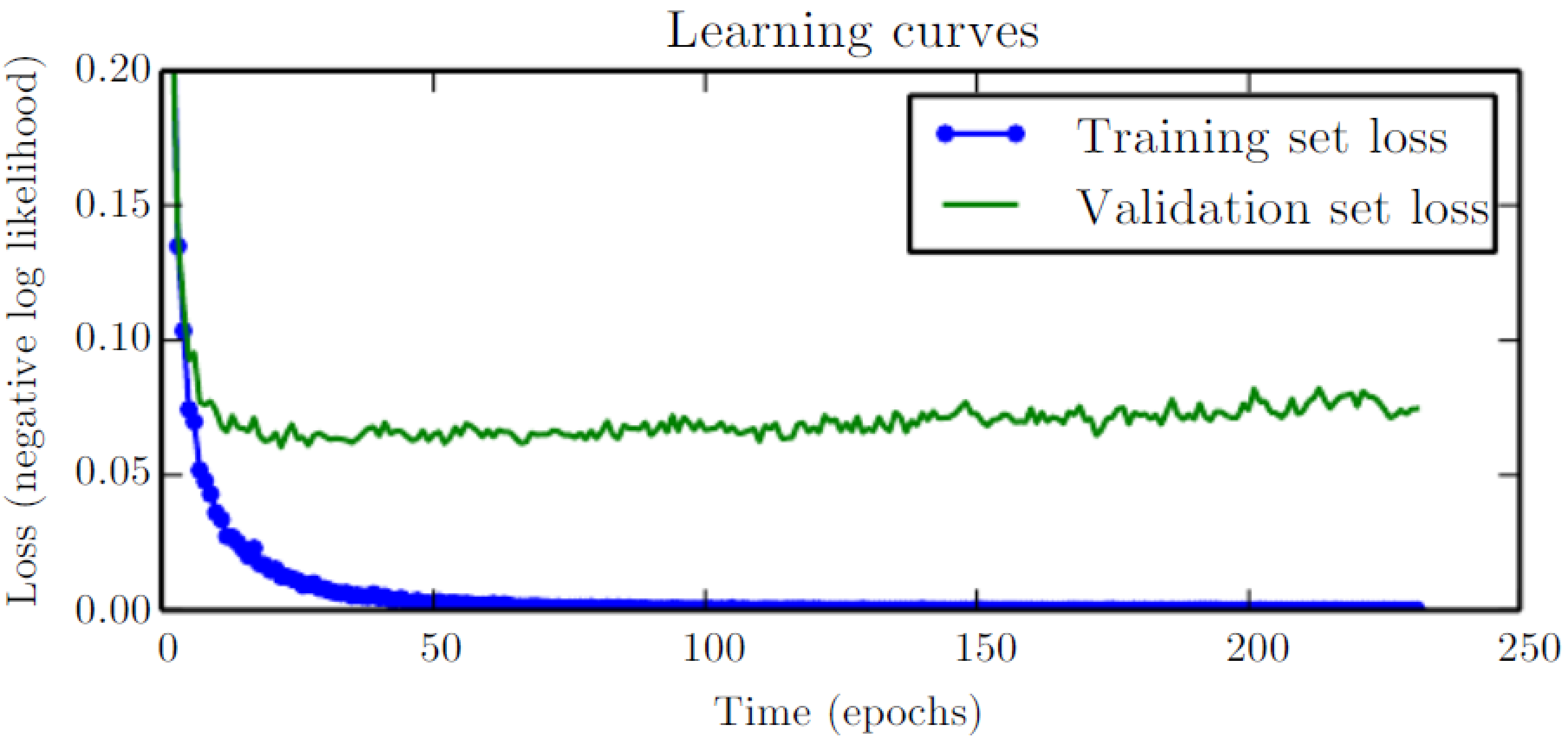
Multi-Task Learning

- Improve generalization by pooling examples from several tasks
  - Assumes that sharing a part of the model among different tasks is justified

- Figure can be split in two parts:
  1. Task-specific parameters (upper part)
  2. Generic parameters, shared across tasks (lower part)

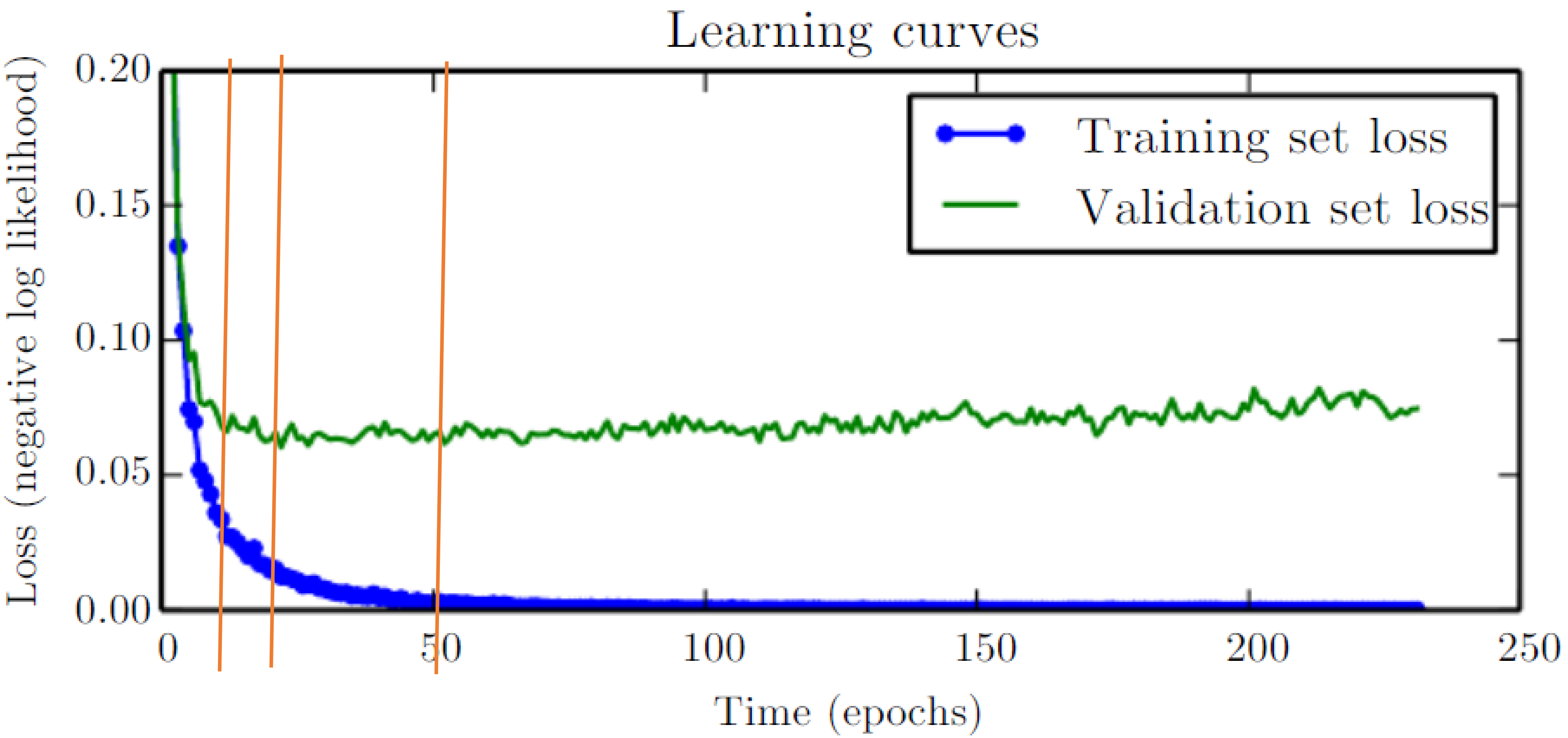


Early stopping



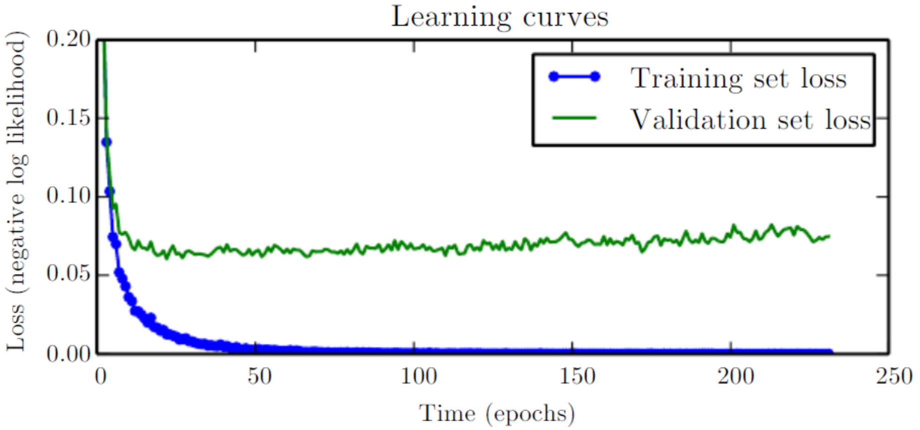
- Overcapacity in model → overfitting
- Training error decreases, but validation error starts to increase after some time.

Early stopping



- Store solution when new minimum in validation error
- No improvement for some time → done

Early stopping

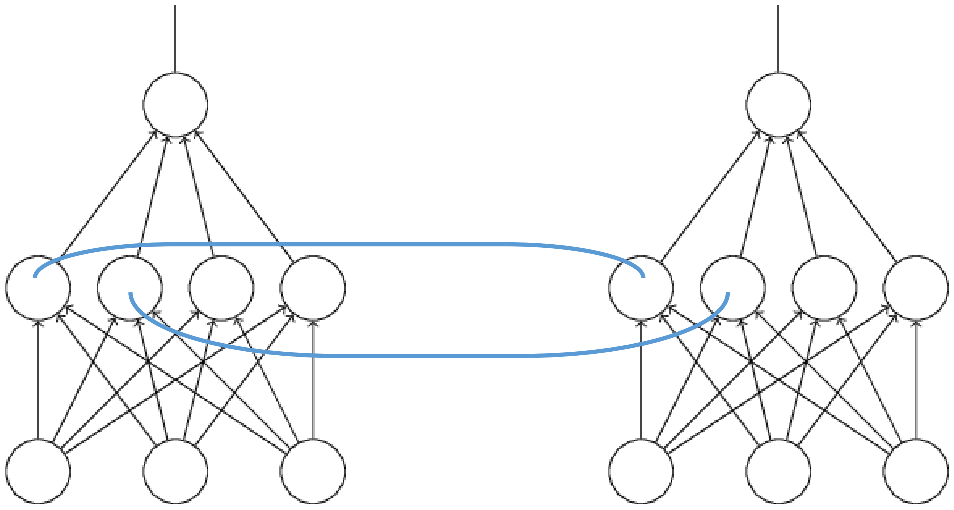


- Effective and simple
- Reduces computational cost. No extra elements are added to model.
- Hyperparameter selection (number of epochs ~ capacity)
- Negative: Must evaluate validation set often
  - Can run this evaluation in parallell if hardware is available
  - Can use small validation set, or infrequent evalutaions.
- Validation set → less data to train on
  - Keep only hyperparameter and train again on all data
  - Keep all parameters, and train again on all data.

Parameter Tying and Parameter Sharing

We have prior knowledge about suitable parameter values.

Usage 1: Two networks that we believe should have somewhat similar parameters. *Parameter tying*.



Add a parameter norm penalty in the cost function. $||\boldsymbol{w}_a + \boldsymbol{w}_b||^2$

Parameter Tying and Parameter Sharing

We have prior knowledge about suitable parameter values.

Usage 2: Force sets of parameters to be equal. *Parameter sharing*

- Far less parameters!
- Convolutional networks
  - Find the cat no matter where in the image it is.
  - The same feature (cat) is searched for at different locations.
  - hidden units with equal weights, but different input subset.

Sparse Representations

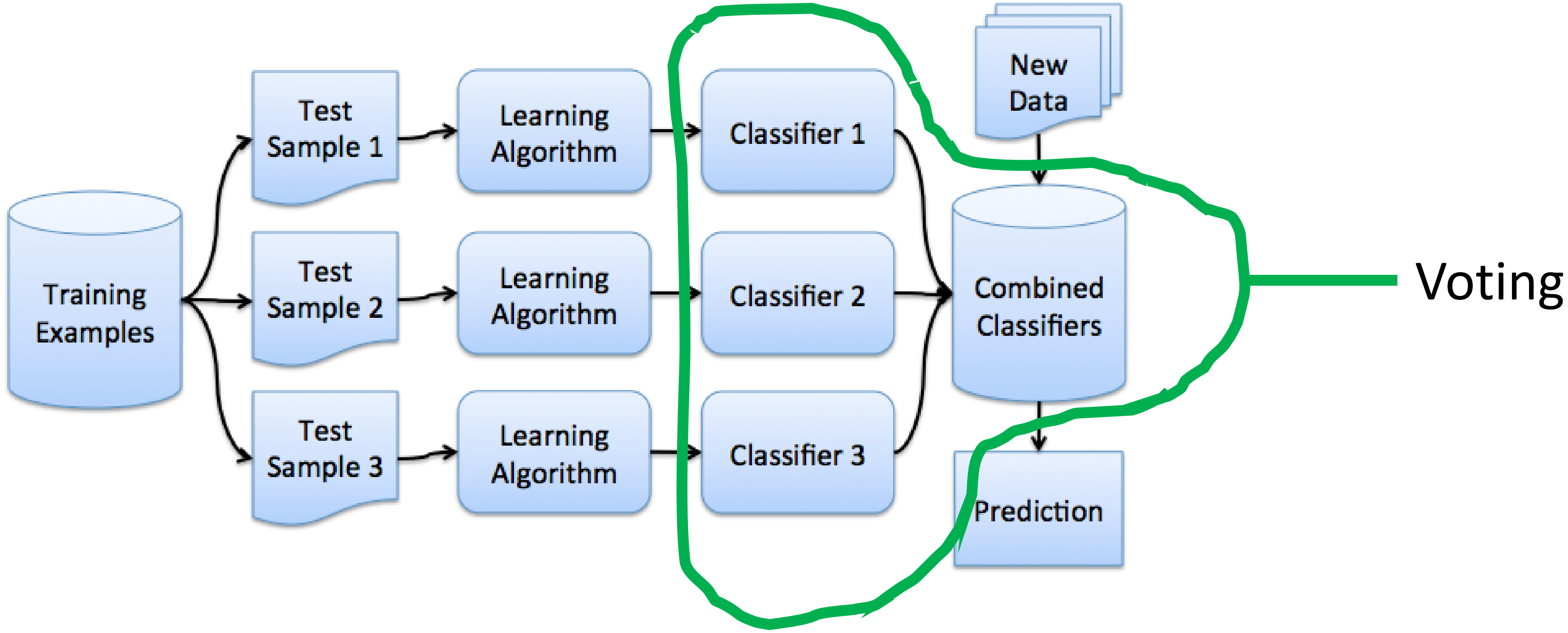
- Sparse parameterization (L1) → Many weights become zero.
- Sparse representation → Many activation outputs become zero.
- Add a penalty term to the cost function.

The goal is to represent data as simple as possible, from one layer to the next. Generalization.

Bagging

Combining several models

- Several models vote on classification



Bagging

Combining several models

- Models will make different errors.
- On average, performs at least as well as any single model
- If models make independent errors → significantly better performance

Bagging

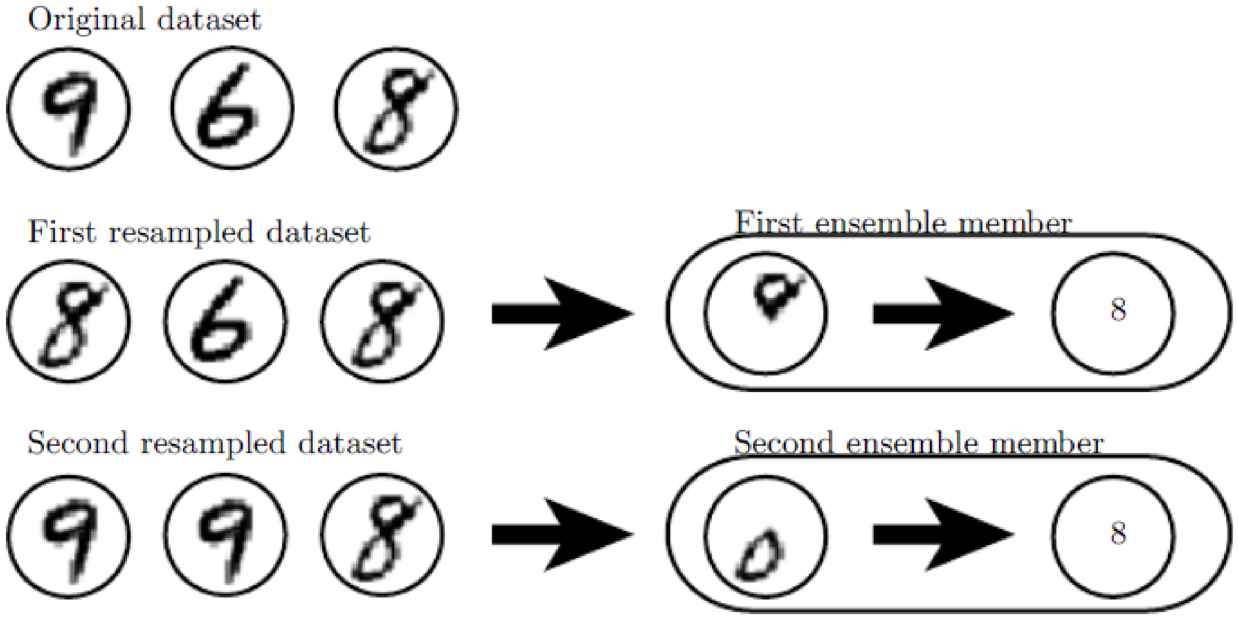
Combining several models

- Bagging is one of many ensemble methods.
- Ensemble methods in general:
  - Each model can use a different objective function.
  - Each model can use a different training algorithm.
- Bagging in particular:
  - Same objective function can be reused.
  - Same training algorithm can be reused.

Bagging

Combining several models

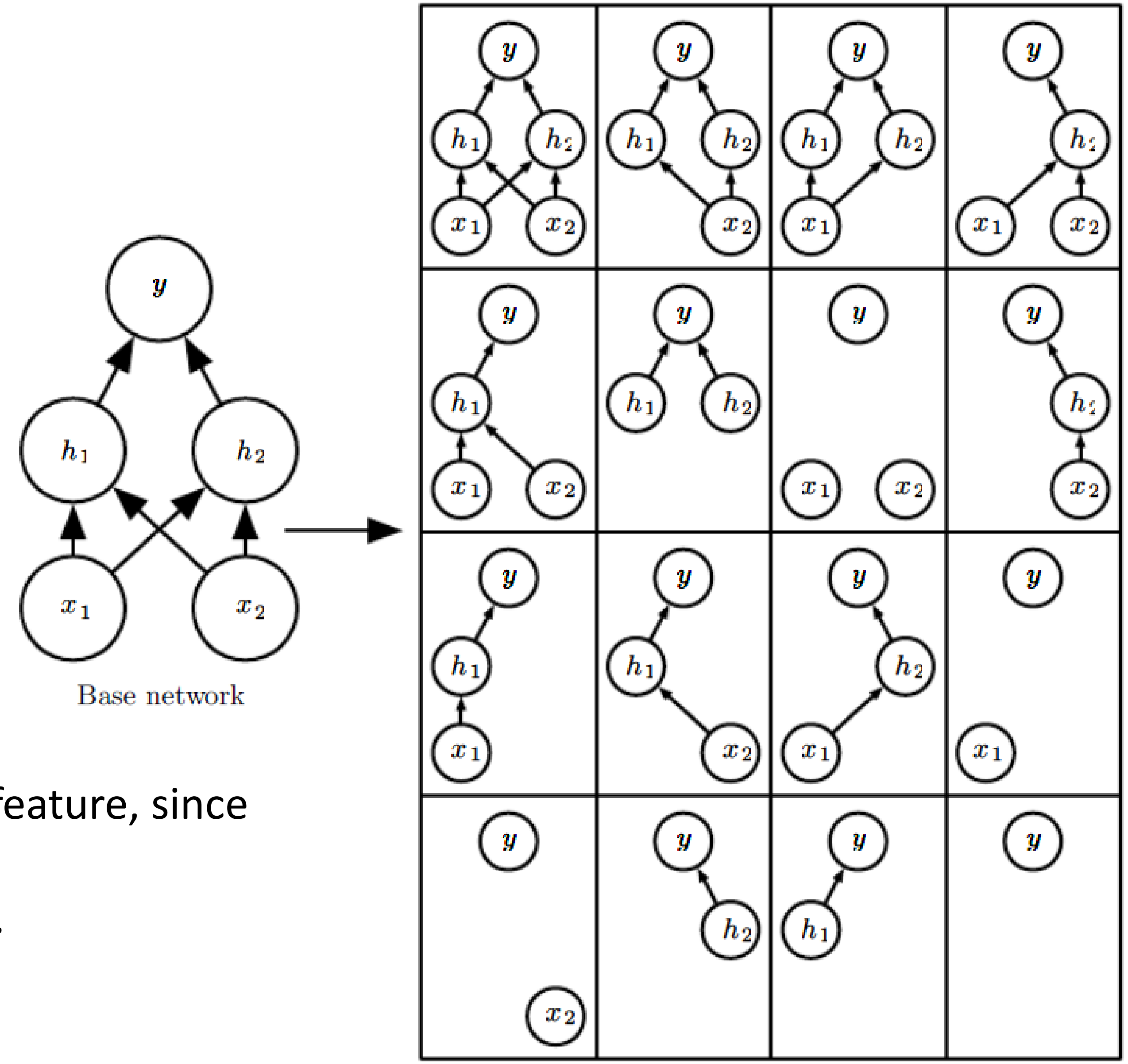
- Solution 1:
  - Random initialization is enough to have models make independent errors.
- Solution 2:
  - Sampling with replacement



Dropout

«Bagging» for large networks

- Use one base network during training.
- For each mini-batch, apply a different mask, telling us which nodes to remove
- After end of mini-batch, all nodes are included again.
- Hyperparameter: Probability of including each node, e.g. 0,5.
- Each model will inherit a different subset of parameters from the last model.
- The network will learn to not rely on a certain feature, since the feature may be missing.
- Infeasible to train on all possible sub-networks.



Base network

Ensemble of Sub-Networks

Dropout

«Bagging» for large networks

Inference

- Average results using several different masks, e.g. 20.
- Even better: Only one inference including all nodes, but activation level from all nodes is multiplied by probability of including that node.
- Probability of node inclusion ½ → divide all activation levels by 2 before infering.

Dropout

«Bagging» for large networks

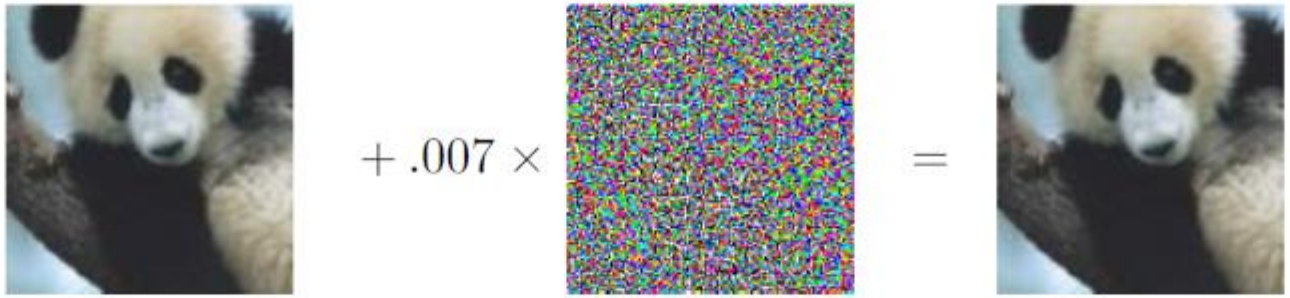
Pros:

- Computationally cheap
- Can be applied to many types of models (feedforward neural networks, recurrent networks, …)
- Each hidden unit is trained to perform well, regardless of which other hidden units are included.

Cons:

- Model capacity is reduced → increase size of model to compensate.

Adversial Training

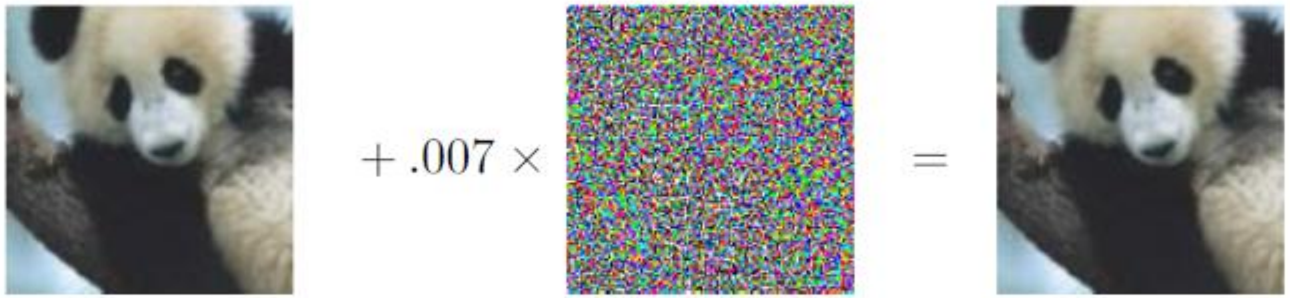


58% sure it is a panda

99% sure it is a gibbon

- Even networks with human level accuracy have a near 100% error rate on adversial data. Not so human after all?
- Adversial training: Training on adversial data with purpose of reducing error rate on original test set.

Adversial Training



58% sure it is a panda

99% sure it is a gibbon

- Even tiny changes to input can result in large changes in cost function.
- AT encourages network to be stable in the neighborhood of the training data. This makes the classifier more general.

Tangents and manifolds

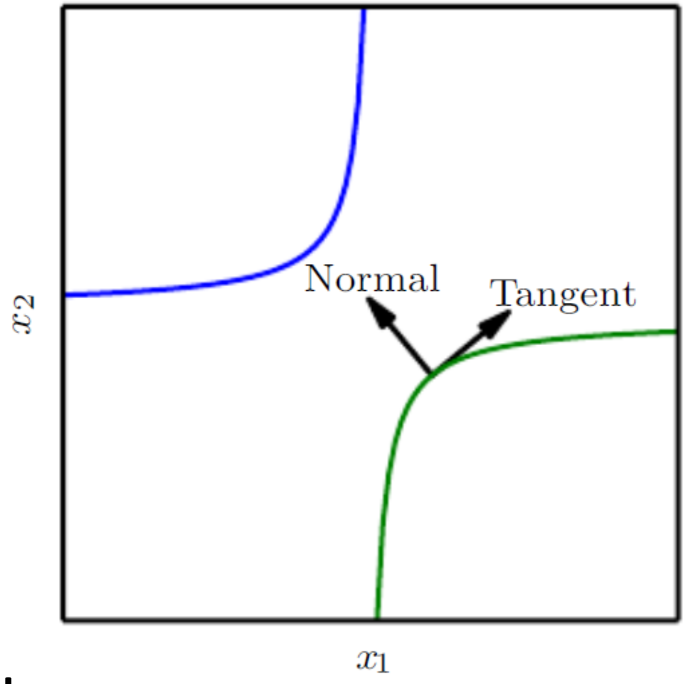
- Aim is to overcome *the curse of dimensionality*
- Assume that data lies near a **low**-dimensional manifold.
- Three algorithms:

Tangents and manifolds

1 – Tangent distance algorithm

- Nearest-neighbor
- Requires knowledge of the manifolds near which probability concentrates in the model
- Assumes that examples on the same manifold share category.
- Nearest-neighbor distance between data points x1 and x2: Distance between the manifolds that they are closest to.
  - The manifold is approximated by its tangent plane at x → Meassure distance between two tangent planes instead.

Tangents and manifolds



2 – Tangent propagation algorithm

- Add penalty to cost function that makes the network more invariant to known factors of variation – movement along manifolds.
- Require gradient of cost function to be orthogonal to the known manifold tangent vectors.
- Tangent vectors must be known from before – from knowledge of the effect of common transformations like translation.

Tangents and manifolds

3 – Manifold tangent classifier

- No need to know the manifold tangent vectors from before.
- Autoencoders can estimate these instead.