AI for Medical Image Classification

Tso-Jung Yen

Institute of Statistical Science
Academia Sinica

tjyen@stat.sinica.edu.tw

Academia Sinica

August 12, 2020

Outline

- **Lecture 1: Stochastic gradient descent algorithm and backpropagation**
  - Gradient descent algorithm
  - Stochastic gradient descent algorithm
  - Adaptive methods
  - Backpropagation algorithm
- **Lecture 2: Methods for efficient model training**
  - Use of pre-trained models
  - Label switching
  - Data augmentation
  - Batch normalization
- **Lecture 3: Criteria for evaluating model performance in classification**

Lecture 1

Tso-Jung Yen

Institute of Statistical Science
Academia Sinica

tjyen@stat.sinica.edu.tw

Academia Sinica

August 12, 2020

Gradient Descent Algorithms

- **Basic idea:**
  - Consider the following optimization problem:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} l(\boldsymbol{\theta}). \tag{1}$$

    Here $\boldsymbol{\theta} \in \mathbb{R}^p$ is a p-dimensional vector.

  - **Differentiability assumption:** If l is twice differentiable on $\mathbb{R}^p$, we may write $l(\boldsymbol{\theta})$ via Taylor's expansion around $\boldsymbol{\theta}' \in \mathbb{R}^p$ as

$$\begin{aligned} l(\boldsymbol{\theta}) &= l(\boldsymbol{\theta}') + \nabla l(\boldsymbol{\theta}')^T(\boldsymbol{\theta} - \boldsymbol{\theta}') + O(||\boldsymbol{\theta} - \boldsymbol{\theta}'||_2^2) \\ &\approx l(\boldsymbol{\theta}') + \nabla l(\boldsymbol{\theta}')^T(\boldsymbol{\theta} - \boldsymbol{\theta}') + \frac{1}{2c}||\boldsymbol{\theta} - \boldsymbol{\theta}'||_2^2. \end{aligned} \tag{2}$$

    where $c > 0$ is a constant.

  - With (2), we may "approximate" (1) as

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \left\{ l(\boldsymbol{\theta}') + \nabla l(\boldsymbol{\theta}')^T(\boldsymbol{\theta} - \boldsymbol{\theta}') + \frac{1}{2c}||\boldsymbol{\theta}' - \boldsymbol{\theta}||_2^2 \right\} \tag{3}$$

    Differentiating (3) with respect to $\boldsymbol{\theta}$ and equaling the derivative to zero yields

$$\nabla l(\boldsymbol{\theta}') - \frac{\boldsymbol{\theta}' - \boldsymbol{\theta}}{c} = 0. \tag{4}$$

    Solving (4) yields $\boldsymbol{\theta} = \boldsymbol{\theta}' - c\nabla l(\boldsymbol{\theta}')$.

Gradient Descent Algorithms

- **Basic idea (contd):**
  - **Iterative scheme:** We can use the iterative scheme

$$\begin{aligned} \boldsymbol{\theta}^{r+1} &= \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^p} \left\{ l(\boldsymbol{\theta}^r) + \nabla l(\boldsymbol{\theta}^r)^T(\boldsymbol{\theta} - \boldsymbol{\theta}^r) + \frac{1}{2c^r}||\boldsymbol{\theta}^r - \boldsymbol{\theta}||_2^2 \right\} \\ &= \boldsymbol{\theta}^r - c^r \nabla l(\boldsymbol{\theta}^r). \end{aligned} \tag{5}$$

    to find a solution to the optimization problem (1).
  - The iterative scheme (5) is an example of the **gradient descent algorithm**.
  - Here c^r is called the **learning rate** (or stepsize) at iteration r.
  - **Stopping criterion:** From (5) we know that

$$||\boldsymbol{\theta}^{r+1} - \boldsymbol{\theta}^r||_2 = c^r ||\nabla l(\boldsymbol{\theta}^r)||_2,$$

    which suggests that we may stop the iterative scheme when $||\nabla l(\boldsymbol{\theta}^r)||_2$ is small.
  - In practice to stop the algorithm, we use the following **stopping criterion**:

$$||\nabla l(\boldsymbol{\theta}^r)||_2 \leq \epsilon. \tag{6}$$

    where ϵ is the tolerance for the error. At the rth step, if (6) is satisfied, we stop the iterative scheme (5).

Gradient Descent Algorithms

- **The descent property:**
  - By letting $\boldsymbol{\theta} = \boldsymbol{\theta}^{r+1}$, $\boldsymbol{\theta}' = \boldsymbol{\theta}^{r}$ in (2) we have

$$\begin{aligned} l(\boldsymbol{\theta}^{r+1}) &\approx l(\boldsymbol{\theta}^{r}) - c^{r}||\nabla l(\boldsymbol{\theta}^{r})||_2^2 + \frac{[c^{r}]^2}{2c}||\nabla l(\boldsymbol{\theta}^{r})||_2^2 \\ &= l(\boldsymbol{\theta}^{r}) - \left(c^{r} - \frac{[c^{r}]^2}{2c}\right)||\nabla l(\boldsymbol{\theta}^{r})||_2^2. \end{aligned}$$

  - From the above result we can see if

$$c^{r} - \frac{[c^{r}]^2}{2c} \geq 0, \tag{7}$$

    then we will have

$$l(\boldsymbol{\theta}^{r+1}) \leq l(\boldsymbol{\theta}^{r}). \tag{8}$$

Gradient Descent Algorithms

- **The descent property (contd):**
  - That means, the sequence $\{\boldsymbol{\theta}^r\}_r$ generated by the iterative scheme (5) leads to a decrease in the loss function l.
  - The inequality (8) is called the **descent property** associated with the sequence $\{\boldsymbol{\theta}^r\}_r$.
  - To make a sequence generated by (5) to satisfy the descent property, we need to ensure (8) holds.
  - Now maximizing (7) yields

$$c^r = c,$$

    which is the optimal choice for the learning rate c^r. This ensures that

$$c^r - \frac{[c^r]^2}{2c} = \frac{c}{2} \geq 0.$$

  - **Important!!** In training deep neural network models, it is difficult to know c priorly.
  - Setting learning rate c^r relies on "heuristics", e.g. trial-and-error.

Stochastic Gradient Descent Algorithms

- **Problem setting:**
  - Consider the following optimization problem:

$$\min_{\boldsymbol{\theta} \in \mathcal{R}^p} \quad \frac{1}{n} \sum_{i=1}^{n} l(\boldsymbol{\theta}; \mathbf{x}_i), \tag{9}$$

    where $\boldsymbol{\theta} \in \mathbb{R}^p$ is a p-dimensional vector of parameters, and $\mathbf{x}_i$ is a data point containing information about the ith observation.
  - The problem (9) is a commonly-seen problem format in **statistics** and **machine learning**, e.g. maximum likelihood estimation.
  - For practical purposes, we assume the n observations are independently observed. Further define

$$h(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} l(\boldsymbol{\theta}; \mathbf{x}_i).$$

Stochastic Gradient Descent Algorithms

- **Problem setting (contd):**
  - Usually one can see the the objective function in (9) as

$$h(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} l(\boldsymbol{\theta}; \mathbf{x}_i) = \mathbb{E}_{\mathbf{x}}[l(\boldsymbol{\theta}; \mathbf{x})], \tag{10}$$

    where $\mathbb{E}_{\mathbf{x}}$ is an expectation operator such that $\mathbb{E}_{\mathbf{x}}(\delta_{\{\mathbf{x}=\mathbf{x}_i\}}) = \mathbb{P}(\mathbf{x} = \mathbf{x}_i) = n^{-1}$, where $\delta_{\{\mathbf{x}=\mathbf{x}_i\}} = 1$ if $\mathbf{x} = \mathbf{x}_i$, and $\delta_{\{\mathbf{x}=\mathbf{x}_i\}} = 0$ otherwise.
  - The representation (10) provides us a way to see the deterministic objective function $h(\boldsymbol{\theta})$ of problem (9) as the expectation of a random objective function $l(\boldsymbol{\theta}; \mathbf{x})$ with $\mathbb{P}(\mathbf{x} = \mathbf{x}_i) = n^{-1}$.

Stochastic Gradient Descent Algorithms

- To find a solution to (9), we consider the following iterative scheme:

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^{r} - c_r \mathbf{v}^{r}, \tag{11}$$

  where $\mathbf{v}^r$ is a p-dimensional vector.

- When $\mathbf{v}^r = \nabla h(\boldsymbol{\theta}^r)$, the iterative scheme (11) is an example of the gradient descent algorithm.

- When $\mathbf{v}^r$ is a random vector such that $\mathbb{E}[\mathbf{v}^r] = \nabla h(\boldsymbol{\theta}^r)$, the iterative scheme is an example of the **stochastic gradient descent algorithm (SGD)**.

Stochastic Gradient Descent Algorithms

- **Application:**
  - In practice, we use the following iterative scheme to compute $\boldsymbol{\theta}^{r+1}$: At the $(r+1)$th iteration, choose i_r *uniformly* from $\{1, 2, \cdots, n\}$ and define

$$\mathbf{v}_{i_r}^r = \nabla l(\boldsymbol{\theta}^r; \mathbf{x}_{i_r}).$$

    In this case we have

$$\mathbb{E}[\mathbf{v}_{i_r}^r] = \mathbb{E}_{\mathbf{X}}[\nabla l(\boldsymbol{\theta}^r; \mathbf{x}_{i_r})] = \frac{1}{n}\sum_{i=1}^{n} \nabla l(\boldsymbol{\theta}^r; \mathbf{x}_{i_r}) = \nabla h(\boldsymbol{\theta}^r).$$

  - According to theory of stochastic gradient descent algorithms (Chapter 8 of Beck, 2017), if $\mathbf{v}_{i_r}^r$ further satisfies some regularity conditions, then we can use the iterative scheme

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^r - c_r \cdot \mathbf{v}_{i_r}^r$$

    to find a solution to the problem (9).

Stochastic Gradient Descent Algorithms

- *Remark 1:* In training deep neural network models, we usually sample a *batch* of $\mathbf{v}_{i_r}^r$'s to compute stochastic approximation of $\nabla h(\boldsymbol{\theta}^r)$, e.g. with batch size B, we sample $\{\mathbf{v}_{i_r}\}_{i=1}^B$ and then compute

$$\frac{1}{B}\sum_{i=1}^{B}\mathbf{v}_{i_r}^r = \frac{1}{B}\sum_{i=1}^{B}\nabla l(\boldsymbol{\theta}^r;\mathbf{x}_{i_r})$$

  to approximate $\nabla h(\boldsymbol{\theta}^r)$.

- *Remark 2:* The Stochastic gradient descent algorithm is useful when (a) the input data are large; and (b) the objective function contains random errors and the expectation of the objective function exists.

Adaptive Methods

- **AdaGrad (Duchi et al., 2011):**
  - The **AdaGrad** iterative scheme takes the following form for updating $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^{r} - c^{r}[\mathbf{H}^{r}]^{-1}\mathbf{g}^{r},$$

where c^r is the learning rate, $\mathbf{H}^r = \text{diag}(\mathbf{u}^r + \epsilon\mathbf{1})$ with $\epsilon \geq 0$ is a scale matrix, and

$$\mathbf{u}^{r} \quad = \quad \left[\sum_{s=1}^{r} \mathbf{g}^{s} \circ \mathbf{g}^{s}\right]^{1/2},$$

and $\mathbf{g}^r$ is a stochastic approximation to the gradient of the loss function.

- **RMSProp (Tieleman and Hinton, 2012):**
  - The **RMSProp** iterative scheme takes the following form for updating $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}^{r+1} = \boldsymbol{\theta}^{r} - c^{r}[\mathbf{H}^{r}]^{-1}\mathbf{g}^{r},$$

where c^r is the learning rate, $\mathbf{H}^r = \text{diag}\{(\mathbf{u}^r + \epsilon\mathbf{1})^{1/2}\}$ with $\epsilon \geq 0$, and

$$\mathbf{u}^{r} = \gamma\mathbf{u}^{r-1} + (1-\gamma)\mathbf{g}^{r} \circ \mathbf{g}^{r}$$

with $\gamma \in [0, 1]$.

Adaptive Methods

- **Adam (Kingma and Ba, 2015):**
  - The **Adam** algorithm extends the ideas of **AdaGrad** and **RMSProp** algorithms by using a *smoothed* version of the gradient vector.
  - The name **Adam** is inspired from **adaptive moment estimation**.
  - The gradient vector is computed by applying exponential moving averaging over all gradient vectors at previous iterations.
  - The smoothed version of the gradient vector at the rth iteration is defined by

$$\tilde{\mathbf{g}}^r = \beta \tilde{\mathbf{g}}^{r-1} + (1-\beta)\mathbf{g}^r,$$

    where $\beta \in [0, 1]$.
  - In addition, the **Adam** algorithm scales the gradient vector by introducing a matrix with diagonal elements $[(1-\gamma^r)^{-1}\mathbf{u}^r]^{1/2} + \epsilon\mathbf{1}$, where $\gamma \in [0, 1]$, $\epsilon \geq 0$, and $\mathbf{v}^r$ is adaptively computed in a way such that

$$\mathbf{u}^r = \gamma \mathbf{u}^{r-1} + (1-\gamma)\mathbf{g}^r \circ \mathbf{g}^r.$$

Adaptive Methods

- **Comparisons:** Wilson et al. (2017) compared three adaptive methods: **AdaGrad**, **RMSProp** and **Adam** with non-adaptive methods: stochastic gradient descent algorithm (SGD), the heavy ball method, and Nesterov's Accelerated Gradient method. They found:
  - Solutions found by adaptive methods were usually generalized worse than those found by non-adaptive methods.
  - Even when the adaptive methods achieved the same training loss or lower than non-adaptive methods, performances on the test data were still worse.
  - Adaptive methods might have faster initial progress on the training data, but their performance quickly reached plateau on the test data.
  - Though conventional wisdom suggested that the **Adam** algorithm do not require tuning, the authors found that tuning the initial learning rate and decay scheme for the **Adam** algorithm yields significant improvements over its default settings in all cases.

Adaptive Methods

- **When adaptive methods work?**
  - According to Wilson et al. (2017), adaptive methods are particularly popular for training **GANs** and **Q-learning** with function approximation.
  - These applications stand out **because they are not about solving optimization (minimization) problems**.
  - The authors guess that the dynamics of adaptive methods may be accidentally well matched to these non-optimization iterative search procedures.
  - It is also possible that carefully tuned non-adaptive methods may work as well or better in these applications.

Backpropagation

- **The deep neural network model:**
  - Consider an L-layer neural network:

$$\begin{aligned}
\mathbf{a}^{[0]} &= \mathbf{x}, \\
\mathbf{z}^{[l]} &= \boldsymbol{\theta}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \text{ for } l = 1, 2, \cdots, L, \\
\mathbf{a}^{[l]} &= \sigma^{[l]} \circ \mathbf{z}^{[l]} \text{ for } l = 1, 2, \cdots, L-1, \\
\widehat{\mathbf{y}} &= \widehat{\mathbf{y}}(\mathbf{z}^{[L]}),
\end{aligned}$$

    where $\mathbf{x}$ is the input (i.e. features), $\mathbf{a}^{[l]}$ and $\boldsymbol{\theta}^{[l]}$ are the vector of activation functions $\sigma^{[l]}$ and the corresponding weight matrix at the lth layer, $\mathbf{z}^{[l]}$ is a function of $\mathbf{a}^{[l-1]}$, and $\widehat{\mathbf{y}}$ is the predicted value generated by the neural network for response $\mathbf{y}$.
  - We assume $\mathbf{a}^{[l]}$ is an m_l-dimensional vector, $\mathbf{z}^{[l]}$ is an m_l-dimensional vector, and $\boldsymbol{\theta}^{[l]}$ and $\mathbf{b}^{[l]}$ are $m_l \times m_{l-1}$ matrix and m_l-dimensional vector, respectively. For practical purposes, we assume $\widehat{\mathbf{y}}$ is a p-dimensional vector.

Backpropagation

- **The deep neural network model (contd):**
  - We estimate $(\boldsymbol{\theta}, \mathbf{b}) = \{\boldsymbol{\theta}^{[l]}, \mathbf{b}^{[l]}\}_{l=1}^{L}$ by

$$(\widehat{\boldsymbol{\theta}}, \widehat{\mathbf{b}}) = \arg\min_{\boldsymbol{\theta}, \mathbf{b}} \mathbb{E}_{\mathbf{x},\mathbf{y}}[f(\boldsymbol{\theta}, \mathbf{b}; \mathbf{x}, \mathbf{y})], \tag{12}$$

    that is, we estimate parameters $\{\boldsymbol{\theta}^{[l]}, \mathbf{b}^{[l]}\}_{l=1}^{L}$ by the minimizing loss function $\mathbb{E}_{\mathbf{x},\mathbf{y}}[f(\boldsymbol{\theta}, \mathbf{b}; \mathbf{x}, \mathbf{y})] = h(\boldsymbol{\theta}, \mathbf{b})$.
  - Solving (12) is usually done by gradient-based algorithms, e.g. gradient descent algorithms, stochastic gradient descent algorithms.
  - Directly computing (or approximating) the gradient vector of the loss function $h(\boldsymbol{\theta}, \mathbf{b})$ with respect to all parameters $\{\boldsymbol{\theta}^{[l]}, \mathbf{b}^{[l]}\}_{l=1}^{L}$ can be very complicated.
  - A clever technique for computing the gradient vector for solving (12) is called the **forward-backward propagation**.
  - The forward step is straightforward: At the lth layer, one first computes $\mathbf{z}^{[l]}$ using input $\mathbf{a}^{[l-1]}$ and parameters $(\boldsymbol{\theta}^{[l]}, \mathbf{b}^{[l]})$ and then computes $\mathbf{a}^{[l]} = \sigma^{[l]} \circ \mathbf{z}^{[l]}$, and then starts at next round computation at the $(l+1)$th layer.
  - *Remark:* One may run the first round forward step by using initial values randomly generated from some distributions.

Backpropagation

- **Backpropagation for gradient computation:**
  - In summary we have

$$\begin{aligned}
\nabla_{\mathbf{z}^{[l]}} h &= [\nabla_{\mathbf{z}^{[l]}} \widehat{\mathbf{y}}][\nabla_{\widehat{\mathbf{y}}} h], \\
\nabla_{\boldsymbol{\theta}^{[l]}} h &= [\nabla_{\boldsymbol{\theta}^{[l]}} \mathbf{z}^{[l]}][\nabla_{\mathbf{z}^{[l]}} h] = \mathcal{A}^{[l-1]} \nabla_{\mathbf{z}^{[l]}} h, \\
\nabla_{\mathbf{b}^{[l]}} h &= [\nabla_{\mathbf{b}^{[l]}} \mathbf{z}^{[l]}][\nabla_{\mathbf{z}^{[l]}} h] = \nabla_{\mathbf{z}^{[l]}} h,
\end{aligned}$$

  for $l = L$, and

$$\begin{aligned}
\nabla_{\mathbf{z}^{[l]}} h &= \mathcal{D}^{[l]} (\boldsymbol{\theta}^{[l+1]})^T \nabla \mathbf{z}^{[l+1]} h, \\
\nabla_{\boldsymbol{\theta}^{[l]}} h &= \mathcal{A}^{[l-1]} \nabla_{\mathbf{z}^{[l]}} h, \\
\nabla_{\mathbf{b}^{[l]}} h &= \nabla_{\mathbf{z}^{[l]}} h,
\end{aligned}$$

  from $l = L - 1$ to $l = 1$.

  - Such a numerical procedure is called the **backpropagation** algorithm, which is a method for computing the gradient vector of a composite function such as the loss function $h(\boldsymbol{\theta}, \mathbf{b})$ used here.

Backpropagation

- **Some remarks on the backpropagation algorithm:**
  - The **backpropagation** algorithm is a method that recursively computes the gradient vector of a composite function based on the idea of the chain rule.
  - It is a popular method in training deep neural network models since they usually have composite loss functions involving many layers.
  - It is a special case of a more general method called the **automatic differentiation** or **algorithmic differentiation**, which is widely used in state-of-the-art deep learning packages such as **TensorFlow** or **PyTorch**.
  - Besides the matrix point of view, one way to learn the backpropagation algorithm is to adopt an approach based on the **computational graph**.

References

A. Beck (2017). *First-Order Methods in Optimization*. SIAM. Parts of materials on stochastic gradient descent algorithms are from Chapter 8 of this book.

F. E. Bottou, L .and Curtis and J. Nocedal (2018). Optimization methods for large-scale machine learning. SIAM Review, 60, 223-311.

J. Duchi, E. Hazan, and Y. Singer (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12, 2121-2159.

I. Goodfellow, Y. Bengio and A. Courville. *Deep Learning*. The MIT Press, Massachsetts, 2016. Chapter 6.5 of the book gives an introduction to the backpropagation algorithm.

D. P. Kingma and J. L. Ba (2015). Adam: a method for stochastic optimization. ICLR.

M. W. Mahoney, J. C. Duchi and A. C. Gilbert (2018). *The Mathematics of Data*. American Mathematical Society. Parts of materials on stochastic projected subgradient algorithms are from Chapter 3 (written by Professor J. Duchi) of this book.

Lecture 2

Tso-Jung Yen

Institute of Statistical Science
Academia Sinica

tjyen@stat.sinica.edu.tw

Academia Sinica

August 12, 2020

Use of Pre-trained Models

- **Idea:**
  - A pre-trained model is a model trained on a dataset different from our dataset and may be designed for doing a task different from our model's task.
  - If the dataset for training the pre-trained model is general and large enough, then the features computed by the pre-trained model may be more likely to reflect "generic spatial structure" of the visual world (Section 5.3 of Chollet (2017)).

Use of Pre-trained Models

- **Example:**
  - Consider the following pre-trained model:

$$\text{PretrainedModel} = \widehat{\Psi}^{\text{dc}} \circ \widehat{\Psi}^{\text{conv}}(\mathbf{x}). \tag{13}$$

    Here $\widehat{\Psi}^{\text{dc}}$ and $\widehat{\Psi}^{\text{conv}}$ are densely connected and convolutional neural network architectures, respectively. We use "hat" to indicate that the pre-trained model has been trained on some dataset and the weights (parameters) have trained values.

  - Our model is given as follows:

$$\text{MyModel} = \Phi^{\text{dc}} \circ \Phi^{\text{conv}}(\mathbf{x}). \tag{14}$$

  - By using pre-trained model (13), our new model may look like the following one:

$$\begin{aligned} \text{MyNewModel} &= \Phi^{\text{dc}} \circ \widehat{\mathbf{z}} \\ &= \Phi^{\text{dc}} \circ \widehat{\Psi}^{\text{conv}}(\mathbf{x}). \end{aligned}$$

    Here $\widehat{\mathbf{z}}$ is the features extracted form $\mathbf{x}$ using the CNN part of the pre-trained model.

  - The use of pre-trained model is a kind of **transfer learning**, a learning approach in that model training relies on using exogenous information provided by other models.

Use of Pre-trained Models

- **Application:**
  - When Φ^{conv} is large, it may be difficult to train (14) if the dataset is small. In this situation, one can apply a pre-trained model to compute $\widehat{\mathbf{z}}$ to replace Φ^{conv} and then only trains Φ^{dc} on the small dataset.
  - In some cases one will freeze all parameters in $\widehat{\Psi}^{\text{conv}}$, i.e. treating $\widehat{\mathbf{z}}$ as a fixed quantity, when training (15). In other cases one may unfreeze top layers of $\widehat{\Psi}^{\text{conv}}$ and run a joint training for (15). It is called **fine-tuning**.
- **Caveat:**
  - If our dataset is very different from the dataset for training the pre-trained model, we should only apply lower layers of $\widehat{\Psi}^{\text{conv}}$ to compute $\widehat{\mathbf{z}}$.
  - It is because lower layers are responsible for capturing local generic features (edges, colors, textures and so on) of an image while upper layers are responsible for capturing abstract concepts of the image.

Label Smoothing

- Assume we have K classes and let $\mathbf{y} = (y^1, y^2, \cdots, y^K)$ denote the label such that $y^k = 1$ if the observation belongs to class k.

- Label smoothing (Szegedy et al., 2016) of $\mathbf{y}$, denoted by $\mathbf{y}^{LS}$, is defined as

$$\mathbf{y}^{LS} = (1 - \alpha)\mathbf{y} + \frac{\alpha}{K}(\mathbf{1} - \mathbf{y}),$$

  where $\alpha \in [0, 1)$ is the label smoothing parameters and $\mathbf{1}$ is a K-dimensional vector of 1's.

- For example, $K = 3$, $\alpha = 0.005$ and $\mathbf{y} = (0, 0, 1)$, then $\mathbf{y}^{LS} = (0.0017, 0.0017, 0.995)$.

- When does label smoothing help? (Müller et al, 2019)

  - The authors found that label smoothing can (a) improve generalization and (b) model calibration;

  - But it may hurt knowledge distillation if a teacher network is trained with label smoothing. **A teacher with better accuracy is not necessarily the one that distills better**.

  - **Explanation:** It is because the relative information between logits is "erased" when the teacher is trained with label smoothing.

Mixup

- Data augmentation is a widely-used approach in machine learning, particularly when training data involving images.
- Commonly-seen data augmentation techniques include rotation, translation, cropping, resizing, flipping, and random erasing.
- **Mixup (Zhang et al., 2018):**
  - **Motivation:** The empirical risk minimization (ERM)-trained neural networks can memorize all data points in the training sample (i.e. training error ≈ 0). However, those neural networks may perform significantly worse on a neighborhood point of that training sample.
  - The **mixup** method is data augmentation technique that encourages the neural network to behave **linearly in-between data points in the training sample**.

Mixup

- **Mixup (contd):**
  - Instead of using available training sample, the method trains models on virtual (synthetic) data that consist of convex combination of two data points $(y, \mathbf{x})$ and $(y', \mathbf{x}')$ randomly selected from the training sample:

$$\begin{aligned} \tilde{\mathbf{x}} &= \lambda \mathbf{x} + (1-\lambda)\mathbf{x}', \\ \tilde{y} &= \lambda y + (1-\lambda)y', \end{aligned}$$

    where λ is a random variable and following Beta(α, α) with $\alpha \in [0, \infty)$.
  - Here α is a hyperparameter that should be decided by researchers before model training.
  - When models are trained with data $(\tilde{x}, \tilde{y})$, the models are said to be trained under the **vicinal risk minimization (VRM)**.
  - **Application to the generative adversarial network:** The generative adversarial network trains a generator G with discriminator D by solving the following minimax optimization problem:

$$\min_G \max_D \qquad \mathbb{E}_{\mathbf{x}}[l(D(\mathbf{x}), 1)] + \mathbb{E}_{\mathbf{z}}[l(D(G(\mathbf{z})), 0)].$$

    Under the **mixup**, the input data becomes $\tilde{y} = \lambda \cdot 1 + (1-\lambda) \cdot 0 = \lambda$ and $\tilde{\mathbf{x}} = \mathbf{x} + (1-\lambda)G(\mathbf{z})$. The above minimax problem becomes

$$\min_G \max_D \qquad \mathbb{E}_{\mathbf{x},\mathbf{z}}[l(D(\mathbf{x} + (1-\lambda)G(\mathbf{z})), \lambda)].$$

Mixup

- **Mixup (contd):**
  - **Advantages of mixup**: It costs little to generate **mixup**-type virtual data $(\tilde{y}, \tilde{\mathbf{x}})$.
  - When training neural networks with the **mixup**-type virtual data:
    - Performances of ResNet-type neural networks on ImageNet data classification improve when setting Beta(α, α) with $\alpha \in [0.1, 0.4]$.
    - When combining **mixup** with the dropout technique, performances of neural networks can also improve on the corrupted label problem.
    - Performances of neural networks can improve significantly in the adversarial learning.

Semisupervised GAN

- **Semi-supervised learning via generative adversarial networks (Salimans et al., 2016):**
  - Assume the classifier model taking a form as

$$C(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) = \frac{\exp[\mathbf{y}^T \boldsymbol{\theta}(\mathbf{x})]}{\sum_{k=1}^{K} \exp[(\boldsymbol{\theta}(\mathbf{x}))_k]},$$

    where $\boldsymbol{\theta}(\mathbf{x})$ is a neural network with K-dimensional output that serves as a machines for extracting features from $\mathbf{x}$.

  - Under supervised learning, we train the neural network $\boldsymbol{\theta}$ by solving the following optimization problem:

$$\widehat{\boldsymbol{\theta}} = \arg\min \mathbb{E}_{\mathbf{y},\mathbf{x}}[l_C(\boldsymbol{\theta}; \mathbf{y}, \mathbf{x})],$$

    where $l_C(\boldsymbol{\theta}; \mathbf{y}, \mathbf{x})$ is the loss function corresponding to the learning problem.

Semisupervised GAN

- **Semisupervised learning GAN (contd):**
  - Consider an observation $\{\mathbf{x}', \cdot\}$ when $\mathbf{y}'$ is missing.
  - **Key idea:** Since $\mathbf{x}'$ is also observed from the **real** world, we can treat the observation as **real** in contrast to some **fake** (artificial) data point simulated from a **generator**.
  - We build a **discriminator** to distinguish a **real** observation from a **fake** one:

$$D(\mathbf{x}; \boldsymbol{\theta}) = \frac{\sum_{k=1}^{K} \exp[(\boldsymbol{\theta}(\mathbf{x}))_k]}{\sum_{k=1}^{K} \exp[(\boldsymbol{\theta}(\mathbf{x}))_k] + 1},$$

  - We build a **generator** to produce **fake** data point **u**:

$$\mathbf{u} = G(\mathbf{z}; \boldsymbol{\Phi}),$$

    where $\boldsymbol{\Phi}$ is a neural network model that transforms the vector **z** to the **fake** data point **u**.

Semisupervised GAN

- **Semisupervised learning GAN (contd):**
  - $\boldsymbol{\Phi}$ is learnt by using both **real** data $\mathbf{x}'$ and **fake** data $\mathbf{u}$.
  - Under the GAN training framework, the **real** data $\mathbf{x}'$ will be labeled with 1 while the **fake** data $\mathbf{u}$ will be labeled with 0.
  - The GAN training problem associated with the **discriminator** D and **generator** G can be formulated as follows:

$$(\widehat{\boldsymbol{\theta}}, \widehat{\boldsymbol{\Phi}}) = \arg \max_{\boldsymbol{\Phi}} \min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}',\mathbf{z}}[l_{D,G}(\boldsymbol{\theta}, \boldsymbol{\Phi}; \mathbf{x}', \mathbf{z})].$$

  - In practice we learn $\boldsymbol{\theta}$ and $\boldsymbol{\Phi}$ by solving the following estimation problem:

$$(\widehat{\boldsymbol{\theta}}, \widehat{\boldsymbol{\Phi}}) = \arg \max_{\boldsymbol{\Phi}} \min_{\boldsymbol{\theta}} \left\{ \mathbb{E}_{\mathbf{y},\mathbf{x}}[l_C(\boldsymbol{\theta}; \mathbf{y}, \mathbf{x})] + \mathbb{E}_{\mathbf{x}',\mathbf{z}}[l_{D,G}(\boldsymbol{\theta}, \boldsymbol{\Phi}; \mathbf{x}', \mathbf{z})] \right\},$$

Batch Normalization

- **Batch normalization (Ioffe and Szegedy, 2015):**
  - The **internal covariate shift** problem occurs when the distribution of inputs of the current layer is constantly changing due changes in parameters of previous layers during the training procedure.
  - According to the authors, due to such changes, it is difficult to determine the **learning rate (stepsize)** and the **initial values**, and therefore slows down the training procedure.
  - The authors introduced the **batch normalization (BN)** layer that carries out standardization (whitening) on inputs of a layer over **each training mini batch sample** using the sample mean and sample variance calculated with **each training mini batch sample**.
  - As claimed by the authors, it would reduce **internal covariate shift** as each of the inputs is standardized.

Batch Normalization

- **Key idea:**
  - **Batch normalization** aims to stabilize the input distribution by using the **moment matching** that forces the current input to have the mean and variance the same as the previous input.

- **Forward propagation:**
  - The **batch normalization** carried out standardization by using mini-batch statistics such as the mini-batch sample mean and variance. With an n-size mini-batch sample of inputs $\mathcal{B} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$, compute the following parameters:

$$
\begin{aligned}
\boldsymbol{\mu}_{\mathcal{B}} &= \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i, \\
\boldsymbol{\sigma}^2_{\mathcal{B}} &= \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \boldsymbol{\mu}_{\mathcal{B}})^2, \\
\widehat{\mathbf{x}}_i &= \text{diag}[(\boldsymbol{\sigma}^2_{\mathcal{B}} + \epsilon\mathbf{1})^{-1/2}](\mathbf{x}_i - \boldsymbol{\mu}_{\mathcal{B}}), \\
\mathbf{y}_i &= \gamma\widehat{\mathbf{x}}_i + \beta,
\end{aligned}
$$

  where γ and β are **scalar-valued learnable** parameters.

Batch Normalization

- **Backpropagation:**
  - Now let l be the loss function. The backpropagation corresponding to a BN layer is given as follows:

$$
\begin{aligned}
\frac{\partial l}{\partial \gamma} &= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial l}{\partial \mathbf{y}_i}\circ \widehat{\mathbf{x}}_i, \\
\frac{\partial l}{\partial \beta} &= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial l}{\partial \mathbf{y}_i}, \\
\frac{\partial l}{\partial \widehat{\mathbf{x}}_i} &= \frac{\partial l}{\partial \mathbf{y}_i}\cdot \gamma, \\
\frac{\partial l}{\partial \boldsymbol{\sigma}_{\mathcal{B}}^2} &= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial l}{\partial \widehat{\mathbf{x}}_i}\circ(\mathbf{x}_i-\boldsymbol{\mu}_{\mathcal{B}})\circ\left[\left(\frac{-1}{2}\right)(\boldsymbol{\sigma}_{\mathcal{B}}^2+\epsilon\mathbf{1})^{-3/2}\right], \\
\frac{\partial l}{\partial \boldsymbol{\mu}_{\mathcal{B}}} &= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial l}{\partial \widehat{\mathbf{x}}_i}\circ[-(\boldsymbol{\sigma}_{\mathcal{B}}^2+\epsilon\mathbf{1})^{-1/2}].
\end{aligned}
$$

Batch Normalization

- **Backpropagation (contd):**
  - The derivative of the loss function with respect to the input of the ith batch sample is

$$\begin{aligned}\frac{\partial l}{\partial \mathbf{x}_i} &= \frac{\partial \widehat{\mathbf{x}}_i^T}{\partial \mathbf{x}_i}\frac{\partial l}{\partial \widehat{\mathbf{x}}_i} + \frac{\partial \boldsymbol{\mu}_{\mathcal{B}}^T}{\partial \mathbf{x}_i}\frac{\partial l}{\partial \boldsymbol{\mu}_{\mathcal{B}}} + \frac{\partial (\boldsymbol{\sigma}_{\mathcal{B}}^2)^T}{\partial \mathbf{x}_i}\frac{\partial l}{\partial \boldsymbol{\sigma}_{\mathcal{B}}^2} \\ &= \frac{\partial l}{\partial \widehat{\mathbf{x}}_i} \circ [(\boldsymbol{\sigma}_{\mathcal{B}}^2 + \epsilon \mathbf{1})^{-1/2}] + \frac{\partial l}{\partial \boldsymbol{\mu}_{\mathcal{B}}} \circ \left[\frac{2}{n}(\mathbf{x}_i - \boldsymbol{\mu}_{\mathcal{B}})\right] + \frac{1}{n}\frac{\partial l}{\partial \boldsymbol{\sigma}_{\mathcal{B}}^2}.\end{aligned}$$

  - **Prediction:** According to the authors, when using a model with **BN** layers for prediction, one needs to compute the standardization for new sample. In this situation, the mean and variance of the **BN** layer are computed as follows (using m mini-batch samples):

$$\begin{aligned}\boldsymbol{\mu}_{\bar{\mathcal{B}}} &= \frac{1}{m}\sum_{j=1}^{m} \widehat{\boldsymbol{\mu}}_{\mathcal{B}_j}, \\ \boldsymbol{\sigma}_{\bar{\mathcal{B}}} &= \frac{1}{m}\sum_{j=1}^{m} \frac{n}{n-1}\widehat{\boldsymbol{\sigma}}_{\mathcal{B}_j}.\end{aligned}$$

Batch Normalization

- **Advantages of BN:** As claimed by the authors, the **BN** layer enables us to adopt a larger learning rate in the training procedure without needing to consider the gradient exploding problem.
- *Remark:* The **BN** layer is **scale-invariant**, that is $\mathrm{BN}(a\mathbf{x}) = \mathrm{BN}(\mathbf{x})$.
- **Some findings:**
  - Santurkar et al. (2018) pointed out that **batch normalization (BN)** does not solve the **internal covariate shift (ICS)** problem as claimed by its inventors.
  - Instead **batch normalization** leads to *more smoothness of the optimization landscape*, and therefore speeds up the training procedure as more aggressive learning rates can be used. It eventually leads to better performance.
  - It can be proved that the Lipschitzness of both the loss and its gradients are improved in the models with the **BN** layers: The gradients become more predictive under the models with the **BN** layers, and larger learning rates can be used to speed up training.

Batch Normalization

- **Question:** Is the effectiveness of the **BN** indeed related to internal covariate shift?
    - **Answer:** No. The effectiveness of the **BN** is related to changes of smoothness of the optimization landscape.
- **Question:** Is **BN** stabilization of layer input distributions even effective in reducing **ICS**?
    - **Answer:** No. **BN** may actually lead to increase in **ICS**.
- The claim that **BN** layers lead to more smoothness of the optimization landscape means that the gradients of the loss associated with the BN model is more "Lipschitz", i.e. the Lipschitz continuous condition is **tighter** than those associated with non-BN models.
- Consequently the l_2-norms of the gradients in the models with the **BN** layers are more regular than those associated with models without the **BN** layers. It is due to reparametrization of the **BN** layer.

Empirical Comparison

- B. Recht, R. Roelofs, L. Schmidt, V. Shankar (2019). Do ImageNet classifiers generalize to ImageNet? (`https://arxiv.org/abs/1902.10811`)
    - The authors investigated whether image classification models trained on CIFAR-10 data or ImageNet data can generalize to **newly created data**.
    - Requirements: The newly created data should follow distributions similar to CIFAR-10 data or ImageNet data.

Empirical Comparison

- The CIFAR-10 data:
  - Published in 2009.
  - A subset of Tiny Images, which is a database containing 80 million RGB color images with 32×32 pixels.
  - Consists of 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.
  - Each category has 6000 32×32 pixel images.
  - Both training and test data are class-balanced. The training dataset contains 50000 images while the test dataset contains 10000 images.

Empirical Comparison

- The ImageNet data:
  - Created from a database consisting of 14 million+ images annotated into $\sim$ 22000 categories.
  - The annotation tasks were done via Amazon Mechanical Turk (MTurk) with 49000 workers from 167 countries.
  - The images have no fixed sizes ($\sim 500 \times 400$ pixels).
  - The ImageNet team has run the yearly ImageNet Large Scale Visual Recognition Challenge (ILSVRC) from 2010.
  - The ILSVRC2012 competition dataset is the benchmark version, consisting of 1.2 million+ training images, 50000 validation images, and 100000 test images of 1000 categories.

Empirical Comparison

- Creating data similar to the CIFAR-10 data:
  - Images were collected from the Tiny Image database.
  - The annotation tasks were done by two graduate students.
  - The resulting dataset consists of 2000 images (32×32 pixels).
- Creating data similar to the ImageNet data:
  - Images were collected from the Flickr image hosting service.
  - The annotation tasks were done by recruiting workers from Amazon MTurks.
  - The resulting dataset consists of 10000 images.

Empirical Comparison

- Models subject for testing with the newly created data:
  - 34 models trained on CIFAR-10 data
  - 67 models trained on ImageNet data
  - Most models were trained by original developers.

Empirical Comparison

- Famous models:
  - (AutoAugment; CIFAR-10 only) E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. AutoAugment: Learning augmentation policies from data. `https://arxiv.org/abs/1805.09501`, *2018*.
  - (ResNet) K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CVPR, 2016*.
  - (DenseNet) G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *CVPR, 2017*.
  - (AlexNet; ImageNet only) A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS, 2012*.
  - (PnasNet; ImageNet only) C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, F.-F. Li, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. *ECCV, 2018*.
  - (VGG) K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. `https://arxiv.org/abs/1409.1556`, *2014*.
  - (Inception; ImageNet only) C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception architecture for computer vision. *CVPR, 2016*.

Empirical Comparison

- Results:
  - For VGG and ResNet, CIFAR-10 trained models suffer 8% drop in accuracy while ImageNet trained models suffer 11% drop in accuracy.
  - The accuracy of the best CIFAR-10 model only drops by 3% from 98.4% to 95.5%.
  - In contrast, the accuracy of the best ImageNet model drops by 11% (from 83% to 72%) in top-1 accuracy and a 6% drop (from 96% to 90%) in top-5 accuracy.
  - Models with higher accuracy on the original test data also have higher accuracy on the newly-created test data, suggesting that robustness improves as accuracy increases.

Empirical Comparison

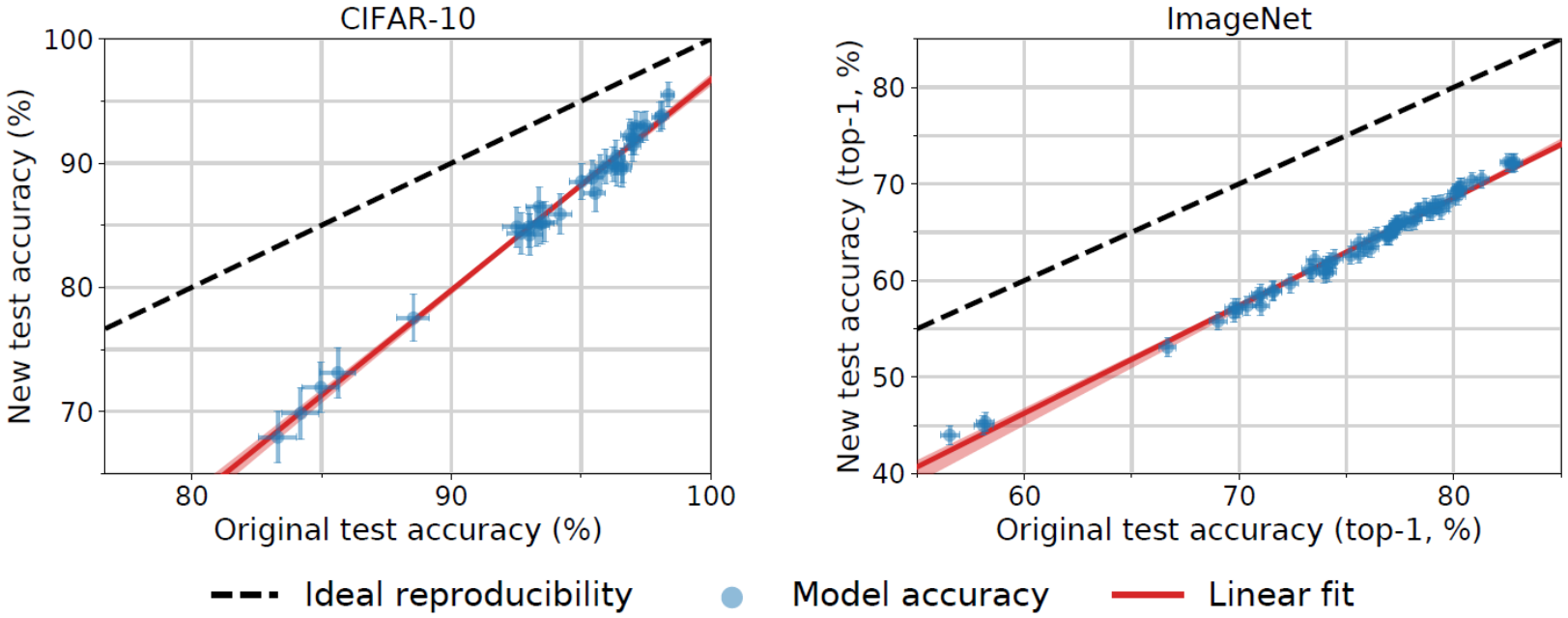

Figure: Accuracy for models tested with newly created data (Recht et al., 2019). Left: Models trained on the CIFAR-10 data; Right: Models trained on the ImageNet data.

Results

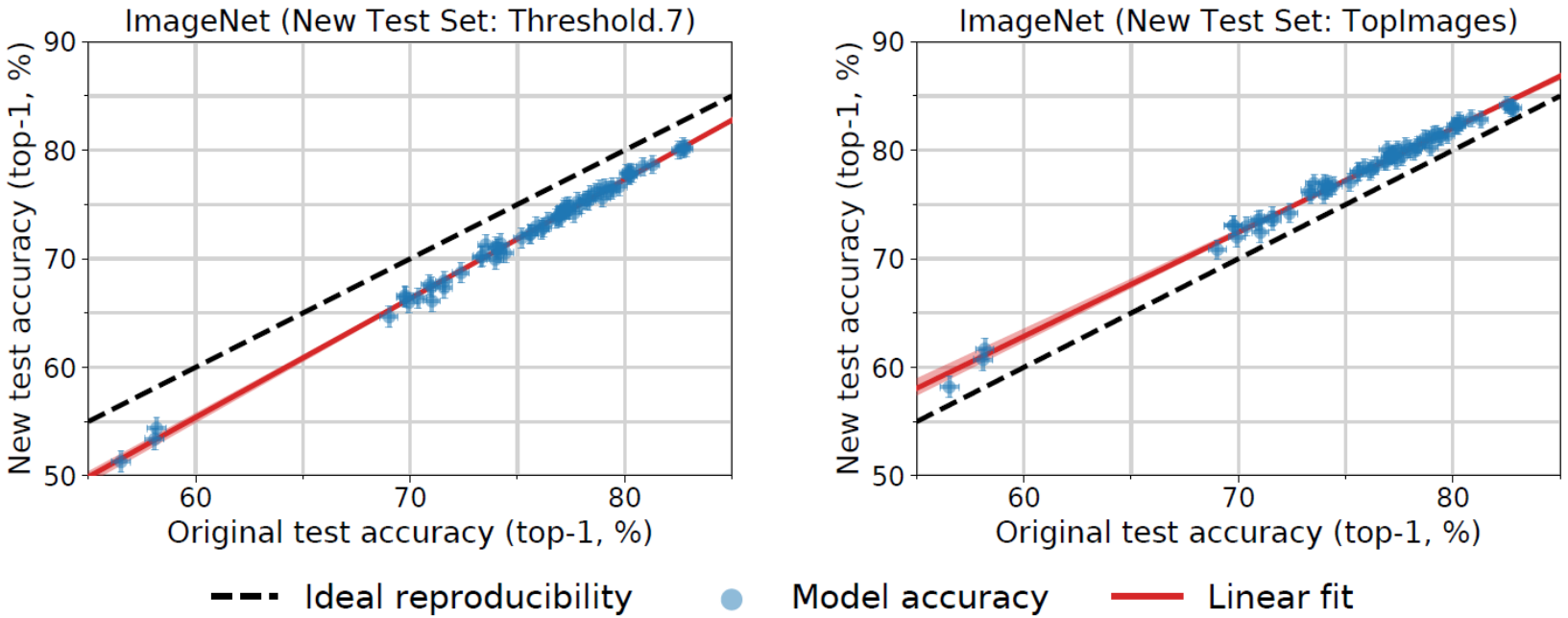

Figure: Accuracy for ImageNet models tested with newly created data (Recht et al., 2019). Left: Models tested with Threshold0.7 data; Right: Models tested with TopImage data.

References

F. Chollet (2017). *Deep Learning with Python*. Manning.

S. Ioffe and C. Szegedy (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. ICML.

T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida (2018). Spectral normalization for generative adversarial networks. ICLR.

R. Müller, S. Kornblith, and G. Hinton (2019). When does label smoothing help? NIPS.

B. Recht, R. Roelofs, L. Schmidt, V. Shankar (2019). Do ImageNet classifiers generalize to ImageNet? Arxiv.

S. Santurkar, D. Tsipras, A. Ilyas, A. Madry (2018). How does batch normalization help optimization?

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen (2016). Improved techniques for training GANs. NIPS.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (2016). Rethinking the inception architecture for computer vision. CVPR.

T. Tieleman and G. Hinton (2012). RMSProp, COURSERA: neural networks for machine learning. Technical report.

A. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht (2017). The marginal value of adaptive gradient methods in machine learning. NIPS.

H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz (2018). mixup: beyond empirical risk minimization. ICLR.

Lecture 3

Tso-Jung Yen

Institute of Statistical Science
Academia Sinica

tjyen@stat.sinica.edu.tw

Academia Sinica

August 12, 2020

Performance Measures

- **Basic concepts:**
  - We take binary classification problems as the example, in which cases are either classified as positive or negative.

    TP = number of positive cases who are **correctly** classified as positive,
    TN = number of negative cases who are **correctly** classified as negative,
    FN = number of positive cases who are **incorrectly** classified as negative,
    FP = number of negative cases who are **incorrectly** classified as positive.

  - The above four quantities are summarized in Table 1.

		prediction	
		positive	negative
truth	positive	TP	FN
	negative	FP	TN

Table: Confusion matrix.

  - *Remark:* Table 1 is also called the **confusion matrix**.

Performance Measures

- **Taxonomy of performance criteria:**
  - Below we consider three groups of performance measures.
  - These groups are not mutually exclusive, and indeed some of the performance measures have different names but share exactly the same mathematical definition.
  - **TPR, FPR, and FDR (statistics):**
    - **TPR** stands for **true positive rate**, **FPR** stands for **false positive rate**. They are defined as

$$\begin{aligned} \text{TPR} &= \frac{\text{TP}}{\text{TP} + \text{FN}}, \\ \text{FPR} &= \frac{\text{FP}}{\text{FP} + \text{TN}}, \end{aligned}$$

      respectively. **FDR** stands for **false discovery rate**, which is defined as

$$\text{FDR} = \frac{\text{FP}}{\text{TP} + \text{FP}}.$$

Performance Measures

- **Taxonomy of performance criteria (contd):**
  - **Sensitivity and specificity (medicine; engineering):**
    - **Sensitivity** is defined as

$$\text{Sensitivity} \quad = \quad \frac{\text{TP}}{\text{TP} + \text{FN}},$$

      which has exactly the same mathematical definition as **TPR**.

    - **Specificity** is defined as

$$\text{Specificity} \quad = \quad \frac{\text{TN}}{\text{FP} + \text{TN}},$$

      Remark: We have Specificity $= 1 -$ FPR, which is also called **true negative rate** or **TNR**.

Performance Measures

- **Taxonomy of performance criteria (contd):**
  - **Recall and precision (machine learning):**
    - **Recall** is defined as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}},$$

      which has exactly the same mathematical definition as **TPR**.
    - **Precision** is defined as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

      Remark: We have $\text{Precision} = 1 - \text{FDR}$.

Performance Measures

- Below we further introduce commonly-seen criteria for evaluating a model's performance in classification.

- **Accuracy (ACC):**

  - It is defined as

$$\begin{aligned} \text{ACC} &= \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{TN} + \text{FP}} \\ &= \frac{\text{number of correctly classified samples}}{\text{number of samples}}. \end{aligned}$$

    The value of **ACC** is between 0 and 1. The larger the **ACC** value, the better the model's performance.

Performance Measures

- **Top-k accuracy (Top-k-ACC):**
  - In multi-class (K classes) classification, one usually makes class prediction by first computing predicted probabilities $(\widehat{p}^1, \widehat{p}^2, \cdots, \widehat{p}^K)$ and ordering the probabilities from the largest to the smallest ones to make the final decision:

    $$\widehat{\mathcal{P}} = (\widehat{p}^{[1]}, \widehat{p}^{[2]}, \cdots, \widehat{p}^{[K]}),$$

    For sample i, define

    $$\widehat{\mathcal{C}}_i^k = \{\text{The indices of the first } k \text{ elements of } \widehat{\mathcal{P}}_i\}.$$

  - Top-1 accuracy is defined as

    $$\text{Top-1-ACC} = \sum_{i=1}^{n} \frac{\mathbb{I}\{y_i^{\text{true}} \in \widehat{\mathcal{C}}_i^1\}}{n}.$$

  - Top-k accuracy is defined as

    $$\text{Top-k-ACC} = \sum_{i=1}^{n} \frac{\mathbb{I}\{y_i^{\text{true}} \in \widehat{\mathcal{C}}_i^k\}}{n}.$$

Performance Measures

- **F1 score:**
  - It is defined as

$$F_1 = \frac{2\mathrm{TP}}{2\mathrm{TP} + \mathrm{FP} + \mathrm{FN}},$$

    which can be expressed as

$$F_1 = \frac{2\mathrm{Recall} \cdot \mathrm{Precision}}{\mathrm{Recall} + \mathrm{Precision}},$$

    i.e. the harmonic mean of **Recall** and **Precision**. The value of **F1 score** is between 0 and 1. The larger the F_1 value, the better the model's performance.

Performance Measures

- **Area under the ROC curve (AUC):**
  - **ROC** stands for **receiver operating characteristic**.
  - An ROC curve is a 2D plot of **Sensitivity** (y-axis) against $1 -$ **Specificity** (x-axis).
  - The value of the **AUC** is between 0 and 1. The larger the **AUC** value, the better the model's performance.
- **Matthews correlation coefficient (MCC):** It is defined as

$$\mathrm{MCC} = \frac{\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN}}{\sqrt{(\mathrm{TP} + \mathrm{FP})(\mathrm{TP} + \mathrm{FN})(\mathrm{TN} + \mathrm{FP})(\mathrm{TN} + \mathrm{FN})}}.$$

  The value of **MCC** is between -1 and 1. The larger the **MCC** value, the better the model's performance (MCC$= 1$ means perfect classification; MCC$= -1$ means totally wrong classification.)