Neural Networks – Part 2

- Training as an Optimization Problem.
- Gradient Descent

CSE 4311 – Neural Networks and Deep Learning

Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

Training a Neural Network

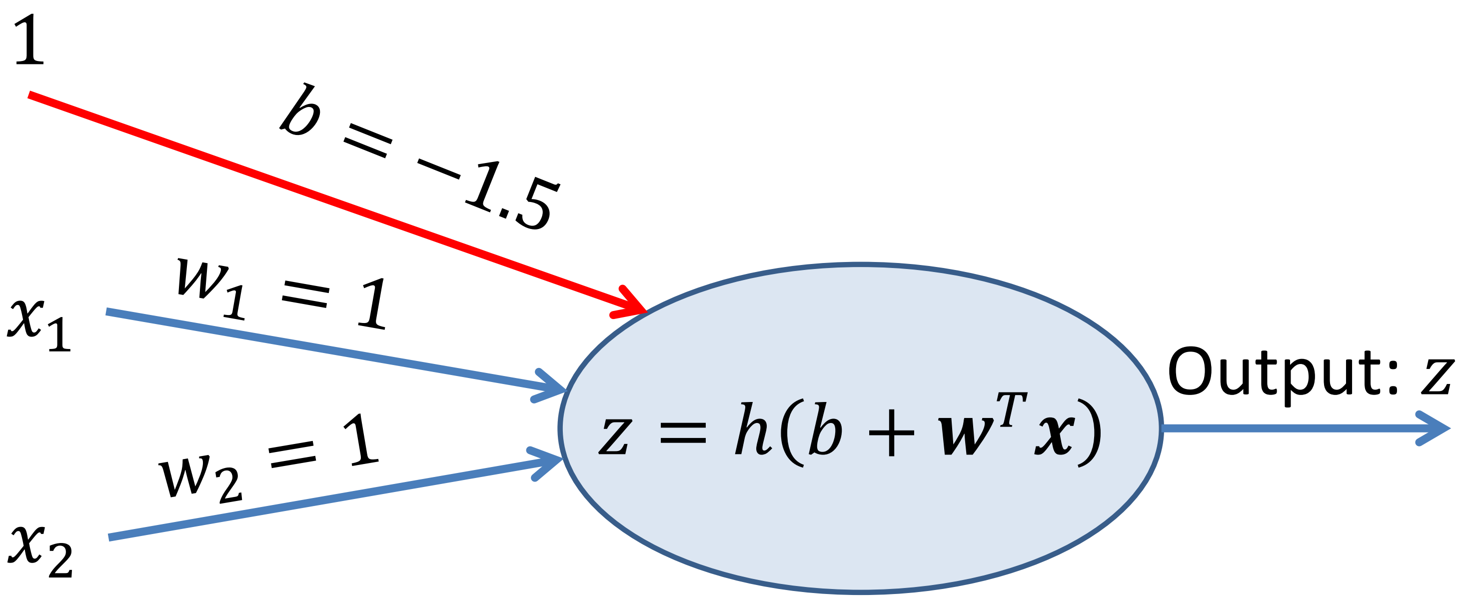
- To train a neural network we need:
  - Values for hyperparameters specifying the network topology (number of layers, units per layer, connectivity of layers).
  - Other hyperparameters, besides network topology.
    - More details later.
  - A training set.
    - This is a typical supervised learning problem, so each element of a training set is a pair of an example input and a target output.
    - Typically, both the input and the output are multidimensional.
  - An optimization criterion.
    - This is a quantitative performance measure, that tells us how well the network performs on the training data.

Plan

- We start with training the simplest neural network:
  - A single perceptron.
- Training a neural network is an **optimization problem**.
  - We will define what that means.
- In general, optimization problems can be solved in different ways.
- For neural networks in particular, we will solve the optimization problem using **gradient descent**.
  - Again, we will define what that means.
- We will apply gradient descent to train a perceptron.
- Then, we will address more general neural networks.

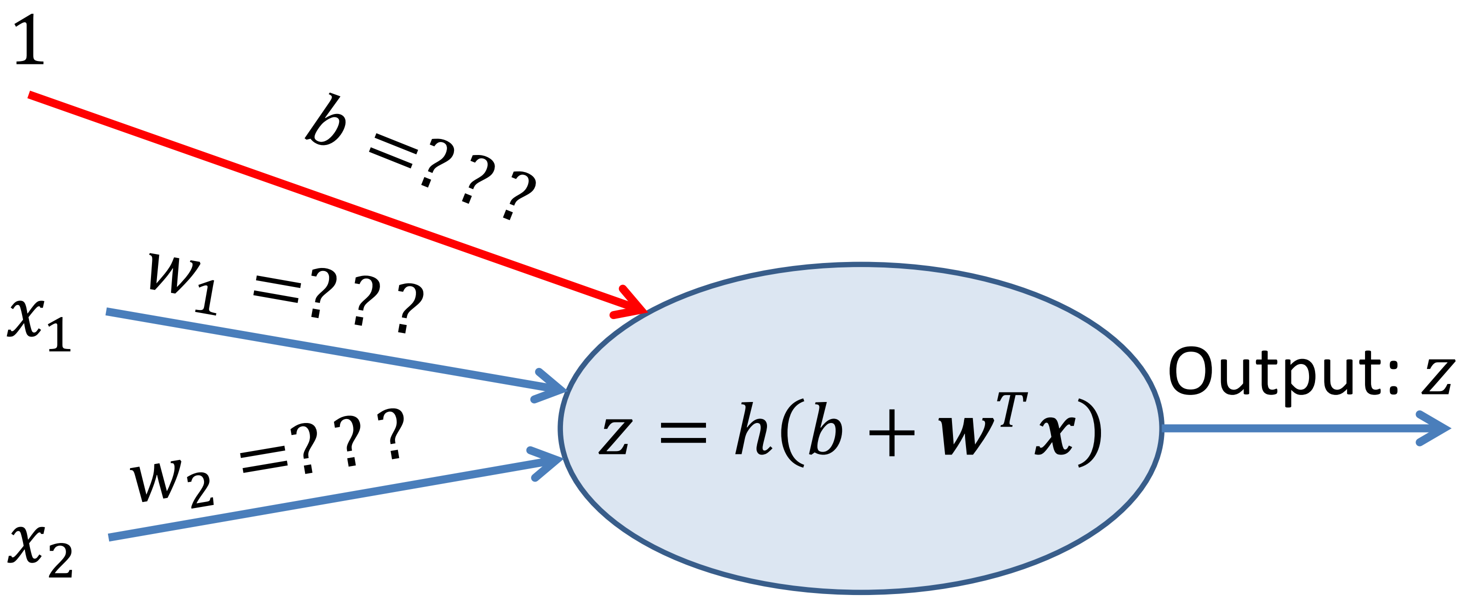
Training the AND Perceptron

- When we discussed the AND perceptron before, we hardcoded the weights.



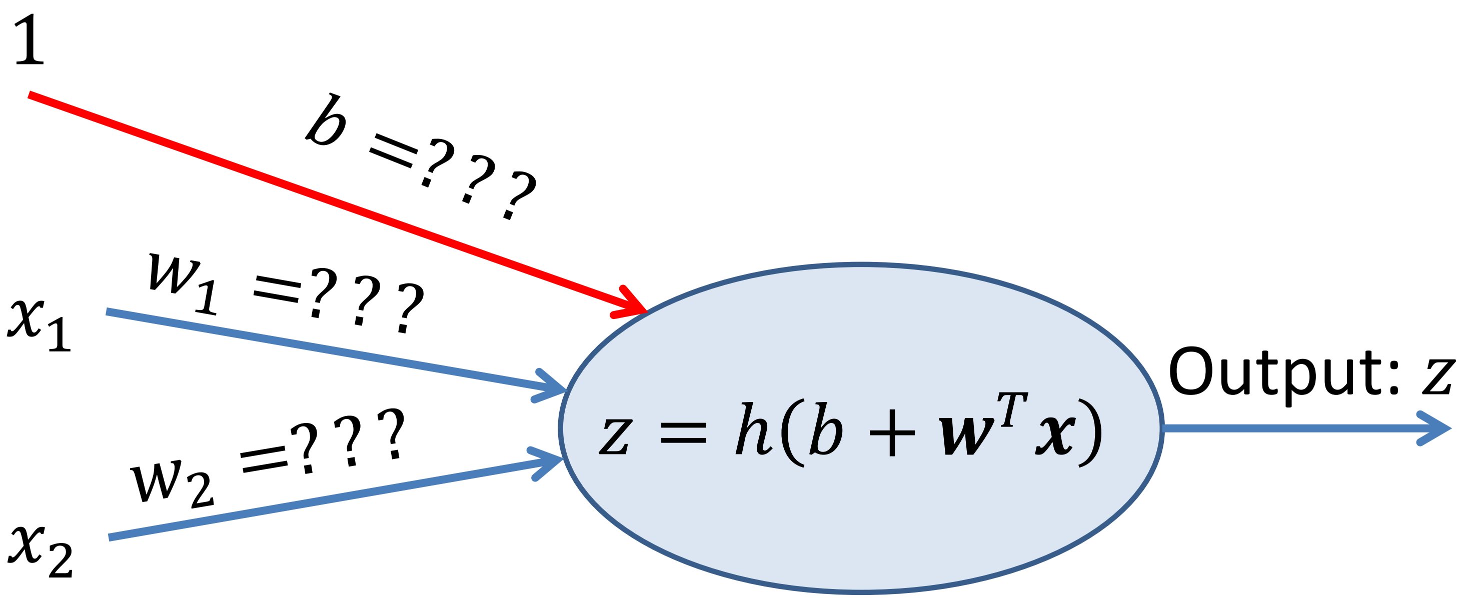
Training the AND Perceptron

- When we discussed the AND perceptron before, we hardcoded the weights.
- Now, as a toy example, we will see how we could train this perceptron, so that we learn the weights using training data.



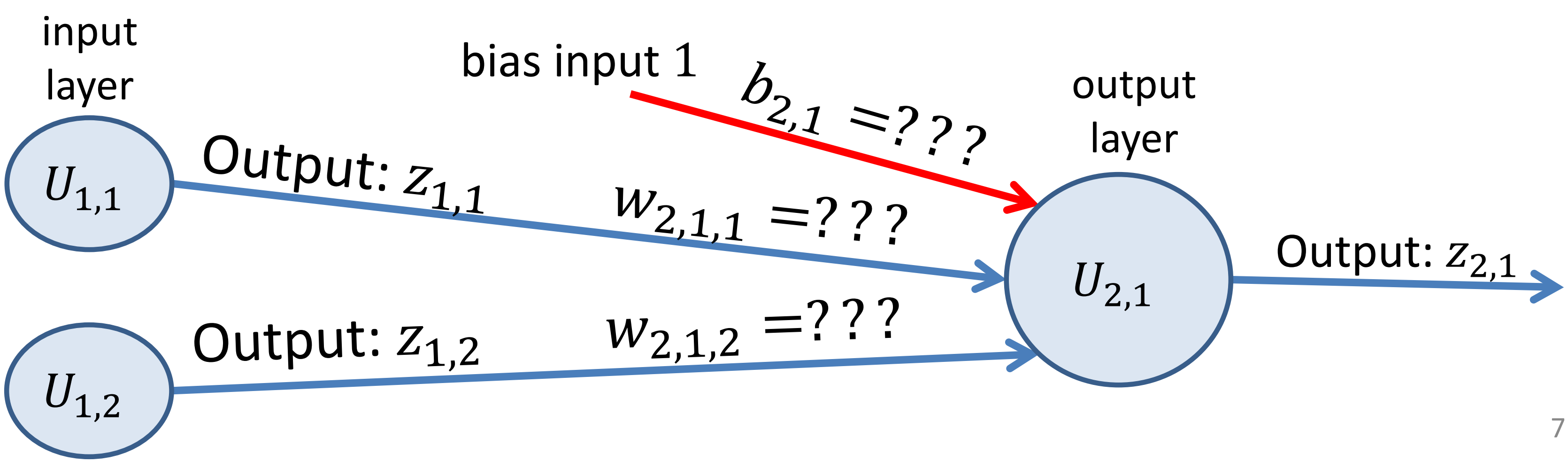
Drawing This as a Neural Network

- If we think of the AND perceptron as a neural network, what topology does it have?
  - How many layers?
  - How many units in each layer?



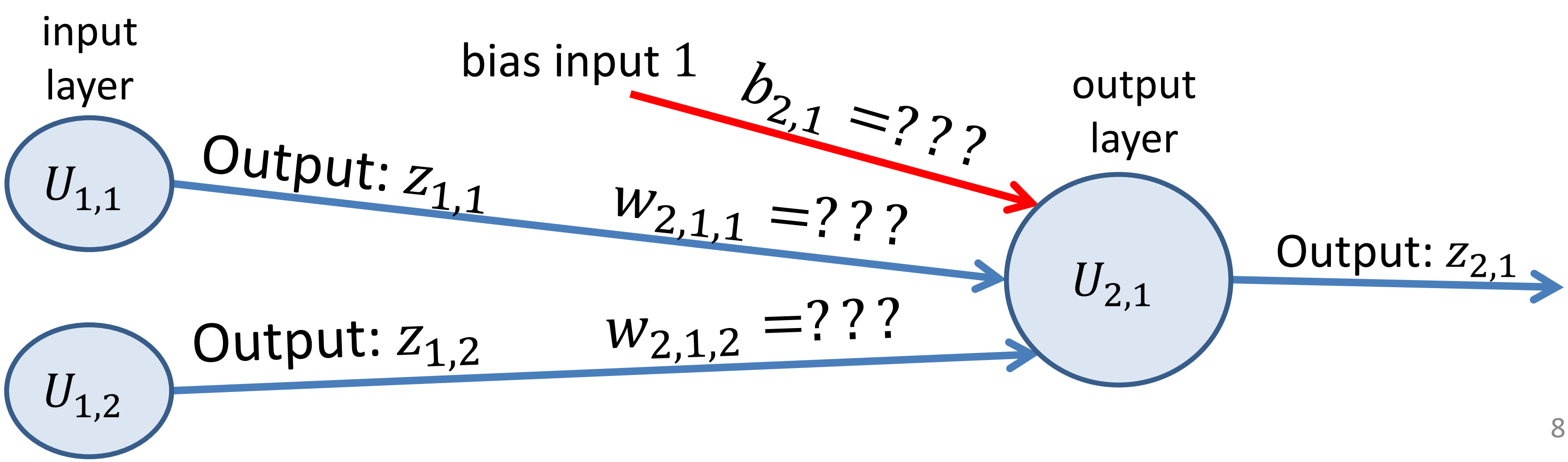
Drawing This as a Neural Network

- If we think of the AND perceptron as a neural network, what topology does it have?
  - How many layers? Two (don't forget the input layer).
  - How many units in each layer?
    - Input layer: 2 units
    - Output layer: 1 unit (the actual perceptron).



Training Set

- This is a toy problem, there are only four possible cases:
  - $\boldsymbol{x}_1 = (0.0,\ 0.0)^T \qquad t_1 = 0$
  - $\boldsymbol{x}_2 = (0.0,\ 1.0)^T \qquad t_2 = 1$
  - $\boldsymbol{x}_3 = (1.0,\ 0.0)^T \qquad t_3 = 1$
  - $\boldsymbol{x}_4 = (1.0,\ 1.0)^T \qquad t_4 = 1$



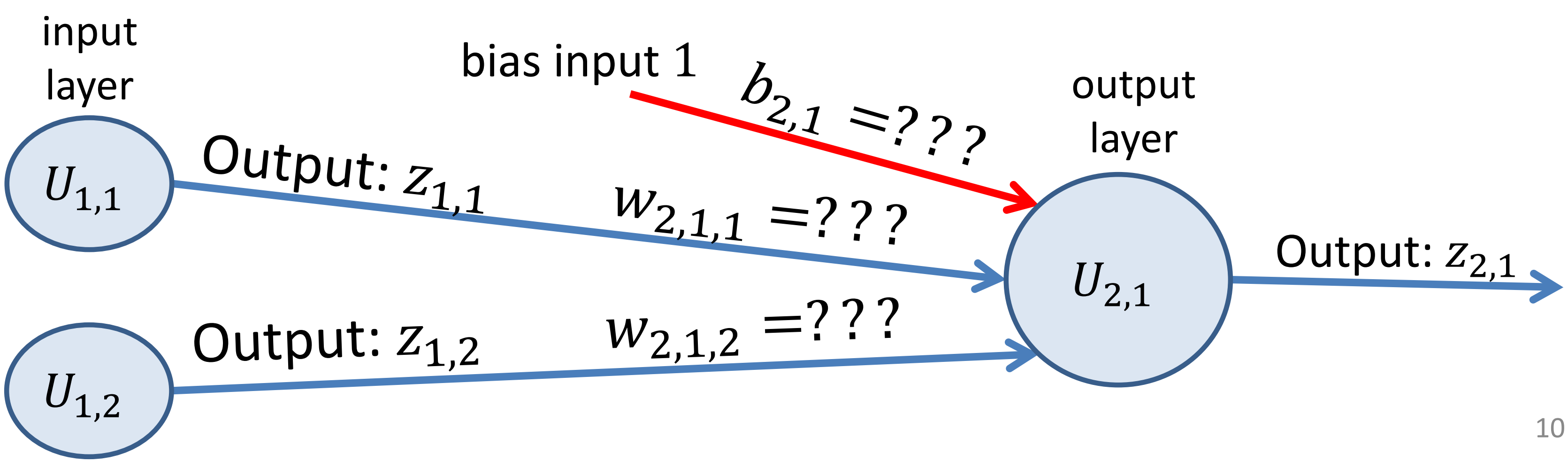
Perceptron Training: Notation for Training Set

- We have a set X of N training inputs.
  - $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$
- Each $\boldsymbol{x}_n$ is a D-dimensional column vector.
  - $\boldsymbol{x}_n = (x_{n,1}, x_{n,2}, \dots, x_{n,D})'$
- We also have a set T of N target outputs.
  - $T = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \dots, \boldsymbol{t}_N\}$
  - $\boldsymbol{t}_n$ is the target output for training example $\boldsymbol{x}_n$.
- **If we are training a single perceptron**, then each $\boldsymbol{t}_n$ is a real number.
- **In the general case**, each $\boldsymbol{t}_n$ is a K-dimensional column vector:
  - $\boldsymbol{t}_n = (t_{n,1}, t_{n,2}, \dots, t_{n,K})'$

Training Goal

- What do we want our training to achieve?

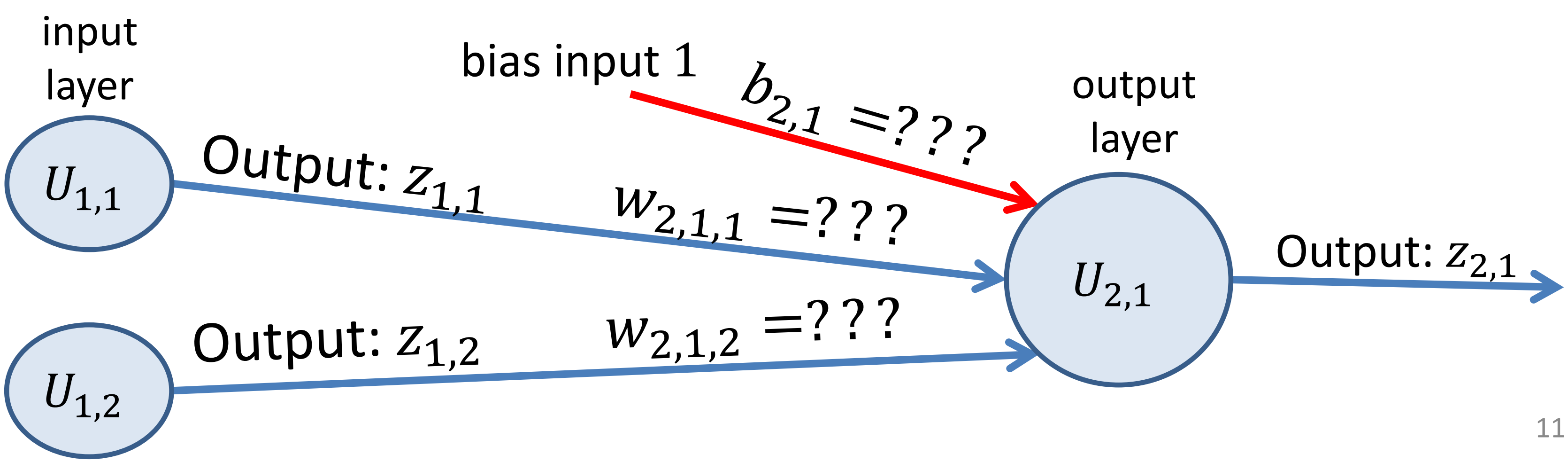
$$\boldsymbol{x}_1 = (0.0,\ 0.0)^T \quad t_1 = 0$$
$$\boldsymbol{x}_2 = (0.0,\ 1.0)^T \quad t_2 = 1$$
$$\boldsymbol{x}_3 = (1.0,\ 0.0)^T \quad t_3 = 1$$
$$\boldsymbol{x}_4 = (1.0,\ 1.0)^T \quad t_4 = 1$$



Training Goal

- What do we want our training to achieve?
  - Intuitively, we want to come up with weights so that each input is mapped to the correct output.
  - We want a general approach, that can be applied to any training set.

$$\boldsymbol{x}_1 = (0.0,\ 0.0)^T \quad t_1 = 0$$
$$\boldsymbol{x}_2 = (0.0,\ 1.0)^T \quad t_2 = 1$$
$$\boldsymbol{x}_3 = (1.0,\ 0.0)^T \quad t_3 = 1$$
$$\boldsymbol{x}_4 = (1.0,\ 1.0)^T \quad t_4 = 1$$



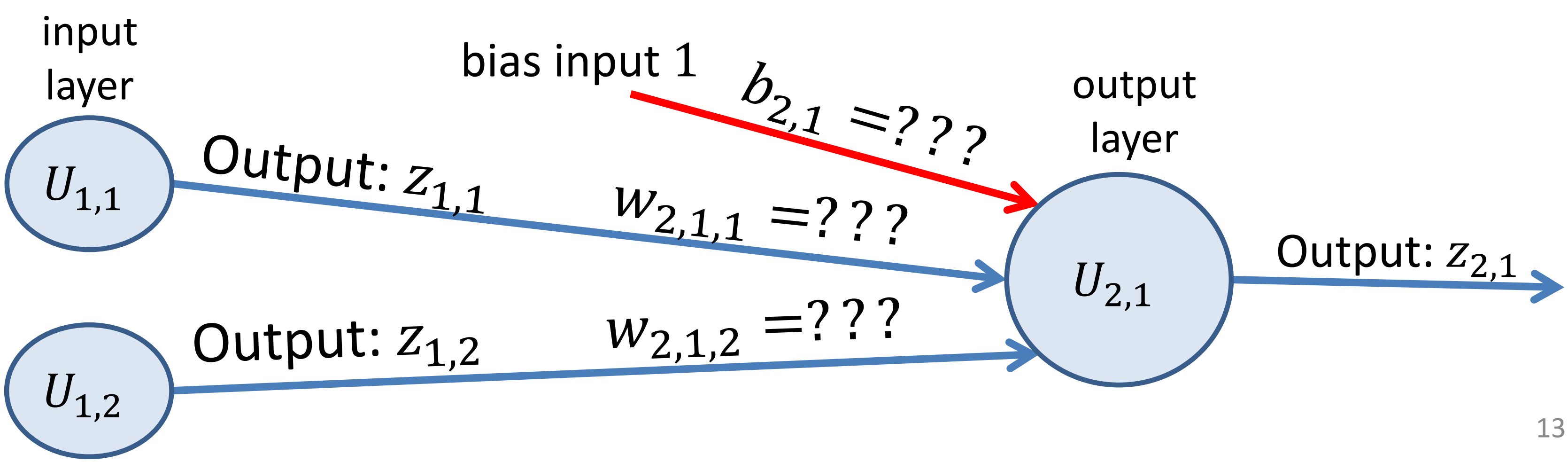
Training as an Optimization Problem

- Training a neural network is an optimization problem.
- In an optimization problem we need to:
  - Define what **parameters** we are optimizing. This is also called the **search space** (the space of possible choices).
  - Define a quantitative **optimization criterion**.
    - For any choice of parameters, this criterion will measure will tell us how good they are.
    - If we have two different choices, this criterion will tell us which choice one is better.
  - Define an **optimization algorithm** for finding a good set of parameters.

Parameters We Optimize

- In a neural network, what parameters are we optimizing?

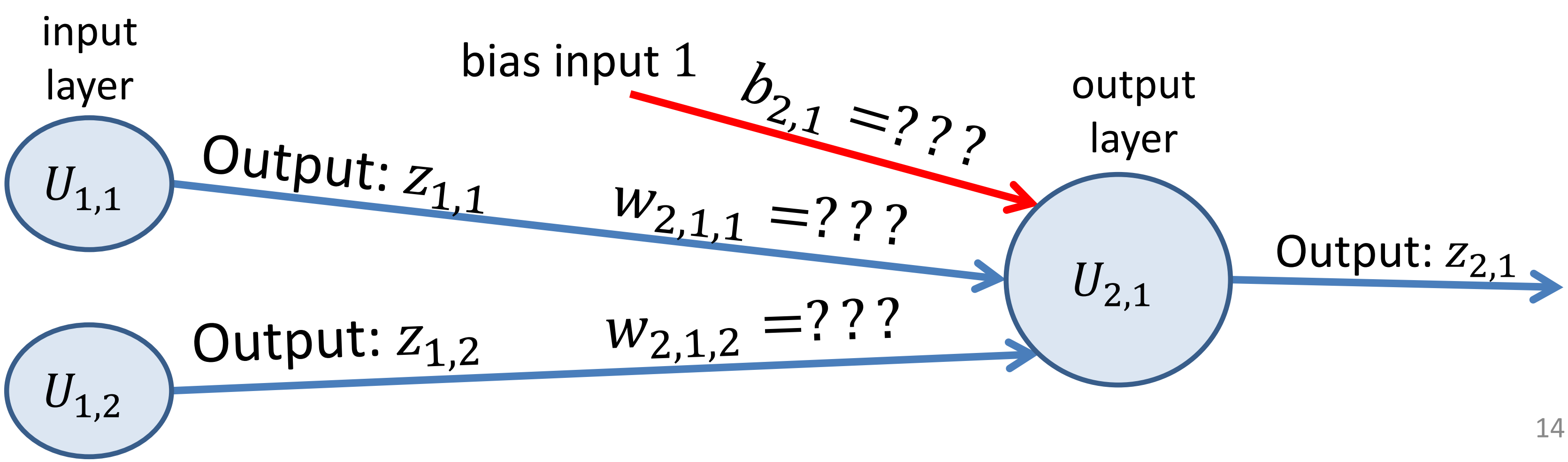
$$\boldsymbol{x}_1 = (0.0,\ 0.0)^T \quad t_1 = 0$$
$$\boldsymbol{x}_2 = (0.0,\ 1.0)^T \quad t_2 = 1$$
$$\boldsymbol{x}_3 = (1.0,\ 0.0)^T \quad t_3 = 1$$
$$\boldsymbol{x}_4 = (1.0,\ 1.0)^T \quad t_4 = 1$$



Parameters We Optimize

- In a neural network, what parameters are we optimizing?
  - Bias weights b and regular weights w.
  - In our toy example, this gives us three values that we have to optimize: $b_{2,1}$, $w_{2,1,1}$, $w_{2,1,2}$.

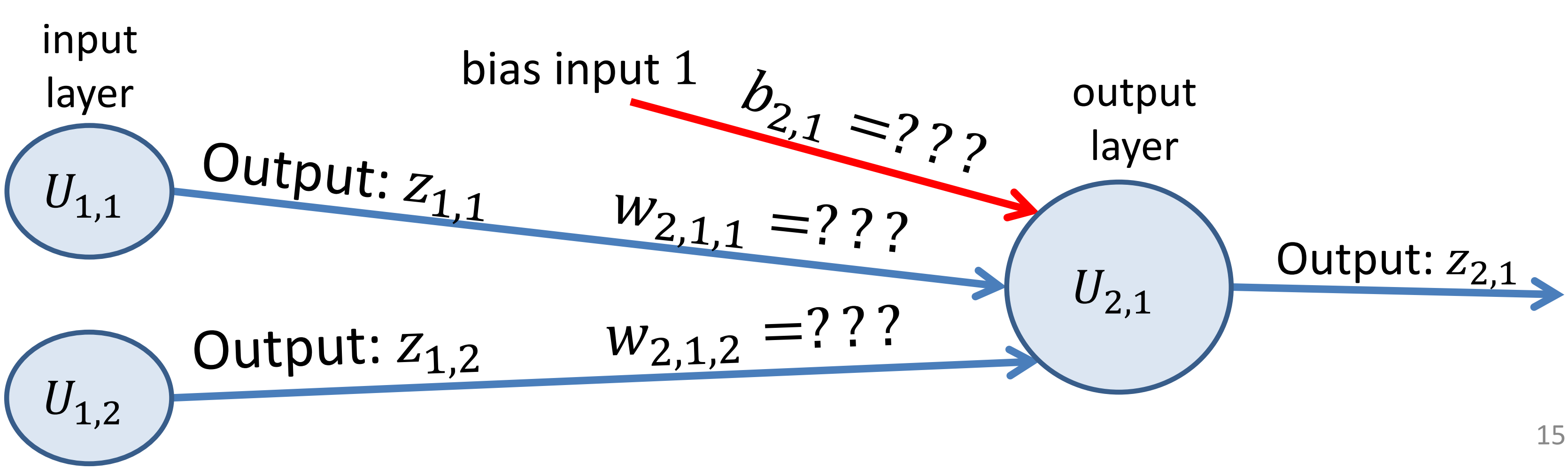
$$
\begin{array}{ll}
\boldsymbol{x}_1 = (0.0,\ 0.0)^T & t_1 = 0 \\
\boldsymbol{x}_2 = (0.0,\ 1.0)^T & t_2 = 1 \\
\boldsymbol{x}_3 = (1.0,\ 0.0)^T & t_3 = 1 \\
\boldsymbol{x}_4 = (1.0,\ 1.0)^T & t_4 = 1
\end{array}
$$



Optimization Criterion

- Suppose that we are considering some values for $b_{2,1}$, $w_{2,1,1}$, $w_{2,1,2}$.
- What quantitative criterion can we use to measure how good (or bad) those values are?

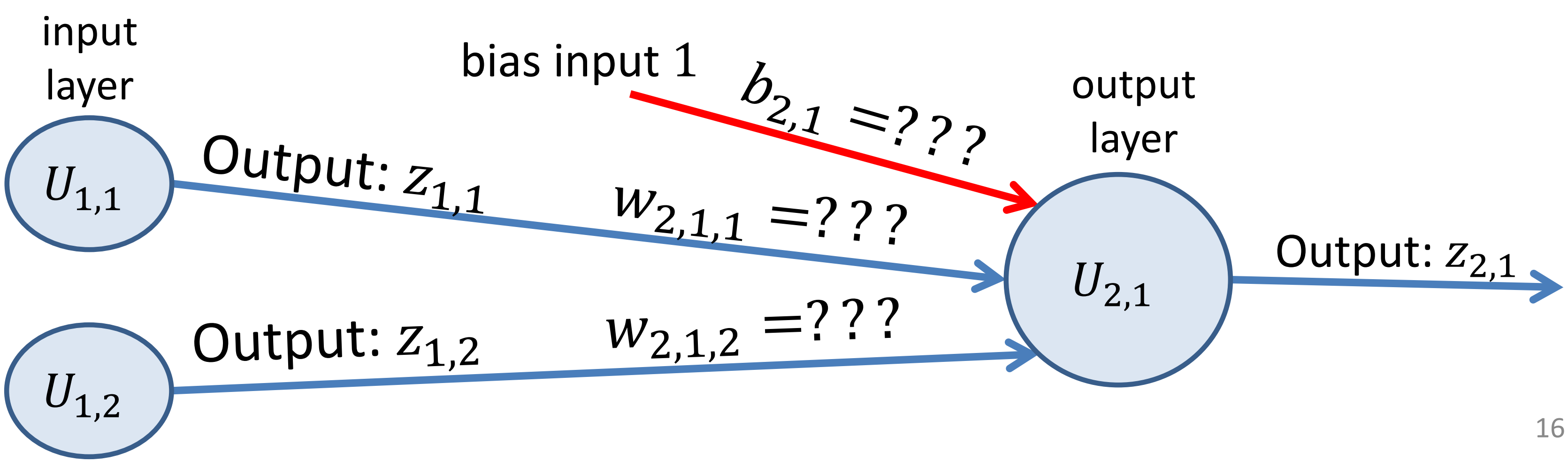
$$\boldsymbol{x}_1 = (0.0,\ 0.0)^T \quad t_1 = 0$$
$$\boldsymbol{x}_2 = (0.0,\ 1.0)^T \quad t_2 = 1$$
$$\boldsymbol{x}_3 = (1.0,\ 0.0)^T \quad t_3 = 1$$
$$\boldsymbol{x}_4 = (1.0,\ 1.0)^T \quad t_4 = 1$$



Optimization Criterion

- Suppose that we are considering some values for $b_{2,1}$, $w_{2,1,1}$, $w_{2,1,2}$.
- What quantitative criterion can we use to measure how good (or bad) those values are?
  - One commonly used measure: sum of squared differences.

$$\boldsymbol{x}_1 = (0.0,\ 0.0)^T \quad t_1 = 0$$
$$\boldsymbol{x}_2 = (0.0,\ 1.0)^T \quad t_2 = 1$$
$$\boldsymbol{x}_3 = (1.0,\ 0.0)^T \quad t_3 = 1$$
$$\boldsymbol{x}_4 = (1.0,\ 1.0)^T \quad t_4 = 1$$



Squared Differences

- A neural network defines a mathematical function $f(\boldsymbol{b}, \boldsymbol{w}, \boldsymbol{x})$:
  - $\boldsymbol{b}$, a list that specifies all the bias weights in the network.
  - $\boldsymbol{w}$, a list that specifies all other weights (non-bias weights) in the network.
  - $\boldsymbol{x}$, the vector that is given as input to the network.
- For our AND example:
  - $\boldsymbol{b}$ is a single number: $b_{2,1}$.
  - $\boldsymbol{w}$ contains two numbers: $w_{2,1,1}$ and $w_{2,1,2}$.
  - $\boldsymbol{x}$ can be any 2-dimensional vector.
- For any training example $\boldsymbol{x}_n$, we define error $E_n(b, \boldsymbol{w})$ as:

$$E_n(\boldsymbol{b}, \boldsymbol{w}) = \frac{1}{2}\left(f(\boldsymbol{b}, \boldsymbol{w}, \boldsymbol{x}_n) - t_n\right)^2$$

- In words, $E_n(b, \boldsymbol{w})$ is the **squared difference** between the output of the neural network and the target output, multiplied (for reasons of convenience, explained later) by $\frac{1}{2}$.

Sum of Squared Differences

- The error E over the entire training set is defined as:

$$E(\boldsymbol{b}, \boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{b}, \boldsymbol{w}) = \sum_{n=1}^{N} \left[\frac{1}{2} \left(f(\boldsymbol{b}, \boldsymbol{w}, \boldsymbol{x}_n) - t_n \right)^2 \right]$$

- This is called the **sum of squared differences (SSD) error**.
  - We simply sum up, over all training examples, the squared difference (squared error) that we get for each example.
- Note that $E(\boldsymbol{b}, \boldsymbol{w})$ is a function of network parameters $\boldsymbol{b}$ and $\boldsymbol{w}$.
  - Different choices of $\boldsymbol{b}$ and $\boldsymbol{w}$ give a different error $E(\boldsymbol{b}, \boldsymbol{w})$.
  - Our training goal is to find values of $\boldsymbol{b}$ and $\boldsymbol{w}$ that **minimize** error $E(\boldsymbol{b}, \boldsymbol{w})$.

Global and Local Optima

- Optimization can be maximization or minimization.
  - We typically want to maximize if our optimization criterion relates to accuracy, fitness, utility…
  - We typically want to minimize if our optimization criterion relates to error, cost, time or space complexity…
- In our case, our optimization criterion is SSD error, so we want to minimize that.
- An <u>**optimum**</u> is a <u>**maximum**</u> when we want to maximize, and a <u>**minimum**</u> when we want to minimize.
- The goal in optimization is to find an optimum.

Global and Local Optima

- What does it mean if we say that a choice of values $\boldsymbol{b_{opt}}$ and $\boldsymbol{w_{opt}}$ **minimizes** the SSD error?

$$E(\boldsymbol{b}, \boldsymbol{w}) = \sum_{n=1}^{N} \left[\frac{1}{2} \left(f(\boldsymbol{b}, \boldsymbol{w}, \boldsymbol{x}_n) - t_n \right)^2 \right]$$

- A term like "minimum", "maximum", "optimum" can be unclear, unless we specify whether it is **global** or **local**.
- To say that $\boldsymbol{b_{opt}}$ and $\boldsymbol{w_{opt}}$ are **globally optimal**, they must satisfy this property:

$$\forall\, (\boldsymbol{b}, \boldsymbol{w}), E\left(\boldsymbol{b_{opt}}, \boldsymbol{w_{opt}}\right) \leq E(\boldsymbol{b}, \boldsymbol{w})$$

- That is, no other choice for $\boldsymbol{b}$ and $\boldsymbol{w}$ can give a lower error.

Global and Local Optima

- For $\boldsymbol{b_{opt}}$ and $\boldsymbol{w_{opt}}$ to be **<u>locally optimal</u>**, they must satisfy a far weaker property:

$$\begin{aligned}&\exists \varepsilon, \text{such that } \forall\, (\boldsymbol{b}, \boldsymbol{w}):\\ &\text{if } \left[\left(\left\|\boldsymbol{b_{opt}} - \boldsymbol{b}\right\| < \varepsilon\right) \text{ and } \left(\left\|\boldsymbol{w_{opt}} - \boldsymbol{w}\right\| < \varepsilon\right)\right]\\ &\text{then } E\left(\boldsymbol{b_{opt}}, \boldsymbol{w_{opt}}\right) \leq E(\boldsymbol{b}, \boldsymbol{w})\end{aligned}$$

- Remember that $||\boldsymbol{x}||$ denotes the Euclidean norm.
- In words, if $\boldsymbol{b_{opt}}$ and $\boldsymbol{w_{opt}}$ are **<u>locally optimal</u>**, it means that we can find no better values for $\boldsymbol{b}, \boldsymbol{w}$ that are relatively close $\boldsymbol{b_{opt}}$ and $\boldsymbol{w_{opt}}$.
  - There may be values $\boldsymbol{b}, \boldsymbol{w}$ that give a far lower SSD error, but they are not very close to $\boldsymbol{b_{opt}}$ and $\boldsymbol{w_{opt}}$.

Global and Local Optima

- In any optimization method, it is important to understand if the result is globally or locally optimal.
- For neural networks, we do **<u>not</u>** have any method that finds globally optimal solutions in a reasonable amount of time (like polynomial time).
- The standard training algorithm (called **<u>backpropagation</u>**) finds a locally optimal solution.
  - Mathematically we wish we could do better.
  - In practice, the results are often good enough, otherwise neural networks would not be as popular.

Training as an Optimization Problem

- As we said, in an optimization problem we need to:
  - Define what **parameters** we are optimizing. This is also called the **search space** (the space of possible choices).
    - For neural networks, what is that?
  - Define a quantitative **optimization criterion**.
    - For neural networks, what is that?
  - Define an **optimization algorithm** for finding a good set of parameters.
    - For neural networks, what is that?

Training as an Optimization Problem

- As we said, in an optimization problem we need to:
  - Define what **parameters** we are optimizing. This is also called the **search space** (the space of possible choices).
    - For neural networks, we search over $\boldsymbol{b}$ and $\boldsymbol{w}$.
  - Define a quantitative **optimization criterion**.
    - For neural networks, we defined the SSD error, which we want to minimize. We will also see and use other choices this semester.
  - Define an **optimization algorithm** for finding a good set of parameters.
    - We have not done this yet, that is our next topic.
    - Preview: the general method that we will use is called **gradient descent**. When used specifically for training neural networks, it is called **backpropagation**.

Gradients and Partial Derivatives

- Gradients is something that is covered in the third semester of the Calculus sequence.
- For easy reference, here is a quick description.
  - Summary: gradients are vectors of partial derivatives.
- Consider this function f:

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

- The partial derivative of f with respect to x is denoted as $\frac{\partial f}{\partial x}$.
- To compute it, we simply compute the derivative with respect to x, pretending that any other variables are constant.
  - In our example, the only other variable is y, so we pretend that y is constant.

Partial Derivatives

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

- Using the sum rule for derivatives:

$$\frac{\partial f(x,y)}{\partial x} = \frac{\partial x^2}{\partial x} + \frac{\partial 2y^2}{\partial x} + \frac{\partial(-600x)}{\partial x} + \frac{\partial(-800y)}{\partial x} + \frac{\partial xy}{\partial x} + \frac{\partial 50}{\partial x}$$

Partial Derivatives

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\frac{\partial f(x, y)}{\partial x} = \frac{\partial x^2}{\partial x} + \frac{\partial 2y^2}{\partial x} + \frac{\partial(-600x)}{\partial x} + \frac{\partial(-800y)}{\partial x} + \frac{\partial xy}{\partial x} + \frac{\partial 50}{\partial x}$$

- $\frac{\partial x^2}{\partial x} = ???$
- $\frac{\partial 2y^2}{\partial x} = ???$
- $\frac{\partial(-600x)}{\partial x} = ???$
- $\frac{\partial(-800y)}{\partial x} = ???$
- $\frac{\partial xy}{\partial x} = ???$
- $\frac{\partial 50}{\partial x} = ???$

Partial Derivatives

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\frac{\partial f(x,y)}{\partial x} = \frac{\partial x^2}{\partial x} + \frac{\partial 2y^2}{\partial x} + \frac{\partial(-600x)}{\partial x} + \frac{\partial(-800y)}{\partial x} + \frac{\partial xy}{\partial x} + \frac{\partial 50}{\partial x}$$

- $\frac{\partial x^2}{\partial x} = 2x$
- $\frac{\partial 2y^2}{\partial x} = 0$. Why? Because **we treat y as a constant**.
- $\frac{\partial(-600x)}{\partial x} = -600$
- $\frac{\partial(-800y)}{\partial x} = 0$. Why? Again, because **we treat y as a constant**.
- $\frac{\partial xy}{\partial x} = y$. Again, **we treat y as a constant**.
- $\frac{\partial 50}{\partial x} = 0$, since 50 is a constant.

Partial Derivatives

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

- Based on the previous calculations, the partial derivative $\frac{\partial f(x,y)}{\partial x}$ is:

$$\frac{\partial f(x, y)}{\partial x} = 2x - 600 + y$$

Partial Derivatives

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

- Now, let's compute the partial derivative of f with respect to y, which is denoted as $\frac{\partial f}{\partial y}$.
- To compute it, we simply compute the derivative with respect to y, pretending that any other variables are constant.
  - In our example, the only other variable is x, so we pretend that x is constant.

Partial Derivatives

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

- Using the sum rule for derivatives:

$$\frac{\partial f(x, y)}{\partial y} = \frac{\partial x^2}{\partial y} + \frac{\partial 2y^2}{\partial y} + \frac{\partial(-600x)}{\partial y} + \frac{\partial(-800y)}{\partial y} + \frac{\partial xy}{\partial y} + \frac{\partial 50}{\partial y}$$

Partial Derivatives

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\frac{\partial f(x, y)}{\partial y} = \frac{\partial x^2}{\partial y} + \frac{\partial 2y^2}{\partial y} + \frac{\partial(-600x)}{\partial y} + \frac{\partial(-800y)}{\partial y} + \frac{\partial xy}{\partial y} + \frac{\partial 50}{\partial y}$$

- $\frac{\partial x^2}{\partial y} =???$
- $\frac{\partial 2y^2}{\partial y} =???$
- $\frac{\partial(-600x)}{\partial y} =???$
- $\frac{\partial(-800y)}{\partial y} =???$
- $\frac{\partial xy}{\partial y} =???$
- $\frac{\partial 50}{\partial y} =???$

Partial Derivatives

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\frac{\partial f(x, y)}{\partial y} = \frac{\partial x^2}{\partial y} + \frac{\partial 2y^2}{\partial y} + \frac{\partial(-600x)}{\partial y} + \frac{\partial(-800y)}{\partial y} + \frac{\partial xy}{\partial y} + \frac{\partial 50}{\partial y}$$

- $\frac{\partial x^2}{\partial y} = 0.$ Now we treat x as a constant.
- $\frac{\partial 2y^2}{\partial y} = 4y$
- $\frac{\partial(-600x)}{\partial y} = 0.$ Again, we treat x as a constant.
- $\frac{\partial(-800y)}{\partial y} = -800$
- $\frac{\partial xy}{\partial y} = x.$ Again, we treat x as a constant.
- $\frac{\partial 50}{\partial y} = 0,$ since 50 is a constant.

Partial Derivatives

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

- Based on the previous calculations, the partial derivative $\frac{\partial f(x,y)}{\partial y}$ is:

$$\frac{\partial f(x, y)}{\partial y} = 4y - 800 + x$$

Gradients

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

- So, the two partial derivatives are:

$$\frac{\partial f(x, y)}{\partial x} = 2x - 600 + y, \qquad \frac{\partial f(x, y)}{\partial y} = 4y - 800 + x$$

- The **gradient vector** $\nabla f(x, y)$ is simply the vector of the partial derivatives.

$$\nabla f(x, y) = \left(\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y} \right) = (2x - 600 + y, 4y - 800 + x)$$

Gradients

- Formally: suppose that f is a function from $\mathbb{R}^D$ to $\mathbb{R}$.
  - In other words, the input to f is a D-dimensional vector, and the output of f is a real number.
- Then, the gradient ∇f is a function from $\mathbb{R}^D$ to $\mathbb{R}^D$.
- If $\boldsymbol{x} = (x_1, x_2, \dots, x_D)$ is a D-dimensional vector, then the gradient vector $\nabla f(\boldsymbol{x})$ is defined as the vector of all partial derivatives $\frac{\partial f(\boldsymbol{x})}{\partial x_i}$:

$$\nabla f(\boldsymbol{x}) = \left(\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \dots, \frac{\partial f(\boldsymbol{x})}{\partial x_D} \right)$$

Gradients and Neural Networks

- Gradients can be used to find local minima.
- In training a neural network, we typically want to find a local minimum of the optimization criterion.
  - For example, the optimization criterion can be the sum of squared differences.
- So, we need to review how gradients are used in such problems.
- The method is called **<u>gradient descent</u>**.

Direction of the Gradient

- The gradient vector points towards the direction where the function increases the fastest.
- The opposite direction is the direction where the function increases the slowest.
- If we look at our previous example:

$$f(x,y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\nabla f(x,y) = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right) = (2x - 600 + y, 4y - 800 + x)$$

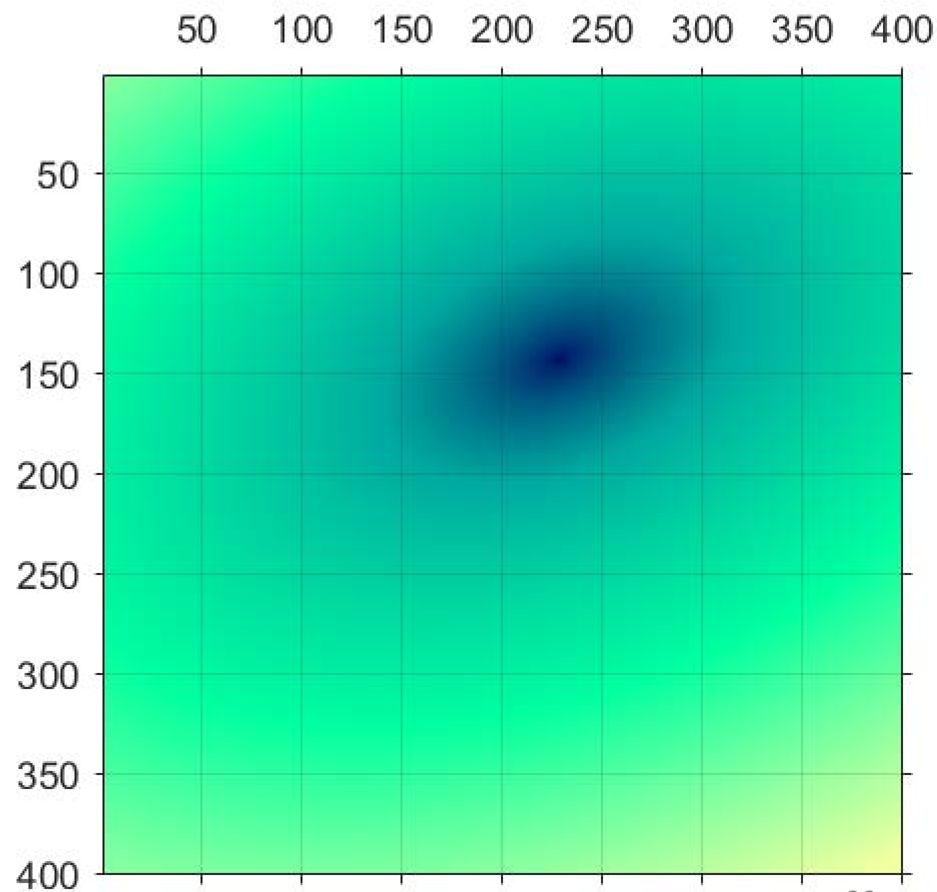
- If we choose any point (x,y), the gradient vector $\nabla f(x,y)$ tells us towards which direction the function increases and decreases the fastest.

Some Examples

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\nabla f(x, y) = \left(\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y}\right) = (2x - 600 + y, 4y - 800 + x)$$

- Here is a visualization of this function, for the region: 0 < x < 400, 0 < y < 400
- Larger values are yellow (see bottom left of figure).
- Middle values are green.
- Low values are blue.
- The lowest values are black.

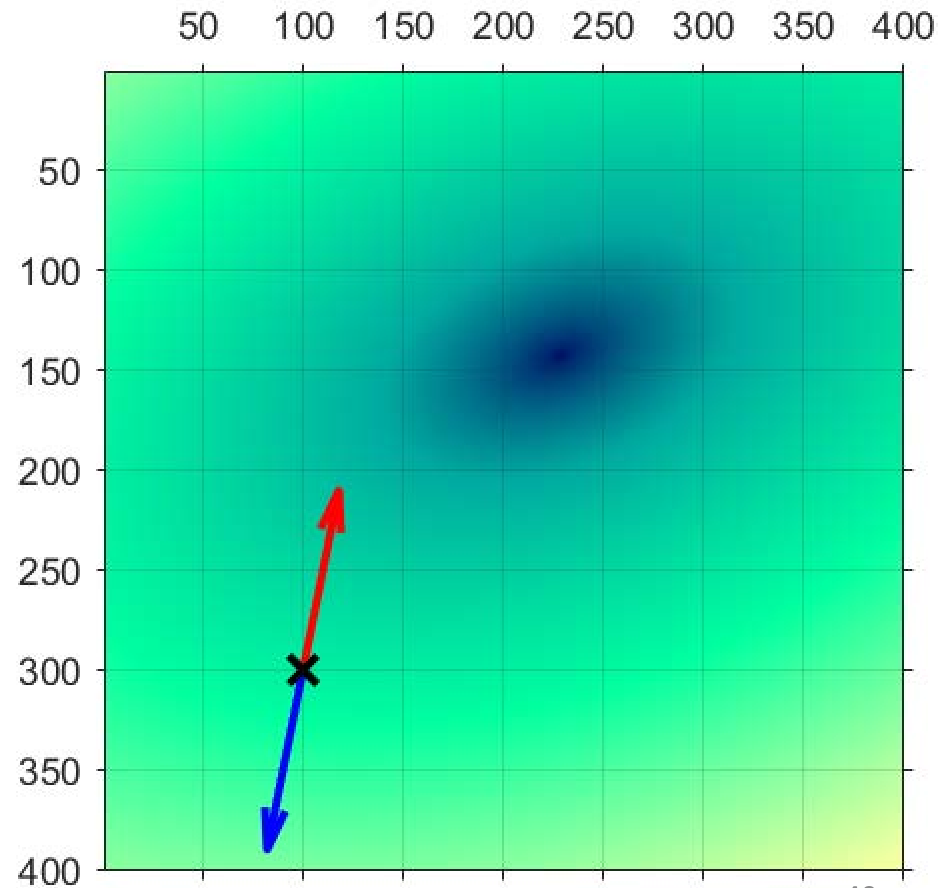


Some Examples

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\nabla f(x, y) = \left(\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y}\right) = (2x - 600 + y, 4y - 800 + x)$$

- We choose (arbitrarily) point (100, 300), shown as ×.
- We calculate the gradient, it is equal to (-100, 500).
- We plot two arrows:
  - The blue arrow points in the direction of the gradient (downwards and a bit to the left).
  - The red arrow points in the opposite direction.

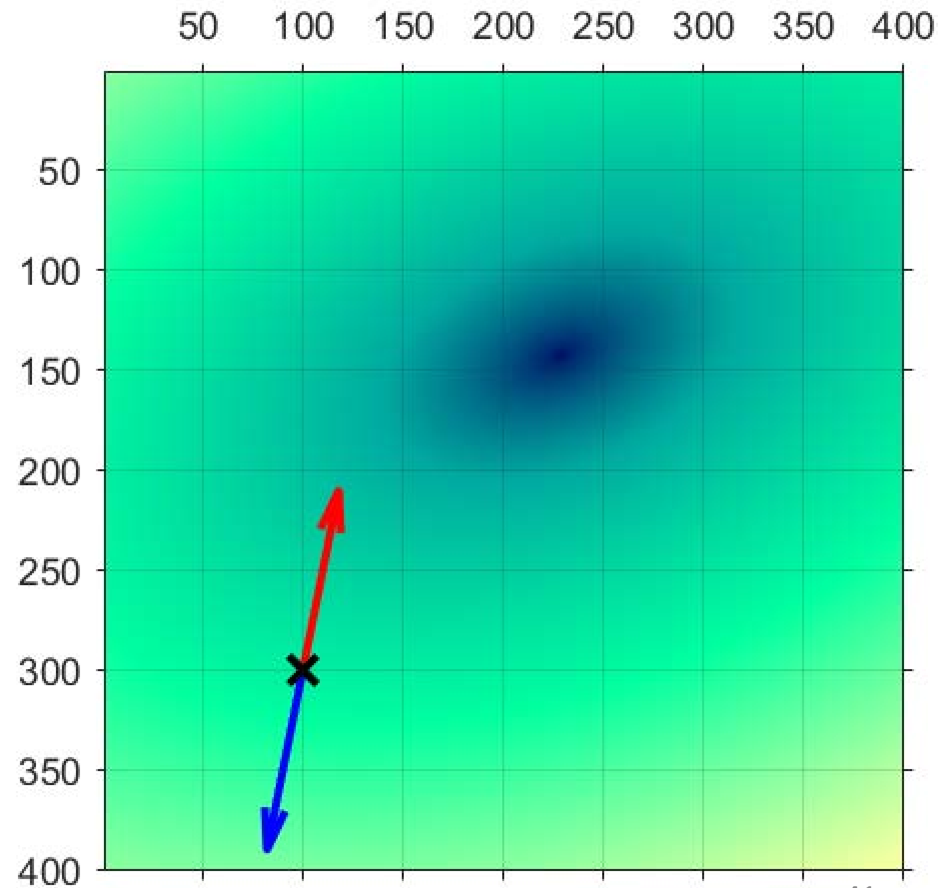


Some Examples

$$f(x,y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\nabla f(x,y) = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right) = (2x - 600 + y, 4y - 800 + x)$$

- We can see that:
  - The function values increase (at least for a while) if we start moving towards the direction of the gradient.
  - The function values decrease (again, at least for a while) if we start moving in the opposite direction.

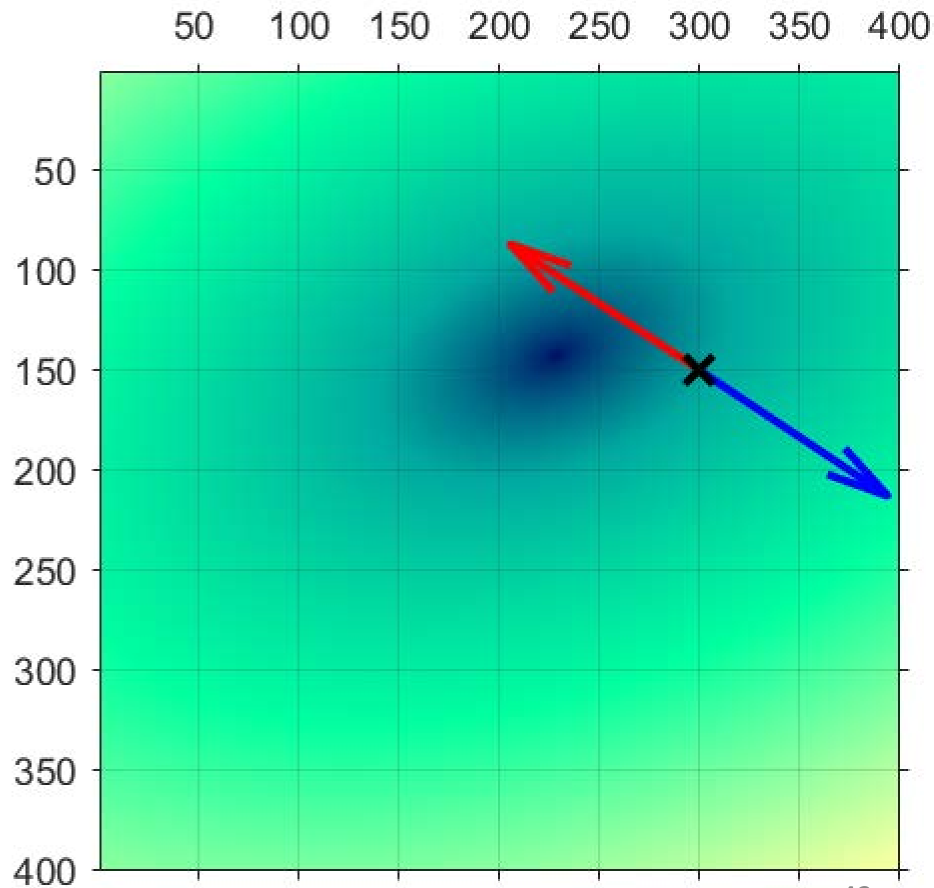


Some Examples

$$f(x,y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\nabla f(x,y) = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right) = (2x - 600 + y, 4y - 800 + x)$$

- We now choose another point, (300, 150), shown as X.
- We calculate the gradient, it is equal to (150, 100).
- Again, we plot two arrows:
  - One pointing towards the direction of the gradient (downwards and to the right).
  - One pointing in the opposite direction.

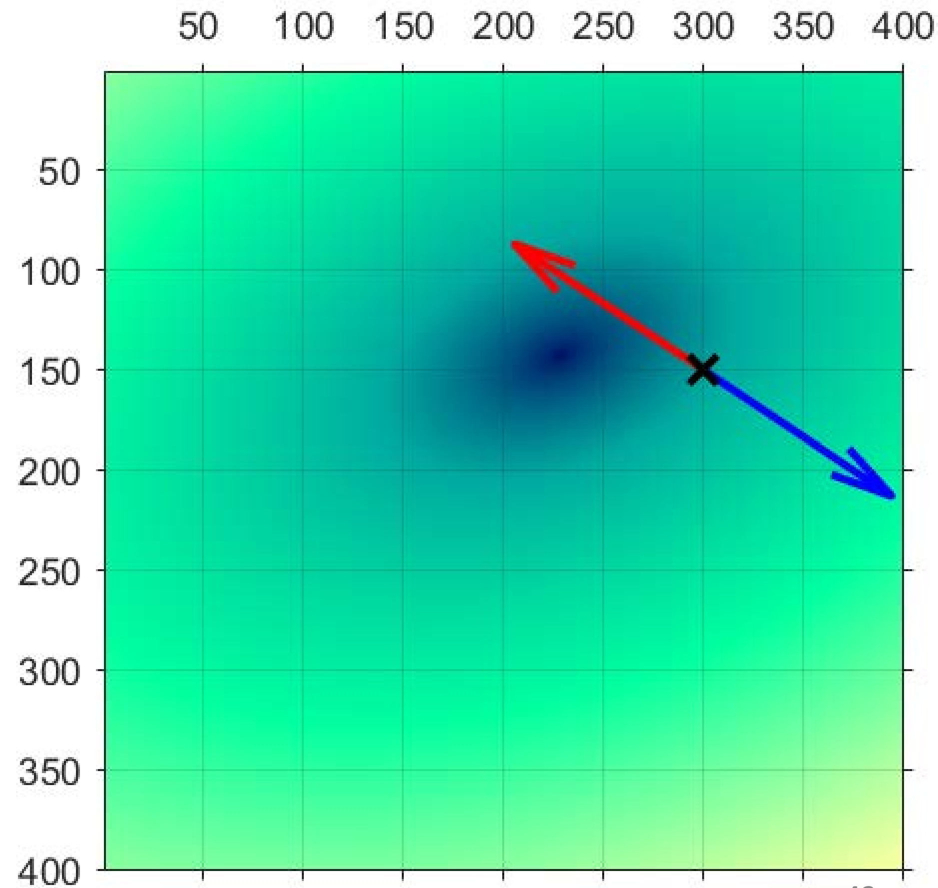


Some Examples

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\nabla f(x, y) = \left(\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y}\right) = (2x - 600 + y, 4y - 800 + x)$$

- Again, we see that:
  - The function values increase if we start moving (at least for a while) towards the direction of the gradient.
  - The function values decrease (at least for a while) in the opposite direction.
    - Note that, in this example, after a bit the values start increasing again.

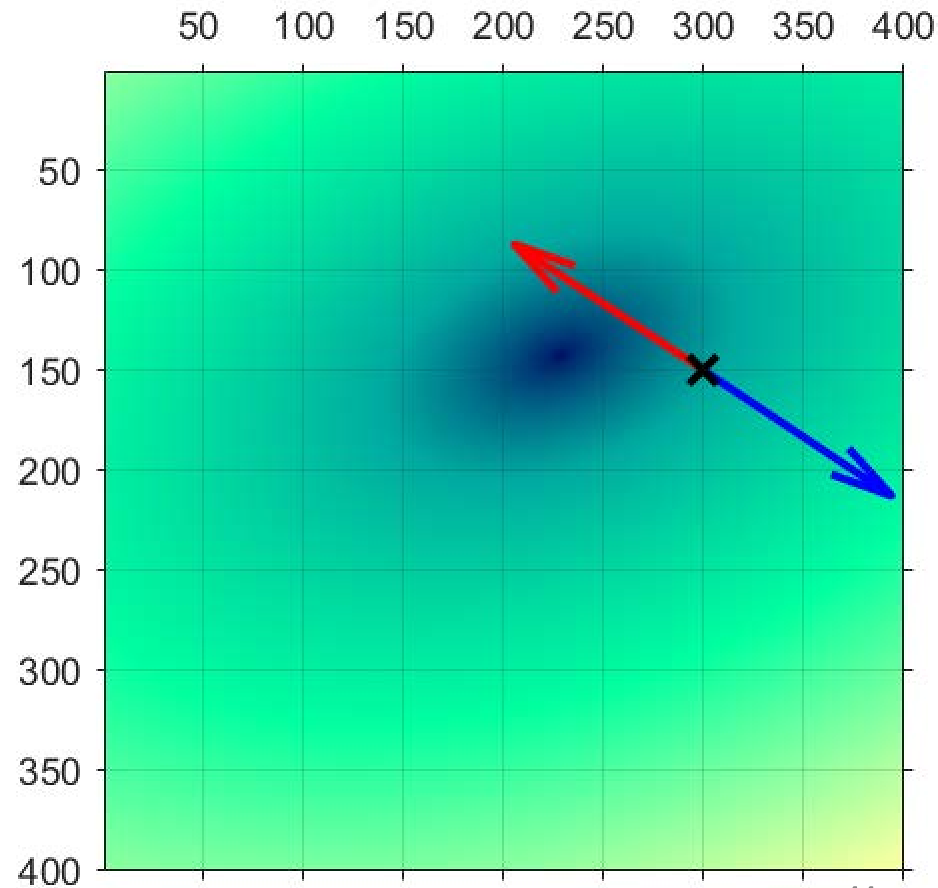


Gradient Descent

$$f(x,y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\nabla f(x,y) = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right) = (2x - 600 + y, 4y - 800 + x)$$

- Suppose that we want to find a local minimum of function f.
- **Gradient descent** is a method for doing that.



Gradient Descent

Gradient descent pseudocode (still too vague, we will see a fully specified version in a bit):

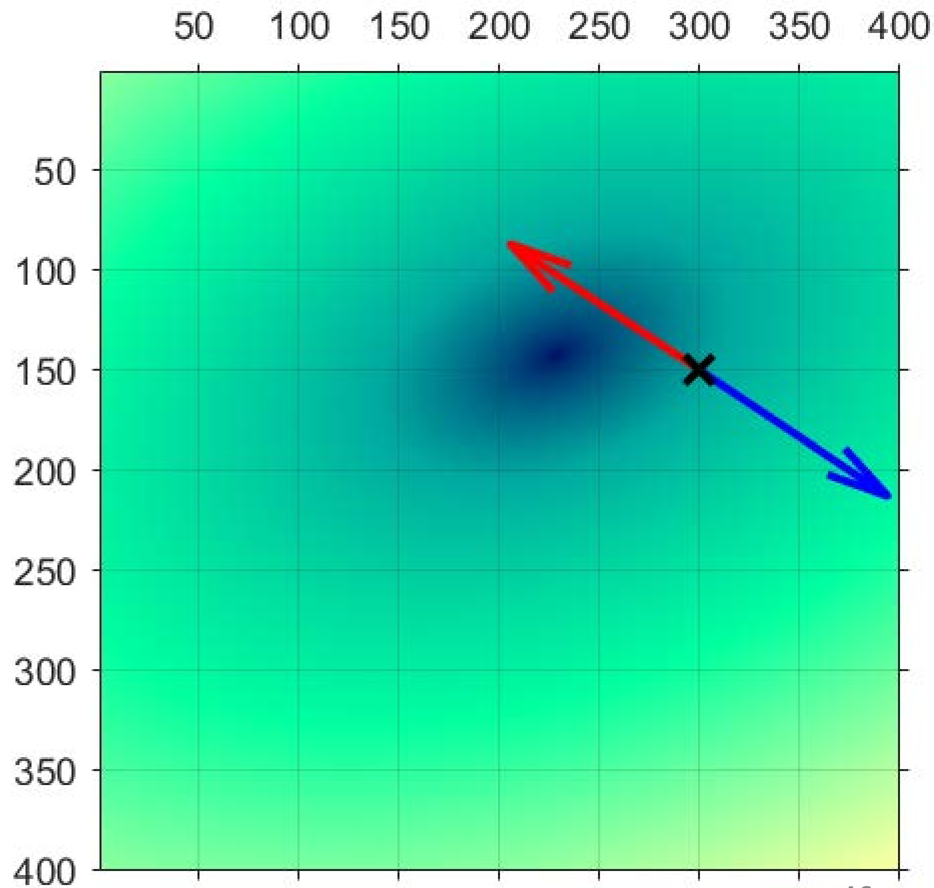
1. Choose (randomly or however else you want) some starting point (x, y).
2. Compute gradient $\nabla f(x, y)$.
3. Compute new (x, y) by starting at (x, y) and moving opposite to the direction of the $\nabla f(x, y)$.
   - This is still too vague: **<u>How much do we move</u>**? We will discuss this in a bit.
4. Decide whether we are done. If we are done, return the new (x, y).
   - For example, check if the distance from the new (x, y) to the old (x, y) is less than a threshold ε.
5. Go back to Step 2.

Gradient Descent

$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\nabla f(x, y) = \left(\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y}\right) = (2x - 600 + y, 4y - 800 + x)$$

- Suppose that we start at position $(x_1, y_1) = (300,150)$.
- The gradient there is $(150,100)$.
- The next position (x_2, y_2) should be obtained by moving "in the opposite direction of the gradient".
- Key question: how far do we move?



Gradient Descent

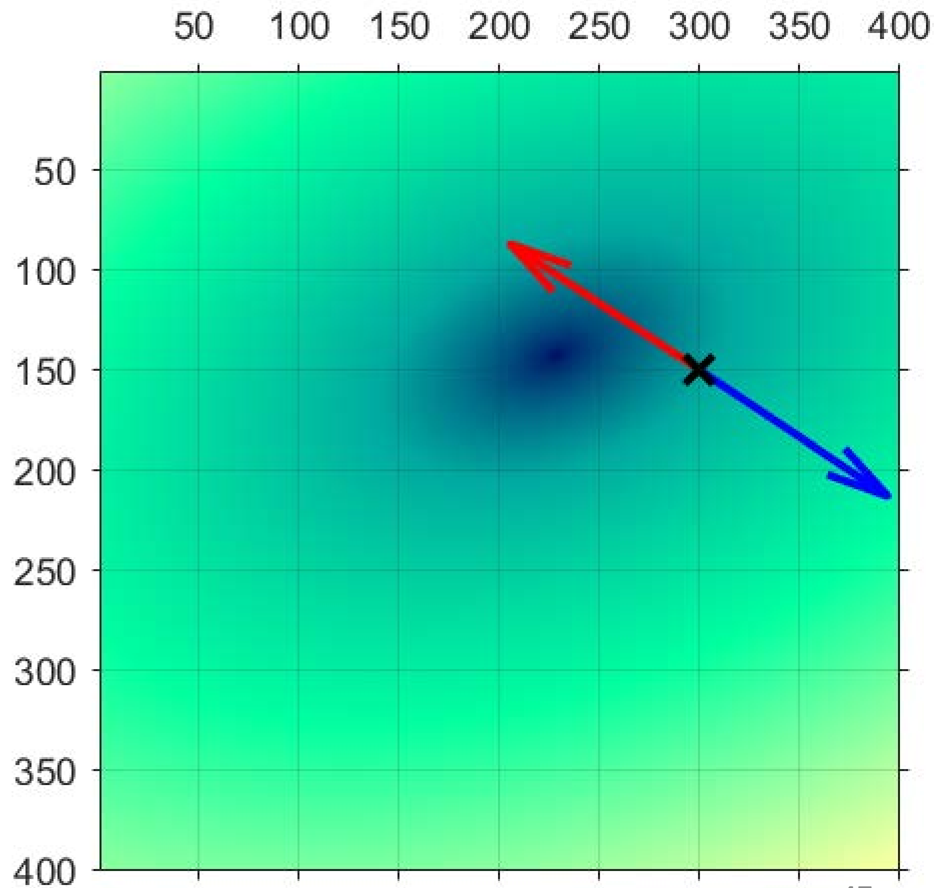
$$f(x,y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\nabla f(x,y) = \left(\frac{\partial f(x,y)}{\partial x}, \frac{\partial f(x,y)}{\partial y}\right) = (2x - 600 + y, 4y - 800 + x)$$

- Suppose that we start at position $(x_1, y_1) = (300,150)$.
- The gradient there is $(150,100)$.
- The next position (x_2, y_2) should be obtained by moving "in the opposite direction of the gradient".
- Mathematically:

$$(x_2, y_2) = (x_1, y_1) - \eta * \nabla f(x_1, y_1)$$

- The question is, what is a good value for η?



Gradient Descent

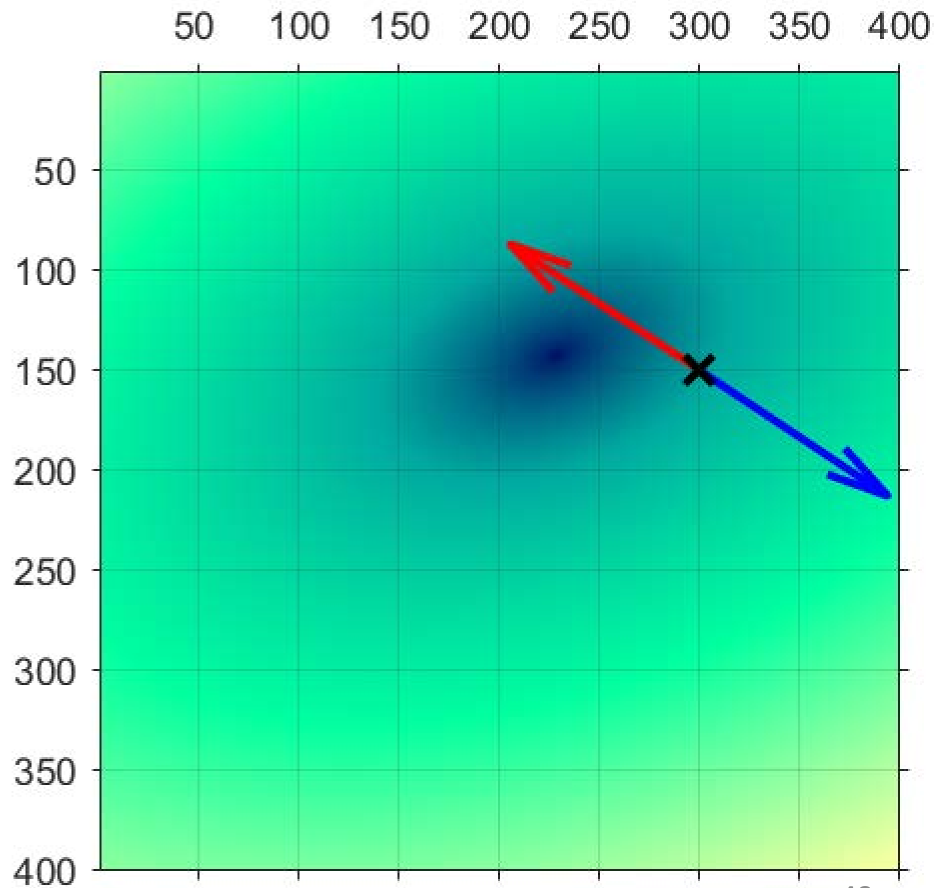
$$f(x, y) = x^2 + 2y^2 - 600x - 800y + x * y + 50$$

$$\nabla f(x, y) = \left(\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y}\right) = (2x - 600 + y, 4y - 800 + x)$$

- $(x_1, y_1) = (300{,}150)$.
- $\nabla f(x_1, y_1) = (150{,}100)$.

$$(x_2, y_2) = (x_1, y_1) - \eta * \nabla f(x_1, y_1)$$

- Parameter η is a hyperparameter.
- There are complicated ways that guarantee a good value for η, in some situations.
- For our example, we will do it the simple and hacky way: start with $\eta = 1$.



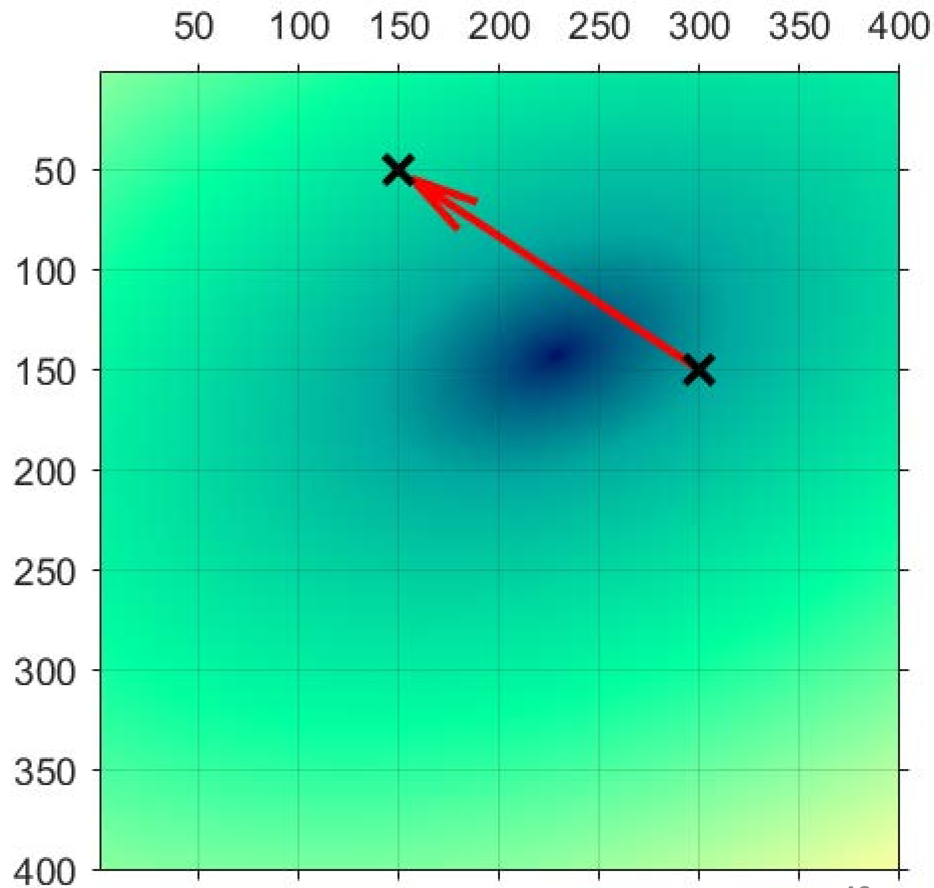
Gradient Descent

- $(x_1, y_1) = (300,150)$.

- $\nabla f(x_1, y_1) = (150,100)$.

$$
\begin{aligned}
(x_2, y_2) &= (x_1, y_1) - \eta * \nabla f(x_1, y_1) \\
&= (300,150) - 1 * (150,100) \\
&= (150,50)
\end{aligned}
$$

- We moved too far.
- Visually, the region around (x_2, y_2) is brighter.
- If we do the math, we can verify that $f(x_2, y_2) > f(x_1, y_1)$.



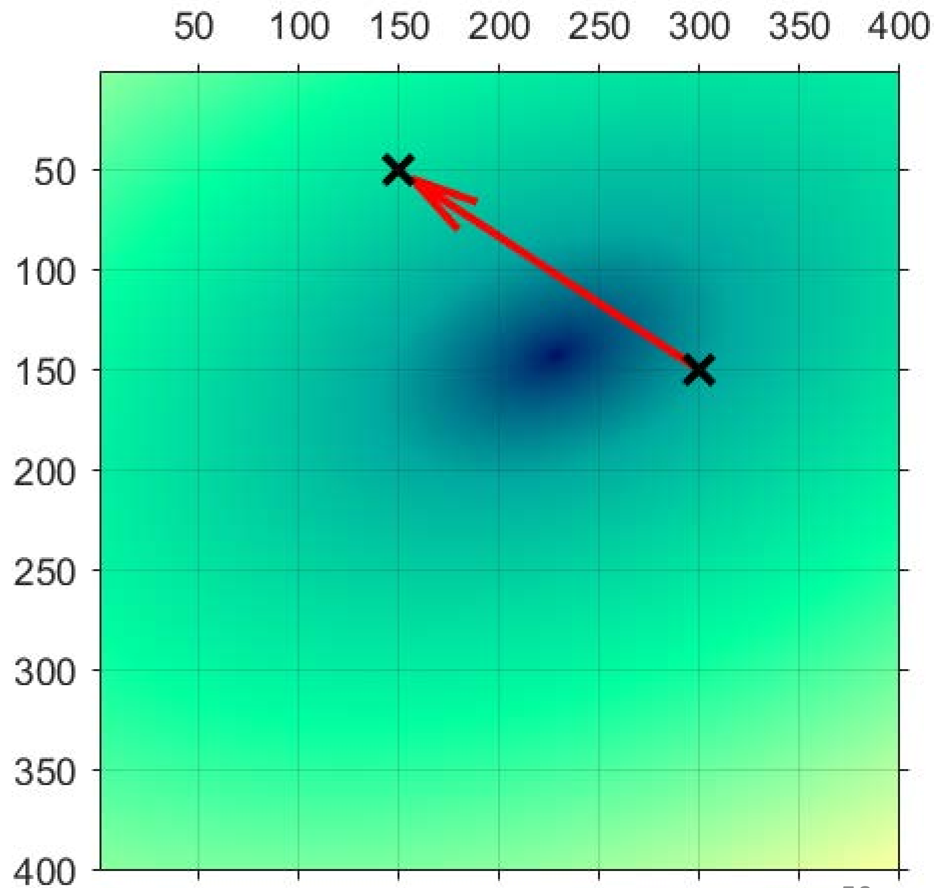
Gradient Descent

- $(x_1, y_1) = (300{,}150)$.

- $\nabla f(x_1, y_1) = (150{,}100)$.

$$\begin{aligned}(x_2, y_2) &= (x_1, y_1) - \eta * \nabla f(x_1, y_1) \\ &= (300{,}150) - 1 * (150{,}100) \\ &= (150{,}50)\end{aligned}$$

- We moved too far.
- However, our code can easily detect and fix this problem.
  - How do we detect the problem?
  - How do we fix it?

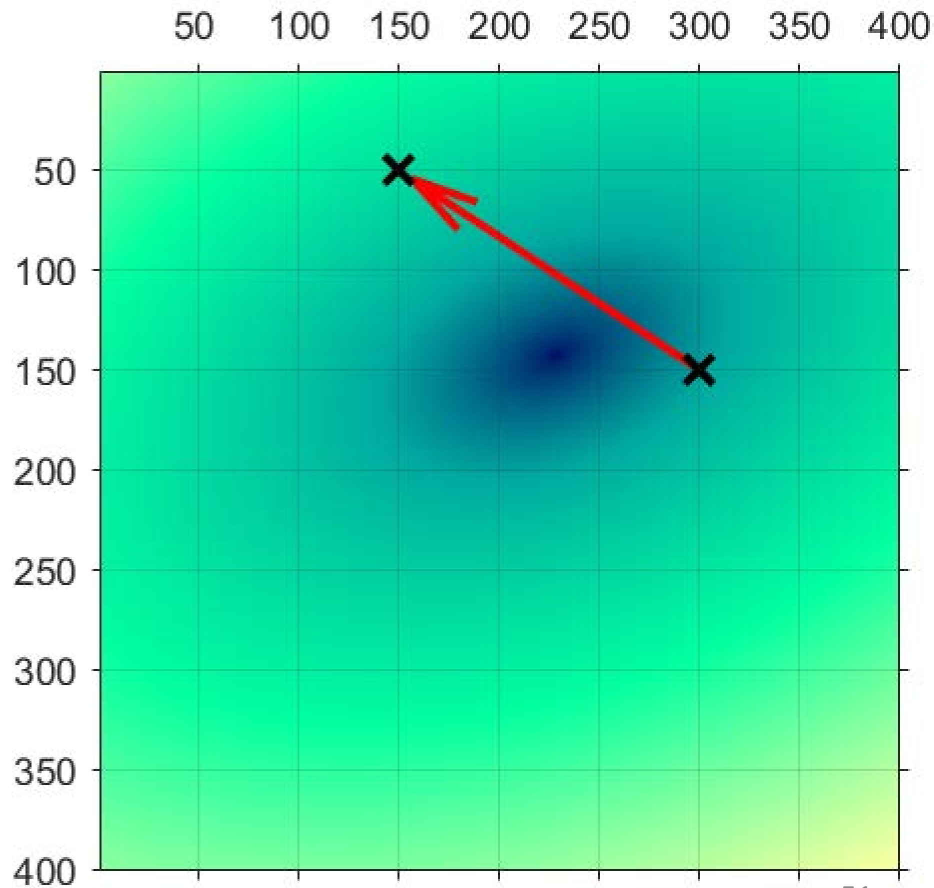


Gradient Descent

- $(x_1, y_1) = (300{,}150)$.
- $\nabla f(x_1, y_1) = (150{,}100)$.

$$\begin{aligned}(x_2, y_2) &= (x_1, y_1) - \eta * \nabla f(x_1, y_1) \\ &= (300{,}150) - 1 * (150{,}100) \\ &= (150{,}50)\end{aligned}$$

- We moved too far.
- How do we detect the problem?
  - If $f(x_2, y_2) > f(x_1, y_1)$, we have a problem.
- How do we fix it?
  - Reset η to a smaller value, like half its previous value, and try again.
- So, what would be the new η?



Gradient Descent

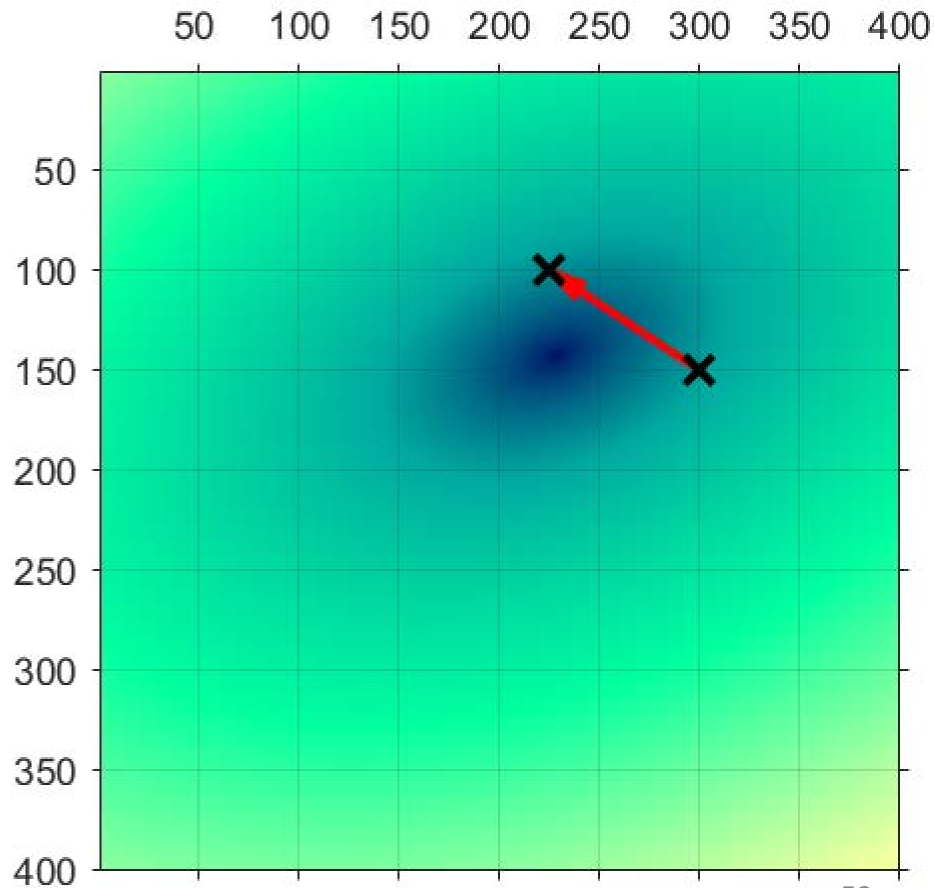
- Our new η is 0.5:

$(x_1, y_1) = (300{,}150)$.

$\nabla f(x_1, y_1) = (150{,}100)$.

$$\begin{aligned}(x_2, y_2) &= (x_1, y_1) - \eta * \nabla f(x_1, y_1) \\ &= (300{,}150) - 0.5 * (150{,}100) \\ &= (225{,}100)\end{aligned}$$

- The function value at (225,100) is indeed smaller than at (300, 150), as we can see by the darker color.
  - Again, we can verify by doing the math.
- What do we do next?



Gradient Descent

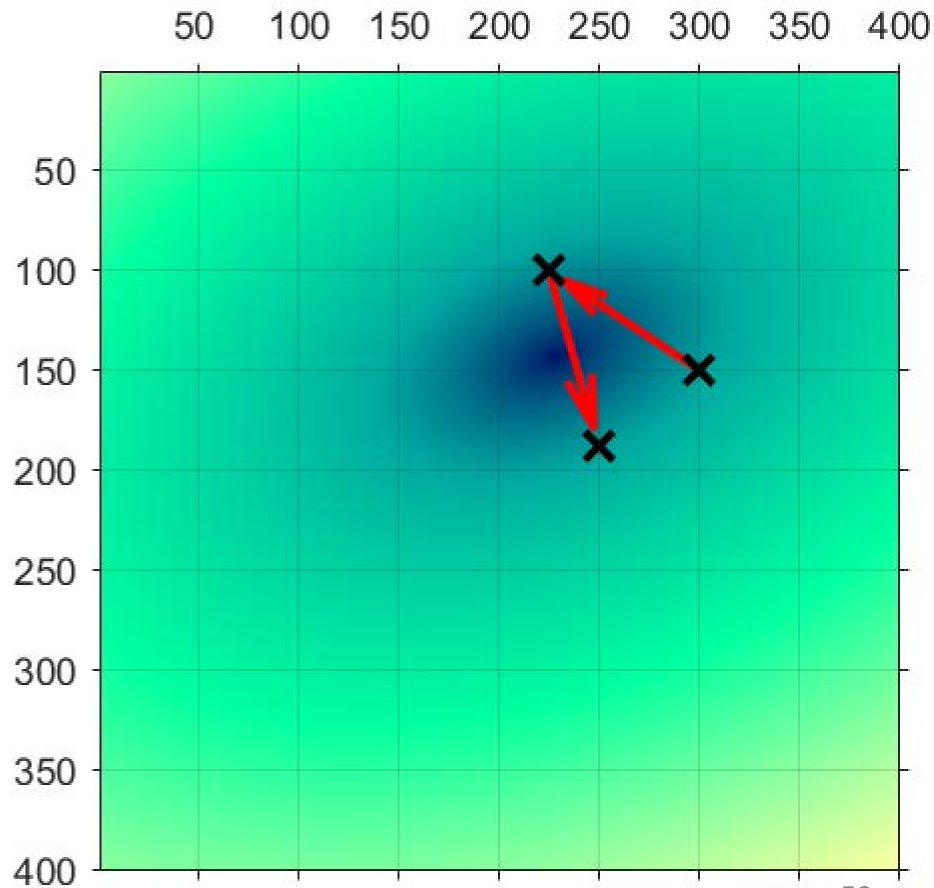
- Our progress so far:

$(x_1, y_1) = (300{,}150)$.
$(x_2, y_2) = (225{,}100)$.

Current value for η is 0.5.

- Next step: compute the next point in our descent, (x_3, y_3), based on the gradient at (x_2, y_2).

$$
\begin{aligned}
(x_3, y_3) &= (x_2, y_2) - \eta * \nabla f(x_2, y_2) \\
&= (225{,}100) - 0.5 * (-50, -175) \\
&= (250, 187.50)
\end{aligned}
$$



Gradient Descent

- Our progress so far:

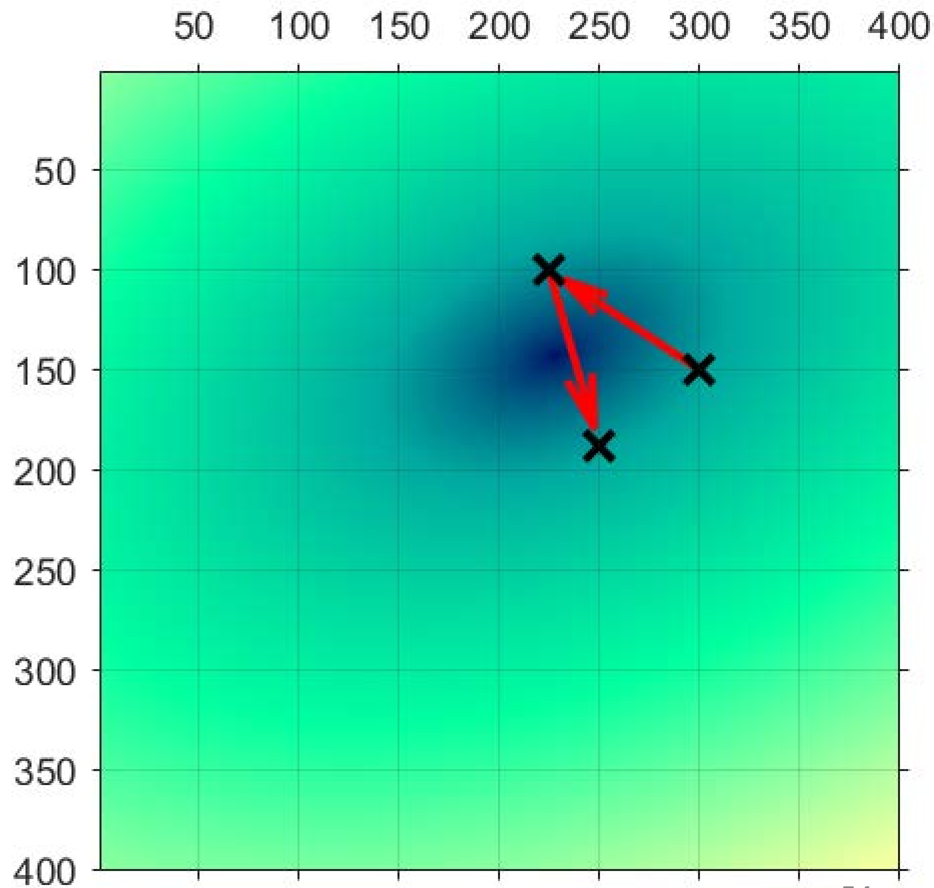
$(x_1, y_1) = (300,150)$.
$(x_2, y_2) = (225,100)$.

Current value for η is 0.5.

- Next step: compute the next point in our descent, (x_3, y_3), based on the gradient at (x_2, y_2).

$$
\begin{aligned}
(x_3, y_3) &= (x_2, y_2) - \eta * \nabla f(x_2, y_2) \\
&= (225,100) - 0.5 * (-50, -175) \\
&= (250, 187.50)
\end{aligned}
$$

- Turns out that, again, η was too large, and $f(x_3, y_3) > f(x_2, y_2)$.
- So, we try again with new $\eta = 0.25$.



Gradient Descent

- Our progress so far:

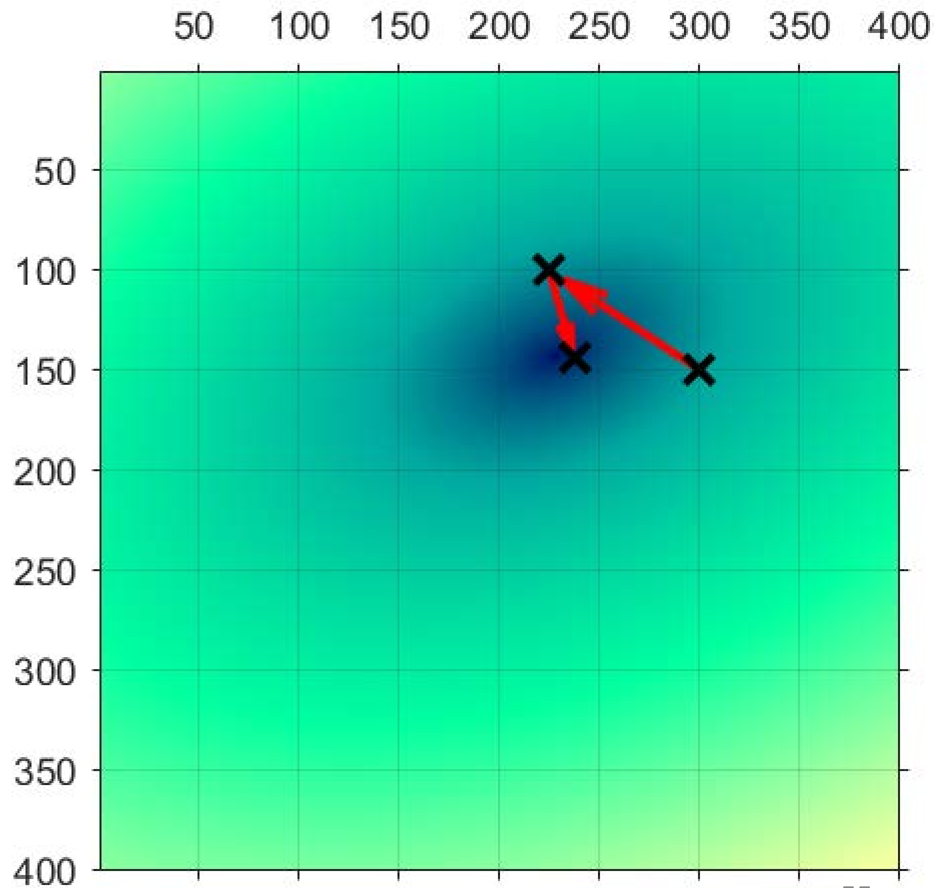
$(x_1, y_1) = (300{,}150)$.
$(x_2, y_2) = (225{,}100)$.

Current value for η is 0.25.

$$
\begin{aligned}
(x_3, y_3) &= (x_2, y_2) - \eta * \nabla f(x_2, y_2) \\
&= (225{,}100) - 0.25 * (-50, -175) \\
&= (237.5, 143.75)
\end{aligned}
$$

- This move was useful:
  $f(x_3, y_3) < f(x_2, y_2)$.

- This is a process that is easy to implement.
- If we continue, after 25 steps, we get (numerically close) to the minimum.



Gradients at Local Minima

$$(x_{t+1}, y_{t+1}) = (x_t, y_t) - \eta * \nabla f(x_t, y_t)$$

- Mathematically, if (x, y) is a local minimum, then $\nabla f(x, y\ ) = ???$

Gradients at Local Minima

$$(x_{t+1}, y_{t+1}) = (x_t, y_t) - \eta * \nabla f(x_t, y_t)$$

- Mathematically, if (x, y) is a local minimum, then $\nabla f(x, y) = \mathbf{0}$ (the zero **vector**, not a single number).
- So, if (x_t, y_t) is a local minimum, there will be no updates anymore.
- In practice, as we get closer and closer to the local minimum, the gradient eventually starts getting closer and closer to the zero vector.
- Therefore, the norm of the gradient vector can be used as a stopping criterion.

Gradient Descent Pseudocode

```
# This is a simplified version, but it still works in many cases
# (x_1, y_1) is the starting point for the descent.
GradientDescent(f, x_1, y_1, η, ε)
        t = 1
        history = [(x_1, y_1)]
        while (‖∇f(x_t, y_t)‖ > ε)
                (x_{t+1}, y_{t+1}) = (x_t, y_t) − η * ∇f(x_t, y_t)
                If (f(x_{t+1}, y_{t+1}) > f(x_t, y_t))
                        η = η/2
                        continue
                Else
                        add (x_{t+1}, y_{t+1}) to end of history
                        t = t + 1
        return (x_t, y_t, history)
```

Further Reading

- Gradient descent is a widely applied method, both in machine learning and in many other fields.
- If you are interested in more details (like how to choose η, a good starting point is these Wikipedia articles:
  - Gradient descent: https://en.wikipedia.org/wiki/Gradient_descent
  - Stochastic gradient descent: https://en.wikipedia.org/wiki/Stochastic_gradient_descent
- Technically, we will train neural networks using **stochastic gradient descent**, but most of the time I will just be using the term "gradient descent".

Next Steps

- Our topic is (still) how to train a neural network.
- We will first apply gradient descent to train a perceptron.
- Then we will apply gradient descent to train a neural network.
- As a reminder, the application of gradient descent to neural networks is called **backpropagation**.