CS/DS 541: Class 3

Jacob Whitehill

Hyperparameter tuning

Hyperparameter tuning

- The values we **optimize** when training a machine learning model — e.g., **w** and *b* for linear regression — are the **parameters** of the model.

- There are also values related to the training process itself — e.g., learning rate ε, batch size $\tilde{n}$, regularization strength α — which are the **hyperparameters** of training.

Hyperparameter tuning

- Both the parameters and hyperparameters can have a huge impact on model performance on test data.
- When estimating the performance of a trained model, it is important to tune both kinds of parameters in a principled way:
    - Training/validation/testing sets
    - Double cross-validation

Training/validation/testing sets

- In an application domain with a large dataset (e.g., 100K examples), it is common to partition it into three subsets:
    - Training (typically 70-80%): optimization of parameters
    - Validation (typically 5-10%): tuning of hyperparameters
    - Testing (typically 5-10%): evaluation of the final model
- For comparison with other researchers' methods, this partition should be fixed.

Training/validation/testing sets

- Hyperparameter tuning works as follows:

1. For each hyperparameter configuration *h*:
    - Train the parameters on the **training** set using *h*.
    - Evaluate the model on the **validation** set.
    - If performance is better than what we got with the best *h* so far (h^*), then save *h* as h^*.
2. Evaluate the model trained with h^* on the **testing** set. (You can train either on training data, or on training+validation data).

Training/validation/testing sets

- Question: Which machine did we evaluate?

Training/validation/testing sets

- Question: Which machine did we evaluate?

- Answer: The one trained in step 2:

2. Evaluate the model trained with h^* on the **testing** set. (You can train either on training data, or on training+validation data).

Cross-validation

- When working with smaller datasets, cross-validation is commonly used so that we can use **all** data for training.
- Assume we already know the best hyperparameters h^*.
- We partition the data into k folds of equal sizes.
- Over k iterations, we train on (k-1) folds and test on the remaining fold.
- We then compute the average accuracy over the k testing folds.

Cross-validation

- \# D=dataset, k=# folds, h=hyperparameter configuration.
  CrossValidation (D, k, h):
      Partition D into k folds F_1, ..., F_k
      For i = 1, ..., k:
          $test = F_i$
          $train = D \setminus F_i$
          Train the model on *train* using h
          $acc[i]$ = Evaluate NN on test
      A = Avg[acc]
      return A

Training/validation/testing sets

- Question: To what machine does the reported accuracy correspond?

Training/validation/testing sets

- Question: To what machine does the reported accuracy correspond?

- Answer: None of them! Cross-validation gives the *expected* accuracy of a classifier that is trained on $(k\text{-}1)/k$ of the data.

Training/validation/testing sets

- Question: To what machine does the reported accuracy correspond?
- Answer: None of them! Cross-validation gives the *expected* accuracy of a classifier that is trained on $(k\text{-}1)/k$ of the data.
- However, we can train another model M using h^* on the entire dataset, and then report A as its accuracy.
- Since M is trained on more data than any of the cross-validation models, its *expected* accuracy should be >= A.

Cross-validation

- But how do we find the best hyperparameters h^* for each fold?

- The typical approach is to use **double cross-validation**, i.e.:

  - For each of the k “outer” folds, run cross-validation in an “inner” loop to determine the best hyperparameter configuration h^* for the kth fold.

Double cross-validation

- # D=dataset, k=# folds, H=set of hyperparameter configurations.
  DoubleCrossValidation (D, k, H):
      Partition D into k folds F_1, ..., F_k

      For each of the k "outer" folds, run cross-validation in an "inner" loop to determine the best hyperparameter configuration h* for the kth fold.

      At the end of the procedure, which "machine" are you evaluating?

      return …

For your reference…

CrossValidation (D, k, h):
    Partition D into k folds F_1, ..., F_k
    For i = 1, ..., k:
        test = F_i
        train = $D \setminus F_i$
        Train the model on *train* using h
        acc[i] = Evaluate NN on test
    A = Avg[*acc*]
    return A

Double cross-validation

- # *D*=dataset, *k*=# folds, *H*=set of hyperparameter configurations.
  DoubleCrossValidation (D, k, H):
      Partition D into k folds $F_1, \ldots, F_k$
      For $i = 1, \ldots, k$:
          $test = F_i$
          $train = D \setminus F_i$
          $A^* = -\infty$
          For h in H:
              A = CrossValidation(*train*, k, h)
              if $A > A^*$:
                  $A^* = A$
                  $h^* = h$
          Train the model on *train* using h^*
          accs[*i*] = Evaluate the model on *test*
      A = Avg[accs]
      return A

For your reference…

CrossValidation (D, k, h):
    Partition D into k folds $F_1, \ldots, F_k$
    For $i = 1, \ldots, k$:
        $test = F_i$
        $train = D \setminus F_i$
        Train the model on *train* using h
        acc[*i*] = Evaluate NN on test
    A = Avg[*acc*]
    return A

Training/validation/testing sets

- Question: To what machine does the reported accuracy correspond?

Training/validation/testing sets

- Question: To what machine does the reported accuracy correspond?
- Answer: None of them!
- In contrast to (single) cross-validation, it's not obvious how to train a model M with accuracy >= A.
- One strategy: return an **ensemble** model whose output is the average of the k models' predictions…but this is rarely done.

Subject independence

- In many machine learning settings, the data are not completely independent from each other — they are *linked* in some way.
- Example:
  - Predict multiple grades for each student based on their Canvas clickstream features (# logins, # forum posts, etc.).

Subject independence

- We could partition the data into folds in different ways:
  - We could randomize across all the data.
  - However, if grades are correlated within each student, then one (or more) training folds can leak information about the testing fold.

	Quiz 1	Quiz 2	Quiz 3
Student 1	45	48	42
Student 2	96	93	93
Student 3	86	86	87
Student 4	10	30	50

Subject independence

- We could partition the data into folds in different ways:
  - Alternatively, we can **stratify** across students, i.e., no student appears in more than 1 fold.
  - With this partition, the cross-validation accuracy estimates the model's performance on a subject *not* used for training.

	Quiz 1	Quiz 2	Quiz 3
Student 1	45	48	42
Student 2	96	93	93
Student 3	86	86	87
Student 4	10	30	50

Optimization of ML models

Optimization of ML models

- Gradient descent is guaranteed to converge to a local minimum (eventually) if the learning rate is small enough relative to the steepness of *f*.

- A function $f : \mathbb{R}^m \rightarrow \mathbb{R}$ is Lipschitz-continuous if:

$$\exists L : \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^m : \|f(\mathbf{x}) - f(\mathbf{y})\|_2 \leq L\|\mathbf{x} - \mathbf{y}\|_2$$

- *L* is essentially an upper bound on the absolute slope of *f*.

Optimization of ML models

- Gradient descent is guaranteed to converge to a local minimum (eventually) if the learning rate is small enough relative to the steepness of *f*.

- A function $f : \mathbb{R}^m \to \mathbb{R}$ is Lipschitz-continuous if:

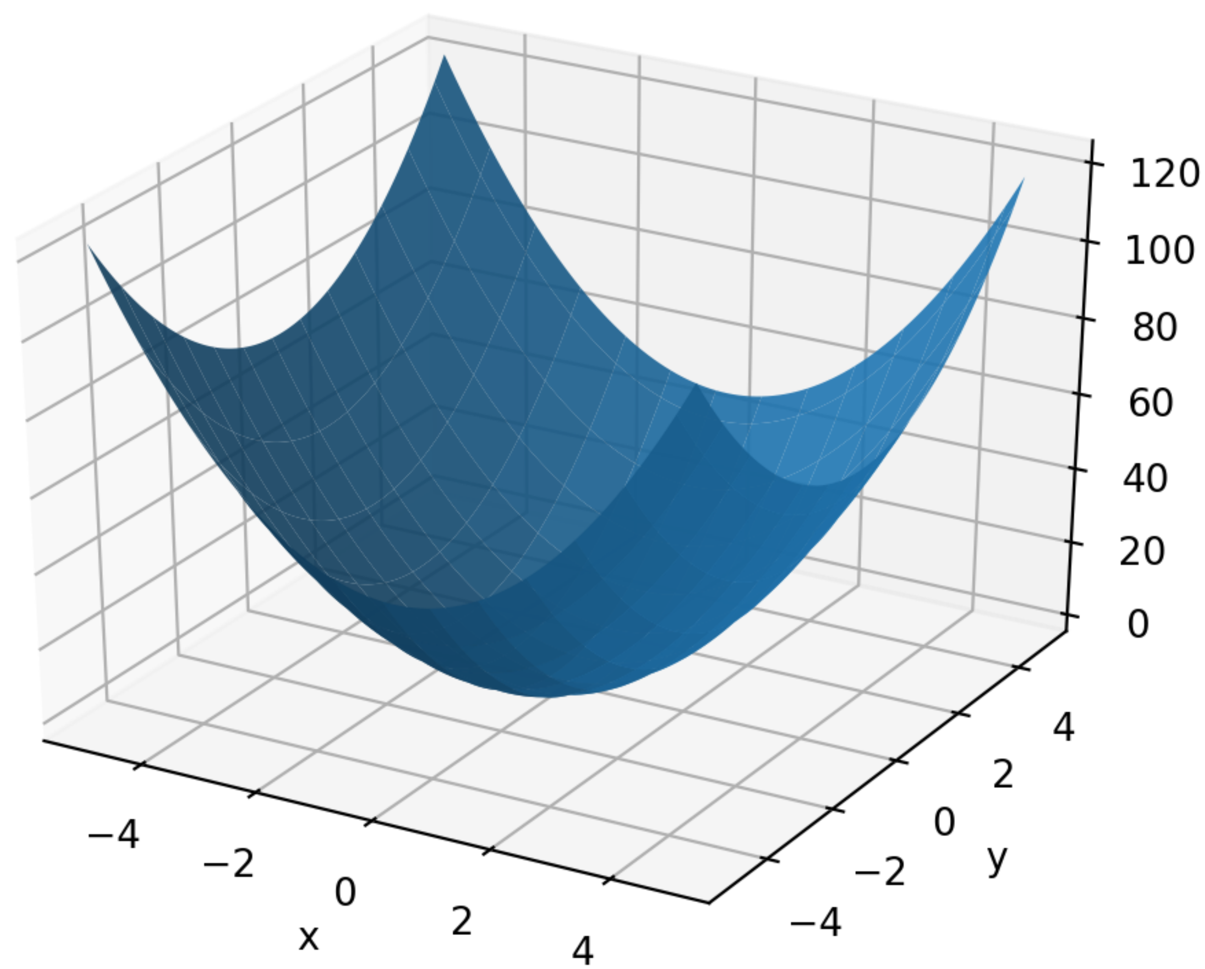
$$\exists L : \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^m : \|f(\mathbf{x}) - f(\mathbf{y})\|_2 \leq L\|\mathbf{x} - \mathbf{y}\|_2$$

- *L* is essentially an upper bound on the absolute slope of *f*.

- For learning rate $\epsilon \leq \frac{1}{L}$, gradient descent will converge to a local minimum linearly, i.e., the error is *O*(1/*k*) in the iterations *k*.

Optimization of ML models

- With linear regression, the cost function f_{MSE} has a single local minimum w.r.t. the weights **w**:



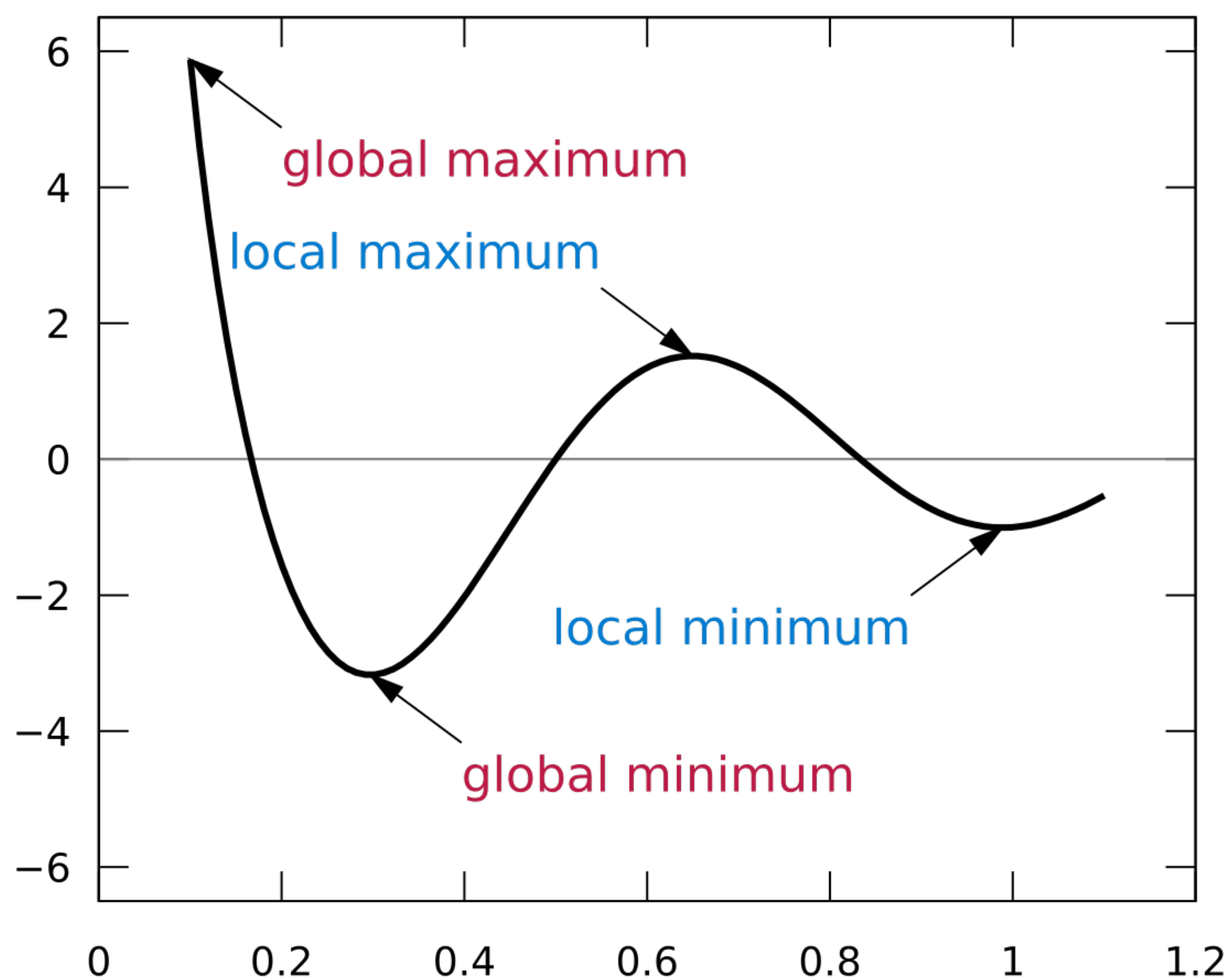
- As long as our learning rate is small enough, we will find **the optimal w**.

Optimization: what can go wrong?

- In general ML and DL models, optimization is usually not so simple, due to:

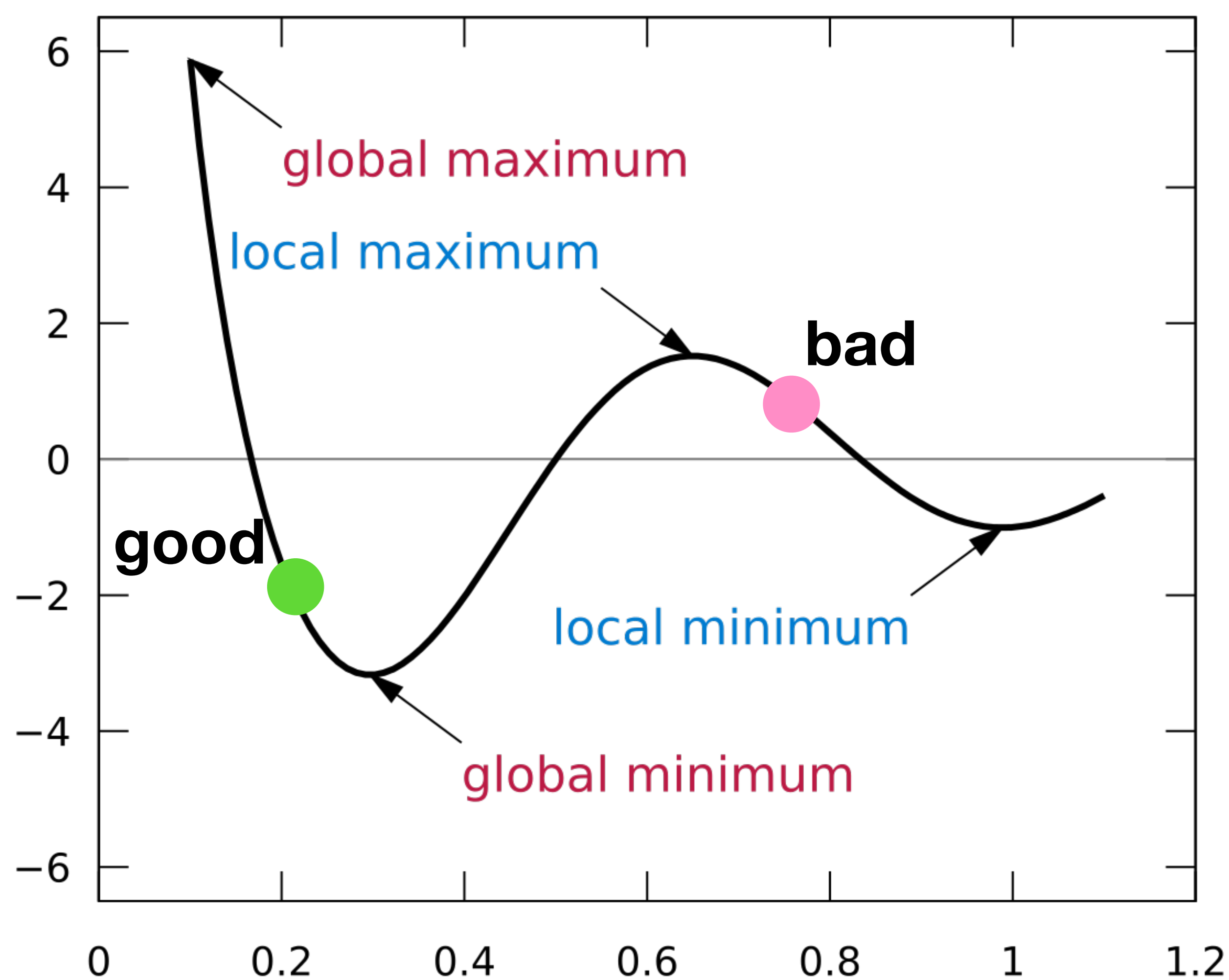
Optimization: what can go wrong?

- In general ML and DL models, optimization is usually not so simple, due to:
  1. Presence of multiple local minima



Optimization: what can go wrong?

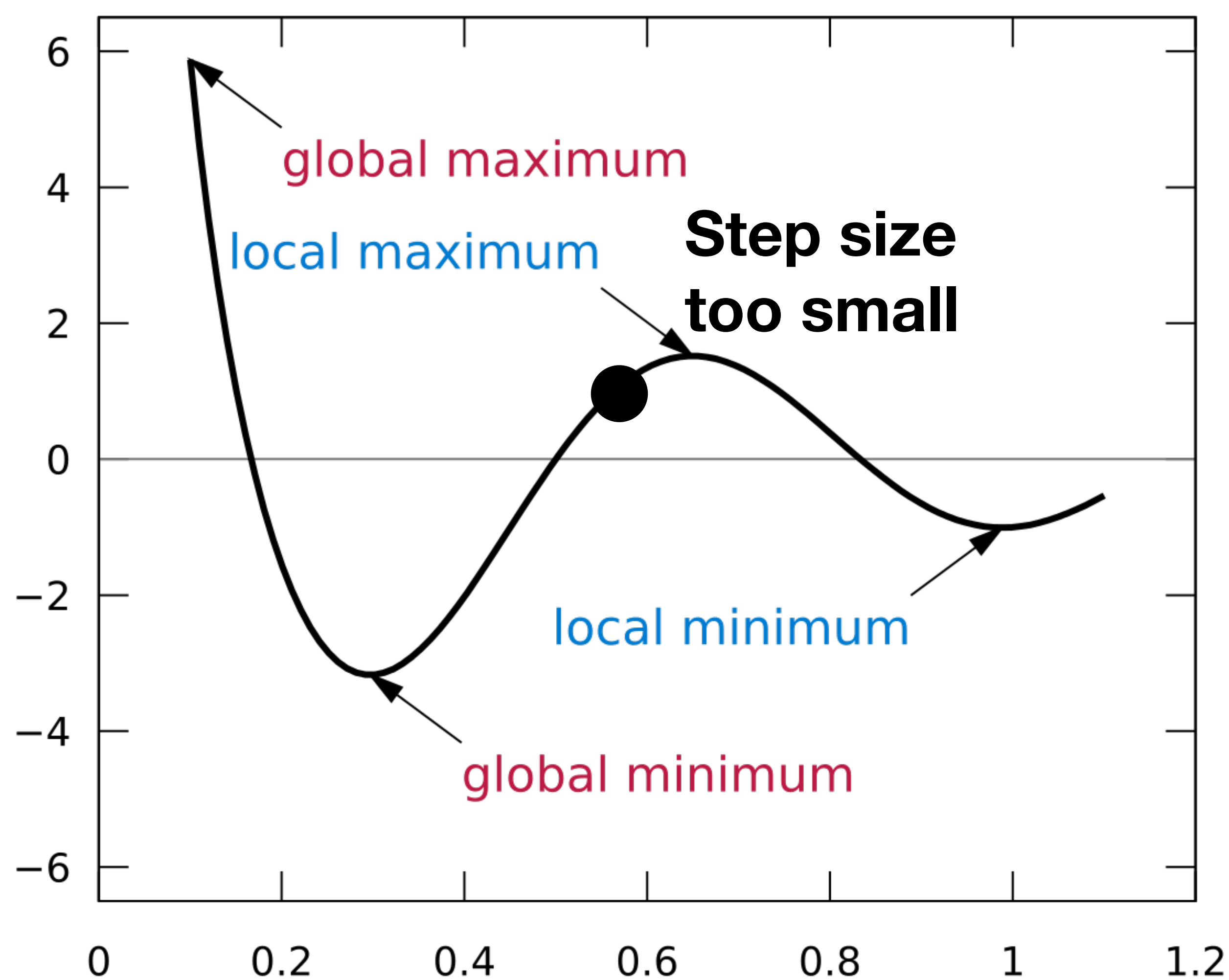
- In general ML and DL models, optimization is usually not so simple, due to:
  2. Bad initialization of the weights **w**.



Optimization: what can go wrong?

- In general ML and DL models, optimization is usually not so simple, due to:

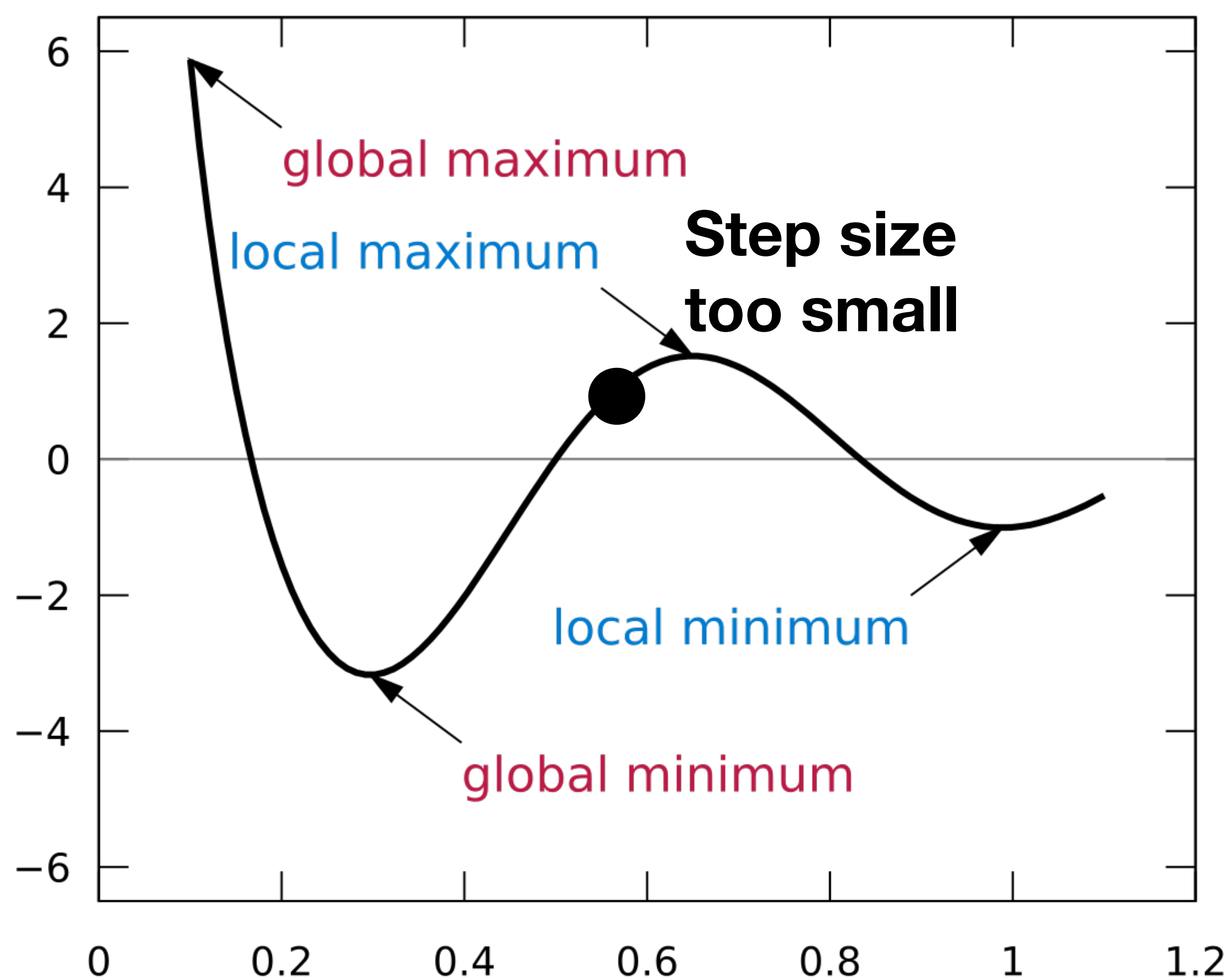
  3. Learning rate is too small.



Optimization: what can go wrong?

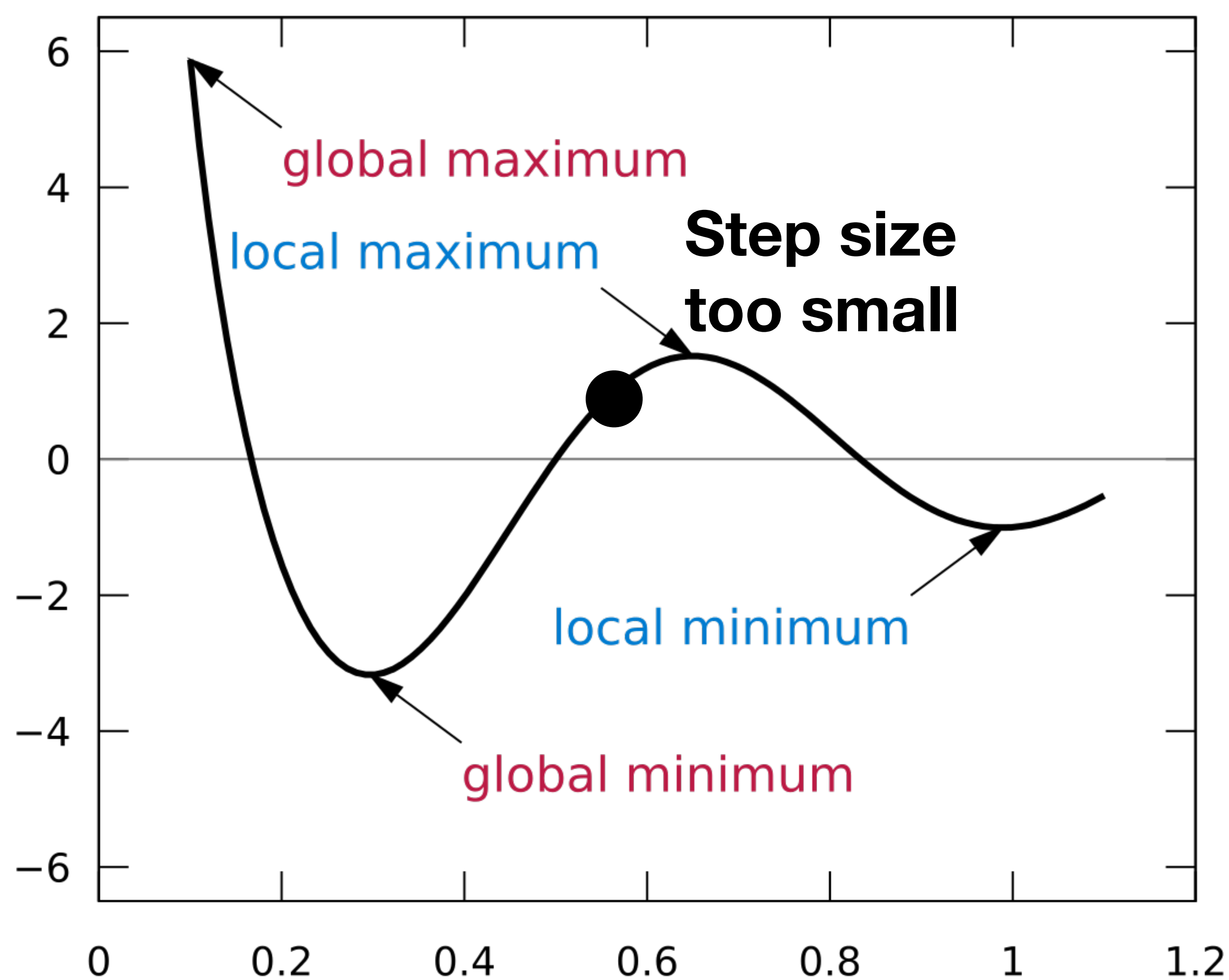
- In general ML and DL models, optimization is usually not so simple, due to:

  3. Learning rate is too small.



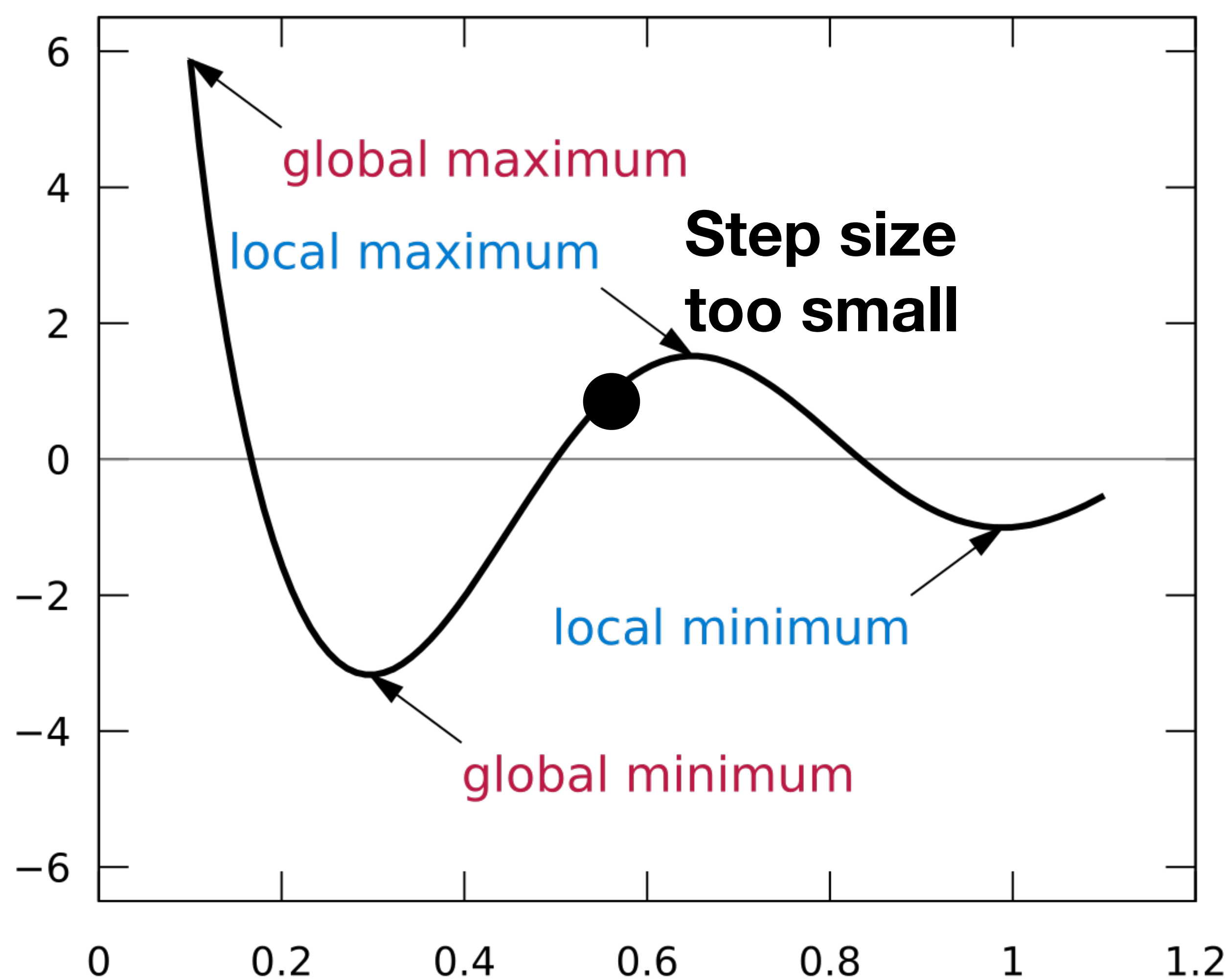
Optimization: what can go wrong?

- In general ML and DL models, optimization is usually not so simple, due to:
    3. Learning rate is too small.



Optimization: what can go wrong?

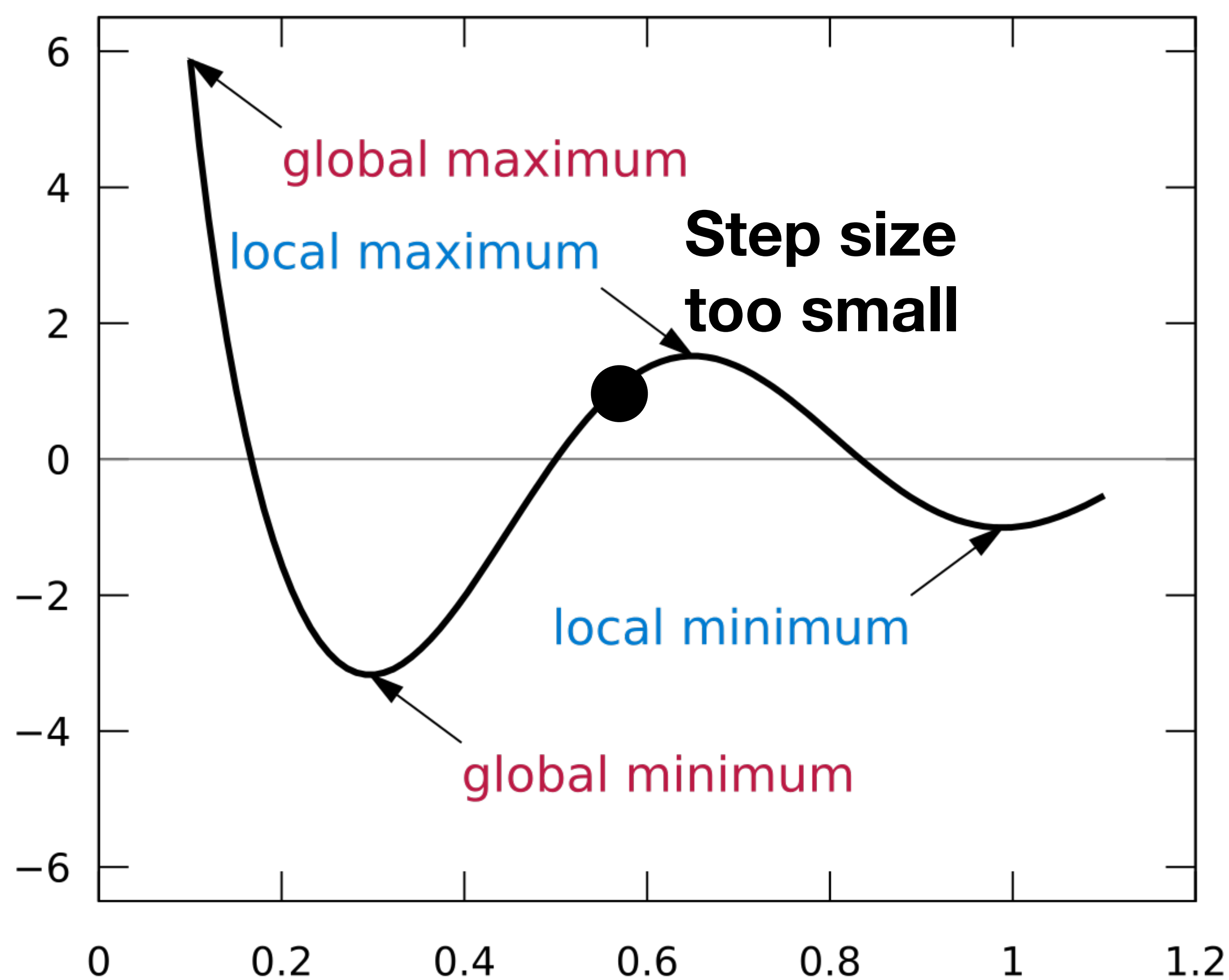
- In general ML and DL models, optimization is usually not so simple, due to:
   3. Learning rate is too small.



Optimization: what can go wrong?

- In general ML and DL models, optimization is usually not so simple, due to:

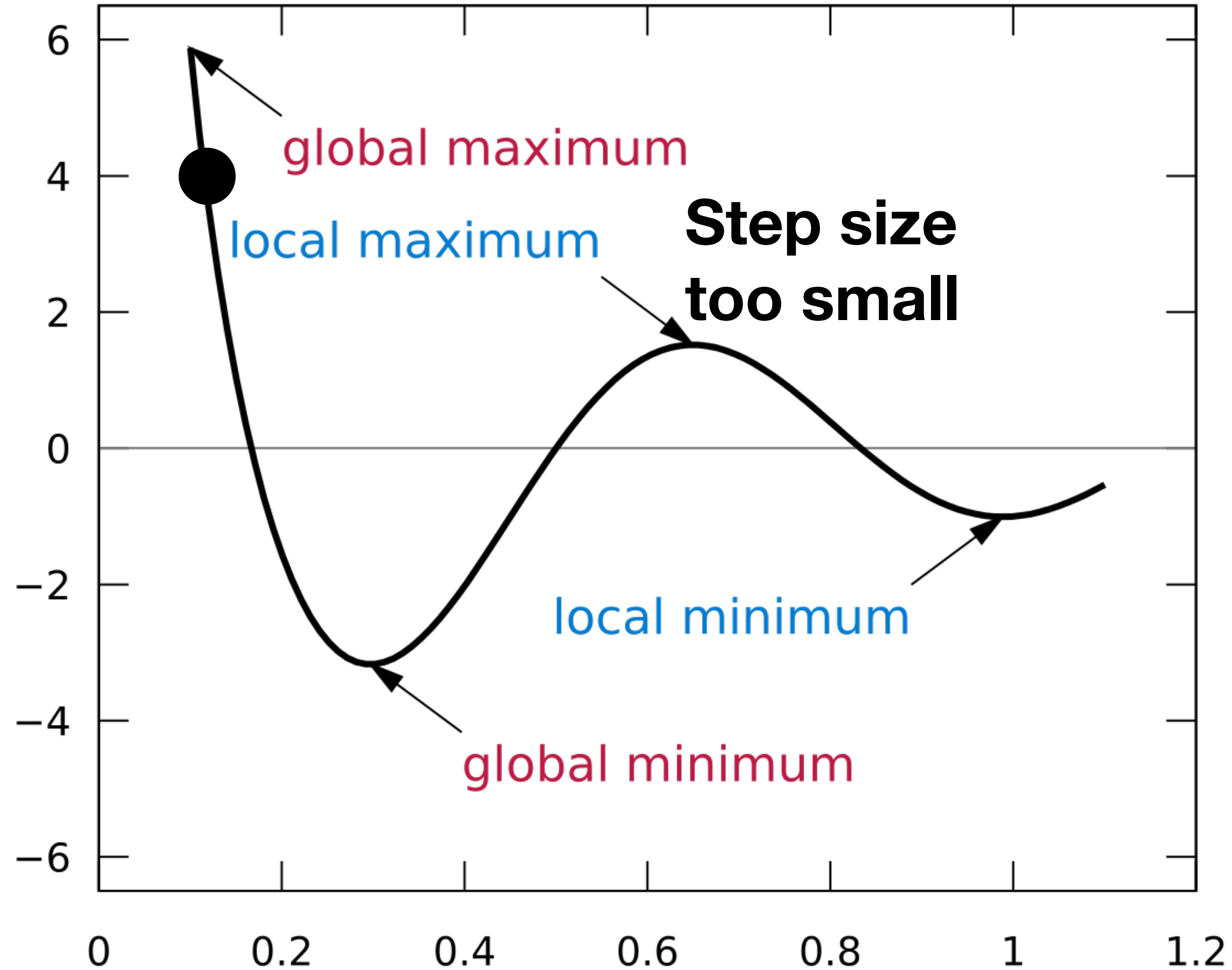
  4. Learning rate is too big.



Optimization: what can go wrong?

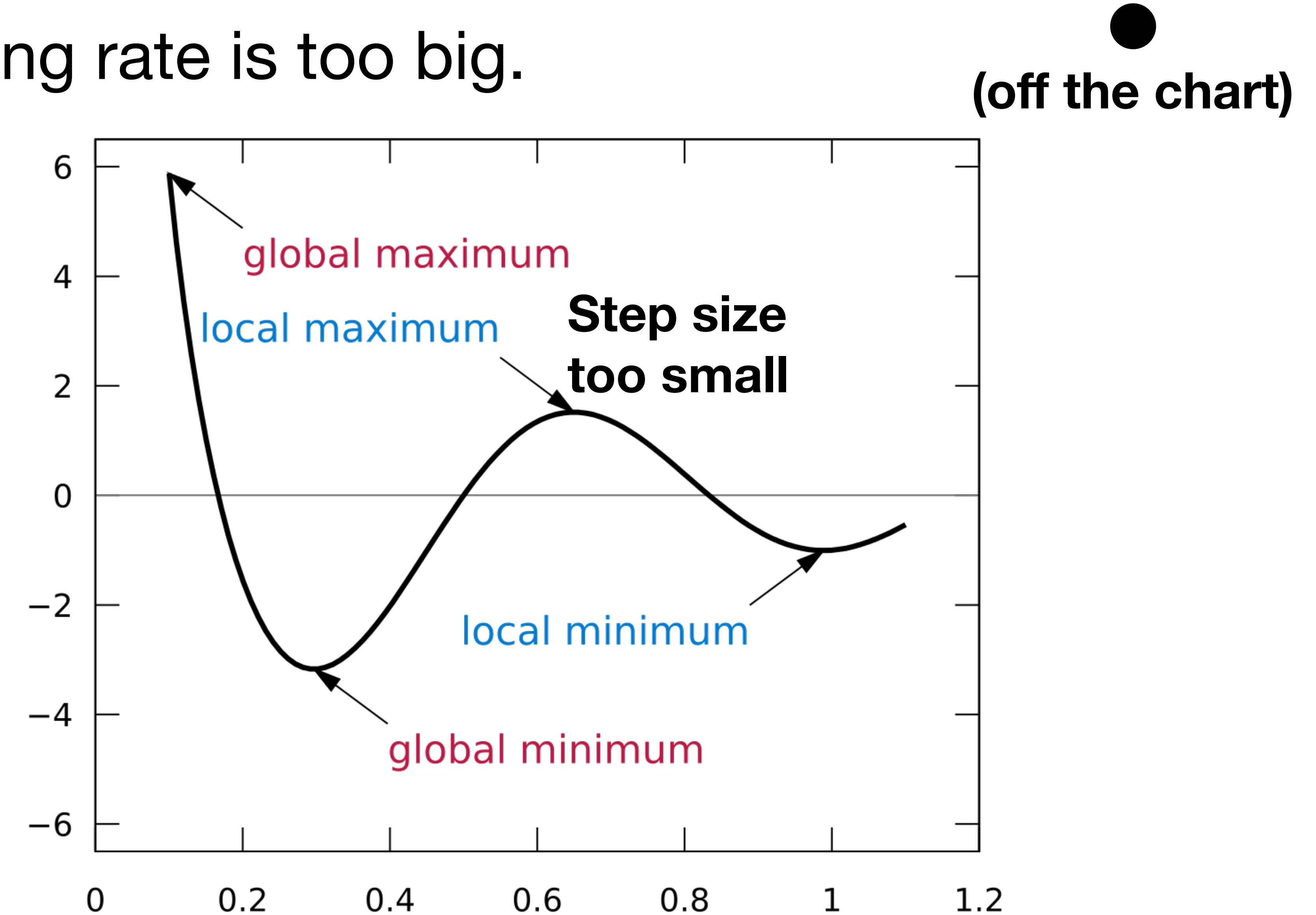
- In general ML and DL models, optimization is usually not so simple, due to:

    4. Learning rate is too big.



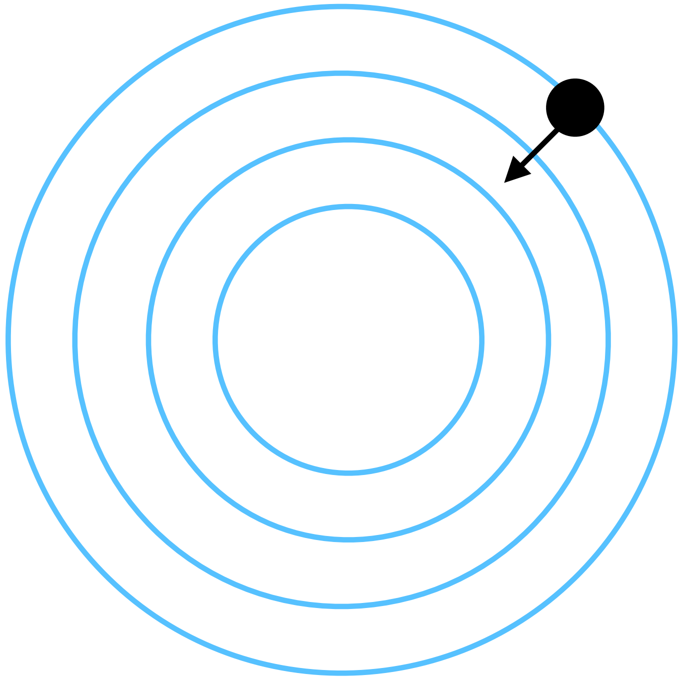
Optimization: what can go wrong?

- In general ML and DL models, optimization is usually not so simple, due to:
    4. Learning rate is too big.



Optimization: what can go wrong?

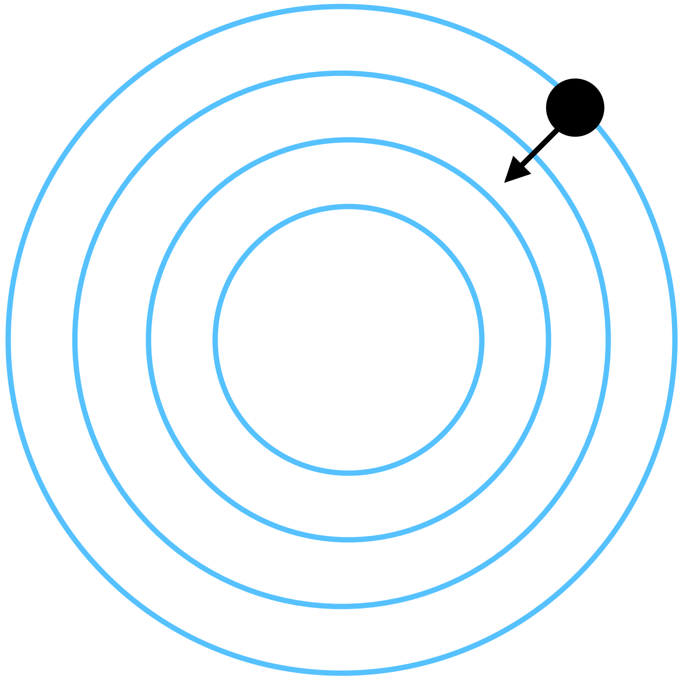
- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.
- Consider the cost *f* whose level sets are shown below:



Optimization: what can go wrong?

- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.

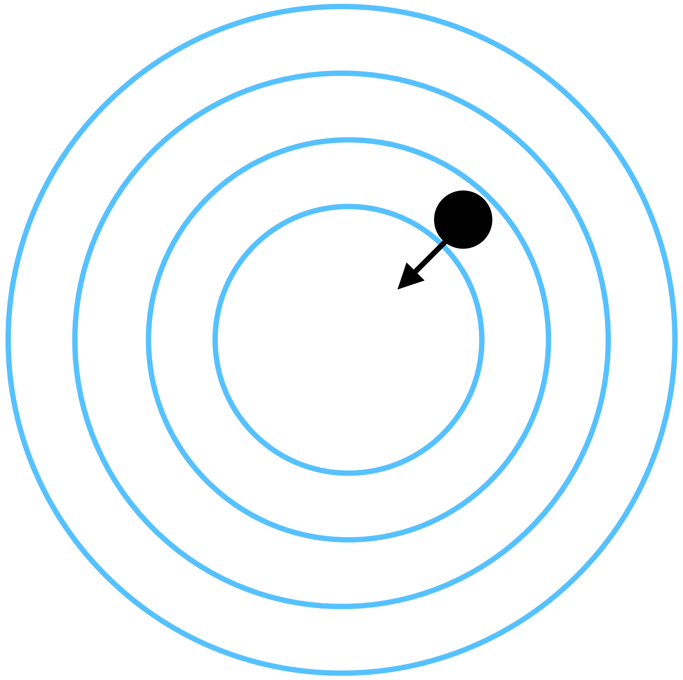
- Gradient descent guides the search along the direction of steepest decrease in *f*.



Optimization: what can go wrong?

- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.

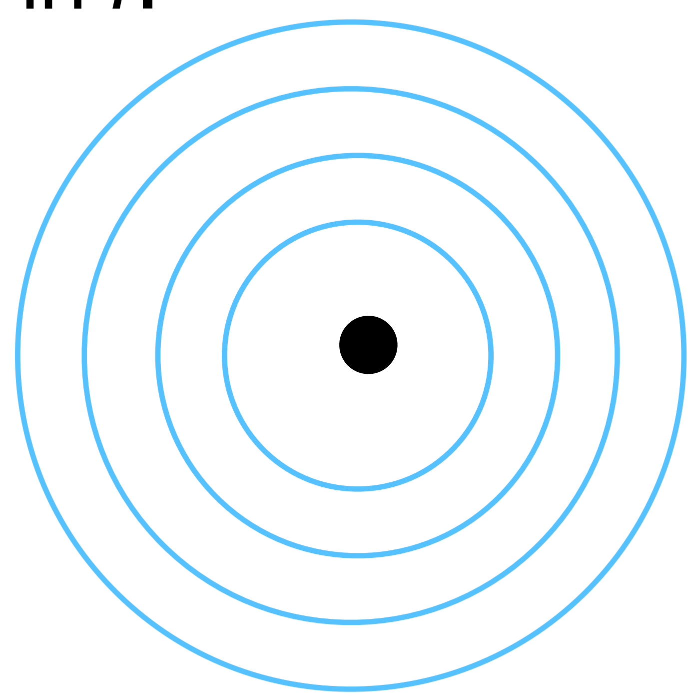
- Gradient descent guides the search along the direction of steepest decrease in *f*.



Optimization: what can go wrong?

- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.

- Gradient descent guides the search along the direction of steepest decrease in *f*.



Optimization: what can go wrong?

- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.

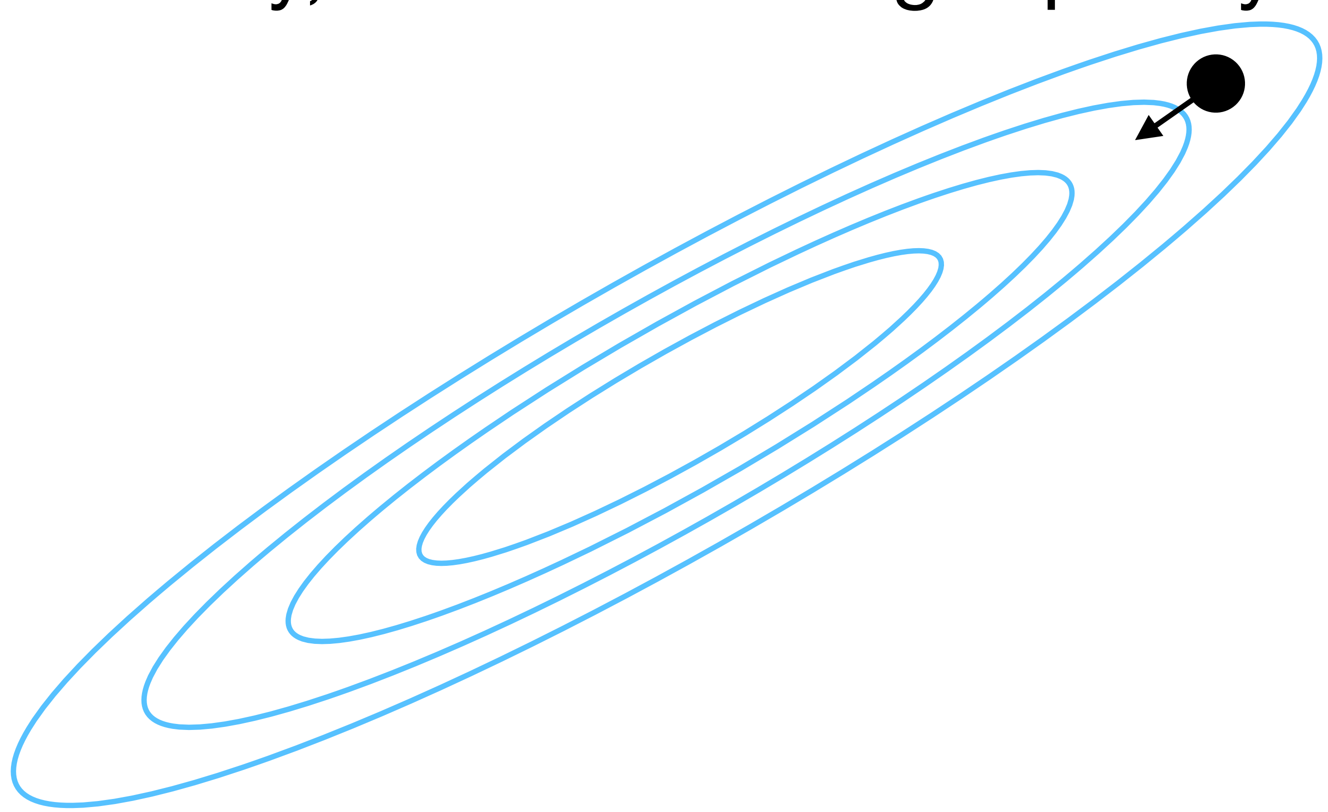
- But what if the level sets are ellipsoids instead of spheres?

Optimization: what can go wrong?

- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.

- But what if the level sets are ellipsoids instead of spheres?

  - If we are lucky, we still converge quickly.

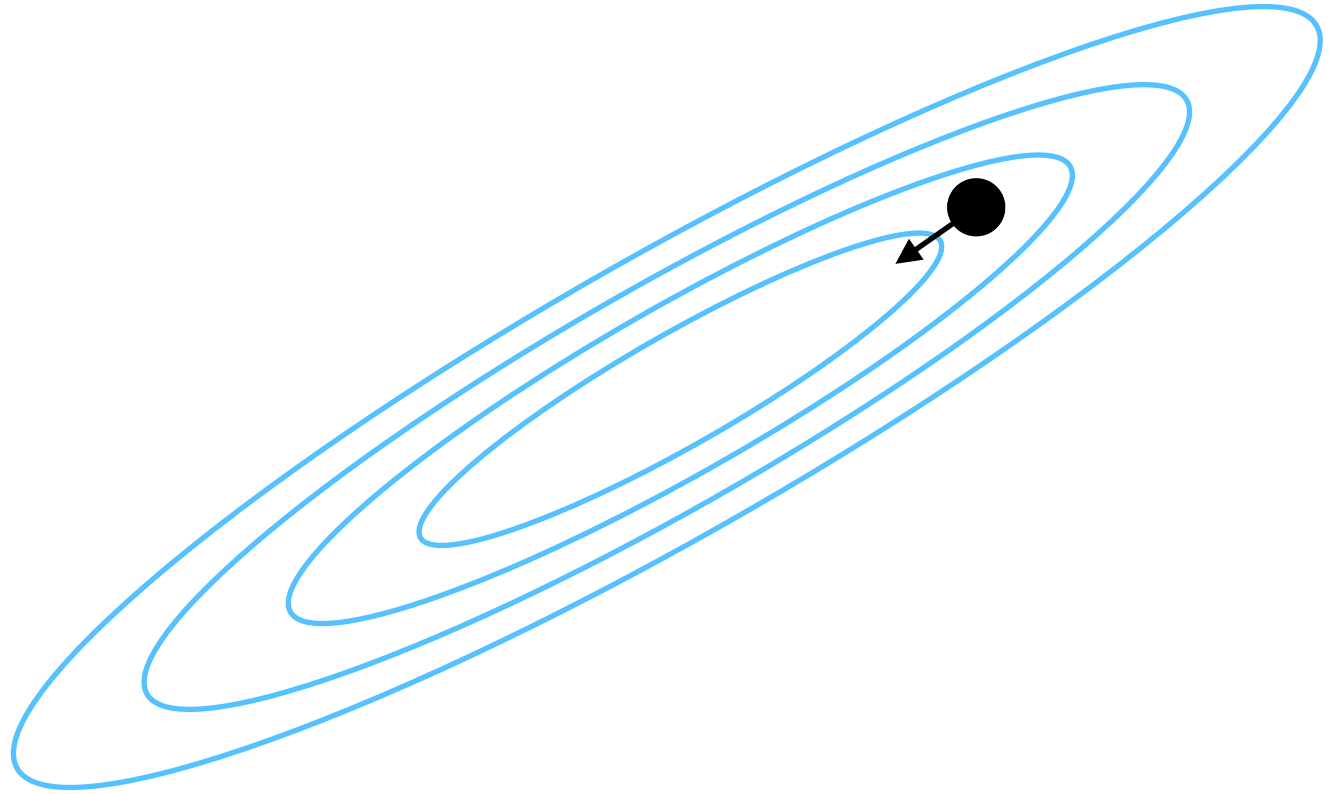


Optimization: what can go wrong?

- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.

- But what if the level sets are ellipsoids instead of spheres?

  - If we are lucky, we still converge quickly.

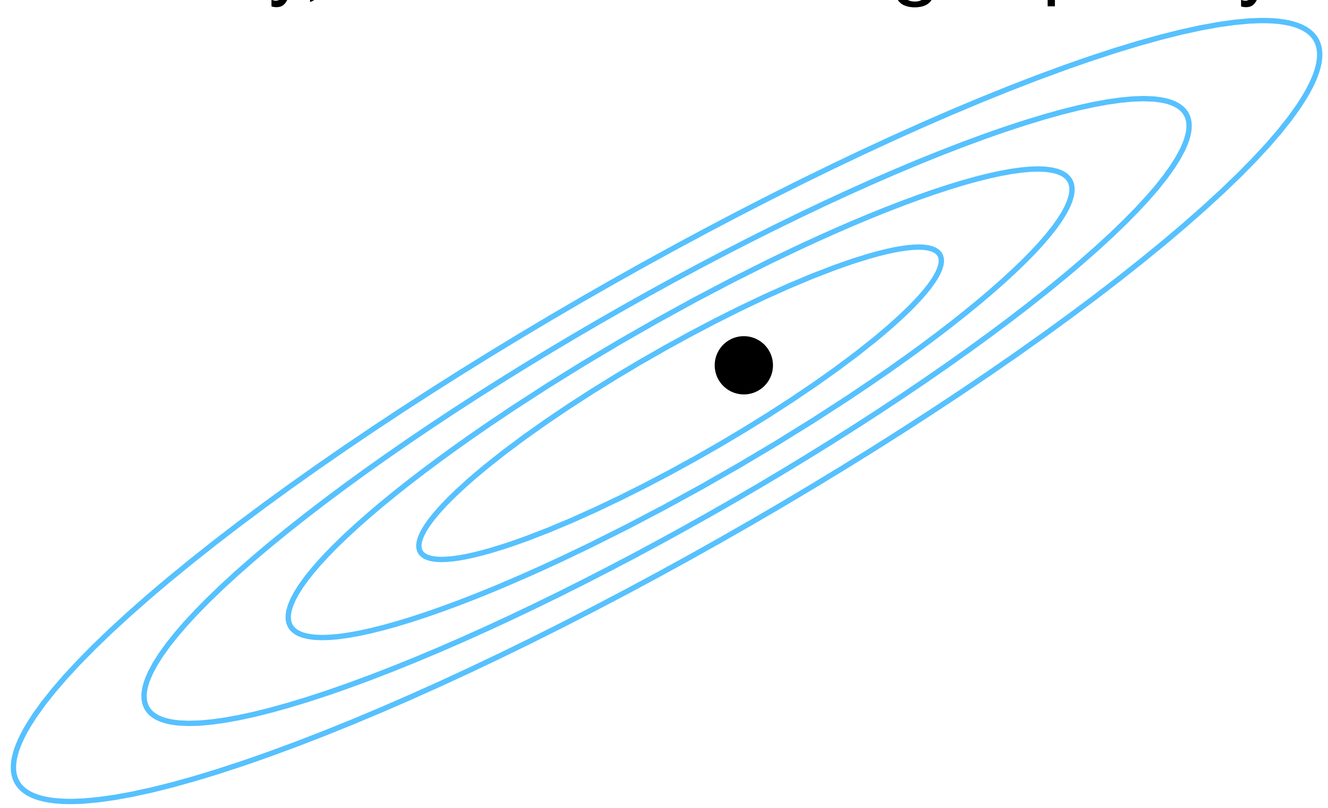


Optimization: what can go wrong?

- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.

- But what if the level sets are ellipsoids instead of spheres?

    - If we are lucky, we still converge quickly.



Optimization: what can go wrong?

- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.

- But what if the level sets are ellipsoids instead of spheres?

    - If we are unlucky, convergence is very slow.

Optimization: what can go wrong?

- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.

- But what if the level sets are ellipsoids instead of spheres?

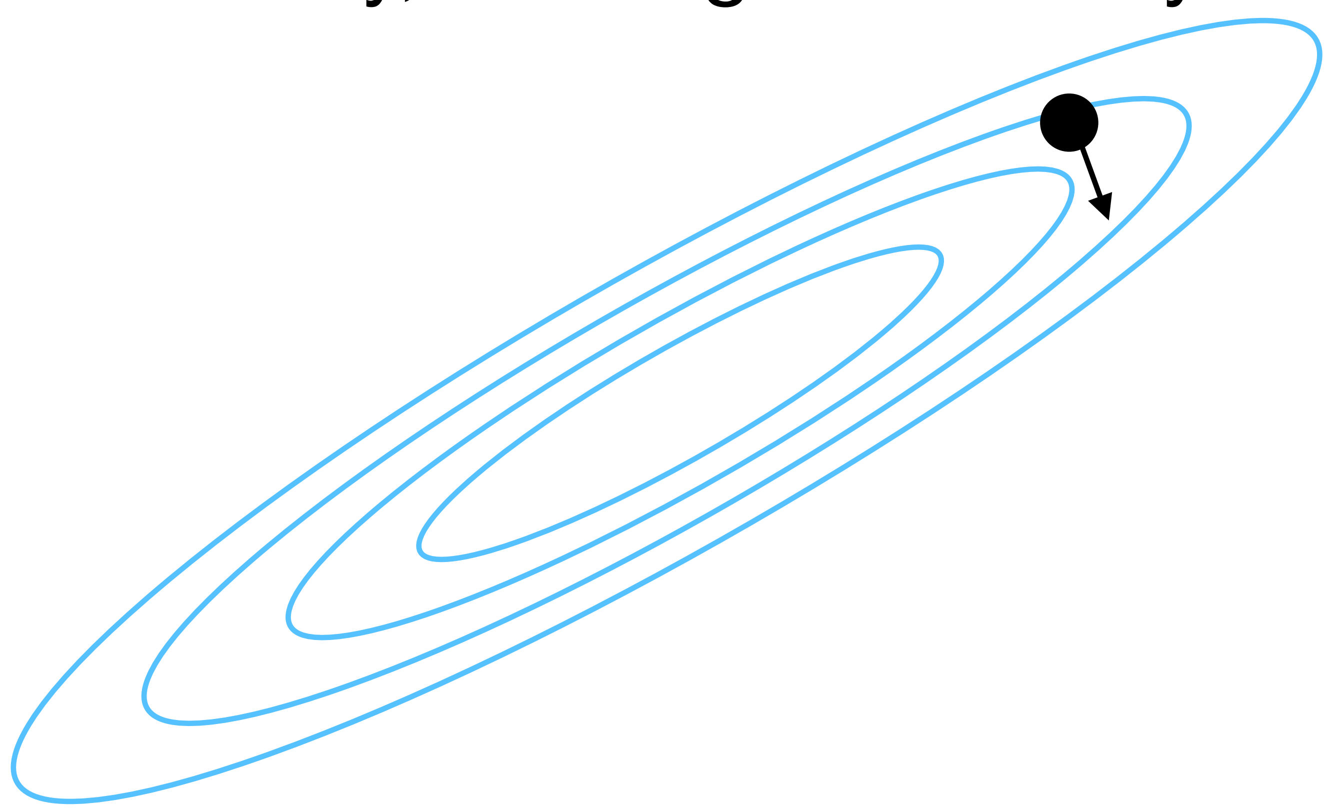
  - If we are unlucky, convergence is very slow.

Optimization: what can go wrong?

- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.

- But what if the level sets are ellipsoids instead of spheres?

  - If we are unlucky, convergence is very slow.

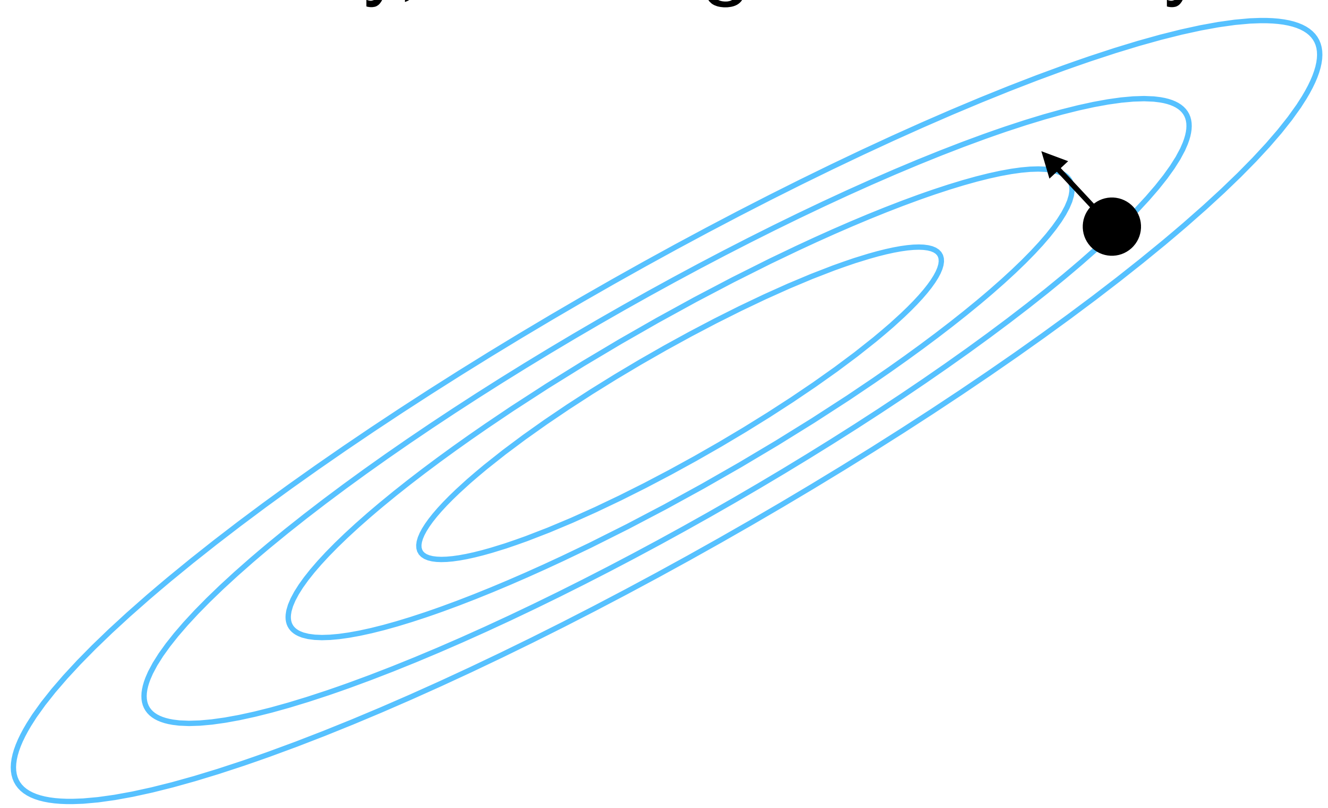


Optimization: what can go wrong?

- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.

- But what if the level sets are ellipsoids instead of spheres?

    - If we are unlucky, convergence is very slow.

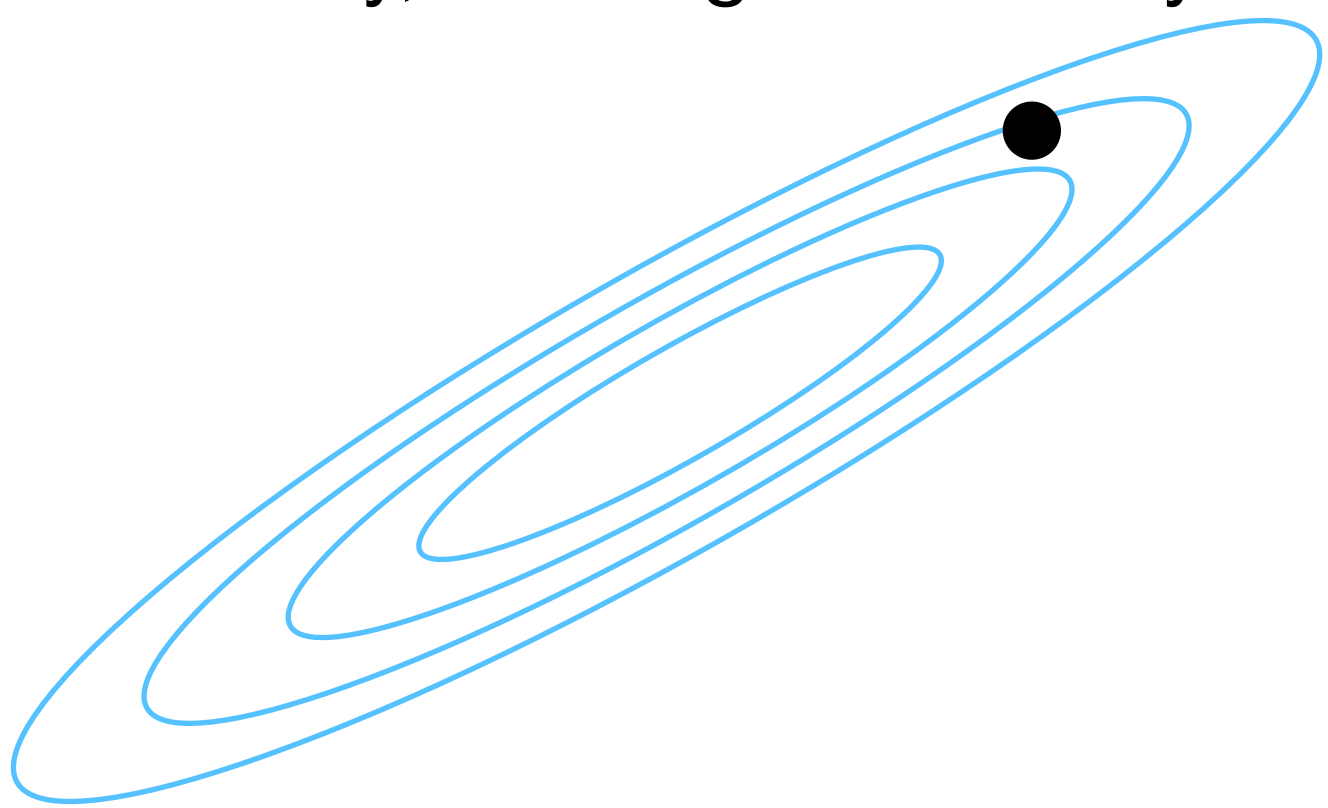


Optimization: what can go wrong?

- With multidimensional weight vectors, badly chosen learning rates can cause more subtle problems.

- But what if the level sets are ellipsoids instead of spheres?

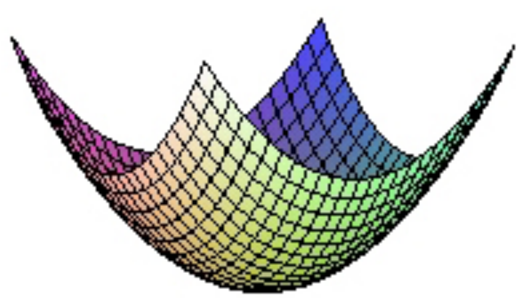
    - If we are unlucky, convergence is very slow.



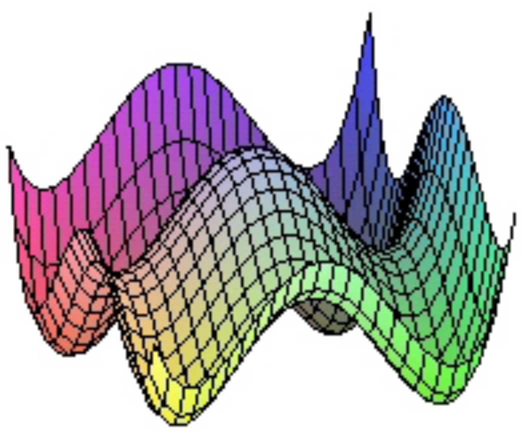
Convexity

Convex ML models

- Linear regression has a loss function that is **convex**.
- With a convex function *f*, *every local minimum is also a global minimum*.



convex



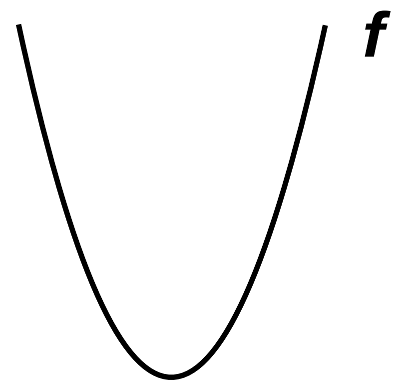
non-convex

- Convex functions are ideal for conducting gradient descent.

https://plus.maths.org/content/convexity

Convexity in 1-d

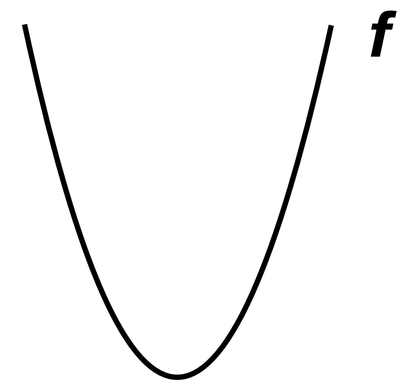
- How can we tell if a 1-d function *f* is convex?



- What property of *f* ensures there is only one local minimum?

Convexity in 1-d

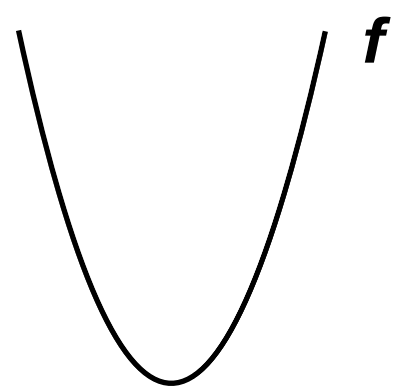
- How can we tell if a 1-d function *f* is convex?



- What property of *f* ensures there is only one local minimum?
    - From left to right, the slope of *f* *never decreases*.

Convexity in 1-d

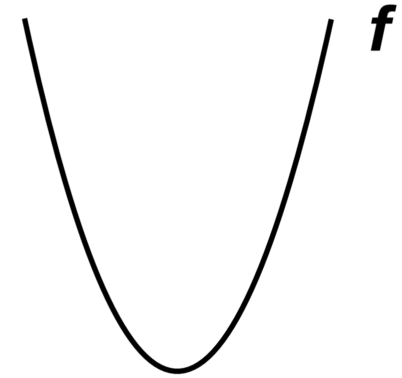
- How can we tell if a 1-d function *f* is convex?



- What property of *f* ensures there is only one local minimum?
    - From left to right, the slope of *f* *never decreases*.
      ==> the derivative of the slope is always non-negative.

Convexity in 1-d

- How can we tell if a 1-d function *f* is convex?



- What property of *f* ensures there is only one local minimum?
  - From left to right, the slope of *f* *never decreases*.
    ==> the derivative of the slope is always non-negative.
    ==> the second derivative of *f* is always non-negative.

Convexity in higher dimensions

- For higher-dimensional *f*, convexity is determined by the the Hessian of *f*.

$$\mathrm{H}[f] = \left[\begin{array}{ccc} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_m} \\ \dots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_m \partial x_1} & \dots & \frac{\partial^2 f}{\partial x_m \partial x_m} \end{array} \right]$$

- For $f : \mathbb{R}^m \rightarrow \mathbb{R}$, *f* is convex if the Hessian matrix is positive semi-definite for *every* input **x**.

Positive semi-definite

- Positive semi-definite is the matrix analog of being “non-negative”.

- A real symmetric matrix **A** is **positive semi-definite (PSD)** if (equivalent conditions):

Positive semi-definite

- Positive semi-definite is the matrix analog of being “non-negative”.

- A real symmetric matrix **A** is **positive semi-definite (PSD)** if (equivalent conditions):
  - All its eigenvalues are ≥ 0.
    - If **A** happens to be diagonal, then its eigenvalues are the diagonal elements.

Positive semi-definite

- Positive semi-definite is the matrix analog of being "non-negative".

- A real symmetric matrix **A** is **positive semi-definite (PSD)** if (equivalent conditions):
  - All its eigenvalues are ≥ 0.
    - If **A** happens to be diagonal, then its eigenvalues are the diagonal elements.
  - For every vector **v**: $\mathbf{v}^\top \mathbf{A} \mathbf{v} \geq 0$
    - Therefore: If there exists *any* vector **v** such that $\mathbf{v}^\top \mathbf{A} \mathbf{v} < 0$, then **A** is *not* PSD.

Example

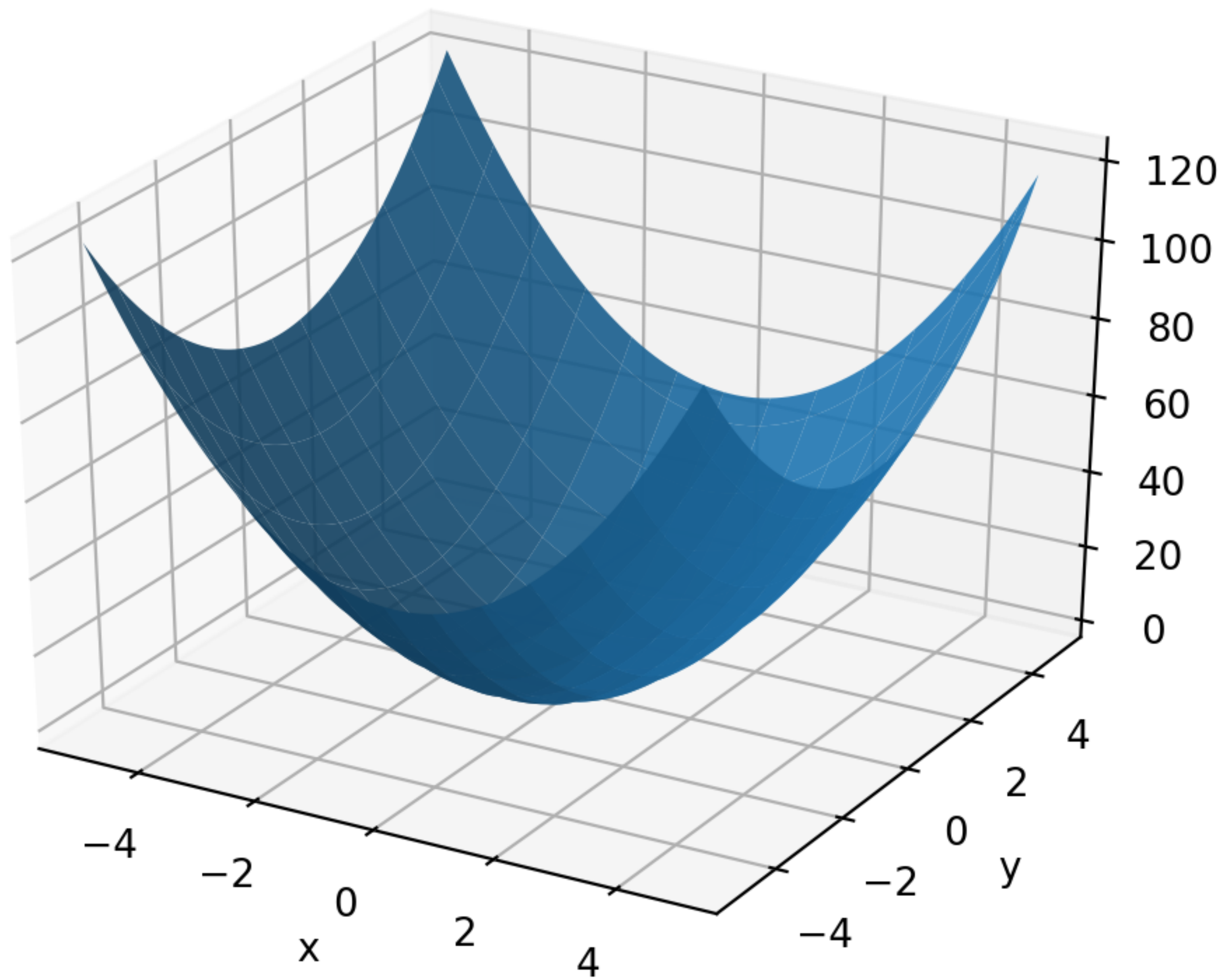
- Suppose $f(x, y) = 3x^2 + 2y^2 - 2$.
- Then the first derivatives are: $\frac{\partial f}{\partial x} = 6x \quad \frac{\partial f}{\partial y} = 4y$
- The Hessian matrix is therefore:

$$\mathbf{H} = \left[\begin{array}{cc} \frac{\partial^2 f}{\partial x \partial x} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y \partial y} \end{array} \right] = \left[\begin{array}{cc} 6 & 0 \\ 0 & 4 \end{array} \right]$$

- Notice that **H** for this *f* does not depend on (x,y).
- Also, **H** is a diagonal matrix (with 6 and 4 on the diagonal). Hence, the eigenvalues are just 6 and 4. Since they are both non-negative, then *f* is convex.

Example

- Graph of $f(x, y) = 3x^2 + 2y^2 - 2$:



Exercise

- Recall: if **H** is the Hessian of *f*, then *f* is convex if — at *every* (x,y), we can show (equivalently):
  - $\mathbf{v}^T\mathbf{H}\mathbf{v} \geq 0$ for *every* $\mathbf{v}$
  - All eigenvalues of **H** are non-negative.

- Which of the following function(s) are convex?

  - $x^2 + y + 5$

  - $x^4 + xy + x^2$

Exercise

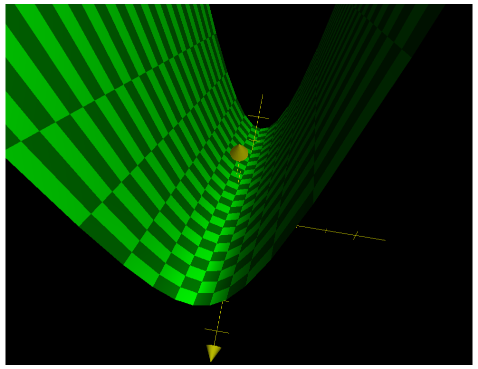
- Recall: if **H** is the Hessian of *f*, then *f* is convex if — at *every* (x,y), we can show (equivalently):
  - $\mathbf{v}^T\mathbf{H}\mathbf{v} \geq 0$ for *every* $\mathbf{v}$
  - All eigenvalues of **H** are non-negative.

- Which of the following function(s) are convex?
  - $x^2 + y + 5$ $\quad \mathbf{H} = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$
  - $x^4 + xy + x^2$

Exercise

- Recall: if **H** is the Hessian of *f*, then *f* is convex if — at *every* (x,y), we can show (equivalently):
  - $\mathbf{v}^T\mathbf{H}\mathbf{v} \geq 0$ for *every* $\mathbf{v}$
  - All eigenvalues of **H** are non-negative.

- Which of the following function(s) are convex?
  - $x^2 + y + 5$ $\quad \mathbf{H} = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$ **Eigenvalues are 2, 0 => PSD.**



  - $x^4 + xy + x^2$

Exercise

- Recall: if **H** is the Hessian of *f*, then *f* is convex if — at *every* (x,y), we can show (equivalently):
  - $\mathbf{v}^T\mathbf{H}\mathbf{v} \geq 0$ for *every* $\mathbf{v}$
  - All eigenvalues of **H** are non-negative.

- Which of the following function(s) are convex?
  - $x^2 + y + 5$
  - $x^4 + xy + x^2$ $\quad \mathbf{H} = \left[\begin{array}{cc} 12x^2 + 2 & 1 \\ 1 & 0 \end{array} \right]$

Exercise

- Recall: if **H** is the Hessian of *f*, then *f* is convex if — at *every* (x,y), we can show (equivalently):
  - $\mathbf{v}^\top\mathbf{H}\mathbf{v} \geq 0$ for *every* **v**
  - All eigenvalues of **H** are non-negative.

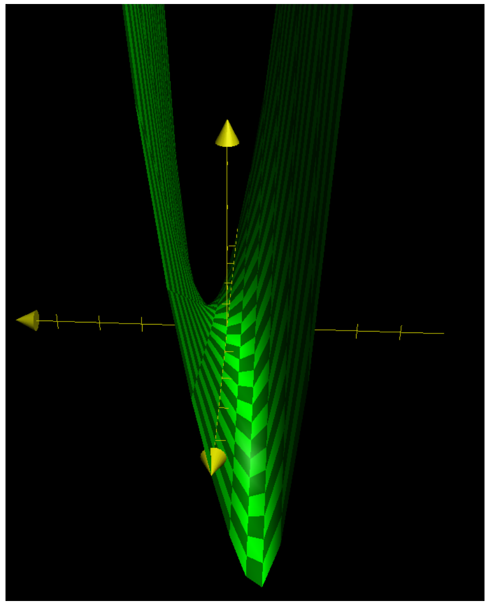
- Which of the following function(s) are convex?
  - $x^2 + y + 5$
  - $x^4 + xy + x^2$ $\quad \mathbf{H} = \left[\begin{array}{cc} 12x^2+2 & 1 \\ 1 & 0 \end{array}\right]$ $\quad x = 1$ $\quad \mathbf{v} = \left[\begin{array}{c} -1 \\ 15 \end{array}\right]$ $\quad \mathbf{v}^\top\mathbf{H}\mathbf{v} = -16$

    Not PSD.

Exercise

- Recall: if **H** is the Hessian of *f*, then *f* is convex if — at *every* (x,y), we can show (equivalently):
  - $\mathbf{v}^\top \mathbf{H} \mathbf{v} \geq 0$ for *every* **v**
  - All eigenvalues of **H** are non-negative.

- Which of the following function(s) are convex?



  - $x^2 + y + 5$
  - $x^4 + xy + x^2$ $\quad \mathbf{H} = \left[\begin{array}{cc} 12x^2 + 2 & 1 \\ 1 & 0 \end{array} \right]$ $\quad x = 1 \quad \mathbf{v} = \left[\begin{array}{c} -1 \\ 15 \end{array} \right]$ $\quad \mathbf{v}^\top \mathbf{H} \mathbf{v} = -16$

Not PSD.

Convexity of linear regression

- How do we know linear regression is a convex ML model?
- First, recall that, for any matrices **A**, **B** that can be multiplied:
  - $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$

Convexity of linear regression

- How do we know linear regression is a convex ML model?

- Next, recall the gradient and Hessian of f_{MSE} (for linear regression):

$$\begin{aligned}\nabla_{\mathbf{w}} f_{\text{MSE}} &= \mathbf{X}(\hat{\mathbf{y}} - \mathbf{y}) \\ &= \mathbf{X}(\mathbf{X}^\top \mathbf{w} - \mathbf{y}) \\ \mathbf{H} &= \mathbf{X}\mathbf{X}^\top\end{aligned}$$

Convexity of linear regression

- How do we know linear regression is a convex ML model?

- Next, recall the gradient and Hessian of f_{MSE} (for linear regression):

$$\begin{aligned} \nabla_{\mathbf{w}} f_{\mathrm{MSE}} &= \mathbf{X}(\hat{\mathbf{y}} - \mathbf{y}) \\ &= \mathbf{X}(\mathbf{X}^\top \mathbf{w} - \mathbf{y}) \\ \mathbf{H} &= \mathbf{X}\mathbf{X}^\top \end{aligned}$$

- For any vector **v**, we have:

$$\begin{aligned} \mathbf{v}^\top \mathbf{X}\mathbf{X}^\top \mathbf{v} &= \left(\mathbf{X}^\top \mathbf{v}\right)^\top \left(\mathbf{X}^\top \mathbf{v}\right) \\ &\geq 0 \end{aligned}$$

Convex ML models

- Prominent convex models in ML include linear regression, logistic regression, softmax regression, and support vector machines (SVM).

- However, models in deep learning are generally not convex.

    - Much DL research is devoted to how to optimize the weights to deliver good generalization performance.

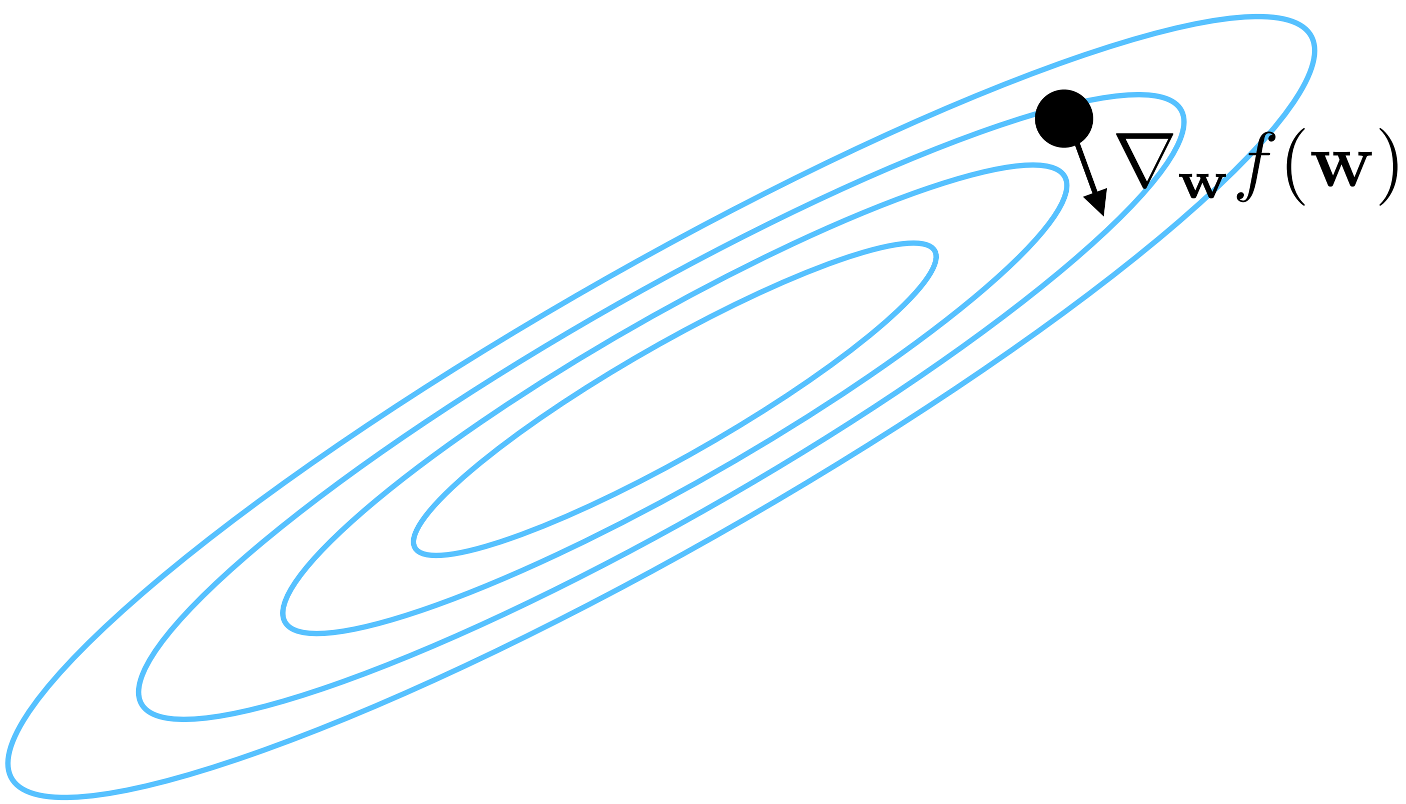
Non-spherical loss functions

Non-spherical loss functions

- As described previously, loss functions that are non-spherical can make hill climbing via gradient descent more difficult:

Curvature

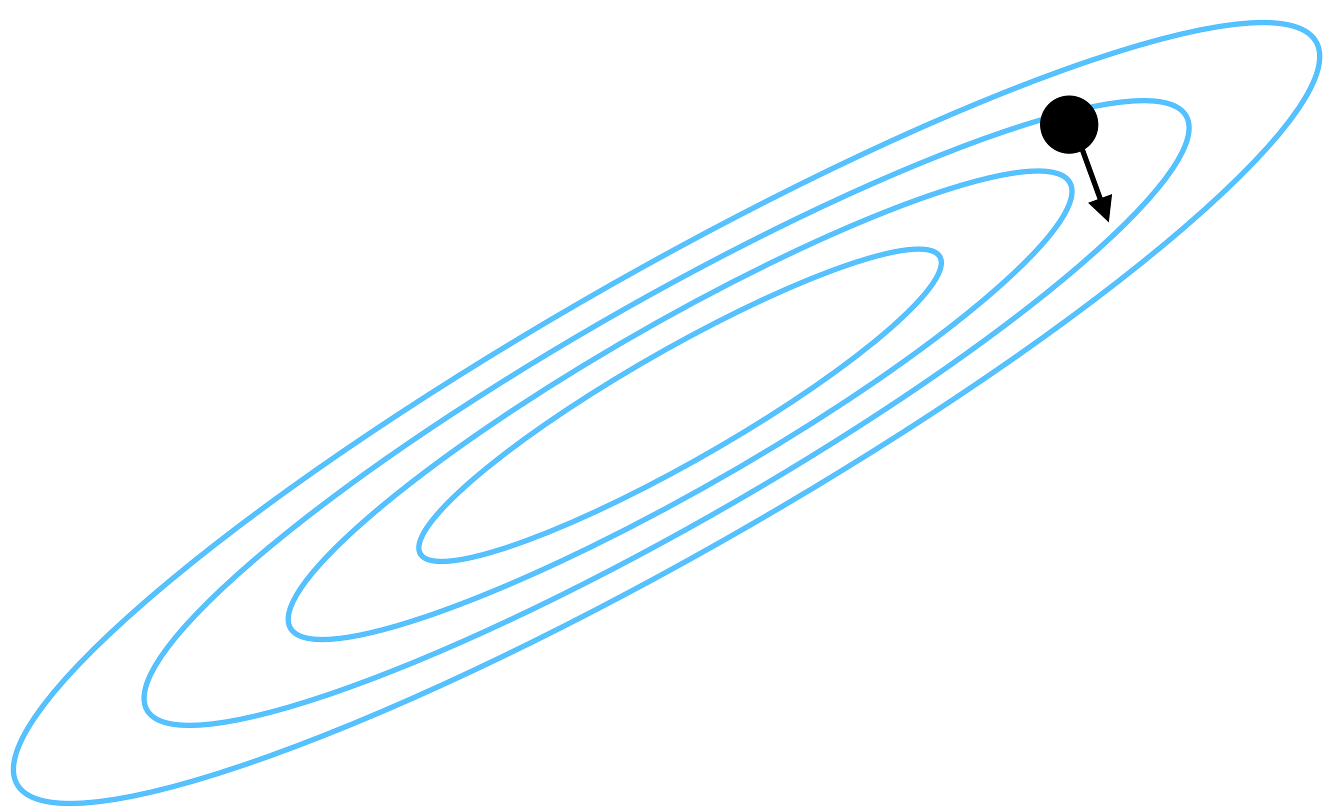
- The problem is that gradient descent only considers slope (1st-order effect), i.e., how f changes with **w**.



Curvature

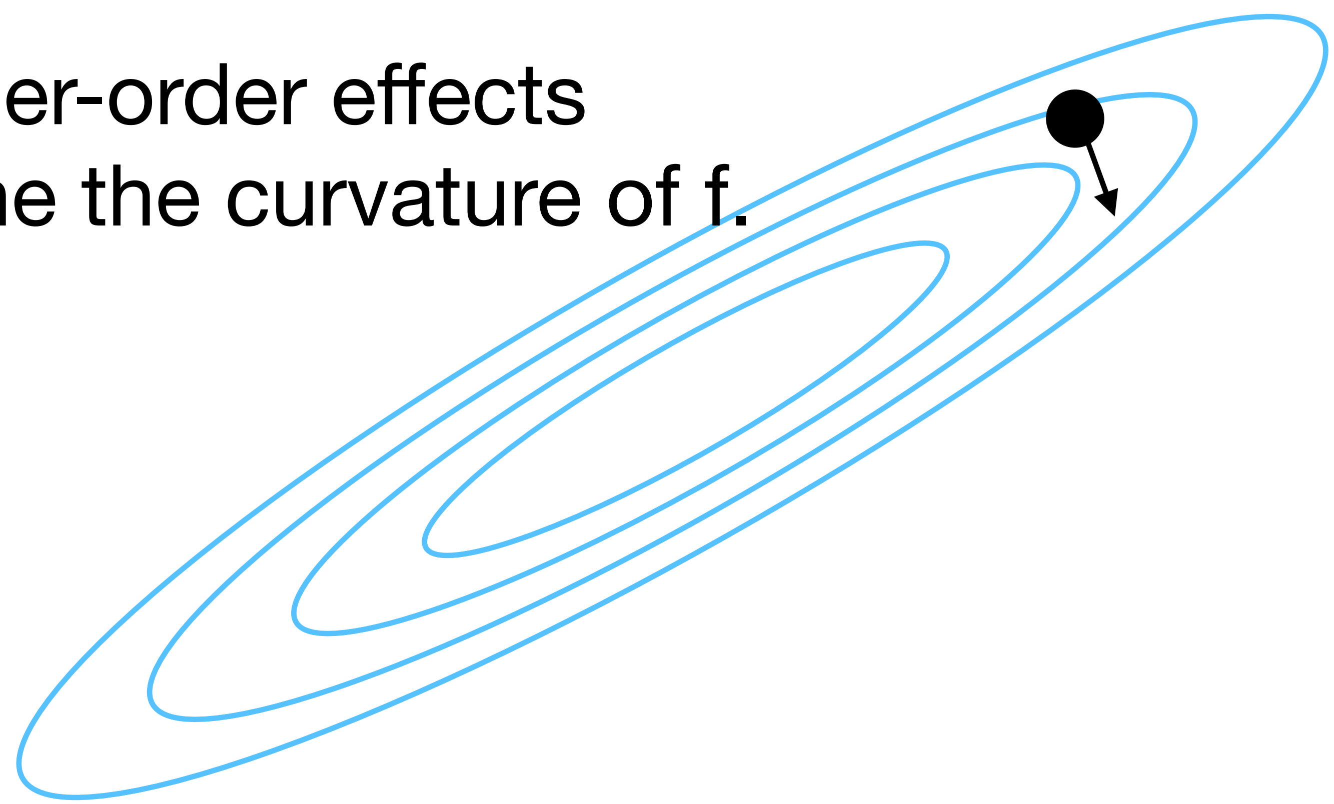
- The problem is that gradient descent only considers slope (1st-order effect), i.e., how f changes with **w**.

- The gradient does not consider how the slope *itself* changes with **w** (2nd-order effect).



Curvature

- The problem is that gradient descent only considers slope (1st-order effect), i.e., how f changes with **w**.
- The gradient does not consider how the slope *itself* changes with **w** (2nd-order effect).
- The higher-order effects determine the curvature of f.



Curvature

- For linear regression with cost f_{MSE},

$$f_{\text{MSE}}(\mathbf{w}) = \frac{1}{2n}(\mathbf{X}^\top\mathbf{w} - \mathbf{y})^\top(\mathbf{X}^\top\mathbf{w} - \mathbf{y})$$

  the Hessian is:

$$\mathbf{H}[f](\mathbf{w}) = \frac{1}{n}\mathbf{X}\mathbf{X}^\top$$

- Hence, **H** is constant and is proportional to the (uncentered) **auto-covariance matrix** of **X**.

$$\mathbb{E}[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^\top]$$

Curvature

- For linear regression with cost f_{MSE},

$$f_{\text{MSE}}(\mathbf{w}) = \frac{1}{2n}(\mathbf{X}^\top\mathbf{w} - \mathbf{y})^\top(\mathbf{X}^\top\mathbf{w} - \mathbf{y})$$

  the Hessian is:

$$\mathbf{H}[f](\mathbf{w}) = \frac{1}{n}\mathbf{X}\mathbf{X}^\top$$

- In other words, the curvature depends solely on the matrix of training data **X**.

$$\mathbb{E}[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^\top]$$

Curvature

- To accelerate optimization of the weights, we can either:
  - Alter the cost function by transforming the input data.
  - Change our optimization method to account for the curvature.

Feature transformations

Whitening transformations

- Gradient descent works best when the level sets of the cost function are spherical.

- We can “spherize” the input features using a **whitening transformation**, which makes the auto-covariance matrix equal the identity matrix **I**.

- We compute this transformation on the training data, and then apply it to both training and testing data.

Whitening transformations

- We can find a whitening transform **T** as follows:
    - Let the auto-covariance* of our training data be $\mathbf{X}\mathbf{X}^T$.

*** (uncentered).**

Whitening transformations

- We can find a whitening transform **T** as follows:
  - Let the auto-covariance* of our training data be $\mathbf{X}\mathbf{X}^{\top}$.
  - We can write its eigendecomposition as:

$$\mathbf{X}\mathbf{X}^{\top}\Phi = \Phi\Lambda$$

where Φ is the matrix of eigenvectors and Λ is the corresponding diagonal matrix of eigenvalues.

*** (uncentered).**

Whitening transformations

- We can find a whitening transform **T** as follows:
  - Let the auto-covariance* of our training data be $\mathbf{X}\mathbf{X}^{\top}$.
  - We can write its eigendecomposition as:

    $$\mathbf{X}\mathbf{X}^{\top}\Phi = \Phi\Lambda$$

    where Φ is the matrix of eigenvectors and Λ is the corresponding diagonal matrix of eigenvalues.
  - For real-valued features, $\mathbf{X}\mathbf{X}^{\top}$ is real and symmetric; hence, Φ is orthonormal. Also, Λ is non-negative.

*** (uncentered).**

Whitening transformations

- We can find a whitening transform **T** as follows:
    - Therefore, we can multiply both sides by Φ^T:

$$\mathbf{X}\mathbf{X}^\top\Phi = \Phi\Lambda$$

$$\Phi^\top\mathbf{X}\mathbf{X}^\top\Phi = \Phi^\top\Phi\Lambda = \Lambda$$

Whitening transformations

- We can find a whitening transform **T** as follows:

    - Therefore, we can multiply both sides by Φ^T:

$$\mathbf{X}\mathbf{X}^\top \Phi = \Phi \Lambda$$

$$\Phi^\top \mathbf{X}\mathbf{X}^\top \Phi = \Phi^\top \Phi \Lambda = \Lambda$$

    - Since Λ is diagonal and non-negative, we can easily compute $\mathbf{\Lambda}^{-\frac{1}{2}}$.

Whitening transformations

- We can find a whitening transform **T** as follows:
  - Therefore, we can multiply both sides by Φ^T:

$$\mathbf{X}\mathbf{X}^\top\Phi = \Phi\Lambda$$

$$\Phi^\top\mathbf{X}\mathbf{X}^\top\Phi = \Phi^\top\Phi\Lambda = \Lambda$$

  - Since Λ is diagonal and non-negative, we can easily compute $\boldsymbol{\Lambda}^{-\frac{1}{2}}$.
  - We then multiply both sides (2x) to obtain **I** on the RHS.

$$\Lambda^{-\frac{1}{2}^\top}\Phi^\top\mathbf{X}\mathbf{X}^\top\Phi\Lambda^{-\frac{1}{2}} = \Lambda^{-\frac{1}{2}^\top}\Lambda\Lambda^{-\frac{1}{2}}$$

Whitening transformations

- We can find a whitening transform **T** as follows:
  - Therefore, we can multiply both sides by Φ^{T}:

$$\mathbf{X}\mathbf{X}^\top \Phi = \Phi \Lambda$$

$$\Phi^\top \mathbf{X}\mathbf{X}^\top \Phi = \Phi^\top \Phi \Lambda = \Lambda$$

  - Since Λ is diagonal and non-negative, we can easily compute $\mathbf{\Lambda}^{-\frac{1}{2}}$.
  - We then multiply both sides (2x) to obtain **I** on the RHS.

$$\Lambda^{-\frac{1}{2}^\top} \Phi^\top \mathbf{X}\mathbf{X}^\top \Phi \Lambda^{-\frac{1}{2}} = \Lambda^{-\frac{1}{2}^\top} \Lambda \Lambda^{-\frac{1}{2}}$$

$$\left(\Lambda^{-\frac{1}{2}^\top} \Phi^\top \mathbf{X}\right)^\top \left(\Lambda^{-\frac{1}{2}^\top} \Phi^\top \mathbf{X}\right) = \mathbf{I}$$

Whitening transformations

- We have thus derived a transform $\mathbf{T} = \Lambda^{-\frac{1}{2}}{}^{\top} \Phi^{\top}$ such that the (uncentered) auto-covariance of the transformed data $\tilde{\mathbf{X}} = \mathbf{T}\mathbf{X}$ is the identity matrix **I**.

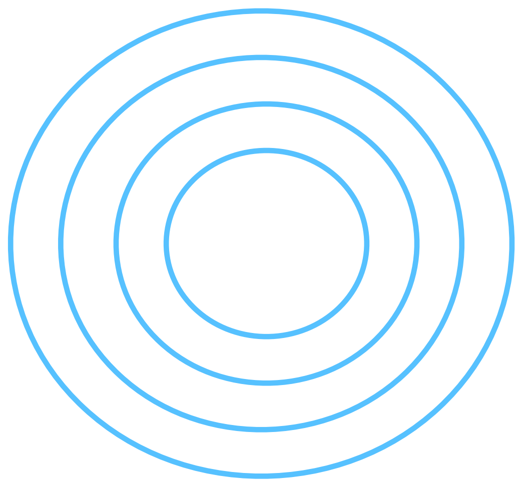
Whitening transformations

- We have thus derived a transform $\mathbf{T} = \Lambda^{-\frac{1}{2}}{}^{\top} \Phi^{\top}$ such that the (uncentered) auto-covariance of the transformed data $\tilde{\mathbf{X}} = \mathbf{T}\mathbf{X}$ is the identity matrix **I**.

- **T** transforms the cost from $f_{\mathrm{MSE}}(\mathbf{w}; \mathbf{X})$

Whitening transformations

- We have thus derived a transform $\mathbf{T} = \Lambda^{-\frac{1}{2}}{}^{\top} \Phi^{\top}$ such that the (uncentered) auto-covariance of the transformed data $\tilde{\mathbf{X}} = \mathbf{T}\mathbf{X}$ is the identity matrix **I**.
- **T** transforms the cost from $f_{\text{MSE}}(\mathbf{w}; \mathbf{X})$ to $f_{\text{MSE}}(\mathbf{w}; \tilde{\mathbf{X}})$:



Whitening transformations

- Whitening transformations are a technique from “classical” ML rather than DL.
    - Time cost is $O(m^3)$, which for high-dimensional feature spaces is too large.
- However, whitening has inspired modern DL techniques such as **batch normalization** (Szegedy & Ioffe, 2015) (more to come later).

Whitening transformations

- Demos (gradient_descent, zscore).