Generalizaiton and Regularizaiton

Tianxiang (Adam) Gao

School of Computing
DePaul University

Outline

Recap: Optimization in Neural Networks

Training Process:

- MLP are **parameterized** function $f_{\boldsymbol{\theta}}$, where $\boldsymbol{\theta} = \{\boldsymbol{W}^{\ell}, \boldsymbol{b}^{\ell}\}$
- The training process involves solving an optimization problem with respect to $\boldsymbol{\theta}$:

$$\min_{\boldsymbol{\theta}} \quad \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

  where ℓ is a **loss** function and $\mathcal{S} := \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{\ell}$ is a **training set**.
- One commonly used method is called **gradient descent**:

$$\boldsymbol{\theta}^{+} = \boldsymbol{\theta} - \eta \nabla \mathcal{L}(\boldsymbol{\theta})$$

  where $\eta > 0$ is a **learning rate**.

Convergence Issues:

- **Small** η leads to slow convergence while **large** η cause oscillations or divergence.
- DNN loss landscapes are highly complex, exhibiting large and varying condition numbers κ
- Ill-conditioned loss landscapes cause **zig-zag patterns** in gradient descent.
- Unstable information propagation in DNNs can result in vanishing or exploding gradients.

Recap: Advanced Optimizers

Improving Optimizations:

- Averaging gradients (or with **momentum**) helps smooth the descent direction.
- A larger η is used in GD with momentum, but training also exhibits **damping** effects in the loss.
- Adaptive methods like RMSProp **rescale** gradients to maintain consistent update magnitudes.
- Adaptive optimizers provide an **adaptive learning rate** for each gradient coordinate.
- SGD with **mini-batch** improves computational efficiency by using small data subsets.

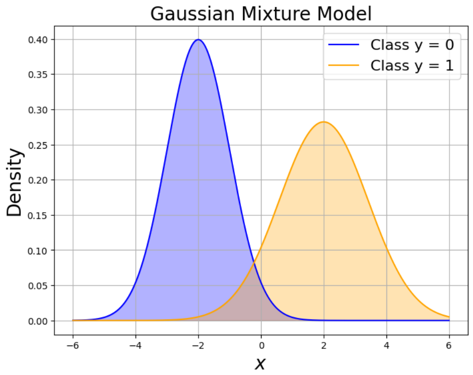
Questions

- What are common activation functions beyond sigmoid and ReLU?
- How should I choose learning rate, width, and depth for my network?
- Does gradient descent always converge? How can I speed up training?
- Does good training performance guarantee good test performance?

Outline

Gaussian Mixture Model

- Assume the output y follows a *discrete uniform distribution* over $\{0,1\}$, meaning $y \sim \mathcal{U}\{0,1\}$.
- For each value of y, the input x follows a *Gaussian distribution*:
  - When $y=0$, x follows $x|y=0 \sim \mathcal{N}(\mu_1, \sigma_1^2)$, *e.g.*, $\mu_1 = 1$ and $\sigma_1 = 1$
  - When $y=1$, x follows $x|y=1 \sim \mathcal{N}(\mu_2, \sigma_2^2)$, *e.g.*, $\mu_2 = 2$ and $\sigma_2 = 2$



- This setup defines a **(binary) Gaussian Mixture Model (GMM)**.
- Both x and y are random variables, with a joint distribution denoted as $\mathcal{D}$, i.e., $(x,y) \sim \mathcal{D}$.

Statistical Learning Theory (SLT)

- Assume the data (x, y) is drawn from an underlying joint distribution $\mathcal{D}$, i.e., $(x, y) \sim \mathcal{D}$.
- The goal of learning is to find a (parameterized) function f such that:

$$f(x) \approx y$$

  for "most" (x, y) pairs in a probabilistic sense.
- The **expected risk** of f is defined as:

$$R(f) := \mathbb{E}_{(x,y)\sim\mathcal{D}}[f(x) - y]^2,$$

  where we use the squared loss to measure the difference between $f(x)$ and y.
- In practice, the distribution $\mathcal{D}$ is **unknown**.
- Instead, we collect a **random training sample** $\mathcal{S} := \{(x_i, y_i)\}_{i=1}^n$ and compute the **empirical risk** or **training error**:

$$R_S(f) := \frac{1}{n}\sum_{i=1}^{n}[f(x_i) - y_i]^2.$$

- By the **law of large numbers**, we have:

$$R_S(f) \longrightarrow R(f) \quad \text{as} \quad n \to \infty.$$

Example of Expected and Empirical Risk using GMM

Suppose (x, y) follows GMM, and the function $f(x) = \theta x$, *i.e.*, *parameterized linear function*.

- The expected risk $R(f)$ is given by

$$\begin{aligned} R(f) =& \mathbb{E}_{(x,y)\sim\mathcal{D}}\ell(f(x), y) \\ =& \int [f(x) - y]^2 p(x, y) dx dy = \int [f(x) - y]^2 p(x|y)p(y) dx dy \\ =& \frac{1}{2}\int [f(x)]^2 p(x|y = 0)dx + \frac{1}{2}\int [f(x) - 1]^2 p(x|y = 1)dx \\ =& \frac{1}{2}\int [\theta x]^2 \cdot \mathcal{N}(x; \mu_1, \sigma_1^2)dx + \frac{1}{2}\int [\theta x - 1]^2 \cdot \mathcal{N}(x; \mu_2, \sigma_1^2)dx \\ \triangleq& R(\theta), \end{aligned}$$

where $p(x, y)$ is the joint density, and $\mathcal{N}(x; \mu, \sigma^2)$ is the Gaussian density defined by

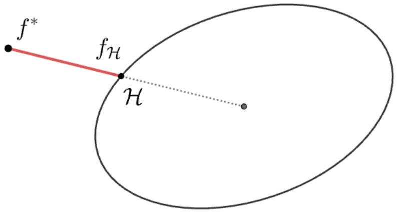
$$\mathcal{N}(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}.$$

- The empirical risk $R_S(f)$ over a training sample is given by

$$R_S(f) = \frac{1}{n}\sum_{i=1}^{n} [\theta x_i - y_i]^2 \triangleq R_S(\theta).$$

Hypothesis Class

In practice, we cannot evaluate all possible functions f. Instead, we restrict our search to a family of functions called a **hypothesis class** $\mathcal{H}$. Each function $h \in \mathcal{H}$ is called a **hypothesis**.



- The collection of all linear models or the collection of all two-layer neural networks:

$$\mathcal{H}_1 = \{h : h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}\}, \qquad \mathcal{H}_2 = \{h : h(\mathbf{x}) = \mathbf{v}^\top \phi(\mathbf{W}\mathbf{x})\}.$$

- A learning algorithm aims to find the best hypothesis $h \in \mathcal{H}$ that minimizes the **expected risk**:

$$f_{\mathcal{H}} := \operatorname*{argmin}_{f \in \mathcal{H}} R(f).$$

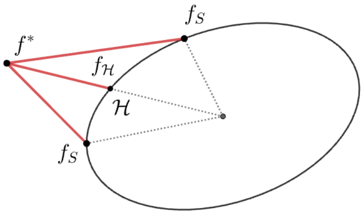
- The difference $\|f^* - f_{\mathcal{H}}\|$ is called the **approximation error**, where f^* is the ground true function.
- The **Universal Approximation Theorem (UAT)** implies $\|f^* - f_{\mathcal{H}}\| \approx 0$ if $\mathcal{H} = \mathcal{H}_2$.

Decomposition of Expected Risk

- Given a learned hypothesis f_S from a sample S, the expected risk of f_S can be decomposed as:

$$R(f_S) = \underbrace{R_S(f_S)}_{\textbf{Training Error}} + \underbrace{[R(f_S) - R_S(f_S)]}_{\textbf{Generalization Error}}.$$

- The **generalization error** is the difference between the expected risk and the empirical risk.



- In practice, the generalization error is estimated using the **test error** on an independent **test set**.

Bounding the Generalization Error

- The generalization error can be upper bounded by the **complexity** of the hypothesis class:

$$\sup_{h\in\mathcal{H}} |R(h) - R_S(h)| \le \text{Complexity Term},$$

where the "Complexity Term" quantifies how **flexible** or **complex** the hypothesis class $\mathcal{H}$ is.

- One commonly used complexity measure is the **(empirical) Rademacher complexity**:

$$\mathfrak{R}_S(\mathcal{H}) := \mathbb{E}_{\sigma_i\sim\mathcal{U}\{-1,1\}}\left[\min_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \ell\Big(h(x_i),\sigma_i\Big)\right],$$

where $\ell(h(x),\sigma) = \sigma h(x)$ and $\sigma_i \in \{-1,1\}$ are i.i.d. Rademacher random variables (uniformly distributed), *i.e.*, $\sigma \sim \mathcal{U}\{-1,1\}$, and the expectation is taken over these random labels.

- **Takeaway**: Rademacher complexity measures the ability of the hypothesis class to fit **random noise** (i.e., how well the hypothesis class can fit random labels).
- Using model complexity, we can derive the following generalization bound:

$$R(f_S) \le R_S(f_S) + \mathfrak{R}_S(\mathcal{H}) + \tilde{\mathcal{O}}(n^{-1}),$$

where the expected risk is upper-bounded by the **training error** and the **complexity of the model**.

Example: Complexity of Linear Models

Let $\mathcal{S} \subseteq \{\boldsymbol{x} : \|\boldsymbol{x}\| \leq R\}$ be a random sample, and consider $\mathcal{H}_1 := \{h : h(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x},\ \|\boldsymbol{w}\| \leq \Lambda\}$.

- The (empirical) Rademacher complexity $\mathfrak{R}_S(\mathcal{H}_1)$ is given by

$$\begin{aligned}\mathfrak{R}_S(\mathcal{H}_1) =& \mathbb{E}_{\sigma_i}\left[\min_{h\in\mathcal{H}_1} \frac{1}{n}\sum_{i=1}^{n} \ell(h(x_i), \sigma_i)\right] = \mathbb{E}_{\sigma_i}\left[\min_{\|\boldsymbol{w}\|\leq\lambda} \frac{1}{n}\sum_{i=1}^{n} \sigma_i \boldsymbol{w}^\top \boldsymbol{x}_i\right] \\ \leq& \frac{\Lambda}{n}\mathbb{E}_{\sigma_i}\left[\left\|\sum_{i=1}^{n}\sigma_i\boldsymbol{x}_i\right\|\right] \leq \frac{\Lambda}{n}\left[\mathbb{E}_{\sigma_i}\left\|\sum_{i=1}^{n}\sigma_i\boldsymbol{x}_i\right\|^2\right]^{1/2} \\ \leq& \frac{\Lambda}{n}\sqrt{nR^2} = \sqrt{\frac{R^2\Lambda^2}{n}},\end{aligned}$$

  where we use the Cauchy-Schwartz and Jensen's inequalities.

- As a result, the generalization error for linear models satisfies (with high probability):

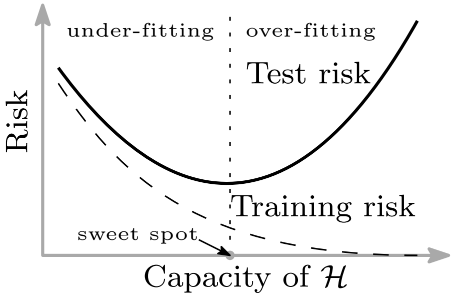
$$R(h_S) \leq R_S(h_S) + \sqrt{\frac{R^2\Lambda^2}{n}} + \tilde{\mathcal{O}}(n^{-1}).$$

- More data improves the empirical risk R_S as an **approximation** of the expected risk R, reducing overfitting, but overall performance still depends on minimizing R_S.

Model Complexity Trade-Off

The expected risk $R(f_S)$ is upper-bounded by the training error and the model complexity:

$$R(f_S) \le R_S(f_S) + \mathfrak{R}_S(\mathcal{H}) + \tilde{\mathcal{O}}(n^{-1}).$$



Key Insights on Generalization Bound

- If the model is too simple, it may fail to fit the training data well. This is known as **underfitting**.
- Conversely, if the model is highly flexible, it may achieve low training error, but perform poorly on unseen data. This is known as **overfitting**.
- The goal is to find a "**sweet spot**" balancing underfitting and overfitting to minimize the overall expected risk.

Optimal Hypothesis f^*

Claim: $f^*(x) = \mathbb{E}[y|x]$ is the **optimal hypothesis** that minimizes the expected risk.

Proof.

For any function f, we can decompose the expected risk as follows:

$$
\begin{aligned}
R(f) =& \mathbb{E}(f-y)^2 = \mathbb{E}(f-f^*+f^*-y)^2 \\
=& \mathbb{E}(f-f^*)^2 + 2\mathbb{E}(f-f^*)(f^*-y) + \mathbb{E}(f^*-y)^2 \\
=& \mathbb{E}(f-f^*)^2 + \mathbb{E}(f^*-y)^2 \\
\geq& \mathbb{E}(f^*-y)^2 \\
=& R(f^*)
\end{aligned}
$$

where the cross term $\mathbb{E}(f-f^*)(f^*-y) = 0$, because $f^*(x) = \mathbb{E}[y|x]$. □

- This is another **existence** result.
- The optimal hypothesis f^* is not directly accessible unless we know the joint distribution $\mathcal{D}$.
- Generally, we may have $R(f^*) \neq 0$. For example, consider $y = \theta x + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$

$$
\begin{aligned}
f^*(x) =& \mathbb{E}[y|x] = \mathbb{E}[\theta x + \varepsilon | x] = \theta x \\
R(f^*) =& \mathbb{E}_x[f^*(x) - y]^2 = \mathbb{E}_x[\theta x - (\theta x + \varepsilon)]^2 = \sigma^2 \quad \Longrightarrow \quad \textbf{irreducible error.}
\end{aligned}
$$

Bias-Variance Decomposition of Expected Risk

- The learned function f_S depends on the random sample S, making f_S a **random variable**.
- Hence, the expected risk $R(f_S)$ is also **random**, and it varies across different random samples S.
- To capture this variability, we consider the expectation of the $R(f_S)$ over all possible samples S, i.e., $\mathbb{E}_S[R(f_S)]$.
- Let $\bar{f} := \mathbb{E}_S[f_S]$, the **expected** or average hypothesis over all random samples S.
- Using $\bar{f}$, we can decompose $\mathbb{E}_S[R(f_S)]$ as follows:

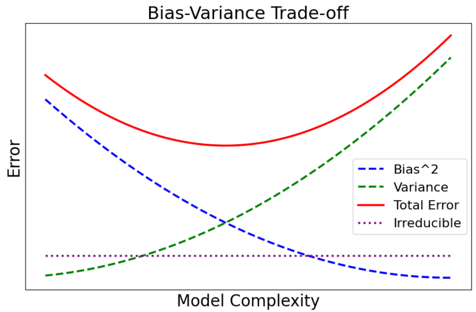
$$\begin{aligned}
\mathbb{E}_S[R(f_S)] =& \mathbb{E}_S\mathbb{E}_{(x,y)\sim\mathcal{D}}[f_S(x) - y]^2 \\
=& \mathbb{E}_S\mathbb{E}_{\mathcal{D}}[f_S - f^*)]^2 + R(f^*) \\
=& \mathbb{E}_S\mathbb{E}_{\mathcal{D}}\left[f_S - \bar{f} + \bar{f} - f^*\right]^2 + R(f^*) \\
=& \mathbb{E}_S\mathbb{E}_{\mathcal{D}}\left[(f_S - \bar{f})^2 + (\bar{f} - f^*)^2\right] + R(f^*) \\
=& \underbrace{\mathbb{E}_S(f_S - \bar{f})^2}_{\textbf{Variance term}} + \underbrace{\mathbb{E}_{\mathcal{D}}(\bar{f} - f^*)^2}_{\textbf{Bias term}} + \underbrace{R(f^*)}_{\textbf{irreducible}}
\end{aligned}$$

where the cross term $\mathbb{E}_{S,\mathcal{D}}(f_S - \mathbb{E}_S[f_S])(\mathbb{E}_S(f_S) - f^*) = 0$ cancels out.

Bias-Variance Trade-Off

The expected risk $\mathbb{E}_S[R(f_S)]$ can be broken down into three parts:

- **Squared Bias**: $\mathbb{E}_{\mathcal{D}}[(f^* - \bar{f})^2]$ measures the error from approximating the optimal function f^* with the learned model f_S. It reflects the error caused by using a simple model that cannot capture all the data patterns.
- **Variance**: $\mathrm{Var}(f_S) = \mathbb{E}_S[(f_S - \bar{f})^2]$ measures how much the learned function f_S varies with different training samples. It represents the error due to the model's sensitivity to fluctuations in the random training sample S.
- **Irreducible Error**: $R(f^*)$ represents the inherent noise in the data, which no model can eliminate. It is the error we cannot reduce.



- **High bias, low variance**: Simple models (e.g., linear models) have *low variance* since they are less sensitive to training data, but have *high bias* because they are too simple to capture all patterns in the data.
- **Low bias, high variance**: Complex models (e.g., polynomial model) have *low bias* as they can model complex relations, but *high variance* due to overfitting to the training data.

Summary of Statistical Learning Theory

- The goal is to find a hypothesis f within a hypothesis class $\mathcal{H}$ that minimizes the **expected risk**:

$$R(f) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[(f(x) - y)^2\right].$$

- Since the underlying distribution $\mathcal{D}$ is **unknown**, we approximate f by minimizing the **empirical risk** based on a **random** training sample S:

$$R_S(f) = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2.$$

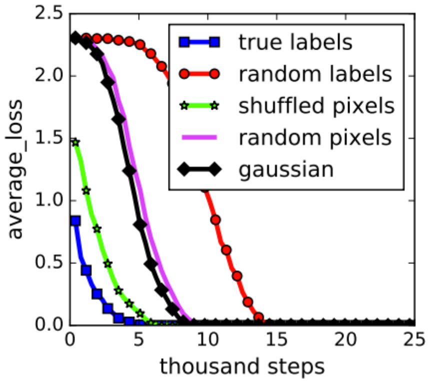
- Using model complexity $\mathfrak{R}_S(\mathcal{H})$, the expected risk is upper bound as:

$$R(f_S) \leq R_S(f_S) + \mathfrak{R}_S(\mathcal{H}) + \tilde{\mathcal{O}}(n^{-1}),$$

- By considering variations across different random training samples S, the expected risk $\mathbb{E}_S[R(f_S)]$ can be decomposed into three components: **bias**, **variance**, and **irreducible error**:
  - **High bias, low variance**: Simple models **underfit** and miss important patterns in the data.
  - **Low bias, high variance**: Complex models **overfit** and perform poorly on unseen data.
- Find the "sweet spot" between underfitting and overfitting to minimize the overall expected risk.

Outline

DNNs Can Fit Random Labels and Random Data



- **Label corruption**: Replace true label with *random label*
- **Shuffled pixels**: The pixels of each image are rearranged using a *fixed* random permutation
- **Random pixels**: Each image has a *unique* random arrangement of pixels).
- **Gaussian**: The pixels in images are replaced with random Gaussian *noise*.
- **Average loss**: *Training error* using the cross-entropy loss

Key Observation

DNNs can perfectly fit random labels or data, achieving **zero training error** even on completely unstructured inputs.

Zhang et al. "Understanding deep learning requires rethinking generalization" ICLR 2017.

Outline

Weight Decay

Weight Decay

- Regularization typically involves adding an extra term, called the **regularizer**, to the training loss:

$$\mathcal{L}_\lambda(\boldsymbol{\theta}) := \mathcal{L}(\boldsymbol{\theta}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2,$$

  where $\lambda > 0$ is the **regularization hyperparameter**, and $\|\cdot\|$ is the Euclidean norm.

- In deep learning, this regularization is known as **weight decay** because gradient descent on the regularized loss automatically shrinks (or decays) parameter $\boldsymbol{\theta}$ by the factor $(1-\eta\lambda)$:

$$\begin{aligned}\boldsymbol{\theta}^+ &= \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}_\lambda(\boldsymbol{\theta}) = \boldsymbol{\theta} - \eta\left[\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) + \lambda\boldsymbol{\theta}\right] \\ &= \underbrace{(1-\eta\lambda)}_{\textbf{decaying weights}}\ \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}).\end{aligned}$$

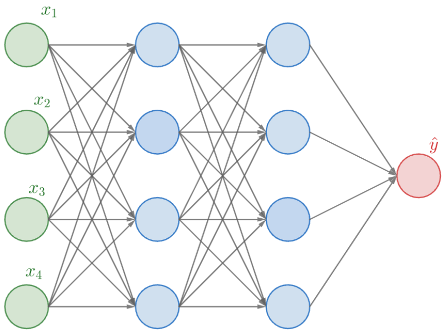
- However, $\boldsymbol{\theta}$ does **not** shrink to zero, as it must maintain a certain value to minimize the cost $\mathcal{L}(\boldsymbol{\theta})$.

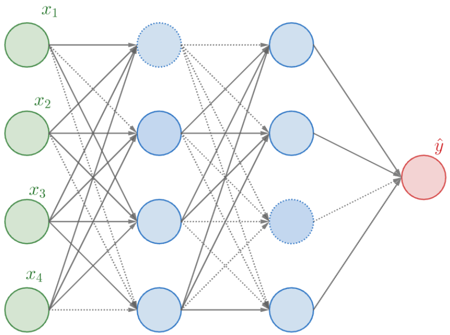
Interpretation: Sparsity

- The regularized optimization can be reformulated as:

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}), \quad \textbf{s.t.} \quad \|\boldsymbol{\theta}\| \leq C_\lambda,$$

  where $C_\lambda > 0$ is a constant that depends on λ.
- In deep learning, $\boldsymbol{\theta}$ is called **sparse** if most parameters are zero or close to zero (i.e., $\boldsymbol{\theta}_i \approx 0$).
- Sparse $\boldsymbol{\theta}$ reduces the flexibility and complexity of the DNN, leading to a **simpler** model.



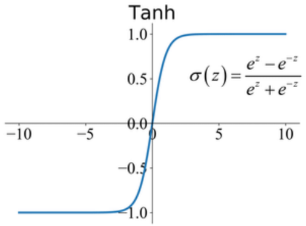


Interpretation: Linearity

Consider a simple two-layer neural network:

$$f_{\boldsymbol{\theta}}(x) = \sum_{i=1}^{n} v_i \phi(w_i x),$$

where $x \in \mathbb{R}$ is a scalar and $\phi(\cdot)$ is $\tanh$.



- When $w_i \approx 0$, then $w_i x \approx 0$, and the network operates near the **linear** region of $\tanh$:

$$v_i \phi(w_i x) \approx v_i (w_i x) \approx (v_i w_i) x = u_i x \quad \Longrightarrow \quad \textbf{a linear model},$$

  where $u_i := v_i w_i$.

- If $v_i \approx 0$, then

$$v_i \phi(w_i x) \approx 0,$$

  indicating **fewer** neurons are used.

Outline

Dropout

Dropout Regularization

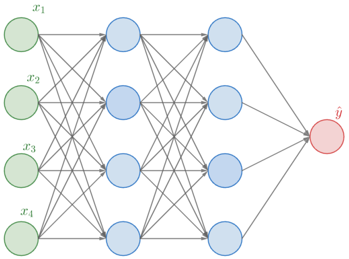
Recall the forward propagation:

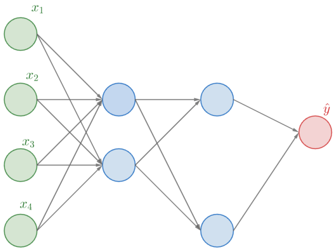
$$\boldsymbol{z}^{\ell} = \boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1}, \quad \boldsymbol{x}^{\ell} = \phi(\boldsymbol{z}^{\ell}).$$

- During **training**, each neuron is randomly **dropped** with probability p (a **hyperparameter**):

$$\boldsymbol{z}^{\ell} = \boldsymbol{W}^{\ell}\left(\boldsymbol{r}^{\ell} \odot \boldsymbol{x}^{\ell-1}\right), \quad \boldsymbol{x}^{\ell} = \phi(\boldsymbol{z}^{\ell}),$$

where $\boldsymbol{r}_i^{\ell} \overset{i.i.d.}{\sim}$ Bernoulli(p) and $\odot$ is element-wise product.



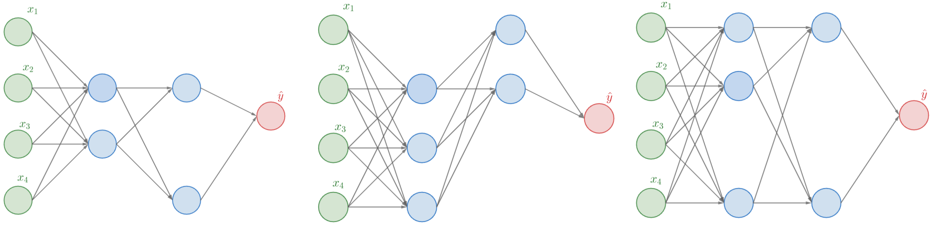


- The gradient update applies only to a **thinned subbnet** of the network.
- At **test time**, dropout is **turned off**, and weights are scaled by p to respect the dropout probability:

$$\boldsymbol{z}^{\ell} = p\boldsymbol{W}^{\ell}\boldsymbol{x}^{\ell-1}.$$

Interpretation: Implicit Ensemble Learning

- By randomly dropping units, a different **thinned subnet** is trained at each gradient descent step.
- With n neurons in the full network,
- At test time, the output is an **ensemble** prediction, aggregating the contributions of all subnets.



Key Insight

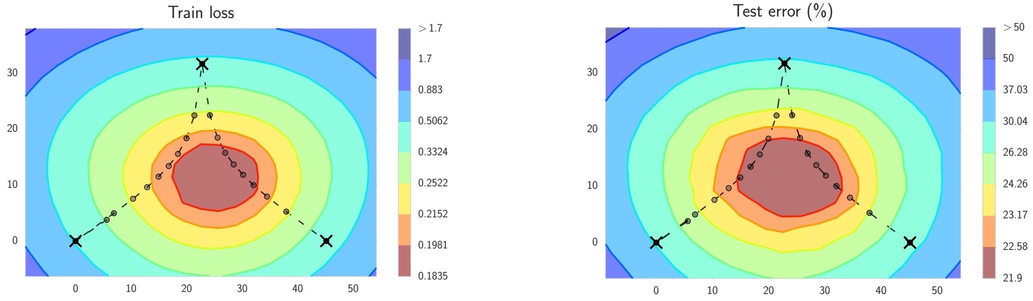
Dropout ensures that **no** single neuron or small group of neurons can dominate the prediction. By **spreading** the responsibility across all units, it improves model robustness to the input change and prevents overfitting.

Outline

Stochastic Weight Averaging

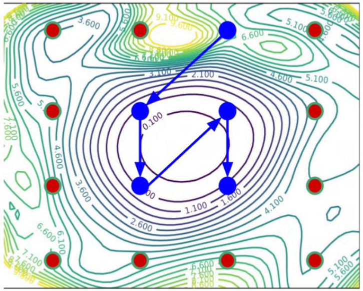
Trajectories of SGD

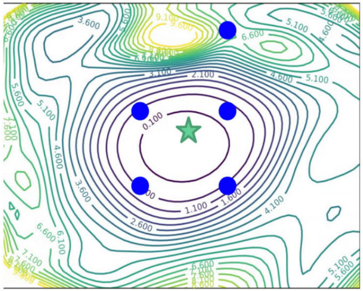
Let us continue to run SGD from a well-trained model and visualize the trajectory



- SGD oscillates around the **periphery** of high-performing solutions, and averaging SGD iterates improves test performance.
- SGD trajectories resemble a high-dimensional Gaussian-like distribution, with most of the mass concentrated in a **thin shell**.

Averaging Weights for Better Performance


Low-precision SGD


Compute Weight Average

- Averaging SGD iterations leads to improved generalization:

$$\bar{\boldsymbol{w}} = \frac{1}{k}\sum_{i=1}^{k} \boldsymbol{w}^i$$

Averaging Weights for Better Performance

- Averaging weights approximates **ensembling** predictions via linearization (if the weights are close):

$$\frac{1}{k}\sum_{i=1}^{k} f(\boldsymbol{w}^i) \approx f\left(\frac{1}{k}\sum_{i=1}^{k} \boldsymbol{w}^i\right) = f(\bar{\boldsymbol{w}})$$

- Moving average formulation:

$$\begin{aligned} \boldsymbol{w}^{k+1} &= \boldsymbol{w}^k - \eta \nabla \mathcal{L}(\boldsymbol{w}^k) \\ \boldsymbol{w}_{\mathsf{swa}}^{k+1} &= (1-\beta^k)\boldsymbol{w}_{\mathsf{swa}}^k + \beta^k \boldsymbol{w}^{k+1} \end{aligned}$$

where $\beta^k = \frac{k}{k+1}$ or $\beta^k = \beta \in (0,1)$.

Summary

- DNNs can fit random labels and data, achieving **zero training error**.
- Weight decay controls large weights, promoting **sparsity**, **linearity**, and **stability**.
- During training, dropout randomly drops units, effectively training an **exponential number** of **thinned subnets** simultaneously.
- At test time, the output is an **ensemble** prediction, aggregating contributions from all subnets.
- SGD oscillates near the **boundary** of local minima, while SWA finds a **centralized** solution in a flatter region.
- SWA approximates **ensemble predictions** through linearization.

Outline

Recap: Hyperparameters in Neural Networks

The training process involves several key hyperparameters:

- **Loss Function** $\ell(\cdot, \cdot)$: Square loss, cross-entropy loss, hinge loss
- **Activation Function** $\phi(\cdot)$: Step, sigmoid, ReLU, tanh, GELU
- **Optimizer**: SGD, Momentum, RMSProp, Adam, AdamW
- **Learning Rate** (η), **Batch Size** (b), **Epochs**
- **Network Type**: MLPs, CNNs, RNNs, Transformers, GNNs
- **Width and Depth**
- **Layers**: Normalization, pooling, dropout, softmax
- **Otherwise**: Initialization (Xavier, He), ℓ_2-regularization, gradient clipping, early stop

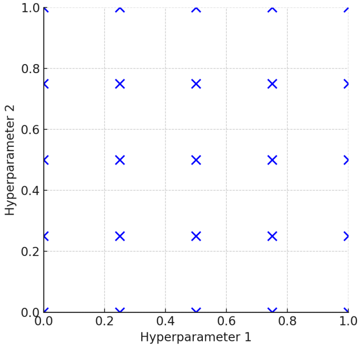
Key Difference: Hyperparameters vs. Trainable Parameters

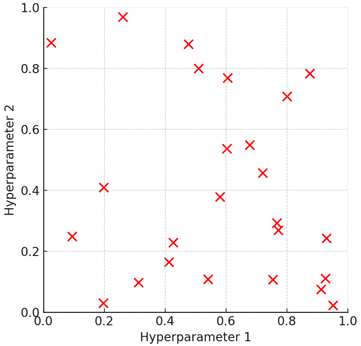
- Hyperparameters are **not trainable**. Unlike weights and biases, they need to be **tuned**.
- Proper tuning is essential for faster convergence during training and achieving good **generalization** performance.

Validation Set

- Split the dataset into three parts: **training set**, **validation set**, and **test set**.
- Build the model using the *training set*.
- Optimize or tune hyperparameters on the *validation set*.
- After tuning, **evaluate** the final model on the test set.
- Suggested split ratios:
  - For datasets between 100 and 1,000,000 samples: 60/20/20.
  - For datasets larger than 1,000,000 samples: 98/1/1.
- Ensure the validation and test sets come from the **same** distribution.
  - Example: Training and validation images from the web, but test images from user cell phones can cause a mismatch.

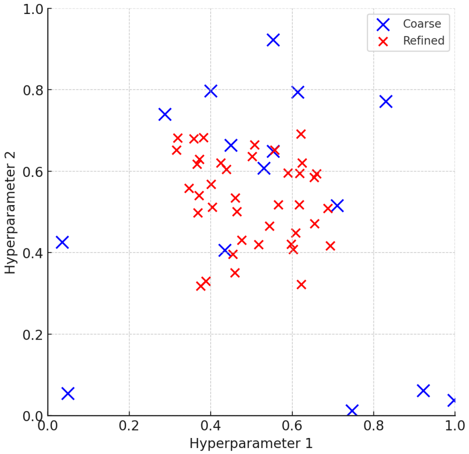
Tuning Process: Grid and Random Search





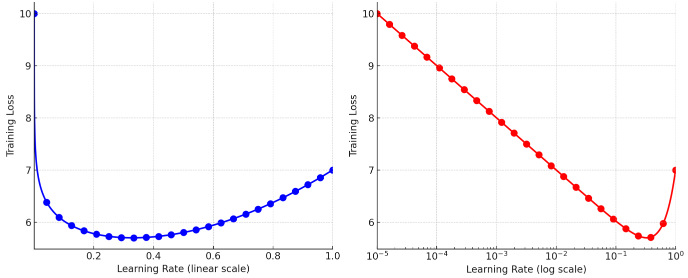
- **Grid Search**: Systematically explores a predefined set of hyperparameters; comprehensive but expensive
- **Random Search**: Randomly sample hyperparameters; more efficient than grid search when some hyperparameters are less important.

Tuning Process: Coarse and Refine



- Start with **coarse** tuning, then **refine** gradually.

Tuning Process: Log Scale



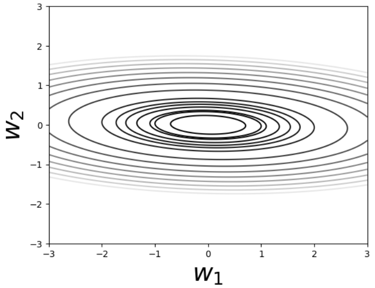
- Use **log scale** for hyperparameter search when appropriate, *e.g.*, learning rate η and smoothing factors β
- **Hyperband/Successive Halving**: Dynamically allocate resources and discard poor configurations early, ideal for deep networks with long training times.
- Leverage **parallelization** to run multiple experiments simultaneously to accelerate the search.

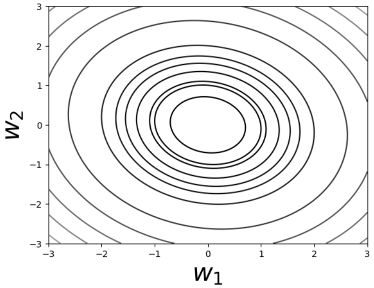
Input Normalization

- Normalize the inputs using **training set**:

$$\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i, \quad \bar{\boldsymbol{x}}_i = \boldsymbol{x}_i - \boldsymbol{\mu}, \quad \boldsymbol{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n} \bar{\boldsymbol{x}}_i^2, \quad \hat{\boldsymbol{x}}_i = \bar{\boldsymbol{x}}_i / \boldsymbol{\sigma},$$

  where all operations are taken element-wise.
- Consider a binary classification problem using linear model: $f_{\boldsymbol{\theta}}(x) = w_1 x_1 + w_2 x_2$
  - if $x_1 = \mathcal{O}(100)$ and $x_2 = \mathcal{O}(1)$, to have output $f_{\boldsymbol{\theta}} = \mathcal{O}(1)$, we must have $w_1 = \mathcal{O}\left(\frac{1}{100}\right)$ and $w_2 = \mathcal{O}(1)$.
  - After normalization, $\bar{x}_1 = \mathcal{O}(1)$ and $\bar{x}_2 = \mathcal{O}(1)$, so we have $w_1 = \mathcal{O}(1)$ and $w_2 = \mathcal{O}(1)$.





- At test time, apply $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ from **training** to test set.

Learning Rate Decay

- Recall that an **epoch** k is one pass through all **mini-batches** in SGD
- Instead of using a fixed learning rate, one can consider using **learning rate decay**

$$\eta_k = \frac{\eta}{k}, \qquad \eta_k = \frac{\eta}{\sqrt{k}}, \qquad \eta_k = (0.95)^k \eta$$

Bag of Tips

Learning Rate η:

- **Log-scale search**: $10^{-5} \sim 10^{-1}$.
- Learning rate schedules: Linearly **warm up**, then decay periodically for smooth convergence.
- Early stopping: Monitor loss curves to detect divergence.

Batch Size b:

- Small batches (*e.g.*, $16 \sim 128$) generalize better, but noisy gradient.
- Large batches (*e.g.*, $256 \sim 4096$) converge faster but may require higher learning rates.
- Rule of Thumb: Use the **largest** batch size that fits in memory, then tune; $\eta' = \eta \times \frac{b'}{b}$.

Weight Decay:

- **Log-scale search**: $10^{-5} \sim 10^{-3}$.
- **For Adam**: Use **AdamW** instead of standard weight decay.

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\boldsymbol{v}}{\sqrt{\boldsymbol{s} + \boldsymbol{\epsilon}}} - \eta\lambda\boldsymbol{w} \quad \Longrightarrow \quad \boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\boldsymbol{v}}{\sqrt{\boldsymbol{s} + \boldsymbol{\epsilon}}} - \lambda\boldsymbol{w}$$

  where weight decay is scaled by the small η in Adam, reducing the regularization effect.
- If validation loss diverges while training loss improves, increase weight decay.

Dropout:

- Start with $0.2 \sim 0.5$ for input layers, $0.5 \sim 0.8$ for hidden layers.
- Combine dropout with ℓ_2-regularization but avoid using it with Batch normalization.

Optimizers:

- **SGD+Momentum**: More stable than vanilla SGD.
- **Adam** works well for most tasks with default values $\beta_1 = 0.9$ and $\beta_2 = 0.999$
- Use **AdamW** for better weight decay handling.
- **RMSProp**: Useful for RNNs and reinforcement learning.

Network Architecture (Depth and Width):

- Start simple and gradually increase the complexity
- More layers (**depth**) improve feature extraction, using skip connections if too deep
- More neurons (**width**) increase capacity and stabilize training

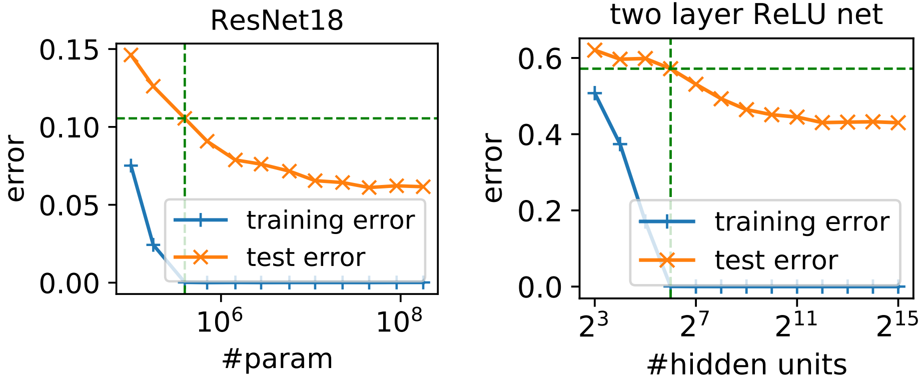
Activation Functions:

- **ReLU**: Standard choice for DNNs.
- **Leaky ReLU**: Fixes dying ReLU problem ($\alpha = 0.01$).
- **GELU**: Used in Transformers.
- **Swish**: Works well in CNNs.

Outline

Overparameterization

- A deep neural network (DNN) is said to be **overparameterized** when the number of neurons or parameters is much larger than the number of training samples.
- This might seem counterintuitive, but it has been found to be surprisingly beneficial in practice.



Behnam, et al. "Towards Understanding the Role of Over-Parametrization in Generalization of Neural Networks," ICLR 2019.

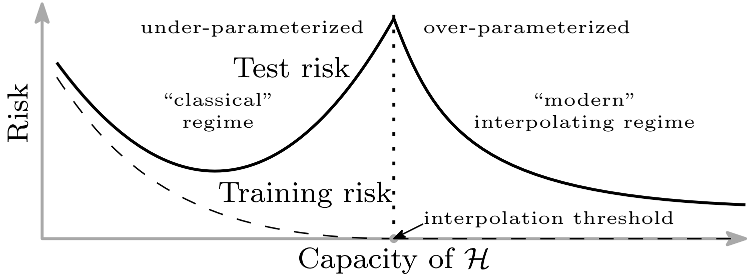
Double Descent

Overparameterized neural networks can **perfectly fit** or **interpolate** the training data.

- Mathematically, there exists a set of parameters $\boldsymbol{\theta}$ such that

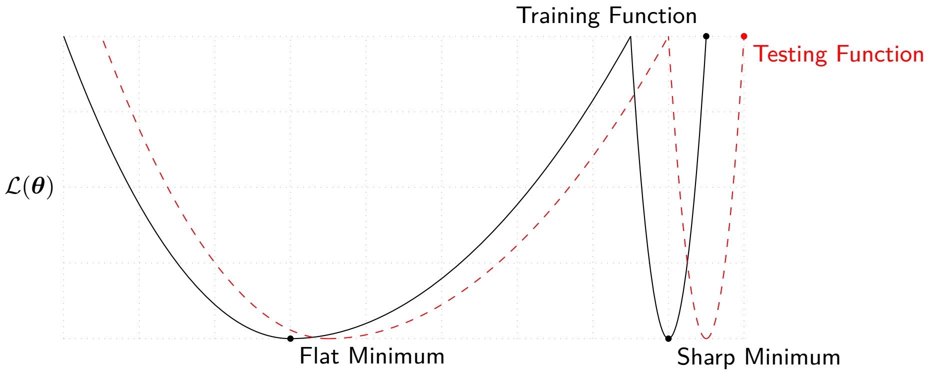
$$f_{\boldsymbol{\theta}}(x_i) = y_i, \quad \forall i \in [n]. \tag{1}$$

- Overparameterization implies there are **infinitely** many interpolation solutions.
- Some interpolation solutions generalize much better than those in the *underparameterized* regime. This phenomenon is called **double descent**.



Implicit Regularization

- It is important to understand that different global minima lead to varying **test** performances.
- A **flat** minimum typically results in better generalization than a **sharp** minimum.
- Different optimizers may converge to different minima, each with different generalization outcomes. This is known as **implicit regularization**.
- Thus, even if your current optimizer achieves low training error, tuning or adjusting it may still be necessary to achieve better test performance.


Training Function
Testing Function
$\mathcal{L}(\boldsymbol{\theta})$
Flat Minimum
Sharp Minimum