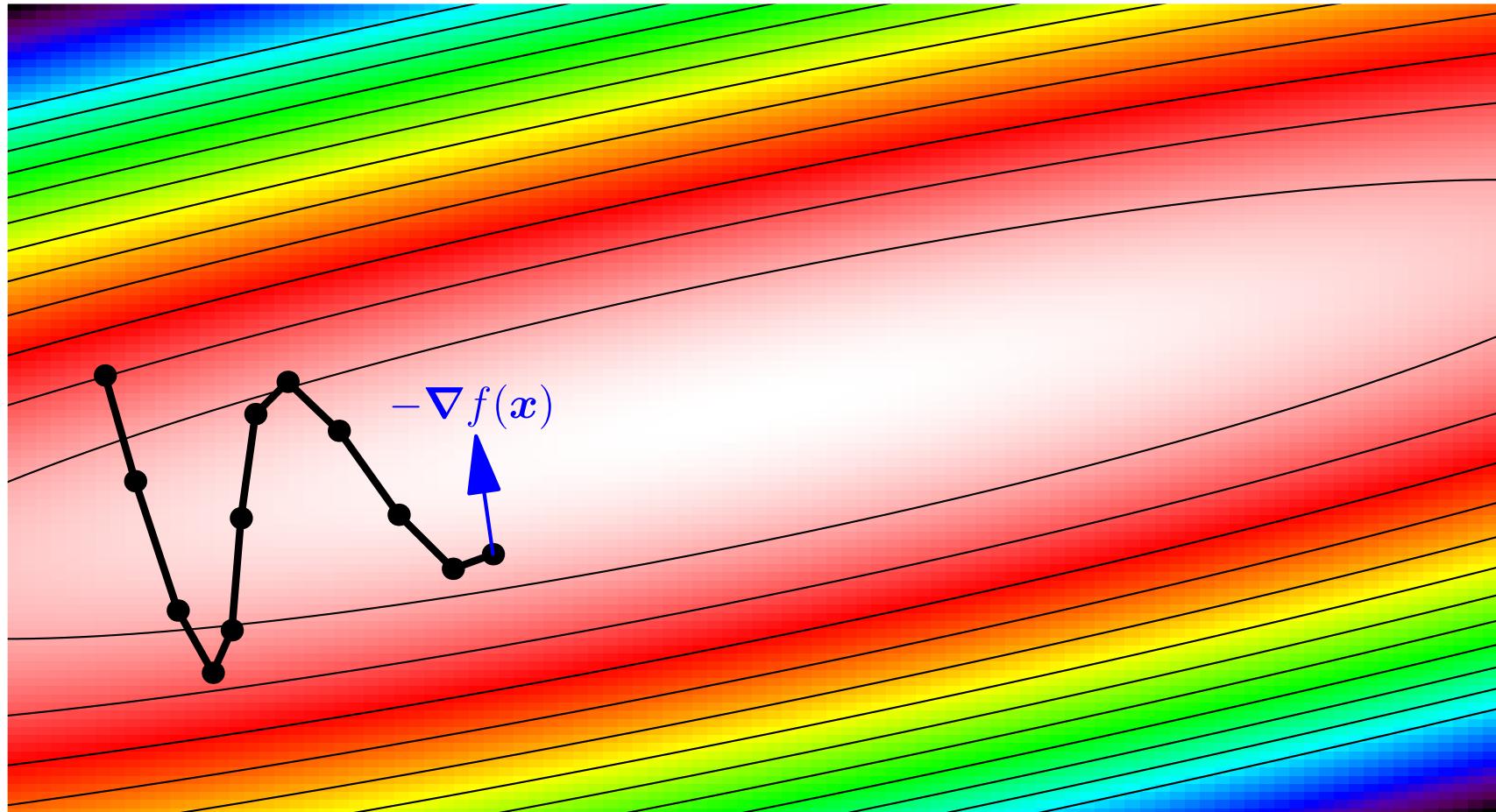


Advanced Machine Learning

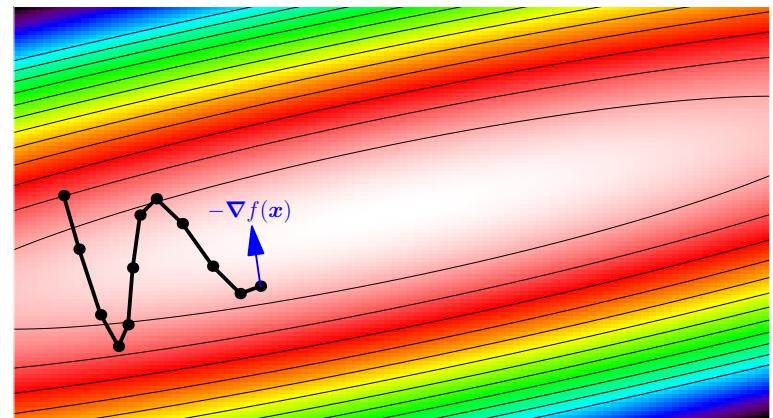
Stochastic Gradient Descent



SGD, momentum, step size, ADAM

Outline

1. SGD
2. Momentum
3. Loss Landscapes



The Next Step

- In most machine learning problems we design a network and a loss function
- The rest requires simple optimisation
- Although Newton and Quasi-Newton methods ensure rapid convergence for many tasks this apparent advantage is delusionary
- Since the raise of deep learning, gradient descent and its variants have become dominant

The Next Step

- In most machine learning problems we design a network and a loss function
- The rest requires simple optimisation
- Although Newton and Quasi-Newton methods ensure rapid convergence for many tasks this apparent advantage is delusionary
- Since the raise of deep learning, gradient descent and its variants have become dominant

The Next Step

- In most machine learning problems we design a network and a loss function
- The rest requires simple optimisation
- Although Newton and Quasi-Newton methods ensure rapid convergence for many tasks this apparent advantage is delusionary
- Since the raise of deep learning, gradient descent and its variants have become dominant

The Next Step

- In most machine learning problems we design a network and a loss function
- The rest requires simple optimisation
- Although Newton and Quasi-Newton methods ensure rapid convergence for many tasks this apparent advantage is delusionary
- Since the raise of deep learning, gradient descent and its variants have become dominant

Mini-Batch Learning

- Rather than computing the gradient of the loss function, e.g.

$$\nabla L_{\mathcal{D}}(\mathbf{w}) = \nabla \sum_{(\mathbf{x},y) \in \mathcal{D}} L(f(\mathbf{x}|\mathbf{w}), y)^2$$

- We often compute an approximation

$$\nabla L_{\mathcal{B}}(\mathbf{w}) = \nabla \sum_{(\mathbf{x},y) \in \mathcal{B}} L(f(\mathbf{x}|\mathbf{w}), y)^2$$

- where $\mathcal{B} \subset \mathcal{D}$ is a randomly sampled subset of the training set
- Usually $|\mathcal{B}| \ll |\mathcal{D}|$

Mini-Batch Learning

- Rather than computing the gradient of the loss function, e.g.

$$\nabla L_{\mathcal{D}}(\mathbf{w}) = \nabla \sum_{(\mathbf{x},y) \in \mathcal{D}} L(f(\mathbf{x}|\mathbf{w}), y)^2$$

- We often compute an approximation

$$\nabla L_{\mathcal{B}}(\mathbf{w}) = \nabla \sum_{(\mathbf{x},y) \in \mathcal{B}} L(f(\mathbf{x}|\mathbf{w}), y)^2$$

- where $\mathcal{B} \subset \mathcal{D}$ is a randomly sampled subset of the training set
- Usually $|\mathcal{B}| \ll |\mathcal{D}|$

Mini-Batch Learning

- Rather than computing the gradient of the loss function, e.g.

$$\nabla L_{\mathcal{D}}(\mathbf{w}) = \nabla \sum_{(\mathbf{x}, y) \in \mathcal{D}} L(f(\mathbf{x}|\mathbf{w}), y)^2$$

- We often compute an approximation

$$\nabla L_{\mathcal{B}}(\mathbf{w}) = \nabla \sum_{(\mathbf{x}, y) \in \mathcal{B}} L(f(\mathbf{x}|\mathbf{w}), y)^2$$

- where $\mathcal{B} \subset \mathcal{D}$ is a randomly sampled subset of the training set
- Usually $|\mathcal{B}| \ll |\mathcal{D}|$

Stochastic Gradient Descent

- Stochastic gradient descent (SGD) follows the gradient of mini-batches
- Computationally this is efficient because it is much quicker to compute $\nabla L_{\mathcal{B}}(\mathbf{w})$ than $\nabla L_{\mathcal{D}}(\mathbf{w})$
- The batch gradients add noise (are stochastic) as we are only sampling the dataset
- This noise might help escape local-minima
- Taking many smaller steps of approximate gradients reduces the likelihood of divergence

Stochastic Gradient Descent

- Stochastic gradient descent (SGD) follows the gradient of mini-batches
- Computationally this is efficient because it is much quicker to compute $\nabla L_{\mathcal{B}}(\mathbf{w})$ than $\nabla L_{\mathcal{D}}(\mathbf{w})$
- The batch gradients add noise (are stochastic) as we are only sampling the dataset
- This noise might help escape local-minima
- Taking many smaller steps of approximate gradients reduces the likelihood of divergence

Stochastic Gradient Descent

- Stochastic gradient descent (SGD) follows the gradient of mini-batches
- Computationally this is efficient because it is much quicker to compute $\nabla L_{\mathcal{B}}(\mathbf{w})$ than $\nabla L_{\mathcal{D}}(\mathbf{w})$
- The batch gradients add noise (are stochastic) as we are only sampling the dataset
- This noise might help escape local-minima
- Taking many smaller steps of approximate gradients reduces the likelihood of divergence

Stochastic Gradient Descent

- Stochastic gradient descent (SGD) follows the gradient of mini-batches
- Computationally this is efficient because it is much quicker to compute $\nabla L_{\mathcal{B}}(\mathbf{w})$ than $\nabla L_{\mathcal{D}}(\mathbf{w})$
- The batch gradients add noise (are stochastic) as we are only sampling the dataset
- This noise might help escape local-minima
- Taking many smaller steps of approximate gradients reduces the likelihood of divergence

Stochastic Gradient Descent

- Stochastic gradient descent (SGD) follows the gradient of mini-batches
- Computationally this is efficient because it is much quicker to compute $\nabla L_{\mathcal{B}}(\mathbf{w})$ than $\nabla L_{\mathcal{D}}(\mathbf{w})$
- The batch gradients add noise (are stochastic) as we are only sampling the dataset
- This noise might help escape local-minima (**but I'm not sure there is compelling evidence for this**)
- Taking many smaller steps of approximate gradients reduces the likelihood of divergence

Stochastic Gradient Descent

- Stochastic gradient descent (SGD) follows the gradient of mini-batches
- Computationally this is efficient because it is much quicker to compute $\nabla L_{\mathcal{B}}(\mathbf{w})$ than $\nabla L_{\mathcal{D}}(\mathbf{w})$
- The batch gradients add noise (are stochastic) as we are only sampling the dataset
- This noise might help escape local-minima (but I'm not sure there is compelling evidence for this)
- Taking many smaller steps of approximate gradients reduces the likelihood of divergence

Automatic Differentiation

- For gradient descent we need to compute the gradient
- For most of my life this was just a pain
- You guys have it easy, modern deep learning frameworks do this for you automatically
- This is a *game changer!* We are no longer afraid of large models
- We can differentiate through algorithms allowing us to train incredible sophisticated networks

Automatic Differentiation

- For gradient descent we need to compute the gradient
- For most of my life this was just a pain
- You guys have it easy, modern deep learning frameworks do this for you automatically
- This is a *game changer!* We are no longer afraid of large models
- We can differentiate through algorithms allowing us to train incredible sophisticated networks

Automatic Differentiation

- For gradient descent we need to compute the gradient
- For most of my life this was just a pain
- You guys have it easy, modern deep learning frameworks do this for you automatically
- This is a *game changer!* We are no longer afraid of large models
- We can differentiate through algorithms allowing us to train incredible sophisticated networks

Automatic Differentiation

- For gradient descent we need to compute the gradient
- For most of my life this was just a pain
- You guys have it easy, modern deep learning frameworks do this for you automatically
- **This is a *game changer!*** We are no longer afraid of large models
- We can differentiate through algorithms allowing us to train incredible sophisticated networks

Automatic Differentiation

- For gradient descent we need to compute the gradient
- For most of my life this was just a pain
- You guys have it easy, modern deep learning frameworks do this for you automatically
- This is a *game changer!* We are no longer afraid of large models
- We can differentiate through algorithms allowing us to train incredible sophisticated networks

Automatic Differentiation

- For gradient descent we need to compute the gradient
- For most of my life this was just a pain
- You guys have it easy, modern deep learning frameworks do this for you automatically
- This is a *game changer!* We are no longer afraid of large models
- We can differentiate through algorithms allowing us to train incredible sophisticated networks

Convergence

- Asymptotic convergence of gradient descent (and SGD) is slower than quasi-Newton methods
- But there are three reasons why we don't really care
 - ★ For complex models we spend a lot of time before we are close to a quadratic minima where asymptotic convergence kicks in
 - ★ These days we often use ReLUs (rectified linear units) which are non-analytic. The convergence we discussed depends on a Taylor expansion that assumes analyticity
 - ★ We want to minimise the generalisation error, but are only minimising a poor surrogate, namely the training error

Convergence

- Asymptotic convergence of gradient descent (and SGD) is slower than quasi-Newton methods
- But there are three reasons why we don't really care
 - ★ For complex models we spend a lot of time before we are close to a quadratic minima where asymptotic convergence kicks in
 - ★ These days we often use ReLUs (rectified linear units) which are non-analytic. The convergence we discussed depends on a Taylor expansion that assumes analyticity
 - ★ We want to minimise the generalisation error, but are only minimising a poor surrogate, namely the training error

Convergence

- Asymptotic convergence of gradient descent (and SGD) is slower than quasi-Newton methods
- But there are three reasons why we don't really care
 - ★ For complex models we spend a lot of time before we are close to a quadratic minima where asymptotic convergence kicks in
 - ★ These days we often use ReLUs (rectified linear units) which are non-analytic. The convergence we discussed depends on a Taylor expansion that assumes analyticity
 - ★ We want to minimise the generalisation error, but are only minimising a poor surrogate, namely the training error

Convergence

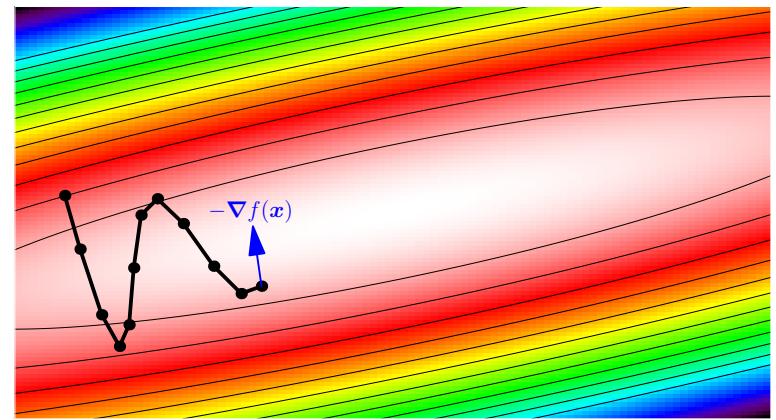
- Asymptotic convergence of gradient descent (and SGD) is slower than quasi-Newton methods
- But there are three reasons why we don't really care
 - ★ For complex models we spend a lot of time before we are close to a quadratic minima where asymptotic convergence kicks in
 - ★ These days we often use ReLUs (rectified linear units) which are non-analytic. The convergence we discussed depends on a Taylor expansion that assumes analyticity
 - ★ We want to minimise the generalisation error, but are only minimising a poor surrogate, namely the training error

Convergence

- Asymptotic convergence of gradient descent (and SGD) is slower than quasi-Newton methods
- But there are three reasons why we don't really care
 - ★ For complex models we spend a lot of time before we are close to a quadratic minima where asymptotic convergence kicks in
 - ★ These days we often use ReLUs (rectified linear units) which are non-analytic. The convergence we discussed depends on a Taylor expansion that assumes analyticity
 - ★ We want to minimise the generalisation error, but are only minimising a poor surrogate, namely the training error

Outline

1. SGD
2. Momentum
3. Loss Landscapes

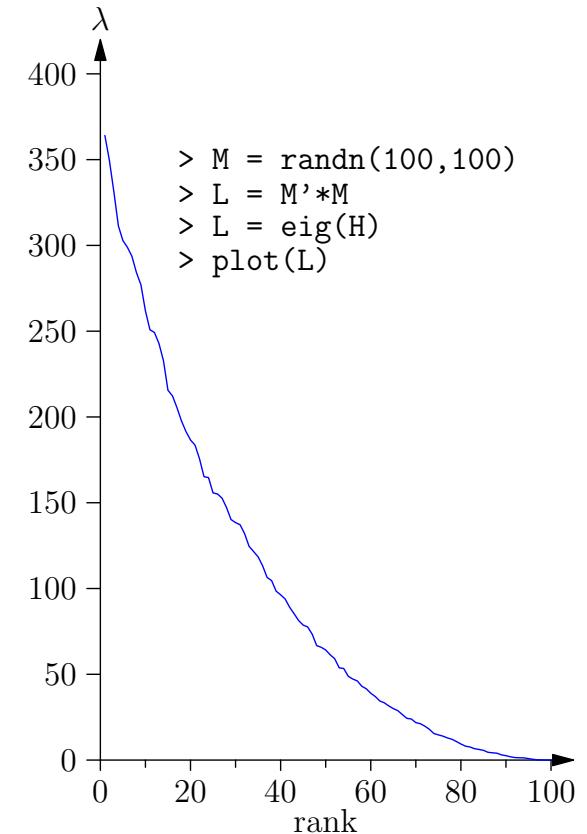


Step Size

- The optimal step size depends on the gradient and the curvature (second derivative)
- For a quadratic minima the minimum is given by

$$x^* = x - \frac{f'(x)}{f''(x)}, \quad x^* = x - H^{-1} \nabla f(x)$$

- In high dimensions the Hessian H will have a spectrum of eigenvalues
- This means there are different scales

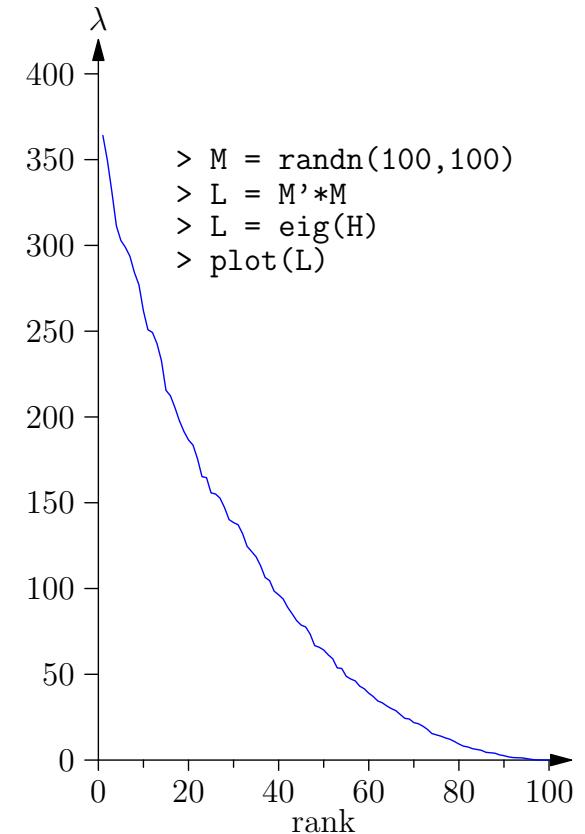


Step Size

- The optimal step size depends on the gradient and the curvature (second derivative)
- For a quadratic minima the minimum is given by

$$x^* = x - \frac{f'(x)}{f''(x)}, \quad x^* = x - H^{-1} \nabla f(x)$$

- In high dimensions the Hessian H will have a spectrum of eigenvalues
- This means there are different scales

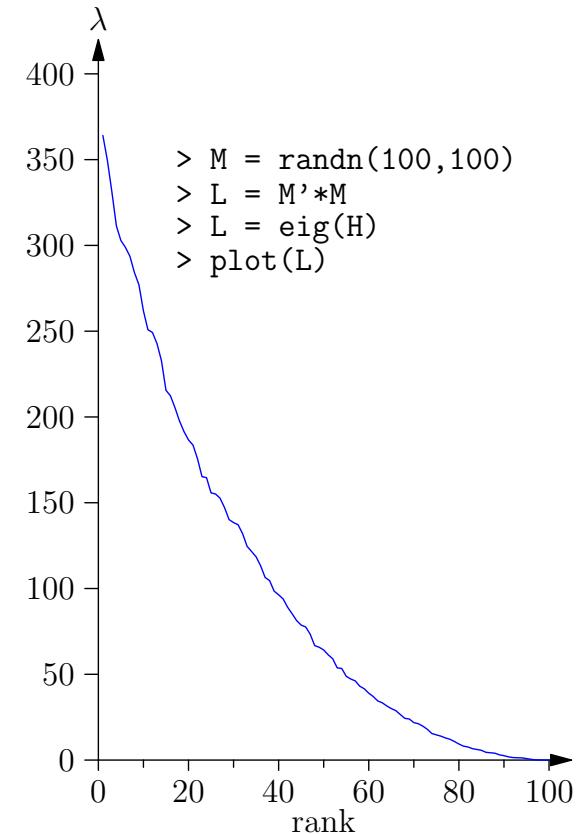


Step Size

- The optimal step size depends on the gradient and the curvature (second derivative)
- For a quadratic minima the minimum is given by

$$x^* = x - \frac{f'(x)}{f''(x)}, \quad x^* = x - H^{-1} \nabla f(x)$$

- In high dimensions the Hessian H will have a spectrum of eigenvalues
- This means there are different scales

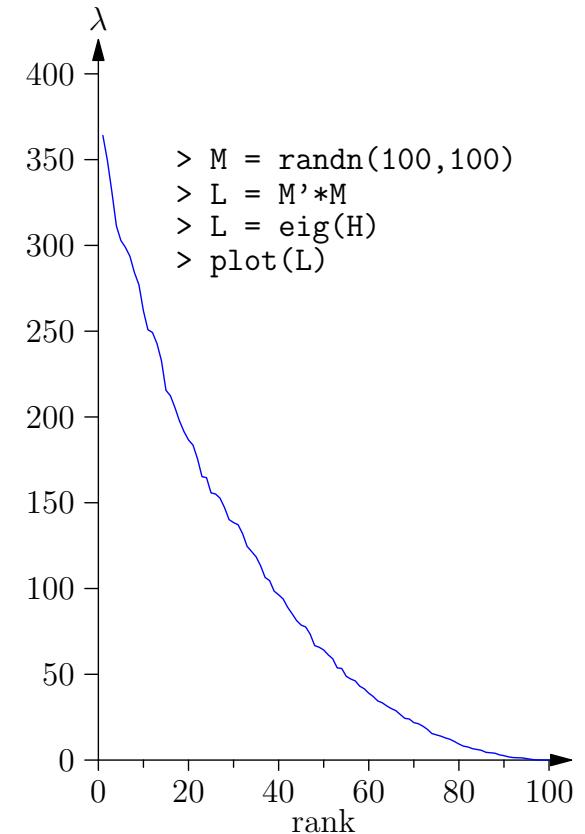


Step Size

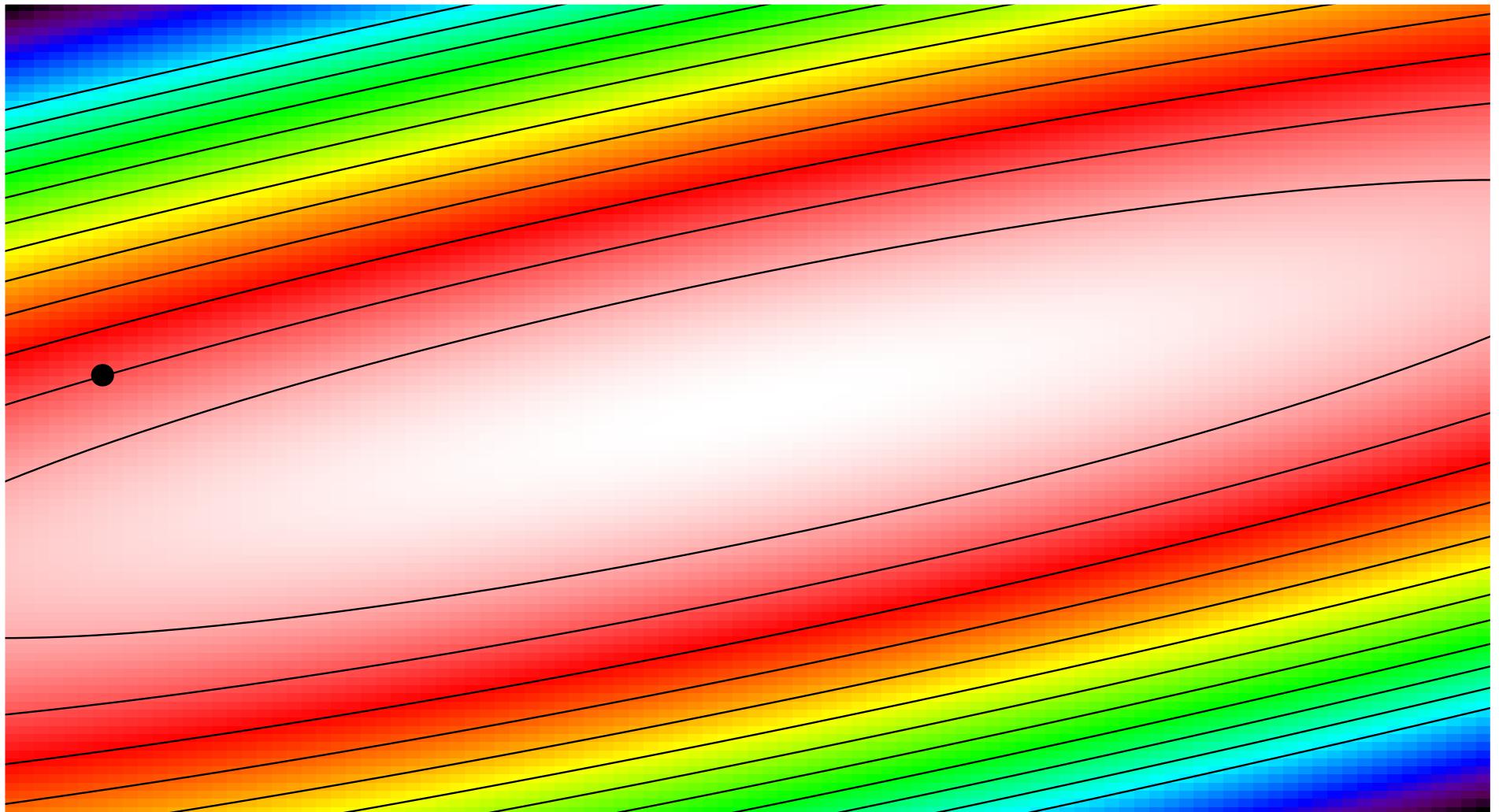
- The optimal step size depends on the gradient and the curvature (second derivative)
- For a quadratic minima the minimum is given by

$$x^* = x - \frac{f'(x)}{f''(x)}, \quad x^* = x - H^{-1} \nabla f(x)$$

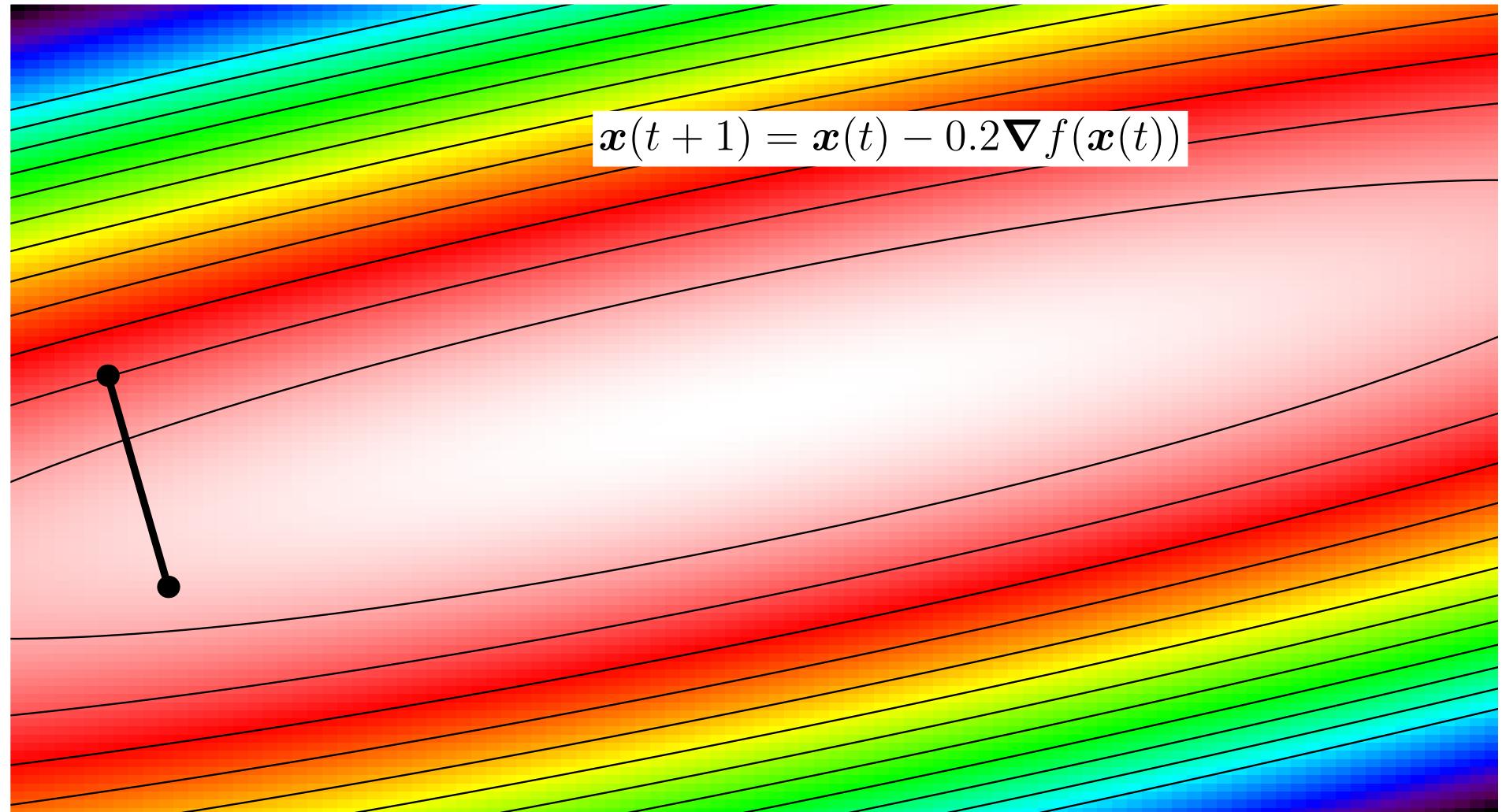
- In high dimensions the Hessian H will have a spectrum of eigenvalues
- This means there are different scales



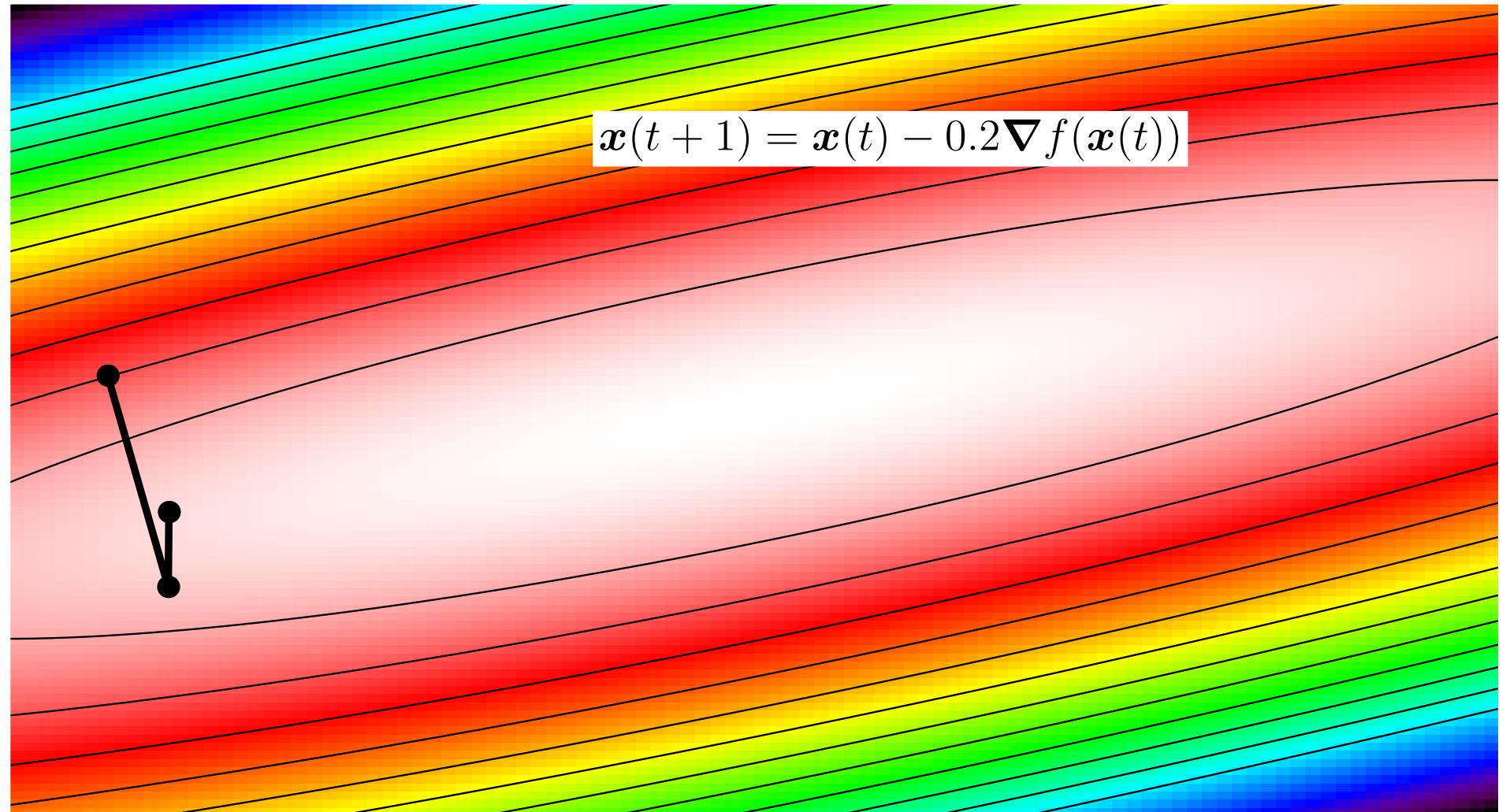
Gradient Descent



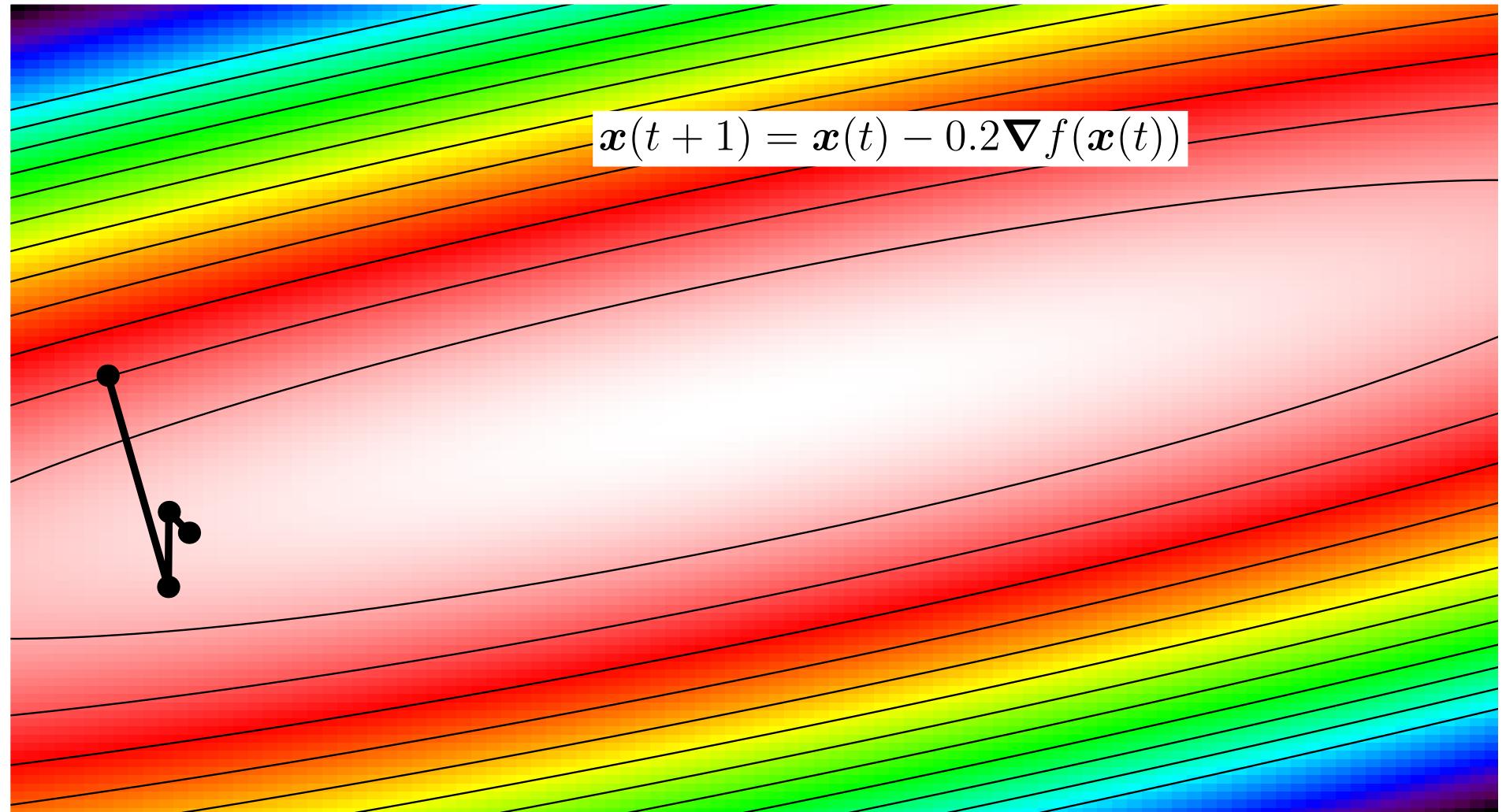
Gradient Descent



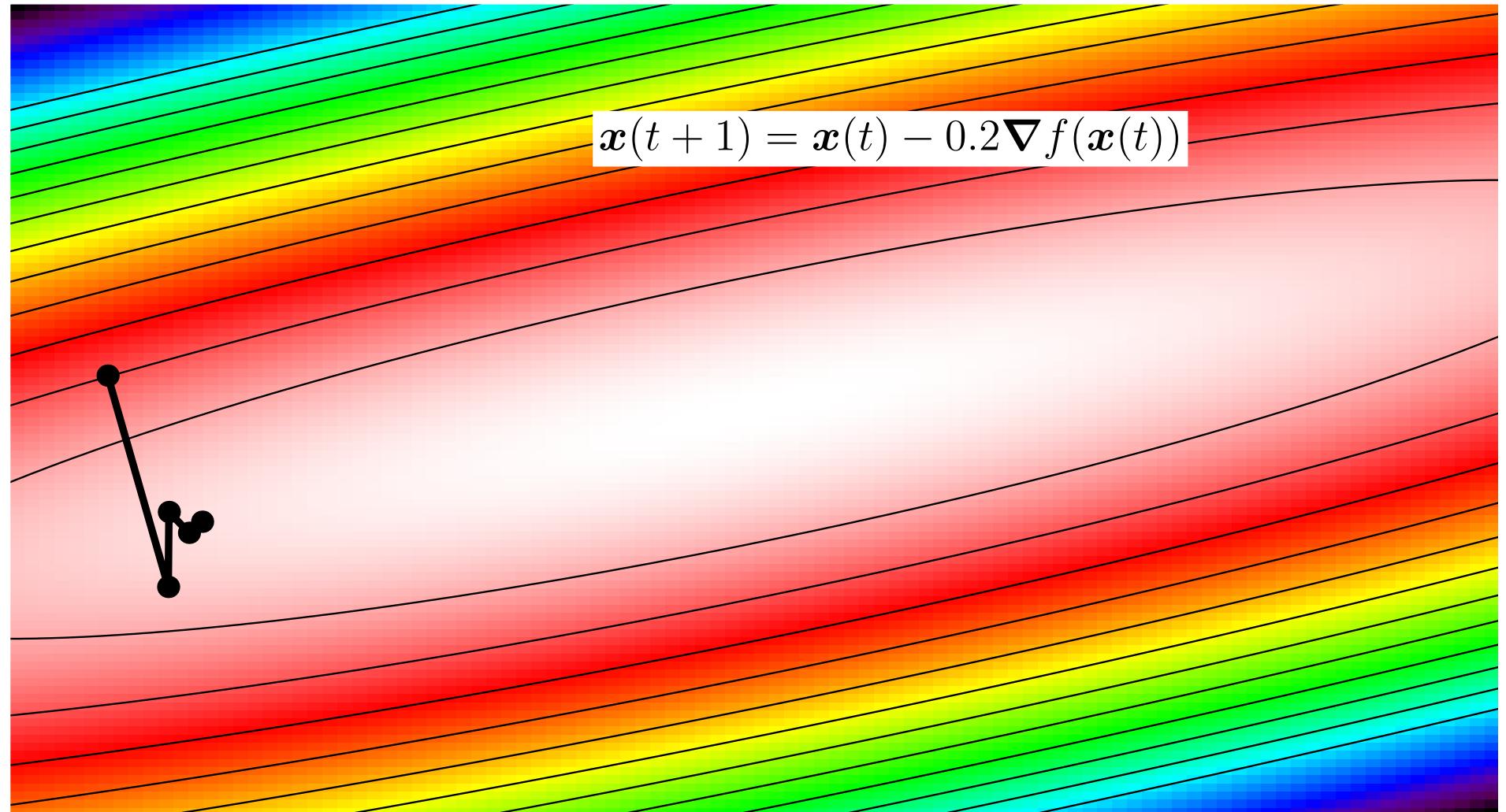
Gradient Descent



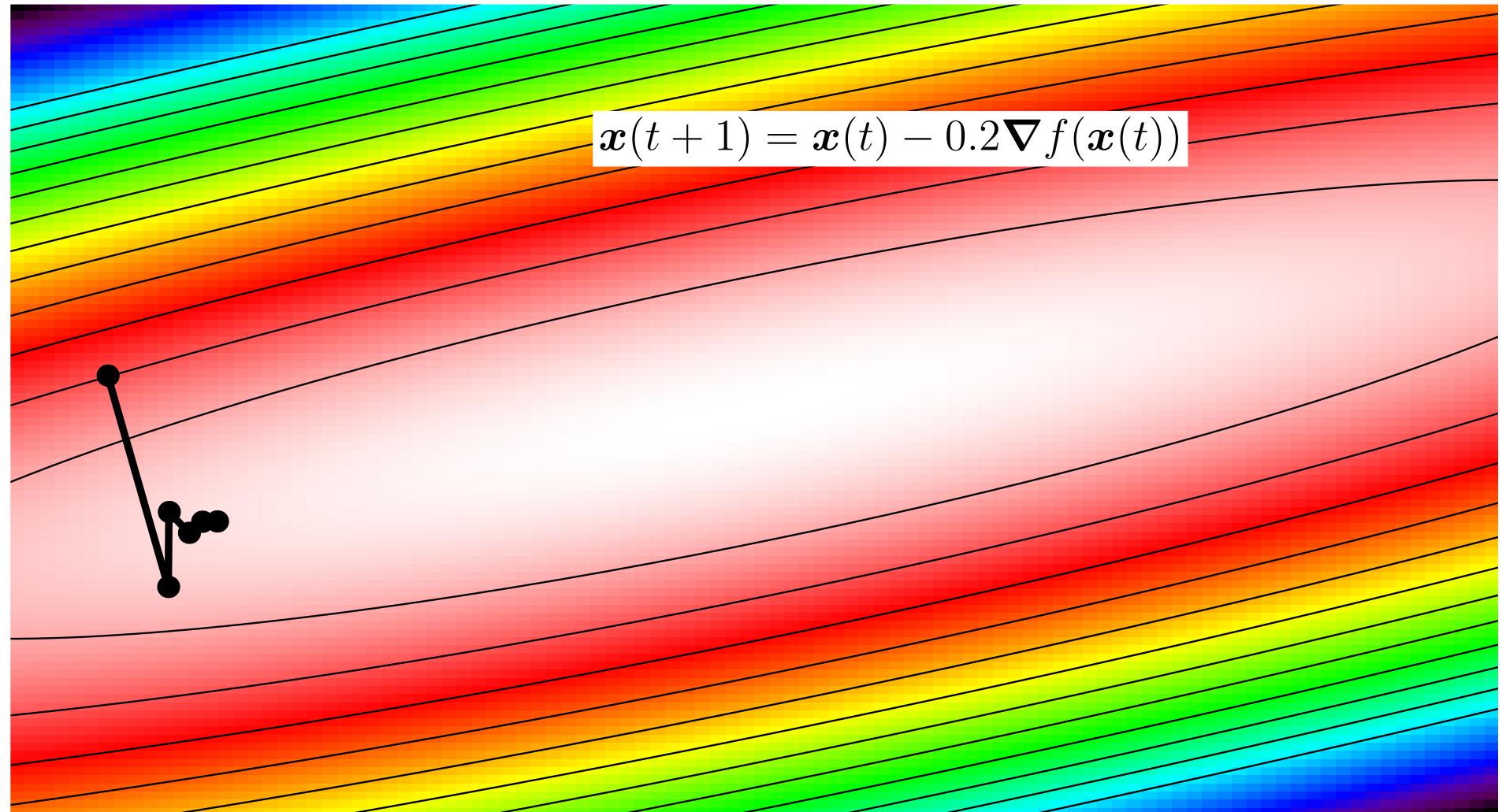
Gradient Descent



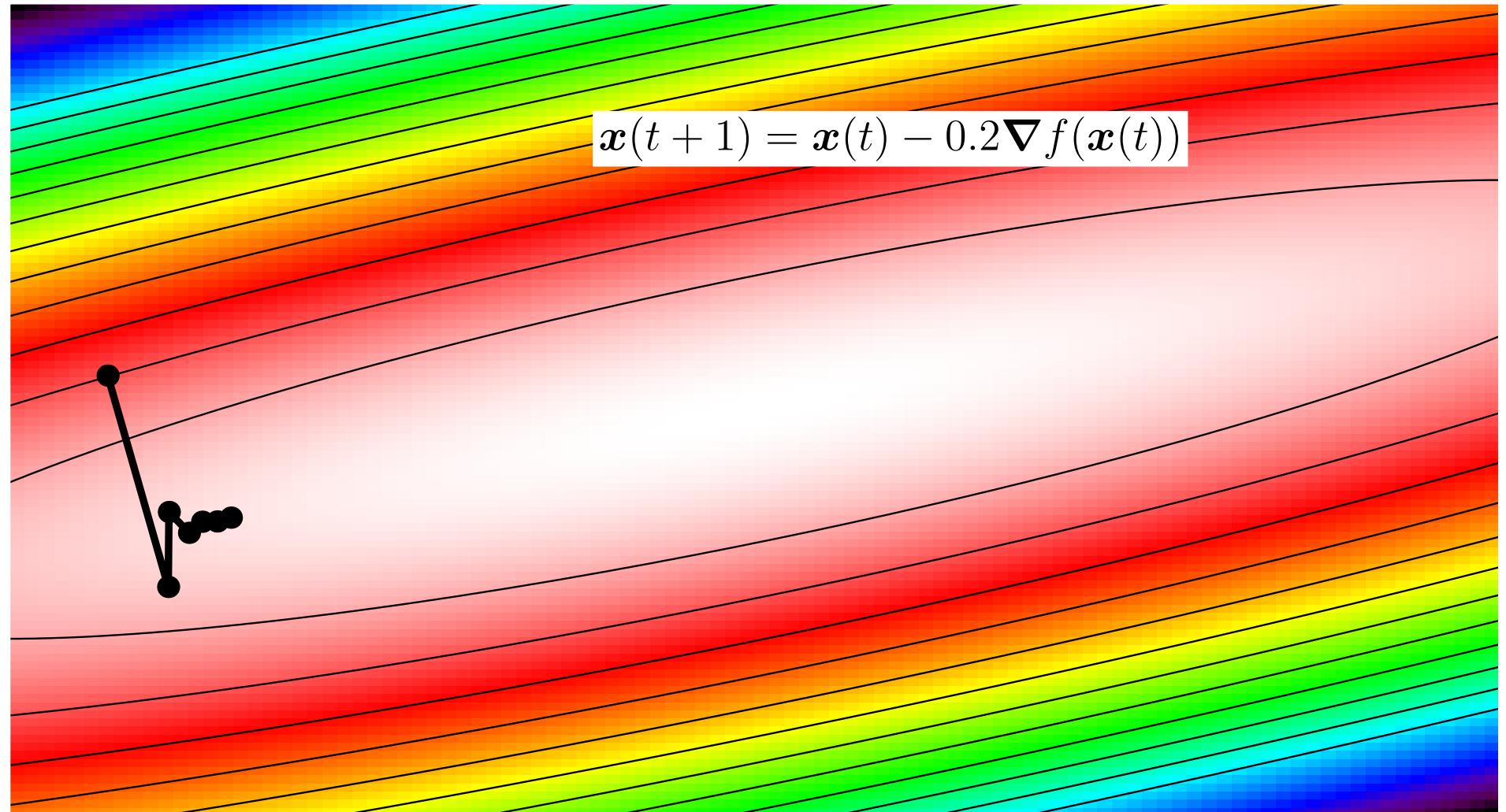
Gradient Descent



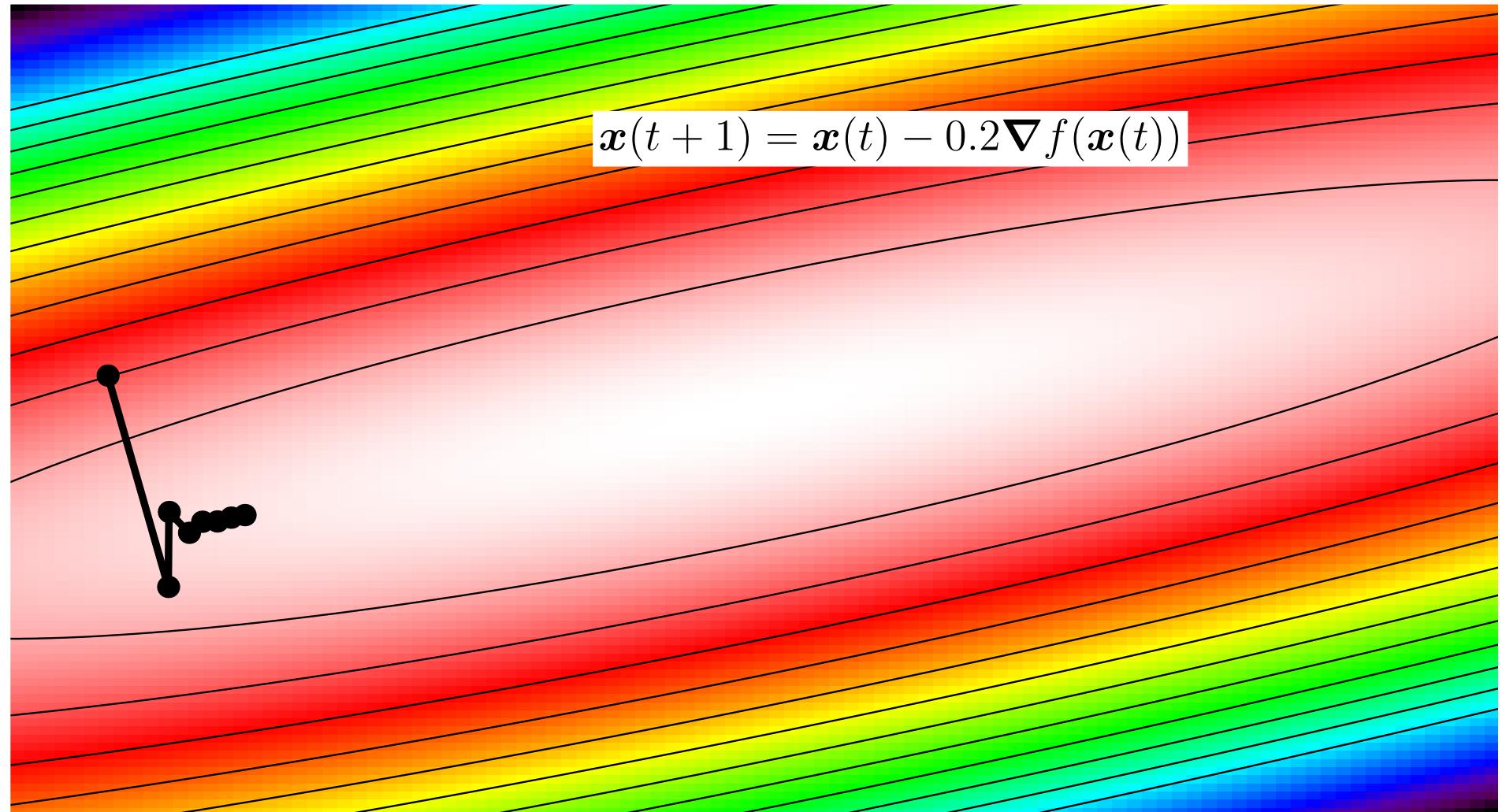
Gradient Descent



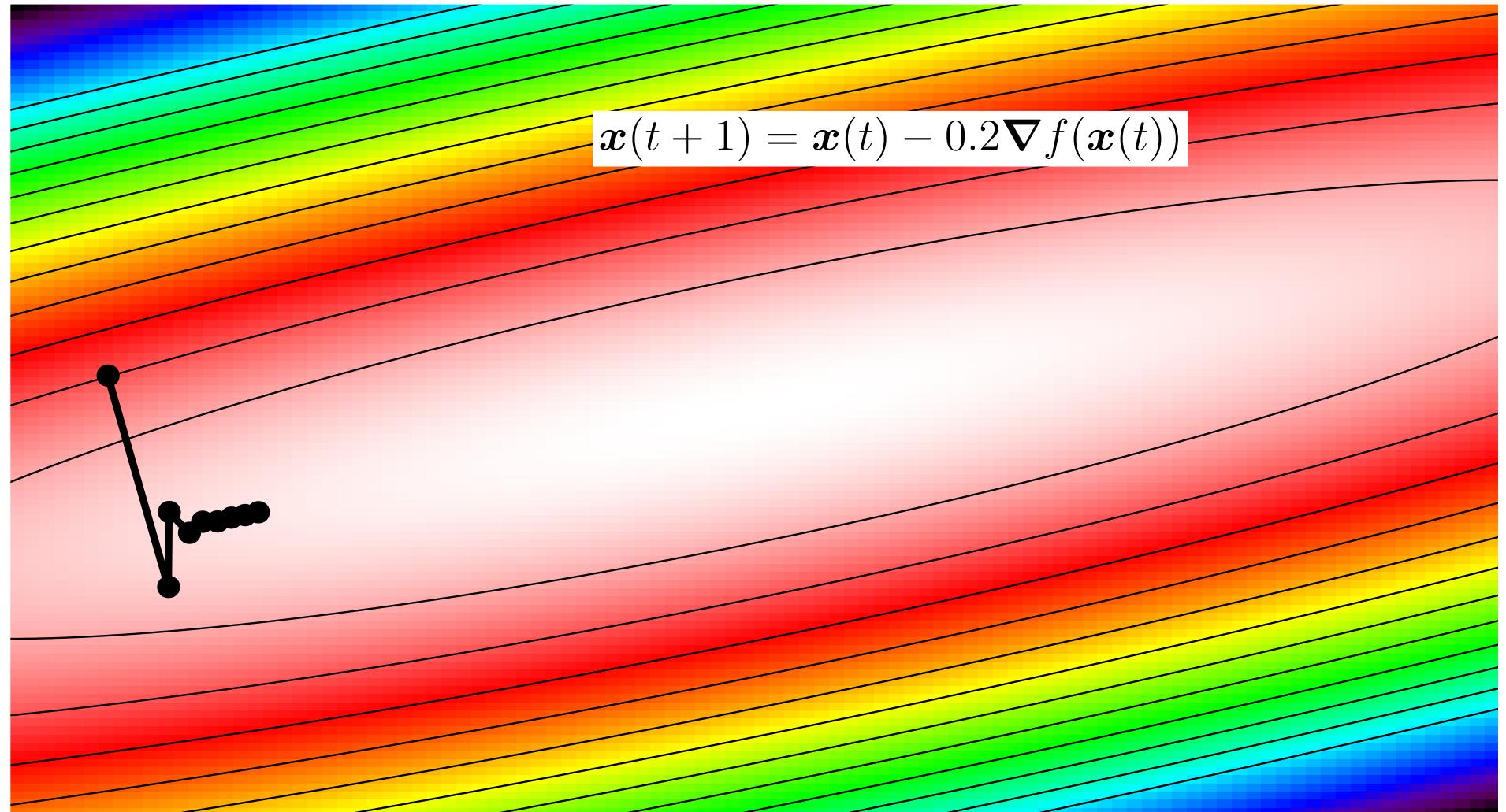
Gradient Descent



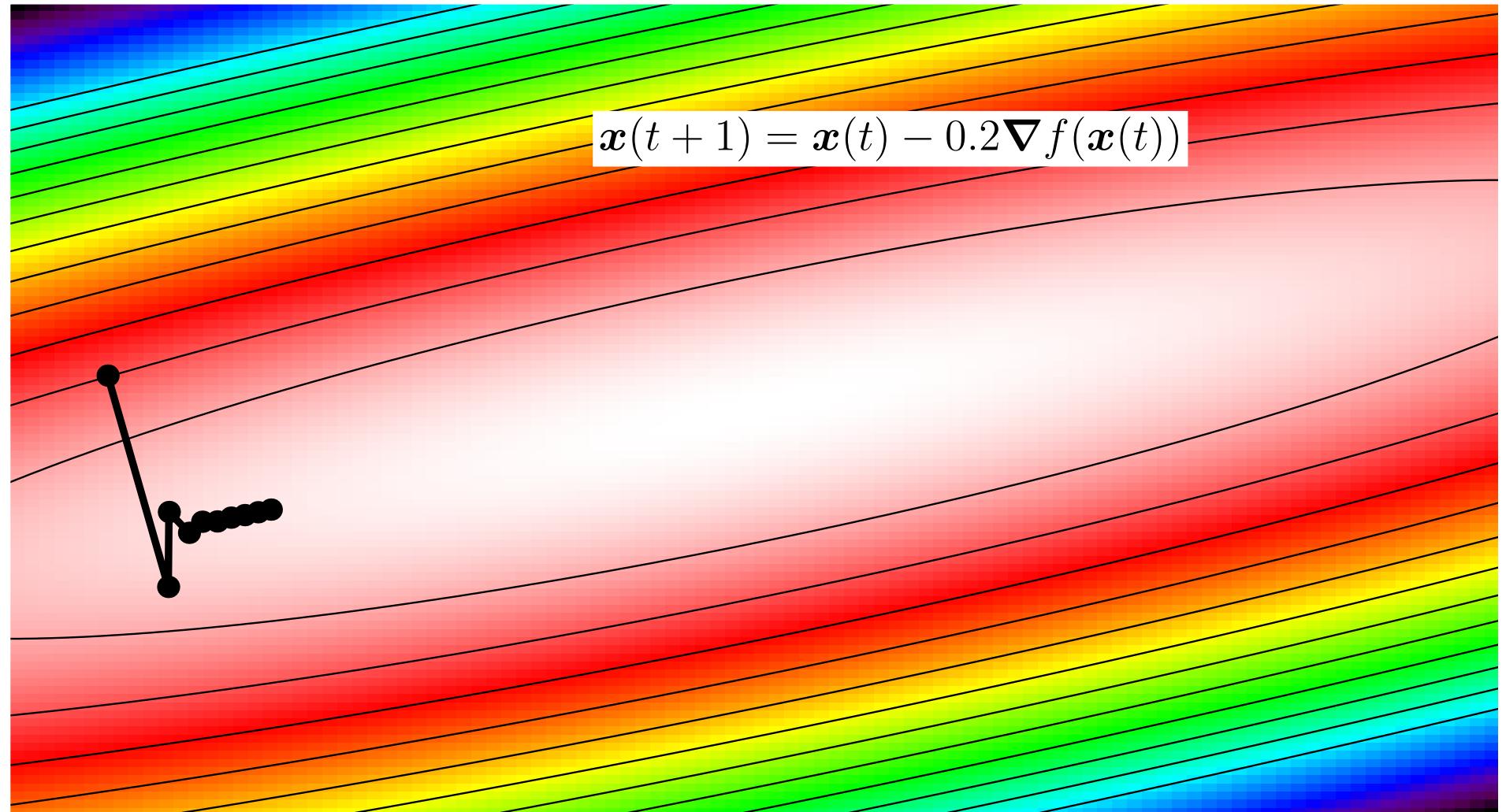
Gradient Descent



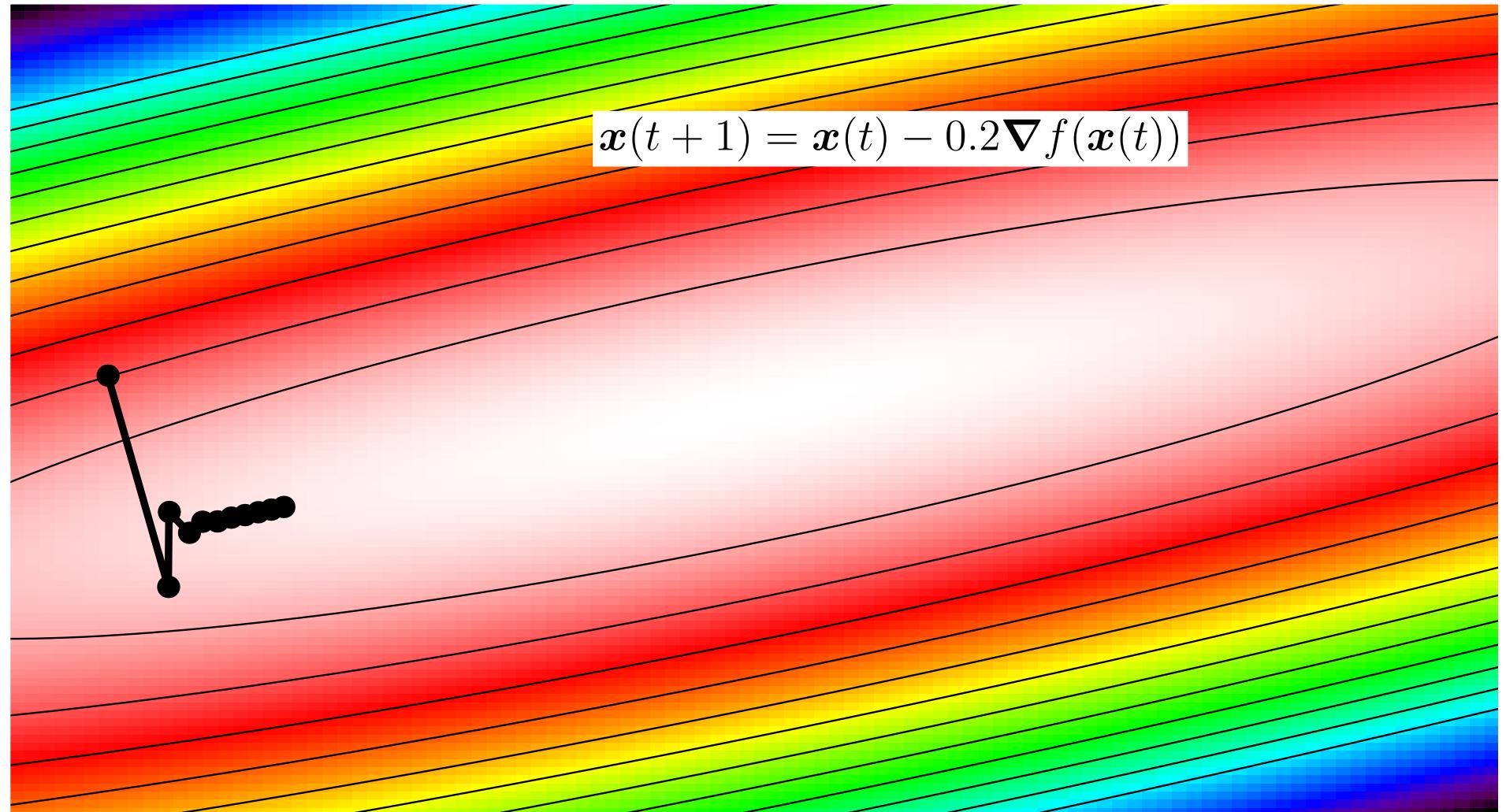
Gradient Descent



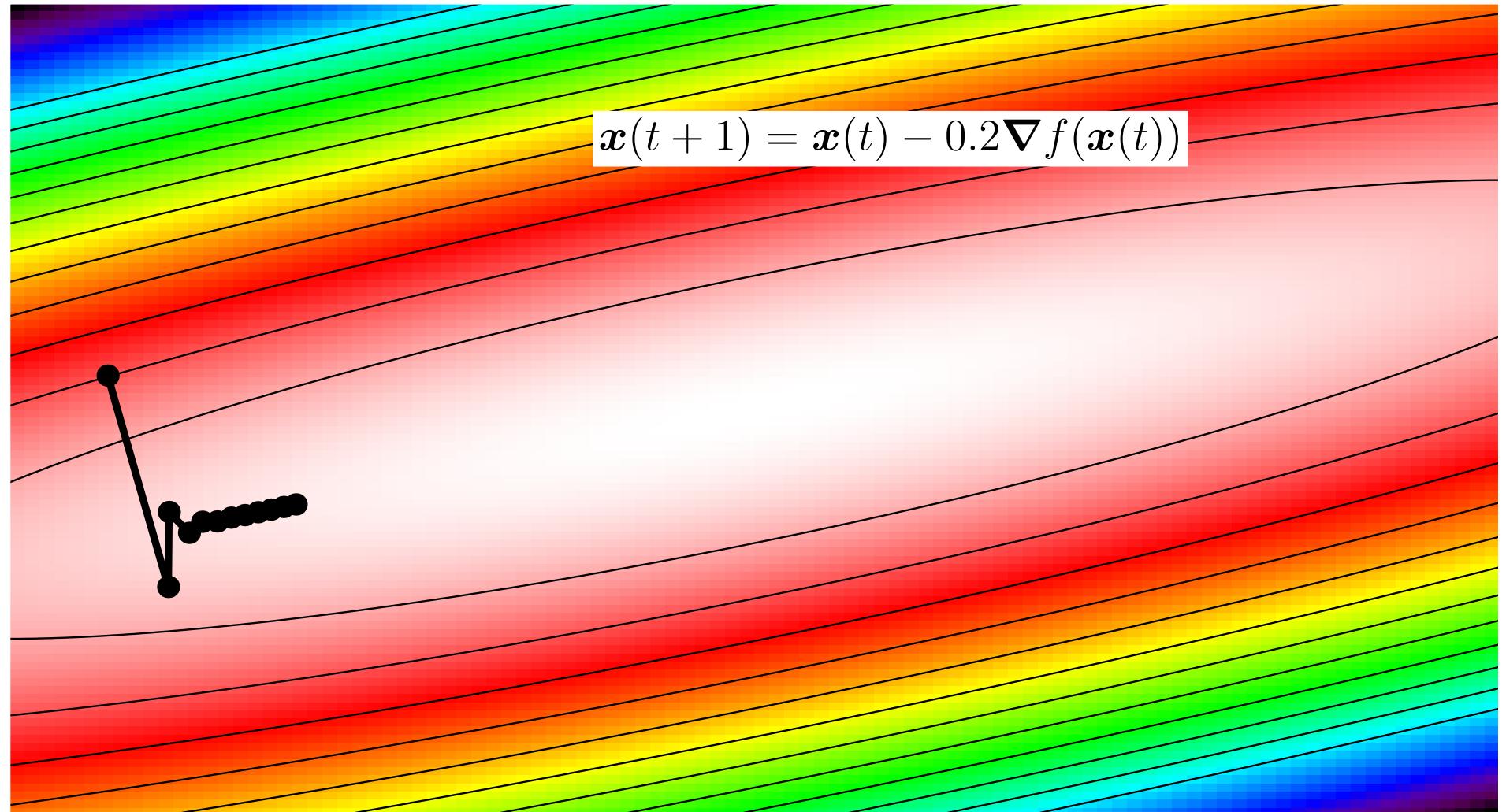
Gradient Descent



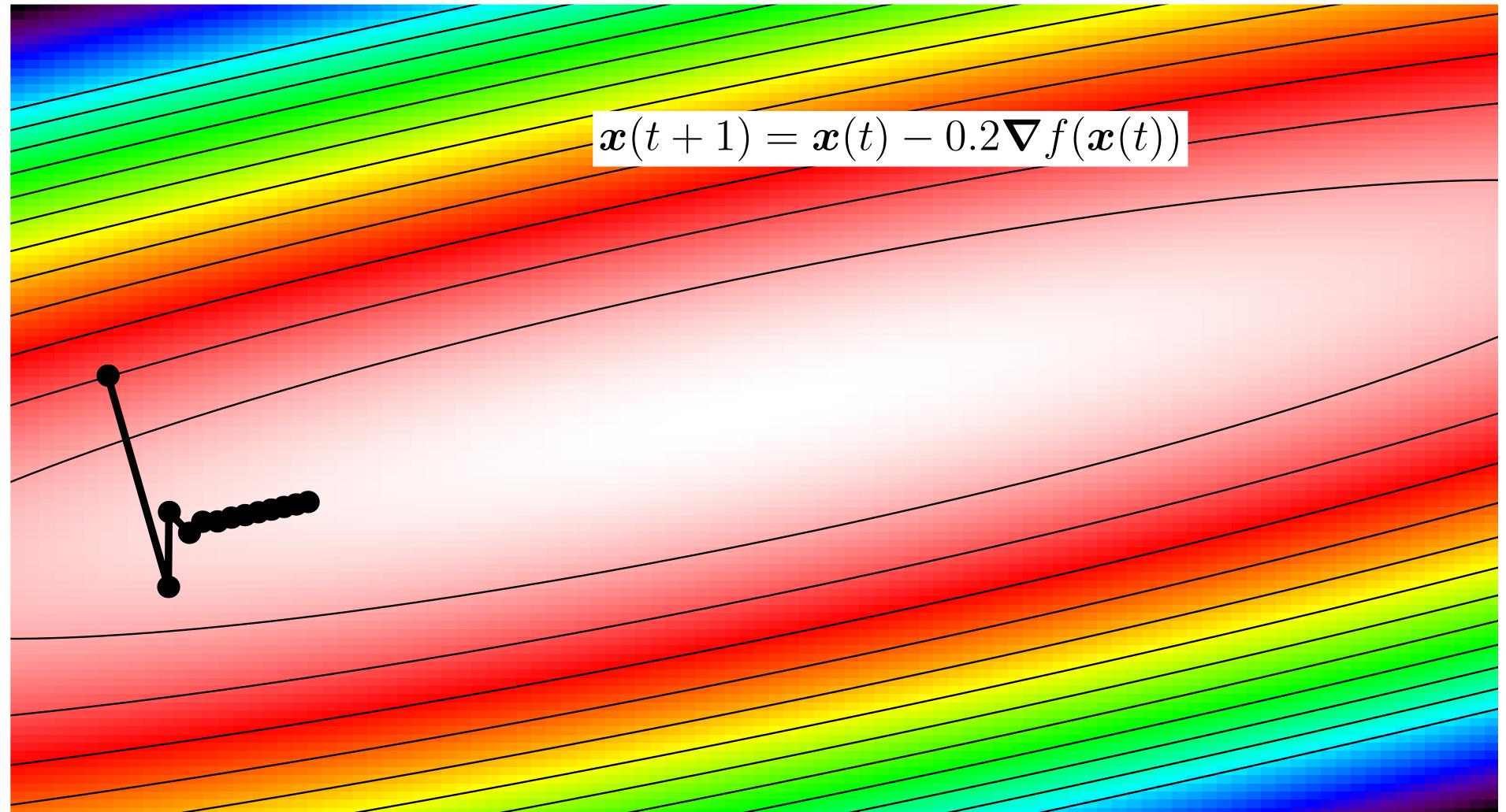
Gradient Descent



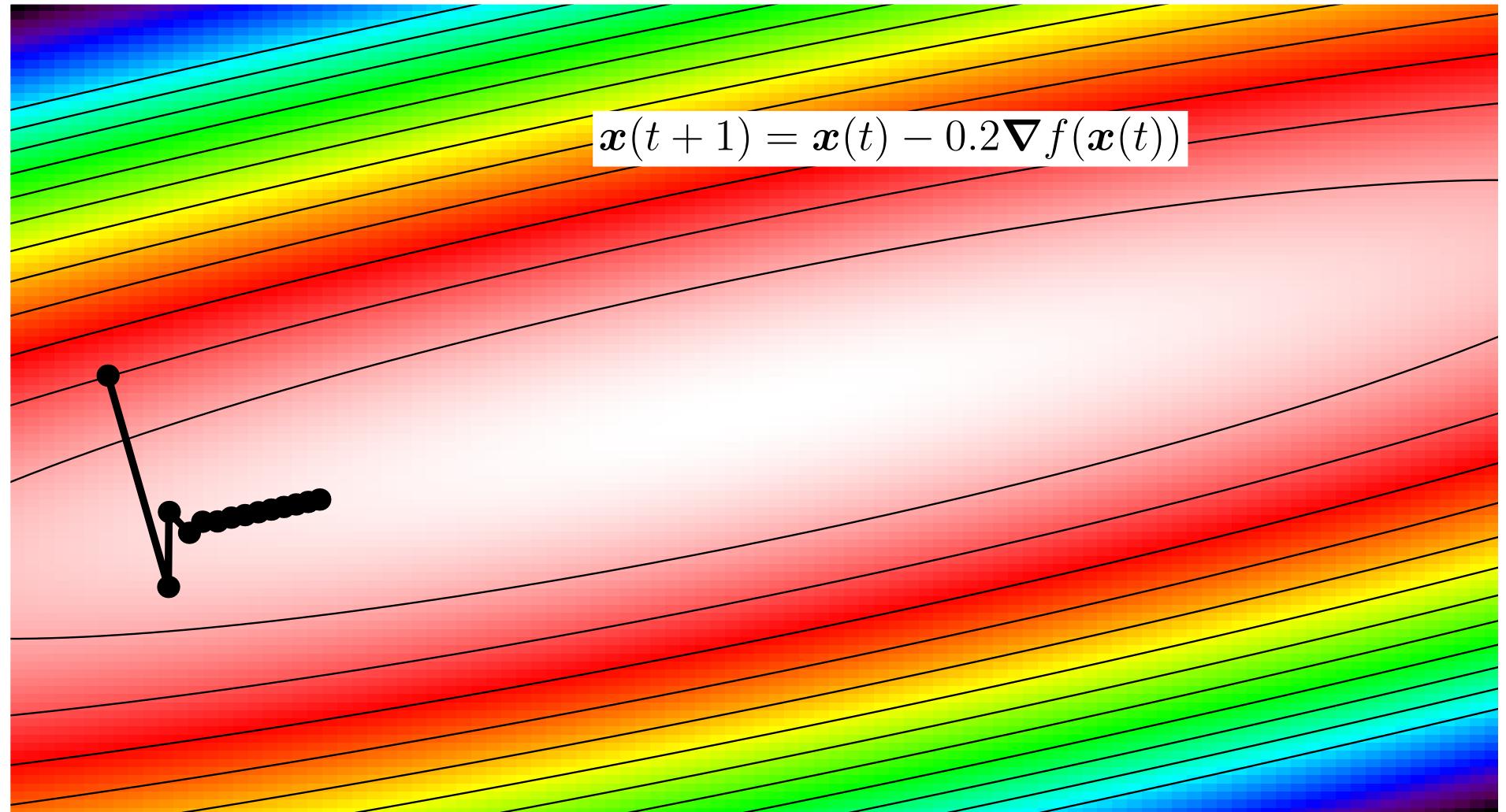
Gradient Descent



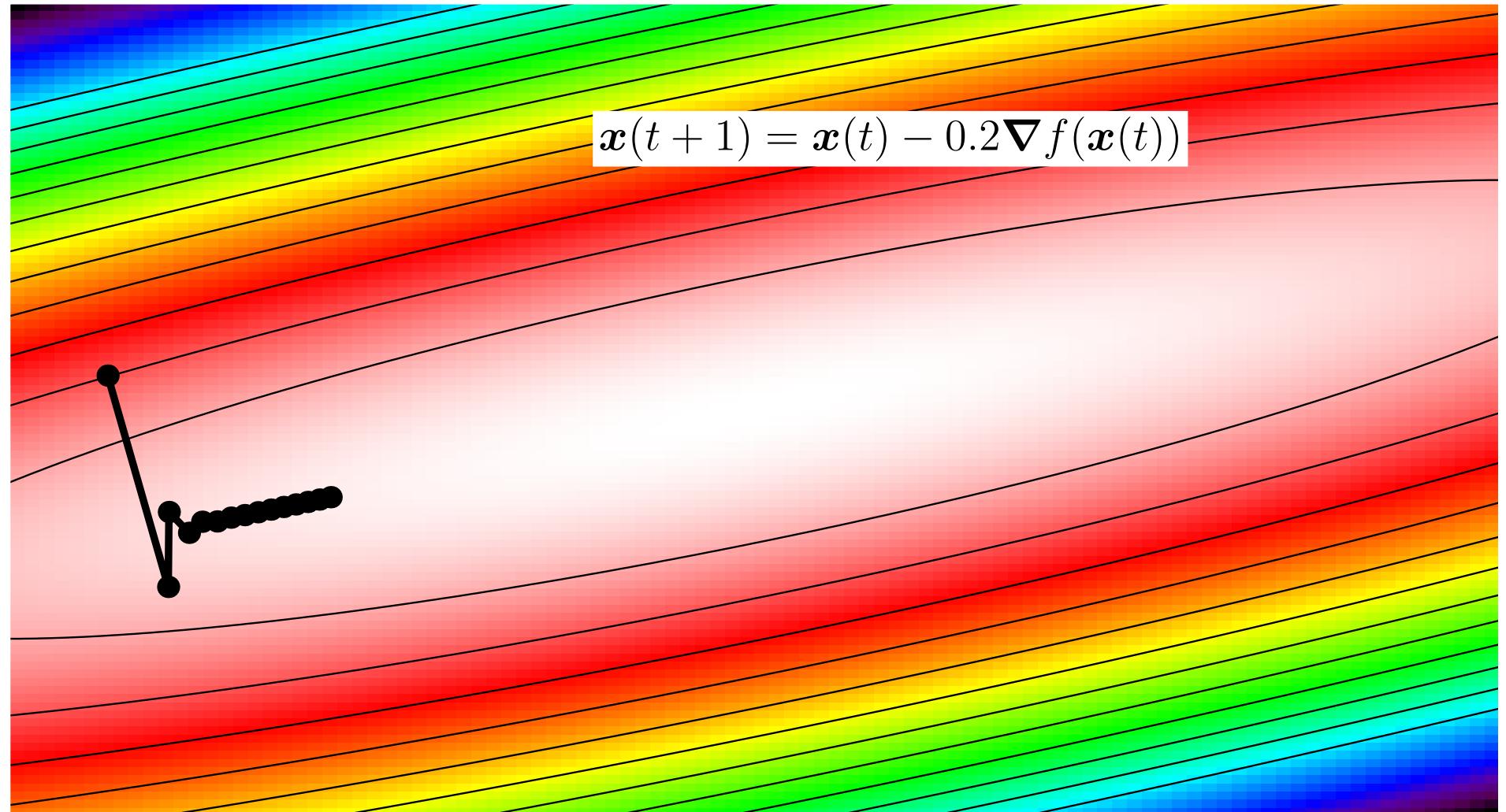
Gradient Descent



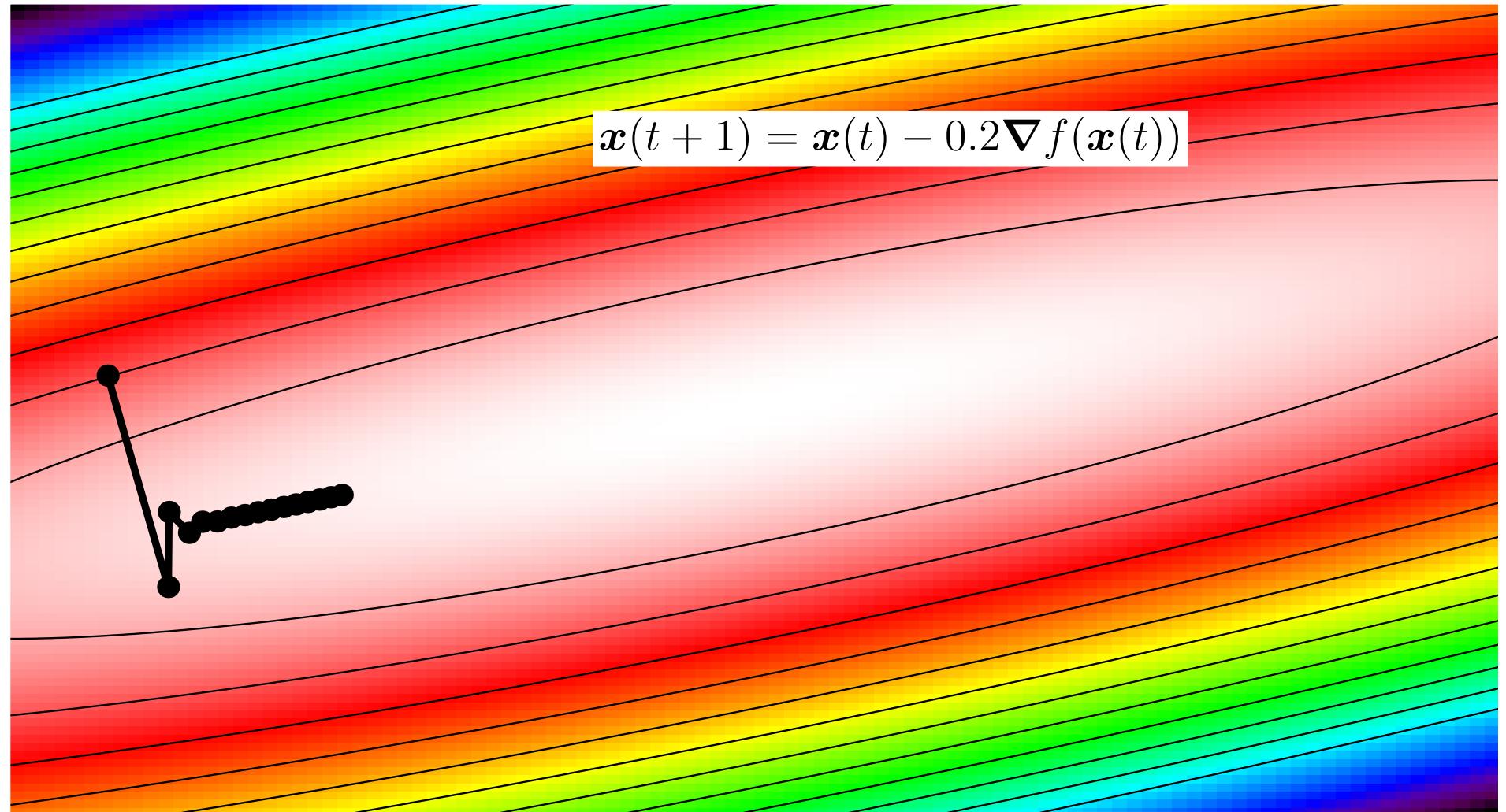
Gradient Descent



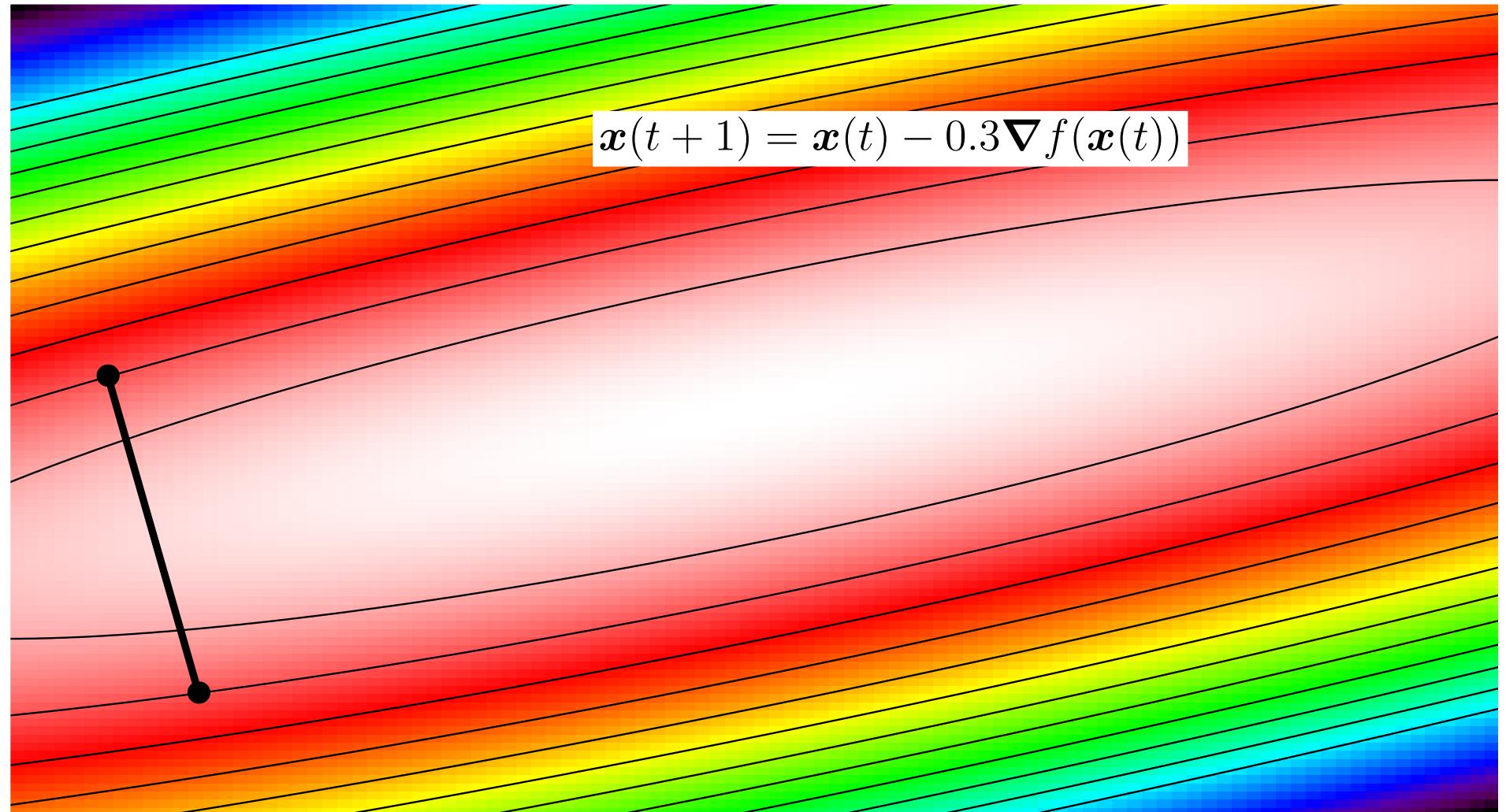
Gradient Descent



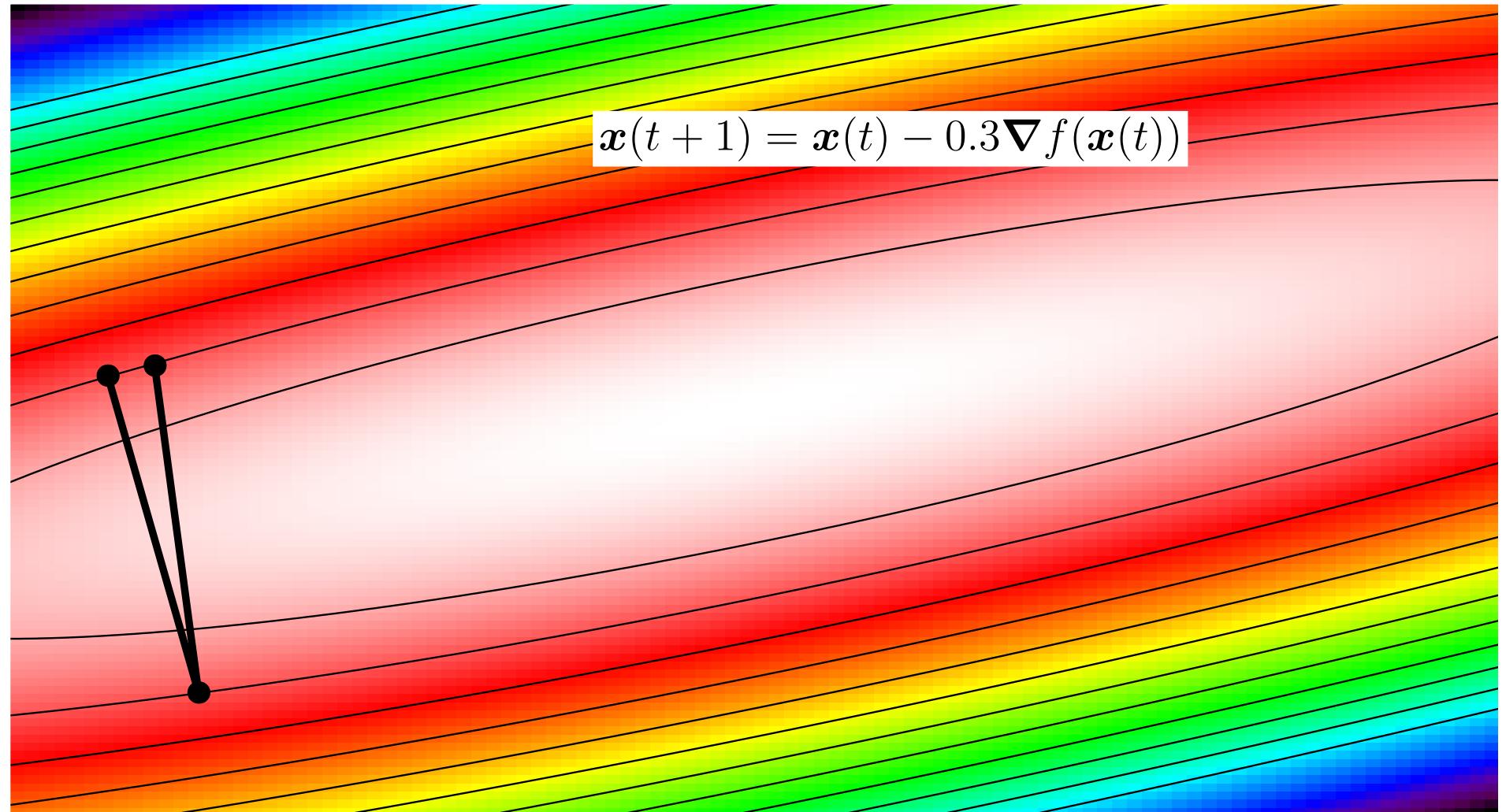
Gradient Descent



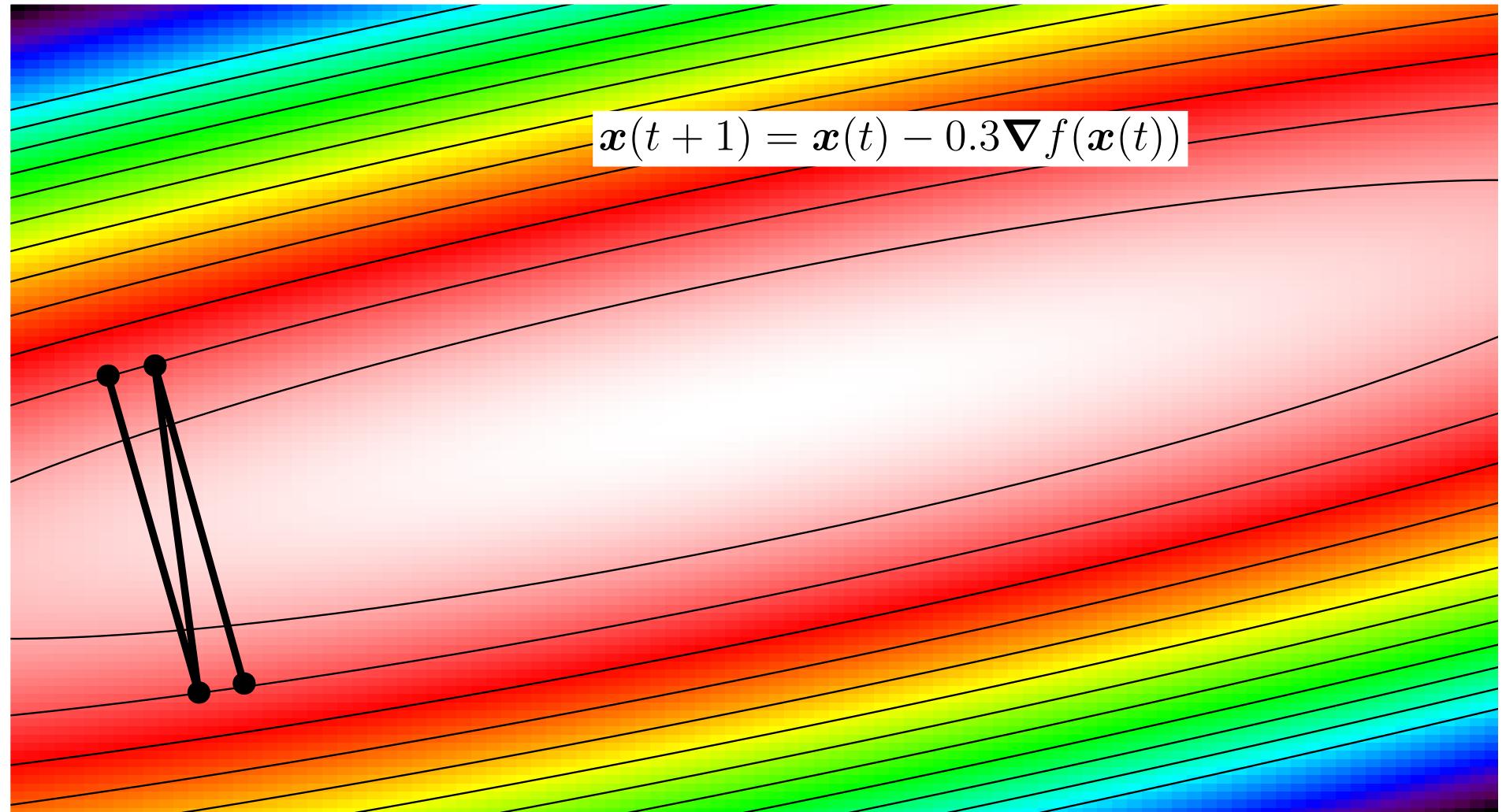
Gradient Descent



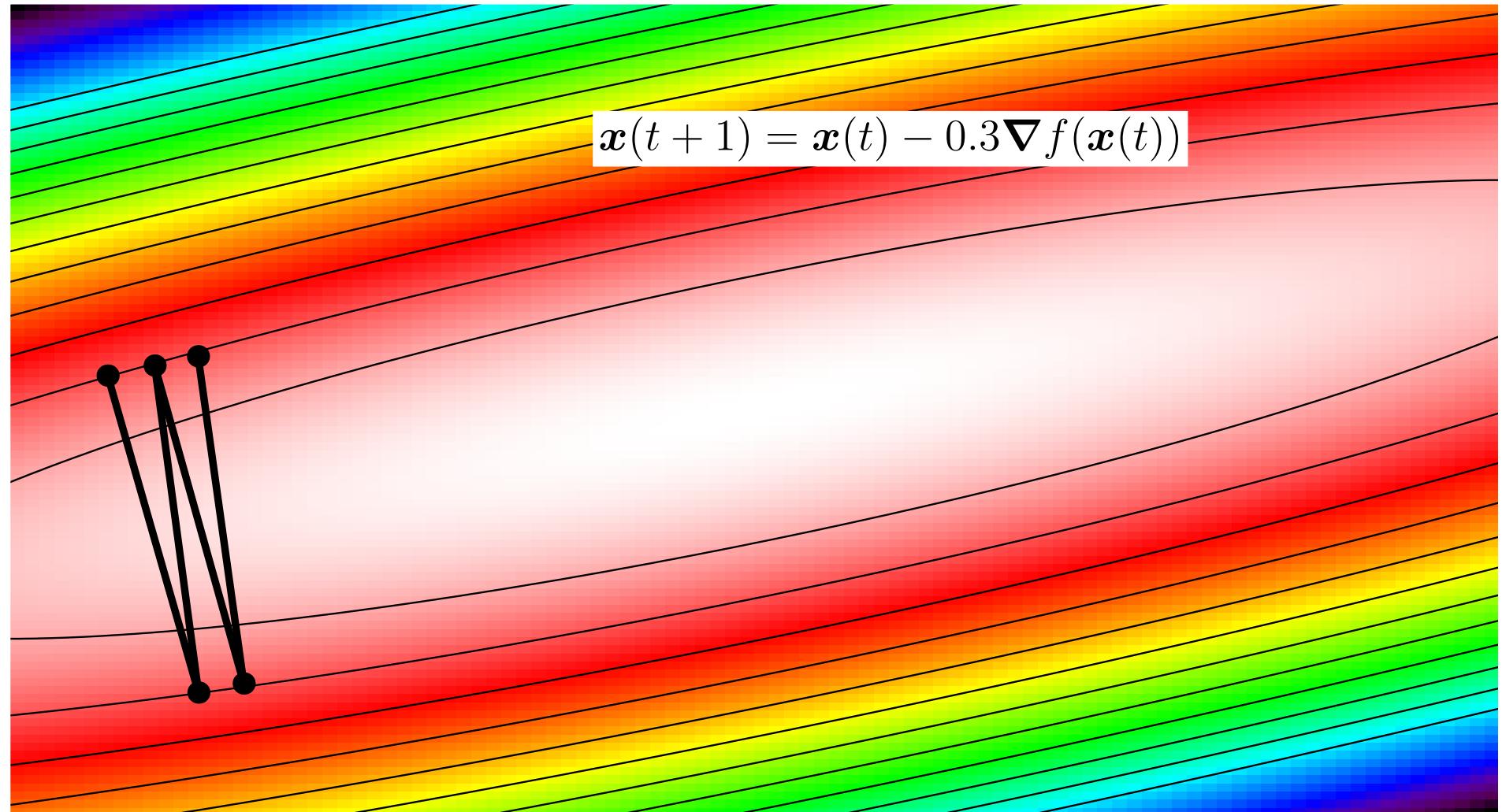
Gradient Descent



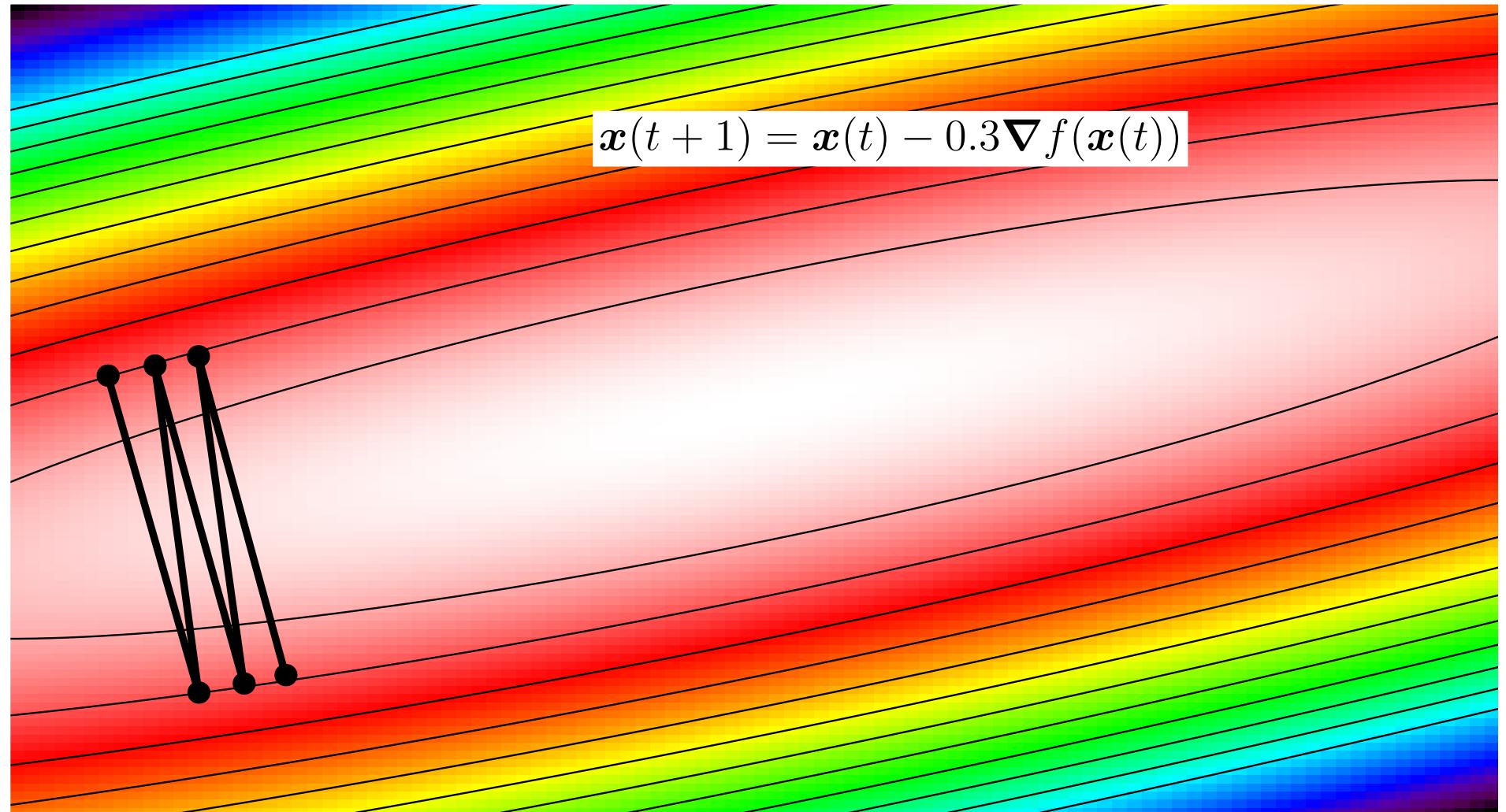
Gradient Descent



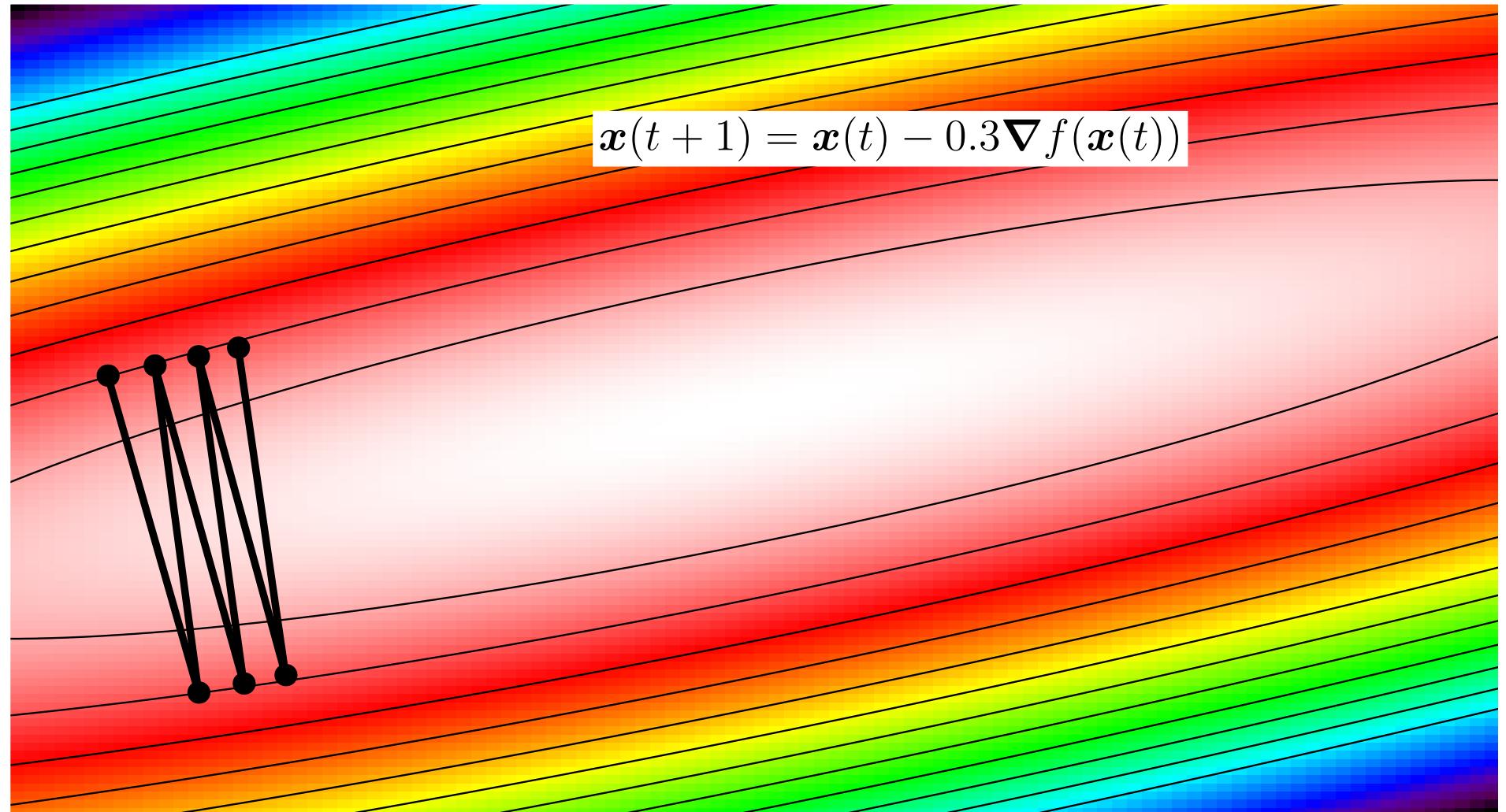
Gradient Descent



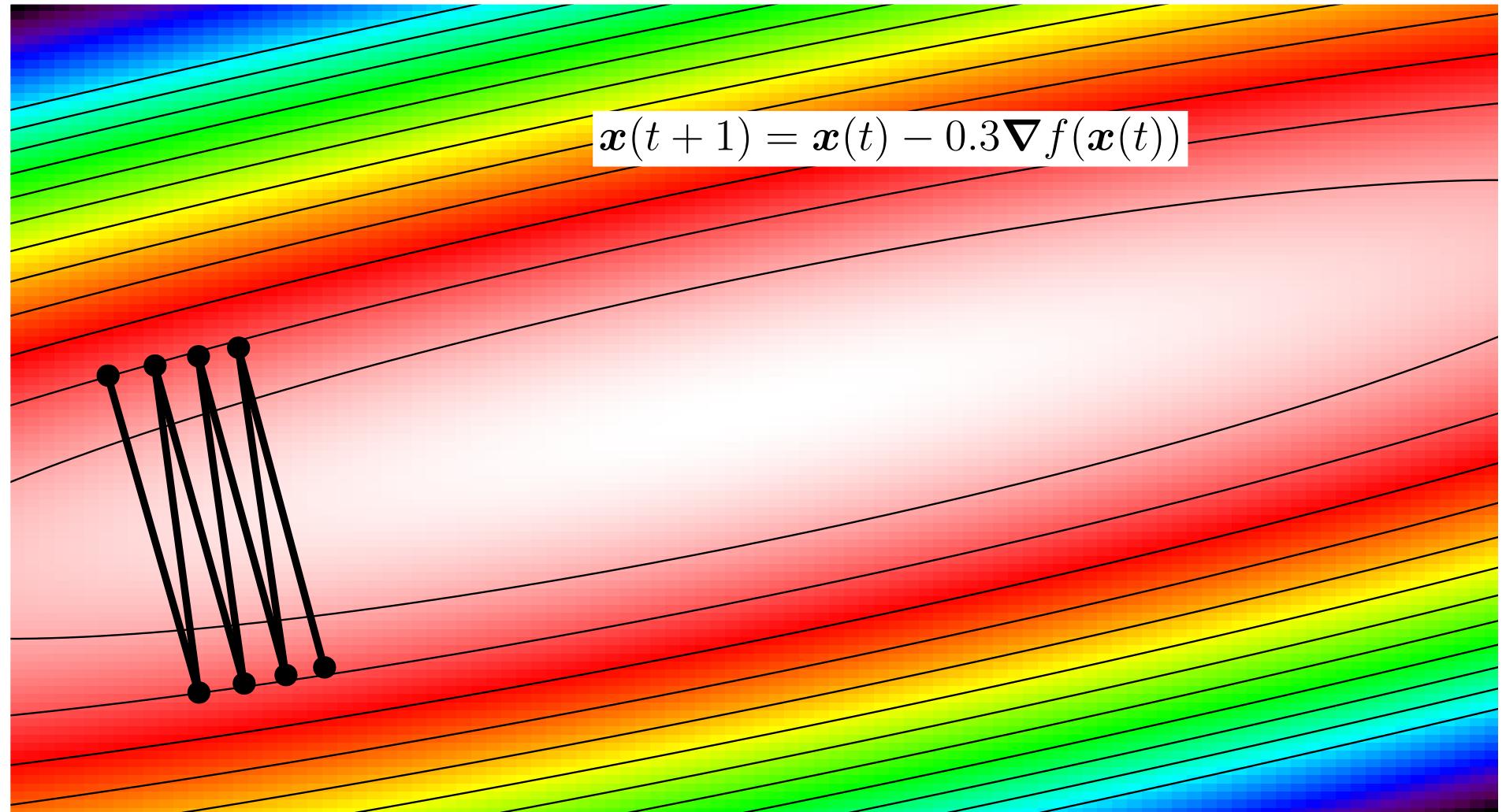
Gradient Descent



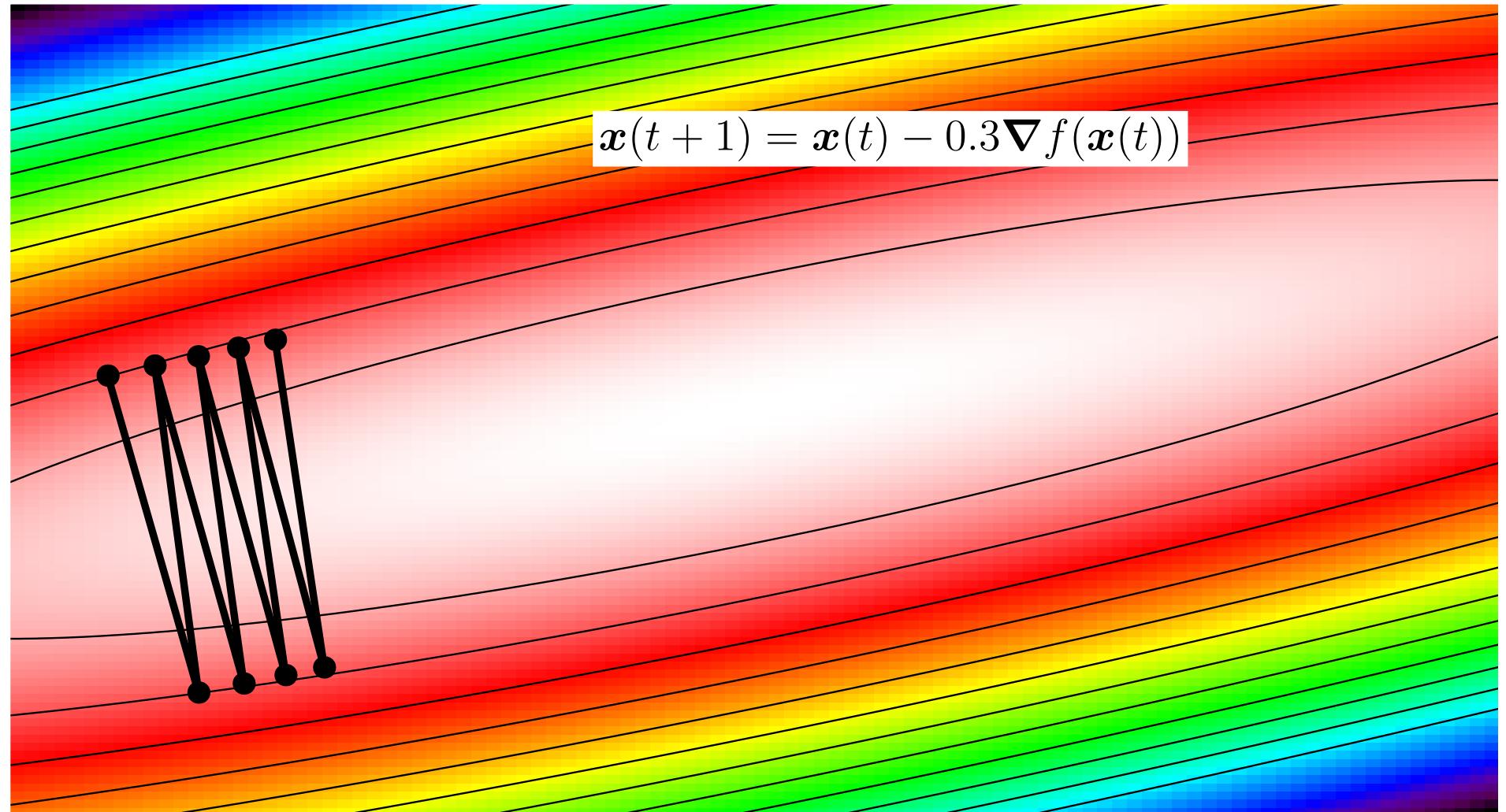
Gradient Descent



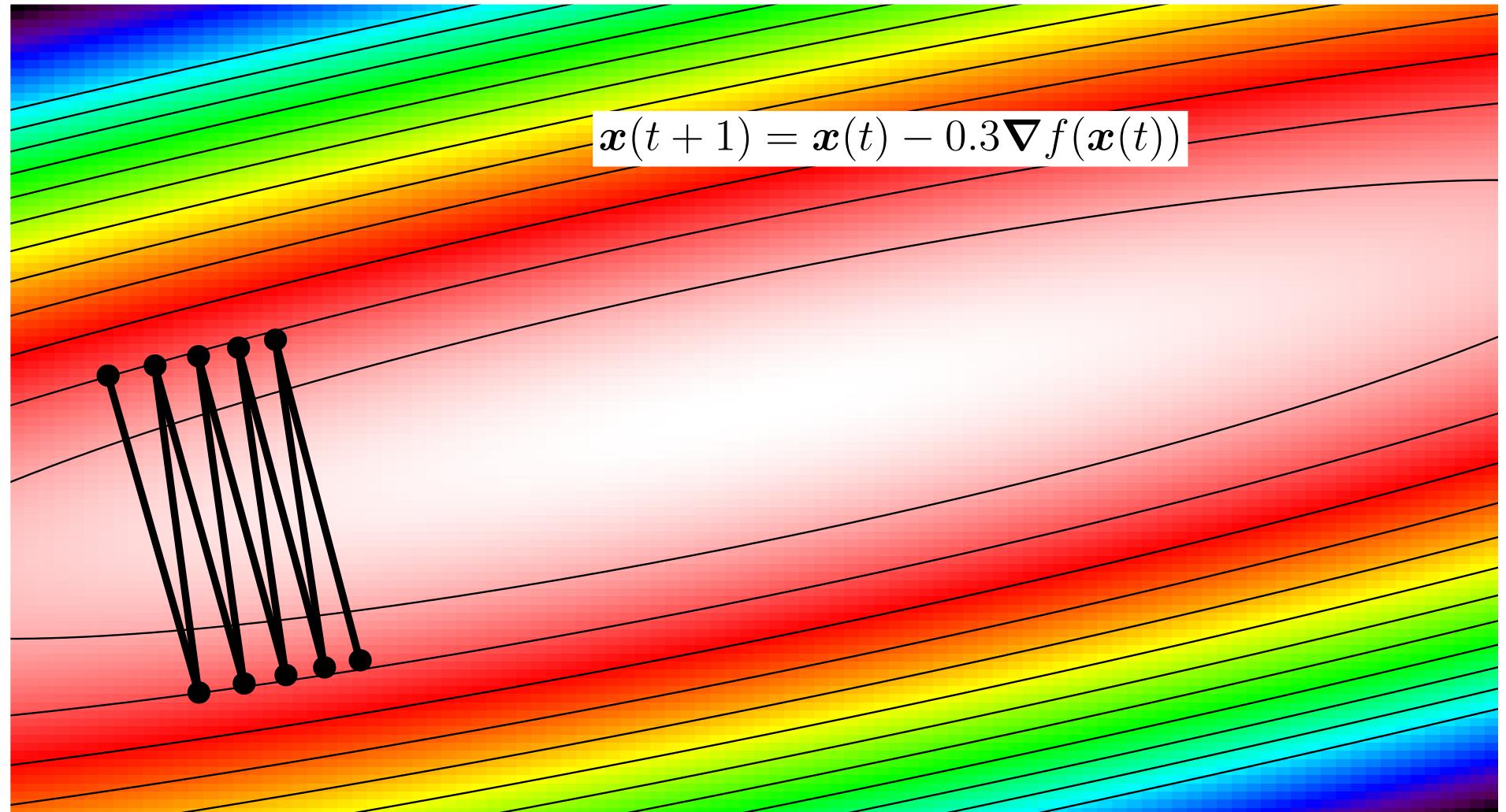
Gradient Descent



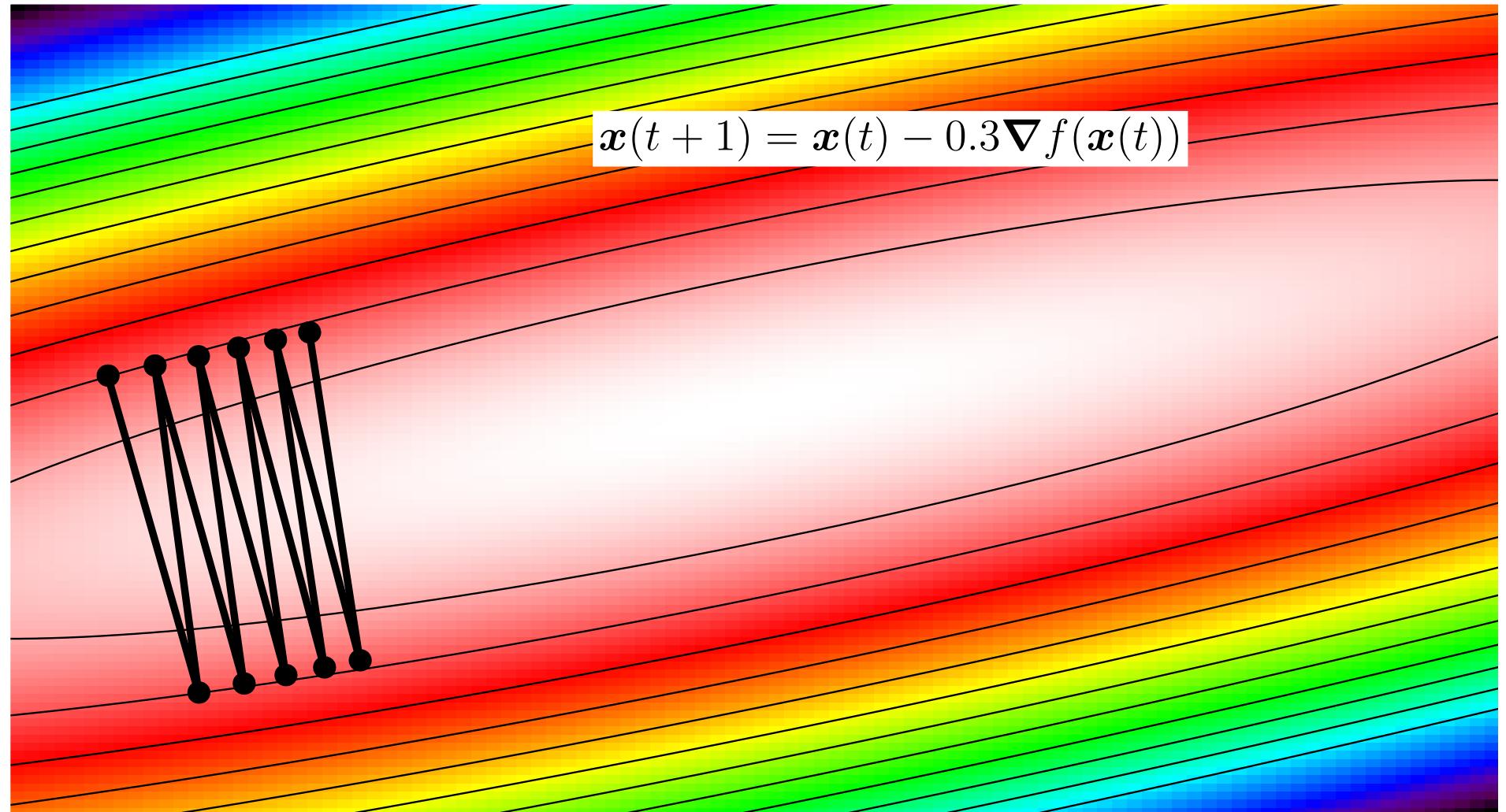
Gradient Descent



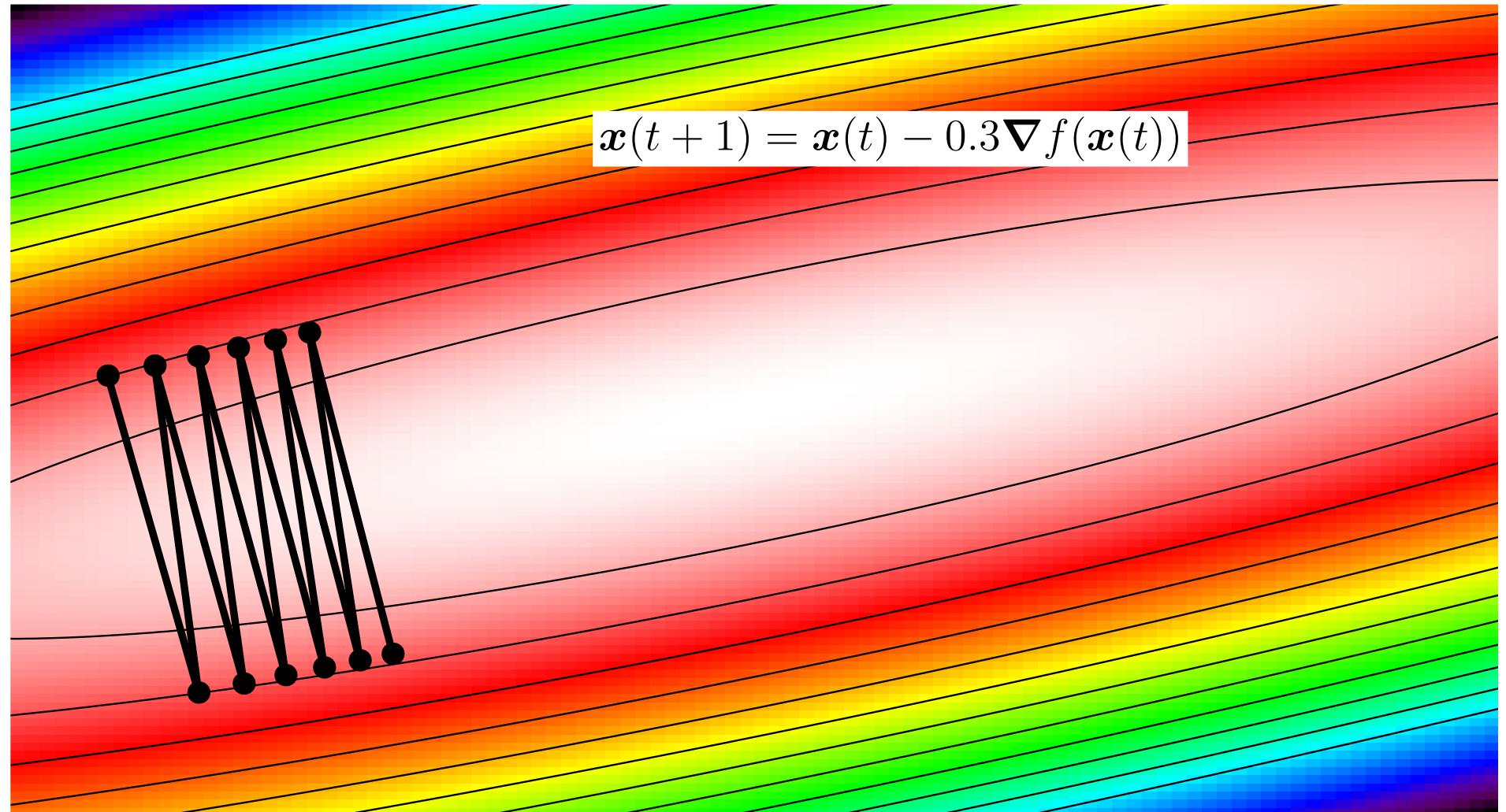
Gradient Descent



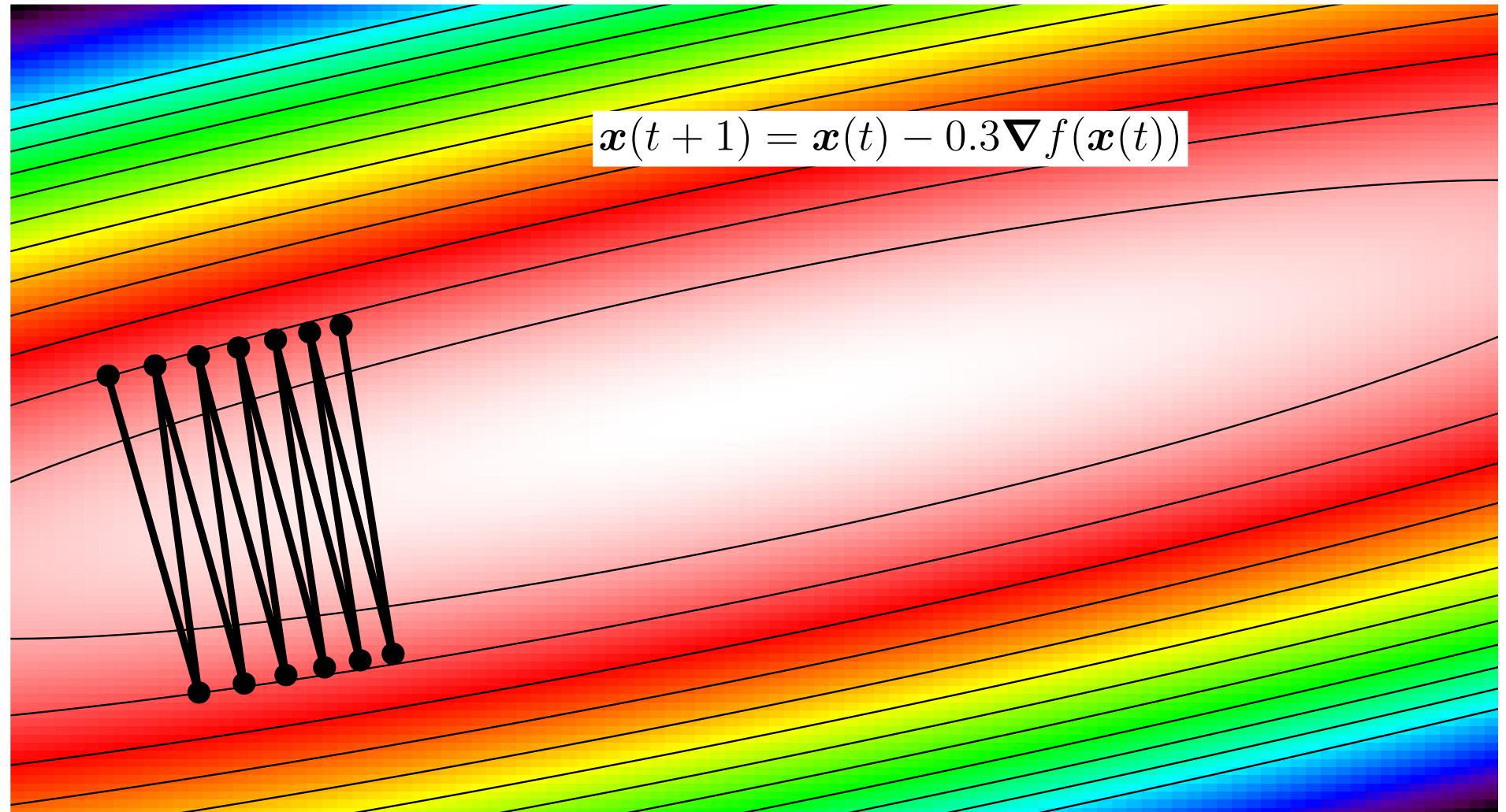
Gradient Descent



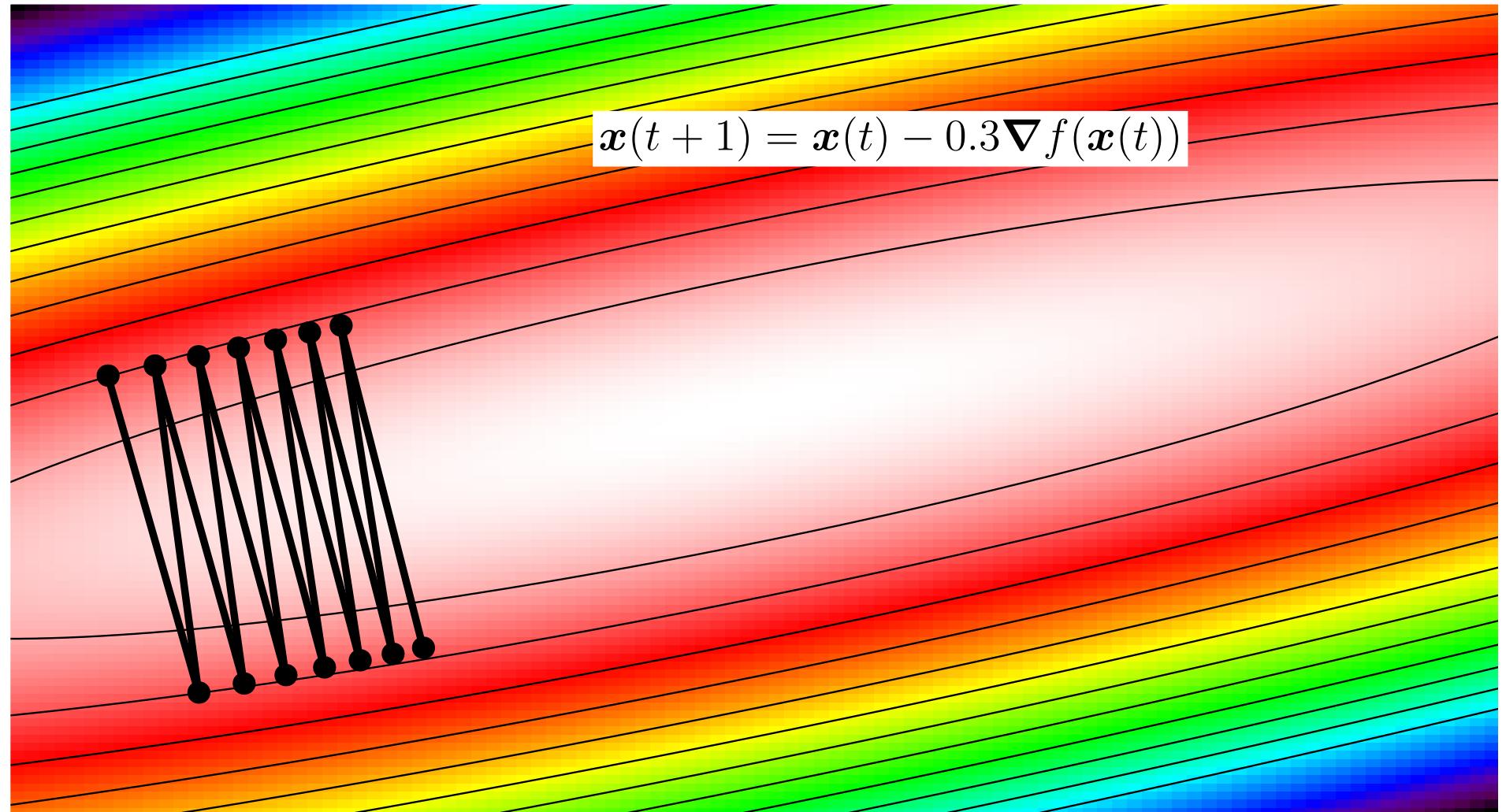
Gradient Descent



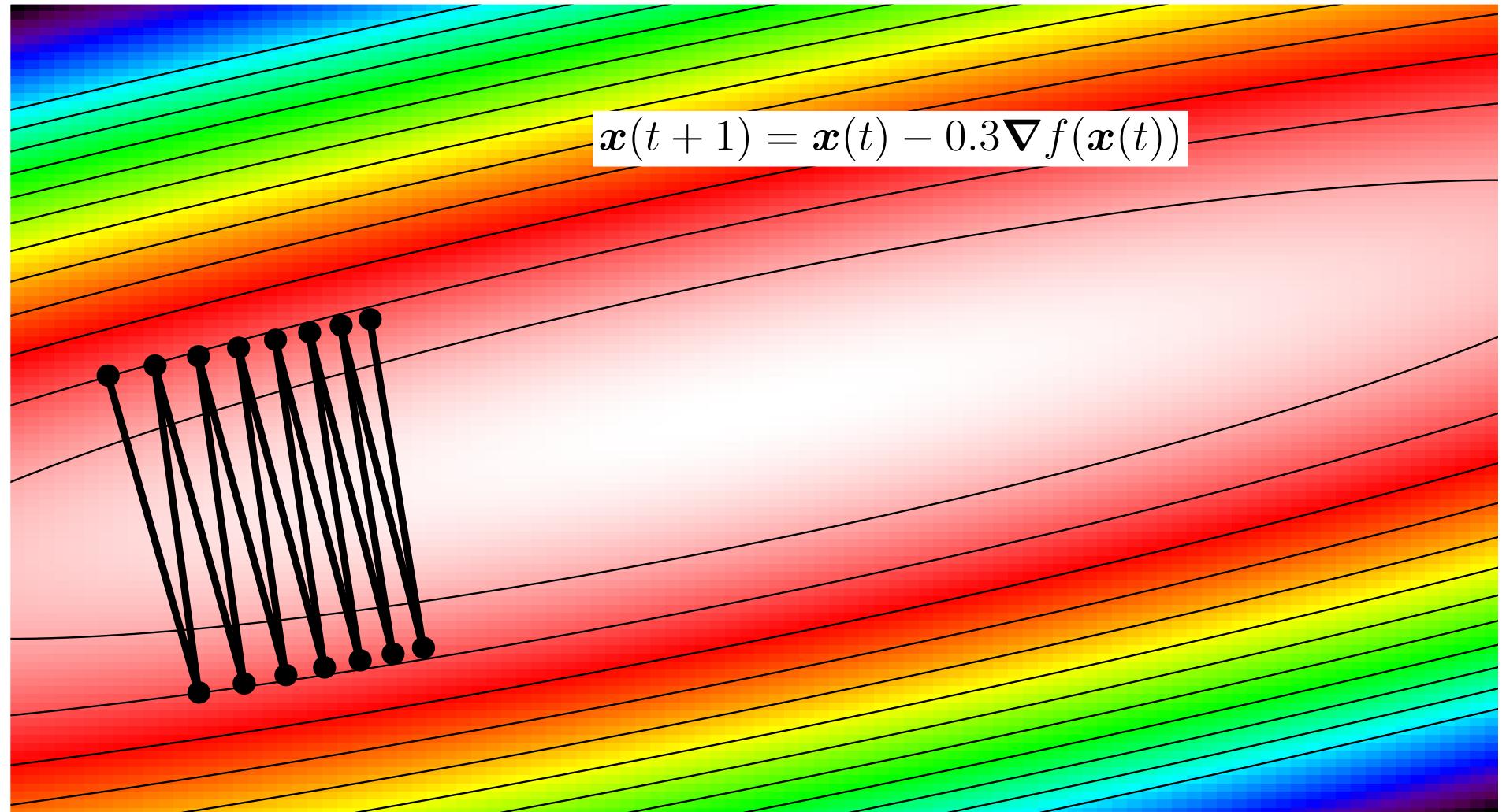
Gradient Descent



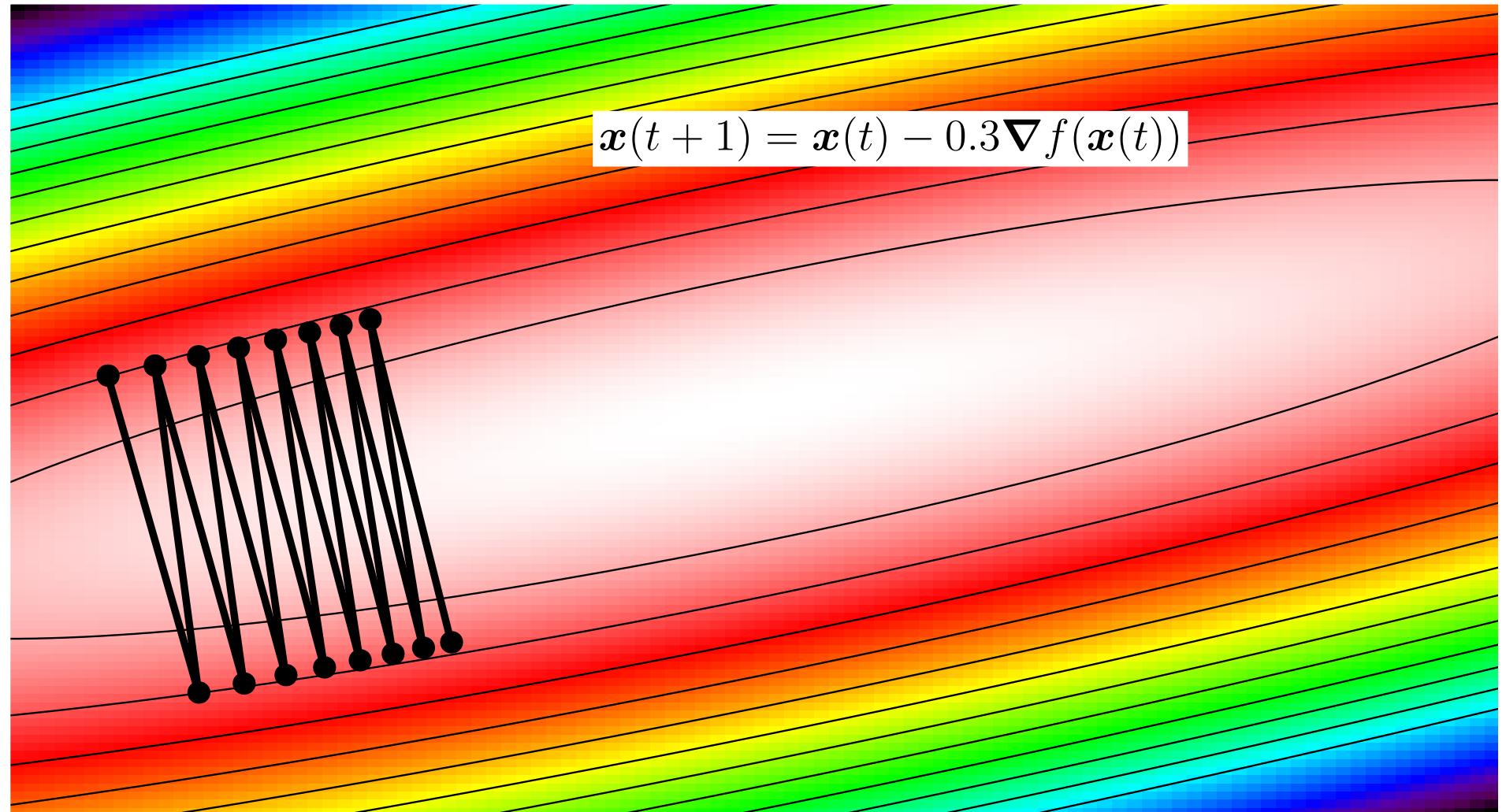
Gradient Descent



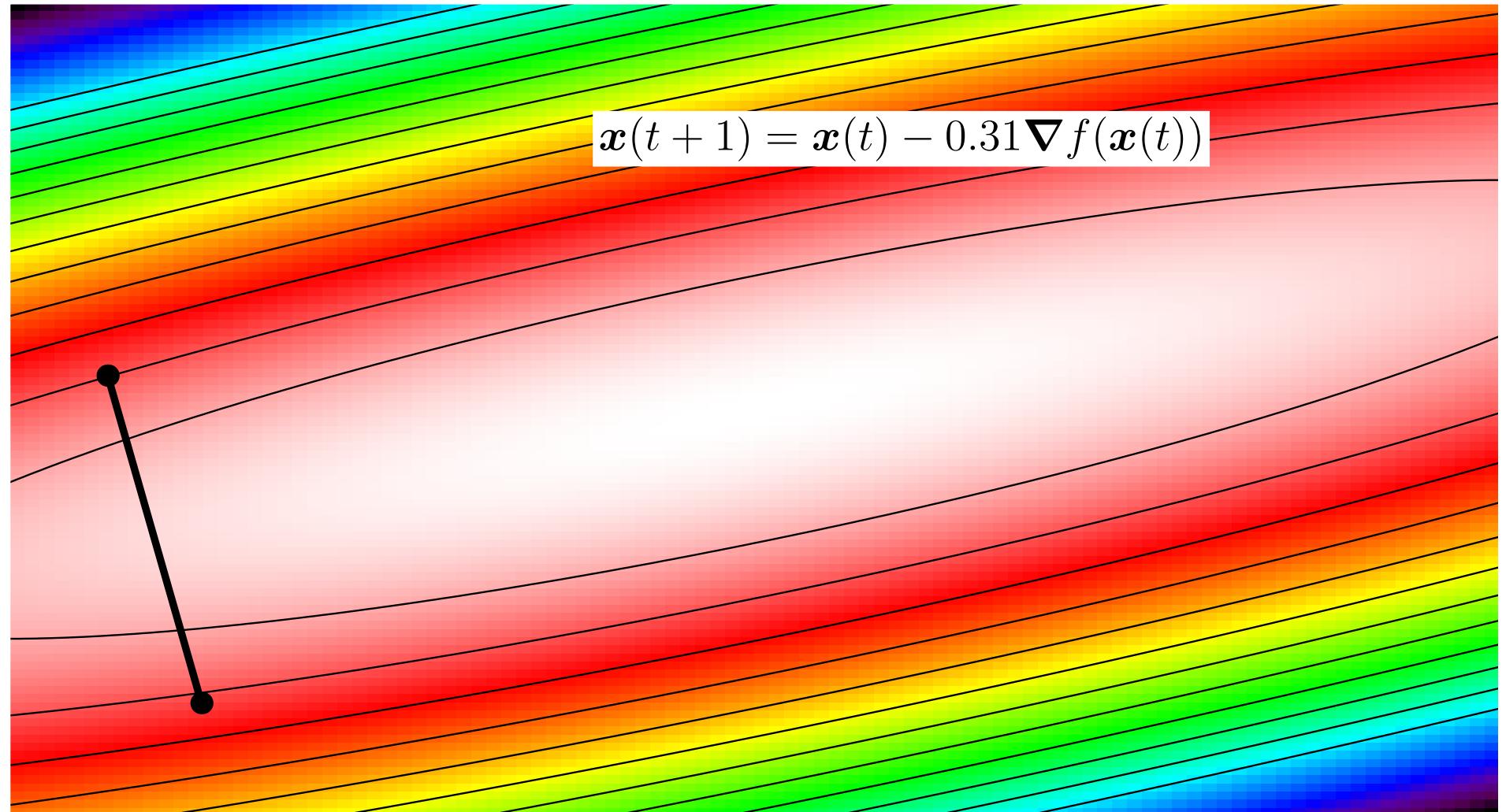
Gradient Descent



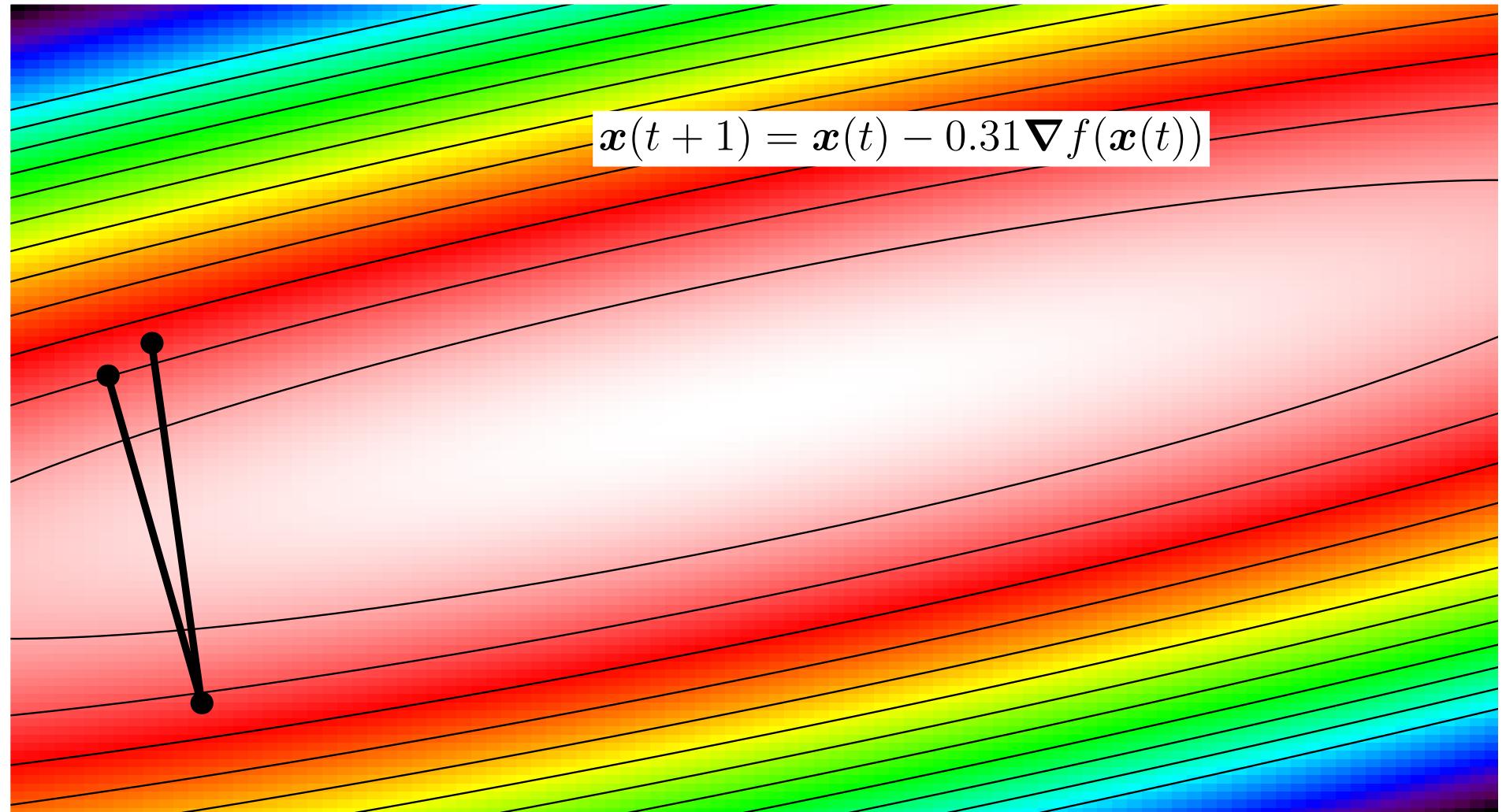
Gradient Descent



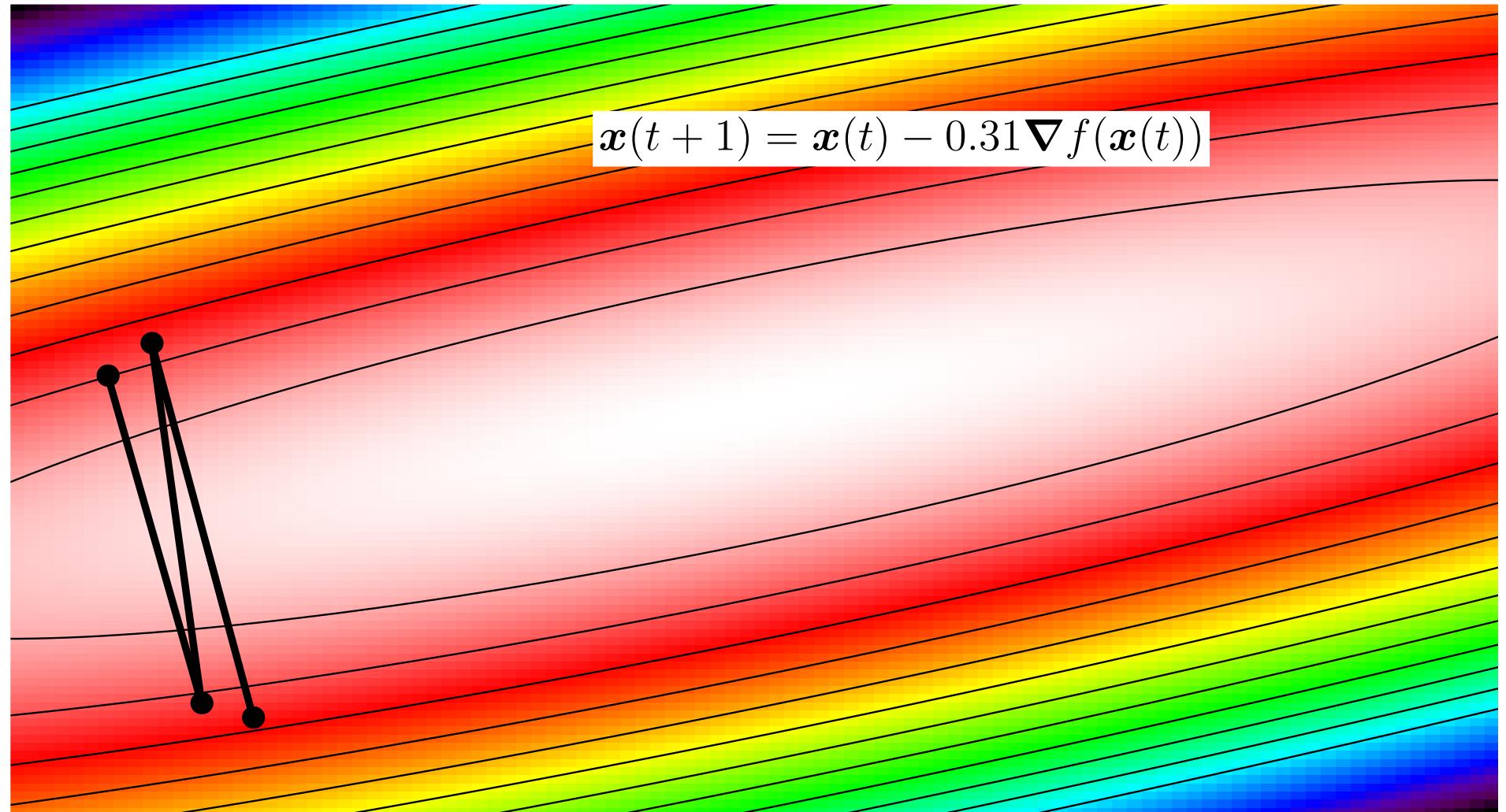
Gradient Descent



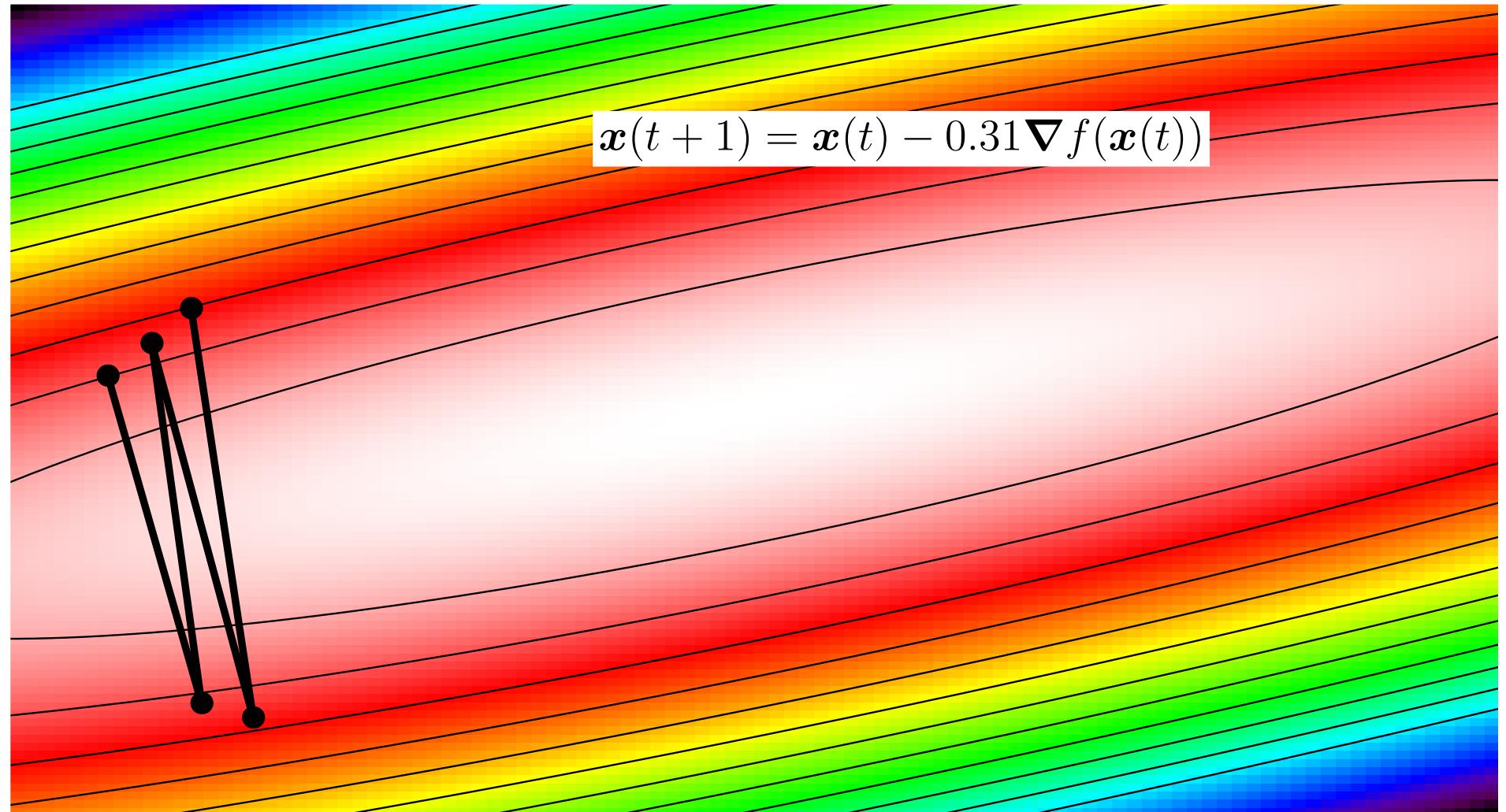
Gradient Descent



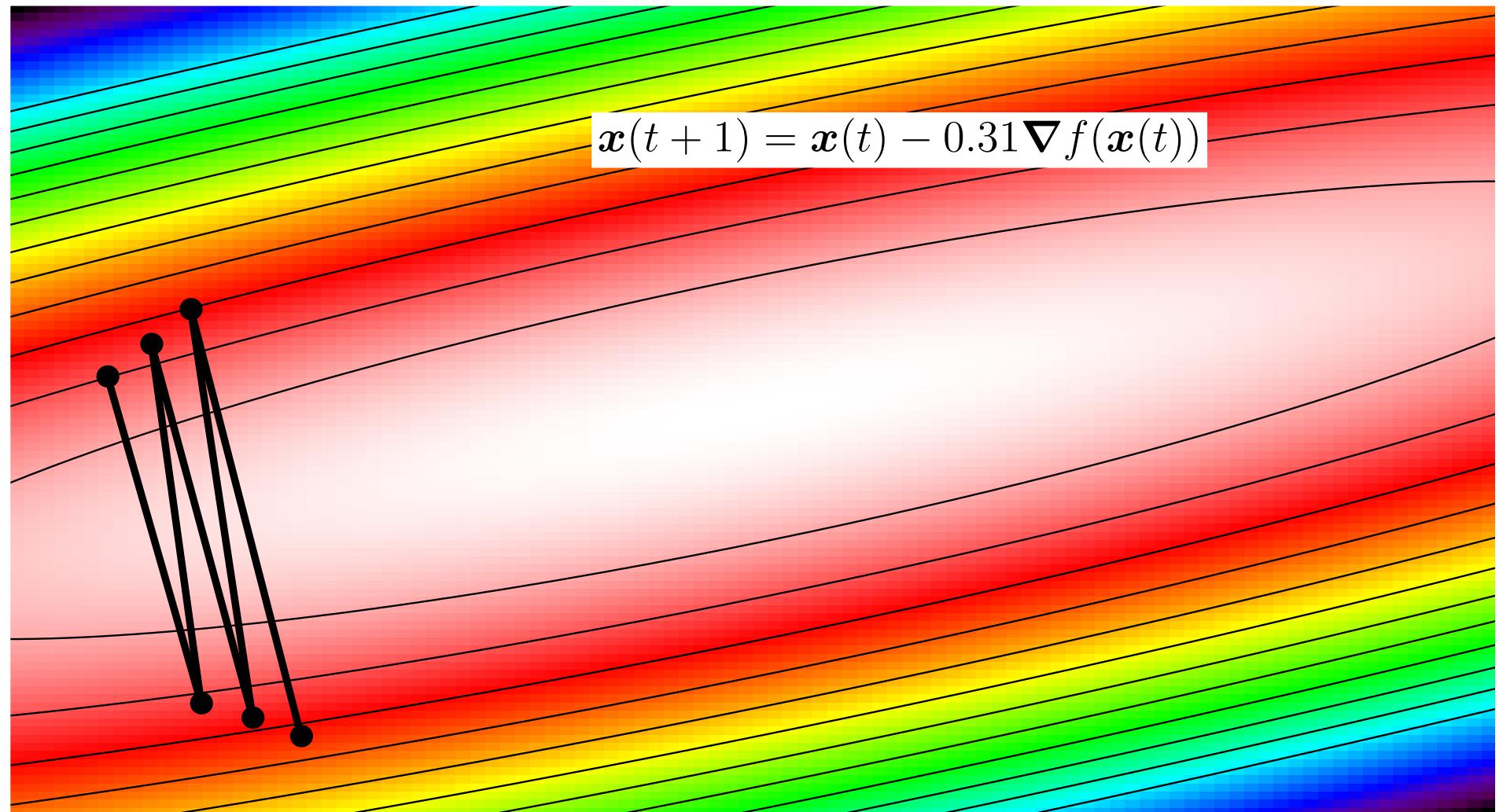
Gradient Descent



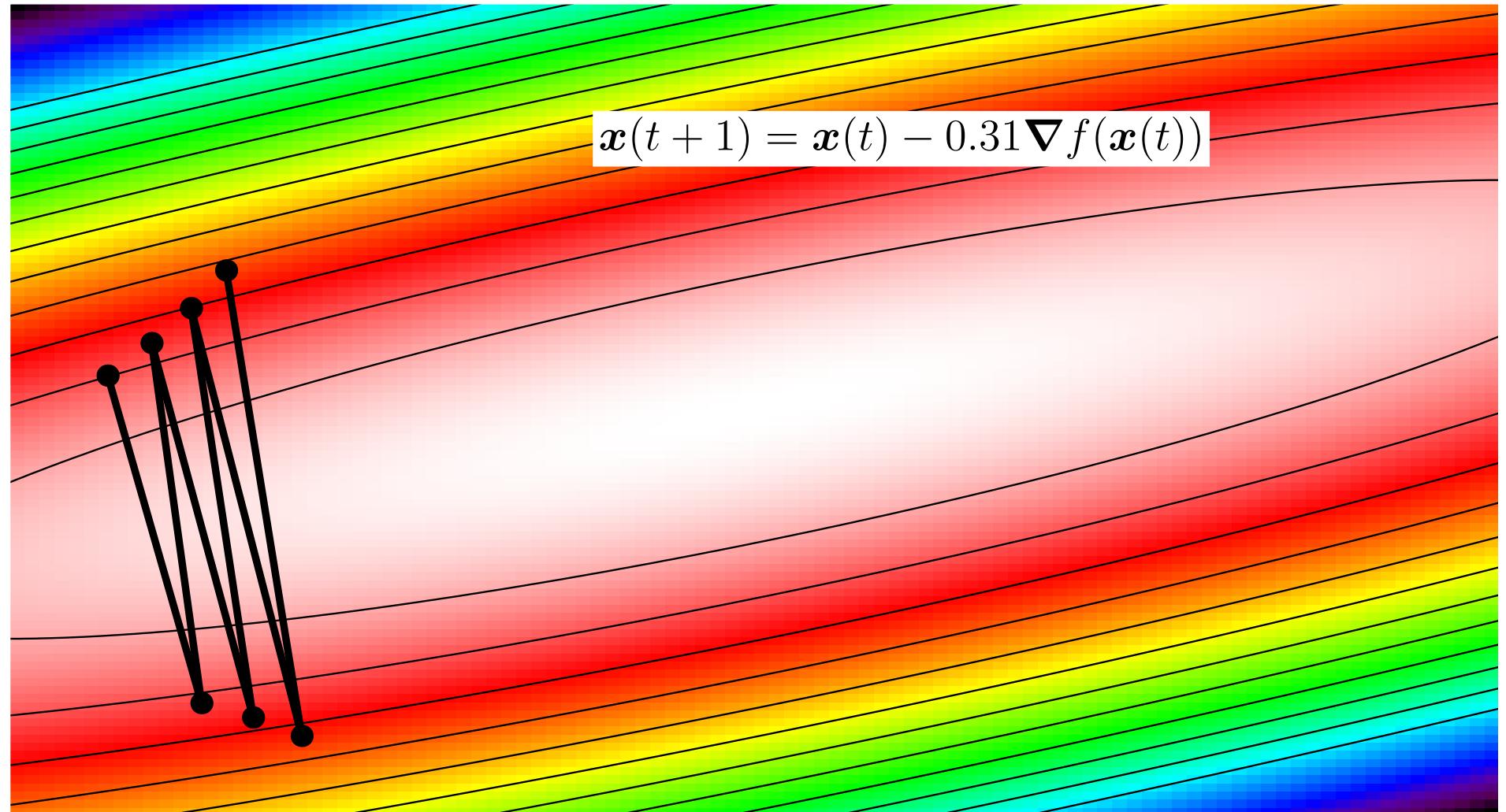
Gradient Descent



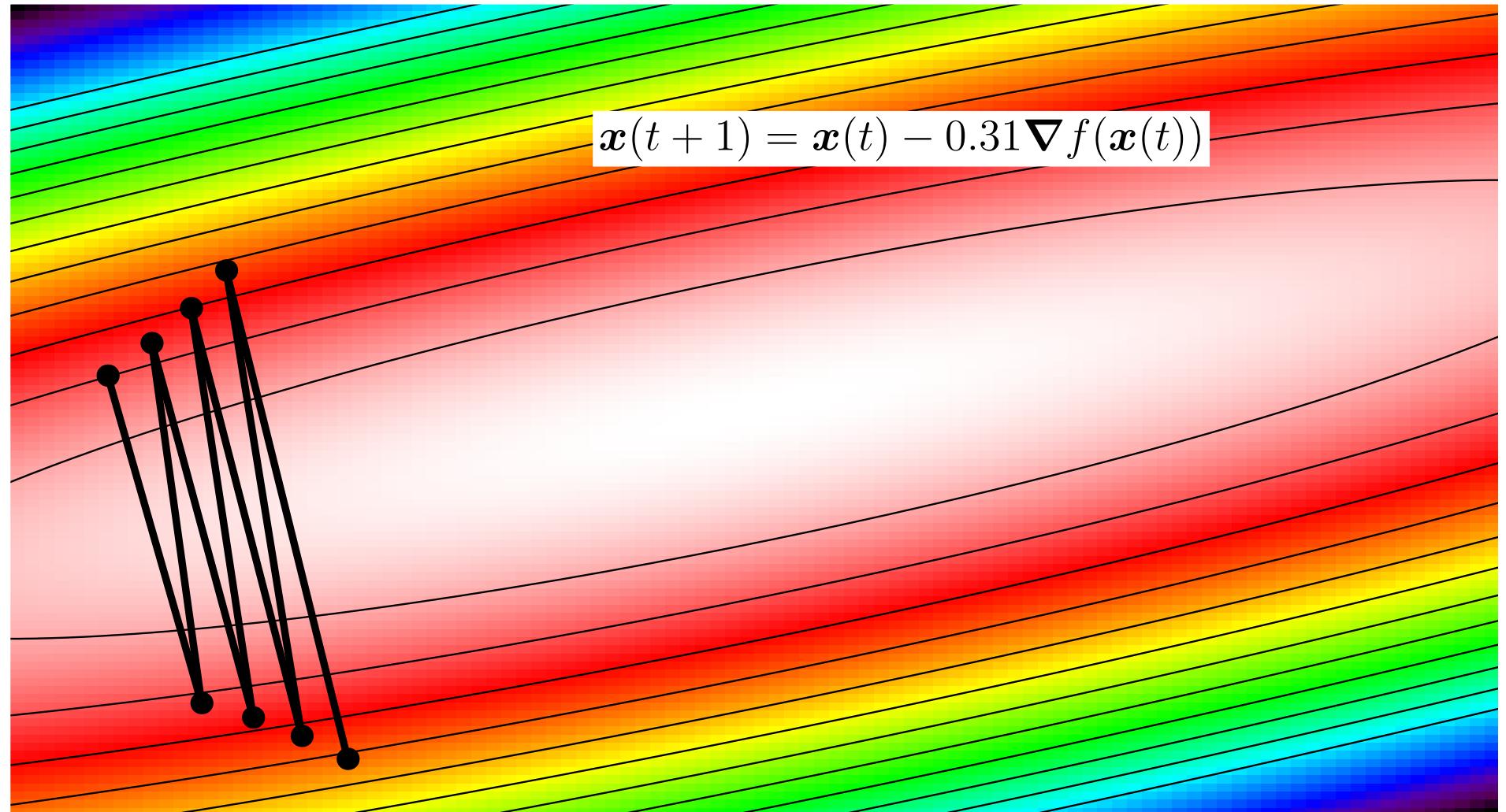
Gradient Descent



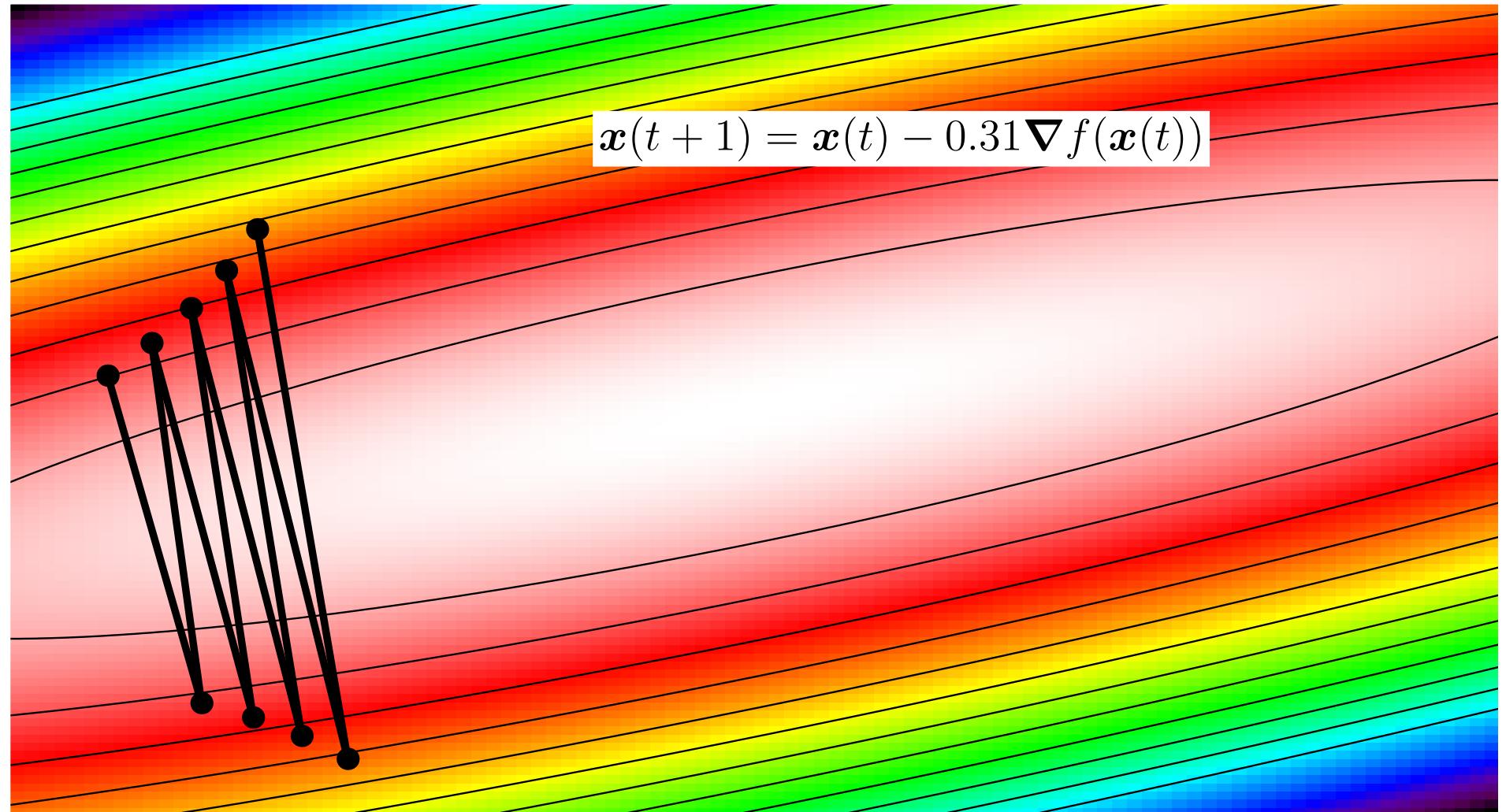
Gradient Descent



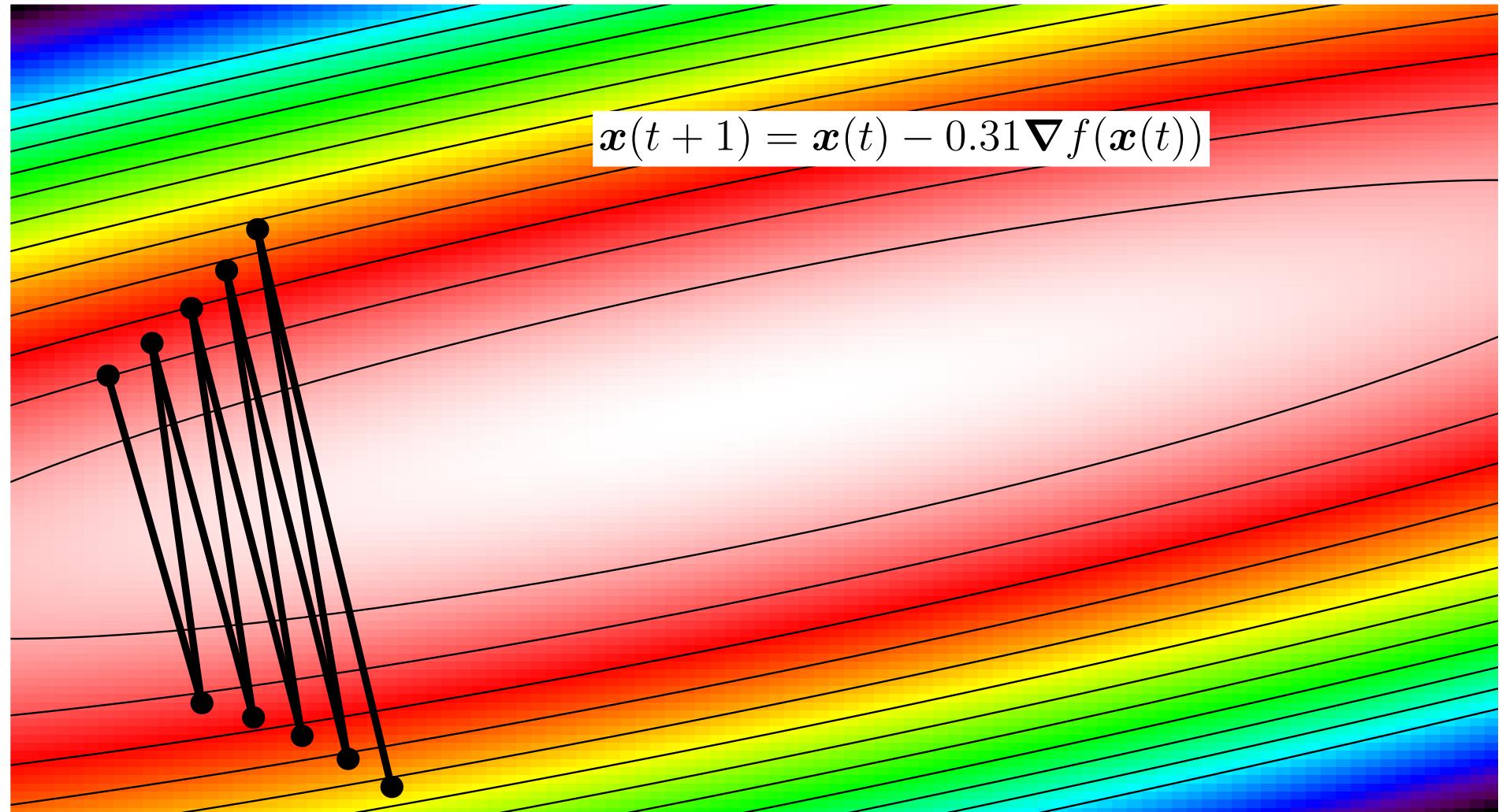
Gradient Descent



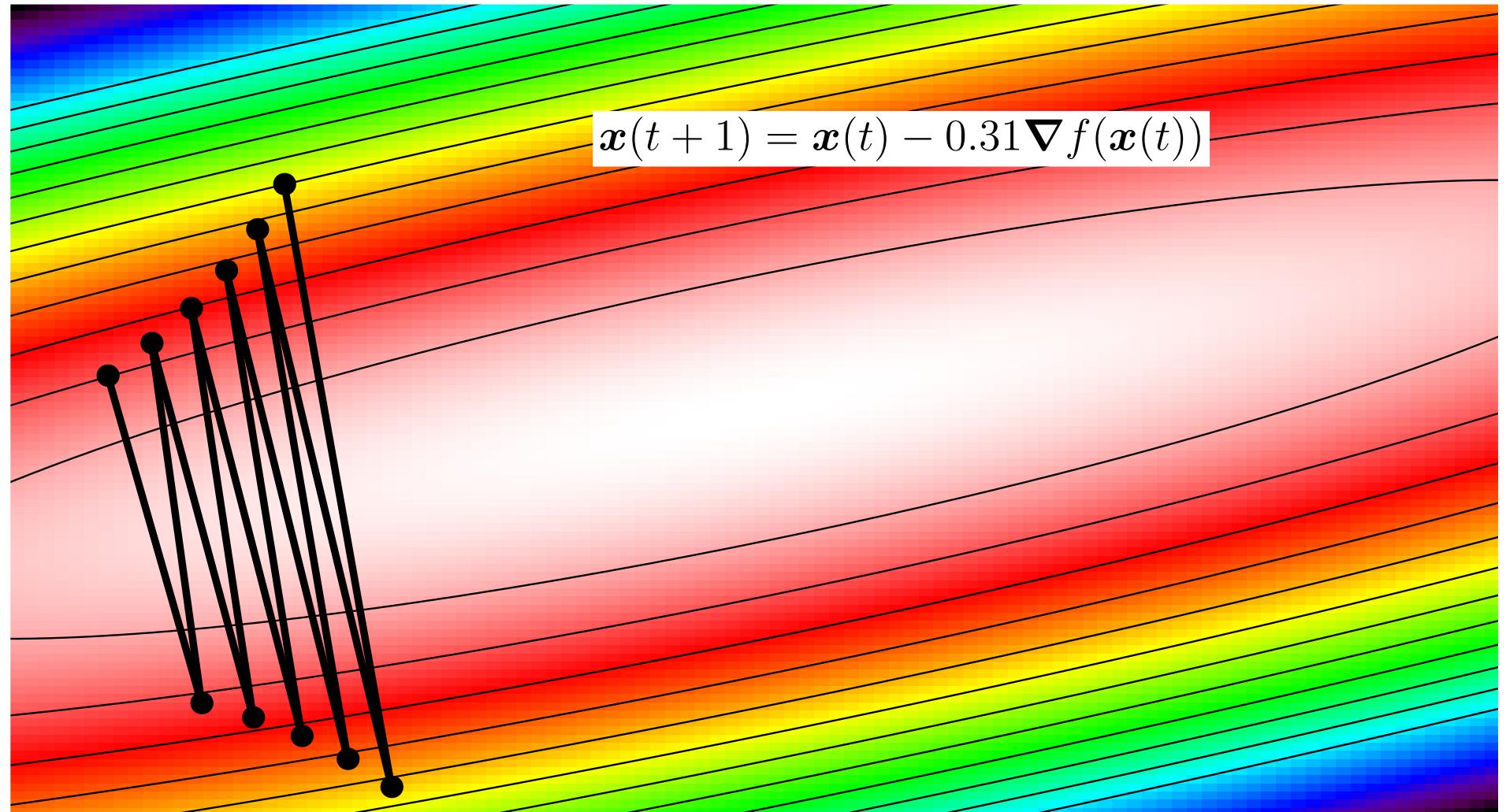
Gradient Descent



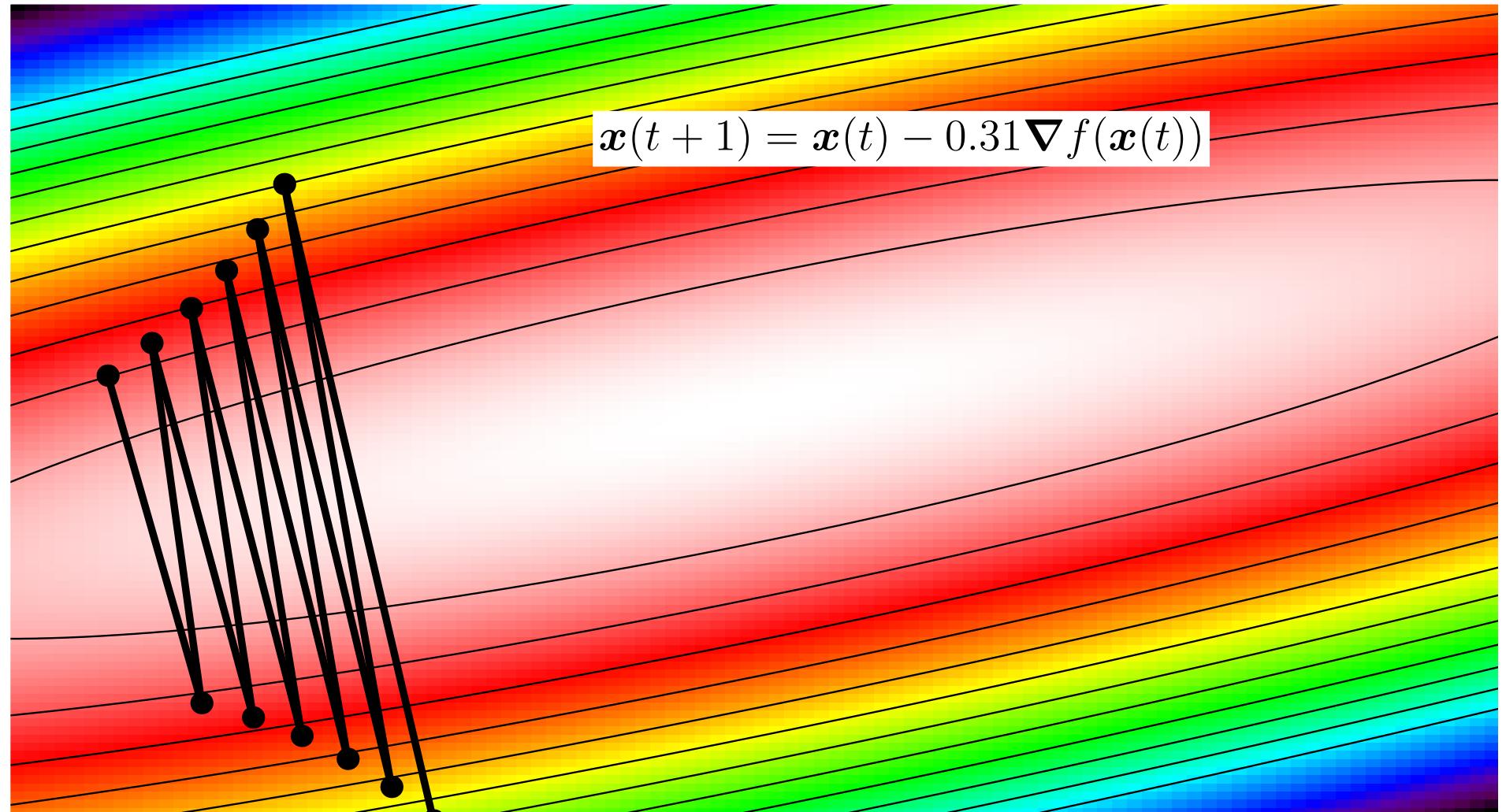
Gradient Descent



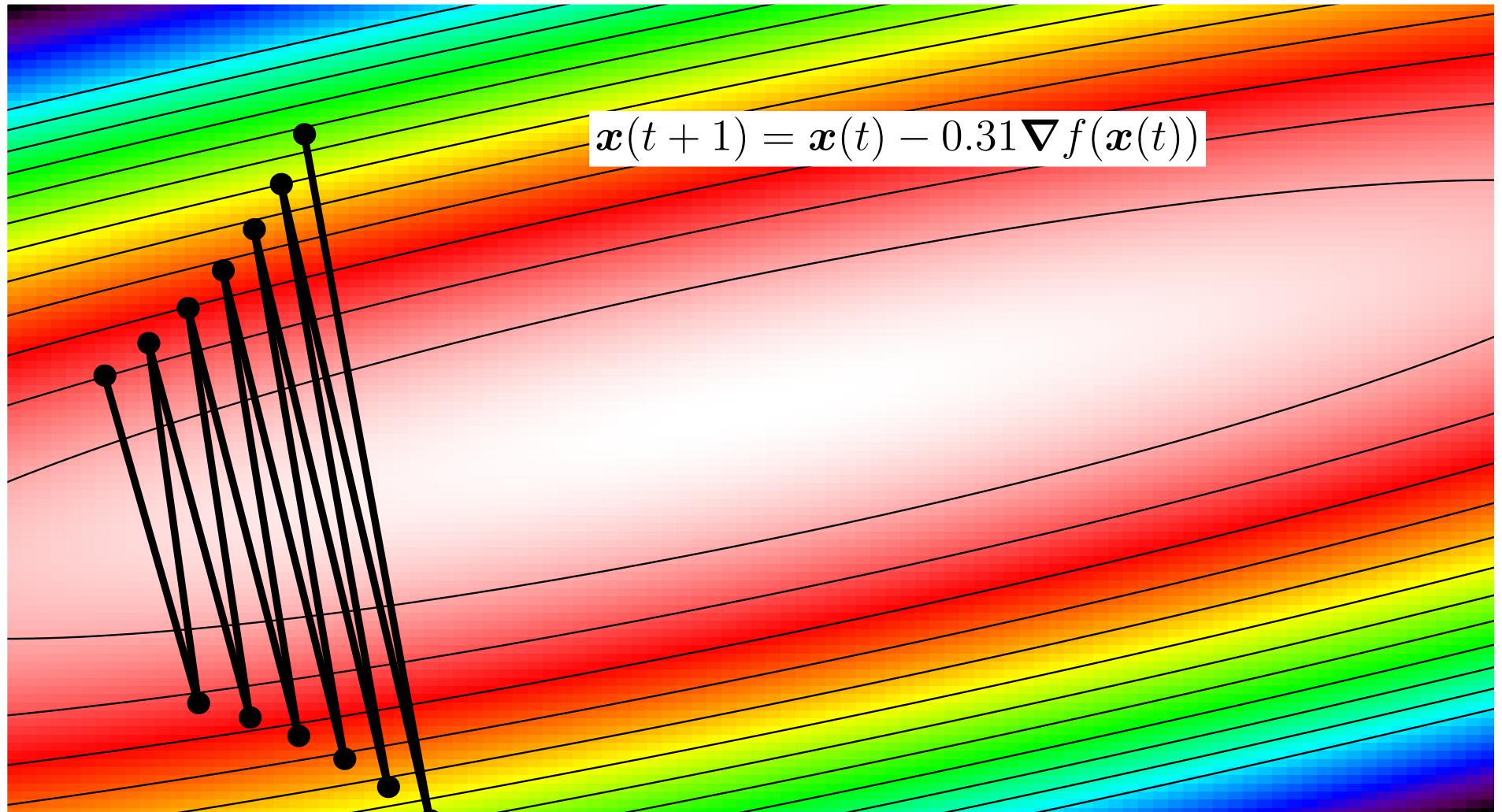
Gradient Descent



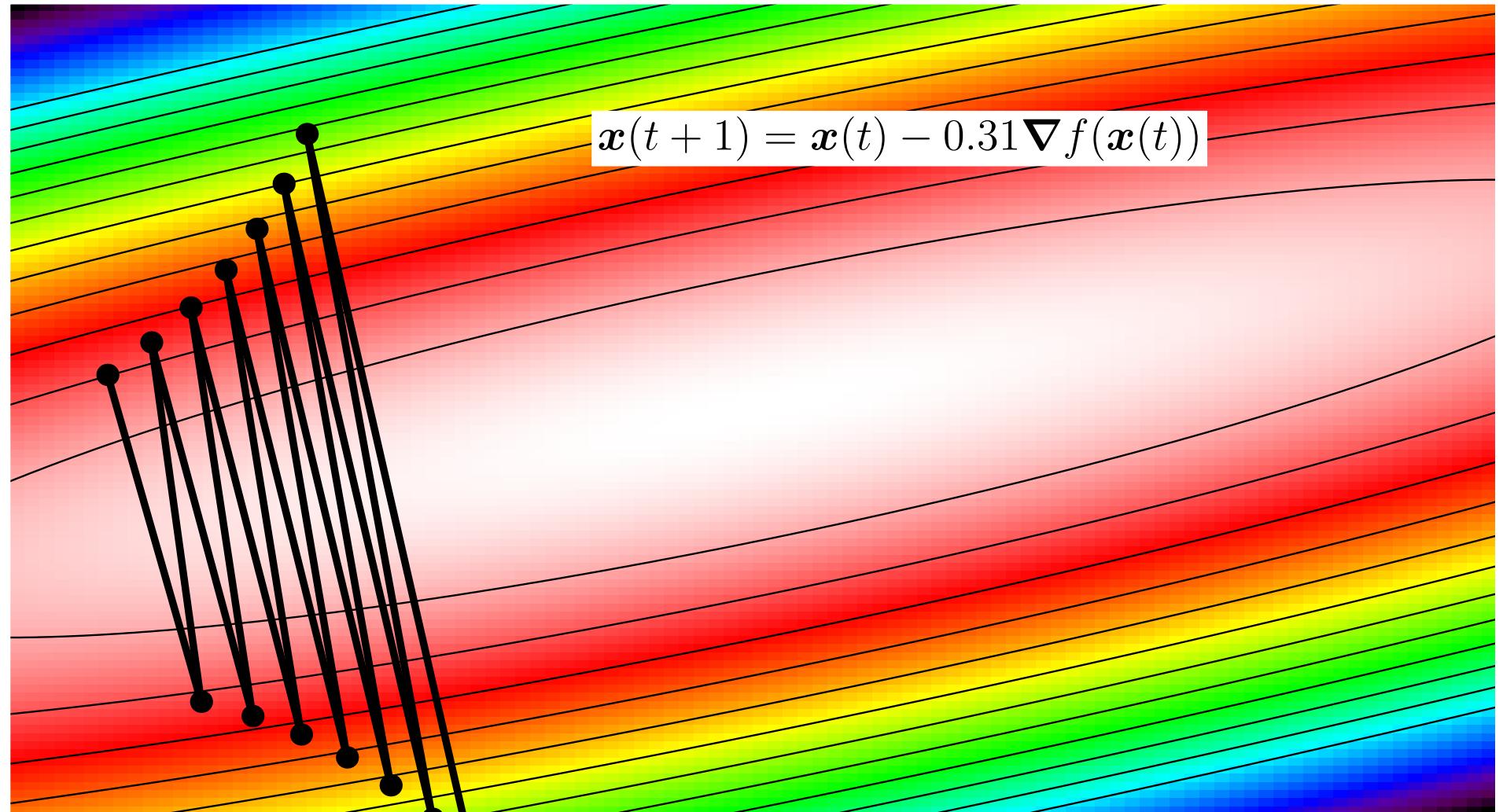
Gradient Descent



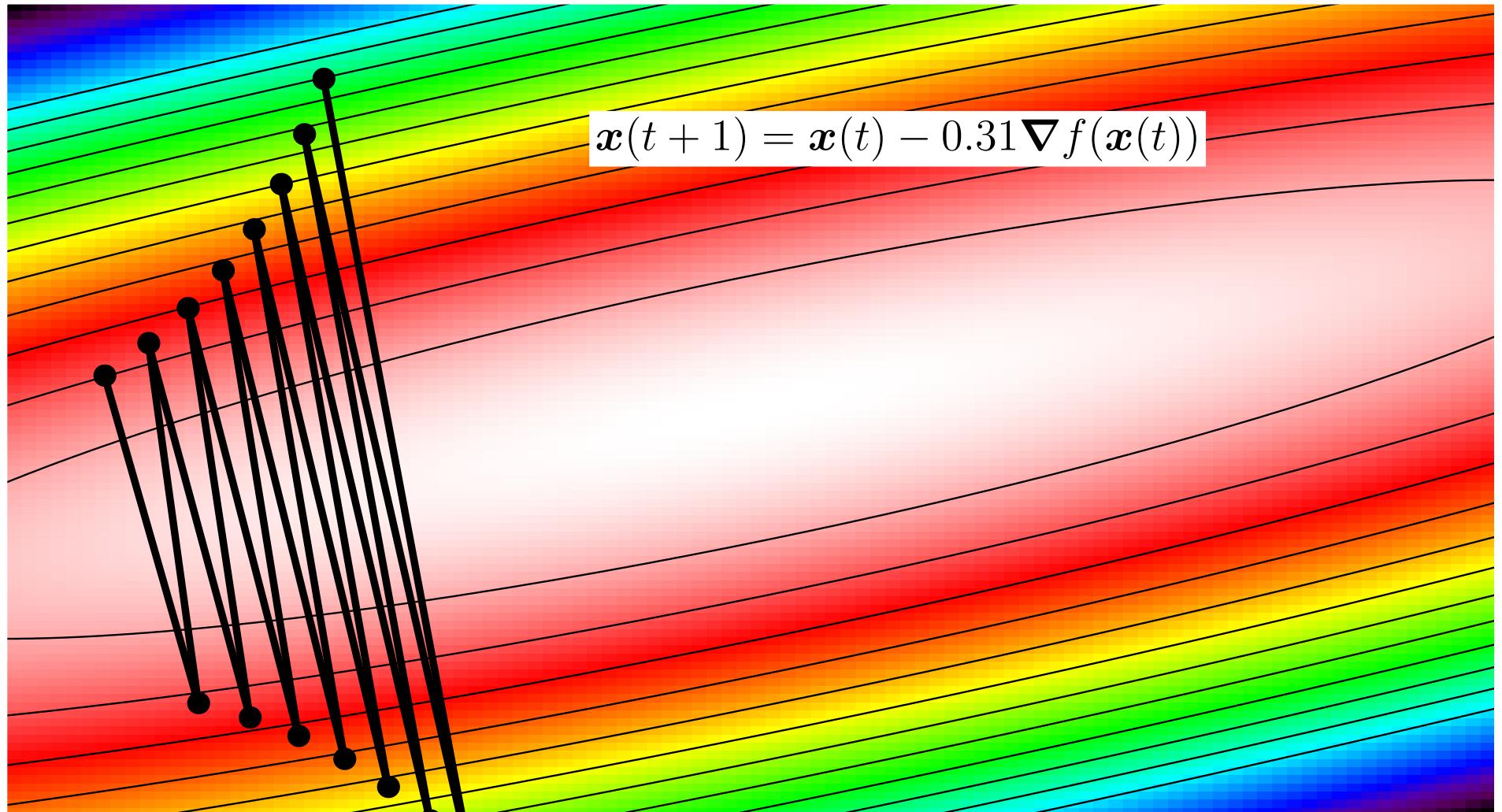
Gradient Descent



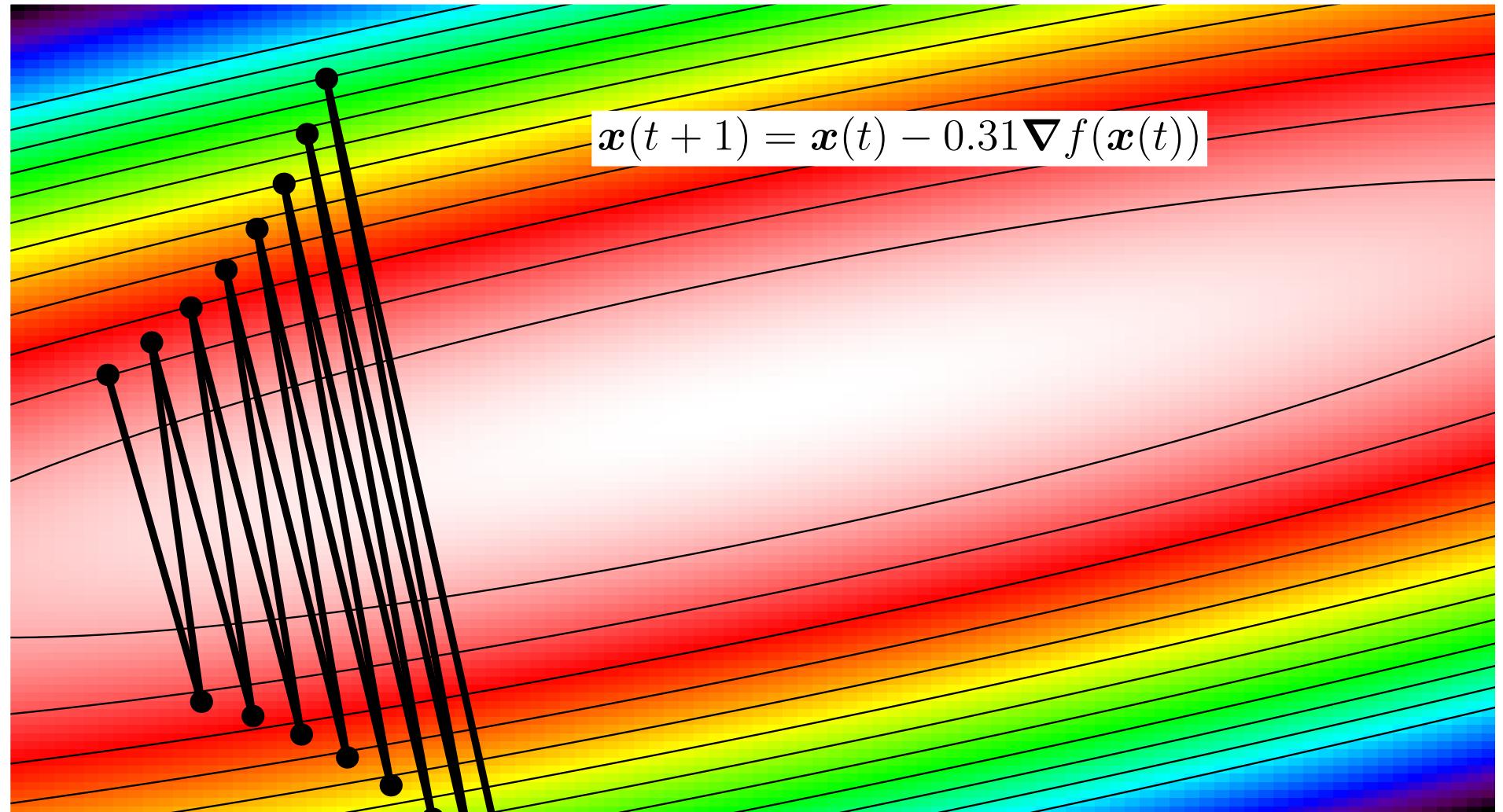
Gradient Descent



Gradient Descent



Gradient Descent



Avoiding Divergence

- We saw in the last lecture that gradient descent can actually diverge from a local optimum if you have a too big step size
- If we use too large a step size we quickly get NaN errors
- If we only use SGD the step size is determined by the largest eigenvalue of the Hessian

Avoiding Divergence

- We saw in the last lecture that gradient descent can actually diverge from a local optimum if you have a too big step size
- If we use too large a step size we quickly get NaN errors
- If we only use SGD the step size is determined by the largest eigenvalue of the Hessian

Avoiding Divergence

- We saw in the last lecture that gradient descent can actually diverge from a local optimum if you have a too big step size
- If we use too large a step size we quickly get NaN errors
- If we only use SGD the step size is determined by the largest eigenvalue of the Hessian

Avoiding Divergence

- We saw in the last lecture that gradient descent can actually diverge from a local optimum if you have a too big step size
- If we use too large a step size we quickly get NaN errors
- If we only use SGD the step size is determined by the largest eigenvalue of the Hessian
- This seems impossibly slow

Avoiding Divergence

- We saw in the last lecture that gradient descent can actually diverge from a local optimum if you have a too big step size
- If we use too large a step size we quickly get NaN errors
- If we only use SGD the step size is determined by the largest eigenvalue of the Hessian
- This seems impossibly slow, but weirdly straightforward SGD is often used in deep learning

Avoiding Divergence

- We saw in the last lecture that gradient descent can actually diverge from a local optimum if you have a too big step size
- If we use too large a step size we quickly get NaN errors
- If we only use SGD the step size is determined by the largest eigenvalue of the Hessian
- This seems impossibly slow, but weirdly straightforward SGD is often used in deep learning (**although nearly always with momentum**)

Momentum

- In high dimensions we zig-zag:
 - ★ stepping consistently in directions with low curvature
 - ★ jumping backwards and forwards past the minimum in directions with high curvature
- By introducing “momentum” we can increase our steps in low curvature directions and decrease it in high curvature directions

$$\mathbf{v}(t + 1) = \gamma \mathbf{v}(t) - \eta \nabla f(\mathbf{x}(t))$$

$$\mathbf{x}(t + 1) = \mathbf{x}(t) + \mathbf{v}(t + 1)$$

Momentum

- In high dimensions we zig-zag:
 - ★ stepping consistently in directions with low curvature
 - ★ jumping backwards and forwards past the minimum in directions with high curvature
- By introducing “momentum” we can increase our steps in low curvature directions and decrease it in high curvature directions

$$\mathbf{v}(t+1) = \gamma \mathbf{v}(t) - \eta \nabla f(\mathbf{x}(t))$$

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \mathbf{v}(t+1)$$

Momentum

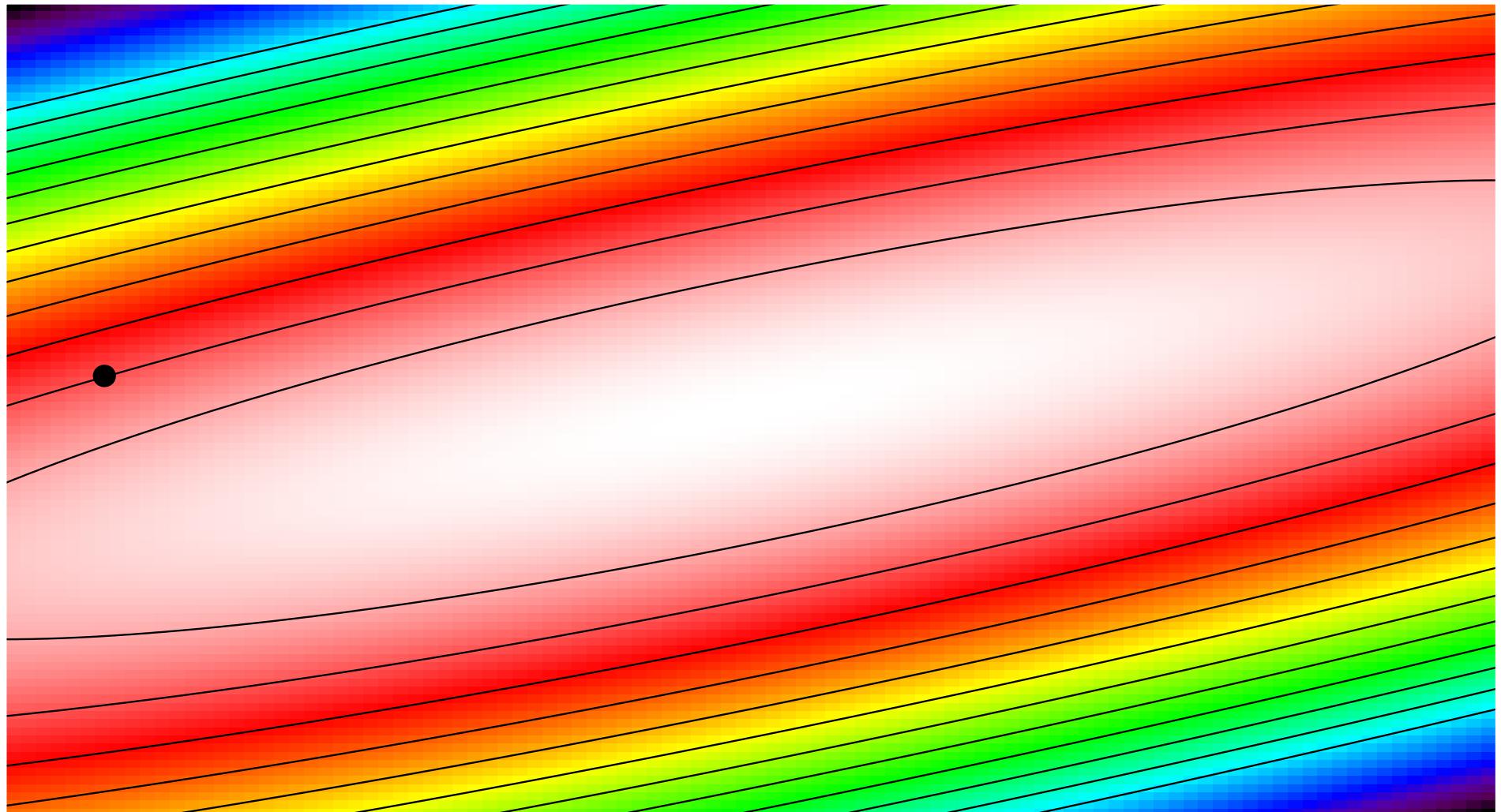
- In high dimensions we zig-zag:
 - ★ stepping consistently in directions with low curvature
 - ★ jumping backwards and forwards past the minimum in directions with high curvature
- By introducing “momentum” we can increase our steps in low curvature directions and decrease it in high curvature directions

$$\mathbf{v}(t + 1) = \gamma \mathbf{v}(t) - \eta \nabla f(\mathbf{x}(t))$$

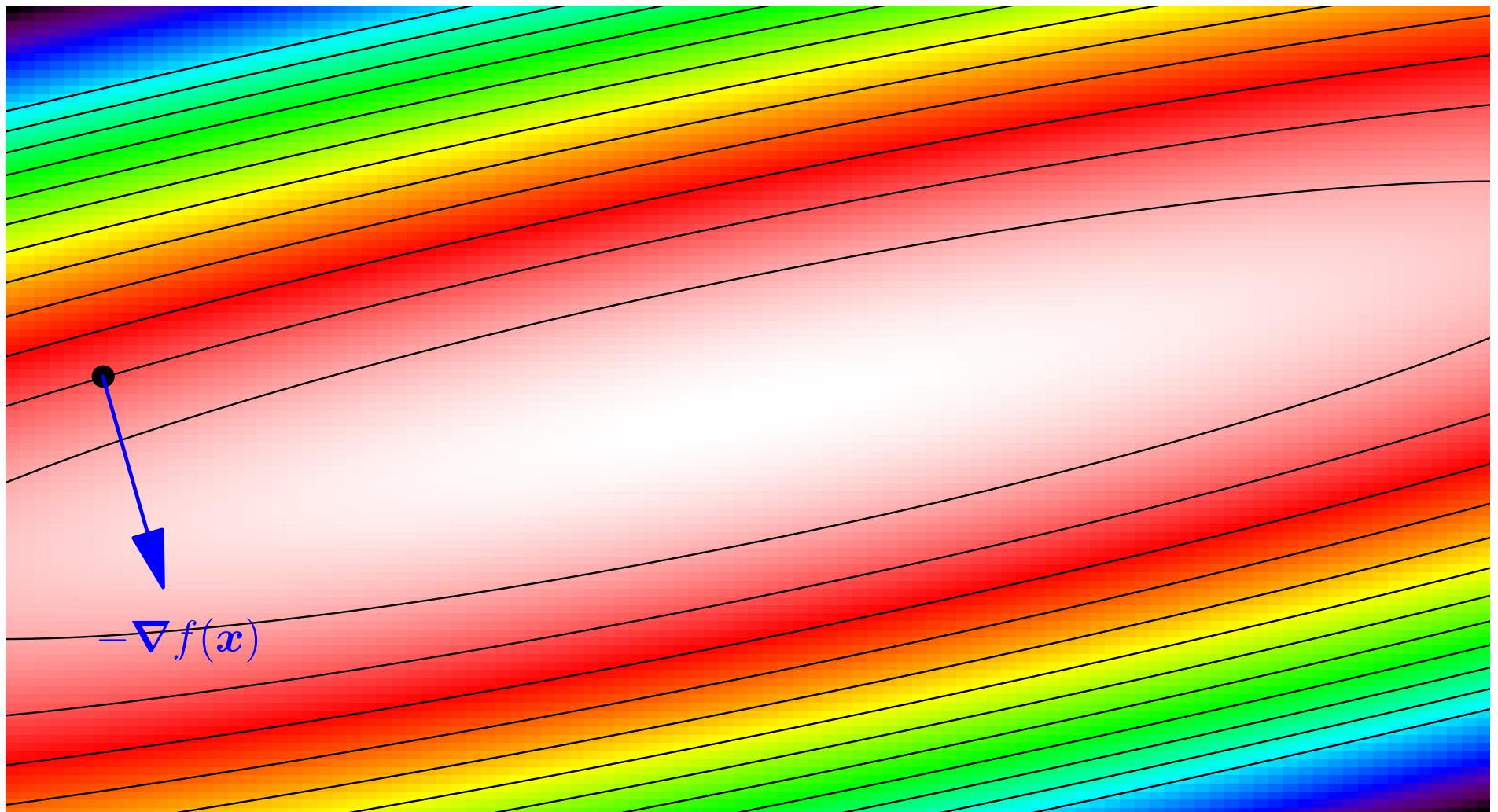
$$\mathbf{x}(t + 1) = \mathbf{x}(t) + \mathbf{v}(t + 1)$$

$0 < \gamma < 1$ is a damping term

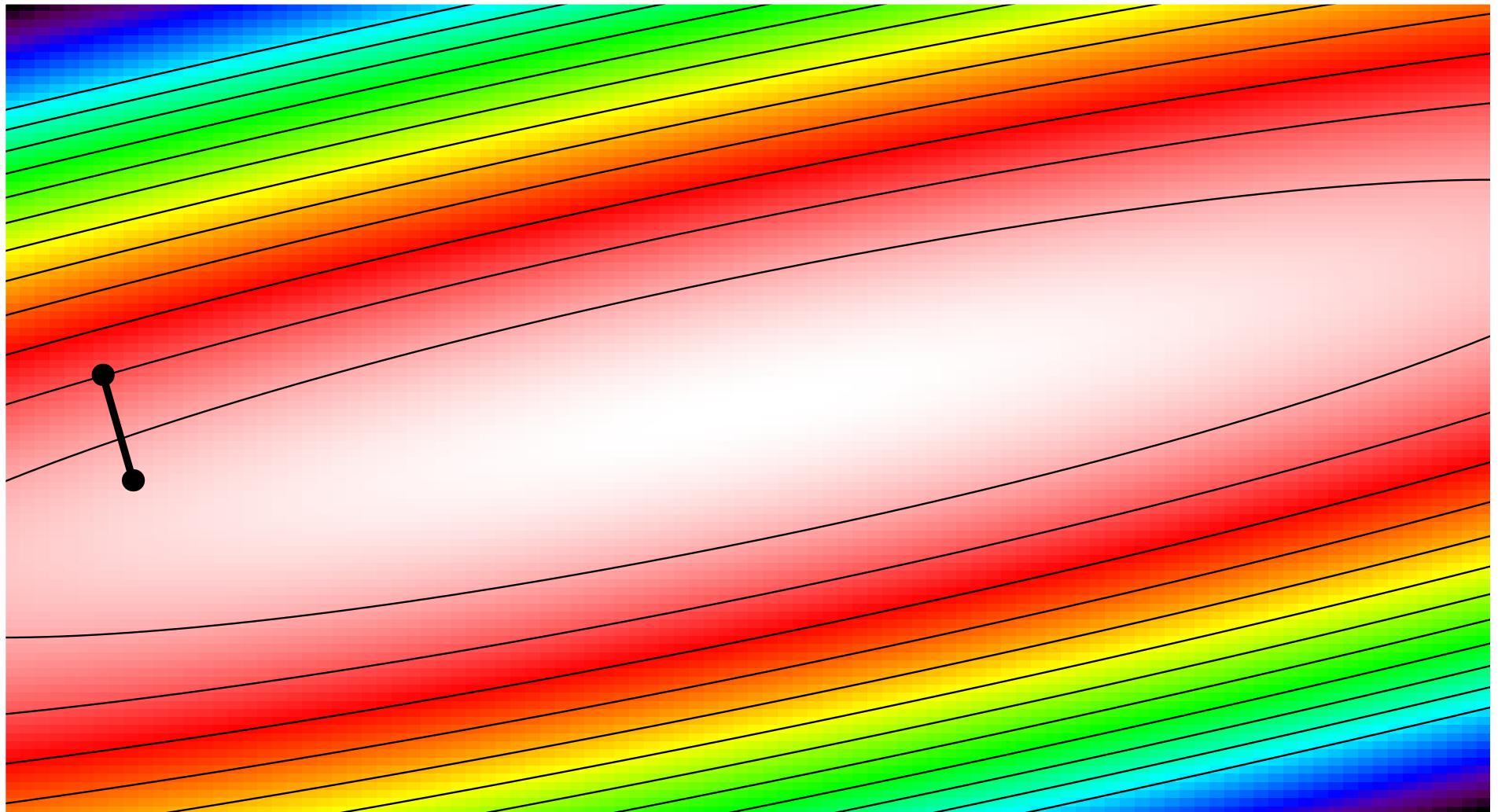
Gradient Descent with Momentum



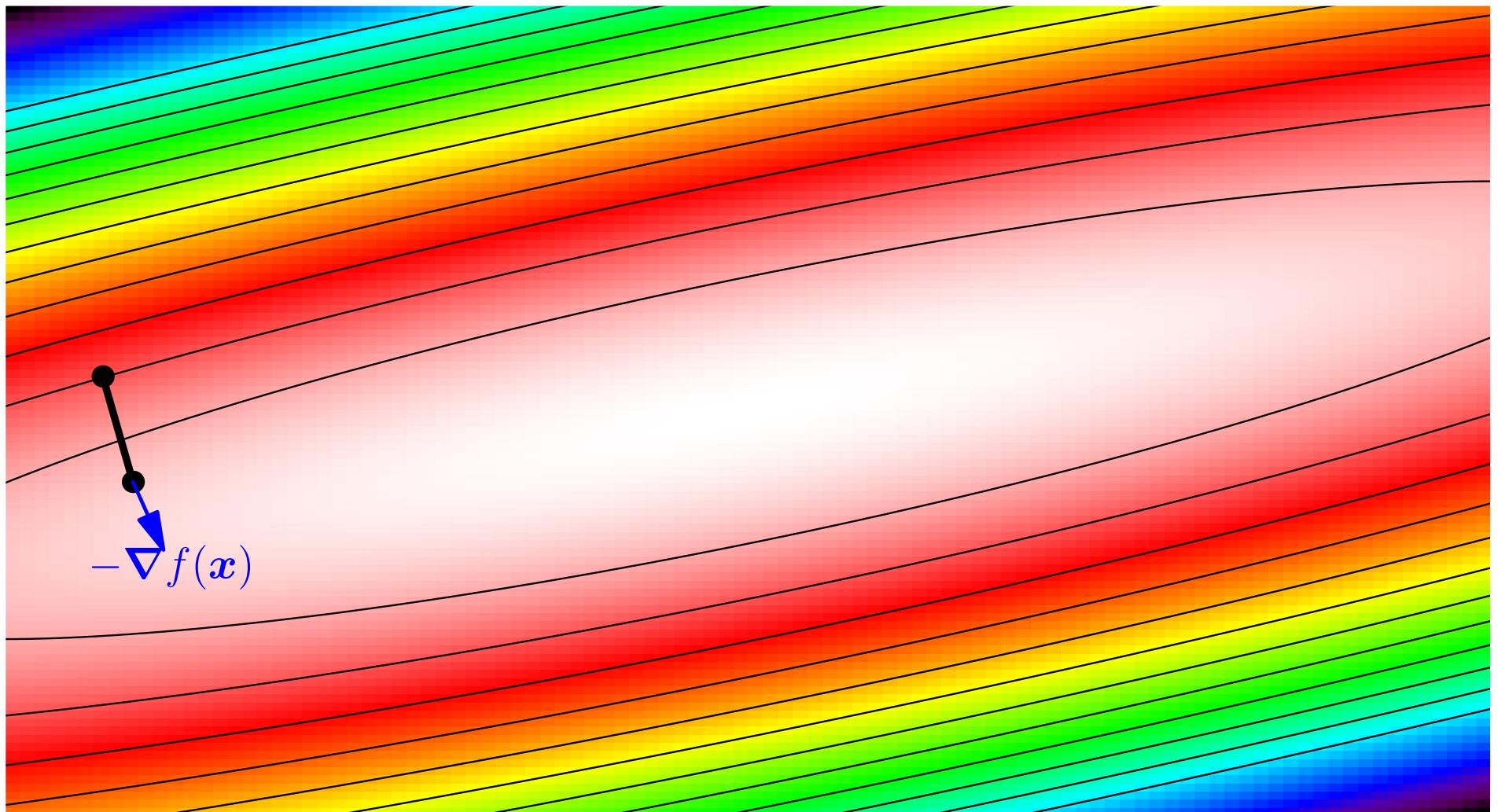
Gradient Descent with Momentum



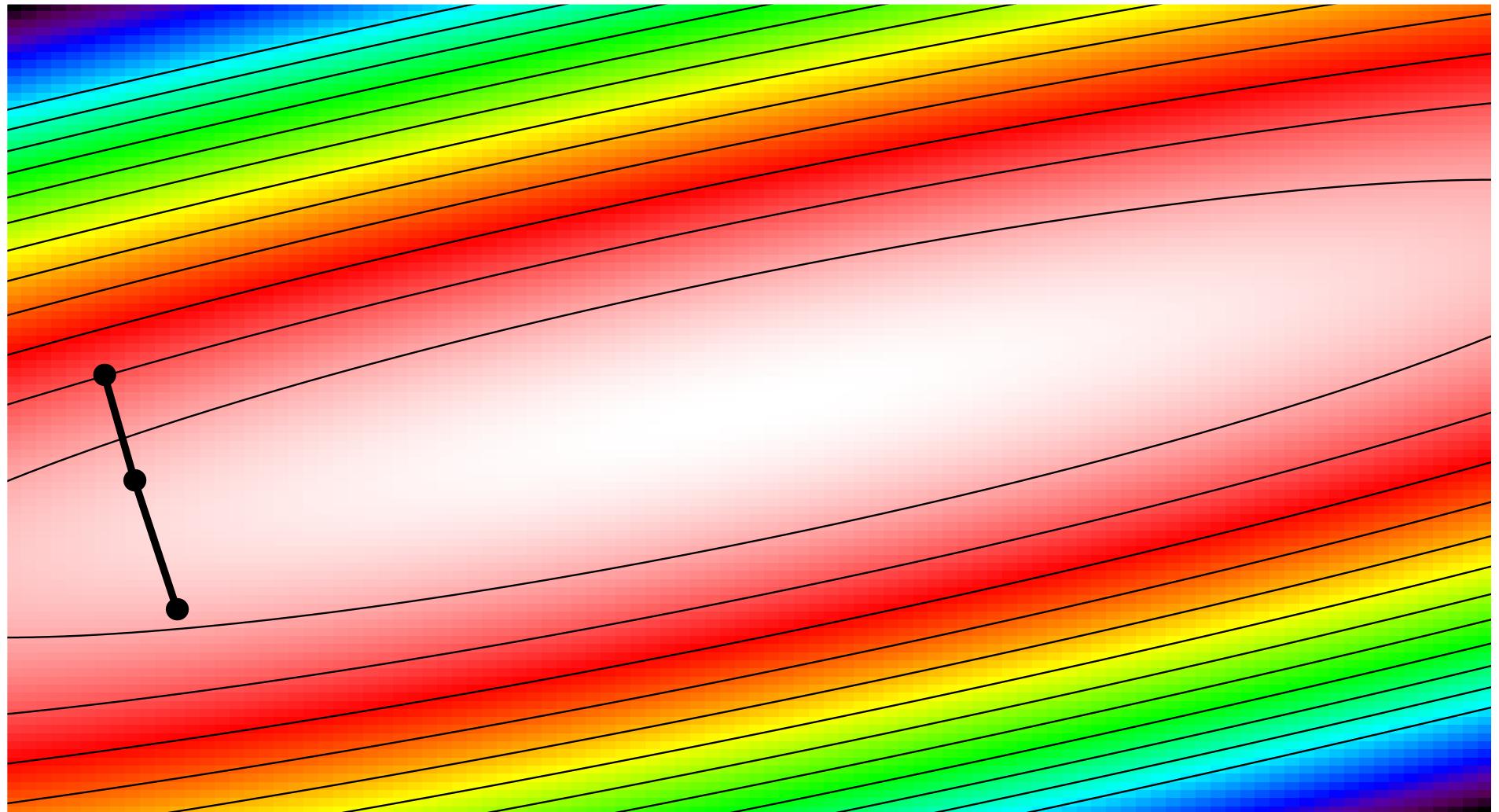
Gradient Descent with Momentum



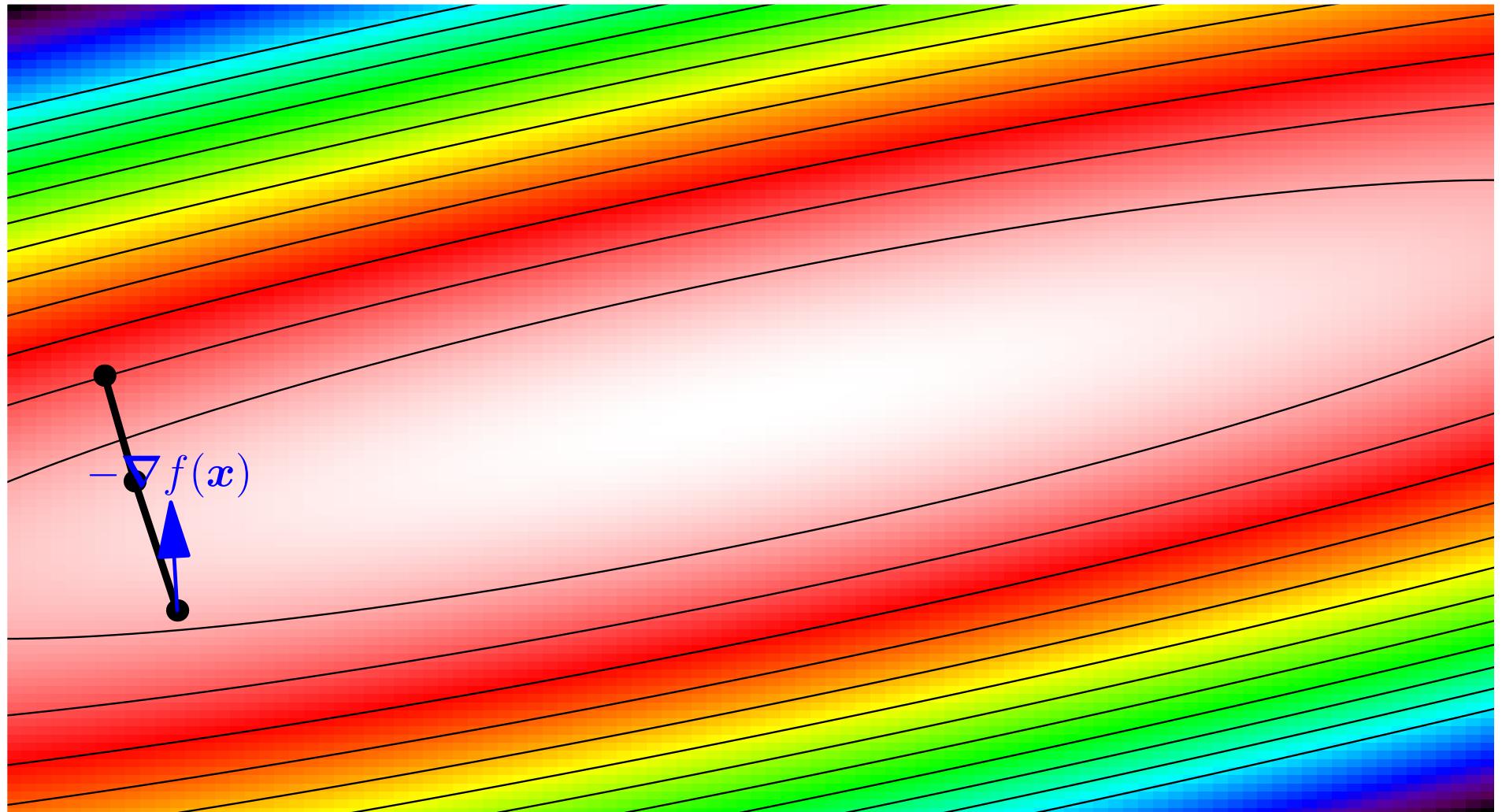
Gradient Descent with Momentum



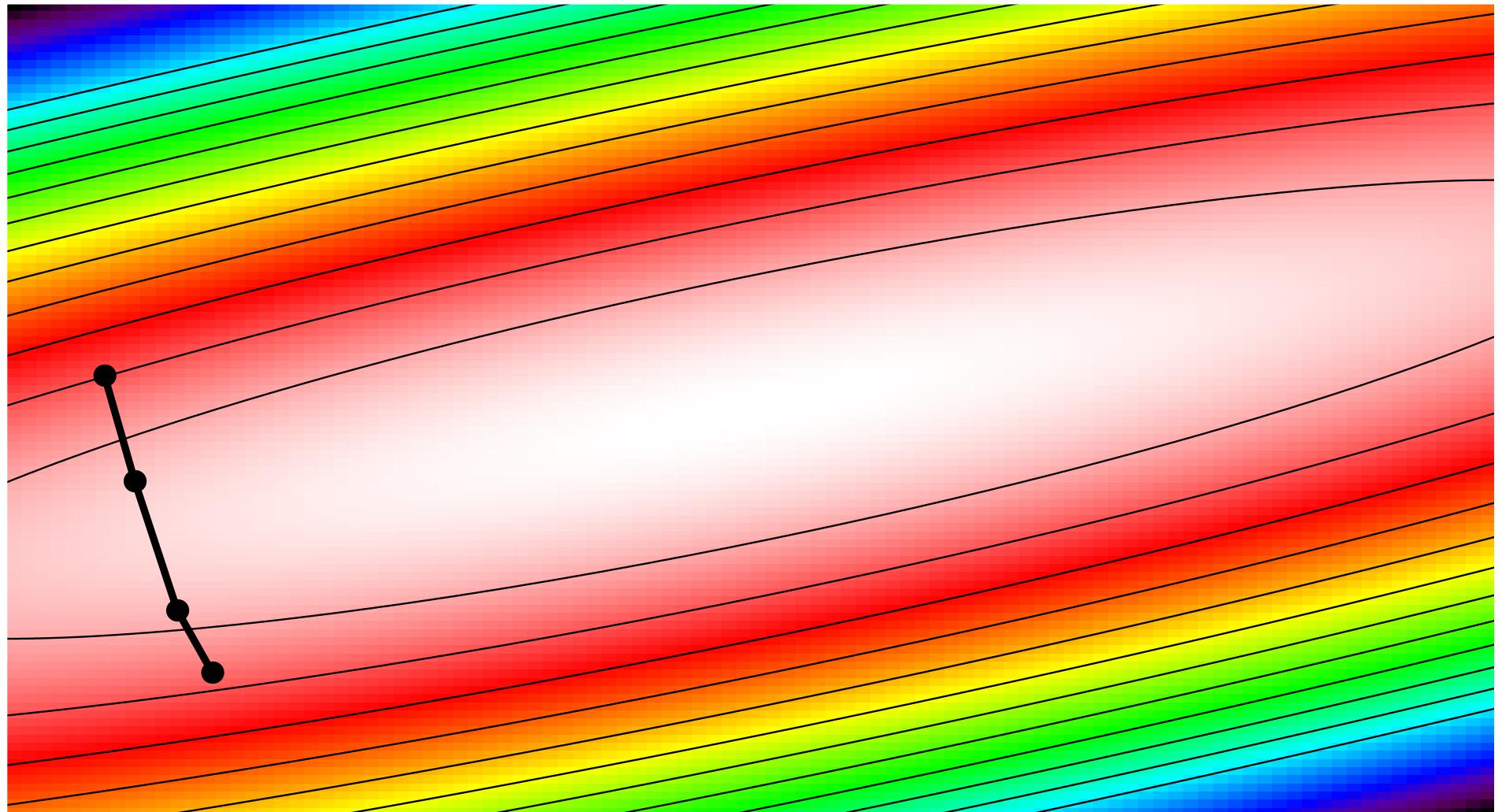
Gradient Descent with Momentum



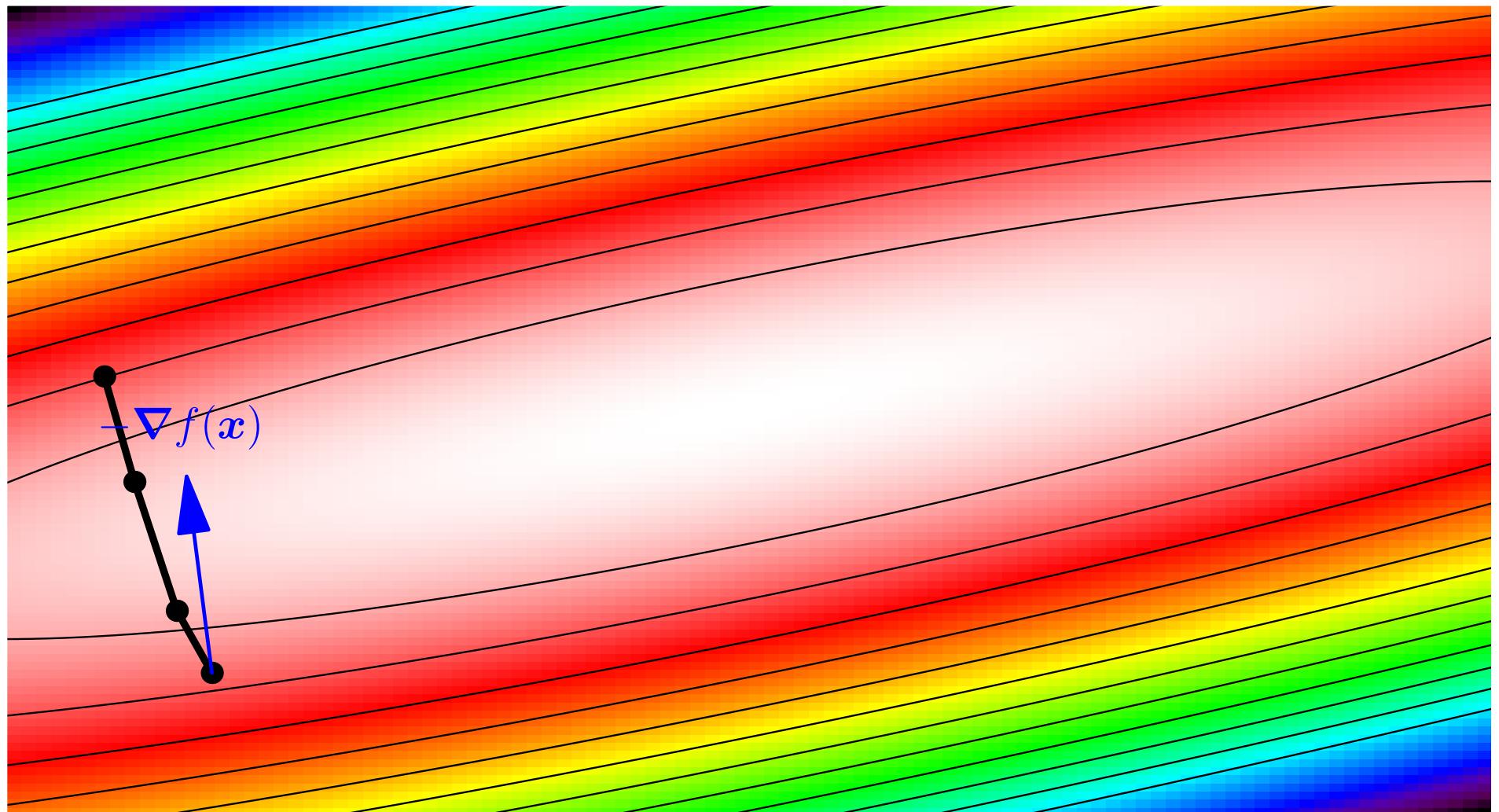
Gradient Descent with Momentum



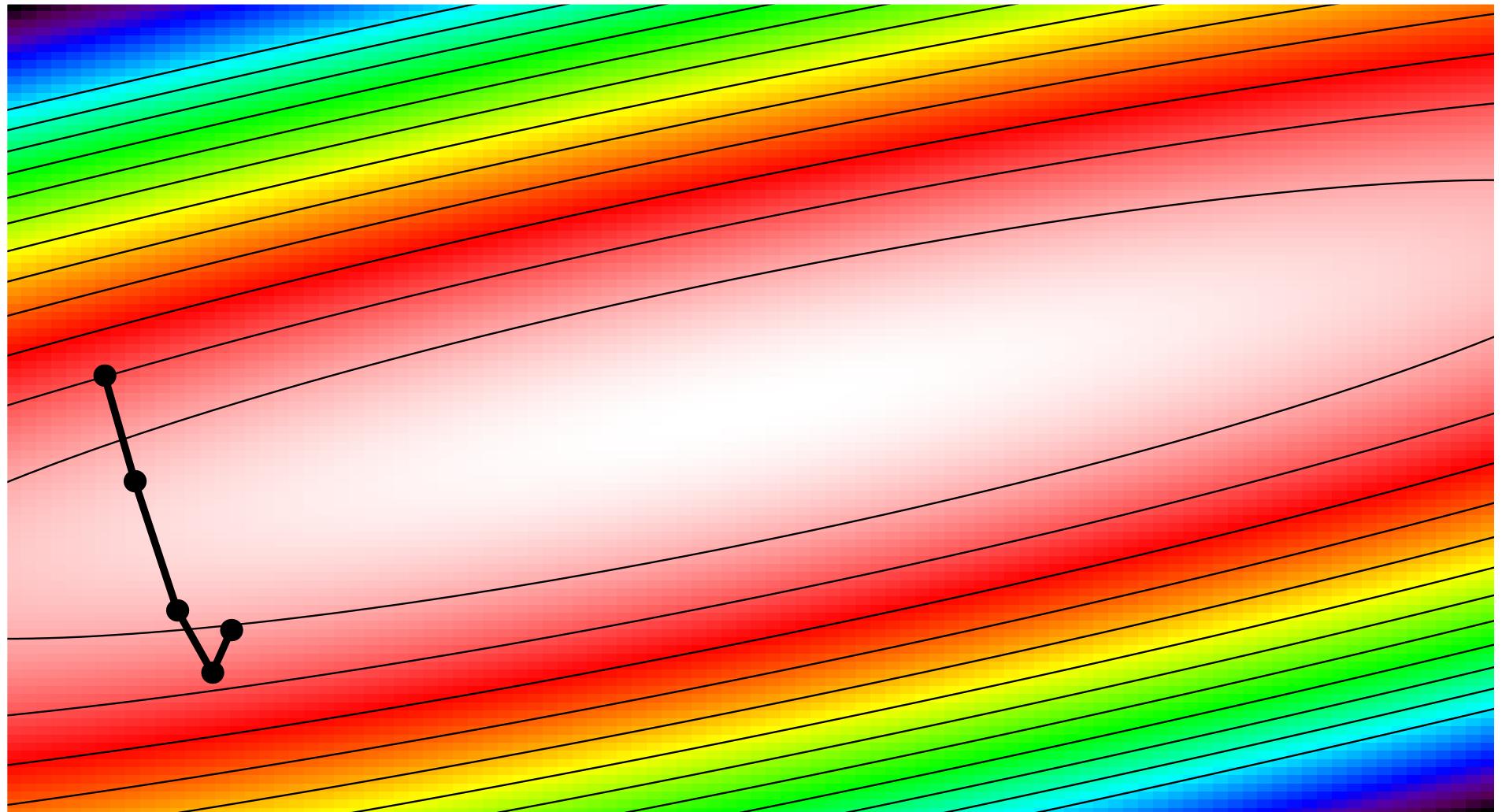
Gradient Descent with Momentum



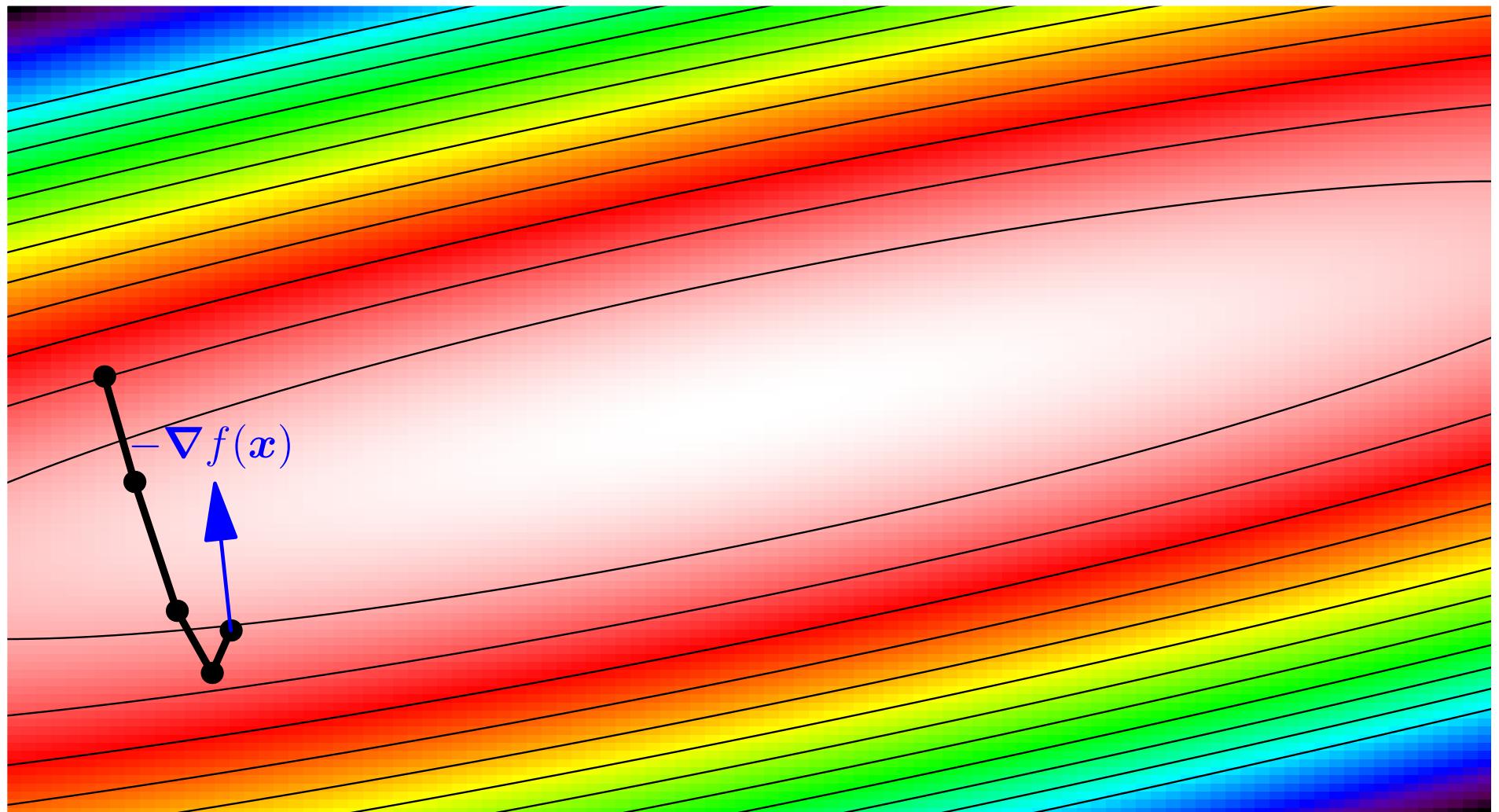
Gradient Descent with Momentum



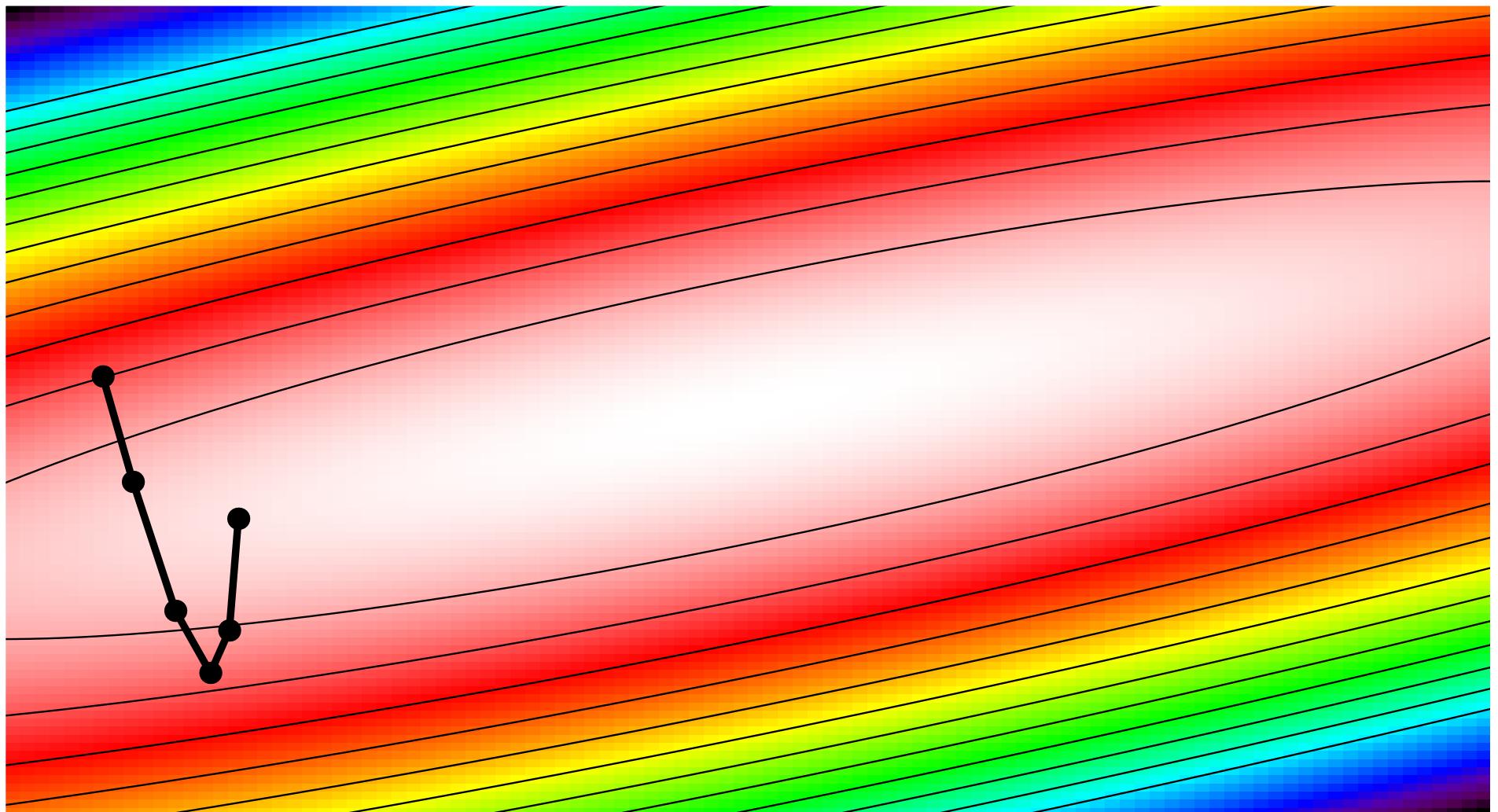
Gradient Descent with Momentum



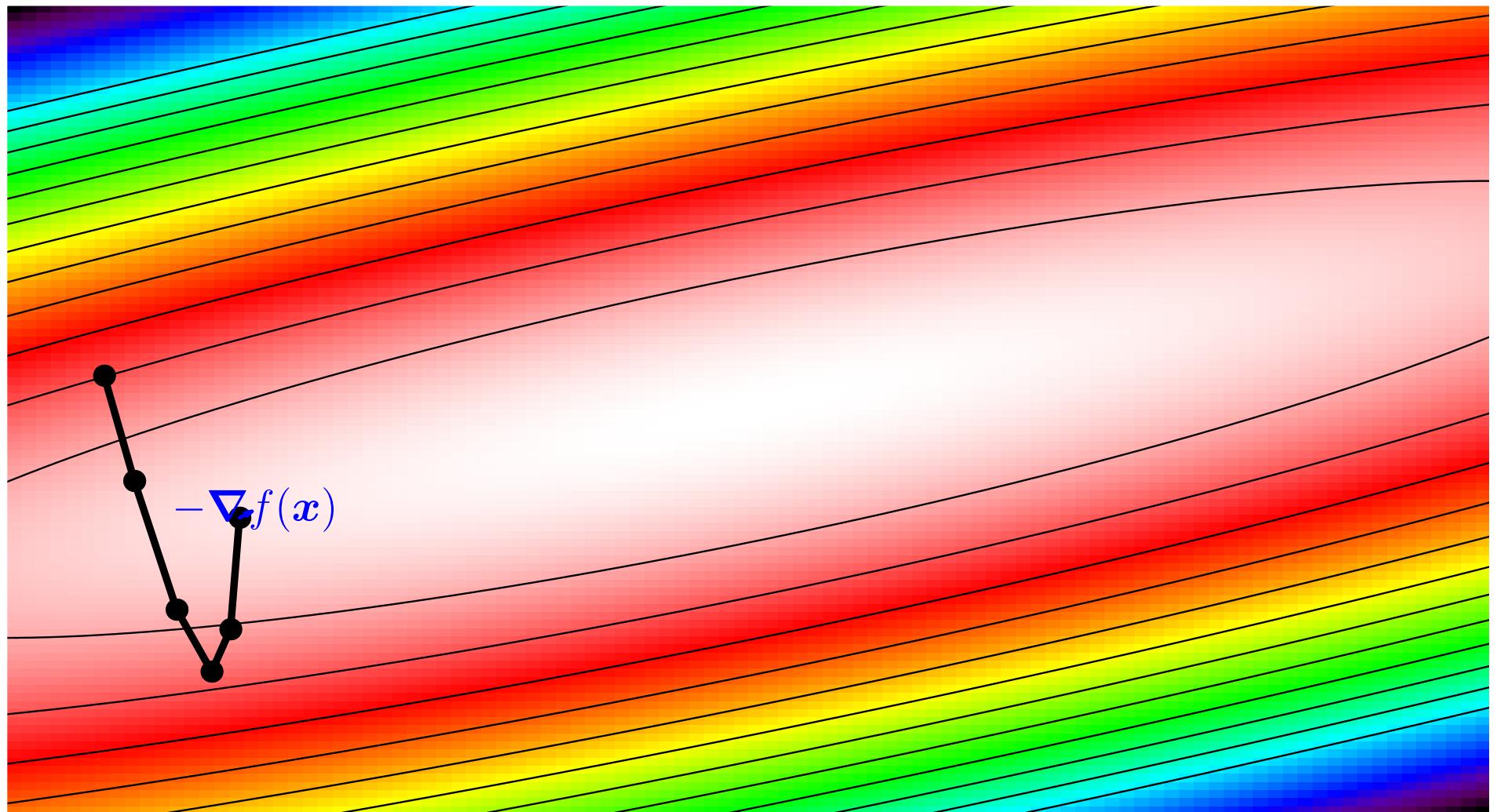
Gradient Descent with Momentum



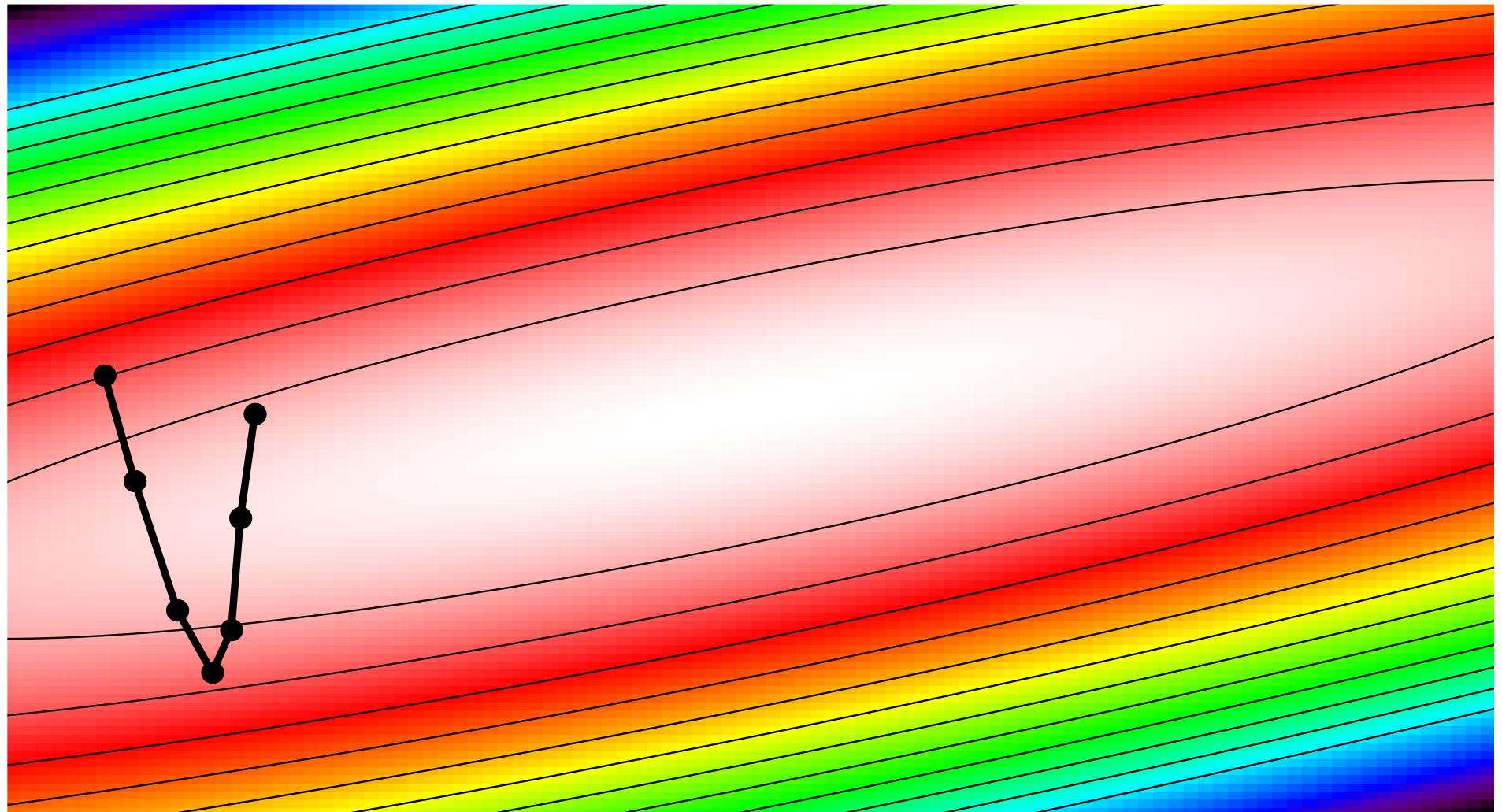
Gradient Descent with Momentum



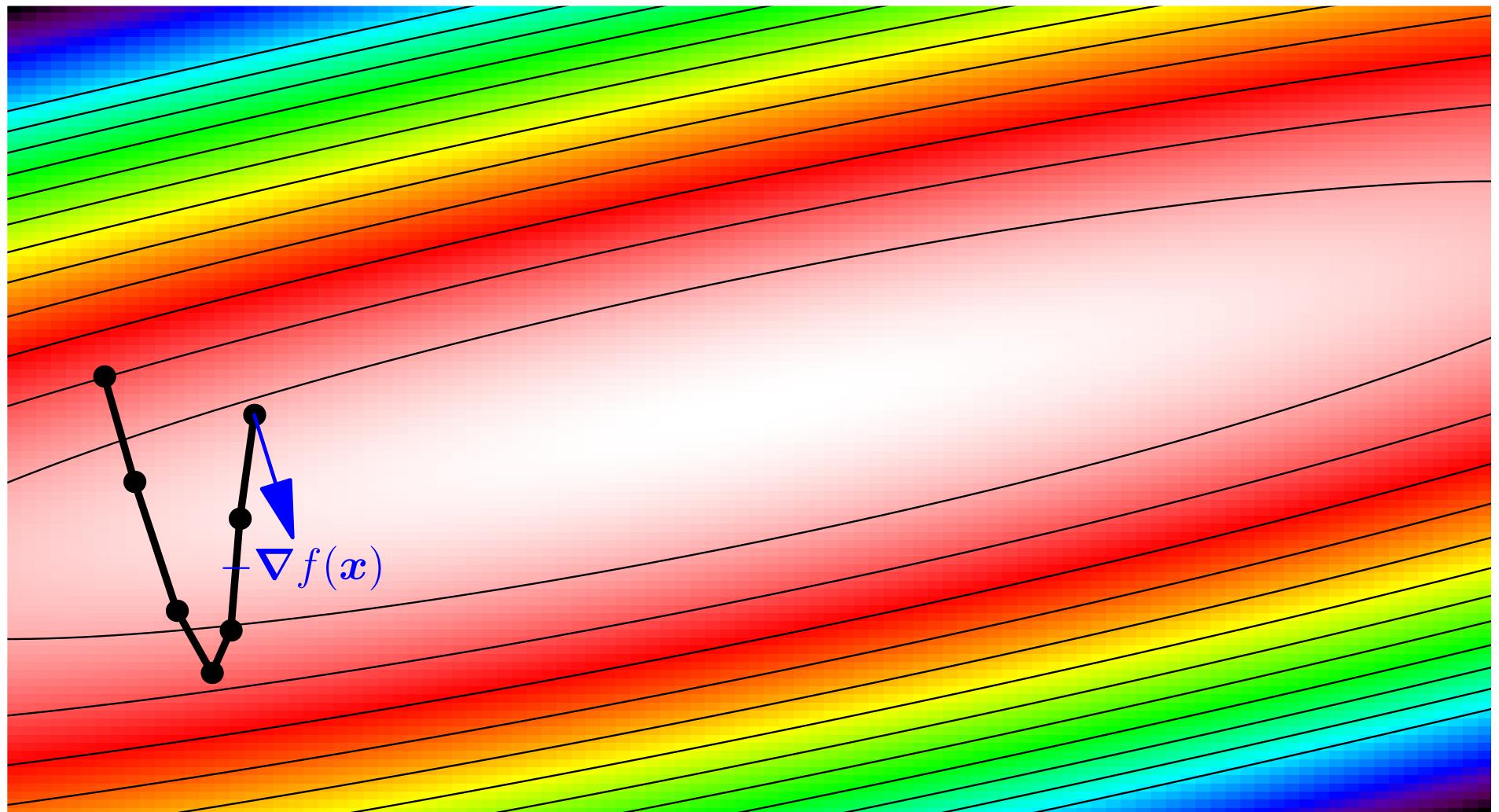
Gradient Descent with Momentum



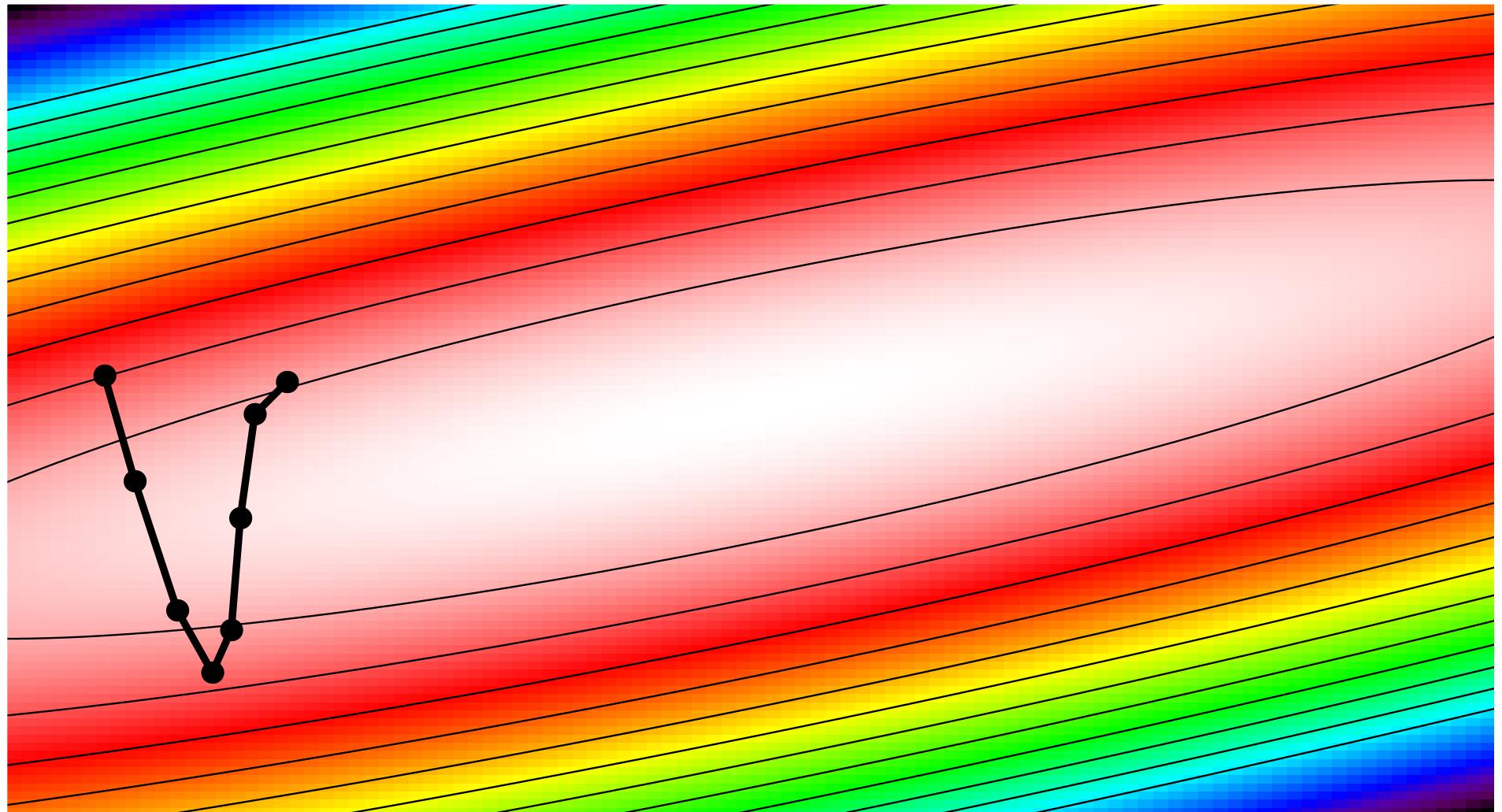
Gradient Descent with Momentum



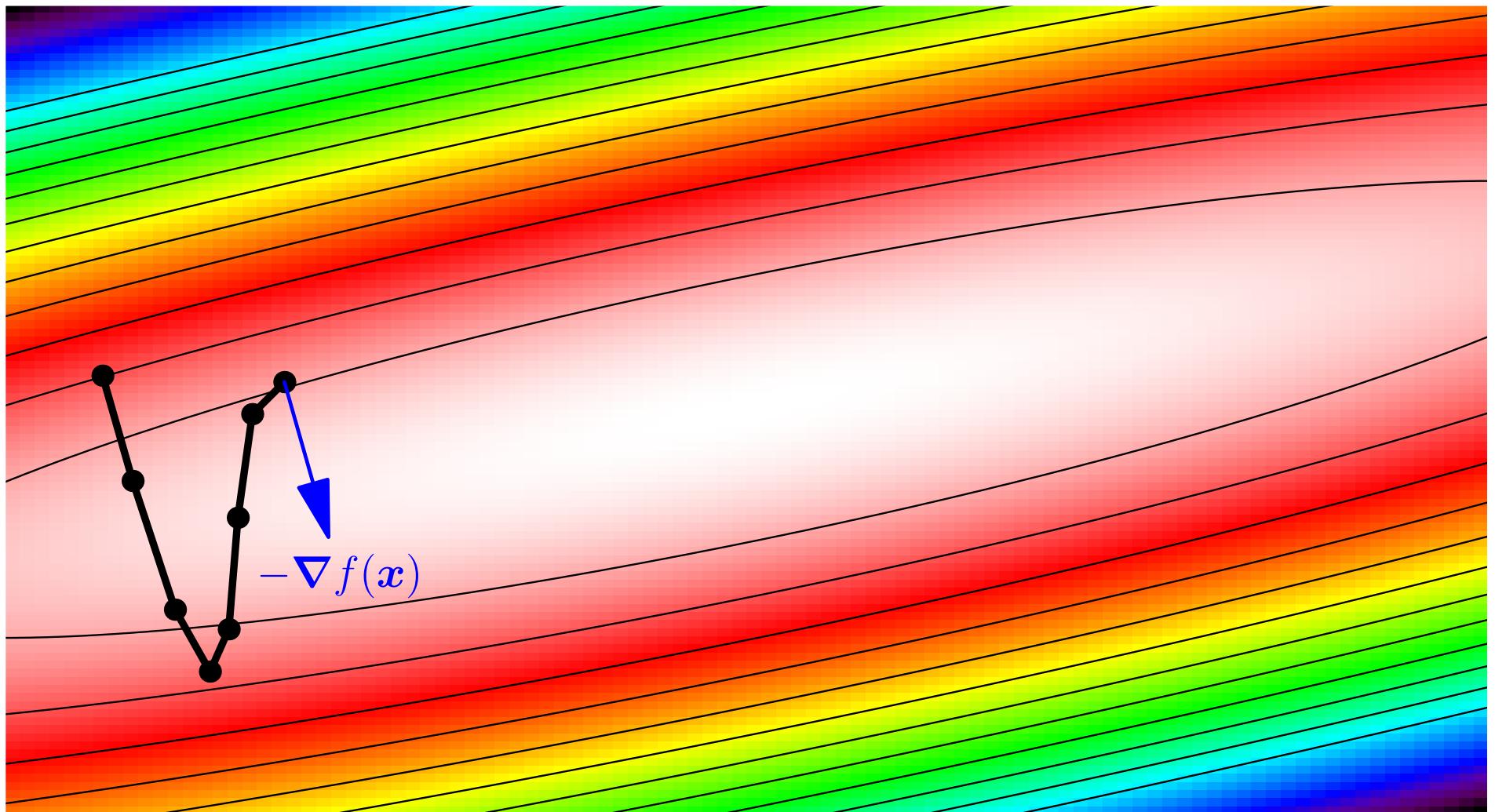
Gradient Descent with Momentum



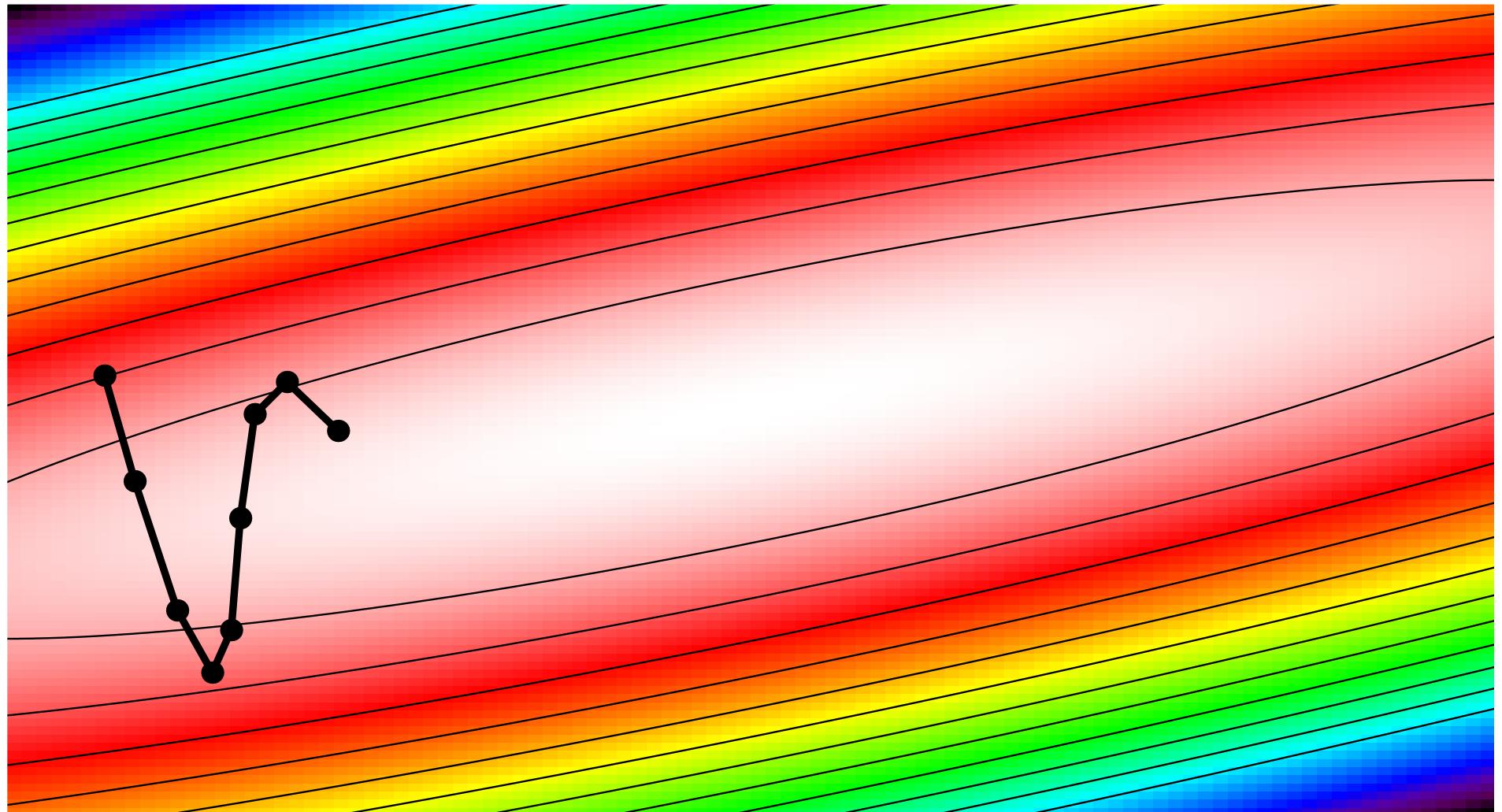
Gradient Descent with Momentum



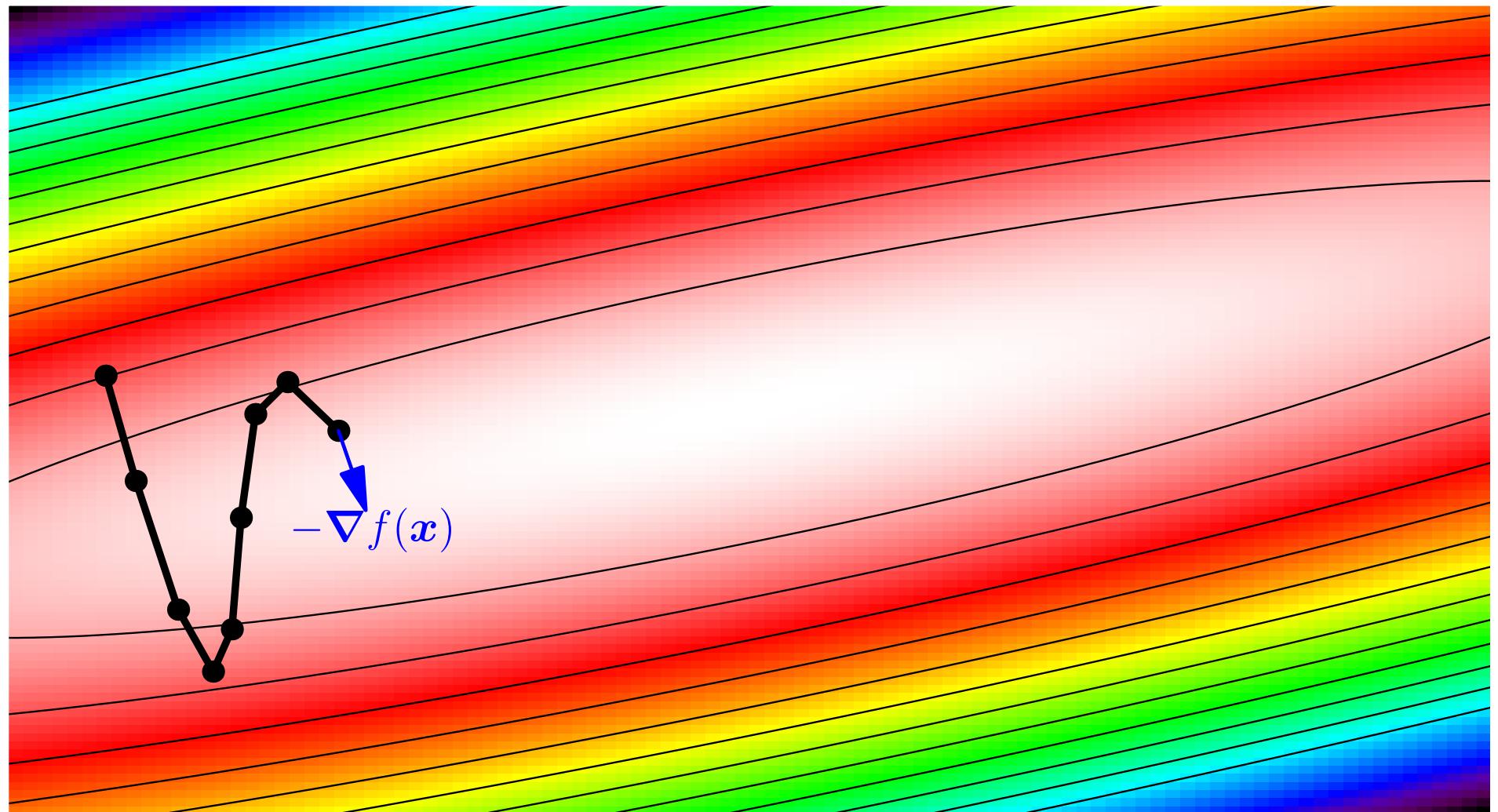
Gradient Descent with Momentum



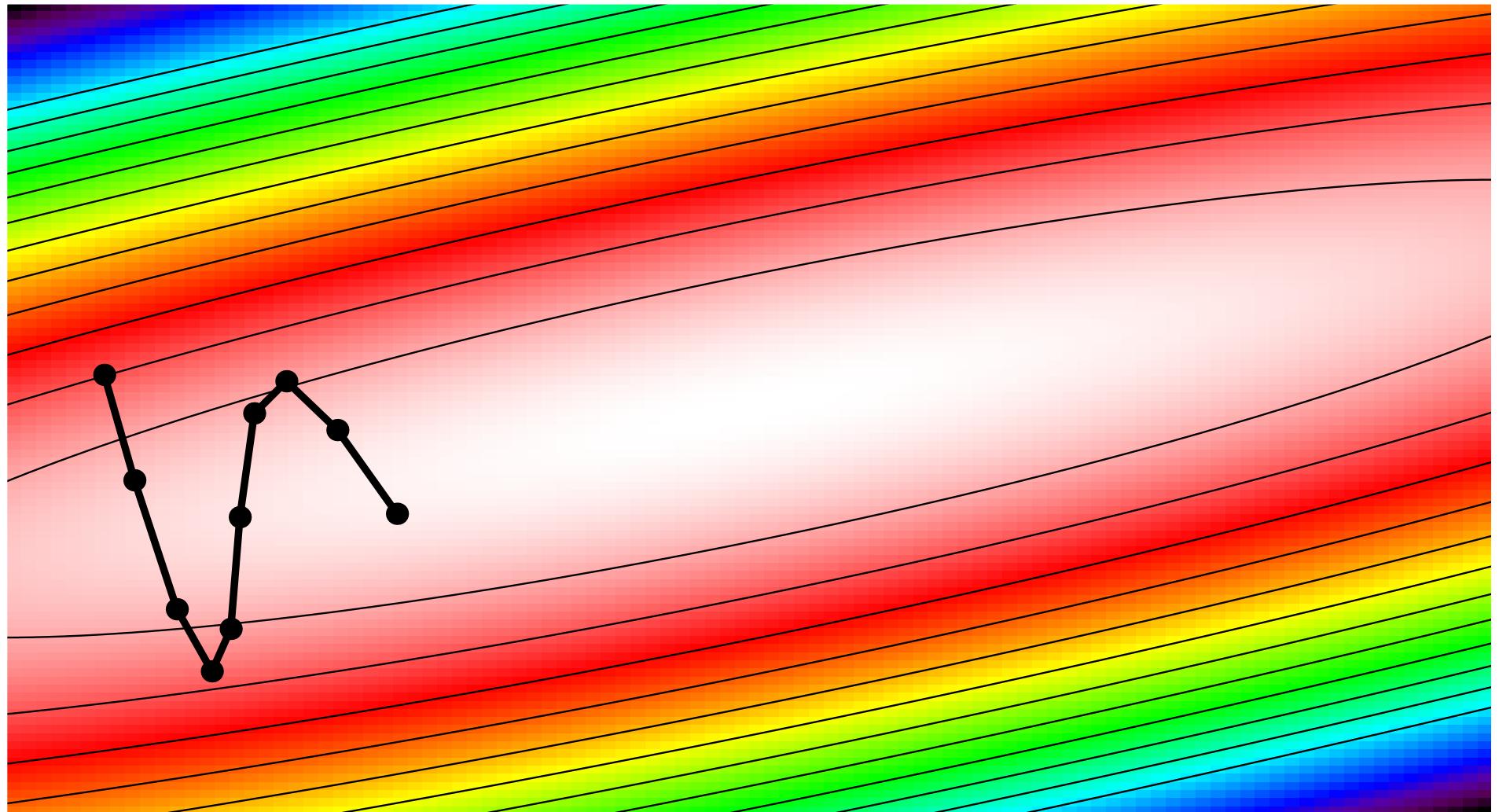
Gradient Descent with Momentum



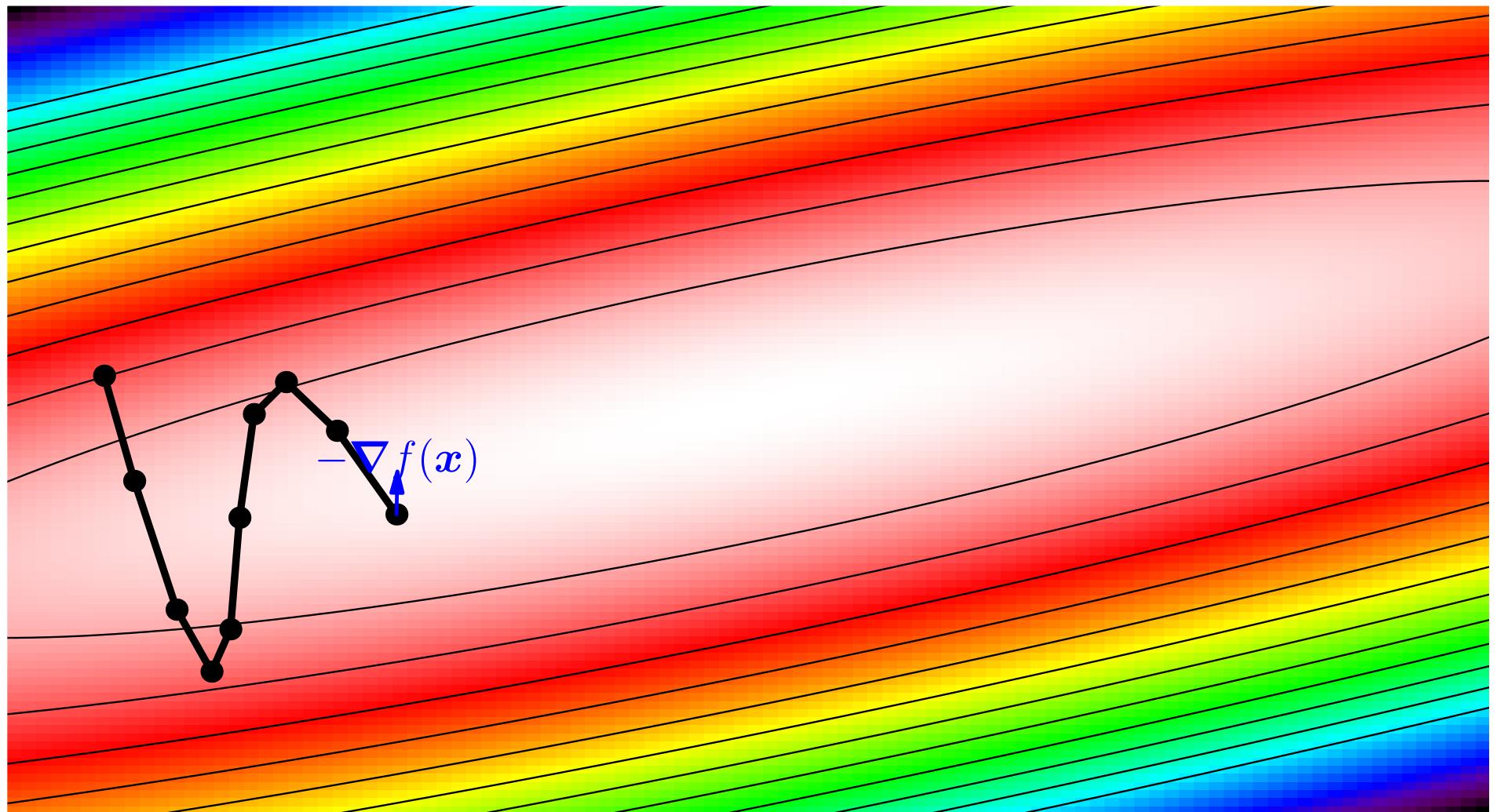
Gradient Descent with Momentum



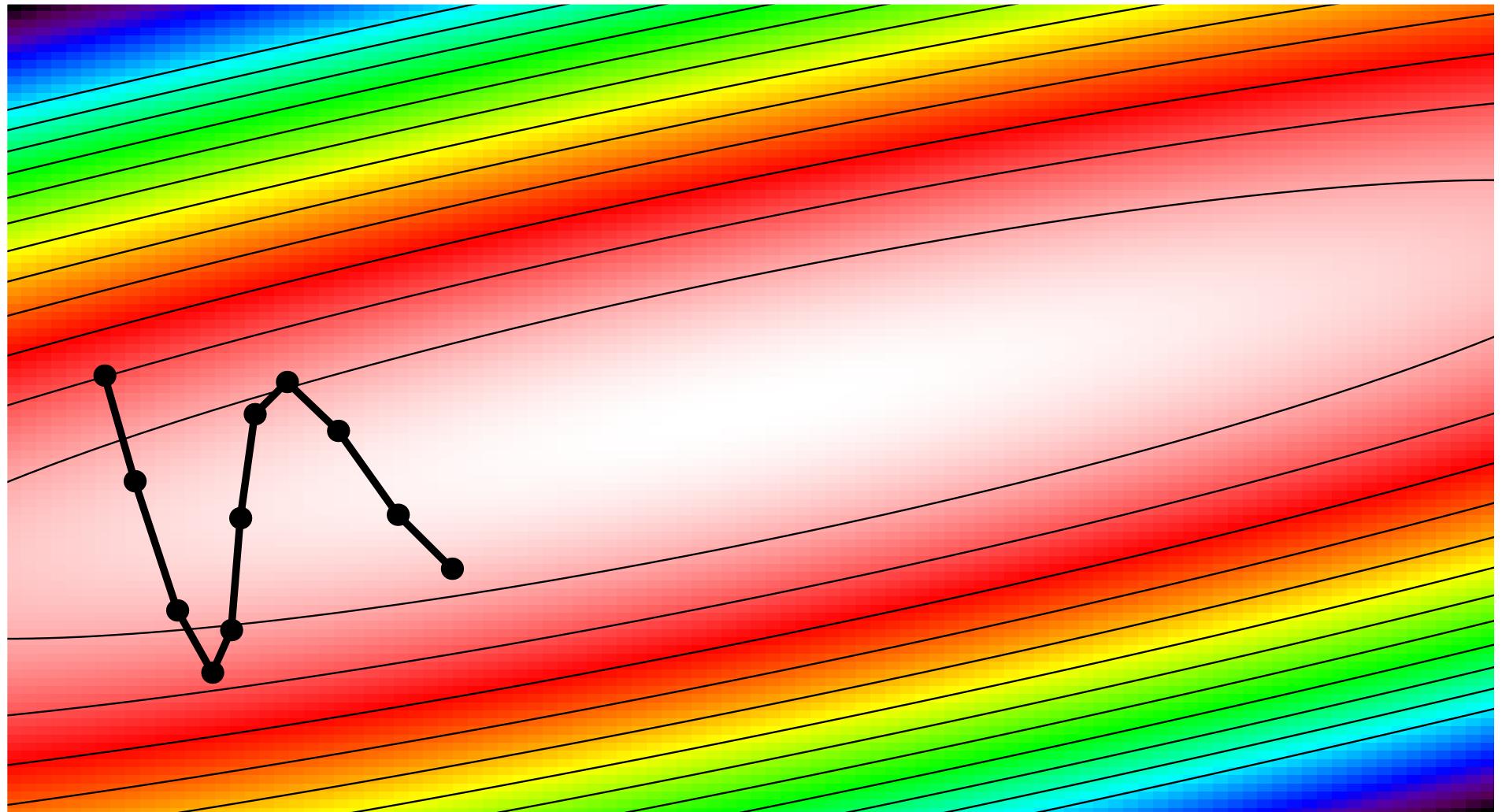
Gradient Descent with Momentum



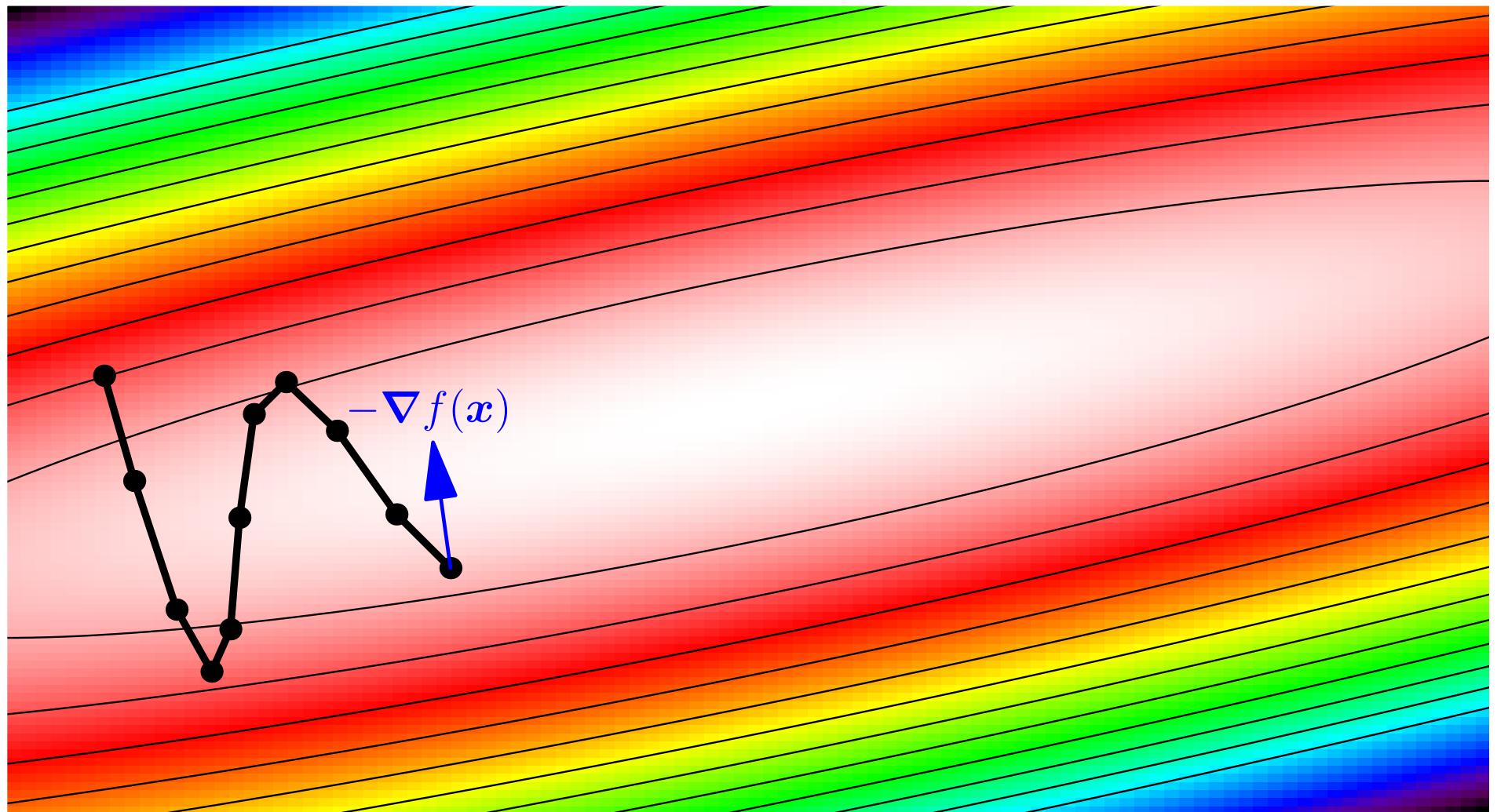
Gradient Descent with Momentum



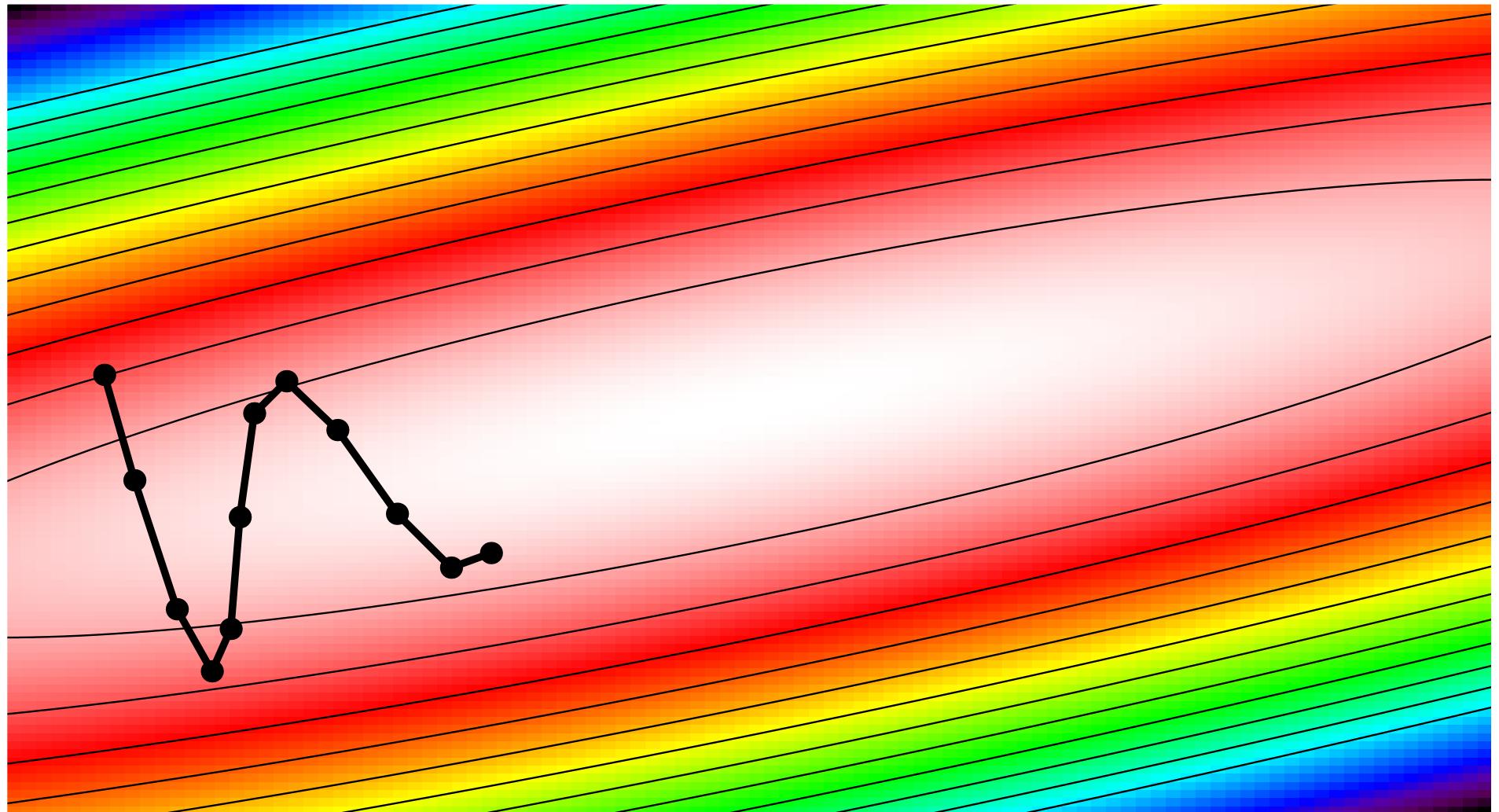
Gradient Descent with Momentum



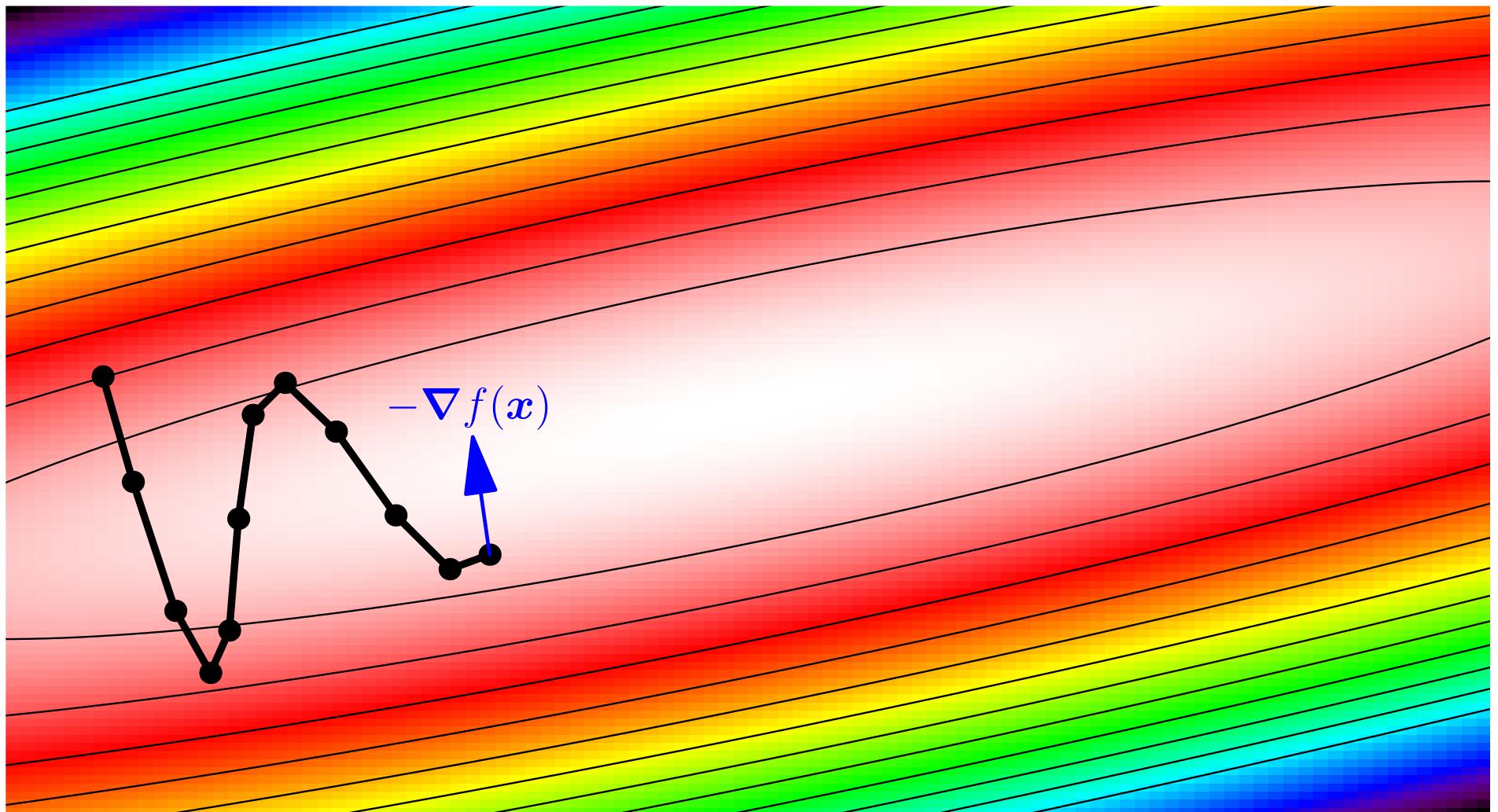
Gradient Descent with Momentum



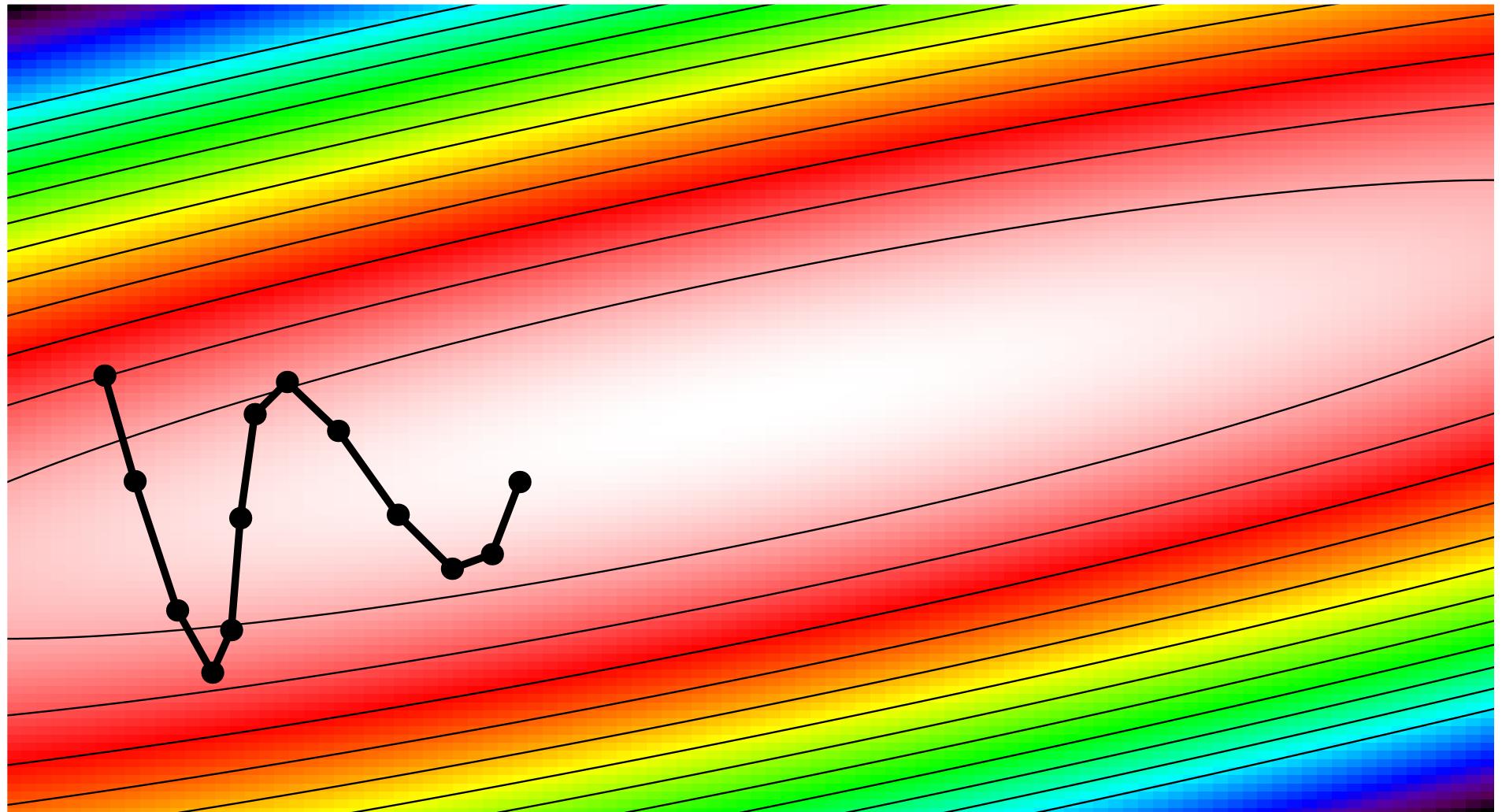
Gradient Descent with Momentum



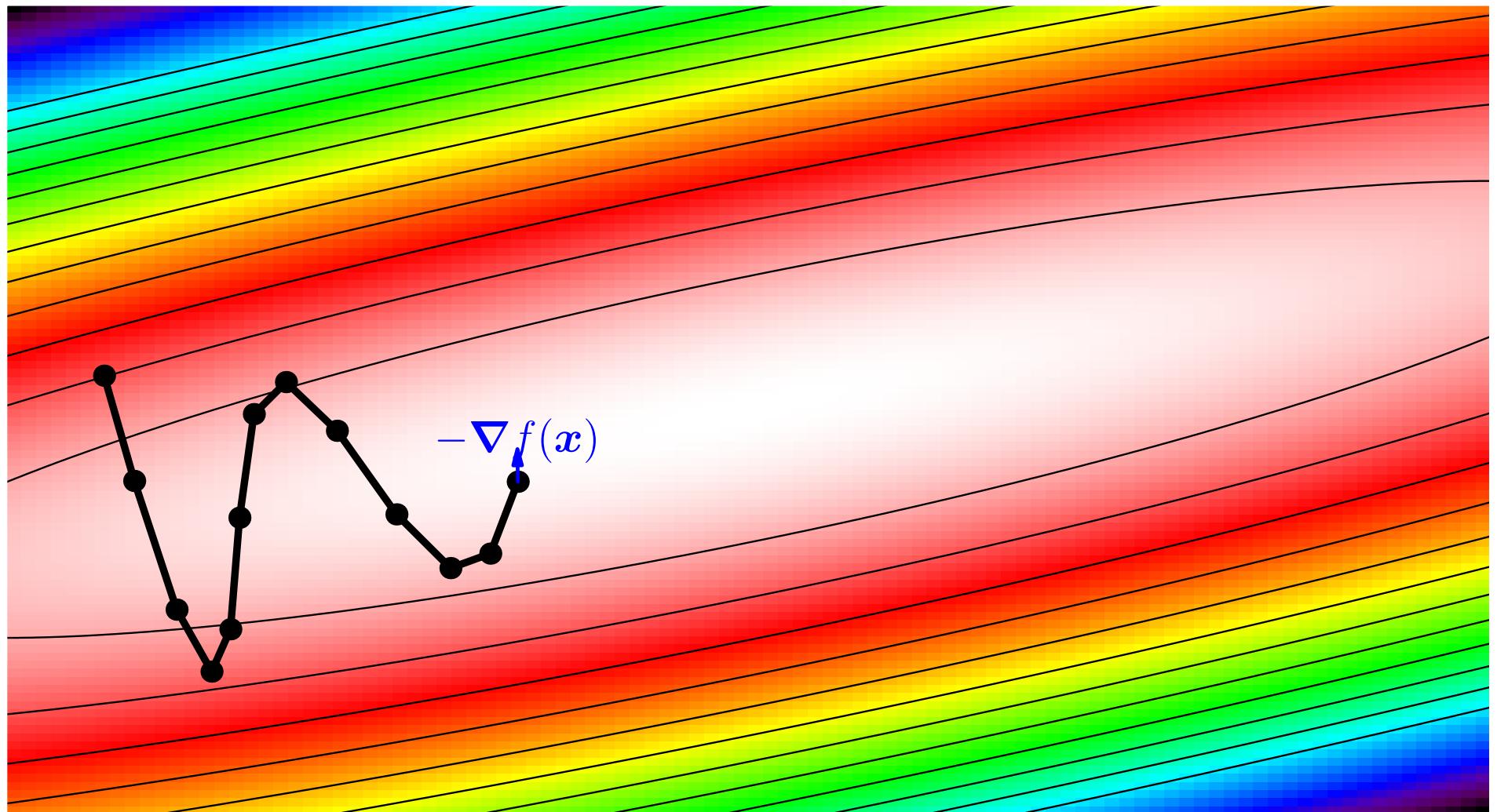
Gradient Descent with Momentum



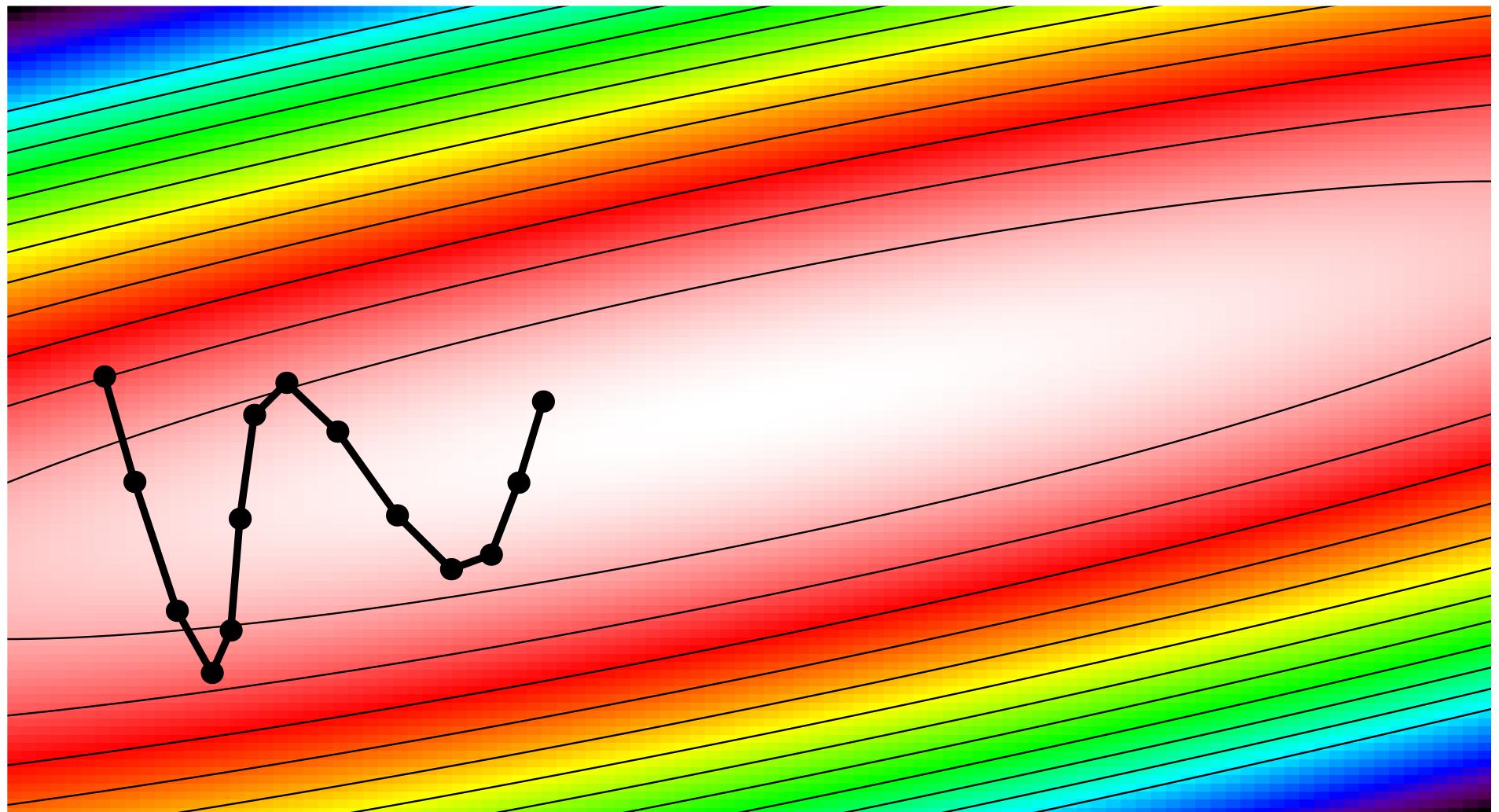
Gradient Descent with Momentum



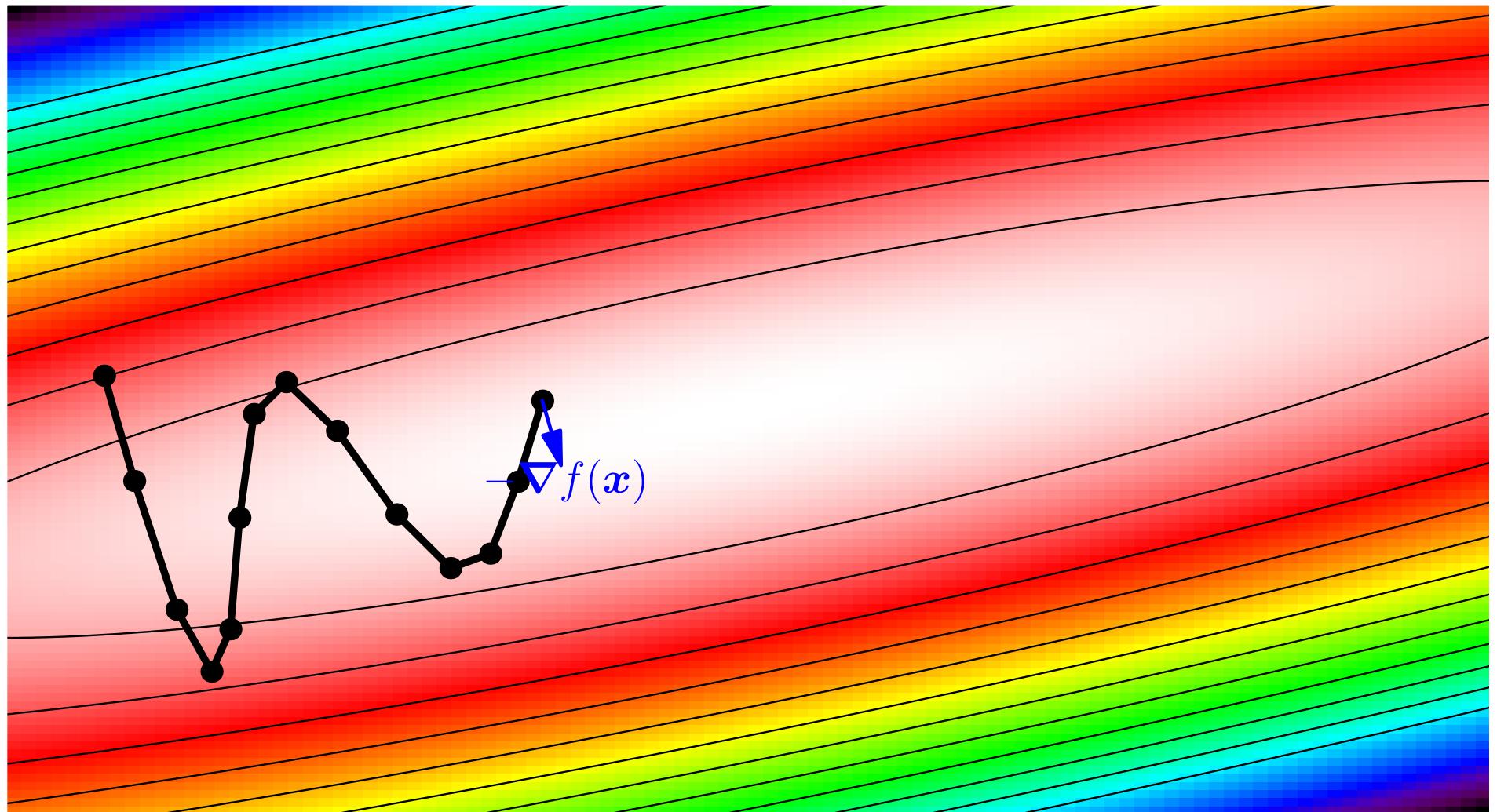
Gradient Descent with Momentum



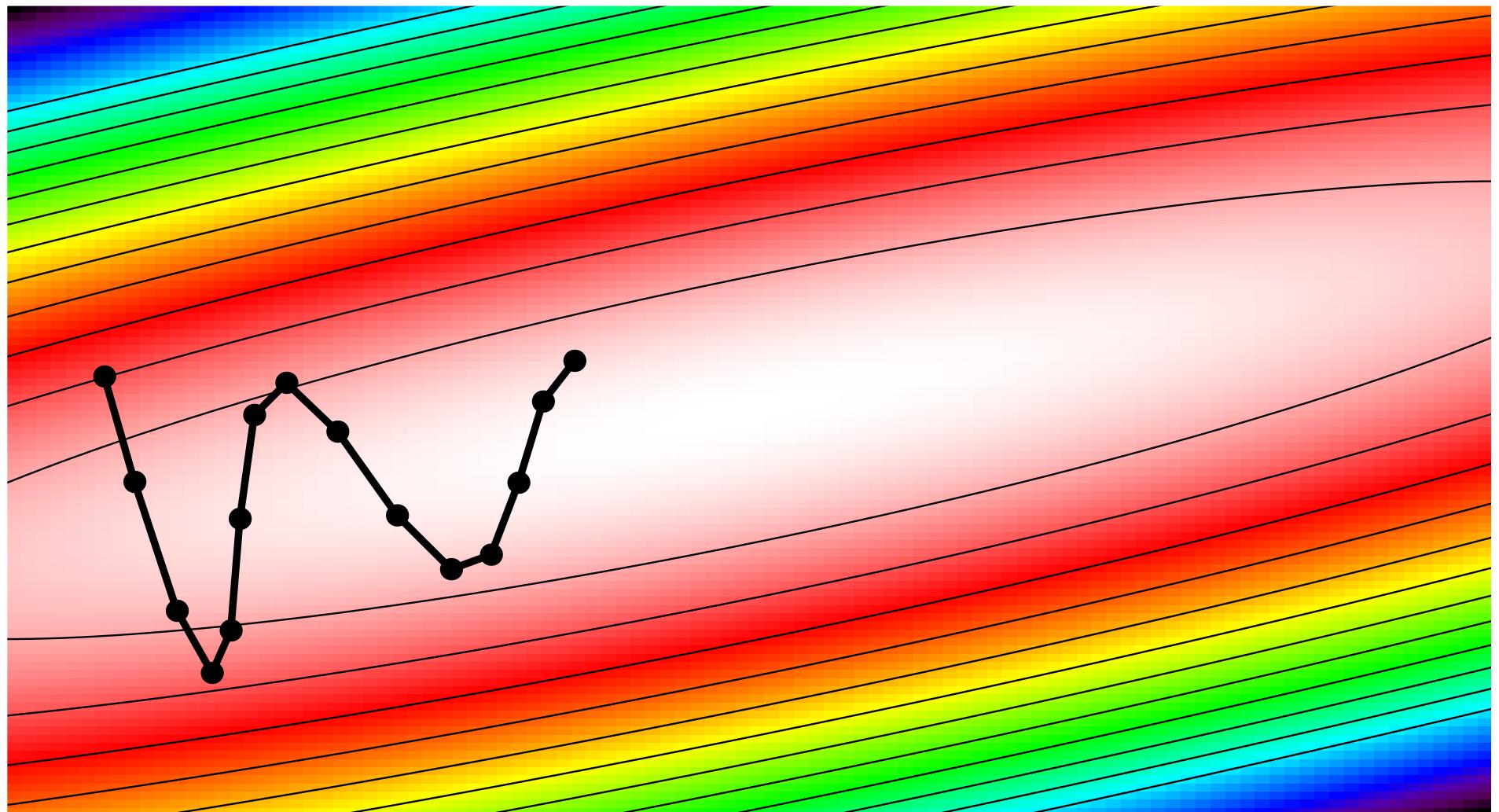
Gradient Descent with Momentum



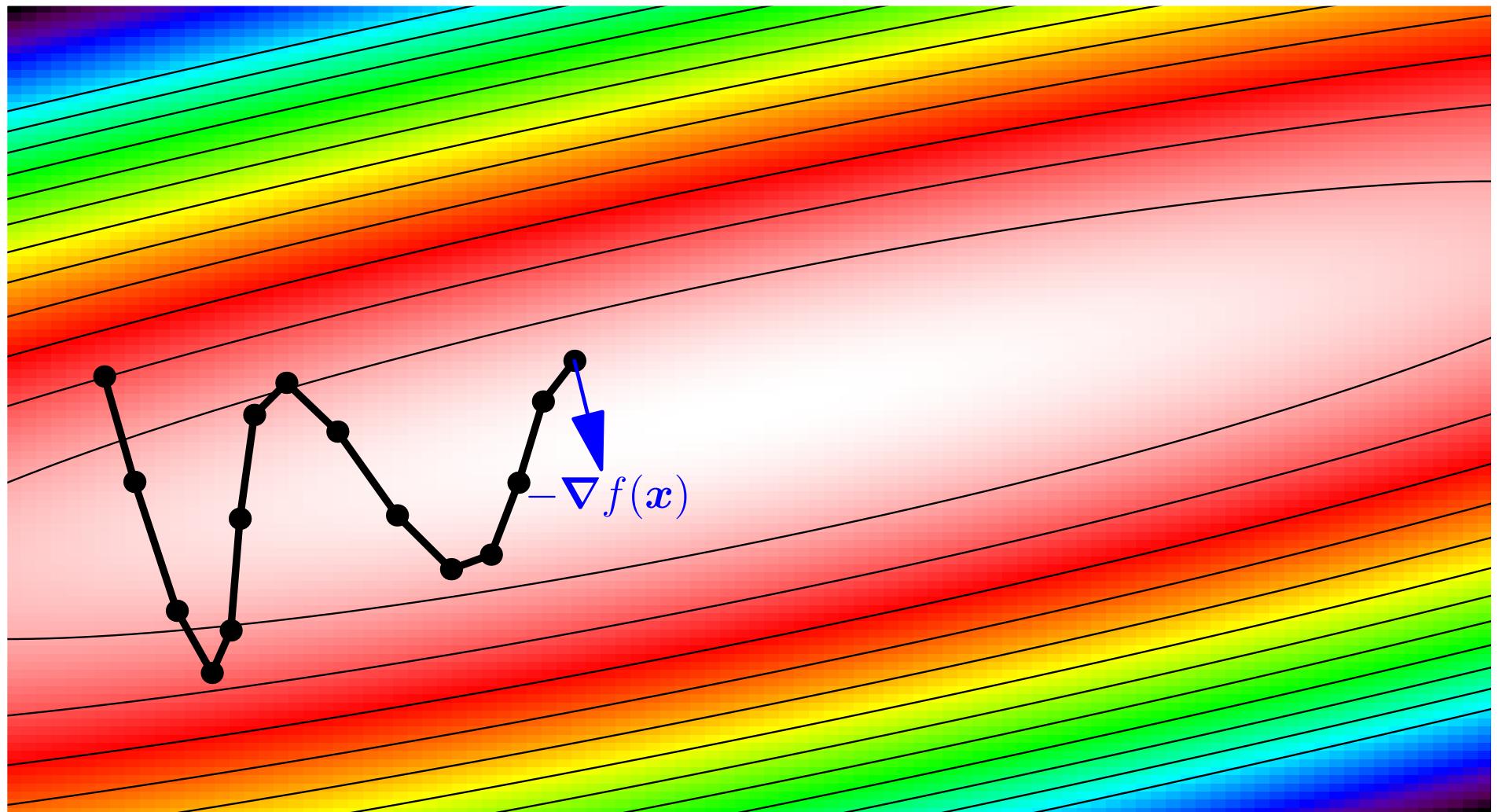
Gradient Descent with Momentum



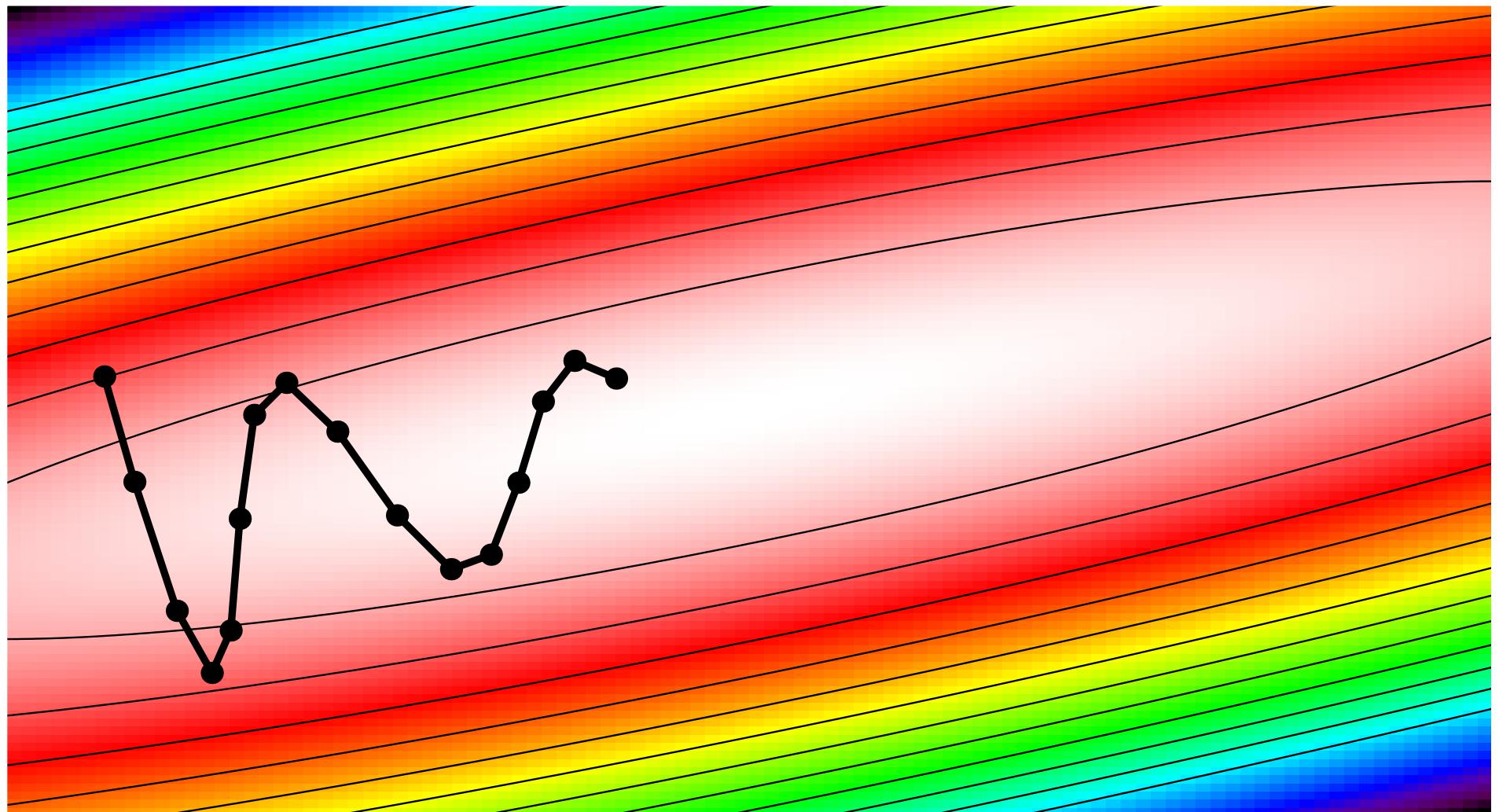
Gradient Descent with Momentum



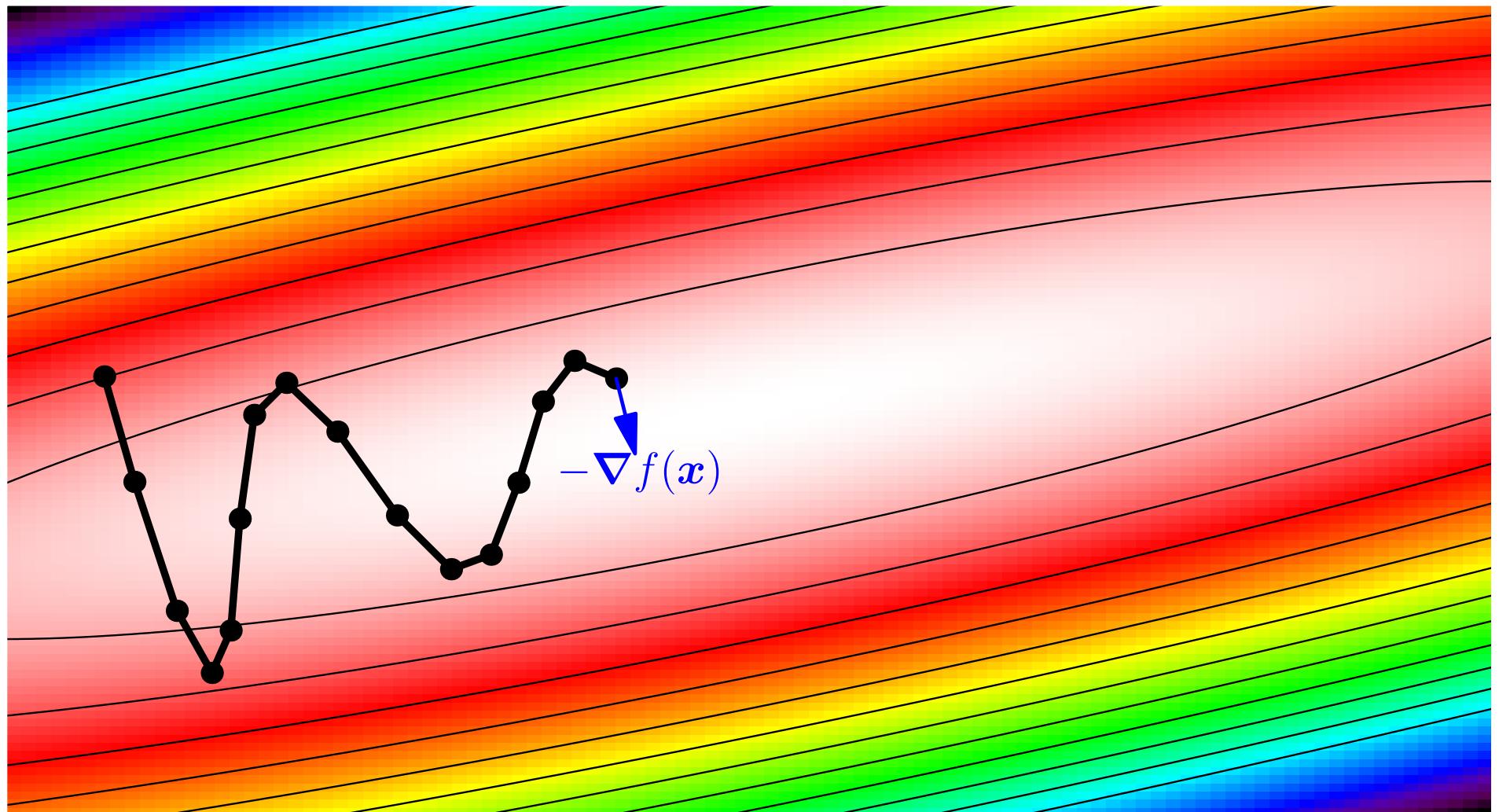
Gradient Descent with Momentum



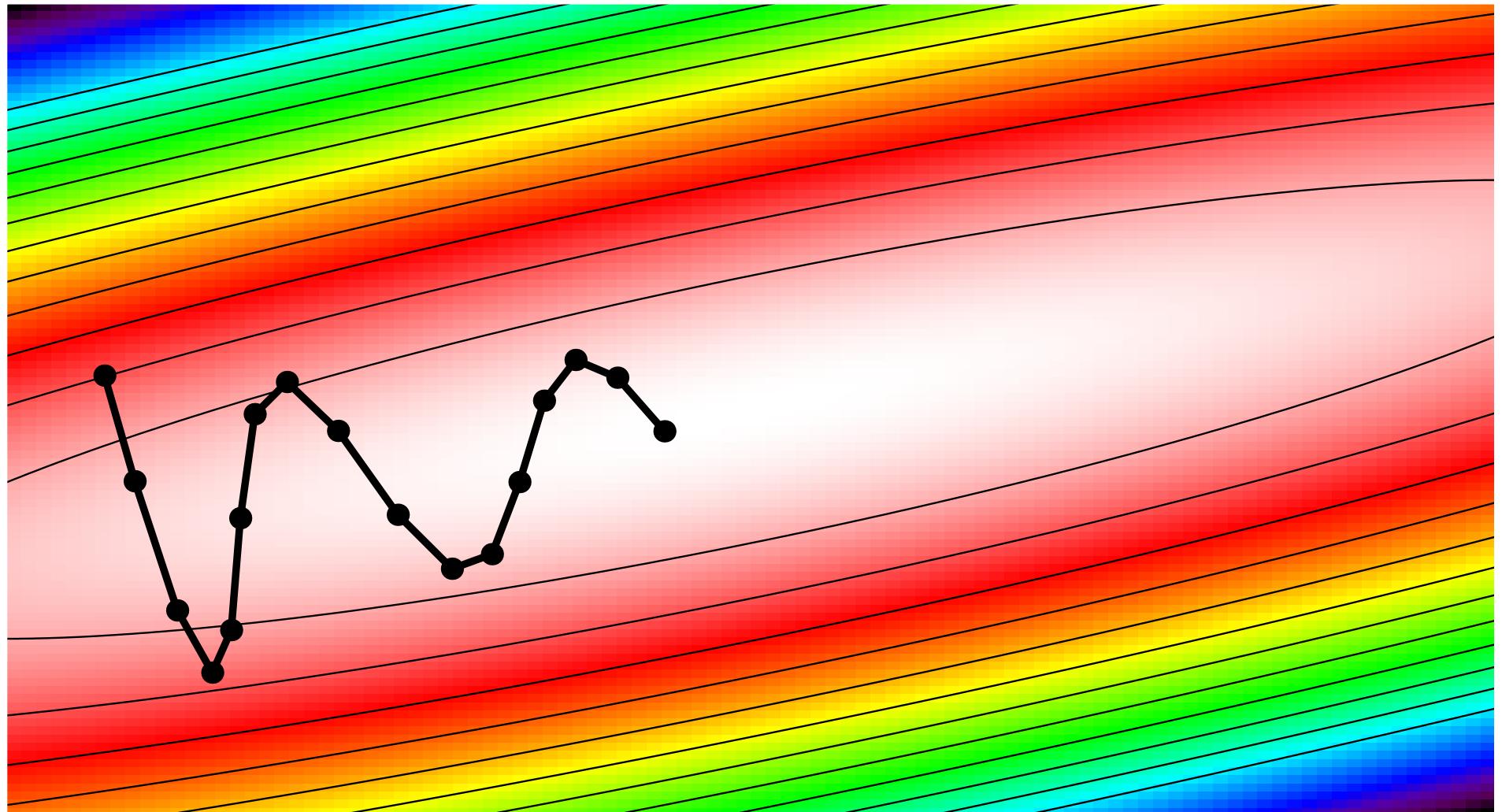
Gradient Descent with Momentum



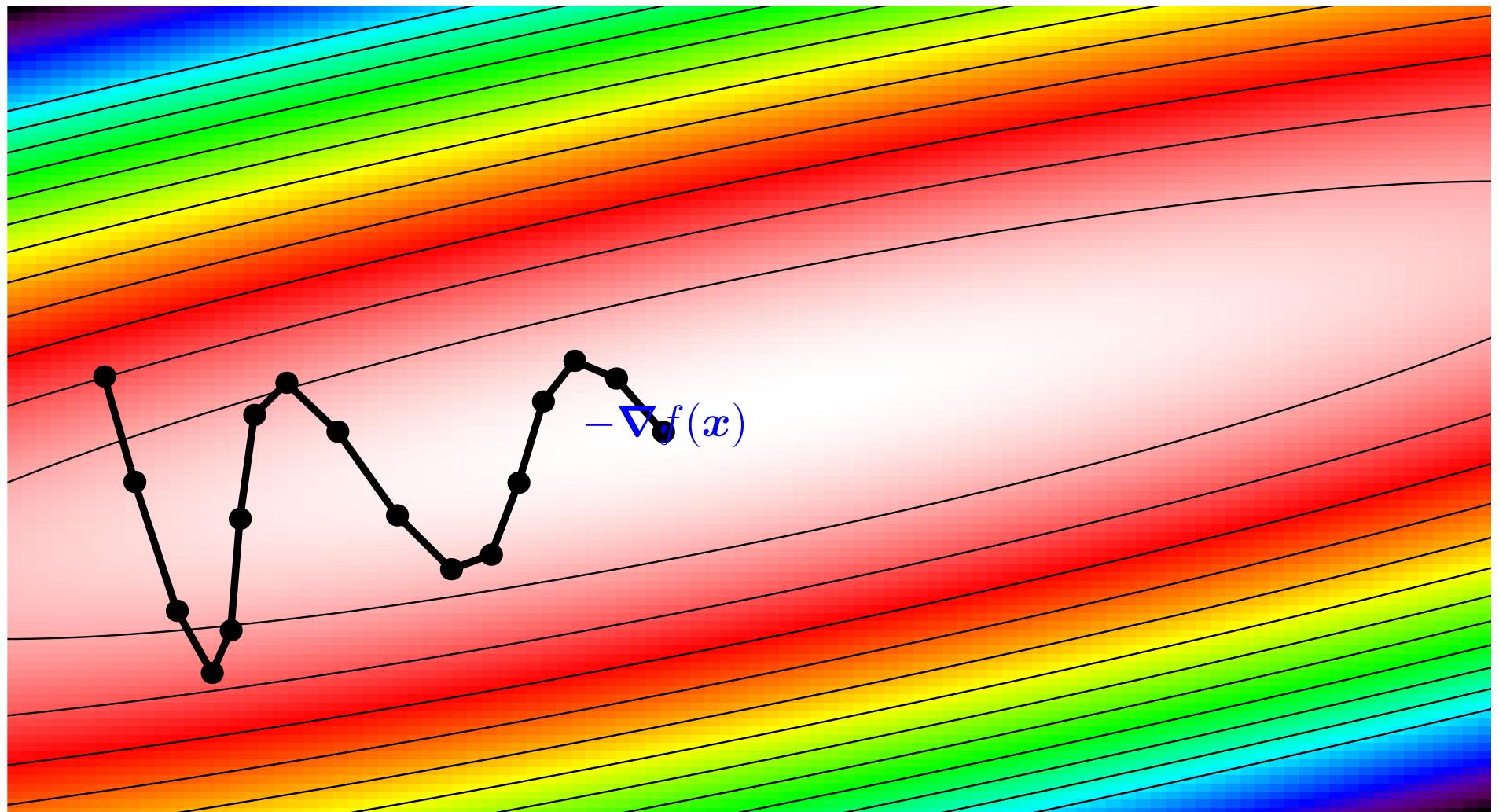
Gradient Descent with Momentum



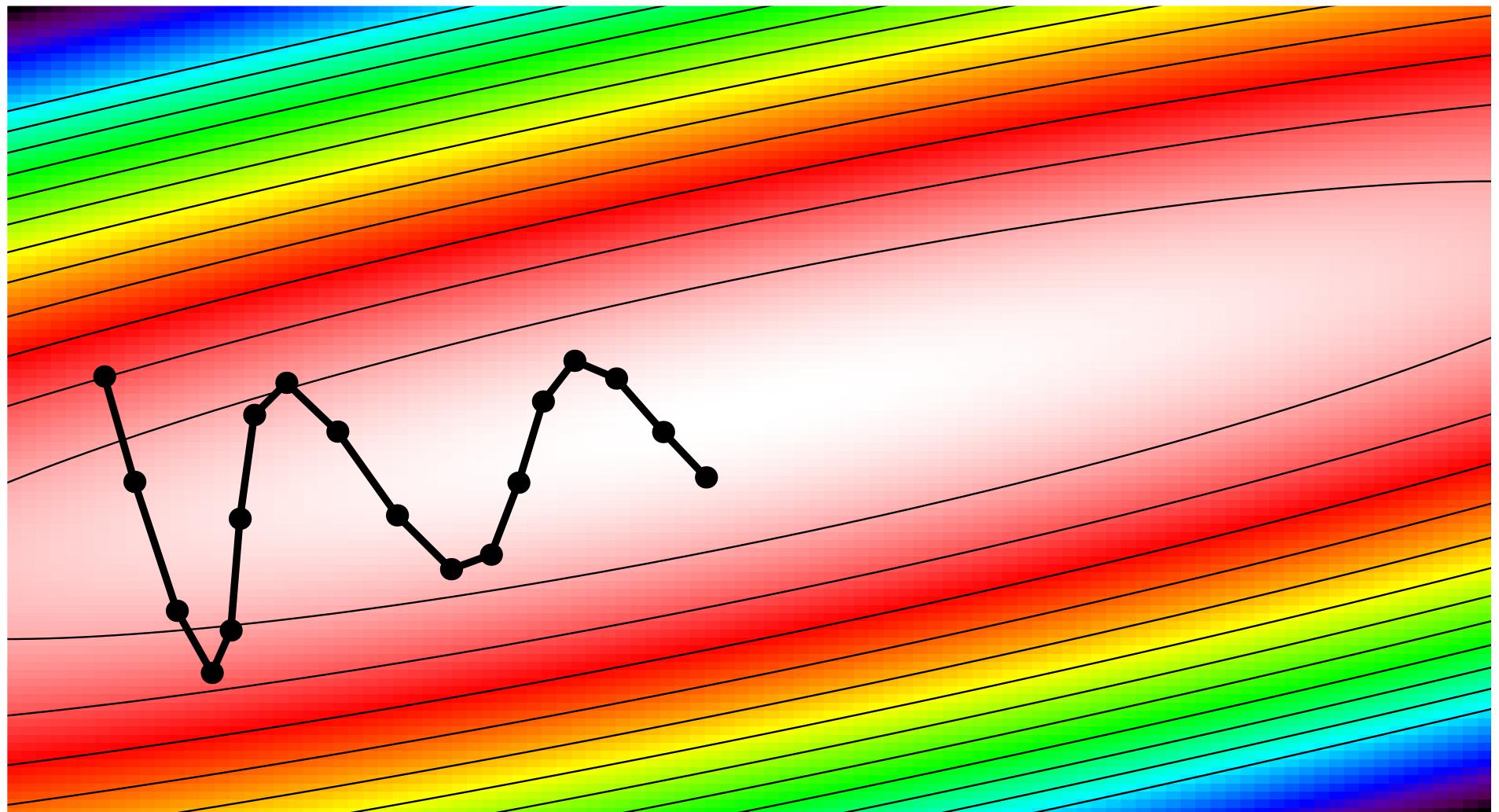
Gradient Descent with Momentum



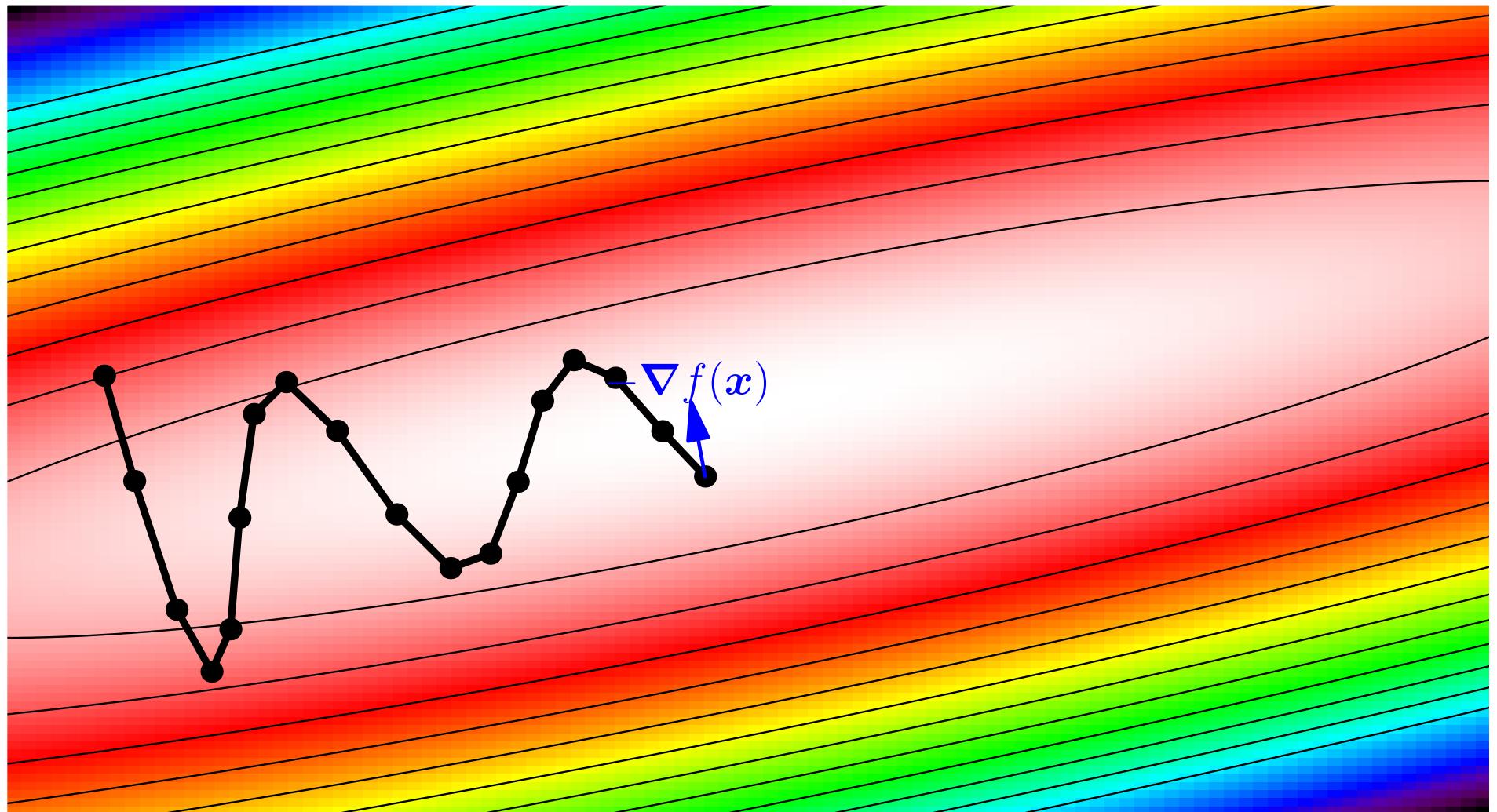
Gradient Descent with Momentum



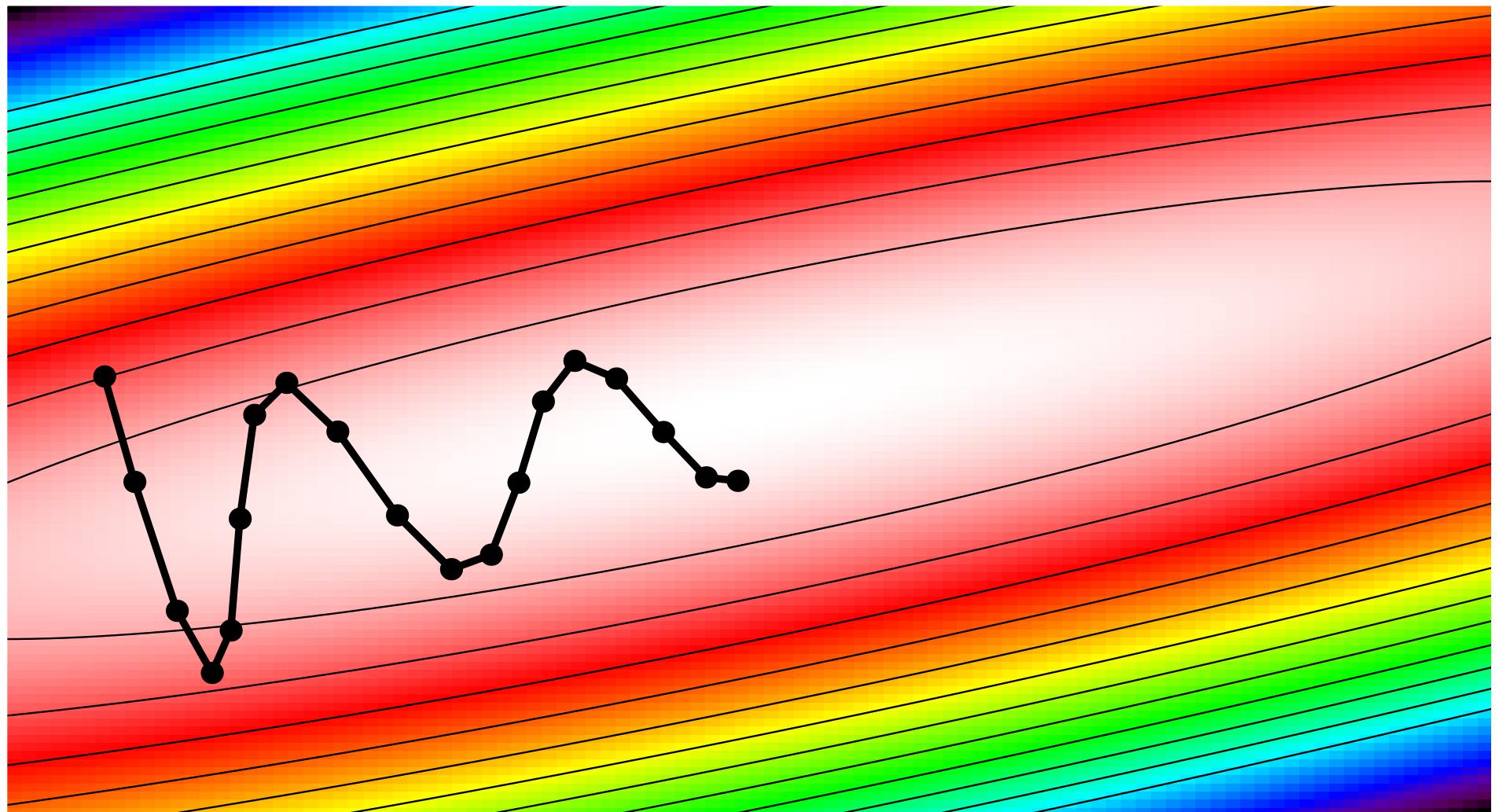
Gradient Descent with Momentum



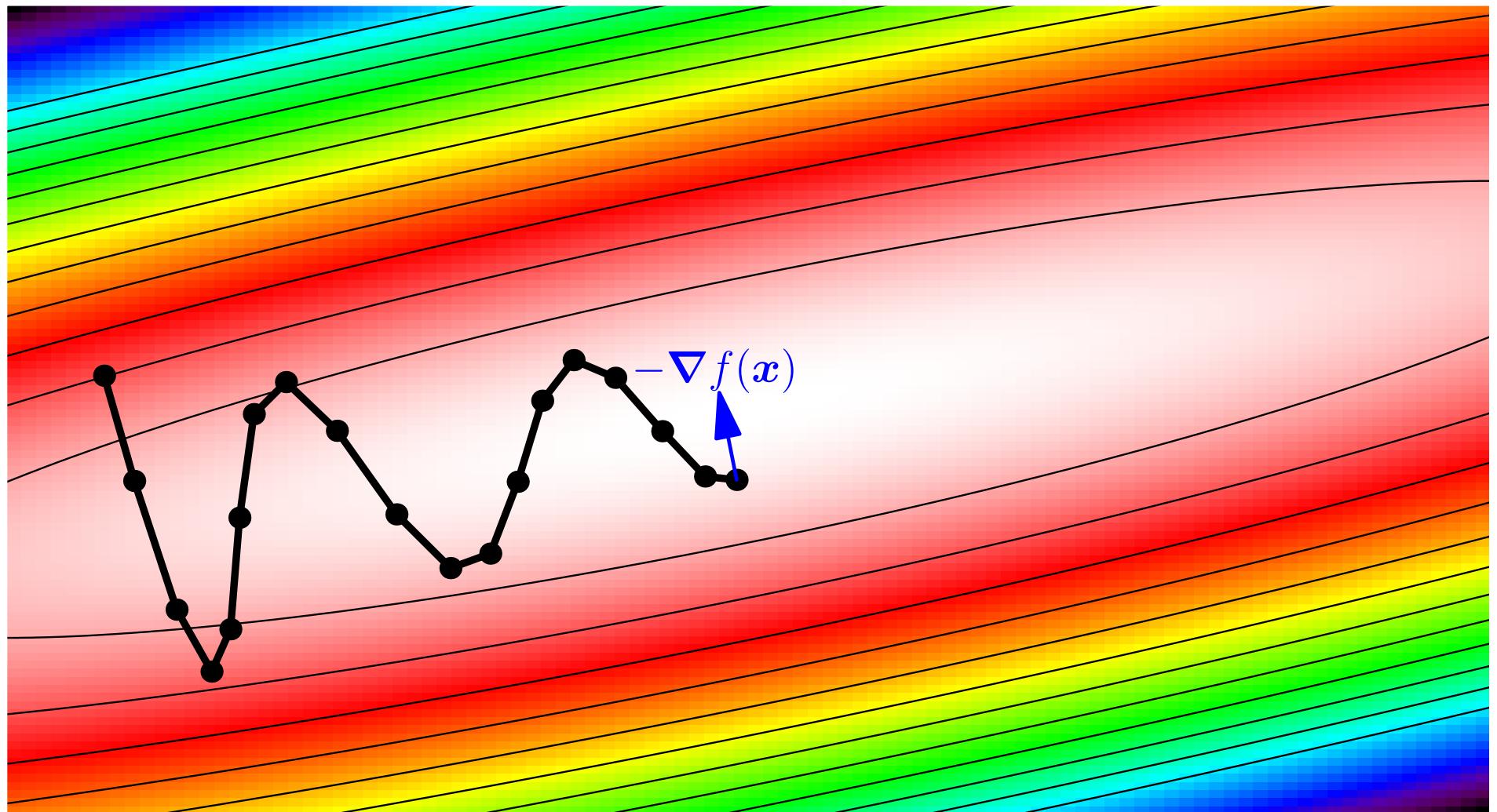
Gradient Descent with Momentum



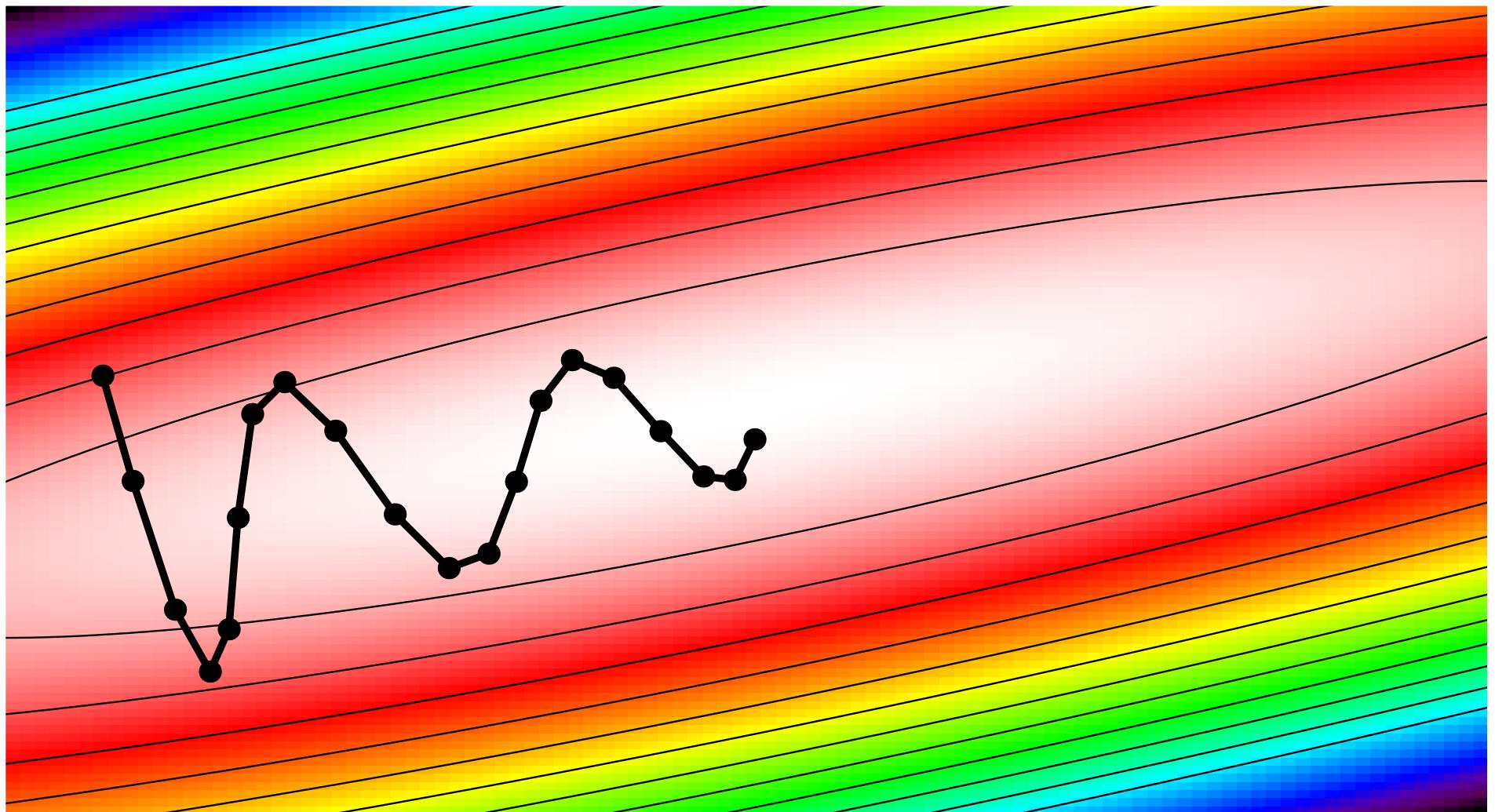
Gradient Descent with Momentum



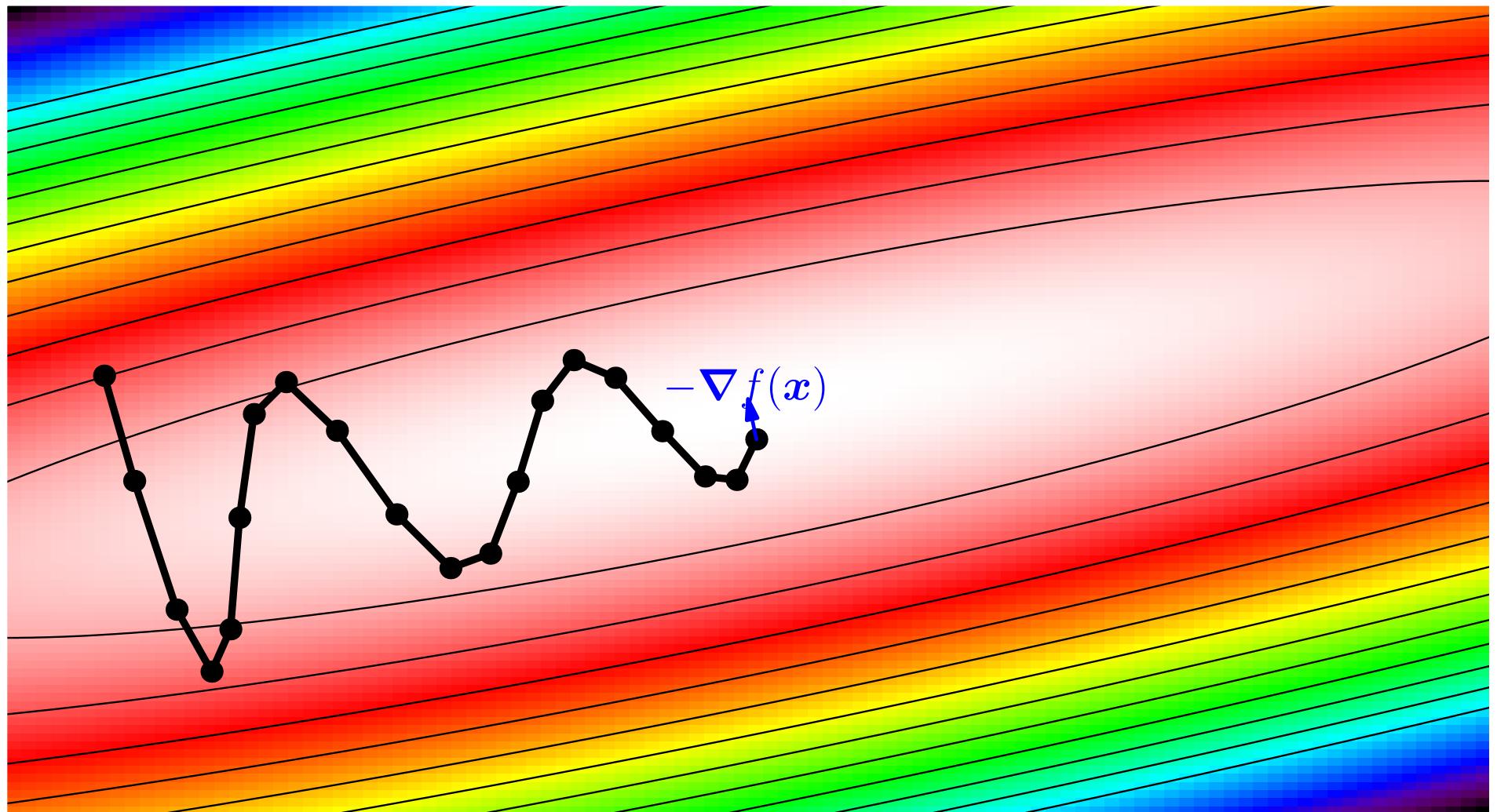
Gradient Descent with Momentum



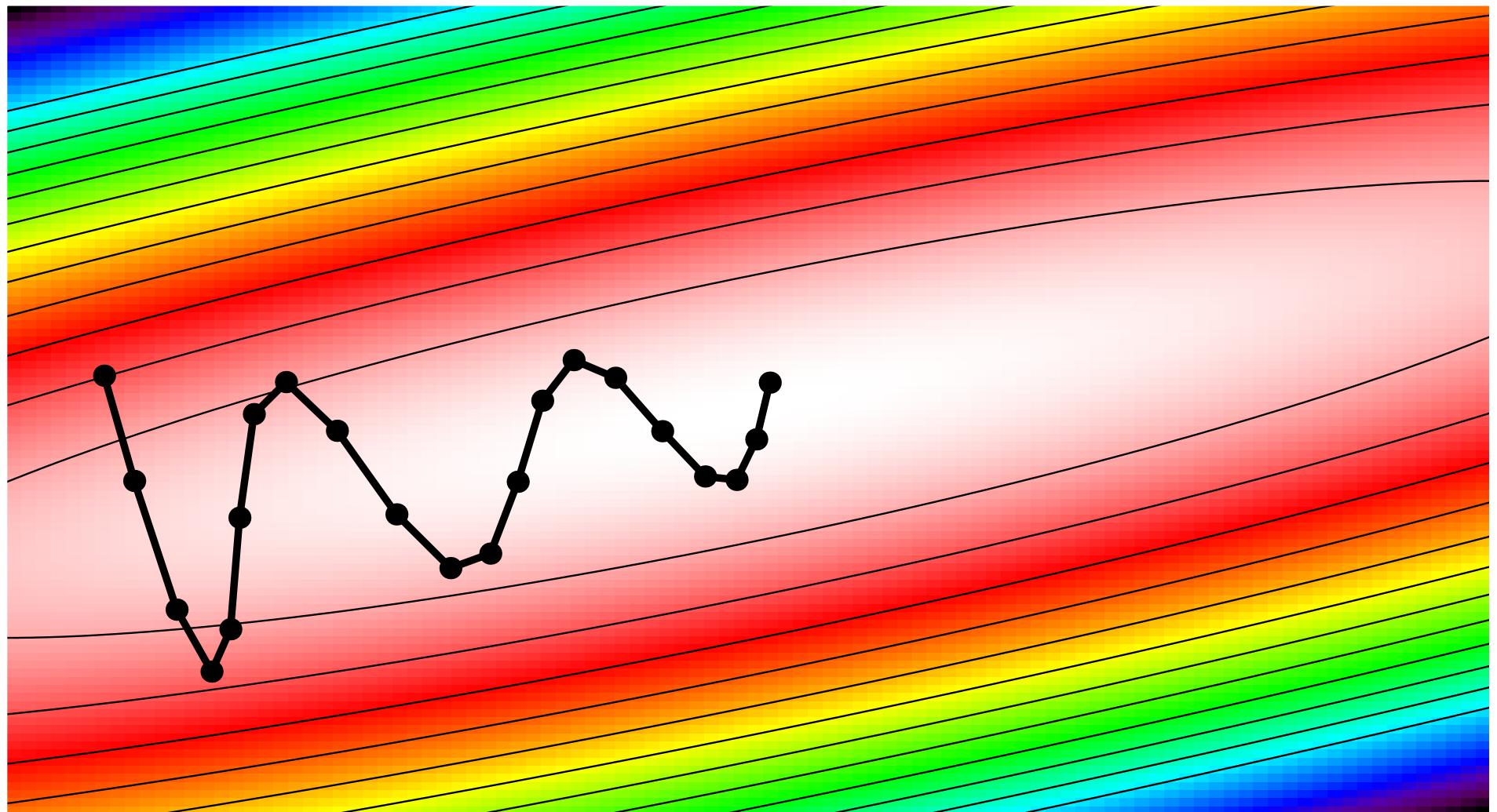
Gradient Descent with Momentum



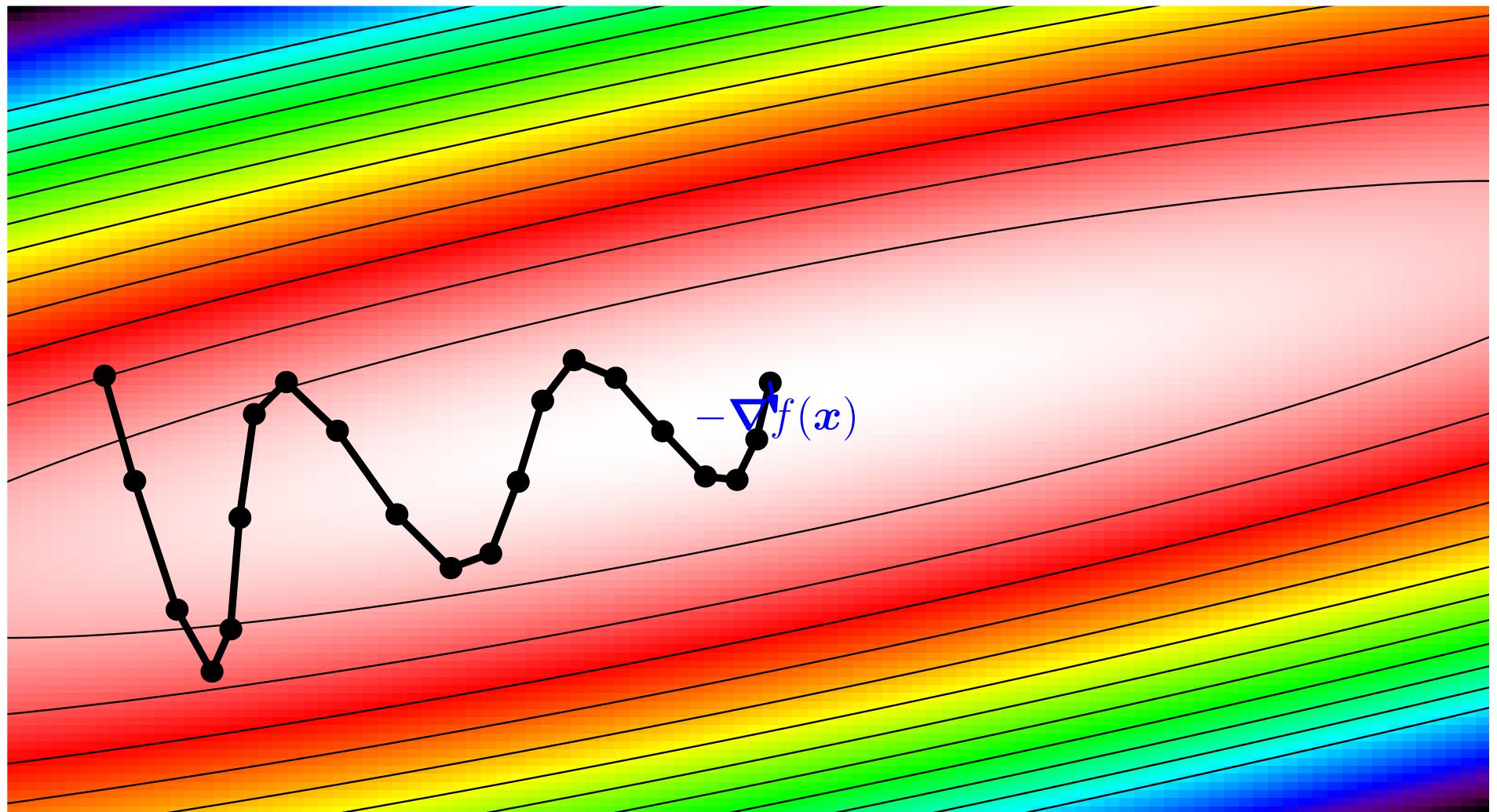
Gradient Descent with Momentum



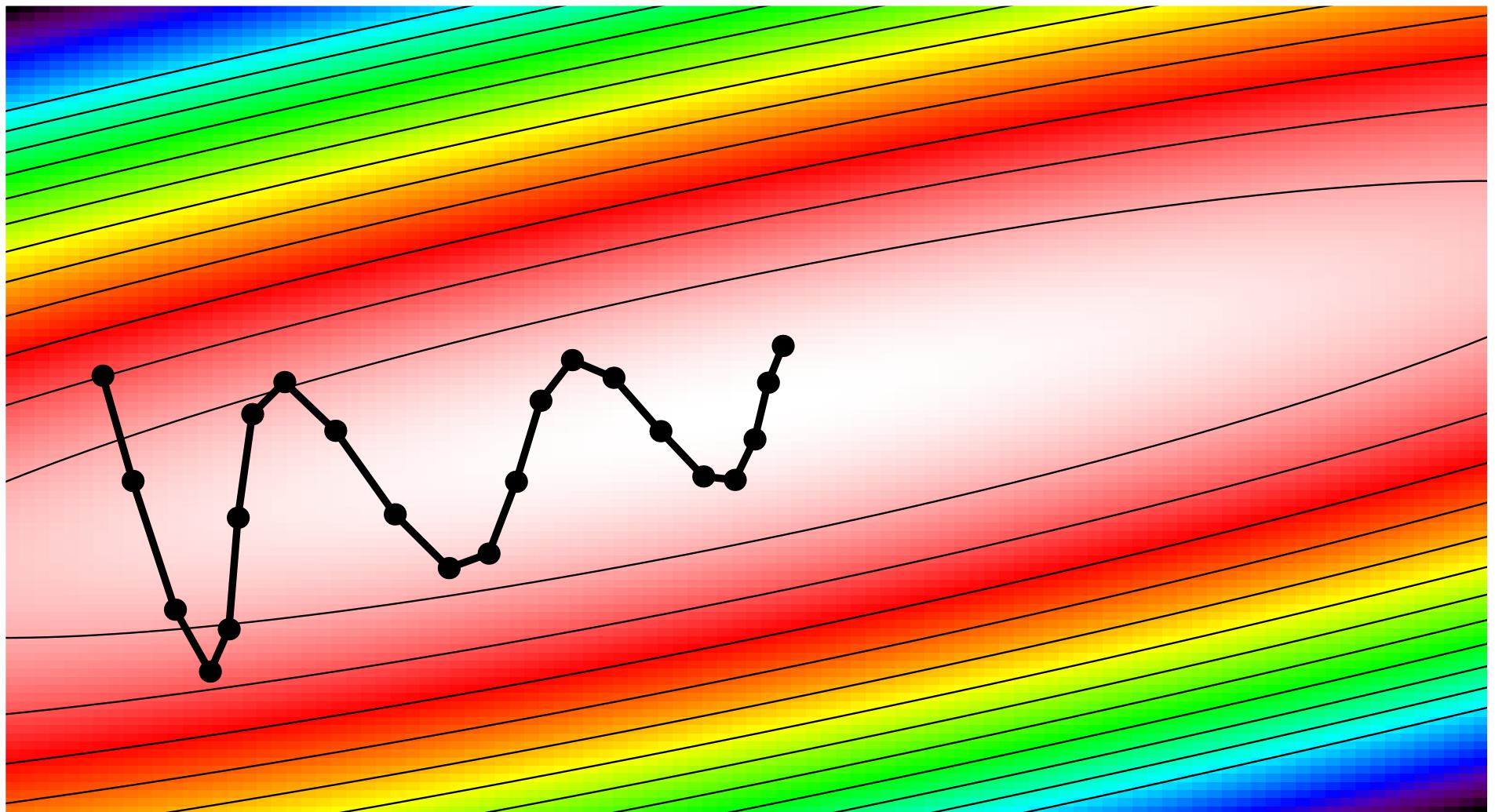
Gradient Descent with Momentum



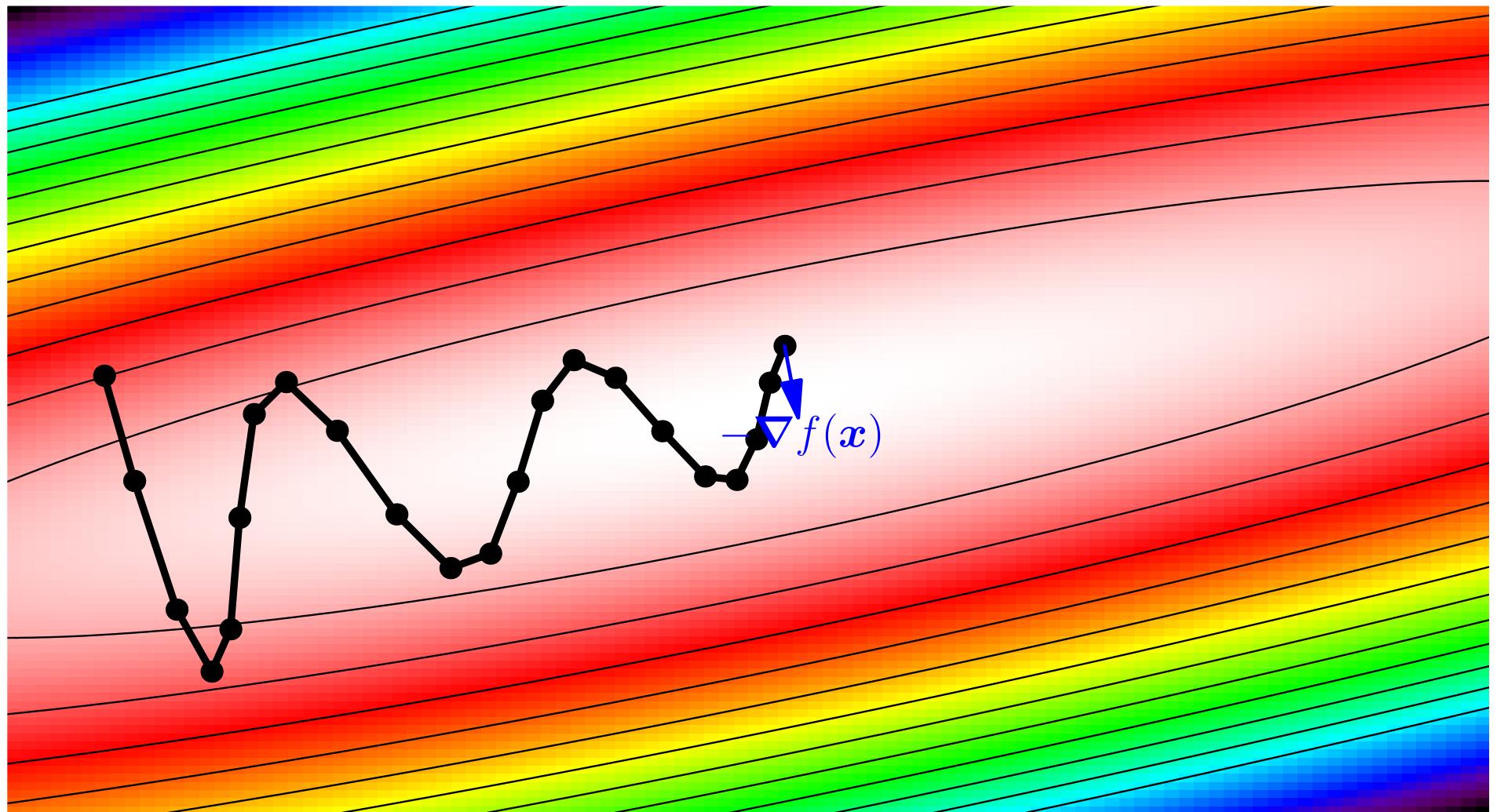
Gradient Descent with Momentum



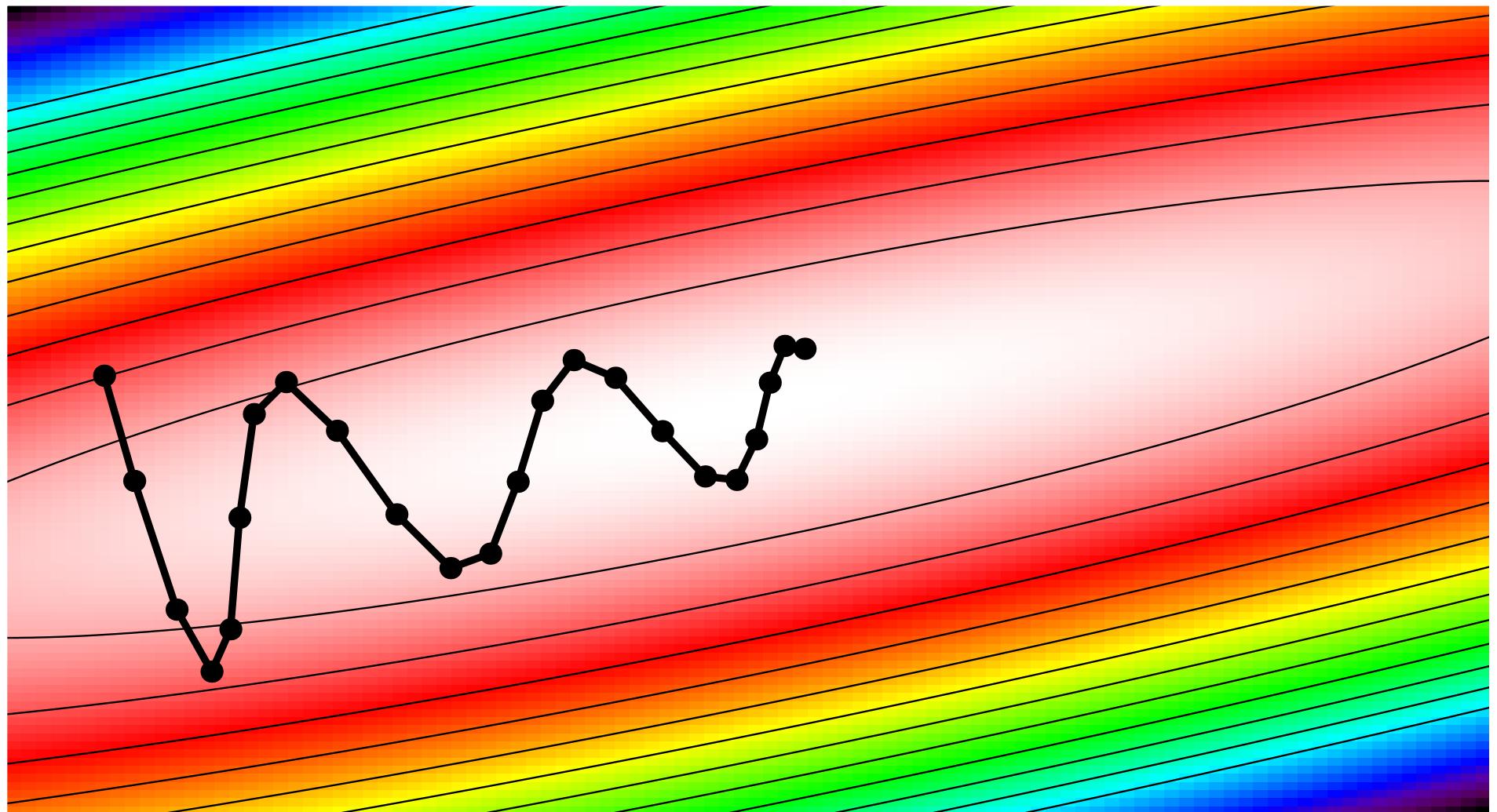
Gradient Descent with Momentum



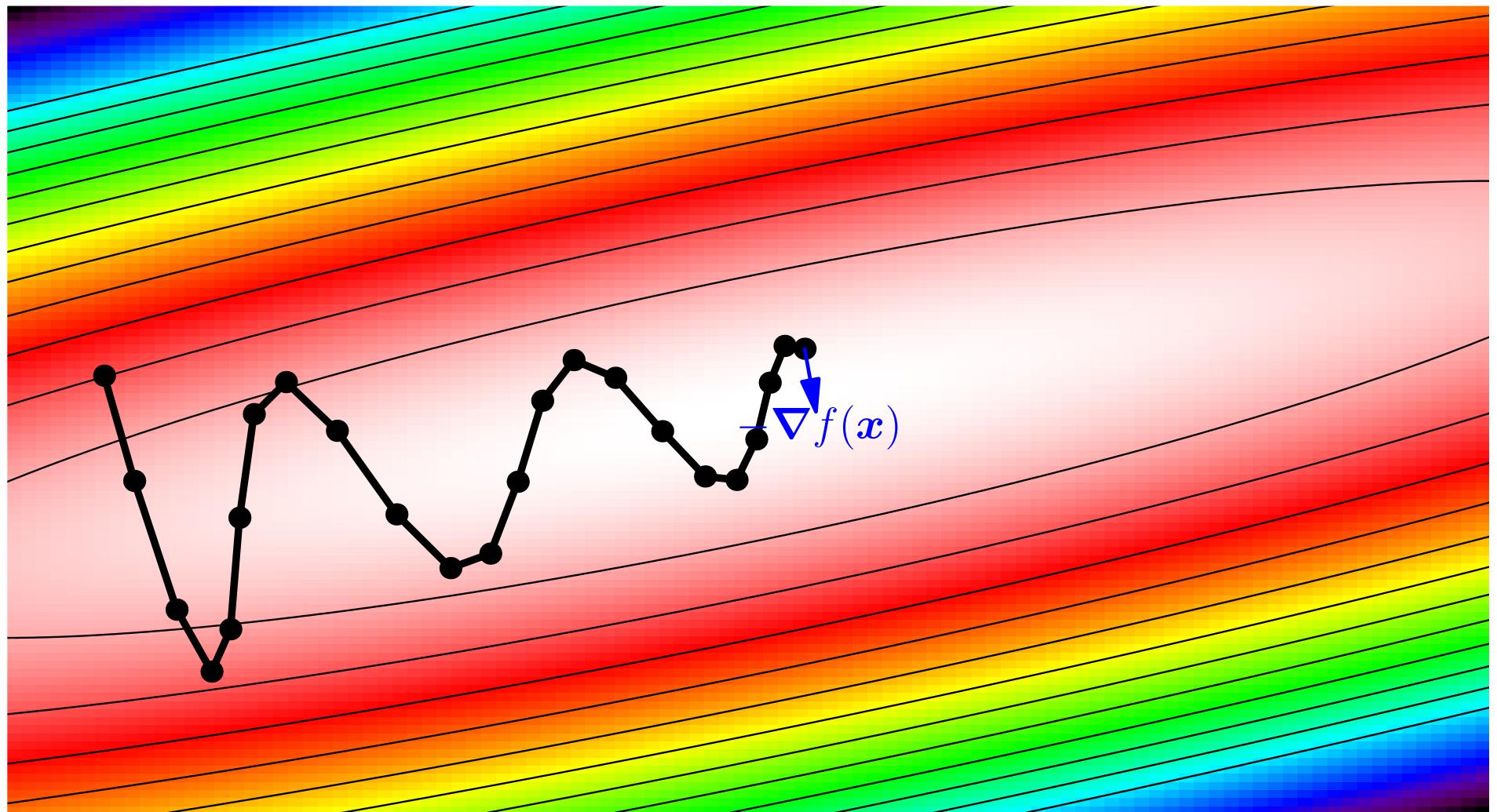
Gradient Descent with Momentum



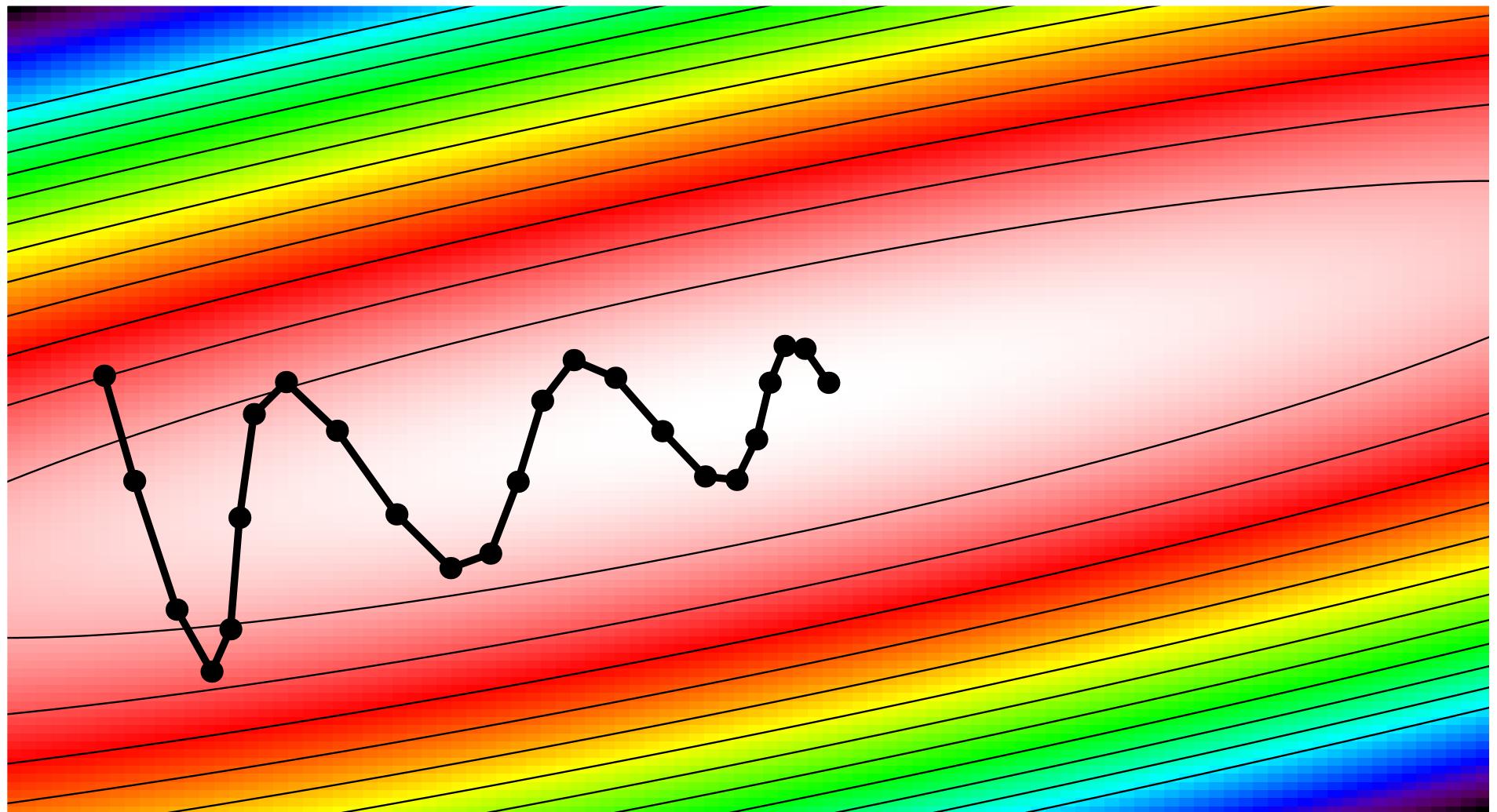
Gradient Descent with Momentum



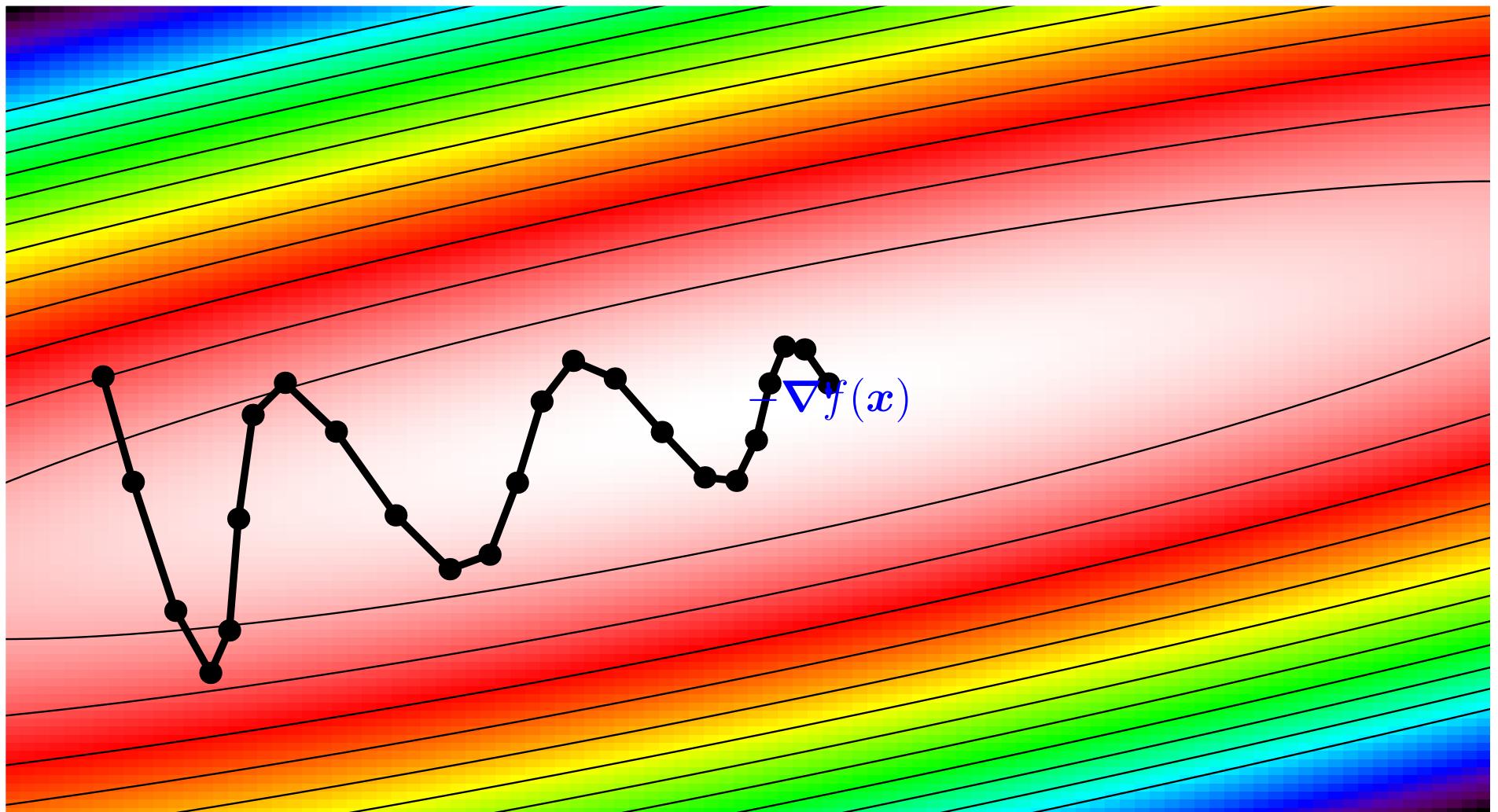
Gradient Descent with Momentum



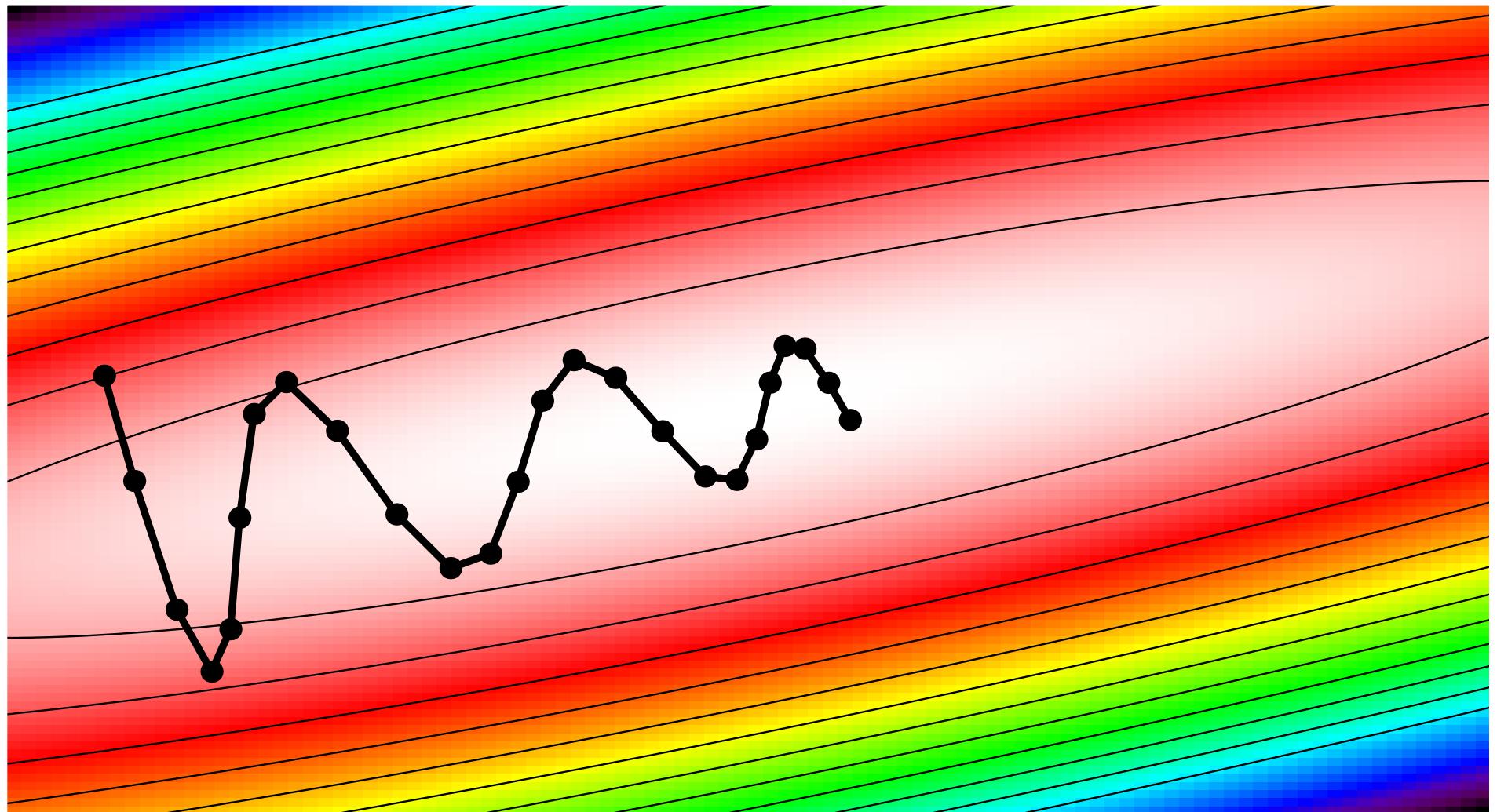
Gradient Descent with Momentum



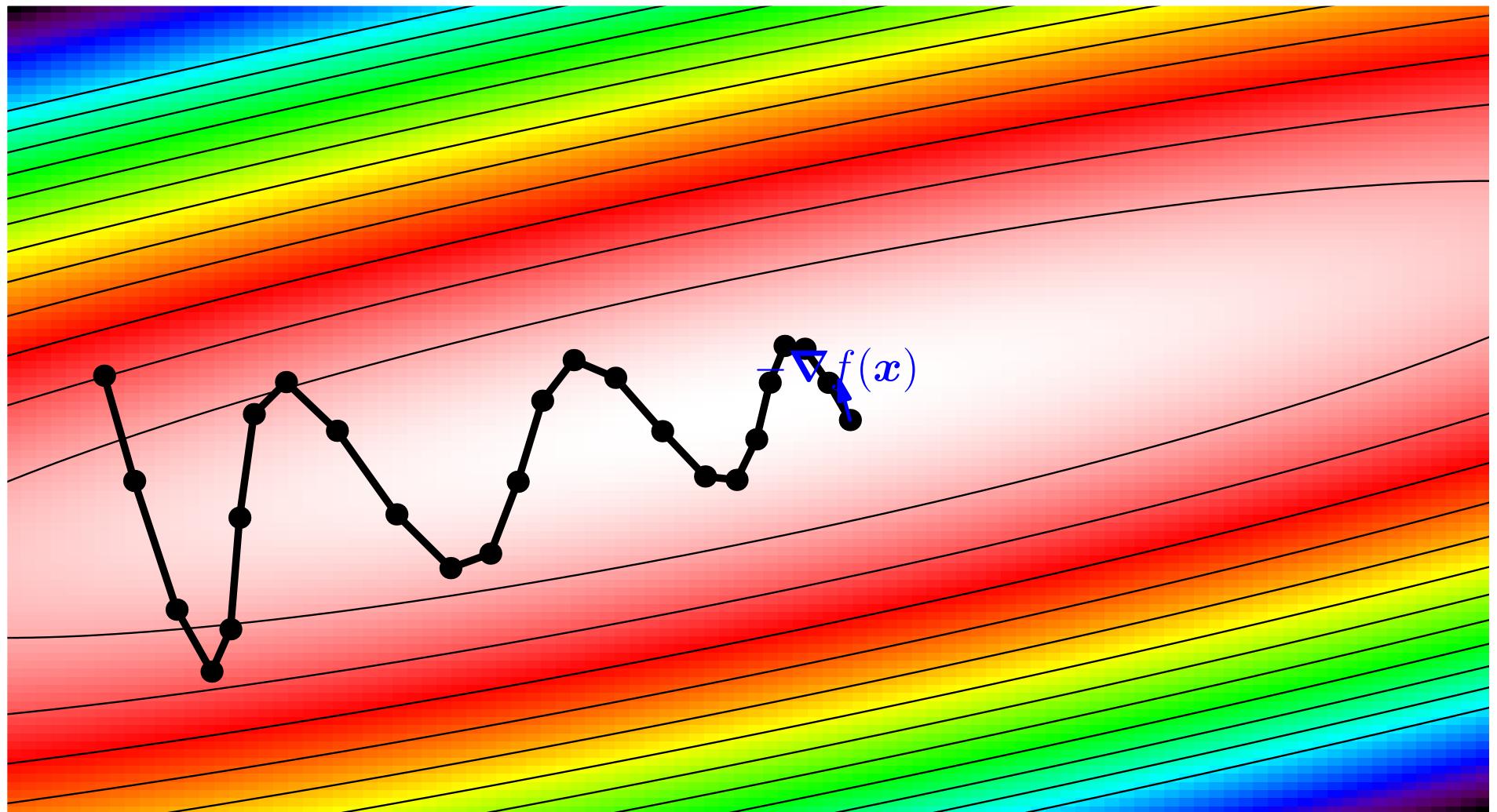
Gradient Descent with Momentum



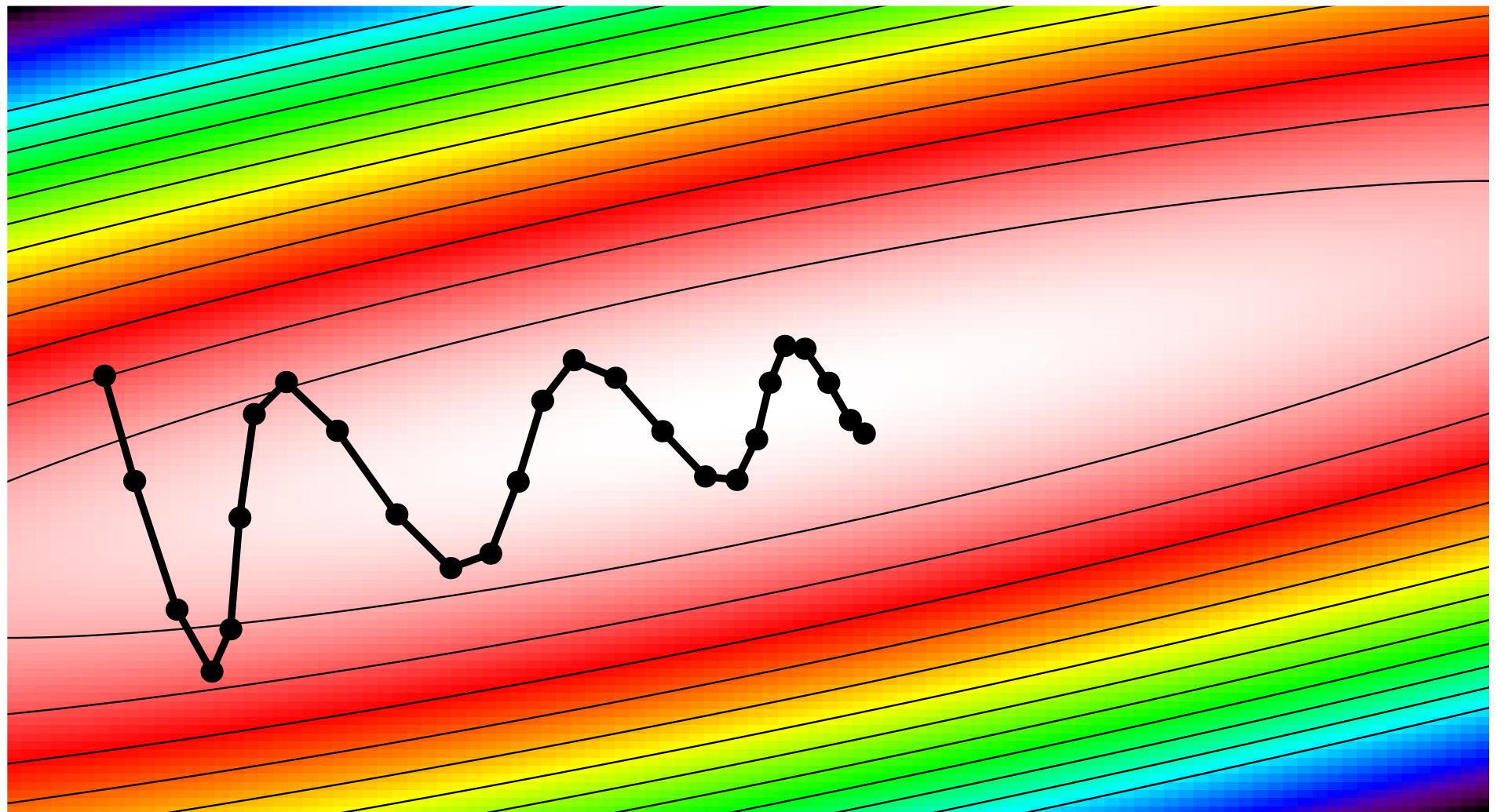
Gradient Descent with Momentum



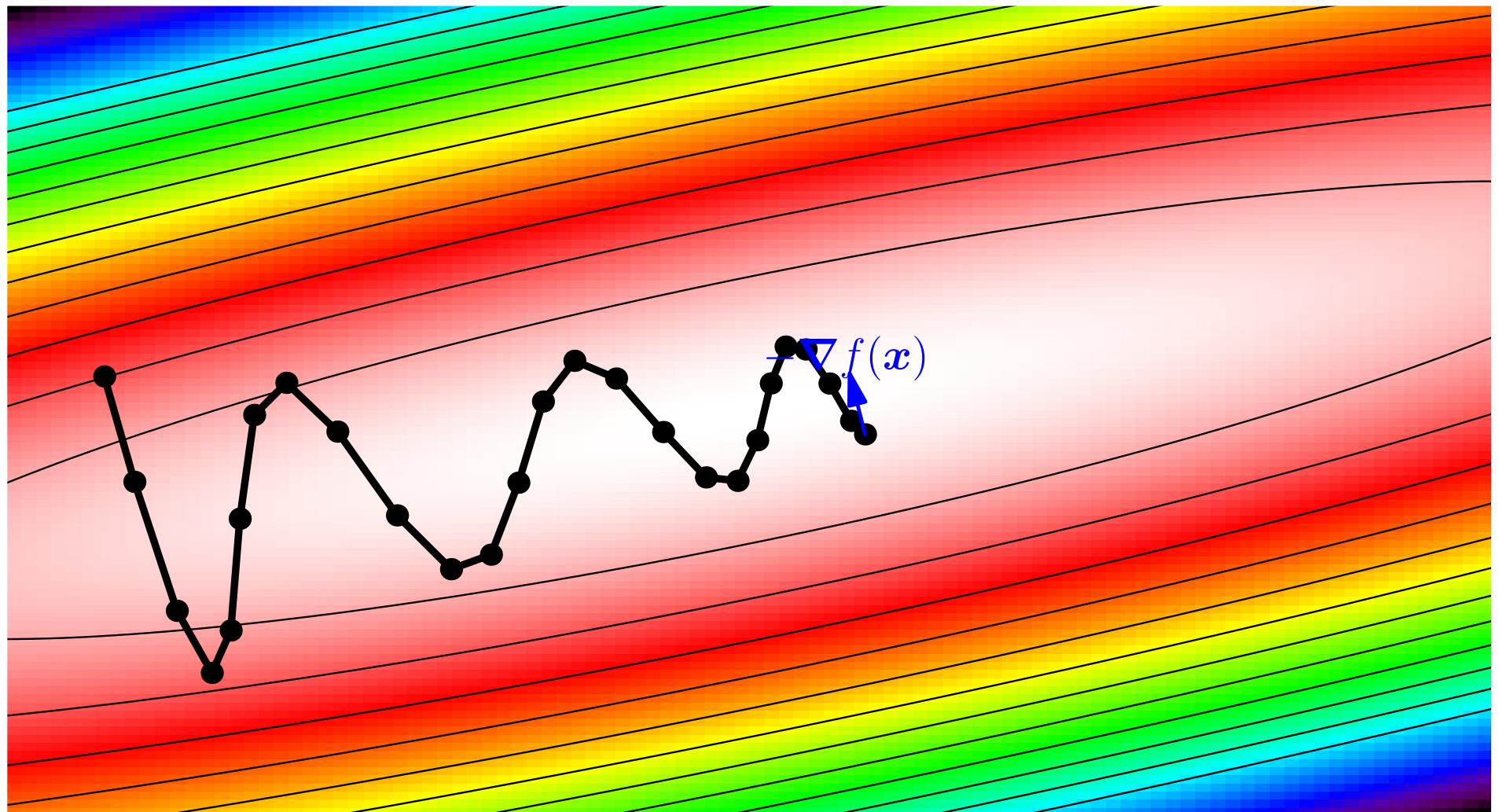
Gradient Descent with Momentum



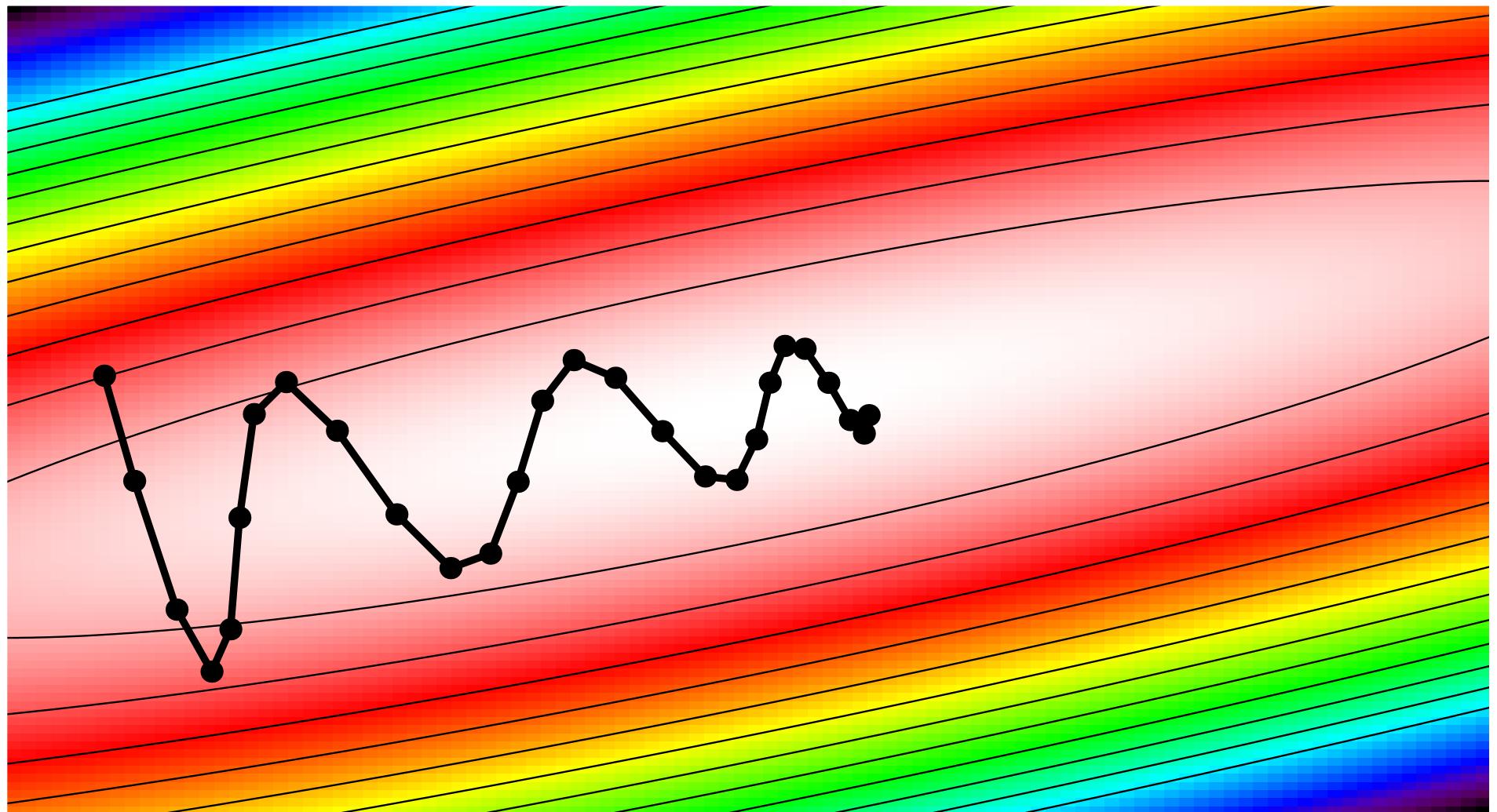
Gradient Descent with Momentum



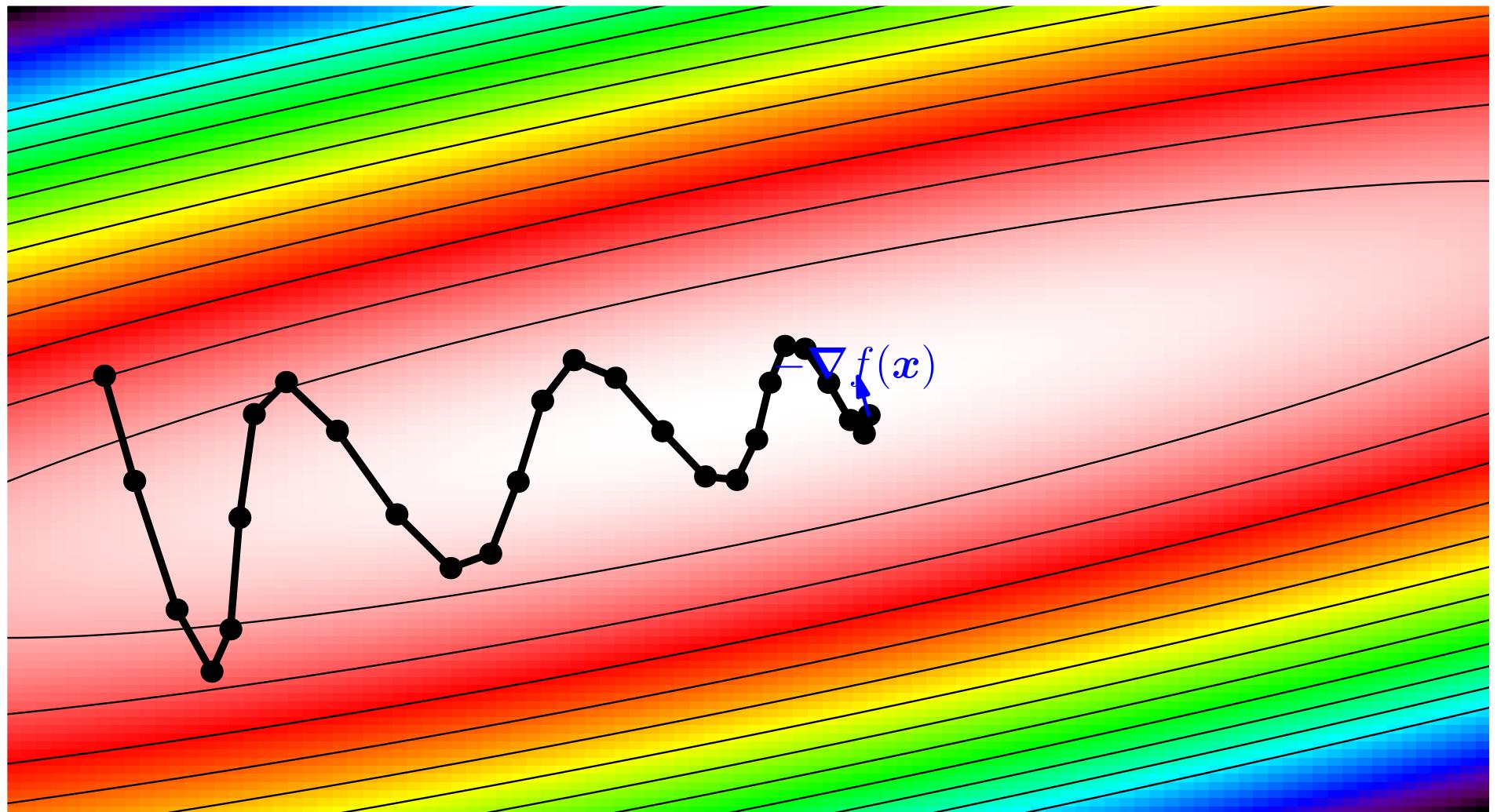
Gradient Descent with Momentum



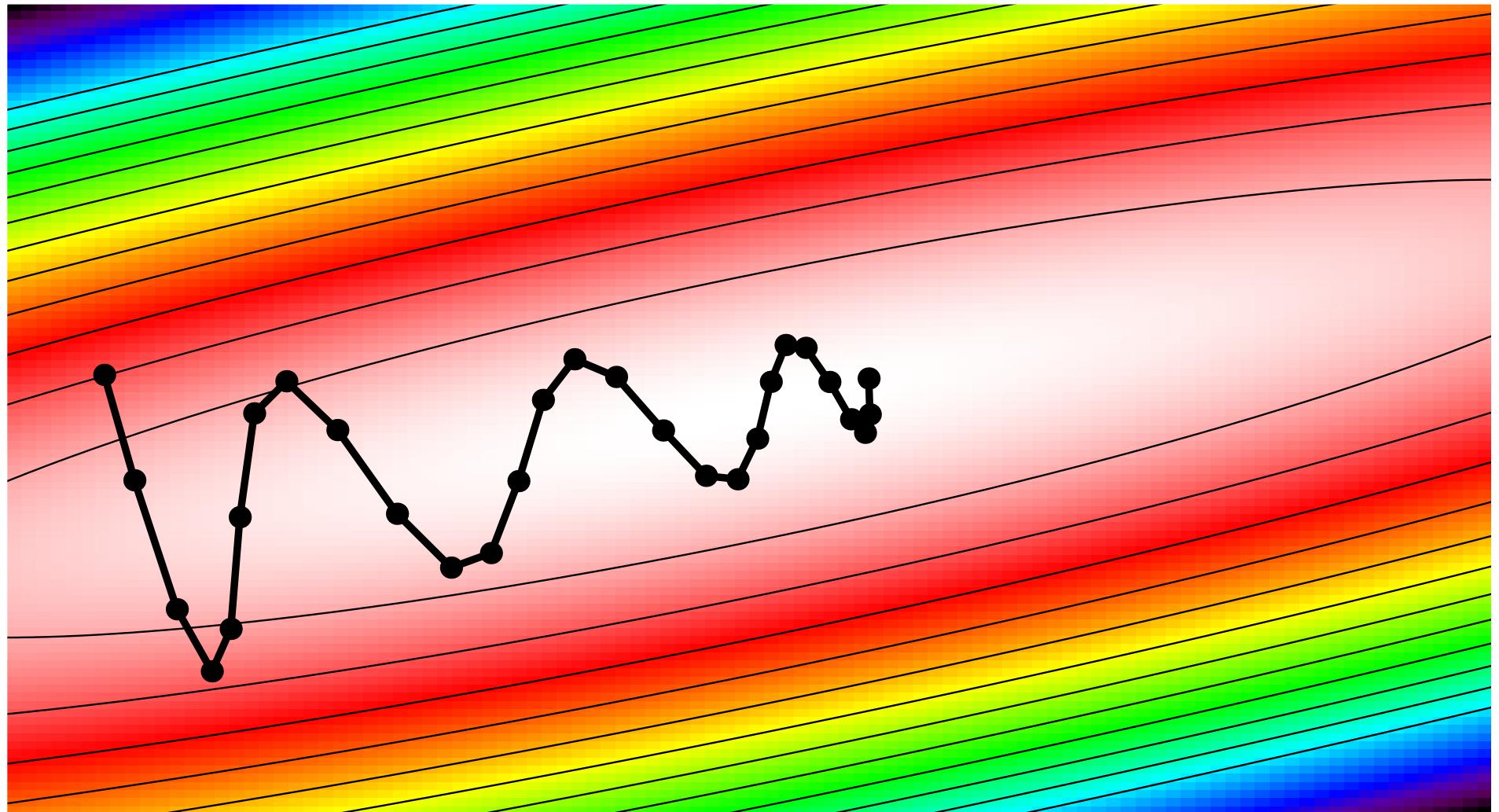
Gradient Descent with Momentum



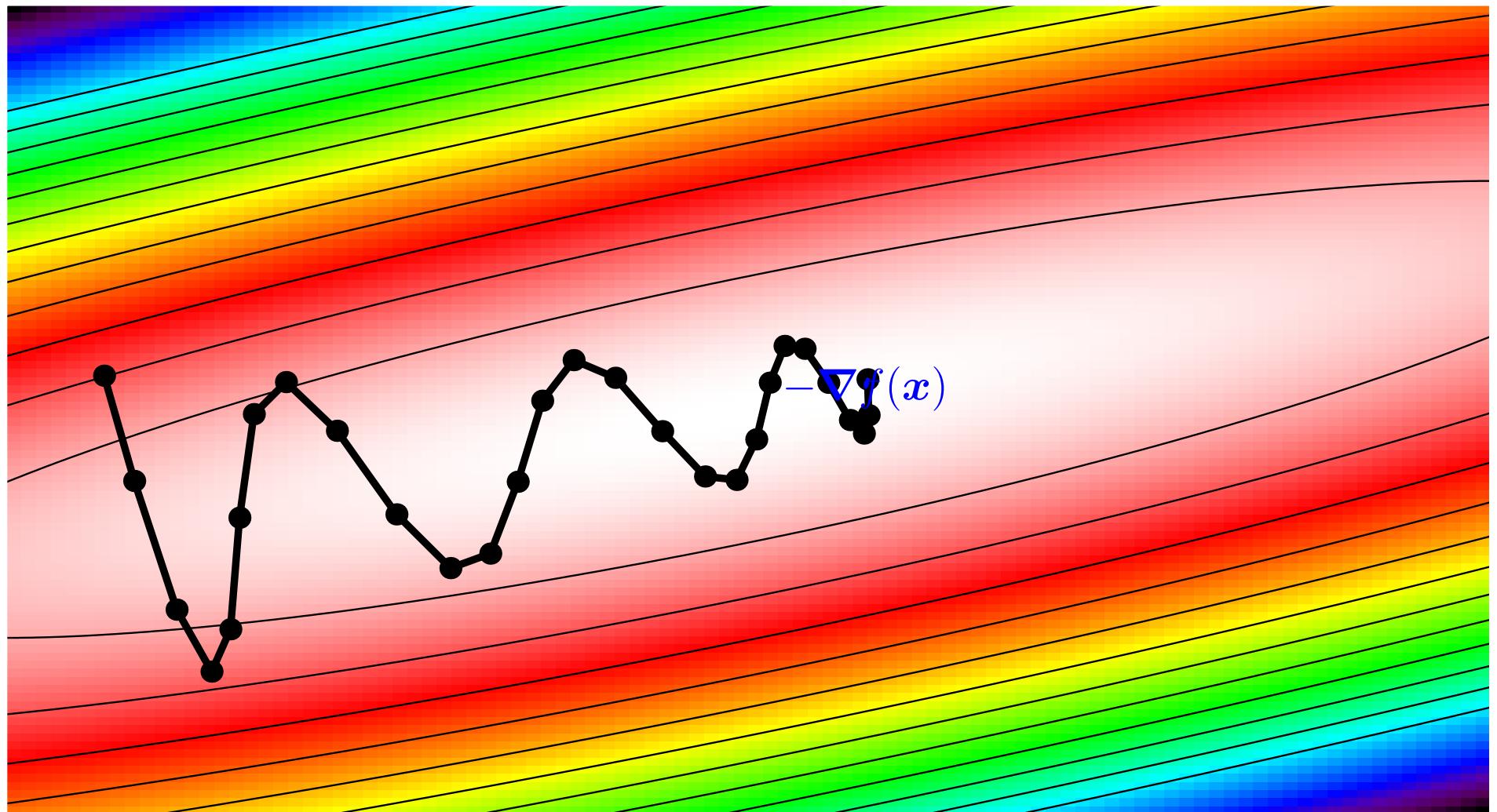
Gradient Descent with Momentum



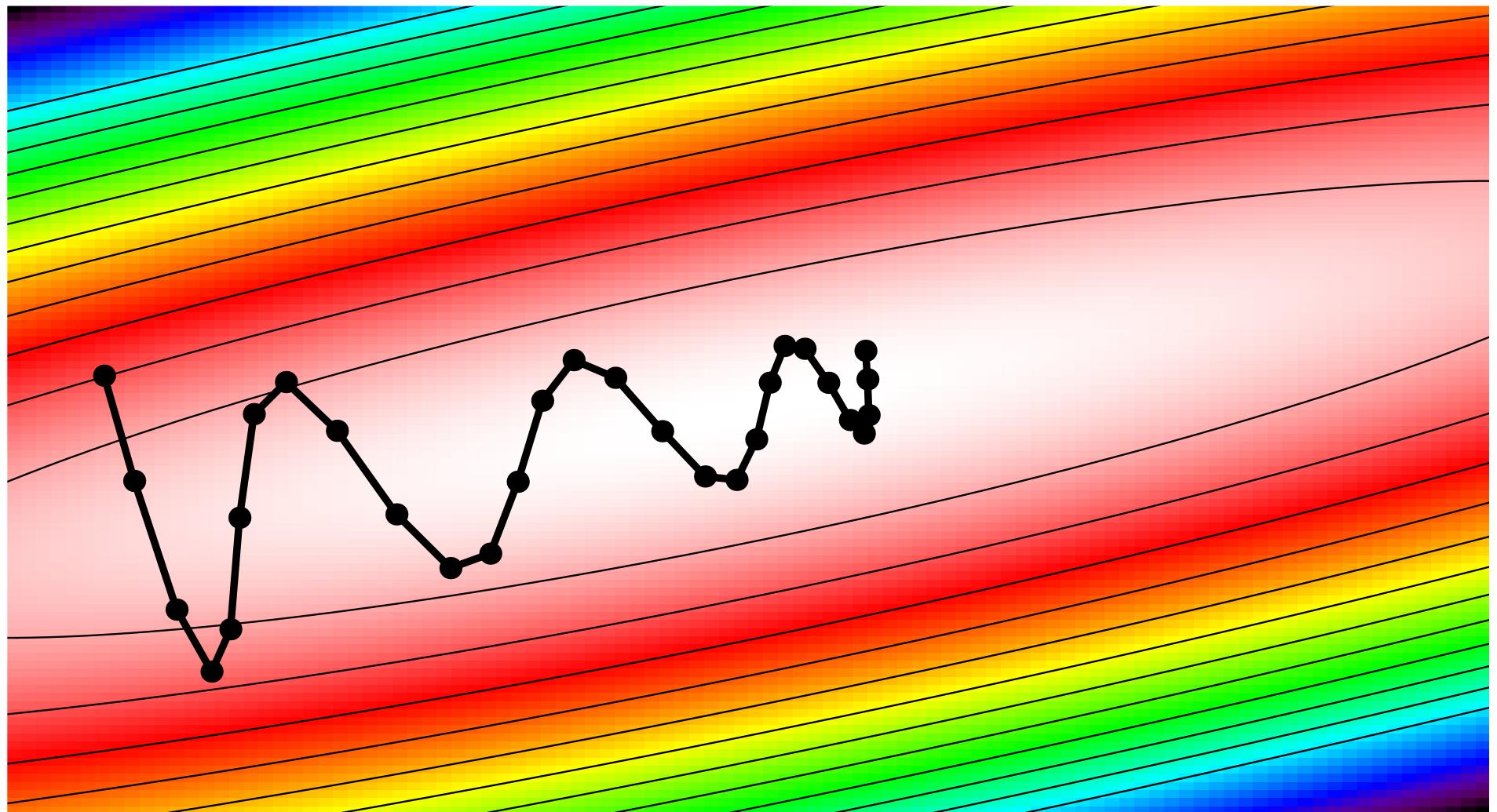
Gradient Descent with Momentum



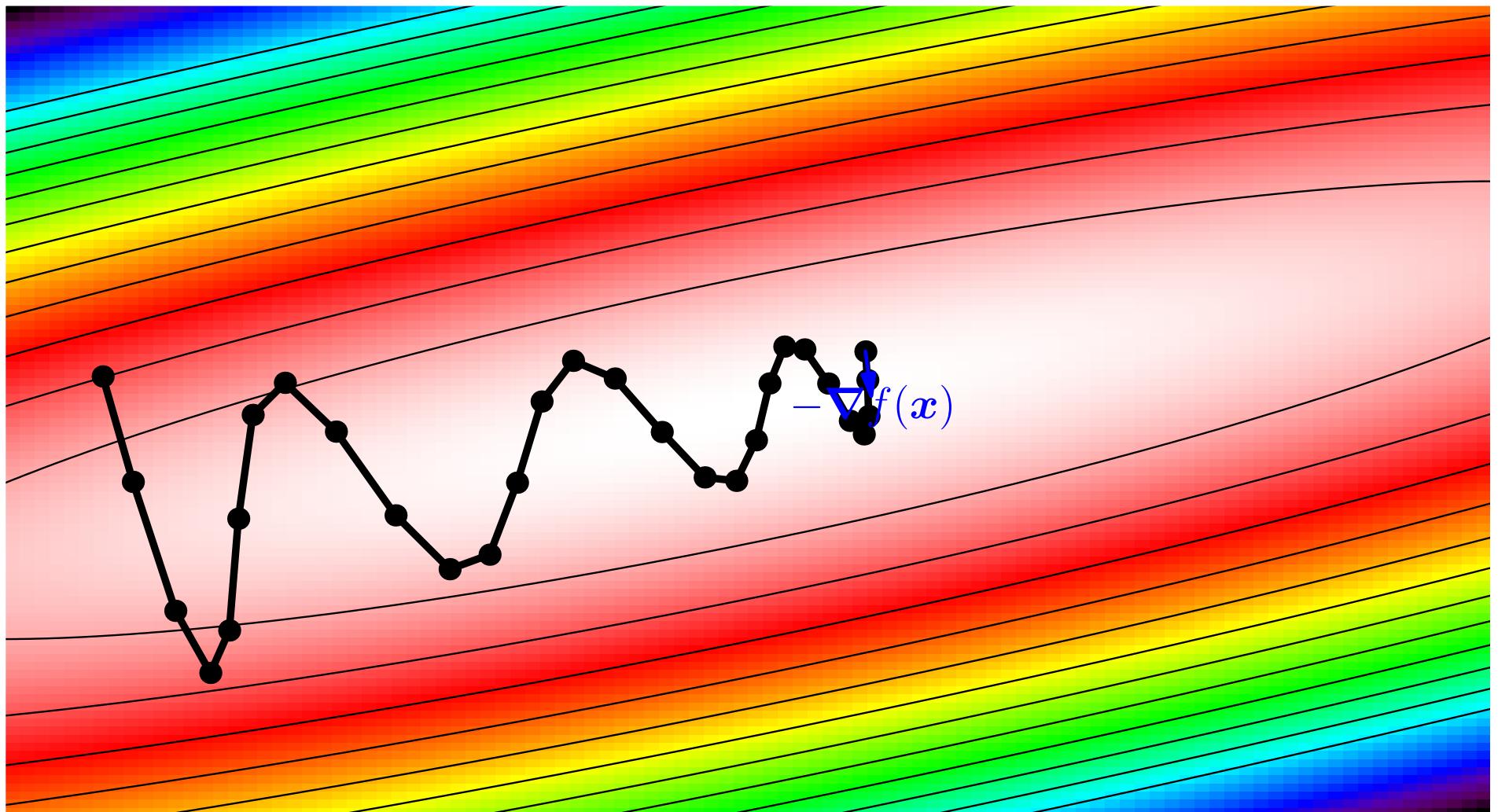
Gradient Descent with Momentum



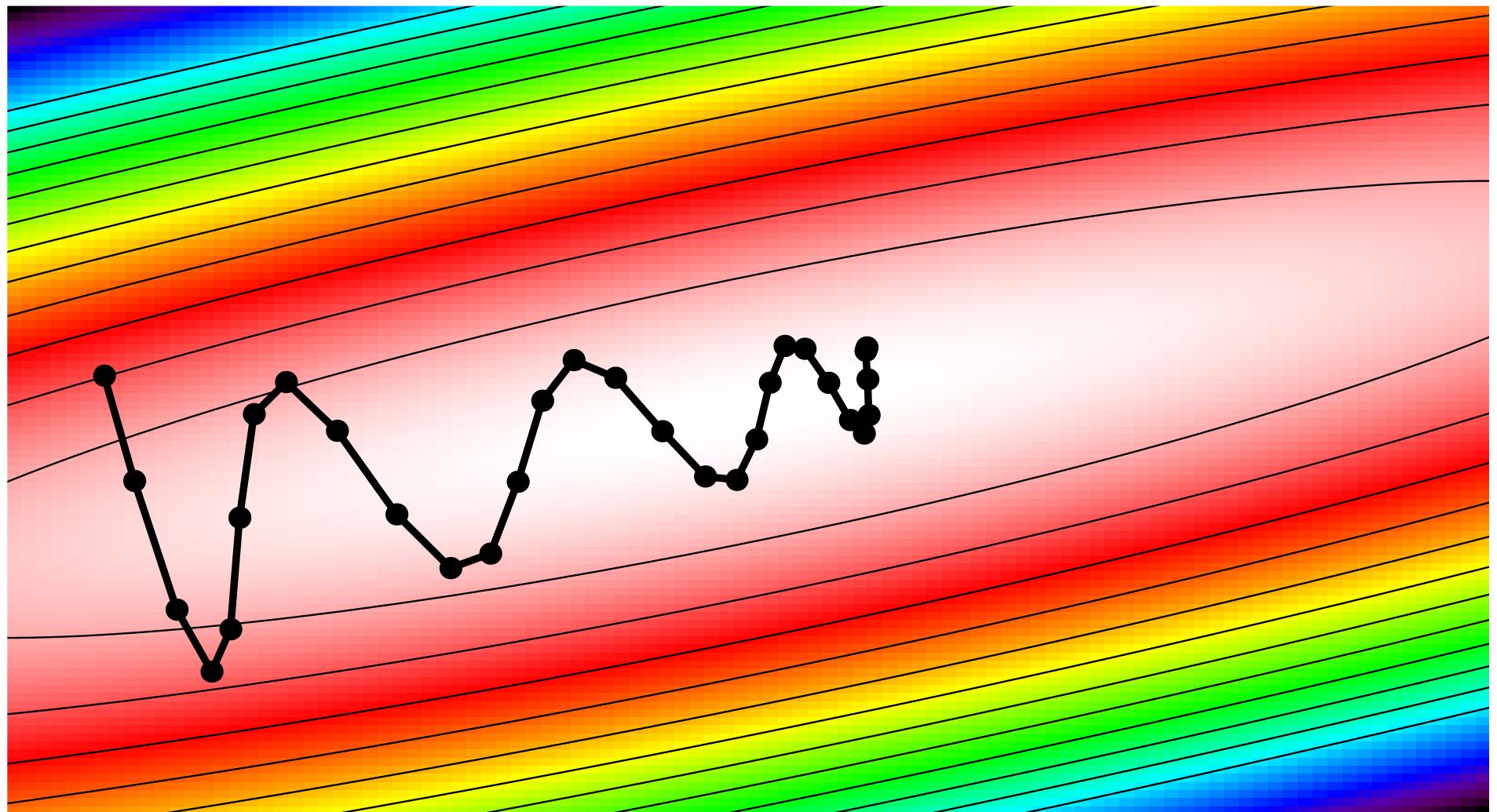
Gradient Descent with Momentum



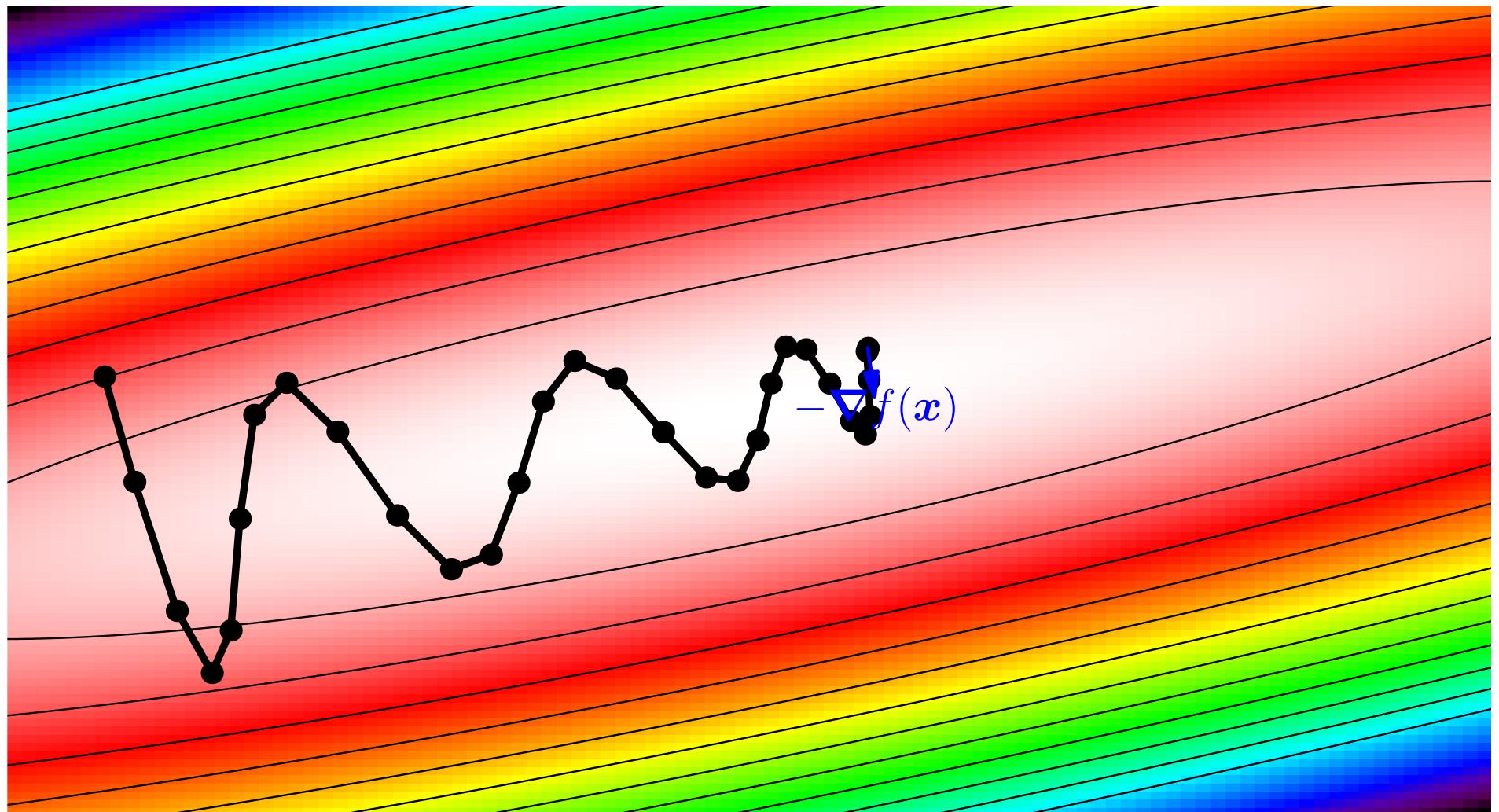
Gradient Descent with Momentum



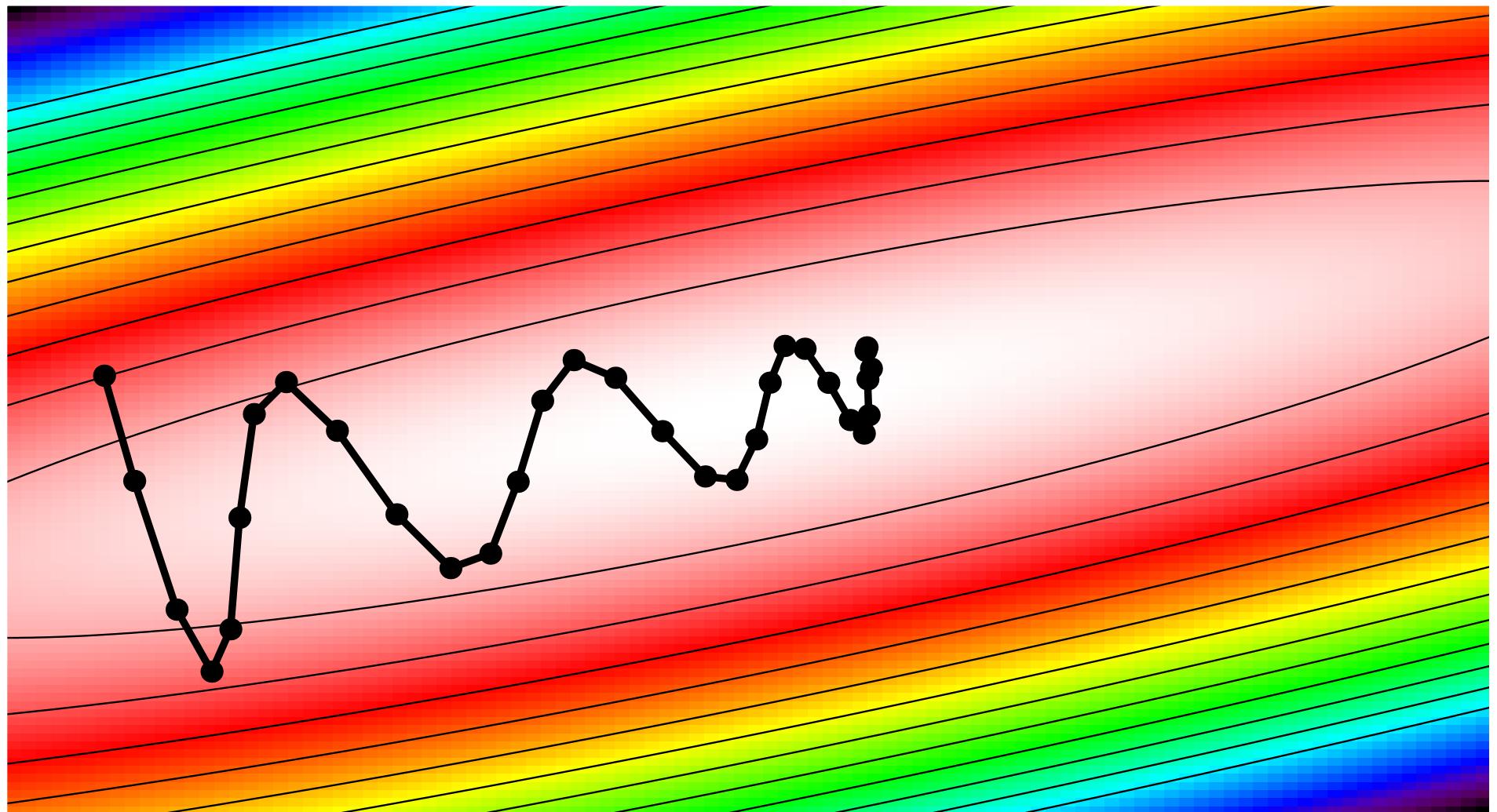
Gradient Descent with Momentum



Gradient Descent with Momentum



Gradient Descent with Momentum



Adaptive Methods

- A major difficulty of high dimensional optimisation is the existence of different scales
- That is, there are some directions where we have to move a lot (low curvature), while other directions we have to make small steps
- In adaptive methods we use a dynamic algorithm to rescale the step size of each parameter (weight)
- We can think of this as a “regularisation” that makes our basins of attraction more spherical

Adaptive Methods

- A major difficulty of high dimensional optimisation is the existence of different scales
- That is, there are some directions where we have to move a lot (low curvature), while other directions we have to make small steps
- In adaptive methods we use a dynamic algorithm to rescale the step size of each parameter (weight)
- We can think of this as a “regularisation” that makes our basins of attraction more spherical

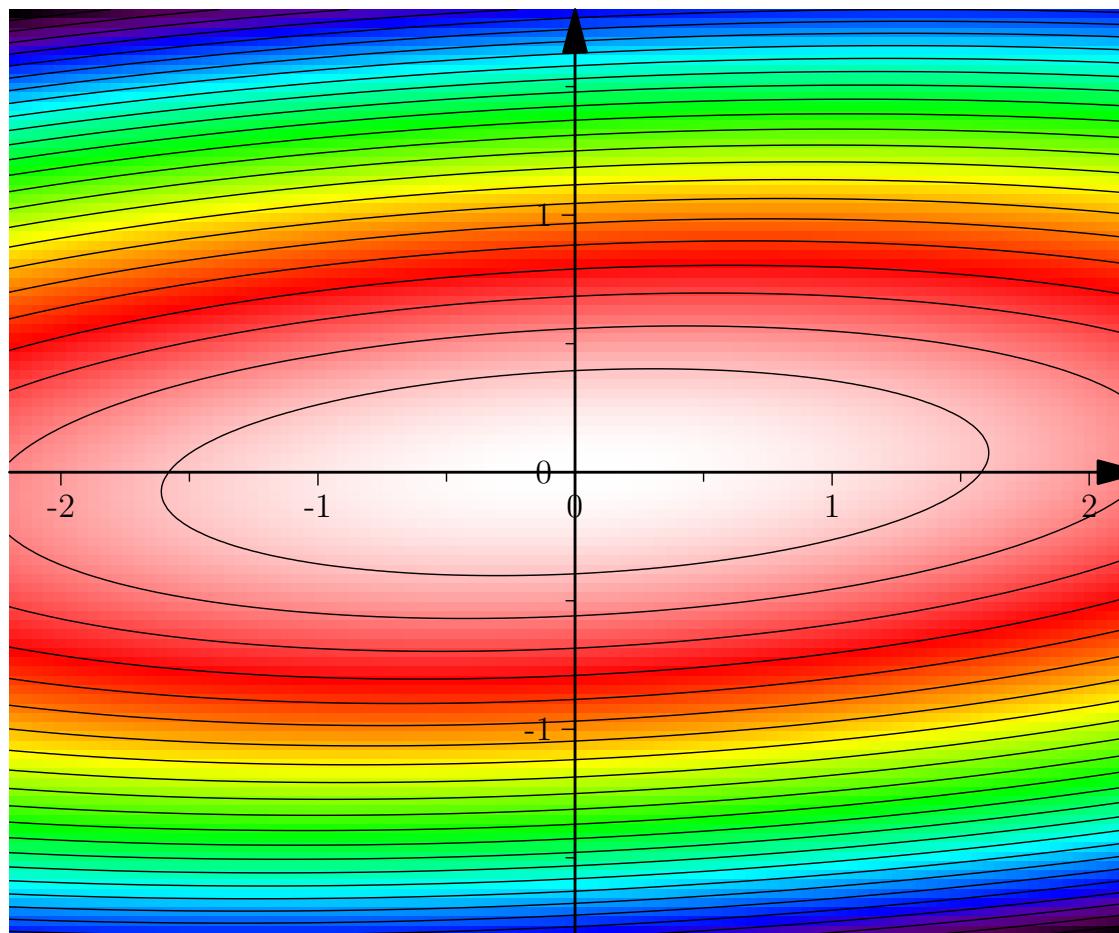
Adaptive Methods

- A major difficulty of high dimensional optimisation is the existence of different scales
- That is, there are some directions were we have to move a lot (low curvature), while other directions we have to make small steps
- In adaptive methods we use a dynamic algorithm to rescale the step size of each parameter (weight)
- We can think of this as a “regularisation” that makes our basins of attraction more spherical

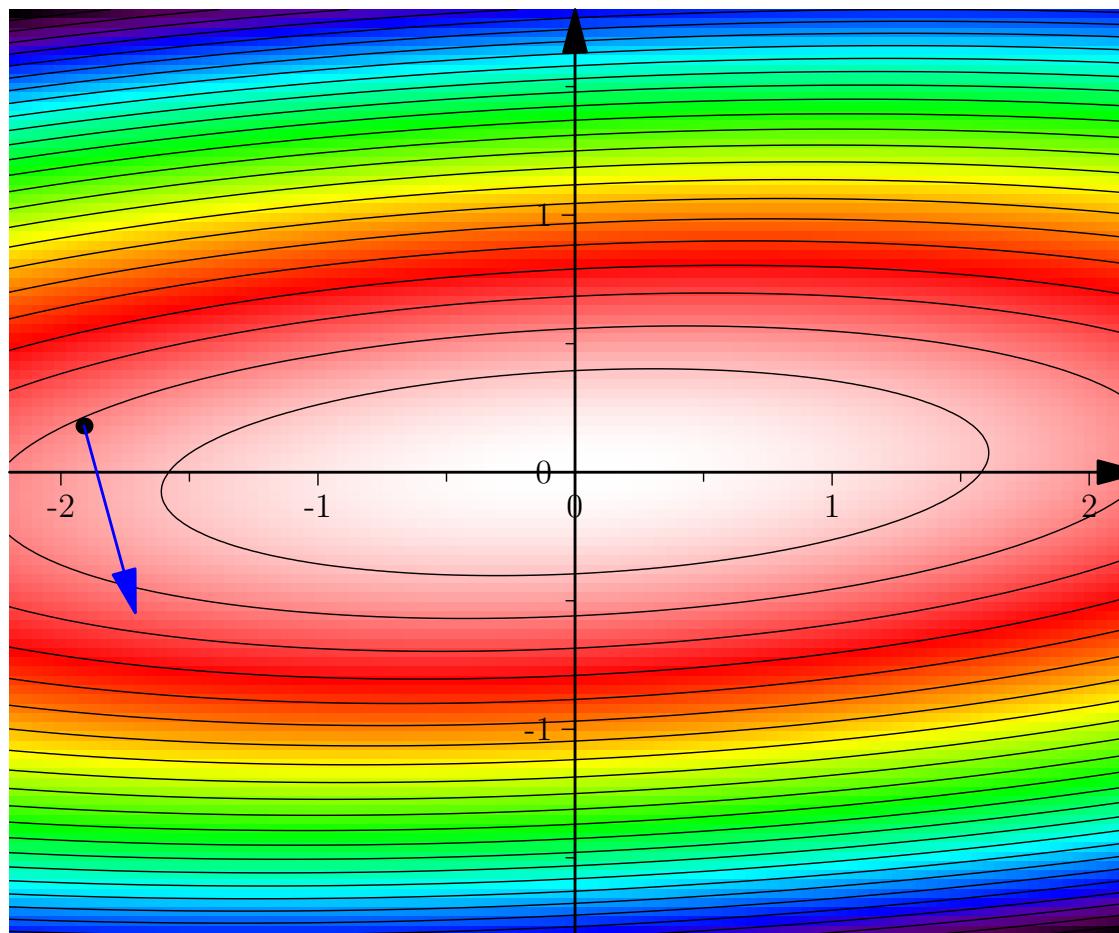
Adaptive Methods

- A major difficulty of high dimensional optimisation is the existence of different scales
- That is, there are some directions where we have to move a lot (low curvature), while other directions we have to make small steps
- In adaptive methods we use a dynamic algorithm to rescale the step size of each parameter (weight)
- We can think of this as a “regularisation” that makes our basins of attraction more spherical

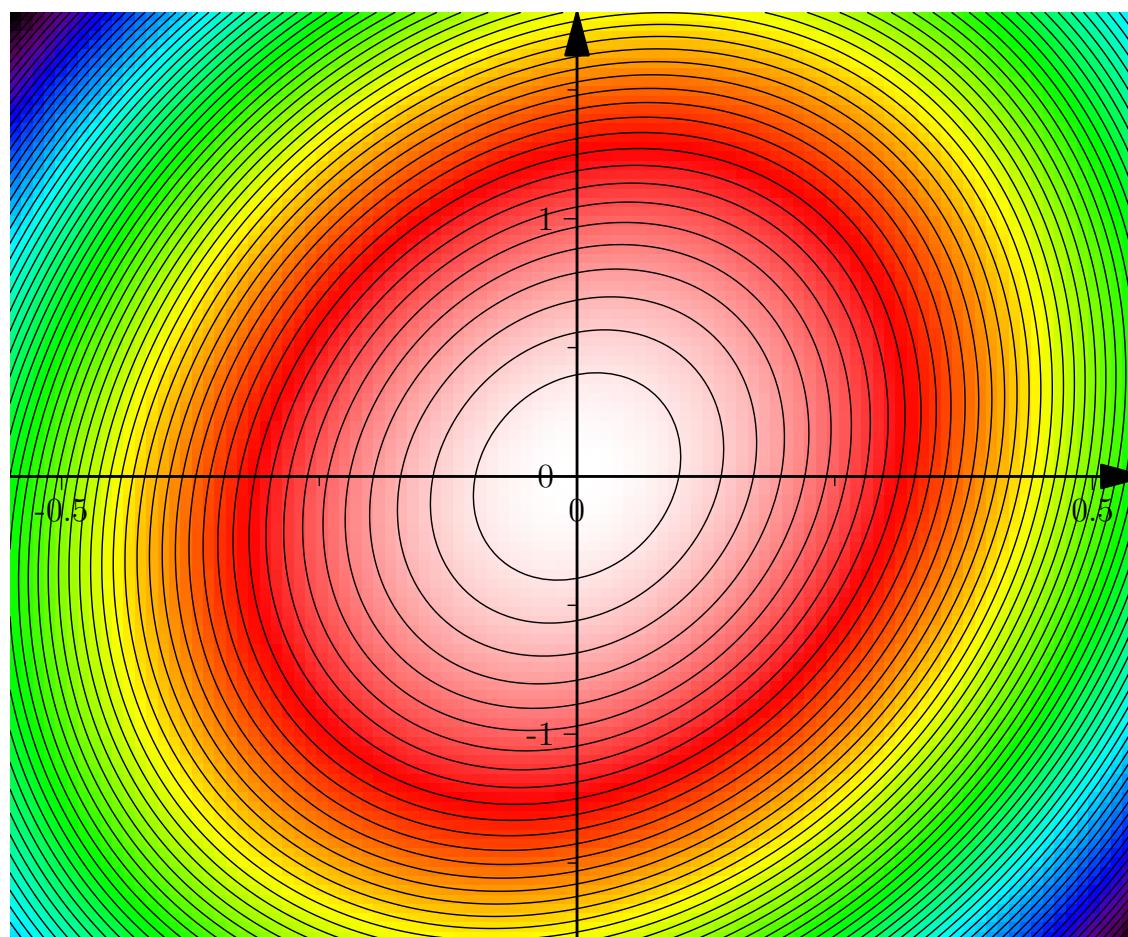
Rescaling Co-ordinates



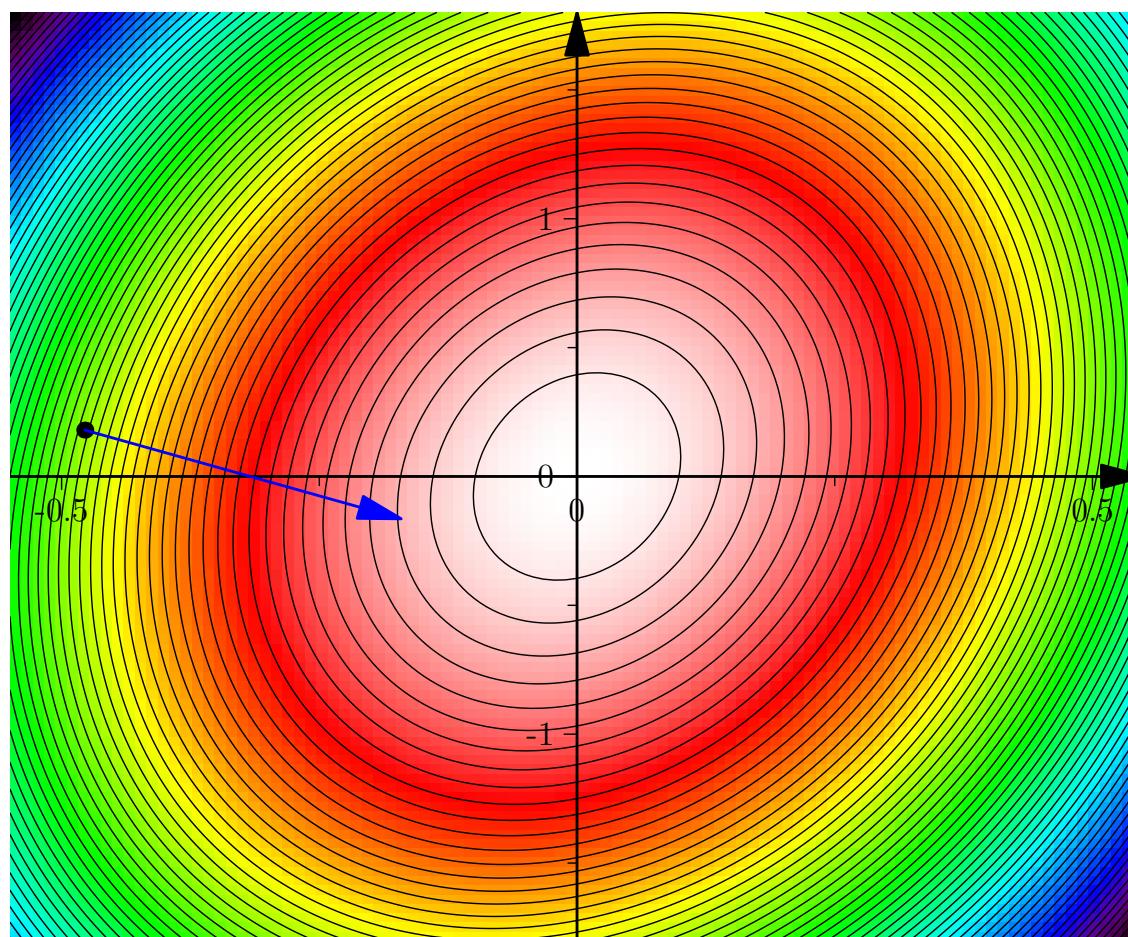
Rescaling Co-ordinates



Rescaling Co-ordinates



Rescaling Co-ordinates



AdaDelta

- We want to rescale the coordinates in proportion to the curvature in that direction
- To estimate the curvature we compute a running average of the squared gradients

$$S_i^g(t+1) = (1 - \gamma) S_i^g(t) + \gamma \left(\frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)} \right)^2$$

- To estimate the relative scale of the weights we compute a running average of the squared weights

$$S_i^w(t+1) = (1 - \gamma) S_i^w(t) + \gamma w_i(t)^2$$

AdaDelta

- We want to rescale the coordinates in proportion to the curvature in that direction
- To estimate the curvature we compute a running average of the squared gradients

$$S_i^g(t+1) = (1 - \gamma) S_i^g(t) + \gamma \left(\frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)} \right)^2$$

- To estimate the relative scale of the weights we compute a running average of the squared weights

$$S_i^w(t+1) = (1 - \gamma) S_i^w(t) + \gamma w_i(t)^2$$

AdaDelta

- We want to rescale the coordinates in proportion to the curvature in that direction
- To estimate the curvature we compute a running average of the squared gradients

$$S_i^g(t+1) = (1 - \gamma) S_i^g(t) + \gamma \left(\frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)} \right)^2$$

- To estimate the relative scale of the weights we compute a running average of the squared weights

$$S_i^w(t+1) = (1 - \gamma) S_i^w(t) + \gamma w_i(t)^2$$

AdaDelta

- The AdaDelta algorithm uses the update

$$w_i(t+1) = w_i(t) - \eta \sqrt{\frac{S_i^w(t+1) + \epsilon}{S_i^g(t+1) + \epsilon}} \frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)}$$

- Note that we can rescale the weights or rescale the gradients yet the update would be the same
- Because we are adaptively changing our step size we can use a single step size throughout the optimisation

AdaDelta

- The AdaDelta algorithm uses the update

$$w_i(t + 1) = w_i(t) - \eta \sqrt{\frac{S_i^w(t + 1) + \epsilon}{S_i^g(t + 1) + \epsilon}} \frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)}$$

- Note that we can rescale the weights or rescale the gradients yet the update would be the same
- Because we are adaptively changing our step size we can use a single step size throughout the optimisation

AdaDelta

- The AdaDelta algorithm uses the update

$$w_i(t + 1) = w_i(t) - \eta \sqrt{\frac{S_i^w(t + 1) + \epsilon}{S_i^g(t + 1) + \epsilon}} \frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)}$$

- Note that we can rescale the weights or rescale the gradients yet the update would be the same—**this is one aspect of a covariant algorithm**
- Because we are adaptively changing our step size we can use a single step size throughout the optimisation

AdaDelta

- The AdaDelta algorithm uses the update

$$w_i(t+1) = w_i(t) - \eta \sqrt{\frac{S_i^w(t+1) + \epsilon}{S_i^g(t+1) + \epsilon}} \frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)}$$

- Note that we can rescale the weights or rescale the gradients yet the update would be the same—this is one aspect of a covariant algorithm
- Because we are adaptively changing our step size we can use a single step size throughout the optimisation

Adding in Momentum

- AdaDelta doesn't use momentum (there is an argument this isn't so important as it has "regularised the landscape")
- Nevertheless we can learn a momentum term

$$M_i(t + 1) = (1 - \beta)M_i(t) + \beta \frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)}$$

$$S_i(t + 1) = (1 - \gamma)S_i(t) + \gamma \left(\frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)} \right)^2$$

- These running averages have a lag time which we can remove

$$\hat{M}_i(t + 1) = \frac{M_i(t + 1)}{1 - (1 - \beta)^t} \quad \hat{S}_i(t + 1) = \frac{S_i(t + 1)}{1 - (1 - \gamma)^t}$$

Adding in Momentum

- AdaDelta doesn't use momentum (there is an argument this isn't so important as it has "regularised the landscape")
- Nevertheless we can learn a momentum term

$$M_i(t+1) = (1 - \beta) M_i(t) + \beta \frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)}$$

$$S_i(t+1) = (1 - \gamma) S_i(t) + \gamma \left(\frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)} \right)^2$$

- These running averages have a lag time which we can remove

$$\hat{M}_i(t+1) = \frac{M_i(t+1)}{1 - (1 - \beta)^t} \quad \hat{S}_i(t+1) = \frac{S_i(t+1)}{1 - (1 - \gamma)^t}$$

Adding in Momentum

- AdaDelta doesn't use momentum (there is an argument this isn't so important as it has "regularised the landscape")
- Nevertheless we can learn a momentum term

$$M_i(t+1) = (1 - \beta) M_i(t) + \beta \frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)}$$

$$S_i(t+1) = (1 - \gamma) S_i(t) + \gamma \left(\frac{\partial L_{\mathcal{B}}(\mathbf{w}(t))}{\partial w_i(t)} \right)^2$$

- These running averages have a lag time which we can remove

$$\hat{M}_i(t+1) = \frac{M_i(t+1)}{1 - (1 - \beta)^t} \quad \hat{S}_i(t+1) = \frac{S_i(t+1)}{1 - (1 - \gamma)^t}$$

ADAM

- Adaptive Moment Estimation (Adam) adapts the scale of the parameters, but also uses momentum
- Update weights

$$w_i(t + 1) = w_i(t) - \frac{\eta}{\sqrt{\hat{S}_i(t + 1)} + \epsilon} \hat{M}_i(t + 1)$$

- ADAM and its variants are very successful in deep learning
- It is very robust so often results in a network learning which might not learn under standard SGD (with momentum)

ADAM

- Adaptive Moment Estimation (Adam) adapts the scale of the parameters, but also uses momentum
- Update weights

$$w_i(t + 1) = w_i(t) - \frac{\eta}{\sqrt{\hat{S}_i(t + 1)} + \epsilon} \hat{M}_i(t + 1)$$

- ADAM and its variants are very successful in deep learning
- It is very robust so often results in a network learning which might not learn under standard SGD (with momentum)

ADAM

- Adaptive Moment Estimation (Adam) adapts the scale of the parameters, but also uses momentum
- Update weights

$$w_i(t + 1) = w_i(t) - \frac{\eta}{\sqrt{\hat{S}_i(t + 1)} + \epsilon} \hat{M}_i(t + 1)$$

- ADAM and its variants are very successful in deep learning
- It is very robust so often results in a network learning which might not learn under standard SGD (with momentum)

ADAM

- Adaptive Moment Estimation (Adam) adapts the scale of the parameters, but also uses momentum
- Update weights

$$w_i(t + 1) = w_i(t) - \frac{\eta}{\sqrt{\hat{S}_i(t + 1)} + \epsilon} \hat{M}_i(t + 1)$$

- ADAM and its variants are very successful in deep learning
- It is very robust so often results in a network learning which might not learn under standard SGD (with momentum)

Covariant Equations

- Vector arithmetic, (matrix multiplication, addition, multiplying by a scale, computing gradients) has a covariant property that it is invariant to the coordinate system
- That is we can translate and rotate our coordinates and get the same result
- That is not true of element-wise multiplication of vectors
- Many ML algorithms do this (including ADAM and adaDelta), but they aren't invariant if we rotate our coordinates

Covariant Equations

- Vector arithmetic, (matrix multiplication, addition, multiplying by a scale, computing gradients) has a covariant property that it is invariant to the coordinate system
- That is we can translate and rotate our coordinates and get the same result
- That is not true of element-wise multiplication of vectors
- Many ML algorithms do this (including ADAM and adaDelta), but they aren't invariant if we rotate our coordinates

Covariant Equations

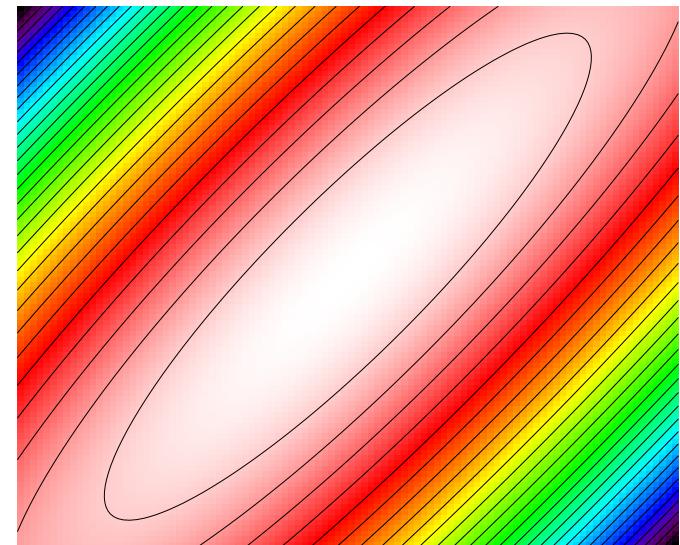
- Vector arithmetic, (matrix multiplication, addition, multiplying by a scale, computing gradients) has a covariant property that it is invariant to the coordinate system
- That is we can translate and rotate our coordinates and get the same result
- That is not true of element-wise multiplication of vectors
- Many ML algorithms do this (including ADAM and adaDelta), but they aren't invariant if we rotate our coordinates

Covariant Equations

- Vector arithmetic, (matrix multiplication, addition, multiplying by a scale, computing gradients) has a covariant property that it is invariant to the coordinate system
- That is we can translate and rotate our coordinates and get the same result
- That is not true of element-wise multiplication of vectors
- Many ML algorithms do this (including ADAM and adaDelta), but they aren't invariant if we rotate our coordinates

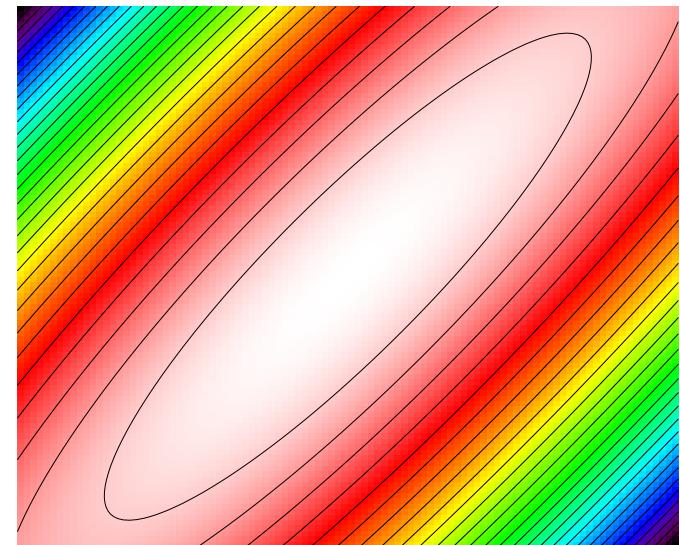
Covariant Equations

- Vector arithmetic, (matrix multiplication, addition, multiplying by a scale, computing gradients) has a covariant property that it is invariant to the coordinate system
- That is we can translate and rotate our coordinates and get the same result
- That is not true of element-wise multiplication of vectors
- Many ML algorithms do this (including ADAM and adaDelta), but they aren't invariant if we rotate our coordinates



Covariant Equations

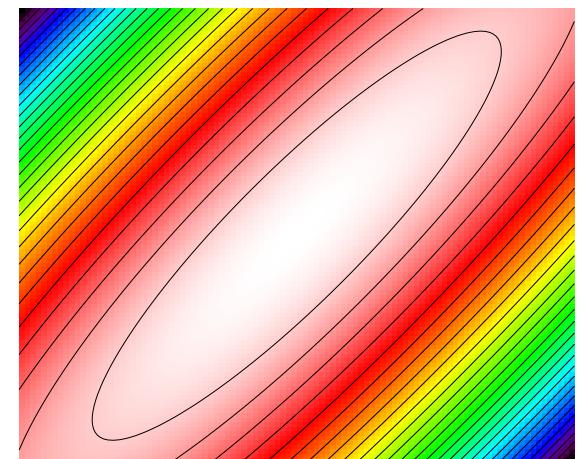
- Vector arithmetic, (matrix multiplication, addition, multiplying by a scale, computing gradients) has a covariant property that it is invariant to the coordinate system
- That is we can translate and rotate our coordinates and get the same result
- That is not true of element-wise multiplication of vectors
- Many ML algorithms do this (including ADAM and adaDelta), but they aren't invariant if we rotate our coordinates



Can't rescale
coordinates

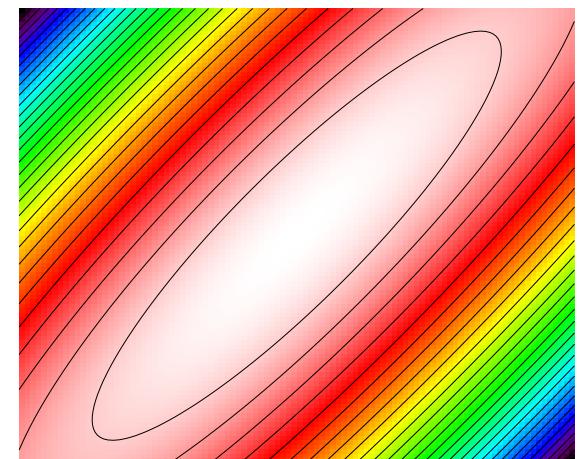
Correlated Weights

- Correlated weights will lead to skewed valleys
- Adaptive methods would be more efficient if they were rotationally invariant
- However, this would slow down the algorithm
- ADAM is a compromise between speed of implementation and effectiveness



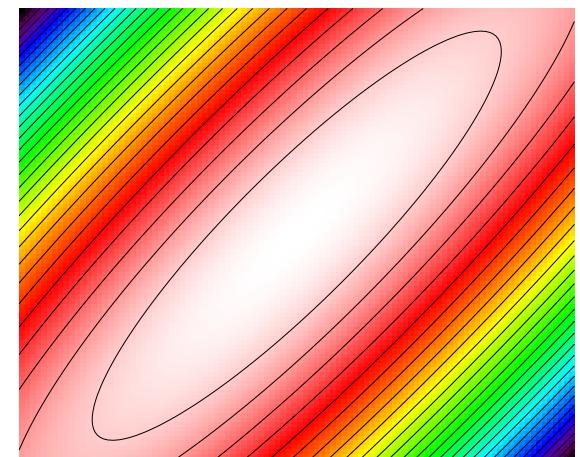
Correlated Weights

- Correlated weights will lead to skewed valleys
- Adaptive methods would be more efficient if they were rotationally invariant
- However, this would slow down the algorithm
- ADAM is a compromise between speed of implementation and effectiveness



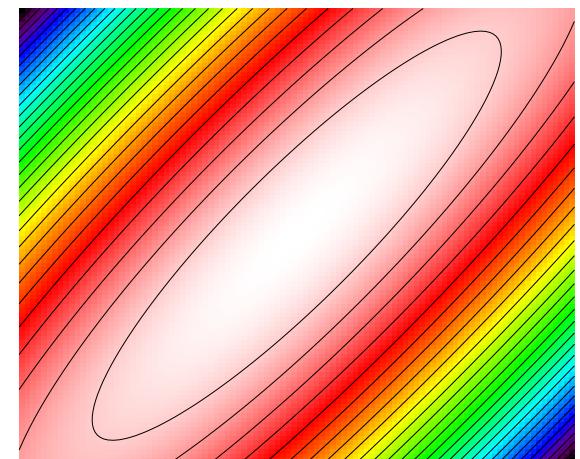
Correlated Weights

- Correlated weights will lead to skewed valleys
- Adaptive methods would be more efficient if they were rotationally invariant
- However, this would slow down the algorithm
- ADAM is a compromise between speed of implementation and effectiveness



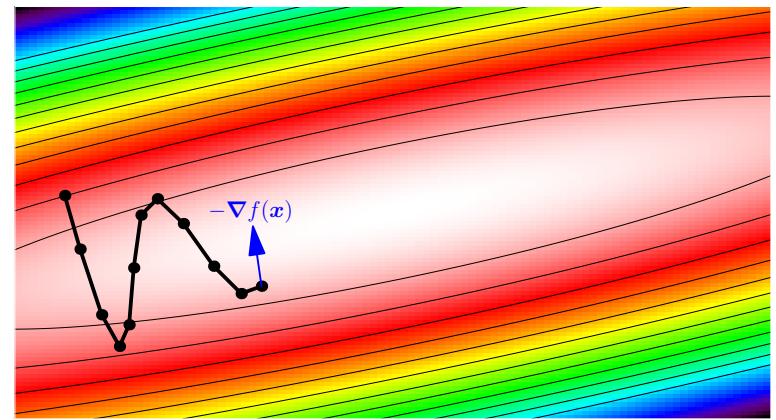
Correlated Weights

- Correlated weights will lead to skewed valleys
- Adaptive methods would be more efficient if they were rotationally invariant
- However, this would slow down the algorithm
- ADAM is a compromise between speed of implementation and effectiveness



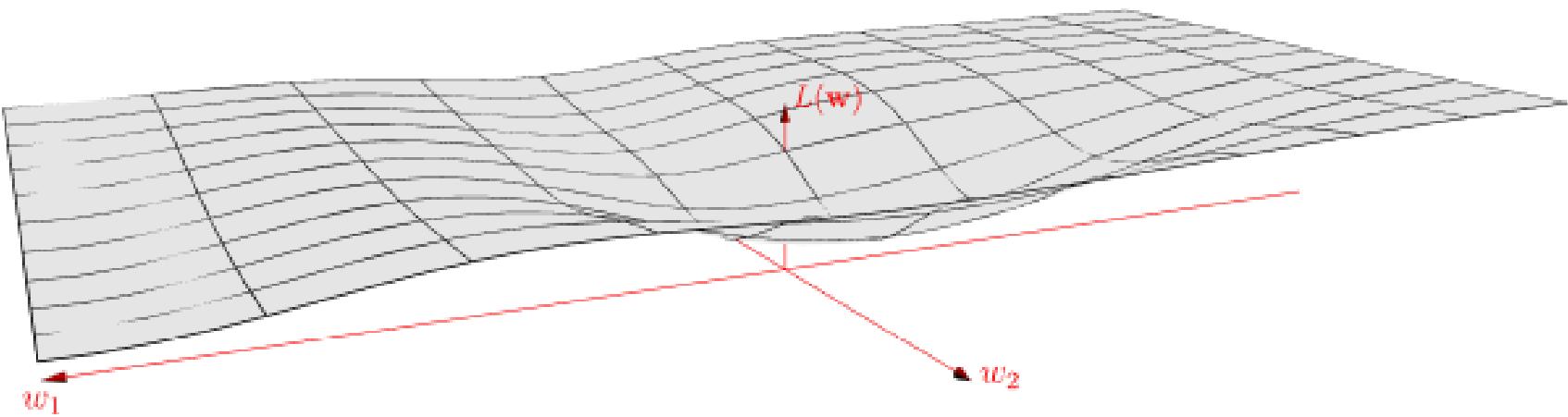
Outline

1. SGD
2. Momentum
3. Loss Landscapes



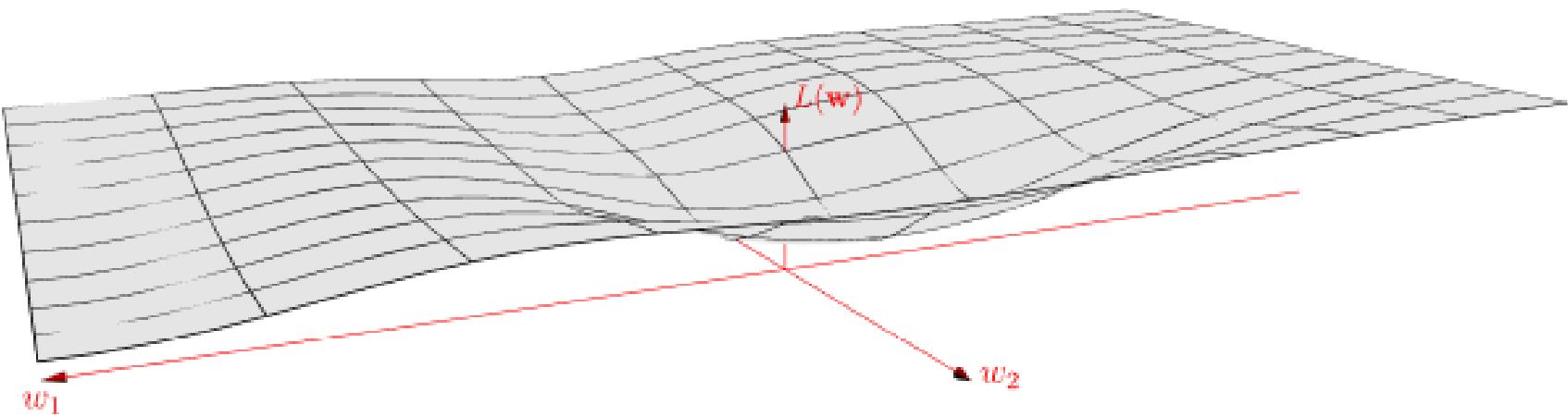
Loss Landscape

- A useful concept for understanding optimisation is the loss landscape
- For every value of the weights w there is some loss associated with it
- Note that this landscape is very high dimensional, so our intuition can be a bit misleading
- The landscape is also huge



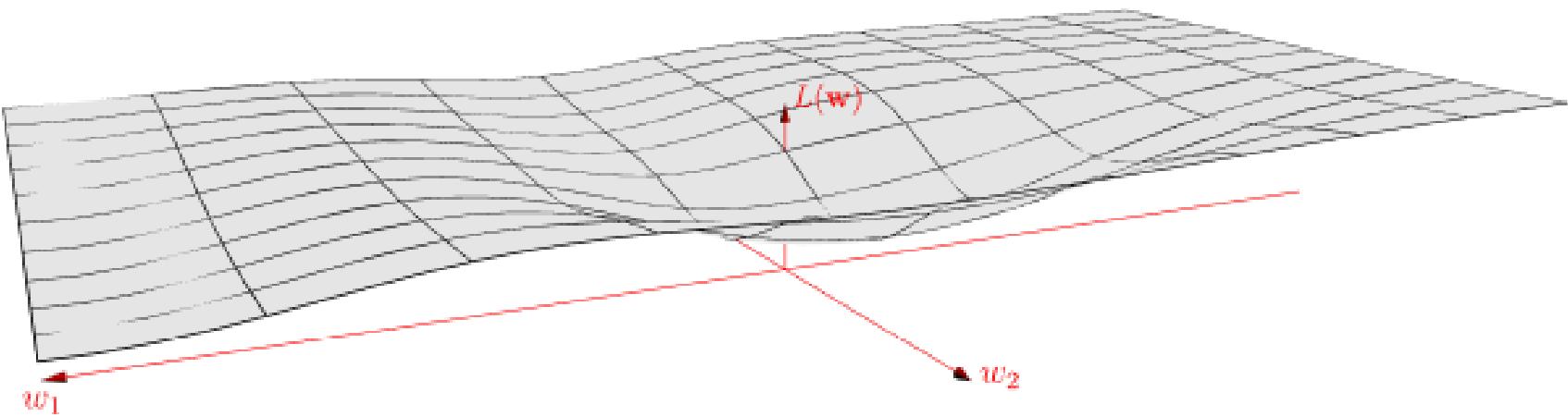
Loss Landscape

- A useful concept for understanding optimisation is the loss landscape
- For every value of the weights w there is some loss associated with it
- Note that this landscape is very high dimensional, so our intuition can be a bit misleading
- The landscape is also huge



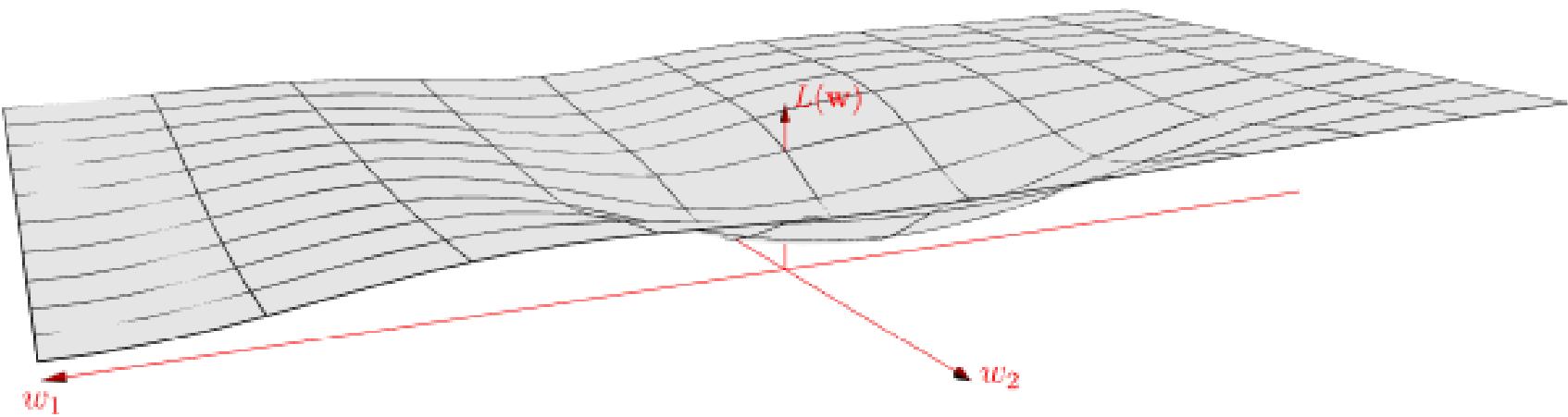
Loss Landscape

- A useful concept for understanding optimisation is the loss landscape
- For every value of the weights w there is some loss associated with it
- Note that this landscape is very high dimensional, so our intuition can be a bit misleading
- The landscape is also huge



Loss Landscape

- A useful concept for understanding optimisation is the loss landscape
- For every value of the weights w there is some loss associated with it
- Note that this landscape is very high dimensional, so our intuition can be a bit misleading
- The landscape is also huge



Global and Local Minima

- Our objective is to minimise our loss
- This would mean finding the global minimum
- There are no algorithms that are guaranteed to find the global minimum
- The best we can hope is to find a local minimum by doing gradient descent
- In practice, our landscapes are so large that we are probably unable to find even a local minimum

Global and Local Minima

- Our objective is to minimise our loss
- This would mean finding the global minimum
- There are no algorithms that are guaranteed to find the global minimum
- The best we can hope is to find a local minimum by doing gradient descent
- In practice, our landscapes are so large that we are probably unable to find even a local minimum

Global and Local Minima

- Our objective is to minimise our loss
- This would mean finding the global minimum
- There are no algorithms that are guaranteed to find the global minimum
- The best we can hope is to find a local minimum by doing gradient descent
- In practice, our landscapes are so large that we are probably unable to find even a local minimum

Global and Local Minima

- Our objective is to minimise our loss
- This would mean finding the global minimum
- There are no algorithms that are guaranteed to find the global minimum
- The best we can hope is to find a local minimum by doing gradient descent
- In practice, our landscapes are so large that we are probably unable to find even a local minimum

Global and Local Minima

- Our objective is to minimise our loss
- This would mean finding the global minimum
- There are no algorithms that are guaranteed to find the global minimum
- The best we can hope is to find a local minimum by doing gradient descent
- In practice, our landscapes are so large that we are probably unable to find even a local minimum

Noisy Optimisation

- Because we are using mini-batches we aren't actually following the correct gradients
- If we reduce our step size then our average gradient is closer to the true gradient
- Thus our optimisation is inherently noisy
- At least for deep learning there is no evidence that the weights stop changing even after learning for a huge amount of time

Noisy Optimisation

- Because we are using mini-batches we aren't actually following the correct gradients
- If we reduce our step size then our average gradient is closer to the true gradient
- Thus our optimisation is inherently noisy
- At least for deep learning there is no evidence that the weights stop changing even after learning for a huge amount of time

Noisy Optimisation

- Because we are using mini-batches we aren't actually following the correct gradients
- If we reduce our step size then our average gradient is closer to the true gradient
- Thus our optimisation is inherently noisy
- At least for deep learning there is no evidence that the weights stop changing even after learning for a huge amount of time

Noisy Optimisation

- Because we are using mini-batches we aren't actually following the correct gradients
- If we reduce our step size then our average gradient is closer to the true gradient
- Thus our optimisation is inherently noisy
- At least for deep learning there is no evidence that the weights stop changing even after learning for a huge amount of time

Valleys in Valleys in Valleys

- My view of the loss landscape is that we have valleys inside bigger valleys inside bigger valleys, and so on
- If our noise is high (we use large step size), we can navigate away from local valleys and move towards the big valley
- But, we can't explore the small valleys
- Often when the rate of improvement slows down, practitioners will reduce their learning rate by a factor of 10 and the rate of improvement jumps up
- Some practitioners cycle through raising and lowering their learning rates which may speed up convergence

Valleys in Valleys in Valleys

- My view of the loss landscape is that we have valleys inside bigger valleys inside bigger valleys, and so on
- If our noise is high (we use large step size), we can navigate away from local valleys and move towards the big valley
- But, we can't explore the small valleys
- Often when the rate of improvement slows down, practitioners will reduce their learning rate by a factor of 10 and the rate of improvement jumps up
- Some practitioners cycle through raising and lowering their learning rates which may speed up convergence

Valleys in Valleys in Valleys

- My view of the loss landscape is that we have valleys inside bigger valleys inside bigger valleys, and so on
- If our noise is high (we use large step size), we can navigate away from local valleys and move towards the big valley
- But, we can't explore the small valleys
- Often when the rate of improvement slows down, practitioners will reduce their learning rate by a factor of 10 and the rate of improvement jumps up
- Some practitioners cycle through raising and lowering their learning rates which may speed up convergence

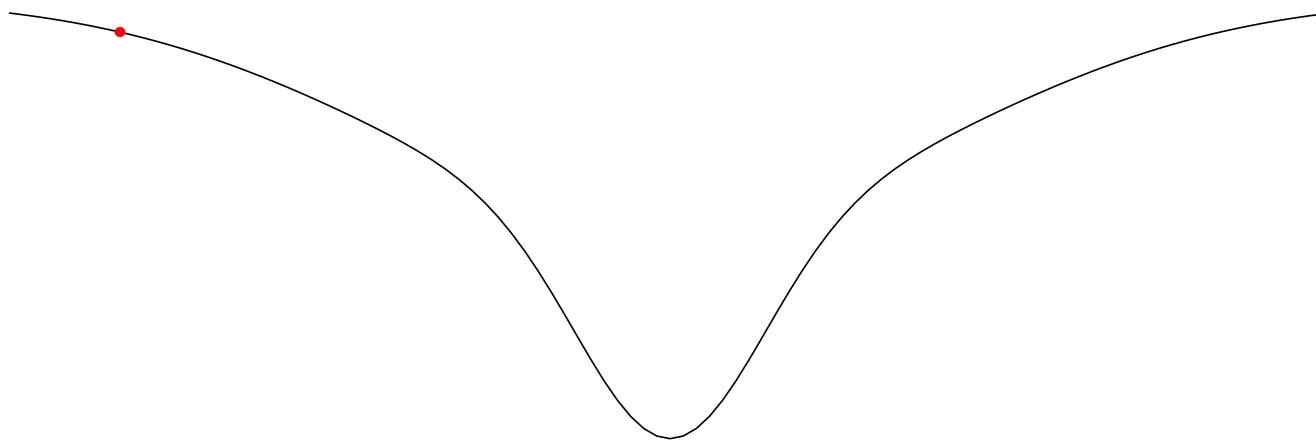
Valleys in Valleys in Valleys

- My view of the loss landscape is that we have valleys inside bigger valleys inside bigger valleys, and so on
- If our noise is high (we use large step size), we can navigate away from local valleys and move towards the big valley
- But, we can't explore the small valleys
- Often when the rate of improvement slows down, practitioners will reduce their learning rate by a factor of 10 and the rate of improvement jumps up
- Some practitioners cycle through raising and lowering their learning rates which may speed up convergence

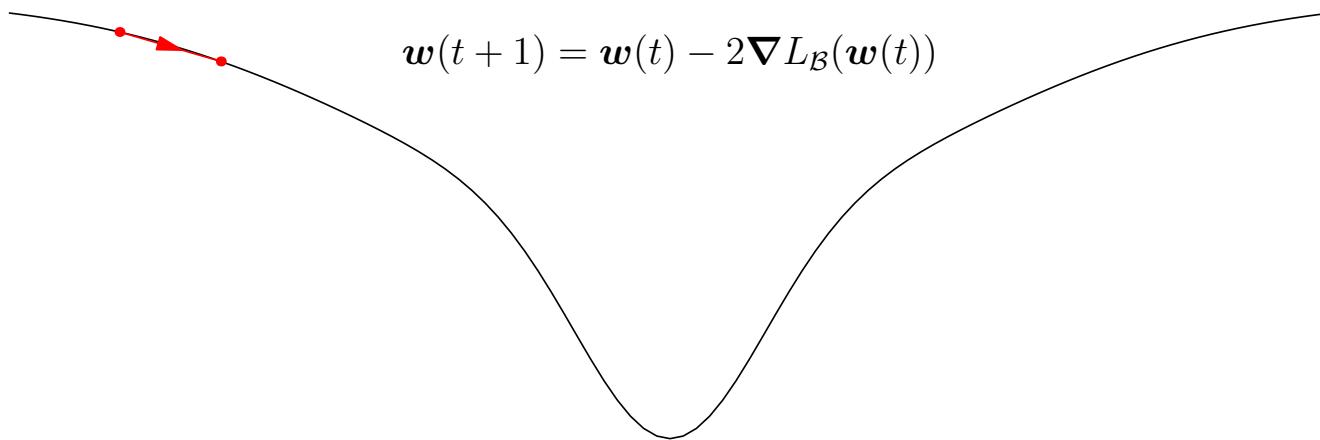
Valleys in Valleys in Valleys

- My view of the loss landscape is that we have valleys inside bigger valleys inside bigger valleys, and so on
- If our noise is high (we use large step size), we can navigate away from local valleys and move towards the big valley
- But, we can't explore the small valleys
- Often when the rate of improvement slows down, practitioners will reduce their learning rate by a factor of 10 and the rate of improvement jumps up
- Some practitioners cycle through raising and lowering their learning rates which may speed up convergence

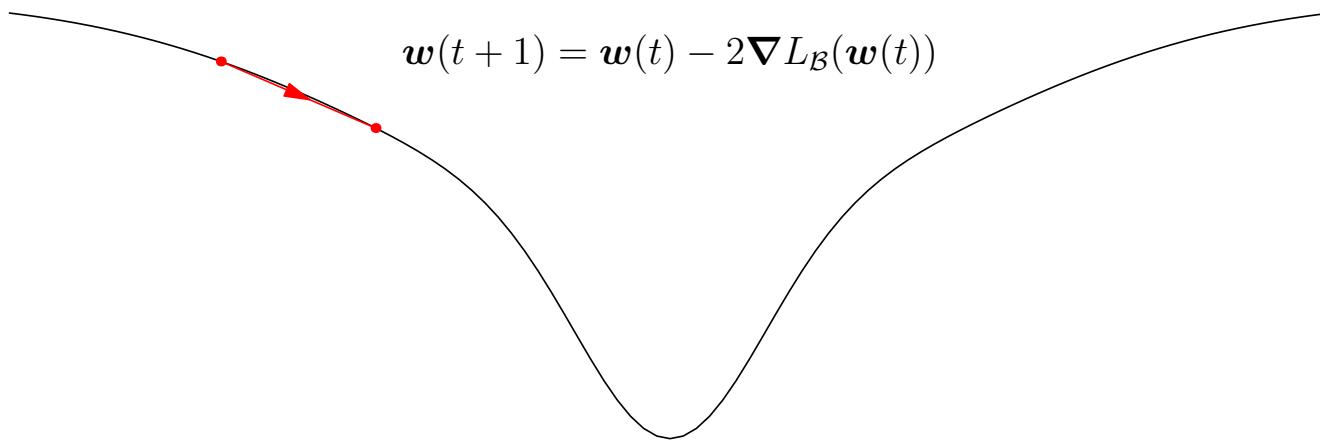
Reducing the Step Size



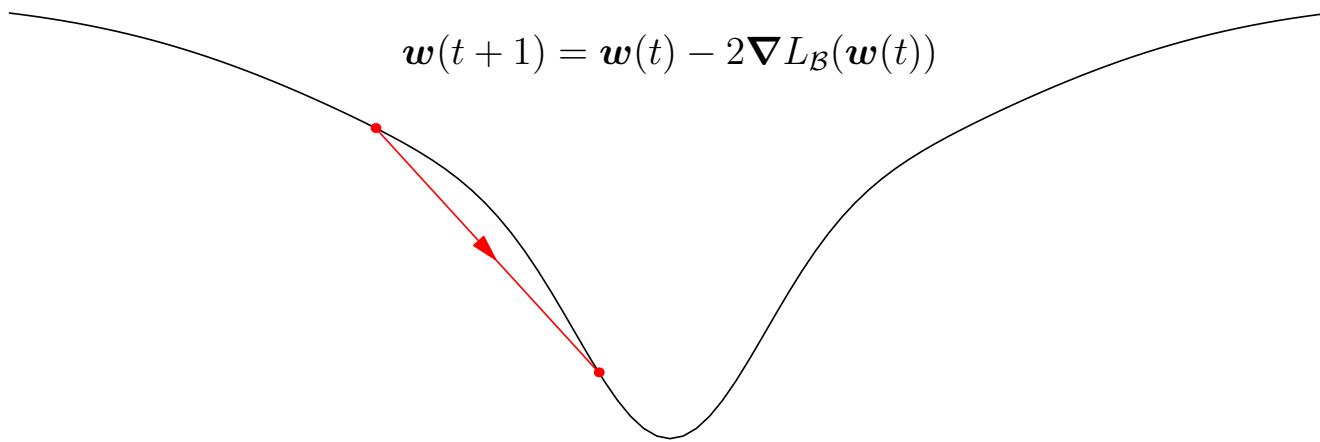
Reducing the Step Size



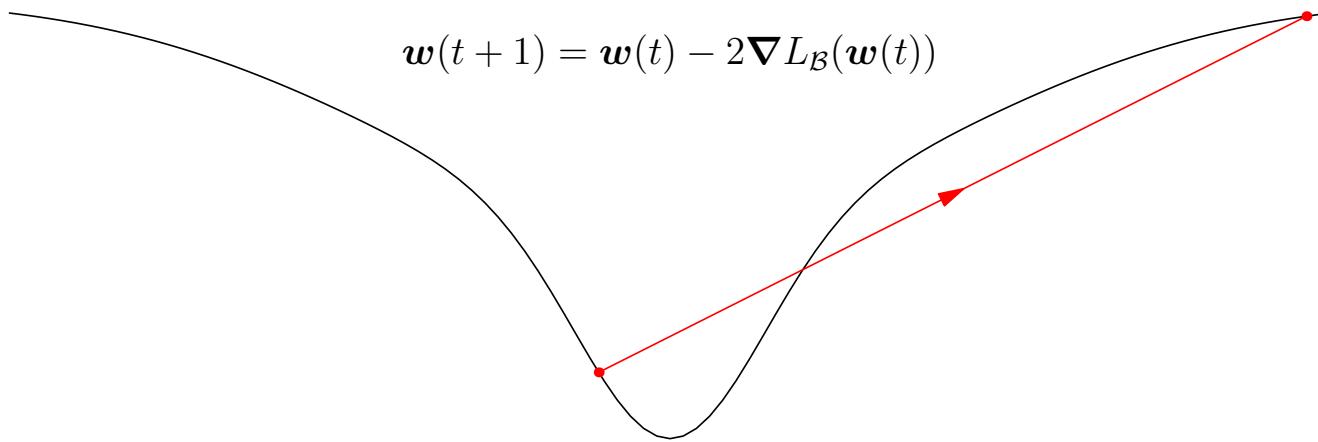
Reducing the Step Size



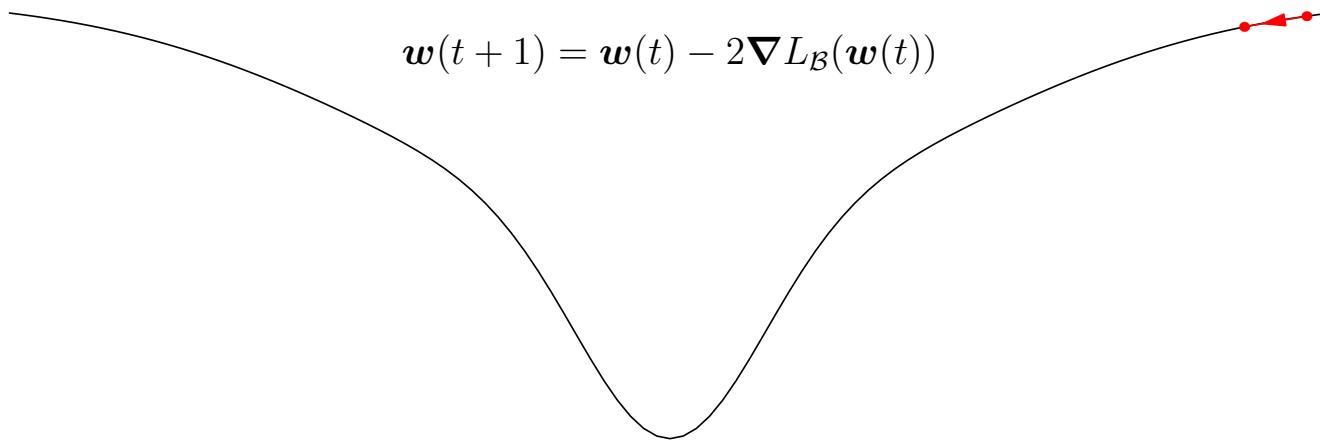
Reducing the Step Size



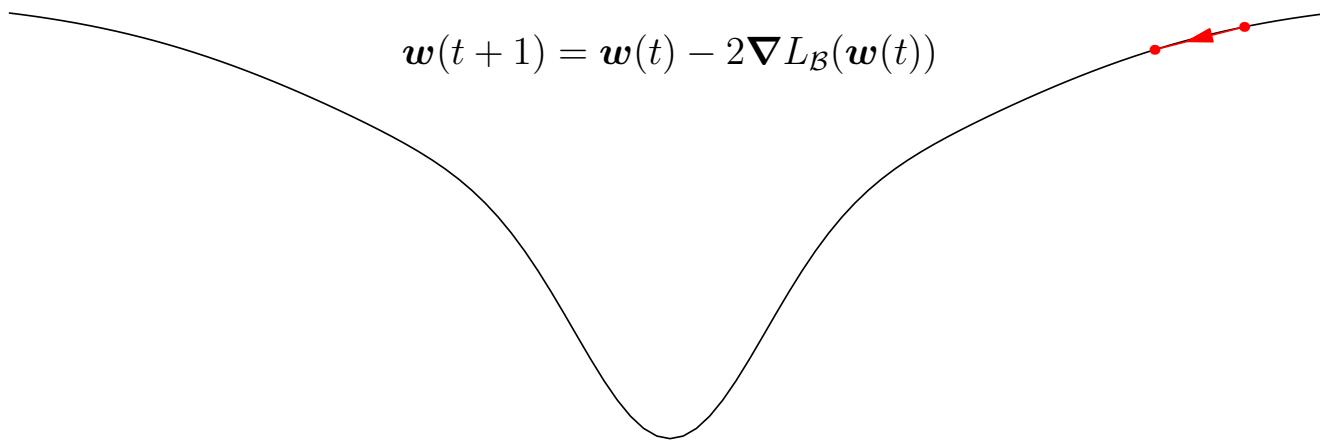
Reducing the Step Size



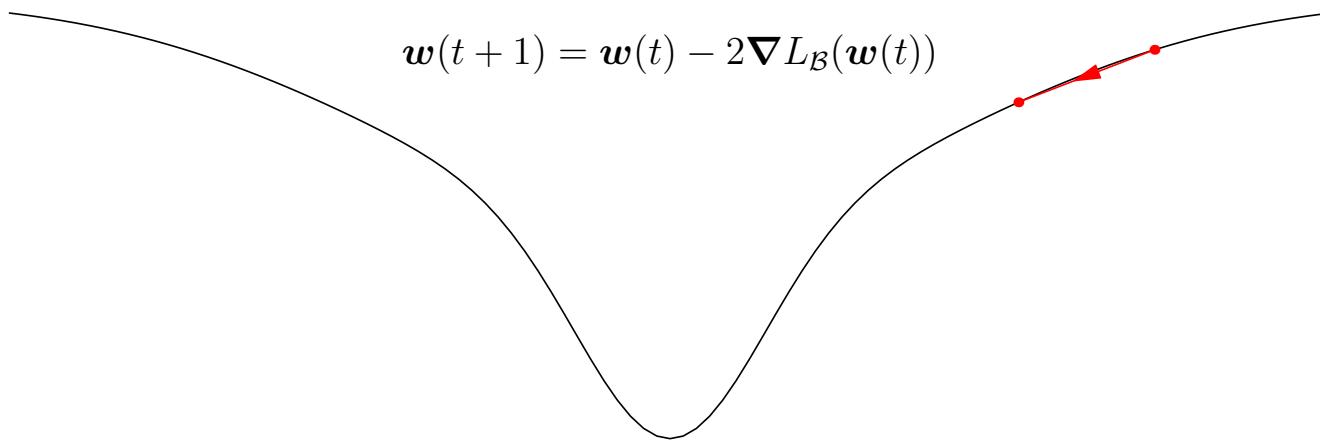
Reducing the Step Size



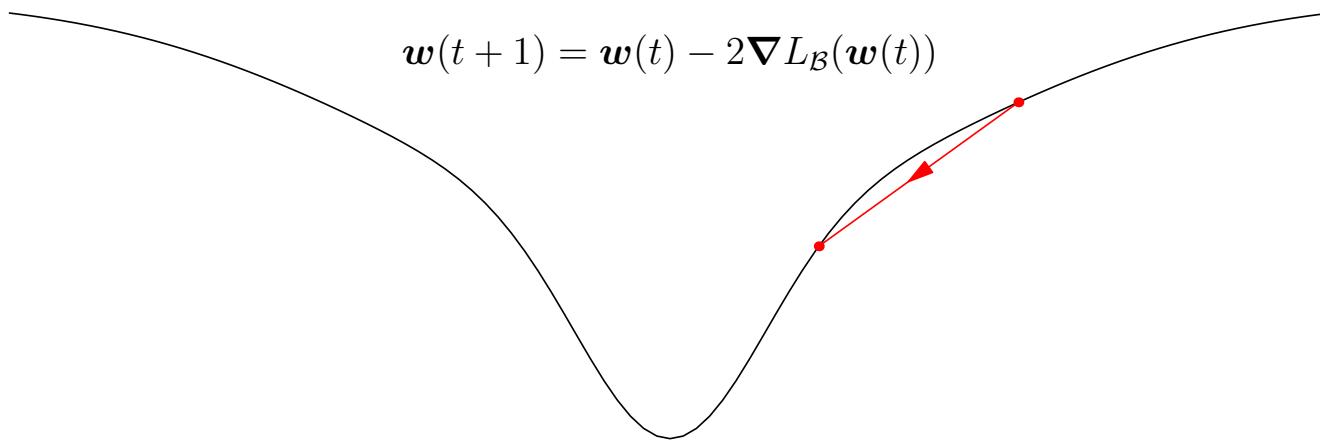
Reducing the Step Size



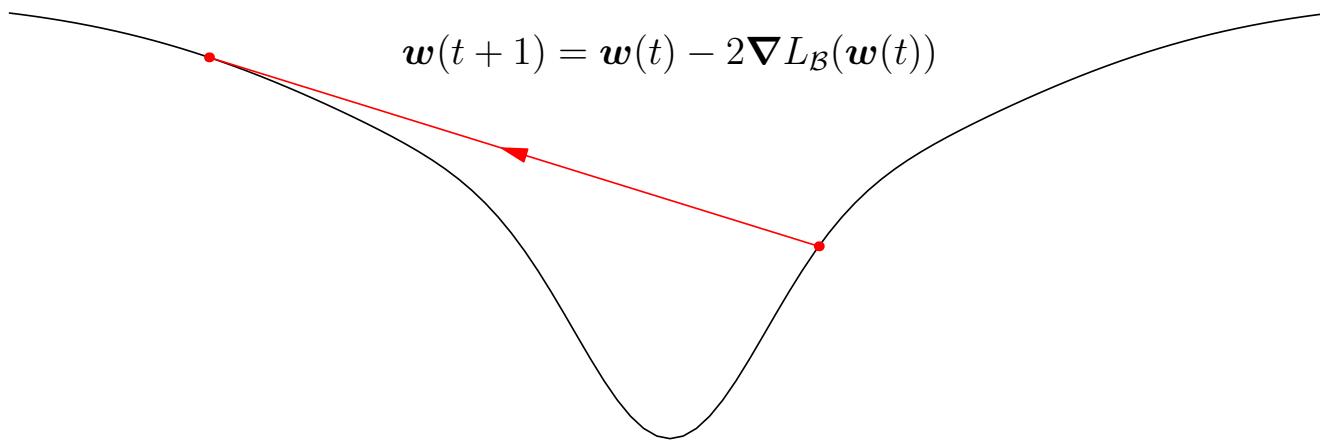
Reducing the Step Size



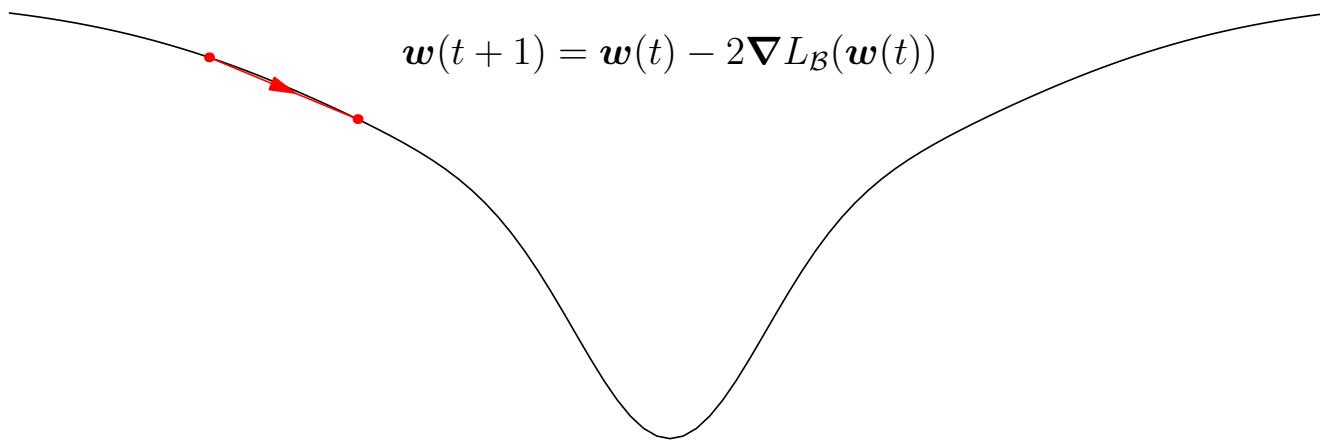
Reducing the Step Size



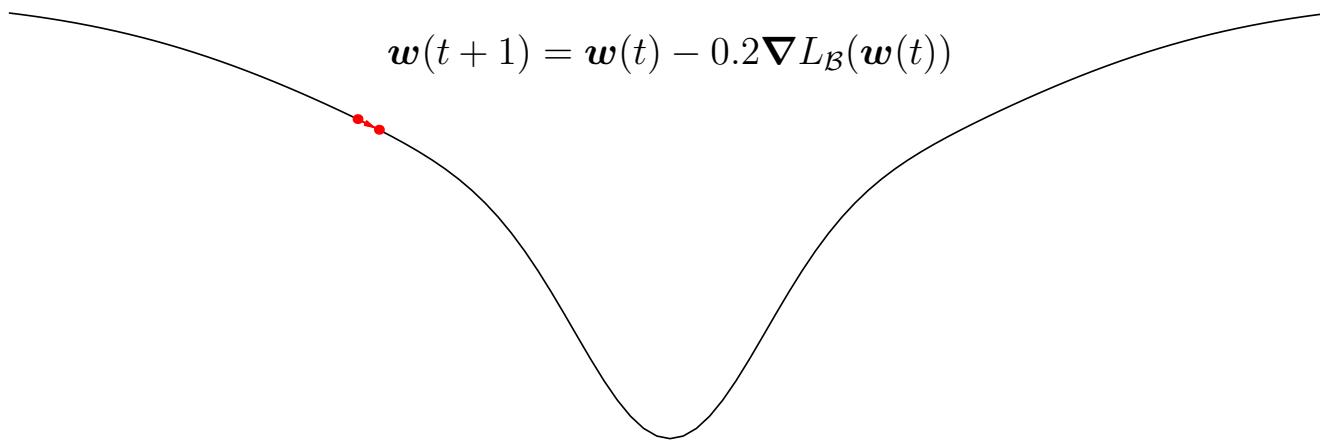
Reducing the Step Size



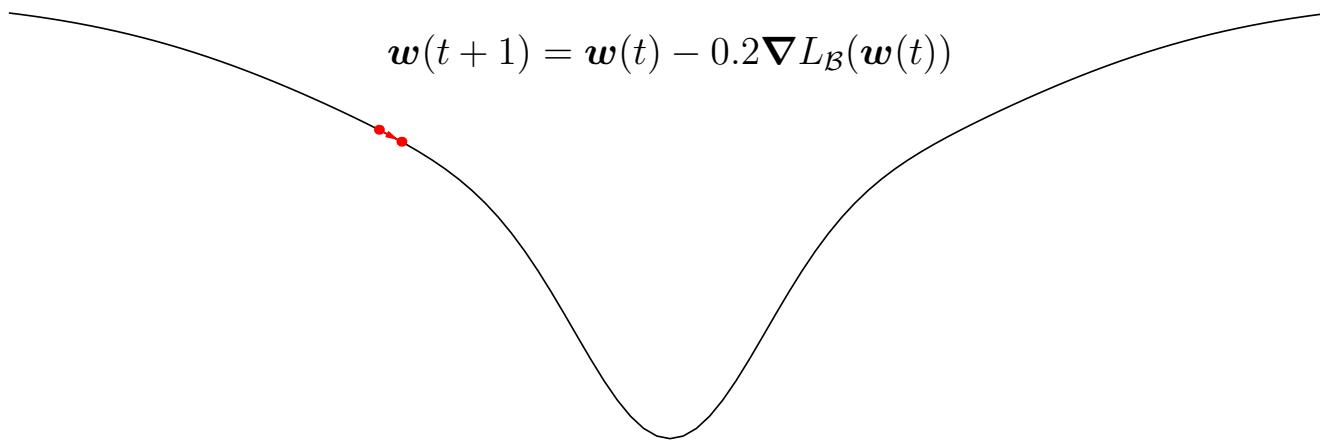
Reducing the Step Size



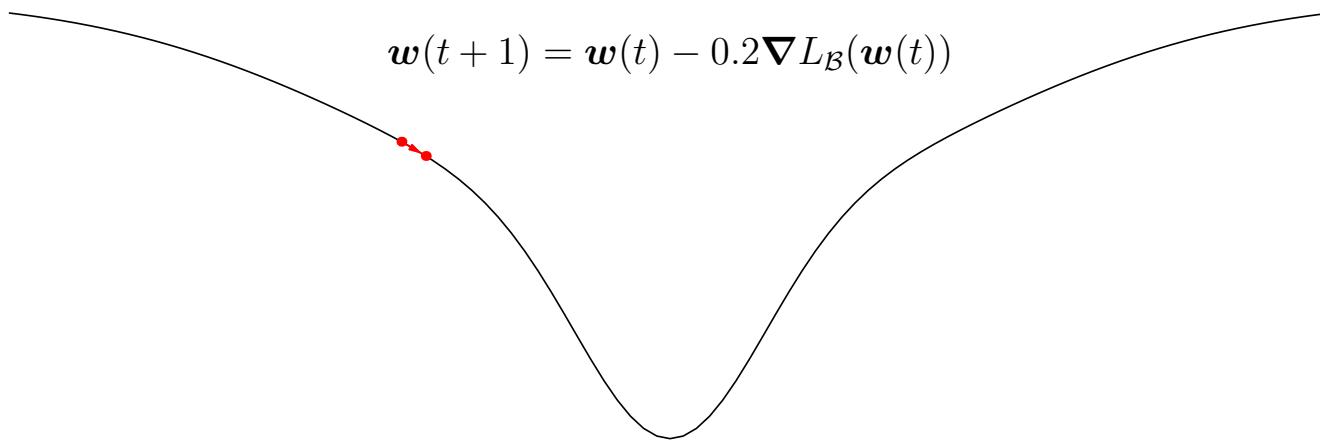
Reducing the Step Size



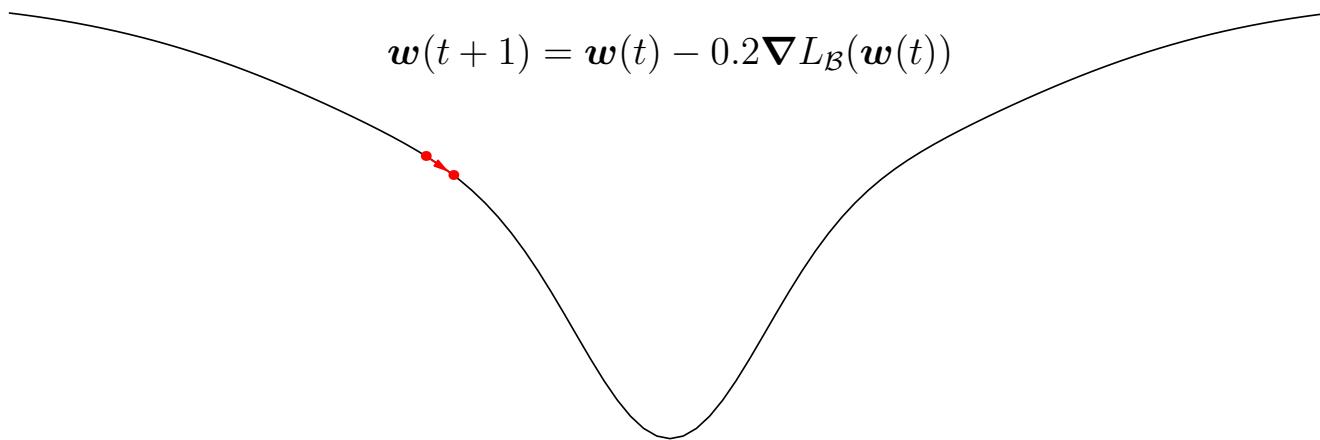
Reducing the Step Size



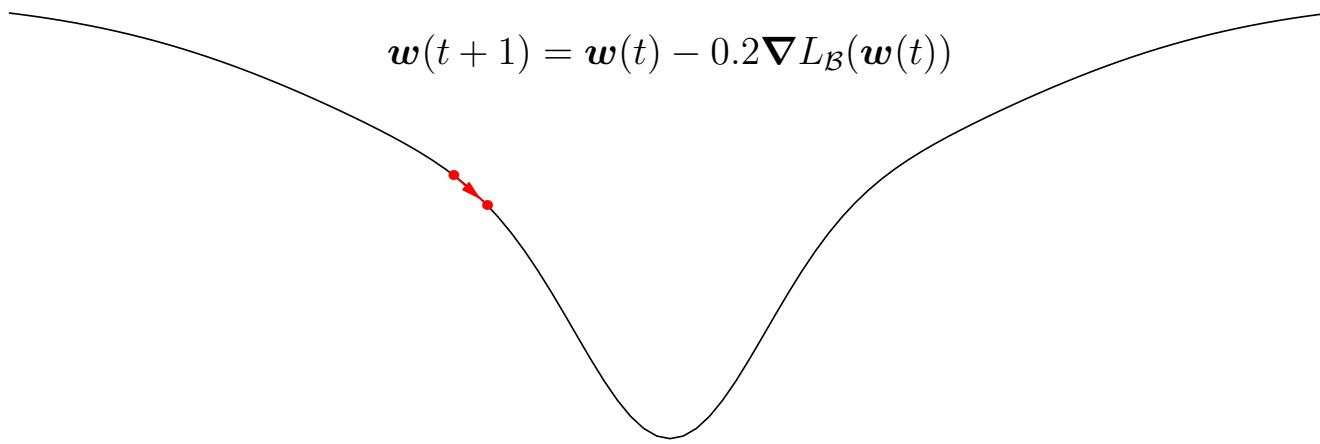
Reducing the Step Size



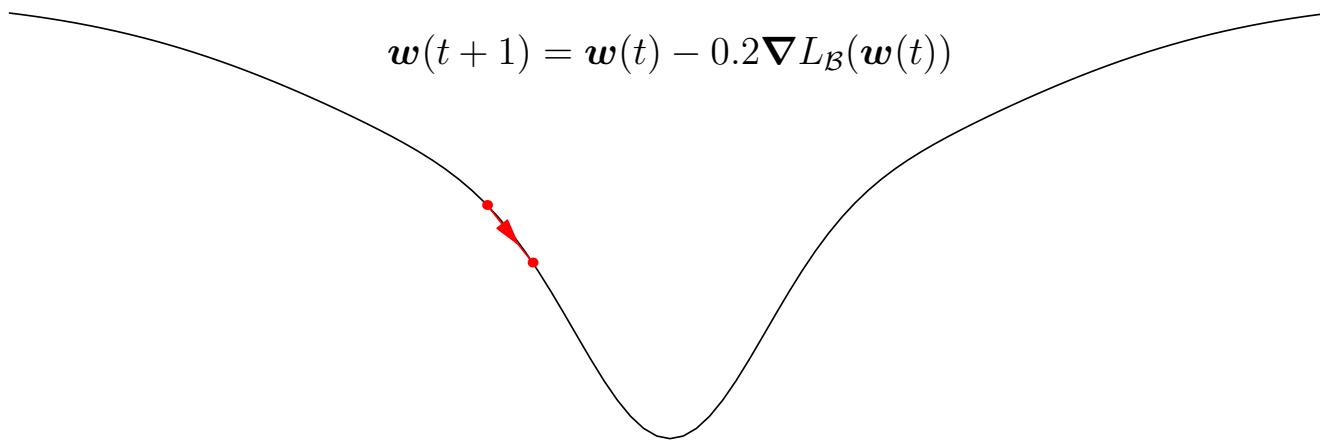
Reducing the Step Size



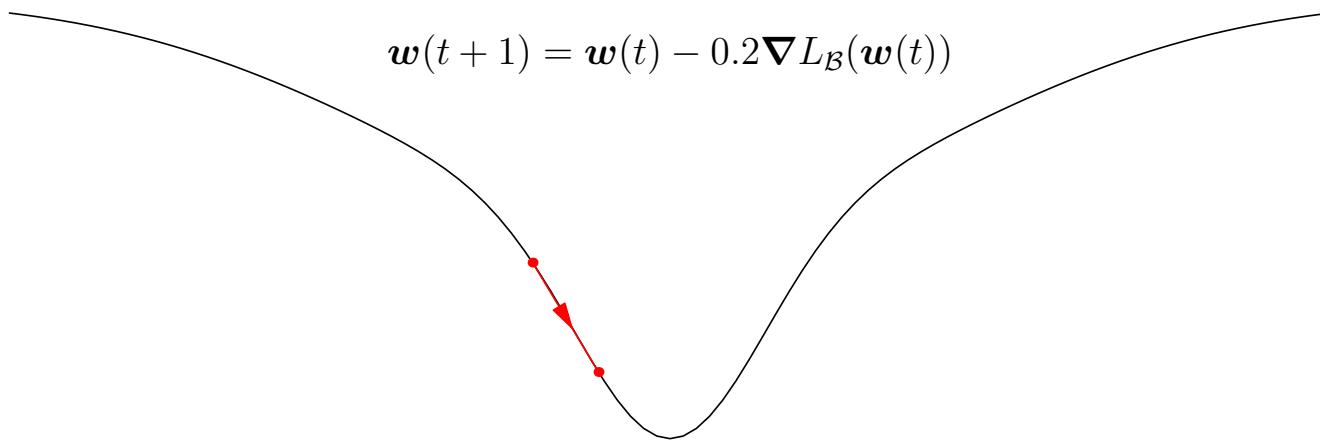
Reducing the Step Size



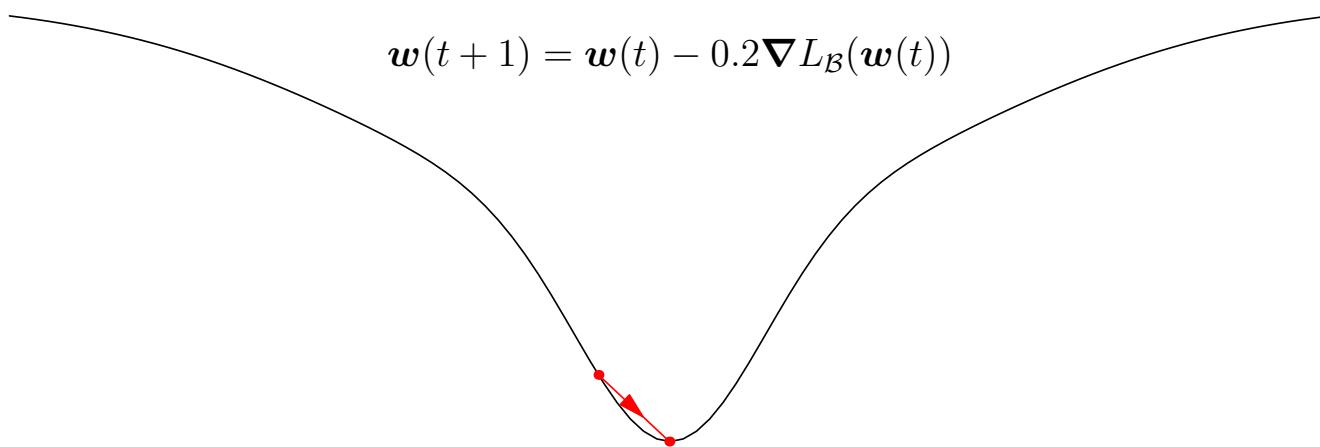
Reducing the Step Size



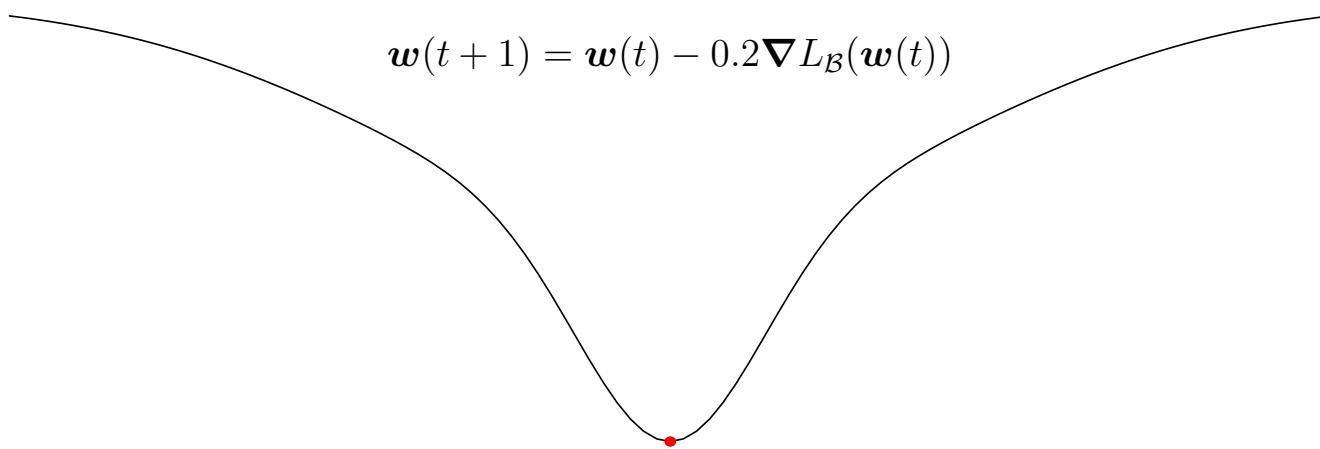
Reducing the Step Size



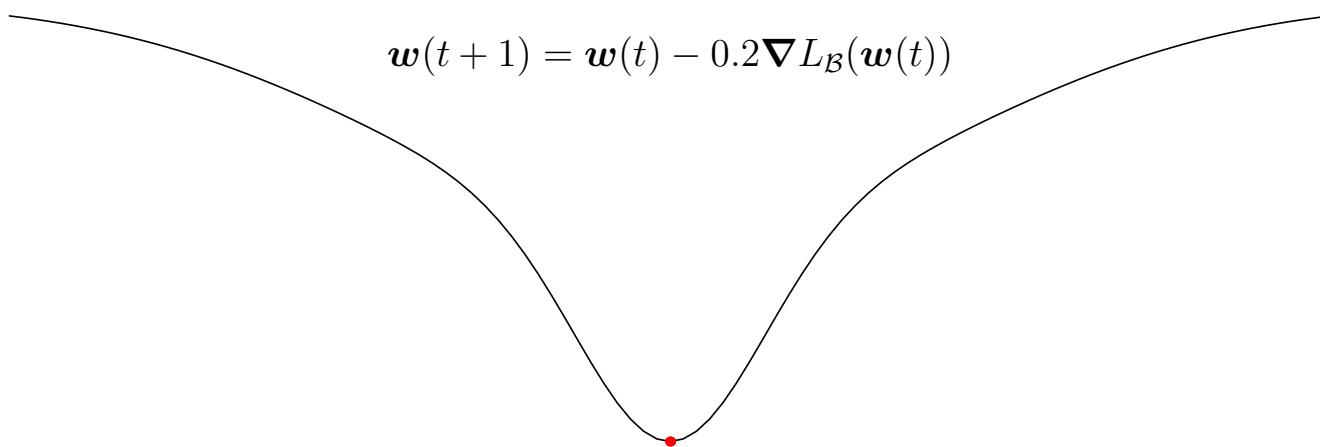
Reducing the Step Size



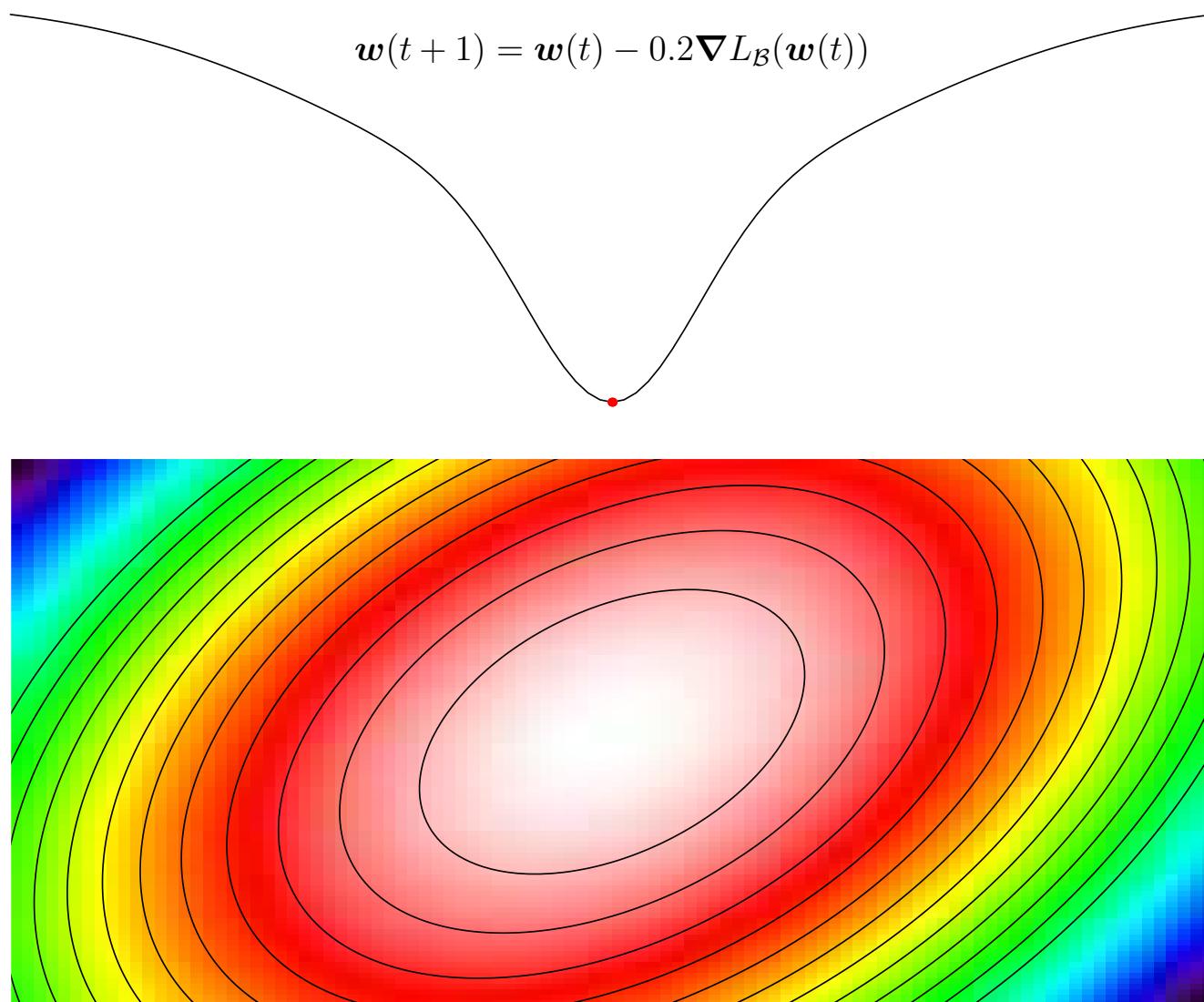
Reducing the Step Size



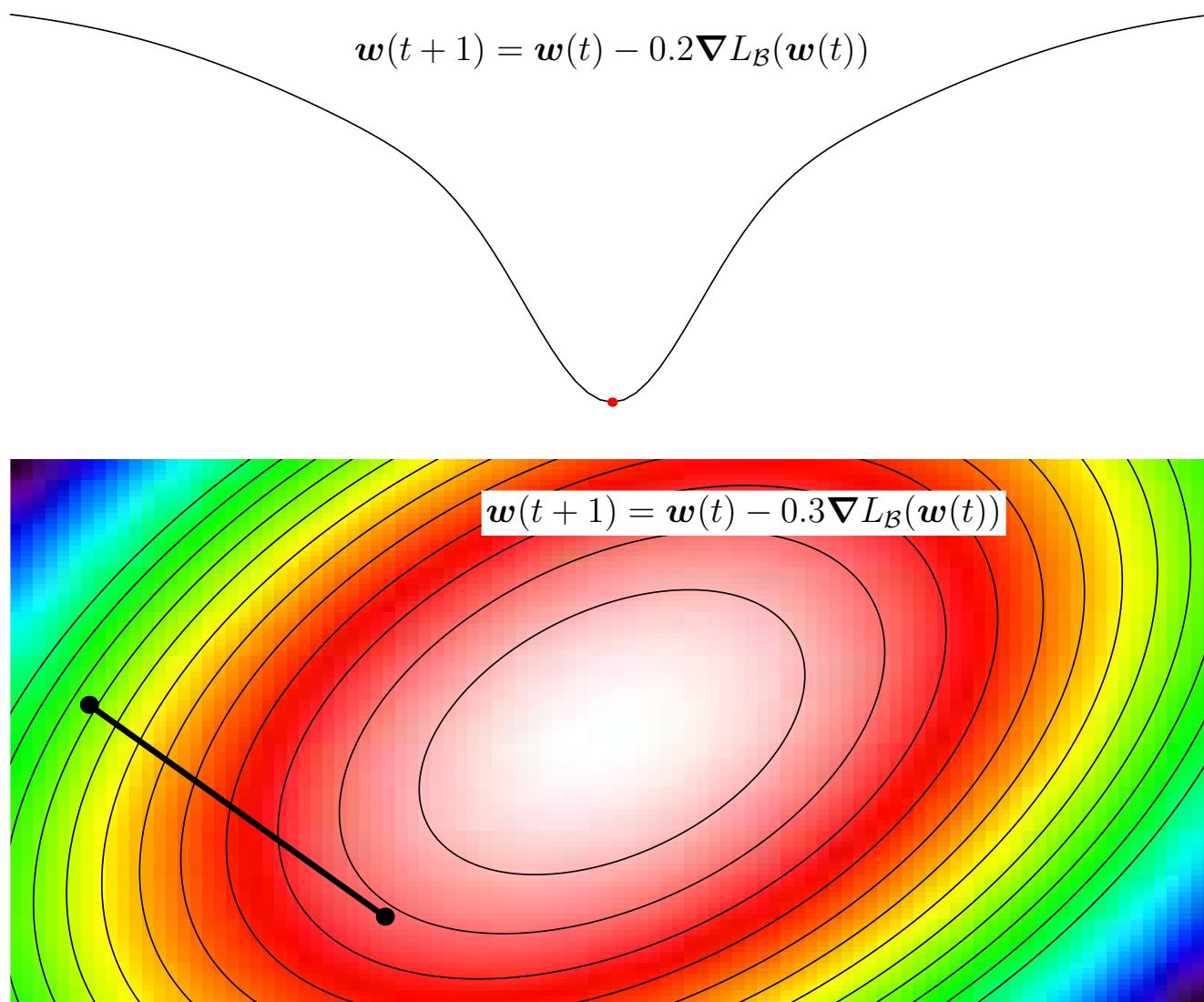
Reducing the Step Size



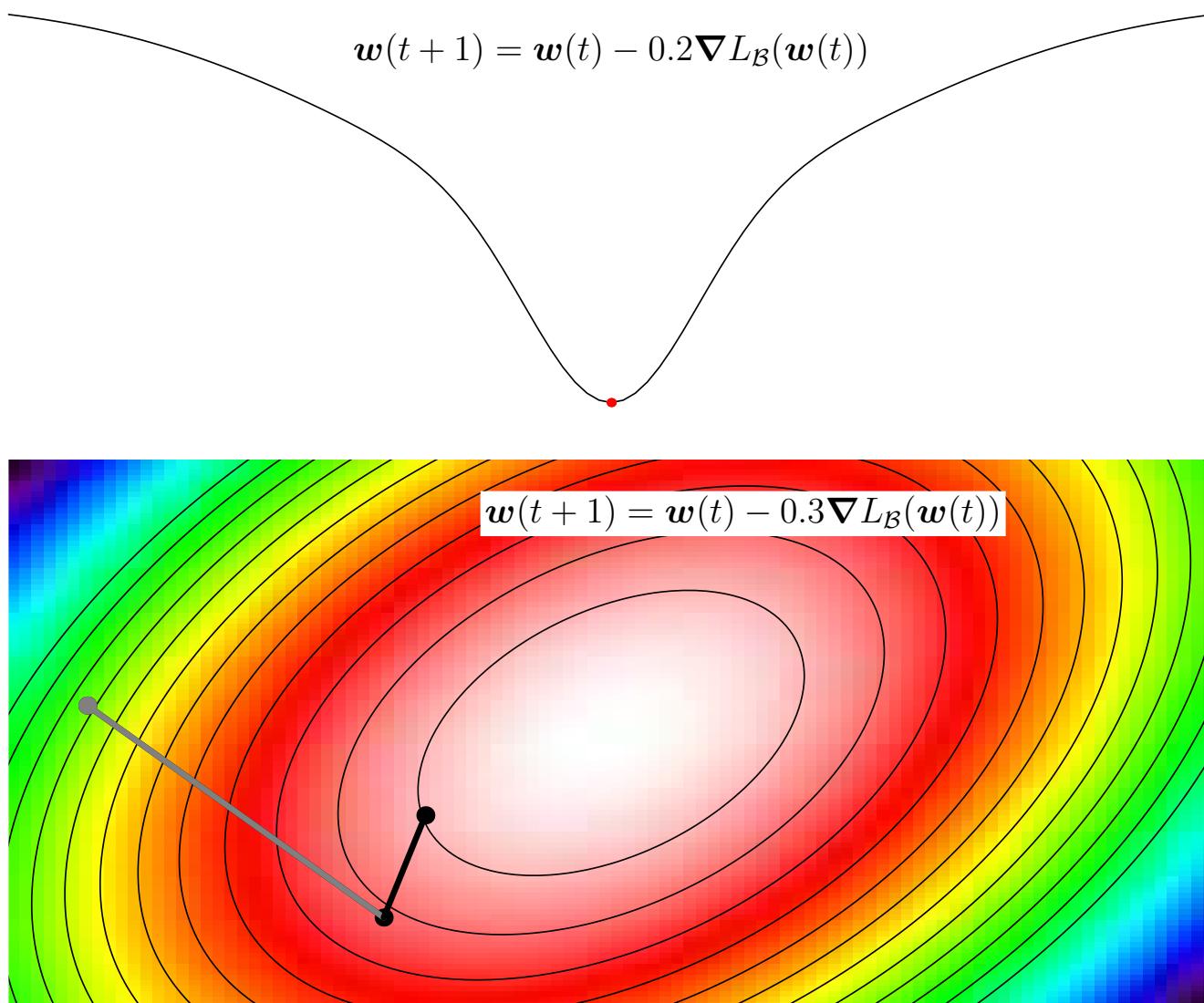
Reducing the Step Size



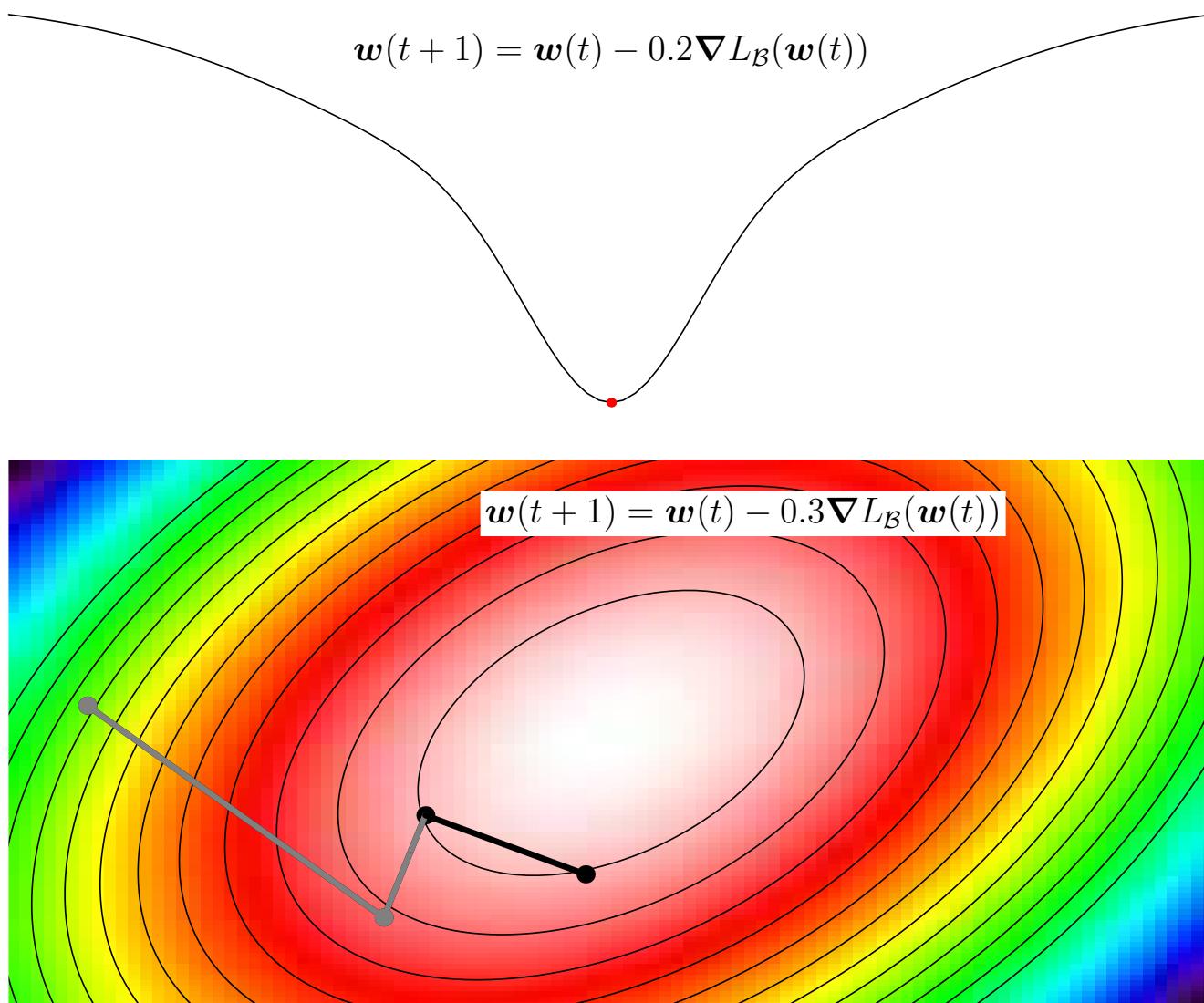
Reducing the Step Size



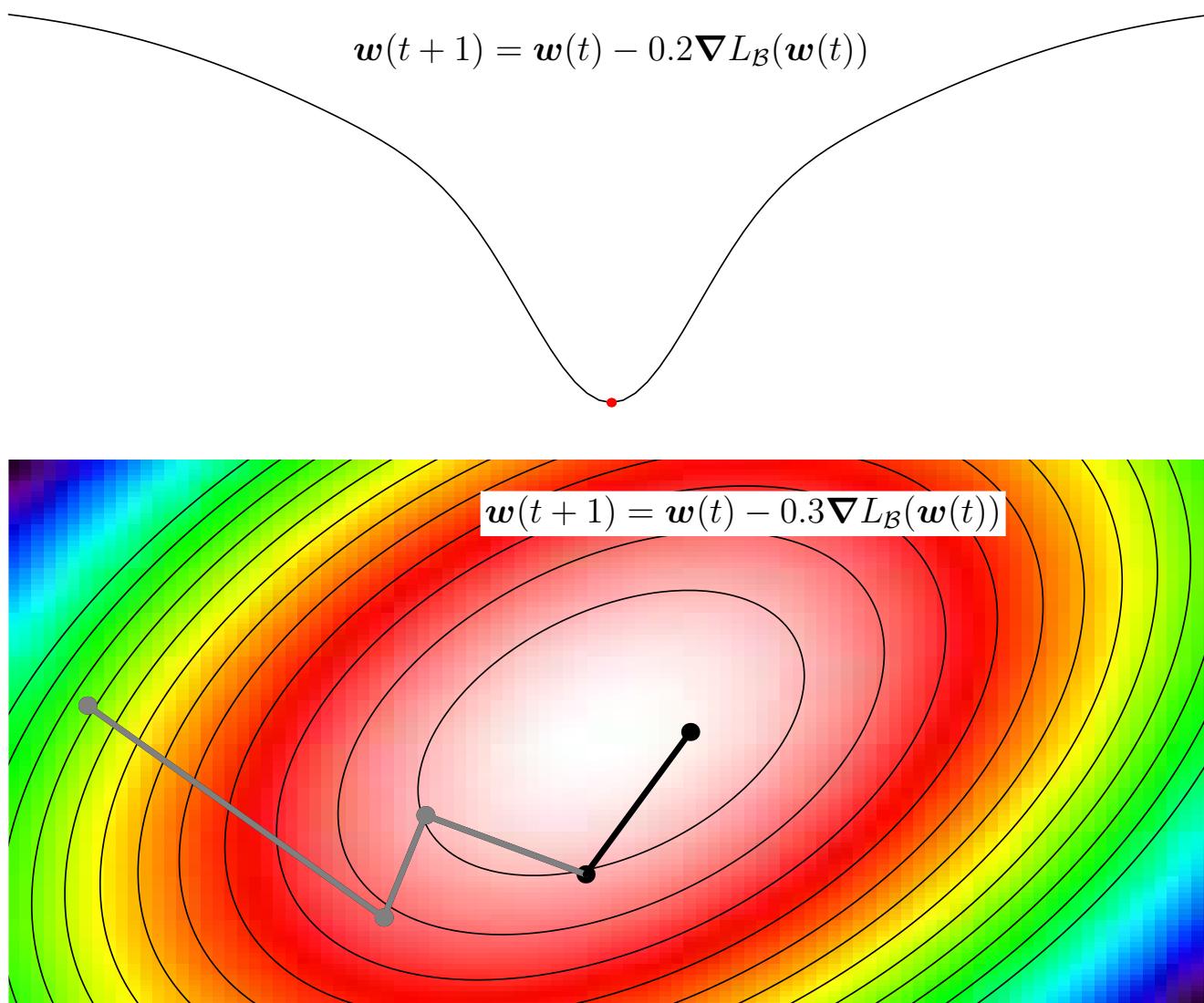
Reducing the Step Size



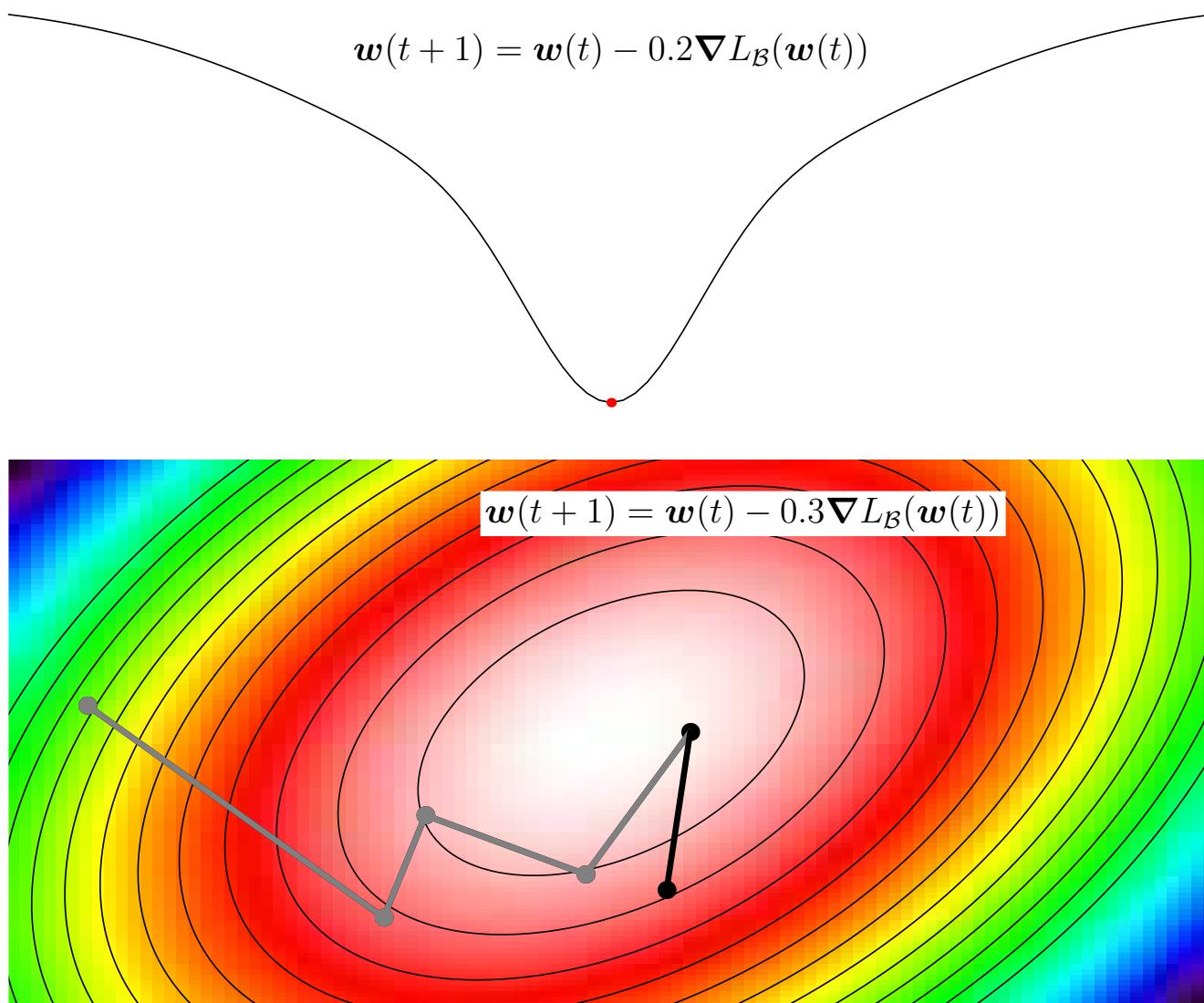
Reducing the Step Size



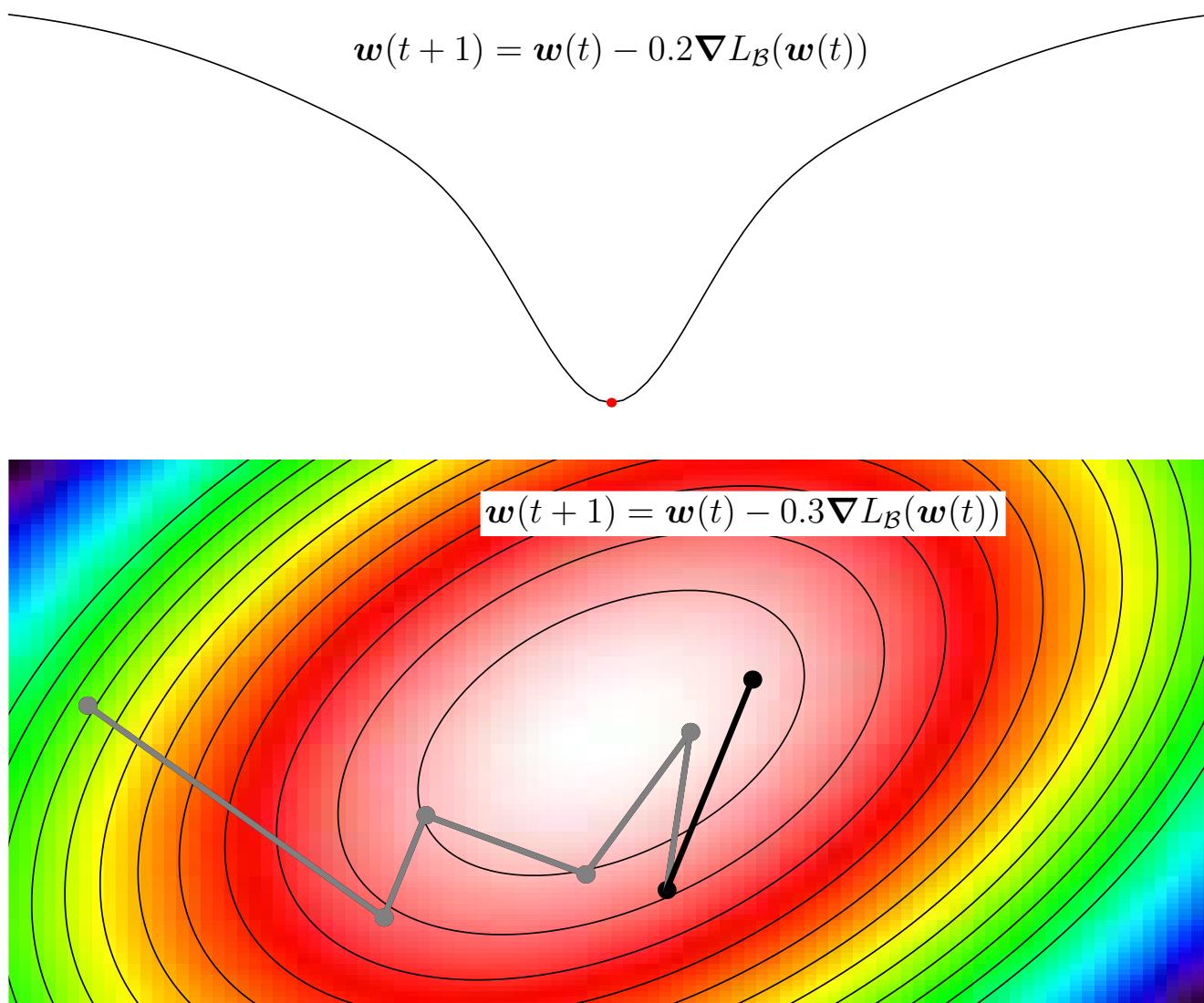
Reducing the Step Size



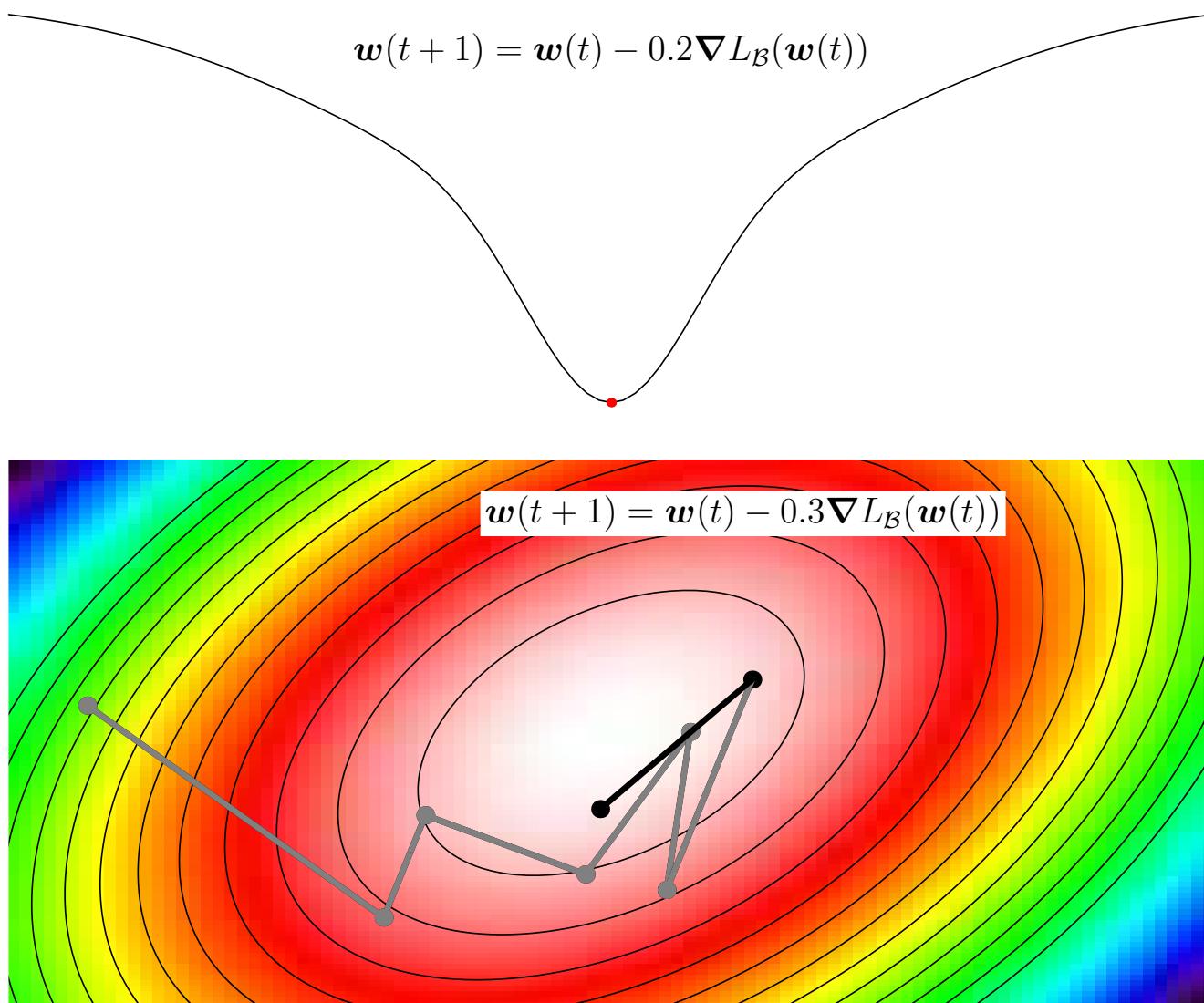
Reducing the Step Size



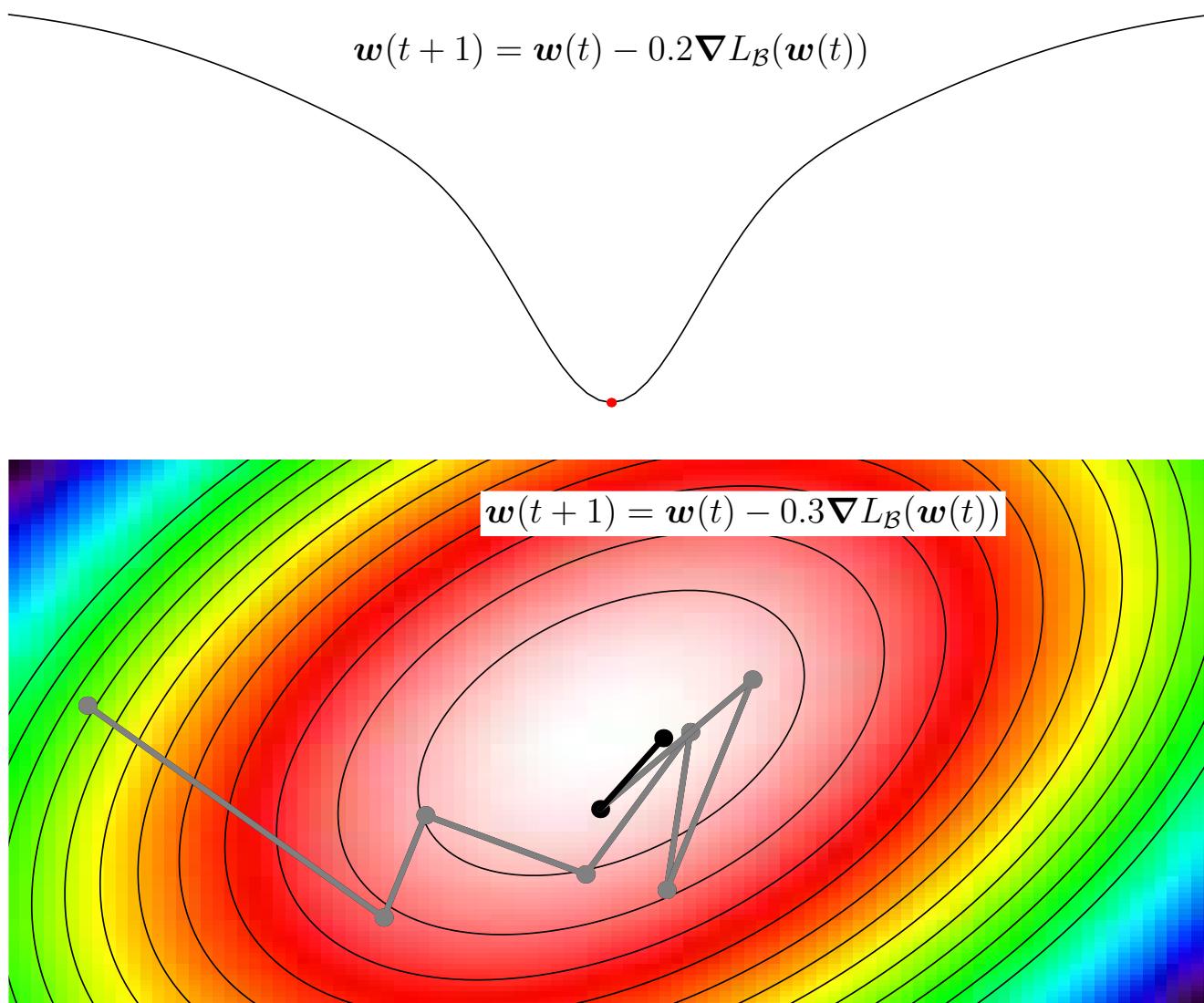
Reducing the Step Size



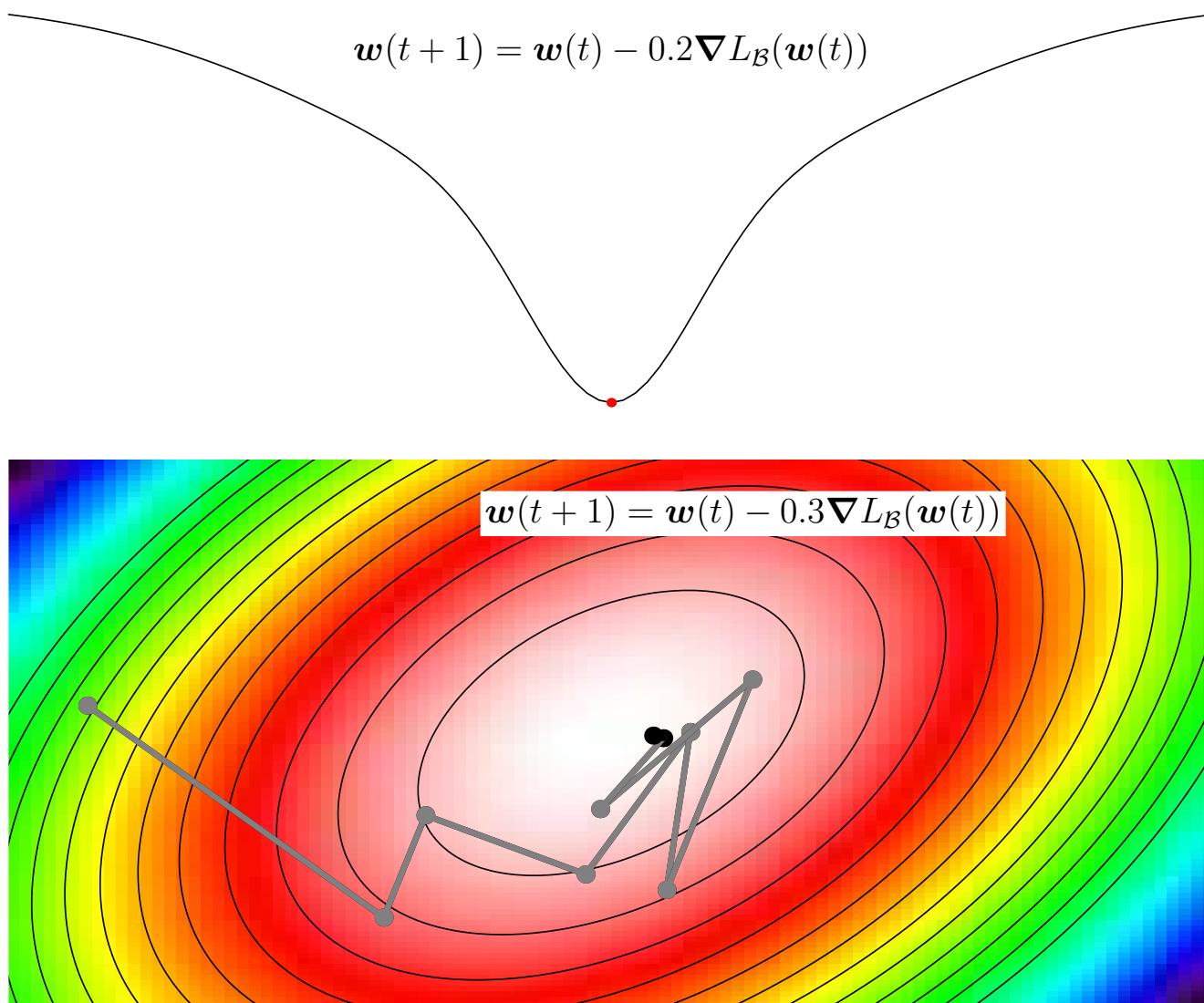
Reducing the Step Size



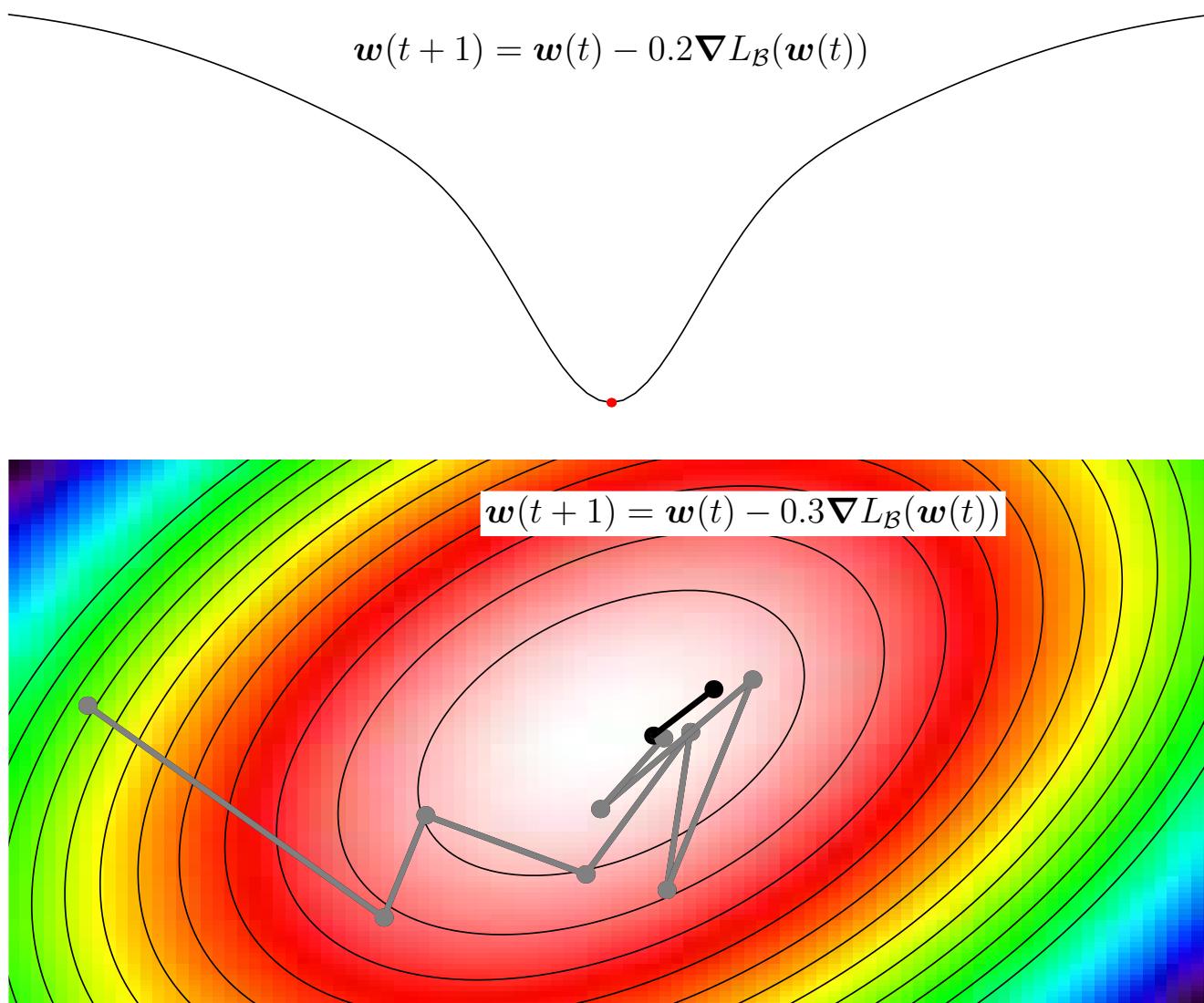
Reducing the Step Size



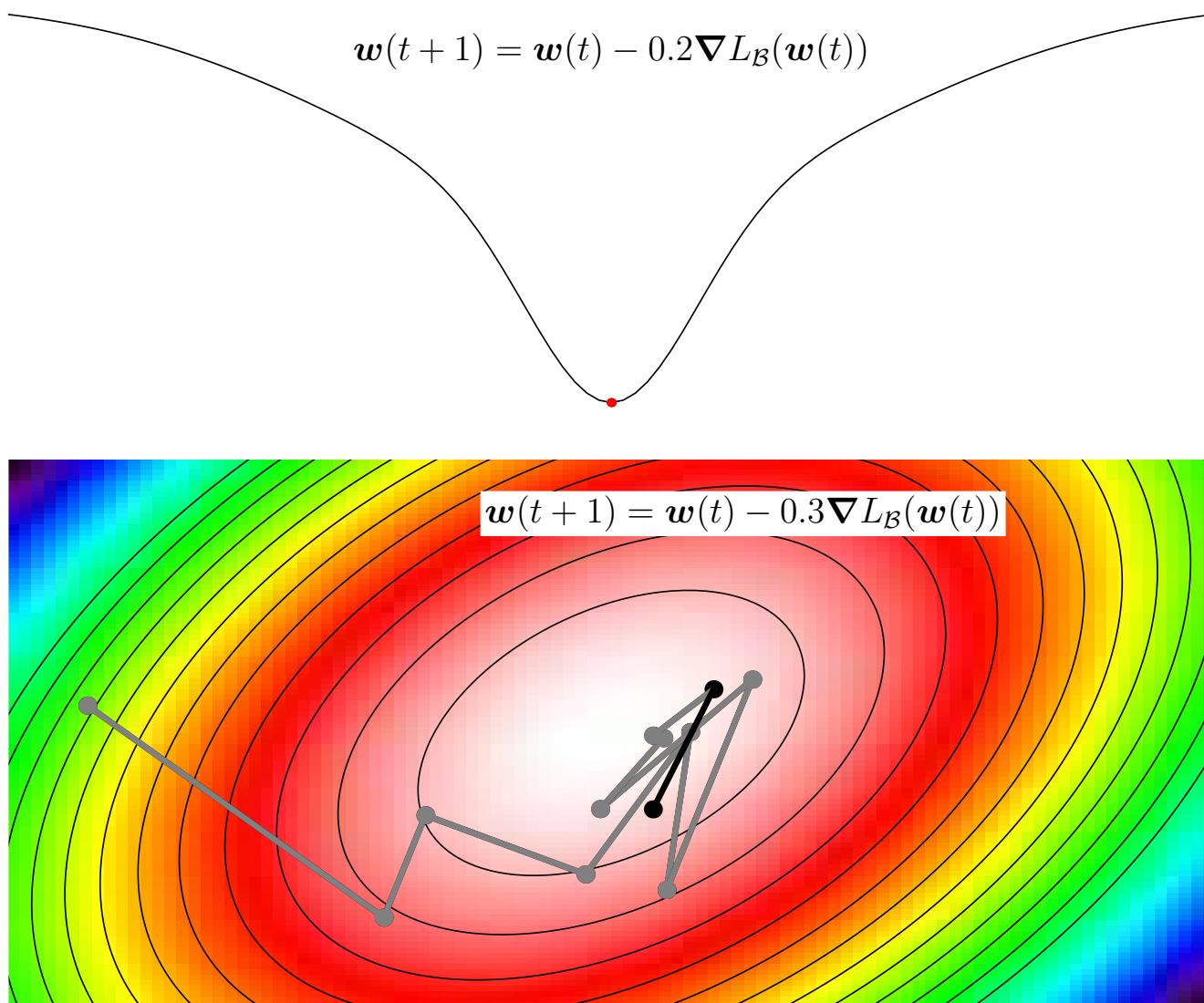
Reducing the Step Size



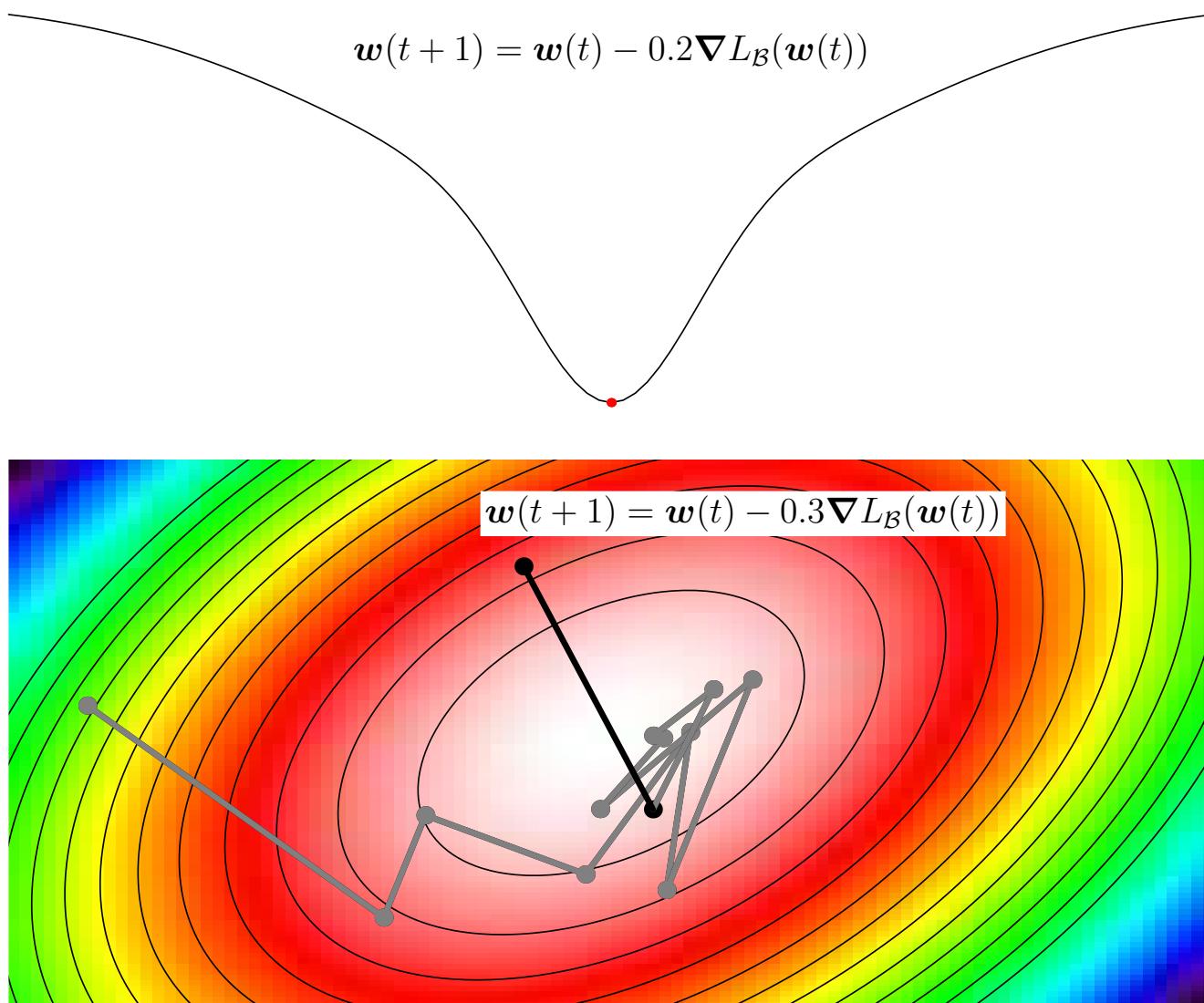
Reducing the Step Size



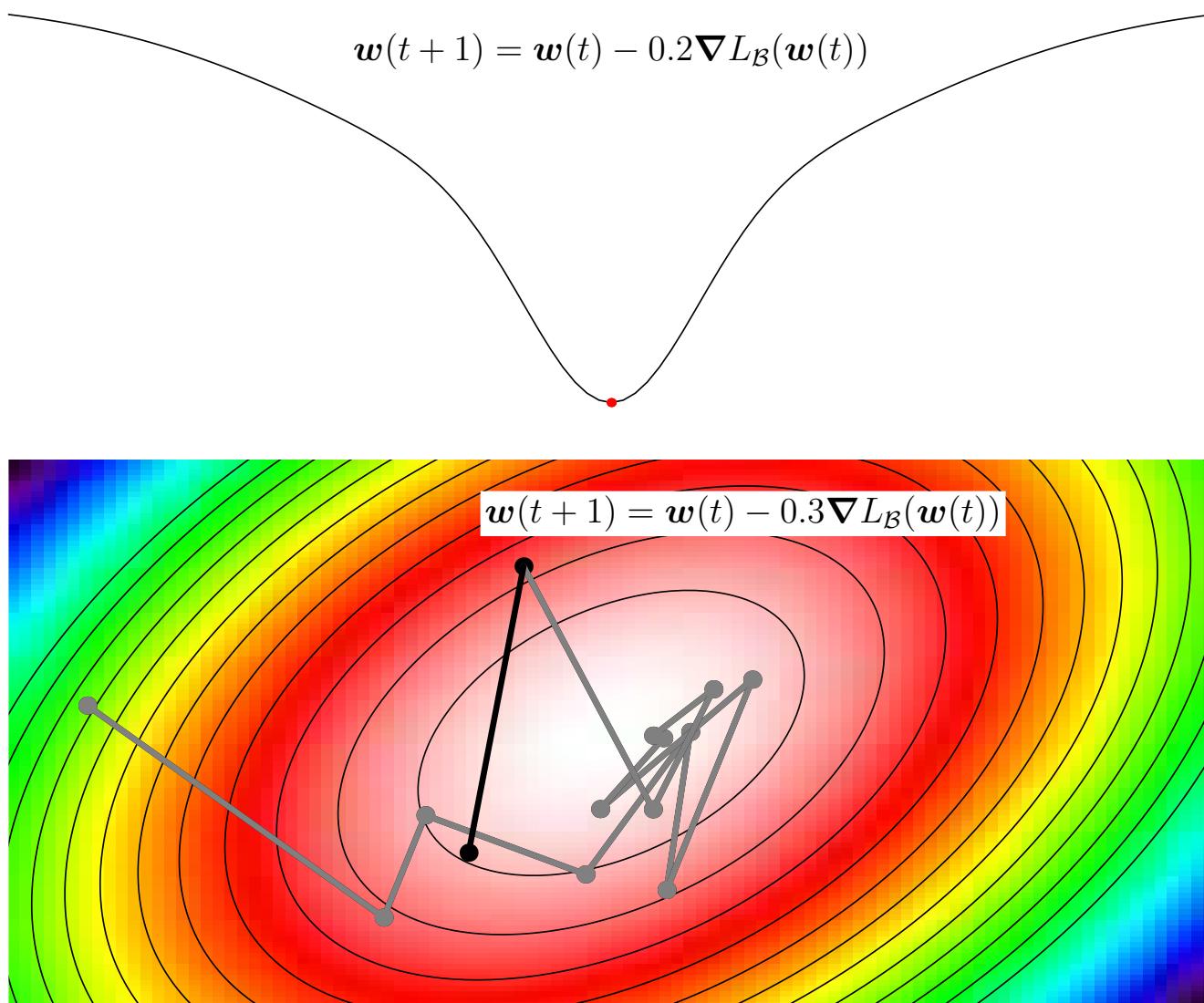
Reducing the Step Size



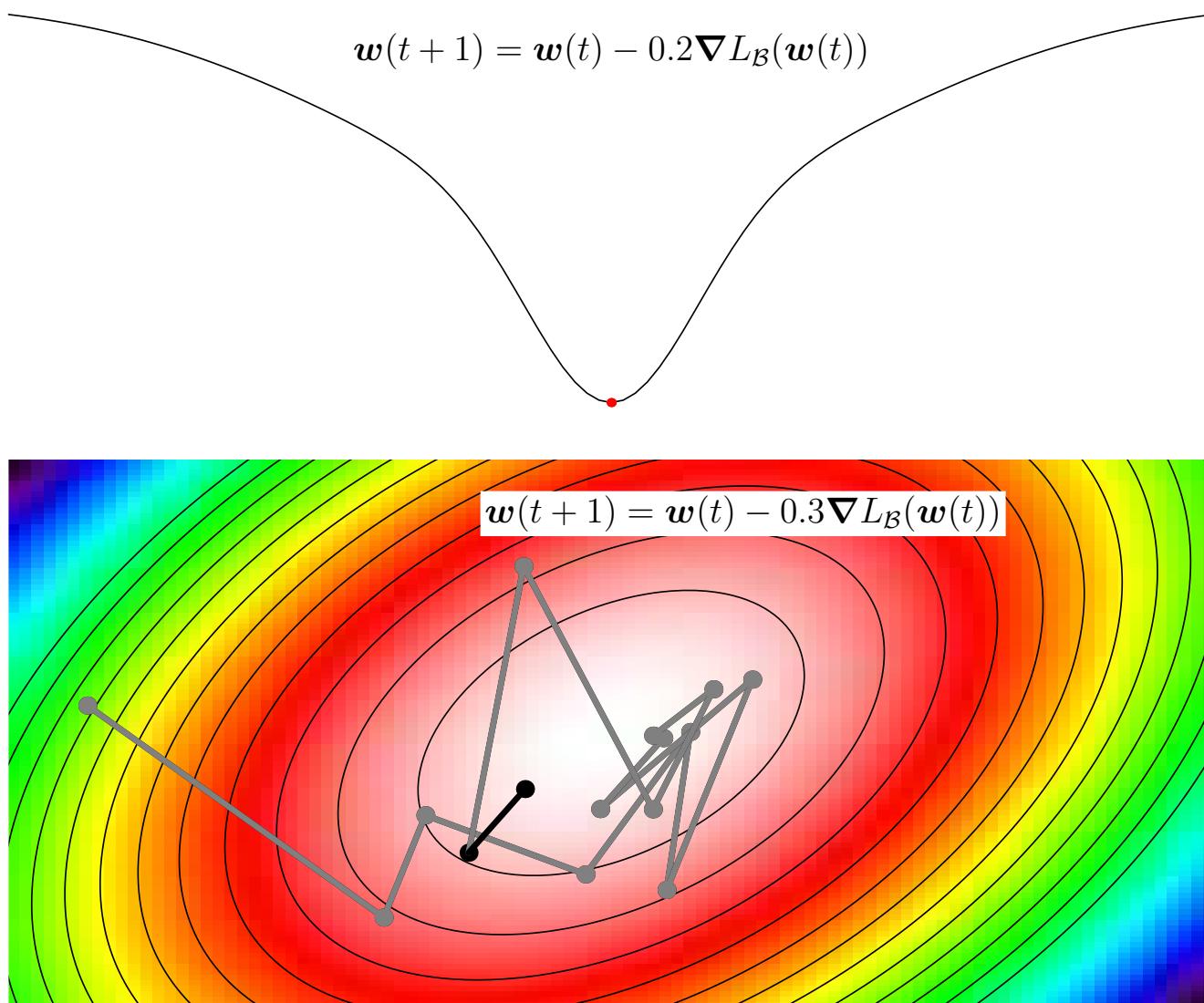
Reducing the Step Size



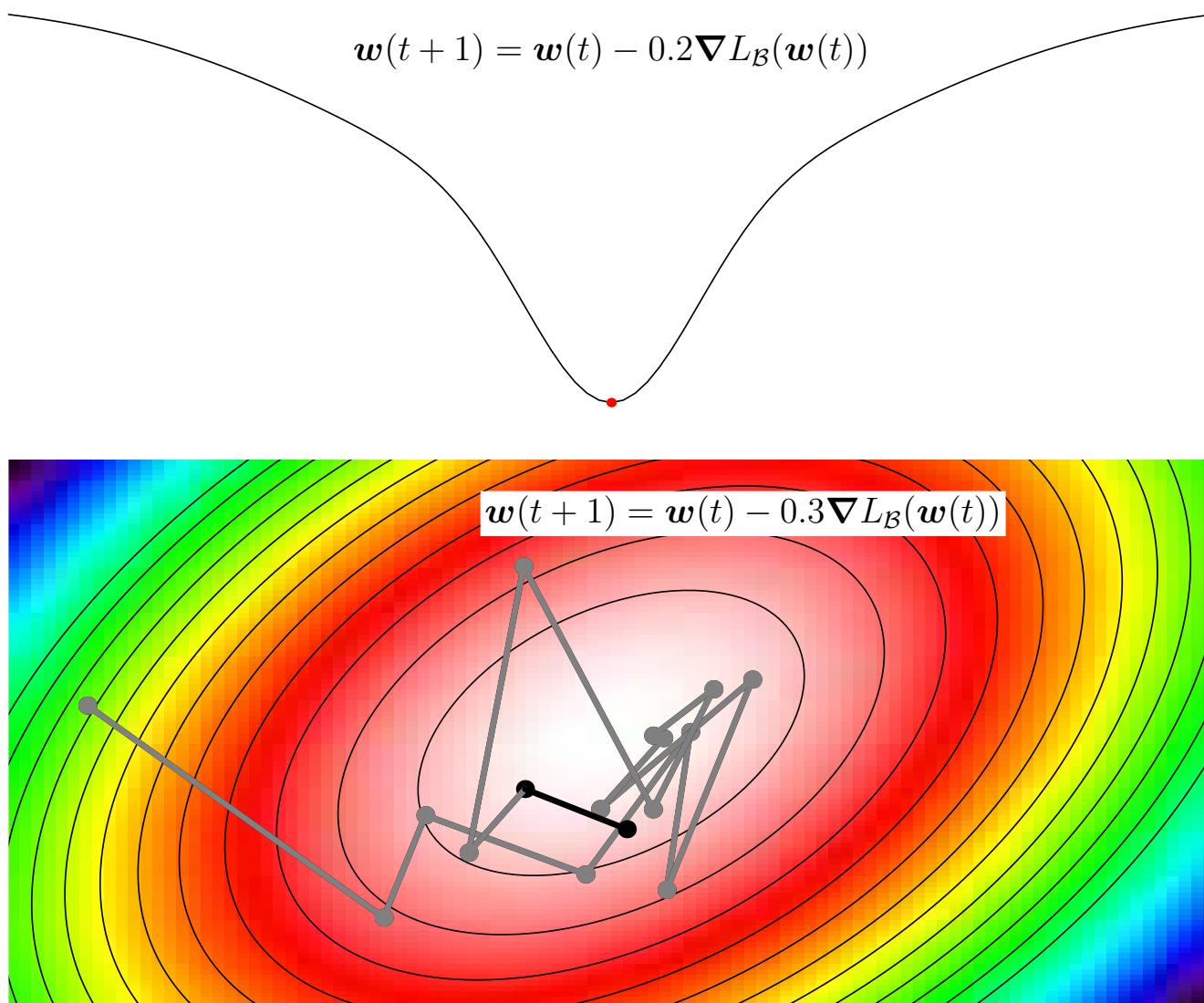
Reducing the Step Size



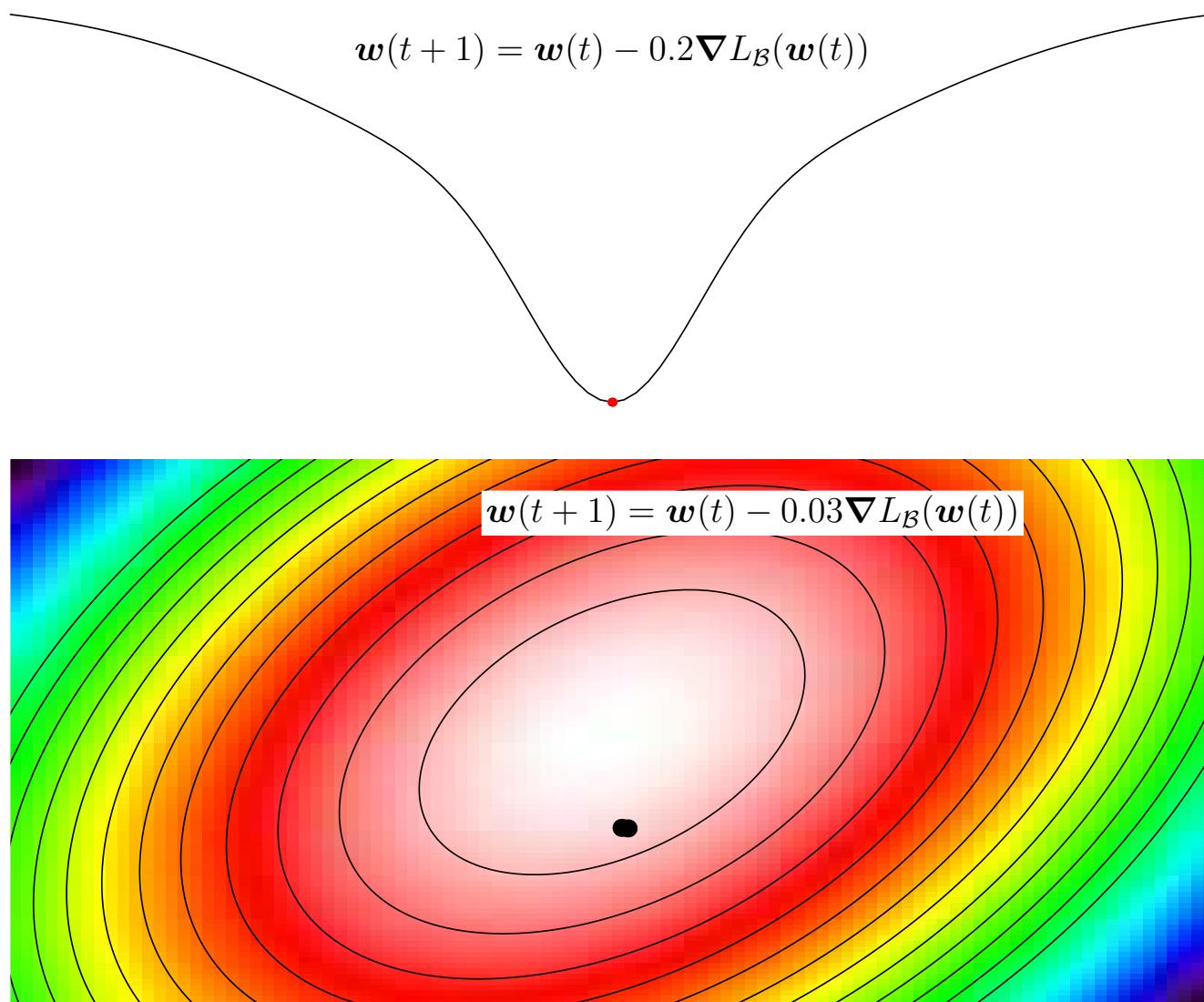
Reducing the Step Size



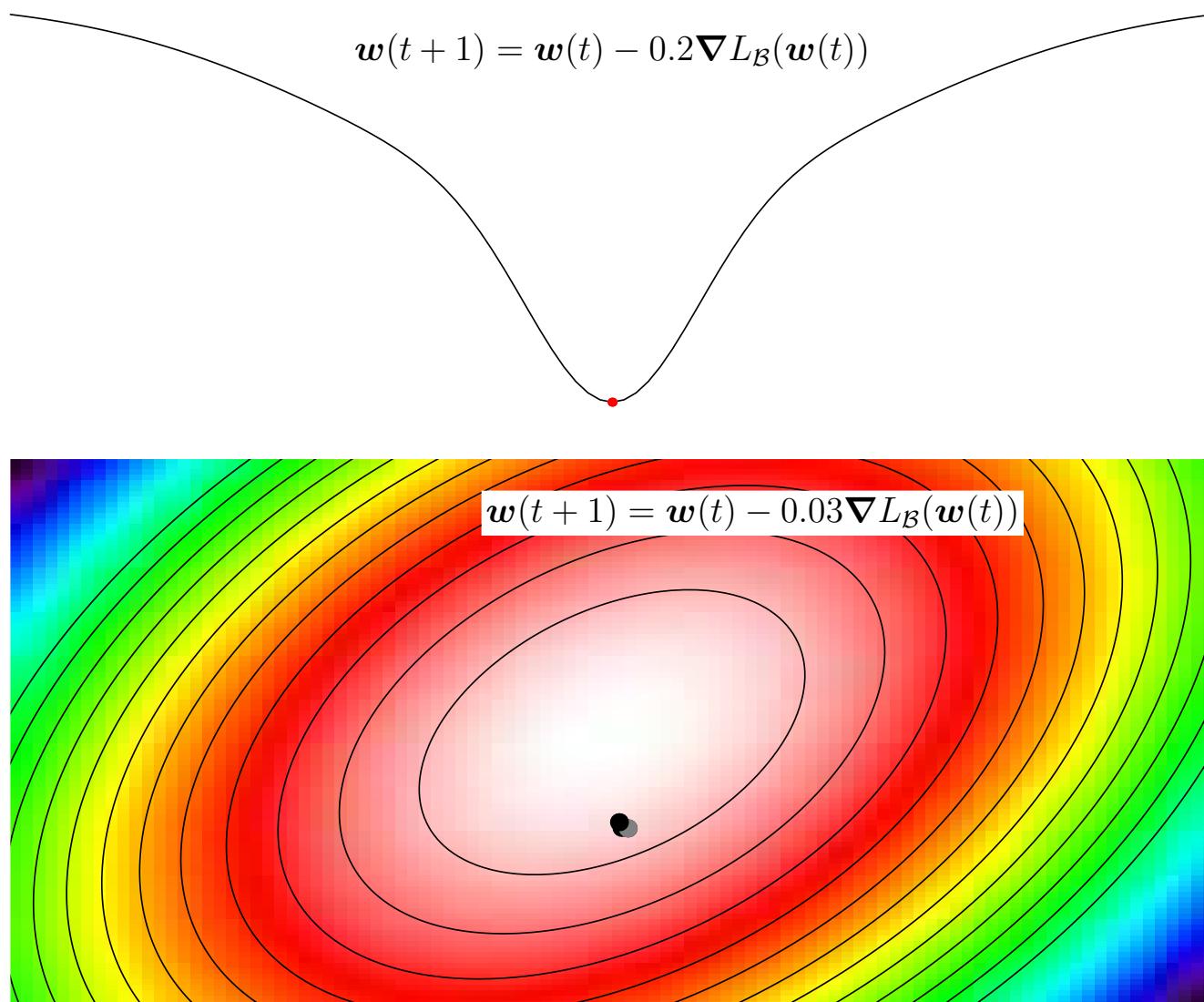
Reducing the Step Size



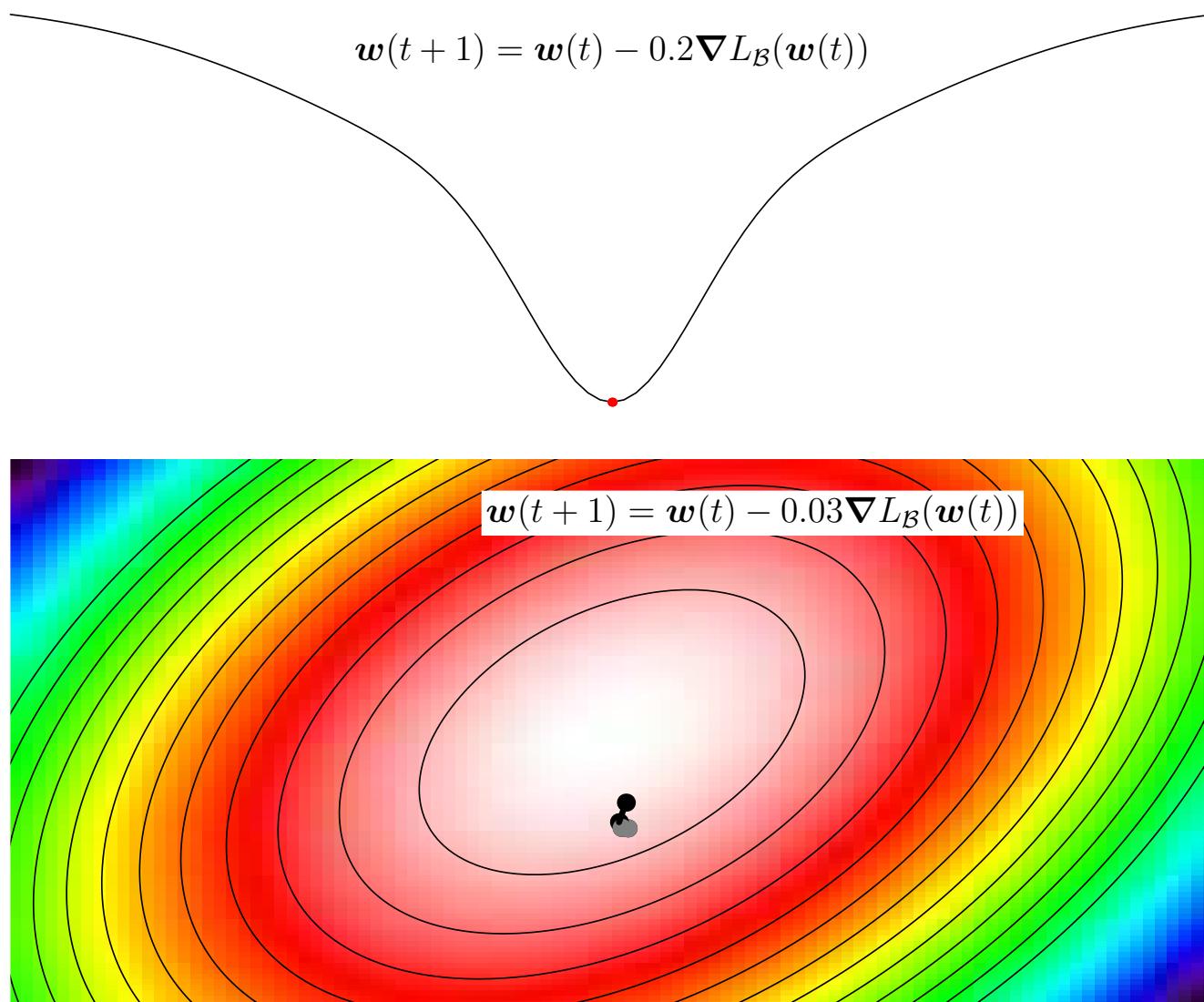
Reducing the Step Size



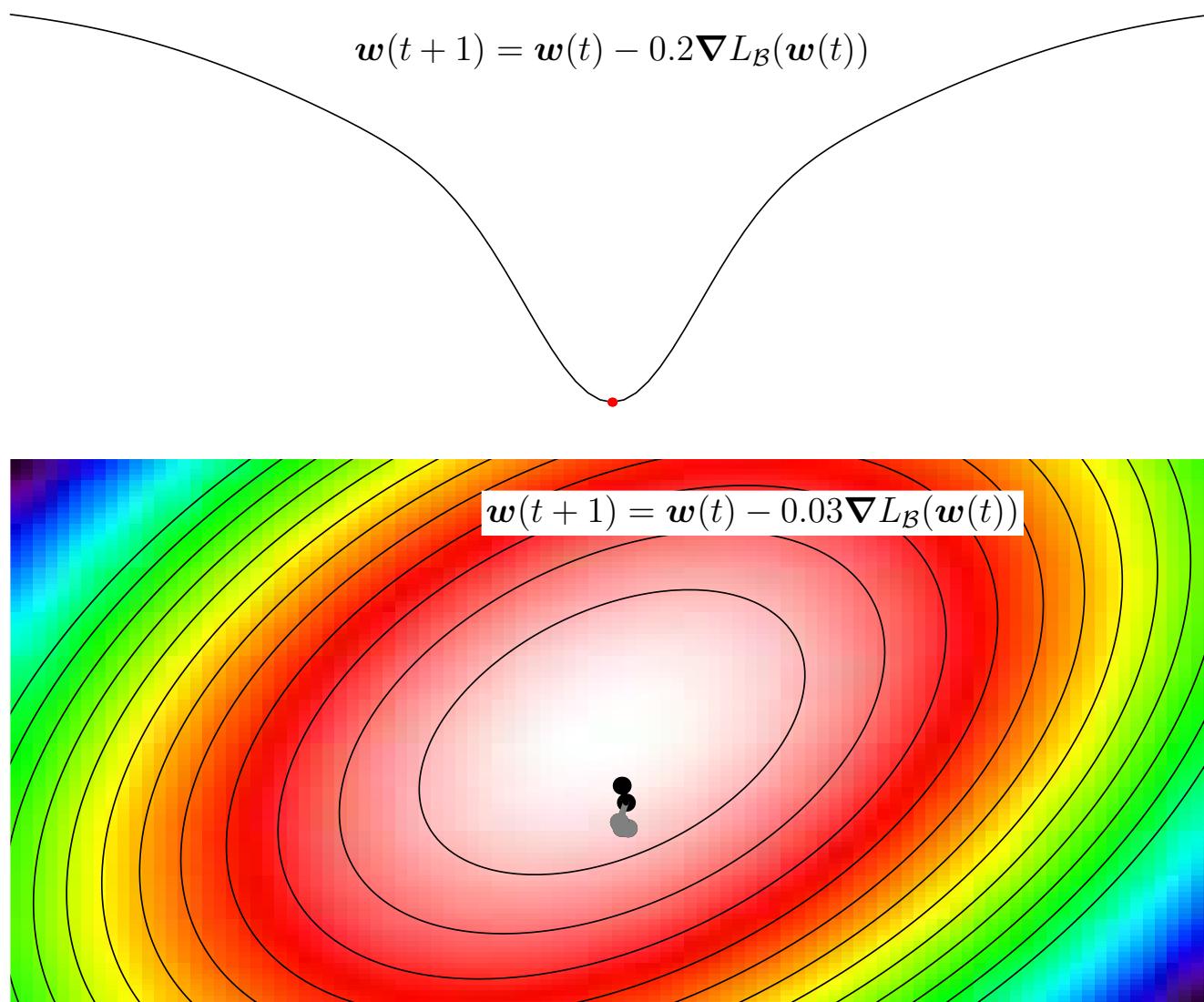
Reducing the Step Size



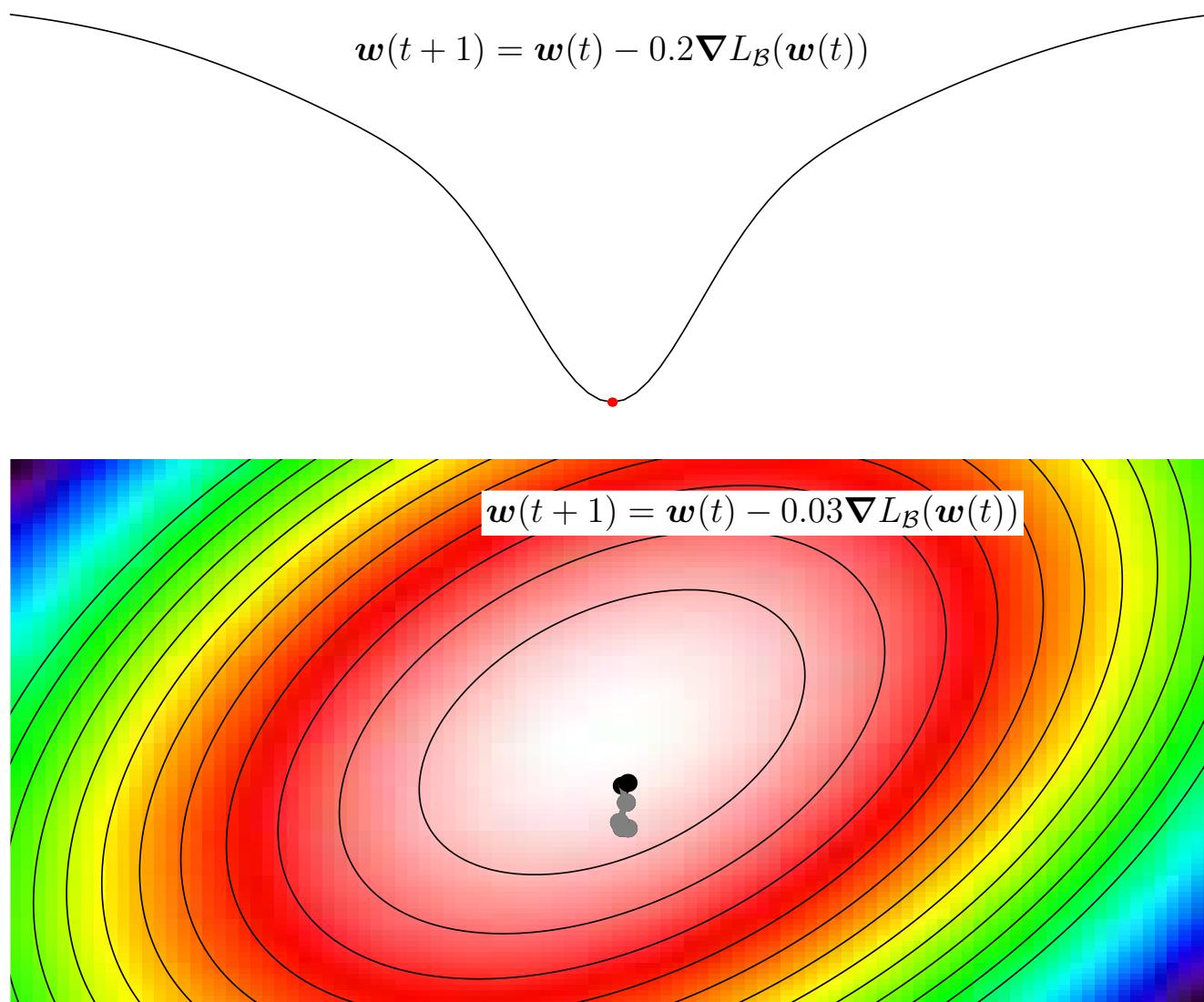
Reducing the Step Size



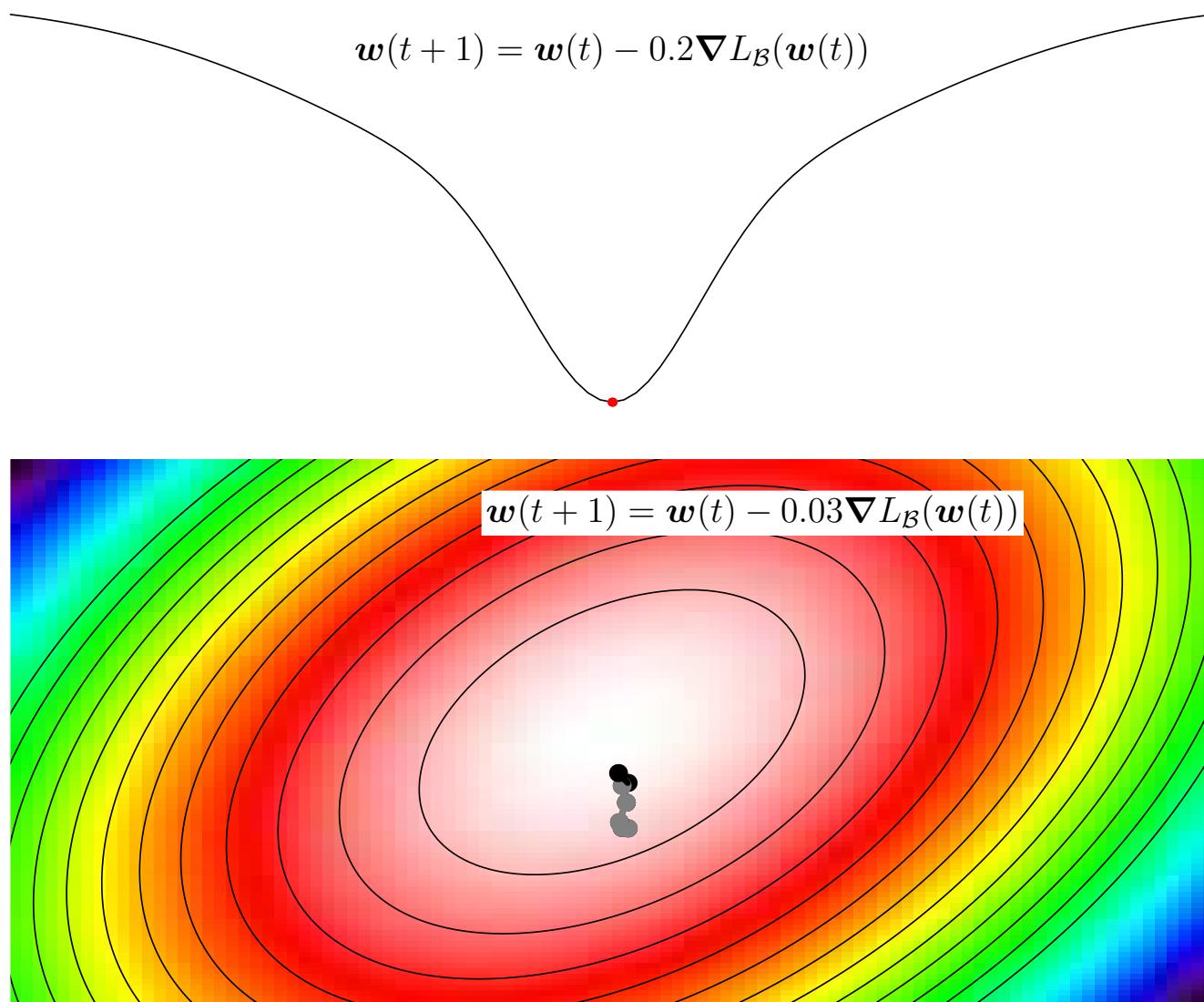
Reducing the Step Size



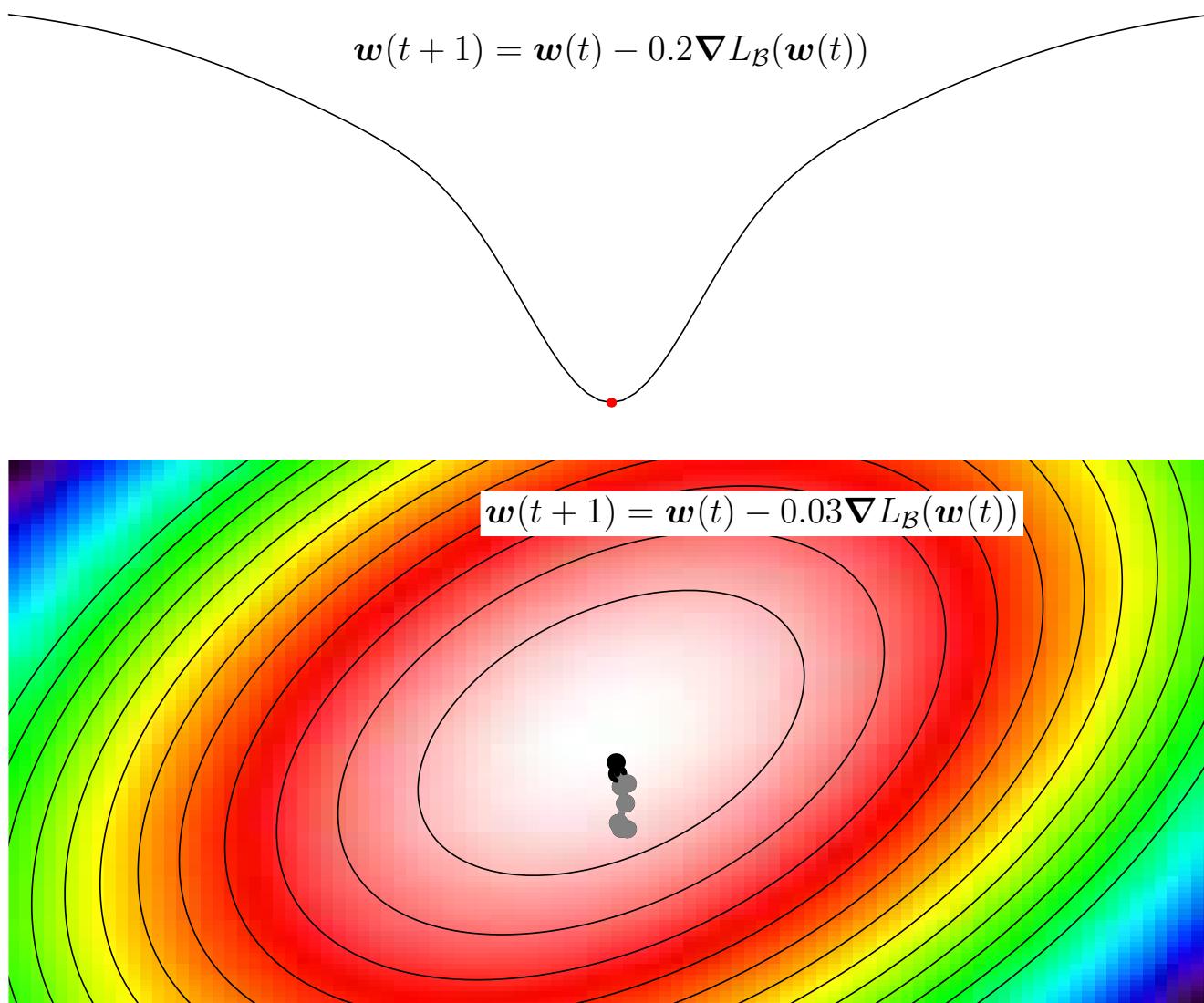
Reducing the Step Size



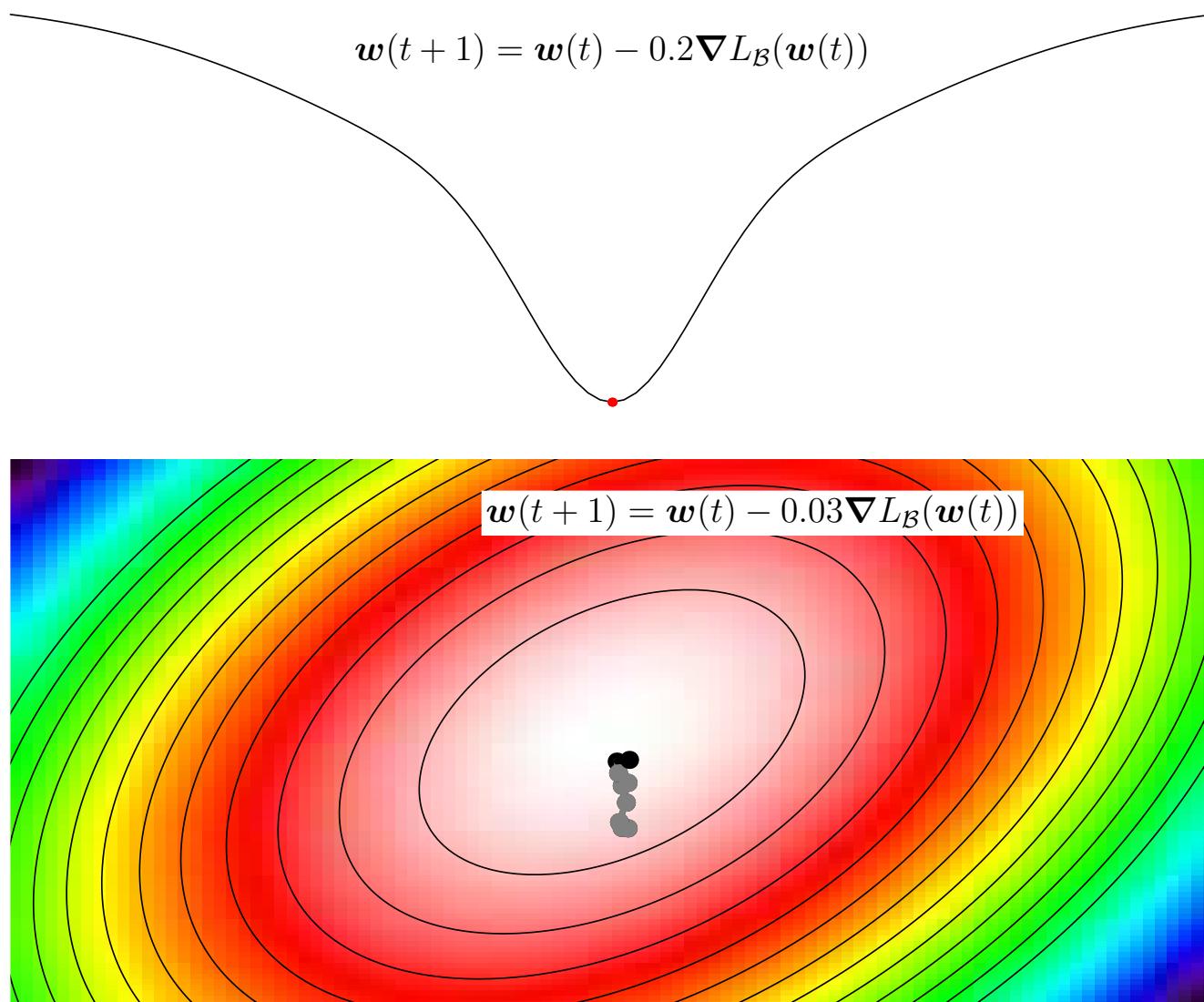
Reducing the Step Size



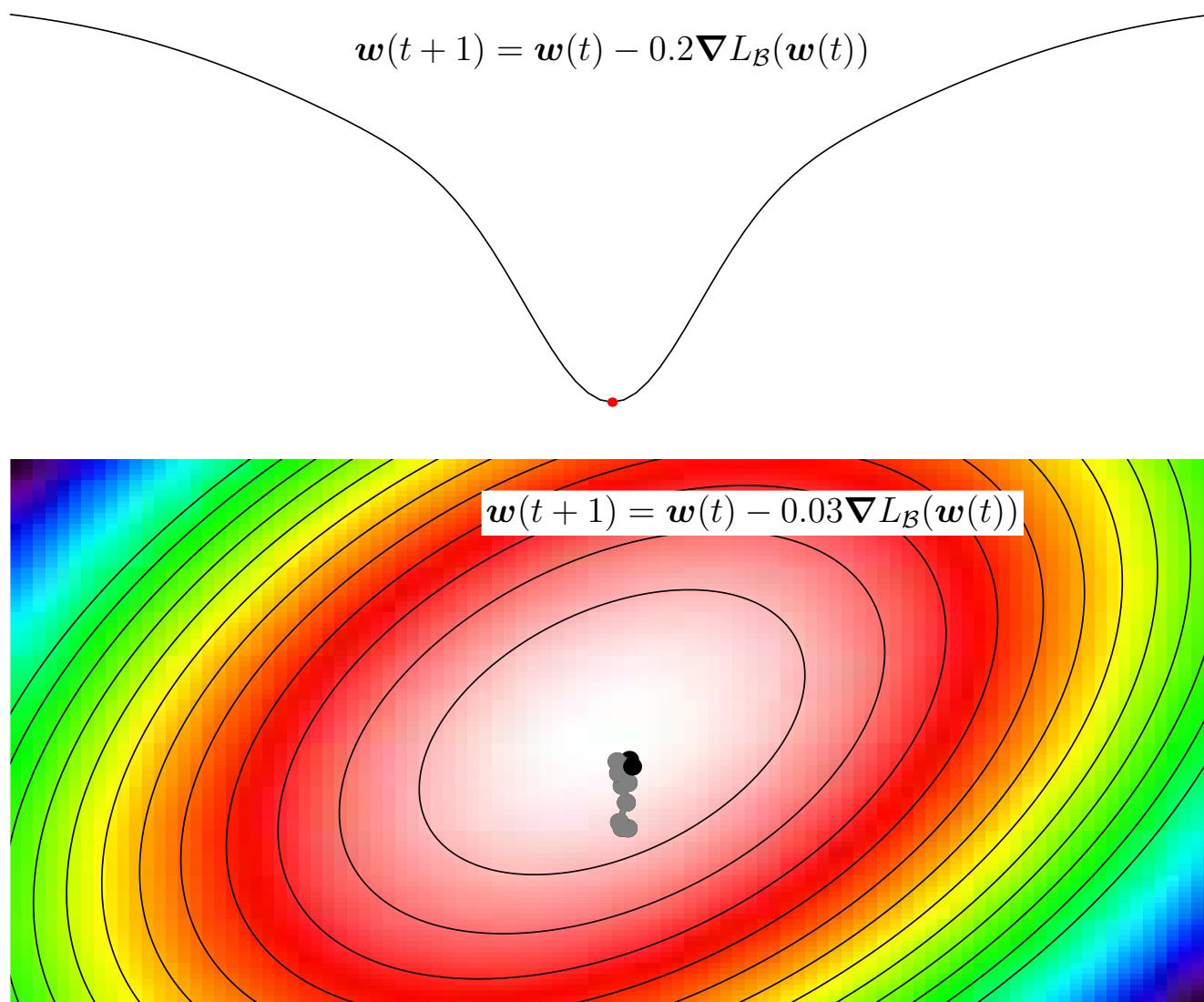
Reducing the Step Size



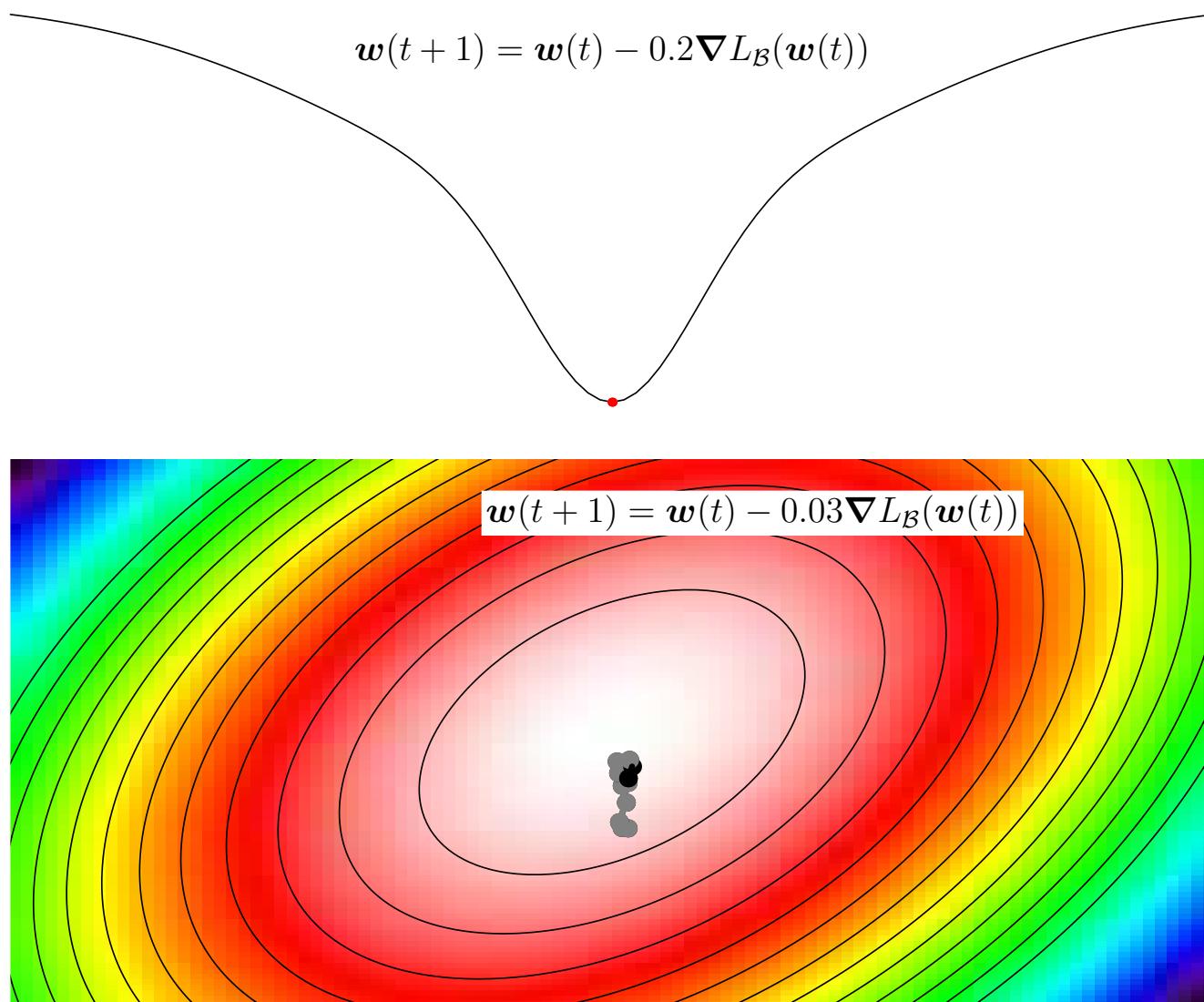
Reducing the Step Size



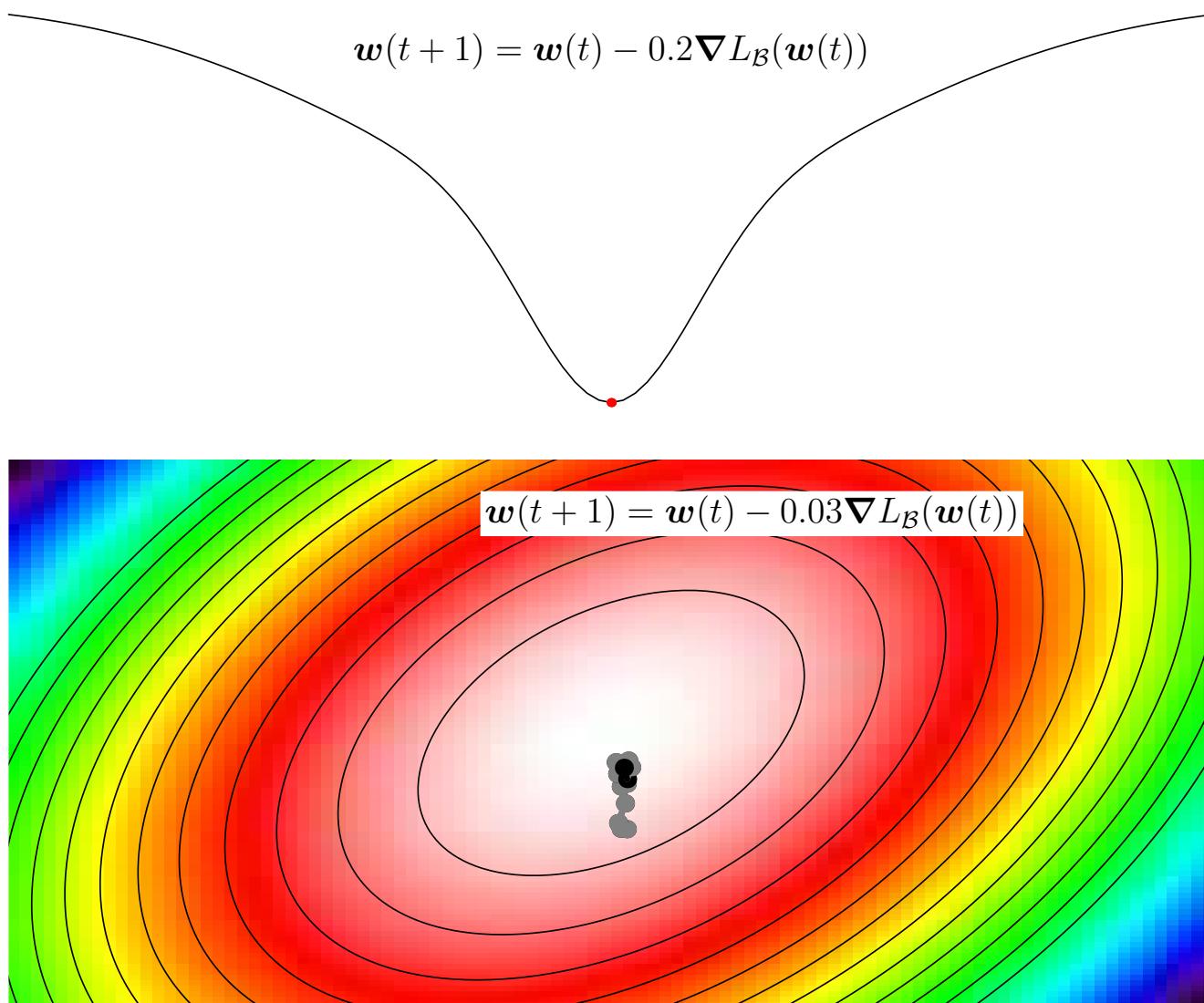
Reducing the Step Size



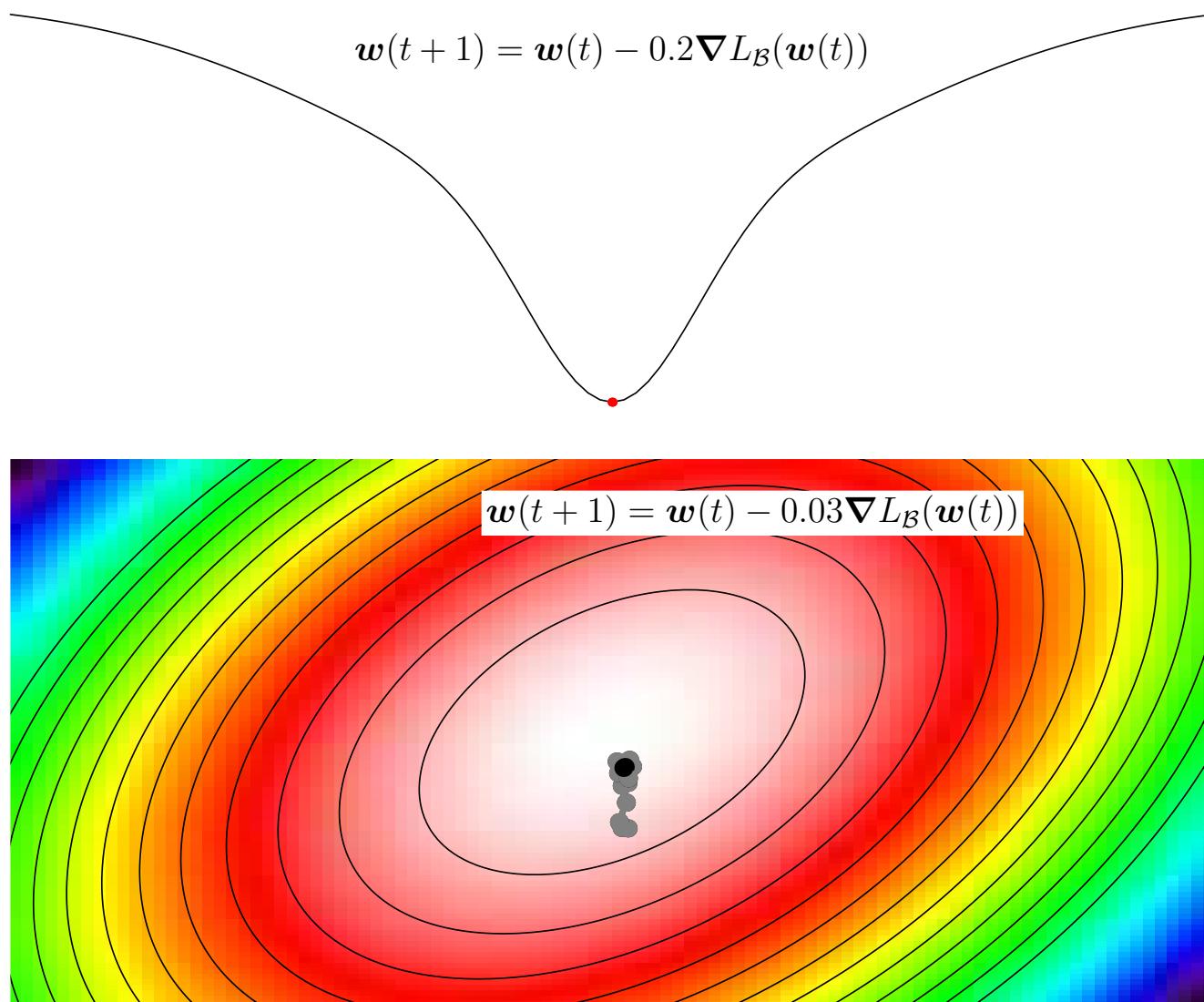
Reducing the Step Size



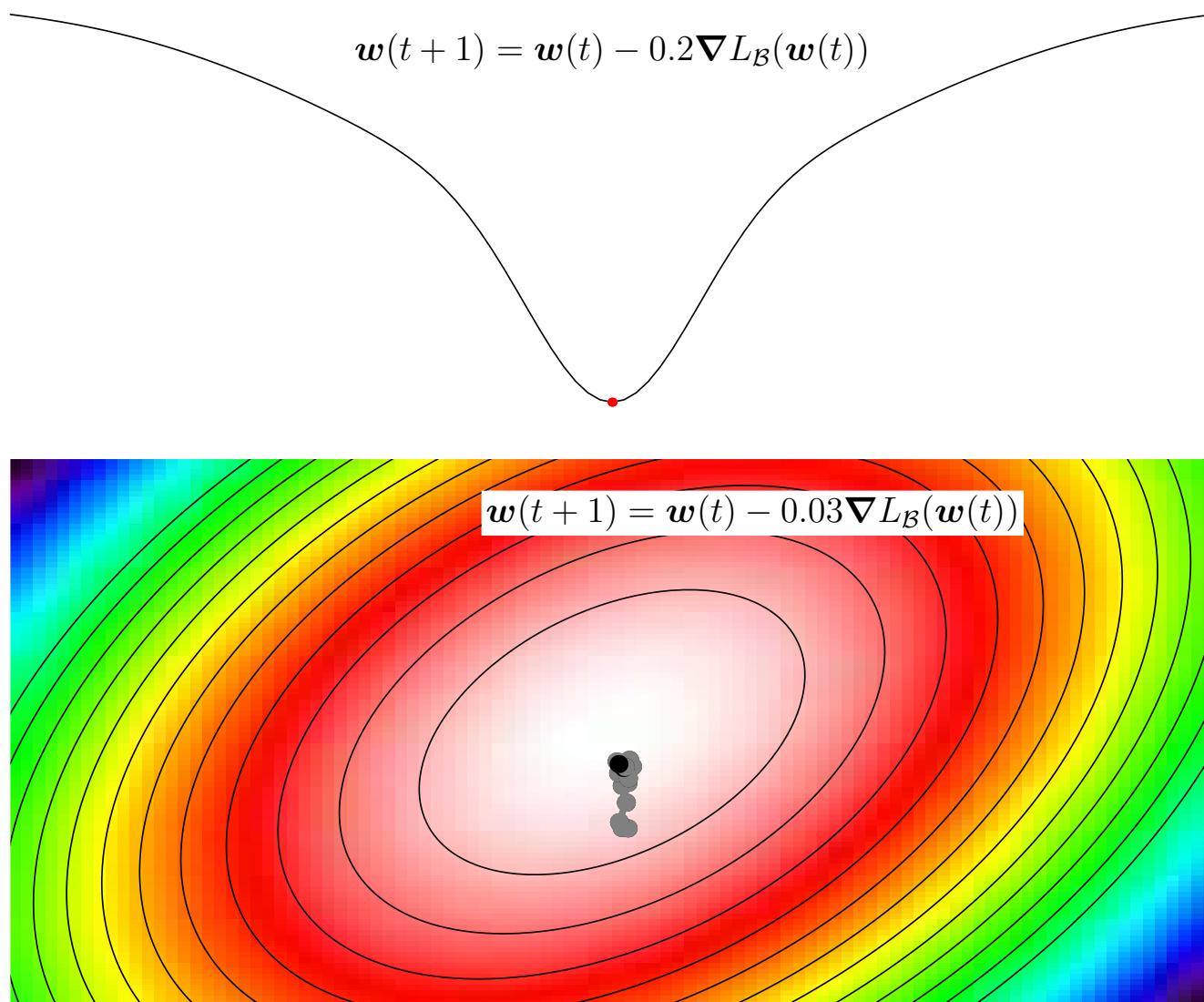
Reducing the Step Size



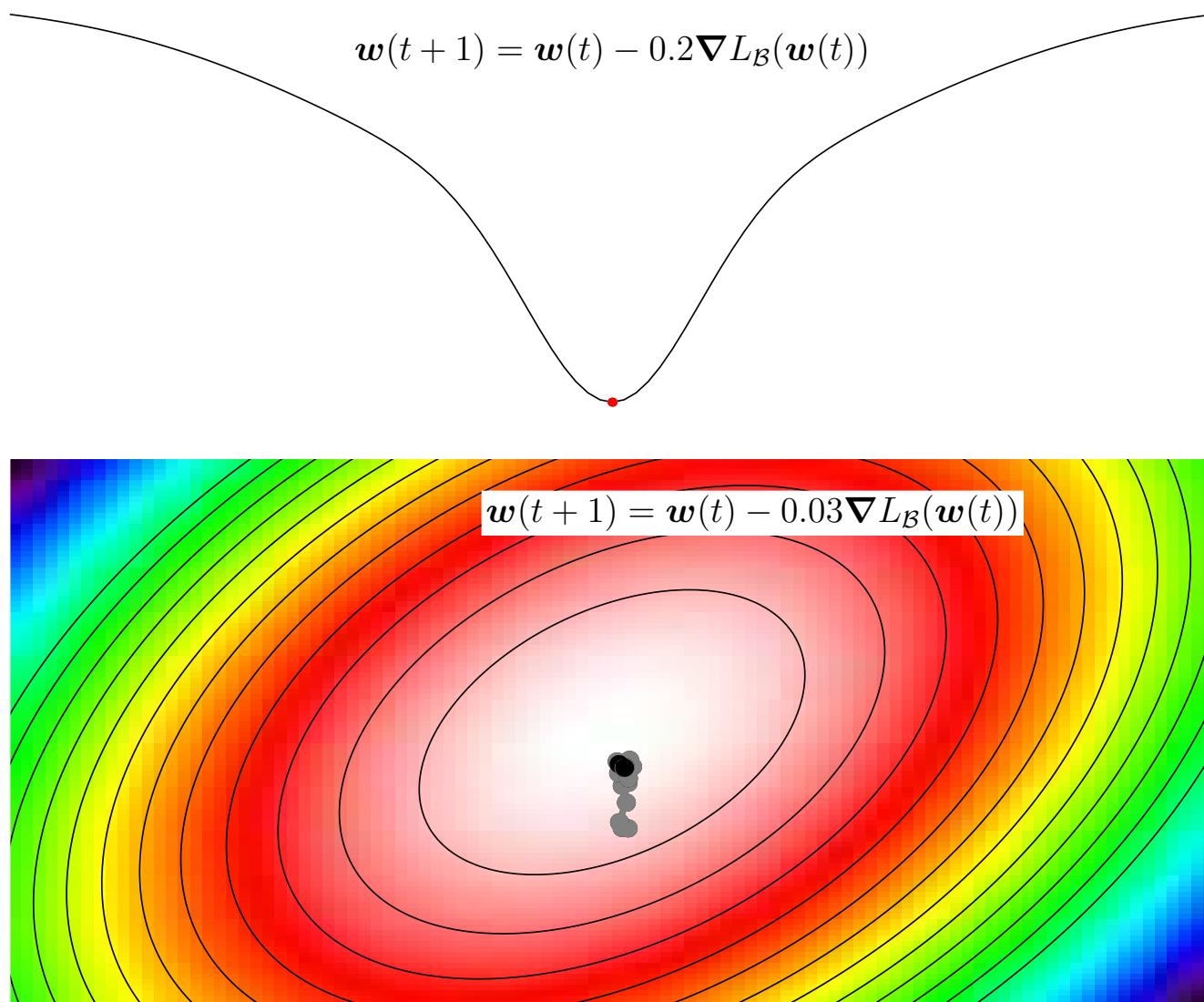
Reducing the Step Size



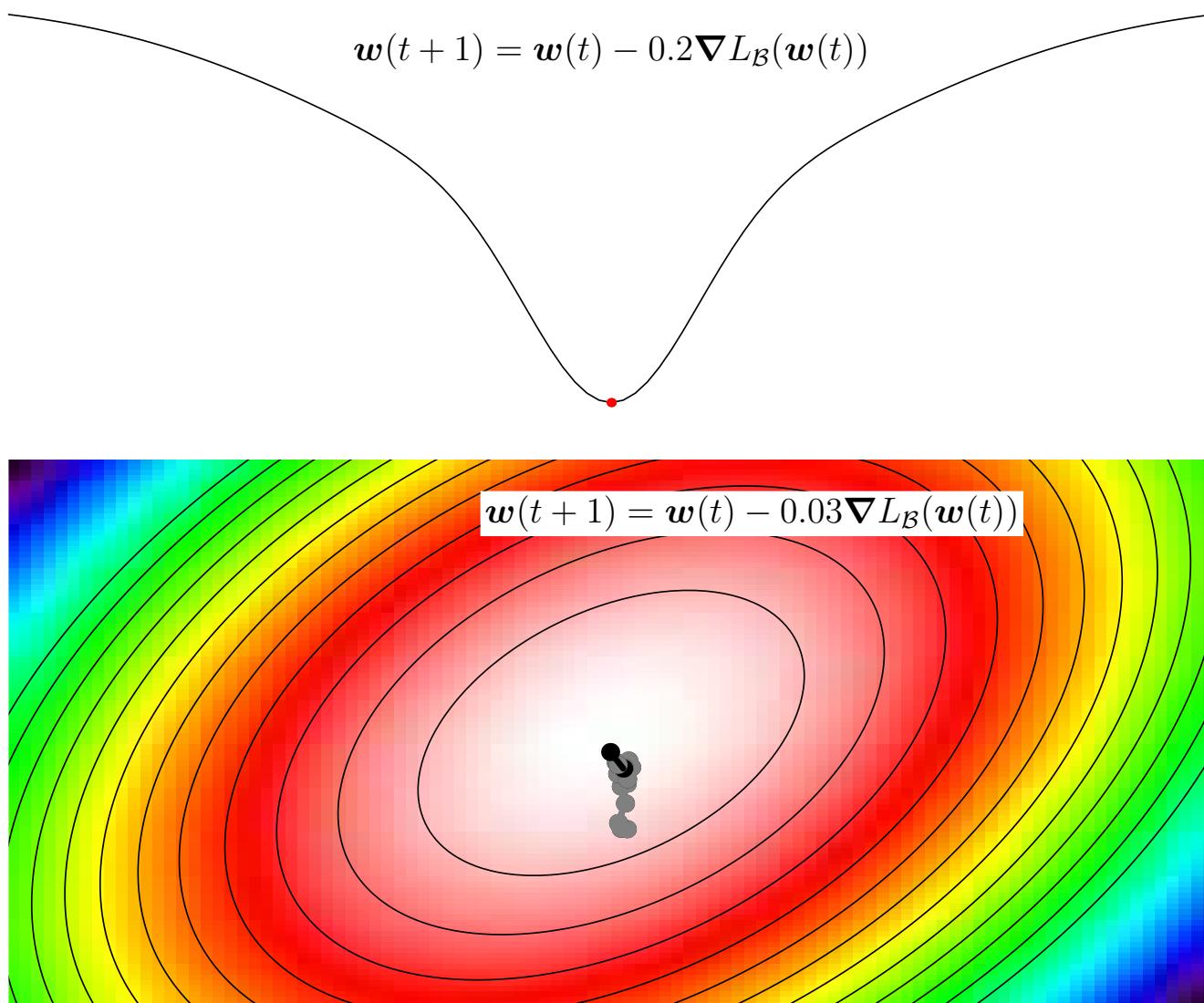
Reducing the Step Size



Reducing the Step Size



Reducing the Step Size



Symmetries

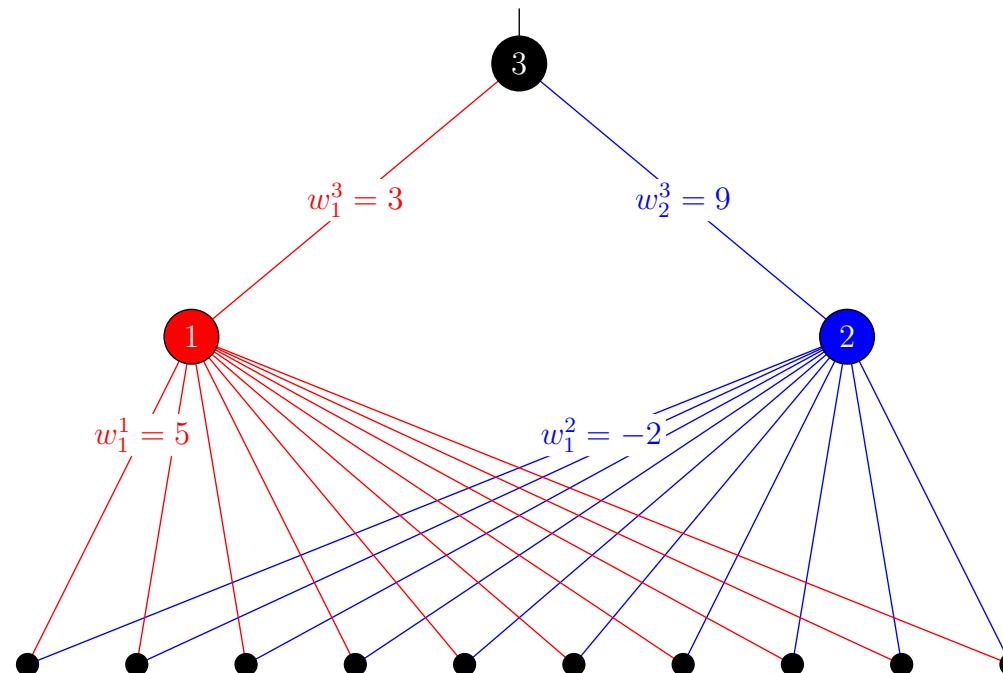
- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers

Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers

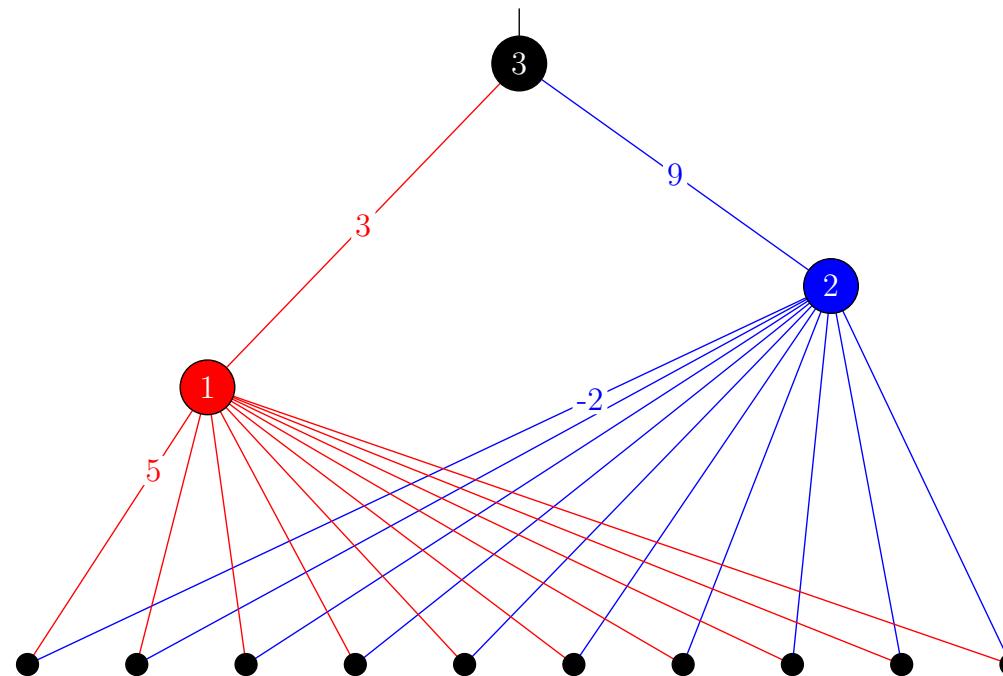
Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers



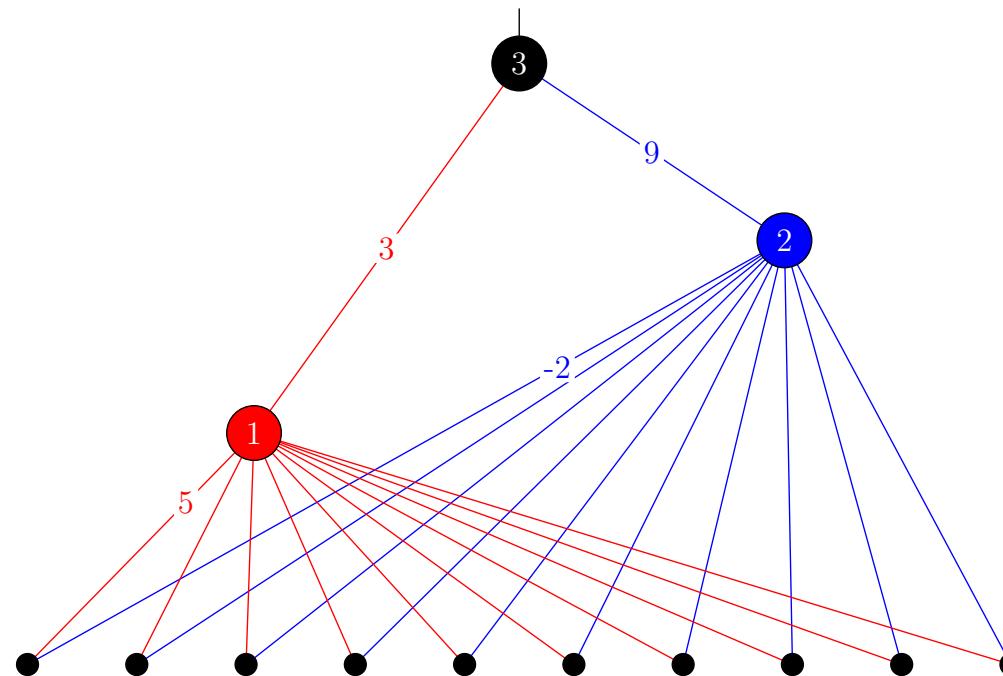
Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers



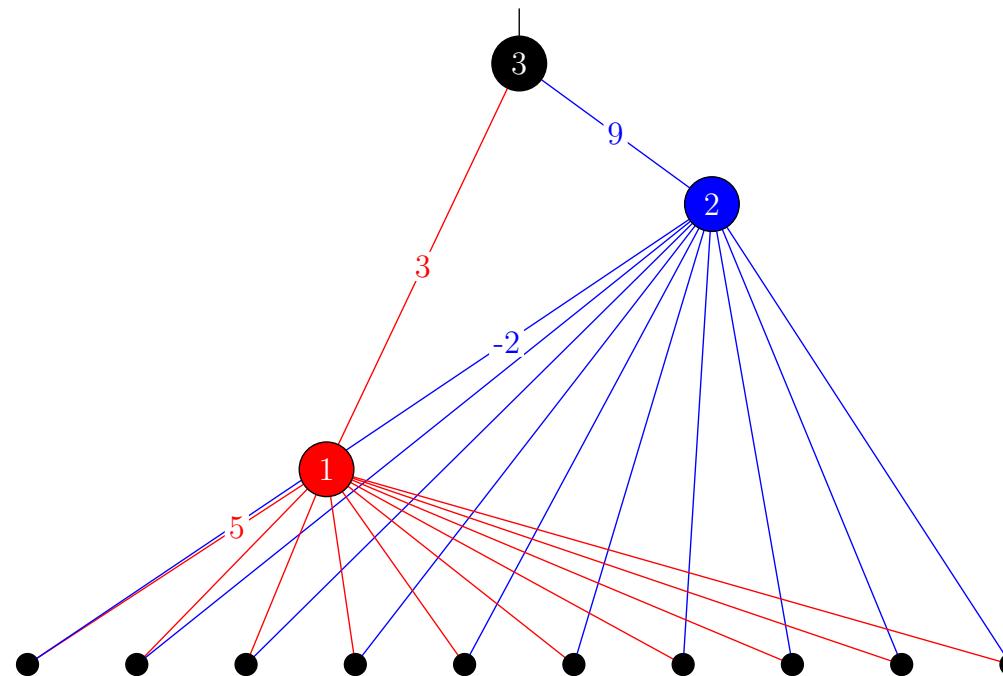
Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers



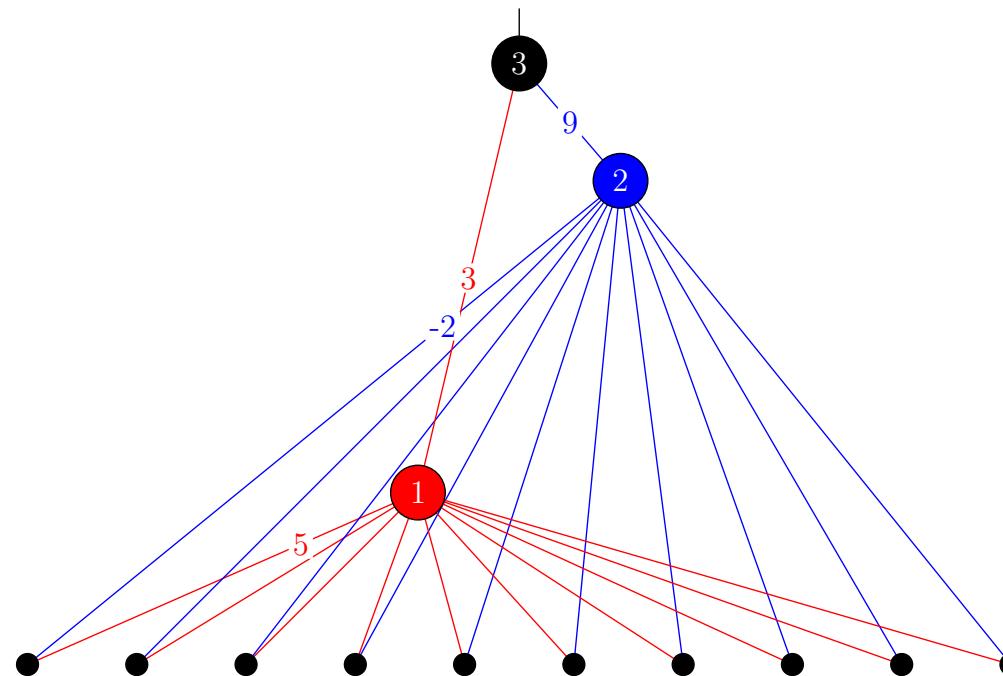
Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers



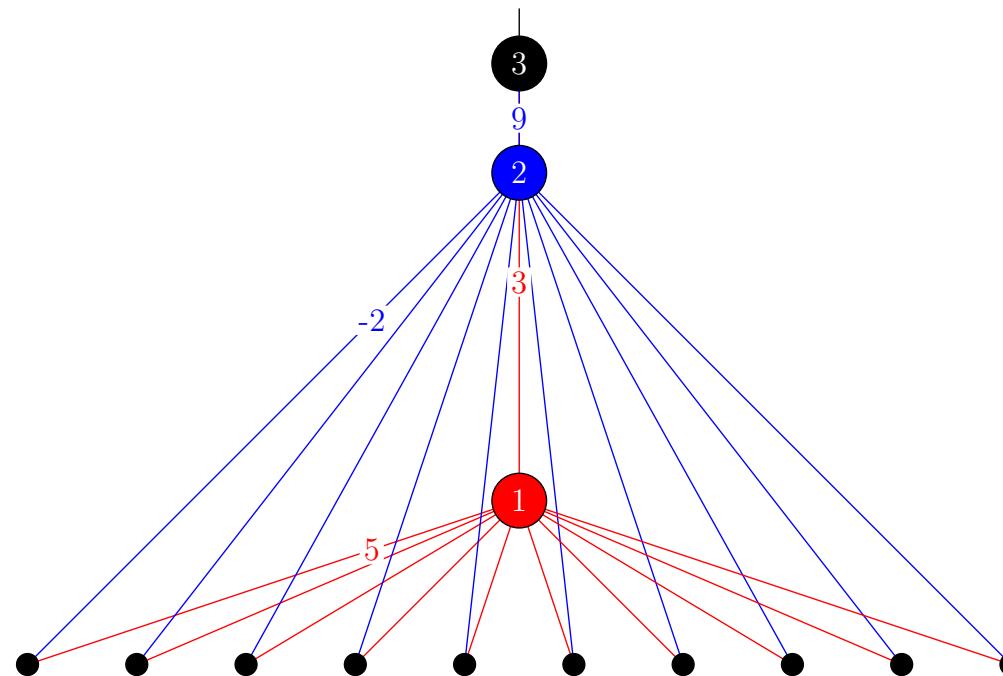
Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers



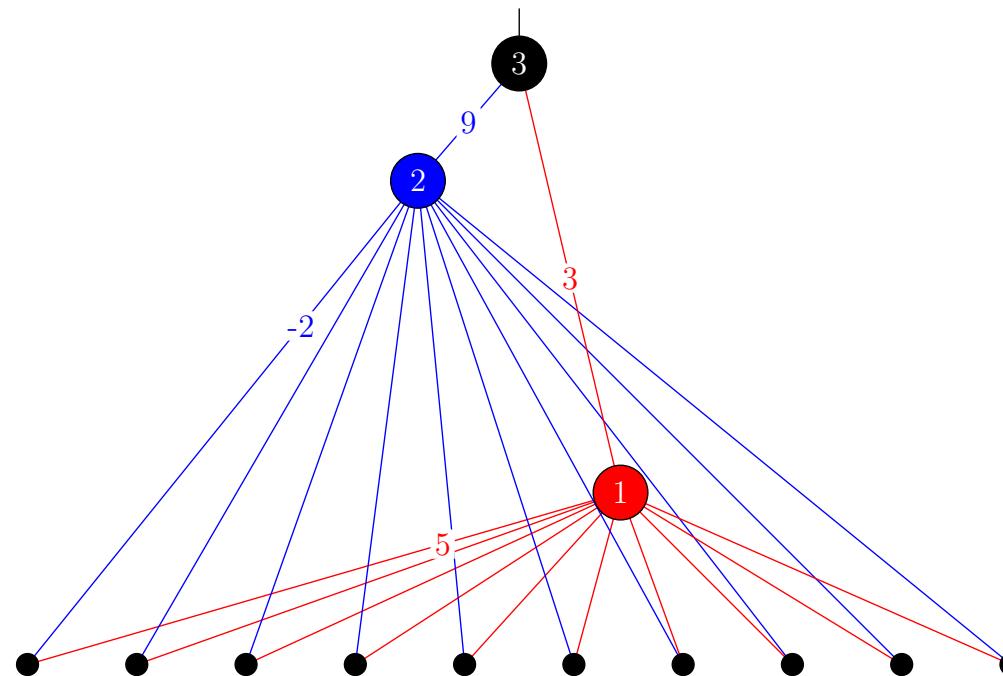
Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers



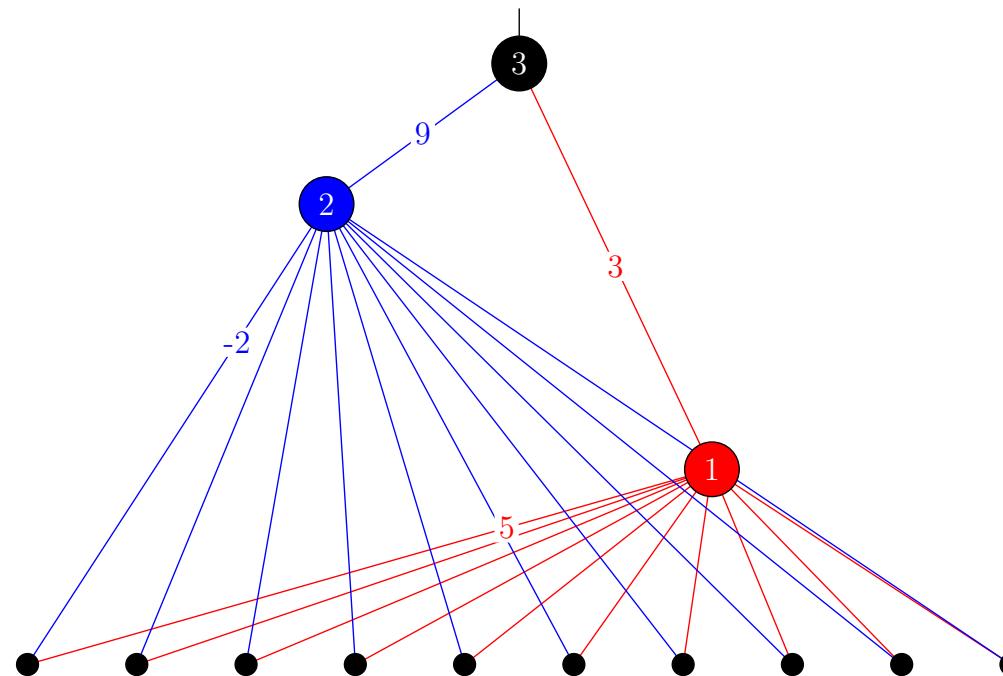
Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers



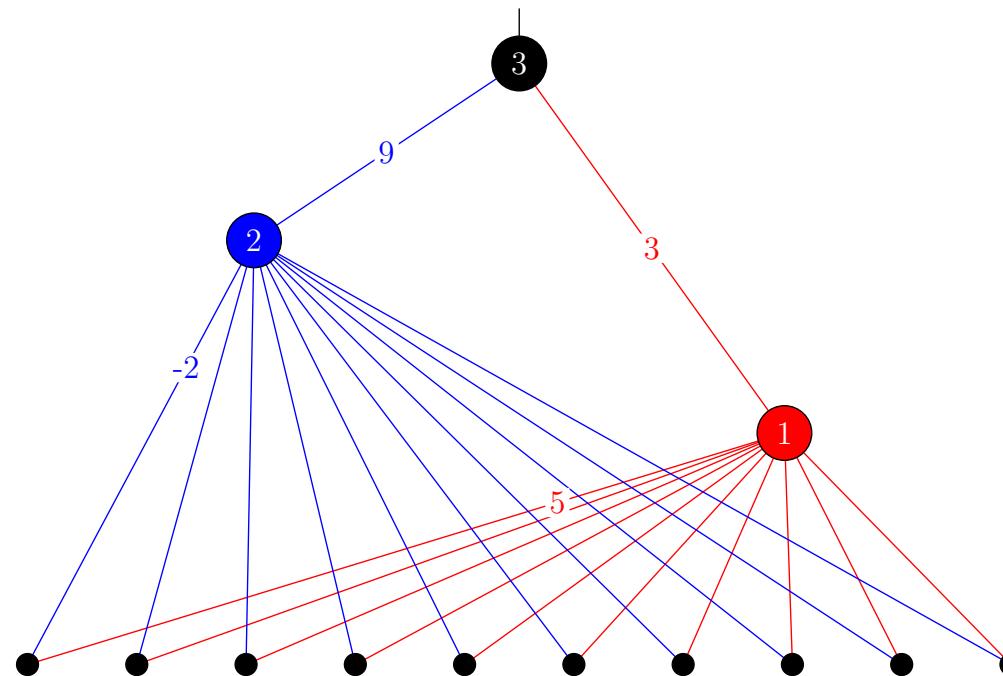
Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers



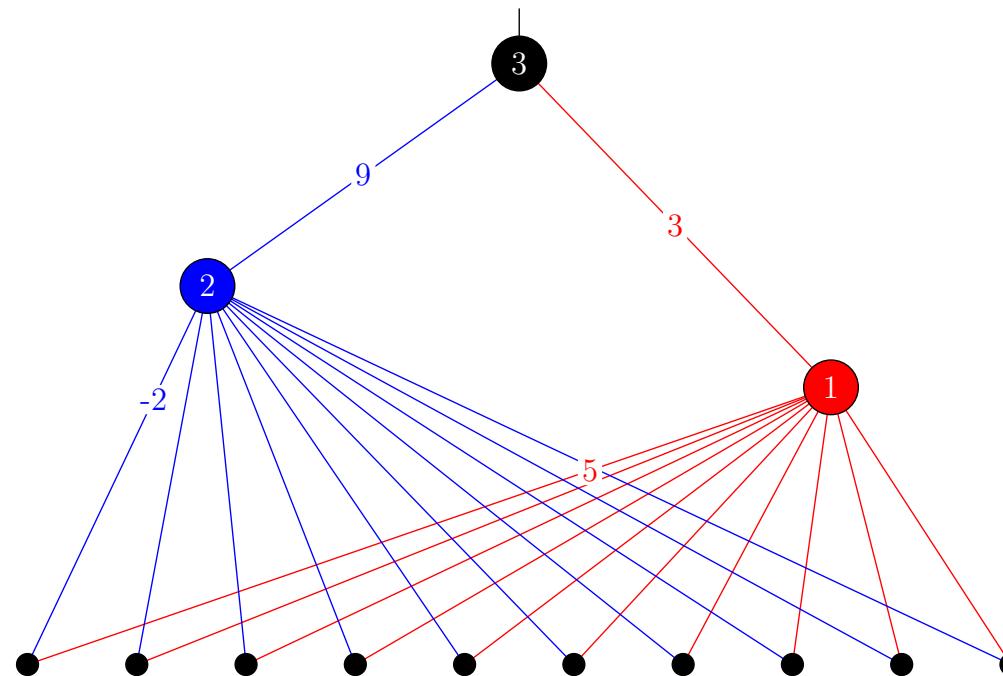
Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers



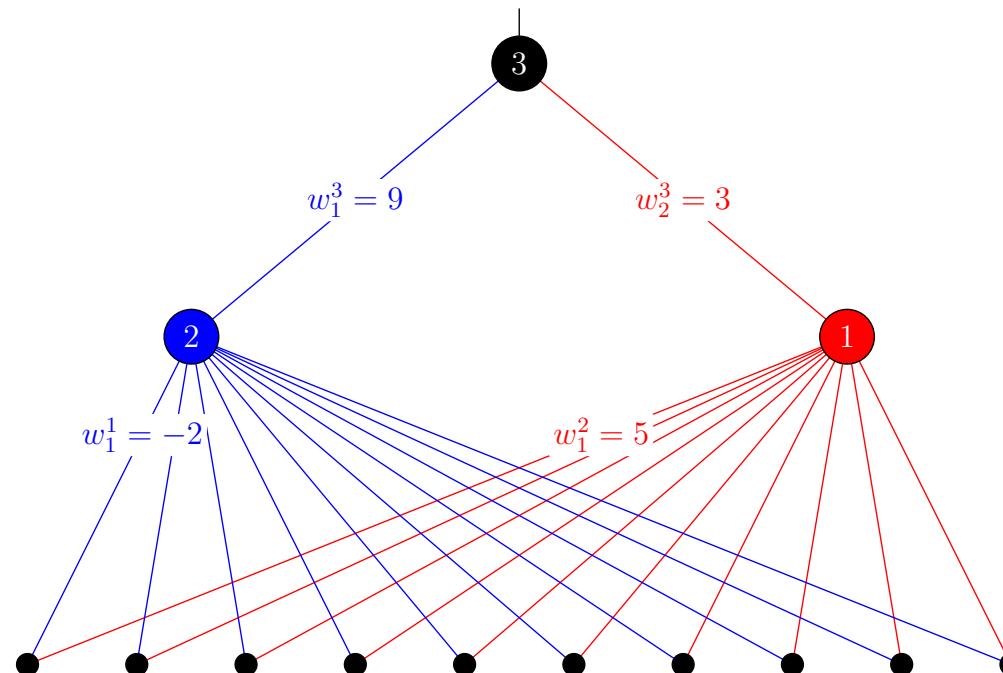
Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers



Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers



More Symmetries

- With n hidden nodes there are $n!$ symmetries
- Often if I multiply all the input and output weights to a node by -1 the network will be the same
- There are also continuous symmetries. For linear networks with two layers performing a mapping $\mathbf{W}_2\mathbf{W}_1$ then for any invertible matrix \mathbf{U} (where $\mathbf{U}\mathbf{U}^{-1} = \mathbf{I}$)

$$\mathbf{W}_2\mathbf{W}_1 = \mathbf{W}_2\mathbf{I}\mathbf{W}_1$$

so a network with weights $\mathbf{W}'_2 = \mathbf{W}_2\mathbf{U}$ and $\mathbf{W}'_1 = \mathbf{U}^{-1}\mathbf{W}_1$ will perform the same mapping

- There will be a huge space of equivalent networks

More Symmetries

- With n hidden nodes there are $n!$ symmetries
- Often if I multiply all the input and output weights to a node by -1 the network will be the same
- There are also continuous symmetries. For linear networks with two layers performing a mapping $\mathbf{W}_2\mathbf{W}_1$ then for any invertible matrix \mathbf{U} (where $\mathbf{U}\mathbf{U}^{-1} = \mathbf{I}$)

$$\mathbf{W}_2\mathbf{W}_1 = \mathbf{W}_2\mathbf{I}\mathbf{W}_1$$

so a network with weights $\mathbf{W}'_2 = \mathbf{W}_2\mathbf{U}$ and $\mathbf{W}'_1 = \mathbf{U}^{-1}\mathbf{W}_1$ will perform the same mapping

- There will be a huge space of equivalent networks

More Symmetries

- With n hidden nodes there are $n!$ symmetries
- Often if I multiply all the input and output weights to a node by -1 the network will be the same
- There are also continuous symmetries. For linear networks with two layers performing a mapping $\mathbf{W}_2\mathbf{W}_1$ then for any invertible matrix \mathbf{U} (where $\mathbf{U}\mathbf{U}^{-1} = \mathbf{I}$)

$$\mathbf{W}_2\mathbf{W}_1 = \mathbf{W}_2\mathbf{I}\mathbf{W}_1$$

so a network with weights $\mathbf{W}'_2 = \mathbf{W}_2\mathbf{U}$ and $\mathbf{W}'_1 = \mathbf{U}^{-1}\mathbf{W}_1$ will perform the same mapping

- There will be a huge space of equivalent networks

More Symmetries

- With n hidden nodes there are $n!$ symmetries
- Often if I multiply all the input and output weights to a node by -1 the network will be the same
- There are also continuous symmetries. For linear networks with two layers performing a mapping $\mathbf{W}_2\mathbf{W}_1$ then for any invertible matrix \mathbf{U} (where $\mathbf{U}\mathbf{U}^{-1} = \mathbf{I}$)

$$\mathbf{W}_2\mathbf{W}_1 = \mathbf{W}_2\mathbf{I}\mathbf{W}_1 = \mathbf{W}_2(\mathbf{U}\mathbf{U}^{-1})\mathbf{W}_1$$

so a network with weights $\mathbf{W}'_2 = \mathbf{W}_2\mathbf{U}$ and $\mathbf{W}'_1 = \mathbf{U}^{-1}\mathbf{W}_1$ will perform the same mapping

- There will be a huge space of equivalent networks

More Symmetries

- With n hidden nodes there are $n!$ symmetries
- Often if I multiply all the input and output weights to a node by -1 the network will be the same
- There are also continuous symmetries. For linear networks with two layers performing a mapping $\mathbf{W}_2\mathbf{W}_1$ then for any invertible matrix \mathbf{U} (where $\mathbf{U}\mathbf{U}^{-1} = \mathbf{I}$)

$$\mathbf{W}_2\mathbf{W}_1 = \mathbf{W}_2\mathbf{I}\mathbf{W}_1 = \mathbf{W}_2(\mathbf{U}\mathbf{U}^{-1})\mathbf{W}_1 = (\mathbf{W}_2\mathbf{U})(\mathbf{U}^{-1}\mathbf{W}_1)$$

so a network with weights $\mathbf{W}'_2 = \mathbf{W}_2\mathbf{U}$ and $\mathbf{W}'_1 = \mathbf{U}^{-1}\mathbf{W}_1$ will perform the same mapping

- There will be a huge space of equivalent networks

More Symmetries

- With n hidden nodes there are $n!$ symmetries
- Often if I multiply all the input and output weights to a node by -1 the network will be the same
- There are also continuous symmetries. For linear networks with two layers performing a mapping $\mathbf{W}_2\mathbf{W}_1$ then for any invertible matrix \mathbf{U} (where $\mathbf{U}\mathbf{U}^{-1} = \mathbf{I}$)

$$\mathbf{W}_2\mathbf{W}_1 = \mathbf{W}_2\mathbf{I}\mathbf{W}_1 = \mathbf{W}_2(\mathbf{U}\mathbf{U}^{-1})\mathbf{W}_1 = (\mathbf{W}_2\mathbf{U})(\mathbf{U}^{-1}\mathbf{W}_1)$$

so a network with weights $\mathbf{W}'_2 = \mathbf{W}_2\mathbf{U}$ and $\mathbf{W}'_1 = \mathbf{U}^{-1}\mathbf{W}_1$ will perform the same mapping

- There will be a huge space of equivalent networks

More Symmetries

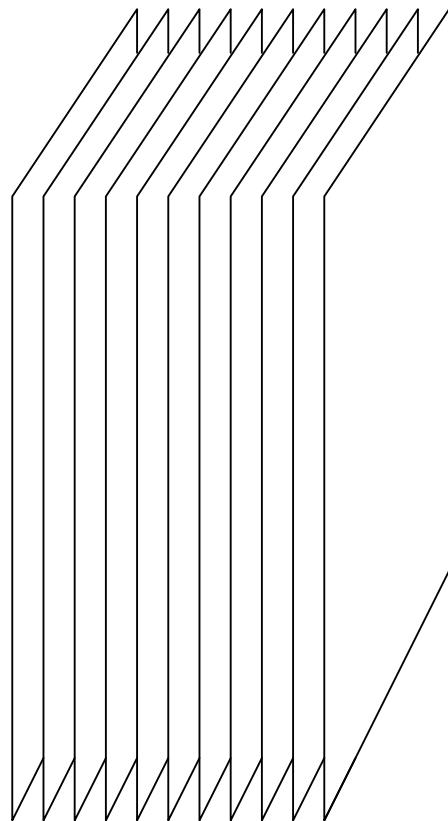
- With n hidden nodes there are $n!$ symmetries
- Often if I multiply all the input and output weights to a node by -1 the network will be the same
- There are also continuous symmetries. For linear networks with two layers performing a mapping $\mathbf{W}_2\mathbf{W}_1$ then for any invertible matrix \mathbf{U} (where $\mathbf{U}\mathbf{U}^{-1} = \mathbf{I}$)

$$\mathbf{W}_2\mathbf{W}_1 = \mathbf{W}_2\mathbf{I}\mathbf{W}_1 = \mathbf{W}_2(\mathbf{U}\mathbf{U}^{-1})\mathbf{W}_1 = (\mathbf{W}_2\mathbf{U})(\mathbf{U}^{-1}\mathbf{W}_1)$$

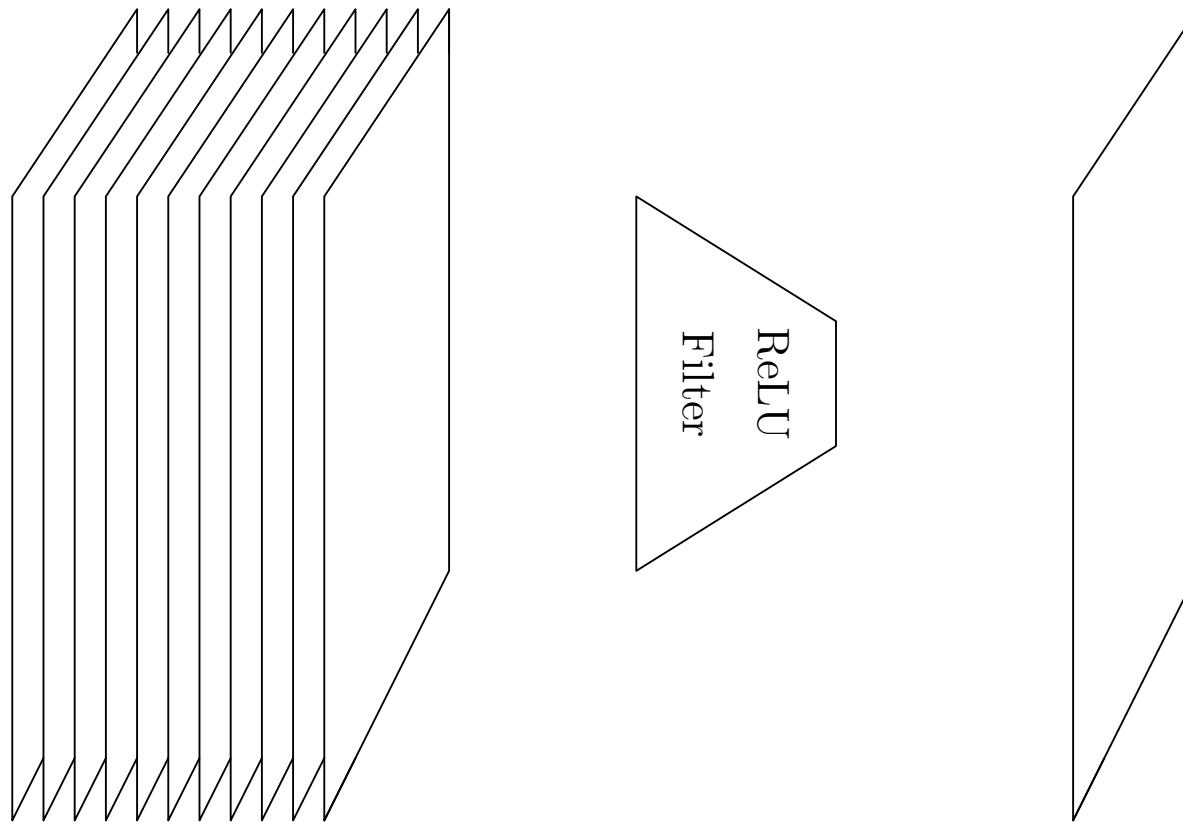
so a network with weights $\mathbf{W}'_2 = \mathbf{W}_2\mathbf{U}$ and $\mathbf{W}'_1 = \mathbf{U}^{-1}\mathbf{W}_1$ will perform the same mapping

- There will be a huge space of equivalent networks

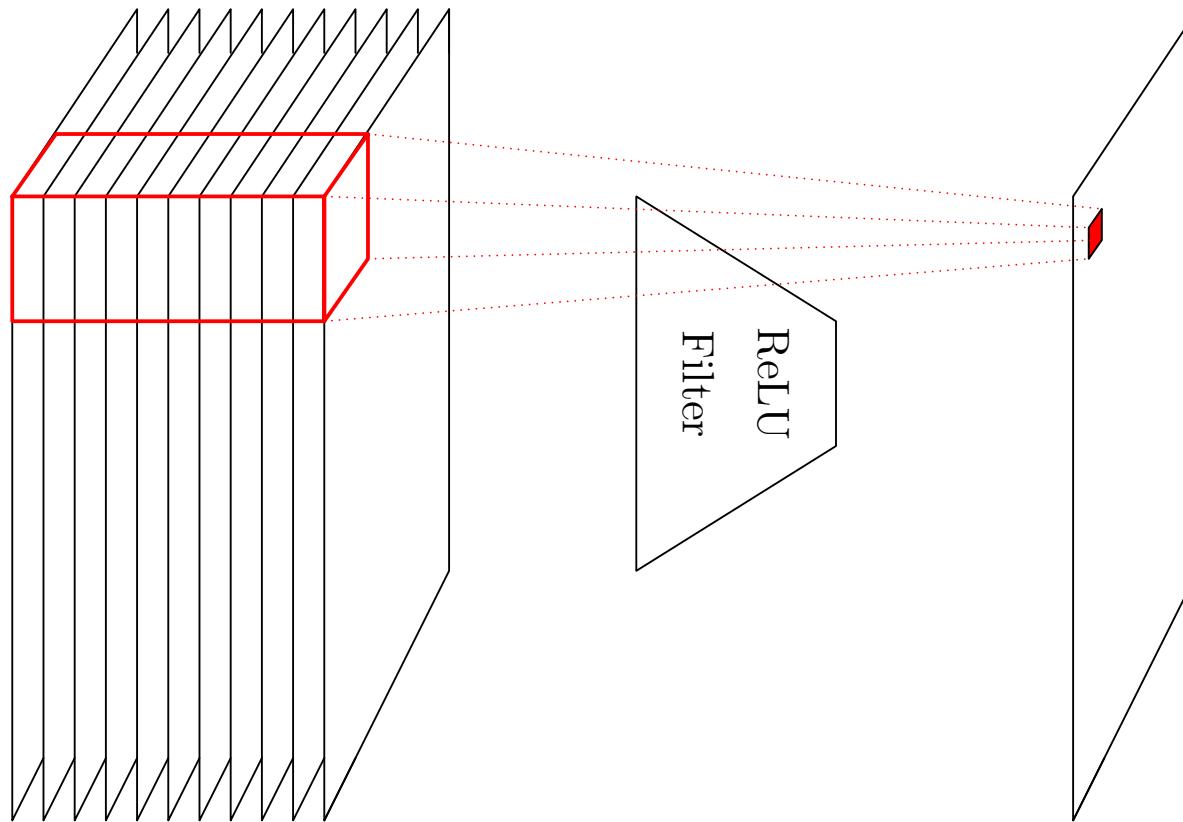
Skip Connections



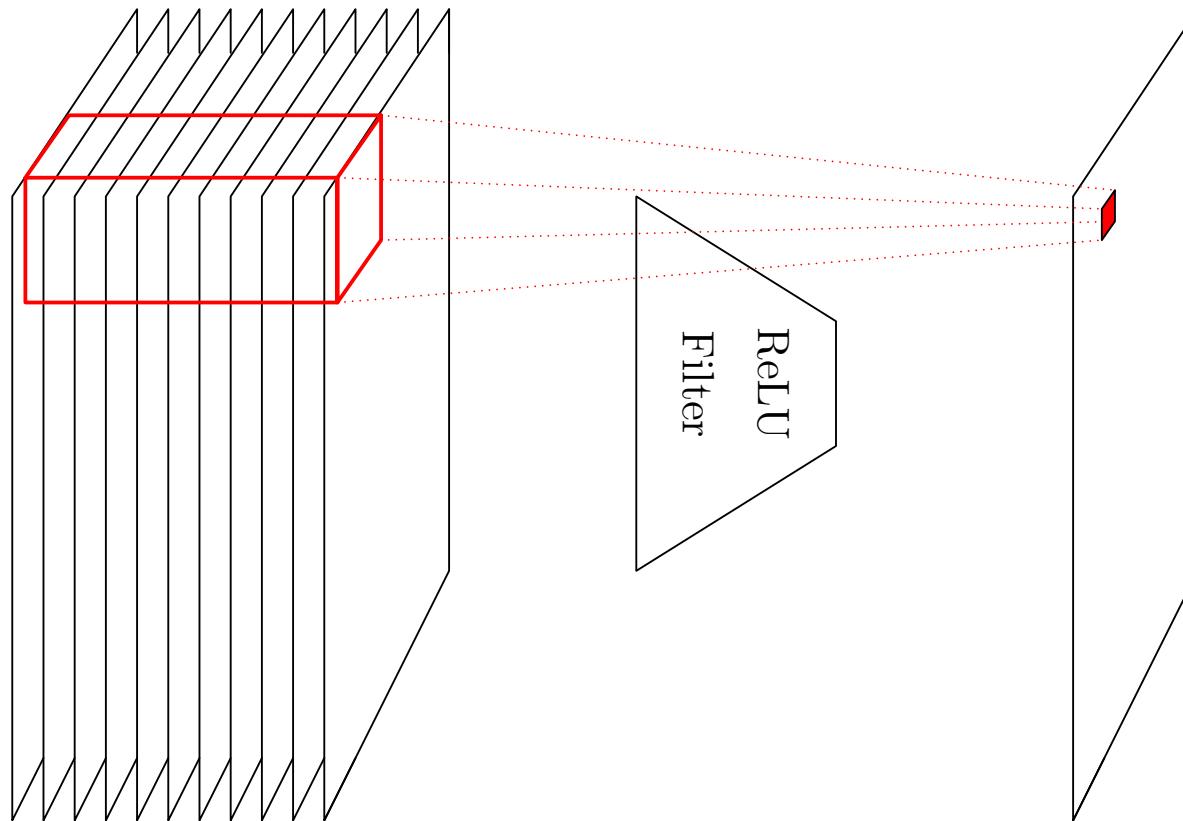
Skip Connections



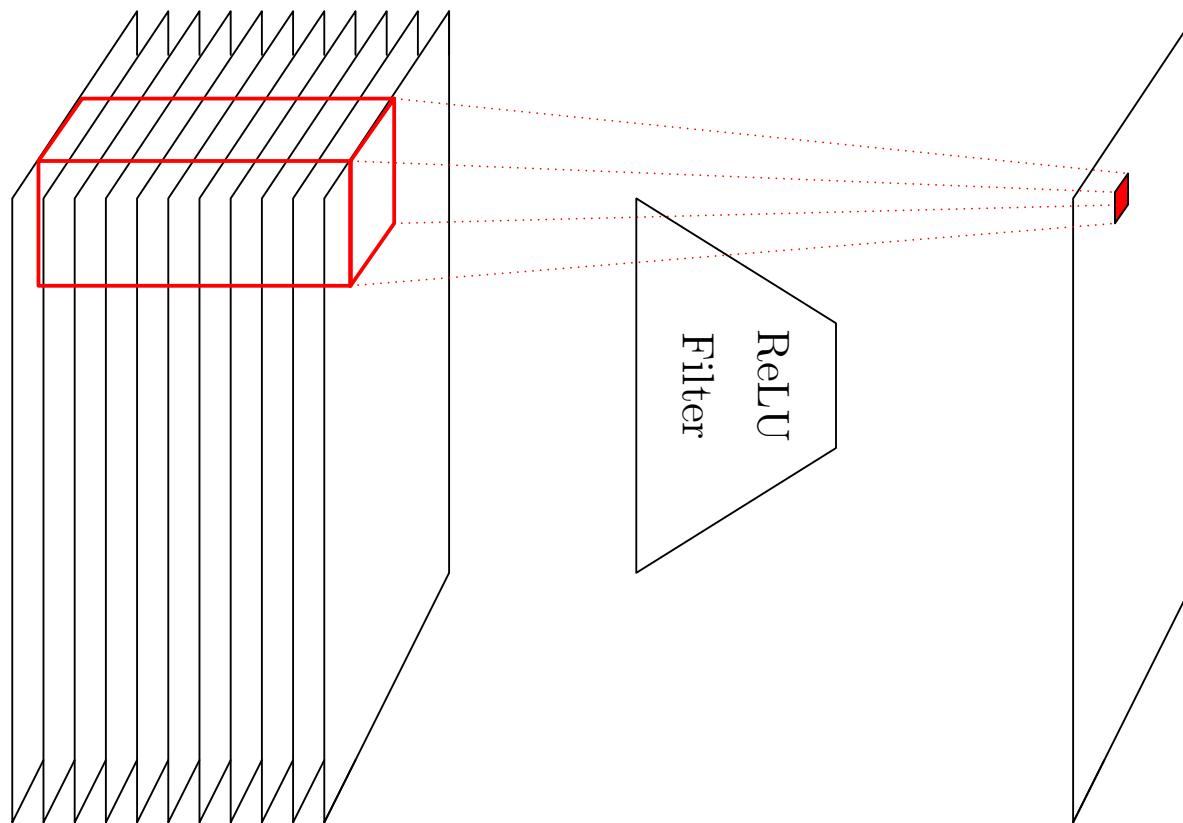
Skip Connections



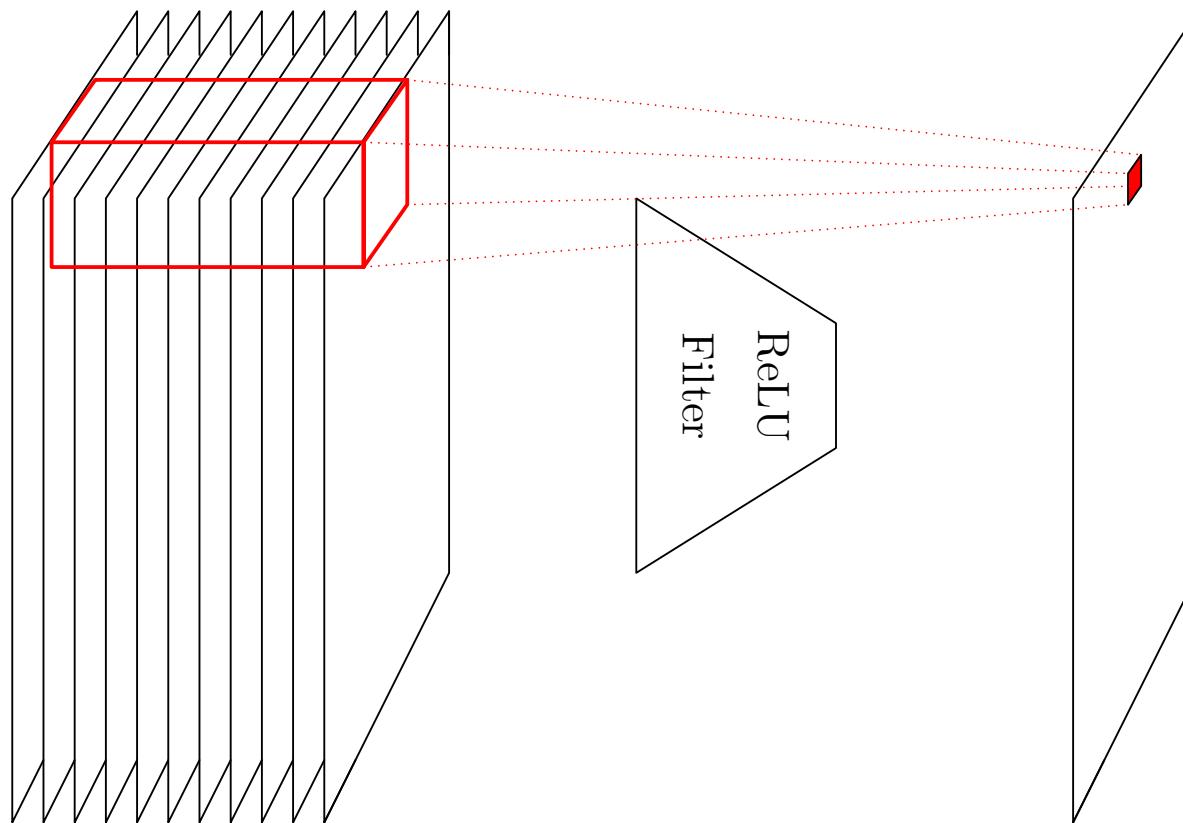
Skip Connections



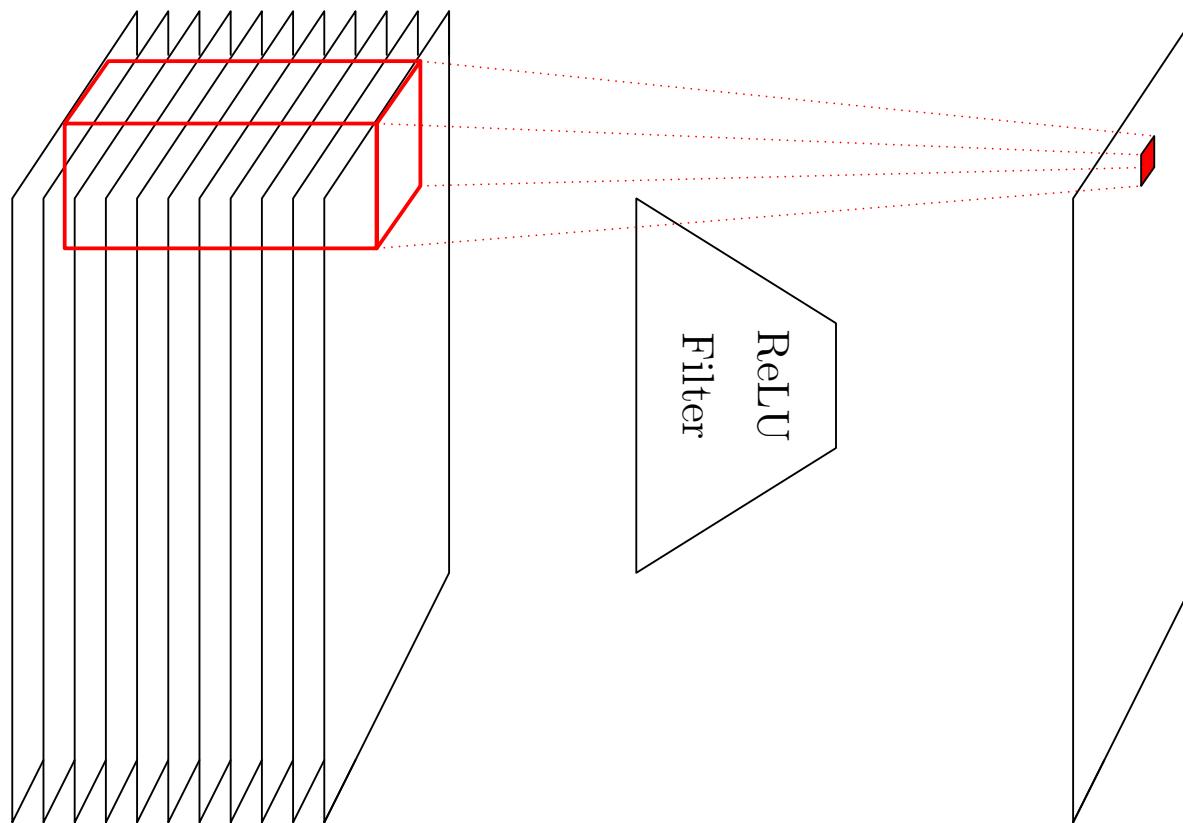
Skip Connections



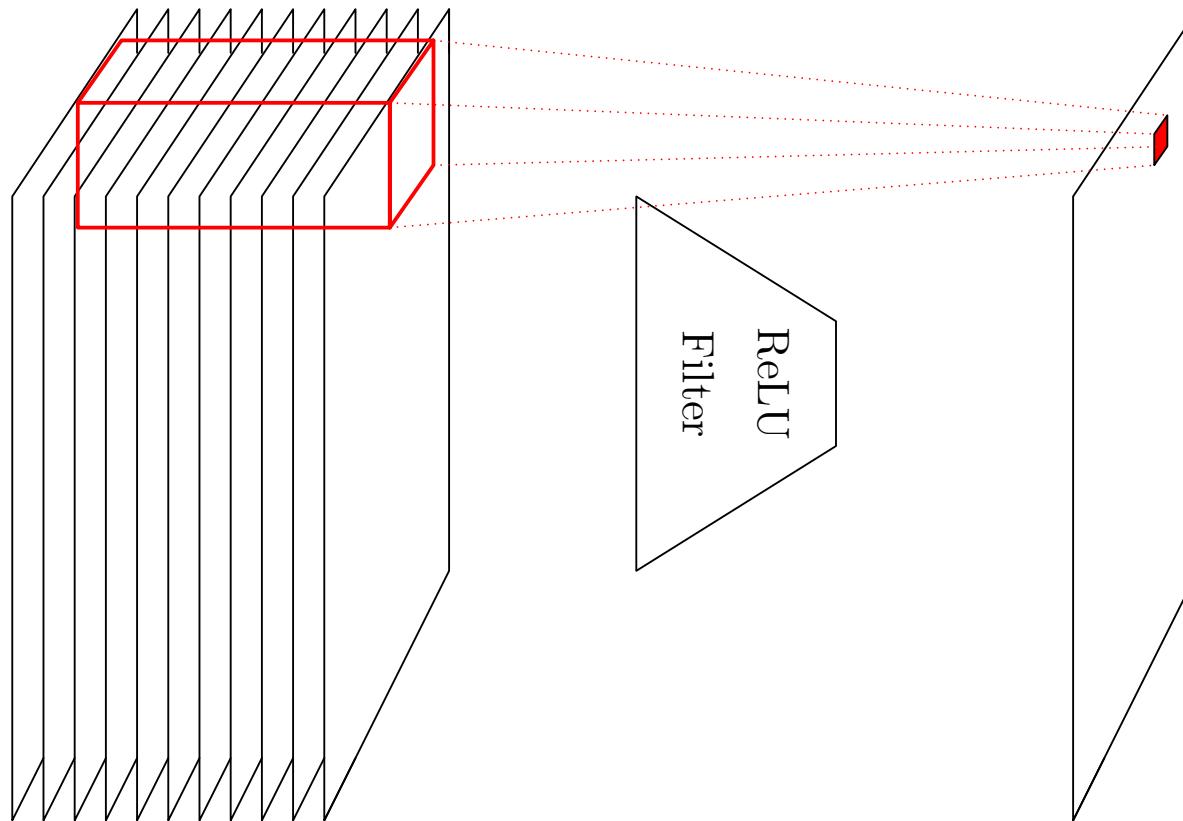
Skip Connections



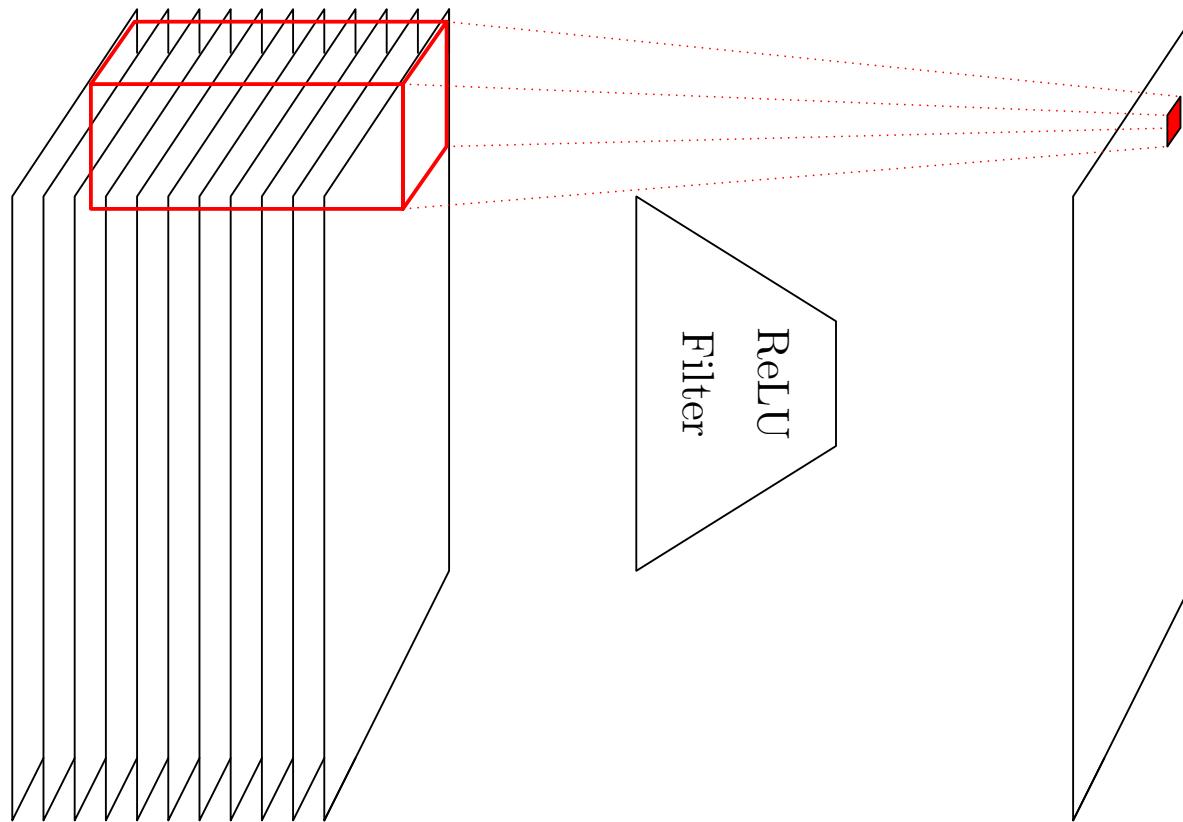
Skip Connections



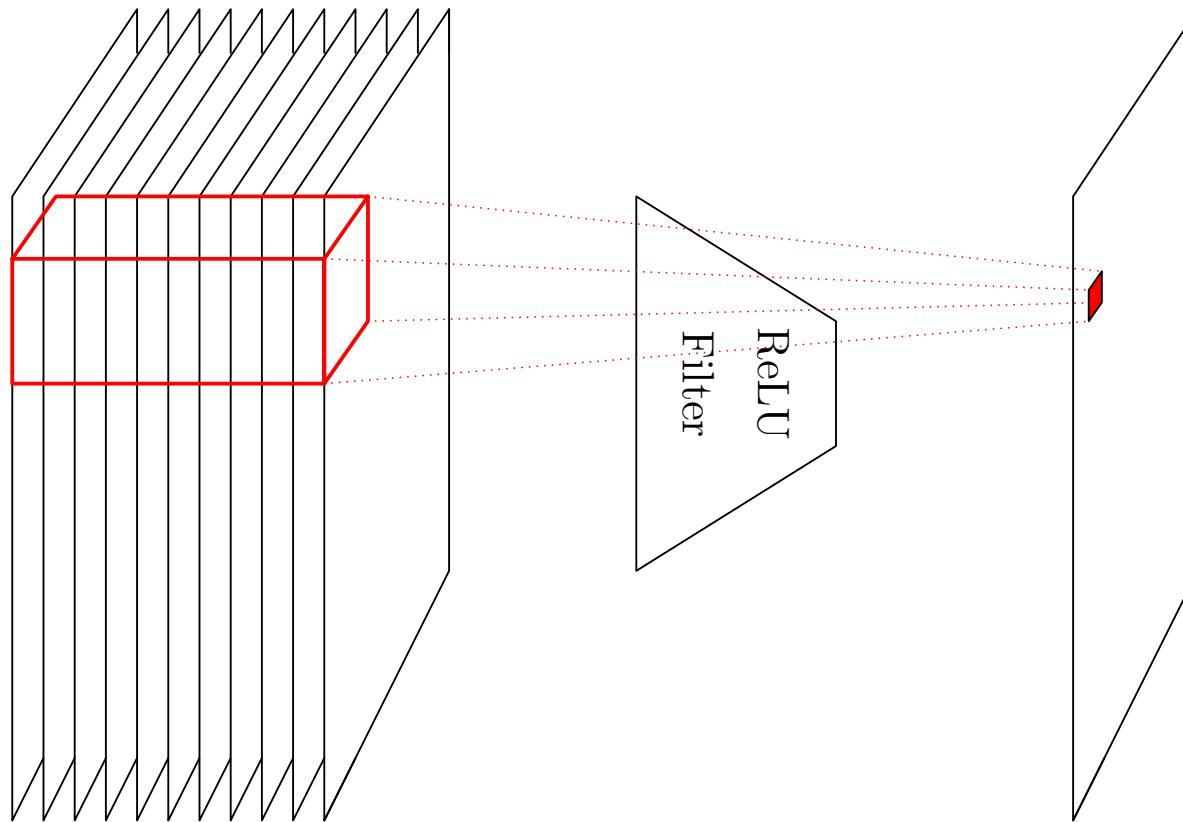
Skip Connections



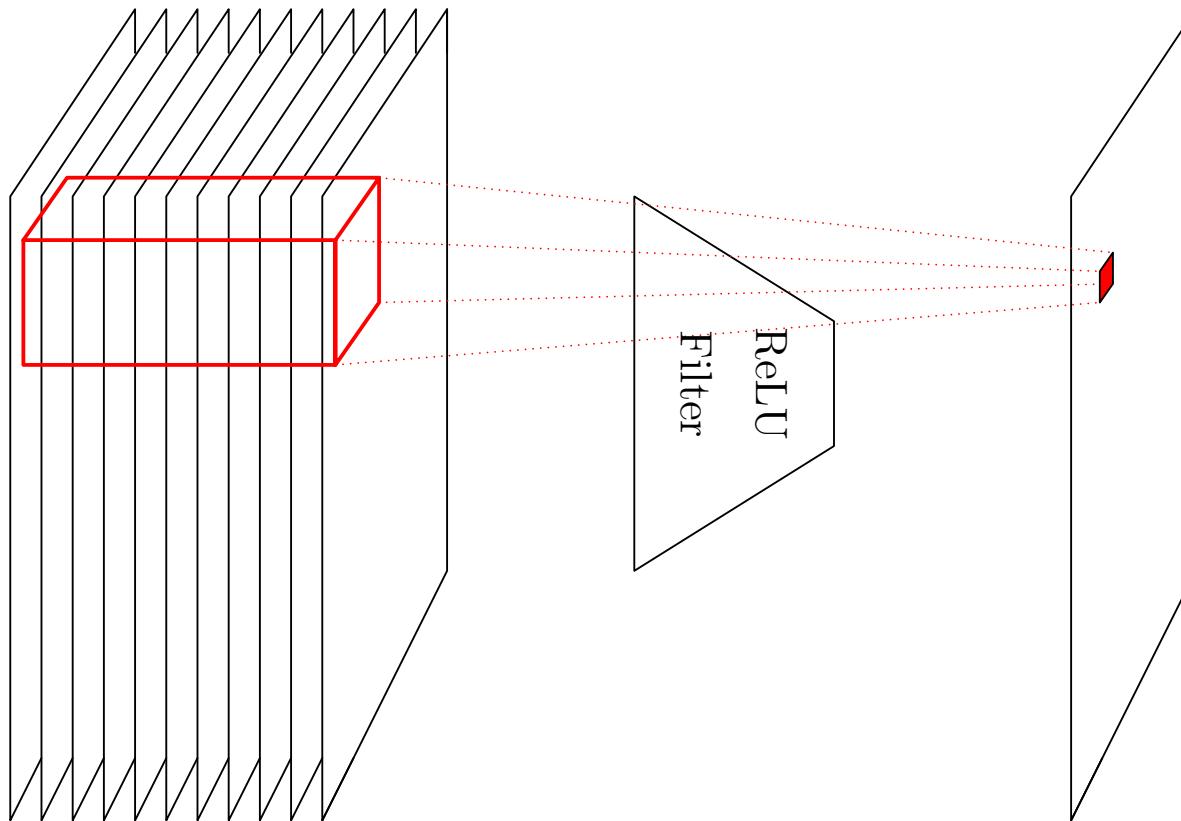
Skip Connections



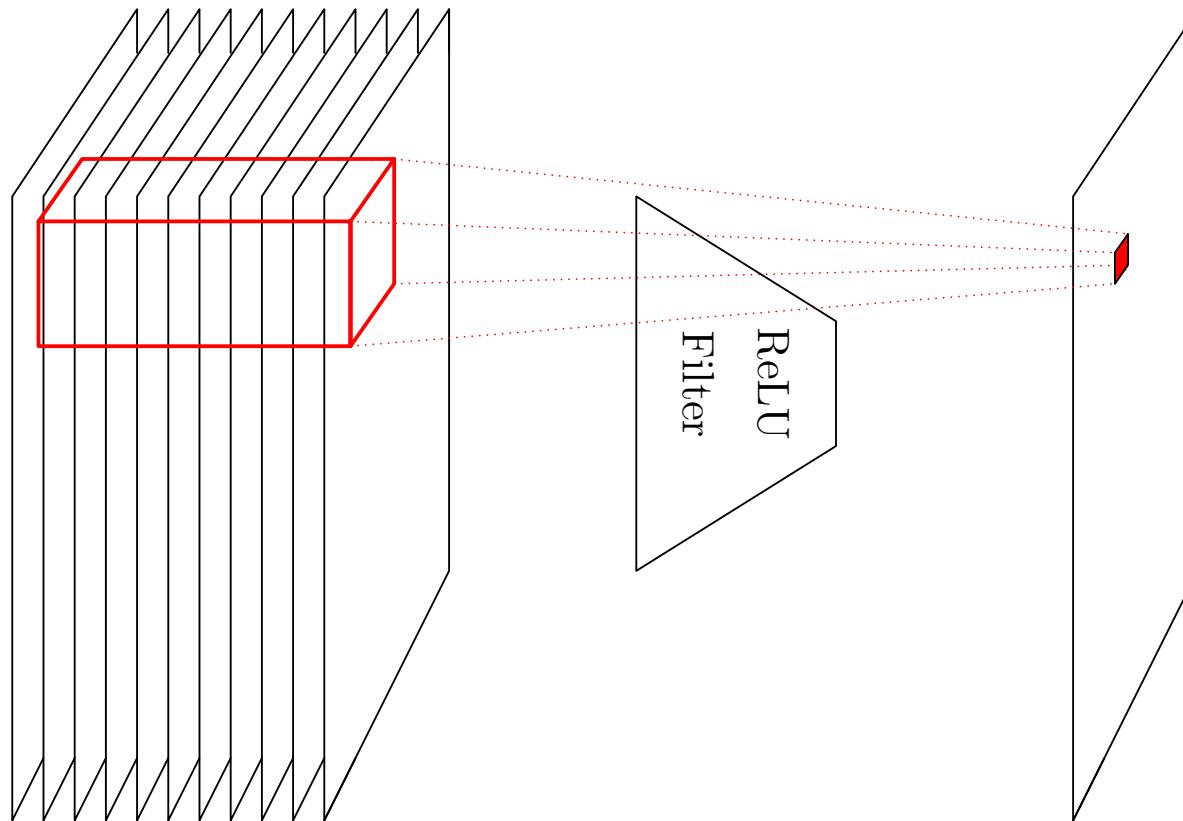
Skip Connections



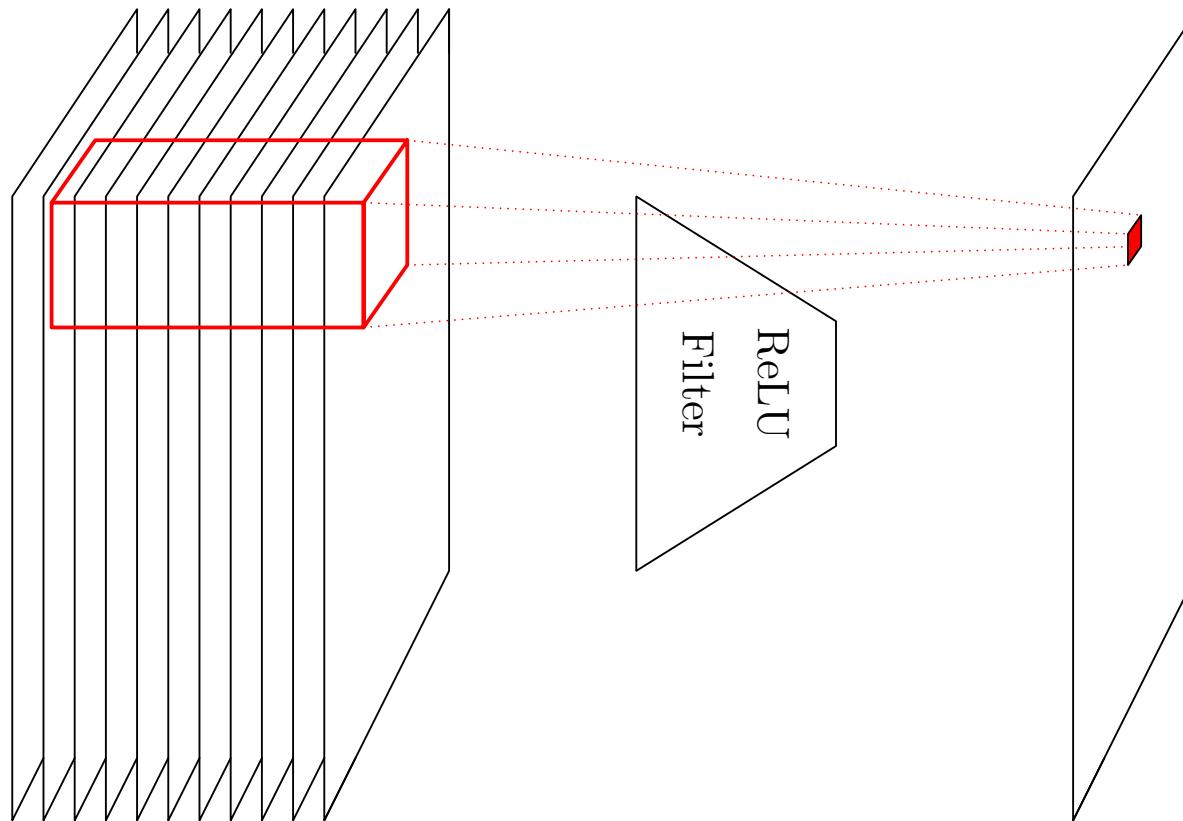
Skip Connections



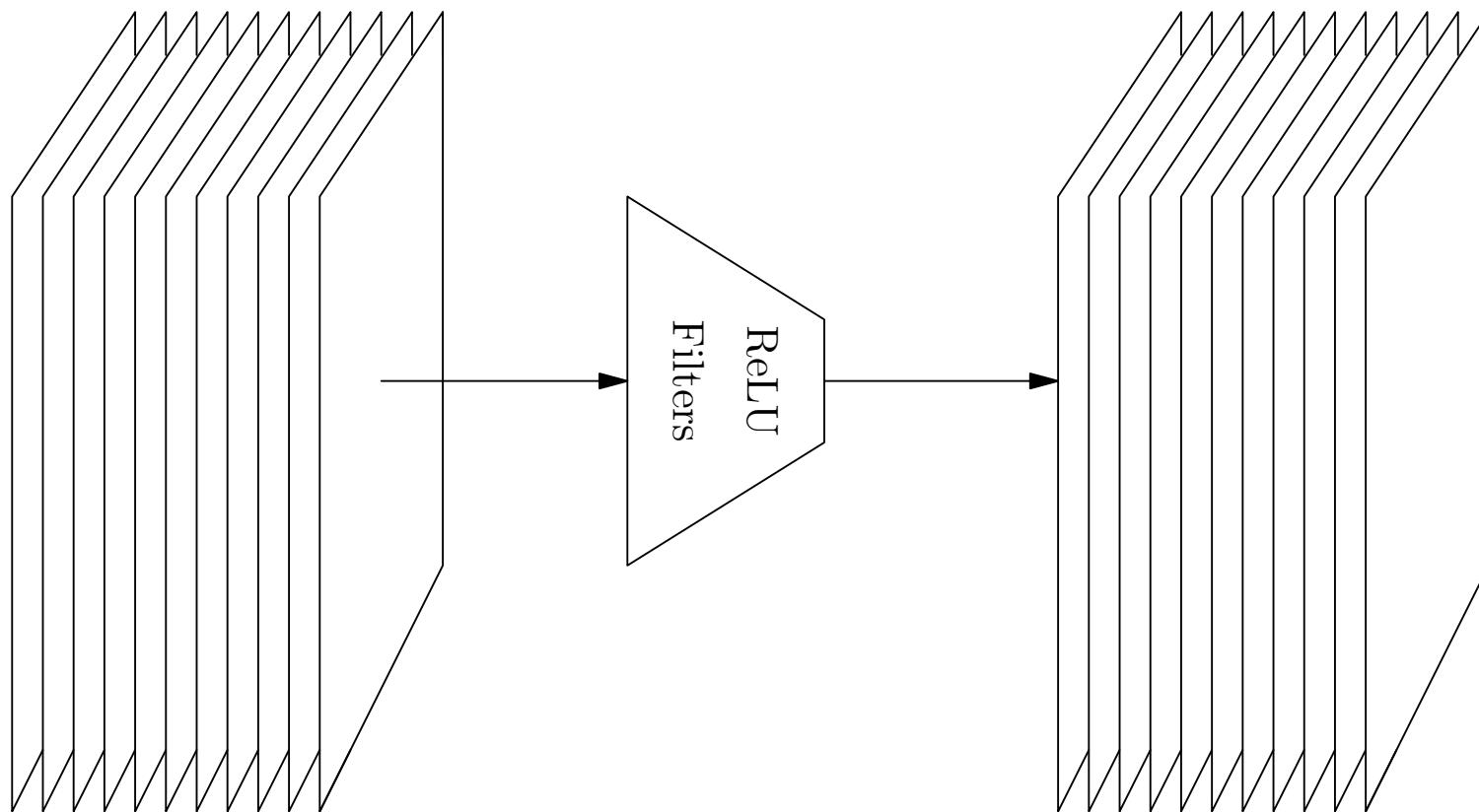
Skip Connections



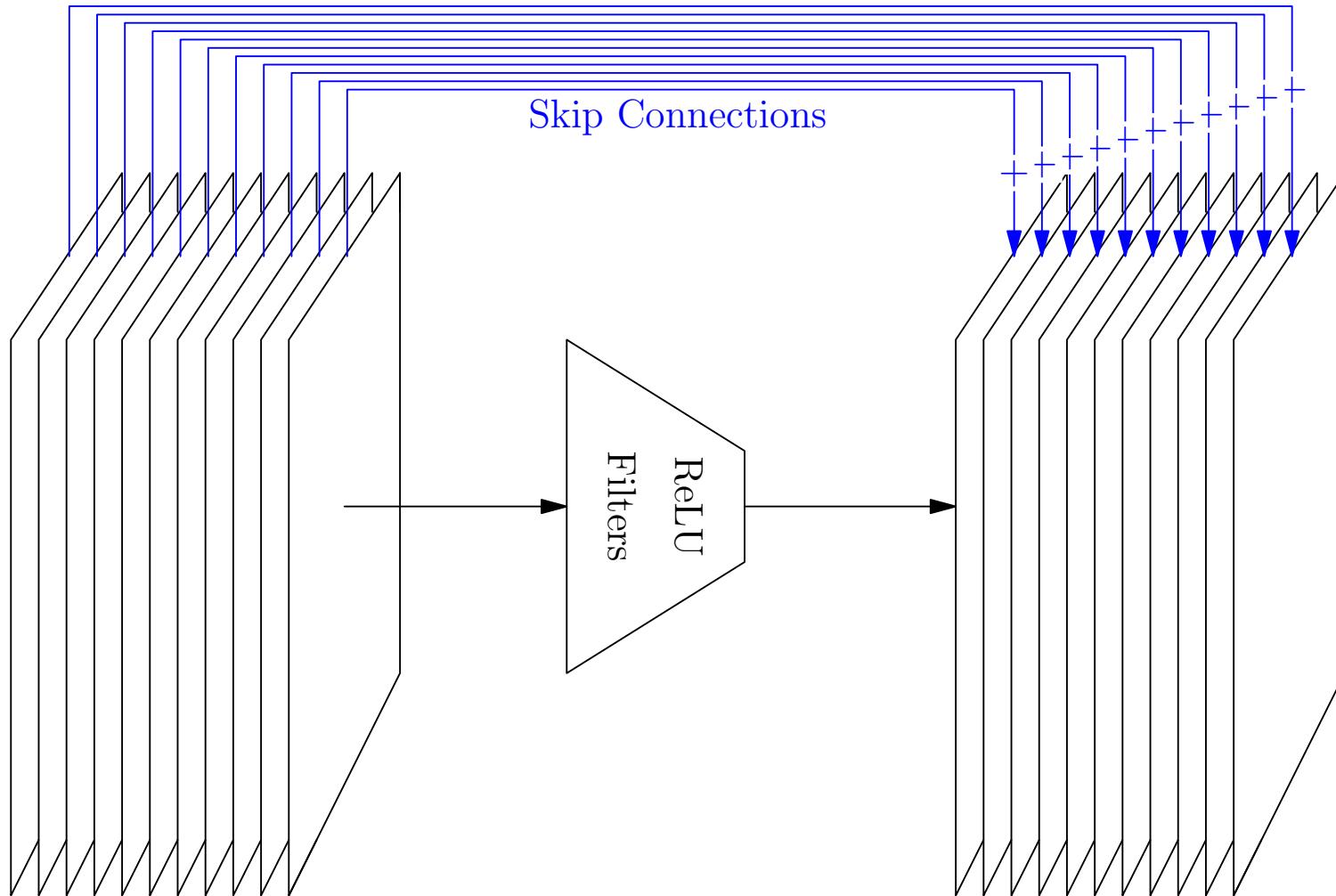
Skip Connections



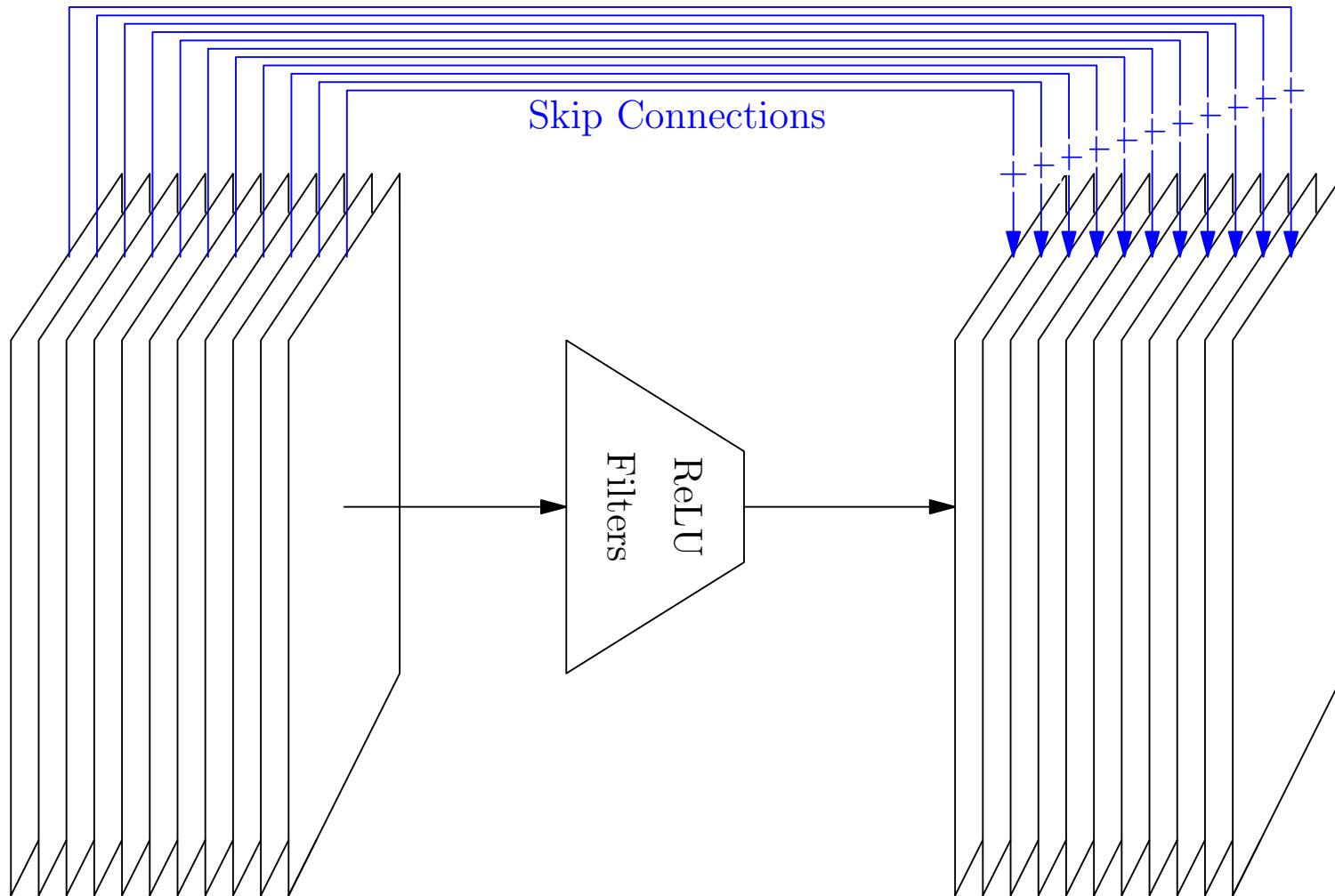
Skip Connections



Skip Connections

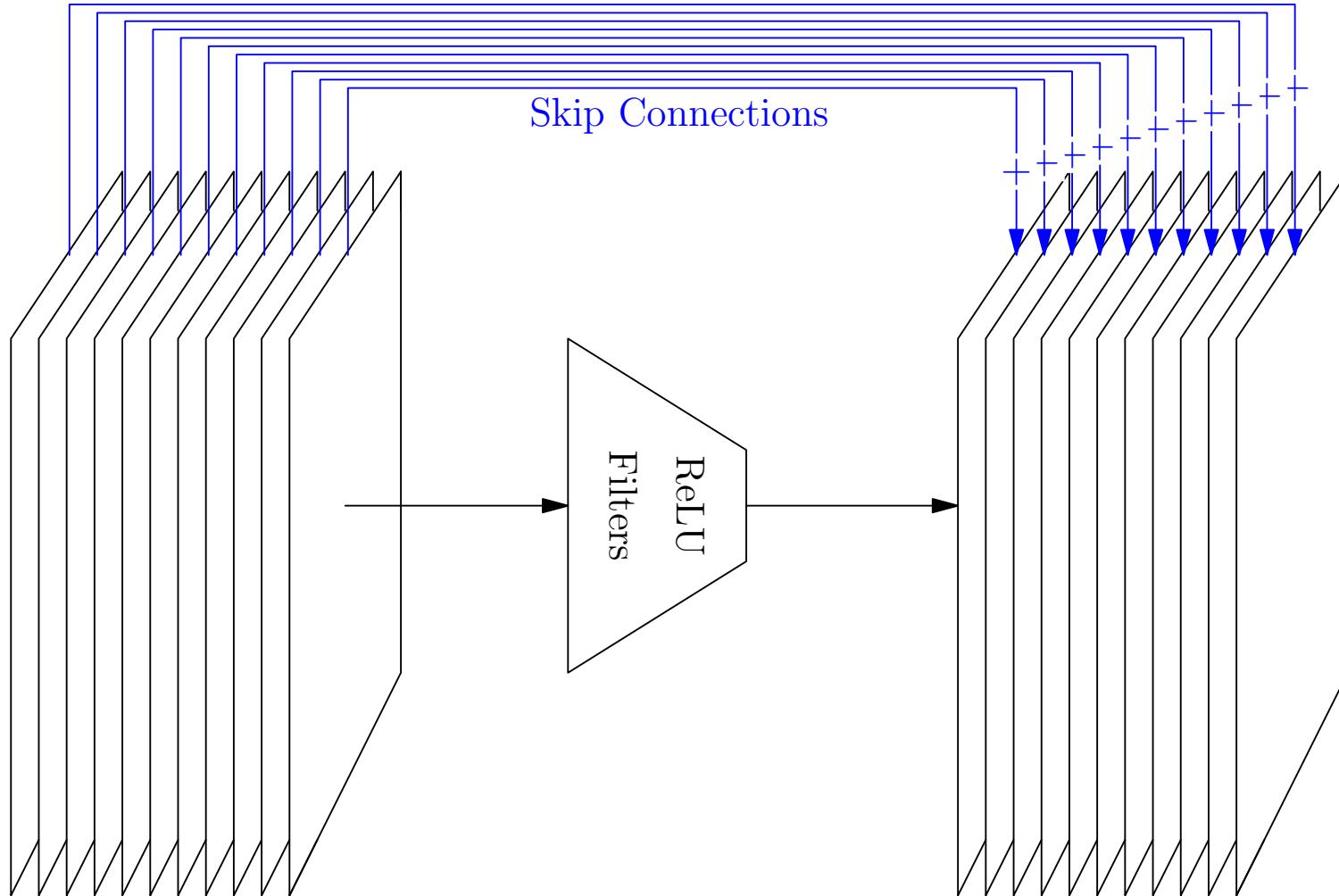


Skip Connections



- Breaks permutation symmetry

Skip Connections



- Breaks permutation symmetry. Used in Resnets, transformer, etc.

Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance

Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance

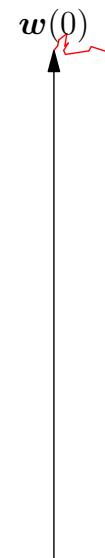
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



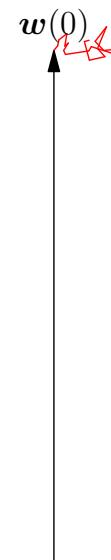
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



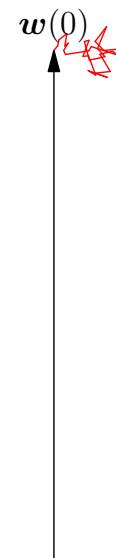
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



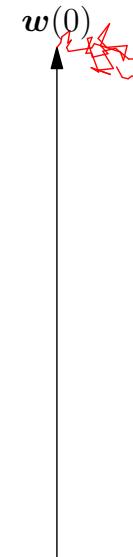
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



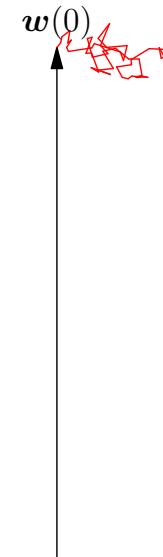
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



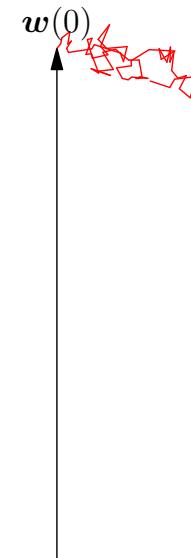
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



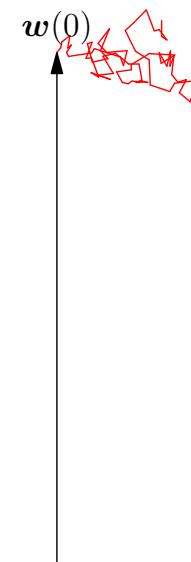
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



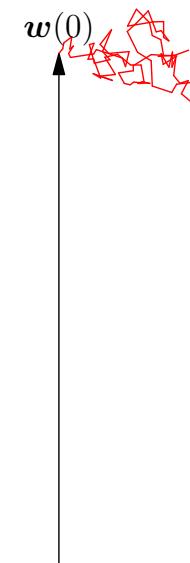
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



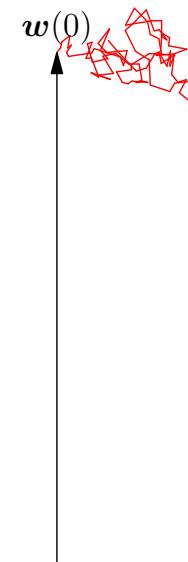
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



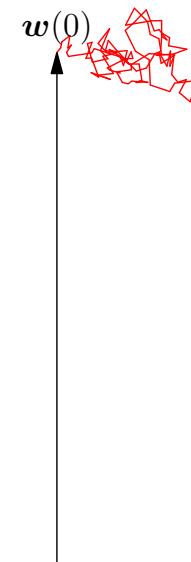
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



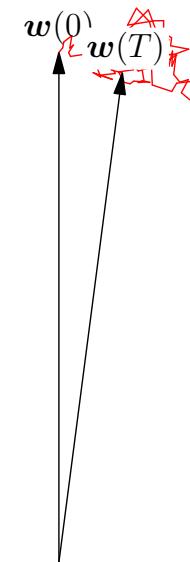
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



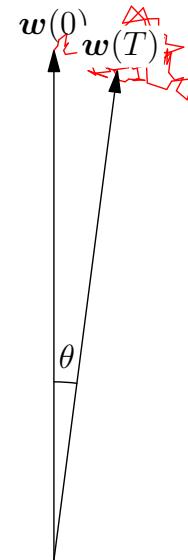
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



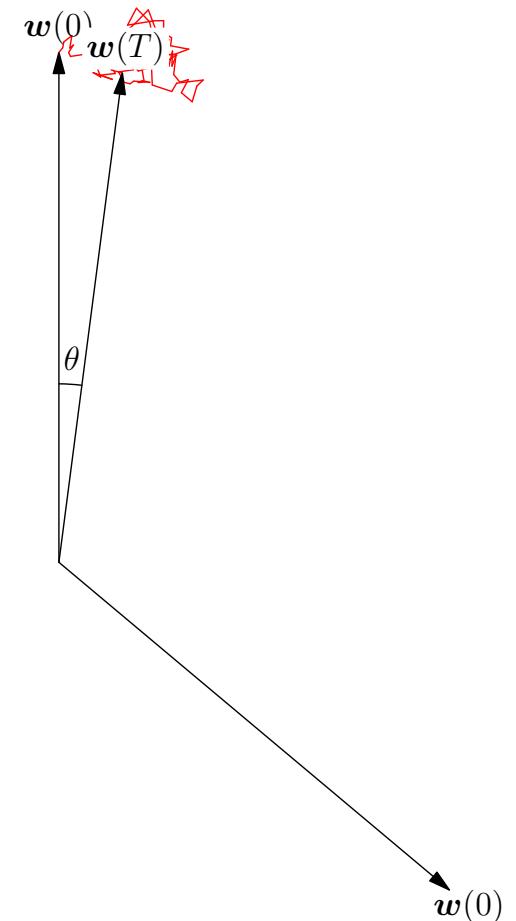
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



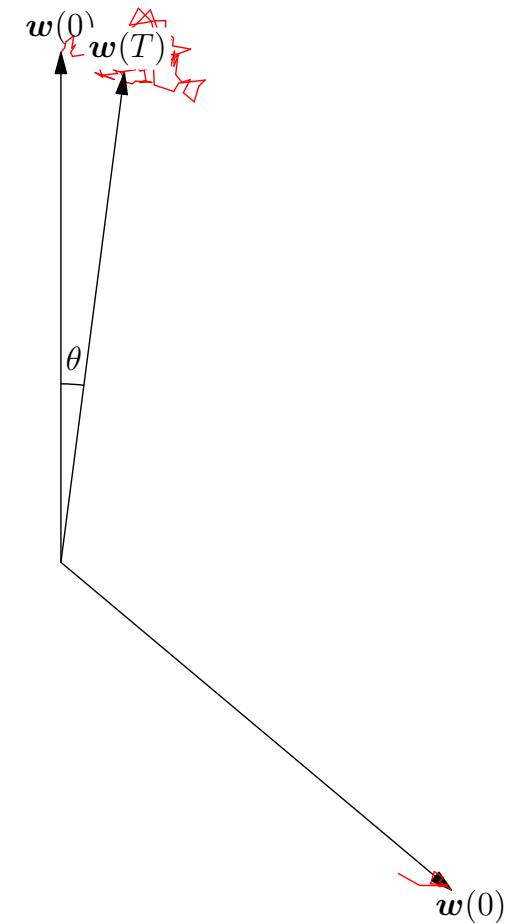
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



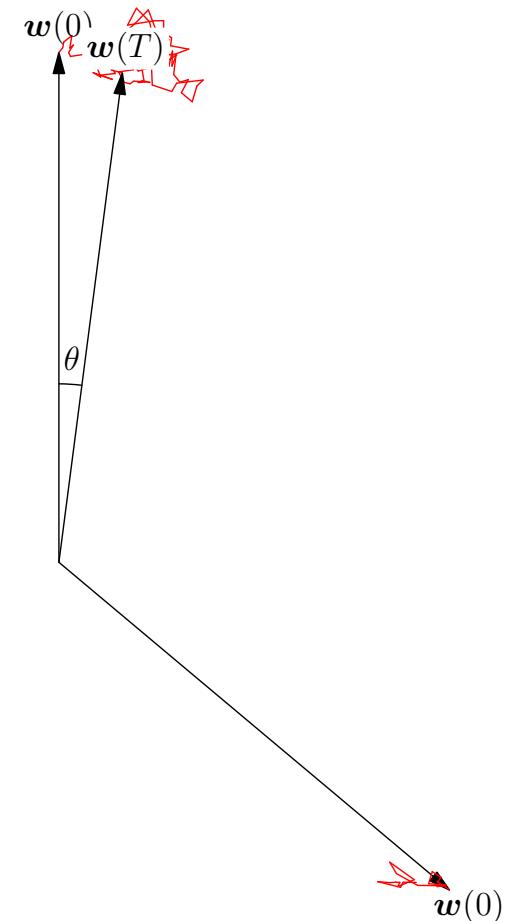
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



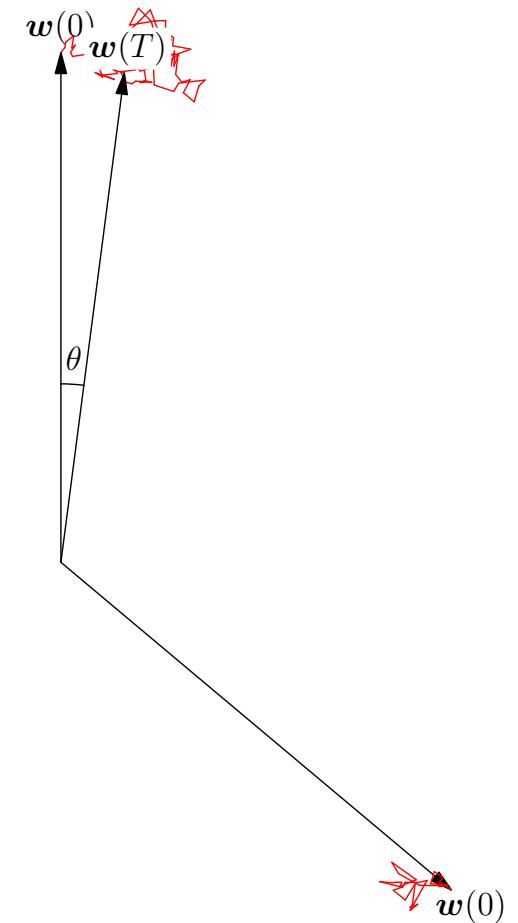
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



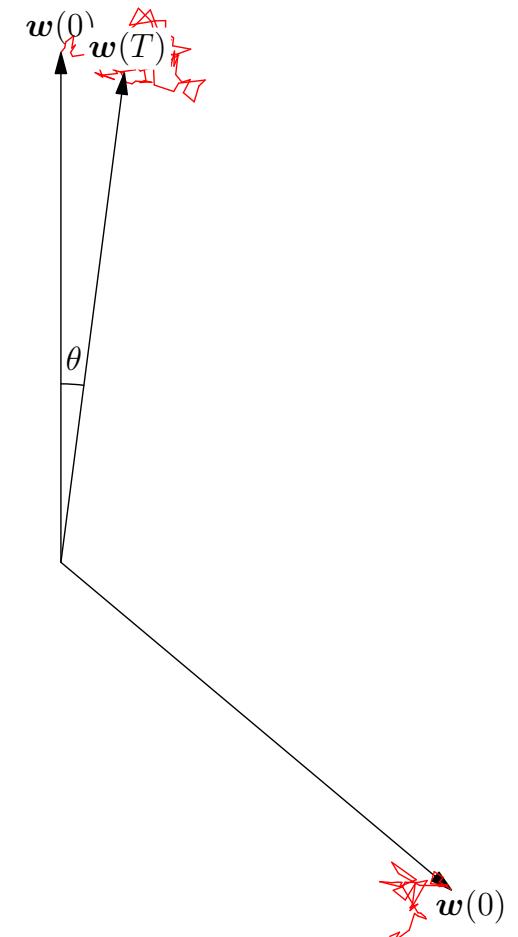
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



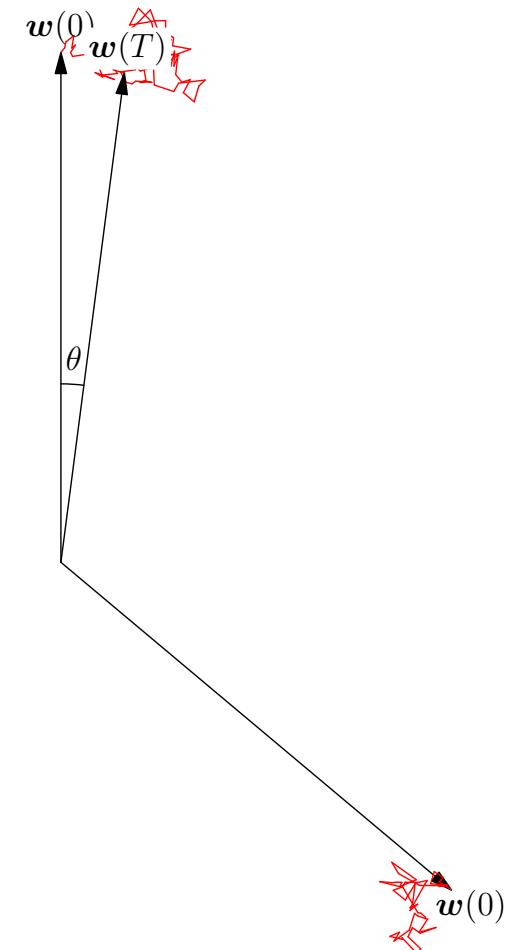
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



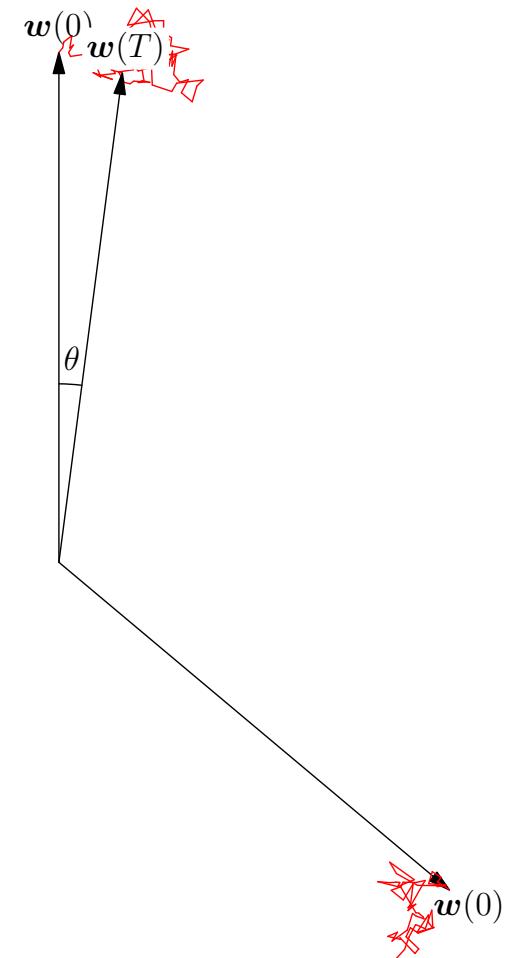
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



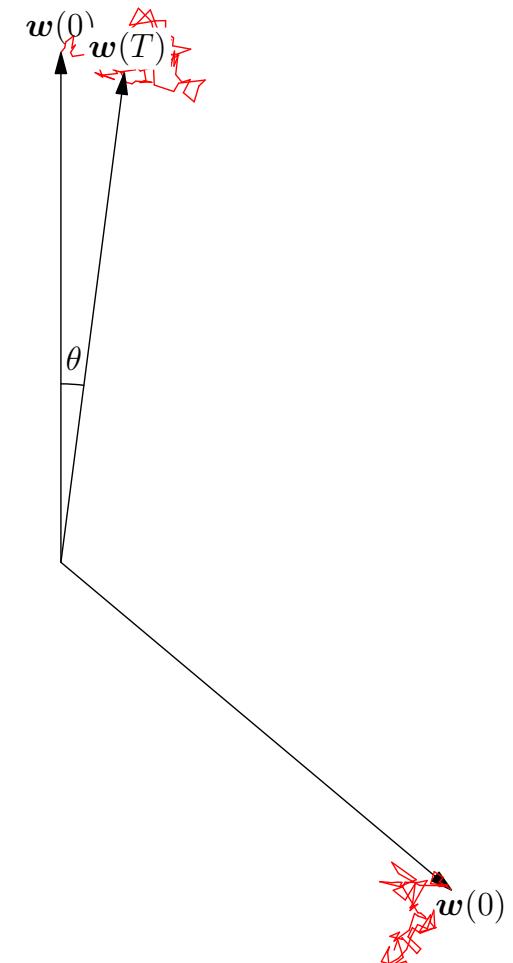
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



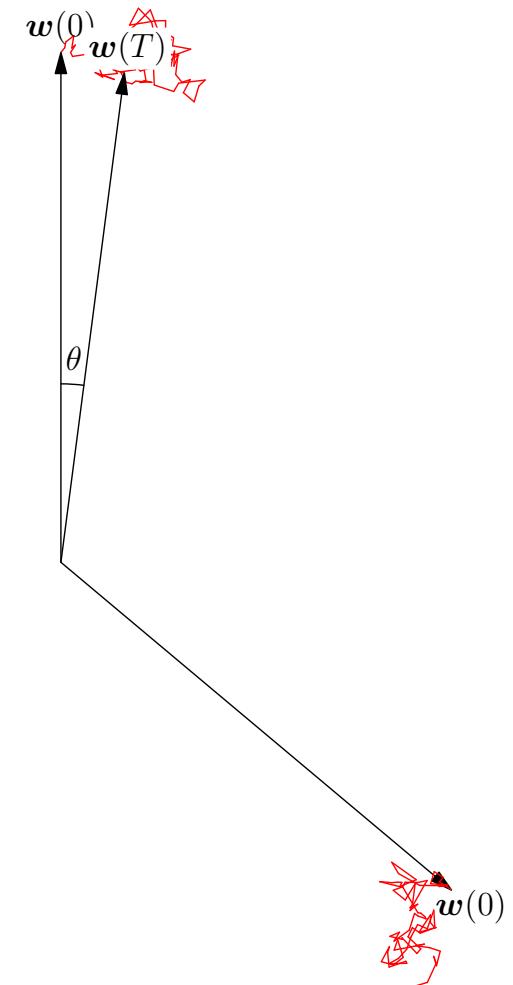
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



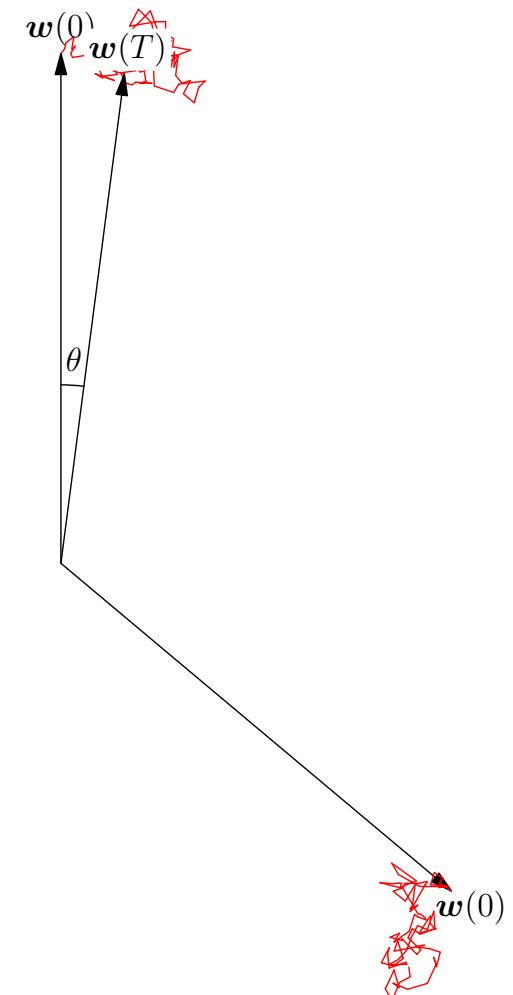
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



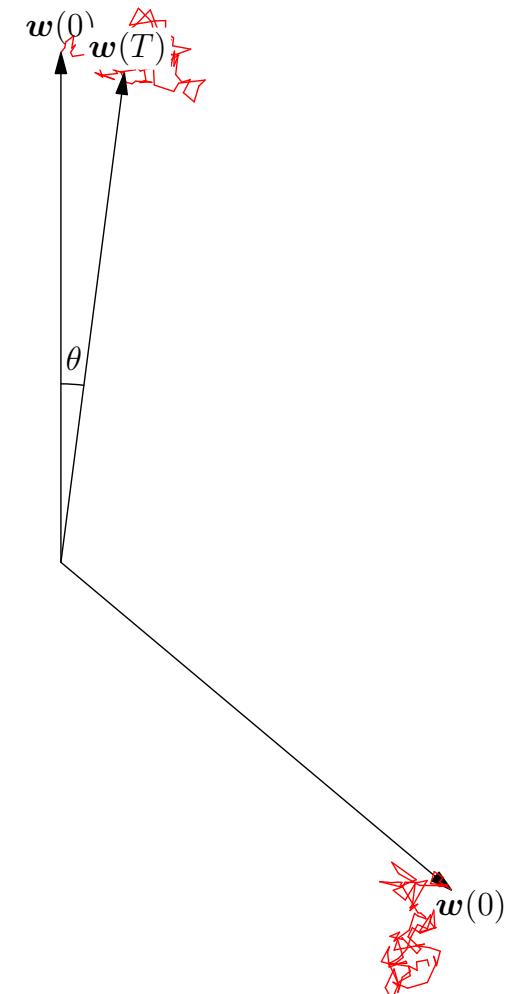
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



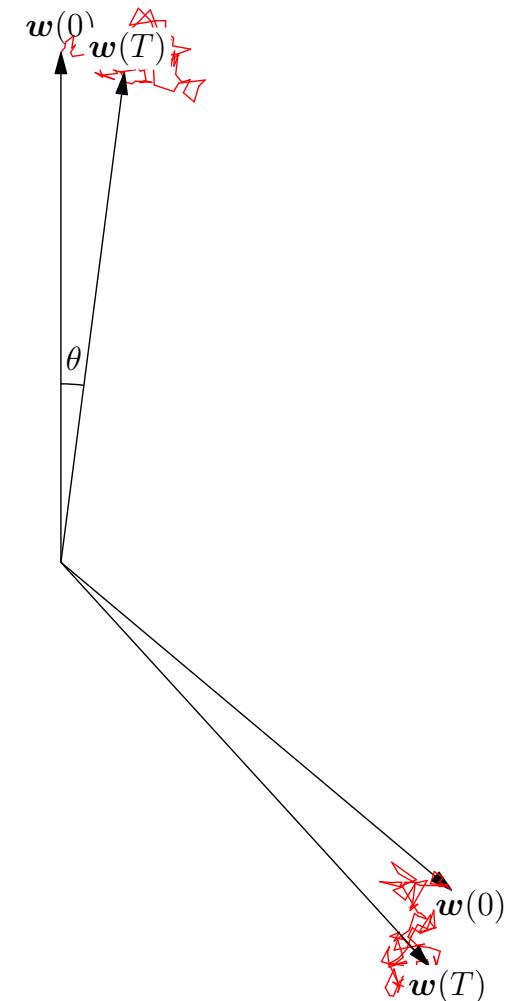
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



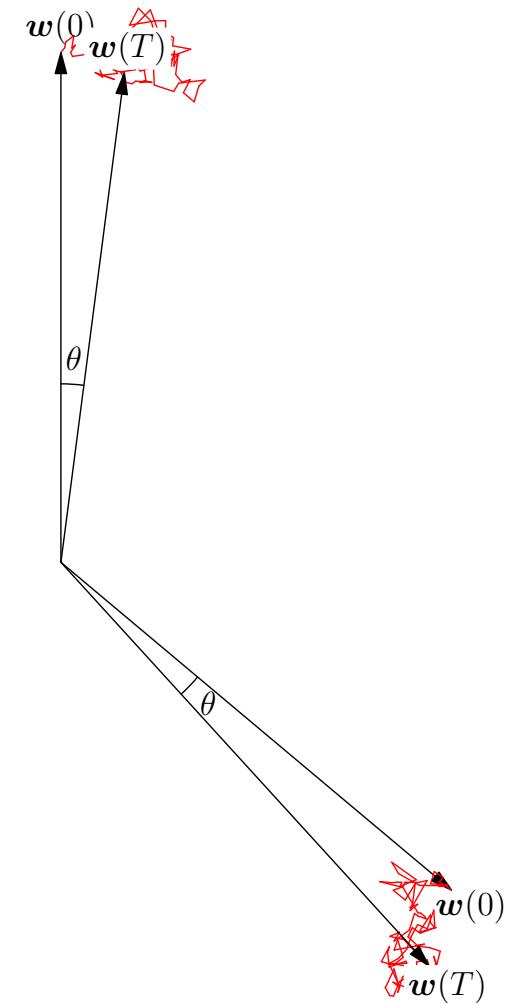
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



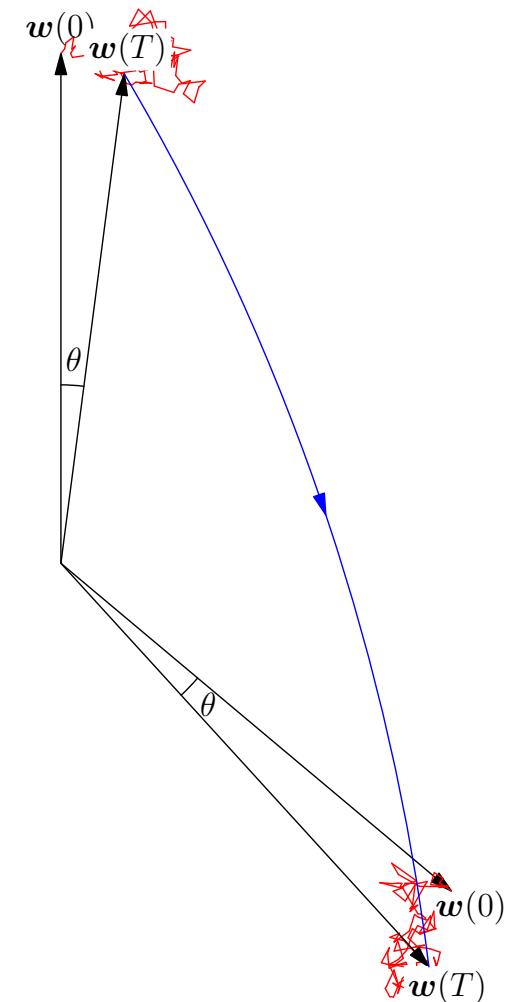
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



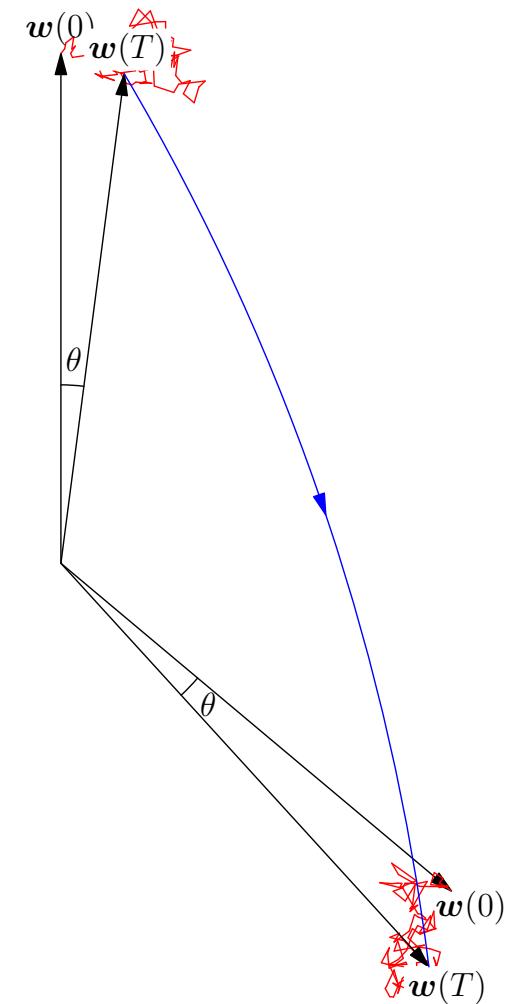
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



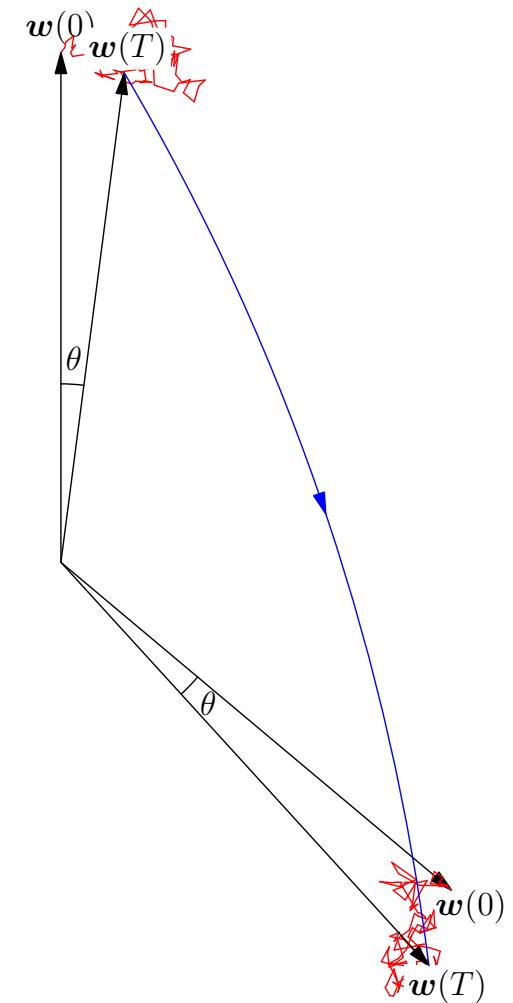
Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance



Lessons

- SGD together with automatic differentiation has revolutionised machine learning
- There are a number of techniques to get the step size right
- Still need to understand that we are exploring a huge and complex loss landscape
- What works is problem specific

Lessons

- SGD together with automatic differentiation has revolutionised machine learning
- There are a number of techniques to get the step size right
- Still need to understand that we are exploring a huge and complex loss landscape
- What works is problem specific

Lessons

- SGD together with automatic differentiation has revolutionised machine learning
- There are a number of techniques to get the step size right:
momentum
- Still need to understand that we are exploring a huge and complex loss landscape
- What works is problem specific

Lessons

- SGD together with automatic differentiation has revolutionised machine learning
- There are a number of techniques to get the step size right: momentum, **adaptive step size**
- Still need to understand that we are exploring a huge and complex loss landscape
- What works is problem specific

Lessons

- SGD together with automatic differentiation has revolutionised machine learning
- There are a number of techniques to get the step size right: momentum, adaptive step size, **ADAM**
- Still need to understand that we are exploring a huge and complex loss landscape
- What works is problem specific

Lessons

- SGD together with automatic differentiation has revolutionised machine learning
- There are a number of techniques to get the step size right: momentum, adaptive step size, ADAM
- Still need to understand that we are exploring a huge and complex loss landscape
- What works is problem specific

Lessons

- SGD together with automatic differentiation has revolutionised machine learning
- There are a number of techniques to get the step size right: momentum, adaptive step size, ADAM
- Still need to understand that we are exploring a huge and complex loss landscape
- What works is problem specific

Lessons

- SGD together with automatic differentiation has revolutionised machine learning
- There are a number of techniques to get the step size right: momentum, adaptive step size, ADAM
- Still need to understand that we are exploring a huge and complex loss landscape
- What works is problem specific (**as always**)