CS 453X: Class 9

Jacob Whitehill

Multi-class versus binary classification

Multi-class versus binary classification

- For the MNIST classification problem, suppose we only really care about distinguishing {0} from {1, 2, …, 9}.
  - This is essentially a binary classification problem (where {1, 2, …, 9} become a single "class").
  - But we could still train a softmax regressor on 10 classes.
- Which approach works better?

Multi-class versus binary classification

- The answer to this question is ultimately **empirical** — there are no theoretical guarantees.

- Intuitive arguments:

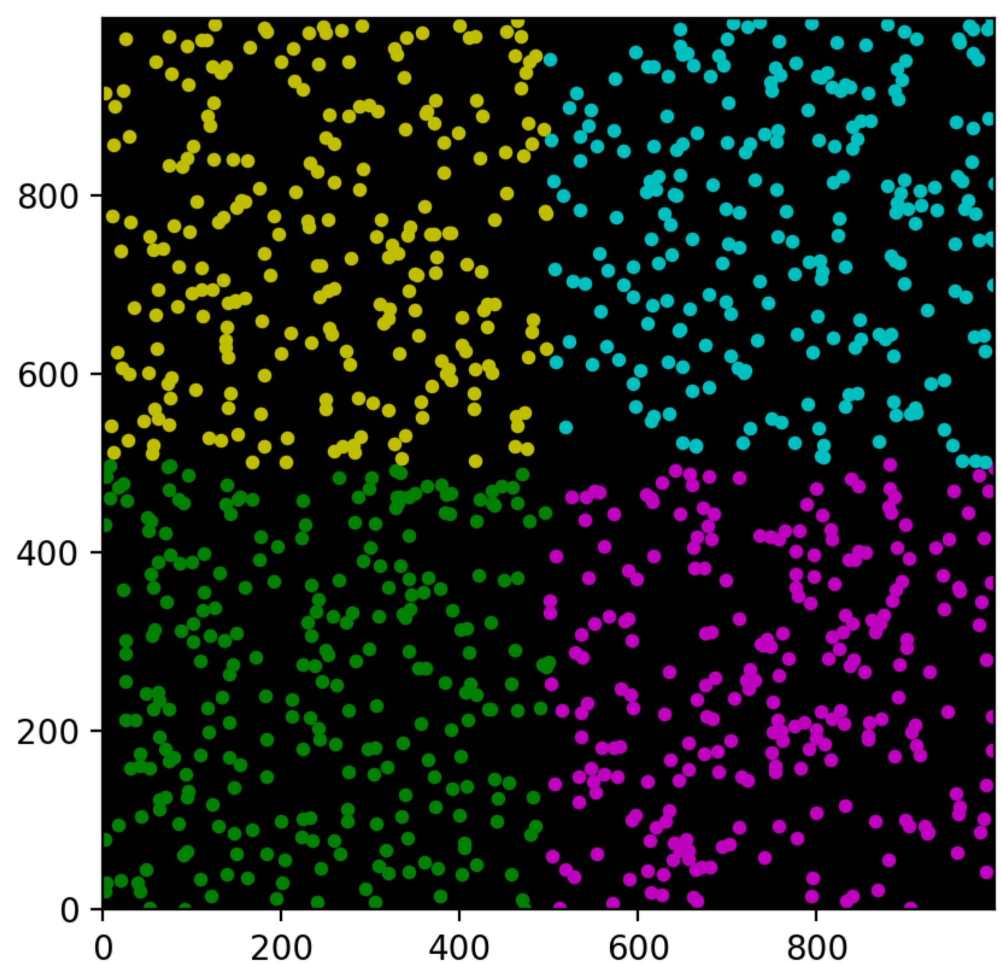
  - Binary classification is simpler, exactly matches the problem objective, and requires fewer parameters.

  - Multi-class classification has more parameters and can capture more structure in the data.

Example on 4 classes

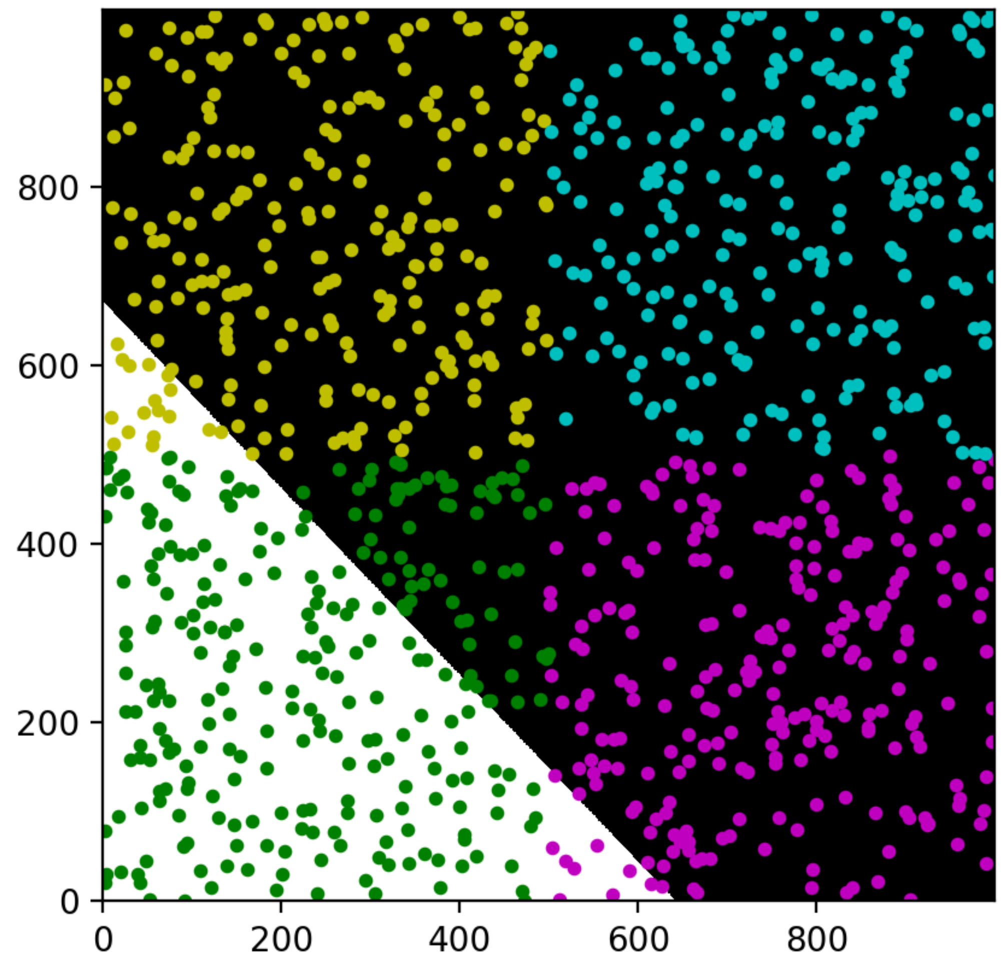
- Here's a simulation of how multi-class classification *can* work better.

- Data points are assigned to classes based on the quadrant in the graph.



Example on 4 classes

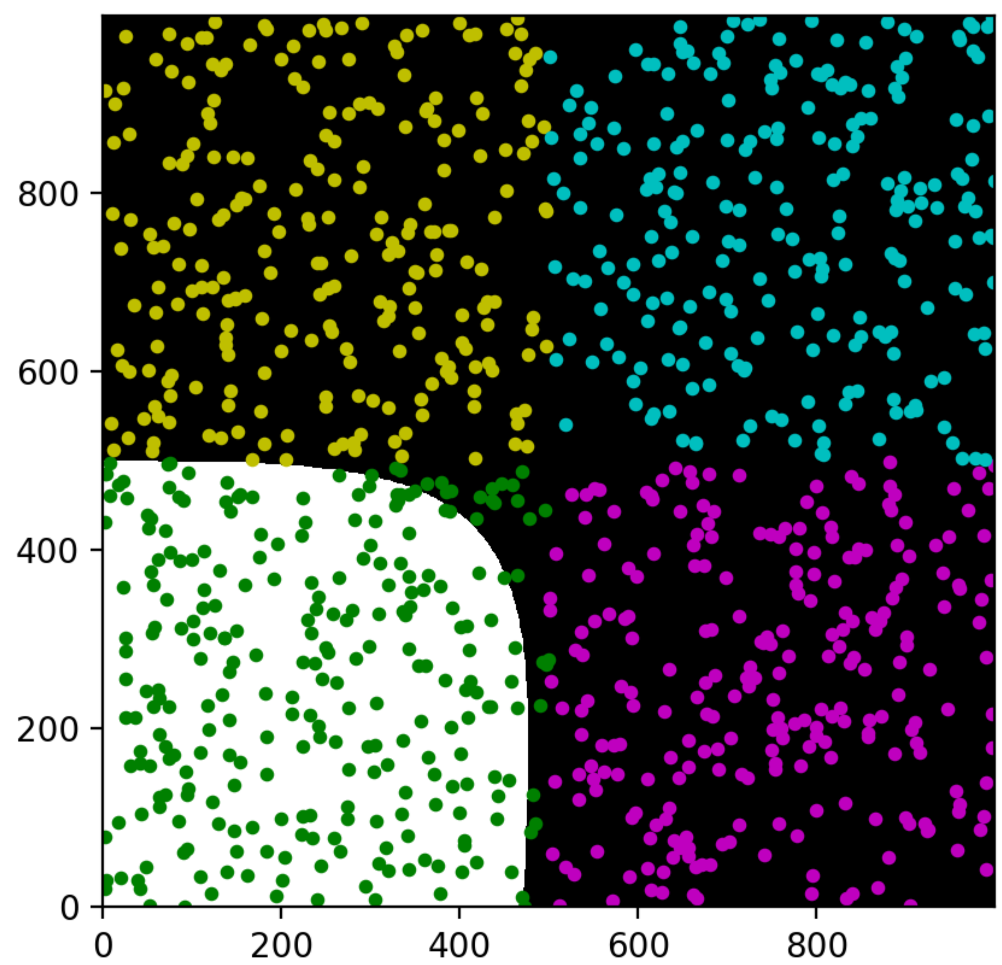
- Logistic regression (1 v {2,3,4}):
  - White background: set of pixels such that $\hat{y}_1 > 0.5$



PC=0.909

Example on 4 classes

- Softmax regression (1 v 2 v 3 v 4):
  - White background: set of pixels such that $\hat{y}_1 > 0.5$



PC=0.991

Stochastic gradient descent

Gradient descent

- With gradient descent, we only update the weights after scanning the *entire* training set.
  - This is slow.
- If the training set contains 60K examples (like in MNIST), then the gradient is an *average* over 60K images.
  - How much would the gradient really change if we just used, say, 30K images? 15K images? 128 images?

$$\nabla_{\mathbf{W}} f_{\mathrm{CE}}(\mathbf{Y}, \hat{\mathbf{Y}}; \mathbf{W}) = \frac{1}{n}\mathbf{X}(\hat{\mathbf{Y}} - \mathbf{Y})$$

Average over entire training set.

Stochastic gradient descent

- This is the idea behind **stochastic gradient descent** (SGD):
  - Randomly sample a small ($\ll n$) **mini-batch** (or sometimes just **batch**) of training examples.
  - Estimate the gradient on just the mini-batch.
  - Update weights based on *mini-batch* gradient estimate.
  - Repeat.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
    - An **epoch** is a single pass through the entire training set.

- Procedure:
    1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
    2. Randomize the order of the examples in the training set.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.
- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.
  3. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.
- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.
  3. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):
     A. Select a mini-batch $\mathcal{J}$ containing the next $\tilde{n}$ examples.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.
- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.
  3. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):
     A. Select a mini-batch $\mathcal{J}$ containing the next $\tilde{n}$ examples.
     B. Compute the gradient on this mini-batch: $\frac{1}{\tilde{n}} \sum_{i \in \mathcal{J}} \nabla_{\mathbf{W}} f(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}; \mathbf{W})$

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.

- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.
  3. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):
     A. Select a mini-batch $\mathcal{J}$ containing the next $\tilde{n}$ examples.
     B. Compute the gradient on this mini-batch: $\frac{1}{\tilde{n}} \sum_{i \in \mathcal{J}} \nabla_{\mathbf{W}} f(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}; \mathbf{W})$
     C. Update the weights based on the current mini-batch gradient.

Stochastic gradient descent

- In practice, SGD is usually conducted over multiple epochs.
  - An **epoch** is a single pass through the entire training set.

- Procedure:
  1. Let $\tilde{n} \ll n$ equal the size of the mini-batch.
  2. Randomize the order of the examples in the training set.
  3. For *i* = 0 to $(\lceil n/\tilde{n} \rceil - 1)$ (one epoch):
     A. Select a mini-batch $\mathcal{J}$ containing the next $\tilde{n}$ examples.
     B. Compute the gradient on this mini-batch: $\frac{1}{\tilde{n}} \sum_{i \in \mathcal{J}} \nabla_{\mathbf{W}} f(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}; \mathbf{W})$
     C. Update the weights based on the current mini-batch gradient.
  4. Repeat (3) until the desired number of epochs is reached.

SGD versus GD: example

- Suppose our training set contains n=8 examples.
- Here is how *regular* gradient descent would proceed:
  - **Initialize weights $\mathbf{w}^{(0)}$ to random values.**

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples.
- Here is how *regular* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - For each round:
    - **Compute gradient on all *n* examples.**

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples.
- Here is how *regular* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - For each round:
    - Compute gradient on all n examples.
    - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \nabla_{\mathbf{w}} f$

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples.

- Here is how *regular* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - For each round:
    - **Compute gradient on all *n* examples.**
    - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \nabla_{\mathbf{w}} f$

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples.
- Here is how *regular* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - For each round:
    - Compute gradient on all n examples.
    - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \nabla_{\mathbf{w}} f$

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - **Initialize weights $\mathbf{w}^{(0)}$ to random values.**

Training examples
1
2
3
4
5
6
7
8

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - **Randomize the order of the training data.**

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=1
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): **e=1**
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=1
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=1
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=1
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): **e=1**
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=1
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=1
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - **Update weights**: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): e=2
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - **Compute gradient on next $\tilde{n}$ examples.**
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\nabla}_{\mathbf{w}} f$

Training examples
4
1
3
5
7
6
8
2

SGD versus GD: example

- Suppose our training set contains n=8 examples with $\tilde{n}$ = 2.
- Here is how *stochastic* gradient descent would proceed:
  - Initialize weights $\mathbf{w}^{(0)}$ to random values.
  - Randomize the order of the training data.
  - For each epoch (e=1, …, E): **e=2**
    - For each round (r=1, …, $\lceil n/\tilde{n} \rceil$):
      - Compute gradient on next $\tilde{n}$ examples.
      - Update weights: $\mathbf{w}^{(t+1)} \longleftarrow \mathbf{w}^{(t)} - \epsilon \tilde{\boldsymbol{\nabla}}_{\mathbf{w}} f$

…

Training examples
4
1
3
5
7
6
8
2

Stochastic gradient descent

- Despite “noise” (statistical inaccuracy) in the mini-batch gradient estimates, we will still converge to local minimum.

- Training can be much faster than regular gradient descent because we adjust the weights *many times* per epoch.

SGD: learning rates

- With SGD, our learning rate ϵ needs to be **annealed** (reduced slowly over time) to guarantee convergence.
  - Otherwise we might just oscillate forever in weight space.
- Necessary conditions:

$$\lim_{T\to\infty} \sum_{t=1}^{T} |\epsilon_t|^2 < \infty$$

Not too big: sum of squared learning rates converges.

SGD: learning rates

- With SGD, our learning rate ϵ needs to be **annealed** (reduced slowly over time) to guarantee convergence.
  - Otherwise we might just oscillate forever in weight space.
- Necessary conditions:

$$\lim_{T\to\infty} \sum_{t=1}^{T} |\epsilon_t|^2 < \infty \qquad \lim_{T\to\infty} \sum_{t=1}^{T} |\epsilon_t| = \infty$$

Not too small: sum of absolute learning rates grows to infinity.

SGD: learning rates

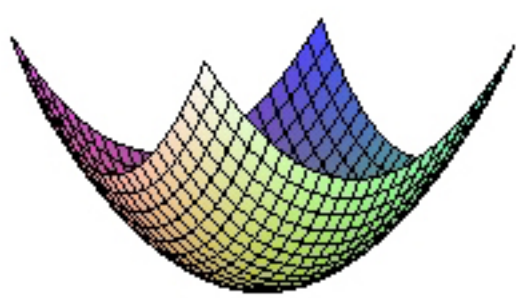
- One common learning rate "schedule" is to multiply ϵ by $c \in (0, 1)$ every *k* rounds.
    - This is called **exponential decay**.
- Another possibility (which avoids the issue) is to set the number of epochs *T* to a finite number.
    - SGD may not fully converge, but the machine might still perform well.
- There are many other strategies.

Convex ML models

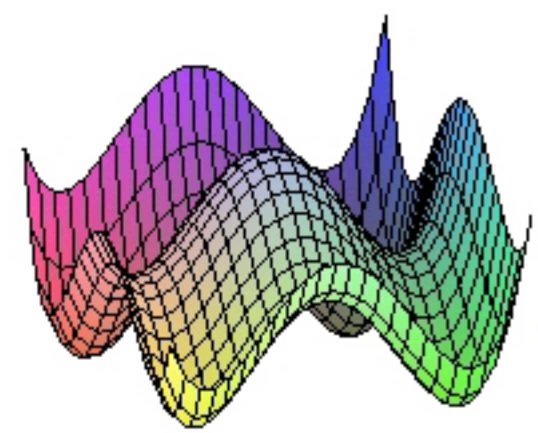
Convex ML models

- The two main ML models we have examined — linear regression and softmax regression — have loss functions that are **convex**.

- With a convex function *f*, every local minimum is also a global minimum.



convex



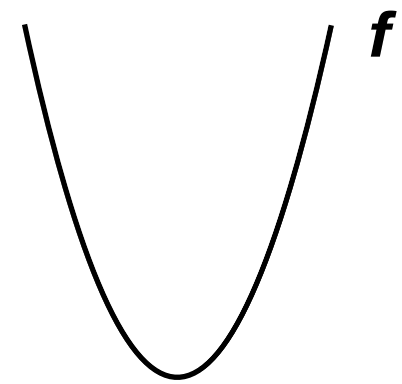
non-convex

- Convex functions are ideal for conducting gradient descent.

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

Convexity in 1-d

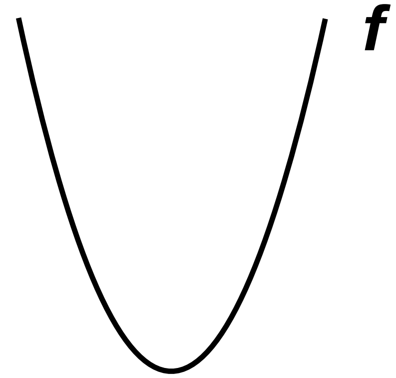
- How can we tell if a 1-d function *f* is convex?



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

Convexity in 1-d

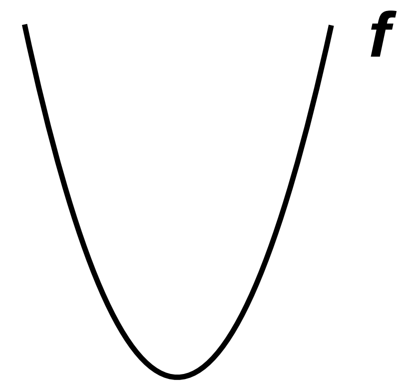
- How can we tell if a 1-d function *f* is convex?



- 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?

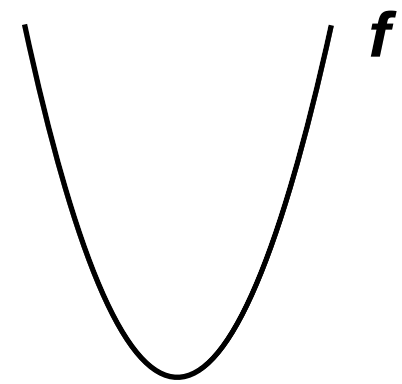


- What property of *f* ensures there is only one local minimum?

    - 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 second derivative matrix, known as the **Hessian** of *f*.

$$\mathbf{H} = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n} \\[2.2ex] \dfrac{\partial^2 f}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2\,\partial x_n} \\[2.2ex] \vdots & \vdots & \ddots & \vdots \\[2.2ex] \dfrac{\partial^2 f}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix}.$$

- For $f : \mathbb{R}^m \to \mathbb{R}$, *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 $\mathbf{A}$ is **positive semi-definite (PSD)** if (equivalent conditions):
  - All its eigenvalues are ≥ 0.
    - If $\mathbf{A}$ happens to be diagonal, then its eigenvalues are the diagonal elements.
  - For every vector $\mathbf{v}$: $\mathbf{v}^T\mathbf{A}\mathbf{v} \geq 0$
    - Therefore: If there exists *any* vector $\mathbf{v}$ such that $\mathbf{v}^T\mathbf{A}\mathbf{v} < 0$, then $\mathbf{A}$ 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 $f(x, y) = 3x^2 + 2y^2 - 2$: