MATH5836: Data and Machine Learning

Week 9: Emerging Topics in Machine Learning

Sarat Moka *UNSW, Sydney*

Key Topics

9.1 Double-Descent

Modern deep networks often operate in a heavily overparameterized regime—usually with far more parameters than training examples. Classical bias–variance theory predicts a U-shaped test-error curve as model complexity grows, but recent work has uncovered a "double-descent" phenomenon: after peaking at the interpolation threshold, the test error descends again as complexity increases further. In this section we present a theoretical sketch of double-descent in a simple linear model, discuss interpolation and minimum-norm solutions, and highlight related phenomena such as benign overfitting and implicit regularization.

9.1.1 Classical Risk Decomposition vs. Modern Double-Descent

- Let $\hat{f}$ be a predictor drawn from a hypothesis class of "size" (complexity) C. The classical bias–variance decomposition writes the expected test risk as

$$\mathcal{R}(C) = \underbrace{\mathbb{E}_x\left[(\mathbb{E}[\hat{f}(x)] - f^*(x))^2\right]}_{\text{Bias}^2} + \underbrace{\mathbb{E}_x\left[\mathrm{Var}[\hat{f}(x)]\right]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible noise}},$$

  where f^* is the ground-truth function and σ^2 the label-noise variance.

- As C grows, bias decreases and variance increases, yielding the familiar U-shaped curve.

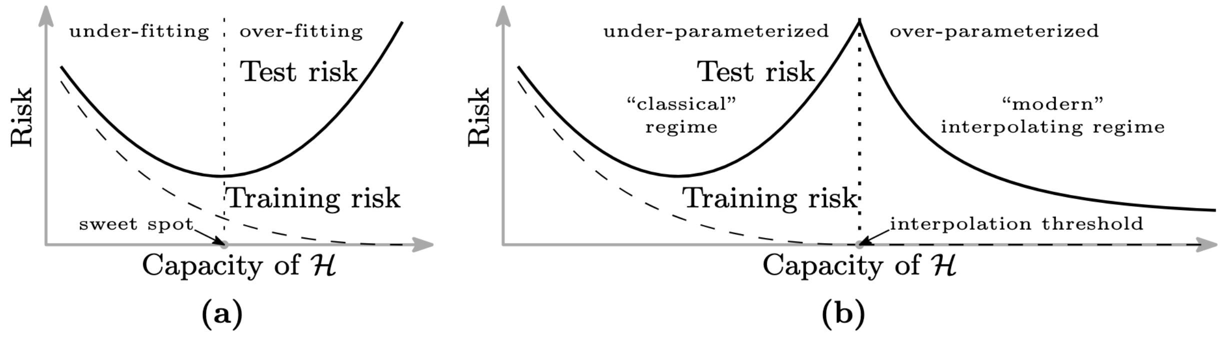


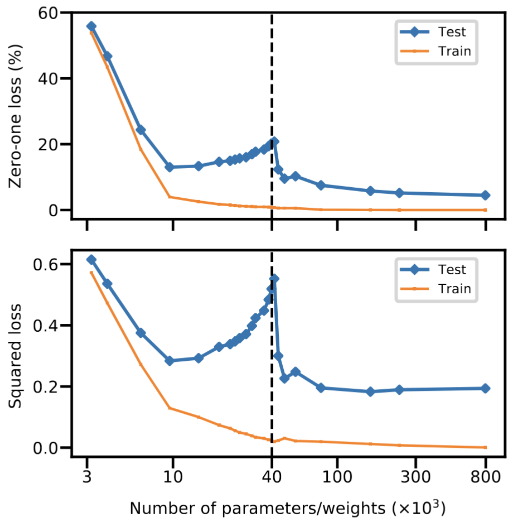
Figure 9.1: Curves for training risk (dashed line) and test risk (solid line). (a) The classical U-shaped risk curve arising from the bias-variance trade-off. (b) The double descent risk curve, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high capacity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk. [Copied from Belkin et al (2019).]

- In contrast, when C surpasses the *interpolation threshold*—the point at which the model can fit the training set exactly—an additional "second descent" in risk emerges. Empirically one observes:

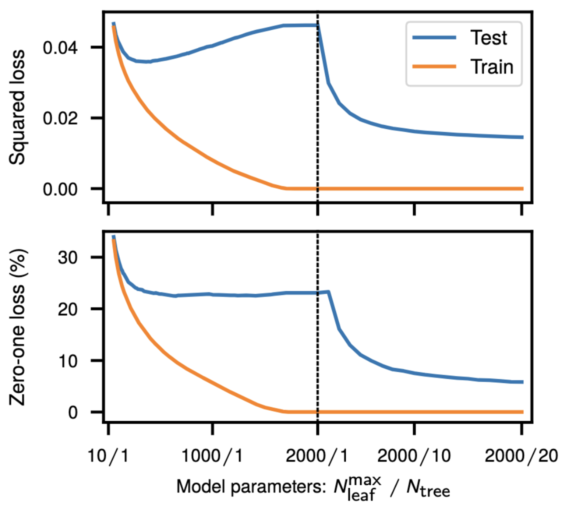
  1. For $C \ll n$: classical under-parameterized U-shape.

2. At $C \approx n$: a sharp peak in test error (variance blows up).
3. For $C \gg n$: a second descent in error (over-parameterized regime).

- Refer to Figures 9.1 and 9.2 for illustrations.
- Moreover, this second descent shows that heavily overparameterized models can nonetheless generalise well—despite zero training error—due to implicit regularisation (e.g. gradient-descent's bias toward minimum-norm interpolates) and benign overfitting effects.



(a) Neural Networks



(b) Random Forests

Figure 9.2: Double descent phenomenon in machine learning models (reference: Belkin et al., 2019). In (b), complexity is controlled by the number of trees N_{tree} and the maximum number of leaves allowed for each tree N_{leaf}^{max}.

9.1.2 Interpolation in Linear Least Squares

Consider the linear regression model

$$y = X\beta^* + \epsilon, \qquad X \in \mathbb{R}^{n\times p}, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I).$$

- *Underdetermined* ($p < n$): the ordinary least-squares (OLS) solution

  $$\hat{\beta}_{\text{OLS}} = (X^\top X)^{-1} X^\top y$$

  is unique but does not interpolate (i.e., complete fit on the points) unless $\epsilon = 0$.

- *Interpolation threshold* ($p = n$): X is square and (w.h.p.) invertible. One can achieve zero training error, and the corresponding unique solution is

  $$\hat{\beta} = X^{-1} y.$$

- *Overparameterized* ($p > n$): infinitely many fits satisfy $X\beta = y$. Gradient descent on the squared loss converges to the *minimum-norm interpolant*

$$\hat{\beta} = \arg \min_{\beta \colon X\beta = y} \|\beta\|_2 = X^+ y.$$

  where X^+ is the pseudo-inverse (i.e., Moore-Penrose inverse) of X.

9.1.3 Epoch-Wise Double-Descent

Beyond model-capacity sweeps, the double-descent phenomenon also manifests over the course of training. Let

$$\mathcal{R}_{\text{train}}(t),\ \mathcal{R}_{\text{test}}(t)$$

denote the empirical (training) and population (test) risks after t epochs of gradient-based optimization. Empirically, one observes:

1. **Initial descent and first ascent.** For $t \in [0, t_{\min}]$, the training risk $\mathcal{R}_{\text{train}}(t)$ decreases monotonically, while the test risk $\mathcal{R}_{\text{test}}(t)$ follows a U-shaped curve—falling until $t \approx t_{\min}$ and then rising as variance dominates.
2. **Interpolation point.** At a critical epoch $t_{\text{int}} \geq t_{\min}$, the model achieves zero training error, $\mathcal{R}_{\text{train}}(t_{\text{int}}) = 0$, and $\mathcal{R}_{\text{test}}$ typically attains its maximum.
3. **Second descent.** For $t > t_{\text{int}}$, further gradient steps continue to decrease $\mathcal{R}_{\text{test}}(t)$ despite zero training loss, leading to a second downward phase and eventual convergence to a test risk below its earlier peak.

This *epoch-wise double-descent* has been documented in deep networks (e.g. Nakkiran et al., 2020) and is known to interact with batch size (which controls gradient noise) and learning-rate schedules (which influence implicit regularization).

References:

- Belkin, M., Hsu, D., Ma, S., & Mandal, S. (2019). Reconciling modern ML practice and the classical bias–variance trade-off. *PNAS*, 116(32):15849–15854.
- Nakkiran, P., Kaplun, G., Bansal, Y., Yang, T., Barak, B., & Sutskever, I. (2020). Deep double descent. *ICML.*
- Hastie, T., Montanari, A., Rosset, S., & Tibshirani, R. (2022). Surprises in high-dimensional ridgeless least squares interpolation. *Biometrika*, 109(3):651–678.

9.2 Model Compression

Deep neural networks often contain millions of parameters and require billions of floating-point operations at inference time. On resource-constrained devices–such as mobile phones, Internet of Things (IoT)–both memory footprint and compute-latency are critical. *Model compression* aims to reduce the size and speed up inference of a trained model while preserving— as much as possible— its predictive accuracy.

9.2.1 Parameter Pruning and Sparsity

Given a parametric model $f(x;\theta)$, $\theta \in \mathbb{R}^d$, trained to minimize a loss $\mathcal{L}(\theta)$ on data $\{(x_i, y_i)\}_{i=1}^N$, unstructured pruning introduces a binary mask $m \in \{0,1\}^d$ over the d parameters:

$$\min_{\theta,m} \mathcal{L}\big(\theta \odot m\big) \quad \text{s.t.} \quad \|m\|_0 \le k,$$

where $\odot$ is element-wise multiplication and $k \ll d$ is a sparsity budget. In practice one often first trains θ, then removes the $d-k$ smallest-magnitude weights:

$$m_j = \begin{cases} 1, & |\theta_j| \ge \tau, \\ 0, & |\theta_j| < \tau, \end{cases}$$

and finally fine-tunes the surviving weights $\theta \odot m$ to recover accuracy. At test time, the forward pass uses only the nonzero weights, reducing both storage and multiply–accumulate (MAC) cost.

References:

- Cheng, Y., Wang, D., Zhou, P. and Zhang, T. (2017). "A survey of model compression and acceleration for deep neural networks." arXiv preprint arXiv:1710.09282.
- Molchanov, Pavlo, et al. "Pruning convolutional neural networks for resource efficient inference." arXiv preprint arXiv:1611.06440 (2016).
- Han, S., Mao, H., Dally, W. (2016). "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding." ICLR 2016.

9.2.2 Post-Training and Quantization-Aware Training

Quantization $Q(\theta)$ is an operation on the parameters that replaces each 32-bit weight $\theta_j \in \mathbb{R}$ by a low-precision code $q_j \in \{0, 1, \ldots, K-1\}$. A simple uniform mapping is

$$q_j = \operatorname{round}\big(\theta_j/\Delta\big), \qquad \hat{\theta}_j = \Delta\, q_j,$$

where Δ is a chosen step-size (e.g. $\Delta = \max_j |\theta_j|/K)$). In *post-training quantization*, one applies quantization $Q(\theta)$ after training. In *quantization-aware training* one instead minimizes

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell\big(f(x_i; Q(\theta)),\, y_i\big),$$

where the forward pass uses $\hat{\theta} = Q(\theta)$ but the gradients flow through a straight-through estimator for Q. Inference then operates entirely in low precision, cutting both model size and arithmetic cost. In particular, during the backward pass, pretend the quantizer Q is the identity function. In other words,

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \hat{\theta}} \underbrace{\frac{\partial Q(\theta)}{\partial \theta}}_{\approx 1} \approx \frac{\partial L}{\partial \hat{\theta}}.$$

Concretely, most implementations simply forward-quantize but then in backprop set

```
theta_q     = Delta * round(theta / Delta)    # quantization
theta_grad = upstream_grad                    # i.e. treat dQ/dtheta = 1
```

so that the gradient "flows through" the quantization unchanged.

References:

- Jacob, Benoit, et al. "Quantization and training of neural networks for efficient integer-arithmetic-only inference." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.
- Banner, Ron, Yury Nahshan, and Daniel Soudry. "Post training 4-bit quantization of convolutional networks for rapid-deployment." Advances in neural information processing systems 32 (2019).

9.2.3 Low-Rank Factorization

Many weight layers $W \in \mathbb{R}^{m\times n}$ have redundancies. One can approximate W by a product of two thin factors,

$$W \approx U\,V^\top, \quad U \in \mathbb{R}^{m\times r},\ V \in \mathbb{R}^{n\times r},\ r \ll \min(m,n),$$

by solving

$$\min_{U,V} \ \|W - U\,V^\top\|_F^2.$$

In practice one replaces W with $\hat{W} = UV^\top$ and *re*trains the network (fine-tuning) by minimizing the original loss $\mathcal{L}$ with respect to U, V. At inference each layer now costs $O(r(m+n))$ MACs instead of $O(mn)$.

References:

- Denton, Emily L., et al. "Exploiting linear structure within convolutional networks for efficient evaluation." Advances in neural information processing systems 27 (2014).
- Tai, Cheng, et al. "Convolutional neural networks with low-rank regularization." arXiv preprint arXiv:1511.06067 (2015).

9.2.4 Knowledge Distillation

A large "teacher" network $f_t(x)$ can be used to train a smaller "student" $f_s(x)$ by matching soft outputs. Let $z_t,\ z_s \in \mathbb{R}^C$ be the teacher's and student's logits over C classes. Define softened probabilities

$$p_t^{(i)} = \text{softmax}\big(z_t^{(i)}/T\big), \quad p_s^{(i)} = \text{softmax}\big(z_s^{(i)}/T\big),$$

where $T > 1$ is a temperature. The student is trained to minimize

$$\mathcal{L}_{\text{KD}} = (1-\alpha)\ \underbrace{\frac{1}{N}\sum_i \ell\big(y_i,\, \text{softmax}(z_s^{(i)})\big)}_{\text{student loss}}\ +\alpha\ \underbrace{T^2\,\text{KL}\big(p_t \parallel p_s\big)}_{\text{distillation loss}},$$

where ℓ is the standard cross-entropy and KL is the Kullback–Leibler divergence. The first term on the left of the above expression is a loss between student network predictions and true labels while the second term is a loss between the predictions of the student and teacher networks.

At test time only the small student f_s is used, yielding a compact model that inherits much of the teacher's accuracy.

References:

- Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." arXiv preprint arXiv:1503.02531 (2015).

9.3 Self-Supervised Learning

In domains like *vision*, *language*, and *time-series analysis*, labelled data is often *scarce or costly* to acquire. *Self-supervised learning (SSL)* addresses this challenge by leveraging the intrinsic structure of unlabelled data $\mathcal{D}_{\text{unlabeled}} = \{x_1, \ldots, x_n\}$.

SSL's primary goal is to enable models to learn transferable representations from unlabelled data by inventing *pretext tasks* that reveal inherent structure. Unlike supervised learning, which relies on explicit human annotations (e.g., class labels or bounding boxes), SSL creates implicit supervision signals by:

- **Exploiting Data Structure**: Using spatial/temporal continuity (e.g., predicting missing image patches or future time steps),
- **Leveraging Invariance**: Teaching models that differently augmented views of the same input (e.g., rotated images, noisy audio clips) should have similar embeddings,
- **Contextual Reasoning**: Recovering masked elements (words, pixels, etc.) from their surroundings, mimicking human reasoning.

The learned representations can then be *transferred* to downstream tasks through a two-phase process:

Phase 1: Representation Learning

- Train encoder f_θ via pretext tasks on $\mathcal{D}_{\text{unlabeled}}$
- Output: Latent representations $\mathbf{z} = f_\theta(\mathbf{x})$ capturing data semantics

Phase 2: Downstream Adaptation

1. **Extract features**: Compute $\mathbf{z}_i = f_\theta(\mathbf{x}_i)$ for labeled downstream data $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$
2. **Train task head**: Learn mapping g_ϕ from features to labels:

$$\min_\phi \frac{1}{N} \sum_{i=1}^{N} \ell(g_\phi(\mathbf{z}_i), y_i)$$

3. **Deploy**: Use composed model $g_\phi \circ f_\theta$ for prediction

This paradigm achieves human-like learning—first understanding patterns through observation (SSL phase), then specializing with minimal guidance (downstream phase)—enabling sample-efficient adaptation to new tasks with as few as 10-100 labeled examples.

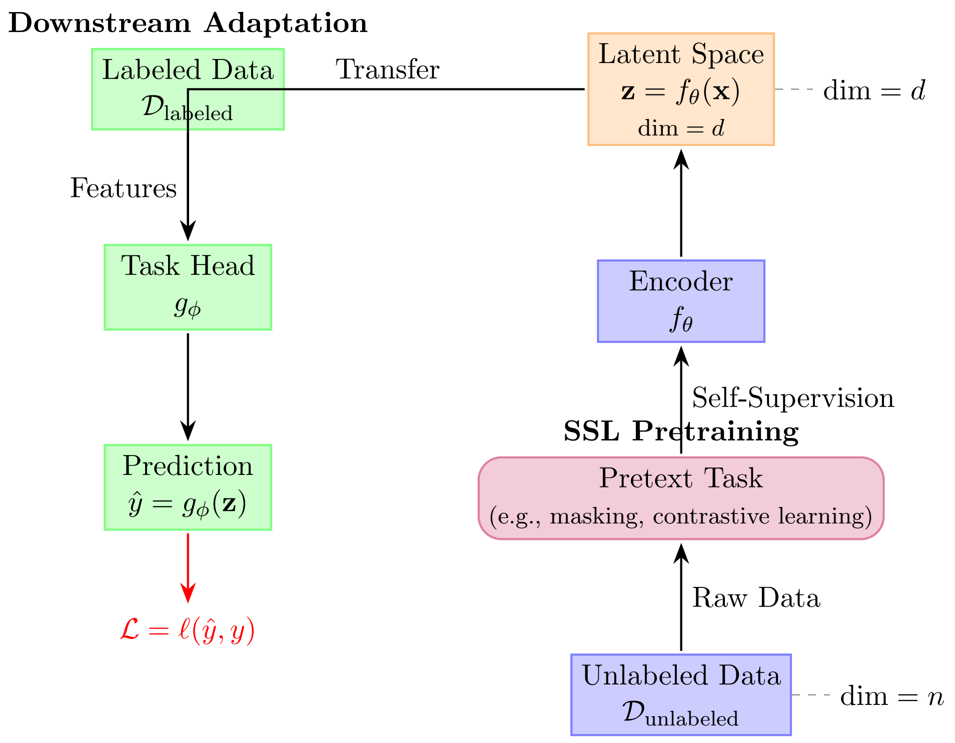

Downstream Adaptation
Labeled Data $\mathcal{D}_{\text{labeled}}$
Transfer
Latent Space $\mathbf{z} = f_\theta(\mathbf{x})$ dim $= d$
dim $= d$
Features
Task Head g_ϕ
Encoder f_θ
Self-Supervision
SSL Pretraining
Prediction $\hat{y} = g_\phi(\mathbf{z})$
Pretext Task (e.g., masking, contrastive learning)
Raw Data
$\mathcal{L} = \ell(\hat{y}, y)$
Unlabeled Data $\mathcal{D}_{\text{unlabeled}}$
dim $= n$

Figure 9.3: Self-supervised learning pipeline: Pretraining on unlabeled data followed by downstream adaptation with labeled examples.

Remark 9.3.1

Self-supervised learning excels when abundant unlabelled data exists but labelled examples are scarce or costly to obtain. By first pretraining on large-scale unlabelled datasets through pretext tasks (like masked prediction or contrastive learning), SSL extracts general-purpose features that capture the underlying data structure. These learned representations then enable effective fine-tuning on downstream tasks with as little as 1% of the labelled data required by traditional supervised methods. This approach is particularly valuable in domains like medical imaging, NLP, and industrial Internet of Things, where unlabelled data is plentiful but expert annotations are expensive. However, SSL provides limited benefits when both labelled and unlabelled datasets are small, or when tasks demand precise, rule-based labelling that can't be inferred from data patterns alone. The paradigm shift lies in replacing human-provided labels with automatically generated learning signals from the data itself, dramatically improving label efficiency while maintaining or even surpassing supervised performance.

9.3.1 Masked Representation Learning

Masked representation learning is a domain-agnostic SSL technique where random portions of structured input are obscured, forcing the model to learn contextual relationships. Given input $\mathbf{x}$ composed of elements $\{x_1, \ldots, x_n\}$ (tokens, pixels, time steps, etc.):

1. Sample mask positions $\mathcal{M} \subseteq \{1, \ldots, n\}$ with masking ratio ρ (typically $\rho \in [0.1, 0.8]$)
2. Generate corrupted input $\tilde{\mathbf{x}}$:
$$\tilde{x}_i = \begin{cases} \texttt{[MASK]} & i \in \mathcal{M} \\ x_i & i \notin \mathcal{M} \end{cases}$$
3. Encode $\tilde{\mathbf{x}}$ with backbone network f_θ to obtain representations:
$$\mathbf{z} = f_\theta(\tilde{\mathbf{x}}) \quad \in \mathbb{R}^{n \times d}$$
4. For each $i \in \mathcal{M}$, reconstruct original element x_i using projection head g_ϕ:
$$\hat{x}_i = g_\phi(z_i)$$
5. Optimize parameters with element-wise reconstruction loss:
$$\min_{\theta,\phi} \sum_{i \in \mathcal{M}} \ell(\hat{x}_i, x_i)$$

Loss Selection by Data Type:

- **Categorical** (tokens/classes): Cross-entropy over vocabulary

- **Continuous** (pixels/sensors): Mean Squared Error (MSE)
- **Structured** (graphs): Domain-specific losses (e.g., edge prediction)

Remark 9.3.2

Implementation varies by domain:

- **Text**: Replace tokens with `[MASK]` ID (e.g., BERT)
- **Vision**: Mask patch embeddings (e.g., MAE)
- **Time-Series**: Zero-out sensor readings (e.g., TS-TCC)

```
# Generic masking pseudocode
def mask_input(x, mask_ratio=0.15):
    n = x.shape[0] # Number of elements
    mask_indices = random.sample(range(n), k=int(n * mask_ratio))
    x_corrupted = x.clone()
    x_corrupted[mask_indices] = MASK_TOKEN # Domain-specific implementation
    return x_corrupted, mask_indices
```

This approach forces the model to develop *contextual understanding* by exploiting relationships between masked and unmasked elements, yielding transferable representations.

9.3.2 Contrastive Learning

Contrastive learning creates supervisory signals by teaching the model to identify *similar* (positive) and *dissimilar* (negative) data representations. Given an anchor sample $x \sim \mathcal{D}_{\text{unlabeled}}$:

1. Generate two augmented views: $\tilde{x}_i = \mathcal{T}_i(x)$, $\tilde{x}_j = \mathcal{T}_j(x)$ using stochastic augmentations $\mathcal{T}$ (e.g., time-series cropping, noise injection).
2. Encode views into latent representations:
$$\mathbf{h}_i = f_\theta(\tilde{x}_i), \quad \mathbf{h}_j = f_\theta(\tilde{x}_j) \in \mathbb{R}^d$$
3. Optimize using the *normalized temperature-scaled cross entropy loss (NT-Xent)*:
$$\mathcal{L}_{\text{contrast}} = -\log \frac{\exp(\text{sim}(\mathbf{h}_i, \mathbf{h}_j)/T)}{\sum_{k=1}^{2B} \mathbb{I}(k \neq i) \exp(\text{sim}(\mathbf{h}_i, \mathbf{h}_k)/T)} \tag{9.1}$$
where:
    - $\text{sim}(\mathbf{u}, \mathbf{v}) = \mathbf{u}^\top \mathbf{v} / \|\mathbf{u}\| \|\mathbf{v}\|$ (cosine similarity)
    - $T > 0$ is a temperature hyperparameter
    - B is the batch size (negative samples come from other instances in batch)

The loss maximizes agreement between positive pairs while repelling negative pairs in the latent space $\mathbb{R}^d$. This paradigm underpins methods like SimCLR and TS-TCC for time-series.

9.3.3 References

- Devlin et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding."
- Chen et al. (2020). "A Simple Framework for Contrastive Learning of Visual Representations (SimCLR)."
- He et al. (2020). "Momentum Contrast for Unsupervised Visual Representation Learning (MoCo)."
- Grill et al. (2020). "Bootstrap Your Own Latent (BYOL): A New Approach to Self-Supervised Learning."