Foundation of Machine Learning

Amir H. Payberah
payberah@kth.se
2020-09-28

The Course Web Page

https://fid3024.github.io

Linear Regression

Linear Regression (1/2)

- Given the dataset of m houses.

Living area	No. of bedrooms	Price
2104	3	400
1600	3	330
2400	3	369
⋮	⋮	⋮

- Predict the prices of other houses, as a function of the size of living area and number of bedrooms?

Linear Regression (2/2)

- Building a model that takes input $\mathbf{x} \in \mathbb{R}^{\mathbf{n}}$ and predicts output $\hat{\mathbf{y}} \in \mathbb{R}$.

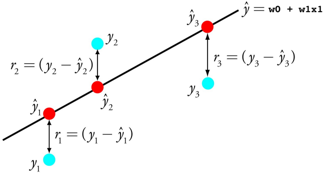
Linear Regression (2/2)

- Building a model that takes input $\mathbf{x} \in \mathbb{R}^n$ and predicts output $\hat{y} \in \mathbb{R}$.
- In linear regression, the output $\hat{y}$ is a linear function of the input $\mathbf{x}$.

$$\hat{y} = f_w(\mathbf{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$
$$\hat{y} = \mathbf{w}^\intercal \mathbf{x}$$

- $\hat{y}$: the predicted value
- n: the number of features
- x_i: the ith feature value
- w_j: the jth model parameter ($\mathbf{w} \in \mathbb{R}^n$)

Loss Function



- For each value of the $\mathbf{w}$, how close the $\hat{y}^{(i)}$ is to the corresponding $y^{(i)}$.
- E.g., Mean Squared Error (MSE)

$$J(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{cost}_{\mathbf{w}}(y^{(i)}, \hat{y}^{(i)}) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2$$

Objective

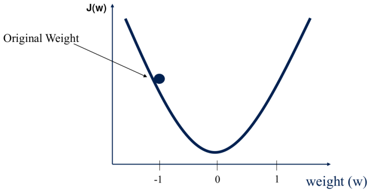
- Minimizing the loss function $J(\mathbf{w})$.
- Gradient descent

Gradient Descent

- Tweaking parameters $\mathbf{w}$ iteratively in order to minimize a loss function $J(\mathbf{w})$.

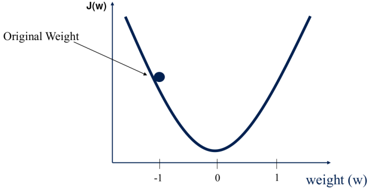
Gradient Descent

- Tweaking parameters **w** iteratively in order to minimize a loss function $J(\mathbf{w})$.
- Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:



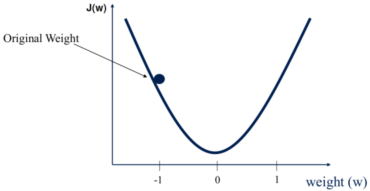
Gradient Descent

- Tweaking parameters $\mathbf{w}$ iteratively in order to minimize a loss function $J(\mathbf{w})$.
- Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
  1. Determine a descent direction $\nabla J(\mathbf{w})$



Gradient Descent

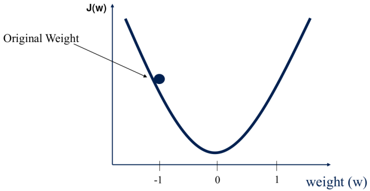
- Tweaking parameters **w** iteratively in order to minimize a loss function $J(\mathbf{w})$.
- Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
  1. Determine a descent direction $\nabla J(\mathbf{w})$
  2. Choose a step size η



Gradient Descent

- Tweaking parameters $\mathbf{w}$ iteratively in order to minimize a loss function $J(\mathbf{w})$.

- Start at a random point, and repeat the following steps, until the stopping criterion is satisfied:
  1. Determine a descent direction $\nabla J(\mathbf{w})$
  2. Choose a step size η
  3. Update the parameters: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J(\mathbf{w})$



Batch Gradient Descent vs. Mini-Batch Stochastic Gradient Descent

- Gradient descent
  - $\mathbf{X}$ is the total dataset.
  - $\mathtt{J}(\mathbf{w}) = \frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \mathbf{X}} \mathtt{cost}_{\mathbf{w}}(\mathtt{y}^{(\mathtt{i})}, \hat{\mathtt{y}}^{(\mathtt{i})})$

Batch Gradient Descent vs. Mini-Batch Stochastic Gradient Descent

- Gradient descent
  - $\mathbf{X}$ is the total dataset.
  - $\mathtt{J}(\mathbf{w}) = \frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \mathbf{X}} \mathtt{cost}_{\mathbf{w}}(\mathtt{y}^{(\mathtt{i})}, \hat{\mathtt{y}}^{(\mathtt{i})}) = \frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \mathbf{X}} \mathtt{l}(\mathbf{x}, \mathbf{w})$

Batch Gradient Descent vs. Mini-Batch Stochastic Gradient Descent

- Gradient descent
  - $\mathbf{X}$ is the total dataset.
  - $\mathtt{J}(\mathbf{w}) = \frac{1}{|\mathbf{X}|}\sum_{\mathbf{x}\in\mathbf{X}} \mathtt{cost}_{\mathbf{w}}(\mathtt{y}^{(\mathtt{i})}, \hat{\mathtt{y}}^{(\mathtt{i})}) = \frac{1}{|\mathbf{X}|}\sum_{\mathbf{x}\in\mathbf{X}} \mathtt{l}(\mathbf{x}, \mathbf{w})$
  - $\mathbf{w} \leftarrow \mathbf{w} - \eta\frac{1}{|\mathbf{X}|}\sum_{\mathbf{x}\in\mathbf{X}} \nabla\mathtt{l}(\mathbf{x}, \mathbf{w})$

Batch Gradient Descent vs. Mini-Batch Stochastic Gradient Descent

- Gradient descent
  - $\mathbf{X}$ is the total dataset.
  - $\mathtt{J}(\mathbf{w}) = \frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \mathbf{X}} \mathtt{cost}_{\mathbf{w}}(\mathtt{y}^{(\mathtt{i})}, \hat{\mathtt{y}}^{(\mathtt{i})}) = \frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \mathbf{X}} \mathtt{l}(\mathbf{x}, \mathbf{w})$
  - $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \mathbf{X}} \nabla \mathtt{l}(\mathbf{x}, \mathbf{w})$

- Mini-batch stochastic gradient descent
  - β is the mini-batch, i.e., a random subset of $\mathbf{X}$.
  - $\mathtt{J}(\mathbf{w}) = \frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \beta} \mathtt{cost}_{\mathbf{w}}(\mathtt{y}^{(\mathtt{i})}, \hat{\mathtt{y}}^{(\mathtt{i})}) = \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \mathtt{l}(\mathbf{x}, \mathbf{w})$

Batch Gradient Descent vs. Mini-Batch Stochastic Gradient Descent

- Gradient descent
  - $\mathbf{X}$ is the total dataset.
  - $\mathtt{J}(\mathbf{w}) = \frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \mathbf{X}} \mathtt{cost}_{\mathbf{w}}(\mathtt{y}^{(\mathtt{i})}, \hat{\mathtt{y}}^{(\mathtt{i})}) = \frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \mathbf{X}} \mathtt{l}(\mathbf{x}, \mathbf{w})$
  - $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \mathbf{X}} \nabla \mathtt{l}(\mathbf{x}, \mathbf{w})$
- Mini-batch stochastic gradient descent
  - β is the mini-batch, i.e., a random subset of $\mathbf{X}$.
  - $\mathtt{J}(\mathbf{w}) = \frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \beta} \mathtt{cost}_{\mathbf{w}}(\mathtt{y}^{(\mathtt{i})}, \hat{\mathtt{y}}^{(\mathtt{i})}) = \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \mathtt{l}(\mathbf{x}, \mathbf{w})$
  - $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \nabla \mathtt{l}(\mathbf{x}, \mathbf{w})$

Binomial Logistic Regression

Binomial Logistic Regression (1/2)

- Given the dataset of m cancer tests.

Tumor size	Cancer
330	1
120	0
400	1
⋮	⋮

- Predict the risk of cancer, as a function of the tumor size?

Binomial Logistic Regression (2/2)

- Linear regression: the model computes the weighted sum of the input features (plus a bias term).

$$\hat{y} = w_0 x_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = \mathbf{w}^\intercal \mathbf{x}$$

Binomial Logistic Regression (2/2)

- Linear regression: the model computes the weighted sum of the input features (plus a bias term).

$$\hat{y} = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = \mathbf{w}^\intercal\mathbf{x}$$

- Binomial logistic regression: the model computes a weighted sum of the input features (plus a bias term), but it outputs the logistic of this result.

$$z = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = \mathbf{w}^\intercal\mathbf{x}$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-\mathbf{w}^\intercal\mathbf{x}}}$$

Loss Function (1/3)

- Naive idea: minimizing the Mean Squared Error (MSE)

$$\mathtt{cost}(\hat{y}^{(i)}, y^{(i)}) = (\hat{y}^{(i)} - y^{(i)})^2$$

$$J(\mathbf{w}) = \frac{1}{m}\sum_{i}^{m} \mathtt{cost}(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

Loss Function (1/3)

- Naive idea: minimizing the Mean Squared Error (MSE)

$$\mathtt{cost}(\hat{y}^{(i)}, y^{(i)}) = (\hat{y}^{(i)} - y^{(i)})^2$$

$$J(\mathbf{w}) = \frac{1}{m}\sum_{i}^{m} \mathtt{cost}(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

$$J(\mathbf{w}) = \mathrm{MSE}(\mathbf{w}) = \frac{1}{m}\sum_{i}^{m}\left(\frac{1}{1 + e^{-\mathbf{w}^\intercal \mathbf{x}^{(i)}}} - y^{(i)}\right)^2$$

Loss Function (1/3)

- Naive idea: minimizing the Mean Squared Error (MSE)

$$\mathtt{cost}(\hat{y}^{(i)}, y^{(i)}) = (\hat{y}^{(i)} - y^{(i)})^2$$

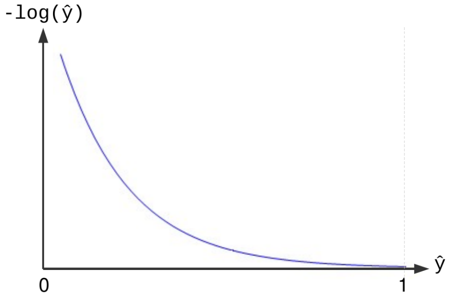
$$J(\mathbf{w}) = \frac{1}{m}\sum_{i}^{m} \mathtt{cost}(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{m}\sum_{i}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

$$J(\mathbf{w}) = \mathtt{MSE}(\mathbf{w}) = \frac{1}{m}\sum_{i}^{m}\left(\frac{1}{1 + e^{-\mathbf{w}^\intercal \mathbf{x}^{(i)}}} - y^{(i)}\right)^2$$

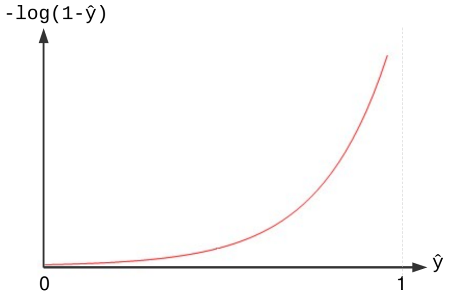
- This cost function is a non-convex function for parameter optimization.

Loss Function (2/3)

$$\texttt{cost}(\hat{y}^{(i)}, y^{(i)}) = \begin{cases} -\log(\hat{y}^{(i)}) & \text{if} \quad y^{(i)} = 1 \\ -\log(1 - \hat{y}^{(i)}) & \text{if} \quad y^{(i)} = 0 \end{cases}$$



when $y = 1$



when $y = 0$

Loss Function (3/3)

- We can define $\mathtt{J}(\mathbf{w})$ as below

$$\mathtt{cost}(\hat{\mathtt{y}}^{(\mathtt{i})}, \mathtt{y}^{(\mathtt{i})}) = \begin{cases} -\mathtt{log}(\hat{\mathtt{y}}^{(\mathtt{i})}) & \text{if} \quad \mathtt{y}^{(\mathtt{i})} = 1 \\ -\mathtt{log}(1 - \hat{\mathtt{y}}^{(\mathtt{i})}) & \text{if} \quad \mathtt{y}^{(\mathtt{i})} = 0 \end{cases}$$

Loss Function (3/3)

- We can define $J(\mathbf{w})$ as below

$$\mathtt{cost}(\hat{y}^{(i)}, y^{(i)}) = \begin{cases} -\mathtt{log}(\hat{y}^{(i)}) & \text{if} \quad y^{(i)} = 1 \\ -\mathtt{log}(1 - \hat{y}^{(i)}) & \text{if} \quad y^{(i)} = 0 \end{cases}$$

$$J(\mathbf{w}) = \frac{1}{m}\sum_{i}^{m}\mathtt{cost}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i}^{m}(y^{(i)}\mathtt{log}(\hat{y}^{(i)}) + (1 - y^{(i)})\mathtt{log}(1 - \hat{y}^{(i)}))$$

Binomial vs. Multinomial Logistic Regression (1/2)

- In a binomial classifier, $y \in \{0, 1\}$, the estimator is $\hat{y} = p(y = 1 \mid \mathbf{x}; \mathbf{w})$.
  - We find one set of parameters $\mathbf{w}$.

$$\mathbf{w}^\intercal = [w_0, w_1, \cdots, w_n]$$

Binomial vs. Multinomial Logistic Regression (1/2)

- In a binomial classifier, $y \in \{0, 1\}$, the estimator is $\hat{y} = p(y = 1 \mid \mathbf{x}; \mathbf{w})$.
  - We find one set of parameters $\mathbf{w}$.

$$\mathbf{w}^\mathsf{T} = [w_0, w_1, \cdots, w_n]$$

- In multinomial classifier, $y \in \{1, 2, \cdots, k\}$, we need to estimate the result for each individual label, i.e., $\hat{y}_j = p(y = j \mid \mathbf{x}; \mathbf{w})$.

Binomial vs. Multinomial Logistic Regression (1/2)

- In a binomial classifier, $y \in \{0, 1\}$, the estimator is $\hat{y} = p(y = 1 \mid \mathbf{x}; \mathbf{w})$.
  - We find one set of parameters $\mathbf{w}$.

$$\mathbf{w}^\intercal = [w_0, w_1, \cdots, w_n]$$

- In multinomial classifier, $y \in \{1, 2, \cdots, k\}$, we need to estimate the result for each individual label, i.e., $\hat{y}_j = p(y = j \mid \mathbf{x}; \mathbf{w})$.
  - We find k set of parameters $\mathbf{W}$.

$$\mathbf{W} = \begin{bmatrix} [w_{0,1}, w_{1,1}, \cdots, w_{n,1}] \\ [w_{0,2}, w_{1,2}, \cdots, w_{n,2}] \\ \vdots \\ [w_{0,k}, w_{1,k}, \cdots, w_{n,k}] \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\intercal \\ \mathbf{w}_2^\intercal \\ \vdots \\ \mathbf{w}_k^\intercal \end{bmatrix}$$

Binomial vs. Multinomial Logistic Regression (2/2)

- In a binary class, $y \in \{0, 1\}$, we use the sigmoid function.

$$\mathbf{w}^\intercal \mathbf{x} = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n$$

$$\hat{y} = p(y = 1 \mid \mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^\intercal \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\intercal \mathbf{x}}}$$

Binomial vs. Multinomial Logistic Regression (2/2)

- In a binary class, $y \in \{0, 1\}$, we use the sigmoid function.

$$\mathbf{w}^\intercal \mathbf{x} = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n$$

$$\hat{y} = p(y = 1 \mid \mathbf{x}; \mathbf{w}) = \sigma(\mathbf{w}^\intercal \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\intercal \mathbf{x}}}$$

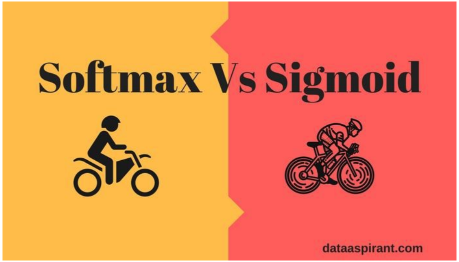
- In multiclasses, $y \in \{1, 2, \cdots, k\}$, we use the softmax function.

$$\mathbf{w}_j^\intercal \mathbf{x} = w_{0,j} x_0 + w_{1,j} x_1 + \cdots + w_{n,j} x_n, 1 \leq j \leq k$$

$$\hat{y}_j = p(y = j \mid \mathbf{x}; \mathbf{w}_j) = \sigma(\mathbf{w}_j^\intercal \mathbf{x}) = \frac{e^{\mathbf{w}_j^\intercal \mathbf{x}}}{\sum_{i=1}^{k} e^{\mathbf{w}_i^\intercal \mathbf{x}}}$$

Sigmoid vs. Softmax

- Sigmoid function: $\sigma(\mathbf{w}^\mathsf{T}\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^\mathsf{T}\mathbf{x}}}$
- Softmax function: $\sigma(\mathbf{w}_j^\mathsf{T}\mathbf{x}) = \frac{e^{\mathbf{w}_j^\mathsf{T}\mathbf{x}}}{\sum_{i=1}^{k} e^{\mathbf{w}_i^\mathsf{T}\mathbf{x}}}$
  - Calculate the probabilities of each target class over all possible target classes.
  - The softmax function for two classes is equivalent the sigmoid function.

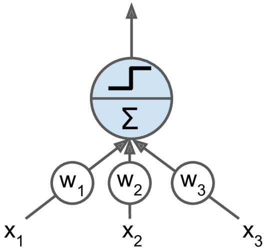


Deep Neural Network

The Linear Threshold Unit (LTU)

- Each input connection is associated with a weight.
- Computes a weighted sum of its inputs and applies a step function to that sum.

- $z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = \mathbf{w}^\intercal \mathbf{x}$
- $\hat{y} = \text{step}(z) = \text{step}(\mathbf{w}^\intercal \mathbf{x})$



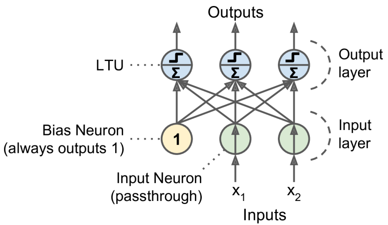
The Perceptron

- The perceptron is a single layer of LTUs.
- Train the model.

The Perceptron

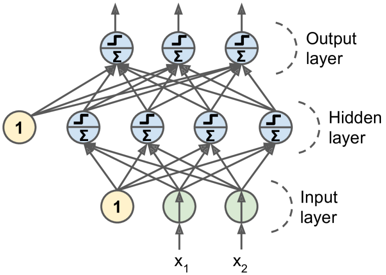
- The perceptron is a single layer of LTUs.
- Train the model.

$$\hat{\mathbf{y}} = \mathtt{f}_{\mathbf{w}}(\mathbf{X})$$
$$\mathtt{J}(\mathbf{w}) = \mathtt{cost}(\mathbf{y}, \hat{\mathbf{y}})$$
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \mathtt{J}(\mathbf{w})$$



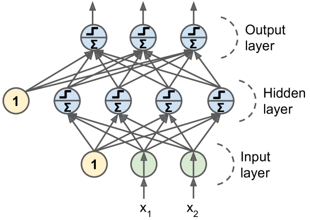
Feedforward Neural Network Architecture

- A feedforward neural network is composed of:
  - One input layer
  - One or more hidden layers
  - One final output layer



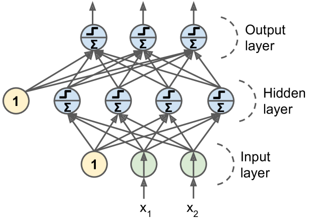
Training Feedforward Neural Networks

- How to train a feedforward neural network?



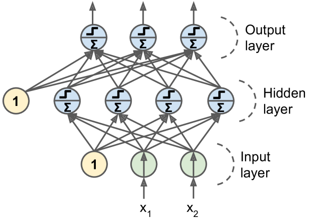
Training Feedforward Neural Networks

- How to train a feedforward neural network?
- For each training instance $\mathbf{x}^{(i)}$ the algorithm does the following steps:



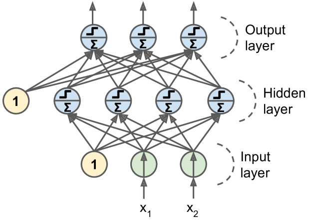
Training Feedforward Neural Networks

- How to train a feedforward neural network?
- For each training instance $\mathbf{x}^{(i)}$ the algorithm does the following steps:
  1. Forward pass: make a prediction (i.e., $\hat{y}^{(i)}$).



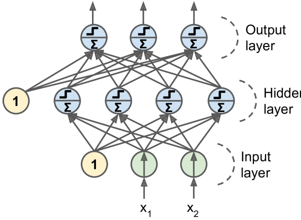
Training Feedforward Neural Networks

- How to train a feedforward neural network?
- For each training instance $\mathbf{x}^{(i)}$ the algorithm does the following steps:
  1. Forward pass: make a prediction (i.e., $\hat{y}^{(i)}$).
  2. Measure the error (i.e., $\mathtt{cost}(\hat{y}^{(i)}, y^{(i)})$).



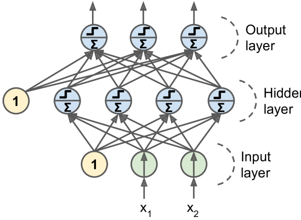
Training Feedforward Neural Networks

- How to train a feedforward neural network?
- For each training instance $\mathbf{x}^{(i)}$ the algorithm does the following steps:
  1. Forward pass: make a prediction (i.e., $\hat{y}^{(i)}$).
  2. Measure the error (i.e., $\mathtt{cost}(\hat{y}^{(i)}, y^{(i)})$).
  3. Backward pass: go through each layer in reverse to measure the error contribution from each connection.



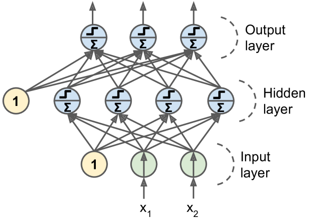
Training Feedforward Neural Networks

- How to train a feedforward neural network?
- For each training instance $\mathbf{x}^{(i)}$ the algorithm does the following steps:
  1. Forward pass: make a prediction (i.e., $\hat{y}^{(i)}$).
  2. Measure the error (i.e., $\texttt{cost}(\hat{y}^{(i)}, y^{(i)})$).
  3. Backward pass: go through each layer in reverse to measure the error contribution from each connection.
  4. Tweak the connection weights to reduce the error (update $\mathbf{W}$ and $\mathbf{b}$).



Training Feedforward Neural Networks

- How to train a feedforward neural network?
- For each training instance $\mathbf{x}^{(\mathtt{i})}$ the algorithm does the following steps:
  1. Forward pass: make a prediction (i.e., $\hat{\mathtt{y}}^{(\mathtt{i})}$).
  2. Measure the error (i.e., $\mathtt{cost}(\hat{\mathtt{y}}^{(\mathtt{i})}, \mathtt{y}^{(\mathtt{i})})$).
  3. Backward pass: go through each layer in reverse to measure the error contribution from each connection.
  4. Tweak the connection weights to reduce the error (update $\mathbf{W}$ and $\mathbf{b}$).
- It's called the backpropagation training algorithm



Generalization

Generalization

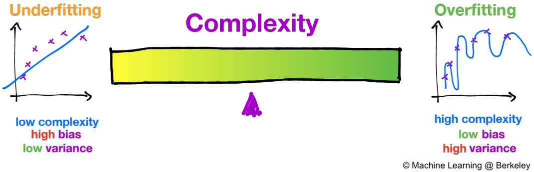
- Generalization: make a model that performs well on test data.
  - Have a small test error.

Generalization

- Generalization: make a model that performs well on test data.
  - Have a small test error.
- Challenges
  1. Make the training error small.
  2. Make the gap between training and test error small.

Generalization

- Generalization: make a model that performs well on test data.
  - Have a small test error.
- Challenges
  1. Make the training error small.
  2. Make the gap between training and test error small.
- Overfitting vs. underfitting



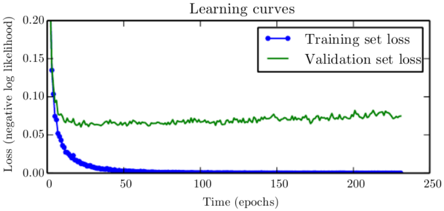
[https://ml.berkeley.edu/blog/2017/07/13/tutorial-4]

Avoiding Overfitting

- Early stopping
- $l1$ and $l2$ regularization
- Max-norm regularization
- Dropout
- Data augmentation

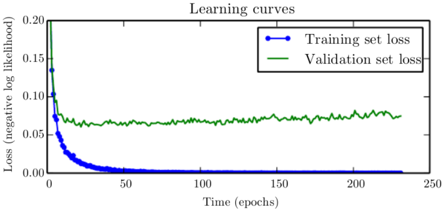
Early Stopping

- As the training steps go by, its prediction error on the training/validation set naturally goes down.



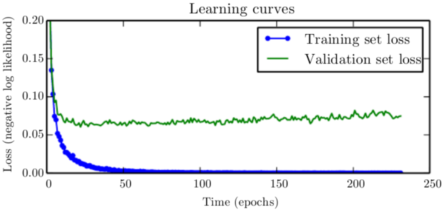
Early Stopping

- As the training steps go by, its prediction error on the training/validation set naturally goes down.
- After a while the validation error stops decreasing and starts to go back up.
  - The model has started to overfit the training data.



Early Stopping

- As the training steps go by, its prediction error on the training/validation set naturally goes down.
- After a while the validation error stops decreasing and starts to go back up.
  - The model has started to overfit the training data.
- In the early stopping, we stop training when the validation error reaches a minimum.



$l1$ and $l2$ Regularization

- Penalize large values of weights w_j.

$$\tilde{J}(\mathbf{w}) = J(\mathbf{w}) + \lambda R(\mathbf{w})$$

$l1$ and $l2$ Regularization

- Penalize large values of weights $\mathtt{w_j}$.

$$\tilde{J}(\mathbf{w}) = J(\mathbf{w}) + \lambda R(\mathbf{w})$$

- $l1$ regression: $R(\mathbf{w}) = \lambda \sum_{i=1}^{n} |w_i|$ is added to the cost function.

$$\tilde{J}(\mathbf{w}) = J(\mathbf{w}) + \lambda \sum_{i=1}^{n} |w_i|$$

$l1$ and $l2$ Regularization

- Penalize large values of weights w_j.

$$\tilde{J}(\mathbf{w}) = J(\mathbf{w}) + \lambda R(\mathbf{w})$$

- $l1$ regression: $R(\mathbf{w}) = \lambda \sum_{i=1}^{n} |w_i|$ is added to the cost function.

$$\tilde{J}(\mathbf{w}) = J(\mathbf{w}) + \lambda \sum_{i=1}^{n} |w_i|$$

- $l2$ regression: $R(\mathbf{w}) = \lambda \sum_{i=1}^{n} w_i^2$ is added to the cost function.

$$\tilde{J}(\mathbf{w}) = J(\mathbf{w}) + \lambda \sum_{i=1}^{n} w_i^2$$

Max-Norm Regularization

- Max-norm regularization: constrains the weights $\mathbf{w}_j$ of the incoming connections for each neuron j.
  - Prevents them from getting too large.

Max-Norm Regularization

- Max-norm regularization: constrains the weights $\mathbf{w}_j$ of the incoming connections for each neuron j.
  - Prevents them from getting too large.

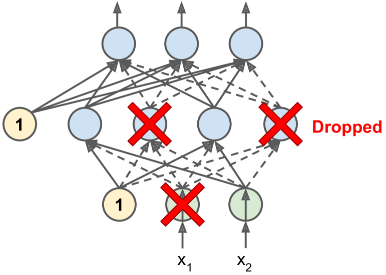
- After each training step, clip $\mathbf{w}_j$ as below, if $||\mathbf{w}_j||_2 > r$:

$$\mathbf{w}_j \leftarrow \mathbf{w}_j \frac{r}{||\mathbf{w}_j||_2}$$

  - r is the max-norm hyperparameter
  - $||\mathbf{w}_j||_2 = (\sum_i w_{i,j}^2)^{\frac{1}{2}} = \sqrt{w_{1,j}^2 + w_{2,j}^2 + \cdots + w_{n,j}^2}$

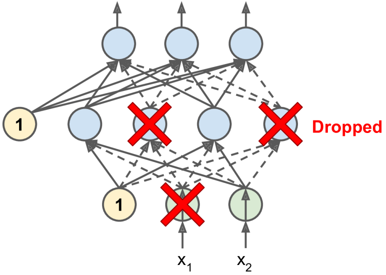
Dropout (1/2)

- At each training step, each neuron drops out temporarily with a probability p.



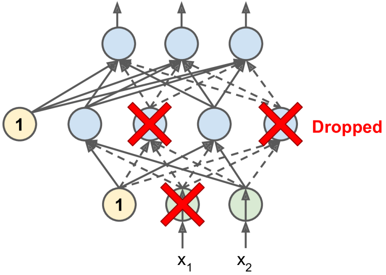
Dropout (1/2)

- At each training step, each neuron drops out temporarily with a probability p.
  - The hyperparameter p is called the dropout rate.



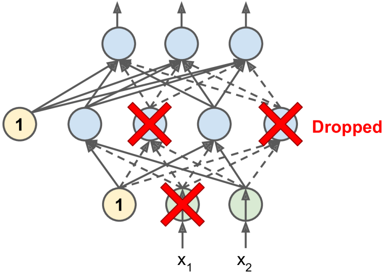
Dropout (1/2)

- At each training step, each neuron drops out temporarily with a probability p.
  - The hyperparameter p is called the dropout rate.
  - A neuron will be entirely ignored during this training step.



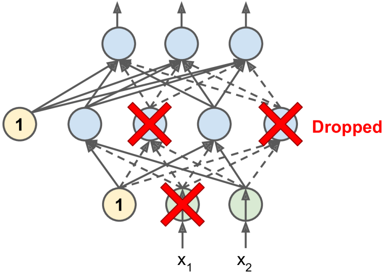
Dropout (1/2)

- At each training step, each neuron drops out temporarily with a probability p.
  - The hyperparameter p is called the dropout rate.
  - A neuron will be entirely ignored during this training step.
  - It may be active during the next step.



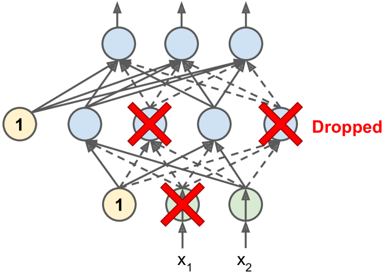
Dropout (1/2)

- At each training step, each neuron drops out temporarily with a probability p.
  - The hyperparameter p is called the dropout rate.
  - A neuron will be entirely ignored during this training step.
  - It may be active during the next step.
  - Exclude the output neurons.



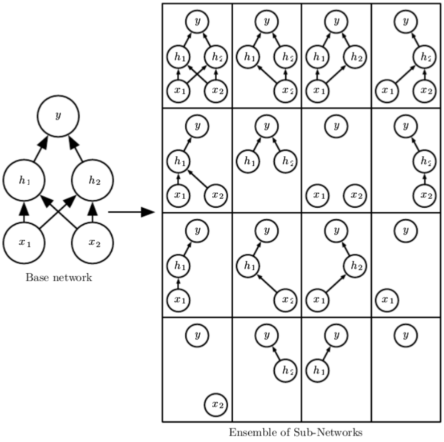
Dropout (1/2)

- At each training step, each neuron drops out temporarily with a probability p.
  - The hyperparameter p is called the dropout rate.
  - A neuron will be entirely ignored during this training step.
  - It may be active during the next step.
  - Exclude the output neurons.
- After training, neurons don't get dropped anymore.



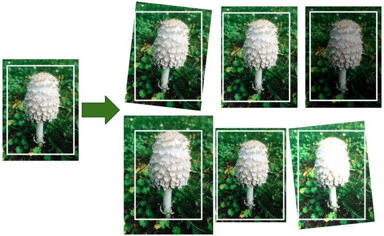
Dropout (2/2)

- Each neuron can be either present or absent.
- 2^N possible networks, where N is the total number of droppable neurons.
  - $N = 4$ in this figure.



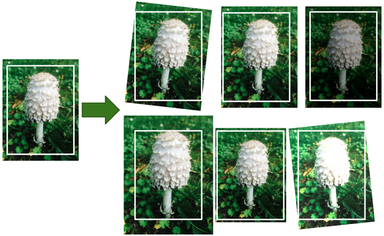
Data Augmentation

- One way to make a model generalize better is to train it on more data.
- This will reduce overfitting.



Data Augmentation

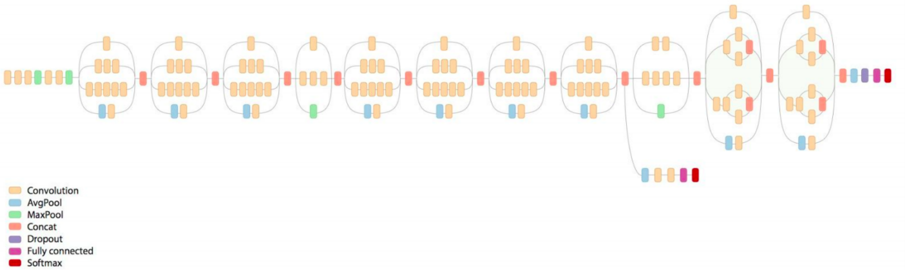
- One way to make a model generalize better is to train it on more data.
- This will reduce overfitting.
- Create fake data and add it to the training set.



Batch Size

Training Deep Neural Networks

- Computationally intensive
- Time consuming

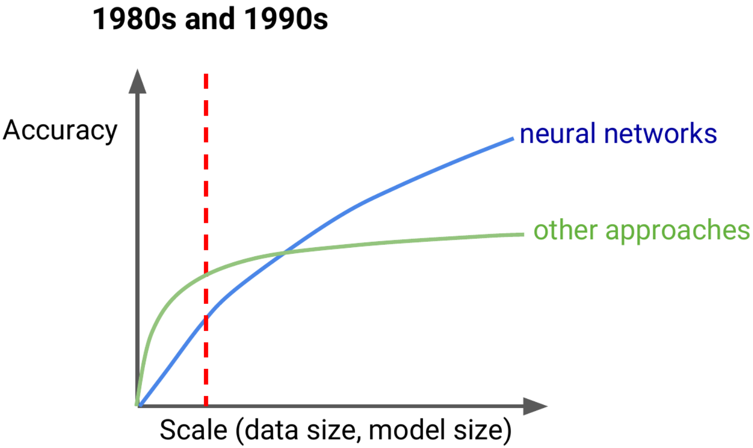


[https://cloud.google.com/tpu/docs/images/inceptionv3onc--oview.png]

Why?

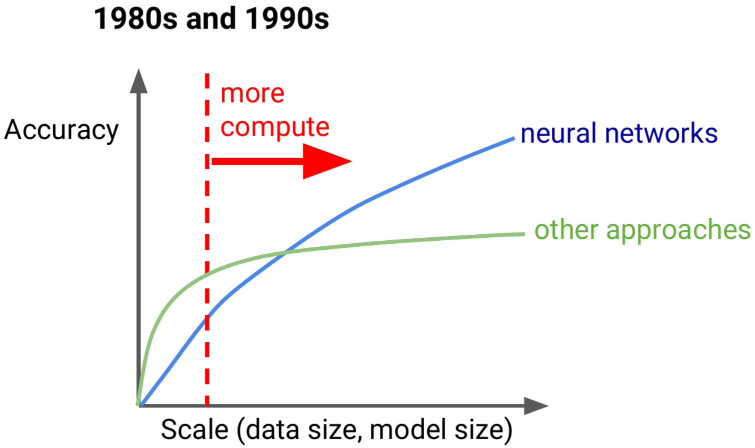
- Massive amount of training dataset
- Large number of parameters

Accuracy vs. Data/Model Size



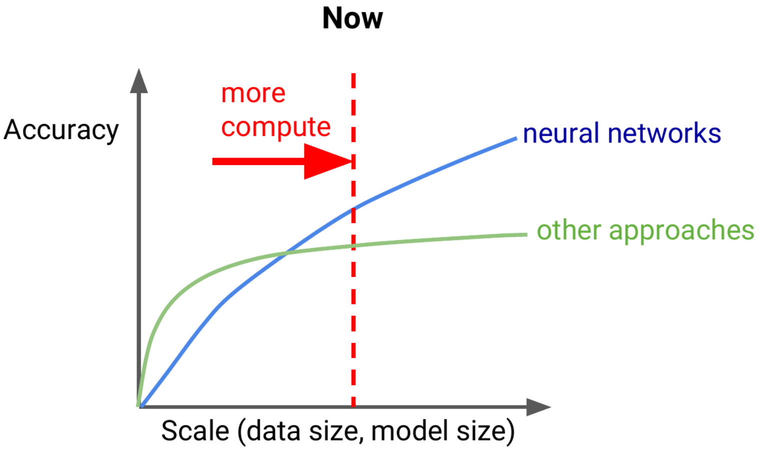
[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

Accuracy vs. Data/Model Size



[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

Accuracy vs. Data/Model Size

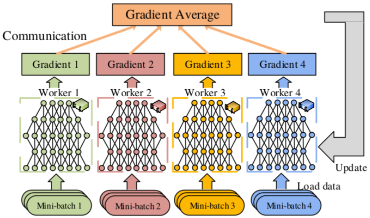


[Jeff Dean at AI Frontiers: Trends and Developments in Deep Learning Research]

Scale Matters

Distributed Gradient Descent (1/2)

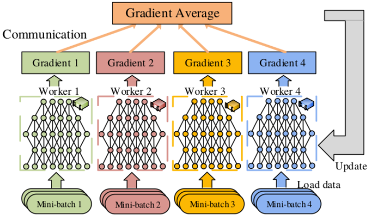
- Replicate a whole model on every device.
- Each device has model replica with a copy of model parameters.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Gradient Descent (2/2)

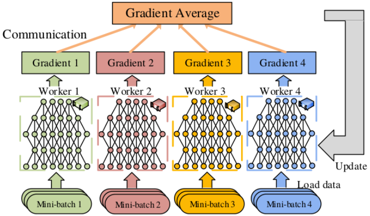
- Parameter Server (PS): maintains global model.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Gradient Descent (2/2)

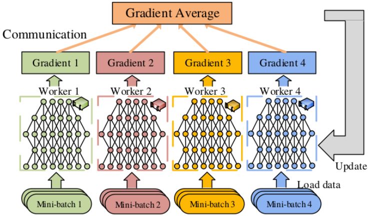
- Parameter Server (PS): maintains global model.
- Once each device completes processing, the weights are transferred to PS, which aggregates all the gradients.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Distributed Gradient Descent (2/2)

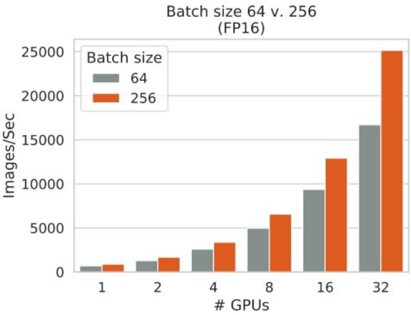
- Parameter Server (PS): maintains global model.
- Once each device completes processing, the weights are transferred to PS, which aggregates all the gradients.
- The PS, then, sends back the results to each device.



[Tang et al., Communication-Efficient Distributed Deep Learning: A Comprehensive Survey, 2020]

Batch Size vs. Number of GPUs

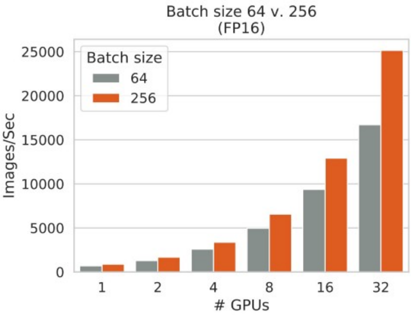
- $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \nabla \mathtt{l}(\mathbf{x}, \mathbf{w})$



[https://medium.com/@emwatz/lessons-for-improving-training-performance-part-1-b5efd0f0dcea]

Batch Size vs. Number of GPUs

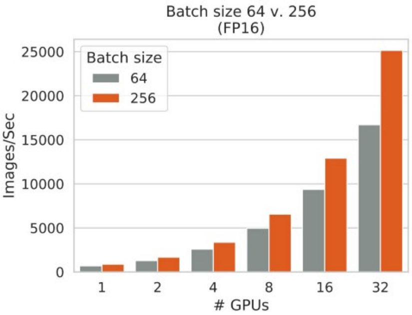
- $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \nabla \mathtt{l}(\mathbf{x}, \mathbf{w})$
- The more samples processed during each batch, the faster a training job will complete.



[https://medium.com/@emwatz/lessons-for-improving-training-performance-part-1-b5efd0f0dcea]

Batch Size vs. Number of GPUs

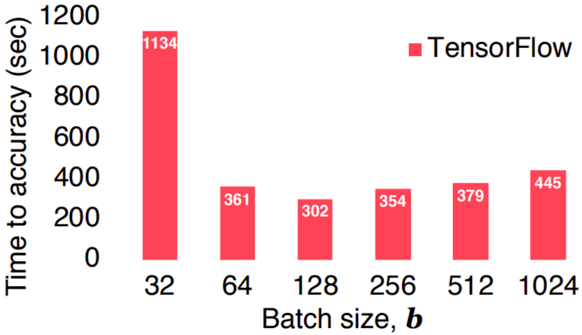
- $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \nabla \mathtt{l}(\mathbf{x}, \mathbf{w})$
- The more samples processed during each batch, the faster a training job will complete.
- E.g., ImageNet + ResNet-50



[https://medium.com/@emwatz/lessons-for-improving-training-performance-part-1-b5efd0f0dcea]

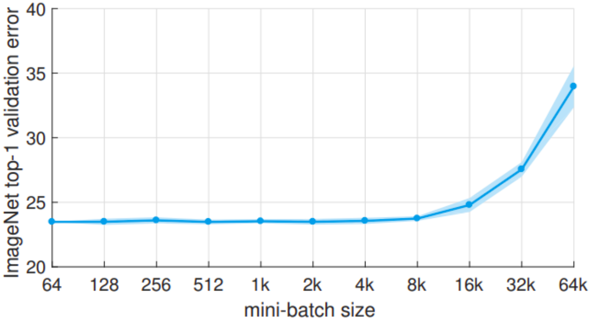
Batch Size vs. Time to Accuracy

- ResNet-32 on Titan X GPU



[Peter Pietzuch - Imperial College London]

Batch Size vs. Validation Error



[Goyal et al., Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, 2018]

Improve the Validation Error

Improve the Validation Error

- Scaling learning rate
- Batch normalization
- Label smoothing
- Momentum

Scaling Learning Rate

- $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \nabla \mathtt{l}(\mathbf{x}, \mathbf{w})$.

Scaling Learning Rate

- $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \nabla \mathtt{l}(\mathbf{x}, \mathbf{w})$.
- Linear scaling: multiply the learning rate by `k`, when the mini batch size is multiplied by `k`.

Scaling Learning Rate

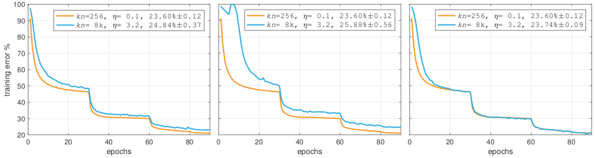
- $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \nabla \mathtt{l}(\mathbf{x}, \mathbf{w})$.
- Linear scaling: multiply the learning rate by $\mathtt{k}$, when the mini batch size is multiplied by $\mathtt{k}$.
- Constant warmup: start with a small learning rate for few epochs, and then increase the learning rate to $\mathtt{k}$ times learning rate.

Scaling Learning Rate

- $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \nabla \mathtt{l}(\mathbf{x}, \mathbf{w})$.
- Linear scaling: multiply the learning rate by k, when the mini batch size is multiplied by k.
- Constant warmup: start with a small learning rate for few epochs, and then increase the learning rate to k times learning rate.
- Gradual warmup: start with a small learning rate, and then gradually increase it by a constant for each epoch till it reaches k times learning rate.

Scaling Learning Rate

- $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{|\beta|} \sum_{\mathbf{x} \in \beta} \nabla \mathtt{l}(\mathbf{x}, \mathbf{w})$.
- Linear scaling: multiply the learning rate by k, when the mini batch size is multiplied by k.
- Constant warmup: start with a small learning rate for few epochs, and then increase the learning rate to k times learning rate.
- Gradual warmup: start with a small learning rate, and then gradually increase it by a constant for each epoch till it reaches k times learning rate.



(a) no warmup (b) constant warmup (c) gradual warmup

[Goyal et al., Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, 2018]

Batch Normalization (1/2)

- Changes in minibatch size change the underlying loss function being optimized.

Batch Normalization (1/2)

- Changes in minibatch size change the underlying loss function being optimized.
- Batch Normalization computes statistics along the minibatch dimension.

Batch Normalization (1/2)

- Changes in minibatch size change the underlying loss function being optimized.
- Batch Normalization computes statistics along the minibatch dimension.

$$\mu_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \mathbf{x}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} (\mathbf{x} - \mu_{\mathcal{B}})^2$$

Batch Normalization (2/2)

- Zero-centering and normalizing the inputs, then scaling and shifting the result.

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}}$$

$$\mathbf{z} = \alpha\hat{\mathbf{x}} + \gamma$$

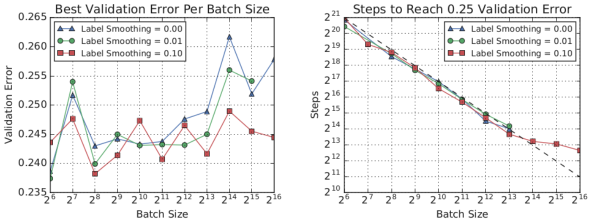
- $\hat{\mathbf{x}}$: the zero-centered and normalized input.
- $\mathbf{z}$: the output of the BN operation, which is a scaled and shifted version of the inputs.
- α: the scaling parameter vector for the layer.
- γ: the shifting parameter (offset) vector for the layer.
- ϵ: a tiny number to avoid division by zero.

Label Smoothing

- A generalization technique.
- Replaces one-hot encoded label vector $\mathbf{y}_{\texttt{hot}}$ with a mixture of $\mathbf{y}_{\texttt{hot}}$ and the uniform distribution.

$$\mathbf{y}_{\texttt{ls}} = (1 - \alpha)\mathbf{y}_{\texttt{hot}} + \alpha / \texttt{K}$$

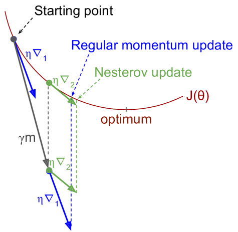
- $\texttt{K}$ is the number of label classes, and α is a hyperparameter.



[Shallue et al., Measuring the Effects of Data Parallelism on Neural Network Training, 2019]

Momentum (1/3)

- Regular gradient descent optimization: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J(\mathbf{w})$

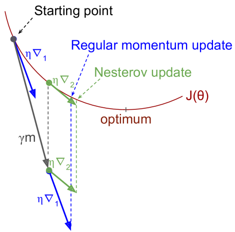


[Aurélien Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2019]

Momentum (1/3)

- Regular gradient descent optimization: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla \mathrm{J}(\mathbf{w})$
- At each iteration, momentum optimization adds the local gradient to the momentum vector $\mathbf{m}$.

$$\mathbf{m} \leftarrow \beta \mathbf{m} + \eta \nabla \mathrm{J}(\mathbf{w})$$
$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{m}$$

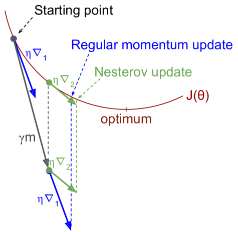


[Aurélien Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2019]

Momentum (2/3)

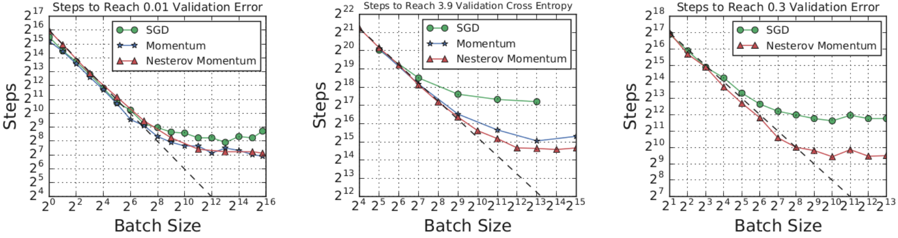
- Nesterov momentum measure the gradient of the cost function slightly ahead in the direction of the momentum.

$$\mathbf{m} = \beta\mathbf{m} + \eta\nabla J(\mathbf{w} + \beta\mathbf{m})$$
$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{m}$$



[Aurélien Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2019]

Momentum (3/3)



(a) Simple CNN on MNIST (b) Transformer Shallow on LM1B (c) ResNet-8 on CIFAR-10

[Shallue et al., Measuring the Effects of Data Parallelism on Neural Network Training, 2019]

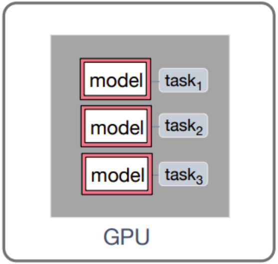
CROSSBOW: Scaling Deep Learning with Small Batch Sizes on Multi-GPU Servers

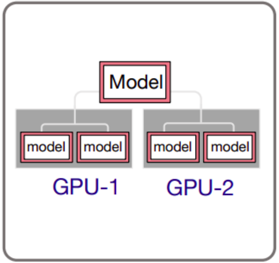
- How to design a deep learning system that scales training with multiple GPUs, even when the preferred batch size is small?

Crossbow

(1) How to increase efficiency with small batches?

(2) How to synchronise model replicas?





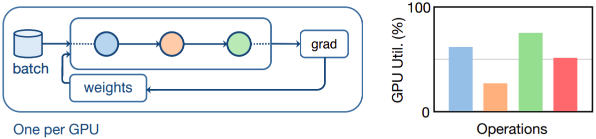
[Peter Pietzuch - Imperial College London]

Problem: Small Batches

- Small batch sizes underutilise GPUs.

Problem: Small Batches

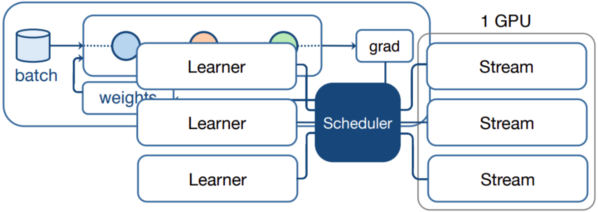
- Small batch sizes underutilise GPUs.
- One batch per GPU: not enough data and instruction parallelism for every operator.



[Peter Pietzuch - Imperial College London]

Idea: Multiple Replicas Per GPU

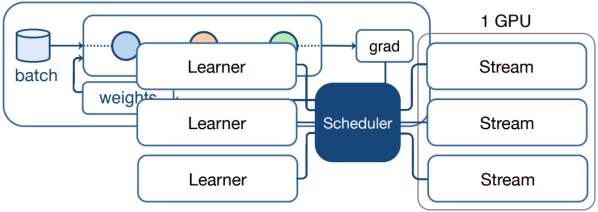
- Train multiple model replicas per GPU.
- A learner is an entity that trains a single model replica independently with a given batch size.



[Peter Pietzuch - Imperial College London]

Idea: Multiple Replicas Per GPU

- Train multiple model replicas per GPU.
- A learner is an entity that trains a single model replica independently with a given batch size.

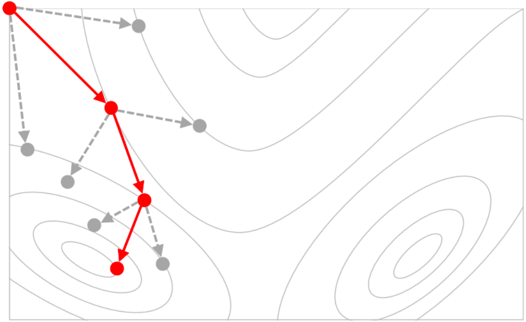


[Peter Pietzuch - Imperial College London]

- But, now we must synchronise a large number of model replicas.

Problem: Similiar Starting Point

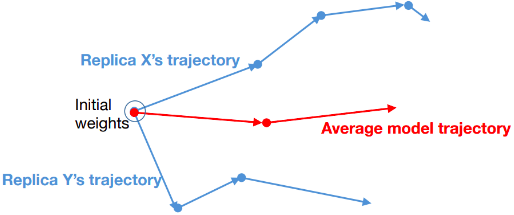
- All learners always start from the same point.
- Limited exploration of parameter space.



[Peter Pietzuch - Imperial College London]

Idea: Independent Replicas

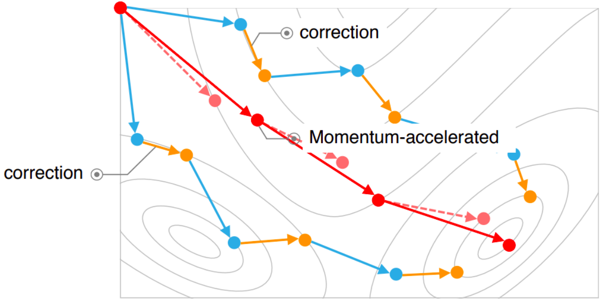
- Maintain independent model replicas.
- Increased exploration of space through parallelism.
- Each model replica uses small batch size.



[Peter Pietzuch - Imperial College London]

Crossbow: Synchronous Model Averaging

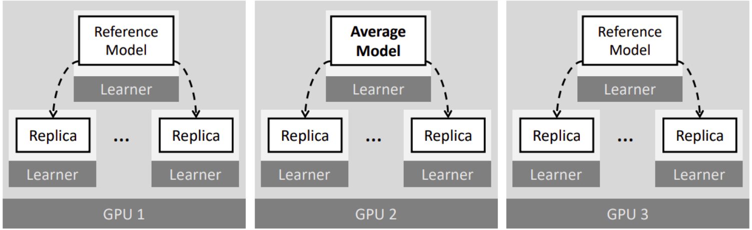
- Allow learners to diverge, but correct trajectories based on average model.
- Accelerate average model trajectory with momentum to find minima faster.



[Peter Pietzuch - Imperial College London]

GPUs with Synchronous Model Averaging

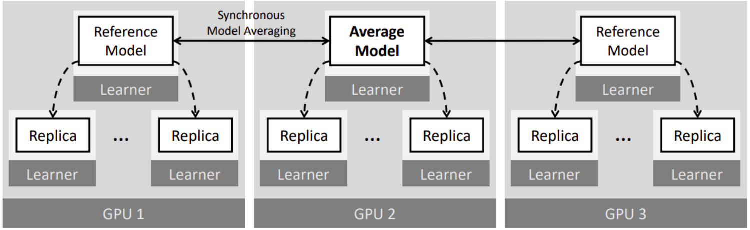
- Synchronously apply corrections to model replicas.



[Peter Pietzuch - Imperial College London]

GPUs with Synchronous Model Averaging

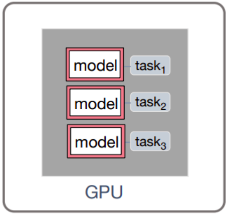
- Ensures consistent view of average model.
- Takes GPU bandwidth into account during synchronisation.



[Peter Pietzuch - Imperial College London]

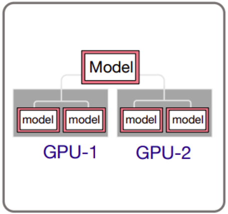
Crossbow

(1) How to increase efficiency with small batches?



Train multiple model replicas per GPU

(2) How to synchronise model replicas?



Use synchronous model averaging

[Peter Pietzuch - Imperial College London]

Summary

Summary

- Stochastic Gradient Descent (SGD)
- Generalization
  - Regularization
  - Max-norm
  - Dropout
- Distributed SGD
- Batch size
  - Scaling learing rate
  - Batch normalization
  - Label smoothing
  - Momntum
- Crossbow

Reference

- P. Goyal et al., Accurate, large minibatch sgd: Training imagenet in 1 hour, 2017
- C. Shallue et al., Measuring the effects of data parallelism on neural network training, 2018
- A. Koliousis et al. CROSSBOW: scaling deep learning with small batch sizes on multi-gpu servers, 2019

Questions?