Train A Softmax Classifier

CPT_S 434/534 Neural network design and application

Example: how to train a softmax classifier

The MNIST Dataset

- $n = 60{,}000$ training samples $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$.
- Each $\mathbf{x}_j$ is a 28×28 image.
- Each y_j is an integer in $\{0, 1, 2, \cdots, 9\}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

The MNIST Dataset

- $n = 60{,}000$ training samples $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)$.
- Each $\mathbf{x}_j$ is a 28×28 image.
- Each y_j is an integer in $\{0, 1, 2, \cdots, 9\}$.

Task: multi-class classification

- Given a 28×28 image, predict the digit.
- Learn a function $\mathbf{f}: \mathbb{R}^{28\times 28} \mapsto \mathbb{R}^{10}$.
- The i-th entry of $\mathbf{f}(\mathbf{x})$ indicates how likely the image $\mathbf{x}$ is the digit i.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

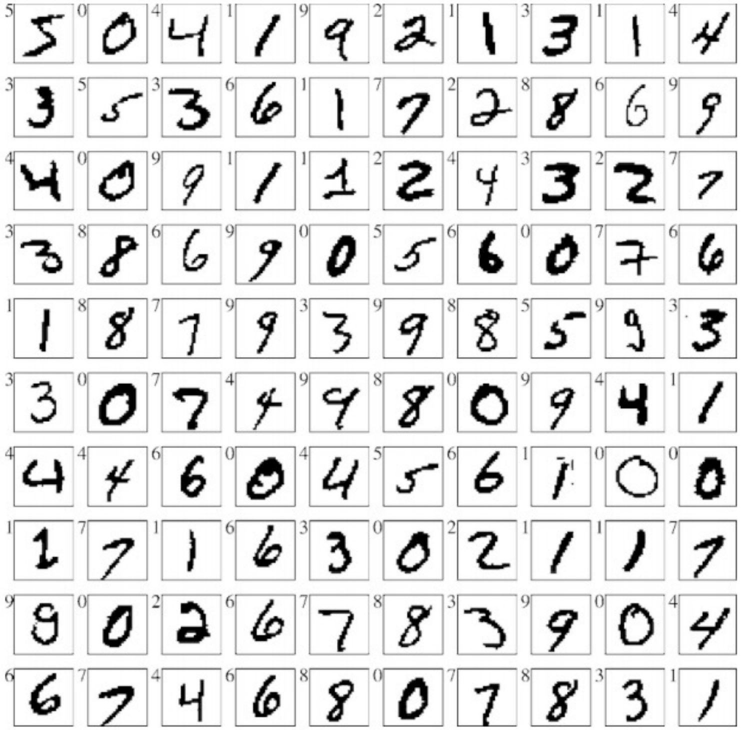
Linear model: softmax classifier

- Vectorize each 28×28 image to a 784-dim vector.
- Add a feature of all ones. (So $\mathbf{x}$ becomes 785-dim.)

  Bias term is absorbed

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



Linear model: softmax classifier

- Vectorize each 28×28 image to a 784-dim vector.
- Add a feature of all ones. (So $\mathbf{x}$ becomes 785-dim.)
- Let $\mathbf{W} \in \mathbb{R}^{10\times785}$ contain the parameters.
- Let $\mathbf{z} = \mathbf{W}\mathbf{x} \in \mathbb{R}^{10}$.
- Output a 10-dim vector:

$$\mathbf{f}(\mathbf{x}) = \mathrm{SoftMax}(\mathbf{z}).$$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Linear model: softmax classifier

- Vectorize each 28×28 image to a 784-dim vector.
- Add a feature of all ones. (So $\mathbf{x}$ becomes 785-dim.)
- Let $\mathbf{W} \in \mathbb{R}^{10\times 785}$ contain the parameters.
- Let $\mathbf{z} = \mathbf{W}\mathbf{x} \in \mathbb{R}^{10}$.
- Output a 10-dim vector:

$$\mathbf{f}(\mathbf{x}) = \text{SoftMax}(\mathbf{z}).$$

$$\text{SoftMax}(\mathbf{z}) = \frac{1}{\sum_{i=0}^{9} \exp(z_i)} [\exp(z_0), \cdots, \exp(z_9)]$$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Learn $\mathbf{W} \in \mathbb{R}^{10\times 785}$ from the training data

- One-hot encode of the labels
  - Originally, a label is a scalar in $\{0, 1, 2, \cdots, 9\}$.
  - The one-hot encode $\mathbf{y}$ is a 10-dim vector $\{0,1\}^{10}$.
  - E.g., the one-hot encode of 2 is $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Learn $\mathbf{W} \in \mathbb{R}^{10\times 785}$ from the training data

- One-hot encode of the labels
  - Originally, a label is a scalar in $\{0, 1, 2, \cdots, 9\}$.
  - The one-hot encode $\mathbf{y}$ is a 10-dim vector $\{0,1\}^{10}$.
  - E.g., the one-hot encode of 2 is $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$.
- Cross-entropy loss:

$$\text{CrossEntropy}(\mathbf{y}, \mathbf{f}) = -\sum_{i=0}^{9} y_i \cdot \log(f_i).$$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Learn $\mathbf{W} \in \mathbb{R}^{10\times 785}$ from the training data

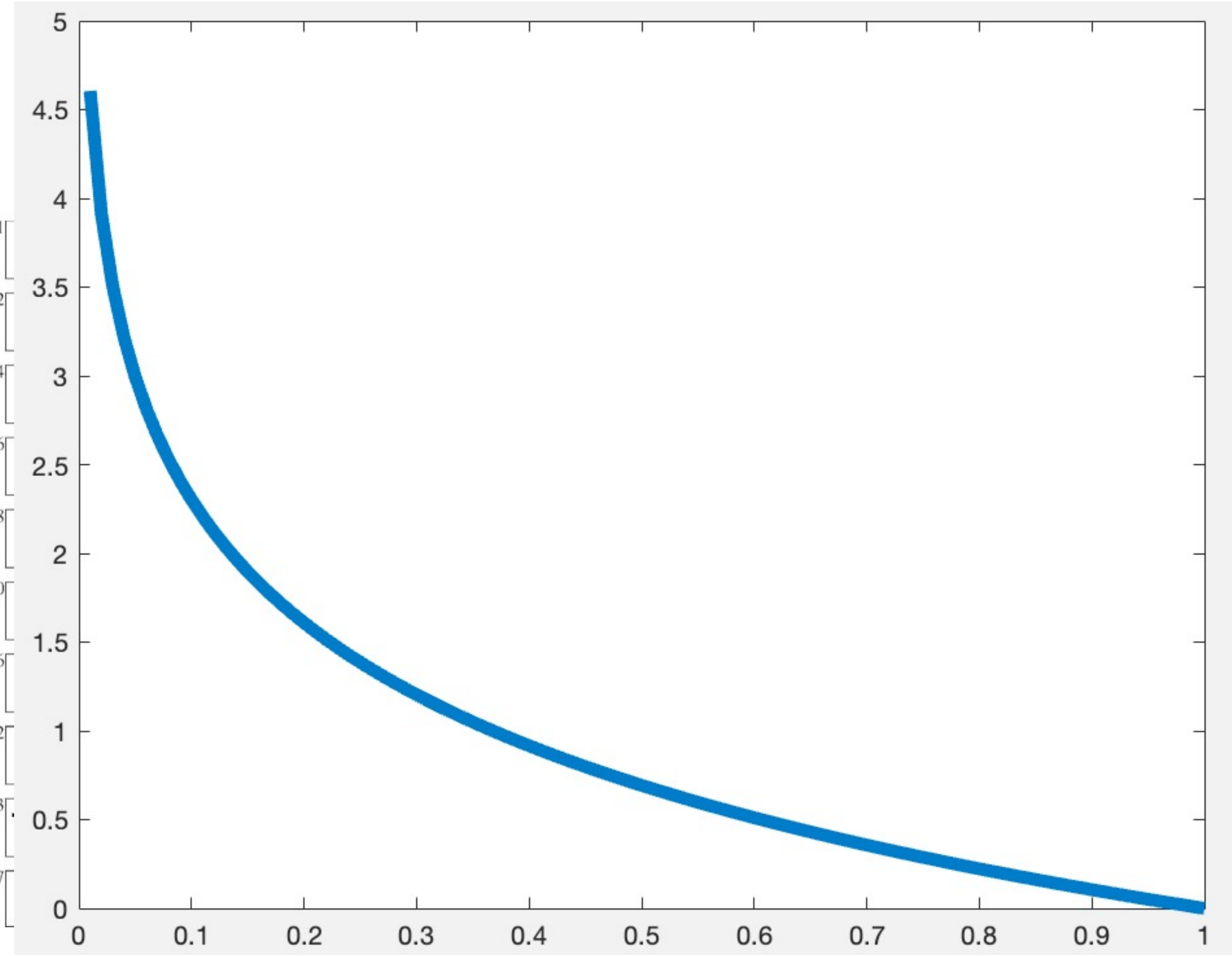
- $\boxed{\text{One-hot}}$ encode of the labels
  - Originally, a label is a scalar in $\{0, 1, 2, \cdots, 9\}$.
  - The one-hot encode $\mathbf{y}$ is a 10-dim vector $\{0,1\}^{10}$.
  - E.g., the one-hot encode of 2 is $[0, 0, \boxed{1}, 0, 0, 0, 0, 0, 0, 0]$.
- Cross-entropy loss:

$$\text{CrossEntropy}(\mathbf{y}, \mathbf{f}) = -\sum_{i=0}^{9} y_i \cdot \log(f_i).$$

$\boxed{\text{Q: how to interpret CE loss?}}$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



e training data

1, 2, ⋯ , 9}.

ı vector $\{0,1\}^{10}$.

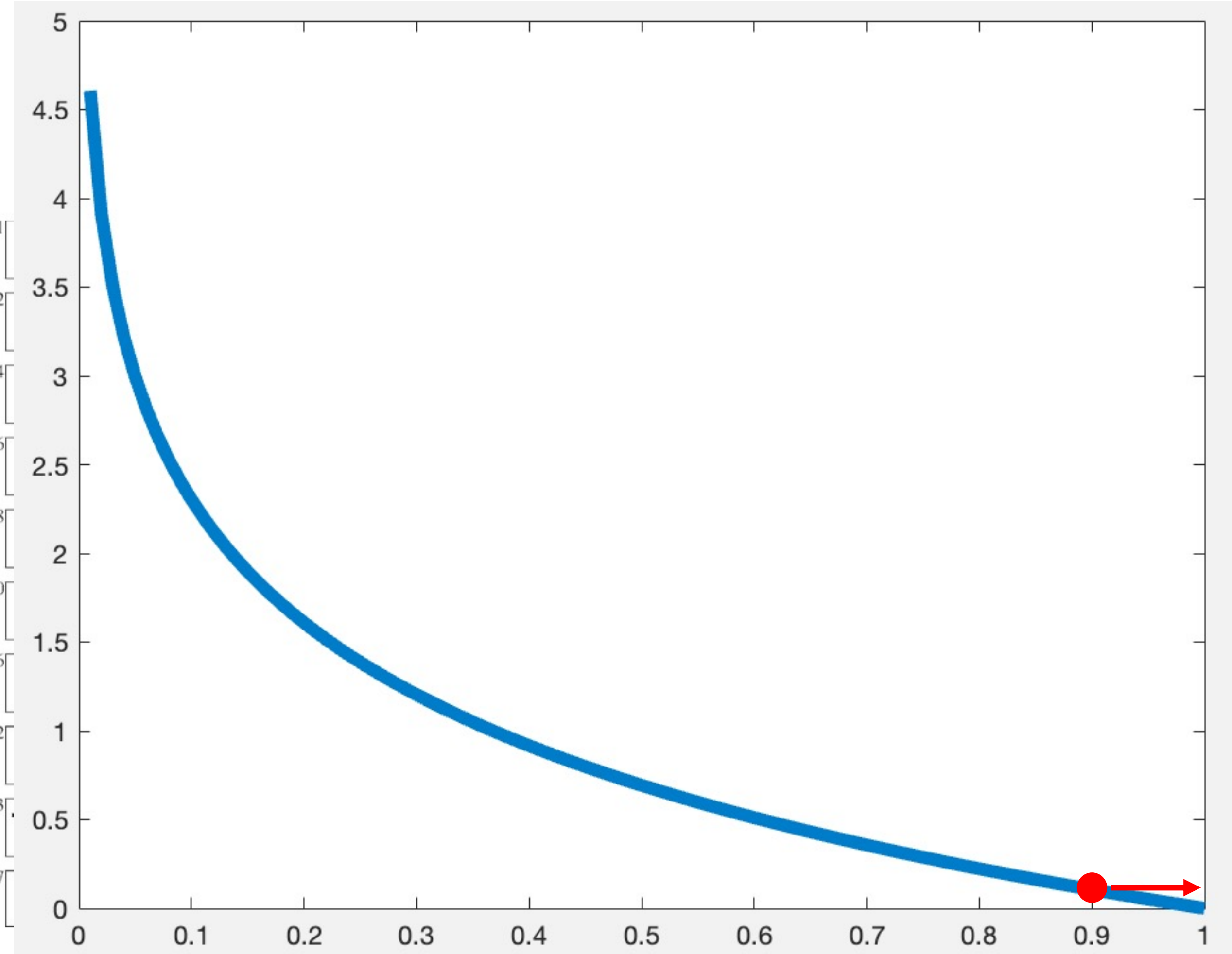
), 0, 1, 0, 0, 0, 0, 0, 0, 0].

$\Sigma_{i=0}^{9} y_i \cdot \log(f_i)$.

$-\log(f_i)$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



e training data

$1, 2, \cdots, 9\}$.

n vector $\{0,1\}^{10}$.

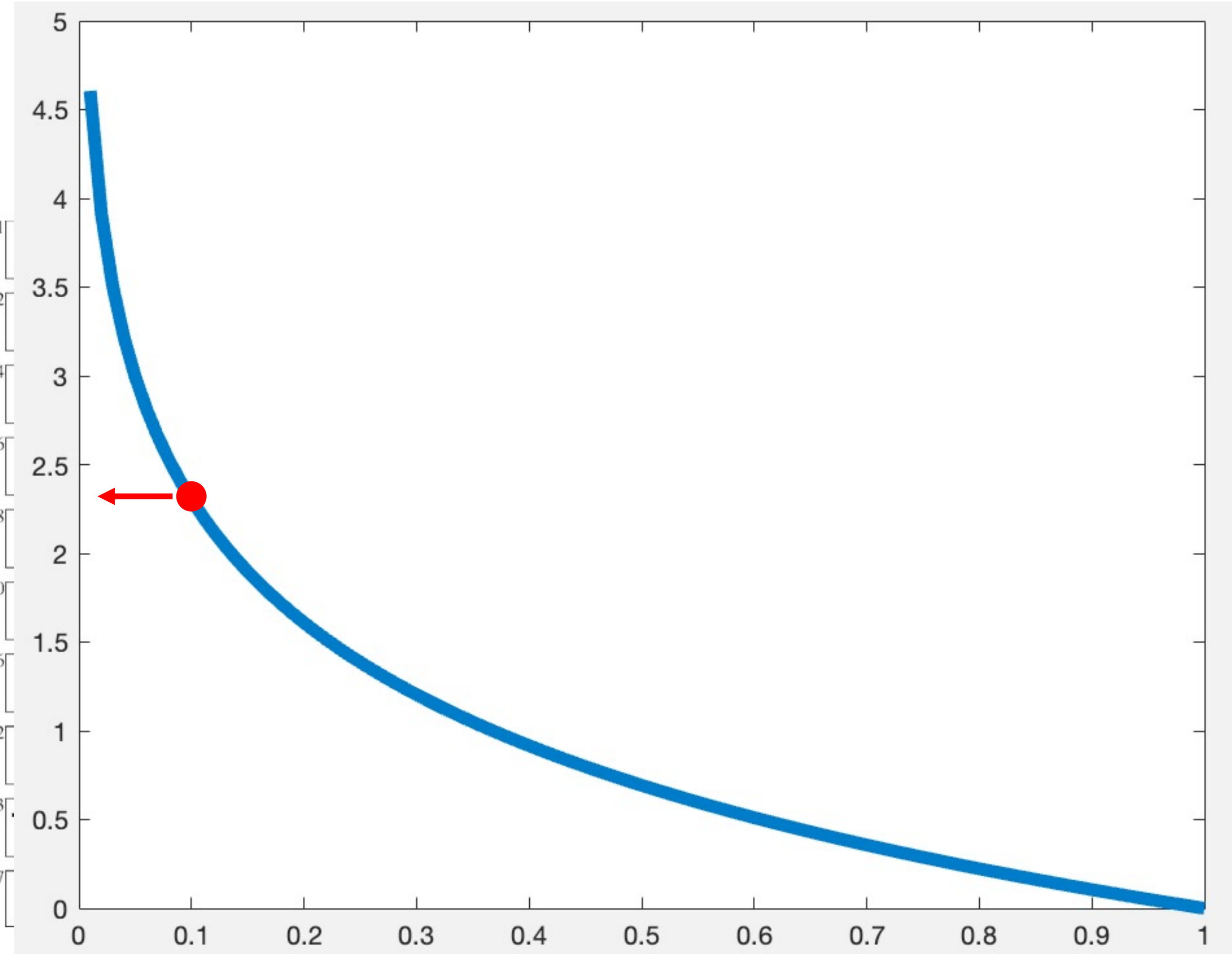
$), 0, 1, 0, 0, 0, 0, 0, 0, 0]$.

$\sum_{i=0}^{9} y_i \cdot \log(f_i)$.

$-\log(f_i)$ $\qquad$ $-\, y_i \cdot \log(f_i) \rightarrow 0$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



e training data

$1, 2, \cdots, 9\}$.

n vector $\{0,1\}^{10}$.

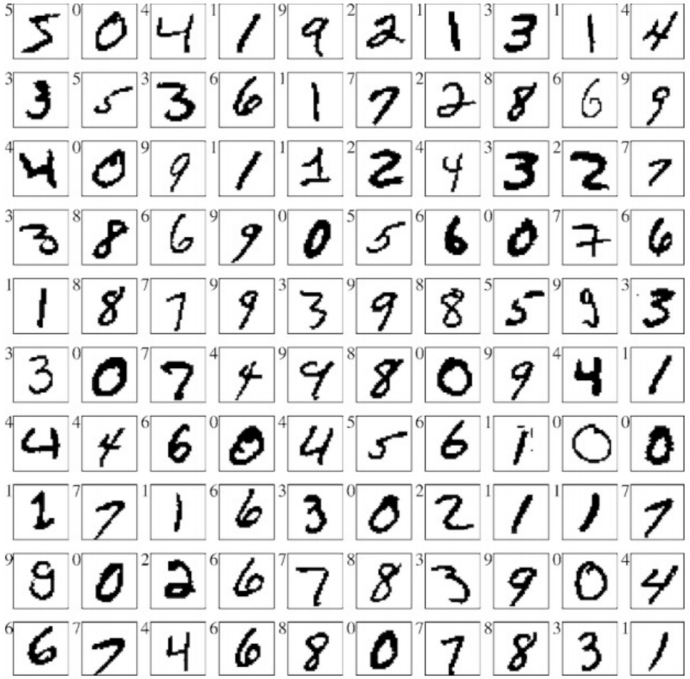
$0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$.

$\sum_{i=0}^{9} y_i \cdot \log(f_i)$.

$-y_i \cdot \log(f_i) \rightarrow \infty$ $\quad$ $-\log(f_i)$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



Learn $\mathbf{W} \in \mathbb{R}^{10\times 785}$ from the training data

- One-hot encode of the labels
  - Originally, a label is a scalar in $\{0, 1, 2, \cdots, 9\}$.
  - The one-hot encode $\mathbf{y}$ is a 10-dim vector $\{0,1\}^{10}$.
  - E.g., the one-hot encode of 2 is $[0, 0, \boxed{1}, 0, 0, 0, 0, 0, 0, 0]$.
- Cross-entropy loss:

$$\text{CrossEntropy}(\mathbf{y}, \mathbf{f}) = -\sum_{i=0}^{9} y_i \cdot \log(f_i).$$

- Solve the optimization model:

$$\mathbf{W}^\star = \underset{\mathbf{W}}{\text{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \text{CrossEntropy}\Big(\mathbf{y}_j, \mathbf{f}(\mathbf{x}_j)\Big) \right\}.$$

W is the parameter of **f**

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

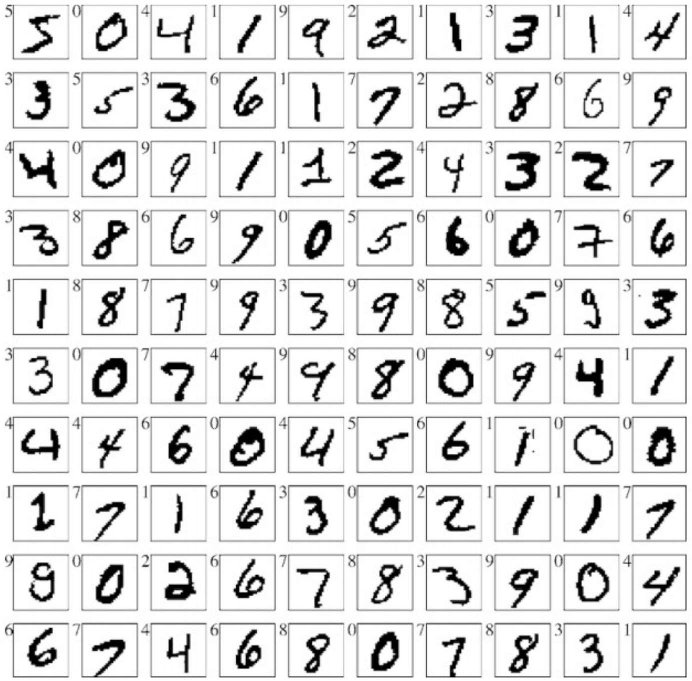
Example: how to train a softmax classifier

Make prediction for a test sample $\mathbf{x}'$

- Now we have $\mathbf{W}^\star \in \mathbb{R}^{10\times 785}$.
- For a test sample $\mathbf{x}'$, compute $\boxed{\mathbf{z} = \mathbf{W}^\star \mathbf{x}' \in \mathbb{R}^{10}}$.
- Make prediction by $\boxed{\text{argmax}\, \mathbf{z}.}$
  - If the 7-th entry of $\mathbf{z}$ is the largest, then the model thinks the image is digit "7".

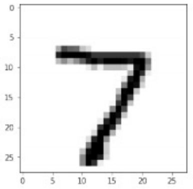
Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier



Make prediction for a test sample $\mathbf{x}'$

- Now we have $\mathbf{W}^\star \in \mathbb{R}^{10\times 785}$.
- For a test sample $\mathbf{x}'$, compute $\mathbf{z} = \mathbf{W}^\star\mathbf{x}' \in \mathbb{R}^{10}$.
- Make prediction by $\operatorname{argmax} \mathbf{z}$.
    - If the 7-th entry of $\mathbf{z}$ is the largest, then the model thinks the image is digit "7".



$\mathbf{z} = [\,-55.7, -141.4\,, 18.1\,, 188.3\,, -91.3\,, -26.8\,, -183.6\,, \boxed{411.2}\,, -142.1\,, 96.2]$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x} \in \mathbb{R}^{785}$.
- $\mathbf{z} = \boxed{\mathbf{W}\,\mathbf{x} \in \mathbb{R}^{10}.}$
- **Output:** $\boxed{\mathbf{f}(\mathbf{x}) = \mathrm{SoftMax}(\mathbf{z}).}$

Trainable parameters: $\mathbf{W} \in \mathbb{R}^{10\times 785}$

Train the function by empirical risk minimization (ERM):

- **Training set:** $(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_n, \mathbf{y}_n) \in \mathbb{R}^{785} \times \mathbb{R}^{10}$.
- **Loss function:** $\mathrm{CrossEntropy}(\mathbf{y}, \mathbf{f}) = -\sum_{i=1}^{10} y_i \cdot \log(\mathbf{f}(\mathbf{x})_i)$.
- **Solve ERM:** $\boxed{\underset{\mathbf{W}}{\mathrm{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \mathrm{CrossEntropy}\left(\mathbf{y}_j, \mathbf{f}(\mathbf{x}_j)\right) \right\}.}$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

- **How to solve** $\underset{\mathbf{W}}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \text{CrossEntropy}\left(\mathbf{y}_j, \mathbf{f}(\mathbf{x}_j)\right) \right\}$?

- **Stochastic gradient descent** $\boxed{\textbf{(SGD)}}$ **with momentum** repeats:
  1. Randomly pick $\boxed{j \text{ from } \{1, 2, \cdots, n\}}$.
  2. Evaluate the gradient $\boxed{\mathbf{G}_j = \frac{\partial \text{CrossEntropy}(\mathbf{y}_j, \mathbf{f}(\mathbf{x}_j))}{\partial \mathbf{W}}} \Big|_{\mathbf{W}=\mathbf{W}_{\text{old}}}$.
  3. Update the momentum: $\mathbf{V}_{\text{new}} = \beta \mathbf{V}_{\text{old}} + \mathbf{G}_j$.
  4. Update $\mathbf{W}$ by $\mathbf{W}_{\text{new}} \leftarrow \mathbf{W}_{\text{old}} - \alpha \, \mathbf{V}_{\text{new}}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(1)} = \mathrm{SoftMax}\left(\mathbf{z}^{(1)}\right) \in \mathbb{R}^{d_1}$.
- **Output:** $\mathbf{f}\left(\mathbf{x}^{(0)}\right) = \mathbf{x}^{(1)}$.

A linear model

Trainable parameter:

- $\mathbf{W}^{(0)} \in \mathbb{R}^{10 \times 785}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input**: vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.

Hidden Layer 1

- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.

Hidden Layer 2

- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.

Output Layer

- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.

- **Output**: $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

MLP

Trainable parameters:

- $\mathbf{W}^{(0)} \in \mathbb{R}^{d_1 \times 785}$,
- $\mathbf{W}^{(1)} \in \mathbb{R}^{d_2 \times d_1}$,
- $\mathbf{W}^{(2)} \in \mathbb{R}^{10 \times d_2}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}\left(\mathbf{z}^{(3)}\right) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}\left(\mathbf{x}^{(0)}\right) = \mathbf{x}^{(3)}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

Build an optimization model:

$$\underset{\mathbf{W}^{(0)},\mathbf{W}^{(1)},\mathbf{W}^{(2)}}{\mathrm{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \boxed{\mathrm{Loss}\big(\mathbf{f}(\mathbf{x}_j),\ \mathbf{y}_j\big)} \right\}$$

E.g., the cross-entropy loss

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{0, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{0, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to solve

$$\operatorname*{argmin}_{\mathbf{W}^{(0)},\mathbf{W}^{(1)},\mathbf{W}^{(2)}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \mathrm{Loss}\left(\mathbf{f}(\mathbf{x}_j),\ \mathbf{y}_j \right) \right\} ?$$

Stochastic gradient descent (SGD):

- $\boxed{\text{Randomly pick } j \text{ from } \{1, 2, \cdots, n\}.}$
- Compute the stochastic gradient w.r.t. $\mathbf{W}^{(0)}$ at the current iteration $\mathbf{W}_{\mathrm{old}}^{(0)}$:

$$\mathbf{g}_j^{(0)} = \frac{\partial\, \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(0)}} \Big|_{\mathbf{W}^{(0)} = \mathbf{W}_{\mathrm{old}}^{(0)}}.$$

- Update $\mathbf{W}^{(0)}$: $\mathbf{W}_{\mathrm{new}}^{(0)} = \mathbf{W}_{\mathrm{old}}^{(0)} - \alpha\, \mathbf{g}_j^{(0)}$.
- Do the same for $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{0, \ \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{0, \ \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to solve

$$\underset{\mathbf{W}^{(0)},\mathbf{W}^{(1)},\mathbf{W}^{(2)}}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \mathrm{Loss}\big(\mathbf{f}(\mathbf{x}_j), \ \mathbf{y}_j \big) \right\} ?$$

Stochastic gradient descent (SGD):

- Randomly pick j from $\{1, 2, \cdots, n\}$.
- Compute the stochastic gradient w.r.t. $\mathbf{W}^{(0)}$ at the current iteration $\mathbf{W}^{(0)}_{\text{old}}$:

$$\mathbf{g}_j^{(0)} = \frac{\partial \, \mathrm{Loss}\big(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j \big)}{\partial \mathbf{W}^{(0)}} \Big|_{\mathbf{W}^{(0)} = \mathbf{W}^{(0)}_{\text{old}}}.$$

- Update $\mathbf{W}^{(0)}$: $\mathbf{W}^{(0)}_{\text{new}} = \mathbf{W}^{(0)}_{\text{old}} - \alpha \, \mathbf{g}_j^{(0)}$.
- Do the same for $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to solve

$$\underset{\mathbf{W}^{(0)},\mathbf{W}^{(1)},\mathbf{W}^{(2)}}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \mathrm{Loss}\big(\mathbf{f}(\mathbf{x}_j),\ \mathbf{y}_j \big) \right\} ?$$

Stochastic gradient descent (SGD):

- Randomly pick j from $\{1, 2, \cdots, n\}$.
- Compute the stochastic gradient w.r.t. $\mathbf{W}^{(0)}$ at the current iteration $\mathbf{W}_{\mathrm{old}}^{(0)}$:

$$\mathbf{g}_j^{(0)} = \frac{\partial\, \mathrm{Loss}\big(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j\big)}{\partial \mathbf{W}^{(0)}} \Big|_{\mathbf{W}^{(0)} = \mathbf{W}_{\mathrm{old}}^{(0)}}.$$

- Update $\mathbf{W}^{(0)}$: $\mathbf{W}_{\mathrm{new}}^{(0)} = \mathbf{W}_{\mathrm{old}}^{(0)} - \alpha\, \mathbf{g}_j^{(0)}$.
- Do the same for $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{0, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{0, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to solve

$$\underset{\mathbf{W}^{(0)},\mathbf{W}^{(1)},\mathbf{W}^{(2)}}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{j=1}^{n} \mathrm{Loss}\big(\mathbf{f}(\mathbf{x}_j),\ \mathbf{y}_j \big) \right\} ?$$

Stochastic gradient descent (SGD):

- Randomly pick j from $\{1, 2, \cdots, n\}$.
- Compute the stochastic gradient w.r.t. $\mathbf{W}^{(0)}$ at the current iteration $\mathbf{W}^{(0)}_{\text{old}}$:

$$\mathbf{g}_j^{(0)} = \frac{\partial\, \mathrm{Loss}\big(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j\big)}{\partial \mathbf{W}^{(0)}} \Big|_{\mathbf{W}^{(0)} = \mathbf{W}^{(0)}_{\text{old}}}.$$

- Update $\mathbf{W}^{(0)}$: $\mathbf{W}^{(0)}_{\text{new}} = \mathbf{W}^{(0)}_{\text{old}} - \alpha\, \mathbf{g}_j^{(0)}$.
- Do the same for $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

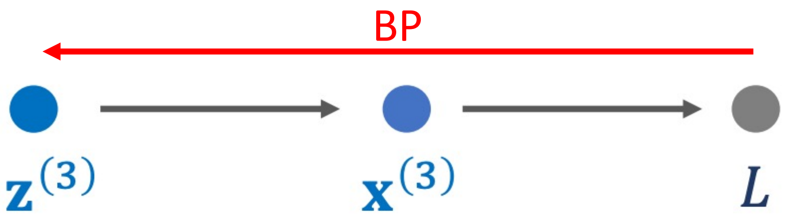
- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{0, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{0, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\boxed{\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}.}$
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\frac{\partial \, \mathrm{Loss}(\, \mathbf{f}(\mathbf{x}_j), \mathbf{y}_j \,)}{\boxed{\partial \mathbf{W}^{(k)}}}$?

Backpropagation:

- Denote $L = \mathrm{Loss}(\, \mathbf{f}(\mathbf{x}_j), \; \mathbf{y}_j \,)$.
- Compute $\boxed{\frac{\partial \, L}{\partial \mathbf{z}^{(3)}}}$.

10-dim vector



Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\boxed{\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}.}$
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\dfrac{\partial \, \mathrm{Loss}(\, \mathbf{f}(\mathbf{x}_j), \mathbf{y}_j \,)}{\partial \mathbf{W}^{(k)}}$ **?**

Backpropagation:

- Denote $L = \mathrm{Loss}(\, \mathbf{f}(\mathbf{x}_j), \; \mathbf{y}_j \,)$.
- Compute $\boxed{\dfrac{\partial L}{\partial \mathbf{z}^{(3)}}}$.
- $\dfrac{\partial L}{\partial \mathbf{z}^{(2)}} = \dfrac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}} \boxed{\dfrac{\partial L}{\partial \mathbf{z}^{(3)}}}$, $\qquad \dfrac{\partial L}{\partial \mathbf{W}^{(2)}} = \dfrac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} \boxed{\dfrac{\partial L}{\partial \mathbf{z}^{(3)}}}$.

$\mathbf{z}^{(3)}$ is a function of $\mathbf{z}^{(2)}$ and $\mathbf{W}^{(2)}$.

Apply the chain rule.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\boxed{\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}.}$
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\dfrac{\partial\, \mathrm{Loss}(\,\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j\,)}{\partial \mathbf{W}^{(k)}}$?

Backpropagation:

- Denote $L = \mathrm{Loss}(\,\mathbf{f}(\mathbf{x}_j),\ \mathbf{y}_j\,)$.
- Compute $\dfrac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\dfrac{\partial L}{\partial \mathbf{z}^{(2)}} = \boxed{\dfrac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}}} \dfrac{\partial L}{\partial \mathbf{z}^{(3)}}, \qquad \boxed{\dfrac{\partial L}{\partial \mathbf{W}^{(2)}}} = \dfrac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} \dfrac{\partial L}{\partial \mathbf{z}^{(3)}}$.

Use it to update $\mathbf{W}^{(2)}$ (e.g., by SGD).

$$\frac{\partial z^{(3)}}{\partial x^{(2)}} = w^{(2)}, \qquad \frac{\partial x^{(2)}}{\partial z^{(2)}} = \begin{cases} 1, & \text{if } z^{(2)} > 0; \\ 0, & \text{else.} \end{cases}$$

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\boxed{\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}.}$
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\frac{\partial \,\mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(k)}}$ **?**

Backpropagation:

- Denote $L = \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)$.
- Compute $\frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\boxed{\frac{\partial L}{\partial \mathbf{z}^{(2)}}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(1)}} \boxed{\frac{\partial L}{\partial \mathbf{z}^{(2)}}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{W}^{(1)}} \boxed{\frac{\partial L}{\partial \mathbf{z}^{(2)}}}$.

Apply the chain rule again.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\boxed{\mathbf{x}^{(1)} = \max\{\mathbf{0}, \ \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}.}$
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \ \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\dfrac{\partial \, \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(k)}}$ **?**

Backpropagation:

- Denote $L = \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \ \mathbf{y}_j)$.
- Compute $\dfrac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\dfrac{\partial L}{\partial \mathbf{z}^{(2)}} = \dfrac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}} \dfrac{\partial L}{\partial \mathbf{z}^{(3)}}$, $\quad \dfrac{\partial L}{\partial \mathbf{W}^{(2)}} = \dfrac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} \dfrac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\dfrac{\partial L}{\partial \mathbf{z}^{(1)}} = \dfrac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(1)}} \dfrac{\partial L}{\partial \mathbf{z}^{(2)}}$, $\quad \boxed{\dfrac{\partial L}{\partial \mathbf{W}^{(1)}}} = \dfrac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{W}^{(1)}} \dfrac{\partial L}{\partial \mathbf{z}^{(2)}}$.

Use it to update $\mathbf{W}^{(1)}$ (e.g., by SGD).

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\boxed{\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}.}$
- $\mathbf{x}^{(1)} = \max\{0, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{0, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\frac{\partial\, \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(k)}}$ **?**

Backpropagation:

- Denote $L = \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j),\ \mathbf{y}_j)$.
- Compute $\frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\boxed{\frac{\partial L}{\partial \mathbf{z}^{(1)}}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(1)}} \frac{\partial L}{\partial \mathbf{z}^{(2)}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{W}^{(1)}} \frac{\partial L}{\partial \mathbf{z}^{(2)}}$.
- $\frac{\partial L}{\partial \mathbf{W}^{(0)}} = \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(0)}} \boxed{\frac{\partial L}{\partial \mathbf{z}^{(1)}}}$.

Apply the chain rule again.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

Define a function $\mathbf{f}: \mathbb{R}^{785} \mapsto \mathbb{R}^{10}$:

- **Input:** vector $\mathbf{x}^{(0)} \in \mathbb{R}^{785}$.
- $\mathbf{z}^{(1)} = \mathbf{W}^{(0)} \mathbf{x}^{(0)} \in \mathbb{R}^{d_1}$.
- $\mathbf{x}^{(1)} = \max\{\mathbf{0}, \mathbf{z}^{(1)}\} \in \mathbb{R}^{d_1}$.
- $\mathbf{z}^{(2)} = \mathbf{W}^{(1)} \mathbf{x}^{(1)} \in \mathbb{R}^{d_2}$.
- $\mathbf{x}^{(2)} = \max\{\mathbf{0}, \mathbf{z}^{(2)}\} \in \mathbb{R}^{d_2}$.
- $\mathbf{z}^{(3)} = \mathbf{W}^{(2)} \mathbf{x}^{(2)} \in \mathbb{R}^{10}$.
- $\mathbf{x}^{(3)} = \mathrm{SoftMax}(\mathbf{z}^{(3)}) \in \mathbb{R}^{10}$.
- **Output:** $\mathbf{f}(\mathbf{x}^{(0)}) = \mathbf{x}^{(3)}$.

How to compute $\frac{\partial\, \mathrm{Loss}(\,\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j\,)}{\partial \mathbf{W}^{(k)}}$?

Backpropagation:

- Denote $L = \mathrm{Loss}(\,\mathbf{f}(\mathbf{x}_j),\ \mathbf{y}_j\,)$.
- Compute $\frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{z}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathbf{z}^{(3)}}{\partial \mathbf{W}^{(2)}} \frac{\partial L}{\partial \mathbf{z}^{(3)}}$.
- $\frac{\partial L}{\partial \mathbf{z}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{z}^{(1)}} \frac{\partial L}{\partial \mathbf{z}^{(2)}}$, $\quad \frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial \mathbf{z}^{(2)}}{\partial \mathbf{W}^{(1)}} \frac{\partial L}{\partial \mathbf{z}^{(2)}}$.
- $\boxed{\frac{\partial L}{\partial \mathbf{W}^{(0)}}} = \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(0)}} \frac{\partial L}{\partial \mathbf{z}^{(1)}}$. Use it to update $\mathbf{W}^{(0)}$.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

1. Randomly pick a sample $(\mathbf{x}_j, \mathbf{y}_j)$.
2. Run a forward pass (from the input $\mathbf{x}^{(0)}$ to the prediction).
3. Run a backward pass (from the loss to $\mathbf{W}^{(0)}$).

Get the derivatives (stochastic gradients):

$$\frac{\partial \text{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(2)}}, \quad \frac{\partial \text{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(1)}}, \quad \frac{\partial \text{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(0)}}.$$

Update $\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \mathbf{W}^{(2)}$ using the derivatives.

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

1. Randomly pick ~~a sample $(\mathbf{x}_j, \mathbf{y}_j)$~~. Several random samples.
2. Run a forward pass (from the input $\mathbf{x}^{(0)}$ to the prediction).
3. Run a backward pass (from the loss to $\mathbf{W}^{(0)}$).

Get the derivatives (stochastic gradients):

$$\cancel{\frac{\partial \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(2)}}, \frac{\partial \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(1)}}, \frac{\partial \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(0)}}}.$$

$$\frac{1}{|\mathcal{J}|}\sum_{j \in \mathcal{J}} \frac{\partial \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(2)}}, \quad \frac{1}{|\mathcal{J}|}\sum_{j \in \mathcal{J}} \frac{\partial \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(1)}}, \quad \frac{1}{|\mathcal{J}|}\sum_{j \in \mathcal{J}} \frac{\partial \mathrm{Loss}(\mathbf{f}(\mathbf{x}_j), \mathbf{y}_j)}{\partial \mathbf{W}^{(0)}}.$$

Mini-batch should always be used! Set batch size $|\mathcal{J}|$ to 16, 32, 64, …

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Example: how to train a softmax classifier

SGD: $\text{BatchSize} = 1.$

- Per-iteration cost is low.
- Lots of iterations to converge.

Mini-Batch: $\text{BatchSize} > 1.$

- Better than the other two, if BatchSize is properly set.

Full Gradient: $\text{BatchSize} = n.$

- Per-iteration cost is n times higher than SGD.
- Convex problem: less number of iterations.
- Neural network: it doesn't work!

See some blogs
https://distill.pub/2017/momentum/
https://ruder.io/optimizing-gradient-descent/

Credit for Shusen Wang https://github.com/wangshusen/DeepLearning/blob/master/Slides/6_NeuralNet_1.pdf

Rethink BP and chain rule

$$f_n\left(\dots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

$$f_i \rightarrow x_i$$

$$\frac{dx_n}{dx_1} = \frac{dx_n}{dx_{n-1}} \cdot \frac{dx_{n-1}}{dx_{n-2}} \cdot \dots \cdot \frac{dx_2}{dx_1} \cdot \frac{dx_1}{dx}$$

$$\frac{dx_n}{dx_i}, for\ i = 1, \dots, n-1$$

Rethink BP and chain rule

$$f(x) \rightarrow \nabla f(x)$$

$$f(g(x)) \rightarrow ?$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \boxed{\frac{dz}{dy}}\boxed{\frac{dy}{dx}}$$

Rethink BP and chain rule

$$f(x) \to \nabla f(x)$$

However, we use **stochastic gradient**, rather than **gradient**

$$f(g(x)) \to ?$$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \boxed{\frac{dz}{dy}}\boxed{\frac{dy}{dx}}$$

Rethink BP and chain rule

$f(x) \rightarrow \nabla f(x)$

$f(g(x)) \rightarrow ?$

However, we use **stochastic gradient**, rather than **gradient**

Stochastic $\widehat{\nabla} f(y) \rightarrow \nabla f(y)$ (approximation)

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \boxed{\frac{dz}{dy}}\boxed{\frac{dy}{dx}}$$

Rethink BP and chain rule

$f(x) \rightarrow \nabla f(x)$

However, we use **stochastic gradient**, rather than **gradient**

imation)

Stochastic approximation methods are a family of iterative methods typically used for root-finding problems or
collected data is corrupted by noise, or for approximating extreme values of functions which cannot be computed

In a nutshell, stochastic approximation algorithms deal with a function of the form $\boxed{f(\theta) = \mathrm{E}_\xi[F(\theta, \xi)]}$ which is
stochastic approximation algorithms use random samples of $F(\theta, \xi)$ to efficiently approximate properties of f su

Recently, stochastic approximations have found extensive applications in the fields of statistics and machine lear
reinforcement learning via temporal differences, and deep learning, and others.[1] Stochastic approximation algo
their theory.[2]

The earliest, and prototypical, algorithms of this kind are the **Robbins–Monro** and **Kiefer–Wolfowitz** algorithms

$$\frac{dz}{dx} = \boxed{\frac{dz}{dy}}\boxed{\frac{dy}{dx}}$$

Screen shot from https://en.wikipedia.org/wiki/Stochastic_approximation

Rethink BP and chain rule

$f(x) \rightarrow \nabla f(x)$

$f(g(x)) \rightarrow ?$

However, we use **stochastic gradient**, rather than **gradient**

Stochastic $\hat{\nabla} f(y) \rightarrow \nabla f(y)$ (approximation)

$E[\hat{\nabla} f(y)] = \nabla f(y)$ (unbiased)

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \boxed{\frac{dz}{dy}}\boxed{\frac{dy}{dx}}$$

Rethink BP and chain rule

$f(x) \rightarrow \nabla f(x)$

$f(g(x)) \rightarrow ?$

However, we use **stochastic gradient**, rather than **gradient**

Stochastic $\hat{\nabla} f(y) \rightarrow \nabla f(y)$ (approximation)

$E[\hat{\nabla} f(y)] = \nabla f(y)$ (unbiased)

$\hat{\nabla} f(y) \neq \nabla f(y)$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\frac{dz}{dx} = \boxed{\frac{dz}{dy}}\boxed{\frac{dy}{dx}}$$

Rethink BP and chain rule

$f(x) \rightarrow \nabla f(x)$

$f(g(x)) \rightarrow ?$

However, we use **stochastic gradient**, rather than **gradient**

Stochastic $\hat{\nabla} f(y) \rightarrow \nabla f(y)$ (approximation)

$E[\hat{\nabla} f(y)] = \nabla f(y)$ (unbiased)

$\hat{\nabla} f(y) \neq \nabla f(y)$

- Chain rule of calculus

$$y = g(x) \text{ and } z = f(g(x)) = f(y)$$

$$\boxed{\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}} \longrightarrow \boxed{\frac{\widehat{dz}}{dx} = \frac{\widehat{dz}}{dy}\frac{\widehat{dy}}{dx}}$$

Composition: Biased

Train an MLP softmax classifier

$$f_n\left(\dots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

- $f_1 \rightarrow ?$
- $f_2 \rightarrow ?$
- $f_3 \rightarrow ?$
- $f_4 \rightarrow ?$
- …

Need to manually implement?

Train an MLP softmax classifier

$$f_n\left(\dots\left(f_2\left(f_1(x)\right)\right)\right) \rightarrow ?$$

- $f_1 \rightarrow ?$
- $f_2 \rightarrow ?$
- $f_3 \rightarrow ?$
- $f_4 \rightarrow ?$
- …

Need to manually implement? ✗

Train an MLP softmax classifier

$$f_n\Big(\dots\Big(f_2\big(f_1(x)\big)\Big)\Big) \rightarrow ?$$

```python
# Fully connected neural network with one hidden layer
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

Train an MLP softmax classifier

Define a function $f: \mathbb{R} \mapsto \mathbb{R}$

Backpropagation:

One iteration:

1. Randomly sample j from $\{1, 2, \cdots, n\}$.
2. **Forward pass**: take x_j as input ($x^{(0)} = x_j$), compute each layer $z^{(1)}, x^{(1)}, z^{(2)}, x^{(2)}, z^{(3)}$.
3. **Backward pass:**
   i. Compute the derivatives $\underbrace{\boxed{\frac{\partial L}{\partial z^{(3)}}}, \boxed{\frac{\partial L}{\partial w^{(2)}}}, \boxed{\frac{\partial L}{\partial z^{(2)}}}, \boxed{\frac{\partial L}{\partial w^{(1)}}}, \boxed{\frac{\partial L}{\partial z^{(1)}}}, \boxed{\frac{\partial L}{\partial w^{(0)}}}}_{\text{Need to compute gradients for each layer?}}$.
   ii. Update $w^{(k)}$ using $\frac{\partial L}{\partial w^{(k)}}$.

Train an MLP softmax classifier

Define a function $f: \mathbb{R} \mapsto \mathbb{R}$

Backpropagation:

One iteration:

1. Randomly sample j from $\{1, 2, \cdots, n\}$.
2. **Forward pass**: take x_j as input ($x^{(0)} = x_j$), compute each layer $z^{(1)}, x^{(1)}, z^{(2)}, x^{(2)}, z^{(3)}$.
3. **Backward pass**:
   i. Compute the derivatives $\underbrace{\boxed{\frac{\partial L}{\partial z^{(3)}}}, \boxed{\frac{\partial L}{\partial w^{(2)}}}, \boxed{\frac{\partial L}{\partial z^{(2)}}}, \boxed{\frac{\partial L}{\partial w^{(1)}}}, \boxed{\frac{\partial L}{\partial z^{(1)}}}, \boxed{\frac{\partial L}{\partial w^{(0)}}}}_{\text{Need to compute gradients for each layer?}}$.
   ii. Update $w^{(k)}$ using $\frac{\partial L}{\partial w^{(k)}}$.

Train an MLP softmax classifier

```
# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model_NN.parameters(), lr=learning_rate, weight_decay=0.00001)
```

```
# Forward pass
outputs = model(images)
loss = criterion(outputs, labels)

# Backward and optimize
optimizer.zero_grad()
loss.backward()
optimizer.step()
```