Regularization + Optimization

Avoidig overfitting

Empirical Risk and surrogate Loss Functions

- Ideally we want to minimize with respect to the true data generating mechanism however it is frequently unknown and a proxy is used
- When minimizing with respect to a known data generating function we minimize the **Risk**
- When the data generating function is unknown we minimize with respect to the **Empirical Risk Minimization**

$$J(\theta) = E_{(x,y)\sim\hat{p}_{data}}[L(f(x;\theta),y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta),y^{(i)})$$

Empirical Risk Function

Surrogate loss functions are used as approximates to our true loss functions because:

- They have efficiency advantages
- Their derivative does not contain useful information (0-1 loss in classification)
  - An example of a surrogate loss function in classification is cross-entropy rather than accuracy
- Their transforms are usually bijective mappings over the space

$$L(y,\hat{y})_i = \sum_k y_i^{(k)} \log(\hat{y}_i^{(k)}) \quad ;k = 1,\ldots,\text{num classes}$$

Cross-entropy for 1 sample from k=1,..,num classes

Challenges fitting a network

Below we list a couple of challenges with Network optimization

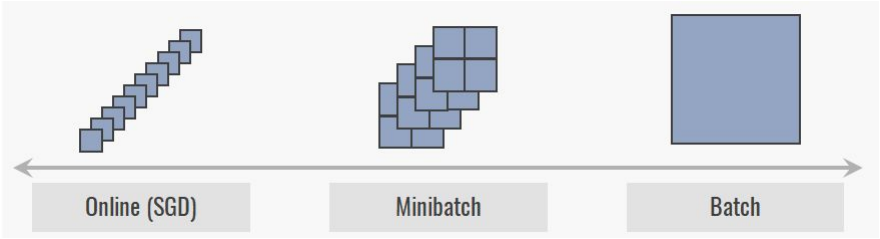
1. Traditional methods may not work as well i.e. L-BFGS
2. Models with high capacity can memorize the training set
3. Second order optimization methods suffer from poor condition numbers and can be computationally expensive
4. NNs are **non-convex functions**
    a. Many local minima
    b. Saddle points
    c. Models are not **identifiable**
    d. Local minima are problematic if they have a high cost in comparison to global minimum
5. Efficiently distributing and parallelizing optimization is challenging

Batch and Stochastic Gradient descent

Batch Gradient Descent
Optimization algorithms that use the entire data set are called **batch** methods.
They are deterministic (batch gradient descent uses the entire training set)

Stochastic gradient descent:

- Use a single example to update parameter (also known as *online* learning)
- Implementation:
  - Randomly samples one element at time
  - Runs through backwards/forwards propagation
  - It then updates the cost function



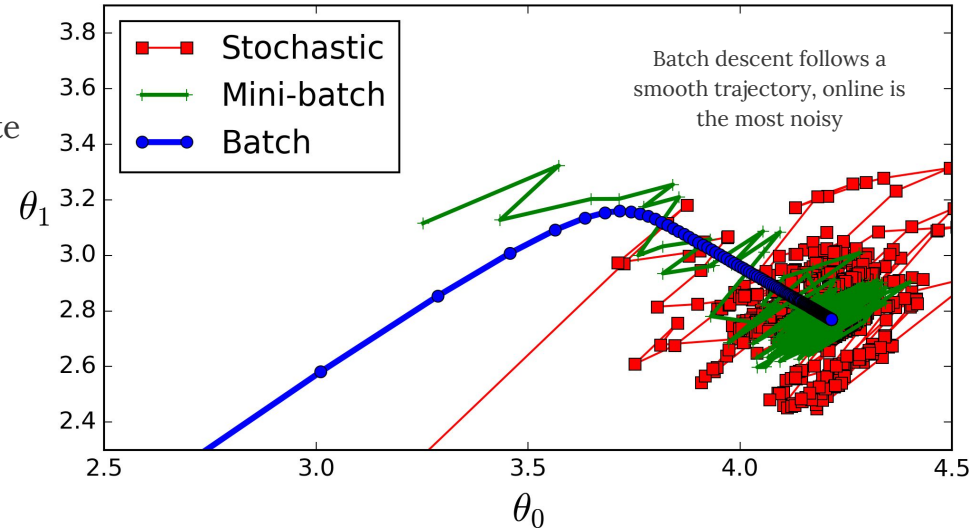
Minibatch Gradient descent

$$\theta_{t+1} = \theta_t - \epsilon(t)\frac{1}{B}\sum_{b=0}^{B-1}\frac{\partial L(\theta, \mathbf{m}_b)}{\partial \theta}$$

In mini-batch, samples are averaged over the partition set B

Mini-batch gradient descent:

- Most parameter learning algorithms fall somewhere in between batch and stochastic and are frequently called **minibatch** methods
- It is now common to refer to them as *stochastic* which is a misnomer
- Implementation:
  - Samples several points at a time and then
  - Applies forwards/backwards prop to update parameters and cost function
- Commonly called *batch gradient descent*
- Synonymous with SGD, although incorrect (people refer to mini-batch as SGD)



Best practices

1. Large batches are more accurate
    - Performance return is is less than linear
    - Bigger batches don't mean you get a linear increase in accuracy
2. Multicore architectures are underutilized with small batches
3. When processing batches in parallel
    - Memory scales with batches
    - This a hardware limitation - (we'll discuss this more in the tool kit section)
4. Some hardware is specialized to training (GPUs) and works better with batch sizes
    - Typical batch sizes follow powers of 2
    - 32-256 are common with 16 sometimes being used for large models
5. Small batches cab offer a regularizing effect
6. Generalization error is minimized with a small batch size (i.e. 1) however should be done with a small learning rate to maintain stability (total runtime will however be much longer)

Momentum learning

Vanilla Momentum

$$\mathbf{v}_{t+1} \leftarrow \alpha \mathbf{v}_t - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)}) \right)$$

$$\theta_{t+1} \leftarrow \theta_t + \mathbf{v}_{t+1}$$

Nesterov Momentum

$$\mathbf{v}_{t+1} \leftarrow \alpha \mathbf{v}_t - \epsilon \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}; \theta + \alpha \mathbf{v}_t), \mathbf{y}^{(i)}) \right)$$

$$\theta_{t+1} \leftarrow \theta_t + \mathbf{v}_{t+1}$$

Adaptive Learning Rate (AdaGrad)

$$\mathbf{r}_0 \leftarrow 0$$

$$\mathbf{g}_t \leftarrow \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)}) \right)$$

$$\mathbf{r}_{t+1} \leftarrow \mathbf{r}_t + \mathbf{g}_t \odot \mathbf{g}_t$$

$$\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot \mathbf{g}_t$$

$$\theta \leftarrow \theta + \Delta\theta$$

Adaptive Moments (Adam)

$$k \leftarrow 0; \mathbf{s}_0 \leftarrow 0; \mathbf{r}_0 \leftarrow 0$$

$$\mathbf{g}_t \leftarrow \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)}) \right)$$

$$\mathbf{s}_{t+1} \leftarrow \rho_1 \mathbf{s}_t + (1 - \rho_1)\mathbf{g}_t \quad ; \quad \mathbf{r}_{t+1} \leftarrow \rho_2 \mathbf{r}_t + (1 - \rho_2)\mathbf{g}_t \odot \mathbf{g}_t$$

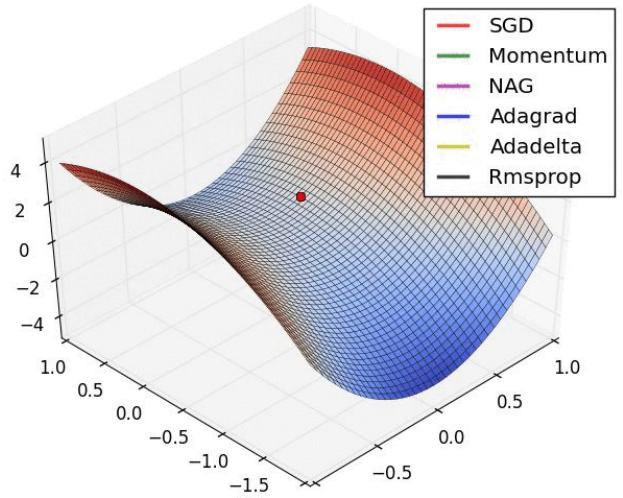
$$k \leftarrow k + 1 \quad \hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^k} \quad \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^k}$$

$$\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot \mathbf{g}_t$$

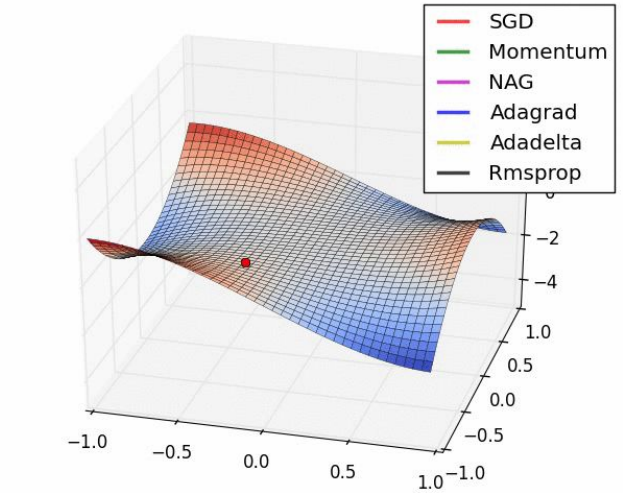
$$\theta \leftarrow \theta + \Delta\theta$$

Which Algorithm to Use?

- No broad concensus
- Schaul et al 2014 suggest adaptive methods (RMSProp and AdaDelta) are robust however do not conclude one method is consistently better



- The most popular methods which yield reasonable results are:
  1. SGD
  2. SGD with momentum
  3. RMSProp
  4. RMSPropr with Momentum
  5. AdaDelta and
  6. Adam

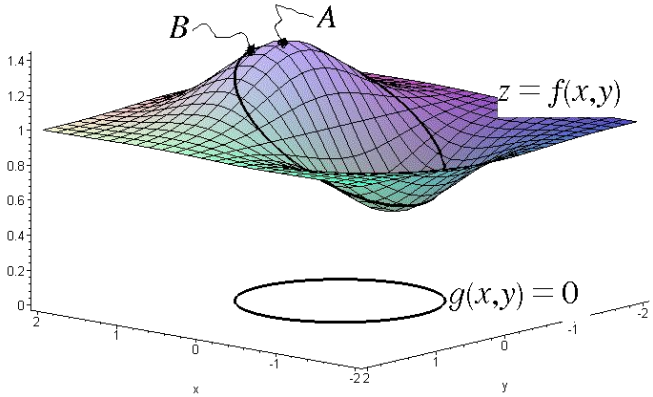


Regularization

"Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."- Goodfellow 5.2.2

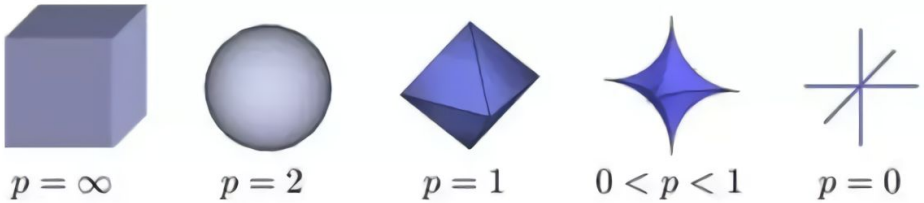
- Common methods
  - Parameter Norm Penalties - L1/L2 regularization
  - Stochastic dropout
  - Noise injection
  - Early stopping
  - Data augmentation
- Other methods (not discussed here):
  - Ensembling (TBD with advanced topics)
  - Explicit optimization using Karush-Kuhn-Tucker conditions
  - Multi-task learning
  - Sparse representations

Norm Penalties

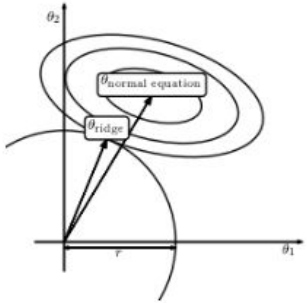


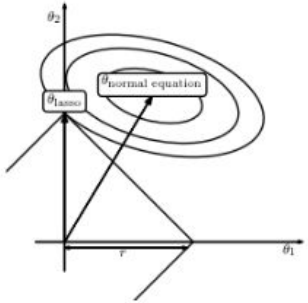
L1/L2 penalties are effectively LaGrange multipliers

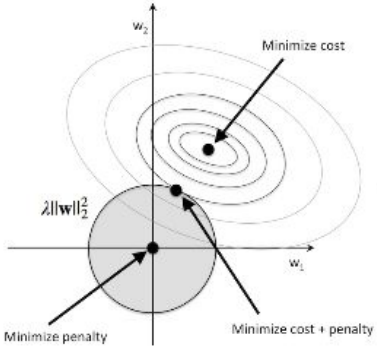
They are a form of constrained optimization

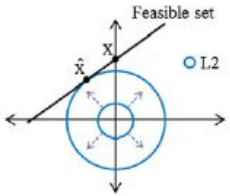


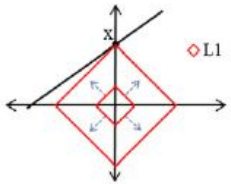
The two most common operation penalties are L1/L2 penalties











L1 penalties for sparsity since the solution set is constrained to the L1 surface

Updating the loss function with penalties

- Norm penalties are added in the cost function:

$$\tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = \mathbf{J}(\theta; \mathbf{X}, \mathbf{y}) + \lambda ||\theta||_{\mathbf{p}}$$

  where $||\theta||_{\mathbf{p}}$ is the p-norm and λ is the penalizing constant and gradient

$$\nabla_\theta \tilde{J}(\theta; \mathbf{X}, \mathbf{y}) = \nabla_\theta \mathbf{J}(\theta; \mathbf{X}, \mathbf{y}) + \nabla_\theta \lambda ||\theta||_{\mathbf{p}}$$

- For L_1 cost function is:

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

- For L_2 the

$$\tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = J(\mathbf{w}; \mathbf{X}, \mathbf{y}) + \lambda \frac{1}{2} ||\mathbf{w}||_2^2$$

- When applying regularizers the L_1 and L_2 gradients are updated as in backprop algorithm as follows:

$$L_1 : \mathbf{w} \leftarrow \mathbf{w} - \epsilon(\nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{X}, \mathbf{y} + \lambda \text{sign}(\mathbf{w}))$$

$$L_2 : \mathbf{w} \leftarrow \mathbf{w} - \epsilon(\nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{X}, \mathbf{y} + \lambda \mathbf{w})$$

Dataset augmentation

- Hand-designed perturbations don't change the content
- Can reduce the generalization error
- Frequently used with image recognition

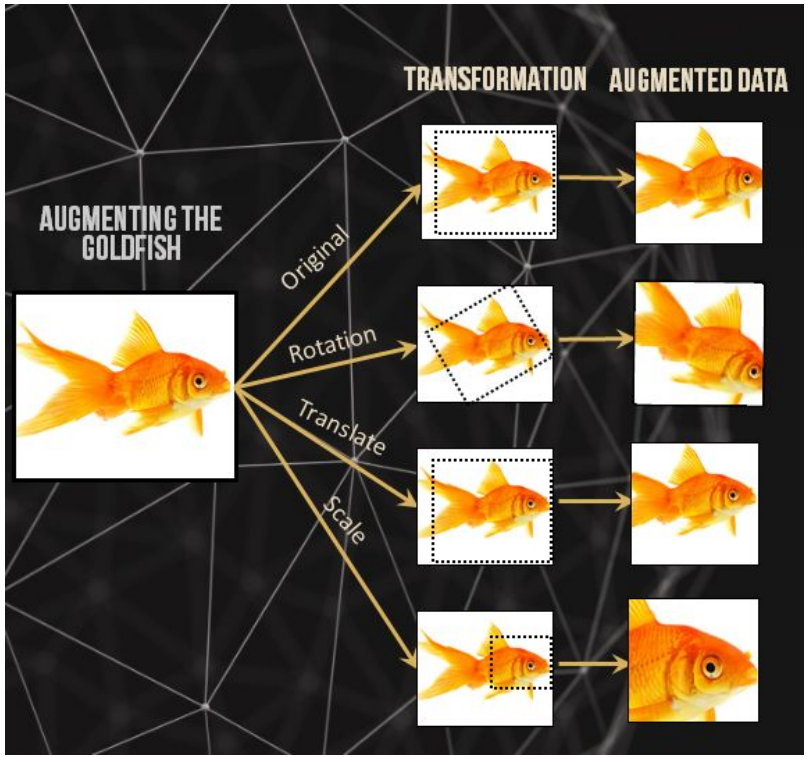
Common transforms include:

1. Rotation
2. Decrease size
3. Scaling
4. Shearing
5. Translation
6. Occlusion
7. Mirroring
8. Lighting shifts

By increasing the number of "exemplars", a model will learn their feature invariance at different resolutions, positions, lighting conditions and symmetries

<u>Caution:</u> certain transformations may result in flipping classes such as 6 to a 9

* Poole et al (2014) showed noise injection acts as a form of regularization



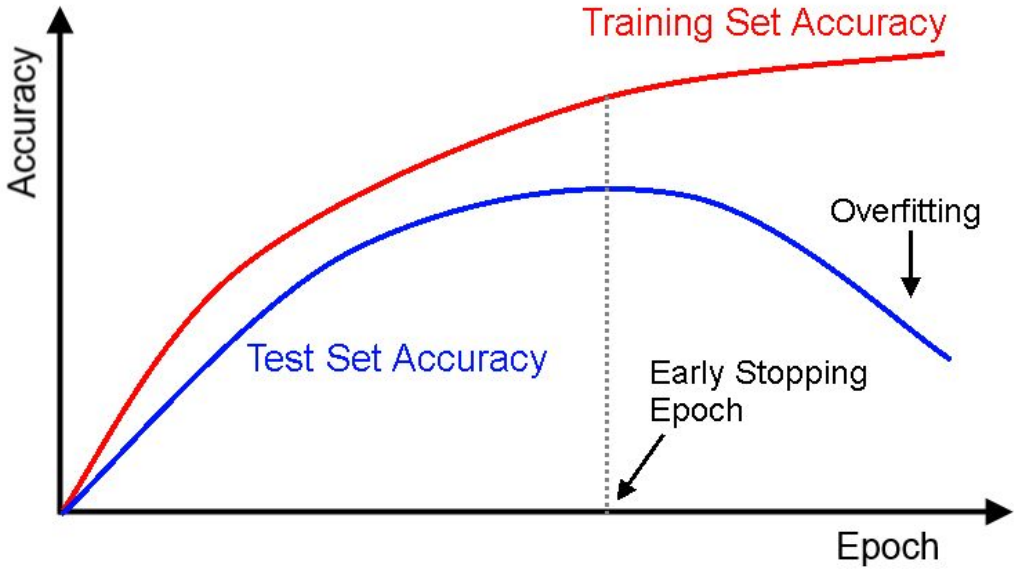
Noise Injection

Noise can be added at the:

- Input:
  - When noise is added at the input it is similar to data augmentation
  - In some cases in may also be interpretted as a penalty norm (Bishop 15a,b)
- Hidden layers:
  - Adding noise to the weights (mainly recurrent neural networks)
  - Interpreted as a stochastic implementation of Bayesian inference over the weights (see notes for calculations)
  - We'll discuss regularization at hidden layers more when discussing drop-out
- Output perturbation:
  - Some data sets have errors in their output values
  - Consider how difficult it may be to differentiate between a 1 and 7 in MNIST
  - Using **label smoothing** regularizes model based on a softmax with kk output values by replacing the hard classification targets 0 and 1 with $\epsilon/(k-1)\epsilon/(k-1)$ and ϵ

Early Stopping

- One of the most effective and simplest forms of regularized
- Look for U shape in training set
- Bishop 1995 argued it has the effect of restricting the procedure to a small volume in the parameter space
- Early stopping is similar to controlling the weight decay
- Under certain conditions early stopping is equivalent to L^2 Regularization



Stochastic dropout

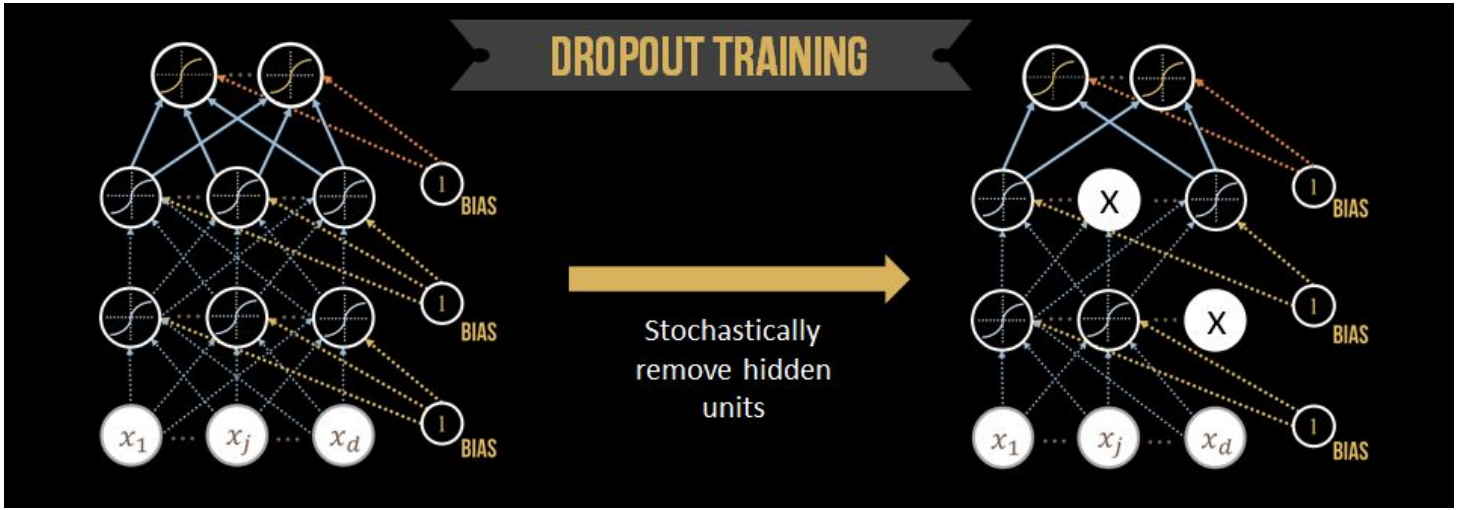
Here we "stochastically" set hidden units to 0 during forward and backward passes

We do this by applying a binary mask

This has the effect of forcing other units to learn patterns rather than memorizing the data

Hidden units cannot co-adapt to other units

Hidden units need to learn features



Importance of Feature Engineering

Deep networks allow us to explore complex hypothesis spaces

However well designed features have additional benefits including:

1. Decreased computational resources
2. Decrease model complexity
3. Less data is required for training

Example: Predicting the time on a clock from an image

In order of complexity of approach (most to least)

- Approach 1: Scan clock and build model to process image
- Approach 2: Get point coordinates for the hands
- Approach 3: Get angle between large and small hands

Accordingly a first approach is to understand and simplify the problem
It greatly decreases model complexity, increases generalizability and subsequently increases accuracy

Representational Capacity

The idea of **representational capacity** is the degree to which a neural network can represent a function f(x)

Representational capacity is important in so far as it helps us determine the appropriate architecture for our neural networks

A well known theorem states that any 2-layer neural network with an infinite number of hidden units can approximate any function f(x)f(x) with arbitrary precision.

- Neural networks, even while shallow, are therefore known as **universal approximators**