AI and Deep Learning

3-2 Regularization

Zhonglei Wang

WISE, SOE, CHOW

XMU

(Version v1)

Contents

1. Explicit regularization

2. Implicit regularizations

3. Noise injection

4. Ensemble learning

Intuition

1. Fully connected neural network often involves too many model parameters
   - Complex models tend to overfit the training data
   - Decrease the generality of the trained model
   - Lead to possible disaster in practice
2. Definitely, we can consider a simpler neural network
   - Simpler models may underfit the training data, however
3. Thus, we may would like to obtain a kind of less complex model without changing the architecture

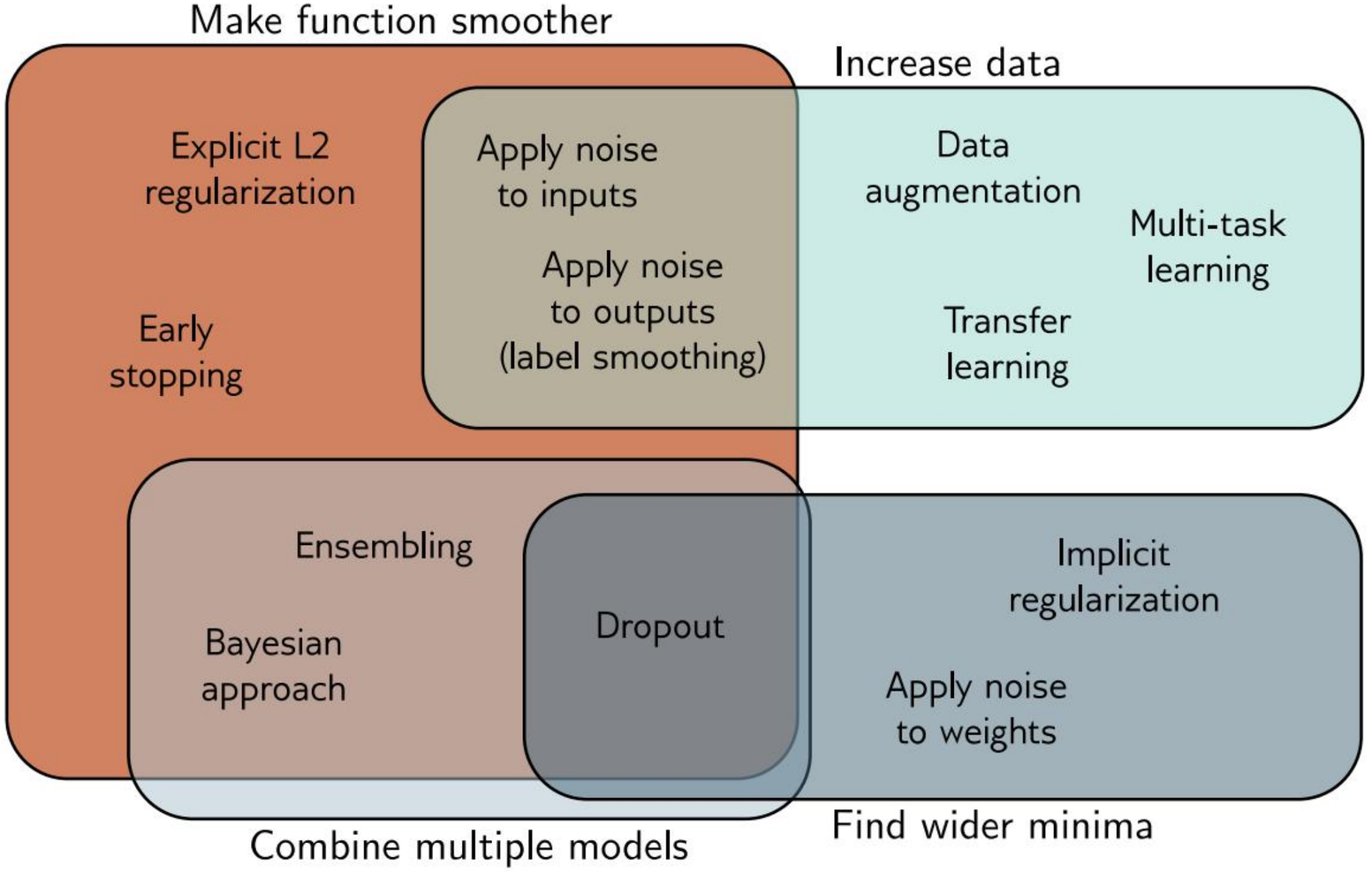
Intuition

1. For deep learning, regularization includes

- Explicit regularization (penalties)
- Implicit regularization (gradient descent algorithms, early stopping, dropout)
- Injection of noise (training examples or the learning procedures)
- Ensemble learning (bagging, boosting, stacking)

Intuition

1. The following is Figure 9.14 of Prince (2024)



Explicit regularization

1. Form:

$$\mathcal{J} = \mathcal{J}_1 + \mathcal{J}_2$$

- $\mathcal{J}_1 = \mathcal{J}_1(\boldsymbol{\theta})$: Original cost function, such as cross entropy
- $\mathcal{J}_2 = \mathcal{J}_2(\boldsymbol{\theta})$: Penalty on the model parameters, such as l_2 or l_1 penalty

2. To update model parameters, we need

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{J}_1}{\partial \boldsymbol{\theta}} + \frac{\partial \mathcal{J}_2}{\partial \boldsymbol{\theta}}$$

- The first term is obtained by backpropagation
- The second term is determined by the form of the penalty

Explicit regularization

1. l_2 penalty (ridge regression or Tikhonov regularization)

$$\mathcal{J}_2 = \frac{\lambda}{2n} \sum_{l=1}^{L} \| \boldsymbol{W}^{[l]} \|_F^2$$

- $\lambda > 0$: Hyperparameter
- $\| \boldsymbol{A} \|_F = (\sum_i \sum_j a_{ij}^2)^{1/2}$: Frobenius norm of a real-valued matrix $\boldsymbol{A} = (a_{ij})$
- Derivative

$$\frac{\partial \mathcal{J}_2}{\partial \boldsymbol{b}^{[l]}} = 0, \quad \frac{\partial \mathcal{J}_2}{\partial \boldsymbol{W}^{[l]}} = \frac{\lambda}{n} \boldsymbol{W}^{[l]} \quad (l = 1, \ldots, L)$$

Explicit regularization

1. Given a learning rate α,
   - Update for bias terms $\{\boldsymbol{b}^{[l]} : l = 1, \ldots, L\}$ remains the same
   - Update for weight terms $\{\boldsymbol{W}^{[l]} : l = 1, \ldots, L\}$ involves a new gradient part
2. Weight terms are updated by

$$\begin{aligned}
\boldsymbol{W}^{[l]} &\leftarrow \boldsymbol{W}^{[l]} - \alpha \cdot \left(\frac{\partial \mathcal{J}_1}{\partial \boldsymbol{W}^{[l]}} + \frac{\partial \mathcal{J}_2}{\partial \boldsymbol{W}^{[l]}} \right) \\
&= \boldsymbol{W}^{[l]} - \alpha \cdot \left(\frac{\partial \mathcal{J}_1}{\partial \boldsymbol{W}^{[l]}} + \frac{\lambda}{n} \boldsymbol{W}^{[l]} \right) \\
&= \left(1 - \frac{\alpha \cdot \lambda}{n} \right) \cdot \boldsymbol{W}^{[l]} - \alpha \cdot \frac{\partial \mathcal{J}_1}{\partial \boldsymbol{W}^{[l]}}
\end{aligned}$$

3. Weight decay regularization: Weight terms get updated AFTER shrinkage

Remarks on weight decay regularization

1. For standard GD, weight decay regularization is equivalent to l_2 regularization
2. For adaptive GD, such as Adam, they are different (Loshchilov and Hutter, 2019)
3. Consider the following task,
   - Achieve weight decay regularization
   - Use the idea of Adam to update model parameters
4. A straightforward idea is to consider
   - Consider a cost function: $\mathcal{J} = \mathcal{J}_1 + \mathcal{J}_2$
   - A standard Adam is used as the optimizor
5. However, the above idea does not work as we want

Remarks on weight decay regularization

1. Where does the problem come from?
   - Check the cost function more carefully: $\mathcal{J} = \mathcal{J}_1 + \mathcal{J}_2$
   - l_2 penalty leads to over-regularization on gradients than they should be
   - Result in suboptimal for adaptive gradient methods (Adam) in certain cases
2. Loshchilov and Hutter (2019) proposed AdamW (Adam with decoupled Weight decay) to solve this problem
   - To get AdamW, we only need to modify Adam slightly
   - AdamW is the default optimizor for many problems

AdamW

1. Goal: Achieve weight decay regularization during training
   - For Adam with l_2-regularization, consider a cost function: $\mathcal{J} = \mathcal{J}_1 + \mathcal{J}_2$
   - For AdamW, weight decay is implicitly implemented based on a cost function: $\mathcal{J}_1$
2. The following is Algorithm 2 of Loshchilov and Hutter (2019)

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2) \boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Explicit regularization

1. l_1 penalty term

$$\mathcal{J}_2 = \frac{\lambda}{n} \sum_{l=1}^{L} \|\boldsymbol{W}^{[l]}\|_1$$

- $\|\boldsymbol{A}\|_1 = \sum_i \sum_j |a_{ij}|$
- Derivative

$$\frac{\partial \mathcal{J}_2}{\partial \boldsymbol{b}^{[l]}} = 0; \quad \frac{\partial \mathcal{J}_2}{\partial \boldsymbol{W}^{[l]}} = \frac{\lambda}{n} \text{sign}(\boldsymbol{W}^{[l]})$$

- It is used to
  - Control the complexity of the model
  - Induce sparsity, which is similar to LASSO

Remarks

1. We "misuse" $\|\boldsymbol{A}\|_1$ to denote the summation of absolute values of elements in $\boldsymbol{A}$
   - More generally, $\|\boldsymbol{A}\|_1 = \max_j \sum_i |a_{ij}|$ denotes the maximum column summation of absolute elements
2. We only implement l_2 or l_1 penalty on the weight terms $\{\boldsymbol{W}^{[l]} : l = 1, \ldots, L\}$
   - We leave the bias terms $\{\boldsymbol{b}^{[l]} : l = 1, \ldots, L\}$ unpenalized
3. See Section 3.4 of Hastie et al. (2009) for a geometric intuition for l_2 and l_1 penalty

Implicit regularization

1. Consider the following gradient descent algorithm:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \nabla \mathcal{J}(\boldsymbol{\theta}^{(t)})$$

   - $\boldsymbol{\theta}^{(t)}$: Model parameter after the tth iteration
   - α : Learning rate
   - $\nabla \mathcal{J}(\boldsymbol{\theta}^{(t)}) = \partial \mathcal{J}(\boldsymbol{\theta}^{(t)}) / \partial \boldsymbol{\theta}$

2. We can treat $\boldsymbol{\theta}^{(t)}$ as a piece-wise linear function of t
3. Barrett and Dherin (2021) analyzed this implicit regularization using ideas from the ODE perspective

Implicit regularization

1. GD is similar to Runge-Kutta method for the initial value problem (IVP)

$$\dot{\boldsymbol{\theta}} = -\nabla \mathcal{J}(\boldsymbol{\theta}), \quad \boldsymbol{\theta}(0) = \boldsymbol{\theta}^{(0)}$$

   - We implicitly assume that $\boldsymbol{\theta} = \boldsymbol{\theta}(t)$ is a function of t
   - $\dot{\boldsymbol{\theta}} = \mathrm{d}\boldsymbol{\theta}/\mathrm{d}t, \nabla \mathcal{J}(\boldsymbol{\theta}) = \partial \mathcal{J}(\boldsymbol{\theta})/\partial \boldsymbol{\theta}$
   - $\boldsymbol{\theta}(0)$: Initial value for $t = 0$
   - $\boldsymbol{\theta}^{(0)}$: Initial value for the gradient descent algorithm
2. When solving the above IVP by Runge-Kutta method, we discretize t
3. Such discretization incurs approximation error
   - Gap between the true (continuous) solution and the numerical (discretized and piece-wise linear) approximation

Implicit regularization

1. Example: Consider the following cost function

$$\mathcal{J}(\boldsymbol{\theta}) = (\theta_1^2 + 1)\theta_2^2$$

- $\boldsymbol{\theta} = (\theta_1, \theta_2)^{\mathrm{T}}$
- Optimal solution: Horizontal line $\theta_2 = 0$
- $\nabla \mathcal{J}(\boldsymbol{\theta}) = (2\theta_1\theta_2^2, 2\theta_2 + 2\theta_1^2\theta_2)^{\mathrm{T}}$

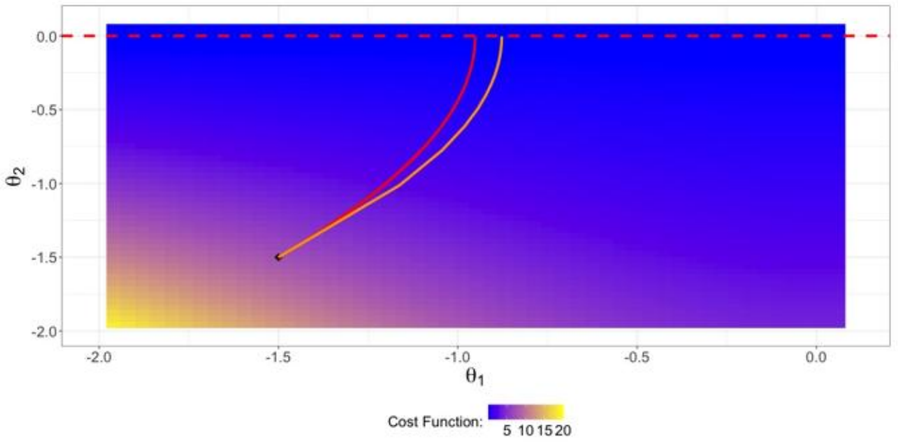
2. Consider the following IVP

$$\dot{\boldsymbol{\theta}} = -\nabla \mathcal{J}(\boldsymbol{\theta}) = \begin{pmatrix} 2\theta_1\theta_2^2 \\ 2\theta_2 + 2\theta_1^2\theta_2 \end{pmatrix}, \quad \boldsymbol{\theta}(0) = (-1.5, 1.5)$$

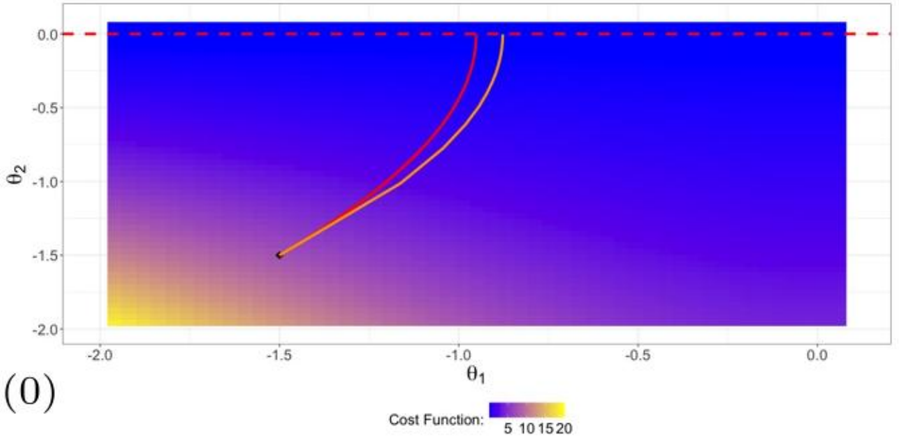
- Both θ_1 and θ_2 are functions of t

Implicit regularization

1. For gradient descent, set learning rate $\alpha = 0.05$
   - True solution
   - Numerical approximation (GD)



Implicit regularization



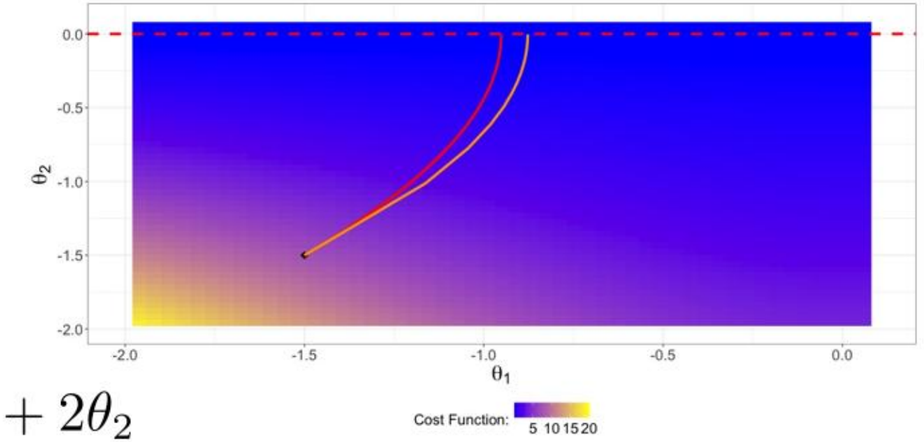
1. The numerical approximation is much closer to the true continuous solution to the following IVP

$$\dot{\boldsymbol{\theta}} = -\nabla \tilde{\mathcal{J}}(\boldsymbol{\theta}), \quad \boldsymbol{\theta}(0) = \boldsymbol{\theta}^{(0)}$$

- $\tilde{\mathcal{J}}(\boldsymbol{\theta}) = \mathcal{J}(\boldsymbol{\theta}) + \lambda \cdot R_{IG}(\boldsymbol{\theta})$
- $\lambda = \alpha \cdot p/4$
- α : Learning rate
- p : Dimensionality of $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_p)^{\mathrm{T}}$
- $R_{IG}(\boldsymbol{\theta}) = p^{-1} \sum_{i=1}^{p} (\partial \mathcal{J}(\boldsymbol{\theta}) / \partial \theta_i)^2$

2. $R_{IG}(\boldsymbol{\theta})$acts like an implicit regularization term for GD
3. For IGR, four predictions of Barrett and Dherin (2021) are:
   - IGR encourages smaller values of $R_{IG}(\boldsymbol{\theta})$ relative to $\mathcal{J}(\boldsymbol{\theta})$
   - IGR encourages the discovery of flatter optima
   - IGR encourages higher test accuracy
   - IGR encourages the discovery of optima that are more robust to parameter perturbations

Implicit regularization



1. Example: Consider the following cost function

$$\mathcal{J}(\boldsymbol{\theta}) = (\theta_1^2 + 1)\theta_2^2$$

- Recall that $\partial\mathcal{J}(\boldsymbol{\theta})/\partial\theta_1 = 2\theta_1\theta_2^2, \quad \partial\mathcal{J}(\boldsymbol{\theta})/\partial\theta_2 = 2\theta_1^2\theta_2 + 2\theta_2$
- The IGR term is

$$R_{IG}(\boldsymbol{\theta}) = \frac{1}{2}\left\{4\theta_1^2\theta_2^4 + (2\theta_1^2\theta_2 + 2\theta_2)\right\}$$

- For learning rate $\alpha = 0.05$, we have $\lambda = 0.05 \times 4/2 = 0.025$
- Denote $\tilde{\mathcal{J}}(\boldsymbol{\theta}) = \mathcal{J}(\boldsymbol{\theta}) + \lambda \cdot R_{IG}(\boldsymbol{\theta})$

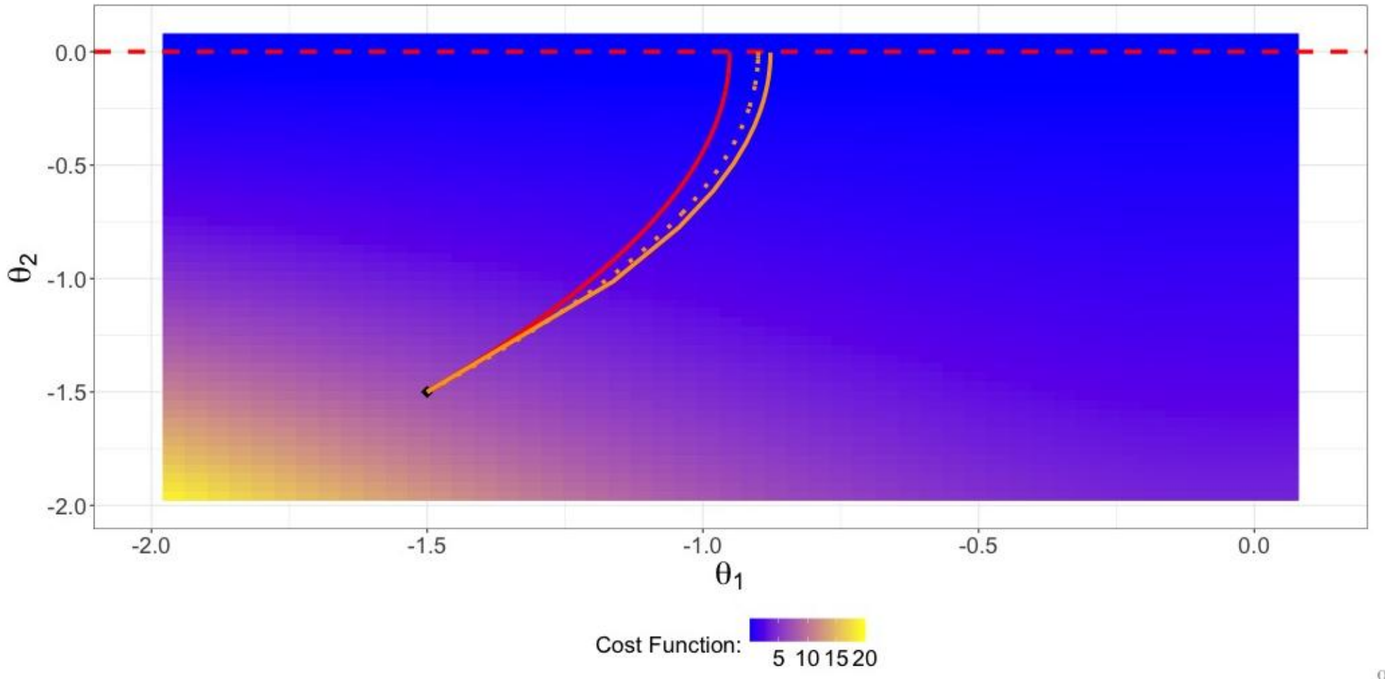
2. Consider the following IVP

$$\dot{\boldsymbol{\theta}} = -\nabla\tilde{\mathcal{J}}(\boldsymbol{\theta}), \quad \boldsymbol{\theta}(0) = (-1.5, 1.5)$$

Implicit regularization

1. For gradient descent, set learning rate $\alpha = 0.05$
   - True solution
   - Numerical approximation (GD)
   - True solution (dashed)

     for the ODE with IGR term



Early stopping

1. The performance on the training dataset improves as training procedure precedes
2. It may overfit the training dataset
   - The performance on the validation/test dataset may not always improve
3. Early stopping monitors the performance on the validataion set
   - Stop the training if the performance does not improve on the validation set

Early stopping

1. The trade-off between early stopping and double descent is between robustness and potential during model training
   - Model performance may exhibit a double descent phenomenon after sufficient iterations
   - Early stopping obtains a robust model that locks in gains promptly at the first risk valley
2. Use early stopping to obtain a robust baseline model
3. If resources permit, try increasing the number of training epochs to explore whether the double descent phenomenon exists

Dropout

1. Commonly used implicit regularization during the training step

   (Srivastava et al., 2014)

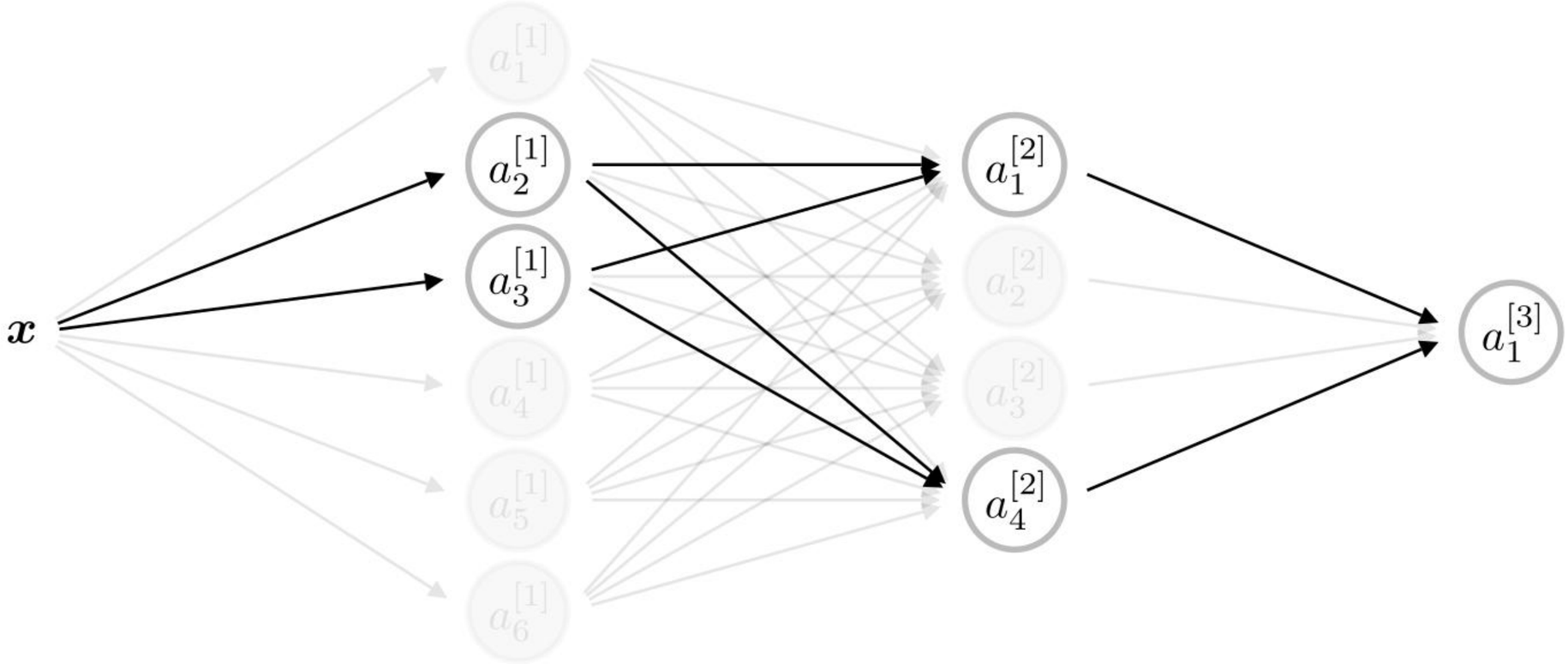
   - Randomly remove neurons in each hidden layer
   - If a neuron is removed, its activated value is set to be 0
   - Removal is done for each training example individually

2. It is also a regularization method of injecting noise
3. Improves the robustness

Dropout

1. For actiavted values of each training exampling in the lth layer ($l = 1, \ldots, L-1$)
   - Set the dropout rate to be $p^{[l]}$
   - Randomly assign them to be 0 using probability $p^{[l]}$
   - Need to rescale the remainings by $(1 - p^{[l]})^{-1}$ to compensate information loss
2. No dropout is applied for the input and output layers
3. No dropout is applied after the model is trained

Dropout


$\boldsymbol{x}$
$a_1^{[1]}$
$a_2^{[1]}$
$a_3^{[1]}$
$a_4^{[1]}$
$a_5^{[1]}$
$a_6^{[1]}$
$a_1^{[2]}$
$a_2^{[2]}$
$a_3^{[2]}$
$a_4^{[2]}$
$a_1^{[3]}$
Input layer
1st hidden layer
2nd hidden layer
Output layer

Dropout

1. Dropout is implemented using a mask matrix for each layer
2. Let $\boldsymbol{M}^{[l]} \in \{0,1\}^{n \times d^{[l]}}$ be a mask matrix
   - Each element of $\boldsymbol{M}^{[l]}$ is a Bernoulli random variable with success probability $1 - p^{[l]}$
3. Forward propagation

$$\boldsymbol{A}_d^{[l]} = \frac{\boldsymbol{A}^{[l]} \circ \boldsymbol{M}^{[l]}}{1 - p^{[l]}}.$$

4. Backpropagation: As usual
   - No additional model parameters are introduced for dropout

Dropout

1. Discussion

- In terms of operation, dropout randomly removes neurons during training
- In terms of implementation, dropout relies on mask matrices and scaling factors
- Essentially, dropout applies multiplier random noise to hidden layers
- Dropout suppresses co-adaptation, improves robustness, and reduces overfitting

Dropout

1. Dropout is multiplying Bernoulli random noise to the neural network
2. Generally, we can add or multiply other noise to the neural network
   - Add noise to the training data
   - Add noise to the weights
   - Perturb the labels

Noise injection

1. Noise injection is a population regularization for model training
   - Inputs (Matsuoka, 1992; Grandvalet et al., 1997)
   - Label smoothing (Müller et al., 2019; Lukasik et al., 2020)
   - Data augmentation (Shorten and Khoshgoftaar, 2019)
   - .....

Ensemble learning

1. To improve generalization of the model, we may consider multiple models
   - Reduce variance from a single overfitted model
   - Reduce bias from a single underfitted model

Ensemble learning

1. Commonly used ensemble learning methods can be generally classified as follows
   - Bagging primarily aims to reduce variance, by training the same model on different bootstrapped samples
     - Random forest
   - Boosting primarily aims to reduce bias, by sequentially using new models to correct mistakes from previous ones
     - AdaBoost, Gradient Boosted Decision Trees, XGBoost
   - Stacking aims to improve predictive accuracy by learning how to best combine a diverse set of strong models (Wolpert, 1992)
     - Meta learning

Summary

1. Regularization improves the model's generalization ability and reduce its accidental dependence on training samples
   - Explicit regularization: Add penalties to the cost function
   - Implicit regularization: Preference emerges from the training algorithm or training procedure
   - Noise injection: Intentionally introducing random perturbations during the training process
   - Ensemble learning: Making more stable combinations at the model level