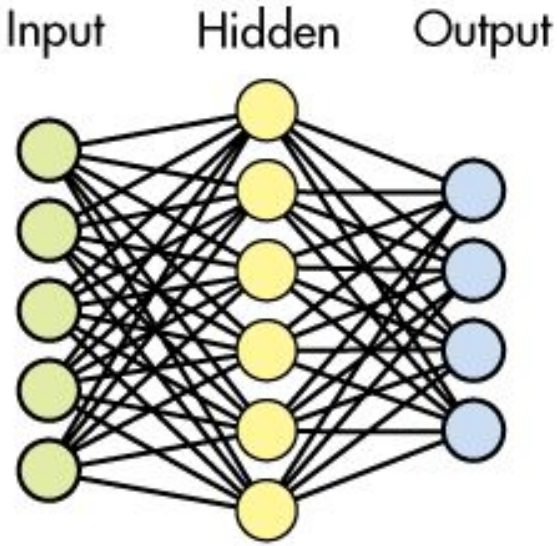
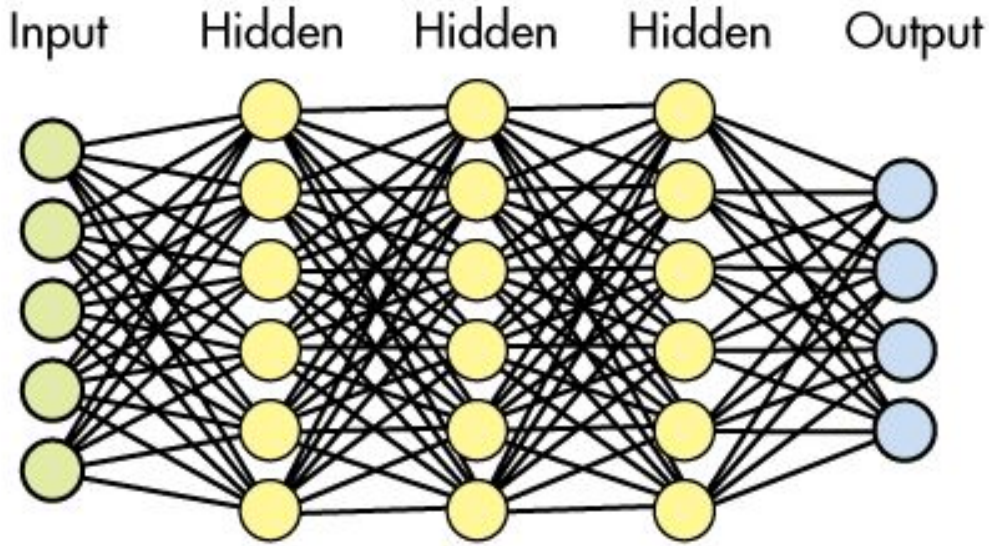
Neural Networks

Ayush Thada

Neural network



Deep neural network



Content

- Is NN generalization of Perceptron?
- Parameters vs Hyperparameters
- Expressiveness of Neural Network
  - Universal Approximation theorem
- Back Propagation
- Overfitting Prevention
  - Dropout
  - L1 & L2 Regularization
- Other Resources

Is NN generalization of Perceptron?

- The perceptron convergence procedure works by ensuring that every time the weights change, they get closer to every “generously feasible” set of weights.
- This type of guarantee cannot be extended to more complex networks in which the average of two good solutions may be a bad solution.
- So “multi-layer” neural networks do not use the perceptron learning procedure.
- They should never have been called multi-layer perceptrons.
- Instead of showing the weights get closer to a good set of weights, show that the actual output values get closer the target values.
- This can be true even for non-convex problems in which there are many quite different sets of weights that work well and averaging two good sets of weights may give a bad set of weights. It is not true for perceptron learning.
- The simplest example is a linear neuron with a squared error measure.

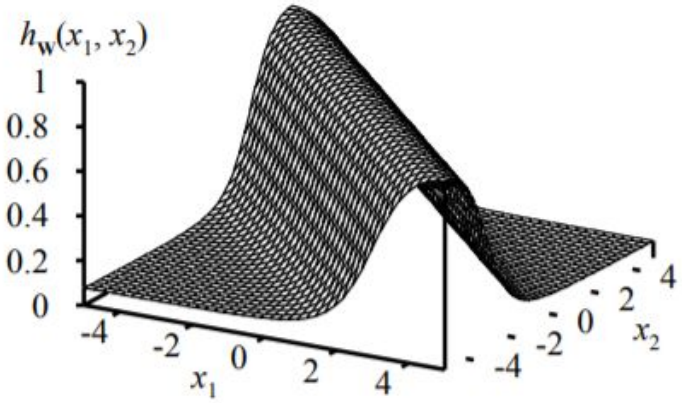
Parameters vs Hyperparameters

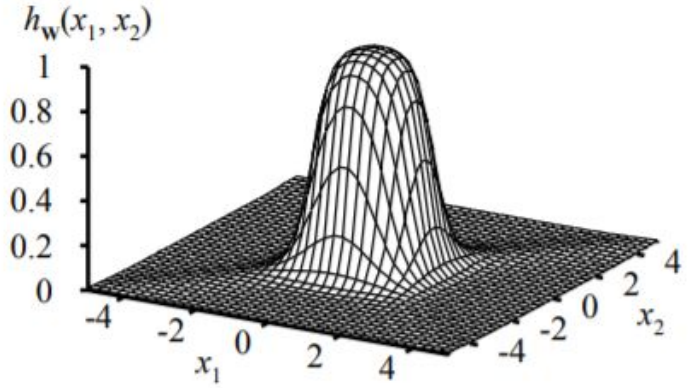
- A model **hyperparameter** is a configuration that is external to the model and whose value cannot be estimated from data.
    - They are often used in processes to help **estimate model parameters**.
    - They are often specified by the practitioner.
    - They can often be **set using heuristics**.
    - They are often tuned for a given predictive modeling problem.
- We cannot know the best value for a model hyperparameter on a given problem. We may use rules of thumb, copy values used on other problems, or search for the best value by trial and error.
- When a machine learning algorithm is tuned for a specific problem, such as when you are using a **grid search or a random searc**h, then you are tuning the **hyperparameters** of the model or order to discover the parameters of the model that result in the most skillful predictions.

- A model **parameter** is a configuration variable that is internal to the model and whose value can be estimated from data.
    - They are required by the model when making predictions.
    - They values define the skill of the model on your problem.
    - They are **estimated** or **learned** from data.
    - They are often not set manually by the practitioner.
    - They are often saved as part of the learned model.
- Parameters are key to machine learning algorithms. They are the part of the model that is learned from historical training data.
- In classical machine learning literature, we may think of the model as the hypothesis and the parameters as the tailoring of the hypothesis to a specific set of data.

Expressiveness of Neural Network

- All continuous functions w/ 2 layers, all functions w/ 3 layers.





- Combine two opposite-facing threshold functions to make a ridge.
- Combine two perpendicular ridges to make a bump
- Add bumps of various sizes and locations to fit any surface
- Proof requires exponentially many hidden units.

- **<u>Universal Approximation theorem:</u>**
  - The universal approximation theorem states that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbf{R}^n$, under mild assumptions on the activation function.
  - The theorem thus states that simple neural networks can represent a wide variety of interesting functions when given **appropriate parameters**; however, it does not touch upon the **algorithmic learnability** of those parameters.
  - Kurt Hornik showed in 1991, that it is not the specific choice of the activation function, but rather the multilayer feedforward architecture itself which gives neural networks the potential of being universal approximators.
  - The output units are always assumed to be linear.

- **Proof of UAT**

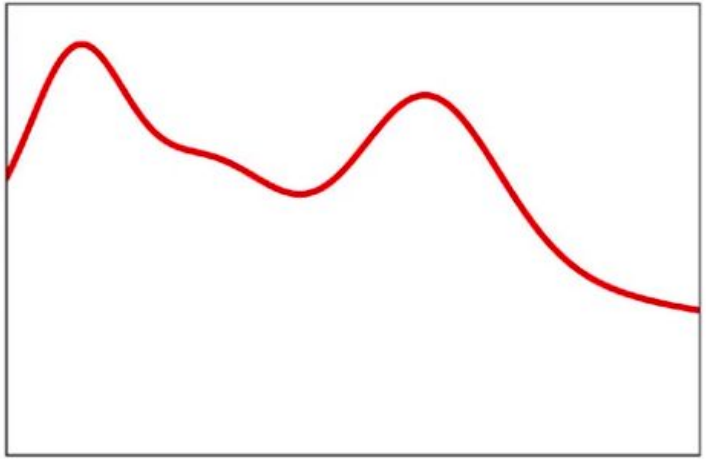


Fig: Original Function

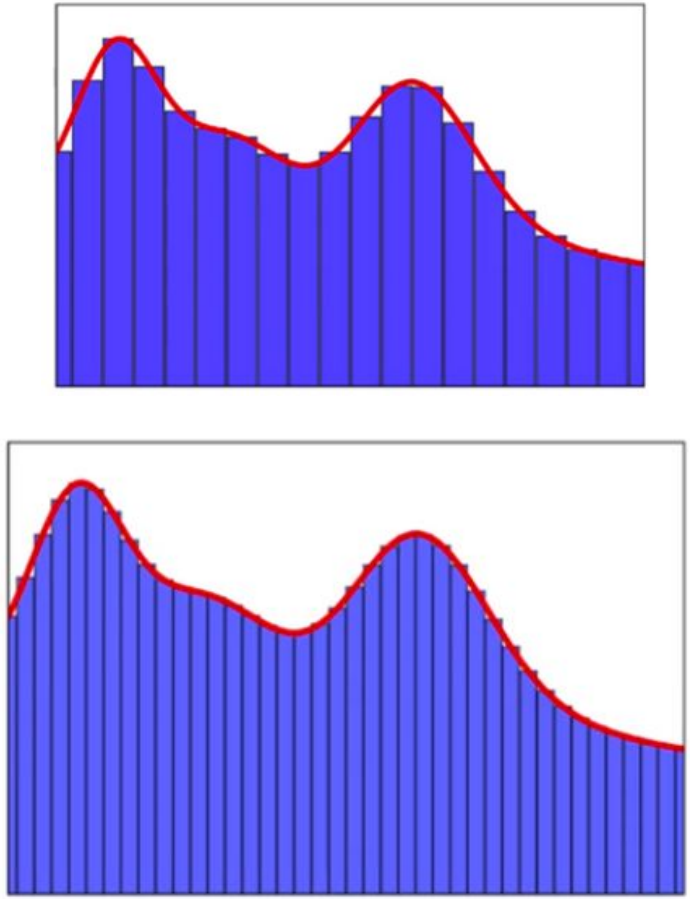
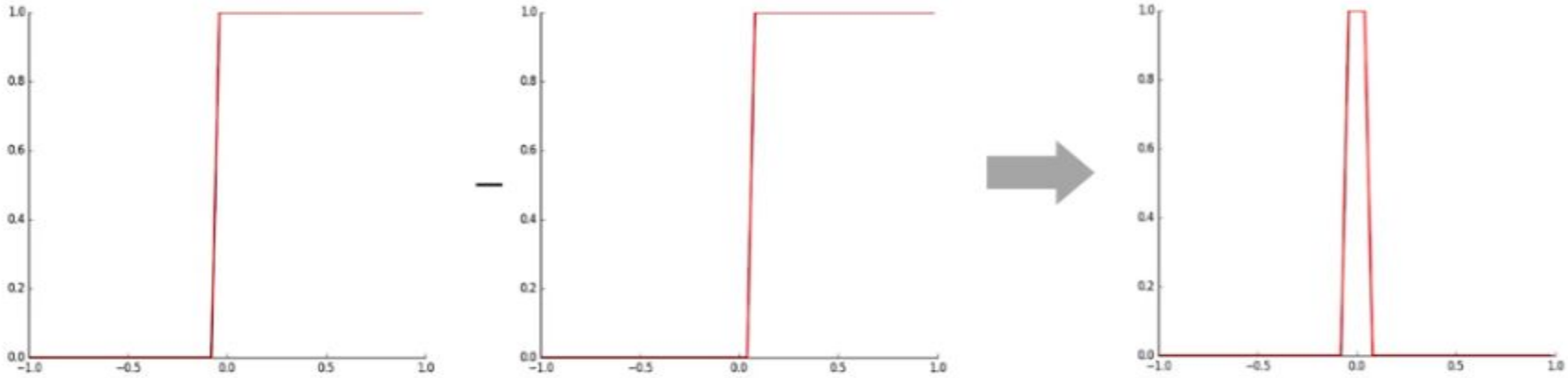
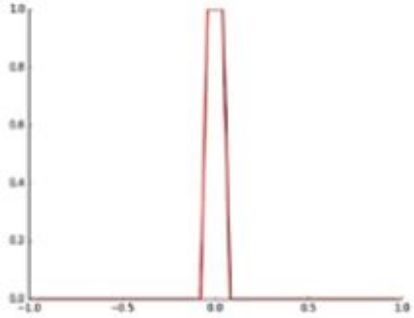


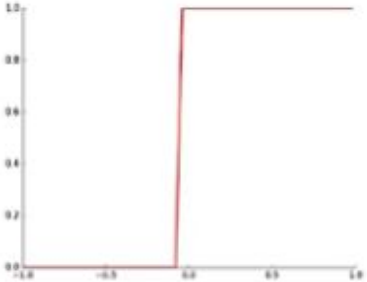
Fig: Combination of simple Function

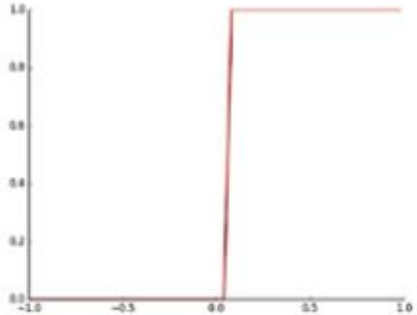
- Let’s take two sigmoid functions having a very steep slope and notice that they have a different place at which they peak.
- The left sigmoid peaks just before zero and right sigmoid peaks just after zero.
- Subtract these two functions the net effect is going to be a tower (rectangular output).

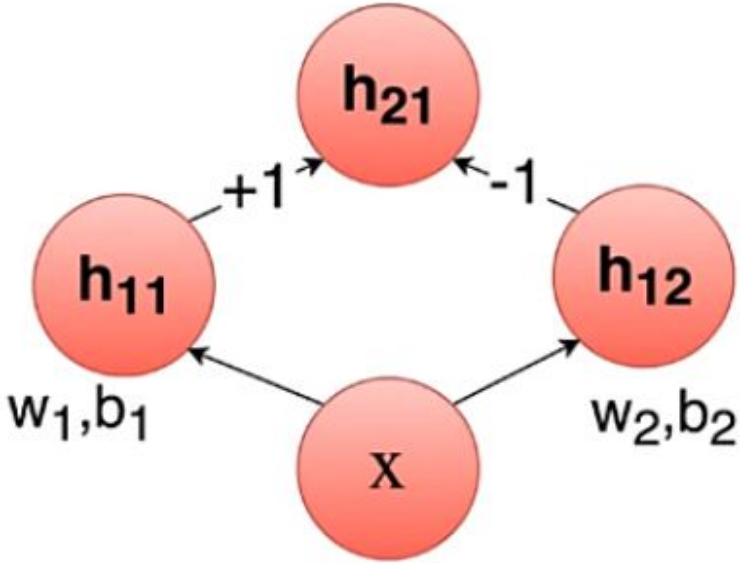


- If we can get the series of these towers, then we can approximate any true function between input and output.



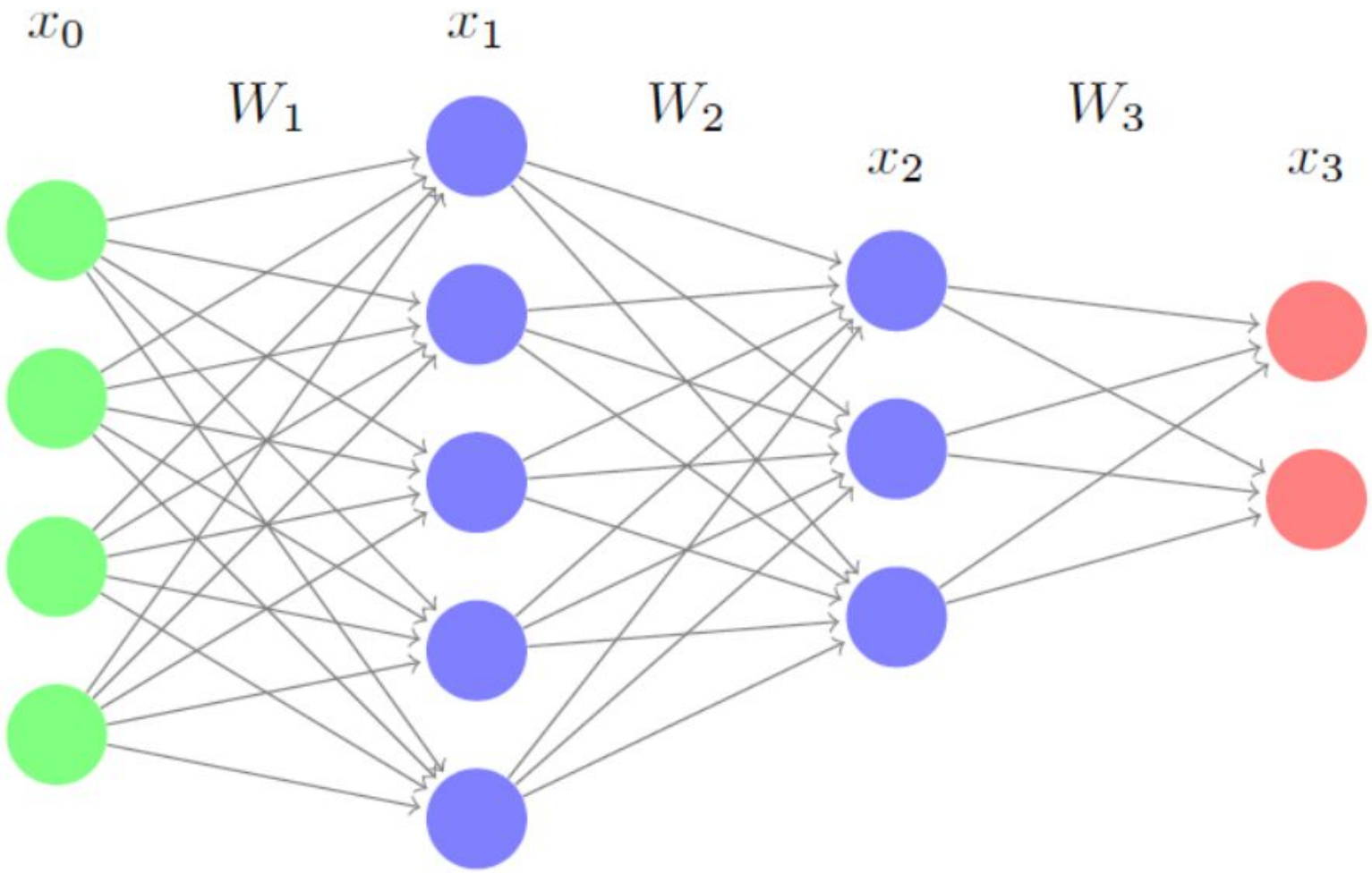





h21
+1
-1
h11
h12
w1,b1
w2,b2
x

Back Propagation

- Backpropagation is an algorithm used to train neural networks, used along with an optimization routine such as gradient descent. It exploits the chain rule of derivation.
- Gradient descent requires access to the gradient of the loss function with respect to all the weights in the network to perform a weight update, in order to minimize the loss function.
- Backpropagation computes these gradients in a systematic way.
- Backpropagation along with Gradient descent is arguably the single most important algorithm for training Deep Neural Networks and could be said to be the driving force behind the recent emergence of Deep Learning.
- The main feature of backpropagation is its iterative, recursive and efficient method for calculating the weights updates to improve the network until it is able to perform the task for which it is being trained. It is closely related to the Gauss–Newton algorithm.


x_0
x_1
W_1
W_2
W_3
x_2
x_3

- The forward propagation equations are as follows:

$$\text{Input} = x_0$$
$$\text{Hidden Layer1 output} = x_1 = f_1(W_1 x_0)$$
$$\text{Hidden Layer2 output} = x_2 = f_2(W_2 x_1)$$
$$\text{Output} = x_3 = f_3(W_3 x_2)$$

- Stochastic update loss function:

$$E = \frac{1}{2}\|z - t\|_2^2$$

- Batch update loss function:

$$E = \frac{1}{2}\sum_{i \in Batch} \|z_i - t_i\|_2^2$$

- Let us look at the loss function from a different perspective. Given an input x_0, output x_3 is determined by W_1, W_2 and W_3. So the only tunable parameters in E are W_1, W_2 and W_3. To reduce the value of the error function, we have to change these weights in the negative direction of the gradient of the loss function with respect to these weights.

$$w = w - \alpha_w \frac{\partial E}{\partial w} \qquad \text{for all the weights } w$$

- Here αw is a scalar for this particular weight, called the learning rate. Its value is decided by the optimization technique used.
- Backpropagation equations can be derived by repeatedly applying the chain rule.

- $$\frac{\partial E}{\partial W_3} = (x_3 - t)\frac{\partial x_3}{\partial W_3}$$
$$= [(x_3 - t) \circ f_3'(W_3 x_2)]\frac{\partial W_3 x_2}{\partial W_3}$$
$$= [(x_3 - t) \circ f_3'(W_3 x_2)]x_2^T$$
- $$\text{Let } \delta_3 = (x_3 - t) \circ f_3'(W_3 x_2)$$
- $$\frac{\partial E}{\partial W_3} = \delta_3 x_2^T$$

- $$\begin{aligned}\frac{\partial E}{\partial W_2} &= (x_3 - t)\frac{\partial x_3}{\partial W_2} \\ &= [(x_3 - t) \circ f_3'(W_3 x_2)]\frac{\partial (W_3 x_2)}{\partial W_2} \\ &= \delta_3 \frac{\partial (W_3 x_2)}{\partial W_2} \\ &= W_3^T \delta_3 \frac{\partial x_2}{\partial W_2} \\ &= [W_3^T \delta_3 \circ f_2'(W_2 x_1)]\frac{\partial W_2 x_1}{\partial W_2} \\ &= \delta_2 x_1^T\end{aligned}$$

- Similarly

$$\frac{\partial E}{\partial W_1} = [W_2^T \delta_2 \circ f_1'(W_1 x_0)] x_0^T$$
$$= \delta_1 x_0^T$$

- We can observe a recursive pattern emerging in the backpropagation equations. The Forward and Backward passes can be summarized on next slide.
- The neural network has L layers. x_0 is the input vector, x_L is the output vector and t is the truth vector. The weight matrices are $W_1, W_2, .., W_L$ and activation functions are $f_1, f_2, .., f_L$.

- **Forward Pass:**

$$x_i = f_i(W_i x_{i-1})$$
$$E = \|x_L - t\|_2^2$$

- **Backward Pass:**

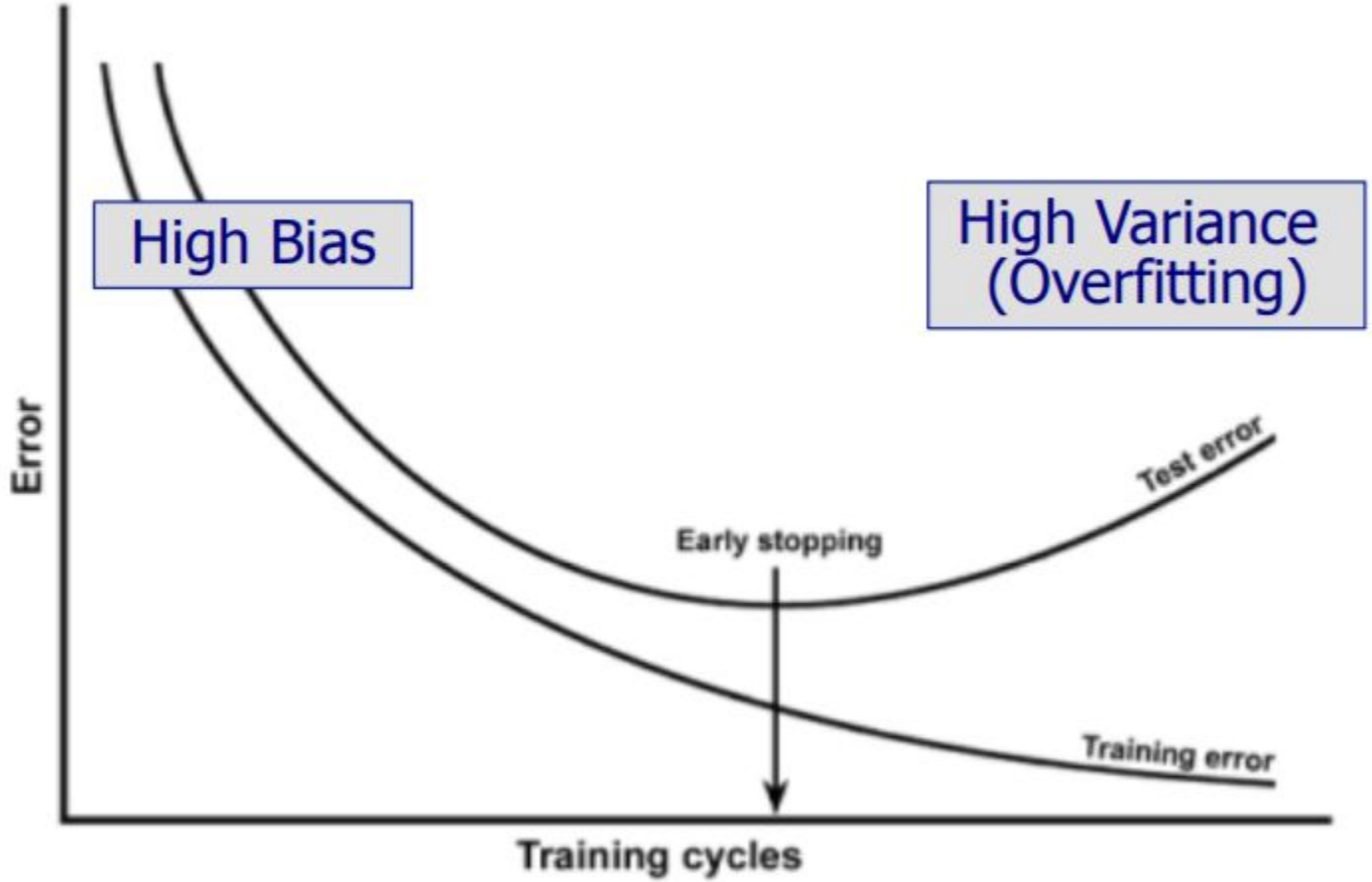
$$\delta_L = (x_L - t) \circ f_L'(W_L x_{L-1})$$
$$\delta_i = W_{i+1}^T \delta_{i+1} \circ f_i'(W_i x_{i-1})$$

- **Weight Update:**

$$\frac{\partial E}{\partial W_i} = \delta_i x_{i-1}^T$$
$$W_i = W_i - \alpha_{W_i} \circ \frac{\partial E}{\partial W_i}$$

Overfitting Prevention

- The training data contains information about the regularities in the mapping from input to output. But it also contains sampling error.
- There will be accidental regularities just because of the particular training cases that were chosen.
- When we fit the model, it cannot tell which regularities are real and which are caused by sampling error.
- So it fits both kinds of regularity. If the model is very flexible it can model the sampling error really well.
- This means the model will not generalize well to unseen data .


High Bias
High Variance
(Overfitting)
Error
Test error
Early stopping
Training error
Training cycles

Prevent Overfitting:

- *Approach 1*: Get more data!
  - Almost always the best bet if you have enough compute power to train on more data.
- *Approach 2*: Use a model that has the right capacity:
  - enough to fit the true regularities.
  - not enough to also fit spurious regularities (if they are weaker).
- *Approach 3*: Average many different models.
  - Use models with different forms.
  - Or train the model on different subsets of the training data (this is called “bagging”).
- *Approach 4*: Regularization
  - L1 & L2 Regularization
  - Dropout

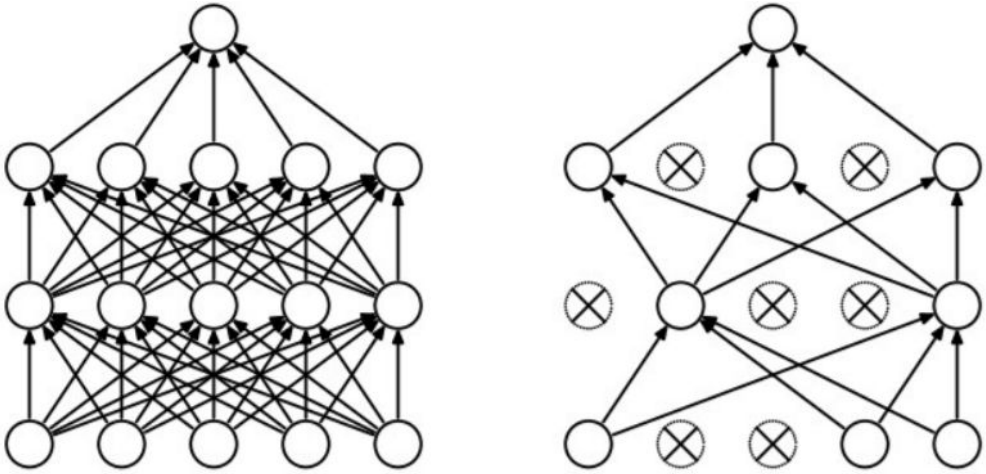
Limiting Capacity of a Neural Net

- The capacity can be controlled in many ways:
  - *Architecture*: Limit the number of hidden layers and the number of units per layer.
  - *Early stopping*: Start with small weights and stop the learning before it overfits.
  - *Weight-decay*: Penalize large weights using penalties or constraints on their squared values (L2 penalty) or absolute values (L1 penalty). This method is a one way of regularizing the weights by putting constraint on model's learning objective.
  - *Noise*: Add noise to the weights or the activities.
- Typically, a combination of several of these methods is used.

Dropout

- An efficient way to average many large neural nets.
- Consider a neural net with one hidden layer.
- Procedure:
    - Each time we present a training example, we randomly omit each hidden unit with probability 0.5.
    - So we are randomly sampling from 2^H different architectures. **[All architectures share weights.]**
- We sample from 2^H models. So only a few of the models ever get trained, and they only get one training example. [**This is as extreme as bagging can get.**]
- The sharing of the weights means that every model is very strongly regularized. [**It's a much better regularizer than L2 or L1 penalties that pull the weights towards zero.**]

- In test time:
  - We could sample many different architectures and take the geometric mean of their output distributions.
  - It better to use all of the hidden units, but to halve their outgoing weights. This exactly computes the geometric mean of the predictions of all 2^H models.



(a) Standard Neural Net

(b) After applying dropout.

L1 & L2 Regularization:

- **L1 regularization** adds an L1 penalty equal to the absolute value of the magnitude of coefficients. In other words, it limits the size of the coefficients. L1 can yield sparse models (i.e. models with few coefficients); Some coefficients can become zero and eliminated.
    - Lasso regression uses this method.
- **L2 regularization** adds an L2 penalty equal to the square of the magnitude of coefficients. L2 will not yield sparse models and all coefficients are shrunk by the same factor (**none are eliminated**).
    - Ridge regression and SVMs use this method.
- **Elastic nets** combine L1 & L2 methods, but do add a hyperparameter

Extra Reading

Backpropagation:

- https://theclevermachine.wordpress.com/2014/09/06/derivation-error-backpropagation-gradient-descent-for-neural-networks/
- https://sudeepraja.github.io/Neural/
- https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf [**Original Paper**]

Gradient descent optimization algorithms

- http://ruder.io/optimizing-gradient-descent/

Universal Approximation theorem

- http://mcneela.github.io/machine_learning/2017/03/21/Universal-Approximation-Theorem.html

Expressive Power of Neural Network

- https://arxiv.org/pdf/1709.02540.pdf
- https://www.cl.cam.ac.uk/~ey204/teaching/ACS/R244_2018_2019/papers/raghu_PMLR_2017.pdf

Text Books

- Chapter 5, **Pattern Recognition and Machine Learning** [Christopher M. Bishop]
- Chapter 4, **Machine Learning** [Tom Mitchell]

Regularization

- http://neuralnetworksanddeeplearning.com/chap3.html [Michael Nielsen]

Questions