Deep Learning

CS256 - Topics in Artificial Intelligence

February 14, 2018

Overview

Regularization

Regularization

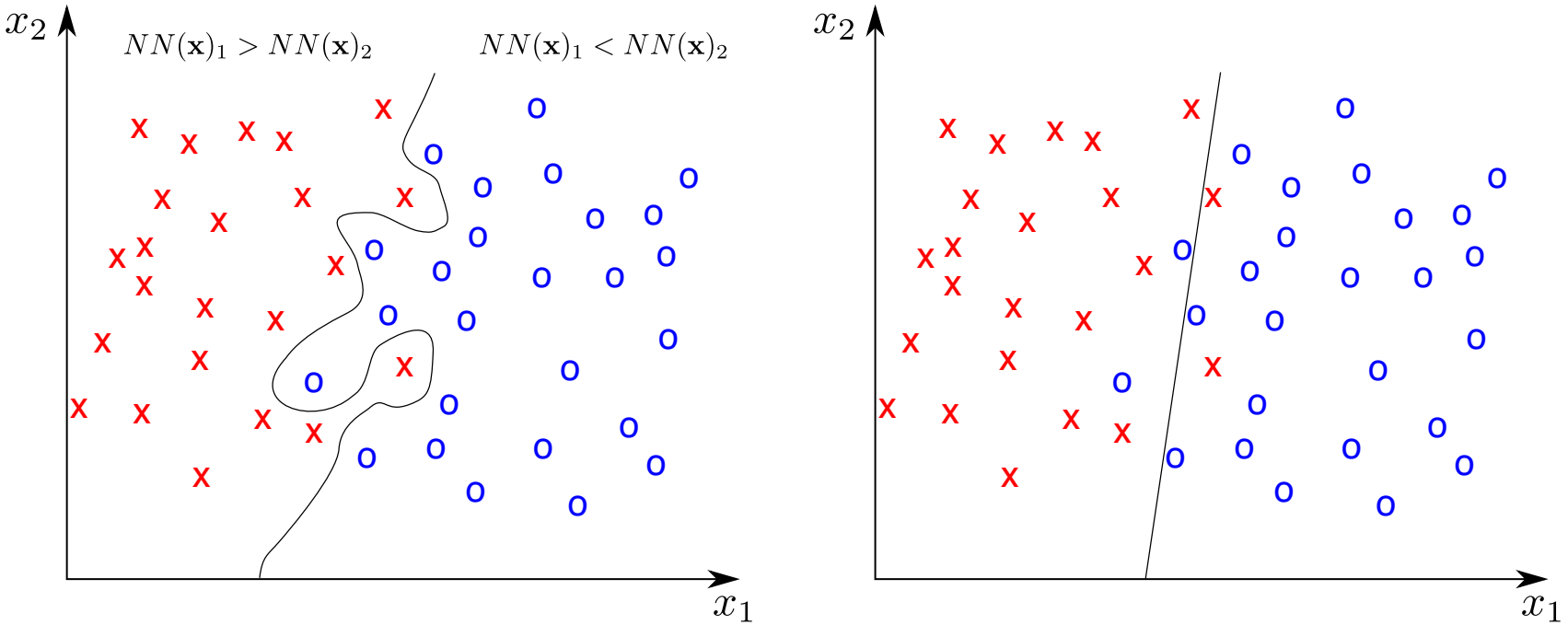
Regularization:

> *Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error. These strategies are known collectively as regularization.*

– Deep Learning; Ian Goodfellow, Yoshua Bengio, Aaron Courvill

Validation Set

We have seen that using a validation set can be used to prevent over-fitting.



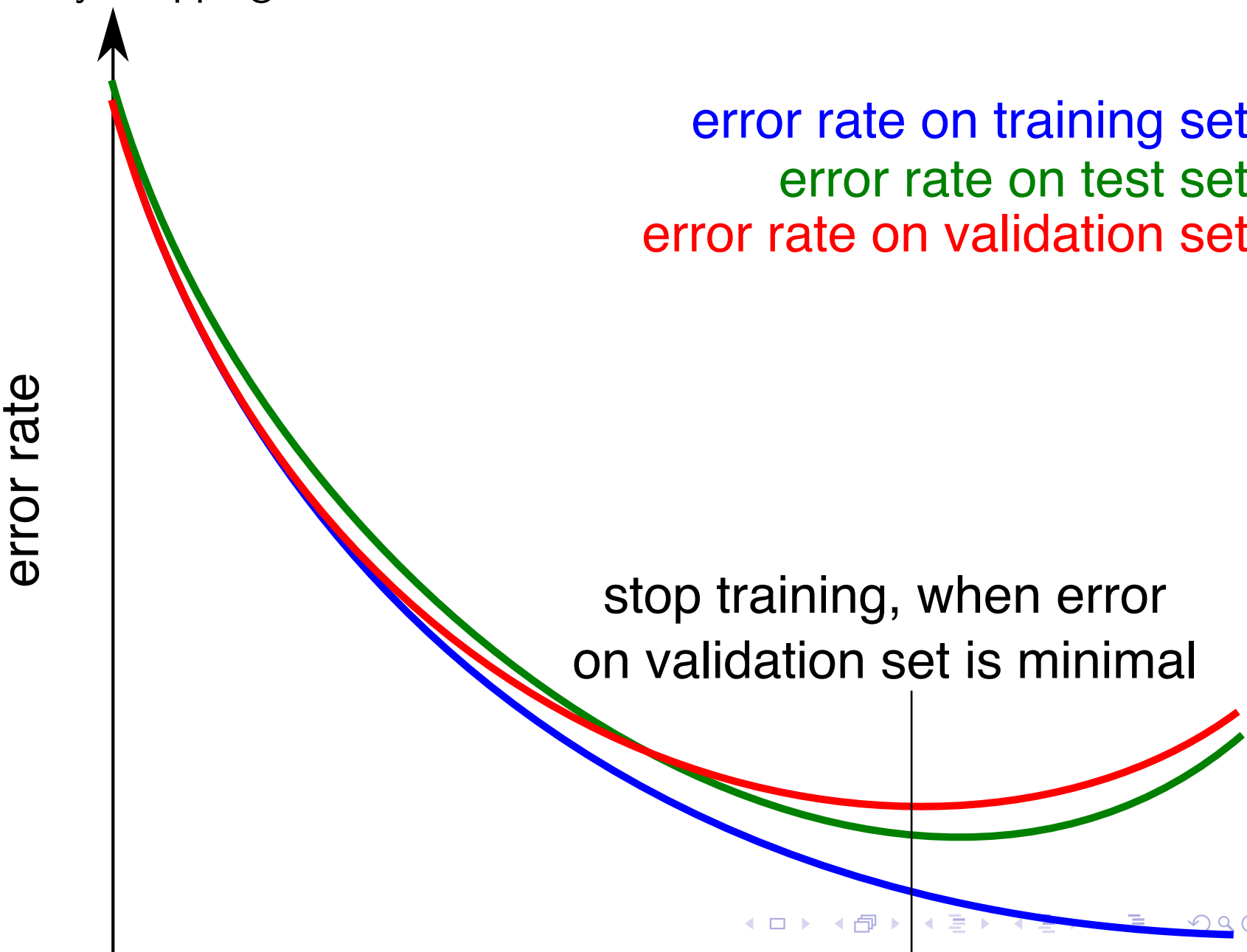
Regularization Overview

An overview of the most frequently used regularization criteria

- Limit the NN
  - Early stopping (validation set)
  - Limit weight range
    - weight decay
    - L1/L2 Regularization
- averaging separate nets
  - bagging
  - dropout layer
- Dataset augmentation
  - add noise to data

Limiting the capabilities of the Neural Network

Early Stopping:



Limiting the capabilities of the Neural Network

Limiting the weight range.

- We want to keep the individual weights close to 0.
- This prevents pathological cases (maybe because of outliers) where weights get too large
- Thus, we introduce a penalty to the error function, namely the L2 norm of the weights

$$E(X, Y, W) \rightarrow \hat{E}(X, Y, W) = E(X, Y, W) + \frac{\alpha}{2} \|W\|_2^2$$

- Now, we have a new hyper-parameter α to regulate the influence of the weights.
  - If $\alpha = 0$, no regularization (old error function)
  - Large values α increase the impact of forcing the weights to be small

L2 Regularization

Let's look at the error function with weight penalty term

$$\hat{E}(X, Y, W) = E(X, Y, W) + \frac{\alpha}{2} \|W\|_2^2$$

The gradient is then

$$\nabla \hat{E}(X, Y, W) = \nabla E(X, Y, W) + \alpha \cdot W$$

Thus, a weight update step is now

$$W' \leftarrow W_i - \eta \left(\nabla E(X, Y, W_i) + \alpha \cdot W_i\right)$$
$$W' \leftarrow (1 - \eta\alpha) W_i - \eta \nabla E(X, Y, W_i)$$

We can see that with limiting the L2 norm of the weights, we multiplicatively shrink all the weights, right before the weight update step.

L2 Regularization

What is the effect for the overall training

- If a weight w_s has little or no impact on the result, the continuous multiplication with $1 - \alpha$ pushes $w_s \rightarrow 0$.
- If a weight w_l has a large effect on the result, the gradient is accordingly large and the w_l stays more or less the same.
- Each value has an influence on the L2 norm. Thus, if unimportant weights disappear, the modified error $\hat{E}$ is smaller.

L2 Regularization

- Cross-entropy training with Softmax output layer can never fully learn the the output

$$\text{softmax}(o_i) = \frac{\exp(o_i)}{\sum_k \exp(o_k)}$$

- The exponentiated output of each node is always strictly positive, hence the network can never produce clean 0 or 1 values
- continued learning can push the weights to be larger and larger, L2 regularization helps here

L1 Regularization

Other weight regularizations

- We have seen L2 regularization, using the (square of) L2 norm

$$\|\cdot\|_2^2 = w_1^2 + w_2^2 + \dots$$

- But we can also use the L1 Norm (not as frequently used)

$$\|\cdot\|_1 = |w_1| + |w_2| + \dots$$

- Thus, the error function is now

$$\hat{E}_1(X, Y, W) = E(X, Y, W) + \alpha \|W\|_1$$

- The gradient is now

$$\nabla \hat{E}_1(X, Y, W) = \nabla E(X, Y, W) + \alpha \cdot \text{sign}(W)$$

- where sign(x) is $+1$ for values larger than 0 and -1 for values smaller than 0

L1 Regularization vs. L2 Regularization

- We can see that the shrinkage does not scale with the magnitude of the weight. L2 weight update

$$w_i \leftarrow (1 - \eta\alpha)w_i - \eta\frac{\partial}{\partial w_i}E$$

- And L1 weight update

$$w_i \leftarrow \begin{cases} w_i - \eta\left(\frac{\partial}{\partial w_i}E + \alpha\right) & w_i > 0 \\ \\ w_i - \eta\left(\frac{\partial}{\partial w_i}E - \alpha\right) & w_i < 0 \end{cases}$$

- L1 regularization results in weight matrices that are sparse, i.e. as many 0 as possible, whereas L2 regularization just reduces the magnitude of the weights

Dataset Augmentation

We can add noise to the dataset.
Instead of using always the same training data

$$(x, y)$$

we add Gaussian noise

$$(x + \mathcal{N}(\sigma, \mu), y)$$

where $\mathcal{N}(\sigma, \mu)$ is a random value according to the Gaussian function with mean σ and variance μ.

Dataset Augmentation

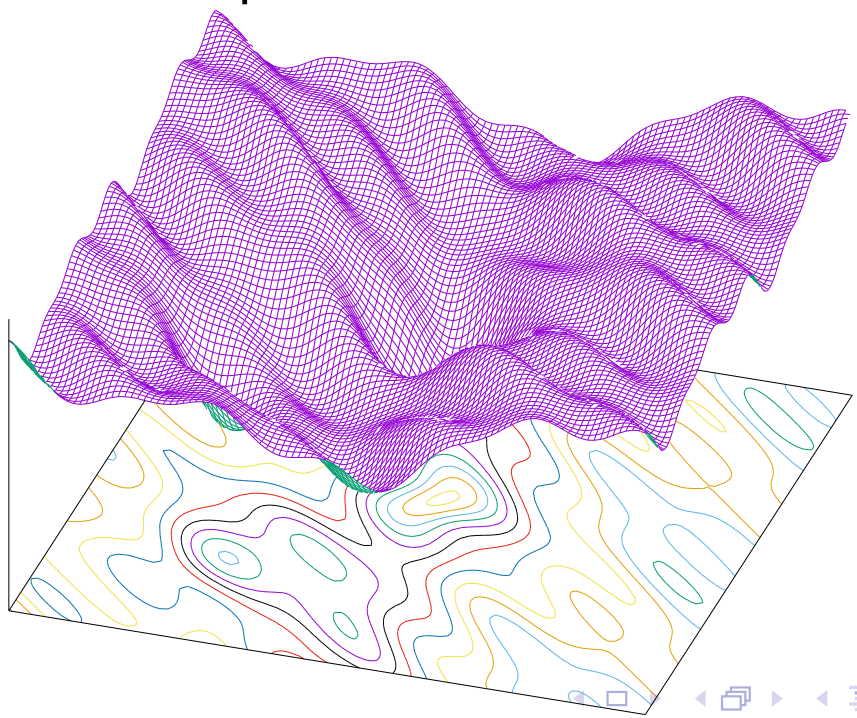
Let the output of the NN of input x and noise be

$$\hat{y}(x)$$

For small learning rates η, this is equivalent to having a regularization term

$$\eta \mathrm{E}\left[\nabla W \cdot \hat{y}(x)\right]$$

where $\mathrm{E}[\cdot]$ is the expectation value. Thus, the learning algorithm prefers flat minima where perturbations do not have a large impact.



Averaging

We can reduce the sensitivity of overtraining by using several, different neural networks and average the output.

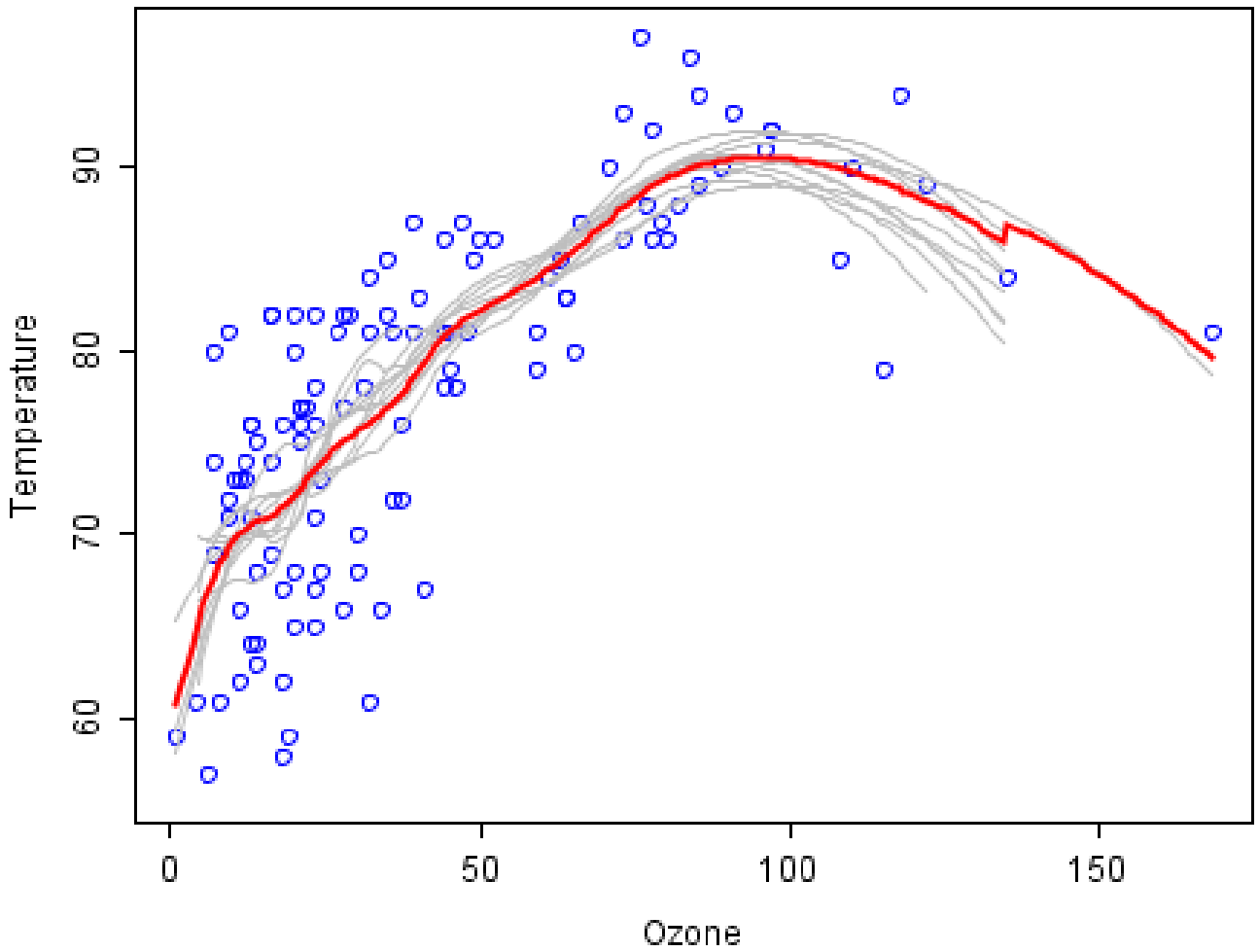
- Use different architectures
- Use different subsets of the training set (aka. Bootstrap aggregating, or Bagging)

Bootstrap aggregating

Bagging helps to create smooth models.
Not all models will make the same mistake

- Chose a random subset of the training data
- Train a model
- Repeat
- Finally, average all models

Example 1



(Wikipedia)

- 100 models, each with 100 randomly chosen data points
- Grey lines: The first 10 models
- Red line: average

Dropout

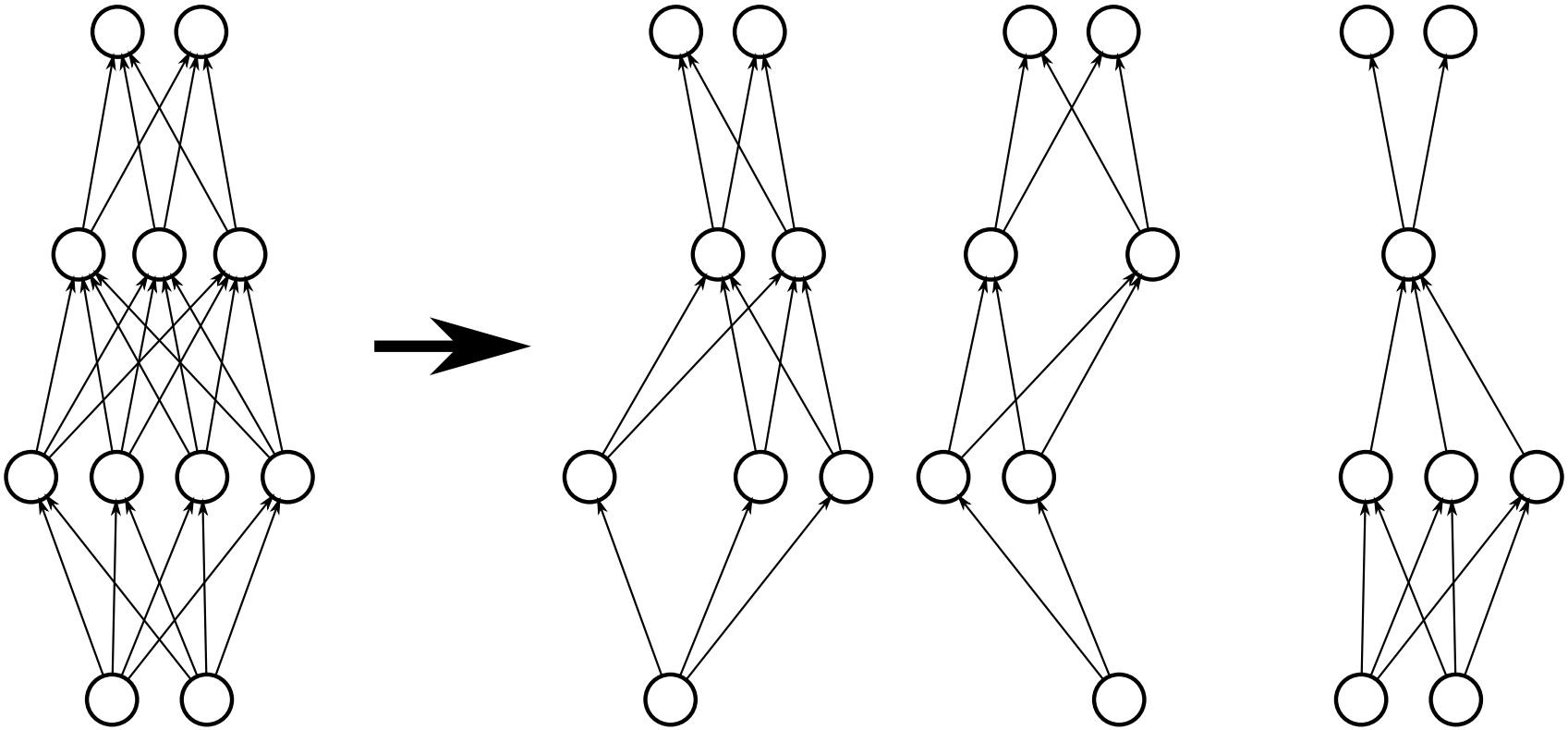
- Bagging has a long history and is not constrained to deep learning/neural network.
- A more recent technique (introduced 2012) is Dropout, i.e. randomly blocking in the network.
- We can think of dropout as an ensemble of neural networks with tied parameters that can be generated from a larger neural network.

Dropout

Algorithm:

1. Set probability of keeping a node p_{keep}
2. For all mini-batches in the training set:
3. For all non-output nodes n:
4. Block node n with probability $1 - p_{\text{keep}}$
5. Do forward pass, backward pass, weight update
6. Unblock nodes

Dropout



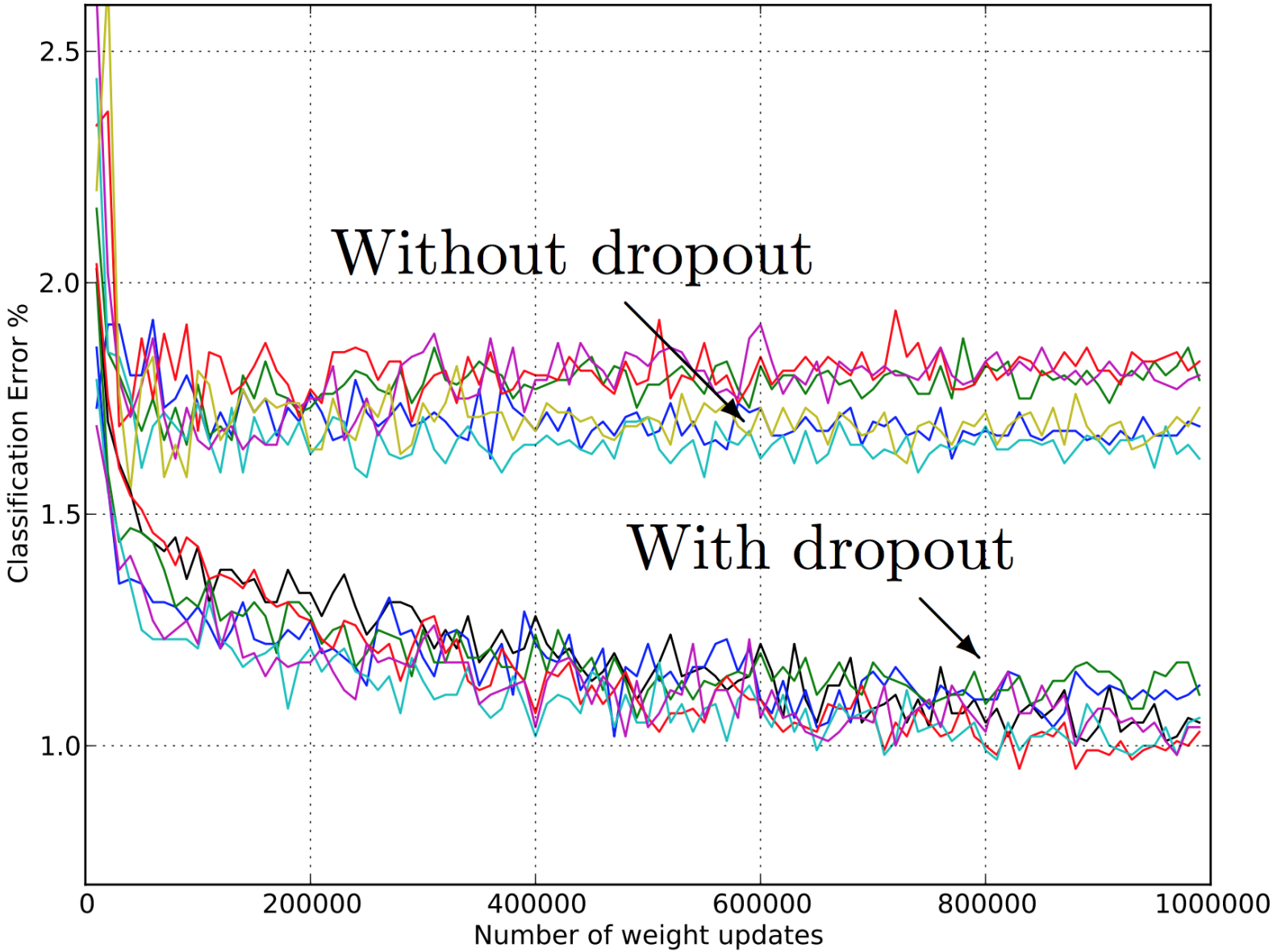
input NN

Ensemble, generated by randomly deleting non-ouput nodes

Bagging vs. Dropout

- In bagging, we chose a number of networks, each trained independently
- In dropout, we create on-the-fly a new network by deleting random nodes. This corresponds to an exponential number of networks, yet with tied weights
  - In fact, not all networks are seen during training
- To make a prediction in bagging, we average the output of all models
- To make a prediction with dropout, we
  - Add all nodes (and weights) to the network
  - Adjust the weights. I.e. weight leaving node n_i are multiplied with p_{keep}, the probability that the node was not deleted.

Dropout Example



(Srivastava, N. et al., "Dropout: A Simple Way to Prevent Neural