Turing Machine and Deep Learning
Lecture 4: Neural Networks

Satchit Chatterji
satchit.chatterji@gmail.com

MSc Artificial Intelligence
University of Amsterdam

2nd June 2023

Tangent: Regularization

Idea: You don't know how flexible your model *needs* to be.

Solution: Fit a large model, but *constrain* the learning/flexibility somehow. $\Rightarrow$ Regularization.
Regularization covered here:

- L2 + L1 Regularization
- Early Stopping (Neural Nets)
- Dropout (Neural Nets)

Tangent: Regularization

Idea: You don't know how flexible your model *needs* to be.

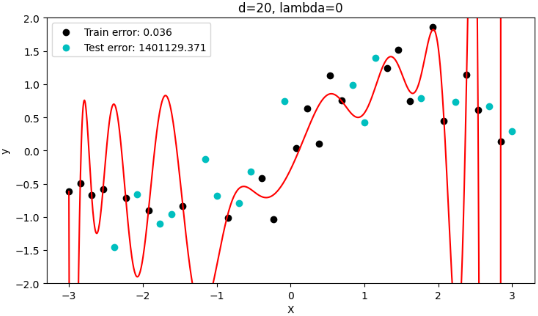
Solution: Fit a large model, but *constrain* the learning/flexibility somehow. $\Rightarrow$ Regularization.
Regularization covered here:

- $\rightarrow$ **L2 + L1 Regularization**
- Early Stopping (Neural Nets)
- Dropout (Neural Nets)

L2 Regularization

Polynomial regression:

$$y = w_0 + w_1 x + w_2 x^2 + \ldots + w_d x^d$$



$$\hat{w} = [0.0, 2.29, 2.1, -2.89, 0.13, -5.71, -12.64, 19.78, 19.04, -19.6, \\ -12.81, 9.44, 4.83, -2.5, -1.09, 0.37, 0.15, -0.03, -0.01, 0.0, 0.0]^T$$

L2 Regularization

Polynomial regression:

$$y = w_0 + w_1 x + w_2 x^2 + ... + w_d x^d$$

L2 Regularization

Polynomial regression:

$$y = w_0 + w_1 x + w_2 x^2 + ... + w_d x^d$$

Learning task:

$$\hat{f} = h_{opt} = \arg\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} (y_i - h(x_i))^2$$

L2 Regularization

Polynomial regression:

$$y = w_0 + w_1 x + w_2 x^2 + ... + w_d x^d$$

Learning task:

$$\hat{f} = h_{opt} = \arg\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} (y_i - h(x_i))^2$$

Regularization: Penalize weights by their size:

$$\text{L2 regularization term} = \lambda \cdot \sum_{i=0}^{d} w_i^2$$

L2 Regularization

Polynomial regression:

$$y = w_0 + w_1 x + w_2 x^2 + ... + w_d x^d$$

Learning task:

$$\hat{f} = h_{opt} = \arg\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} (y_i - h(x_i))^2$$

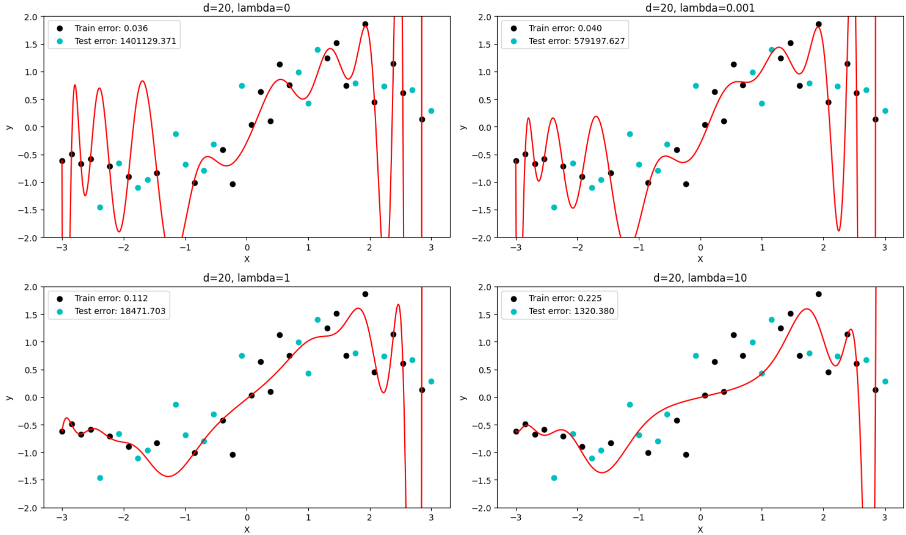
Regularization: Penalize weights by their size:

$$\text{L2 regularization term} = \lambda \cdot \sum_{i=0}^{d} w_i^2$$

New learning task:

$$\hat{f} = h_{opt} = \arg\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} (y_i - h(x_i))^2 + \lambda \cdot \sum_{i=0}^{d} w_i^2$$

L2 Regularization Example



L2 Weight Analysis

$\lambda = 0$:
$$\hat{w} = [0.0, 2.29, 2.1, -2.89, 0.13, -5.71, -12.64, 19.78, 19.04, -19.6,\\ -12.81, 9.44, 4.83, -2.5, -1.09, 0.37, 0.15, -0.03, -0.01, 0.0, 0.0]^T$$

$\lambda = 0.001$:
$$\hat{w} = [0.0, 2.33, 2.91, -5.66, -5.19, 6.59, 0.13, 1.23, 5.18, -5.98,\\ -4.69, 3.88, 2.03, -1.18, -0.5, 0.19, 0.07, -0.02, -0.01, 0.0, 0.0]^T$$

$\lambda = 1$: $\hat{w} = [0.0, 0.89, 0.03, 0.36, -0.07, 0.09, -0.07, -0.19, -0.01, -0.19, 0.06, 0.24, -0.01, -0.09, -0.0, 0.02, 0.0, -0.0, -0.0, 0.0, 0.0]^T$

$\lambda = 10$: $\hat{w} = [0.0, 0.26, -0.03, 0.15, -0.05, 0.1, -0.04, 0.03, 0.01, -0.03, 0.05, 0.0, -0.03, -0.0, 0.01, 0.0, -0.0, -0.0, 0.0, 0.0, 0.0]^T$

Neural Networks

Recap: Supervised ML

Given

Input-output pairs of the form:

$$S = (x_i, y_i)$$

$$x_i \in \mathbb{R}^K, y_i \in \mathbb{R}^M, i = \{1, \ldots, n\}$$

Assumption

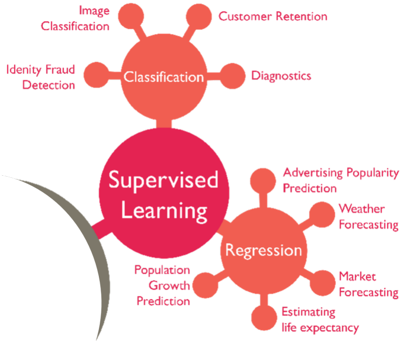
Data is generated by a ‘true’ function f with some noise:

$$y_i = f(x_i) + \nu_i$$

Goal

‘Learn’ an approximation $\hat{f}$ that is close to the real function f over all S:

$$\hat{f}(x_i) \approx f(x_i) \;\; \forall i = \{1, \ldots, n\}$$



SML view on NNs

A neural network $\mathcal{N}_\theta$ is

- a non-linear function $\mathbb{R}^K \mapsto \mathbb{R}^M$ parameterised by weights θ.
- Trained with some loss function $\mathcal{L} : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}^{\geq 0}$
- Goal: find θ_{opt} such that:

$$\theta_{opt} = \underset{\theta \in \Theta}{\operatorname{argmin}}\, E[\mathcal{L}(\mathcal{N}_\theta(X), Y)]$$

- Best we can do: find θ_{opt} such that:

$$\theta_{opt} = \underset{\theta \in \Theta}{\operatorname{argmin}}\, \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\mathcal{N}_\theta(x_i), y_i)$$

- *Generally* using an iterative method such as *gradient descent*

Regularization in Neural Networks

Regularization in Neural Networks

There are a *lot* of ways to regularize the weights/learning in neural networks.

- L2 + L1 Regularization *(covered)*
- Early Stopping
- Dropout

Early Stopping

Objective: Find the point during training where the model performs well on the validation set and prevent it from continuing to learn and overfit the training data.
Process: Stop learning when the model's performance on the validation set starts doing *worse*.
Benefits: Prevents overfitting, saves computational resources, provides regularization
Considerations: Validation set size and quality, careful selection of stopping point

Dropout

Objective: Improve generalization by randomly dropping out neurons during training.
Process: Neurons are probabilistically "dropped out" with a specified probability during each training iteration – the incoming weights associated with this neuron is zero.
Effect: Forces the network to learn redundant representations and reduces interdependence among neurons $\Rightarrow$ equivalent to an ensemble of neural networks (source)
Implementation: Dropout is applied only during training, *not* during testing or inference (though there are some uncertainty quantification methods that disregard this)
Benefits: Prevents overfitting, improves model robustness, acts as a form of ensemble learning.
Considerations: Dropout probability selection, impact on training time.