Regularization in Deep Learning

Deep Learning (ENGG*6600*01)

School of Engineering,
University of Guelph, ON, Canada

Course Instructor: Benyamin Ghojogh

Summer 2023

Prerequisite for This Lecture

- The prerequisite for this lecture is the lecture of overfitting and regularization on my YouTube channel.
- The name of video is “Overfitting, Cross Validation, Regularization, and L1 and L2 Norm Regularization in Machine Learning” in my YouTube channel.
- The link of video: `https://www.youtube.com/watch?v=wds4KdXQJIA`
- Please see that lecture first before watching this lecture. We assume you have studied it.

Weight Decay

Weight Decay

- Recall the ℓ_2 regularization from the lecture of overfitting. If we replace the objective variable $\boldsymbol{x}$ with the vector of neural network weights $\boldsymbol{w}$, we will have:

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad \widetilde{J}(\boldsymbol{w};\theta) := J(\boldsymbol{w};\theta) + \frac{\alpha}{2}\,||\boldsymbol{w}||_2^2, \tag{1}$$

  which can be the loss function optimized in a neural network [1].
- Penalizing the weights with regularization is referred to as **weight decay** [2, 3].
- This penalty prevents neural network from becoming too non-linear (complex) and thus overfitted.
- The reason is that according to **non-linear activation functions** such as hyperbolic tangent, **very large weights** (very positive or very negative) are in the **very non-linear parts of the activation functions**.
- Although neural network should not be completely linear in order to be able to learn non-linear patterns, it should not be very non-linear as well, not to be overfitted to the training data.
- Penalizing the weights makes the weights relatively small (where the activation functions are almost linear) to have a balance in linearity and non-linearity.
- In the lecture of overfitting, we proved that the result of Eq. (1) is:

$$\boldsymbol{w}^\dagger = \boldsymbol{U}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{U}^\top\boldsymbol{w}^*, \tag{2}$$

  where $\boldsymbol{w}^*$ is the solution of non-regularized problem and $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ contain the eigenvectors and eigenvalues of Hessian of J, respectively.
- It has the similar interpretations as we discussed before in the lecture of overfitting.

Noise Injection to Input in Neural Networks

Noise Injection to Input in Neural Networks

- In training neural networks, it is beneficial to **add noise to the input** [4].
- One perspective to why adding noise to input helps better training of network is **data augmentation** [5, 6]. Data augmentation is useful for training deep networks because they have a **huge number of weights (parameters)** and if we do not introduce enough training data to them, they will overfit to the training data.
- Another interpretation of noise injection to input is **regularization** [7, 1]. Assume that the optimization of neural network is:

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad J := \mathbb{E}((\widehat{y}(\boldsymbol{x}) - y)^2), \tag{3}$$

  where $\boldsymbol{x}$, $\widehat{y}(\boldsymbol{x})$, and y are the input, the estimation (output) of network, and the training label, respectively.
- We add noise $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$ to the input, so the objective function changes to:

$$\begin{aligned}\widetilde{J} &:= \mathbb{E}((\widehat{y}(\boldsymbol{x}+\boldsymbol{\varepsilon}) - y)^2) = \mathbb{E}(\widehat{y}^2(\boldsymbol{x}+\boldsymbol{\varepsilon}) - 2y\widehat{y}(\boldsymbol{x}+\boldsymbol{\varepsilon}) + y^2) \\ &= \mathbb{E}(\widehat{y}^2(\boldsymbol{x}+\boldsymbol{\varepsilon})) - 2\mathbb{E}(y\widehat{y}(\boldsymbol{x}+\boldsymbol{\varepsilon})) + \mathbb{E}(y^2).\end{aligned}$$

- Assuming that the variance of noise is small, the Taylor series expansion of $\widehat{y}(\boldsymbol{x}+\boldsymbol{\varepsilon})$ is:

$$\widehat{y}(\boldsymbol{x}+\boldsymbol{\varepsilon}) = \widehat{y}(\boldsymbol{x}) + \boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{\varepsilon}^\top \nabla^2_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x})\, \boldsymbol{\varepsilon} + o(\boldsymbol{\varepsilon}^3).$$

Noise Injection to Input in Neural Networks

- We had:

$$\begin{aligned}
\widetilde{J} &= \mathbb{E}(\widehat{y}^2(\boldsymbol{x}+\boldsymbol{\varepsilon})) - 2\mathbb{E}(y\widehat{y}(\boldsymbol{x}+\boldsymbol{\varepsilon})) + \mathbb{E}(y^2), \\
\widehat{y}(\boldsymbol{x}+\boldsymbol{\varepsilon}) &= \widehat{y}(\boldsymbol{x}) + \boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{\varepsilon}^\top \nabla^2_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x})\, \boldsymbol{\varepsilon} + o(\boldsymbol{\varepsilon}^3).
\end{aligned}$$

- Therefore:

$$\begin{aligned}
\widetilde{J} \approx\, &\mathbb{E}\Big(\big(\widehat{y}(\boldsymbol{x}) + \boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{\varepsilon}^\top \nabla^2_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x})\, \boldsymbol{\varepsilon}\big)^2\Big) \\
&- 2\mathbb{E}\Big(y\widehat{y}(\boldsymbol{x}) + y\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x}) + \frac{1}{2} y\boldsymbol{\varepsilon}^\top \nabla^2_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x})\, \boldsymbol{\varepsilon}\Big) + \mathbb{E}(y^2) \\
&= \mathbb{E}\Big(\widehat{y}(\boldsymbol{x})^2 + y^2 - 2y\widehat{y}(\boldsymbol{x})\Big) - 2\mathbb{E}\Big(\frac{1}{2} y\boldsymbol{\varepsilon}^\top \nabla^2_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x})\, \boldsymbol{\varepsilon}\Big) \\
&+ \mathbb{E}\Big(\widehat{y}(\boldsymbol{x})\boldsymbol{\varepsilon}^\top \nabla^2_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x})\boldsymbol{\varepsilon} + (\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x}))^2 + o(\boldsymbol{\varepsilon}^3)\Big).
\end{aligned}$$

- The first term, $\mathbb{E}(\widehat{y}(\boldsymbol{x})^2 + y^2 - 2y\widehat{y}(\boldsymbol{x})) = \mathbb{E}((\widehat{y}(\boldsymbol{x}) - y)^2)$, is the loss function before adding the noise to the input, according to Eq. (3).
- Also, because of $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$, we have $\mathbb{E}(\boldsymbol{\varepsilon}^\top \boldsymbol{\varepsilon}) = \sigma^2$. As the noise and the input are independent, the following term is simplified as:

$$\mathbb{E}\Big((\boldsymbol{\varepsilon}^\top \nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x}))^2\Big) \overset{\perp\!\!\!\perp}{=} \mathbb{E}(\boldsymbol{\varepsilon}^\top \boldsymbol{\varepsilon})\, \mathbb{E}(||\nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x})||_2^2) = \sigma^2\, \mathbb{E}(||\nabla_{\boldsymbol{x}} \widehat{y}(\boldsymbol{x})||_2^2).$$

Noise Injection to Input in Neural Networks

- The rest of expression is simplified as:

$$\begin{aligned}
&\mathbb{E}\Big(\widehat{y}(\boldsymbol{x})\boldsymbol{\varepsilon}^\top \nabla^2_{\boldsymbol{x}}\widehat{y}(\boldsymbol{x})\boldsymbol{\varepsilon}\Big) - 2\mathbb{E}\Big(\frac{1}{2} y\boldsymbol{\varepsilon}^\top \nabla^2_{\boldsymbol{x}}\widehat{y}(\boldsymbol{x})\,\boldsymbol{\varepsilon}\Big) \\
&= \mathbb{E}\Big(\widehat{y}(\boldsymbol{x})\boldsymbol{\varepsilon}^\top \nabla^2_{\boldsymbol{x}}\widehat{y}(\boldsymbol{x})\boldsymbol{\varepsilon}\Big) - \mathbb{E}\Big(y\boldsymbol{\varepsilon}^\top \nabla^2_{\boldsymbol{x}}\widehat{y}(\boldsymbol{x})\,\boldsymbol{\varepsilon}\Big) \\
&\overset{\perp\!\!\!\perp}{=} \mathbb{E}(\boldsymbol{\varepsilon}^\top\boldsymbol{\varepsilon})\,\mathbb{E}\big(\widehat{y}(\boldsymbol{x})\nabla^2_{\boldsymbol{x}}\widehat{y}(\boldsymbol{x})\big) - \mathbb{E}(\boldsymbol{\varepsilon}^\top\boldsymbol{\varepsilon})\,\mathbb{E}\big(y\nabla^2_{\boldsymbol{x}}\widehat{y}(\boldsymbol{x})\big) \\
&= \sigma^2\,\mathbb{E}\Big(\big(\widehat{y}(\boldsymbol{x}) - y\big)\nabla^2_{\boldsymbol{x}}\widehat{y}(\boldsymbol{x})\Big).
\end{aligned}$$

- Hence, the overall loss function after noise injection to the input is simplified to:

$$\widetilde{J} \approx J + \sigma^2\,\mathbb{E}\Big(\big(\widehat{y}(\boldsymbol{x}) - y\big)\nabla^2_{\boldsymbol{x}}\widehat{y}(\boldsymbol{x})\Big) + \sigma^2\,\mathbb{E}(||\nabla_{\boldsymbol{x}}\widehat{y}(\boldsymbol{x})||_2^2), \tag{4}$$

  which is a regularized optimization problem with ℓ_2 norm penalty.

- The penalty is on the second derivatives of outputs of neural network. This means that we do not want to have significant changes in the output of neural network. This penalization prevents from overfitting.

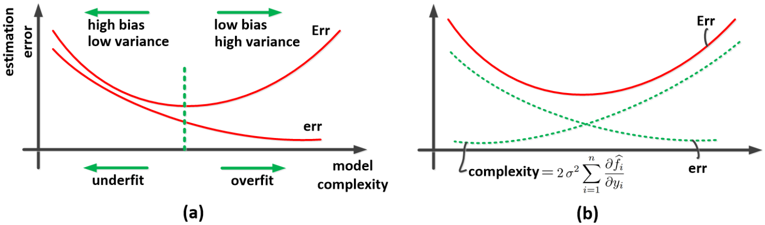
Noise Injection to Input in Neural Networks

- Note that the technique of adding noise to the input is also used in denoising autoencoders [8].
- Moreover, an overcomplete autoencoder with one hidden layer [1] (where the number of hidden neurons is greater than the dimension of data) needs a noisy input; otherwise, the mapping in the autoencoder will be just copying the input to output without learning a latent space.
- It is also noteworthy that **injecting noise to the weights** of neural network [1, 9] can be interpreted similar to injecting noise to the input. Therefore, noise injection to the weights can also be interpreted as regularization where the regularization penalty term is $\sigma^2\,\mathbb{E}(||\nabla_{\boldsymbol{w}}\widehat{y}(\boldsymbol{x})||_2^2)$ where $\boldsymbol{w}$ is the vector of weights [1].

Early Stopping in Neural Networks

Early Stopping in Neural Networks

- As we mentioned in the explanations of overfitting lecture, we train neural network up to a point where the overfitting is starting. This is referred to as **early stopping** [10, 11] which helps avoid overfitting [12].



Early Stopping in Neural Networks

- Recall the overfitting lecture:
  - A regularized optimization:

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad \widetilde{J}(\boldsymbol{w};\theta) := J(\boldsymbol{w};\theta) + \frac{\alpha}{2}\,||\boldsymbol{w}||_2^2. \tag{5}$$

    The ℓ_2 norm regularization is also referred to as *ridge regression* or *Tikhonov regularization* [1].
  - Suppose $\boldsymbol{w}^*$ is minimizer of the $J(\boldsymbol{w};\theta)$, i.e.:

$$\nabla J(\boldsymbol{w}^*;\theta) = 0. \tag{6}$$

  - The Taylor series expansion of $J(\boldsymbol{w};\theta)$ up to the second derivative at $\boldsymbol{w}^*$ gives:

$$\widehat{J}(\boldsymbol{w};\theta) \approx J(\boldsymbol{w}^*;\theta) + \frac{1}{2}(\boldsymbol{w}-\boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w}-\boldsymbol{w}^*), \tag{7}$$

    where $\boldsymbol{H} \in \mathbb{R}^{d\times d}$ is the Hessian.
  - If we apply eigenvalue decomposition on the Hessian matrix, we will have:

$$\boldsymbol{H} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top, \tag{8}$$

    where $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ contain the eigenvectors and eigenvalues, respectively.

Early Stopping in Neural Networks

- According to Eq. (7), we have:

$$\nabla_{\boldsymbol{w}} \widehat{J}(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^*) + \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*) \overset{(6)}{=} \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*).$$

- The gradient descent (with η as the learning rate) used in back-propagation of neural network is [13]:

$$\boldsymbol{w}^{(t)} := \boldsymbol{w}^{(t-1)} - \eta \nabla_{\boldsymbol{w}} \widehat{J}(\boldsymbol{w}^{(t)}) = \boldsymbol{w}^{(t-1)} - \eta \boldsymbol{H}(\boldsymbol{w}^{(t-1)} - \boldsymbol{w}^*)$$
$$\implies \boldsymbol{w}^{(t)} - \boldsymbol{w}^* = (\boldsymbol{I} - \eta \boldsymbol{H})(\boldsymbol{w}^{(t-1)} - \boldsymbol{w}^*),$$

  where t is the index of iteration.

- According to Eq. (8), we have:

$$\boldsymbol{w}^{(t)} - \boldsymbol{w}^* = (\boldsymbol{I} - \eta\, \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top)(\boldsymbol{w}^{(t-1)} - \boldsymbol{w}^*).$$

  Assuming the initial weights are $\boldsymbol{w}^{(0)} = 0$, we have:

$$\boldsymbol{w}^{(1)} - \boldsymbol{w}^* = -(\boldsymbol{I} - \eta\, \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top)\boldsymbol{w}^* \implies \boldsymbol{w}^{(1)} = \big(\boldsymbol{I} - (\boldsymbol{I} - \eta\, \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top)\big)\boldsymbol{w}^*$$
$$\overset{(a)}{\implies} \boldsymbol{w}^{(1)} = \big(\boldsymbol{U}\boldsymbol{U}^\top - (\boldsymbol{U}\boldsymbol{U}^\top - \eta\, \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top)\big)\boldsymbol{w}^* \implies \boldsymbol{w}^{(1)} = \boldsymbol{U}\big(\boldsymbol{I} - (\boldsymbol{I} - \eta\, \boldsymbol{\Lambda})\big)\boldsymbol{U}^\top \boldsymbol{w}^*,$$

  where (a) is because $\boldsymbol{U}$ is a non-truncated orthogonal matrix so $\boldsymbol{U}\boldsymbol{U}^\top = \boldsymbol{I}$.

Early Stopping in Neural Networks

- We found:

$$\boldsymbol{w}^{(1)} = \boldsymbol{U}\big(\boldsymbol{I} - (\boldsymbol{I} - \eta\,\boldsymbol{\Lambda})\big)\boldsymbol{U}^\top \boldsymbol{w}^*.$$

- By induction, we have:

$$\begin{aligned}&\boldsymbol{w}^{(t)} = \boldsymbol{U}\big(\boldsymbol{I} - (\boldsymbol{I} - \eta\,\boldsymbol{\Lambda})^t\big)\boldsymbol{U}^\top \boldsymbol{w}^* \implies \boldsymbol{U}^\top\boldsymbol{w}^{(t)} = \boldsymbol{U}^\top\boldsymbol{U}\big(\boldsymbol{I} - (\boldsymbol{I} - \eta\,\boldsymbol{\Lambda})^t\big)\boldsymbol{U}^\top \boldsymbol{w}^*,\\ &\overset{(a)}{\implies} \boldsymbol{U}^\top\boldsymbol{w}^{(t)} = \big(\boldsymbol{I} - (\boldsymbol{I} - \eta\,\boldsymbol{\Lambda})^t\big)\boldsymbol{U}^\top \boldsymbol{w}^*, \qquad (9)\end{aligned}$$

where (a) is because $\boldsymbol{U}$ is an orthogonal matrix so $\boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{I}$.

- On the other hand, recall Eq. (2):

$$\begin{aligned}&\boldsymbol{w}^\dagger = \boldsymbol{U}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{U}^\top\boldsymbol{w}^*,\\ &\implies \boldsymbol{U}^\top\boldsymbol{w}^\dagger = (\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{U}^\top\boldsymbol{w}^*,\\ &\overset{(a)}{\implies} \boldsymbol{U}^\top\boldsymbol{w}^\dagger = \big(\boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\alpha\big)\boldsymbol{U}^\top\boldsymbol{w}^*, \qquad (10)\end{aligned}$$

where (a) is because of an expression rearrangement asserted in [1].

- Comparing Eqs. (9) and (10) shows that early stopping can be seen as a ℓ_2 norm regularization or weight decay [1].

Early Stopping in Neural Networks

- The Eqs. (9) and (10):

$$\boldsymbol{U}^\top \boldsymbol{w}^{(t)} = \big(\boldsymbol{I} - (\boldsymbol{I} - \eta\,\boldsymbol{\Lambda})^t\big)\,\boldsymbol{U}^\top \boldsymbol{w}^*,$$
$$\boldsymbol{U}^\top \boldsymbol{w}^\dagger = \big(\boldsymbol{I} - (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\alpha\big)\,\boldsymbol{U}^\top \boldsymbol{w}^*.$$

- Actually, the Eqs. (9) and (10) are equivalent if:

$$(\boldsymbol{I} - \eta\,\boldsymbol{\Lambda})^t = (\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\alpha, \tag{11}$$

  for some η, t, and α.

- If we take the logarithm from these expressions and use Taylor series expansion for $\log(1+x)$, we have:

$$\log(\boldsymbol{I} - \eta\,\boldsymbol{\Lambda})^t = t\log(\boldsymbol{I} - \eta\,\boldsymbol{\Lambda}) \approx -t\,(\eta\boldsymbol{\Lambda} + \frac{1}{2}\eta^2\boldsymbol{\Lambda}^2 + \frac{1}{3}\eta^3\boldsymbol{\Lambda}^3 + \cdots), \tag{12}$$

$$\begin{aligned}\log(\boldsymbol{\Lambda} + \alpha \boldsymbol{I})^{-1}\alpha &= -\log(\boldsymbol{\Lambda} + \alpha \boldsymbol{I}) + \log\alpha = -\log(\alpha(\boldsymbol{I} + \frac{1}{\alpha}\boldsymbol{\Lambda})) + \log\alpha \\ &= -\log\alpha - \log(\boldsymbol{I} + \frac{1}{\alpha}\boldsymbol{\Lambda}) + \log\alpha \approx \frac{-1}{\alpha}\boldsymbol{\Lambda} + \frac{1}{2\alpha^2}\boldsymbol{\Lambda}^2 - \frac{1}{3\alpha^3}\boldsymbol{\Lambda}^3 + \cdots.\end{aligned} \tag{13}$$

- Equating Eqs. (12) and (13) because of Eq. (11) gives us:

$$\alpha \approx \frac{1}{t\,\eta}, \quad t \approx \frac{1}{\alpha\,\eta}. \tag{14}$$

Early Stopping in Neural Networks

- We found:

$$\alpha \approx \frac{1}{t\,\eta}, \quad t \approx \frac{1}{\alpha\,\eta},$$

- which shows that the inverse of number of iterations is proportional to the weight decay (ℓ_2 norm) regularization parameter.
- In other words, the more training iterations we have, the less we are penalizing the weights and the more the network might get overfitted.
- Moreover, some empirical studies [14] show that noise injection and weight decay have more effectiveness than early stopping for avoiding overfitting, although early stopping has its own merits.

Dropout

Bagging

- **Bagging** is short for **Bootstrap AGGregatING**, first proposed by [15] (1996).
- It is a meta algorithm which can be used with any model (classifier, regression, etc).
- The definition of **bootstrapping** is as follows. Suppose we have a sample $\{\boldsymbol{x}_i\}_{i=1}^n$ with size n where $f(\boldsymbol{x})$ is the unknown distribution of the sample, i.e., $\boldsymbol{x}_i \overset{iid}{\sim} f(\boldsymbol{x})$. We would like to sample from this distribution but we do not know the $f(\boldsymbol{x})$. **Approximating the sampling from the distribution by randomly sampling from the available sample** is named bootstrapping. In bootstrapping, we use **simple random sampling with replacement**. The drawn sample is named **bootstrap sample**.
- In bagging, we draw k **bootstrap** samples each with some sample size. Then, we train the model h_j using the j-th bootstrap sample, $\forall j \in \{1, \ldots, k\}$. Hence, we have k trained models rather than one model. Finally, we **aggregate** the results of estimations of the k models for an instance $\boldsymbol{x}$:

$$\widehat{f}(\boldsymbol{x}) = \frac{1}{k} \sum_{j=1}^{k} h_j(\boldsymbol{x}). \tag{15}$$

- If the model is classifier, we should probably use sign function:

$$\widehat{f}(\boldsymbol{x}) = \text{sign}\big(\frac{1}{k} \sum_{j=1}^{k} h_j(\boldsymbol{x})\big). \tag{16}$$

Bagging

- Let e_j denote the error of the j-th model in estimation of the observation of an instance. Suppose this error is a random variable with normal distribution having mean zero, i.e., $e_j \overset{iid}{\sim} \mathcal{N}(0, s)$ where $s := \sigma^2$.
- We denote the covariance of estimations of two trained models using two different bootstrap samples by c.
- Therefore, we have:

$$\mathbb{E}(e_j^2) = s \implies \mathbb{V}\text{ar}(e_j) = \mathbb{E}(e_j^2) - (\mathbb{E}(e_j))^2 = s - 0 = s \implies \mathbb{V}\text{ar}(h_j(\boldsymbol{x})) = s, \tag{17}$$

$$\begin{aligned}\mathbb{E}(e_j\, e_\ell) = c \implies & \mathbb{C}\text{ov}(e_j, e_\ell) = \mathbb{E}(e_j\, e_\ell) - \mathbb{E}(e_j)\mathbb{E}(e_\ell) = c - (0 \times 0) = c \\ & \implies \mathbb{C}\text{ov}(h_j(\boldsymbol{x}), h_\ell(\boldsymbol{x})) = c,\end{aligned} \tag{18}$$

for all $j, \ell \in \{1, \dots, k\}, j \neq \ell$.

- According to Eqs. (15), (17), and (18), we have:

$$\begin{aligned}\mathbb{V}\text{ar}\big(\widehat{f}(\boldsymbol{x})\big) &= \frac{1}{k^2}\mathbb{V}\text{ar}\Big(\sum_{j=1}^{k} h_j(\boldsymbol{x})\Big) = \frac{1}{k^2}\sum_{j=1}^{k}\mathbb{V}\text{ar}(h_j(\boldsymbol{x})) + \frac{1}{k^2}\sum_{j=1}^{k}\sum_{\ell=1,\ell\neq j}^{k}\mathbb{C}\text{ov}(h_j(\boldsymbol{x}), h_\ell(\boldsymbol{x})) \\ &= \frac{1}{k^2}ks + \frac{1}{k^2}k(k-1)c = \frac{1}{k}s + \frac{k-1}{k}c.\end{aligned} \tag{19}$$

Bagging

- We found:

$$\mathbb{V}\text{ar}\big(\widehat{f}(\boldsymbol{x})\big) = \frac{1}{k}s + \frac{k-1}{k}c.$$

- The obtained expression has an interesting interpretation: If two trained models with two different bootstrap samples are very correlated, we will have $c \approx s$, thus:

$$\lim_{c \rightarrow s} \mathbb{V}\text{ar}\big(\widehat{f}(\boldsymbol{x})\big) = \frac{1}{k}s + \frac{k-1}{k}s = s, \tag{20}$$

  and if the two trained models are very different (uncorrelated), we will have $c \approx 0$, hence:

$$\lim_{c \rightarrow 0} \mathbb{V}\text{ar}\big(\widehat{f}(\boldsymbol{x})\big) = \frac{1}{k}s + \frac{k-1}{k}0 = \frac{1}{k}s. \tag{21}$$

- This means that if the trained models are **very correlated** in bagging, there is **not any difference from using only one model**; however, if we have **different** trained models, the **variance of estimation improves significantly by the factor of** k.
- This also implies that **bagging never is destructive**; it **either is not effective or improves** the estimation in terms of variance [16, 15].
- The more complex model usually has more variance and less bias. Therefore, the more variance corresponds to overfitting. As bagging helps **decrease the variance** of estimation, it helps **prevent overfitting**. Therefore, bagging is a meta algorithm useful to have less variance and not to get overfitted [17].
- Bagging can be seen as an **ensemble learning** method [18] which is useful because of **model averaging** [19, 20].

Dropout

- One of the examples of using bagging in machine learning is **random forest** [21].
- Another example of bagging is **dropout** in neural networks (2014) [22].
- According to dropout, in every iteration of **training** phase, the **neurons are randomly removed** with **probability** $p = 0.5$, i.e., we sample from a **Bernoulli distribution**.
- This makes the training phase as training different neural networks as we have different models in bagging.
- In the test time, **all the neurons** are used but their **output is multiplied by the** p. This imitates the **model averaging** of bagging in Eq. (15).
- That is why dropout prevents neural network from **overfitting**.
- Another intuition of why dropout works is making the neural network **sparse** which is very effective because of principal of sparsity [23, 24] or Occam's razor [25] introduced before.

Acknowledgment

- Some slides are based on our tutorial paper: "The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial" [26]
- Some slides are inspired by the textbook: Trevor Hastie, Robert Tibshirani, Jerome Friedman, "The elements of statistical learning: Data Mining, Inference, and Prediction", Springer, 2009 [23].
- Some slides are inspired by the textbook: Ian Goodfellow, Yoshua Bengio, Aaron Courville. "Deep learning". MIT press, 2016 [1].
- Another textbook suitable for sparsity in machine learning is: Robert Tibshirani, Martin Wainwright, Trevor Hastie, "Statistical learning with sparsity: the lasso and generalizations", Chapman and Hall/CRC, 2015 [24].
- Some slides of this slide deck are inspired by teachings of Prof. Ali Ghodsi (at University of Waterloo, Department of Statistics).

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*.
MIT press, 2016.

[2] A. Krogh and J. A. Hertz, “A simple weight decay can improve generalization,” in *Advances in neural information processing systems*, pp. 950–957, 1992.

[3] C.-T. Chiu, K. Mehrotra, C. K. Mohan, and S. Ranka, “Modifying training algorithms for improved fault tolerance,” in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN’94)*, vol. 1, pp. 333–338, IEEE, 1994.

[4] K. Matsuoka, “Noise injection into inputs in back-propagation learning,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 3, pp. 436–440, 1992.

[5] D. A. Van Dyk and X.-L. Meng, “The art of data augmentation,” *Journal of Computational and Graphical Statistics*, vol. 10, no. 1, pp. 1–50, 2001.

[6] T. DeVries and G. W. Taylor, “Dataset augmentation in feature space,” *arXiv preprint arXiv:1702.05538*, 2017.

[7] Y. Grandvalet, S. Canu, and S. Boucheron, “Noise injection: Theoretical prospects,” *Neural Computation*, vol. 9, no. 5, pp. 1093–1108, 1997.

[8] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, ACM, 2008.

References (cont.)

[9] K. Ho, C.-s. Leung, and J. Sum, “On weight-noise-injection training,” in *International Conference on Neural Information Processing*, pp. 919–926, Springer, 2008.

[10] L. Prechelt, “Early stopping-but when?,” in *Neural Networks: Tricks of the trade*, pp. 55–69, Springer, 1998.

[11] Y. Yao, L. Rosasco, and A. Caponnetto, “On early stopping in gradient descent learning,” *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, 2007.

[12] R. Caruana, S. Lawrence, and C. L. Giles, “Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping,” in *Advances in neural information processing systems*, pp. 402–408, 2001.

[13] S. Boyd and L. Vandenberghe, *Convex optimization*.
Cambridge university press, 2004.

[14] R. M. Zur, Y. Jiang, L. L. Pesce, and K. Drukker, “Noise injection for training artificial neural networks: A comparison with weight decay and early stopping,” *Medical physics*, vol. 36, no. 10, pp. 4810–4818, 2009.

[15] L. Breiman, “Bagging predictors,” *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[16] P. L. Bühlmann and B. Yu, “Explaining bagging,” in *Research report/Seminar für Statistik, Eidgenössische Technische Hochschule Zürich*, vol. 92, Seminar für Statistik, Eidgenössische Technische Hochschule (ETH), 2000.

References (cont.)

[17] L. Breiman, "Arcing classifier (with discussion and a rejoinder by the author)," *The annals of statistics*, vol. 26, no. 3, pp. 801–849, 1998.

[18] R. Polikar, "Ensemble learning," in *Ensemble machine learning*, pp. 1–34, Springer, 2012.

[19] J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky, "Bayesian model averaging: a tutorial," *Statistical science*, pp. 382–401, 1999.

[20] G. Claeskens and N. L. Hjort, *Model selection and model averaging*.
Cambridge Books, Cambridge University Press, 2008.

[21] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[23] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning: Data Mining, Inference, and Prediction*, vol. 2.
Springer series in statistics, New York, NY, USA, 2009.

[24] R. Tibshirani, M. Wainwright, and T. Hastie, *Statistical learning with sparsity: the lasso and generalizations*.
Chapman and Hall/CRC, 2015.

References (cont.)

[25] P. Domingos, “The role of Occam’s razor in knowledge discovery,” *Data mining and knowledge discovery*, vol. 3, no. 4, pp. 409–425, 1999.

[26] B. Ghojogh and M. Crowley, “The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial,” *arXiv preprint arXiv:1905.12787*, 2019.