Chapter 5:
Overfitting & Regularization

5.1 Overfitting & Underfitting

In the previous chapters, we learned about prediction functions but never talked about how well they describe our data. It could be the case that our prediction function is too simple, making our predictions not accurate enough. It could also be the case, that our prediction function is too complex, basically learning the data by heart and not being effective at predictions.

In Machine Learning we want to find a middle ground, the prediction function should not be too complex nor too simple.

Let us take the example of the gaussian kernel with positive bandwidth ($\sigma > 0$).

$$k(\mathbf{x}, \tilde{\mathbf{x}}) := \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} - \tilde{\mathbf{x}}\|^2\right)$$

Figure 1: Various prediction functions. We see that the middle one with moderate width describes our data best.

Definition: Underfitting
Learning a too simple classifier that is not complex enough to describe the training data well is called ***underfitting****.*
Definition: Overfitting
Learning a too complex classifier that fits the training data "too well" (does not "generalize" to new data) is called ***overfitting****.*

We can observe the underfitting and overfitting behavior in other learning machines as well. Let us take the k-nearest neighbor algorithm:

- Choosing k small we don't look at enough points to make a generalized prediction, leading to overfitting.

- Choosing k large, we include points that might be too far away, leading to underfitting.

Our high level idea to avoid overfitting and underfitting, will be:

1. Choose a complex enough classifier to avoid underfitting.
2. Use a technique called **Regularization** to avoid overfitting.

5.2 Unifying Loss View

We would like to take a more abstract view on the learning machines we studied so far (SVM, LR, ANN).
Let us take a look at the following optimization problem:

$$\min_{[W,]b,\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \ell\left(y_i\left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle + b\right)\right) \left[+\frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\text{Fro}}^2\right] \tag{UE}$$

1. Let $\ell(t) := \max\{0, 1-t\}$ and $\phi := id$ and by leaving out the bracketed values, we get our standard soft margin SVM:

$$\min_{b,\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right)\right) \tag{SVM}$$

2. Let $\ell(t) := \max\{0, 1-t\}$ and $\phi := \phi_k$ and by leaving out the bracketed values, we get our kernel SVM:

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \max\left(0, 1 - y_i\left(\langle \mathbf{w}, \phi_k\left(\mathbf{x}_i\right)\rangle + b\right)\right) \tag{Kernel SVM}$$

3. Let $\ell(t) := \ln((1+\exp(-t))$ and $\phi := id$ and by leaving out the bracketed values, we get logistic regression:

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \ln\left(1 + \exp\left(-y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right)\right)\right) \tag{LR}$$

4. Let $\ell(t) := \ln((1+\exp(-t))$ and $\phi := \phi_k$ and by leaving out the bracketed values, we get the kernelized logistic regression:

$$\min_{b\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \ln\left(1 + \exp\left(-y_i\left(\langle \mathbf{w}, \phi_k\left(\mathbf{x}_i\right)\rangle + b\right)\right)\right) \tag{Kernel LR}$$

5. Let $\ell(t) := \ln((1+\exp(-t))$ and $\phi := \phi_W$ and taking into account the bracketed values we get our ANN.

$$\min_{b,\mathbf{w},W} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\text{Fro}}^2 + C\sum_{i=1}^{n} \ln\left(1 + \exp\left(-y_i\left(\mathbf{w}^\top \phi_W\left(\mathbf{x}_i\right) + b\right)\right)\right) \tag{ANN}$$

We can summarize this result in the following image:

loss ℓ \ map ϕ	id	ϕ_K	ϕ_W
hinge	linear SVM	kernel SVM	—[1]
logistic	linear LR	kernel LR	ANN

Figure 2: The 5 big learning machines summarized into one equation.

The unifying equation (UE) contains, for every training example $(\mathbf{x}_i, y_i)$ a term

$$\ell(\underbrace{y_i \langle \mathbf{w}, \phi(\mathbf{x}) \rangle + b}_{=:t_i})$$

which we call the loss of the i-th example.

Figure 3: Large t lead to lower $\ell(t)$ values

We try to minimize:

$$\sum_{i=1}^{n} \ell\left(y_i \left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right) \rangle + b\right)\right)$$

which promotes large t_i values.

This is intuitive as $t_i > 0$ means we classify i-th example correctly.

[1] The hinge loss is theoretically possible but uncommon in neural networks.

In theory we are allowed to choose arbitrary loss functions. Let us have a look at the 0-1 loss function.

Definition: 0-1 Loss

The function $\ell(t) := \text{sign}(-t)$ *is called* $\mathbf{0-1}$ ***loss.***

Figure 4: The hinge loss and 0-1 loss in comparison

Observation

We look at

$$\ell(t_i) := -\text{sign}(y_i\left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle + b\right))$$

as discussed in Chapter 1, $\ell(t_i) = 1$ when the label is misclassified ($t_i < 0$) and $\ell(t_i) = 0$ otherwise. Thus the cumulative $0-1$ loss

$$\sum_{i=1}^{n} \ell\left(y_i\left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle + b\right)\right)$$

measures the number of training errors. Sadly, optimizing over this discrete problem is NP-hard so we still need to look at the convex alternatives as the hinge loss and logistic regression.

5.3 Regularization

Let us recall the unifiying equation with parameters $\theta := (b, \mathbf{w}, [, W])$ with the bracketed terms only for ANNs:

$$\min_{[W,]b,\mathbf{w}} \underbrace{\frac{1}{2}\|\mathbf{w}\|^2 \left[+\frac{1}{2}\sum_{l=1}^{L}\|W_l\|^2\right]}_{\text{regularizer } R(\theta)} + \underbrace{C}_{\text{regularization constant}} \left(\underbrace{\sum_{i=1}^{n} \ell\left(y_i\left(\langle \mathbf{w}, \phi\left(\mathbf{x}_i\right)\rangle + b\right)\right)}_{\text{loss } L(\theta)}\right)$$

We want to discuss the influence of the regularization constant C.

Increasing the value of C means to increase the importance of the loss term $L(\theta)$ and decrease the importance of the regularizer term $R(\theta)$.

The regularizer term $R(\theta)$ will impose a penalty on the complexity of our chosen prediction function.
Regularization is a general concept having to do with restriction on choice of parameters. We see that:

High regularization constant $C \implies$ low regularization $\implies$ high overfitting.

Low regularization constant $C \implies$ high regularization $\implies$ low overfitting.

Theorem 5.3.1

$$\min_{\theta} \underbrace{R(\theta)}_{\text{regularizer}} + \underbrace{C}_{\text{regularization constant}} \underbrace{L(\theta)}_{\text{Loss}}$$

is equivalent to

$$\begin{array}{ll} \min_{\theta} & L(\theta) \\ \text{s.t.} & R(\theta) \leq \tilde{C} \end{array}$$

for an appropriate choice of $\tilde{C}$.

Proof:
By the Lagrangian duality theorem (L),

$$\begin{aligned} & \min_{\theta} L(\theta) \quad \text{s.t.} \quad R(\theta) \leq \tilde{C} \\ \overset{(L)}{=} & \max_{\lambda \geq 0} \min_{\theta} L(\theta) + \lambda(R(\theta) - \tilde{C}) \\ \overset{(L)}{=} & \min_{\theta} \max_{\lambda \geq 0} L(\theta) + \lambda(R(\theta) - \tilde{C}) \\ = & \min_{\theta} L(\theta) + \lambda^*(R(\theta) - \tilde{C}) \\ = & \min_{\theta} L(\theta) + \underbrace{\lambda^*}_{=:C} R(\theta) \end{aligned}$$

where (θ^*, λ^*) is the optimal value the above min-max problem. $\square$

The proof is important because it tells us how to generate a regularizer from a hard constraint. In other words, the regularizers origin is a constraint.

By changing $\tilde{C}$ we can control the size of the set

$$\{\theta : R(\theta) \leq \tilde{C}\}.$$

The larger the set, the more likely our learning machine will pick a function that describes our training data too well.

We risk that our prediction function overfits.

5.4 Regularization in Deep Learning

Deep neural networks have a high risk of overfitting due to their complexity. We discuss some special strategies to avoid overfitting in the setting of neural networks.

Consider a large neural network involving many layers and neurons. Clearly, using a lot of SGD iterations, the neural network can adapt to our training data much better than a small network leading to the potential of overfitting, transitioning to our first regularization strategy in neural networks.

Trick 1: Early Stopping

- Split the training data into two sets:
  1. New smaller training set
  2. Validation Set
- Monitor the validation set errors every couple of mini batch iterations
- Stop SGD when the validation error goes up.

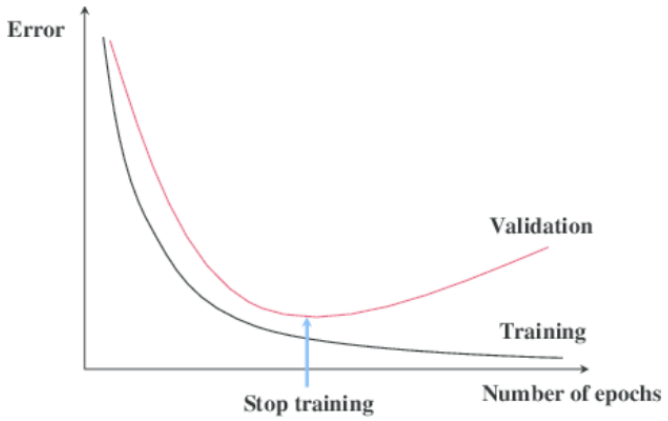

Figure 5: Initially as the number of iterations increase, the neural network describes our control data (Validation Set) better. The validation error decreases. Once it starts overfitting the validation error will start increasing. This is our stopping point.

Trick 2: Batch normalization
Batch normalization is a strategy in which we attempt to normalize the input into the activation function. The gradient of activation functions is typically most informative when its input has mean 0 and variance 1, however this is dependant on the activation function. Some examples:

- ReLU, if its input is very large (very small) its gradient is 1 (0). As such ReLU does not seem to be relevant unless its input is both large and small within a batch, in which case its mean will be around 0.
- All sigmoidal functions (i.e. the sigmoid, tanh, err) have vanishing gradients, i.e. for very small or very large inputs their gradient are close to 0. As such, we want their input to have mean 0 and a small variance.

With this in mind we can learn parameters that will allow us to normalize the input of an activation function. Batch normalization first normalizes the input by learning the mean μ and variance σ and subtracting this learned mean from the input and dividing by the variance, bringing the input to mean 0 and variance 1. It then additionally allows learning parameters γ and β which rescale and shift the normalized input. After batch normalization the input to the activation can have any learned mean and variance. In effect the optimization algorithm can then optimize the optimization.

Algorithm Batch Normalization

Denote, for a neuron f, its activation values on a mini-batch by $\mathbf{f} = (f_1, \dots f_B)$. Then, for each mini-batch in the SGD optimization and every neuron in the ANN, do:

1: Center each neuron activation: $\forall i = 1, \dots, B : f_i \leftarrow f_i - \mu_f$

2: Normalize the spread: $f_i \leftarrow f_i / \sigma_f$

3: Overwrite the neuron's activation by the learned parameters (γ, β) with $\gamma \mathbf{f} + \beta$

Definition: Ensemble Methods *Ensemble methods are methods that aggregate the prediction from multiple predictors*

Example: Ensemble Method

1. Train k machine learning algorithms, resulting in prediction functions $f_1, \dots, f_k$
2. Predict by majority vote: $f(\mathbf{x}) = +1$ if $|i : f_i(\mathbf{x}) = +1| \geq |i : f_i(\mathbf{x}) = -1|$ and $f(\mathbf{x}) = -1$ elsewise

The drawback to Ensemble Methods is that training multiple deep ANN is very costly. We try to bypass this problem by using the following trick.

Trick 3: Dropout Regularization
We simulate many ANN by randomly hiding some neurons from our original net in each SGD step. This process is formally called Dropout Regularization.

Dropout regularization randomly hides in each mini-batch SGD iteration a fixed percentage of randomly selected neurons of the network. The intuition is that the network can not fixate on any one feature, as this feature might not be present during optimization, as it might be hidden. As such the network learns

more general representations for concepts.

Trick 4: Data Augmentation
The optimal setting for optimization is online learning. In this setting there is an infinite amount of data and in each optimization step it uses new previously unseen data. Obviously online learning is not very realistic save for a very few prblems. However, data augmentation creates "new training data" from old data. For example an image of a cat could be mirrored, translated, noised, rescaled, zoomed in, zoomed out, etc. This leads to more informative training data.

It promotes generalization, as we are adding variety to our training data.

Trick 5: Transfer Learning
Assuming we have a small training dataset, we can do the following:

1. Pre-training: Download an ANN from the web that was pre-trained on huge amounts of images or alternatively train an ANN on some huge dataset.

2. Fine-tuning: Run SGD optimization on our small dataset, but using the ANN from the previous step, effectively "fine-tuning" it to the small dataset.

This regularization strategy is known as transfer learning − that is, transfering information into a learning problem from a related problem.