Deep Learning Book

Chapter 7
Regularization for Deep Learning

Botian Shi
botianshi@bit.edu.cn

March 14, 2017

You can download the LaTeX source code of this file from Here.

Generalization and Strategy

- How to make an algorithm that will perform well not just on the training data, but also on new inputs? (Generalization)
- Many strategies designed to reduce the test error, possibly at the expense of increased training error.
- These strategies are known collectively as **regularization**.
- Many regularization algorithm have been developed.
- Developing more effective regularization strategies is one of the major research efforts in the field.
- In this chapter, we describe regularization in more detail, focusing on regularization strategies for deep models or models that may be used as building blocks to form deep models.

Generalization and Strategy

- How to make an algorithm that will perform well not just on the training data, but also on new inputs? (Generalization)
- Many strategies designed to reduce the test error, possibly at the expense of increased training error.
- These strategies are known collectively as **regularization**.
- Many regularization algorithm have been developed.
- Developing more effective regularization strategies is one of the major research efforts in the field.
- In this chapter, we describe regularization in more detail, focusing on regularization strategies for deep models or models that may be used as building blocks to form deep models.

Generalization and Strategy

- How to make an algorithm that will perform well not just on the training data, but also on new inputs? (Generalization)
- Many strategies designed to reduce the test error, possibly at the expense of increased training error.
- These strategies are known collectively as **regularization**.
- Many regularization algorithm have been developed.
- Developing more effective regularization strategies is one of the major research efforts in the field.
- In this chapter, we describe regularization in more detail, focusing on regularization strategies for deep models or models that may be used as building blocks to form deep models.

Generalization and Strategy

- How to make an algorithm that will perform well not just on the training data, but also on new inputs? (Generalization)
- Many strategies designed to reduce the test error, possibly at the expense of increased training error.
- These strategies are known collectively as **regularization**.
- Many regularization algorithm have been developed.
- Developing more effective regularization strategies is one of the major research efforts in the field.
- In this chapter, we describe regularization in more detail, focusing on regularization strategies for deep models or models that may be used as building blocks to form deep models.

Generalization and Strategy

- How to make an algorithm that will perform well not just on the training data, but also on new inputs? (Generalization)
- Many strategies designed to reduce the test error, possibly at the expense of increased training error.
- These strategies are known collectively as **regularization**.
- Many regularization algorithm have been developed.
- Developing more effective regularization strategies is one of the major research efforts in the field.
- In this chapter, we describe regularization in more detail, focusing on regularization strategies for deep models or models that may be used as building blocks to form deep models.

Generalization and Strategy

- How to make an algorithm that will perform well not just on the training data, but also on new inputs? (Generalization)
- Many strategies designed to reduce the test error, possibly at the expense of increased training error.
- These strategies are known collectively as **regularization**.
- Many regularization algorithm have been developed.
- Developing more effective regularization strategies is one of the major research efforts in the field.
- In this chapter, we describe regularization in more detail, focusing on regularization strategies for deep models or models that may be used as building blocks to form deep models.

Generalization and Strategy

- There are many regularization strategies.
  1. Put extra constrains on a machine learning model. (Adding restrictions on the parameter values.)
  2. Add extra terms in the objective function that can be thought of as corresponding to a soft constraint on the parameter values.
- If chosen carefully, these extra constraints and penalties can lead to improved performance on the test set.
- Sometimes these constraints and penalties are designed to
  1. **encode** specific kinds of **prior knowledge**.
  2. express a generic preference for a simpler model class in order to promote generalization.
  3. make an under-determined problem determined. (Provide more information)
- Other forms of regularization, known as ensemble methods, combine multiple hypotheses that explain the training data.

Generalization and Strategy

- There are many regularization strategies.
    1. Put extra constrains on a machine learning model. (Adding restrictions on the parameter values.)
    2. Add extra terms in the objective function that can be thought of as corresponding to a soft constraint on the parameter values.
- If chosen carefully, these extra constraints and penalties can lead to improved performance on the test set.
- Sometimes these constraints and penalties are designed to
    1. **encode** specific kinds of **prior knowledge**.
    2. express a generic preference for a simpler model class in order to promote generalization.
    3. make an under-determined problem determined. (Provide more information)
- Other forms of regularization, known as ensemble methods, combine multiple hypotheses that explain the training data.

Generalization and Strategy

- There are many regularization strategies.
    1. Put extra constrains on a machine learning model. (Adding restrictions on the parameter values.)
    2. Add extra terms in the objective function that can be thought of as corresponding to a soft constraint on the parameter values.
- If chosen carefully, these extra constraints and penalties can lead to improved performance on the test set.
- Sometimes these constraints and penalties are designed to
    1. **encode** specific kinds of **prior knowledge**.
    2. express a generic preference for a simpler model class in order to promote generalization.
    3. make an under-determined problem determined. (Provide more information)
- Other forms of regularization, known as ensemble methods, combine multiple hypotheses that explain the training data.

Generalization and Strategy

- There are many regularization strategies.
  1. Put extra constrains on a machine learning model. (Adding restrictions on the parameter values.)
  2. Add extra terms in the objective function that can be thought of as corresponding to a soft constraint on the parameter values.
- If chosen carefully, these extra constraints and penalties can lead to improved performance on the test set.
- Sometimes these constraints and penalties are designed to
  1. **encode** specific kinds of **prior knowledge**.
  2. express a generic preference for a simpler model class in order to promote generalization.
  3. make an under-determined problem determined. (Provide more information)
- Other forms of regularization, known as ensemble methods, combine multiple hypotheses that explain the training data.

Generalization and Strategy

- There are many regularization strategies.
    1. Put extra constrains on a machine learning model. (Adding restrictions on the parameter values.)
    2. Add extra terms in the objective function that can be thought of as corresponding to a soft constraint on the parameter values.
- If chosen carefully, these extra constraints and penalties can lead to improved performance on the test set.
- Sometimes these constraints and penalties are designed to
    1. **encode** specific kinds of **prior knowledge**.
    2. express a generic preference for a simpler model class in order to promote generalization.
    3. make an under-determined problem determined. (Provide more information)
- Other forms of regularization, known as ensemble methods, combine multiple hypotheses that explain the training data.

Generalization and Strategy

- There are many regularization strategies.
  1. Put extra constrains on a machine learning model. (Adding restrictions on the parameter values.)
  2. Add extra terms in the objective function that can be thought of as corresponding to a soft constraint on the parameter values.
- If chosen carefully, these extra constraints and penalties can lead to improved performance on the test set.
- Sometimes these constraints and penalties are designed to
  1. **encode** specific kinds of **prior knowledge**.
  2. express a generic preference for a simpler model class in order to promote generalization.
  3. make an under-determined problem determined. (Provide more information)
- Other forms of regularization, known as ensemble methods, combine multiple hypotheses that explain the training data.

Generalization and Strategy

- Principle: Treading increased bias for reduced variance.
- An effective regularizer is one that makes a profitable trade, **reducing variance** significantly while not overly **increasing the bias**.
- In practice, **an overly complex model family does not necessarily include the target function or the true data generating process, or even a close approximation**.
- We almost never have access to the true data generating process so we can never know for sure **if the model family being estimated includes the generating process or not**.

Generalization and Strategy

- Principle: Treading increased bias for reduced variance.
- An effective regularizer is one that makes a profitable trade, **reducing variance** significantly while not overly **increasing the bias**.
- In practice, **an overly complex model family does not necessarily include the target function or the true data generating process, or even a close approximation**.
- We almost never have access to the true data generating process so we can never know for sure **if the model family being estimated includes the generating process or not**.

Generalization and Strategy

- Principle: Treading increased bias for reduced variance.
- An effective regularizer is one that makes a profitable trade, **reducing variance** significantly while not overly **increasing the bias**.
- In practice, **an overly complex model family does not necessarily include the target function or the true data generating process, or even a close approximation**.
- We almost never have access to the true data generating process so we can never know for sure **if the model family being estimated includes the generating process or not**.

Generalization and Strategy

- Principle: Treading increased bias for reduced variance.
- An effective regularizer is one that makes a profitable trade, **reducing variance** significantly while not overly **increasing the bias**.
- In practice, **an overly complex model family does not necessarily include the target function or the true data generating process, or even a close approximation**.
- We almost never have access to the true data generating process so we can never know for sure **if the model family being estimated includes the generating process or not**.

Generalization and Strategy

- However, most applications of deep learning algorithms are to domains where the true data generating process is almost certainly outside the model family.
- Deep learning algorithms are typically applied to **extremely complicated domains** such as images, audio sequences and text, for which the true generation process essentially involves **simulating the entire universe**.
- To some extent, we are always trying to fit a square peg(the data generating process) into a round hole (our model family) 『持方枘 (ruì) 而欲内圆凿』.
- What this means is that controlling the complexity of the model is not a simple matter of finding the model of the right size, with the **right number of parameters**.
- Instead, we might find that the best fitting model is a large model that has been regularized appropriately.
- We now review several strategies for how to create such a large, deep, regularized model.

Generalization and Strategy

- However, most applications of deep learning algorithms are to domains where the true data generating process is almost certainly outside the model family.
- Deep learning algorithms are typically applied to **extremely complicated domains** such as images, audio sequences and text, for which the true generation process essentially involves **simulating the entire universe**.
- To some extent, we are always trying to fit a square peg(the data generating process) into a round hole (our model family) 『持方枘 (ruì) 而欲内圆凿』.
- What this means is that controlling the complexity of the model is not a simple matter of finding the model of the right size, with the **right number of parameters**.
- Instead, we might find that the best fitting model is a large model that has been regularized appropriately.
- We now review several strategies for how to create such a large, deep, regularized model.

Generalization and Strategy

- However, most applications of deep learning algorithms are to domains where the true data generating process is almost certainly outside the model family.
- Deep learning algorithms are typically applied to **extremely complicated domains** such as images, audio sequences and text, for which the true generation process essentially involves **simulating the entire universe**.
- To some extent, we are always trying to fit a square peg(the data generating process) into a round hole (our model family) 『持方枘 (ruì) 而欲内圆凿』.
- What this means is that controlling the complexity of the model is not a simple matter of finding the model of the **right size**, with the **right number of parameters**.
- Instead, we might find that the best fitting model is a large model that has been regularized appropriately.
- We now review several strategies for how to create such a large, deep, regularized model.

Generalization and Strategy

- However, most applications of deep learning algorithms are to domains where the true data generating process is almost certainly outside the model family.
- Deep learning algorithms are typically applied to **extremely complicated domains** such as images, audio sequences and text, for which the true generation process essentially involves **simulating the entire universe**.
- To some extent, we are always trying to fit a square peg(the data generating process) into a round hole (our model family) 『持方枘 (ruì) 而欲内圆凿』.
- What this means is that controlling the complexity of the model is not a simple matter of finding the model of the right size, with the **right number of parameters**.
- Instead, we might find that the best fitting model is a large model that has been regularized appropriately.
- We now review several strategies for how to create such a large, deep, regularized model.

Generalization and Strategy

- However, most applications of deep learning algorithms are to domains where the true data generating process is almost certainly outside the model family.
- Deep learning algorithms are typically applied to **extremely complicated domains** such as images, audio sequences and text, for which the true generation process essentially involves **simulating the entire universe**.
- To some extent, we are always trying to fit a square peg(the data generating process) into a round hole (our model family) 『持方枘 (ruì) 而欲内圆凿』.
- What this means is that controlling the complexity of the model is not a simple matter of finding the model of the right size, with the **right number of parameters**.
- Instead, we might find that the best fitting model is a large model that has been regularized appropriately.
- We now review several strategies for how to create such a large, deep, regularized model.

Generalization and Strategy

- However, most applications of deep learning algorithms are to domains where the true data generating process is almost certainly outside the model family.
- Deep learning algorithms are typically applied to **extremely complicated domains** such as images, audio sequences and text, for which the true generation process essentially involves **simulating the entire universe**.
- To some extent, we are always trying to fit a square peg(the data generating process) into a round hole (our model family) 『持方枘 (ruì) 而欲内圆凿』.
- What this means is that controlling the complexity of the model is not a simple matter of finding the model of the right size, with the **right number of parameters**.
- Instead, we might find that the best fitting model is a large model that has been regularized appropriately.
- We now review several strategies for how to create such a large, deep, regularized model.

Parameter Norm Penalties

- Regularization has been used for decades prior to the advent of deep learning.
- Linear models allow simple straightforward and effective regularization strategies.
- Most approaches are based on limiting the capacity of models by adding a **parameter norm penalty** $\Omega(\theta)$ to the objective function J:

$$\tilde{J}(\boldsymbol{\theta}; X, y) = J(\boldsymbol{\theta}; X, y) + \alpha\Omega(\boldsymbol{\theta})$$

  where $\alpha \in [0, +\infty)$ weights the relative contribution of the norm penalty term.
- Setting α to 0 results in no regularization. Larger values of α correspond to more regularization.
- Optimize both J and norm
- Different Ω has different result.

Parameter Norm Penalties

- Regularization has been used for decades prior to the advent of deep learning.
- Linear models allow simple straightforward and effective regularization strategies.
- Most approaches are based on limiting the capacity of models by adding a **parameter norm penalty** $\Omega(\theta)$ to the objective function J:

$$\tilde{J}(\boldsymbol{\theta}; X, y) = J(\boldsymbol{\theta}; X, y) + \alpha\Omega(\boldsymbol{\theta})$$

  where $\alpha \in [0, +\infty)$ weights the relative contribution of the norm penalty term.
- Setting α to 0 results in no regularization. Larger values of α correspond to more regularization.
- Optimize both J and norm
- Different Ω has different result.

Parameter Norm Penalties

- Regularization has been used for decades prior to the advent of deep learning.
- Linear models allow simple straightforward and effective regularization strategies.
- Most approaches are based on limiting the capacity of models by adding a **parameter norm penalty** $\Omega(\theta)$ to the objective function J:

$$\tilde{J}(\boldsymbol{\theta}; X, \boldsymbol{y}) = J(\boldsymbol{\theta}; X, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

  where $\alpha \in [0, +\infty)$ weights the relative contribution of the norm penalty term.
- Setting α to 0 results in no regularization. Larger values of α correspond to more regularization.
- Optimize both J and norm
- Different Ω has different result.

Parameter Norm Penalties

- Regularization has been used for decades prior to the advent of deep learning.
- Linear models allow simple straightforward and effective regularization strategies.
- Most approaches are based on limiting the capacity of models by adding a **parameter norm penalty** $\Omega(\theta)$ to the objective function J:

$$\tilde{J}(\boldsymbol{\theta}; X, y) = J(\boldsymbol{\theta}; X, y) + \alpha\Omega(\boldsymbol{\theta})$$

  where $\alpha \in [0, +\infty)$ weights the relative contribution of the norm penalty term.
- Setting α to 0 results in no regularization. Larger values of α correspond to more regularization.
- Optimize both J and norm
- Different Ω has different result.

Parameter Norm Penalties

- Regularization has been used for decades prior to the advent of deep learning.
- Linear models allow simple straightforward and effective regularization strategies.
- Most approaches are based on limiting the capacity of models by adding a **parameter norm penalty** $\Omega(\theta)$ to the objective function J:

$$\tilde{J}(\boldsymbol{\theta}; X, y) = J(\boldsymbol{\theta}; X, y) + \alpha\Omega(\boldsymbol{\theta})$$

  where $\alpha \in [0, +\infty)$ weights the relative contribution of the norm penalty term.
- Setting α to 0 results in no regularization. Larger values of α correspond to more regularization.
- Optimize both J and norm
- Different Ω has different result.

Parameter Norm Penalties

- Regularization has been used for decades prior to the advent of deep learning.
- Linear models allow simple straightforward and effective regularization strategies.
- Most approaches are based on limiting the capacity of models by adding a **parameter norm penalty** $\Omega(\theta)$ to the objective function J:

$$\tilde{J}(\boldsymbol{\theta}; X, y) = J(\boldsymbol{\theta}; X, y) + \alpha\Omega(\boldsymbol{\theta})$$

  where $\alpha \in [0, +\infty)$ weights the relative contribution of the norm penalty term.
- Setting α to 0 results in no regularization. Larger values of α correspond to more regularization.
- Optimize both J and norm
- Different Ω has different result.

Parameter Norm Penalties

- Regularization has been used for decades prior to the advent of deep learning.
- Linear models allow simple straightforward and effective regularization strategies.
- Most approaches are based on limiting the capacity of models by adding a **parameter norm penalty** $\Omega(\theta)$ to the objective function J:
  $$\tilde{J}(\boldsymbol{\theta}; X, y) = J(\boldsymbol{\theta}; X, y) + \alpha\Omega(\boldsymbol{\theta})$$
  where $\alpha \in [0, +\infty)$ weights the relative contribution of the norm penalty term.
- Setting α to 0 results in no regularization. Larger values of α correspond to more regularization.
- Optimize both J and norm
- Different Ω has different result.

Parameter Norm Penalties

- We penalize **only the weights** of the affine transformation at each layer and leaves the biases unregularized.
- We do not induce too much variance by leaving the biases unregularized.
- Regularizing the bias parameters can introduce a significant amount of under-fitting.
- We therefore use the vector $\boldsymbol{w}$ to indicate all of the weights that should be affected by a norm penalty, while the vector $\boldsymbol{\theta}$ denotes all of the parameters, including both $\boldsymbol{w}$ and the unregularized parameters.
- Sometime we use a separate penalty with a different α coefficient for each layer.
- But it can be expensive to search for the correct value of multiple hyper-parameters, it is still reasonable to use the same weight decay at all layers just to reduce the size of search space.

Parameter Norm Penalties

- We penalize **only the weights** of the affine transformation at each layer and leaves the biases unregularized.
- We do not induce too much variance by leaving the biases unregularized.
- Regularizing the bias parameters can introduce a significant amount of under-fitting.
- We therefore use the vector w to indicate all of the weights that should be affected by a norm penalty, while the vector θ denotes all of the parameters, including both w and the unregularized parameters.
- Sometime we use a separate penalty with a different α coefficient for each layer.
- But it can be expensive to search for the correct value of multiple hyper-parameters, it is still reasonable to use the same weight decay at all layers just to reduce the size of search space.

Parameter Norm Penalties

- We penalize **only the weights** of the affine transformation at each layer and leaves the biases unregularized.
- We do not induce too much variance by leaving the biases unregularized.
- Regularizing the bias parameters can introduce a significant amount of under-fitting.
- We therefore use the vector $\boldsymbol{w}$ to indicate all of the weights that should be affected by a norm penalty, while the vector $\boldsymbol{\theta}$ denotes all of the parameters, including both $\boldsymbol{w}$ and the unregularized parameters.
- Sometime we use a separate penalty with a different α coefficient for each layer.
- But it can be expensive to search for the correct value of multiple hyper-parameters, it is still reasonable to use the same weight decay at all layers just to reduce the size of search space.

Parameter Norm Penalties

- We penalize **only the weights** of the affine transformation at each layer and leaves the biases unregularized.
- We do not induce too much variance by leaving the biases unregularized.
- Regularizing the bias parameters can introduce a significant amount of under-fitting.
- We therefore use the vector $\boldsymbol{w}$ to indicate all of the weights that should be affected by a norm penalty, while the vector $\boldsymbol{\theta}$ denotes all of the parameters, including both $\boldsymbol{w}$ and the unregularized parameters.
- Sometime we use a separate penalty with a different α coefficient for each layer.
- But it can be expensive to search for the correct value of multiple hyper-parameters, it is still reasonable to use the same weight decay at all layers just to reduce the size of search space.

Parameter Norm Penalties

- We penalize **only the weights** of the affine transformation at each layer and leaves the biases unregularized.
- We do not induce too much variance by leaving the biases unregularized.
- Regularizing the bias parameters can introduce a significant amount of under-fitting.
- We therefore use the vector $\boldsymbol{w}$ to indicate all of the weights that should be affected by a norm penalty, while the vector $\boldsymbol{\theta}$ denotes all of the parameters, including both $\boldsymbol{w}$ and the unregularized parameters.
- Sometime we use a separate penalty with a different α coefficient for each layer.
- But it can be expensive to search for the correct value of multiple hyper-parameters, it is still reasonable to use the same weight decay at all layers just to reduce the size of search space.

Parameter Norm Penalties

- We penalize **only the weights** of the affine transformation at each layer and leaves the biases unregularized.
- We do not induce too much variance by leaving the biases unregularized.
- Regularizing the bias parameters can introduce a significant amount of under-fitting.
- We therefore use the vector $\boldsymbol{w}$ to indicate all of the weights that should be affected by a norm penalty, while the vector $\boldsymbol{\theta}$ denotes all of the parameters, including both $\boldsymbol{w}$ and the unregularized parameters.
- Sometime we use a separate penalty with a different α coefficient for each layer.
- But it can be expensive to search for the correct value of multiple hyper-parameters, it is still reasonable to use the same weight decay at all layers just to reduce the size of search space.

L^2 Parameter Regularization

- The L^2 norm penalty commonly known as *weight decay*.

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|w\|_2^2$$

- This regularization strategy drives the weights closer to the origin. (as well as *ridge regression* or *Tikhonov regularization*)
- We can gain some insight into the behavior of weight decay regularization. (assume no bias for simplification)

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^T w + J(w; X, y)$$

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

- The update

$$w \leftarrow w - \epsilon(\alpha w + \nabla_w J(w; X, y))$$

$$w \leftarrow (1 - \epsilon\alpha) w - \epsilon \nabla_w J(w; X, y)$$

- Shrink the weight vector by a constant factor on each step.
- What happens over the entire course of training?

L^2 Parameter Regularization

- The L^2 norm penalty commonly known as *weight decay*.

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|w\|_2^2$$

- This regularization strategy drives the weights closer to the origin. (as well as *ridge regression* or *Tikhonov regularization*)
- We can gain some insight into the behavior of weight decay regularization. (assume no bias for simplification)

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^T w + J(w; X, y)$$

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

- The update

$$w \leftarrow w - \epsilon(\alpha w + \nabla_w J(w; X, y))$$

$$w \leftarrow (1 - \epsilon\alpha) w - \epsilon \nabla_w J(w; X, y)$$

- Shrink the weight vector by a constant factor on each step.
- What happens over the entire course of training?

L^2 Parameter Regularization

- The L^2 norm penalty commonly known as *weight decay*.

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|w\|_2^2$$

- This regularization strategy drives the weights closer to the origin. (as well as *ridge regression* or *Tikhonov regularization*)
- We can gain some insight into the behavior of weight decay regularization. (assume no bias for simplification)

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^T w + J(w; X, y)$$

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

- The update

$$w \leftarrow w - \epsilon(\alpha w + \nabla_w J(w; X, y))$$

$$w \leftarrow (1 - \epsilon\alpha) w - \epsilon \nabla_w J(w; X, y)$$

- Shrink the weight vector by a constant factor on each step.
- What happens over the entire course of training?

L^2 Parameter Regularization

- The L^2 norm penalty commonly known as *weight decay*.

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|w\|_2^2$$

- This regularization strategy drives the weights closer to the origin. (as well as *ridge regression* or *Tikhonov regularization*)
- We can gain some insight into the behavior of weight decay regularization. (assume no bias for simplification)

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^T w + J(w; X, y)$$

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

- The update

$$w \leftarrow w - \epsilon(\alpha w + \nabla_w J(w; X, y))$$

$$w \leftarrow (1 - \epsilon\alpha) w - \epsilon \nabla_w J(w; X, y)$$

- Shrink the weight vector by a constant factor on each step.
- What happens over the entire course of training?

L^2 Parameter Regularization

- The L^2 norm penalty commonly known as *weight decay*.

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|w\|_2^2$$

- This regularization strategy drives the weights closer to the origin. (as well as *ridge regression* or *Tikhonov regularization*)
- We can gain some insight into the behavior of weight decay regularization. (assume no bias for simplification)

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^T w + J(w; X, y)$$

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

- The update

$$w \leftarrow w - \epsilon(\alpha w + \nabla_w J(w; X, y))$$

$$w \leftarrow (1 - \epsilon\alpha) w - \epsilon \nabla_w J(w; X, y)$$

- Shrink the weight vector by a constant factor on each step.
- What happens over the entire course of training?

L^2 Parameter Regularization

- The L^2 norm penalty commonly known as *weight decay*.

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|w\|_2^2$$

- This regularization strategy drives the weights closer to the origin. (as well as *ridge regression* or *Tikhonov regularization*)
- We can gain some insight into the behavior of weight decay regularization. (assume no bias for simplification)

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^T w + J(w; X, y)$$

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

- The update

$$w \leftarrow w - \epsilon(\alpha w + \nabla_w J(w; X, y))$$

$$w \leftarrow (1 - \epsilon\alpha) w - \epsilon \nabla_w J(w; X, y)$$

- Shrink the weight vector by a constant factor on each step.
- What happens over the entire course of training?

L^2 Parameter Regularization

- The L^2 norm penalty commonly known as *weight decay*.

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|w\|_2^2$$

- This regularization strategy drives the weights closer to the origin. (as well as *ridge regression* or *Tikhonov regularization*)
- We can gain some insight into the behavior of weight decay regularization. (assume no bias for simplification)

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^T w + J(w; X, y)$$

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

- The update

$$w \leftarrow w - \epsilon(\alpha w + \nabla_w J(w; X, y))$$

$$w \leftarrow (1 - \epsilon\alpha) w - \epsilon \nabla_w J(w; X, y)$$

- Shrink the weight vector by a constant factor on each step.
- What happens over the entire course of training?

L^2 Parameter Regularization

- The L^2 norm penalty commonly known as *weight decay*.

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|w\|_2^2$$

- This regularization strategy drives the weights closer to the origin. (as well as *ridge regression* or *Tikhonov regularization*)
- We can gain some insight into the behavior of weight decay regularization. (assume no bias for simplification)

$$\tilde{J}(w; X, y) = \frac{\alpha}{2} w^T w + J(w; X, y)$$

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

- The update

$$w \leftarrow w - \epsilon(\alpha w + \nabla_w J(w; X, y))$$

$$w \leftarrow (1 - \epsilon\alpha) w - \epsilon \nabla_w J(w; X, y)$$

- Shrink the weight vector by a constant factor on each step.
- What happens over the entire course of training?

Recall: Quadratic Approximation

- In mathematics, approximation theory is concerned with how functions can best be approximated with simpler functions.
- **local linear approximation** and **taylor expansion**
  1. For example, when the independent variable of function $y = x^3$ changes, which is Δx, the variation of y is

$$\Delta y = (x + \Delta x)^3 - x^3 = 3x^2 \Delta x + 3x(\Delta x)^2 + (\Delta x)^3$$

  2. When $\Delta x \to 0$, omit last two terms: $\Delta y = 3x^2 \Delta x$
  3. In general:

$$\Delta y = f(x_0 + \Delta x) - f(x_0) \approx f'(x_0) \times \Delta x$$

$$\Delta y = f(x) - f(x_0),\ \Delta x = x - x_0$$

$$f(x) - f(x_0) = f'(x_0) \times (x - x_0)$$

$$f(x) = f(x_0) + f'(x_0)(x - x_0)$$

  4. In order to improve the precision, we can use second-order approximation, which is the second-order Taylor series expansion.

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2$$

Recall: Quadratic Approximation

- In mathematics, approximation theory is concerned with how functions can best be approximated with simpler functions.
- **local linear approximation** and **taylor expansion**
  1. For example, when the independent variable of function $y = x^3$ changes, which is Δx, the variation of y is

$$\Delta y = (x + \Delta x)^3 - x^3 = 3x^2 \Delta x + 3x(\Delta x)^2 + (\Delta x)^3$$

  2. When $\Delta x \to 0$, omit last two terms: $\Delta y = 3x^2 \Delta x$
  3. In general:

$$\Delta y = f(x_0 + \Delta x) - f(x_0) \approx f'(x_0) \times \Delta x$$

$$\Delta y = f(x) - f(x_0),\ \Delta x = x - x_0$$

$$f(x) - f(x_0) = f'(x_0) \times (x - x_0)$$

$$f(x) = f(x_0) + f'(x_0)(x - x_0)$$

  4. In order to improve the precision, we can use second-order approximation, which is the second-order Taylor series expansion.

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2$$

Recall: Quadratic Approximation

- In mathematics, approximation theory is concerned with how functions can best be approximated with simpler functions.
- **local linear approximation** and **taylor expansion**
  1. For example, when the independent variable of function $y = x^3$ changes, which is Δx, the variation of y is

$$\Delta y = (x + \Delta x)^3 - x^3 = 3x^2 \Delta x + 3x(\Delta x)^2 + (\Delta x)^3$$

  2. When $\Delta x \to 0$, omit last two terms: $\Delta y = 3x^2 \Delta x$
  3. In general:

$$\Delta y = f(x_0 + \Delta x) - f(x_0) \approx f'(x_0) \times \Delta x$$

$$\Delta y = f(x) - f(x_0),\ \Delta x = x - x_0$$
$$f(x) - f(x_0) = f'(x_0) \times (x - x_0)$$
$$f(x) = f(x_0) + f'(x_0)(x - x_0)$$

  4. In order to improve the precision, we can use second-order approximation, which is the second-order Taylor series expansion.

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2$$

Recall: Quadratic Approximation

- In mathematics, approximation theory is concerned with how functions can best be approximated with simpler functions.
- **local linear approximation** and **taylor expansion**
  1. For example, when the independent variable of function $y = x^3$ changes, which is Δx, the variation of y is

$$\Delta y = (x + \Delta x)^3 - x^3 = 3x^2 \Delta x + 3x(\Delta x)^2 + (\Delta x)^3$$

  2. When $\Delta x \to 0$, omit last two terms: $\Delta y = 3x^2 \Delta x$
  3. In general:

$$\Delta y = f(x_0 + \Delta x) - f(x_0) \approx f'(x_0) \times \Delta x$$

$$\Delta y = f(x) - f(x_0),\ \Delta x = x - x_0$$

$$f(x) - f(x_0) = f'(x_0) \times (x - x_0)$$

$$f(x) = f(x_0) + f'(x_0)(x - x_0)$$

  4. In order to improve the precision, we can use second-order approximation, which is the second-order Taylor series expansion.

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2$$

Recall: Quadratic Approximation

- In mathematics, approximation theory is concerned with how functions can best be approximated with simpler functions.
- **local linear approximation** and **taylor expansion**
  1. For example, when the independent variable of function $y = x^3$ changes, which is Δx, the variation of y is

$$\Delta y = (x + \Delta x)^3 - x^3 = 3x^2 \Delta x + 3x(\Delta x)^2 + (\Delta x)^3$$

  2. When $\Delta x \to 0$, omit last two terms: $\Delta y = 3x^2 \Delta x$
  3. In general:

$$\Delta y = f(x_0 + \Delta x) - f(x_0) \approx f'(x_0) \times \Delta x$$

$$\Delta y = f(x) - f(x_0),\ \Delta x = x - x_0$$

$$f(x) - f(x_0) = f'(x_0) \times (x - x_0)$$

$$f(x) = f(x_0) + f'(x_0)(x - x_0)$$

  4. In order to improve the precision, we can use second-order approximation, which is the second-order Taylor series expansion.

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2$$

Recall: Quadratic Approximation

- In mathematics, approximation theory is concerned with how functions can best be approximated with simpler functions.
- **local linear approximation** and **taylor expansion**
  1. For example, when the independent variable of function $y = x^3$ changes, which is Δx, the variation of y is

$$\Delta y = (x + \Delta x)^3 - x^3 = 3x^2\Delta x + 3x(\Delta x)^2 + (\Delta x)^3$$

  2. When $\Delta x \to 0$, omit last two terms: $\Delta y = 3x^2\Delta x$
  3. In general:

$$\Delta y = f(x_0 + \Delta x) - f(x_0) \approx f'(x_0) \times \Delta x$$

$$\Delta y = f(x) - f(x_0),\ \Delta x = x - x_0$$

$$f(x) - f(x_0) = f'(x_0) \times (x - x_0)$$

$$f(x) = f(x_0) + f'(x_0)(x - x_0)$$

  4. In order to improve the precision, we can use second-order approximation, which is the second-order Taylor series expansion.

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2$$

Recall: Quadratic Approximation

- In mathematics, approximation theory is concerned with how functions can best be approximated with simpler functions.
- **local linear approximation** and **taylor expansion**
  1. For example, when the independent variable of function $y = x^3$ changes, which is Δx, the variation of y is

$$\Delta y = (x + \Delta x)^3 - x^3 = 3x^2 \Delta x + 3x(\Delta x)^2 + (\Delta x)^3$$

  2. When $\Delta x \to 0$, omit last two terms: $\Delta y = 3x^2 \Delta x$
  3. In general:

$$\Delta y = f(x_0 + \Delta x) - f(x_0) \approx f'(x_0) \times \Delta x$$

$$\Delta y = f(x) - f(x_0),\ \Delta x = x - x_0$$

$$f(x) - f(x_0) = f'(x_0) \times (x - x_0)$$

$$f(x) = f(x_0) + f'(x_0)(x - x_0)$$

  4. In order to improve the precision, we can use second-order approximation, which is the second-order Taylor series expansion.

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2$$

L^2 Parameter Regularization

- Let $w^* = \arg\min_w J(w)$ (unregularized training cost)
- Making a quadratic approximation to the objective function in the neighborhood of the value of the weights. (In DLBook, they used $\hat{J}(\theta)$, but here we use $\hat{J}(w)$ to explain easier)

$$\hat{J}(w) = J(w^*) + \frac{1}{2}(w - w^*)^T \mathrm{H}(w - w^*)$$

- Where H is the Hessian matrix of J with respect to w evaluated at w^*.
- There is no first-order term in this quadratic approximation, because w^* is defined to be a minimum, where the gradient vanishes.
- The minimum of $\hat{J}$ occurs where its gradient

$$\nabla_w \hat{J}(w) = \mathrm{H}(w - w^*)$$

is equal to 0.

L^2 Parameter Regularization

- Let $w^* = \arg\min_w J(w)$ (unregularized training cost)
- Making a quadratic approximation to the objective function in the neighborhood of the value of the weights. (In DLBook, they used $\hat{J}(\theta)$, but here we use $\hat{J}(w)$ to explain easier)

$$\hat{J}(w) = J(w^*) + \frac{1}{2}(w - w^*)^T \mathrm{H}(w - w^*)$$

- Where H is the Hessian matrix of J with respect to w evaluated at w^*.
- There is no first-order term in this quadratic approximation, because w^* is defined to be a minimum, where the gradient vanishes.
- The minimum of $\hat{J}$ occurs where its gradient

$$\nabla_w \hat{J}(w) = \mathrm{H}(w - w^*)$$

is equal to 0.

L^2 Parameter Regularization

- Let $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$ (unregularized training cost)
- Making a quadratic approximation to the objective function in the neighborhood of the value of the weights. (In DLBook, they used $\hat{J}(\boldsymbol{\theta})$, but here we use $\hat{J}(\boldsymbol{w})$ to explain easier)

$$\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \mathbf{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

- Where $\mathbf{H}$ is the Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$.
- There is no first-order term in this quadratic approximation, because $\boldsymbol{w}^*$ is defined to be a minimum, where the gradient vanishes.
- The minimum of $\hat{J}$ occurs where its gradient

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \mathbf{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

is equal to 0.

L^2 Parameter Regularization

- Let $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$ (unregularized training cost)
- Making a quadratic approximation to the objective function in the neighborhood of the value of the weights. (In DLBook, they used $\hat{J}(\boldsymbol{\theta})$, but here we use $\hat{J}(\boldsymbol{w})$ to explain easier)

$$\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \mathbf{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

- Where $\mathbf{H}$ is the Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$.
- There is no first-order term in this quadratic approximation, because $\boldsymbol{w}^*$ is defined to be a minimum, where the gradient vanishes.
- The minimum of $\hat{J}$ occurs where its gradient

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \mathbf{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

is equal to 0.

L^2 Parameter Regularization

- Let $\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w})$ (unregularized training cost)
- Making a quadratic approximation to the objective function in the neighborhood of the value of the weights. (In DLBook, they used $\hat{J}(\boldsymbol{\theta})$, but here we use $\hat{J}(\boldsymbol{w})$ to explain easier)

$$\hat{J}(\boldsymbol{w}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \mathbf{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

- Where $\mathbf{H}$ is the Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$.
- There is no first-order term in this quadratic approximation, because $\boldsymbol{w}^*$ is defined to be a minimum, where the gradient vanishes.
- The minimum of $\hat{J}$ occurs where its gradient

$$\nabla_{\boldsymbol{w}} \hat{J}(\boldsymbol{w}) = \mathbf{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

is equal to 0.

L^2 Parameter Regularization

- To study the effect of weight decay, we modify $\nabla_w \hat{J}(w) = \mathsf{H}(w - w^*)$ by adding the weight decay gradient.
- We can solve for the minimum of the regularized version of $\hat{J}$.
- We use the variable $\tilde{w}$ to represent the location of the minimum.

$$\alpha \tilde{w} + \mathsf{H}(\tilde{w} - w^*) = 0$$

$$(\mathsf{H} + \alpha \mathsf{I})\tilde{w} = \mathsf{H}w^*$$

$$\tilde{w} = \frac{\mathsf{H}w^*}{(\mathsf{H} + \alpha \mathsf{I})}$$

- As α approaches 0, the regularized solution $\tilde{w}$ approaches w^*.
- But what happens as α grows?

L^2 Parameter Regularization

- To study the effect of weight decay, we modify $\nabla_w \hat{J}(w) = \mathsf{H}(w - w^*)$ by adding the weight decay gradient.
- We can solve for the minimum of the regularized version of $\hat{J}$.
- We use the variable $\tilde{w}$ to represent the location of the minimum.

$$\alpha \tilde{w} + \mathsf{H}(\tilde{w} - w^*) = 0$$

$$(\mathsf{H} + \alpha \mathsf{I})\tilde{w} = \mathsf{H}w^*$$

$$\tilde{w} = \frac{\mathsf{H}w^*}{(\mathsf{H} + \alpha \mathsf{I})}$$

- As α approaches 0, the regularized solution $\tilde{w}$ approaches w^*.
- But what happens as α grows?

L^2 Parameter Regularization

- To study the effect of weight decay, we modify $\nabla_w \hat{J}(w) = H(w - w^*)$ by adding the weight decay gradient.
- We can solve for the minimum of the regularized version of $\hat{J}$.
- We use the variable $\tilde{w}$ to represent the location of the minimum.

$$\alpha \tilde{w} + H(\tilde{w} - w^*) = 0$$

$$(H + \alpha I)\tilde{w} = Hw^*$$

$$\tilde{w} = \frac{Hw^*}{(H + \alpha I)}$$

- As α approaches 0, the regularized solution $\tilde{w}$ approaches w^*.
- But what happens as α grows?

L^2 Parameter Regularization

- Because H is real and symmetric, we can decompose it into a diagonal matrix $\mathbf{\Lambda}$ and an orthonormal basis of eigenvectors, Q, such that $\mathrm{H} = Q\mathbf{\Lambda}Q^T$.
- Applying the decomposition $\tilde{w} = (\mathrm{H} + \alpha \mathrm{I})^{-1}\mathrm{H}w^*$

$$
\begin{aligned}
\tilde{w} &= (Q\mathbf{\Lambda}Q^T + \alpha I)^{-1} Q\mathbf{\Lambda}Q^T w^* \\
&= \left[Q(\mathbf{\Lambda} + \alpha I)Q^T\right]^{-1} Q\mathbf{\Lambda}Q^T w^* \\
&= Q(\mathbf{\Lambda} + \alpha I)^{-1}\mathbf{\Lambda}Q^T w^* \\
&= Q\frac{\mathbf{\Lambda}}{\mathbf{\Lambda} + \alpha I}Q^T w^*
\end{aligned}
$$

- We see that the effect of weight decay is to rescale w^* along the axes defined by the eigenvectors of H.
- Specifically, the component of w^* that is aligned with the i-th eigenvector of H is rescaled by a factor of $\frac{\lambda_i}{\lambda_i + \alpha}$

L^2 Parameter Regularization

- Because H is real and symmetric, we can decompose it into a diagonal matrix $\boldsymbol{\Lambda}$ and an orthonormal basis of eigenvectors, Q, such that $\mathsf{H} = Q\boldsymbol{\Lambda}Q^T$.
- Applying the decomposition $\tilde{w} = (\mathsf{H} + \alpha \mathsf{I})^{-1}\mathsf{H}w^*$

$$\begin{aligned}
\tilde{w} &= (Q\boldsymbol{\Lambda}Q^T + \alpha I)^{-1} Q\boldsymbol{\Lambda}Q^T w^* \\
&= \left[Q(\boldsymbol{\Lambda} + \alpha I)Q^T\right]^{-1} Q\boldsymbol{\Lambda}Q^T w^* \\
&= Q(\boldsymbol{\Lambda} + \alpha I)^{-1}\boldsymbol{\Lambda}Q^T w^* \\
&= Q\frac{\boldsymbol{\Lambda}}{\boldsymbol{\Lambda} + \alpha I}Q^T w^*
\end{aligned}$$

- We see that the effect of weight decay is to rescale w^* along the axes defined by the eigenvectors of H.
- Specifically, the component of w^* that is aligned with the i-th eigenvector of H is rescaled by a factor of $\frac{\lambda_i}{\lambda_i + \alpha}$

L^2 Parameter Regularization

- Because H is real and symmetric, we can decompose it into a diagonal matrix $\boldsymbol{\Lambda}$ and an orthonormal basis of eigenvectors, Q, such that $\mathsf{H} = Q\boldsymbol{\Lambda}Q^T$.
- Applying the decomposition $\tilde{w} = (\mathsf{H} + \alpha \mathsf{I})^{-1}\mathsf{H}w^*$

$$
\begin{aligned}
\tilde{w} &= (Q\boldsymbol{\Lambda}Q^T + \alpha I)^{-1} Q\boldsymbol{\Lambda}Q^T w^* \\
&= \left[Q(\boldsymbol{\Lambda} + \alpha I)Q^T\right]^{-1} Q\boldsymbol{\Lambda}Q^T w^* \\
&= Q(\boldsymbol{\Lambda} + \alpha I)^{-1}\boldsymbol{\Lambda}Q^T w^* \\
&= Q\frac{\boldsymbol{\Lambda}}{\boldsymbol{\Lambda} + \alpha I}Q^T w^*
\end{aligned}
$$

- We see that the effect of weight decay is to rescale w^* along the axes defined by the eigenvectors of H.
- Specifically, the component of w^* that is aligned with the i-th eigenvector of H is rescaled by a factor of $\frac{\lambda_i}{\lambda_i + \alpha}$

L^2 Parameter Regularization

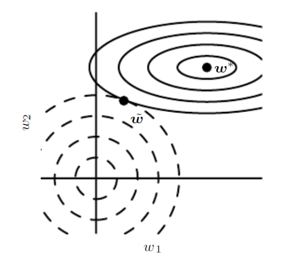
- Because H is real and symmetric, we can decompose it into a diagonal matrix $\boldsymbol{\Lambda}$ and an orthonormal basis of eigenvectors, Q, such that $\mathsf{H} = Q\boldsymbol{\Lambda}Q^T$.
- Applying the decomposition $\tilde{w} = (\mathsf{H} + \alpha \mathsf{I})^{-1}\mathsf{H}w^*$

$$\begin{aligned}
\tilde{w} &= (Q\boldsymbol{\Lambda}Q^T + \alpha I)^{-1} Q\boldsymbol{\Lambda}Q^T w^* \\
&= \left[Q(\boldsymbol{\Lambda} + \alpha I)Q^T\right]^{-1} Q\boldsymbol{\Lambda}Q^T w^* \\
&= Q(\boldsymbol{\Lambda} + \alpha I)^{-1}\boldsymbol{\Lambda}Q^T w^* \\
&= Q\frac{\boldsymbol{\Lambda}}{\boldsymbol{\Lambda} + \alpha I}Q^T w^*
\end{aligned}$$

- We see that the effect of weight decay is to rescale w^* along the axes defined by the eigenvectors of H.
- Specifically, the component of w^* that is aligned with the i-th eigenvector of H is rescaled by a factor of $\frac{\lambda_i}{\lambda_i + \alpha}$

This effect is illustrated in figure:

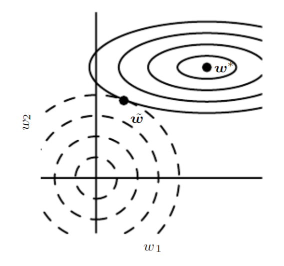
Fig. 1: An illustration of the effect of L^2 (or weight decay) regularization on the value of the optimal w



- The solid ellipses represent contours of equal value of the **unregularized objective**.
- The dotted circles represent contours of equal value of the L^2 regularizer.
- At the point $\tilde{w}$, these competing objectives reach an equilibrium.

This effect is illustrated in figure:

Fig. 1: An illustration of the effect of L^2 (or weight decay) regularization on the value of the optimal w

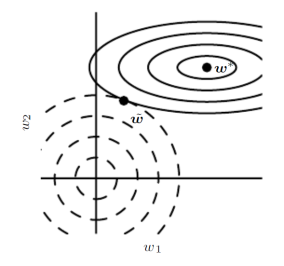


- The solid ellipses represent contours of equal value of the **unregularized objective**.
- The dotted circles represent contours of equal value of the L^2 regularizer.
- At the point $\tilde{w}$, these competing objectives reach an equilibrium.

L^2 Parameter Regularization

This effect is illustrated in figure:

Fig. 1: An illustration of the effect of L^2 (or weight decay) regularization on the value of the optimal w

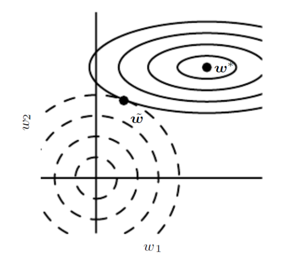


- The solid ellipses represent contours of equal value of the **unregularized objective**.
- The dotted circles represent contours of equal value of the L^2 regularizer.
- At the point $\tilde{w}$, these competing objectives reach an equilibrium.

L^2 Parameter Regularization

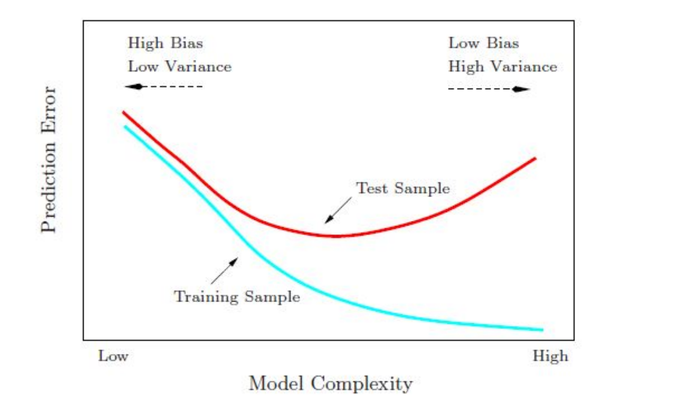
This effect is illustrated in figure:

Fig. 1: An illustration of the effect of L^2 (or weight decay) regularization on the value of the optimal $\boldsymbol{w}$



- The solid ellipses represent contours of equal value of the **unregularized objective**.
- The dotted circles represent contours of equal value of the L^2 regularizer.
- At the point $\tilde{\boldsymbol{w}}$, these competing objectives reach an equilibrium.

L^2 Parameter Regularization



L^2 Parameter Regularization

- How do these effects relate to machine learning in particular?
- We can find out by studying linear regression, the cost function is the sum of squared errors:

$$(Xw - y)^T(Xw - y)$$

- Add L^2 regularization, the objective function changes to:

$$(Xw - y)^T(Xw - y) + \frac{1}{2}\alpha w^T w$$

- This changes the normal equations for the solution from:

$$w = (X^TX)^{-1}X^Ty \text{ to } w = (X^TX + \alpha I)^{-1}X^Ty$$

- The new matrix has the addition of α to the diagonal.
- Diagonal correspond to the variance of each input feature.
- We can see that L^2 regularization causes the learning algorithm to "perceive" the input X as having higher variance, which makes it shrink the weights on features whose covariance with the output target is low compared to this added variance.

L^2 Parameter Regularization

- How do these effects relate to machine learning in particular?
- We can find out by studying linear regression, the cost function is the sum of squared errors:

$$(Xw - y)^T (Xw - y)$$

- Add L^2 regularization, the objective function changes to:

$$(Xw - y)^T (Xw - y) + \frac{1}{2} \alpha w^T w$$

- This changes the normal equations for the solution from:

$$w = (X^T X)^{-1} X^T y \text{ to } w = (X^T X + \alpha I)^{-1} X^T y$$

- The new matrix has the addition of α to the diagonal.
- Diagonal correspond to the variance of each input feature.
- We can see that L^2 regularization causes the learning algorithm to "perceive" the input X as having higher variance, which makes it shrink the weights on features whose covariance with the output target is low compared to this added variance.

L^2 Parameter Regularization

- How do these effects relate to machine learning in particular?
- We can find out by studying linear regression, the cost function is the sum of squared errors:

$$(Xw - y)^T(Xw - y)$$

- Add L^2 regularization, the objective function changes to:

$$(Xw - y)^T(Xw - y) + \frac{1}{2}\alpha w^T w$$

- This changes the normal equations for the solution from:

$$w = (X^TX)^{-1}X^Ty \text{ to } w = (X^TX + \alpha I)^{-1}X^Ty$$

- The new matrix has the addition of α to the diagonal.
- Diagonal correspond to the variance of each input feature.
- We can see that L^2 regularization causes the learning algorithm to "perceive" the input X as having higher variance, which makes it shrink the weights on features whose covariance with the output target is low compared to this added variance.

L^2 Parameter Regularization

- How do these effects relate to machine learning in particular?
- We can find out by studying linear regression, the cost function is the sum of squared errors:

$$(Xw - y)^T (Xw - y)$$

- Add L^2 regularization, the objective function changes to:

$$(Xw - y)^T (Xw - y) + \frac{1}{2}\alpha w^T w$$

- This changes the normal equations for the solution from:

$$w = (X^T X)^{-1} X^T y \text{ to } w = (X^T X + \alpha I)^{-1} X^T y$$

- The new matrix has the addition of α to the diagonal.
- Diagonal correspond to the variance of each input feature.
- We can see that L^2 regularization causes the learning algorithm to "perceive" the input X as having higher variance, which makes it shrink the weights on features whose covariance with the output target is low compared to this added variance.

L^2 Parameter Regularization

- How do these effects relate to machine learning in particular?
- We can find out by studying linear regression, the cost function is the sum of squared errors:

$$(Xw - y)^T(Xw - y)$$

- Add L^2 regularization, the objective function changes to:

$$(Xw - y)^T(Xw - y) + \frac{1}{2}\alpha w^T w$$

- This changes the normal equations for the solution from:

$$w = (X^T X)^{-1} X^T y \text{ to } w = (X^T X + \alpha I)^{-1} X^T y$$

- The new matrix has the addition of α to the diagonal.
- Diagonal correspond to the variance of each input feature.
- We can see that L^2 regularization causes the learning algorithm to ”perceive” the input X as having higher variance, which makes it shrink the weights on features whose covariance with the output target is low compared to this added variance.

L^2 Parameter Regularization

- How do these effects relate to machine learning in particular?
- We can find out by studying linear regression, the cost function is the sum of squared errors:

$$(Xw - y)^T(Xw - y)$$

- Add L^2 regularization, the objective function changes to:

$$(Xw - y)^T(Xw - y) + \frac{1}{2}\alpha w^T w$$

- This changes the normal equations for the solution from:

$$w = (X^T X)^{-1} X^T y \text{ to } w = (X^T X + \alpha I)^{-1} X^T y$$

- The new matrix has the addition of α to the diagonal.
- Diagonal correspond to the variance of each input feature.
- We can see that L^2 regularization causes the learning algorithm to "perceive" the input X as having higher variance, which makes it shrink the weights on features whose covariance with the output target is low compared to this added variance.

L^2 Parameter Regularization

- How do these effects relate to machine learning in particular?
- We can find out by studying linear regression, the cost function is the sum of squared errors:

$$(Xw - y)^T(Xw - y)$$

- Add L^2 regularization, the objective function changes to:

$$(Xw - y)^T(Xw - y) + \frac{1}{2}\alpha w^T w$$

- This changes the normal equations for the solution from:

$$w = (X^T X)^{-1} X^T y \text{ to } w = (X^T X + \alpha I)^{-1} X^T y$$

- The new matrix has the addition of α to the diagonal.
- Diagonal correspond to the variance of each input feature.
- We can see that L^2 regularization causes the learning algorithm to "perceive" the input X as having higher variance, which makes it shrink the weights on features whose covariance with the output target is low compared to this added variance.

L^1 Regularization

- L^1 regularization on the model parameter w is defined as:

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

- We will now discuss the effect of L^1 regularization on the simple linear regression model, with no bias parameters, that we studied in our analysis of L^2 regularization.
- In particular, we are interested in delineating the differences between L^1 and L^2 forms of regularization.

L^1 Regularization

- L^1 regularization on the model parameter w is defined as:

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

- We will now discuss the effect of L^1 regularization on the simple linear regression model, with no bias parameters, that we studied in our analysis of L^2 regularization.
- In particular, we are interested in delineating the differences between L^1 and L^2 forms of regularization.

L^1 Regularization

- As with L^2 weight decay, L^1 weight decay controls the strength of the regularization by scaling the penalty Ω using a positive hyperparameter α.
- Thus, the regularized objective function $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is given by

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

  with the corresponding gradient:

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \operatorname{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

  where $\operatorname{sign}(\boldsymbol{w})$ is simply the sign of $\boldsymbol{w}$ applied element-wise.

L^1 Regularization

- As with L^2 weight decay, L^1 weight decay controls the strength of the regularization by scaling the penalty Ω using a positive hyperparameter α.
- Thus, the regularized objective function $\tilde{J}(w; X, y)$ is given by

$$\tilde{J}(w; X, y) = \alpha \|w\|_1 + J(w; X, y)$$

with the corresponding gradient:

$$\nabla_w \tilde{J}(w; X, y) = \alpha \mathrm{sign}(w) + \nabla_w J(w; X, y)$$

where $\mathrm{sign}(w)$ is simply the sign of w applied element-wise.

L^1 Regularization

- As with L^2 weight decay, L^1 weight decay controls the strength of the regularization by scaling the penalty Ω using a positive hyperparameter α.
- Thus, the regularized objective function $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is given by

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

  with the corresponding gradient:

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \operatorname{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

  where $\operatorname{sign}(\boldsymbol{w})$ is simply the sign of $\boldsymbol{w}$ applied element-wise.

L^1 Regularization

- As with L^2 weight decay, L^1 weight decay controls the strength of the regularization by scaling the penalty Ω using a positive hyperparameter α.
- Thus, the regularized objective function $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is given by

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

  with the corresponding gradient:

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \operatorname{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

  where $\operatorname{sign}(\boldsymbol{w})$ is simply the sign of $\boldsymbol{w}$ applied element-wise.

$$\nabla_w \tilde{J}(w; X, y) = \alpha \mathrm{sign}(w) + \nabla_w J(w; X, y)$$

- From this equation, we can see that the effect of L^1 regularization is quite different from that of L^2 regularization.
- We can see that the regularization contribution to the gradient no longer scales linearly with each w_i; instead it is a constant factor with a sign equal to $\mathrm{sign}(w_i)$.
- One consequence of this form of the gradient is that we will not necessarily see clean algebraic solutions to quadratic approximations of $J(X, y; w)$ as we did for L^2 regularization.

$$\nabla_w \tilde{J}(w; X, y) = \alpha \mathrm{sign}(w) + \nabla_w J(w; X, y)$$

- From this equation, we can see that the effect of L^1 regularization is quite different from that of L^2 regularization.
- We can see that the regularization contribution to the gradient no longer scales linearly with each w_i; instead it is a constant factor with a sign equal to $\mathrm{sign}(w_i)$.
- One consequence of this form of the gradient is that we will not necessarily see clean algebraic solutions to quadratic approximations of $J(X, y; w)$ as we did for L^2 regularization.

$$\nabla_w \tilde{J}(w; X, y) = \alpha \mathrm{sign}(w) + \nabla_w J(w; X, y)$$

- From this equation, we can see that the effect of L^1 regularization is quite different from that of L^2 regularization.
- We can see that the regularization contribution to the gradient no longer scales linearly with each w_i; instead it is a constant factor with a sign equal to $\mathrm{sign}(w_i)$.
- One consequence of this form of the gradient is that we will not necessarily see clean algebraic solutions to quadratic approximations of $J(X, y; w)$ as we did for L^2 regularization.

L^1 Regularization

- Out simple linear model has a quadratic cost function that we can represent via its Taylor series.
- Alternately, we could imaging that this is a truncated Taylor series approximating the cost function of a more sophisticated model.
- The gradient in this setting is given by

$$\nabla_w \tilde{J}(w) = \mathbf{H}(w - w^*)$$

- Because the L^1 penalty does not admit clean algebraic expressions in the case of a full general Hessian, we will also make the further simplifying assumption that the Hessian is a diagonal, $\mathbf{H} = \mathrm{diag}([H_{1,1}, \dots, H_{n,n}])$, where each $H_{i,i} > 0$.
- This assumption holds if the data for the linear regression problem has been preprocessed to remove all correlation between the input features, which may be accomplished using PCA.

L^1 Regularization

- Out simple linear model has a quadratic cost function that we can represent via its Taylor series.
- Alternately, we could imaging that this is a truncated Taylor series approximating the cost function of a more sophisticated model.
- The gradient in this setting is given by

$$\nabla_w \tilde{J}(w) = \mathbf{H}(w - w^*)$$

- Because the L^1 penalty does not admit clean algebraic expressions in the case of a full general Hessian, we will also make the further simplifying assumption that the Hessian is a diagonal, $\mathbf{H} = \mathrm{diag}([H_{1,1}, \ldots, H_{n,n}])$, where each $H_{i,i} > 0$.
- This assumption holds if the data for the linear regression problem has been preprocessed to remove all correlation between the input features, which may be accomplished using PCA.

L^1 Regularization

- Out simple linear model has a quadratic cost function that we can represent via its Taylor series.
- Alternately, we could imaging that this is a truncated Taylor series approximating the cost function of a more sophisticated model.
- The gradient in this setting is given by

$$\nabla_w \tilde{J}(w) = \mathsf{H}(w - w^*)$$

- Because the L^1 penalty does not admit clean algebraic expressions in the case of a full general Hessian, we will also make the further simplifying assumption that the Hessian is a diagonal, $\mathbf{H} = \mathrm{diag}([H_{1,1}, \ldots, H_{n,n}])$, where each $H_{i,i} > 0$.
- This assumption holds if the data for the linear regression problem has been preprocessed to remove all correlation between the input features, which may be accomplished using PCA.

L^1 Regularization

- Out simple linear model has a quadratic cost function that we can represent via its Taylor series.
- Alternately, we could imaging that this is a truncated Taylor series approximating the cost function of a more sophisticated model.
- The gradient in this setting is given by

$$\nabla_w \tilde{J}(w) = \mathsf{H}(w - w^*)$$

- Because the L^1 penalty does not admit clean algebraic expressions in the case of a full general Hessian, we will also make the further simplifying assumption that the Hessian is a diagonal, $\mathsf{H} = \mathrm{diag}([H_{1,1}, \ldots, H_{n,n}])$, where each $H_{i,i} > 0$.
- This assumption holds if the data for the linear regression problem has been preprocessed to remove all correlation between the input features, which may be accomplished using PCA.

L^1 Regularization

- Out simple linear model has a quadratic cost function that we can represent via its Taylor series.
- Alternately, we could imaging that this is a truncated Taylor series approximating the cost function of a more sophisticated model.
- The gradient in this setting is given by

$$\nabla_w \tilde{J}(w) = \mathsf{H}(w - w^*)$$

- Because the L^1 penalty does not admit clean algebraic expressions in the case of a full general Hessian, we will also make the further simplifying assumption that the Hessian is a diagonal, $\mathsf{H} = \mathrm{diag}([H_{1,1}, \ldots, H_{n,n}])$, where each $H_{i,i} > 0$.
- This assumption holds if the data for the linear regression problem has been preprocessed to remove all correlation between the input features, which may be accomplished using PCA.

L^1 Regularization

- Our quadratic approximation of the L^1 regularized objective function decomposes into a sum over the parameters:

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2} H_{i,i} (\boldsymbol{w}_i - \boldsymbol{w}_i^*)^2 + \alpha \left| w_i \right| \right]$$

- The problem of minimizing this approximate cost function has an analytical solution (for each dimension i), with the following form:

$$w_i = \mathrm{sign}(w_i^*) max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}$$

L^1 Regularization

- Our quadratic approximation of the L^1 regularized objective function decomposes into a sum over the parameters:

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2} H_{i,i}(\boldsymbol{w}_i - \boldsymbol{w}_i^*)^2 + \alpha |w_i|\right]$$

- The problem of minimizing this approximate cost function has an analytical solution (for each dimension i), with the following form:

$$w_i = \mathrm{sign}(w_i^*) max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}$$

L^1 Regularization

$$w_i = \mathrm{sign}(w_i^*) max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}$$

- Consider the situation where $w_i^* > 0$ for all i. There are two possible outcomes:
  1. The case where $w_i^* \leq \frac{\alpha}{H_{i,i}}$. Here the optimal value of w_i under the regularized objective is simply $w_i = 0$. This occurs because the contribution of $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ to the regularized objective $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is overwhelmed–in direction i–by the L^1 regularization which pushes the value of w_i to zero.
  2. The case where $w_i^* > \frac{\alpha}{H_{i,i}}$. In this case, the regularization does not move the optimal value of w_i to zero but instead it just shifts it in that direction by a distance equal to $\frac{\alpha}{H_{i,i}}$.
- A similar process happens when $w_i^* < 0$, but with the L^1 penalty making w_i less negative by $\frac{\alpha}{H_{i,i}}$, or 0.

L^1 Regularization

$$w_i = \text{sign}(w_i^*) max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}$$

- Consider the situation where $w_i^* > 0$ for all i. There are two possible outcomes:
  1. The case where $w_i^* \leq \frac{\alpha}{H_{i,i}}$. Here the optimal value of w_i under the regularized objective is simply $w_i = 0$. This occurs because the contribution of $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ to the regularized objective $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is overwhelmed–in direction i–by the L^1 regularization which pushes the value of w_i to zero.
  2. The case where $w_i^* > \frac{\alpha}{H_{i,i}}$. In this case, the regularization does not move the optimal value of w_i to zero but instead it just shifts it in that direction by a distance equal to $\frac{\alpha}{H_{i,i}}$.
- A similar process happens when $w_i^* < 0$, but with the L^1 penalty making w_i less negative by $\frac{\alpha}{H_{i,i}}$, or 0.

L^1 Regularization

$$w_i = \mathrm{sign}(w_i^*) max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}$$

- Consider the situation where $w_i^* > 0$ for all i. There are two possible outcomes:
  1. The case where $w_i^* \leq \frac{\alpha}{H_{i,i}}$. Here the optimal value of w_i under the regularized objective is simply $w_i = 0$. This occurs because the contribution of $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ to the regularized objective $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is overwhelmed–in direction i–by the L^1 regularization which pushes the value of w_i to zero.
  2. The case where $w_i^* > \frac{\alpha}{H_{i,i}}$. In this case, the regularization does not move the optimal value of w_i to zero but instead it just shifts it in that direction by a distance equal to $\frac{\alpha}{H_{i,i}}$.
- A similar process happens when $w_i^* < 0$, but with the L^1 penalty making w_i less negative by $\frac{\alpha}{H_{i,i}}$, or 0.

L^1 Regularization

$$w_i = \text{sign}(w_i^*) max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}$$

- Consider the situation where $w_i^* > 0$ for all i. There are two possible outcomes:
  1. The case where $w_i^* \leq \frac{\alpha}{H_{i,i}}$. Here the optimal value of w_i under the regularized objective is simply $w_i = 0$. This occurs because the contribution of $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ to the regularized objective $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is overwhelmed–in direction i–by the L^1 regularization which pushes the value of w_i to zero.
  2. The case where $w_i^* > \frac{\alpha}{H_{i,i}}$. In this case, the regularization does not move the optimal value of w_i to zero but instead it just shifts it in that direction by a distance equal to $\frac{\alpha}{H_{i,i}}$.
- A similar process happens when $w_i^* < 0$, but with the L^1 penalty making w_i less negative by $\frac{\alpha}{H_{i,i}}$, or 0.

L^1 Regularization

$$w_i = \text{sign}(w_i^*) max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}$$

- Consider the situation where $w_i^* > 0$ for all i. There are two possible outcomes:
  1. The case where $w_i^* \leq \frac{\alpha}{H_{i,i}}$. Here the optimal value of w_i under the regularized objective is simply $w_i = 0$. This occurs because the contribution of $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ to the regularized objective $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is overwhelmed–in direction i–by the L^1 regularization which pushes the value of w_i to zero.
  2. The case where $w_i^* > \frac{\alpha}{H_{i,i}}$. In this case, the regularization does not move the optimal value of w_i to zero but instead it just shifts it in that direction by a distance equal to $\frac{\alpha}{H_{i,i}}$.
- A similar process happens when $w_i^* < 0$, but with the L^1 penalty making w_i less negative by $\frac{\alpha}{H_{i,i}}$, or 0.

L^1 Regularization

- In comparison to L^2 regularization, L^1 regularization results in a solution that is more *sparse*.
- Sparsity in this context refers to the fact that some parameters have an optimal value of zero.
- The sparsity property induced by L^1 regularization has been used extensively as a *feature selection* mechanism.
- Feature selection simplifies a machine learning problem by choosing which subset of the available features should be used.
- In particular, the well known LASSO ([Tibshirani, 1996]) (least absolute shrinkage and selection operator) model integrates an L^1 penalty with a linear model and a least squares cost function.

L^1 Regularization

- In comparison to L^2 regularization, L^1 regularization results in a solution that is more *sparse*.
- Sparsity in this context refers to the fact that some parameters have an optimal value of zero.
- The sparsity property induced by L^1 regularization has been used extensively as a *feature selection* mechanism.
- Feature selection simplifies a machine learning problem by choosing which subset of the available features should be used.
- In particular, the well known LASSO ([Tibshirani, 1996]) (least absolute shrinkage and selection operator) model integrates an L^1 penalty with a linear model and a least squares cost function.

L^1 Regularization

- In comparison to L^2 regularization, L^1 regularization results in a solution that is more *sparse*.
- Sparsity in this context refers to the fact that some parameters have an optimal value of zero.
- The sparsity property induced by L^1 regularization has been used extensively as a *feature selection* mechanism.
- Feature selection simplifies a machine learning problem by choosing which subset of the available features should be used.
- In particular, the well known LASSO ([Tibshirani, 1996]) (least absolute shrinkage and selection operator) model integrates an L^1 penalty with a linear model and a least squares cost function.

L^1 Regularization

- In comparison to L^2 regularization, L^1 regularization results in a solution that is more *sparse*.
- Sparsity in this context refers to the fact that some parameters have an optimal value of zero.
- The sparsity property induced by L^1 regularization has been used extensively as a *feature selection* mechanism.
- Feature selection simplifies a machine learning problem by choosing which subset of the available features should be used.
- In particular, the well known LASSO ([Tibshirani, 1996]) (least absolute shrinkage and selection operator) model integrates an L^1 penalty with a linear model and a least squares cost function.

L^1 Regularization

- In comparison to L^2 regularization, L^1 regularization results in a solution that is more *sparse*.
- Sparsity in this context refers to the fact that some parameters have an optimal value of zero.
- The sparsity property induced by L^1 regularization has been used extensively as a *feature selection* mechanism.
- Feature selection simplifies a machine learning problem by choosing which subset of the available features should be used.
- In particular, the well known LASSO ([Tibshirani, 1996]) (least absolute shrinkage and selection operator) model integrates an L^1 penalty with a linear model and a least squares cost function.

Sparsity? L^1 and L^2

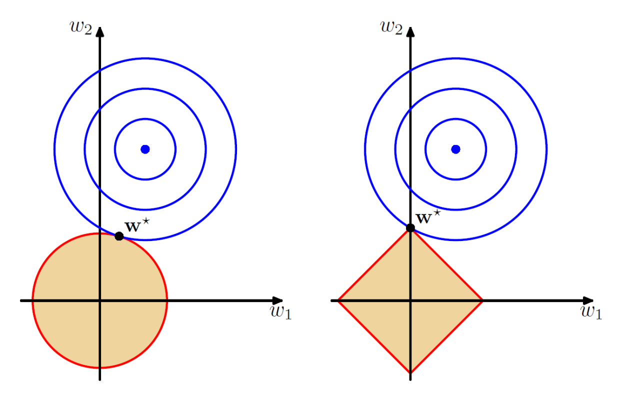


Fig. 2: Plot of the contours of the unregularized error function (blue) along with the constraint region for the quadratic regularizer on the left and the lasso regularizer on the right.

Norm Penalties as Constrained Optimization

- Consider the cost function regularized by a parameter norm penalty:

$$\tilde{J}(\boldsymbol{\theta}; X, y) = J(\boldsymbol{\theta}; X, y) + \alpha\Omega(\boldsymbol{\theta})$$

- If we want to constrain $\Omega(\boldsymbol{\theta})$ to be less than some constant k, we could construct a generalized Lagrange function

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\alpha}; X, y) = J(\boldsymbol{\theta}; X, y) + \alpha(\Omega(\theta) - k)$$

- The solution to the constrained problem is given by

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\alpha})$$

Norm Penalties as Constrained Optimization

- Consider the cost function regularized by a parameter norm penalty:
$$\tilde{J}(\boldsymbol{\theta}; X, y) = J(\boldsymbol{\theta}; X, y) + \alpha\Omega(\boldsymbol{\theta})$$
- If we want to constrain $\Omega(\boldsymbol{\theta})$ to be less than some constant k, we could construct a generalized Lagrange function
$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\alpha}; X, y) = J(\boldsymbol{\theta}; X, y) + \alpha(\Omega(\theta) - k)$$
- The solution to the constrained problem is given by
$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\alpha})$$

Norm Penalties as Constrained Optimization

- Consider the cost function regularized by a parameter norm penalty:

$$\tilde{J}(\boldsymbol{\theta}; X, y) = J(\boldsymbol{\theta}; X, y) + \alpha\Omega(\boldsymbol{\theta})$$

- If we want to constrain $\Omega(\boldsymbol{\theta})$ to be less than some constant k, we could construct a generalized Lagrange function

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\alpha}; X, y) = J(\boldsymbol{\theta}; X, y) + \alpha(\Omega(\theta) - k)$$

- The solution to the constrained problem is given by

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\alpha})$$

Norm Penalties as Constrained Optimization

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\alpha})$$

- Solving this problem requires modifying both $\boldsymbol{\theta}$ and α.
- Many different procedures are possible–some may use gradient descent, while others may use analytical solutions for where the gradient is zero–but in all procedures α must increase whenever $\Omega(\boldsymbol{\theta}) > k$ and decrease whenever $\Omega(\boldsymbol{\theta}) < k$.
- All positive α encourage $\Omega(\boldsymbol{\theta})$ to shrink.
- The optimal value a^* will encourage $\Omega(\theta)$ to shrink, but not so strongly to make $\Omega(\boldsymbol{\theta})$ become less than k.

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\alpha})$$

- Solving this problem requires modifying both $\boldsymbol{\theta}$ and α.
- Many different procedures are possible–some may use gradient descent, while others may use analytical solutions for where the gradient is zero–but in all procedures α must increase whenever $\Omega(\boldsymbol{\theta}) > k$ and decrease whenever $\Omega(\boldsymbol{\theta}) < k$.
- All positive α encourage $\Omega(\boldsymbol{\theta})$ to shrink.
- The optimal value α^* will encourage $\Omega(\theta)$ to shrink, but not so strongly to make $\Omega(\boldsymbol{\theta})$ become less than k.

Norm Penalties as Constrained Optimization

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\alpha})$$

- Solving this problem requires modifying both $\boldsymbol{\theta}$ and α.
- Many different procedures are possible–some may use gradient descent, while others may use analytical solutions for where the gradient is zero–but in all procedures α must increase whenever $\Omega(\boldsymbol{\theta}) > k$ and decrease whenever $\Omega(\boldsymbol{\theta}) < k$.
- All positive α encourage $\Omega(\boldsymbol{\theta})$ to shrink.
- The optimal value α^* will encourage $\Omega(\theta)$ to shrink, but not so strongly to make $\Omega(\boldsymbol{\theta})$ become less than k.

Norm Penalties as Constrained Optimization

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \max_{\alpha, \alpha \geq 0} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\alpha})$$

- Solving this problem requires modifying both $\boldsymbol{\theta}$ and α.
- Many different procedures are possible–some may use gradient descent, while others may use analytical solutions for where the gradient is zero–but in all procedures α must increase whenever $\Omega(\boldsymbol{\theta}) > k$ and decrease whenever $\Omega(\boldsymbol{\theta}) < k$.
- All positive α encourage $\Omega(\boldsymbol{\theta})$ to shrink.
- The optimal value a^* will encourage $\Omega(\theta)$ to shrink, but not so strongly to make $\Omega(\boldsymbol{\theta})$ become less than k.

Norm Penalties as Constrained Optimization

- To gain some insight into the effect of the constraint, we can fix α^* and view the problem as just a function of θ:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \alpha^*) = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; X, y) + \alpha * \Omega(\boldsymbol{\theta})$$

- This is exactly the same as the regularized training problem of minimizing $\tilde{J}$.
- We can thus think of a parameter norm penalty as imposing a constraint on the weights.
- If Ω is the L^2 norm, then the weights are constrained to lie in an L^2 ball.
- If Ω is the L^1 norm, then the weights are constrained to lie in a region of limited L^1 norm.

Norm Penalties as Constrained Optimization

- To gain some insight into the effect of the constraint, we can fix α^* and view the problem as just a function of θ:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \alpha^*) = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha * \Omega(\boldsymbol{\theta})$$

- This is exactly the same as the regularized training problem of minimizing $\tilde{J}$.
- We can thus think of a parameter norm penalty as imposing a constraint on the weights.
- If Ω is the L^2 norm, then the weights are constrained to lie in an L^2 ball.
- If Ω is the L^1 norm, then the weights are constrained to lie in a region of limited L^1 norm.

Norm Penalties as Constrained Optimization

- To gain some insight into the effect of the constraint, we can fix α^* and view the problem as just a function of θ:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \alpha^*) = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha * \Omega(\boldsymbol{\theta})$$

- This is exactly the same as the regularized training problem of minimizing $\tilde{J}$.
- We can thus think of a parameter norm penalty as imposing a constraint on the weights.
- If Ω is the L^2 norm, then the weights are constrained to lie in an L^2 ball.
- If Ω is the L^1 norm, then the weights are constrained to lie in a region of limited L^1 norm.

Norm Penalties as Constrained Optimization

- To gain some insight into the effect of the constraint, we can fix α^* and view the problem as just a function of θ:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \alpha^*) = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha * \Omega(\boldsymbol{\theta})$$

- This is exactly the same as the regularized training problem of minimizing $\tilde{J}$.
- We can thus think of a parameter norm penalty as imposing a constraint on the weights.
- If Ω is the L^2 norm, then the weights are constrained to lie in an L^2 ball.
- If Ω is the L^1 norm, then the weights are constrained to lie in a region of limited L^1 norm.

Regularization and Under-Constrained Problems

- In some cases, regularization is necessary.
- Many linear models in machine learning, including linear regression and PCA, depend on inverting the matrix $X^T X$.
- This is not possible whenever $X^T X$ is singular.
- This matrix can be singular whenever the data generating distribution truly has no variance in some direction, or when no variance in **observed** in some direction because there are fewer examples (rows of X) than input features (columns of X).
- In this case, many forms of regularization correspond to inverting $X^T X + \alpha I$ instead. This regularized matrix is guaranteed to be invertible.

Regularization and Under-Constrained Problems

- In some cases, regularization is necessary.
- Many linear models in machine learning, including linear regression and PCA, depend on inverting the matrix $X^T X$.
- This is not possible whenever $X^T X$ is singular.
- This matrix can be singular whenever the data generating distribution truly has no variance in some direction, or when no variance in **observed** in some direction because there are fewer examples (rows of X) than input features (columns of X).
- In this case, many forms of regularization correspond to inverting $X^T X + \alpha I$ instead. This regularized matrix is guaranteed to be invertible.

Regularization and Under-Constrained Problems

- In some cases, regularization is necessary.
- Many linear models in machine learning, including linear regression and PCA, depend on inverting the matrix $X^T X$.
- This is not possible whenever $X^T X$ is singular.
- This matrix can be singular whenever the data generating distribution truly has no variance in some direction, or when no variance in **observed** in some direction because there are fewer examples (rows of X) than input features (columns of X).
- In this case, many forms of regularization correspond to inverting $X^T X + \alpha I$ instead. This regularized matrix is guaranteed to be invertible.

Regularization and Under-Constrained Problems

- In some cases, regularization is necessary.
- Many linear models in machine learning, including linear regression and PCA, depend on inverting the matrix X^TX.
- This is not possible whenever X^TX is singular.
- This matrix can be singular whenever the data generating distribution truly has no variance in some direction, or when no variance in **observed** in some direction because there are fewer examples (rows of X) than input features (columns of X).
- In this case, many forms of regularization correspond to inverting $X^TX + \alpha I$ instead. This regularized matrix is guaranteed to be invertible.

Regularization and Under-Constrained Problems

- In some cases, regularization is necessary.
- Many linear models in machine learning, including linear regression and PCA, depend on inverting the matrix X^TX.
- This is not possible whenever X^TX is singular.
- This matrix can be singular whenever the data generating distribution truly has no variance in some direction, or when no variance in **observed** in some direction because there are fewer examples (rows of X) than input features (columns of X).
- In this case, many forms of regularization correspond to inverting $X^TX + \alpha I$ instead. This regularized matrix is guaranteed to be invertible.

Regularization and Under-Constrained Problems

- These linear problems have closed form solutions when the relevant matrix is invertible.
- It is also possible for a problem with no closed form solution to be underdetermined.
- An example is logistic regression applied to a problem where the classes are linearly separable.
- If a weight vector w is able to achieve perfect classification, then $2w$ will also achieve perfect classification and higher likelihood.
- An iterative optimization procedure like SGD will continually increase the magnitude of w and, in theory, will never halt.
- Most forms of regularization are able to guarantee the convergence of iterative methods applied to underdetermined problems.
- For example, weight decay will cause gradient descent to quit increasing the magnitude of the weights when the slope of the likelihood is equal to the weight decay coefficient.

Regularization and Under-Constrained Problems

- These linear problems have closed form solutions when the relevant matrix is invertible.
- It is also possible for a problem with no closed form solution to be underdetermined.
- An example is logistic regression applied to a problem where the classes are linearly separable.
- If a weight vector w is able to achieve perfect classification, then $2w$ will also achieve perfect classification and higher likelihood.
- An iterative optimization procedure like SGD will continually increase the magnitude of w and, in theory, will never halt.
- Most forms of regularization are able to guarantee the convergence of iterative methods applied to underdetermined problems.
- For example, weight decay will cause gradient descent to quit increasing the magnitude of the weights when the slope of the likelihood is equal to the weight decay coefficient.

Regularization and Under-Constrained Problems

- These linear problems have closed form solutions when the relevant matrix is invertible.
- It is also possible for a problem with no closed form solution to be underdetermined.
- An example is logistic regression applied to a problem where the classes are linearly separable.
- If a weight vector $\boldsymbol{w}$ is able to achieve perfect classification, then $2\boldsymbol{w}$ will also achieve perfect classification and higher likelihood.
- An iterative optimization procedure like SGD will continually increase the magnitude of $\boldsymbol{w}$ and, in theory, will never halt.
- Most forms of regularization are able to guarantee the convergence of iterative methods applied to underdetermined problems.
- For example, weight decay will cause gradient descent to quit increasing the magnitude of the weights when the slope of the likelihood is equal to the weight decay coefficient.

Regularization and Under-Constrained Problems

- These linear problems have closed form solutions when the relevant matrix is invertible.
- It is also possible for a problem with no closed form solution to be underdetermined.
- An example is logistic regression applied to a problem where the classes are linearly separable.
- If a weight vector $\boldsymbol{w}$ is able to achieve perfect classification, then $2\boldsymbol{w}$ will also achieve perfect classification and higher likelihood.
- An iterative optimization procedure like SGD will continually increase the magnitude of $\boldsymbol{w}$ and, in theory, will never halt.
- Most forms of regularization are able to guarantee the convergence of iterative methods applied to underdetermined problems.
- For example, weight decay will cause gradient descent to quit increasing the magnitude of the weights when the slope of the likelihood is equal to the weight decay coefficient.

Regularization and Under-Constrained Problems

- These linear problems have closed form solutions when the relevant matrix is invertible.
- It is also possible for a problem with no closed form solution to be underdetermined.
- An example is logistic regression applied to a problem where the classes are linearly separable.
- If a weight vector $\boldsymbol{w}$ is able to achieve perfect classification, then $2\boldsymbol{w}$ will also achieve perfect classification and higher likelihood.
- An iterative optimization procedure like SGD will continually increase the magnitude of $\boldsymbol{w}$ and, in theory, will never halt.
- Most forms of regularization are able to guarantee the convergence of iterative methods applied to underdetermined problems.
- For example, weight decay will cause gradient descent to quit increasing the magnitude of the weights when the slope of the likelihood is equal to the weight decay coefficient.

Regularization and Under-Constrained Problems

- These linear problems have closed form solutions when the relevant matrix is invertible.
- It is also possible for a problem with no closed form solution to be underdetermined.
- An example is logistic regression applied to a problem where the classes are linearly separable.
- If a weight vector $\boldsymbol{w}$ is able to achieve perfect classification, then $2\boldsymbol{w}$ will also achieve perfect classification and higher likelihood.
- An iterative optimization procedure like SGD will continually increase the magnitude of $\boldsymbol{w}$ and, in theory, will never halt.
- Most forms of regularization are able to guarantee the convergence of iterative methods applied to underdetermined problems.
- For example, weight decay will cause gradient descent to quit increasing the magnitude of the weights when the slope of the likelihood is equal to the weight decay coefficient.

Regularization and Under-Constrained Problems

- These linear problems have closed form solutions when the relevant matrix is invertible.
- It is also possible for a problem with no closed form solution to be underdetermined.
- An example is logistic regression applied to a problem where the classes are linearly separable.
- If a weight vector w is able to achieve perfect classification, then $2w$ will also achieve perfect classification and higher likelihood.
- An iterative optimization procedure like SGD will continually increase the magnitude of w and, in theory, will never halt.
- Most forms of regularization are able to guarantee the convergence of iterative methods applied to underdetermined problems.
- For example, weight decay will cause gradient descent to quit increasing the magnitude of the weights when the slope of the likelihood is equal to the weight decay coefficient.

- We can solve underdetermined linear equations using the Moore-Penrose pseudoinverse. Recall that one definition of the pseudoinverse X^+ of a matrix X is

$$X^+ = \lim_{a \to 0} (X^T X + \alpha I)^{-1} X^T$$

- We can now recognize this equation as performing linear regression with weight decay.
- We can interpret the pseudoinverse as stabilizing underdetermined problems using regularization.

- We can solve underdetermined linear equations using the Moore-Penrose pseudoinverse. Recall that one definition of the pseudoinverse X^+ of a matrix X is

$$X^+ = \lim_{a \to 0} (X^T X + \alpha I)^{-1} X^T$$

- We can now recognize this equation as performing linear regression with weight decay.
- We can interpret the pseudoinverse as stabilizing underdetermined problems using regularization.

- We can solve underdetermined linear equations using the Moore-Penrose pseudoinverse. Recall that one definition of the pseudoinverse X^+ of a matrix X is

$$X^+ = \lim_{a \to 0} (X^T X + \alpha I)^{-1} X^T$$

- We can now recognize this equation as performing linear regression with weight decay.
- We can interpret the pseudoinverse as stabilizing underdetermined problems using regularization.

Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data.
- In practice, the amount of data we have is limited.
- Create fake data and add it to the training set.
- This approach is easiest for classification.
- A classifier needs to take a complicated, high dimensional input x and summarize it with a single category identity y.
- This means that the main task facing a classifier is to be invariant to a wide variety of transformations.
- We can generate new (x, y) pairs easily just by transforming the x inputs in our training set.

Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data.
- In practice, the amount of data we have is limited.
- Create fake data and add it to the training set.
- This approach is easiest for classification.
- A classifier needs to take a complicated, high dimensional input x and summarize it with a single category identity y.
- This means that the main task facing a classifier is to be invariant to a wide variety of transformations.
- We can generate new (x, y) pairs easily just by transforming the x inputs in our training set.

Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data.
- In practice, the amount of data we have is limited.
- Create fake data and add it to the training set.
- This approach is easiest for classification.
- A classifier needs to take a complicated, high dimensional input x and summarize it with a single category identity y.
- This means that the main task facing a classifier is to be invariant to a wide variety of transformations.
- We can generate new (x, y) pairs easily just by transforming the x inputs in our training set.

Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data.
- In practice, the amount of data we have is limited.
- Create fake data and add it to the training set.
- This approach is easiest for classification.
- A classifier needs to take a complicated, high dimensional input x and summarize it with a single category identity y.
- This means that the main task facing a classifier is to be invariant to a wide variety of transformations.
- We can generate new (x, y) pairs easily just by transforming the x inputs in our training set.

Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data.
- In practice, the amount of data we have is limited.
- Create fake data and add it to the training set.
- This approach is easiest for classification.
- A classifier needs to take a complicated, high dimensional input x and summarize it with a single category identity y.
- This means that the main task facing a classifier is to be invariant to a wide variety of transformations.
- We can generate new (x, y) pairs easily just by transforming the x inputs in our training set.

Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data.
- In practice, the amount of data we have is limited.
- Create fake data and add it to the training set.
- This approach is easiest for classification.
- A classifier needs to take a complicated, high dimensional input x and summarize it with a single category identity y.
- This means that the main task facing a classifier is to be invariant to a wide variety of transformations.
- We can generate new (x, y) pairs easily just by transforming the x inputs in our training set.

Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data.
- In practice, the amount of data we have is limited.
- Create fake data and add it to the training set.
- This approach is easiest for classification.
- A classifier needs to take a complicated, high dimensional input x and summarize it with a single category identity y.
- This means that the main task facing a classifier is to be invariant to a wide variety of transformations.
- We can generate new (x, y) pairs easily just by transforming the x inputs in our training set.

Dataset Augmentation

- The best way to make a machine learning model generalize better is to train it on more data.
- In practice, the amount of data we have is limited.
- Create fake data and add it to the training set.
- This approach is easiest for classification.
- A classifier needs to take a complicated, high dimensional input $\boldsymbol{x}$ and summarize it with a single category identity y.
- This means that the main task facing a classifier is to be invariant to a wide variety of transformations.
- We can generate new $(\boldsymbol{x}, y)$ pairs easily just by transforming the $\boldsymbol{x}$ inputs in our training set.

Dataset Augmentation

- Dataset augmentation has been a particularly effective technique for a specific classification problem: object recognition.
- Images are high dimensional and include an enormous variety of factors of variation, many of which can be easily simulated.
- One must be careful not to apply transformations that would change the correct class. (e.g. ’6’ and ’9’, ’b’ and ’d’).

Dataset Augmentation

- Dataset augmentation has been a particularly effective technique for a specific classification problem: object recognition.
- Images are high dimensional and include an enormous variety of factors of variation, many of which can be easily simulated.
- One must be careful not to apply transformations that would change the correct class. (e.g. ’6’ and ’9’, ’b’ and ’d’).

Dataset Augmentation

- Dataset augmentation has been a particularly effective technique for a specific classification problem: object recognition.
- Images are high dimensional and include an enormous variety of factors of variation, many of which can be easily simulated.
- One must be careful not to apply transformations that would change the correct class. (e.g. ’6’ and ’9’, ’b’ and ’d’).

Dataset Augmentation

- Dataset augmentation is effective for speech recognition task as well (Jaitly and Hinton [2013]).
- Inject noise in the input to a neural network can also be seen as a form of data augmentation (Sietsma and Dow [1991]).
- For many classification and even some regression tasks, the task should still be possible to solve even if small random noise is added to the input.
- One way to improve the robustness of neural networks is simply to train them with random noise applied to their inputs.
- Input noise injection is part of some unsupervised learning algorithms such as the denoising autoencoder (Vincent et al. [2008]).
- Dropout, a powerful regularization strategy can be seen as a process of constructing new inputs by **multiplying** by noise.

Dataset Augmentation

- Dataset augmentation is effective for speech recognition task as well (Jaitly and Hinton [2013]).
- Inject noise in the input to a neural network can also be seen as a form of data augmentation (Sietsma and Dow [1991]).
- For many classification and even some regression tasks, the task should still be possible to solve even if small random noise is added to the input.
- One way to improve the robustness of neural networks is simply to train them with random noise applied to their inputs.
- Input noise injection is part of some unsupervised learning algorithms such as the denoising autoencoder (Vincent et al. [2008]).
- Dropout, a powerful regularization strategy can be seen as a process of constructing new inputs by **multiplying** by noise.

Dataset Augmentation

- Dataset augmentation is effective for speech recognition task as well (Jaitly and Hinton [2013]).
- Inject noise in the input to a neural network can also be seen as a form of data augmentation (Sietsma and Dow [1991]).
- For many classification and even some regression tasks, the task should still be possible to solve even if small random noise is added to the input.
- One way to improve the robustness of neural networks is simply to train them with random noise applied to their inputs.
- Input noise injection is part of some unsupervised learning algorithms such as the denoising autoencoder (Vincent et al. [2008]).
- Dropout, a powerful regularization strategy can be seen as a process of constructing new inputs by **multiplying** by noise.

Dataset Augmentation

- Dataset augmentation is effective for speech recognition task as well (Jaitly and Hinton [2013]).
- Inject noise in the input to a neural network can also be seen as a form of data augmentation (Sietsma and Dow [1991]).
- For many classification and even some regression tasks, the task should still be possible to solve even if small random noise is added to the input.
- One way to improve the robustness of neural networks is simply to train them with random noise applied to their inputs.
- Input noise injection is part of some unsupervised learning algorithms such as the denoising autoencoder (Vincent et al. [2008]).
- Dropout, a powerful regularization strategy can be seen as a process of constructing new inputs by **multiplying** by noise.

Dataset Augmentation

- Dataset augmentation is effective for speech recognition task as well (Jaitly and Hinton [2013]).
- Inject noise in the input to a neural network can also be seen as a form of data augmentation (Sietsma and Dow [1991]).
- For many classification and even some regression tasks, the task should still be possible to solve even if small random noise is added to the input.
- One way to improve the robustness of neural networks is simply to train them with random noise applied to their inputs.
- Input noise injection is part of some unsupervised learning algorithms such as the denoising autoencoder (Vincent et al. [2008]).
- Dropout, a powerful regularization strategy can be seen as a process of constructing new inputs by **multiplying** by noise.

Dataset Augmentation

- Dataset augmentation is effective for speech recognition task as well (Jaitly and Hinton [2013]).
- Inject noise in the input to a neural network can also be seen as a form of data augmentation (Sietsma and Dow [1991]).
- For many classification and even some regression tasks, the task should still be possible to solve even if small random noise is added to the input.
- One way to improve the robustness of neural networks is simply to train them with random noise applied to their inputs.
- Input noise injection is part of some unsupervised learning algorithms such as the denoising autoencoder (Vincent et al. [2008]).
- Dropout, a powerful regularization strategy can be seen as a process of constructing new inputs by **multiplying** by noise.

- When comparing machine learning benchmark results, it is important to take the effect of dataset augmentation into account.
- Often, hand-designed dataset augmentation schemes can dramatically reduce the generalization error.
- When comparing machine learning algorithm A and machine learning algorithm B, it is necessary to make sure that both algorithms were evaluated using the same hand-designed dataset augmentation schemes.

Dataset Augmentation

- When comparing machine learning benchmark results, it is important to take the effect of dataset augmentation into account.
- Often, hand-designed dataset augmentation schemes can dramatically reduce the generalization error.
- When comparing machine learning algorithm A and machine learning algorithm B, it is necessary to make sure that both algorithms were evaluated using the same hand-designed dataset augmentation schemes.

Dataset Augmentation

- When comparing machine learning benchmark results, it is important to take the effect of dataset augmentation into account.
- Often, hand-designed dataset augmentation schemes can dramatically reduce the generalization error.
- When comparing machine learning algorithm A and machine learning algorithm B, it is necessary to make sure that both algorithms were evaluated using the same hand-designed dataset augmentation schemes.

Noise Robustness

- For some models, the addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights (Bishop [1995b,a]).
- Noise injection can be much more powerful than simply shrinking the parameters, especially when the noise is added to the hidden units.
- Noise applied to the hidden units is such an important topic; the dropout algorithm describe later.
- Another way that noise can be added into the weights.
- This technique has been used primarily in the context of recurrent neural networks (Jim et al. [1996], Graves [2011]).
- This can also be interpreted as equivalent (under some assumptions) to a more traditional form of regularization.

Noise Robustness

- For some models, the addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights (Bishop [1995b,a]).
- Noise injection can be much more powerful than simply shrinking the parameters, especially when the noise is added to the hidden units.
- Noise applied to the hidden units is such an important topic; the dropout algorithm describe later.
- Another way that noise can be added into the weights.
- This technique has been used primarily in the context of recurrent neural networks (Jim et al. [1996], Graves [2011]).
- This can also be interpreted as equivalent (under some assumptions) to a more traditional form of regularization.

Noise Robustness

- For some models, the addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights (Bishop [1995b,a]).
- Noise injection can be much more powerful than simply shrinking the parameters, especially when the noise is added to the hidden units.
- Noise applied to the hidden units is such an important topic; the dropout algorithm describe later.
- Another way that noise can be added into the weights.
- This technique has been used primarily in the context of recurrent neural networks (Jim et al. [1996], Graves [2011]).
- This can also be interpreted as equivalent (under some assumptions) to a more traditional form of regularization.

Noise Robustness

- For some models, the addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights (Bishop [1995b,a]).
- Noise injection can be much more powerful than simply shrinking the parameters, especially when the noise is added to the hidden units.
- Noise applied to the hidden units is such an important topic; the dropout algorithm describe later.
- Another way that noise can be added into the weights.
- This technique has been used primarily in the context of recurrent neural networks (Jim et al. [1996], Graves [2011]).
- This can also be interpreted as equivalent (under some assumptions) to a more traditional form of regularization.

Noise Robustness

- For some models, the addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights (Bishop [1995b,a]).
- Noise injection can be much more powerful than simply shrinking the parameters, especially when the noise is added to the hidden units.
- Noise applied to the hidden units is such an important topic; the dropout algorithm describe later.
- Another way that noise can be added into the weights.
- This technique has been used primarily in the context of recurrent neural networks (Jim et al. [1996], Graves [2011]).
- This can also be interpreted as equivalent (under some assumptions) to a more traditional form of regularization.

Noise Robustness

- For some models, the addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights (Bishop [1995b,a]).
- Noise injection can be much more powerful than simply shrinking the parameters, especially when the noise is added to the hidden units.
- Noise applied to the hidden units is such an important topic; the dropout algorithm describe later.
- Another way that noise can be added into the weights.
- This technique has been used primarily in the context of recurrent neural networks (Jim et al. [1996], Graves [2011]).
- This can also be interpreted as equivalent (under some assumptions) to a more traditional form of regularization.

Noise Robustness

- We study the regression setting, where we wish to train a function $\tilde{y}(\boldsymbol{x})$ that maps a set of features $\boldsymbol{x}$ to a scalar using the least-squares cost function between the model predictions $\tilde{y}(\boldsymbol{x})$ and the true values y:

$$J = \mathbb{E}_{p(x,y)}\left[(\hat{y}(x) - y)^2\right]$$

- The training set with m examples: $(\boldsymbol{x}^{(1)}, y^{(1)}), \dots, (\boldsymbol{x}^{(m)}, y^{(m)})$.
- We now assume that with each input presentation we also include a random perturbation $\epsilon_W \mathcal{N}(\epsilon; \mathbf{0}, \eta I)$ of the network weights.
- We denote the perturbed model as $\hat{y}_{\epsilon_W}(x)$. The objective function thus becomes:

$$\begin{aligned}\tilde{J}_W &= \mathbb{E}_{p(\mathbf{x},y,\epsilon_W)}\left[(\hat{y}_{\epsilon_w}(x) - y)^2\right] \\ &= \mathbb{E}_{p(\mathbf{x},y,\epsilon_W)}\left[\hat{y}^2_{\epsilon_W}(\boldsymbol{x} - 2y\hat{y}_{\epsilon_W} + y^2)\right]\end{aligned}$$

Noise Robustness

- We study the regression setting, where we wish to train a function $\tilde{y}(\boldsymbol{x})$ that maps a set of features $\boldsymbol{x}$ to a scalar using the least-squares cost function between the model predictions $\tilde{y}(\boldsymbol{x})$ and the true values y:

$$J = \mathbb{E}_{p(x,y)}\left[(\hat{y}(x) - y)^2\right]$$

- The training set with m examples: $(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})$.
- We now assume that with each input presentation we also include a random perturbation $\epsilon_W \mathcal{N}(\epsilon; \mathbf{0}, \eta I)$ of the network weights.
- We denote the perturbed model as $\hat{y}_{\epsilon_W}(x)$. The objective function thus becomes:

$$\begin{aligned} \tilde{J}_W &= \mathbb{E}_{p(\mathbf{x},y,\epsilon_W)}\left[(\hat{y}_{\epsilon_w}(x) - y)^2\right] \\ &= \mathbb{E}_{p(\mathbf{x},y,\epsilon_W)}\left[\hat{y}^2_{\epsilon_W}(\mathbf{x} - 2y\hat{y}_{\epsilon_W} + y^2)\right] \end{aligned}$$

Noise Robustness

- We study the regression setting, where we wish to train a function $\tilde{y}(\boldsymbol{x})$ that maps a set of features $\boldsymbol{x}$ to a scalar using the least-squares cost function between the model predictions $\tilde{y}(\boldsymbol{x})$ and the true values y:

$$J = \mathbb{E}_{p(x,y)}\left[(\hat{y}(x) - y)^2\right]$$

- The training set with m examples: $(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})$.
- We now assume that with each input presentation we also include a random perturbation $\epsilon_W\, \mathcal{N}(\boldsymbol{\epsilon}; \mathbf{0}, \eta \boldsymbol{I})$ of the network weights.
- We denote the perturbed model as $\hat{y}_{\epsilon_W}(\boldsymbol{x})$. The objective function thus becomes:

$$\begin{aligned} \tilde{J}_W &= \mathbb{E}_{p(\mathbf{x},y,\epsilon_W)}\left[(\hat{y}_{\epsilon_w}(\boldsymbol{x}) - y)^2\right] \\ &= \mathbb{E}_{p(\mathbf{x},y,\epsilon_W)}\left[\hat{y}^2_{\epsilon_W}(\boldsymbol{x} - 2y\hat{y}_{\epsilon_W} + y^2)\right] \end{aligned}$$

Noise Robustness

- We study the regression setting, where we wish to train a function $\tilde{y}(\boldsymbol{x})$ that maps a set of features $\boldsymbol{x}$ to a scalar using the least-squares cost function between the model predictions $\tilde{y}(\boldsymbol{x})$ and the true values y:

$$J = \mathbb{E}_{p(x,y)} \left[(\hat{y}(x) - y)^2 \right]$$

- The training set with m examples: $(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)})$.
- We now assume that with each input presentation we also include a random perturbation $\epsilon_W \mathcal{N}(\boldsymbol{\epsilon}; \boldsymbol{0}, \eta \boldsymbol{I})$ of the network weights.
- We denote the perturbed model as $\hat{y}_{\epsilon_W}(\boldsymbol{x})$. The objective function thus becomes:

$$\begin{aligned} \tilde{J}_W &= \mathbb{E}_{p(\boldsymbol{x},y,\epsilon_W)} \left[(\hat{y}_{\epsilon_W}(\boldsymbol{x}) - y)^2 \right] \\ &= \mathbb{E}_{p(\boldsymbol{x},y,\epsilon_W)} \left[\hat{y}^2_{\epsilon_W}(\boldsymbol{x} - 2y\hat{y}_{\epsilon_W} + y^2) \right] \end{aligned}$$

Noise Robustness

$$\tilde{J}_W = \mathbb{E}_{p(\mathbf{x},y,\epsilon_W)} \left[\hat{y}^2_{\epsilon_W}(\mathbf{x} - 2y\hat{y}_{\epsilon_W} + y^2)\right]$$

- For small η, the minimization of J with added weight noise (with covariance $\eta\mathbf{I}$) is equivalent to minimization of J with an additional regularization.
- This form of regularization encourages the parameters to go to regions of parameter space where small perturbations of the weights have a relatively small influence on the output.
- In other words, it pushes the model weights, finding points that are not merely minimal, but minimal surrounded by flat regions (Hochreiter et al. [1995]).

Noise Robustness

$$\tilde{J}_W = \mathbb{E}_{p(\boldsymbol{x}, y, \epsilon_W)} \left[\hat{y}_{\epsilon_W}^2 (\boldsymbol{x} - 2y\hat{y}_{\epsilon_W} + y^2) \right]$$

- For small η, the minimization of J with added weight noise (with covariance $\eta \boldsymbol{I}$) is equivalent to minimization of J with an additional regularization.
- This form of regularization encourages the parameters to go to regions of parameter space where small perturbations of the weights have a relatively small influence on the output.
- In other words, it pushes the model weights, finding points that are not merely minimal, but minimal surrounded by flat regions (Hochreiter et al. [1995]).

Noise Robustness

$$\tilde{J}_W = \mathbb{E}_{p(\boldsymbol{x},y,\epsilon_W)}\left[\hat{y}^2_{\epsilon_W}(\boldsymbol{x} - 2y\hat{y}_{\epsilon_W} + y^2)\right]$$

- For small η, the minimization of J with added weight noise (with covariance $\eta \boldsymbol{I}$) is equivalent to minimization of J with an additional regularization.
- This form of regularization encourages the parameters to go to regions of parameter space where small perturbations of the weights have a relatively small influence on the output.
- In other words, it pushes the model weights, finding points that are not merely minimal, but minimal surrounded by flat regions (Hochreiter et al. [1995]).

Noise Robustness
Injecting Noise at the Output Target

- Most datasets have some amount of mistakes in the y labels.
- It can be harmful to maximize $\log p(y|\boldsymbol{x})$ when y is a mistake.
- One way to prevent this is to explicitly model the noise on the labels.
- For example, we can assume that for some small constant ϵ, the training set label y is correct with probability $1 - \epsilon$, and otherwise any of the other possible labels might be correct.

Noise Robustness
Injecting Noise at the Output Target

- Most datasets have some amount of mistakes in the y labels.
- It can be harmful to maximize $\log p(y|x)$ when y is a mistake.
- One way to prevent this is to explicitly model the noise on the labels.
- For example, we can assume that for some small constant ϵ, the training set label y is correct with probability $1 - \epsilon$, and otherwise any of the other possible labels might be correct.

- Most datasets have some amount of mistakes in the y labels.
- It can be harmful to maximize $\log p(y|x)$ when y is a mistake.
- One way to prevent this is to explicitly model the noise on the labels.
- For example, we can assume that for some small constant ϵ, the training set label y is correct with probability $1 - \epsilon$, and otherwise any of the other possible labels might be correct.

Noise Robustness
Injecting Noise at the Output Target

- Most datasets have some amount of mistakes in the y labels.
- It can be harmful to maximize $\log p(y|\boldsymbol{x})$ when y is a mistake.
- One way to prevent this is to explicitly model the noise on the labels.
- For example, we can assume that for some small constant ϵ, the training set label y is correct with probability $1 - \epsilon$, and otherwise any of the other possible labels might be correct.

Semi-Supervised Learning

- In the paradigm of semi-supervised learning, both unlabeled examples from $P(x)$ and labeled examples from $P(x, y)$ are used to estimate $P(y|x)$ or predict y from x.
- In the context of deep learning, semi-supervised learning usually refers to learning a representation $h = f(x)$. The goal is to learn a representation so that **examples from the same class have similar representations**.
- Unsupervised learning can provide useful cues for **how to group examples in representations space**.
- Examples that cluster tightly in the input space should be mapped to similar representations.
- A linear classifier in the new space may achieve better generalization in many cases.

Semi-Supervised Learning

- In the paradigm of semi-supervised learning, both unlabeled examples from $P(\mathbf{x})$ and labeled examples from $P(\mathbf{x}, \mathbf{y})$ are used to estimate $P(\mathbf{y}|\mathbf{x})$ or predict $\mathbf{y}$ from $\mathbf{x}$.
- In the context of deep learning, semi-supervised learning usually refers to learning a representation $h = f(x)$. The goal is to learn a representation so that **examples from the same class have similar representations**.
- Unsupervised learning can provide useful cues for **how to group examples in representations space**.
- Examples that cluster tightly in the input space should be mapped to similar representations.
- A linear classifier in the new space may achieve better generalization in many cases.

Semi-Supervised Learning

- In the paradigm of semi-supervised learning, both unlabeled examples from $P(x)$ and labeled examples from $P(x, y)$ are used to estimate $P(y|x)$ or predict y from x.
- In the context of deep learning, semi-supervised learning usually refers to learning a representation $h = f(x)$. The goal is to learn a representation so that **examples from the same class have similar representations**.
- Unsupervised learning can provide useful cues for **how to group examples in representations space**.
- Examples that cluster tightly in the input space should be mapped to similar representations.
- A linear classifier in the new space may achieve better generalization in many cases.

Semi-Supervised Learning

- In the paradigm of semi-supervised learning, both unlabeled examples from $P(\mathbf{x})$ and labeled examples from $P(\mathbf{x}, \mathbf{y})$ are used to estimate $P(\mathbf{y}|\mathbf{x})$ or predict $\mathbf{y}$ from $\mathbf{x}$.
- In the context of deep learning, semi-supervised learning usually refers to learning a representation $\mathbf{h} = f(\mathbf{x})$. The goal is to learn a representation so that **examples from the same class have similar representations**.
- Unsupervised learning can provide useful cues for **how to group examples in representations space**.
- Examples that cluster tightly in the input space should be mapped to similar representations.
- A linear classifier in the new space may achieve better generalization in many cases.

Semi-Supervised Learning

- In the paradigm of semi-supervised learning, both unlabeled examples from $P(\mathbf{x})$ and labeled examples from $P(\mathbf{x}, \mathbf{y})$ are used to estimate $P(\mathbf{y}|\mathbf{x})$ or predict $\mathbf{y}$ from $\mathbf{x}$.
- In the context of deep learning, semi-supervised learning usually refers to learning a representation $\boldsymbol{h} = f(\boldsymbol{x})$. The goal is to learn a representation so that **examples from the same class have similar representations**.
- Unsupervised learning can provide useful cues for **how to group examples in representations space**.
- Examples that cluster tightly in the input space should be mapped to similar representations.
- A linear classifier in the new space may achieve better generalization in many cases.

Semi-Supervised Learning

- One can construct models in which a generative model of either (x) or $P(x, y)$ shares parameters with a discriminative model of $P(y|x)$.
- The generative criterion then express a particular form of prior belief about the solution to the supervised learning problem, namely that the structure of $P(x)$ is connected to the structure of $P(y|x)$ in a way that is captured by the shared parameterization.
- By controlling how much of the generative criterion is included in the total criterion, one can find a better trade-off than with a purely generative or purely discriminative training criterion.
- Hinton and Salakhutdinov [2008] describe a method for learning the kernel function of a kernel machine used for regression, in which the usage of unlabeled examples for modeling $P(x)$ improves $P(y|x)$ quite significantly.

Semi-Supervised Learning

- One can construct models in which a generative model of either (x) or $P(x, y)$ shares parameters with a discriminative model of $P(y|x)$.
- The generative criterion then express a particular form of prior belief about the solution to the supervised learning problem, namely that the structure of $P(x)$ is connected to the structure of $P(y|x)$ in a way that is captured by the shared parameterization.
- By controlling how much of the generative criterion is included in the total criterion, one can find a better trade-off than with a purely generative or purely discriminative training criterion.
- Hinton and Salakhutdinov [2008] describe a method for learning the kernel function of a kernel machine used for regression, in which the usage of unlabeled examples for modeling $P(x)$ improves $P(y|x)$ quite significantly.

Semi-Supervised Learning

- One can construct models in which a generative model of either (x) or $P(x, y)$ shares parameters with a discriminative model of $P(y|x)$.
- The generative criterion then express a particular form of prior belief about the solution to the supervised learning problem, namely that the structure of $P(x)$ is connected to the structure of $P(y|x)$ in a way that is captured by the shared parameterization.
- By controlling how much of the generative criterion is included in the total criterion, one can find a better trade-off than with a purely generative or purely discriminative training criterion.
- Hinton and Salakhutdinov [2008] describe a method for learning the kernel function of a kernel machine used for regression, in which the usage of unlabeled examples for modeling $P(x)$ improves $P(y|x)$ quite significantly.

Semi-Supervised Learning

- One can construct models in which a generative model of either (x) or $P(x, y)$ shares parameters with a discriminative model of $P(y|x)$.
- The generative criterion then express a particular form of prior belief about the solution to the supervised learning problem, namely that the structure of $P(x)$ is connected to the structure of $P(y|x)$ in a way that is captured by the shared parameterization.
- By controlling how much of the generative criterion is included in the total criterion, one can find a better trade-off than with a purely generative or purely discriminative training criterion.
- Hinton and Salakhutdinov [2008] describe a method for learning the kernel function of a kernel machine used for regression, in which the usage of unlabeled examples for modeling $P(x)$ improves $P(y|x)$ quite significantly.

Multi-Task Learning

- Multi-task learning is a way to improve generalization by pooling the examples arising out of several tasks.
- In the same way that additional training examples put more pressure on the parameters of the model towards values that generalize well, when part of a model is shared across tasks, model often yield better generalization.

Multi-Task Learning

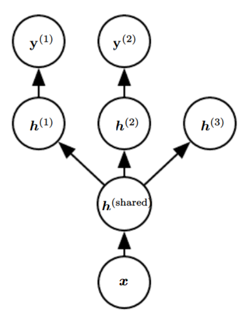
- Multi-task learning is a way to improve generalization by pooling the examples arising out of several tasks.
- In the same way that additional training examples put more pressure on the parameters of the model towards values that generalize well, when part of a model is shared across tasks, model often yield better generalization.

Multi-Task Learning

- Multi-task learning is a way to improve generalization by pooling the examples arising out of several tasks.
- In the same way that additional training examples put more pressure on the parameters of the model towards values that generalize well, when part of a model is shared across tasks, model often yield better generalization.

Multi-Task Learning

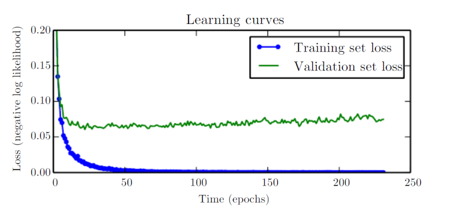
- Here is a very common form of multi-task learning.
- Different supervised tasks (predicting $y^{(i)}$ given x) share the same input x, as well as some intermediate-level representation $h^{(\text{shared})}$ capturing a common pool of factors.
- The model has two kinds of parts:
  1. Task-specific parameters (which only benefit from the examples of their task to achieve good generalization). These are the upper layers.
  2. Generic parameters, shared across all the tasks (which benefit from the pooled data of all the tasks). These are the lower years.



- The factors that explain the variations are shared across two or more tasks.

Early Stopping

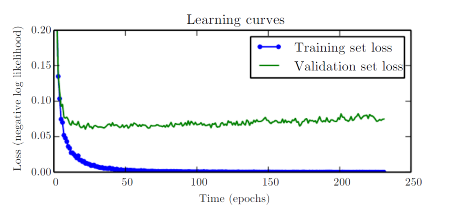
- When training large models with sufficient representational capacity to overfit the task, we often observe that training error decreases steadily over time, but validation set error begins to rise again.



- This behavior occurs very reliably.
- This means we can obtain a model with better validation set error (hopefully better test set error) by returning to the parameter setting at the point in time with the lowest validation set error.

Early Stopping

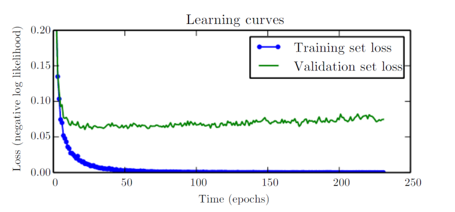
- When training large models with sufficient representational capacity to overfit the task, we often observe that training error decreases steadily over time, but validation set error begins to rise again.



- This behavior occurs very reliably.
- This means we can obtain a model with better validation set error (hopefully better test set error) by returning to the parameter setting at the point in time with the lowest validation set error.

Early Stopping

- When training large models with sufficient representational capacity to overfit the task, we often observe that training error decreases steadily over time, but validation set error begins to rise again.



- This behavior occurs very reliably.
- This means we can obtain a model with better validation set error (hopefully better test set error) by returning to the parameter setting at the point in time with the lowest validation set error.

Early Stopping

- Every time the error on the validation set improves, we store a copy of the model parameters.
- When the training algorithm terminates, we return these parameters, rather the latest parameters.

Early Stopping

- Every time the error on the validation set improves, we store a copy of the model parameters.
- When the training algorithm terminates, we return these parameters, rather the latest parameters.

Early Stopping

Algorithm 1 Early Stopping Algorithm

Let n be the number of steps between evaluations.
Let p be the "patience", the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta_o}$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta_o}$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$; $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$; $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$; $i^* \leftarrow i$; $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Early Stopping

Algorithm 2 Early Stopping Algorithm

Let n be the number of steps between evaluations.
Let p be the "patience", the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$; $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$; $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$; $i^* \leftarrow i$; $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Early Stopping

Algorithm 3 Early Stopping Algorithm

Let n be the number of steps between evaluations.
Let p be the ”patience”, the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$; $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$; $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$; $i^* \leftarrow i$; $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Early Stopping

Algorithm 4 Early Stopping Algorithm

Let n be the number of steps between evaluations.
Let p be the "patience", the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta_o}$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta_o}$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$; $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$; $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$; $i^* \leftarrow i$; $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Early Stopping

Algorithm 5 Early Stopping Algorithm

Let n be the number of steps between evaluations.
Let p be the "patience", the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta_o}$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta_o}$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$; $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$; $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$; $i^* \leftarrow i$; $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Early Stopping

Algorithm 6 Early Stopping Algorithm

Let n be the number of steps between evaluations.
Let p be the "patience", the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta_o}$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta_o}$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$; $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$; $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$; $i^* \leftarrow i$; $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Algorithm 7 Early Stopping Algorithm

Let n be the number of steps between evaluations.
Let p be the "patience", the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta_o}$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta_o}$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$; $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$; $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$; $i^* \leftarrow i$; $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Algorithm 8 Early Stopping Algorithm

Let n be the number of steps between evaluations.
Let p be the "patience", the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta_o}$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta_o}$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$; $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$; $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$; $i^* \leftarrow i$; $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Early Stopping

Algorithm 9 Early Stopping Algorithm

Let n be the number of steps between evaluations.
Let p be the "patience", the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta_o}$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta_o}$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$; $v' \leftarrow$ ValidationSetError($\boldsymbol{\theta}$)
  if $v' < v$ **then**
    $j \leftarrow 0$; $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$; $i^* \leftarrow i$; $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Early Stopping

Algorithm 10 Early Stopping Algorithm

Let n be the number of steps between evaluations.
Let p be the "patience", the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta_o}$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta_o}$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$; $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$; $\boldsymbol{\theta^*} \leftarrow \boldsymbol{\theta}$; $i^* \leftarrow i$; $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Early Stopping

Algorithm 11 Early Stopping Algorithm

Let n be the number of steps between evaluations.
Let p be the "patience", the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$; $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$; $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$; $i^* \leftarrow i$; $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

Algorithm 12 Early Stopping Algorithm

Let n be the number of steps between evaluations.
Let p be the "patience", the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta_o}$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta_o}$; $i \leftarrow 0$; $j \leftarrow 0$; $v \leftarrow \infty$; $i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$; $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$; $\boldsymbol{\theta^*} \leftarrow \boldsymbol{\theta}$; $i^* \leftarrow i$; $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta^*}$, best number of training steps is i^*.

Early Stopping

- One way to think of early stopping is as a very efficient hyperparameter selection algorithm.
- In this view, the number of training steps is just another hyperparameter.
- The only significant cost to choosing this hyperparameter automatically via early stopping is running the validation set evaluation periodically during training.
- An additional cost to early stopping is the need to maintain a copy of the best parameters. This cost is generally negligible. (GPU->CPU/MEMORY->HDD).

Early Stopping

- One way to think of early stopping is as a very efficient hyperparameter selection algorithm.
- In this view, the number of training steps is just another hyperparameter.
- The only significant cost to choosing this hyperparameter automatically via early stopping is running the validation set evaluation periodically during training.
- An additional cost to early stopping is the need to maintain a copy of the best parameters. This cost is generally negligible. (GPU->CPU/MEMORY->HDD).

Early Stopping

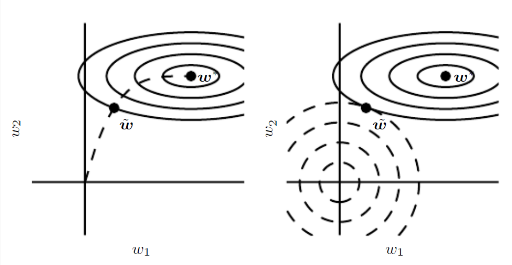
- One way to think of early stopping is as a very efficient hyperparameter selection algorithm.
- In this view, the number of training steps is just another hyperparameter.
- The only significant cost to choosing this hyperparameter automatically via early stopping is running the validation set evaluation periodically during training.
- An additional cost to early stopping is the need to maintain a copy of the best parameters. This cost is generally negligible. (GPU->CPU/MEMORY->HDD).

Early Stopping

- One way to think of early stopping is as a very efficient hyperparameter selection algorithm.
- In this view, the number of training steps is just another hyperparameter.
- The only significant cost to choosing this hyperparameter automatically via early stopping is running the validation set evaluation periodically during training.
- An additional cost to early stopping is the need to maintain a copy of the best parameters. This cost is generally negligible. (GPU->CPU/MEMORY->HDD).

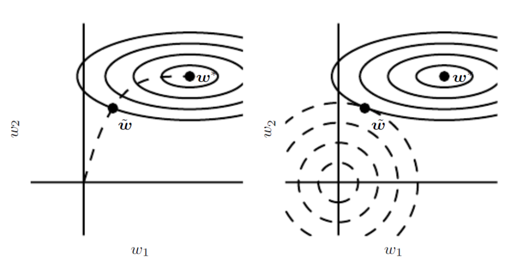
Early Stopping

- How early stopping acts as a regularizer?
- Bishop [1995b] , Sjöberg and Ljung [1995] argued that early stopping has the effect of restricting the optimization procedure to a relatively small volume of parameter space in the neighborhood of the initial parameter value $\boldsymbol{\theta_o}$.



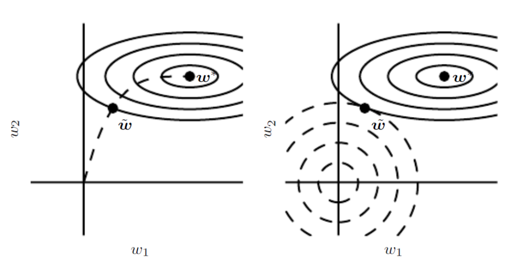
Early Stopping

- How early stopping acts as a regularizer?
- Bishop [1995b] , Sjöberg and Ljung [1995] argued that early stopping has the effect of restricting the optimization procedure to a relatively small volume of parameter space in the neighborhood of the initial parameter value $\boldsymbol{\theta_o}$.



Early Stopping

- How early stopping acts as a regularizer?
- Bishop [1995b] , Sjöberg and Ljung [1995] argued that early stopping has the effect of restricting the optimization procedure to a relatively small volume of parameter space in the neighborhood of the initial parameter value $\boldsymbol{\theta_o}$.



Early Stopping

- In order to compare with classical L^2 regularization, we examine a simple setting where the only parameters are linear weights ($\boldsymbol{\theta} = \boldsymbol{w}$).
- We can model the cost function J with a quadratic approximation in the neighborhood of the empirically optimal value of the weights $\boldsymbol{w}^*$:

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

  where $\boldsymbol{H}$ is Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$.
- Given the assumption that $\boldsymbol{w}^*$ is a minimum of $J(\boldsymbol{w})$, we know that $\boldsymbol{H}$ is positive semidefinite.

Early Stopping

- In order to compare with classical L^2 regularization, we examine a simple setting where the only parameters are linear weights $(\boldsymbol{\theta} = \boldsymbol{w})$.
- We can model the cost function J with a quadratic approximation in the neighborhood of the empirically optimal value of the weights $\boldsymbol{w}^*$:

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

  where $\boldsymbol{H}$ is Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$.
- Given the assumption that $\boldsymbol{w}^*$ is a minimum of $J(\boldsymbol{w})$, we know that $\boldsymbol{H}$ is positive semidefinite.

Early Stopping

- In order to compare with classical L^2 regularization, we examine a simple setting where the only parameters are linear weights ($\boldsymbol{\theta} = \boldsymbol{w}$).
- We can model the cost function J with a quadratic approximation in the neighborhood of the empirically optimal value of the weights $\boldsymbol{w}^*$:

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^T \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*)$$

  where $\boldsymbol{H}$ is Hessian matrix of J with respect to $\boldsymbol{w}$ evaluated at $\boldsymbol{w}^*$.
- Given the assumption that $\boldsymbol{w}^*$ is a minimum of $J(\boldsymbol{w})$, we know that $\boldsymbol{H}$ is positive semidefinite.

Early Stopping

- Under a local Taylor series approximation, the gradient:

$$\nabla_w \hat{J}(w) = H(w - w^*)$$

- We are going to study the trajectory followed by the parameter vector during training.
- For simplicity, let us set the initial parameter vector to the origin, that is $w^{(0)} = \mathbf{0}$.
- Let us suppose that we update the parameters via gradient descent:

$$\begin{aligned} w^{(\tau)} &= w^{(\tau-1)} - \epsilon \nabla_w J(w^{(\tau-1)}) \\ &= w^{(\tau-1)} - \epsilon H(w^{(\tau-1)} - w^*) \\ w^{(\tau)} - w^* &= (I - \epsilon H)(w^{(\tau-1)} - w^*) \end{aligned}$$

Early Stopping

- Under a local Taylor series approximation, the gradient:

$$\nabla_w \hat{J}(w) = H(w - w^*)$$

- We are going to study the trajectory followed by the parameter vector during training.
- For simplicity, let us set the initial parameter vector to the origin, that is $w^{(0)} = \mathbf{0}$.
- Let us suppose that we update the parameters via gradient descent:

$$\begin{aligned} w^{(\tau)} &= w^{(\tau-1)} - \epsilon \nabla_w J(w^{(\tau-1)}) \\ &= w^{(\tau-1)} - \epsilon H(w^{(\tau-1)} - w^*) \\ w^{(\tau)} - w^* &= (I - \epsilon H)(w^{(\tau-1)} - w^*) \end{aligned}$$

Early Stopping

- Under a local Taylor series approximation, the gradient:

$$\nabla_w \hat{J}(w) = H(w - w^*)$$

- We are going to study the trajectory followed by the parameter vector during training.
- For simplicity, let us set the initial parameter vector to the origin, that is $w^{(0)} = \mathbf{0}$.
- Let us suppose that we update the parameters via gradient descent:

$$\begin{aligned} w^{(\tau)} &= w^{(\tau-1)} - \epsilon \nabla_w J(w^{(\tau-1)}) \\ &= w^{(\tau-1)} - \epsilon H(w^{(\tau-1)} - w^*) \\ w^{(\tau)} - w^* &= (I - \epsilon H)(w^{(\tau-1)} - w^*) \end{aligned}$$

Early Stopping

- Under a local Taylor series approximation, the gradient:

$$\nabla_w \hat{J}(w) = H(w - w^*)$$

- We are going to study the trajectory followed by the parameter vector during training.
- For simplicity, let us set the initial parameter vector to the origin, that is $w^{(0)} = \mathbf{0}$.
- Let us suppose that we update the parameters via gradient descent:

$$\begin{aligned} w^{(\tau)} &= w^{(\tau-1)} - \epsilon \nabla_w J(w^{(\tau-1)}) \\ &= w^{(\tau-1)} - \epsilon H(w^{(\tau-1)} - w^*) \\ w^{(\tau)} - w^* &= (I - \epsilon H)(w^{(\tau-1)} - w^*) \end{aligned}$$

Early Stopping

- Let us now rewrite this expression in the space of the eigenvectors of H, exploiting the eigendecomposition of $H: H = Q\mathbf{\Lambda}Q^T$, where $\mathbf{\Lambda}$ is a diagonal matrix and Q is an orthonormal basis of eigenvectors.

$$\begin{aligned} w^{(\tau)} - w^* &= (I - \epsilon Q\mathbf{\Lambda}Q^T)(w^{(\tau-1)} - w^*) \\ Q^T(w^{(\tau)} - w^*) &= (I - \epsilon\mathbf{\Lambda})Q^T(w^{(\tau-1)} - w^*) \end{aligned}$$

- Assuming that $w^{(0)} = 0$ and that ϵ is chosen to be small enough to guarantee $|1 - \epsilon\lambda_i| < 1$, the parameter trajectory during training training after τ parameter updates is as follows:

$$Q^T w^{(\tau)} = [I - (I - \epsilon\Lambda)^\tau]\, Q^T w^*$$

Early Stopping

- Let us now rewrite this expression in the space of the eigenvectors of H, exploiting the eigendecomposition of $H : H = Q\boldsymbol{\Lambda}Q^T$, where $\boldsymbol{\Lambda}$ is a diagonal matrix and Q is an orthonormal basis of eigenvectors.

$$\begin{aligned} w^{(\tau)} - w^* &= (I - \epsilon Q\boldsymbol{\Lambda}Q^T)(w^{(\tau-1)} - w^*) \\ Q^T(w^{(\tau)} - w^*) &= (I - \epsilon\boldsymbol{\Lambda})Q^T(w^{(\tau-1)} - w^*) \end{aligned}$$

- Assuming that $w^{(0)} = 0$ and that ϵ is chosen to be small enough to guarantee $|1 - \epsilon\lambda_i| < 1$, the parameter trajectory during training training after τ parameter updates is as follows:

$$Q^T w^{(\tau)} = [I - (I - \epsilon\Lambda)^\tau]\, Q^T w^*$$

Early Stopping

- In L^2 regularization:

$$\begin{aligned} \tilde{w} &= Q(\boldsymbol{\Lambda} + \boldsymbol{\alpha} I)^{-1} \boldsymbol{\Lambda} Q^T w^* && (1) \\ Q^T \tilde{w} &= (\Lambda + \alpha I)^{-1} \Lambda Q^T w^* && (2) \\ Q^T \tilde{w} &= \left[I - (\boldsymbol{\Lambda} + \alpha I)^{-1} \alpha \right] Q^T w^* && (3) \end{aligned}$$

- Compare with $Q^T w^{(\tau)} = [I - (I - \epsilon\Lambda)^\tau] Q^T w^*$, we can find:

$$(I - \epsilon\Lambda)^\tau = (\Lambda + \alpha I)^{-1} \alpha$$

- Then L^2 regularization and early stopping is equivalent.
- Going even further, by taking logarithms and using the series expansion for $\log(1 + x)$, if all λ_i are small then:

$$\tau \approx \frac{1}{\epsilon\alpha} \quad ; \quad \alpha \approx \frac{1}{\tau\epsilon} \tag{4}$$

- That is, under these assumptions, the number of training iterations τ plays a role inversely proportional to the L^2 regularization parameter, and the inverse of $\tau\epsilon$ plays the role of the weight decay coefficient.

Early Stopping

- In L^2 regularization:

$$\begin{aligned} \tilde{w} &= Q(\boldsymbol{\Lambda} + \boldsymbol{\alpha} I)^{-1}\boldsymbol{\Lambda} Q^T w^* && (1) \\ Q^T\tilde{w} &= (\Lambda + \alpha I)^{-1}\Lambda Q^T w^* && (2) \\ Q^T\tilde{w} &= \left[I - (\boldsymbol{\Lambda} + \alpha I)^{-1}\alpha\right] Q^T w^* && (3) \end{aligned}$$

- Compare with $Q^T w^{(\tau)} = [I - (I - \epsilon\Lambda)^\tau] \, Q^T w^*$, we can find:

$$(I - \epsilon\Lambda)^\tau = (\Lambda + \alpha I)^{-1}\alpha$$

- Then L^2 regularization and early stopping is equivalent.
- Going even further, by taking logarithms and using the series expansion for $\log(1 + x)$, if all λ_i are small then:

$$\tau \approx \frac{1}{\epsilon\alpha} \quad ; \quad \alpha \approx \frac{1}{\tau\epsilon} \qquad (4)$$

- That is, under these assumptions, the number of training iterations τ plays a role inversely proportional to the L^2 regularization parameter, and the inverse of $\tau\epsilon$ plays the role of the weight decay coefficient.

Early Stopping

- In L^2 regularization:

$$\begin{aligned} \tilde{w} &= Q(\boldsymbol{\Lambda} + \boldsymbol{\alpha} I)^{-1} \boldsymbol{\Lambda} Q^T w^* & (1) \\ Q^T \tilde{w} &= (\Lambda + \alpha I)^{-1} \Lambda Q^T w^* & (2) \\ Q^T \tilde{w} &= \left[I - (\boldsymbol{\Lambda} + \alpha I)^{-1} \alpha \right] Q^T w^* & (3) \end{aligned}$$

- Compare with $Q^T w^{(\tau)} = [I - (I - \epsilon\Lambda)^\tau] \, Q^T w^*$, we can find:

$$(I - \epsilon\Lambda)^\tau = (\Lambda + \alpha I)^{-1} \alpha$$

- Then L^2 regularization and early stopping is equivalent.
- Going even further, by taking logarithms and using the series expansion for $\log(1 + x)$, if all λ_i are small then:

$$\tau \approx \frac{1}{\epsilon\alpha} \quad ; \quad \alpha \approx \frac{1}{\tau\epsilon} \qquad (4)$$

- That is, under these assumptions, the number of training iterations τ plays a role inversely proportional to the L^2 regularization parameter, and the inverse of $\tau\epsilon$ plays the role of the weight decay coefficient.

Early Stopping

- In L^2 regularization:

$$\begin{aligned}\tilde{w} &= Q(\boldsymbol{\Lambda} + \boldsymbol{\alpha} I)^{-1}\boldsymbol{\Lambda} Q^T w^* &(1)\\ Q^T\tilde{w} &= (\Lambda + \alpha I)^{-1}\Lambda Q^T w^* &(2)\\ Q^T\tilde{w} &= \left[I - (\boldsymbol{\Lambda} + \alpha I)^{-1}\alpha\right] Q^T w^* &(3)\end{aligned}$$

- Compare with $Q^T w^{(\tau)} = [I - (I - \epsilon\Lambda)^\tau]\, Q^T w^*$, we can find:

$$(I - \epsilon\Lambda)^\tau = (\Lambda + \alpha I)^{-1}\alpha$$

- Then L^2 regularization and early stopping is equivalent.
- Going even further, by taking logarithms and using the series expansion for $\log(1 + x)$, if all λ_i are small then:

$$\tau \approx \frac{1}{\epsilon\alpha} \quad ; \quad \alpha \approx \frac{1}{\tau\epsilon} \qquad (4)$$

- That is, under these assumptions, the number of training iterations τ plays a role inversely proportional to the L^2 regularization parameter, and the inverse of $\tau\epsilon$ plays the role of the weight decay coefficient.

Early Stopping

- In L^2 regularization:

$$\begin{aligned}
\tilde{w} &= Q(\boldsymbol{\Lambda} + \boldsymbol{\alpha} I)^{-1}\boldsymbol{\Lambda} Q^T w^* && (1)\\
Q^T\tilde{w} &= (\Lambda + \alpha I)^{-1}\Lambda Q^T w^* && (2)\\
Q^T\tilde{w} &= \left[I - (\boldsymbol{\Lambda} + \alpha I)^{-1}\alpha\right] Q^T w^* && (3)
\end{aligned}$$

- Compare with $Q^T w^{(\tau)} = [I - (I - \epsilon\Lambda)^\tau]\, Q^T w^*$, we can find:

$$(I - \epsilon\Lambda)^\tau = (\Lambda + \alpha I)^{-1}\alpha$$

- Then L^2 regularization and early stopping is equivalent.
- Going even further, by taking logarithms and using the series expansion for $\log(1 + x)$, if all λ_i are small then:

$$\tau \approx \frac{1}{\epsilon\alpha} \quad ; \quad \alpha \approx \frac{1}{\tau\epsilon} \qquad (4)$$

- That is, under these assumptions, the number of training iterations τ plays a role inversely proportional to the L^2 regularization parameter, and the inverse of $\tau\epsilon$ plays the role of the weight decay coefficient.

Parameter Tying and Parameter Sharing

- Thus far, we have discussed adding constraints or penalties to the parameters.
- However, sometimes we may need other ways to express our prior knowledge about suitable values of the model parameters.
- Sometimes we might not know precisely what values that parameters should take but we know, from knowledge of the domain and model architecture, that there should be some dependencies between the model parameters.
- A common type of dependency that we often want to express is that certain parameters should be close to one another.

Parameter Tying and Parameter Sharing

- Thus far, we have discussed adding constraints or penalties to the parameters.
- However, sometimes we may need other ways to express our prior knowledge about suitable values of the model parameters.
- Sometimes we might not know precisely what values that parameters should take but we know, from knowledge of the domain and model architecture, that there should be some dependencies between the model parameters.
- A common type of dependency that we often want to express is that certain parameters should be close to one another.

Parameter Tying and Parameter Sharing

- Thus far, we have discussed adding constraints or penalties to the parameters.
- However, sometimes we may need other ways to express our prior knowledge about suitable values of the model parameters.
- Sometimes we might not know precisely what values that parameters should take but we know, from knowledge of the domain and model architecture, that there should be some dependencies between the model parameters.
- A common type of dependency that we often want to express is that certain parameters should be close to one another.

Parameter Tying and Parameter Sharing

- Thus far, we have discussed adding constraints or penalties to the parameters.
- However, sometimes we may need other ways to express our prior knowledge about suitable values of the model parameters.
- Sometimes we might not know precisely what values that parameters should take but we know, from knowledge of the domain and model architecture, that there should be some dependencies between the model parameters.
- A common type of dependency that we often want to express is that certain parameters should be close to one another.

Parameter Tying and Parameter Sharing

- Thus far, we have discussed adding constraints or penalties to the parameters.
- However, sometimes we may need other ways to express our prior knowledge about suitable values of the model parameters.
- Sometimes we might not know precisely what values that parameters should take but we know, from knowledge of the domain and model architecture, that there should be some dependencies between the model parameters.
- A common type of dependency that we often want to express is that certain parameters should be close to one another.

Parameter Tying and Parameter Sharing

Consider the following scenario:

- We have two models performing the same classification task.
- But with somewhat different input distributions.
- Formally, we have model A with parameters $\boldsymbol{w}^{(A)}$ and model B with parameters $\boldsymbol{w}^{(B)}$.
- The two models map the input to different, but related outputs: $\hat{y}^{(A)} = f(\boldsymbol{w}^{(A)}, \boldsymbol{x})$ and $\hat{y}^{(B)} = g(\boldsymbol{w}^{(B)}, \boldsymbol{x})$.

Parameter Tying and Parameter Sharing

Consider the following scenario:

- We have two models performing the same classification task.
- But with somewhat different input distributions.
- Formally, we have model A with parameters $\boldsymbol{w}^{(A)}$ and model B with parameters $\boldsymbol{w}^{(B)}$.
- The two models map the input to different, but related outputs: $\hat{y}^{(A)} = f(\boldsymbol{w}^{(A)}, \boldsymbol{x})$ and $\hat{y}^{(B)} = g(\boldsymbol{w}^{(B)}, \boldsymbol{x})$.

Parameter Tying and Parameter Sharing

Consider the following scenario:

- We have two models performing the same classification task.
- But with somewhat different input distributions.
- Formally, we have model A with parameters $\boldsymbol{w}^{(A)}$ and model B with parameters $\boldsymbol{w}^{(B)}$.
- The two models map the input to different, but related outputs: $\hat{y}^{(A)} = f(\boldsymbol{w}^{(A)}, \boldsymbol{x})$ and $\hat{y}^{(B)} = g(\boldsymbol{w}^{(B)}, \boldsymbol{x})$.

Parameter Tying and Parameter Sharing

Consider the following scenario:

- We have two models performing the same classification task.
- But with somewhat different input distributions.
- Formally, we have model A with parameters $\boldsymbol{w}^{(A)}$ and model B with parameters $\boldsymbol{w}^{(B)}$.
- The two models map the input to different, but related outputs: $\hat{y}^{(A)} = f(\boldsymbol{w}^{(A)}, \boldsymbol{x})$ and $\hat{y}^{(B)} = g(\boldsymbol{w}^{(B)}, \boldsymbol{x})$.

Consider the following scenario:

- We have two models performing the same classification task.
- But with somewhat different input distributions.
- Formally, we have model A with parameters $\boldsymbol{w}^{(A)}$ and model B with parameters $\boldsymbol{w}^{(B)}$.
- The two models map the input to different, but related outputs: $\hat{y}^{(A)} = f(\boldsymbol{w}^{(A)}, \boldsymbol{x})$ and $\hat{y}^{(B)} = g(\boldsymbol{w}^{(B)}, \boldsymbol{x})$.

Parameter Tying and Parameter Sharing

- Let us imagine that the tasks are similar enough (perhaps with similar input and output distributions) that we believe the model parameters should be close to each other: $\forall i, w_i^{(A)}$ should be close to $w_i^{(B)}$. We can leverage this information through regularization.
- Specifically, we can use a parameter norm penalty of the form: $\Omega(\boldsymbol{w}^{(A)}, \boldsymbol{w}^{(B)}) = \|\boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)}\|_2^2$. Here we used an L^2 penalty, but other choices are also possible.

Parameter Tying and Parameter Sharing

- Let us imagine that the tasks are similar enough (perhaps with similar input and output distributions) that we believe the model parameters should be close to each other: $\forall i, w_i^{(A)}$ should be close to $w_i^{(B)}$. We can leverage this information through regularization.
- Specifically, we can use a parameter norm penalty of the form: $\Omega(\boldsymbol{w}^{(A)}, \boldsymbol{w}^{(B)}) = \|\boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)}\|_2^2$. Here we used an L^2 penalty, but other choices are also possible.

Parameter Tying and Parameter Sharing

- This kind of approach was proposed by Lasserre et al. [2006], who regularized the parameters of one model, trained as a classifier in a supervised paradigm, to be close to the parameters of another model, trained in an unsupervised paradigm (to capture the distribution of the observed input data).
- The architectures were constructed such that many of the parameters in the classifier model could paired to corresponding parameters in the unsupervised model.

Parameter Tying and Parameter Sharing

- This kind of approach was proposed by Lasserre et al. [2006], who regularized the parameters of one model, trained as a classifier in a supervised paradigm, to be close to the parameters of another model, trained in an unsupervised paradigm (to capture the distribution of the observed input data).
- The architectures were constructed such that many of the parameters in the classifier model could paired to corresponding parameters in the unsupervised model.

Parameter Tying and Parameter Sharing

- While a parameter norm penalty is one way to regularize parameters to be close to one another, the more popular way is to use constraints: **to force sets of parameters to be equal**.
- This method of regularization is often referred to as *parameter sharing*, where we interpret the various models or model components as sharing a unique set of parameters.
- A significant advantage of parameter sharing over regularizing the parameters to be close (via a norm penalty) is that only a subset of the parameters need to be stored in memory.
- In certain models- such as the Convolutional Neural Network – this can lead to significant reduction in the memory footprint of the model.

Parameter Tying and Parameter Sharing

- While a parameter norm penalty is one way to regularize parameters to be close to one another, the more popular way is to use constraints: **to force sets of parameters to be equal**.
- This method of regularization is often referred to as *parameter sharing*, where we interpret the various models or model components as sharing a unique set of parameters.
- A significant advantage of parameter sharing over regularizing the parameters to be close (via a norm penalty) is that only a subset of the parameters need to be stored in memory.
- In certain models- such as the Convolutional Neural Network – this can lead to significant reduction in the memory footprint of the model.

Parameter Tying and Parameter Sharing

- While a parameter norm penalty is one way to regularize parameters to be close to one another, the more popular way is to use constraints: **to force sets of parameters to be equal**.
- This method of regularization is often referred to as *parameter sharing*, where we interpret the various models or model components as sharing a unique set of parameters.
- A significant advantage of parameter sharing over regularizing the parameters to be close (via a norm penalty) is that only a subset of the parameters need to be stored in memory.
- In certain models- such as the Convolutional Neural Network – this can lead to significant reduction in the memory footprint of the model.

Parameter Tying and Parameter Sharing

- While a parameter norm penalty is one way to regularize parameters to be close to one another, the more popular way is to use constraints: **to force sets of parameters to be equal**.
- This method of regularization is often referred to as *parameter sharing*, where we interpret the various models or model components as sharing a unique set of parameters.
- A significant advantage of parameter sharing over regularizing the parameters to be close (via a norm penalty) is that only a subset of the parameters need to be stored in memory.
- In certain models- such as the Convolutional Neural Network – this can lead to significant reduction in the memory footprint of the model.

Parameter Tying and Parameter Sharing
Convolutional Neural Networks

- By far the most popular and extensive use of parameter sharing occurs in *convolutional neural networks* (CNNs) applied to computer vision.
- Natural images have many statistical properties that are invariant to translation.
- CNNs take this property into account by sharing parameters across multiple image locations.
- The same feature (a hidden unit with the same weights) is computed over different locations in the input.
- This means that we can find a object with the same object detector whether the object appears at column i or column $i + 1$ in the image.
- Parameter sharing has allowed CNNs to dramatically lower the number of unique model parameters and to significantly increase network sizes without requiring a corresponding increase in training data.

Parameter Tying and Parameter Sharing
Convolutional Neural Networks

- By far the most popular and extensive use of parameter sharing occurs in *convolutional neural networks* (CNNs) applied to computer vision.
- Natural images have many statistical properties that are invariant to translation.
- CNNs take this property into account by sharing parameters across multiple image locations.
- The same feature (a hidden unit with the same weights) is computed over different locations in the input.
- This means that we can find a object with the same object detector whether the object appears at column i or column $i + 1$ in the image.
- Parameter sharing has allowed CNNs to dramatically lower the number of unique model parameters and to significantly increase network sizes without requiring a corresponding increase in training data.

Parameter Tying and Parameter Sharing
Convolutional Neural Networks

- By far the most popular and extensive use of parameter sharing occurs in *convolutional neural networks* (CNNs) applied to computer vision.
- Natural images have many statistical properties that are invariant to translation.
- CNNs take this property into account by sharing parameters across multiple image locations.
- The same feature (a hidden unit with the same weights) is computed over different locations in the input.
- This means that we can find a object with the same object detector whether the object appears at column i or column $i+1$ in the image.
- Parameter sharing has allowed CNNs to dramatically lower the number of unique model parameters and to significantly increase network sizes without requiring a corresponding increase in training data.

Parameter Tying and Parameter Sharing
Convolutional Neural Networks

- By far the most popular and extensive use of parameter sharing occurs in *convolutional neural networks* (CNNs) applied to computer vision.
- Natural images have many statistical properties that are invariant to translation.
- CNNs take this property into account by sharing parameters across multiple image locations.
- The same feature (a hidden unit with the same weights) is computed over different locations in the input.
- This means that we can find a object with the same object detector whether the object appears at column i or column $i+1$ in the image.
- Parameter sharing has allowed CNNs to dramatically lower the number of unique model parameters and to significantly increase network sizes without requiring a corresponding increase in training data.

Parameter Tying and Parameter Sharing
Convolutional Neural Networks

- By far the most popular and extensive use of parameter sharing occurs in *convolutional neural networks* (CNNs) applied to computer vision.
- Natural images have many statistical properties that are invariant to translation.
- CNNs take this property into account by sharing parameters across multiple image locations.
- The same feature (a hidden unit with the same weights) is computed over different locations in the input.
- This means that we can find a object with the same object detector whether the object appears at column i or column $i+1$ in the image.
- Parameter sharing has allowed CNNs to dramatically lower the number of unique model parameters and to significantly increase network sizes without requiring a corresponding increase in training data.

Parameter Tying and Parameter Sharing
Convolutional Neural Networks

- By far the most popular and extensive use of parameter sharing occurs in *convolutional neural networks* (CNNs) applied to computer vision.
- Natural images have many statistical properties that are invariant to translation.
- CNNs take this property into account by sharing parameters across multiple image locations.
- The same feature (a hidden unit with the same weights) is computed over different locations in the input.
- This means that we can find a object with the same object detector whether the object appears at column i or column $i + 1$ in the image.
- Parameter sharing has allowed CNNs to dramatically lower the number of unique model parameters and to significantly increase network sizes without requiring a corresponding increase in training data.

Parameter Tying and Parameter Sharing
Convolutional Neural Networks

- By far the most popular and extensive use of parameter sharing occurs in *convolutional neural networks* (CNNs) applied to computer vision.
- Natural images have many statistical properties that are invariant to translation.
- CNNs take this property into account by sharing parameters across multiple image locations.
- The same feature (a hidden unit with the same weights) is computed over different locations in the input.
- This means that we can find a object with the same object detector whether the object appears at column i or column $i+1$ in the image.
- Parameter sharing has allowed CNNs to dramatically lower the number of unique model parameters and to significantly increase network sizes without requiring a corresponding increase in training data.

Sparse Representation

- Weight decay acts by placing a penalty directly on the model parameters.
- Another strategy is to place a penalty on the activations of the units in a neural network, encouraging their activations to be sparse.
- This indirectly imposes a complicated penalty on the model parameters.

Sparse Representation

- Weight decay acts by placing a penalty directly on the model parameters.
- Another strategy is to place a penalty on the activations of the units in a neural network, encouraging their activations to be sparse.
- This indirectly imposes a complicated penalty on the model parameters.

Sparse Representation

- Weight decay acts by placing a penalty directly on the model parameters.
- Another strategy is to place a penalty on the activations of the units in a neural network, encouraging their activations to be sparse.
- This indirectly imposes a complicated penalty on the model parameters.

Sparse Representation

- Weight decay acts by placing a penalty directly on the model parameters.
- Another strategy is to place a penalty on the activations of the units in a neural network, encouraging their activations to be sparse.
- This indirectly imposes a complicated penalty on the model parameters.

Sparse Representation

- We have already discussed how L^1 penalization induces a sparse parametrization – meaning that many of the parameters become zero (or close to zero).
- Representational sparsity, on the other hand, describes a representation where many of the elements of the representation are zero (or close to zero).

Sparse Representation

- We have already discussed how L^1 penalization induces a sparse parametrization – meaning that many of the parameters become zero (or close to zero).
- Representational sparsity, on the other hand, describes a representation where many of the elements of the representation are zero (or close to zero).

Sparse Representation

- A simplified view of this distinction can be illustrated in the context of linear regression:

$$\underbrace{\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix}}_{\boldsymbol{y} \in \mathbb{R}^m} = \underbrace{\begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix}}_{\mathbf{A} \in \mathbb{R}^{m \times n}} \underbrace{\begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}}_{\boldsymbol{x} \in \mathbb{R}^n}$$

$$\underbrace{\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix}}_{\boldsymbol{y} \in \mathbb{R}^m} = \underbrace{\begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix}}_{\mathbf{B} \in \mathbb{R}^{m \times n}} \underbrace{\begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}}_{\boldsymbol{h} \in \mathbb{R}^n}$$

Sparse Representation

- A simplified view of this distinction can be illustrated in the context of linear regression:

$$\underbrace{\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix}}_{\boldsymbol{y} \in \mathbb{R}^m} = \underbrace{\begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix}}_{\mathbf{A} \in \mathbb{R}^{m \times n}} \underbrace{\begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}}_{\boldsymbol{x} \in \mathbb{R}^n}$$

$$\underbrace{\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix}}_{\boldsymbol{y} \in \mathbb{R}^m} = \underbrace{\begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix}}_{\mathbf{B} \in \mathbb{R}^{m \times n}} \underbrace{\begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}}_{\boldsymbol{h} \in \mathbb{R}^n}$$

Sparse Representation

- Representational regularization is accomplished by the same sorts of mechanisms that we have used in parameter regularization.
- Norm penalty regularization of representation is performed by adding to the loss function J a norm penalty on the **representation**. This penalty is denoted $\Omega(\boldsymbol{h})$. As before, we denote the regularized loss function by $\tilde{J}$:

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{h})$$

- Just as an L^1 penalty on the parameters induces parameter sparsity, an L^1 penalty on the elements of the representation induces representational sparsity:

$$\Omega(\boldsymbol{h}) = \|\boldsymbol{h}\|_1 = \sum_i |h_i|$$

Sparse Representation

- Representational regularization is accomplished by the same sorts of mechanisms that we have used in parameter regularization.
- Norm penalty regularization of representation is performed by adding to the loss function J a norm penalty on the **representation**. This penalty is denoted $\Omega(\boldsymbol{h})$. As before, we denote the regularized loss function by $\tilde{J}$:

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{h})$$

- Just as an L^1 penalty on the parameters induces parameter sparsity, an L^1 penalty on the elements of the representation induces representational sparsity:

$$\Omega(\boldsymbol{h}) = \|\boldsymbol{h}\|_1 = \sum_i |h_i|$$

Sparse Representation

- Representational regularization is accomplished by the same sorts of mechanisms that we have used in parameter regularization.
- Norm penalty regularization of representation is performed by adding to the loss function J a norm penalty on the **representation**. This penalty is denoted $\Omega(\boldsymbol{h})$. As before, we denote the regularized loss function by $\tilde{J}$:

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{h})$$

- Just as an L^1 penalty on the parameters induces parameter sparsity, an L^1 penalty on the elements of the representation induces representational sparsity:

$$\Omega(\boldsymbol{h}) = \|\boldsymbol{h}\|_1 = \sum_i |h_i|$$

Sparse Representation

- Of course, the L^1 penalty is only one choice of penalty that can result in a sparse representation.
- Others include the penalty derived from a Student-t prior on the representation (Olshausen and Field [2005], Bergstra et al. [2011]) and KL divergence penalties (Larochelle and Bengio [2008]) that are especially useful for representations with elements constrained to lie on the unit interval.
- Lee et al. [2008] and Goodfellow et al. [2009] both provide examples of strategies based on regularizing the average activation across several examples, $\frac{1}{m}\sum_i \boldsymbol{h}^{(i)}$, to be near some target value, such as a vector with 0.01 for each entry.

- Of course, the L^1 penalty is only one choice of penalty that can result in a sparse representation.
- Others include the penalty derived from a Student-t prior on the representation (Olshausen and Field [2005], Bergstra et al. [2011]) and KL divergence penalties (Larochelle and Bengio [2008]) that are especially useful for representations with elements constrained to lie on the unit interval.
- Lee et al. [2008] and Goodfellow et al. [2009] both provide examples of strategies based on regularizing the average activation across several examples, $\frac{1}{m}\sum_i \boldsymbol{h}^{(i)}$, to be near some target value, such as a vector with 0.01 for each entry.

Sparse Representation

- Of course, the L^1 penalty is only one choice of penalty that can result in a sparse representation.
- Others include the penalty derived from a Student-t prior on the representation (Olshausen and Field [2005], Bergstra et al. [2011]) and KL divergence penalties (Larochelle and Bengio [2008]) that are especially useful for representations with elements constrained to lie on the unit interval.
- Lee et al. [2008] and Goodfellow et al. [2009] both provide examples of strategies based on regularizing the average activation across several examples, $\frac{1}{m}\sum_i \boldsymbol{h}^{(i)}$, to be near some target value, such as a vector with 0.01 for each entry.

Sparse Representation

- Other approaches obtain representational sparsity with a hard constraint on the activation values.
- For example, *orthogonal matching pursuit* (Pati et al. [1993]) encodes an input $\boldsymbol{x}$ with representation $\boldsymbol{h}$ that solves the constrained optimization problem

$$\arg\min_{\boldsymbol{h}, \|\boldsymbol{h}\|_0 < k} \|\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}\|^2$$

  where $\|\boldsymbol{h}\|_0$ is the number of non-zero entries of $\boldsymbol{h}$.
- This problem can be solved efficiently when $\boldsymbol{W}$ is constrained to be orthogonal.
- This method is often called OMP-k with the value of k specified to indicate the number of non-zero features allowed.
- Coates et al. [2010] demonstrated that OMP-1 can be very a effective feature extractor for deep architectures.

Sparse Representation

- Other approaches obtain representational sparsity with a hard constraint on the activation values.
- For example, *orthogonal matching pursuit* (Pati et al. [1993]) encodes an input $\boldsymbol{x}$ with representation $\boldsymbol{h}$ that solves the constrained optimization problem

$$\arg\min_{\boldsymbol{h}, \|\boldsymbol{h}\|_0 < k} \|\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}\|^2$$

  where $\|\boldsymbol{h}\|_0$ is the number of non-zero entries of $\boldsymbol{h}$.
- This problem can be solved efficiently when $\boldsymbol{W}$ is constrained to be orthogonal.
- This method is often called OMP-k with the value of k specified to indicate the number of non-zero features allowed.
- Coates et al. [2010] demonstrated that OMP-1 can be very a effective feature extractor for deep architectures.

Sparse Representation

- Other approaches obtain representational sparsity with a hard constraint on the activation values.
- For example, *orthogonal matching pursuit* (Pati et al. [1993]) encodes an input $\boldsymbol{x}$ with representation $\boldsymbol{h}$ that solves the constrained optimization problem

$$\arg\min_{\boldsymbol{h}, \|\boldsymbol{h}\|_0 < k} \|\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}\|^2$$

  where $\|\boldsymbol{h}\|_0$ is the number of non-zero entries of $\boldsymbol{h}$.
- This problem can be solved efficiently when $\boldsymbol{W}$ is constrained to be orthogonal.
- This method is often called OMP-k with the value of k specified to indicate the number of non-zero features allowed.
- Coates et al. [2010] demonstrated that OMP-1 can be very a effective feature extractor for deep architectures.

Sparse Representation

- Other approaches obtain representational sparsity with a hard constraint on the activation values.
- For example, *orthogonal matching pursuit* (Pati et al. [1993]) encodes an input $\boldsymbol{x}$ with representation $\boldsymbol{h}$ that solves the constrained optimization problem

$$\arg\min_{\boldsymbol{h}, \|\boldsymbol{h}\|_0 < k} \|\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}\|^2$$

  where $\|\boldsymbol{h}\|_0$ is the number of non-zero entries of $\boldsymbol{h}$.
- This problem can be solved efficiently when $\boldsymbol{W}$ is constrained to be orthogonal.
- This method is often called OMP-k with the value of k specified to indicate the number of non-zero features allowed.
- Coates et al. [2010] demonstrated that OMP-1 can be very a effective feature extractor for deep architectures.

Sparse Representation

- Other approaches obtain representational sparsity with a hard constraint on the activation values.
- For example, *orthogonal matching pursuit* (Pati et al. [1993]) encodes an input $\boldsymbol{x}$ with representation $\boldsymbol{h}$ that solves the constrained optimization problem

$$\arg\min_{\boldsymbol{h}, \|\boldsymbol{h}\|_0 < k} \|\boldsymbol{x} - \boldsymbol{W}\boldsymbol{h}\|^2$$

  where $\|\boldsymbol{h}\|_0$ is the number of non-zero entries of $\boldsymbol{h}$.
- This problem can be solved efficiently when $\boldsymbol{W}$ is constrained to be orthogonal.
- This method is often called OMP-k with the value of k specified to indicate the number of non-zero features allowed.
- Coates et al. [2010] demonstrated that OMP-1 can be very a effective feature extractor for deep architectures.

- Bagging(short for boostrap aggregating) is a technique for reducing generalization error by combining several models (Breiman [1996]).
- The idea is to train several different models separately, then have all of the models vote on the output for test examples.
- This is an example of a general strategy in machine learning called *model averaging*.
- Techniques employing this strategy are known as *ensemble methods*.

Bagging and Other Ensemble Methods

- Bagging(short for boostrap aggregating) is a technique for reducing generalization error by combining several models (Breiman [1996]).
- The idea is to train several different models separately, then have all of the models vote on the output for test examples.
- This is an example of a general strategy in machine learning called *model averaging*.
- Techniques employing this strategy are known as *ensemble methods*.

Bagging and Other Ensemble Methods

- Bagging(short for boostrap aggregating) is a technique for reducing generalization error by combining several models (Breiman [1996]).
- The idea is to train several different models separately, then have all of the models vote on the output for test examples.
- This is an example of a general strategy in machine learning called *model averaging*.
- Techniques employing this strategy are known as *ensemble methods*.

Bagging and Other Ensemble Methods

- Bagging(short for boostrap aggregating) is a technique for reducing generalization error by combining several models (Breiman [1996]).
- The idea is to train several different models separately, then have all of the models vote on the output for test examples.
- This is an example of a general strategy in machine learning called *model averaging*.
- Techniques employing this strategy are known as *ensemble methods*.

Bagging and Other Ensemble Methods

- Bagging(short for boostrap aggregating) is a technique for reducing generalization error by combining several models (Breiman [1996]).
- The idea is to train several different models separately, then have all of the models vote on the output for test examples.
- This is an example of a general strategy in machine learning called *model averaging*.
- Techniques employing this strategy are known as *ensemble methods*.

Bagging and Other Ensemble Methods

- The reason that model averaging works is that different models will usually not make all the same errors on the test set.
- Consider for example a set of k regression models.
- Suppose that each model makes an error ϵ_i on each example, with the errors drawn from a zero-mean multivariate normal distribution with variance $\mathbb{E}[\epsilon_u^2] = v$ and covariance $\mathbb{E}[\epsilon_i \epsilon_j] = c$.
- Then the error made by the average prediction of all the ensemble models is $\frac{1}{k}\sum_i \epsilon_i$.
- The expected squared error of the ensemble predictor is

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i \left(\epsilon_i^2 + \sum_{i \neq j} \epsilon_i \epsilon_j\right)\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

Bagging and Other Ensemble Methods

- The reason that model averaging works is that different models will usually not make all the same errors on the test set.
- Consider for example a set of k regression models.
- Suppose that each model makes an error ϵ_i on each example, with the errors drawn from a zero-mean multivariate normal distribution with variance $\mathbb{E}[\epsilon_u^2] = v$ and covariance $\mathbb{E}[\epsilon_i \epsilon_j] = c$.
- Then the error made by the average prediction of all the ensemble models is $\frac{1}{k}\sum_i \epsilon_i$.
- The expected squared error of the ensemble predictor is

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i\left(\epsilon_i^2 + \sum_{i \neq j} \epsilon_i \epsilon_j\right)\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

- The reason that model averaging works is that different models will usually not make all the same errors on the test set.
- Consider for example a set of k regression models.
- Suppose that each model makes an error ϵ_i on each example, with the errors drawn from a zero-mean multivariate normal distribution with variance $\mathbb{E}[\epsilon_u^2] = v$ and covariance $\mathbb{E}[\epsilon_i \epsilon_j] = c$.
- Then the error made by the average prediction of all the ensemble models is $\frac{1}{k}\sum_i \epsilon_i$.
- The expected squared error of the ensemble predictor is

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i \left(\epsilon_i^2 + \sum_{i \neq j} \epsilon_i \epsilon_j\right)\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

Bagging and Other Ensemble Methods

- The reason that model averaging works is that different models will usually not make all the same errors on the test set.
- Consider for example a set of k regression models.
- Suppose that each model makes an error ϵ_i on each example, with the errors drawn from a zero-mean multivariate normal distribution with variance $\mathbb{E}[\epsilon_u^2] = v$ and covariance $\mathbb{E}[\epsilon_i \epsilon_j] = c$.
- Then the error made by the average prediction of all the ensemble models is $\frac{1}{k}\sum_i \epsilon_i$.
- The expected squared error of the ensemble predictor is

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i \left(\epsilon_i^2 + \sum_{i \neq j} \epsilon_i \epsilon_j\right)\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

- The reason that model averaging works is that different models will usually not make all the same errors on the test set.
- Consider for example a set of k regression models.
- Suppose that each model makes an error ϵ_i on each example, with the errors drawn from a zero-mean multivariate normal distribution with variance $\mathbb{E}[\epsilon_u^2] = v$ and covariance $\mathbb{E}[\epsilon_i \epsilon_j] = c$.
- Then the error made by the average prediction of all the ensemble models is $\frac{1}{k}\sum_i \epsilon_i$.
- The expected squared error of the ensemble predictor is

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k^2}\mathbb{E}\left[\sum_i \left(\epsilon_i^2 + \sum_{i \neq j} \epsilon_i \epsilon_j\right)\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

- In the case where the errors are perfectly correlated and $c = v$, the mean squared error reduces to v, so the model average does not help at all.
- In the case where the errors are perfectly uncorrelated and $c = 0$, the expected squared error of the ensemble is only $\frac{1}{k}v$. This means that the expected squared error of the ensemble decreases linearly with the ensemble size.
- In other words, on average, the ensemble will perform at least as well as any of its members, and if the members make independent errors, the ensemble will perform significantly better than it members.

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

- In the case where the errors are perfectly correlated and $c = v$, the mean squared error reduces to v, so the model average does not help at all.
- In the case where the errors are perfectly uncorrelated and $c = 0$, the expected squared error of the ensemble is only $\frac{1}{k}v$. This means that the expected squared error of the ensemble decreases linearly with the ensemble size.
- In other words, on average, the ensemble will perform at least as well as any of its members, and if the members make independent errors, the ensemble will perform significantly better than it members.

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k}v + \frac{k-1}{k}c$$

- In the case where the errors are perfectly correlated and $c = v$, the mean squared error reduces to v, so the model average does not help at all.
- In the case where the errors are perfectly uncorrelated and $c = 0$, the expected squared error of the ensemble is only $\frac{1}{k}v$. This means that the expected squared error of the ensemble decreases linearly with the ensemble size.
- In other words, on average, the ensemble will perform at least as well as any of its members, and if the members make independent errors, the ensemble will perform significantly better than it members.

Bagging and Other Ensemble Methods

- Different ensemble methods construct the ensemble of the models in different ways.
- For example, each member of the ensemble could be formed by training a completely different kind of model using a different algorithm or objective function.
- Bagging is a method that allows the same kind of model, training algorithm and objective function to be reused several times.
- Specifically, bagging involves constructing k different datasets.
- Each dataset has the same number of examples as the original dataset, but each dataset is constructed by sampling with replacement from the origin dataset.
- This means that, with high probability, each dataset is missing some of the examples from the original dataset and also contains several duplicate examples.
- Model i is then trained on dataset i.

Bagging and Other Ensemble Methods

- Different ensemble methods construct the ensemble of the models in different ways.
- For example, each member of the ensemble could be formed by training a completely different kind of model using a different algorithm or objective function.
- Bagging is a method that allows the same kind of model, training algorithm and objective function to be reused several times.
- Specifically, bagging involves constructing k different datasets.
- Each dataset has the same number of examples as the original dataset, but each dataset is constructed by sampling with replacement from the origin dataset.
- This means that, with high probability, each dataset is missing some of the examples from the original dataset and also contains several duplicate examples.
- Model i is then trained on dataset i.

Bagging and Other Ensemble Methods

- Different ensemble methods construct the ensemble of the models in different ways.
- For example, each member of the ensemble could be formed by training a completely different kind of model using a different algorithm or objective function.
- Bagging is a method that allows the same kind of model, training algorithm and objective function to be reused several times.
- Specifically, bagging involves constructing k different datasets.
- Each dataset has the same number of examples as the original dataset, but each dataset is constructed by sampling with replacement from the origin dataset.
- This means that, with high probability, each dataset is missing some of the examples from the original dataset and also contains several duplicate examples.
- Model i is then trained on dataset i.

Bagging and Other Ensemble Methods

- Different ensemble methods construct the ensemble of the models in different ways.
- For example, each member of the ensemble could be formed by training a completely different kind of model using a different algorithm or objective function.
- Bagging is a method that allows the same kind of model, training algorithm and objective function to be reused several times.
- Specifically, bagging involves constructing k different datasets.
- Each dataset has the same number of examples as the original dataset, but each dataset is constructed by sampling with replacement from the origin dataset.
- This means that, with high probability, each dataset is missing some of the examples from the original dataset and also contains several duplicate examples.
- Model i is then trained on dataset i.

Bagging and Other Ensemble Methods

- Different ensemble methods construct the ensemble of the models in different ways.
- For example, each member of the ensemble could be formed by training a completely different kind of model using a different algorithm or objective function.
- Bagging is a method that allows the same kind of model, training algorithm and objective function to be reused several times.
- Specifically, bagging involves constructing k different datasets.
- Each dataset has the same number of examples as the original dataset, but each dataset is constructed by sampling with replacement from the origin dataset.
- This means that, with high probability, each dataset is missing some of the examples from the original dataset and also contains several duplicate examples.
- Model i is then trained on dataset i.

Bagging and Other Ensemble Methods

- Different ensemble methods construct the ensemble of the models in different ways.
- For example, each member of the ensemble could be formed by training a completely different kind of model using a different algorithm or objective function.
- Bagging is a method that allows the same kind of model, training algorithm and objective function to be reused several times.
- Specifically, bagging involves constructing k different datasets.
- Each dataset has the same number of examples as the original dataset, but each dataset is constructed by sampling with replacement from the origin dataset.
- This means that, with high probability, each dataset is missing some of the examples from the original dataset and also contains several duplicate examples.
- Model i is then trained on dataset i.

Bagging and Other Ensemble Methods

- Different ensemble methods construct the ensemble of the models in different ways.
- For example, each member of the ensemble could be formed by training a completely different kind of model using a different algorithm or objective function.
- Bagging is a method that allows the same kind of model, training algorithm and objective function to be reused several times.
- Specifically, bagging involves constructing k different datasets.
- Each dataset has the same number of examples as the original dataset, but each dataset is constructed by sampling with replacement from the origin dataset.
- This means that, with high probability, each dataset is missing some of the examples from the original dataset and also contains several duplicate examples.
- Model i is then trained on dataset i.

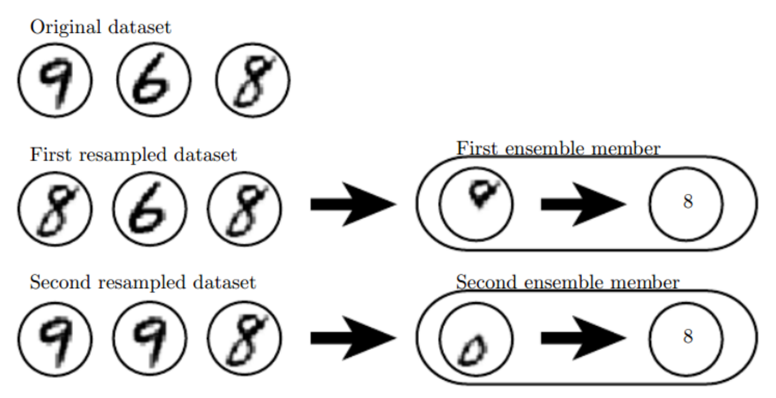


Fig. 3: A cartoon depiction of how bagging works

Bagging and Other Ensemble Methods

- Neural networks reach a wide enough variety of solution points that they can often benefit from model averaging even if all of the models are trained on the same dataset.
- Differences in random initialization, random selection of minibatches, differences in hyperparameters, or different outcomes of non-deterministic implementations of neural networks are often enough to cause different members of the ensemble to make partially independent errors.

Bagging and Other Ensemble Methods

- Neural networks reach a wide enough variety of solution points that they can often benefit from model averaging even if all of the models are trained on the same dataset.
- Differences in random initialization, random selection of minibatches, differences in hyperparameters, or different outcomes of non-deterministic implementations of neural networks are often enough to cause different members of the ensemble to make partially independent errors.

- Model averaging is an extremely powerful and reliable method for reducing generalization error.
- Its use is usually discouraged when benchmarking algorithms for scientific papers.
- Machine learning contest are usually won by methods using model averaging over dozens of models.
- Not all techniques for constructing ensembles are designed to make the ensemble more regularized than the individual models.
- For example, a technique called *boosting* (Freund et al. [1996]) constructs an ensemble with higher capacity than the individual models.

Bagging and Other Ensemble Methods

- Model averaging is an extremely powerful and reliable method for reducing generalization error.
- Its use is usually discouraged when benchmarking algorithms for scientific papers.
- Machine learning contest are usually won by methods using model averaging over dozens of models.
- Not all techniques for constructing ensembles are designed to make the ensemble more regularized than the individual models.
- For example, a technique called *boosting* (Freund et al. [1996]) constructs an ensemble with higher capacity than the individual models.

Bagging and Other Ensemble Methods

- Model averaging is an extremely powerful and reliable method for reducing generalization error.
- Its use is usually discouraged when benchmarking algorithms for scientific papers.
- Machine learning contest are usually won by methods using model averaging over dozens of models.
- Not all techniques for constructing ensembles are designed to make the ensemble more regularized than the individual models.
- For example, a technique called *boosting* (Freund et al. [1996]) constructs an ensemble with higher capacity than the individual models.

Bagging and Other Ensemble Methods

- Model averaging is an extremely powerful and reliable method for reducing generalization error.
- Its use is usually discouraged when benchmarking algorithms for scientific papers.
- Machine learning contest are usually won by methods using model averaging over dozens of models.
- Not all techniques for constructing ensembles are designed to make the ensemble more regularized than the individual models.
- For example, a technique called *boosting* (Freund et al. [1996]) constructs an ensemble with higher capacity than the individual models.

Bagging and Other Ensemble Methods

- Model averaging is an extremely powerful and reliable method for reducing generalization error.
- Its use is usually discouraged when benchmarking algorithms for scientific papers.
- Machine learning contest are usually won by methods using model averaging over dozens of models.
- Not all techniques for constructing ensembles are designed to make the ensemble more regularized than the individual models.
- For example, a technique called *boosting* (Freund et al. [1996]) constructs an ensemble with higher capacity than the individual models.

- Boosting has been applied to build ensembles of neural networks (Schwenk and Bengio [1998]) by incrementally adding neural networks to the ensemble.
- Boosting has also been applied interpreting an individual neural network as an ensemble (Bengio et al. [2006]), incrementally adding hidden units to the neural network.

Bagging and Other Ensemble Methods

- Boosting has been applied to build ensembles of neural networks (Schwenk and Bengio [1998]) by incrementally adding neural networks to the ensemble.
- Boosting has also been applied interpreting an individual neural network as an ensemble (Bengio et al. [2006]), incrementally adding hidden units to the neural network.

Dropout

- Dropout ([Srivastava et al., 2014]) provides a computationally inexpensive but powerful method of regularizing a broad family of models.
- Dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks.
- Bagging involves training multiple models, and evaluating multiple models on each test example.
- This seems impractical when each model is a large neural network, since training and evaluating such networks is costly in terms of runtime and memory.
- It is common to use ensembles of five to ten neural networks. but more than this rapidly becomes unwieldy.
- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.

Dropout

- Dropout ([Srivastava et al., 2014]) provides a computationally inexpensive but powerful method of regularizing a broad family of models.
- Dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks.
- Bagging involves training multiple models, and evaluating multiple models on each test example.
- This seems impractical when each model is a large neural network, since training and evaluating such networks is costly in terms of runtime and memory.
- It is common to use ensembles of five to ten neural networks. but more than this rapidly becomes unwieldy.
- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.

Dropout

- Dropout ([Srivastava et al., 2014]) provides a computationally inexpensive but powerful method of regularizing a broad family of models.
- Dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks.
- Bagging involves training multiple models, and evaluating multiple models on each test example.
- This seems impractical when each model is a large neural network, since training and evaluating such networks is costly in terms of runtime and memory.
- It is common to use ensembles of five to ten neural networks. but more than this rapidly becomes unwieldy.
- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.

Dropout

- Dropout ([Srivastava et al., 2014]) provides a computationally inexpensive but powerful method of regularizing a broad family of models.
- Dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks.
- Bagging involves training multiple models, and evaluating multiple models on each test example.
- This seems impractical when each model is a large neural network, since training and evaluating such networks is costly in terms of runtime and memory.
- It is common to use ensembles of five to ten neural networks. but more than this rapidly becomes unwieldy.
- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.

Dropout

- Dropout ([Srivastava et al., 2014]) provides a computationally inexpensive but powerful method of regularizing a broad family of models.
- Dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks.
- Bagging involves training multiple models, and evaluating multiple models on each test example.
- This seems impractical when each model is a large neural network, since training and evaluating such networks is costly in terms of runtime and memory.
- It is common to use ensembles of five to ten neural networks. but more than this rapidly becomes unwieldy.
- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.

Dropout

- Dropout ([Srivastava et al., 2014]) provides a computationally inexpensive but powerful method of regularizing a broad family of models.
- Dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks.
- Bagging involves training multiple models, and evaluating multiple models on each test example.
- This seems impractical when each model is a large neural network, since training and evaluating such networks is costly in terms of runtime and memory.
- It is common to use ensembles of five to ten neural networks. but more than this rapidly becomes unwieldy.
- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.

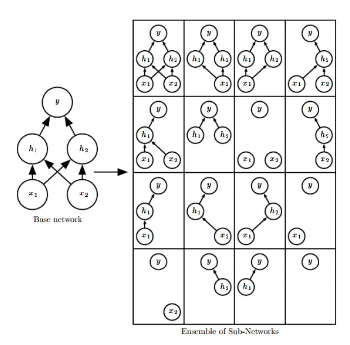
Dropout

- Dropout ([Srivastava et al., 2014]) provides a computationally inexpensive but powerful method of regularizing a broad family of models.
- Dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks.
- Bagging involves training multiple models, and evaluating multiple models on each test example.
- This seems impractical when each model is a large neural network, since training and evaluating such networks is costly in terms of runtime and memory.
- It is common to use ensembles of five to ten neural networks. but more than this rapidly becomes unwieldy.
- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.

Dropout

Specifically, dropout trains the ensemble consisting of all sub-networks that can be formed by removing non-output units from an underlying base network.

- In the most modern neural networks, based on a series of affine transformations and nonlinearities, we can effectively remove a unit from a network by multiplying its output value by zero.
- This procedure requires some slight modification for models such as radial basis function networks, which take the difference between the unit's state and some reference value.



Dropout

- Here, we present the dropout algorithm in terms of multiplication by zero for simplicity, but it can be trivially modified to work with other operations that remove a unit from the network.
- Recall that to learn with bagging, we define k different models, construct k different datasets by sampling from the training set with replacement, and then train model i on dataset i.
- Dropout aims to approximate this process, but with an exponentially large number of neural networks.

Dropout

- Here, we present the dropout algorithm in terms of multiplication by zero for simplicity, but it can be trivially modified to work with other operations that remove a unit from the network.
- Recall that to learn with bagging, we define k different models, construct k different datasets by sampling from the training set with replacement, and then train model i on dataset i.
- Dropout aims to approximate this process, but with an exponentially large number of neural networks.

Dropout

- Here, we present the dropout algorithm in terms of multiplication by zero for simplicity, but it can be trivially modified to work with other operations that remove a unit from the network.
- Recall that to learn with bagging, we define k different models, construct k different datasets by sampling from the training set with replacement, and then train model i on dataset i.
- Dropout aims to approximate this process, but with an exponentially large number of neural networks.

Dropout

- Specifically, to train with dropout, we use a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent.
- Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network.
- The mask for each unit is sampled independently from all of the others.
- The probability of sampling a mask value of one is a hyperparameter fixed before training begins.
- Typically, an input unit is included with probability 0.8 and a hidden unit is included with probability 0.5.
- We then run forward propagation, back-propagation, and the learning update as usual.

Dropout

- Specifically, to train with dropout, we use a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent.
- Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network.
- The mask for each unit is sampled independently from all of the others.
- The probability of sampling a mask value of one is a hyperparameter fixed before training begins.
- Typically, an input unit is included with probability 0.8 and a hidden unit is included with probability 0.5.
- We then run forward propagation, back-propagation, and the learning update as usual.

Dropout

- Specifically, to train with dropout, we use a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent.
- Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network.
- The mask for each unit is sampled independently from all of the others.
- The probability of sampling a mask value of one is a hyperparameter fixed before training begins.
- Typically, an input unit is included with probability 0.8 and a hidden unit is included with probability 0.5.
- We then run forward propagation, back-propagation, and the learning update as usual.

Dropout

- Specifically, to train with dropout, we use a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent.
- Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network.
- The mask for each unit is sampled independently from all of the others.
- The probability of sampling a mask value of one is a hyperparameter fixed before training begins.
- Typically, an input unit is included with probability 0.8 and a hidden unit is included with probability 0.5.
- We then run forward propagation, back-propagation, and the learning update as usual.

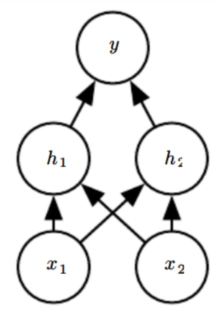
Dropout

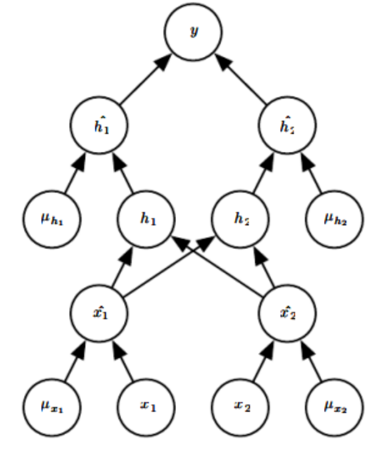
- Specifically, to train with dropout, we use a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent.
- Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network.
- The mask for each unit is sampled independently from all of the others.
- The probability of sampling a mask value of one is a hyperparameter fixed before training begins.
- Typically, an input unit is included with probability 0.8 and a hidden unit is included with probability 0.5.
- We then run forward propagation, back-propagation, and the learning update as usual.

Dropout

- Specifically, to train with dropout, we use a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent.
- Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network.
- The mask for each unit is sampled independently from all of the others.
- The probability of sampling a mask value of one is a hyperparameter fixed before training begins.
- Typically, an input unit is included with probability 0.8 and a hidden unit is included with probability 0.5.
- We then run forward propagation, back-propagation, and the learning update as usual.

Dropout





Dropout

- More formally, suppose that a mask vector μ specifies which units to include, and $J(\boldsymbol{\theta}, \boldsymbol{\mu})$ defines the cost of the model defined by parameters $\boldsymbol{\theta}$ and mask $\boldsymbol{\mu}$.
- Then dropout training consists in minimizing $\mathbb{E}_{\boldsymbol{\mu}} J(\boldsymbol{\theta}, \boldsymbol{\mu})$.
- The expectation contains exponentially many terms but we can obtain an unbiased estimate of its gradient by sampling values of μ

Dropout

- More formally, suppose that a mask vector μ specifies which units to include, and $J(\boldsymbol{\theta}, \boldsymbol{\mu})$ defines the cost of the model defined by parameters $\boldsymbol{\theta}$ and mask $\boldsymbol{\mu}$.
- Then dropout training consists in minimizing $\mathbb{E}_{\boldsymbol{\mu}} J(\boldsymbol{\theta}, \boldsymbol{\mu})$.
- The expectation contains exponentially many terms but we can obtain an unbiased estimate of its gradient by sampling values of μ

Dropout

- More formally, suppose that a mask vector μ specifies which units to include, and $J(\boldsymbol{\theta}, \boldsymbol{\mu})$ defines the cost of the model defined by parameters $\boldsymbol{\theta}$ and mask $\boldsymbol{\mu}$.
- Then dropout training consists in minimizing $\mathbb{E}_{\boldsymbol{\mu}} J(\boldsymbol{\theta}, \boldsymbol{\mu})$.
- The expectation contains exponentially many terms but we can obtain an unbiased estimate of its gradient by sampling values of μ

Dropout

- Dropout training is not quite the same as bagging traing.
- In the case of bagging, the models are all independent.
- In the case of dropout, the models share parameters, with each model inheriting a different subset of parameters from the parent neural network. This parameter sharing makes it possible to represent an exponential number of models with a tractable amount of memory.
- In the case of bagging, each model is trained to convergence on its respective training set.

Dropout

- Dropout training is not quite the same as bagging traing.
- In the case of bagging, the models are all independent.
- In the case of dropout, the models share parameters, with each model inheriting a different subset of parameters from the parent neural network. This parameter sharing makes it possible to represent an exponential number of models with a tractable amount of memory.
- In the case of bagging, each model is trained to convergence on its respective training set.

Dropout

- Dropout training is not quite the same as bagging traing.
- In the case of bagging, the models are all independent.
- In the case of dropout, the models share parameters, with each model inheriting a different subset of parameters from the parent neural network. This parameter sharing makes it possible to represent an exponential number of models with a tractable amount of memory.
- In the case of bagging, each model is trained to convergence on its respective training set.

Dropout

- Dropout training is not quite the same as bagging traing.
- In the case of bagging, the models are all independent.
- In the case of dropout, the models share parameters, with each model inheriting a different subset of parameters from the parent neural network. This parameter sharing makes it possible to represent an exponential number of models with a tractable amount of memory.
- In the case of bagging, each model is trained to convergence on its respective training set.

Dropout

- In the case of dropout, typically most models are not explicitly trained at all–usually, the model is large enough that it would be infeasible to sample all possible subnetworks within the lifetime of the universe.
- Instead, a tiny fraction of the possible sub-networks are each trained for a single step, and the parameter sharing causes the remaining sub-networks to arrive at good settings of the parameters.
- Beyond these, dropout follows the bagging algorithm.

Dropout

- In the case of dropout, typically most models are not explicitly trained at all–usually, the model is large enough that it would be infeasible to sample all possible subnetworks within the lifetime of the universe.
- Instead, a tiny fraction of the possible sub-networks are each trained for a single step, and the parameter sharing causes the remaining sub-networks to arrive at good settings of the parameters.
- Beyond these, dropout follows the bagging algorithm.

Dropout

- In the case of dropout, typically most models are not explicitly trained at all–usually, the model is large enough that it would be infeasible to sample all possible subnetworks within the lifetime of the universe.
- Instead, a tiny fraction of the possible sub-networks are each trained for a single step, and the parameter sharing causes the remaining sub-networks to arrive at good settings of the parameters.
- Beyond these, dropout follows the bagging algorithm.

Dropout

- To make a prediction, a bagged ensemble must accumulate votes from all of its members. We refer to this process as *inference* in this context.
- Now, we assume that the model's role is to output a probability distribution.
- In the case of bagging, each model i produces a probability distribution $p^{(i)}(y|\mathbf{x})$.
- The prediction of the ensemble is given by the arithmetic mean of all of these distributions

$$\frac{1}{k}\sum_{i=1}^{k} p^{(i)}(y|\mathbf{x})$$

Dropout

- To make a prediction, a bagged ensemble must accumulate votes from all of its members. We refer to this process as *inference* in this context.
- Now, we assume that the model's role is to output a probability distribution.
- In the case of bagging, each model i produces a probability distribution $p^{(i)}(y|\mathbf{x})$.
- The prediction of the ensemble is given by the arithmetic mean of all of these distributions

$$\frac{1}{k}\sum_{i=1}^{k} p^{(i)}(y|\mathbf{x})$$

Dropout

- To make a prediction, a bagged ensemble must accumulate votes from all of its members. We refer to this process as *inference* in this context.
- Now, we assume that the model's role is to output a probability distribution.
- In the case of bagging, each model i produces a probability distribution $p^{(i)}(y|\mathbf{x})$.
- The prediction of the ensemble is given by the arithmetic mean of all of these distributions

$$\frac{1}{k}\sum_{i=1}^{k} p^{(i)}(y|\mathbf{x})$$

Dropout

- To make a prediction, a bagged ensemble must accumulate votes from all of its members. We refer to this process as *inference* in this context.
- Now, we assume that the model's role is to output a probability distribution.
- In the case of bagging, each model i produces a probability distribution $p^{(i)}(y|\mathbf{x})$.
- The prediction of the ensemble is given by the arithmetic mean of all of these distributions

$$\frac{1}{k}\sum_{i=1}^{k} p^{(i)}(y|\mathbf{x})$$

Dropout

- In the case of dropout, each sub-model defined by mask vector $\boldsymbol{\mu}$ defines a probability distribution $p(y|\boldsymbol{x}, \boldsymbol{\mu})$.
- The arithmetic mean over all masks is given by

$$\sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y|\boldsymbol{x}, \boldsymbol{\mu})$$

  where $p(\boldsymbol{\mu})$ is the probability distribution that was used to sample μ at training time.
- Because this sum includes an exponential number of terms, it is intractable to evaluate except in cases where the structure of the model permits some form of simplification.
- We can approximate the inference with sampling, by averaging together the output from many masks.

Dropout

- In the case of dropout, each sub-model defined by mask vector $\boldsymbol{\mu}$ defines a probability distribution $p(y|\mathbf{x}, \boldsymbol{\mu})$.
- The arithmetic mean over all masks is given by

$$\sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y|\mathbf{x}, \boldsymbol{\mu})$$

  where $p(\boldsymbol{\mu})$ is the probability distribution that was used to sample μ at training time.
- Because this sum includes an exponential number of terms, it is intractable to evaluate except in cases where the structure of the model permits some form of simplification.
- We can approximate the inference with sampling, by averaging together the output from many masks.

Dropout

- In the case of dropout, each sub-model defined by mask vector $\boldsymbol{\mu}$ defines a probability distribution $p(y|\mathbf{x}, \boldsymbol{\mu})$.
- The arithmetic mean over all masks is given by

$$\sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y|\mathbf{x}, \boldsymbol{\mu})$$

  where $p(\boldsymbol{\mu})$ is the probability distribution that was used to sample μ at training time.
- Because this sum includes an exponential number of terms, it is intractable to evaluate except in cases where the structure of the model permits some form of simplification.
- We can approximate the inference with sampling, by averaging together the output from many masks.

Dropout

- In the case of dropout, each sub-model defined by mask vector $\boldsymbol{\mu}$ defines a probability distribution $p(y|\mathbf{x}, \boldsymbol{\mu})$.
- The arithmetic mean over all masks is given by

$$\sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(y|\mathbf{x}, \boldsymbol{\mu})$$

  where $p(\boldsymbol{\mu})$ is the probability distribution that was used to sample μ at training time.
- Because this sum includes an exponential number of terms, it is intractable to evaluate except in cases where the structure of the model permits some form of simplification.
- We can approximate the inference with sampling, by averaging together the output from many masks.

Dropout

- However, there is an even better approach, that allows us to obtain a good approximation to the predictions of the entire ensemble, at the cost of only one forward propagation.
- To do so, we change to using the geometric mean rather than the arithmetic mean of the ensemble member's prediction distributions.
- Warde-Farley et al. [2013] present arguments and empirical evidence that the geometric mean performs comparably to the arithmetic mean in this context.

Dropout

- However, there is an even better approach, that allows us to obtain a good approximation to the predictions of the entire ensemble, at the cost of only one forward propagation.
- To do so, we change to using the geometric mean rather than the arithmetic mean of the ensemble member's prediction distributions.
- Warde-Farley et al. [2013] present arguments and empirical evidence that the geometric mean performs comparably to the arithmetic mean in this context.

Dropout

- However, there is an even better approach, that allows us to obtain a good approximation to the predictions of the entire ensemble, at the cost of only one forward propagation.
- To do so, we change to using the geometric mean rather than the arithmetic mean of the ensemble member's prediction distributions.
- Warde-Farley et al. [2013] present arguments and empirical evidence that the geometric mean performs comparably to the arithmetic mean in this context.

Dropout

- The geometric mean of multiple probability distributions is not guaranteed to be a probability distribution.
- To guarantee that the result is a probability distribution, we impose the requirement that none of the sub-models assigns probability 0 to any event, and we renormalize the resulting distribution.
- The unnormalized probability distribution defined directly by the geometric mean is given by

$$\tilde{p}_{\text{ensemble}}(y|\mathbf{x}) = \sqrt[2^d]{\prod_{\boldsymbol{\mu}} p(y|\mathbf{x}, \boldsymbol{\mu})}$$

where d is the number of units that may be dropped.

Dropout

- The geometric mean of multiple probability distributions is not guaranteed to be a probability distribution.
- To guarantee that the result is a probability distribution, we impose the requirement that none of the sub-models assigns probability 0 to any event, and we renormalize the resulting distribution.
- The unnormalized probability distribution defined directly by the geometric mean is given by

$$\tilde{p}_{\text{ensemble}}(y|\mathbf{x}) = \sqrt[2^d]{\prod_{\boldsymbol{\mu}} p(y|\mathbf{x}, \boldsymbol{\mu})}$$

where d is the number of units that may be dropped.

Dropout

- The geometric mean of multiple probability distributions is not guaranteed to be a probability distribution.
- To guarantee that the result is a probability distribution, we impose the requirement that none of the sub-models assigns probability 0 to any event, and we renormalize the resulting distribution.
- The unnormalized probability distribution defined directly by the geometric mean is given by

$$\tilde{p}_{\text{ensemble}}(y|\mathbf{x}) = \sqrt[2^d]{\prod_{\boldsymbol{\mu}} p(y|\mathbf{x}, \boldsymbol{\mu})}$$

where d is the number of units that may be dropped.

Dropout

- Here we use a uniform distribution over $\boldsymbol{\mu}$ to simplify the presentation, but non-uniform distributions are also possible.
- To make predictions we must re-normalize the ensemble:

$$p_{\text{ensemble}}(y|\mathbf{x}) = \frac{\tilde{p}_{\text{ensemble}}(y|\mathbf{x})}{\sum_{y'} \tilde{p}_{\text{ensemble}}(y'|\mathbf{x})}$$

- The key insight (Hinton et al. [2012]) involved in dropout is that we can approximate p_{ensemble} by evaluating $p(y|\mathbf{x})$ in one model: the model with all units, but with the weight going out of unit i multiplied by the probability of including unit i.
- We call this approach the *weight scaling inference rule*.
- There is not yet any theoretical argument for the accuracy of this approximate inference rule in deep nonlinear networks, but empirically it performs very well.

Dropout

- Here we use a uniform distribution over $\boldsymbol{\mu}$ to simplify the presentation, but non-uniform distributions are also possible.
- To make predictions we must re-normalize the ensemble:

$$p_{\text{ensemble}}(y|x) = \frac{\tilde{p}_{\text{ensemble}}(y|x)}{\sum_{y'} \tilde{p}_{\text{ensemble}}(y'|x)}$$

- The key insight (Hinton et al. [2012]) involved in dropout is that we can approximate p_{ensemble} by evaluating $p(y|x)$ in one model: the model with all units, but with the weight going out of unit i multiplied by the probability of including unit i.
- We call this approach the *weight scaling inference rule*.
- There is not yet any theoretical argument for the accuracy of this approximate inference rule in deep nonlinear networks, but empirically it performs very well.

Dropout

- Here we use a uniform distribution over $\boldsymbol{\mu}$ to simplify the presentation, but non-uniform distributions are also possible.
- To make predictions we must re-normalize the ensemble:

$$p_{\text{ensemble}}(y|\mathbf{x}) = \frac{\tilde{p}_{\text{ensemble}}(y|\mathbf{x})}{\sum_{y'} \tilde{p}_{\text{ensemble}}(y'|\mathbf{x})}$$

- The key insight (Hinton et al. [2012]) involved in dropout is that we can approximate p_{ensemble} by evaluating $p(y|\mathbf{x})$ in one model: the model with all units, but with the weight going out of unit i multiplied by the probability of including unit i.
- We call this approach the *weight scaling inference rule*.
- There is not yet any theoretical argument for the accuracy of this approximate inference rule in deep nonlinear networks, but empirically it performs very well.

Dropout

- Here we use a uniform distribution over $\boldsymbol{\mu}$ to simplify the presentation, but non-uniform distributions are also possible.
- To make predictions we must re-normalize the ensemble:

$$p_{\text{ensemble}}(y|\mathbf{x}) = \frac{\tilde{p}_{\text{ensemble}}(y|\mathbf{x})}{\sum_{y'} \tilde{p}_{\text{ensemble}}(y'|\mathbf{x})}$$

- The key insight (Hinton et al. [2012]) involved in dropout is that we can approximate p_{ensemble} by evaluating $p(y|\mathbf{x})$ in one model: the model with all units, but with the weight going out of unit i multiplied by the probability of including unit i.
- We call this approach the *weight scaling inference rule*.
- There is not yet any theoretical argument for the accuracy of this approximate inference rule in deep nonlinear networks, but empirically it performs very well.

Dropout

- Here we use a uniform distribution over $\boldsymbol{\mu}$ to simplify the presentation, but non-uniform distributions are also possible.
- To make predictions we must re-normalize the ensemble:

$$p_{\text{ensemble}}(y|\mathbf{x}) = \frac{\tilde{p}_{\text{ensemble}}(y|\mathbf{x})}{\sum_{y'} \tilde{p}_{\text{ensemble}}(y'|\mathbf{x})}$$

- The key insight (Hinton et al. [2012]) involved in dropout is that we can approximate p_{ensemble} by evaluating $p(y|\mathbf{x})$ in one model: the model with all units, but with the weight going out of unit i multiplied by the probability of including unit i.
- We call this approach the *weight scaling inference rule*.
- There is not yet any theoretical argument for the accuracy of this approximate inference rule in deep nonlinear networks, but empirically it performs very well.

Dropout

- Because we usually use an inclusion probability of $\frac{1}{2}$, the weight scaling rule usually amounts to dividing the weights by 2 at the end of training, and then using the model as usual.
- Another way to achieve the same result is to multiply the states of the units by 2 during training.
- Either way, the goal is to make sure that the expected total input to a unit at test time is roughly the same as the expected total input to that unit at train time, even though half the units at train time are missing on average.

Dropout

- Because we usually use an inclusion probability of $\frac{1}{2}$, the weight scaling rule usually amounts to dividing the weights by 2 at the end of training, and then using the model as usual.
- Another way to achieve the same result is to multiply the states of the units by 2 during training.
- Either way, the goal is to make sure that the expected total input to a unit at test time is roughly the same as the expected total input to that unit at train time, even though half the units at train time are missing on average.

Dropout

- Because we usually use an inclusion probability of $\frac{1}{2}$, the weight scaling rule usually amounts to dividing the weights by 2 at the end of training, and then using the model as usual.
- Another way to achieve the same result is to multiply the states of the units by 2 during training.
- Either way, the goal is to make sure that the expected total input to a unit at test time is roughly the same as the expected total input to that unit at train time, even though half the units at train time are missing on average.

Dropout

- For many classes of models that do not have nonlinear hidden units, the weight scaling inference rule is exact.
- For a simple example, consider a softmax regression classifier with n input variables represented by the vector $\mathbf{v}$:

$$P(\mathrm{y} = y|\mathbf{v}) = \mathrm{softmax}(\mathbf{W}^T\mathbf{v} + b)_y$$

- We can index into the family of sub-models by element-wise multiplication of the input with a binary vector d:

$$P(\mathrm{y} = y|\mathbf{v}; d) = \mathrm{softmax}(\mathbf{W}^T(d \odot \mathbf{v}) + b)_y$$

- For many classes of models that do not have nonlinear hidden units, the weight scaling inference rule is exact.
- For a simple example, consider a softmax regression classifier with n input variables represented by the vector $\mathbf{v}$:

$$P(\mathrm{y} = y|\mathbf{v}) = \mathrm{softmax}(\mathbf{W}^T\mathbf{v} + b)_y$$

- We can index into the family of sub-models by element-wise multiplication of the input with a binary vector d:

$$P(\mathrm{y} = y|\mathbf{v}; d) = \mathrm{softmax}(\mathbf{W}^T(d \odot \mathbf{v}) + b)_y$$

Dropout

- For many classes of models that do not have nonlinear hidden units, the weight scaling inference rule is exact.
- For a simple example, consider a softmax regression classifier with n input variables represented by the vector $\mathbf{v}$:

$$P(\mathrm{y} = y|\mathbf{v}) = \mathrm{softmax}(\mathbf{W}^T\mathbf{v} + b)_y$$

- We can index into the family of sub-models by element-wise multiplication of the input with a binary vector d:

$$P(\mathrm{y} = y|\mathbf{v}; d) = \mathrm{softmax}(\mathbf{W}^T(d \odot \mathbf{v}) + b)_y$$

Dropout

- The ensemble predictor is defined by re-normalizing the geometric mean over all ensemble members' predictions

$$P_{\text{ensemble}}(\mathrm{y} = y|\mathbf{v}) = \frac{\tilde{P}_{\text{ensemble}}(\mathrm{y} = y|\mathbf{v})}{\sum_{y'} \tilde{P}_{\text{ensemble}}(\mathrm{y} = y'|\boldsymbol{v})}$$

where

$$\tilde{P}_{\text{ensemble}}(\mathrm{y} = y|\boldsymbol{v}) = \sqrt[2^n]{\prod_{\boldsymbol{d} \in \{0,1\}^n} P(\mathrm{y} = y|\mathbf{v}; \boldsymbol{d})}$$

- To see that the weight scaling rule is exact, we can simplify

$$
\begin{aligned}
\tilde{P}_{\text{ensemble}}(\mathrm{y} = y|\mathbf{v}) &= \sqrt[2^n]{\prod_{d\in\{0,1\}^n} P(\mathrm{y} = y|\mathbf{v}; d)} \\
&= \sqrt[2^n]{\prod_{d\in\{0,1\}^n} \text{softmax}\left(\mathbf{W}^T(d \odot \mathbf{v}) + b\right)_y} \\
&= \sqrt[2^n]{\prod_{d\in\{0,1\}^n} \frac{\exp\left(\mathbf{W}_{y,:}^T(d \odot \mathbf{v}) + b\right)}{\sum_{y'} \exp\left(\mathbf{W}_{y',:}^T(d \odot \mathbf{v}) + b\right)}} \\
&= \frac{\sqrt[2^n]{\prod_{d\in\{0,1\}^n} \exp\left(\mathbf{W}_{y,:}^T(d \odot \mathbf{v}) + b\right)}}{\sqrt[2^n]{\prod_{d\in\{0,1\}^n} \sum_{y'} \exp\left(\mathbf{W}_{y',:}^T(d \odot \mathbf{v}) + b\right)}}
\end{aligned}
$$

Dropout

- To see that the weight scaling rule is exact, we can simplify

$$
\begin{aligned}
\tilde{P}_{\text{ensemble}}(\mathrm{y}=y|\mathbf{v}) &= \sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n} P(\mathrm{y}=y|\mathbf{v};\boldsymbol{d})} \\
&= \sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n} \text{softmax}\left(\mathbf{W}^T(\boldsymbol{d}\odot\mathbf{v})+\boldsymbol{b}\right)_y} \\
&= \sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n} \frac{\exp\left(\mathbf{W}_{y,:}^T(\boldsymbol{d}\odot\mathbf{v})+\boldsymbol{b}\right)}{\sum_{y'}\exp\left(\mathbf{W}_{y',:}^T(\boldsymbol{d}\odot\mathbf{v})+\boldsymbol{b}\right)}} \\
&= \frac{\sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n}\exp\left(\mathbf{W}_{y,:}^T(\boldsymbol{d}\odot\mathbf{v})+\boldsymbol{b}\right)}}{\sqrt[2^n]{\prod_{\boldsymbol{d}\in\{0,1\}^n}\sum_{y'}\exp\left(\mathbf{W}_{y',:}^T(\boldsymbol{d}\odot\mathbf{v})+\boldsymbol{b}\right)}}
\end{aligned}
$$

Dropout

- Because $\tilde{P}$ will be normalized, we can safely ignore multiplication by factors that are constant with respect to y:

$$
\begin{aligned}
\tilde{P}_{\text{ensemble}}(\mathrm{y} = y|\mathbf{v}) &\propto \sqrt[2^n]{\prod_{d\in\{0,1\}^n} \exp\left(\mathbf{W}_{y,:}^T (d \odot \mathbf{v}) + b\right)} \\
&= \exp\left(\frac{1}{2^n}\sum_{d\in\{0,1\}^n} \mathbf{W}_{y,:}^T (d \odot \mathbf{v}) + b\right) \\
&= \exp\left(\frac{1}{2}\mathbf{W}_{y,:}^T \mathbf{v} + b\right)
\end{aligned}
$$

- Substituting this back into $P_{\text{ensemble}}(\mathrm{y} = y|\mathbf{v}) = \frac{\tilde{P}_{\text{ensemble}}(\mathrm{y}=y|\mathbf{v})}{\sum_{y'} \tilde{P}_{\text{ensemble}}(\mathrm{y}=y'|v)}$ we obtain a softmax classifier with weights $\frac{1}{2}\mathbf{W}$

Dropout

- Because $\tilde{P}$ will be normalized, we can safely ignore multiplication by factors that are constant with respect to y:

$$
\begin{aligned}
\tilde{P}_{\text{ensemble}}(\mathrm{y} = y|\mathbf{v}) &\propto \sqrt[2^n]{\prod_{d \in \{0,1\}^n} \exp\left(\mathbf{W}_{y,:}^T (d \odot \mathbf{v}) + b\right)} \\
&= \exp\left(\frac{1}{2^n} \sum_{d \in \{0,1\}^n} \mathbf{W}_{y,:}^T (d \odot \mathbf{v}) + b\right) \\
&= \exp\left(\frac{1}{2}\mathbf{W}_{y,:}^T \mathbf{v} + b\right)
\end{aligned}
$$

- Substituting this back into $P_{\text{ensemble}}(\mathrm{y} = y|\mathbf{v}) = \frac{\tilde{P}_{\text{ensemble}}(\mathrm{y}=y|\mathbf{v})}{\sum_{y'} \tilde{P}_{\text{ensemble}}(\mathrm{y}=y'|v)}$ we obtain a softmax classifier with weights $\frac{1}{2}\mathbf{W}$

Dropout

- Because $\tilde{P}$ will be normalized, we can safely ignore multiplication by factors that are constant with respect to y:

$$
\begin{aligned}
\tilde{P}_{\text{ensemble}}(\mathrm{y} = y|\mathbf{v}) &\propto \sqrt[2^n]{\prod_{d\in\{0,1\}^n} \exp\left(\mathbf{W}_{y,:}^T(d\odot\mathbf{v}) + b\right)} \\
&= \exp\left(\frac{1}{2^n}\sum_{d\in\{0,1\}^n} \mathbf{W}_{y,:}^T(d\odot\mathbf{v}) + b\right) \\
&= \exp\left(\frac{1}{2}\mathbf{W}_{y,:}^T\mathbf{v} + b\right)
\end{aligned}
$$

- Substituting this back into $P_{\text{ensemble}}(\mathrm{y} = y|\mathbf{v}) = \frac{\tilde{P}_{\text{ensemble}}(\mathrm{y}=y|\mathbf{v})}{\sum_{y'}\tilde{P}_{\text{ensemble}}(\mathrm{y}=y'|v)}$ we obtain a softmax classifier with weights $\frac{1}{2}\mathbf{W}$

Dropout

- The weight scaling rule is also exact in other settings, including regression networks with conditionally normal outputs, and deep networks that have hidden layers without nonlinearities.
- However, the weight scaling rule is only a approximation for deep models that have nonlinearities.
- Though the approximation has not been theoretically characterized, it often works well, empirically.
- Goodfellow et al. [2013b] found experimentally that the weight scaling approximation can work better (in terms of classification accuracy) than Monte Carlo approximations to the ensemble predictor.
- This held true even when the Monte Carlo approximation was allowed to sample up to 1,000 sub-networks.

Dropout

- The weight scaling rule is also exact in other settings, including regression networks with conditionally normal outputs, and deep networks that have hidden layers without nonlinearities.
- However, the weight scaling rule is only a approximation for deep models that have nonlinearities.
- Though the approximation has not been theoretically characterized, it often works well, empirically.
- Goodfellow et al. [2013b] found experimentally that the weight scaling approximation can work better (in terms of classification accuracy) than Monte Carlo approximations to the ensemble predictor.
- This held true even when the Monte Carlo approximation was allowed to sample up to 1,000 sub-networks.

Dropout

- The weight scaling rule is also exact in other settings, including regression networks with conditionally normal outputs, and deep networks that have hidden layers without nonlinearities.
- However, the weight scaling rule is only a approximation for deep models that have nonlinearities.
- Though the approximation has not been theoretically characterized, it often works well, empirically.
- Goodfellow et al. [2013b] found experimentally that the weight scaling approximation can work better (in terms of classification accuracy) than Monte Carlo approximations to the ensemble predictor.
- This held true even when the Monte Carlo approximation was allowed to sample up to 1,000 sub-networks.

Dropout

- The weight scaling rule is also exact in other settings, including regression networks with conditionally normal outputs, and deep networks that have hidden layers without nonlinearities.
- However, the weight scaling rule is only a approximation for deep models that have nonlinearities.
- Though the approximation has not been theoretically characterized, it often works well, empirically.
- Goodfellow et al. [2013b] found experimentally that the weight scaling approximation can work better (in terms of classification accuracy) than Monte Carlo approximations to the ensemble predictor.
- This held true even when the Monte Carlo approximation was allowed to sample up to 1,000 sub-networks.

Dropout

- The weight scaling rule is also exact in other settings, including regression networks with conditionally normal outputs, and deep networks that have hidden layers without nonlinearities.
- However, the weight scaling rule is only a approximation for deep models that have nonlinearities.
- Though the approximation has not been theoretically characterized, it often works well, empirically.
- Goodfellow et al. [2013b] found experimentally that the weight scaling approximation can work better (in terms of classification accuracy) than Monte Carlo approximations to the ensemble predictor.
- This held true even when the Monte Carlo approximation was allowed to sample up to 1,000 sub-networks.

Dropout

- Gal and Ghahramani [2015] found that some models obtain better classification accuracy using twenty samples and the Monte Carlo approximation. It appears that the optimal choice of inference approximation is problem-depend.
- Srivastava et al. [2014] showed that dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, filter norm constraints and sparse activity regularization.
- One advantage of dropout is that it is very computationally cheap.
- Another significant advantage of dropout is that it does not significantly limit the type of model or training procedure that can be used.
- This includes feedforward neural networks, probabilistic models such as restricted Boltzmann machines (Srivastava et al. [2014]), and recurrent neural networks (Bayer and Osendorfer [2014], Pascanu et al. [2013]).

Dropout

- Gal and Ghahramani [2015] found that some models obtain better classification accuracy using twenty samples and the Monte Carlo approximation. It appears that the optimal choice of inference approximation is problem-depend.
- Srivastava et al. [2014] showed that dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, filter norm constraints and sparse activity regularization.
- One advantage of dropout is that it is very computationally cheap.
- Another significant advantage of dropout is that it does not significantly limit the type of model or training procedure that can be used.
- This includes feedforward neural networks, probabilistic models such as restricted Boltzmann machines (Srivastava et al. [2014]), and recurrent neural networks (Bayer and Osendorfer [2014], Pascanu et al. [2013]).

Dropout

- Gal and Ghahramani [2015] found that some models obtain better classification accuracy using twenty samples and the Monte Carlo approximation. It appears that the optimal choice of inference approximation is problem-depend.
- Srivastava et al. [2014] showed that dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, filter norm constraints and sparse activity regularization.
- One advantage of dropout is that it is very computationally cheap.
- Another significant advantage of dropout is that it does not significantly limit the type of model or training procedure that can be used.
- This includes feedforward neural networks, probabilistic models such as restricted Boltzmann machines (Srivastava et al. [2014]), and recurrent neural networks (Bayer and Osendorfer [2014], Pascanu et al. [2013]).

Dropout

- Gal and Ghahramani [2015] found that some models obtain better classification accuracy using twenty samples and the Monte Carlo approximation. It appears that the optimal choice of inference approximation is problem-depend.
- Srivastava et al. [2014] showed that dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, filter norm constraints and sparse activity regularization.
- One advantage of dropout is that it is very computationally cheap.
- Another significant advantage of dropout is that it does not significantly limit the type of model or training procedure that can be used.
- This includes feedforward neural networks, probabilistic models such as restricted Boltzmann machines (Srivastava et al. [2014]), and recurrent neural networks (Bayer and Osendorfer [2014], Pascanu et al. [2013]).

Dropout

- Gal and Ghahramani [2015] found that some models obtain better classification accuracy using twenty samples and the Monte Carlo approximation. It appears that the optimal choice of inference approximation is problem-depend.
- Srivastava et al. [2014] showed that dropout is more effective than other standard computationally inexpensive regularizers, such as weight decay, filter norm constraints and sparse activity regularization.
- One advantage of dropout is that it is very computationally cheap.
- Another significant advantage of dropout is that it does not significantly limit the type of model or training procedure that can be used.
- This includes feedforward neural networks, probabilistic models such as restricted Boltzmann machines (Srivastava et al. [2014]), and recurrent neural networks (Bayer and Osendorfer [2014], Pascanu et al. [2013]).

Dropout

- Wager et al. [2013] showed that, when applied to linear regression, dropout is equivalent to L^2 weight decay, with a different weight decay coefficient for each input feature.
- The magnitude of each feature's weight decay coefficient is determined by its variance.
- Similar results hold for other linear models.
- For deep models, dropout is not equivalent to weight decay.

Dropout

- Wager et al. [2013] showed that, when applied to linear regression, dropout is equivalent to L^2 weight decay, with a different weight decay coefficient for each input feature.
- The magnitude of each feature's weight decay coefficient is determined by its variance.
- Similar results hold for other linear models.
- For deep models, dropout is not equivalent to weight decay.

Dropout

- Wager et al. [2013] showed that, when applied to linear regression, dropout is equivalent to L^2 weight decay, with a different weight decay coefficient for each input feature.
- The magnitude of each feature's weight decay coefficient is determined by its variance.
- Similar results hold for other linear models.
- For deep models, dropout is not equivalent to weight decay.

Dropout

- Wager et al. [2013] showed that, when applied to linear regression, dropout is equivalent to L^2 weight decay, with a different weight decay coefficient for each input feature.
- The magnitude of each feature's weight decay coefficient is determined by its variance.
- Similar results hold for other linear models.
- For deep models, dropout is not equivalent to weight decay.

Dropout

- The power of dropout arises from the fact that the masking noise is applied to the hidden units.
- This can be seen as a form of highly intelligent, adaptive destruction of the information content of the input rather than destruction of the raw values of the input.
- For example, if the model learns a hidden unit h_i that detects a face by finding the nose, then dropping h_i corresponds to erasing the information that there is a nose in the image.
- The model must learn another h_i, either that redundantly encodes the presence of a nose, or that detects the face by another feature, such as the mouth.
- Destroying extracted features rather than original values allows the destruction process to make use of all of the knowledge about the input distribution that the model has acquired so far.

Dropout

- The power of dropout arises from the fact that the masking noise is applied to the hidden units.
- This can be seen as a form of highly intelligent, adaptive destruction of the information content of the input rather than destruction of the raw values of the input.
- For example, if the model learns a hidden unit h_i that detects a face by finding the nose, then dropping h_i corresponds to erasing the information that there is a nose in the image.
- The model must learn another h_i, either that redundantly encodes the presence of a nose, or that detects the face by another feature, such as the mouth.
- Destroying extracted features rather than original values allows the destruction process to make use of all of the knowledge about the input distribution that the model has acquired so far.

Dropout

- The power of dropout arises from the fact that the masking noise is applied to the hidden units.
- This can be seen as a form of highly intelligent, adaptive destruction of the information content of the input rather than destruction of the raw values of the input.
- For example, if the model learns a hidden unit h_i that detects a face by finding the nose, then dropping h_i corresponds to erasing the information that there is a nose in the image.
- The model must learn another h_i, either that redundantly encodes the presence of a nose, or that detects the face by another feature, such as the mouth.
- Destroying extracted features rather than original values allows the destruction process to make use of all of the knowledge about the input distribution that the model has acquired so far.

Dropout

- The power of dropout arises from the fact that the masking noise is applied to the hidden units.
- This can be seen as a form of highly intelligent, adaptive destruction of the information content of the input rather than destruction of the raw values of the input.
- For example, if the model learns a hidden unit h_i that detects a face by finding the nose, then dropping h_i corresponds to erasing the information that there is a nose in the image.
- The model must learn another h_i, either that redundantly encodes the presence of a nose, or that detects the face by another feature, such as the mouth.
- Destroying extracted features rather than original values allows the destruction process to make use of all of the knowledge about the input distribution that the model has acquired so far.

Dropout

- The power of dropout arises from the fact that the masking noise is applied to the hidden units.
- This can be seen as a form of highly intelligent, adaptive destruction of the information content of the input rather than destruction of the raw values of the input.
- For example, if the model learns a hidden unit h_i that detects a face by finding the nose, then dropping h_i corresponds to erasing the information that there is a nose in the image.
- The model must learn another h_i, either that redundantly encodes the presence of a nose, or that detects the face by another feature, such as the mouth.
- Destroying extracted features rather than original values allows the destruction process to make use of all of the knowledge about the input distribution that the model has acquired so far.

Adversarial Training

- Im many cases, neural networks have begun to reach human performance when evaluated on an i.i.d. test set. It is natural therefore to wonder whether these models have obtained a true human-level understanding of these tasks.
- In order to probe the level of understanding a network has of the underlying task, we can search for examples that the models misclassifies.
- Szegedy et al. [2013] found that even neural networks that perform at human level accuracy have a nearly 100% error rate on examples that are intentionally constructed by using an optimization procedure to search for an input x' near a data point x such that the model output is very different at x'
- In many cases, x' can be so similar to x' that a human observer cannot tell the difference between the original example and the *adversarial example*, but the network can make highly different predictions.

Adversarial Training

- Im many cases, neural networks have begun to reach human performance when evaluated on an i.i.d. test set. It is natural therefore to wonder whether these models have obtained a true human-level understanding of these tasks.
- In order to probe the level of understanding a network has of the underlying task, we can search for examples that the models misclassifies.
- Szegedy et al. [2013] found that even neural networks that perform at human level accuracy have a nearly 100% error rate on examples that are intentionally constructed by using an optimization procedure to search for an input x' near a data point x such that the model output is very different at x'
- In many cases, x' can be so similar to x' that a human observer cannot tell the difference between the original example and the *adversarial example*, but the network can make highly different predictions.

Adversarial Training

- Im many cases, neural networks have begun to reach human performance when evaluated on an i.i.d. test set. It is natural therefore to wonder whether these models have obtained a true human-level understanding of these tasks.
- In order to probe the level of understanding a network has of the underlying task, we can search for examples that the models misclassifies.
- Szegedy et al. [2013] found that even neural networks that perform at human level accuracy have a nearly 100% error rate on examples that are intentionally constructed by using an optimization procedure to search for an input x' near a data point x such that the model output is very different at x'
- In many cases, x' can be so similar to x' that a human observer cannot tell the difference between the original example and the *adversarial example*, but the network can make highly different predictions.

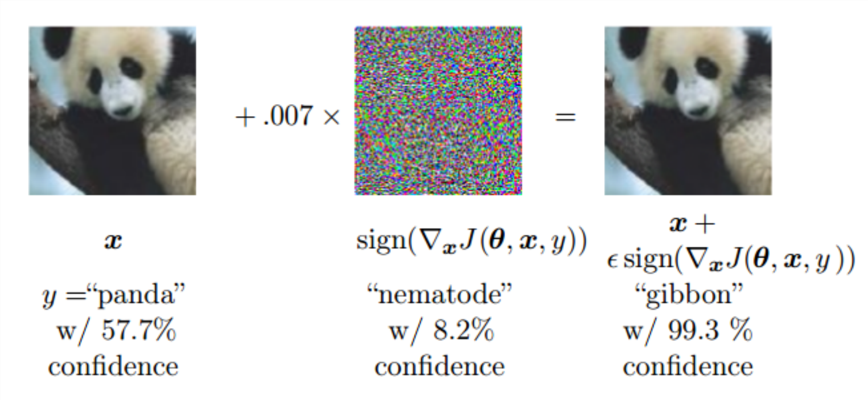
Adversarial Training

- Im many cases, neural networks have begun to reach human performance when evaluated on an i.i.d. test set. It is natural therefore to wonder whether these models have obtained a true human-level understanding of these tasks.
- In order to probe the level of understanding a network has of the underlying task, we can search for examples that the models misclassifies.
- Szegedy et al. [2013] found that even neural networks that perform at human level accuracy have a nearly 100% error rate on examples that are intentionally constructed by using an optimization procedure to search for an input x' near a data point x such that the model output is very different at x'
- In many cases, x' can be so similar to x' that a human observer cannot tell the difference between the original example and the *adversarial example*, but the network can make highly different predictions.

Adversarial Training

- Im many cases, neural networks have begun to reach human performance when evaluated on an i.i.d. test set. It is natural therefore to wonder whether these models have obtained a true human-level understanding of these tasks.
- In order to probe the level of understanding a network has of the underlying task, we can search for examples that the models misclassifies.
- Szegedy et al. [2013] found that even neural networks that perform at human level accuracy have a nearly 100% error rate on examples that are intentionally constructed by using an optimization procedure to search for an input x' near a data point x such that the model output is very different at x'
- In many cases, x' can be so similar to x' that a human observer cannot tell the difference between the original example and the *adversarial example*, but the network can make highly different predictions.

Adversarial Training



Adversarial Training

- Adversarial examples are interesting in the context of regularization because one can reduce the error rate on the original i.i.d. test set via *adversarial training*– training on adversarially perturbed examples from from the training set.
- Goodfellow et al. [2013a] showed that one of the primary causes of these adversarial examples is excessive linearity.
- Neural networks are built out of primarily linear building blocks.
- In some experiments the overall function they implement proves to be highly linear as a result.
- Unfortunately, the value of a linear function can change very rapidly if it has numerous inputs.
- If we change each input by ϵ, then a linear function with weights w can change by as much as $\epsilon||w||_1$, which can be a very large amount if w is high-dimensional.

Adversarial Training

- Adversarial examples are interesting in the context of regularization because one can reduce the error rate on the original i.i.d. test set via *adversarial training*– training on adversarially perturbed examples from from the training set.
- Goodfellow et al. [2013a] showed that one of the primary causes of these adversarial examples is excessive linearity.
- Neural networks are built out of primarily linear building blocks.
- In some experiments the overall function they implement proves to be highly linear as a result.
- Unfortunately, the value of a linear function can change very rapidly if it has numerous inputs.
- If we change each input by ϵ, then a linear function with weights w can change by as much as $\epsilon||w||_1$, which can be a very large amount if w is high-dimensional.

Adversarial Training

- Adversarial examples are interesting in the context of regularization because one can reduce the error rate on the original i.i.d. test set via *adversarial training*– training on adversarially perturbed examples from from the training set.
- Goodfellow et al. [2013a] showed that one of the primary causes of these adversarial examples is excessive linearity.
- Neural networks are built out of primarily linear building blocks.
- In some experiments the overall function they implement proves to be highly linear as a result.
- Unfortunately, the value of a linear function can change very rapidly if it has numerous inputs.
- If we change each input by ϵ, then a linear function with weights w can change by as much as $\epsilon||w||_1$, which can be a very large amount if w is high-dimensional.

Adversarial Training

- Adversarial examples are interesting in the context of regularization because one can reduce the error rate on the original i.i.d. test set via *adversarial training*– training on adversarially perturbed examples from from the training set.
- Goodfellow et al. [2013a] showed that one of the primary causes of these adversarial examples is excessive linearity.
- Neural networks are built out of primarily linear building blocks.
- In some experiments the overall function they implement proves to be highly linear as a result.
- Unfortunately, the value of a linear function can change very rapidly if it has numerous inputs.
- If we change each input by ϵ, then a linear function with weights w can change by as much as $\epsilon||w||_1$, which can be a very large amount if w is high-dimensional.

Adversarial Training

- Adversarial examples are interesting in the context of regularization because one can reduce the error rate on the original i.i.d. test set via *adversarial training*– training on adversarially perturbed examples from from the training set.
- Goodfellow et al. [2013a] showed that one of the primary causes of these adversarial examples is excessive linearity.
- Neural networks are built out of primarily linear building blocks.
- In some experiments the overall function they implement proves to be highly linear as a result.
- Unfortunately, the value of a linear function can change very rapidly if it has numerous inputs.
- If we change each input by ϵ, then a linear function with weights $\boldsymbol{w}$ can change by as much as $\epsilon||\boldsymbol{w}||_1$, which can be a very large amount if $\boldsymbol{w}$ is high-dimensional.

Adversarial Training

- Adversarial examples are interesting in the context of regularization because one can reduce the error rate on the original i.i.d. test set via *adversarial training*– training on adversarially perturbed examples from from the training set.
- Goodfellow et al. [2013a] showed that one of the primary causes of these adversarial examples is excessive linearity.
- Neural networks are built out of primarily linear building blocks.
- In some experiments the overall function they implement proves to be highly linear as a result.
- Unfortunately, the value of a linear function can change very rapidly if it has numerous inputs.
- If we change each input by ϵ, then a linear function with weights w can change by as much as $\epsilon\|w\|_1$, which can be a very large amount if w is high-dimensional.

Adversarial Training

- Adversarial training discourages this highly sensitive locally linear behavior by encouraging the network to be locally constant in the neighborhood of the training data.
- This can be seen as a way of explicitly introducing a local constancy prior into supervised neural networks.

Adversarial Training

- Adversarial training discourages this highly sensitive locally linear behavior by encouraging the network to be locally constant in the neighborhood of the training data.
- This can be seen as a way of explicitly introducing a local constancy prior into supervised neural networks.

Adversarial Training

- Adversarial training helps to illustrate the power of using a large function family in combination with aggressive regularization.
- Purely linear models, like logistic regression, are not able to resist adversarial examples because they are forced to be linear.
- Neural networks are able to represent functions that can range from nearly linear to nearly locally constant and thus have the flexibility to capture linear trends in the training data while still learning to resist local perturbation.

Adversarial Training

- Adversarial training helps to illustrate the power of using a large function family in combination with aggressive regularization.
- Purely linear models, like logistic regression, are not able to resist adversarial examples because they are forced to be linear.
- Neural networks are able to represent functions that can range from nearly linear to nearly locally constant and thus have the flexibility to capture linear trends in the training data while still learning to resist local perturbation.

Adversarial Training

- Adversarial training helps to illustrate the power of using a large function family in combination with aggressive regularization.
- Purely linear models, like logistic regression, are not able to resist adversarial examples because they are forced to be linear.
- Neural networks are able to represent functions that can range from nearly linear to nearly locally constant and thus have the flexibility to capture linear trends in the training data while still learning to resist local perturbation.

Adversarial Training

- Adversarial examples also provide a means of accomplishing semi-supervised learning.
- At a point $\boldsymbol{x}$ that is not associated with a label in the dataset, the model itself assigns some label $\hat{y}$.
- The model's label $\hat{y}$ may not be the true label, but if the model is high quality, then $\hat{y}$ has a high probability of providing the true label.
- We can seek an adversarial example $\boldsymbol{x}'$ that causes the classifier to output a label y' with $y' \neq \hat{y}$.
- The classifier may then be trained to assign the same label to $\boldsymbol{x}$ and $\boldsymbol{x}'$.
- Adversarial examples generated using not the true label but a label provided by a trained model are called *virtual adversarial examples*.
- This encourages the classifier to learn a function that is robust to small changes.

Adversarial Training

- Adversarial examples also provide a means of accomplishing semi-supervised learning.
- At a point $\boldsymbol{x}$ that is not associated with a label in the dataset, the model itself assigns some label $\hat{y}$.
- The model's label $\hat{y}$ may not be the true label, but if the model is high quality, then $\hat{y}$ has a high probability of providing the true label.
- We can seek an adversarial example $\boldsymbol{x}'$ that causes the classifier to output a label y' with $y' \neq \hat{y}$.
- The classifier may then be trained to assign the same label to $\boldsymbol{x}$ and $\boldsymbol{x}'$.
- Adversarial examples generated using not the true label but a label provided by a trained model are called *virtual adversarial examples*.
- This encourages the classifier to learn a function that is robust to small changes.

Adversarial Training

- Adversarial examples also provide a means of accomplishing semi-supervised learning.
- At a point $\boldsymbol{x}$ that is not associated with a label in the dataset, the model itself assigns some label $\hat{y}$.
- The model's label $\hat{y}$ may not be the true label, but if the model is high quality, then $\hat{y}$ has a high probability of providing the true label.
- We can seek an adversarial example $\boldsymbol{x}'$ that causes the classifier to output a label y' with $y' \neq \hat{y}$.
- The classifier may then be trained to assign the same label to $\boldsymbol{x}$ and $\boldsymbol{x}'$.
- Adversarial examples generated using not the true label but a label provided by a trained model are called *virtual adversarial examples*.
- This encourages the classifier to learn a function that is robust to small changes.

Adversarial Training

- Adversarial examples also provide a means of accomplishing semi-supervised learning.
- At a point $\boldsymbol{x}$ that is not associated with a label in the dataset, the model itself assigns some label $\hat{y}$.
- The model's label $\hat{y}$ may not be the true label, but if the model is high quality, then $\hat{y}$ has a high probability of providing the true label.
- We can seek an adversarial example $\boldsymbol{x}'$ that causes the classifier to output a label y' with $y' \neq \hat{y}$.
- The classifier may then be trained to assign the same label to $\boldsymbol{x}$ and $\boldsymbol{x}'$.
- Adversarial examples generated using not the true label but a label provided by a trained model are called *virtual adversarial examples*.
- This encourages the classifier to learn a function that is robust to small changes.

Adversarial Training

- Adversarial examples also provide a means of accomplishing semi-supervised learning.
- At a point $\boldsymbol{x}$ that is not associated with a label in the dataset, the model itself assigns some label $\hat{y}$.
- The model's label $\hat{y}$ may not be the true label, but if the model is high quality, then $\hat{y}$ has a high probability of providing the true label.
- We can seek an adversarial example $\boldsymbol{x}'$ that causes the classifier to output a label y' with $y' \neq \hat{y}$.
- The classifier may then be trained to assign the same label to $\boldsymbol{x}$ and $\boldsymbol{x}'$.
- Adversarial examples generated using not the true label but a label provided by a trained model are called *virtual adversarial examples*.
- This encourages the classifier to learn a function that is robust to small changes.

Adversarial Training

- Adversarial examples also provide a means of accomplishing semi-supervised learning.
- At a point $\boldsymbol{x}$ that is not associated with a label in the dataset, the model itself assigns some label $\hat{y}$.
- The model's label $\hat{y}$ may not be the true label, but if the model is high quality, then $\hat{y}$ has a high probability of providing the true label.
- We can seek an adversarial example $\boldsymbol{x}'$ that causes the classifier to output a label y' with $y' \neq \hat{y}$.
- The classifier may then be trained to assign the same label to $\boldsymbol{x}$ and $\boldsymbol{x}'$.
- Adversarial examples generated using not the true label but a label provided by a trained model are called *virtual adversarial examples*.
- This encourages the classifier to learn a function that is robust to small changes.

Adversarial Training

- Adversarial examples also provide a means of accomplishing semi-supervised learning.
- At a point $\boldsymbol{x}$ that is not associated with a label in the dataset, the model itself assigns some label $\hat{y}$.
- The model's label $\hat{y}$ may not be the true label, but if the model is high quality, then $\hat{y}$ has a high probability of providing the true label.
- We can seek an adversarial example $\boldsymbol{x}'$ that causes the classifier to output a label y' with $y' \neq \hat{y}$.
- The classifier may then be trained to assign the same label to $\boldsymbol{x}$ and $\boldsymbol{x}'$.
- Adversarial examples generated using not the true label but a label provided by a trained model are called *virtual adversarial examples*.
- This encourages the classifier to learn a function that is robust to small changes.

Tangent Distance, Tangent Prop, and Manifold Tangent Classifier

- The *tangent prop* algorithm (Simard et al. [1991]) trains a neural net classifier with an extra penalty to make each output $f(\boldsymbol{x})$ of the neural net locally invariant to known factors of variation.
- The directional derivative of f at $\boldsymbol{x}$ in the directions $\boldsymbol{v}^{(i)}$ be small by adding a regularization penalty Ω:

$$\Omega(f) = \sum_i \left((\nabla_x f(x))^2 \, v^{(i)} \right)^2$$

- The *tangent prop* algorithm (Simard et al. [1991]) trains a neural net classifier with an extra penalty to make each output $f(\boldsymbol{x})$ of the neural net locally invariant to known factors of variation.
- The directional derivative of f at $\boldsymbol{x}$ in the directions $\boldsymbol{v}^{(i)}$ be small by adding a regularization penalty Ω:

$$\Omega(f) = \sum_i \left((\nabla_{\boldsymbol{x}} f(\boldsymbol{x}))^2 \, \boldsymbol{v}^{(i)} \right)^2$$

References

Justin Bayer and Christian Osendorfer. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

Yoshua Bengio, Nicolas Le Roux, Pascal Vincent, Olivier Delalleau, and Patrice Marcotte. Convex neural networks. *Advances in neural information processing systems*, 18:123, 2006.

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

Chris M Bishop. Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116, 1995a.

Christopher M Bishop. Regularization and complexity control in feed-forward networks. 1995b.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Adam Coates, Honglak Lee, and Andrew Y Ng. An analysis of single-layer networks in unsupervised feature learning. *Ann Arbor*, 1001(48109):2, 2010.

Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156, 1996.

Yarin Gal and Zoubin Ghahramani. Bayesian convolutional neural networks with bernoulli approximate variational inference. *arXiv preprint arXiv:1506.02158*, 2015.

Ian Goodfellow, Honglak Lee, Quoc V Le, Andrew Saxe, and Andrew Y Ng. Measuring invariances in deep networks. In *Advances in neural information processing systems*, pages 646–654, 2009.

Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013a.

Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C Courville, and Yoshua Bengio. Maxout networks. *ICML (3)*, 28:1319–1327, 2013b.

Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Using deep belief nets to learn covariance kernels for gaussian processes. In *Advances in neural information processing systems*, pages 1249–1256, 2008.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Sepp Hochreiter, Jürgen Schmidhuber, et al. Simplifying neural nets by discovering flat minima. *Advances in Neural Information Processing Systems*, pages 529–536, 1995.

Navdeep Jaitly and Geoffrey E Hinton. Vocal tract length perturbation (vtlp) improves speech recognition. In *Proc. ICML Workshop on Deep Learning for Audio, Speech and Language*, 2013.

Kam-Chuen Jim, C Lee Giles, and Bill G Horne. An analysis of noise in recurrent neural networks: Convergence and generalization. *IEEE Transactions on neural networks*, 7(6):1424–1438, 1996.

Hugo Larochelle and Yoshua Bengio. Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th international conference on Machine learning*, pages 536–543. ACM, 2008.

Julia A Lasserre, Christopher M Bishop, and Thomas P Minka. Principled hybrids of generative and discriminative models. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 87–94. IEEE, 2006.

Honglak Lee, Chaitanya Ekanadham, and Andrew Y Ng. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pages 873–880, 2008.

Bruno A Olshausen and David J Field. How close are we to understanding v1? *Neural computation*, 17(8):1665–1699, 2005.

Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

Yagyensh Chandra Pati, Ramin Rezaiifar, and PS Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 40–44. IEEE, 1993.

Holger Schwenk and Yoshua Bengio. Training methods for adaptive boosting of neural networks. *Advances in Neural Information Processing Systems*, pages 647–653, 1998.

Jocelyn Sietsma and Robert JF Dow. Creating artificial neural networks that generalize. *Neural networks*, 4(1):67–79, 1991.

Patrice Simard, Bernard Victorri, Yann LeCun, and John S Denker. Tangent prop-a formalism for specifying selected invariances in an adaptive network. In *NIPS*, volume 91, pages 895–903, 1991.

Jonas Sjöberg and Lennart Ljung. Overtraining, regularization and searching for a minimum, with application to neural networks. *International Journal of Control*, 62(6):1391–1407, 1995.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

Stefan Wager, Sida Wang, and Percy S Liang. Dropout training as adaptive regularization. In *Advances in neural information processing systems*, pages 351–359, 2013.

David Warde-Farley, Ian J Goodfellow, Aaron Courville, and Yoshua Bengio. An empirical analysis of dropout in piecewise linear networks. *arXiv preprint arXiv:1312.6197*, 2013.