EE 628
Deep Learning
Fall 2019

Lecture 5

02/20/2019

Sergul Aydore

Applied Scientist

Amazon Web Services

Overview

- Last lecture we covered
    - Multilayer Perceptron

- Today, we will cover
    - Overfitting/underfitting
    - Backpropagation

Model Selection, Underfitting, Overfitting

- As machine learning scientists, our goal is to discover general patterns.

Model Selection, Underfitting, Overfitting

- As machine learning scientists, our goal is to discover general patterns.
- Not to memorize our training set!

Model Selection, Underfitting, Overfitting

- As machine learning scientists, our goal is to discover general patterns.
- Not to memorize our training set!
- How to discover patters that **generalize** is the fundamental problem of machine learning.

Model Selection, Underfitting, Overfitting

- As machine learning scientists, our goal is to discover general patterns.
- Not to memorize our training set!
- How to discover patters that **generalize** is the fundamental problem of machine learning.
- The danger is that when we train models, we access just a small sample of data.

Model Selection, Underfitting, Overfitting

- As machine learning scientists, our goal is to discover general patterns.
- Not to memorize our training set!
- How to discover patters that **generalize** is the fundamental problem of machine learning.
- The danger is that when we train models, we access just a small sample of data.
- The phenomena of fitting our training data more closely than we fit the underlying distribution is called **overfitting**.

Model Selection, Underfitting, Overfitting

- As machine learning scientists, our goal is to discover general patterns.
- Not to memorize our training set!
- How to discover patters that **generalize** is the fundamental problem of machine learning.
- The danger is that when we train models, we access just a small sample of data.
- The phenomena of fitting our training data more closely than we fit the underlying distribution is called **overfitting**.
- The techniques used to combat overfitting are called **generalization**.

Training Error and Generalization Error

- The training error is the error of our model as calculated on the training data set.

Training Error and Generalization Error

- The training error is the error of our model as calculated on the training data set.
- The generalization error is the expectation of our model's error were we to apply it to an infinite data points from the same underlying distribution as our original sample.

Training Error and Generalization Error

- The training error is the error of our model as calculated on the training data set.
- The generalization error is the expectation of our model's error were we to apply it to an infinite data points from the same underlying distribution as our original sample.
- In practice, we must estimate the generalization error by applying our model to an independent test set that were withheld from our training set.

Training Error and Generalization Error

- The training error is the error of our model as calculated on the training data set.
- The generalization error is the expectation of our model's error were we to apply it to an infinite data points from the same underlying distribution as our original sample.
- In practice, we must estimate the generalization error by applying our model to an independent test set that were withheld from our training set.
- Example: a student memorizing all exam questions to prepare an exam in future

Statistical Learning Theory

- In the standard supervised learning setting, we assume that both the training data and the test data are drawn *independently* from *identical distributions*.

Statistical Learning Theory

- In the standard supervised learning setting, we assume that both the training data and the test data are drawn *independently* from *identical distributions*.
- This is called the i.i.d assumption.

Statistical Learning Theory

- In the standard supervised learning setting, we assume that both the training data and the test data are drawn *independently* from *identical distributions*.
- This is called the i.i.d assumption.
- Sometimes we can get away with minor violations of the i.i.d assumption

Statistical Learning Theory

- In the standard supervised learning setting, we assume that both the training data and the test data are drawn *independently* from *identical distributions*.
- This is called the i.i.d assumption.
- Sometimes we can get away with minor violations of the i.i.d assumption
- But some violations can cause trouble

Factors that tend to influence the generalizability

- The number of tunable parameters
  - When the number of tunable parameters, sometimes called the degrees of freedom, is large, models tend to be more susceptible to overfitting.

Factors that tend to influence the generalizability

- The number of tunable parameters
  - When the number of tunable parameters, sometimes called the degrees of freedom, is large, models tend to be more susceptible to overfitting.
- The values taken by the parameters.
  - When weights can take a wider range of values, models can be more susceptible to over fitting.

Factors that tend to influence the generalizability

- The number of tunable parameters
  - When the number of tunable parameters, sometimes called the degrees of freedom, is large, models tend to be more susceptible to overfitting.
- The values taken by the parameters.
  - When weights can take a wider range of values, models can be more susceptible to over fitting.
- The number of training examples.
  - It’s trivially easy to overfit a dataset containing only one or two examples even if your model is simple

Model Selection

- In machine learning, we usually select our final model after evaluating several candidate models.

Model Selection

- In machine learning, we usually select our final model after evaluating several candidate models.
- In order to determine the best among our candidate models, we will typically employ a validation set.

Model Selection

- In machine learning, we usually select our final model after evaluating several candidate models.
- In order to determine the best among our candidate models, we will typically employ a validation set.
- Validation Set:
  - We should never rely on the test data for model selection
  - And yet we cannot rely solely on the training data for model selection
  - Solution: split data in three ways: training, testing and validation

Model Selection

- In machine learning, we usually select our final model after evaluating several candidate models.
- In order to determine the best among our candidate models, we will typically employ a validation set.
- Validation Set:
  - We should never rely on the test data for model selection
  - And yet we cannot rely solely on the training data for model selection
  - Solution: split data in three ways: training, testing and validation
- What if we cannot afford to holdout enough data
  - Solution: K-fold Cross Validation

Underfitting or Overfitting

- Underfitting:
  - when our training error and validation error are both substantial but there is a little gap between them
  - If the model is unable to reduce the training error, that could mean that our model is too simple
  - since the generalization gap between our training and validation errors is small, we have reason to believe that we could get away with a more complex model.

Underfitting or Overfitting

- Underfitting:
  - when our training error and validation error are both substantial but there is a little gap between them
  - If the model is unable to reduce the training error, that could mean that our model is too simple
  - since the generalization gap between our training and validation errors is small, we have reason to believe that we could get away with a more complex model.
- Overfitting:
  - when our training error is significantly lower than our validation error

Underfitting or Overfitting

- Underfitting:
  - when our training error and validation error are both substantial but there is a little gap between them
  - If the model is unable to reduce the training error, that could mean that our model is too simple
  - since the generalization gap between our training and validation errors is small, we have reason to believe that we could get away with a more complex model.
- Overfitting:
  - when our training error is significantly lower than our validation error
- Whether we overfit or underfit can depend both on the complexity of our model and the size of the available training datasets

Model Complexity

- An example using polynomials $\hat{y} = \sum_{i=0}^{d} x^i w_i$

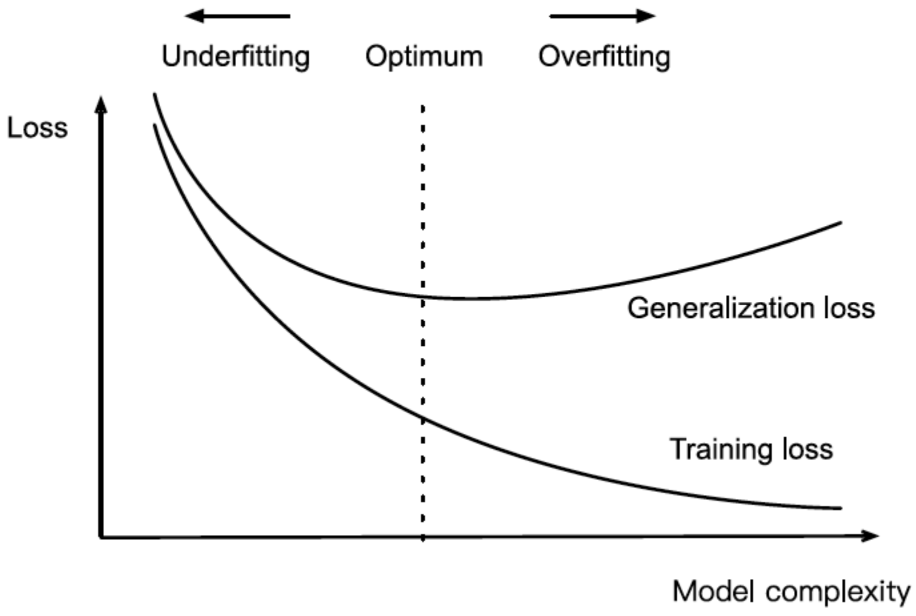


Fig. 6.4.1: Influence of Model Complexity on Underfitting and Overfitting

Data Set Size

- The fewer samples we have in the training dataset, the more likely (and more severely) we are to encounter overfitting.

Data Set Size

- The fewer samples we have in the training dataset, the more likely (and more severely) we are to encounter overfitting.
- As we increase the amount of training data, the generalization error typically decreases.

Data Set Size

- The fewer samples we have in the training dataset, the more likely (and more severely) we are to encounter overfitting.
- As we increase the amount of training data, the generalization error typically decreases.
- Given more data, we might profitably attempt to fit a more complex

model.

Data Set Size

- The fewer samples we have in the training dataset, the more likely (and more severely) we are to encounter overfitting.
- As we increase the amount of training data, the generalization error typically decreases.
- Given more data, we might profitably attempt to fit a more complex

model.

- In part, the current success of deep learning owes to the current abundance of massive datasets due to internet companies, cheap storage, connected devices, and the broad digitization of the economy.

Weight Decay (ℓ_2 regularization)

- Probably the most widely-used technique for regularization

Weight Decay (ℓ_2 regularization)

- Probably the most widely-used technique for regularization
- Intuition: among all functions f, the function $f = 0$ is the simplest.

Weight Decay (ℓ_2 regularization)

- Probably the most widely-used technique for regularization
- Intuition: among all functions f, the function $f = 0$ is the simplest.
- We can measure functions by their proximity to 0.

Weight Decay (ℓ_2 regularization)

- Probably the most widely-used technique for regularization
- Intuition: among all functions f, the function $f = 0$ is the simplest.
- We can measure functions by their proximity to 0.
- Consider a linear function $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$

Weight Decay (ℓ_2 regularization)

- Probably the most widely-used technique for regularization
- Intuition: among all functions f, the function $f = 0$ is the simplest.
- We can measure functions by their proximity to 0.
- Consider a linear function $f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$
- f is simple if its weight vector is small

Weight Decay (ℓ_2 regularization)

- Probably the most widely-used technique for regularization
- Intuition: among all functions f, the function $f = 0$ is the simplest.
- We can measure functions by their proximity to 0.
- Consider a linear function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$
- f is simple if its weight vector is small
- We can measure this via $||\mathbf{w}||^2$

Weight Decay (ℓ_2 regularization)

- Probably the most widely-used technique for regularization
- Intuition: among all functions f, the function $f = 0$ is the simplest.
- We can measure functions by their proximity to 0.
- Consider a linear function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$
- f is simple if its weight vector is small
- We can measure this via $||\mathbf{w}||^2$
- We can add this as penalty term to our loss function

Weight Decay (ℓ_2 regularization)

- Probably the most widely-used technique for regularization
- Intuition: among all functions f, the function $f = 0$ is the simplest.
- We can measure functions by their proximity to 0.
- Consider a linear function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$
- f is simple if its weight vector is small
- We can measure this via $||\mathbf{w}||^2$
- We can add this as penalty term to our loss function
- For linear regression problem, our loss becomes:

$$l(\mathbf{w}, b) + \frac{\lambda}{2}\|w\|^2$$

Regularization constant $\lambda \geq 0$ governs the amount of regularization

Weight Decay (ℓ_2 regularization)

- Why not other norms?

Weight Decay (ℓ_2 regularization)

- Why not other norms?
- In fact, several other choices are valid and popular in statistics.
  - L2 regularized regression is called ridge regression
  - L1 regularized regression is called lasso regression

Weight Decay (ℓ_2 regularization)

- Why not other norms?
- In fact, several other choices are valid and popular in statistics.
  - L2 regularized regression is called ridge regression
  - L1 regularized regression is called lasso regression
- What is the SGD update for L2 regularized regression?

Dropout

- What constitutes to simple model?
  - Small number of features
  - Small norm of the basis functions
  - The function should be robust under small changes in the input

Dropout

- What constitutes to simple model?
  - Small number of features
  - Small norm of the basis functions
  - The function should be robust under small changes in the input
- In 1995, Christopher Bishop showed that training with input noise is equivalent to Tikhonov regularization

Dropout

- What constitutes to simple model?
  - Small number of features
  - Small norm of the basis functions
  - The function should be robust under small changes in the input
- In 1995, Christopher Bishop showed that training with input noise is equivalent to Tikhonov regularization
- In 2014, Siravastana applied Bishop's idea to internal layers of the network
  - Dropout – widely used in neural networks
  - On each iteration, drop out zeroes out some fraction of the nodes in each layer before calculating the subsequent layer in training

Dropout

- The key challenge: how to inject noise without introducing bias?
  - i.e. we want to perturb the inputs to each layer during training in such a way that the expected value of the layer is equal to the value it would have taken had we not introduced any noise at all.

Dropout

- The key challenge: how to inject noise without introducing bias?
  - i.e. we want to perturb the inputs to each layer during training in such a way that the expected value of the layer is equal to the value it would have taken had we not introduced any noise at all.
- Dropout at each node with probability p is defined as:

$$h' = \begin{cases} 0 & \text{with probability } p \\ \frac{h}{1-p} & \text{otherwise} \end{cases}$$

Dropout

- The key challenge: how to inject noise without introducing bias?
  - i.e. we want to perturb the inputs to each layer during training in such a way that the expected value of the layer is equal to the value it would have taken had we not introduced any noise at all.
- Dropout at each node with probability p is defined as:

$$h' = \begin{cases} 0 & \text{with probability } p \\ \frac{h}{1-p} & \text{otherwise} \end{cases}$$

- Show the expectations remain unchanged!

Dropout in Practice

- In the image below h_2 and h_5 are removed
- This way, calculation of the output cannot be overly dependent on any one element of h_1, h_2, h_3, h_4, h_5.

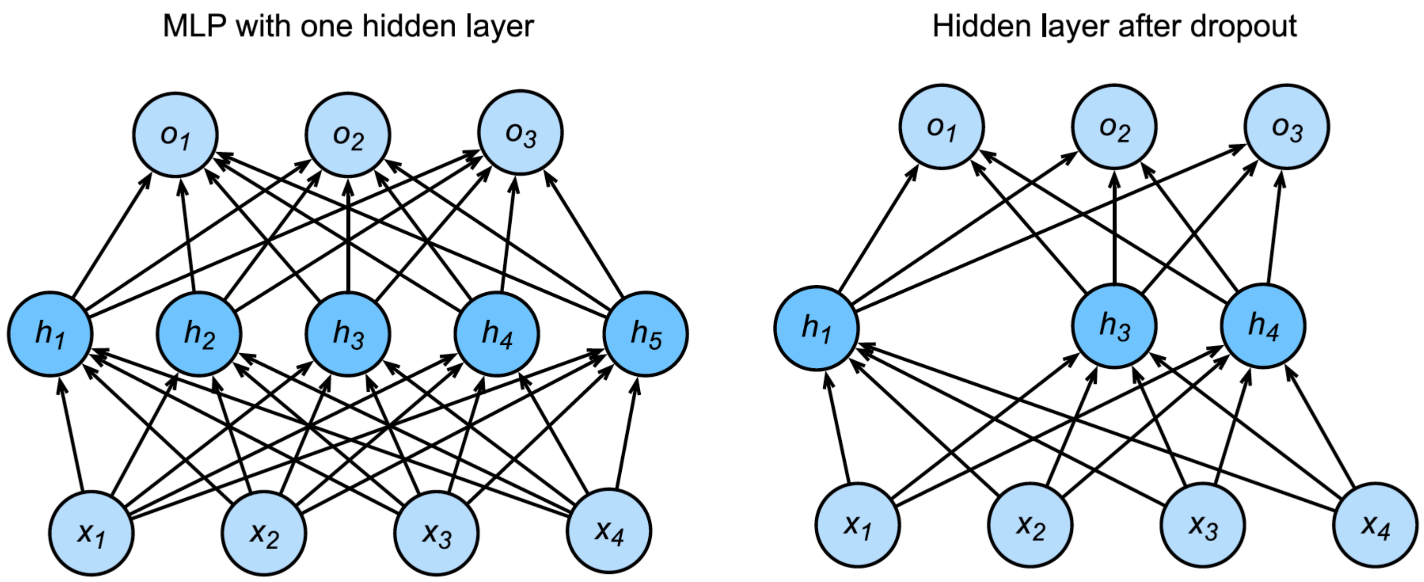


Fig. 6.6.1: MLP before and after dropout

- At test time, we typically do not use dropout.

Implementation of dropout

- From scratch
- concise

Forward Propagation, Backward Propagation, and Computational Graphs

- So far, we relied on backward function of the autograd module to compute the gradients

Forward Propagation, Backward Propagation, and Computational Graphs

- So far, we relied on backward function of the autograd module to compute the gradients
- The automatic calculation of gradients profoundly simplifies the implementation of deep learning algorithms

Forward Propagation, Backward Propagation, and Computational Graphs

- So far, we relied on backward function of the autograd module to compute the gradients
- The automatic calculation of gradients profoundly simplifies the implementation of deep learning algorithms
- Now, we will discuss some of the details of backward propagation

Forward Propagation, Backward Propagation, and Computational Graphs

- So far, we relied on backward function of the autograd module to compute the gradients
- The automatic calculation of gradients profoundly simplifies the implementation of deep learning algorithms
- Now, we will discuss some of the details of backward propagation
- To start, we will focus our exposition on a simple multilayer perceptron with
  - a single hidden layer and
  - $\ell 2$ norm regularization.

Really simple example

- We want
  - $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$
  - for $\mathrm{f(x, y, z) = (x + y)z}$
  - where $x = -2, y = 5, z = -4$
- Draw the computation graph

Reference: http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture04.pdf

Forward Propagation

- calculation and storage of intermediate variables from input layer to output layer.

Forward Propagation

- calculation and storage of intermediate variables from input layer to output layer.
- Let $\mathbf{x} \in R^d$ be the input example

Forward Propagation

- calculation and storage of intermediate variables from input layer to output layer.
- Let $\mathbf{x} \in R^d$ be the input example
- Intermediate variable is $\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$

Forward Propagation

- calculation and storage of intermediate variables from input layer to output layer.
- Let $\mathbf{x} \in R^d$ be the input example
- Intermediate variable is $\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$
- After inputting the intermediate variable into the activation function ϕ, we get $\mathbf{h} = \phi(\mathbf{z})$.

Forward Propagation

- calculation and storage of intermediate variables from input layer to output layer.
- Let $\mathbf{x} \in R^d$ be the input example
- Intermediate variable is $\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$
- After inputting the intermediate variable into the activation function ϕ, we get $\mathbf{h} = \phi(\mathbf{z})$.
- Let $\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$ be the output variable

Forward Propagation

- calculation and storage of intermediate variables from input layer to output layer.
- Let $\mathbf{x} \in R^d$ be the input example
- Intermediate variable is $\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$
- After inputting the intermediate variable into the activation function ϕ, we get $\mathbf{h} = \phi(\mathbf{z})$.
- Let $\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$ be the output variable
- Loss term for a single data example is $L = l(y, \mathbf{o})$

Forward Propagation

- calculation and storage of intermediate variables from input layer to output layer.
- Let $\mathbf{x} \in R^d$ be the input example
- Intermediate variable is $\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$
- After inputting the intermediate variable into the activation function ϕ, we get $\mathbf{h} = \phi(\mathbf{z})$.
- Let $\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$ be the output variable
- Loss term for a single data example is $L = l(y, \mathbf{o})$
- The regularization term is $s = \frac{\lambda}{2}(||\mathbf{W}^{(1)}||_F^2 + ||\mathbf{W}^{(2)}||_F^2)$

Forward Propagation

- calculation and storage of intermediate variables from input layer to output layer.
- Let $\mathbf{x} \in R^d$ be the input example
- Intermediate variable is $\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x}$
- After inputting the intermediate variable into the activation function ϕ, we get $\mathbf{h} = \phi(\mathbf{z})$.
- Let $\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}$ be the output variable
- Loss term for a single data example is $L = l(y, \mathbf{o})$
- The regularization term is $s = \frac{\lambda}{2}(||\mathbf{W}^{(1)}||_F^2 + ||\mathbf{W}^{(2)}||_F^2)$
- Finally, the model's regularized loss (*objective function*) on a given data example is $J = L + s$

Computational Graph of Forward Propagation

- Plotting computational graphs helps us visualize the dependencies of operators and variables within the calculation

Computational Graph of Forward Propagation

- Plotting computational graphs helps us visualize the dependencies of operators and variables within the calculation

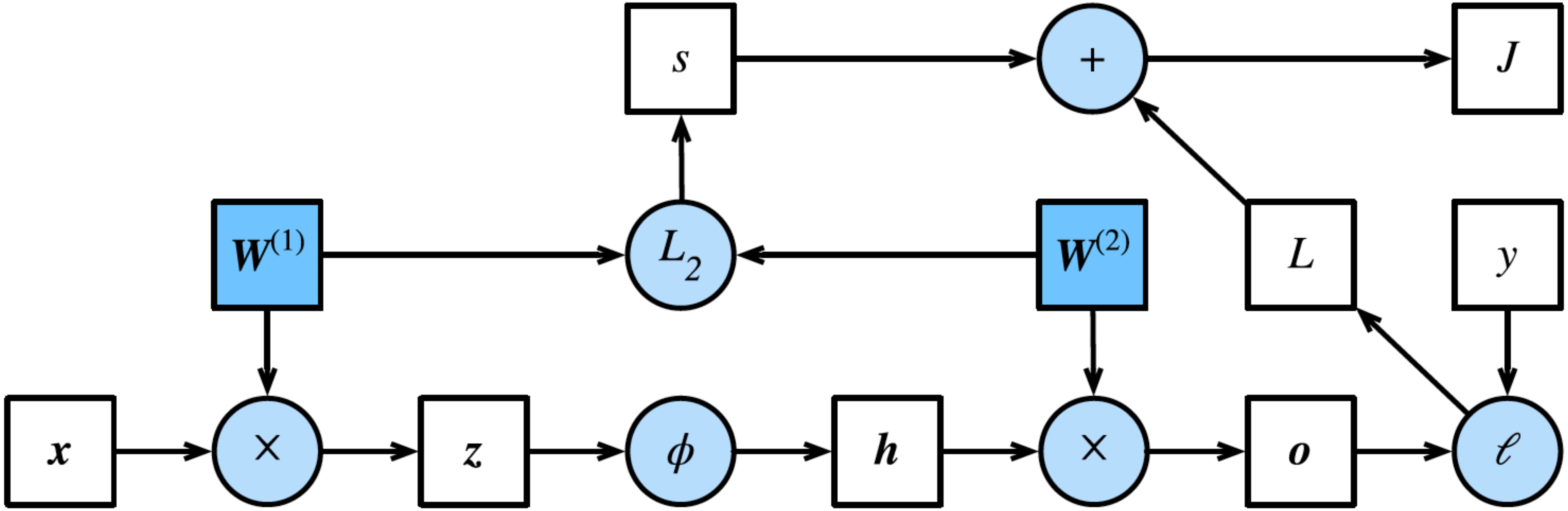


Fig. 6.7.1: Computational Graph

Backpropagation

- the method of calculating the gradient of neural network parameters

Backpropagation

- the method of calculating the gradient of neural network parameters
- in the order of the output layer to the input layer according to the **chain rule** in calculus.

Backpropagation

- the method of calculating the gradient of neural network parameters
- in the order of the output layer to the input layer according to the **chain rule** in calculus.
- What gradients do we need in our example?

Backpropagation

- $J = L + s$, then we have $\frac{\partial J}{\partial L} =?$, $\frac{\partial J}{\partial s} =?$

Backpropagation

- $J = L + s$, then we have $\frac{\partial J}{\partial L} =?$, $\frac{\partial J}{\partial s} =?$
- Next, we compute the gradient of the objective function with respect to the output layer $\frac{\partial J}{\partial \mathbf{o}} = prod\ (\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \mathbf{o}}) = \frac{\partial L}{\partial \mathbf{o}}$ (what is the size?)

Backpropagation

- $J = L + s$, then we have $\frac{\partial J}{\partial L} = ?$, $\frac{\partial J}{\partial s} = ?$
- Next, we compute the gradient of the objective function with respect to the output layer $\frac{\partial J}{\partial \mathbf{o}} = prod\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \mathbf{o}}\right) = \frac{\partial L}{\partial \mathbf{o}}$ (what is the size?)
- Next, we calculate the gradients of the regularization term $\frac{\partial s}{\partial \mathbf{W}^{(1)}} = \lambda \mathbf{W}^{(1)}$ and $\frac{\partial s}{\partial \mathbf{W}^{(2)}} = \lambda \mathbf{W}^{(2)}$.

Backpropagation

- $J = L + s$, then we have $\frac{\partial J}{\partial L} = ?$, $\frac{\partial J}{\partial s} = ?$
- Next, we compute the gradient of the objective function with respect to the output layer $\frac{\partial J}{\partial \mathbf{o}} = prod\ (\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \mathbf{o}}) = \frac{\partial L}{\partial \mathbf{o}}$ (what is the size?)
- Next, we calculate the gradients of the regularization term $\frac{\partial s}{\partial \mathbf{W}^{(1)}} = \lambda \mathbf{W}^{(1)}$ and $\frac{\partial s}{\partial \mathbf{W}^{(2)}} = \lambda \mathbf{W}^{(2)}$.
- Now we are able to calculate the gradient $\frac{\partial J}{\partial \mathbf{W}^{(2)}} = prod\left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial o}{\partial \mathbf{W}^{(2)}}\right) + prod\ \left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial \mathbf{W}^{(2)}}\right) = \frac{\partial J}{\partial \mathbf{o}} \mathbf{h}^T + \lambda \mathbf{W}^{(2)}$

Backpropagation

- $J = L + s$, then we have $\frac{\partial J}{\partial L} =?$, $\frac{\partial J}{\partial s} =?$
- Next, we compute the gradient of the objective function with respect to the output layer $\frac{\partial J}{\partial \mathbf{o}} = prod\ (\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \mathbf{o}}) = \frac{\partial L}{\partial \mathbf{o}}$ (what is the size?)
- Next, we calculate the gradients of the regularization term $\frac{\partial s}{\partial \mathbf{W}^{(1)}} = \lambda \mathbf{W}^{(1)}$ and $\frac{\partial s}{\partial \mathbf{W}^{(2)}} = \lambda \mathbf{W}^{(2)}$.
- Now we are able to calculate the gradient $\frac{\partial J}{\partial \mathbf{W}^{(2)}} = prod\left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial o}{\partial \mathbf{W}^{(2)}}\right) + prod\ \left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial \mathbf{W}^{(2)}}\right) = \frac{\partial J}{\partial \mathbf{o}} \mathbf{h}^T + \lambda \mathbf{W}^{(2)}$
- To obtain the gradient with respect to $\mathbf{W}^{(1)}$, we need to continue backpropagation along the output layer to the hidden layer $\frac{\partial J}{\partial \mathbf{h}} = prod\ \left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial \mathbf{o}}{\partial \mathbf{h}}\right) = W^{(2)T} \frac{\partial J}{\partial \mathbf{o}}$. Calculating gradient with respect to $\mathbf{z}$, requires derivative of the activation function $\frac{\partial J}{\partial z} = prod\ \left(\frac{\partial J}{\partial \mathbf{h}}, \frac{\partial \mathbf{h}}{\partial \mathbf{z}}\right) = \frac{\partial J}{\partial \mathbf{h}} \odot \phi'(\mathbf{z})$.

Backpropagation

- $J = L + s$, then we have $\frac{\partial J}{\partial L} =?$, $\frac{\partial J}{\partial s} =?$
- Next, we compute the gradient of the objective function with respect to the output layer $\frac{\partial J}{\partial \mathbf{o}} = prod\ (\frac{\partial J}{\partial L}, \frac{\partial L}{\partial \mathbf{o}}) = \frac{\partial L}{\partial \mathbf{o}}$ (what is the size?)
- Next, we calculate the gradients of the regularization term $\frac{\partial s}{\partial \mathbf{W}^{(1)}} = \lambda \mathbf{W}^{(1)}$ and $\frac{\partial s}{\partial \mathbf{W}^{(2)}} = \lambda \mathbf{W}^{(2)}$.
- Now we are able to calculate the gradient $\frac{\partial J}{\partial \mathbf{W}^{(2)}} = prod\left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial o}{\partial \mathbf{W}^{(2)}}\right) + prod\ \left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial \mathbf{W}^{(2)}}\right) = \frac{\partial J}{\partial \mathbf{o}} \mathbf{h}^T + \lambda \mathbf{W}^{(2)}$
- To obtain the gradient with respect to $\mathbf{W}^{(1)}$, we need to continue backpropagation along the output layer to the hidden layer $\frac{\partial J}{\partial \mathbf{h}} = prod\ \left(\frac{\partial J}{\partial \mathbf{o}}, \frac{\partial \mathbf{o}}{\partial \mathbf{h}}\right) = W^{(2)T} \frac{\partial J}{\partial \mathbf{o}}$. Calculating gradient with respect to $\mathbf{z}$, requires derivative of the activation function $\frac{\partial J}{\partial \mathbf{z}} = prod\ \left(\frac{\partial J}{\partial \mathbf{h}}, \frac{\partial \mathbf{h}}{\partial \mathbf{z}}\right) = \frac{\partial J}{\partial \mathbf{h}} \odot \phi'(\mathbf{z})$.
- Finally we obtain $\frac{\partial J}{\partial \mathbf{W}^{(1)}} = prod\left(\frac{\partial J}{\partial \mathbf{z}}, \frac{\partial \mathbf{z}}{\partial \mathbf{W}^{(1)}}\right) + prod\ \left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial \mathbf{W}^{(1)}}\right) = \frac{\partial J}{\partial \mathbf{z}} \mathbf{x}^T + \lambda \mathbf{W}^{(1)}$.

Training a Model

- forward and backward propagation depend on each other.

Training a Model

- forward and backward propagation depend on each other.
- for forward propagation, we traverse the compute graph in the direction of dependencies and compute all the variables on its path

Training a Model

- forward and backward propagation depend on each other.
- for forward propagation, we traverse the compute graph in the direction of dependencies and compute all the variables on its path
- These are then used for backpropagation where the compute order on the graph is reversed.

Training a Model

- forward and backward propagation depend on each other.
- for forward propagation, we traverse the compute graph in the direction of dependencies and compute all the variables on its path
- These are then used for backpropagation where the compute order on the graph is reversed.
- we need to retain the intermediate values until backpropagation is complete

Training a Model

- forward and backward propagation depend on each other.
- for forward propagation, we traverse the compute graph in the direction of dependencies and compute all the variables on its path
- These are then used for backpropagation where the compute order on the graph is reversed.
- we need to retain the intermediate values until backpropagation is complete
- This is also one of the reasons why backpropagation requires significantly more memory

Numerical Stability and Initialization

- Which nonlinearity function we use
- How we decide to initialize our parameters
- can play important role in convergence

Vanishing and Exploding Gradients

- Consider a deep neural network with d layers, input $\mathbf{x}$ and output $\mathbf{o}$.

Vanishing and Exploding Gradients

- Consider a deep neural network with d layers, input $\mathbf{x}$ and output $\mathbf{o}$.
- Each layer satisfies $\mathbf{h}^{t+1} = f_t(\mathbf{h}^t)$ and thus $\mathbf{o} = f_d \circ \ldots \circ f_1(\mathbf{x})$

Vanishing and Exploding Gradients

- Consider a deep neural network with d layers, input $\mathbf{x}$ and output $\mathbf{o}$.
- Each layer satisfies $\mathbf{h}^{t+1} = f_t(\mathbf{h}^t)$ and thus $\mathbf{o} = f_d \circ \ldots \circ f_1(\mathbf{x})$
- We can write the gradient of $\mathbf{o}$ with respect to any set of parameters $\mathbf{W}_t$ at layer t simply as

$$\partial_{\mathbf{W}_t} \mathbf{o} = \underbrace{\partial_{\mathbf{h}^{d-1}} \mathbf{h}^d}_{:=\mathbf{M}_d} \cdot \ldots \cdot \underbrace{\partial_{\mathbf{h}^t} \mathbf{h}^{t+1}}_{:=\mathbf{M}_t} \underbrace{\partial_{\mathbf{W}_t} \mathbf{h}^t}_{:=\mathbf{v}_t} .$$

Vanishing and Exploding Gradients

- Consider a deep neural network with d layers, input $\mathbf{x}$ and output $\mathbf{o}$.
- Each layer satisfies $\mathbf{h}^{t+1} = f_t(\mathbf{h}^t)$ and thus $\mathbf{o} = f_d \circ \ldots \circ f_1(\mathbf{x})$
- We can write the gradient of $\mathbf{o}$ with respect to any set of parameters $\mathbf{W}_t$ at layer t simply as

$$\partial_{\mathbf{W}_t} \mathbf{o} = \underbrace{\partial_{\mathbf{h}^{d-1}} \mathbf{h}^d}_{:= \mathbf{M}_d} \cdot \ldots \cdot \underbrace{\partial_{\mathbf{h}^t} \mathbf{h}^{t+1}}_{:= \mathbf{M}_t} \underbrace{\partial_{\mathbf{W}_t} \mathbf{h}^t}_{:= \mathbf{v}_t}.$$

- In other words, it is the product of $d - t$ matrices and a gradient vector

Vanishing and Exploding Gradients

- Consider a deep neural network with d layers, input $\mathbf{x}$ and output $\mathbf{o}$.
- Each layer satisfies $\mathbf{h}^{t+1} = f_t(\mathbf{h}^t)$ and thus $\mathbf{o} = f_d \circ \ldots \circ f_1(\mathbf{x})$
- We can write the gradient of $\mathbf{o}$ with respect to any set of parameters $\mathbf{W}_t$ at layer t simply as

$$\partial_{\mathbf{W}_t} \mathbf{o} = \underbrace{\partial_{\mathbf{h}^{d-1}} \mathbf{h}^d}_{:=\mathbf{M}_d} \cdot \ldots \cdot \underbrace{\partial_{\mathbf{h}^t} \mathbf{h}^{t+1}}_{:=\mathbf{M}_t} \underbrace{\partial_{\mathbf{W}_t} \mathbf{h}^t}_{:=\mathbf{v}_t}.$$

- In other words, it is the product of $d - t$ matrices and a gradient vector
- This product might be too large or too small!

Vanishing Gradients

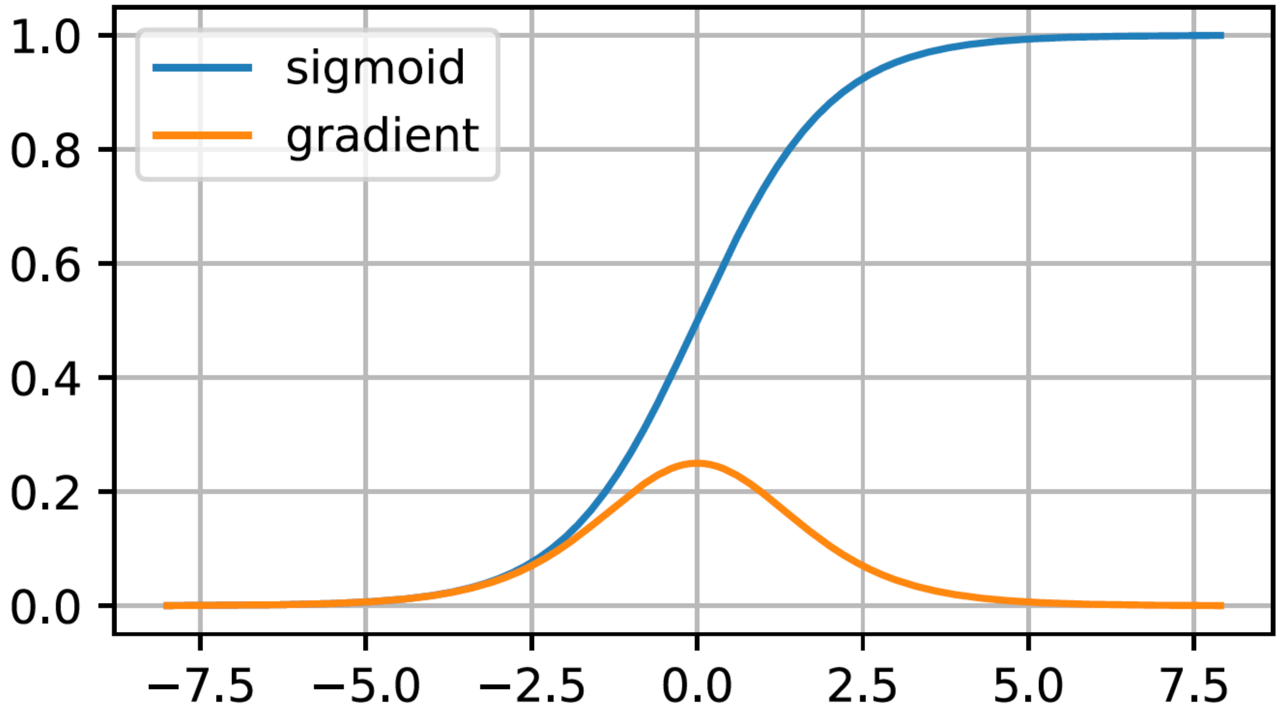
- Major problem for vanishing gradients is usually related to the activation function

Vanishing Gradients

- Major problem for vanishing gradients is usually related to the activation function
- Sigmoid function outputs 0 gradients unless we are in a perfect zone.

Vanishing Gradients

- Major problem for vanishing gradients is usually related to the activation function
- Sigmoid function outputs 0 gradients unless we are in a perfect zone.
- That is why ReLUs have become the default choice !



Exploding Gradients

- The opposite problem of vanishing gradients

Exploding Gradients

- The opposite problem of vanishing gradients
- Let’s draw 100 Gaussian random matrices and multiply them with some initial matrix 100 times.

Exploding Gradients

- The opposite problem of vanishing gradients
- Let’s draw 100 Gaussian random matrices and multiply them with some initial matrix 100 times.
- The matrix product explodes

```
A single matrix
[[ 2.2122064   0.7740038   1.0434405   1.1839255 ]
 [ 1.8917114  -1.2347414  -1.771029   -0.45138445]
 [ 0.57938355 -1.856082   -1.9768796  -0.20801921]
 [ 0.2444218  -0.03716067 -0.48774993 -0.02261727]]
<NDArray 4x4 @cpu(0)>
After multiplying 100 matrices
[[ 3.1575275e+20 -5.0052276e+19  2.0565092e+21 -2.3741922e+20]
 [-4.6332600e+20  7.3445046e+19 -3.0176513e+21  3.4838066e+20]
 [-5.8487235e+20  9.2711797e+19 -3.8092853e+21  4.3977330e+20]
 [-6.2947415e+19  9.9783660e+18 -4.0997977e+20  4.7331174e+19]]
<NDArray 4x4 @cpu(0)>
```

Symmetry

- Imagine what would happen if we initialized all of the parameters of some layer as $\mathbf{W}_l = c$ for some constant c.

Symmetry

- Imagine what would happen if we initialized all of the parameters of some layer as $\mathbf{W}_l = c$ for some constant c.
- In this case, the gradients for all dimensions are identical

Symmetry

- Imagine what would happen if we initialized all of the parameters of some layer as $\mathbf{W}_l = c$ for some constant c.
- In this case, the gradients for all dimensions are identical
- SGD would never break symmetry but dropout regularization would.

Parameter Initialization

- One way of addressing the issues raised above is through careful initialization of the weight vectors
- PyTorch uses uniform initialization by default for Linear layers

```
Attributes:
    weight: the learnable weights of the module of shape
        :math:`(\text{out\_features}, \text{in\_features})`. The values are
        initialized from :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})`, where
        :math:`k = \frac{1}{\text{in\_features}}`
    bias:   the learnable bias of the module of shape :math:`(\text{out\_features})`.
            If :attr:`bias` is ``True``, the values are initialized from
            :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
            :math:`k = \frac{1}{\text{in\_features}}`
```

Parameter Initialization

- One way of addressing the issues raised above is through careful initialization of the weight vectors

Xavier Initialization

- Let’s look at the scale distributions of the activations of the hidden unit h_i: $h_i = \sum_{j=1}^{n_{in}} W_{ij} x_j$

Xavier Initialization

- Let's look at the scale distributions of the activations of the hidden unit h_i: $h_i = \sum_{j=1}^{n_{in}} W_{ij} x_j$
- Assume that W_{ij} are i.i.d and have zero mean and variance σ^2

Xavier Initialization

- Let's look at the scale distributions of the activations of the hidden unit h_i: $h_i = \sum_{j=1}^{n_{in}} W_{ij} x_j$
- Assume that W_{ij} are i.i.d and have zero mean and variance σ^2
- Also assume that the data is zero mean with variance γ^2

Xavier Initialization

- Let's look at the scale distributions of the activations of the hidden unit h_i: $h_i = \sum_{j=1}^{n_{in}} W_{ij} x_j$
- Assume that W_{ij} are i.i.d and have zero mean and variance σ^2
- Also assume that the data is zero mean with variance γ^2
- In this case, the mean and variance of h_i is:

$$\mathrm{E}[h_i] = \sum_{j=1}^{n_{in}} \mathrm{E}[W_{ij} x_j] = 0 \qquad \mathrm{E}[h_i^2] = \sum_{j=1}^{n_{in}} \mathrm{E}[W_{ij}^2 x_j^2] = \sum_{j=1}^{n_{in}} \mathrm{E}[W_{ij}^2]\mathrm{E}[x_j^2] = n_{in}\sigma^2\gamma^2$$

Xavier Initialization

- Let's look at the scale distributions of the activations of the hidden unit h_i: $h_i = \sum_{j=1}^{n_{in}} W_{ij} x_j$
- Assume that W_{ij} are i.i.d and have zero mean and variance σ^2
- Also assume that the data is zero mean with variance γ^2
- In this case, the mean and variance of h_i is:

$$\mathrm{E}[h_i] = \sum_{j=1}^{n_{in}} \mathrm{E}[W_{ij} x_j] = 0 \qquad \mathrm{E}[h_i^2] = \sum_{j=1}^{n_{in}} \mathrm{E}[W_{ij}^2 x_j^2] = \sum_{j=1}^{n_{in}} \mathrm{E}[W_{ij}^2]\mathrm{E}[x_j^2] = n_{in}\sigma^2\gamma^2$$

- To keep the variance fixed, we need $n_{in}\sigma^2 = 1$.

Xavier Initialization

- Let's look at the scale distributions of the activations of the hidden unit h_i: $h_i = \sum_{j=1}^{n_{in}} W_{ij} x_j$
- Assume that W_{ij} are i.i.d and have zero mean and variance σ^2
- Also assume that the data is zero mean with variance γ^2
- In this case, the mean and variance of h_i is:

$$\mathrm{E}[h_i] = \sum_{j=1}^{n_{in}} \mathrm{E}[W_{ij} x_j] = 0 \qquad \mathrm{E}[h_i^2] = \sum_{j=1}^{n_{in}} \mathrm{E}[W_{ij}^2 x_j^2] = \sum_{j=1}^{n_{in}} \mathrm{E}[W_{ij}^2]\mathrm{E}[x_j^2] = n_{in}\sigma^2\gamma^2$$

- To keep the variance fixed, we need $n_{in}\sigma^2 = 1$.
- Similarly, we need $n_{out}\sigma^2 = 1$ for backpropagation which leaves us in a dilemma.

Xavier Initialization

- Let's look at the scale distributions of the activations of the hidden unit h_i: $h_i = \sum_{j=1}^{n_{in}} W_{ij} x_j$
- Assume that W_{ij} are i.i.d and have zero mean and variance σ^2
- Also assume that the data is zero mean with variance γ^2
- In this case, the mean and variance of h_i is:

$$\mathrm{E}[h_i] = \sum_{j=1}^{n_{in}} \mathrm{E}[W_{ij} x_j] = 0 \qquad \mathrm{E}[h_i^2] = \sum_{j=1}^{n_{in}} \mathrm{E}[W_{ij}^2 x_j^2] = \sum_{j=1}^{n_{in}} \mathrm{E}[W_{ij}^2]\mathrm{E}[x_j^2] = n_{in}\sigma^2\gamma^2$$

- To keep the variance fixed, we need $n_{in}\sigma^2 = 1$.
- Similarly, we need $n_{out}\sigma^2 = 1$ for backpropagation which leaves us in a dilemma.
- Xavier initialization simply tries to satisfy: $\frac{1}{2}(n_{in} + n_{out})\sigma^2 = 1$