Deep Learning

Lecture 4: Designing models to generalise

Chris G. Willcocks

Durham University

Lecture Overview

1 Generalisation theory

- universal approximation theorem
- empirical risk minimisation
- no free lunch theorem and Occam's razor
- increasing capacity through double descent

2 Function design

- natural signals
- convolutional neural networks
- recurrent neural networks
- deep residual networks

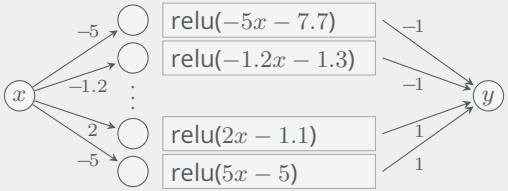
3 Regularisation

- early stopping and annealing
- the effect on model capacity
- data augmentation
- dropout and Tikhonov regularization
- ensembles

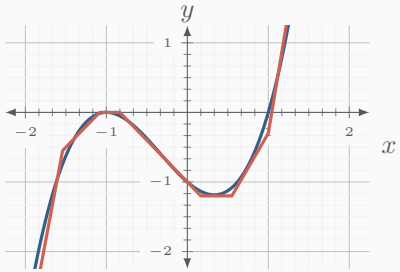
Theorem: universal function approximator

Arbitrary width

A network with a single hidden layer, containing a finite number of neurons, can approximate any continuous function under mild assumptions.



original function $f(x)=x^3+x^2-x-1$



$$f(x)=-r(-5x-7.7)-r(-1.2x-1.3)-r(1.2x+1)+r(1.2x-0.2)+r(2x-1.1)+r(5x-5)$$

Example ReLU weights by Brendan Fortuner

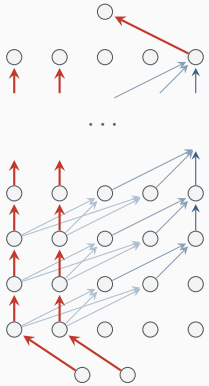
Theorem: universal function approximator

Arbitrary depth (fixed width)
Does the theorem still hold for fixed width and arbitrary depth? **Yes!**

For a network of n inputs and m outputs, [1] show universal approximaton holds true for:

- width $n + m + 2$ for almost any activation function
- width $n + m + 1$ for most activation functions

Short YouTube visual proof



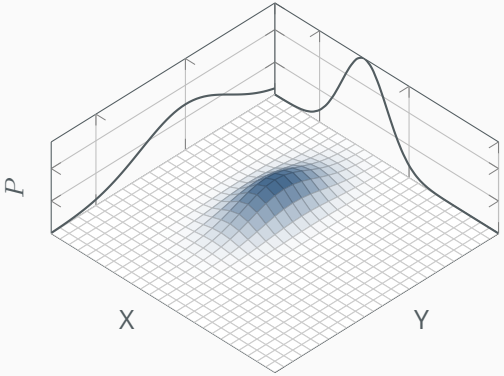
Learning the data distribution

So what is it we want exactly?

- $P(Y|X)$ discriminative model (classification)
- $P(X|Y)$ conditional generative model
- $P(X,Y)$ generative model

We want to learn the probability density function of our data (natures distribution)

The data distribution $P(X,Y)$



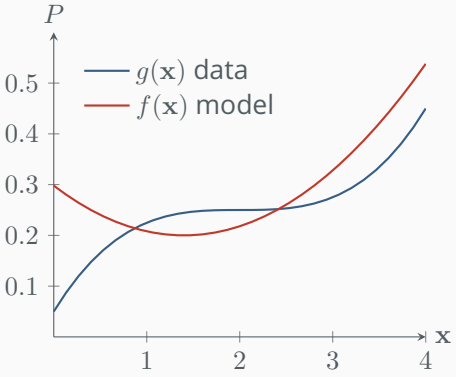
Finding a model

Lets illustrate this in 1D, but try to imagine it in ND. Given the target probability distribution of our data

$$g(\mathbf{x}) = P(X, Y)$$

we want to find (design) a model $f(\mathbf{x}, \theta)$ with parameters θ and optimise θ such that

$$f(\mathbf{x}, \theta) = g(\mathbf{x})$$



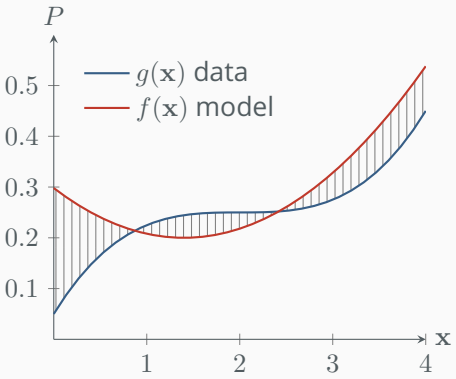
Optimising the model

The total error is therefore the entire area. Modifying the parameters θ will cause the area to change, where we want to find

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \int \mathcal{L}(f(\mathbf{x}, \theta), g(\mathbf{x}))\mathrm{d}\mathbf{x}.$$

where $\mathcal{L}$ is a 'loss function', e.g. a 0-1 loss function $\mathcal{L}(\hat{x}, x) = \mathbb{I}(\hat{x} \neq x)$ or a mean squared error loss.

The solution is a function f that has the capacity to exactly represent $g(\mathbf{x})$



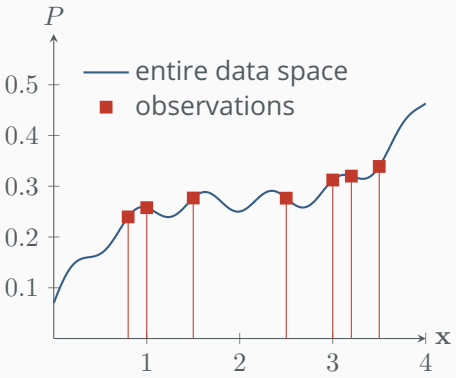
The generalisation problem

However there's a big problem:

In practice, we can't observe all of $g(\mathbf{x})$

This means:

1. We don't know how smooth the function is between the observations
2. Noise can be difficult to interpret
3. Optimisation is highly sensitive to the sampling process



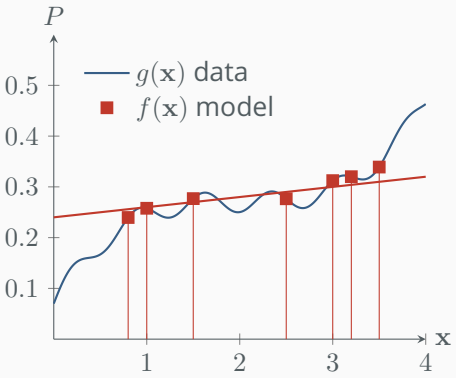
Definition: empirical risk

The expected error (or risk) is the average error over the entire space, which we can't compute:

$$\mathbb{E}[\mathcal{L}(f(\mathbf{x},\theta),g(\mathbf{x}))] = \int \mathcal{L}(f(\mathbf{x},\theta),g(\mathbf{x}))\mathrm{d}\mathbf{x}.$$

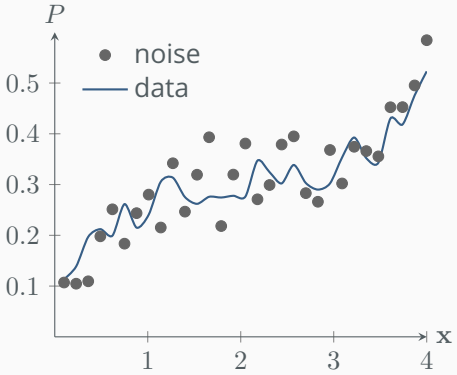
Therefore we minimise the **empirical estimate** of the risk as an average over the samples:

$$\mathbb{E}[\mathcal{L}(f(\mathbf{x},\theta),g(\mathbf{x}))] \approx \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(f(\mathbf{x},\theta),g(\mathbf{x})).$$



The role of noise

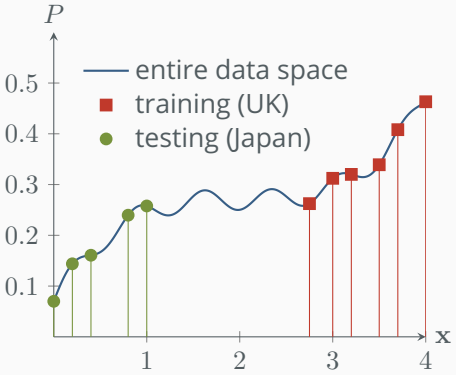
Given that the shape of the distribution outside of the observations is unknown, it is easy to overfit to noise.



Out-of-distribution data

Usually the training dataset collection process draws samples from the data space in a way that is not **independent and identically distributed** (abbreviated i.i.d.) to the expected testing (operational) conditions of the model.

Sampling data in a way that is representative of the task/testing/operational distribution is extraordinarily difficult to do properly. It is often a worthwhile investment though!

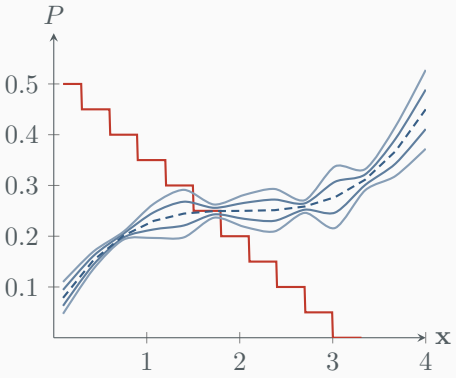


Definition: no free lunch theorem

We can use methods such as cross validation to empirically choose the best method for our particular problem. However, there is no universally best model — this is sometimes called the no free lunch theorem [2].

Definition: Occam's razor

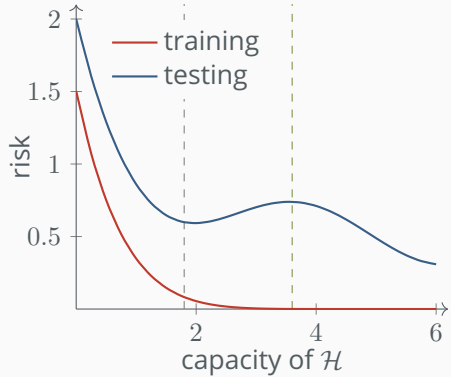
'Prefer the simplest hypothesis $\mathcal{H}$ that fits the data.' In the case of deep learning, this implies the smoothest function that fits the data.



Definition: double descent

Traditionally, we know that increasing the parameters lowers the bias (fitting), but the variance (test risk) will eventually reach a 'sweet spot' (first dashed line) and start to increase again.

The full story has a double descent curve [3], as higher capacity functions past the interpolation threshold (second dashed line) lead again to smoother fitting (Occam's razor).



Natural signals

In nature, the data signal follows patterns:

- There are repetitions in **space** and **time**
- There are various **symmetries**
- Signals are **hierarchical**

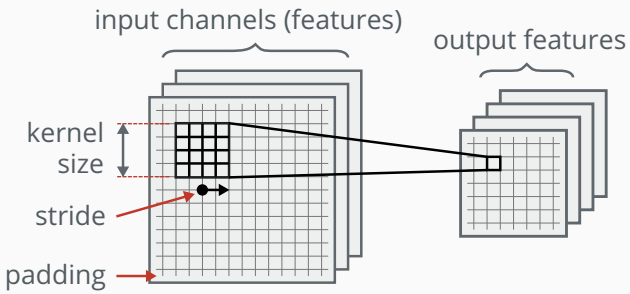
Therefore we can design our functions to fit these patterns effectively

Further reading about deep learning through a physics lens in [4]

Definition: CNNs

A convolution sums the Hadamard product between a sliding area and the convolution kernel. Each output feature map does this for each input channel.

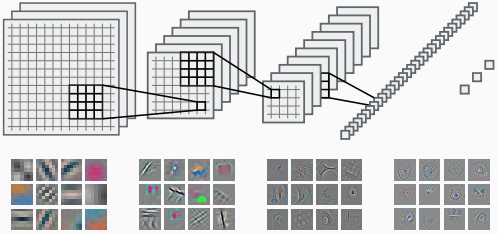
This 2D convolution example is a 4×4 kernel with a stride of 2 and 1 padding. It has 3 input channels and 4 output features.



Hierarchical design

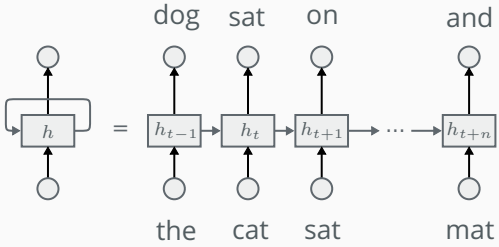
Each feature map output is essentially an image whose intensity values are ‘feature detectors’ from the layer before. The images here are learnt kernel weights.

The ‘width’ vs ‘depth’ problem is based on what type of detectors you need from the dataset, such as to minimise the task risk.



Definition: recurrent neural network

RNNs reuse parameters across multiple timesteps. They can be unrolled to better understand their dynamic behaviour.

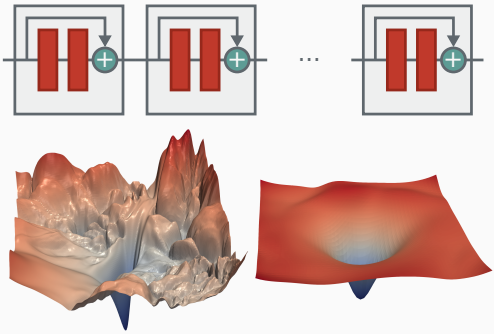


Definition: residual connections

These are shortcuts that skip over two or three layers that contain nonlinearities and batch normalisation between then.

Pseudocode: residual block

```
class ResidualBlock(nn.Module):
  def init(n):
    res_block = [
    nn.Conv2d(in_f=n, out_f=n,3,1,1),
    nn.BatchNorm2d(n),
    nn.ReLU(),
    nn.Conv2d(in_f=n, out_f=n,3,1,1),
    nn.BatchNorm2d(n) ]
  def forward(x):
    return F.relu(x + res_block(x))
```



loss landscape without residuals

loss landscape with residuals [5]

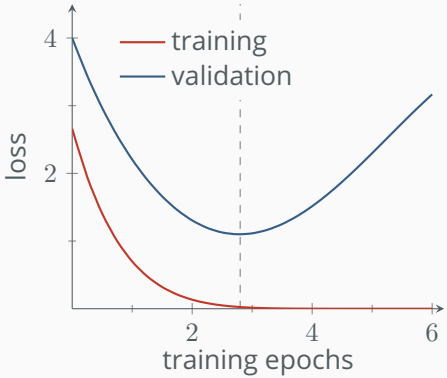
Before we define regularisation, first lets examine two ways to prevent overfitting in high-capacity models:

Definition: early stopping

This is just where we stop training early.

Definition: annealing

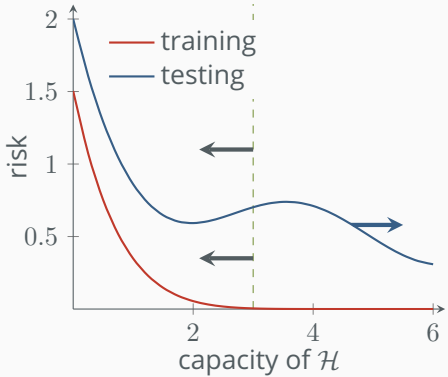
If we decrease the learning rate slowly to zero, this has a similar effect to early stopping (but allows more experience in the process).



Regularisation the effect on model capacity

Definition: regularisation

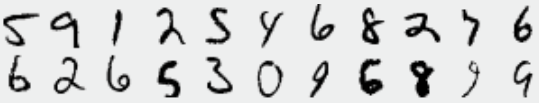
Regularisation is where we add prior information about functions in $\mathcal{H}$ such as to reduce generalisation error.



Definition: data augmentation

If it is expected that small transformations (e.g. rotations, zooms, flips, blurs) will occur in testing, the training samples can be augmented.

However too much augmentation (e.g. too much zoom) will result in poor fitting. In the extreme case it may even change the class label, for example $180°$ rotations in MNIST:



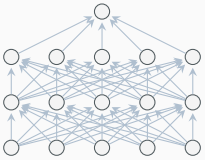
Definition: dropout

Dropout is where each hidden unit is set to zero with some probability (e.g. 0.2). The network can't rely on any one weight, so it spreads its weights out.

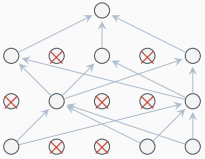
Definition: Tikhonov regularization

Tikhonov regularization (also called **weight decay** or L_2 **regularisation** has a similar effect:

$$L' = L + \lambda ||\mathbf{w}||_2^2$$



standard network

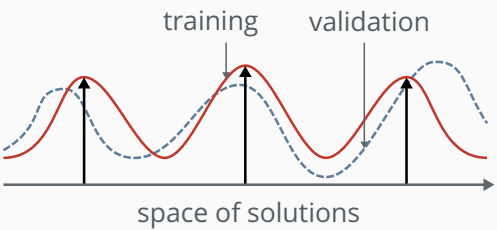


after dropout

Definition: an ensemble

An ensemle is where multiple different models are trained, and then the predictions are combined at test time, for example by averaging or max voting.

This simple technique has shown to be highly successful in winning kaggle competitions, where there is evidence to suggest the success is due to the ability for ensembles to capture multiple modes of the solution space [6].



Take Away Points

Summary

In summary, designing architectures:

- is a scientific process
- choose the right functions to fit the data
- choose your experiments carefully
- consider the double descent graph
- what do you know about the signal?
- what does the task need?

References I

[1] Patrick Kidger and Terry Lyons. “Universal approximation with deep narrow networks”. In: Conference on Learning Theory. 2020, pp. 2306–2327.

[2] Kevin P Murphy. Machine learning: a probabilistic perspective. MIT press, 2012.

[3] Mikhail Belkin et al. “Reconciling modern machine-learning practice and the classical bias–variance trade-off”. In: Proceedings of the National Academy of Sciences 116.32 (2019), pp. 15849–15854.

[4] Henry W Lin, Max Tegmark, and David Rolnick. “Why does deep and cheap learning work so well?” In: Journal of Statistical Physics 168.6 (2017), pp. 1223–1247.

[5] Hao Li et al. “Visualizing the loss landscape of neural nets”. In: Advances in Neural Information Processing Systems. 2018, pp. 6389–6399.

[6] Stanislav Fort, Huiyi Hu, and Balaji Lakshminarayanan. “Deep ensembles: A loss landscape perspective”. In: arXiv preprint arXiv:1912.02757 (2019).