RADONC AI CURRICULUM

Introduction to Deep Learning

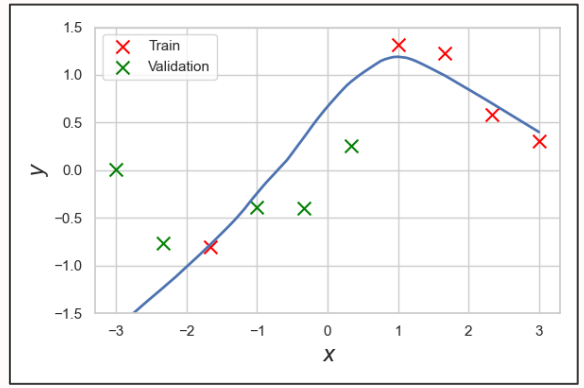
LECTURE 3: Generalization & Learning Algorithms

ANDREW Y. K. FOONG, PH.D.

April 17th 2026

MAYO CLINIC

Radiation Oncology
AI & Data Analytics
AIDA



Today's lecture

1. Generalization
   - *Going beyond the training data*
2. Some mathematics
   - *What is a probability distribution?*
3. Generalization revisited
   - *Mathematically describing generalization*
4. Toy example of generalization
   - *Generalization in a regression problem*
5. Inductive biases
   - *How modeling choices affect generalization*
6. The curse of dimensionality
   - *Why optimization is hard*
7. Tuning gradient descent
   - *Optimizing better*
8. Beyond optimization
   - *The problem of overfitting*
9. Q&A

Generalization

Going beyond the training data

Generalization

- Learning *not* just about memorization.
- About generalizing to new scenarios.
  - *The ability to solve the task well on data that is not included in the training data.*
- Two kinds of generalization:
  1. IN-DISTRIBUTION GENERALIZATION (or just, generalization)—the ability to solve the task well on data that comes from the *same distribution* as the training data.
  2. OUT-OF-DISTRIBUTION GENERALIZATION (or OOD generalization)—the ability to solve the task well on data that comes *a different distribution* from the training data.
- To define this, we need to understand the concept of the *data distribution*.
- And to understand data distributions, we need to understand *probability distributions*.

Some Mathematics

What is a probability distribution?

Probability distributions

- Roughly speaking, a PROBABILITY DISTRIBUTION is a *mathematical description of a random process.*

EXAMPLE: flipping a coin

- When flipping a coin, we don't know in advance what the outcome will be.
- We know that it must be *either* HEADS or TAILS.
- If the coin is fair, we have no reason to believe it's any more likely to be HEADS than TAILS, vice versa.
- Hence the probability of HEADS and TAILS must be *equal*.
- The probability of HEADS added up with the probability of TAILS *must* equal 1.
- The only possibility is that both have probability 0.5.

Probability distributions

EXAMPLE: flipping a coin—described mathematically

- We can express this as a set equations:
  1. $\Pr(\text{HEADS}) = \Pr(\text{TAILS})$
  2. $\Pr(\text{HEADS}) + \Pr(\text{TAILS}) = 1$
- By substituting equation 1 into equation 2, we get:

$$\Pr(\text{HEADS}) + \Pr(\text{HEADS}) = 1$$
$$2\Pr(\text{HEADS}) = 1$$
$$\Pr(\text{HEADS}) = \tfrac{1}{2}$$

- "Pr" is a *probability distribution*:
  - It is a *function* that, given an outcome, assigns a probability value to that outcome.
  - If we know $\Pr(x)$ for all possible outcomes x, then we know how likely *any* event is going to be.

Probability distributions

EXAMPLE: flipping a biased coin

- When flipping a *biased* coin, we don't know in advance what the outcome will be.
- We know that it *must* be either HEADS or TAILS.
- But unlike the fair coin, the outcomes are no longer equally probable.

> *The way we specify probability distributions depends on the information we have about the problem, and what assumptions we are willing to make.*

- For a biased coin, unless we obtain more *information* about the bias, or make *assumptions* about it, we cannot proceed.
- We know there is *some* probability distribution.
- We know that $\Pr(\text{HEADS}) + \Pr(\text{TAILS}) = 1$.
- But we are unable to provide more details.

Data distributions

- Return to the *regression problem*: predicting y from x.

EXAMPLE:

- Assume x represents height, and y represents weight.
- Assume we trained a neural network, $y = f(x)$ that predicts height from weight.
- Assume the mean-squared error loss on the training data is very low.
- How can we *measure* if it generalizes to new data?
- Idea:
  1. Identify loss function that we care about—e.g., mean-squared error (note, does *not* have to be same as *training* loss function).
  2. Sample new examples (x, y) from the same PROBABILITY DISTRIBUTION that generated the training data.
  3. Calculate the average (*the expected*) value of the loss across the new examples.

Data distributions

- What *is* the probability distribution that generated the training data?

EXAMPLE: height and weight at Med City University

- Volunteers were sourced from the campus of Med City University in Summer 2024.
- Their height and weight, (x, y) were recorded.
- MATHEMATICAL REPRESENTATION:
  - The pair (x, y) *was distributed according to* $\Pr(x, y)$, where:
    - $\Pr(x, y)$ represents the process of collecting volunteers from the Med City University Campus in Summer 2024.
    - *Notation*: $(x, y) \sim \Pr(x, y)$.
    - We don't know the precise value of $\Pr(x, y)$ for any given x, y value.
    - But we know that *samples* of x, y distributed according to $\Pr(x, y)$ were used to train the model.

Generalization Revisited

Mathematically describing generalization

In-distribution generalization

- ORIGINAL QUESTION: how can we measure if the model generalizes?
- IN-DISTRIBUTION GENERALIZATION asks if the loss is low for *other samples* from the *same data distribution*.
- Straightforward procedure to measure:
  1. Collect all samples $(x, y) \sim \Pr(x, y)$—e.g., by conducting a survey.
  2. Hold out some of the data (e.g., 20%). This is called a *test set* or *holdout set*.
  3. Train and tune the model using all data *except* the test set.
  4. Evaluate the average (e.g., mean) value of the loss function on the test set:
     - Low test loss: generalizes in-distribution.
     - High test loss: fails to generalize in-distribution.

In-distribution generalization

ORIGINAL QUESTION: how can we measure if the model generalizes?

EXAMPLE: height and weight at Med City University

1. Record height x and weight y of 100 students surveyed at Med City University in Summer 2024.
2. Randomly set aside 20 students as the TEST SET.
3. Train neural network f on remaining 80 students as the TRAINING SET.
4. Compute $f(x)$ for all students' heights, and compare with ground truth weights y.
5. TRAINING SET mean-squared error of ≈ 16 kg^2, root mean-squared error of ≈ 4 kg.
6. TEST SET mean-squared error of ≈ 25 kg^2, root mean-squared error of ≈ 5 kg.
7. If 5 kg RMSE is acceptable, we say *the model generalizes in-distribution.*
8. If not, we say *the model fails to generalize in-distribution.*

Out-of-distribution generalization

- OUT-OF-DISTRIBUTION GENERALIZATION asks if the loss is low for *samples* from *another data distribution we are interested in*.

EXAMPLE: Winter at Med City University

- Training data recorded in Summer 2024.
- Model deployed in Winter 2024.
- Exercise and eating habits change, relationship between height and weight altered.
- MATHEMATICAL REPRESENTATION:
  - In Winter, the pair (x, y) is distributed according to $\Pr_W(x, y)$, *not* $\Pr(x, y)$.
  - $\Pr_W(x, y)$ represents the process of collecting volunteers from the Med City University Campus in Winter 2024.
  - We do not know the values of $\Pr_W(x, y)$, but believe it is significantly different from $\Pr(x, y)$.

Out-of-distribution generalization

- OUT-OF-DISTRIBUTION GENERALIZATION is significantly more challenging for most models than in-distribution generalization.
- But can be measured in the same way, except the TEST SET is now from a *different distribution than the training set.*
- OOD generalization is a common challenge in AI for healthcare:
  - Train on data collected from one hospital, deploy in another.
  - Train on data collected from before a certain time point, test on data after that time.
  - Train on data from high-risk patients only, test on all patients, etc….
- Synonymous with the OOD generalization problem:
  - *"OOD data"*
  - *"Data shift"*
  - *"Distribution shift"*

Validation sets

- Common to introduce an *additional* split of the data: the VALIDATION SET.
- The validation set is used to tune *model hyperparameters*.

EXAMPLE: Test set poisoning

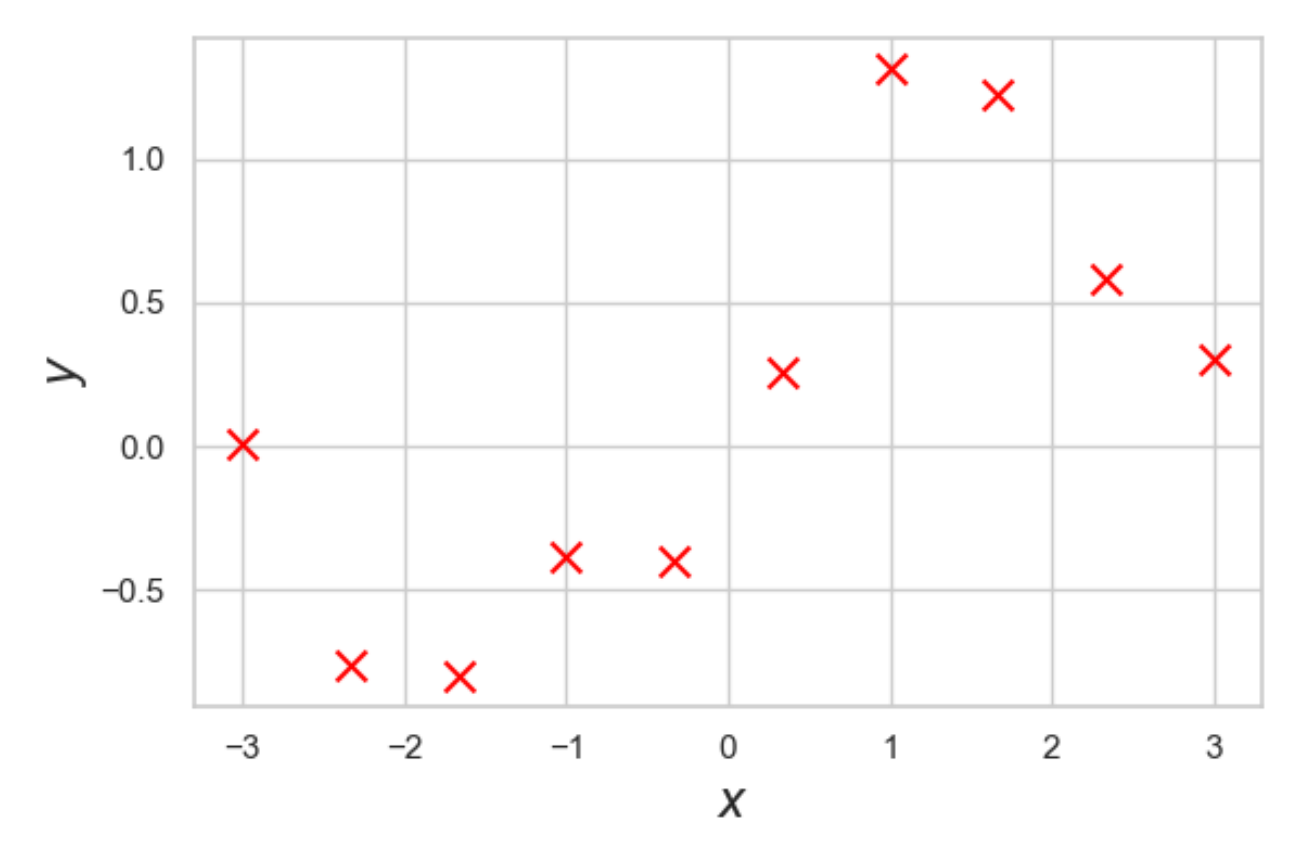
- A researcher is unsure whether to use a neural network with 3, 100, or 1000 neurons.
- They try all three and select the one with the best generalization performance on the *test set*.
- They publish the results of the best-performing network only.
- This is known as test-set poisoning:
  - The researcher made a *choice* about the model based on *which model did best on the test set.*
  - In deployment, we won't know in advance which model will do best.
  - This is effectively cheating—treating the *test data* like it was *training data.*
- SOLUTION: split off a validation set before training.
  - Can be used to choose any aspect of the model without fear of poisoning.
  - The test set is *only used for calculating the final generalization performance*.

Toy Example of Generalization

Generalization in a regression problem

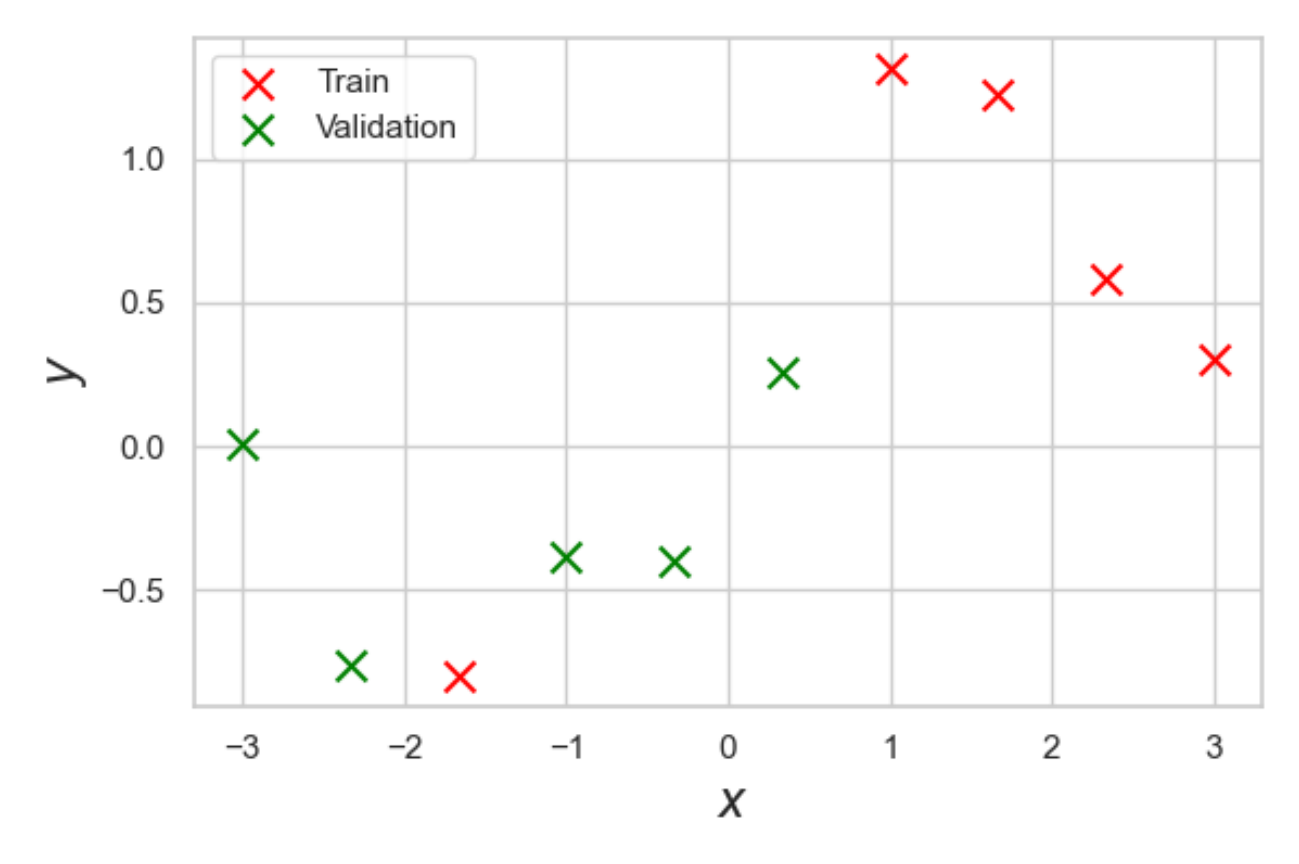
Generalization in one dimension

- Given training data.



Generalization in one dimension

- Given training data.
- Randomly split into TRAIN and VALIDATION set to measure *in-distribution* generalization:

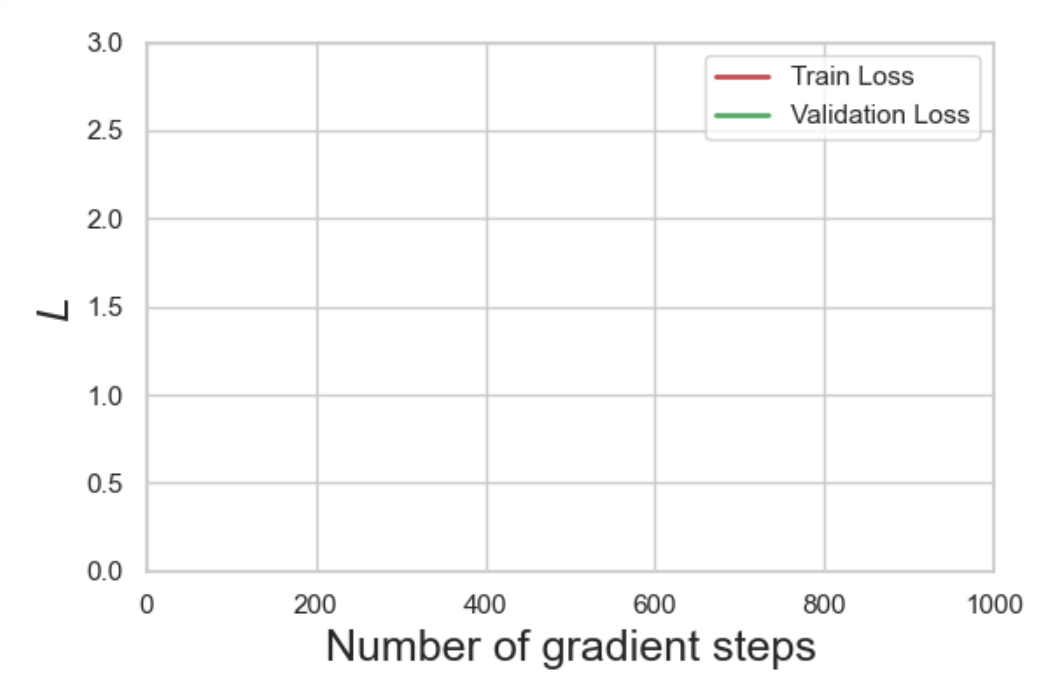


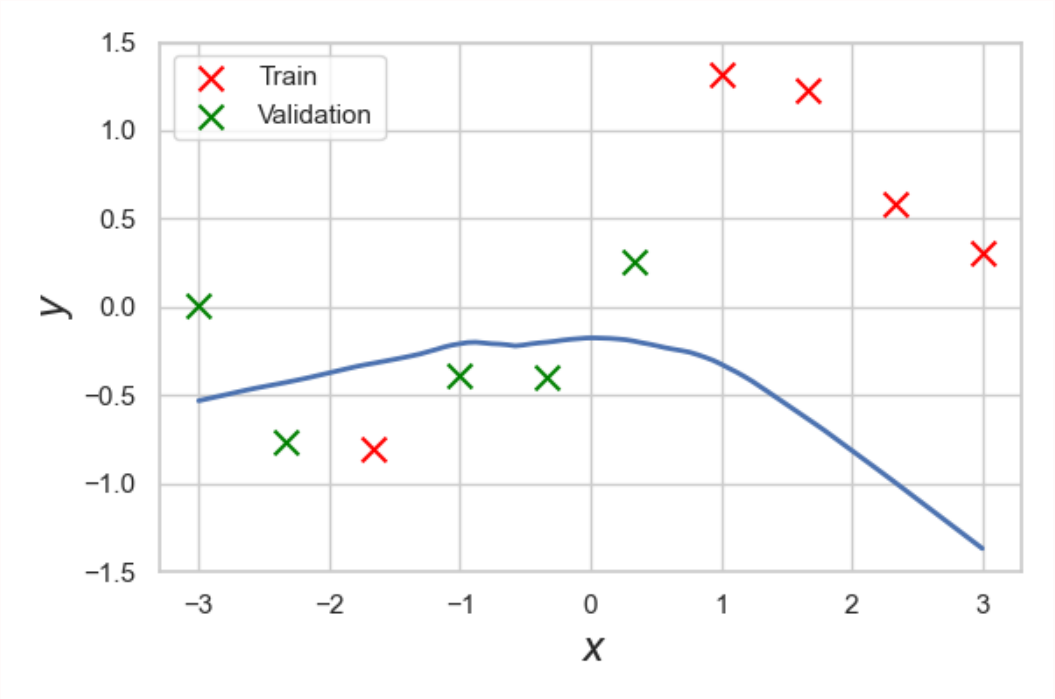
NOTE:

- For simplicity, will not show a test set here.
- A real evaluation should always have a separate test set.

Generalization in one dimension

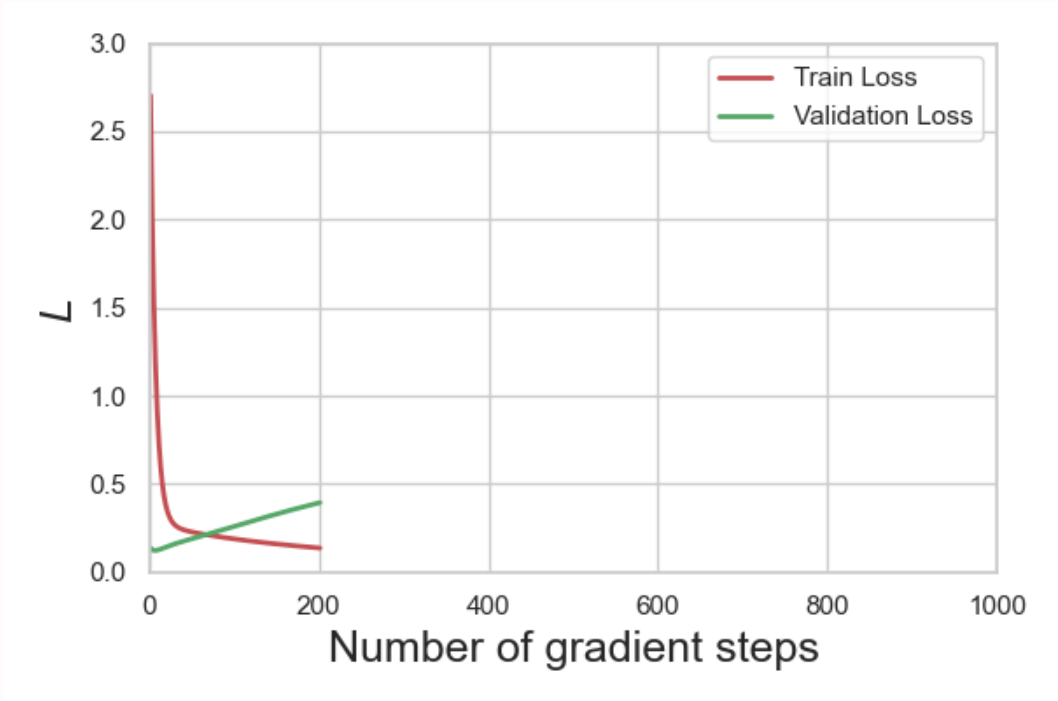
- Train neural network on TRAIN SET using gradient descent, but keep VALIDATION SET hidden.

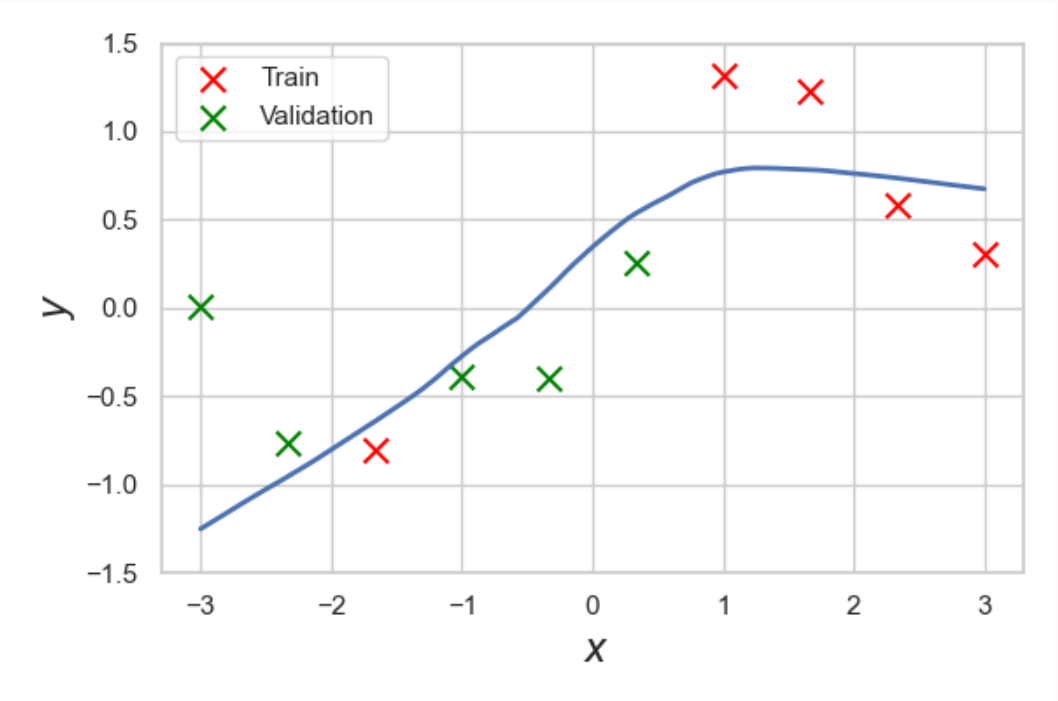




Generalization in one dimension

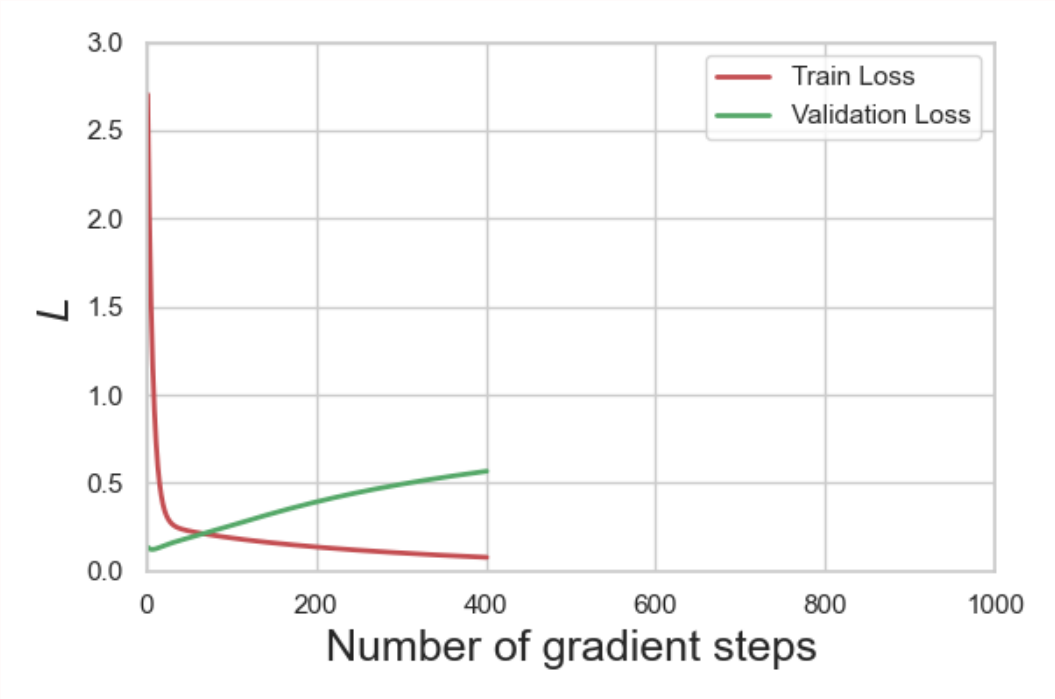
- Train neural network on TRAIN SET using gradient descent, but keep VALIDATION SET hidden.

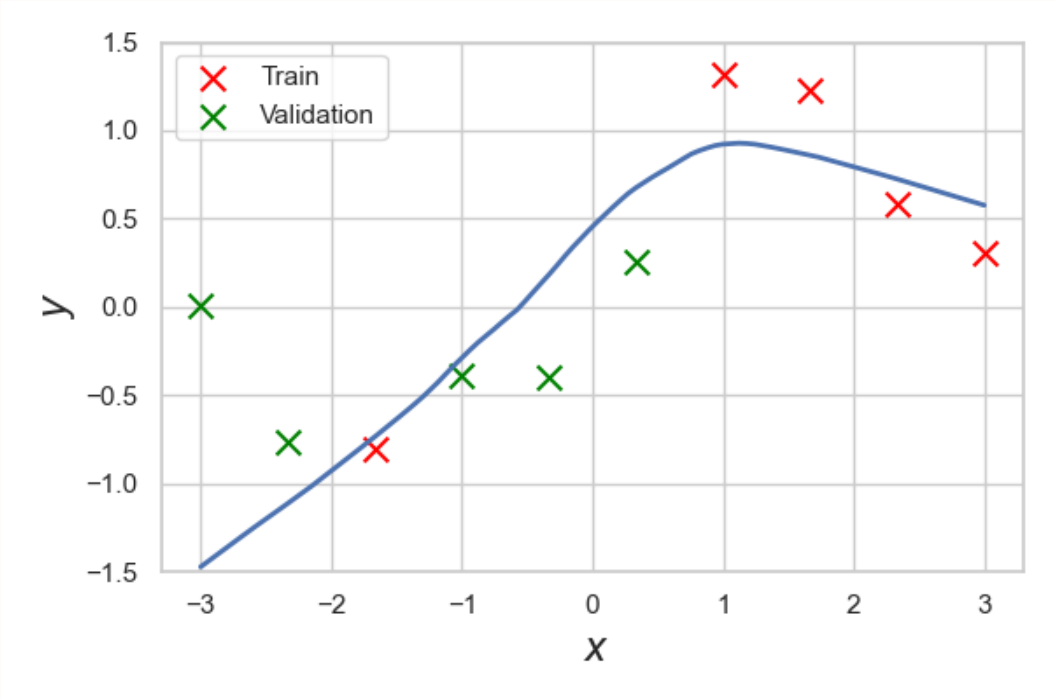




Generalization in one dimension

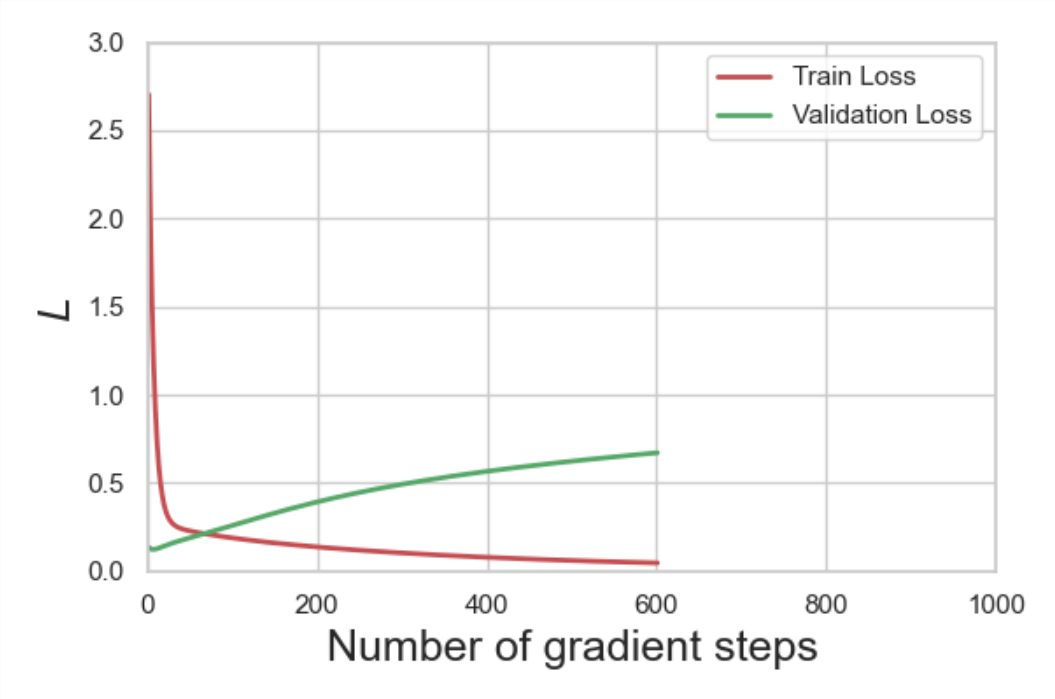
- Train neural network on TRAIN SET using gradient descent, but keep VALIDATION SET hidden.

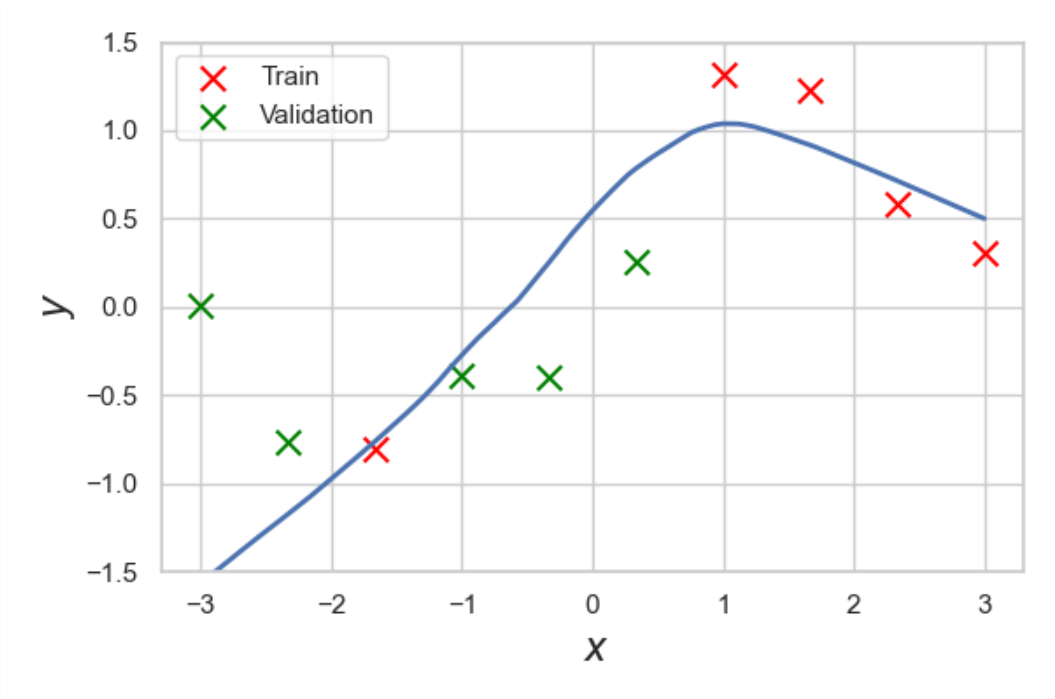




Generalization in one dimension

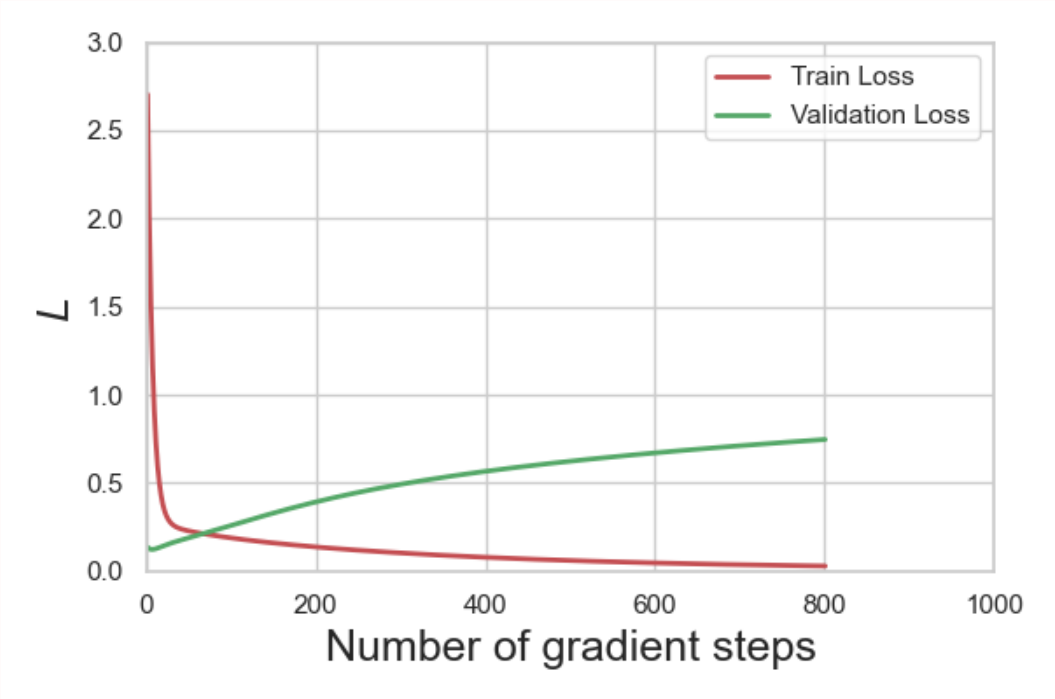
- Train neural network on TRAIN SET using gradient descent, but keep VALIDATION SET hidden.

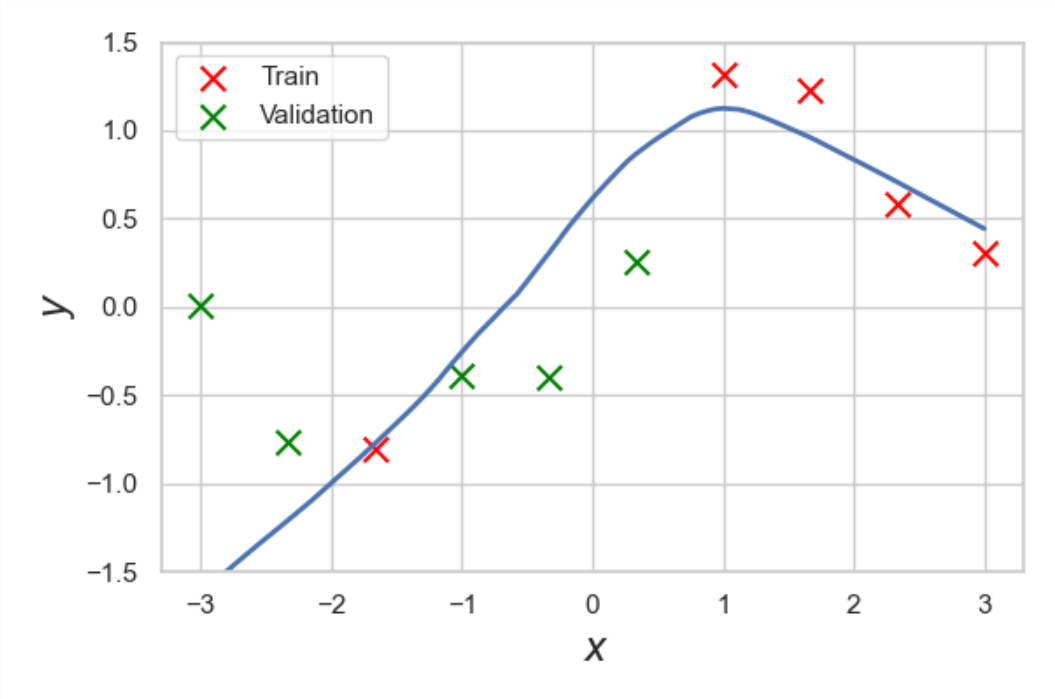




Generalization in one dimension

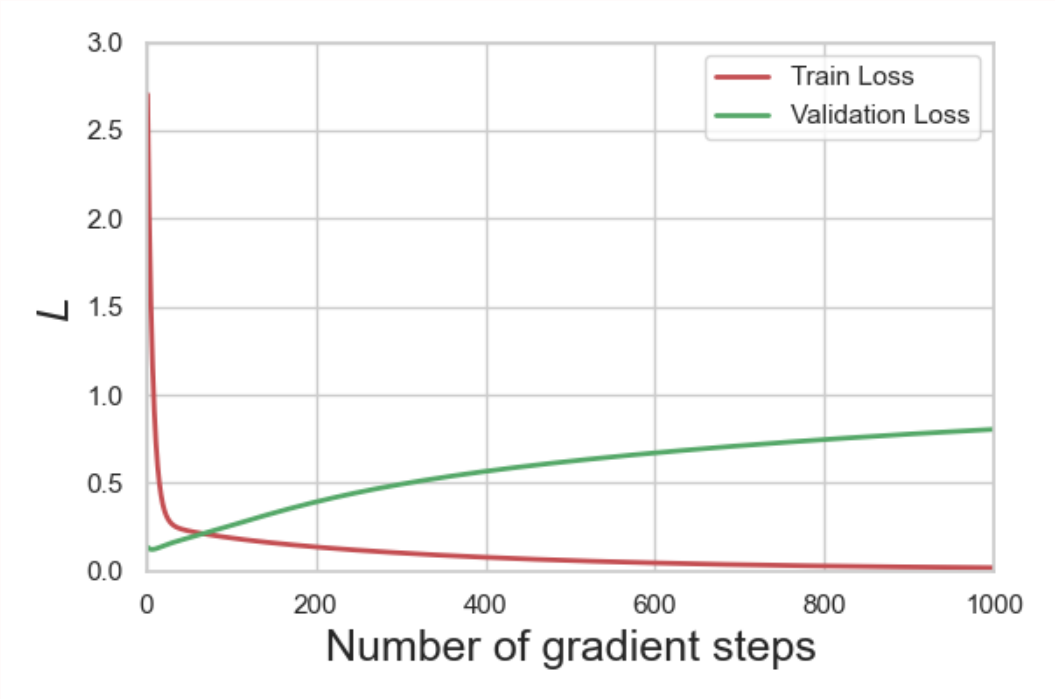
- Train neural network on TRAIN SET using gradient descent, but keep VALIDATION SET hidden.

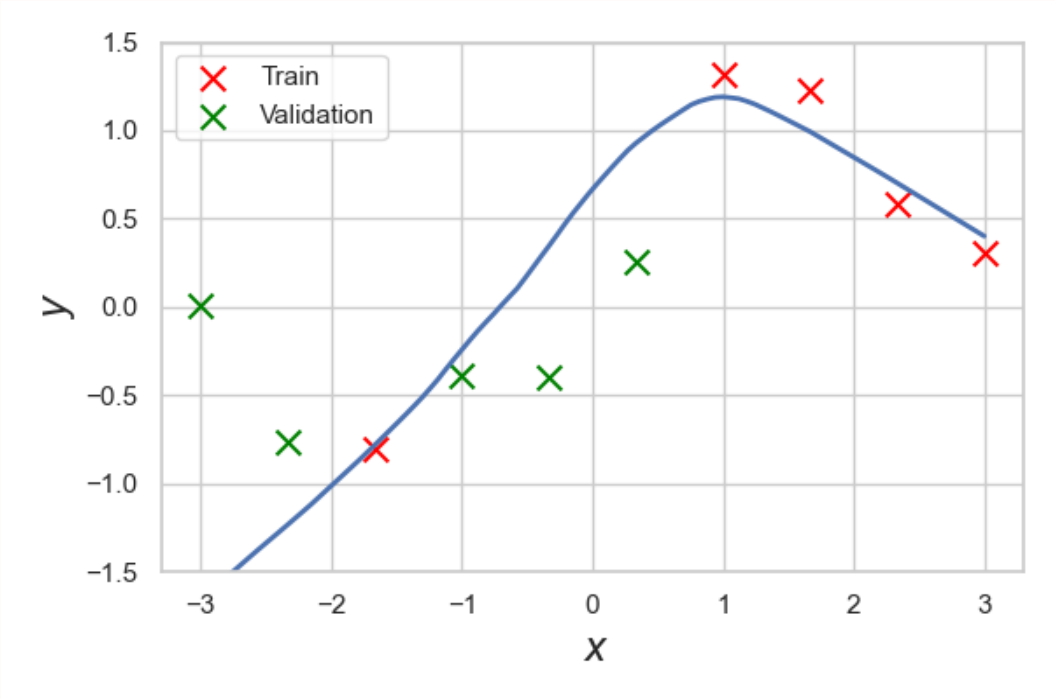




Generalization in one dimension

- Train neural network on TRAIN SET using gradient descent, but keep VALIDATION SET hidden.

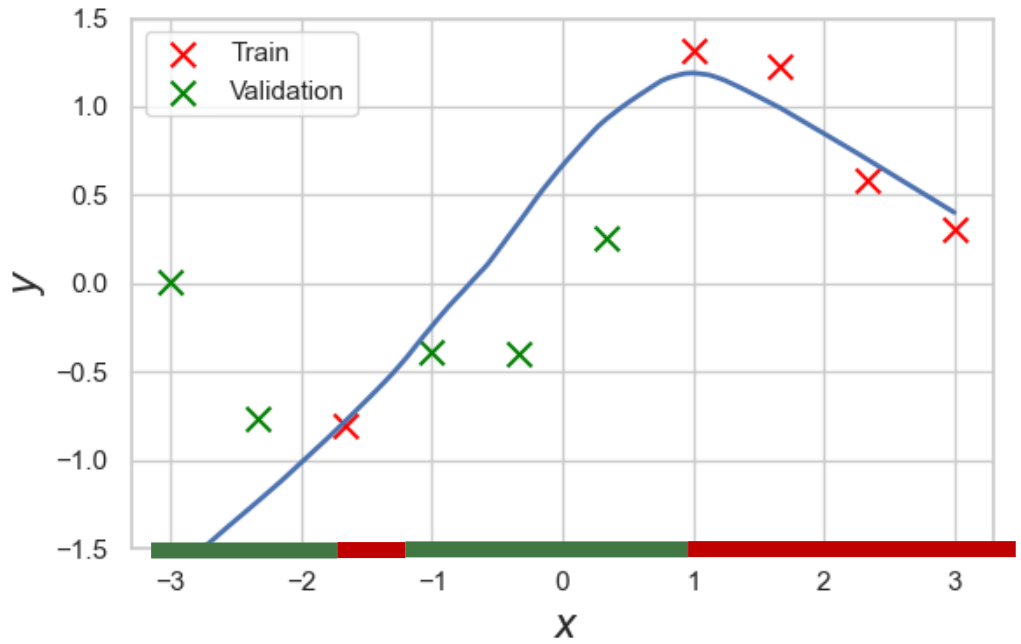




- Train loss goes to zero but validation loss increases!
- Network does *not* generalize in-distribution.

What's going on?

- **RED REGION:**
  - Fit determined by *data*.
  - Loss ⟶ zero as long as enough neurons.
- **GREEN REGION:**
  - *Many ways to extrapolate* while getting low train loss.
  - Fit determined by:
    - Number of neurons.
    - Number of gradient descent steps.
    - Choice of non-linearity, ReLU vs tanh vs GELU etc.
    - Potentially many other factors.
  - Different choices extrapolate differently.
  - Some might generalize well, others might not.
  - Hard to know for sure in advance. *Deep learning is empirical.*

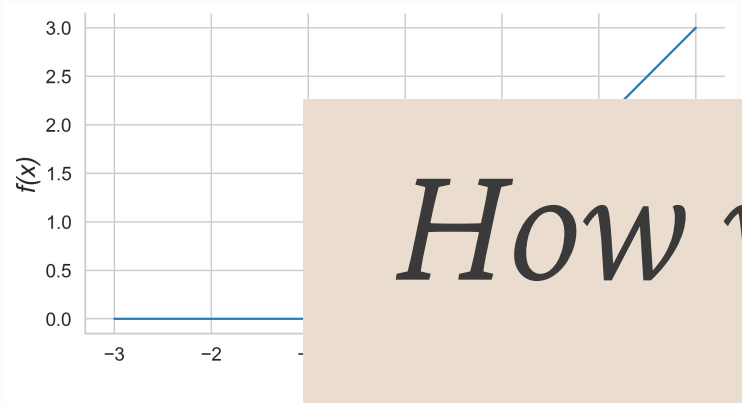


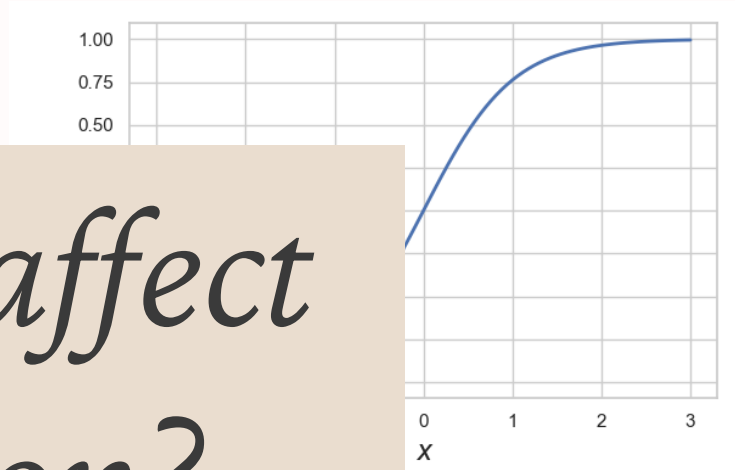
Inductive biases

How modeling choices affect generalization

Generalization experiment

EXAMPLE: How the choice of non-linearity affects generalization



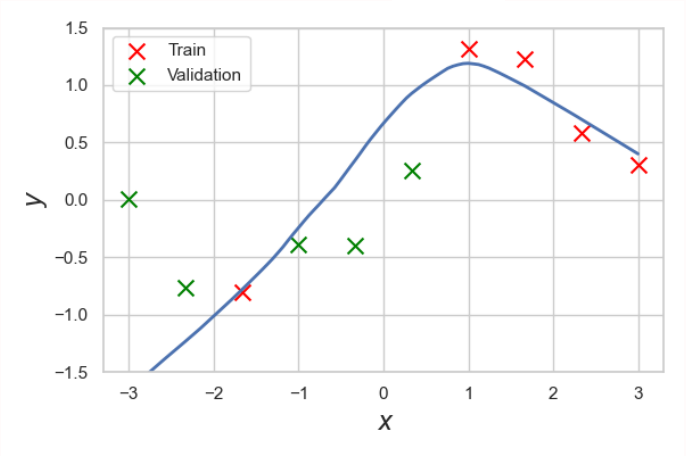


How will this affect generalization?

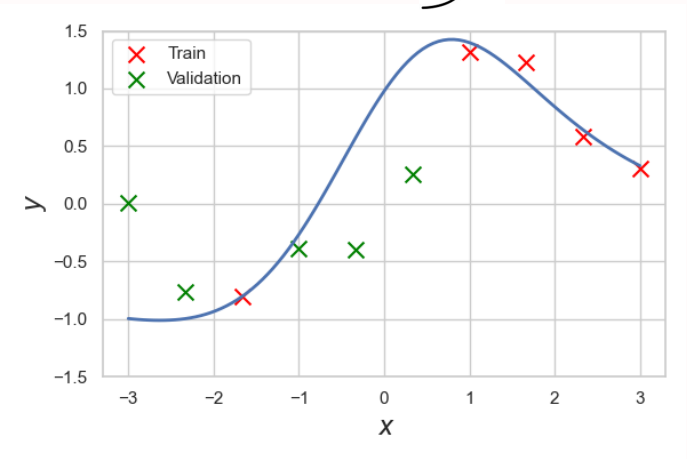
Generalization experiment

EXAMPLE: How the choice of non-linearity affects generalization

ReLU:



tanh:



- Tanh network flattens out on the left of the curve: *generalizes differently*.
- Neither tanh nor ReLU better for *all* data distributions:
  - Tanh better for data that *flattens out*.
  - ReLU better for data that *continues in straight lines*.
  - Need to match choices to data distribution.

Inductive biases

- Broader idea: INDUCTIVE BIASES.
  - INDUCTION:
    The inference of a general law from particular instances.
  - INDUCTIVE BIASES:
    Choices we make that bias the network to infer things in one way as opposed to another.
  - Not to be confused with *biases parameters* in a network, *b*.
- In absence of data:
  - ReLU network *tends* to generalize in straight lines.
  - Tanh network *tends* to generalize in flat lines.

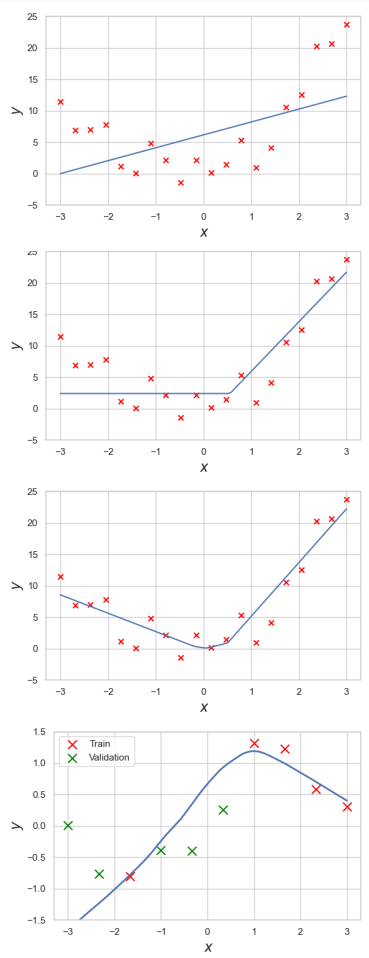
ReLU networks have an inductive bias towards straight lines.

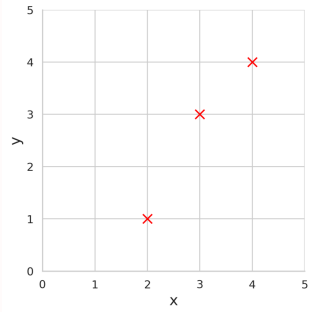
Inductive biases

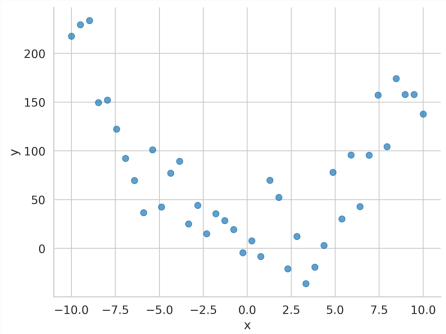
- Much of deep learning research consists in:
  1. Finding out how each network choice influences inductive biases
  2. Choosing accordingly based on our understanding of the data distribution.
- This process *is highly empirical*.

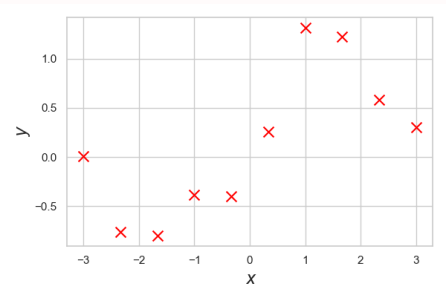
Inductive biases

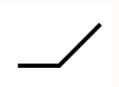
Predictive behavior = Training data + Inductive biases



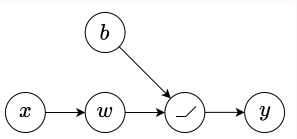




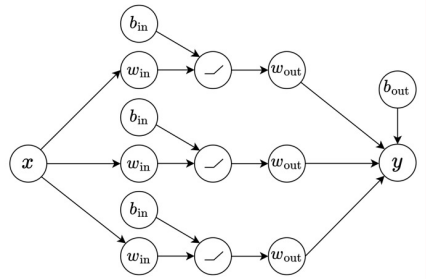


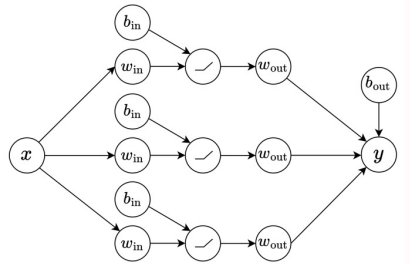


vs.

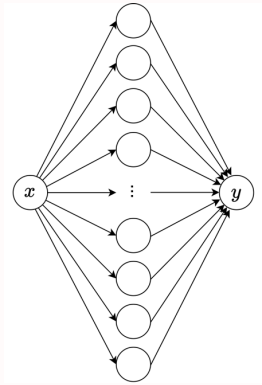


vs.





vs.



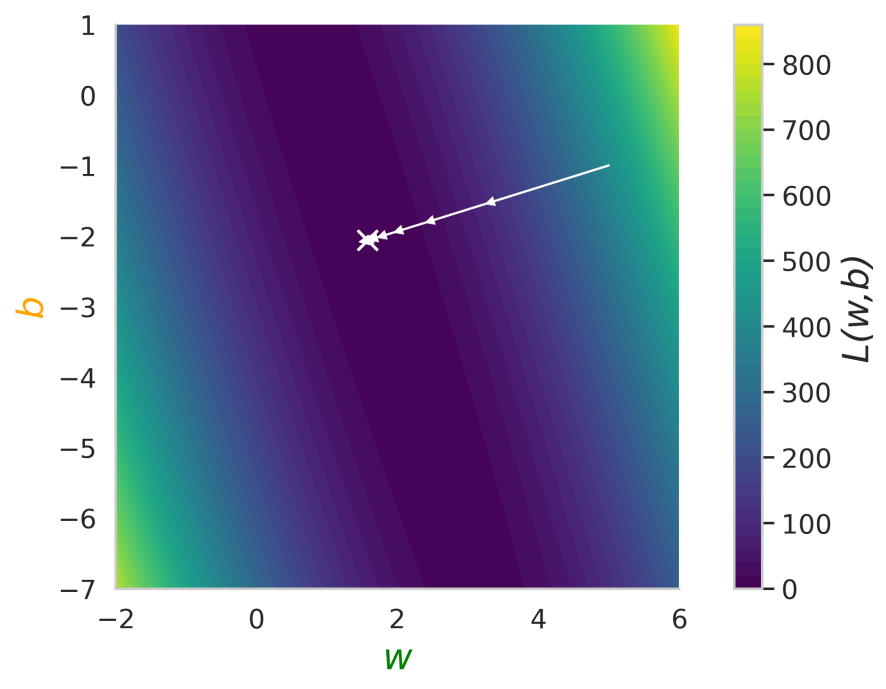
The Curse of Dimensionality

Why optimization is hard

Gradient descent recap

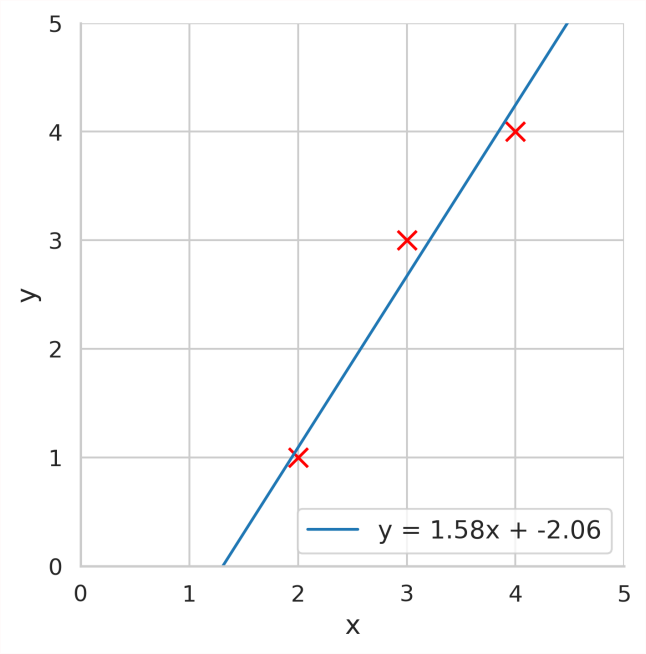
- *Gradient descent* is our first example of a learning algorithm.
- PROCEDURE:
  1. Choose parameters $\mathbf{w}$ *randomly* (terrible loss!).
  2. Calculate derivative/gradient of $L(\mathbf{w})$, called $\nabla L(\mathbf{w})$—easy on a computer.
  3. Adjust $\mathbf{w}$ by small amount in (opposite) direction of gradient.
  4. Repeat 2–3 until $L(\mathbf{w})$ is low (good fit!).
- Gradient descent is how *all* neural networks learn!
  - *But with many variations.*

Gradient descent recap



Contour plot of L(w, b)

⇔



Corresponding regression fit

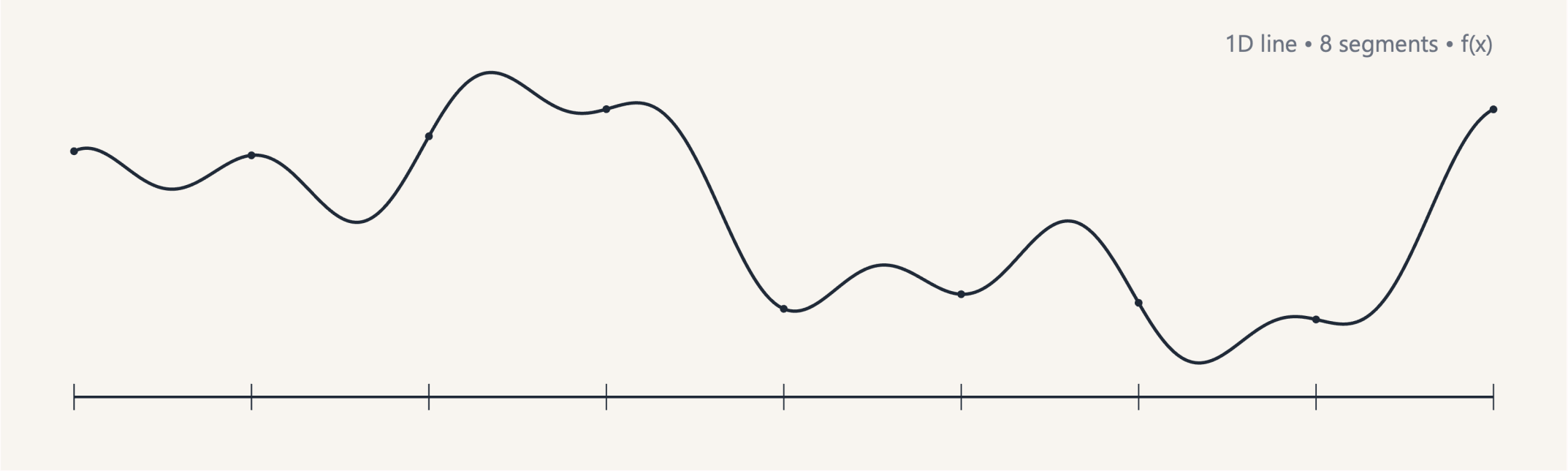
Grid search

- The simplest way to optimize a function is through *grid search*.
  1. Divide the input-space into a grid.
  2. Calculate the output at every grid point.
  3. Pick the input that has the smallest value of the output.
- ONE DIMENSION:
  1. Define grid: $x_1, x_2, \ldots, x_N$.
  2. Calculate: $f(x_1), f(x_2), \ldots, f(x_N)$.
  3. Choose value of x that minimizes $f(x)$.

Grid search

EXAMPLE: N = 8

- Computational cost: 8 function evaluations.

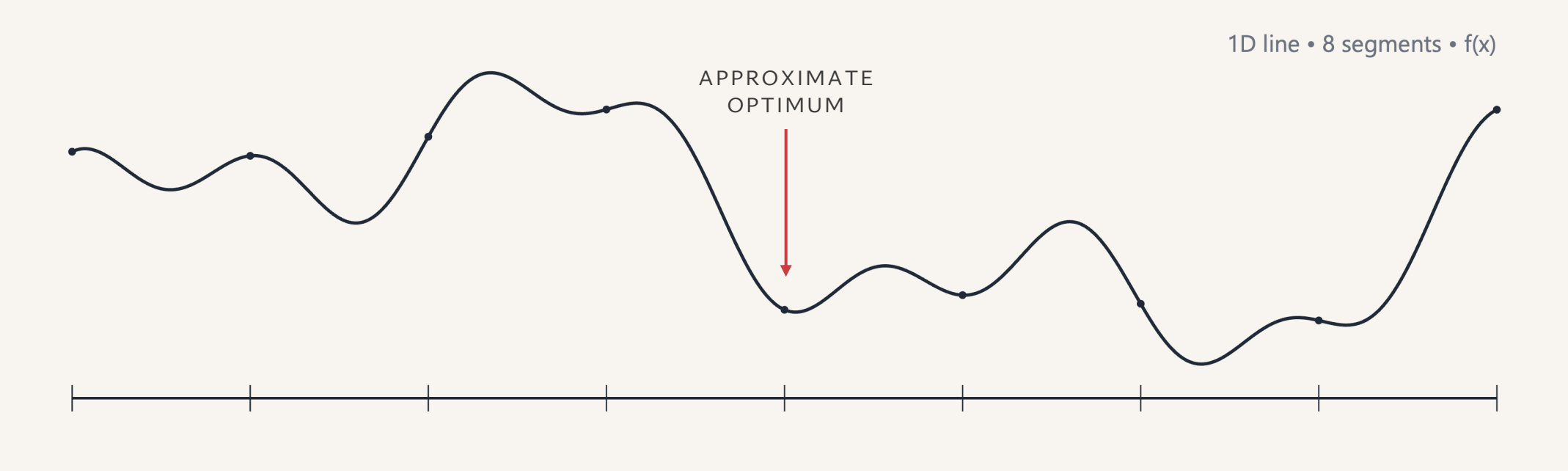


Grid search in one dimension

Grid search

EXAMPLE: N = 8

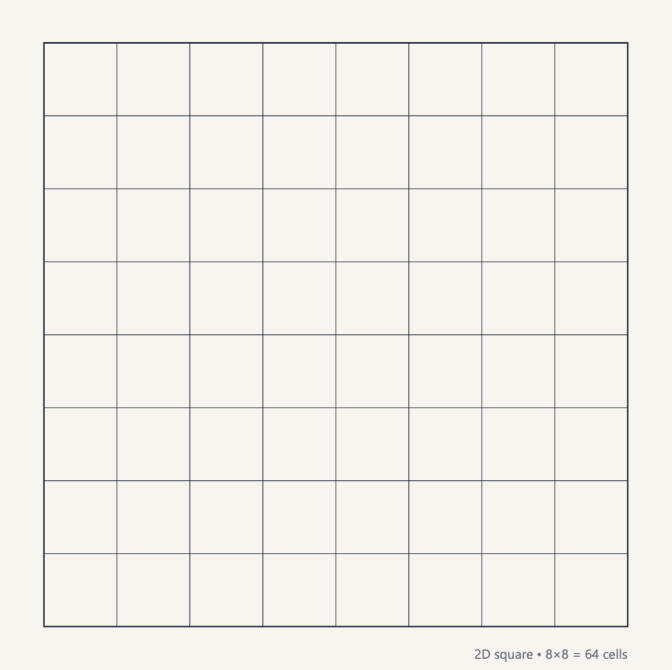
- Computational cost: 8 function evaluations.



Grid search in one dimension

Grid search

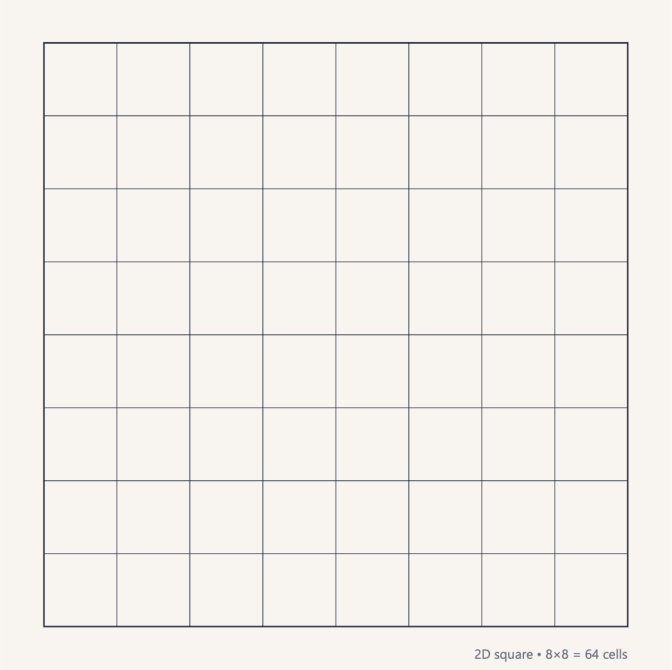
- TWO DIMENSIONS:
  1. Define grid: $x_{1,1}, x_{1,2}, \ldots, x_{1,N}, x_{2,1}, x_{2,2}, \ldots, x_{2,N}, \ldots, x_{N,1}, x_{N,2}, \ldots, x_{N,N}$.
  2. Calculate: $f(x_{1,1}), f(x_{1,2}), \ldots, f(x_{1,N}), f(x_{2,1}), f(x_{2,2}), \ldots, f(x_{2,N}), \ldots, f(x_{N,1}), f(x_{N,2}), \ldots, f(x_{N,N})$.
  3. Choose value of x that minimizes $f(x)$.



Input space for grid search in two dimensions

Grid search

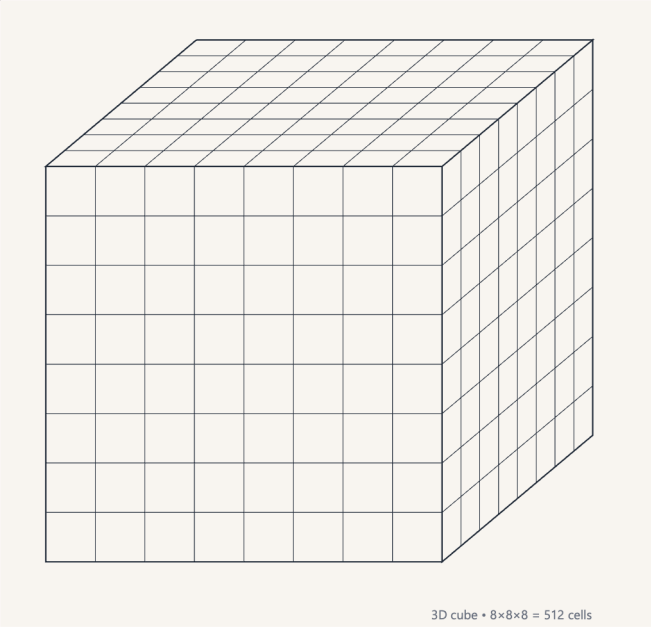
- TWO DIMENSIONS:
  - Computational cost: 8 × 8 = 64 function evaluations.



Input space for grid search in two dimensions

Grid search

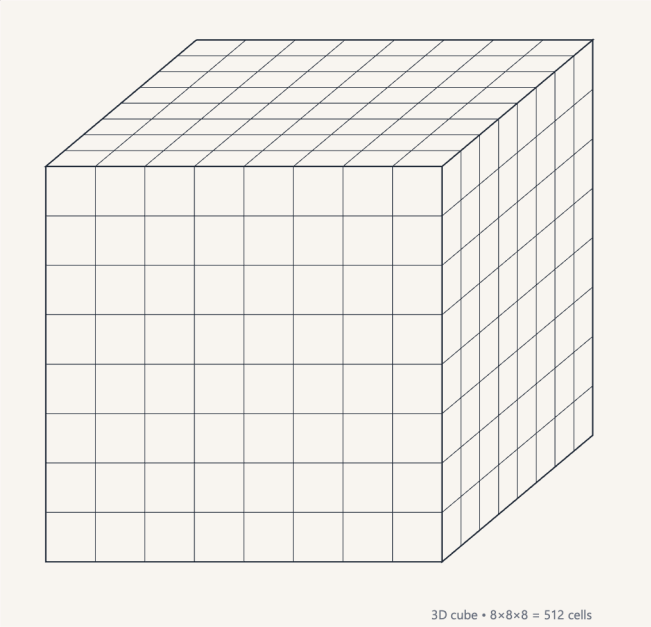
- THREE DIMENSIONS:
  1. Define grid: $x_{1,1,1}, \ldots, x_{N,N,N}$.
  2. Calculate: $f(x_{1,1,1}), f(x_{N,N,N})$.
  3. Choose value of x that minimizes $f(x)$.



Input space for grid search in three dimensions

Grid search

- THREE DIMENSIONS:
  - Computational cost: $8 \times 8 \times 8 = 512$ function evaluations.

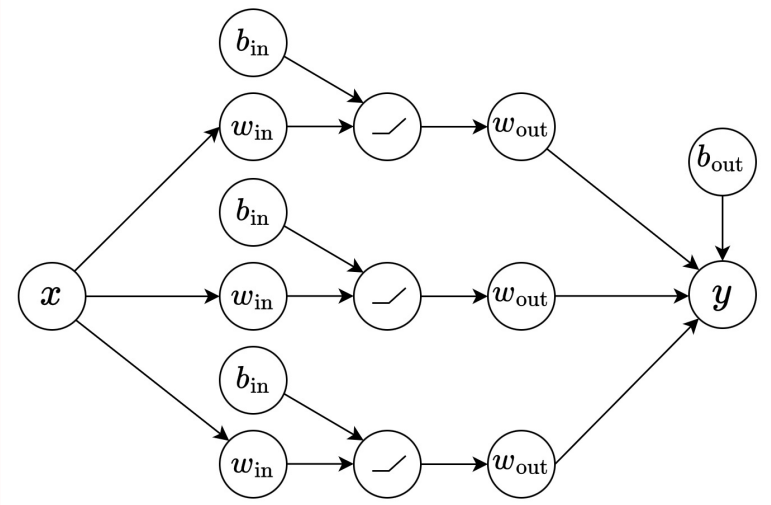


Input space for grid search in three dimensions

Grid search

- NEURAL NETWORK WITH THREE NEURONS:
  - Model has ten parameters
  - Loss function has ten inputs:

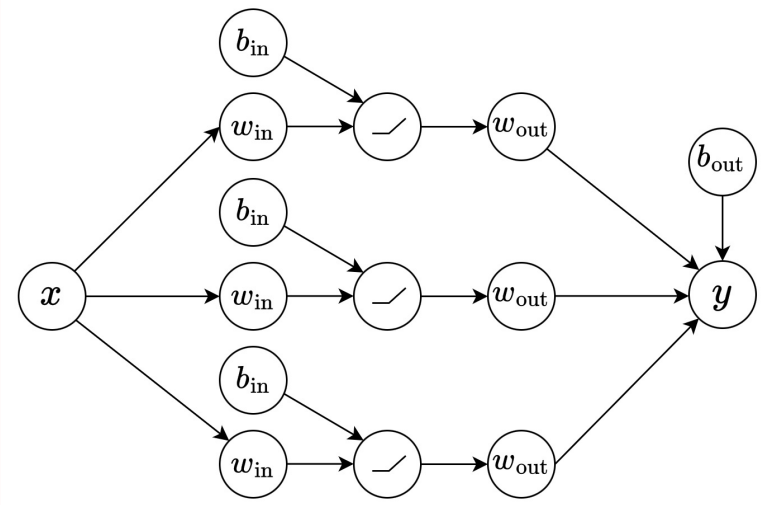
$$L(w_{\text{in},1}, w_{\text{in},2}, w_{\text{in},3}, b_{\text{in},1}, b_{\text{in},2}, b_{\text{in},3}, w_{\text{out},1}, w_{\text{out},2}, w_{\text{out},3}, b_{\text{out}})$$



Three-neuron network

Grid search

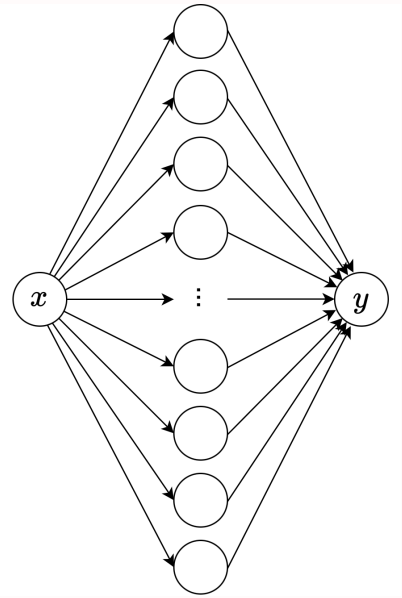
- **NEURAL NETWORK WITH THREE NEURONS:**
  - Computational cost: $8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 \times 8 = 8^{10} = 1{,}073{,}741{,}824$ function evaluations.
  - One function evaluation on a modern GPU $\approx 10^{-11}$ s.
  - Total time $\approx 10^{-2}$ s—still feasible!



Three-neuron network

Grid search

- NEURAL NETWORK WITH A HUNDRED NEURONS:
  - Computational cost $\approx 8^{300} = 8.45 \times 10^{270}$ function evaluations.
  - One function evaluation on a modern GPU $\approx 10^{-11}$ s.
  - Total time $\approx 8 \times 10^{259}$ s.

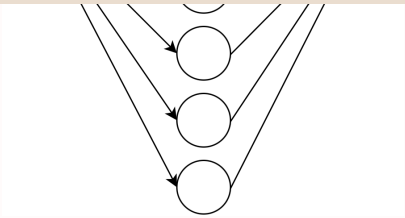


One hundred-neuron network

Grid search

- NEURAL NETWORK WITH A HUNDRED NEURONS:
  - Computational cost ≈ $8^{300} = 8.45 \times 10^{270}$ function evaluations.
  - O
  - T

If every atom in the observable universe were a supercomputer running at today's top GPU speed, the total time to perform grid search would exceed the lifetime of black holes by roughly 10^{150} orders of magnitude.



One hundred-neuron network

The curse of dimensionality

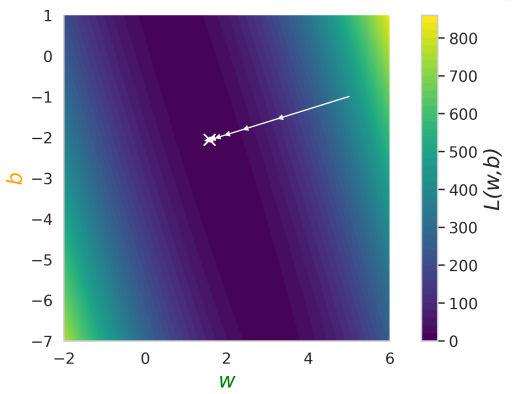
- Spaces get *exponentially* larger with increasing dimension.
- This is the CURSE OF DIMENSIONALITY.
- It's not just optimization:
  - Visualization
  - Exploration
  - Understanding
  - …become exponentially harder as dimensionality increases.
- ChatGPT requires optimization of roughly 1 *trillion* parameters.
- Deep learning advances have been powered by scaling models up.
- *Grid search doesn't scale.*

Gradient descent vs dimensionality

- GRADIENT DESCENT is one of the few algorithms with any hope of optimizing high-dimensional functions.
- *All* deep learning optimizers are modifications of gradient descent.

Gradient descent vs dimensionality

- COMPUTATIONAL COST OF GRADIENT DESCENT:
  - At each gradient iteration:
    1. Compute $f_{\mathbf{w}}(x)$ for the current value of $\mathbf{w}$—scales with number of parameters, N.
       - *If number of parameters doubles, cost roughly doubles.*
    2. Compute loss function—doesn't depend on N.
       - *If number of parameters doubles, cost is unchanged.*
    3. Compute gradient $\nabla L(\mathbf{w})$ for the current value of $\mathbf{w}$—scales with number of parameters, N.
       - *If number of parameters doubles, cost roughly doubles.*
  - Repeat for the number of gradient iterations.
  - Total cost depends on:
    - Number of parameters, N.
    - Number of gradient iterations.
  - *Cost increases proportionately (linearly) with N, not exponentially.*



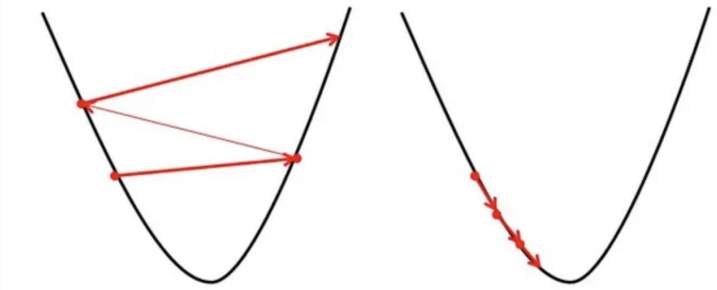
Iterations of gradient descent

Tuning gradient descent

Optimizing better

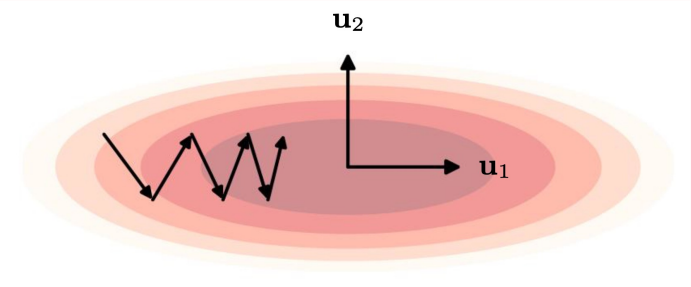
Learning rate

- GRADIENT DESCENT:
  1. Choose parameters $\mathbf{w}$ *randomly* (terrible loss!).
  2. Calculate derivative/gradient of $L(\mathbf{w})$, called $\nabla L(\mathbf{w})$—easy on a computer.
  3. Adjust $\mathbf{w}$ by small amount in (opposite) direction of gradient.
  4. Repeat 2–3 until $L(\mathbf{w})$ is low (good fit!).
- *How small is small?*
  - The size of the step taken is the LEARNING RATE.
  - If too small, optimization takes ages to finish.
  - If too large, optimization can *fail entirely*:

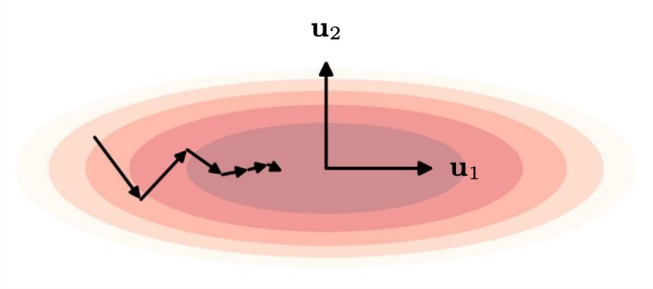


Momentum

- *Another problem*: in two or more dimensions, loss surface can have VALLEYS.
- The optimizer takes many gradient steps to reach the bottom.
- SOLUTION: add *momentum* to the gradient update.
  - Compute a running-average of the gradient values.
  - When taking steps, step in the direction of the running-average.



Gradient descent in a loss valley



Gradient descent with momentum
in a loss valley

Stochastic gradient descent

- *Another problem*: we often have *many* datapoints.
  - For ChatGPT this amounts to *trillions*.
  - Computing $f(x)$ for every input x at every iteration is infeasible.
- STOCHASTIC GRADIENT DESCENT:
  - At each iteration:
    - Pick a small subset of datapoints at random (*stochastic*).
    - Compute the loss only on those datapoints.
    - Compute the gradient only on those datapoints.
    - Take a step based on this partial gradient.
- Two advantages:
  1. Can train neural networks on *massive* datasets.
  2. Randomness can prevent overfitting—a kind of REGULARIZATION.

Beyond Optimization

The problem of overfitting

Some mathematics

- Is lower training always better?
- Mean-squared error loss is *lowest* if the training error is *zero*.
- Zero training loss: model fits the data *perfectly*.
- Can this happen?

Some mathematics

UNIVERSAL APPROXIMATION THEOREM:

Any dataset can be perfectly fit with a neuron network, provided there are sufficiently many neurons. This is true:

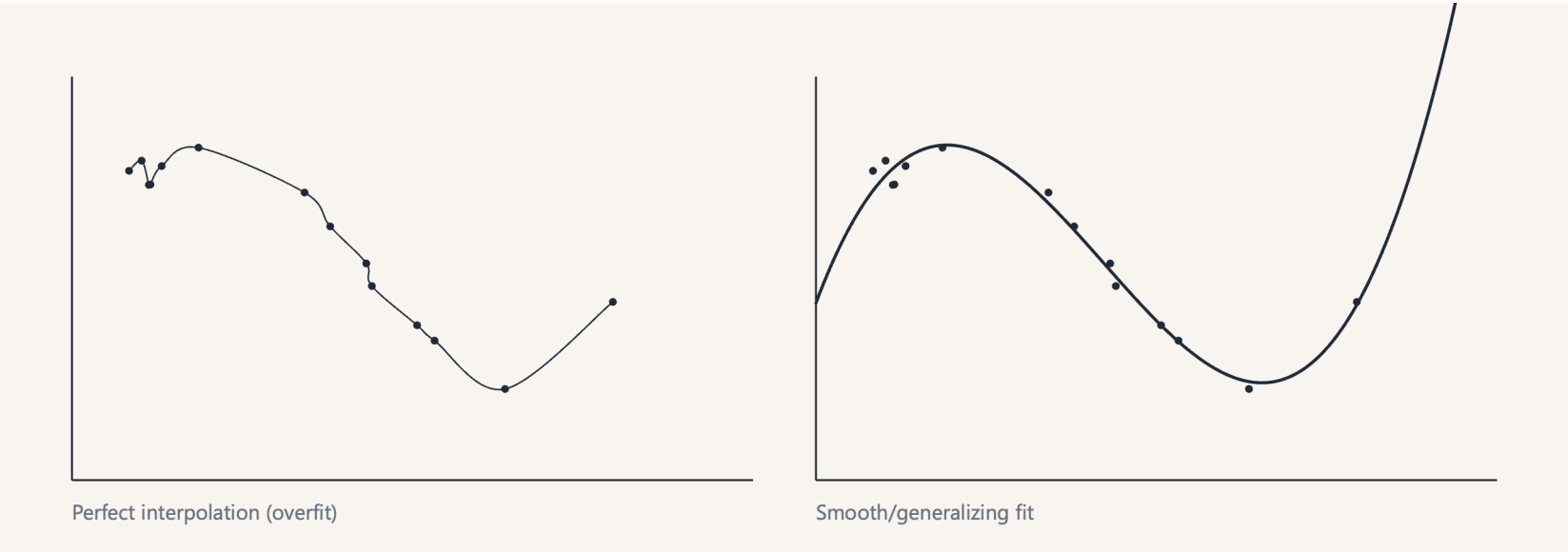
- *Regardless of how many inputs or outputs there are.*
- *For any number of hidden layers.*

NOTE: *this theorem only states that there exists a setting of the weights that can fit any dataset. It does not say how to find it.*

In principle, at least, *zero training loss is achievable.*

Overfitting

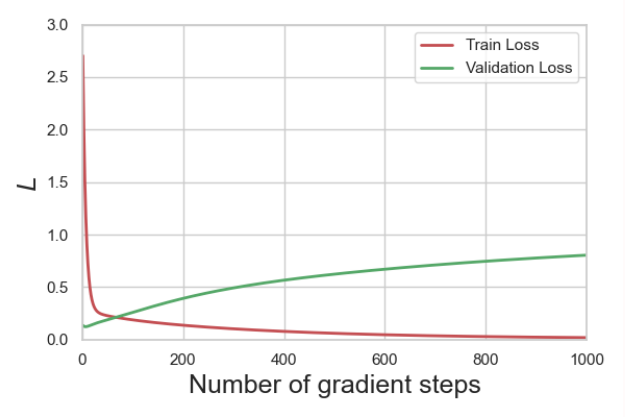
- But: *is zero training loss always desirable?*
- Ultimately, we don't care about training loss, only *test loss*.
- Low training loss is just a *proxy* for successful learning.
- Often the *lowest* training loss is worse than a model with *moderately low* training loss.

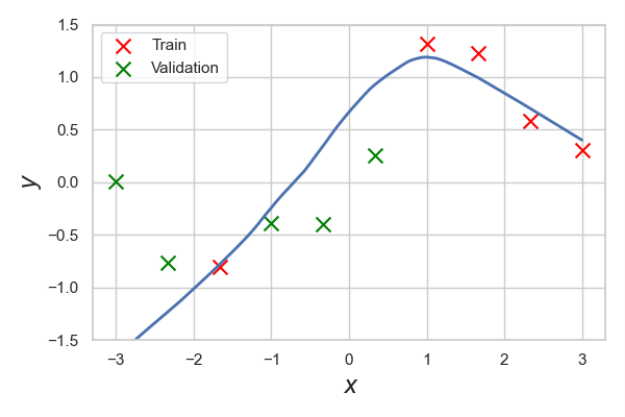


Overfitting vs generalizing fit

Overfitting

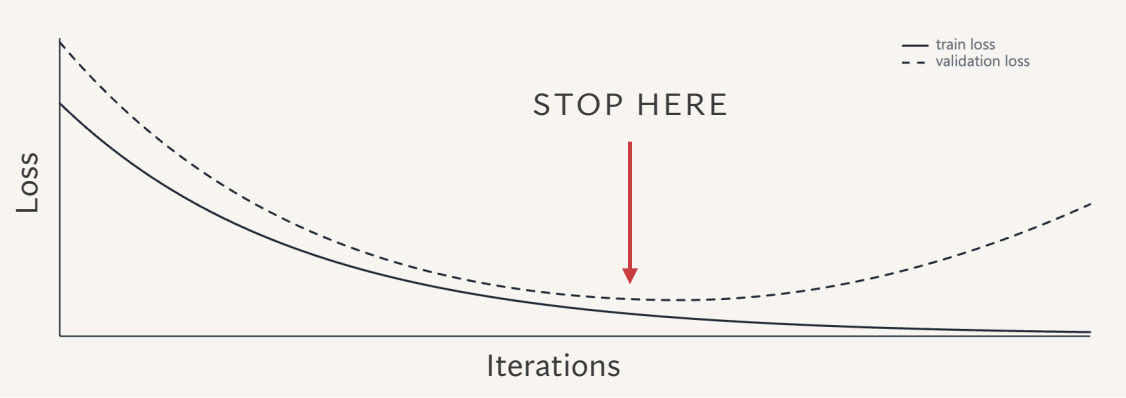
- How do we know if we've fit *just enough* but not *too much?*
- We rely on the VALIDATION LOSS.
- Track the validation loss during training:
  - Validation loss much higher than train loss: OVERFITTING.
  - Validation loss similar to train loss:
    - If both are high: UNDERFITTING—consider adding more neurons, optimizing for longer…
    - If both are low: GENERALIZING—the desired outcome.





Regularization

- REGULARIZATION is any technique that seeks to prevent overfitting.
- Two common methods (out of many):
  - *Weight decay*:
    - Overfitting might be associated with *large* values for the weights.
    - IDEA: add a penalty term to the loss function to keep the weights *small*.
    - $L_{\text{NEW}}(\mathbf{w}) = L(\mathbf{w}) + w_1^2 + w_2^2 + w_3^2 + \ldots + w_N^2$.
  - *Early stopping*:
    - Overfitting might be associated with running gradient descent for *too many iterations*.
    - IDEA: if we train for less time, the model "might not have enough time" to overfit.



Question & Answer