Deep Learning (BEV033DLE)
Lecture 2.
Czech Technical University in Prague

- Neural networks are universal approximators
- Testing networks & loss functions
- Generalisation errors for neural classifiers & regressors
Neural networks as universal approximators

Neural networks are *universal approximators* if we do not restrict the network architecture.

**Boolean functions:** Every boolean function $f : \{\pm 1\}^n \rightarrow \{\pm 1\}$ can be written in conjunctive normal form, i.e. as a conjunction over disjunctive clauses.

**Theorem 1.** *Every boolean function can be represented by a network with binary units and two layers.*

*Remark.* Notice, that the number of neurons grows exponentially with $n$. Implementing e.g. the parity function will require $O(2^n)$ neurons. It can be implemented much more efficiently by a deep network with $O(\log n)$ neurons if we do not restrict its depth.

**Real valued functions:** consider real valued functions $f : [0, 1]^n \rightarrow \mathbb{R}$ that are Lipshitz continuous

$$|f(x) - f(x')| \leq \rho \|x - x'\| \quad \forall x, x' \in [0, 1]^n.$$  

To approximate such function by a network:

- **Partition** : $[0, 1]^n$ into sufficiently small boxes.

- **Design** a network that first decides which box the input vector belongs to and then predicts the average value of $f$ at this box.
Neural networks as universal approximators

Theorem 2. (Cybenko, 1989) Every smooth function on \([0,1]^n\) can be approximated arbitrarily well by a network with sigmoid units and two layers. In other words, given a smooth function \(f : [0,1]^n \to \mathbb{R}\) and an \(\epsilon > 0\), there is a sum

\[
G(x) = \sum_{j=1}^{N} \alpha_j S(w^T_j x + b_j)
\]

s.t. \(|f(x) - G(x)| \leq \epsilon\) for all \(x \in [0,1]^n\).

Remarks:

- There are also “dual” universal approximation theorems that restrict the width of the network (i.e. number of units per layer) and allow arbitrary network depth.
- We limit the expressive power once we fix a network architecture.
Validating & testing neural networks

Given a network, we want to validate its performance on a test set. How *large* shall we choose this set & what *precisely* shall we measure?

- The relation between measurements $x \in \mathcal{X}$ and hidden states $y \in \mathcal{Y}$ is given by a joint probability distribution $p(x, y)$, which is *unknown*.
- The network $h: \mathcal{X} \to \mathcal{Y}$ predicts hidden states $y$, given measurements $x$.
- The loss $\ell(y, y')$ defines the cost incurred by a wrong prediction $y' = h(x)$, if the true hidden state was $y$. Examples:
  - classification, $y$ is categorical: 0/1 loss $\ell(y, y') = [y \neq y']$
  - classification, $y$ is a sequence: Hamming distance $\ell(y, y') = \sum_i [y_i \neq y'_i]$
  - regression, $y \in \mathbb{R}^n$: L1 norm $\ell(y, y') = \|y - y'\|_1$

We want to estimate the risk, i.e. the *expected loss*

$$R(h) = \sum_{x,y} p(x,y) \ell(y,h(x)) \approx \frac{1}{m} \sum_{(x,y) \in \mathcal{T}^m} \ell(x,h(y)) = R_{\mathcal{T}^m}(h)$$

where $\mathcal{T}^m = \{(x^j, y^j) \mid j = 1, \ldots, m\}$ is a test set of i.i.d. examples $x, y \sim p(x,y)$. 
Validating & testing neural networks

Can we upper bound the deviation $|R_{T^m}(h) - R(h)|$?

$$T^m \sim p(x,y) \Rightarrow \mathbb{P}\left(|R(h) - R_{T^m}(h)| > \varepsilon\right) < ??$$

- **Chebyshev inequality:** $\mathbb{P}\left(|R(h) - R_{T^m}(h)| > \varepsilon\right) < \frac{\mathbb{V}[\ell(y, h(x))]}{m\varepsilon^2}$, converges slowly for $m \to \infty$.

- **Hoeffding inequality:** $\mathbb{P}\left(|R(h) - R_{T^m}(h)| > \varepsilon\right) < 2e^{-\frac{2m\varepsilon^2}{(\Delta \ell)^2}}$, where $\Delta \ell = \ell_{max} - \ell_{min}$.

**Example 1.** Consider a classifier with 0/1 loss. What test set size $m$ ensures that $R_{T^m}(h) - 0.01 < R(h) < R_{T^m}(h) + 0.01$ with probability 95%?

Answer: By using Hoeffding inequality, we get $m \approx 2 \cdot 10^4$.

**Example 2.** We train a network and keep several checkpoints with best training accuracy. Then we want to choose the best network from this set $\mathcal{H}$ by comparing their performance on some validation set $T^m$. How large shall we choose $m$?

Answer: use the Hoeffding inequality for a finite set of predictors

$$\mathbb{P}\left(\max_{h \in \mathcal{H}}|R(h) - R_{T^m}(h)| > \varepsilon\right) < 2|\mathcal{H}|^{-\frac{2m\varepsilon^2}{(\Delta \ell)^2}}$$
Learning neural networks: generalisation & overfitting

Given an i.i.d. training set $\mathcal{T}^m = \{(x^j, y^j) \mid j = 1, \ldots, m\}$, we want to train a network $y = h(x, w)$ by minimising its empirical risk, i.e. expected loss on the training set

$$\frac{1}{m} \sum_{(x, y) \in \mathcal{T}^m} \ell(y, h(x, w)) \rightarrow \min_w$$

Often we can not minimise this objective by gradient descent: e.g. classification with 0/1 loss. Let us make a virtue of necessity and consider a different learning criterion: the negative log-likelihood.

- last layer of the network: class scores + softmax, its outputs $h_k(x, w)$ are interpreted as conditional class probabilities $h_k(x, w) = p_w(y = k \mid x)$

- the learning criterion (NLL) reads

$$-\frac{1}{m} \sum_{(x, y) \in \mathcal{T}^m} \log p_w(y \mid x) = -\frac{1}{m} \sum_{(x, y) \in \mathcal{T}^m} \log h_y(x, w) \rightarrow \min_w$$

and is differentiable in $w$.

Advantage: we can estimate the prediction uncertainty.
Generalisation error (bounds) We fix a network architecture. This defines an infinite network class $\mathcal{H}$. We choose the network $h_m \in \mathcal{H}$ with the best performance on a training set $\mathcal{T}^m$. For this we minimise the learning criterion by stochastic gradient descent (SGD).

Can we bound the generalisation error? We would expect the following behaviour for training sets $\mathcal{T}^m$ with fixed size $m$.

![Risk diagram](image)

Generalisation bounds for $\sup_{h \in \mathcal{H}} |R(h) - R_{\mathcal{T}^m}(h)|$ provided by VC dimension estimates for networks are not tight enough. Large networks with $|E| > 10^6$ parameters would require billions of training examples.

Neural networks in typical applications are in an overparametrised regime outside of this plot!
Learning neural networks: generalisation & overfitting

Example 3 (Zhang et al., ICLR, 2018). Image classification on CIFAR (10 classes, $\sim 5 \cdot 10^4$ training examples, tackled by networks with $\sim 10^5$ parameters. The networks learned by SGD and additional regularisers (e.g. data augmentation, dropout, etc.) Achieved accuracy $> 95\%$, generalisation error $< 5\%$. Such networks can learn data with random labels.

Double descent phenomenon: Current ongoing research seems to indicate that SGD, when used for training over-parametrised networks, is choosing smooth predictors with small norm. This leads to unexpected behaviour:

Belkin et al., PNAS, 2019: network with a single hidden layer learned on MNIST
Kernel and rich regimes in overparametrised models:

Let us compare kernel SVMs with neural networks

- the body of the network performs a mapping of the inputs to some feature space, the last network layer represents a linear classifier on this features,
- the “kernel mapping” is parametrised and thus learnable.

At the downside: It may happen that an overparametrised network will learn in a linearised “kernel” regime (e.g. depending on its initialisation):

- The network weights $w$ remain close to their initialisation $w_0$ during learning $\iff$ the network output can be linearised $h(x, w) \approx h(x, w_0) + \nabla_w h(x, w_0)^T (w - w_0)$.
- the network essentially learns a linear classifier for the kernel mapping $x \mapsto \phi(x) = \nabla_w h(x, w_0)$. 