SPATIAL ECOLOGY

Intro to optimizers
&
Perceptron

Antonio Fonseca

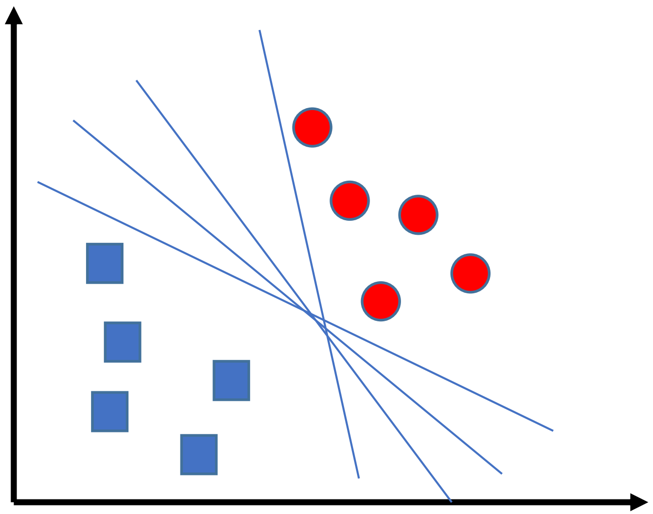
Agenda

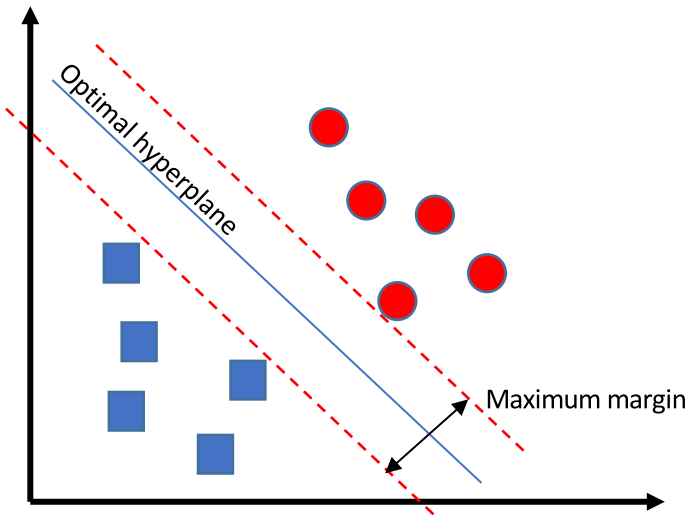
1) Perceptron

- Intro to optimization
- Perceptron
- Optimizers
- Hands-on tutorial

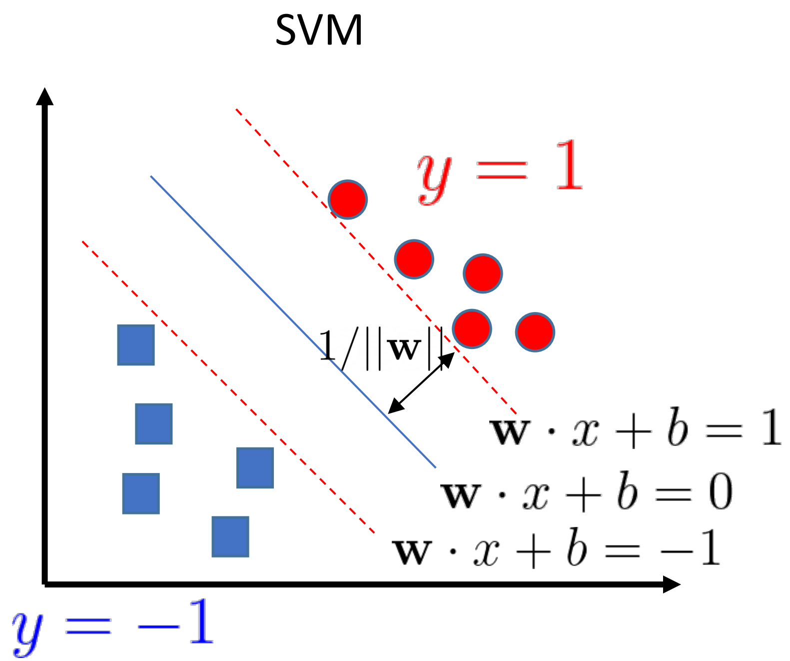
Support Vector Machine

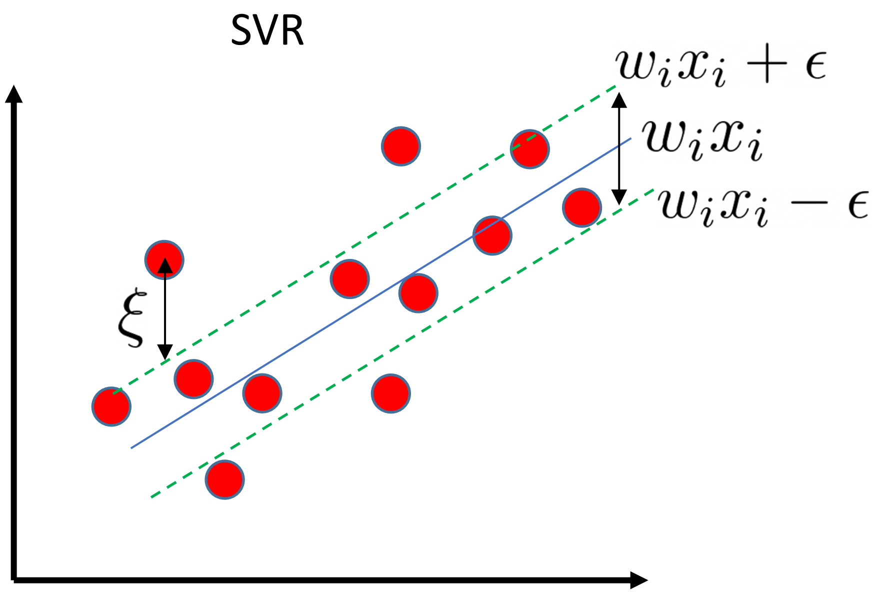
Find the optimal hyperplane in an N-dimensional space that distinctly classifies the data points.





Support Vector Machine for Regression





SVM Optimization

Hinge loss function

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

Loss function for the SVM

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

Updating the weights:

No misclassification

$$w = w - \alpha \cdot (2\lambda w)$$

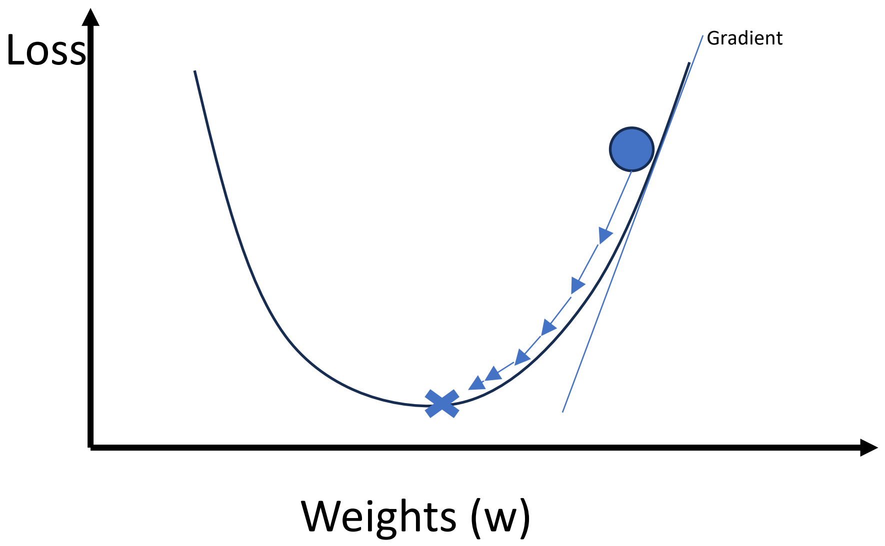
Misclassification

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

Gradients

$$\frac{\delta}{\delta w_k} \lambda \parallel w \parallel^2 \quad = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k} \big(1 - y_i \langle x_i, w \rangle\big)_+ \quad = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$



Intro to optimization

Review on Linear Regression

Task (T)

Input $x \in \mathbb{R}^n$
Weights $w \in \mathbb{R}^n$

$$\hat{y} = w^T x$$

$$f(x, w) = x_1 w_1 + x_2 w_2 + \cdots + x_n w_n$$

Performance (P)

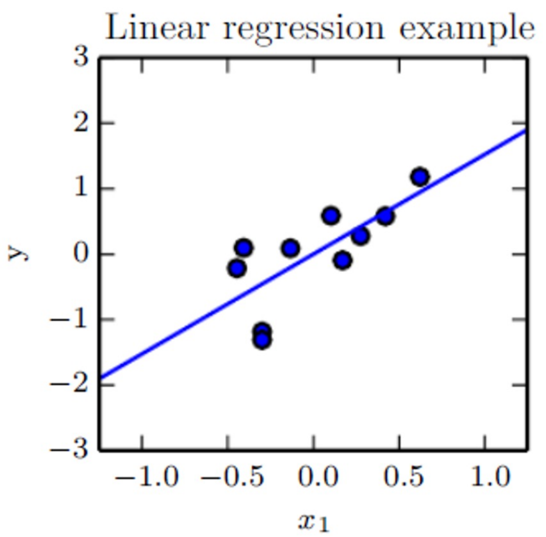
$$MSE_{test} = \frac{1}{m} \sum_i (\hat{y}_{test} - y_{test})_i^2$$

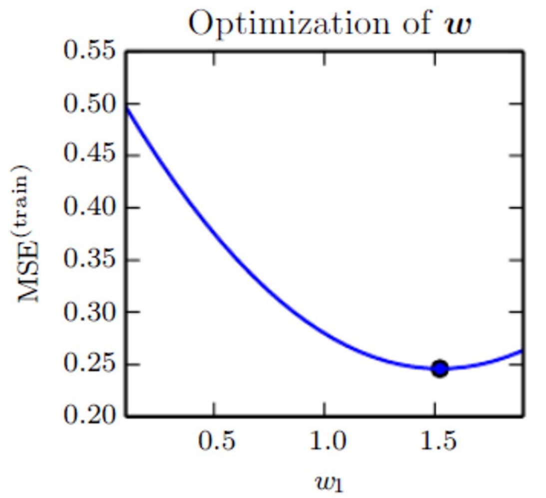
Dataset

$$(X, y) \begin{cases} (X_{train}, y_{train}) \\ (X_{test}, y_{test}) \end{cases}$$

Training

$$\nabla_w \left(\frac{1}{m} \sum_i (w^T X_{train} - y_{train})_i^2 \right) = 0$$





Solves linear problems

Can’t solve more complex problems (e.g., XOR problem)

Linear Regression Optimization

- Add an offset w_0: $f(\boldsymbol{x}; \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x} + \mathrm{w}_0$, $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots (\boldsymbol{x}_n, y_n)\}$

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{n} (\boldsymbol{w}^T \boldsymbol{x}_i + w_0 - y_i)^2$$

$$= \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}; \mathcal{D})$$

- Set $\frac{\partial L(\boldsymbol{w};\mathcal{D})}{\partial w_i} = 0$ for each i

Mean squared error loss

Rewrite:

$$
\begin{aligned}
(X\mathbf{w}-\mathbf{y})^T(X\mathbf{w}-\mathbf{y}) &= (\mathbf{w}^T X^T - \mathbf{y}^T)(X\mathbf{w}-\mathbf{y}) \\
&= \mathbf{w}^T X^T X\mathbf{w} - \mathbf{w}^T X^T \mathbf{y} - \mathbf{y}^T X\mathbf{w} + \mathbf{y}^T\mathbf{y} \\
&= \mathbf{w}^T X^T X\mathbf{w} - 2\mathbf{w}^T X^T \mathbf{y} + \mathbf{y}^T\mathbf{y}.
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial}{\partial w}\mathbf{w}^T X^T X\mathbf{w} - 2\mathbf{w}^T X^T \mathbf{y} + \mathbf{y}^T\mathbf{y} &= 0 \\
2X^T X\mathbf{w} - 2X^T\mathbf{y} &= 0 \\
X^T X\mathbf{w} &= X^T\mathbf{y} \\
\mathbf{w} &= (X^T X)^{-1} X^T\mathbf{y}
\end{aligned}
$$

Regularization

- Ridge regression: penalize with L2 norm

$$\boldsymbol{w}^* = \arg\min \sum_i L(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i) + \lambda \sum_{j=1}^{m} w_j^2$$

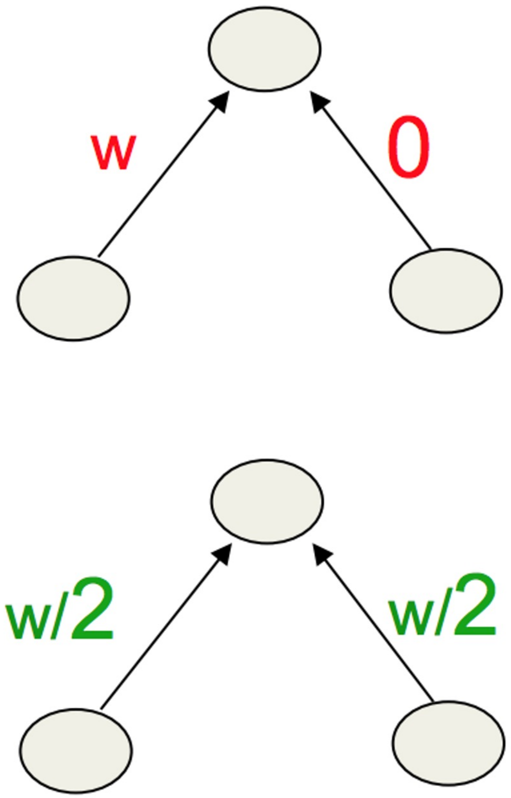
  - Closed form solution exists $\boldsymbol{w}^* = (\lambda I + X^T X)^{-1} X^T \boldsymbol{y}$

- LASSO regression: penalize with L1 norm

$$\boldsymbol{w}^* = \arg\min \sum_i L(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i) + \lambda \sum_{j=1}^{m} |w_j|$$

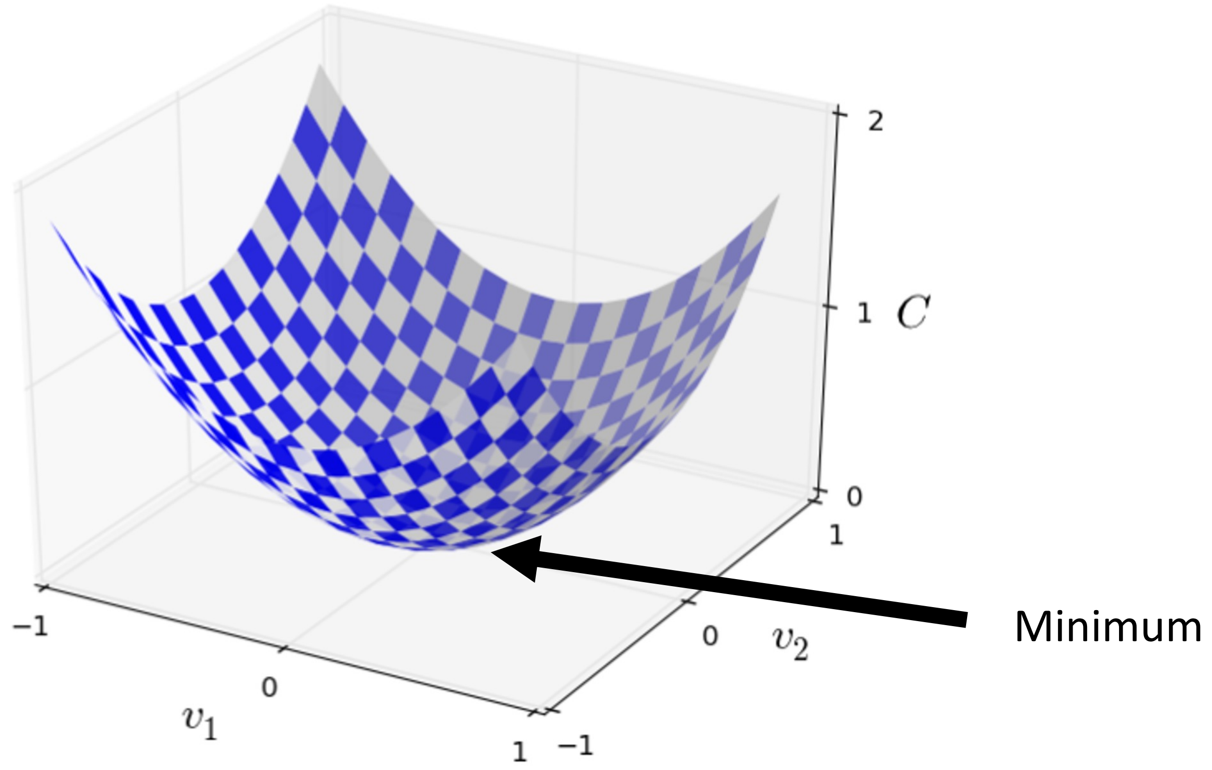
  - No closed form solution but still convex (optimal solution can be found)

Regularization



- Prefers to share smaller weights
- Makes model smoother
- More Convex

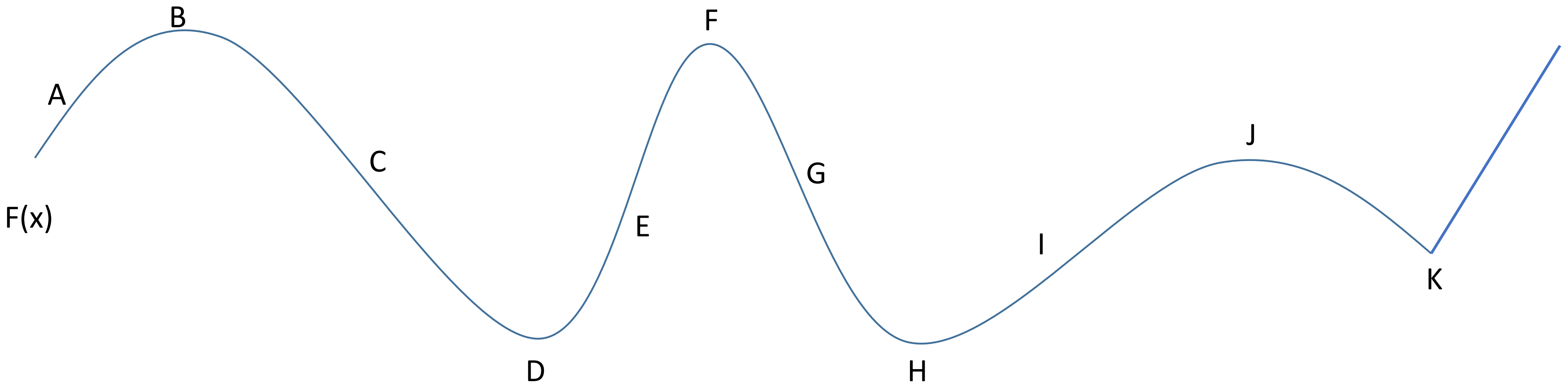
Loss Minimization



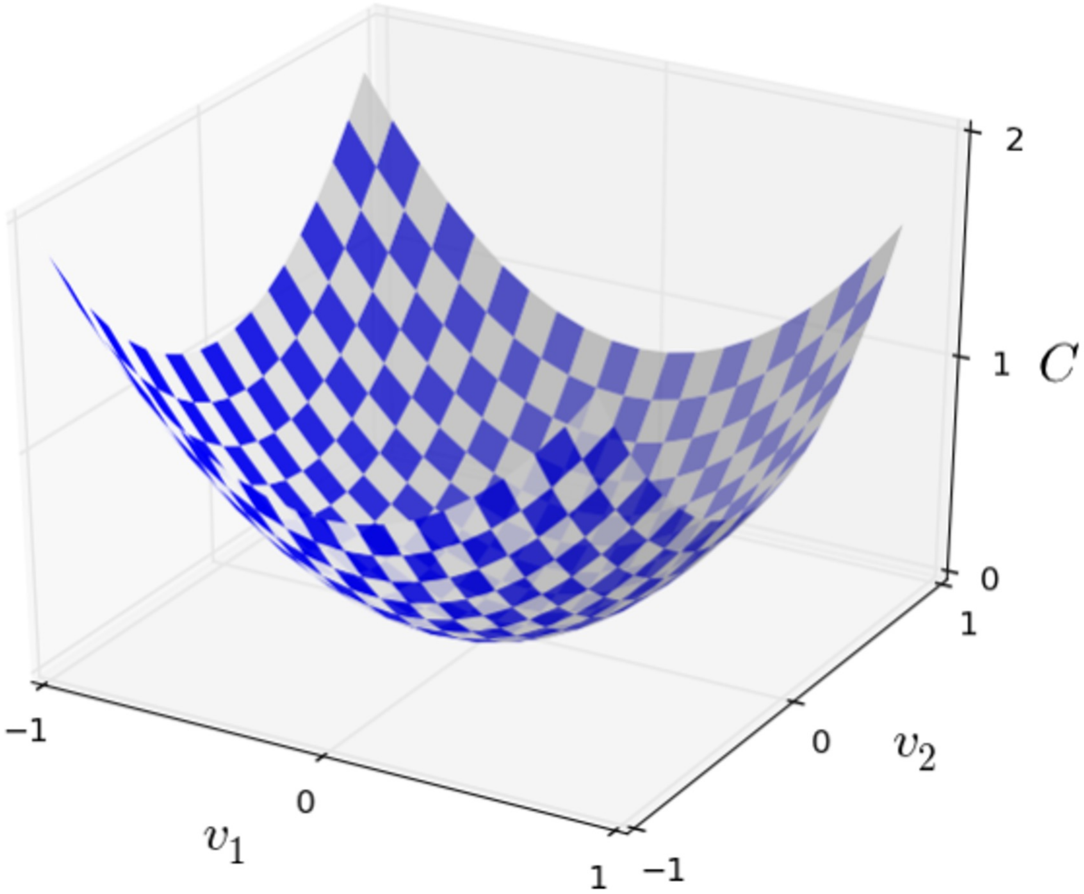
Convex loss functions can be solved by differentiation, at the point where Loss is minimum the derivative wrt to parameters should be 0!

More on the derivatives

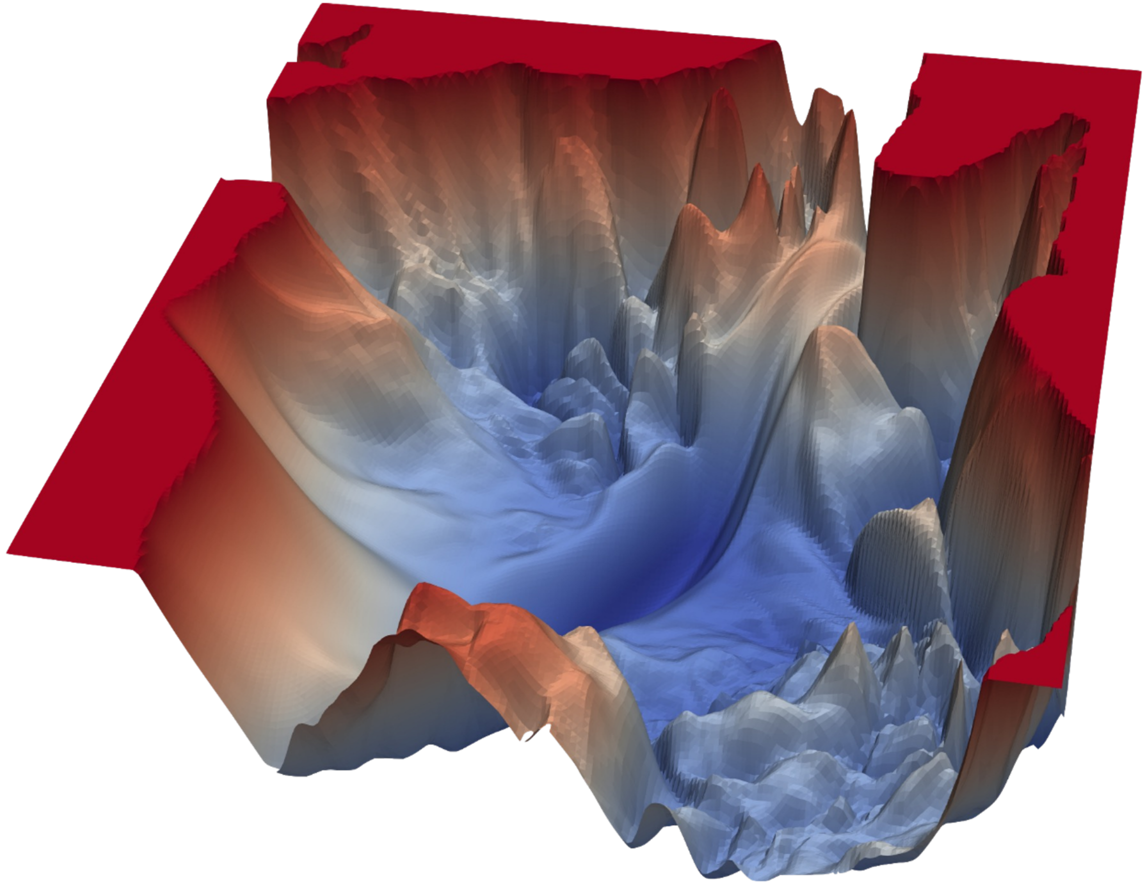
https://tinyurl.com/GeoComp2024



Expectation

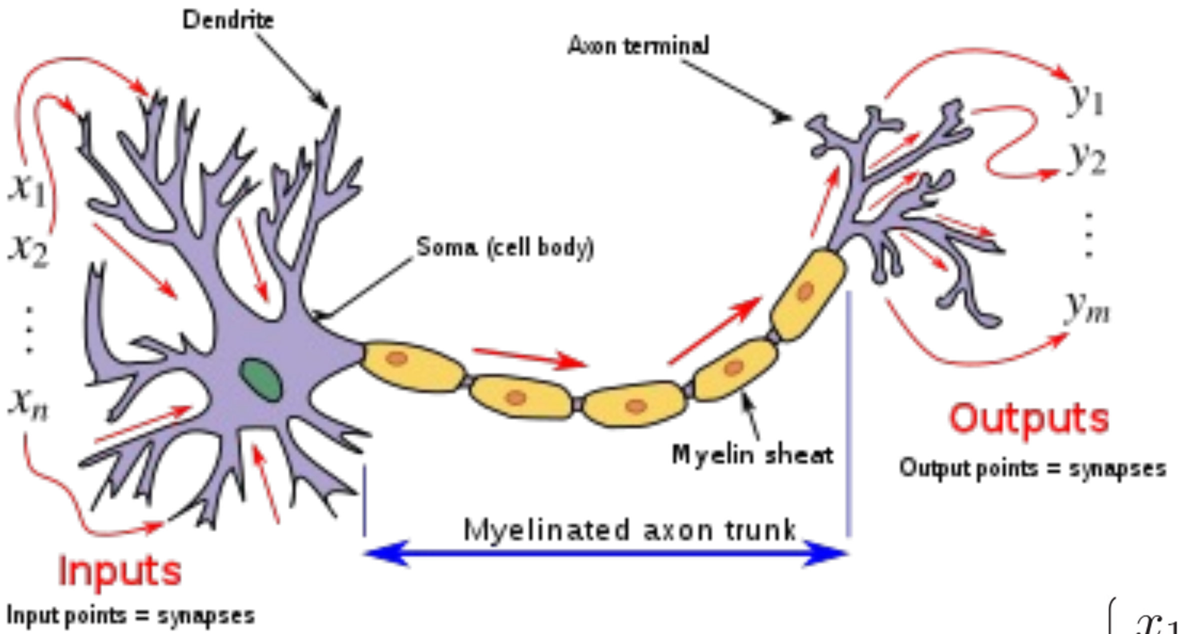


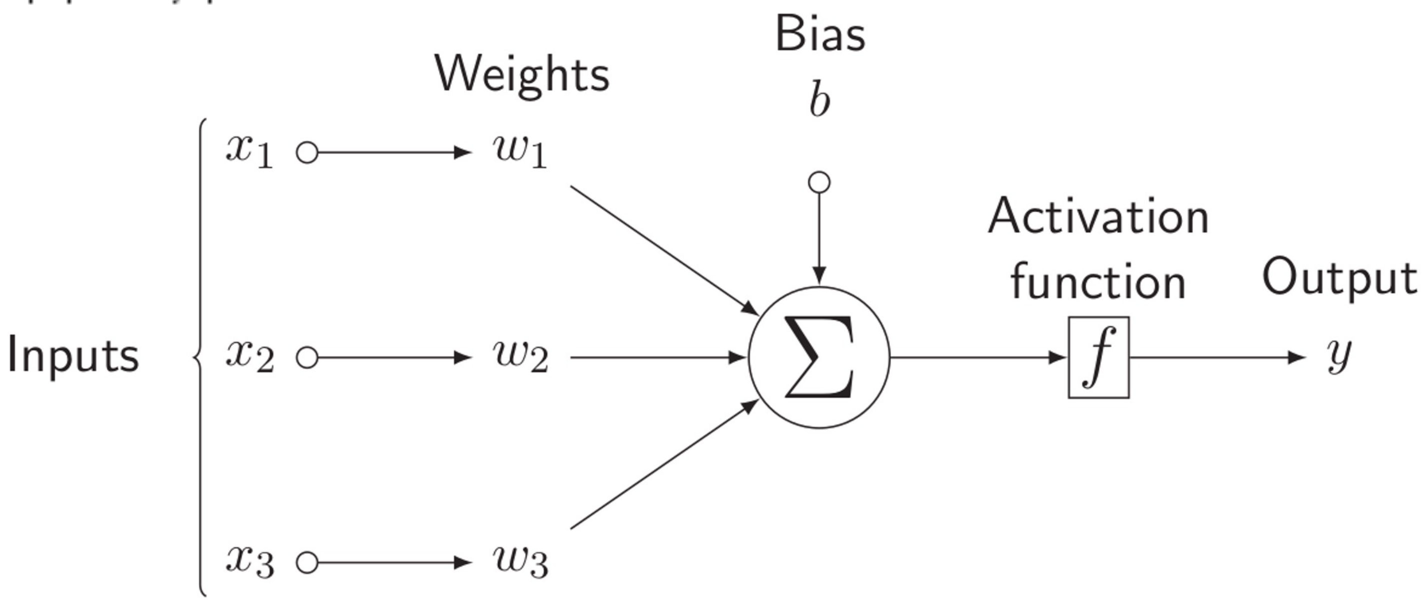
Reality



Perceptron

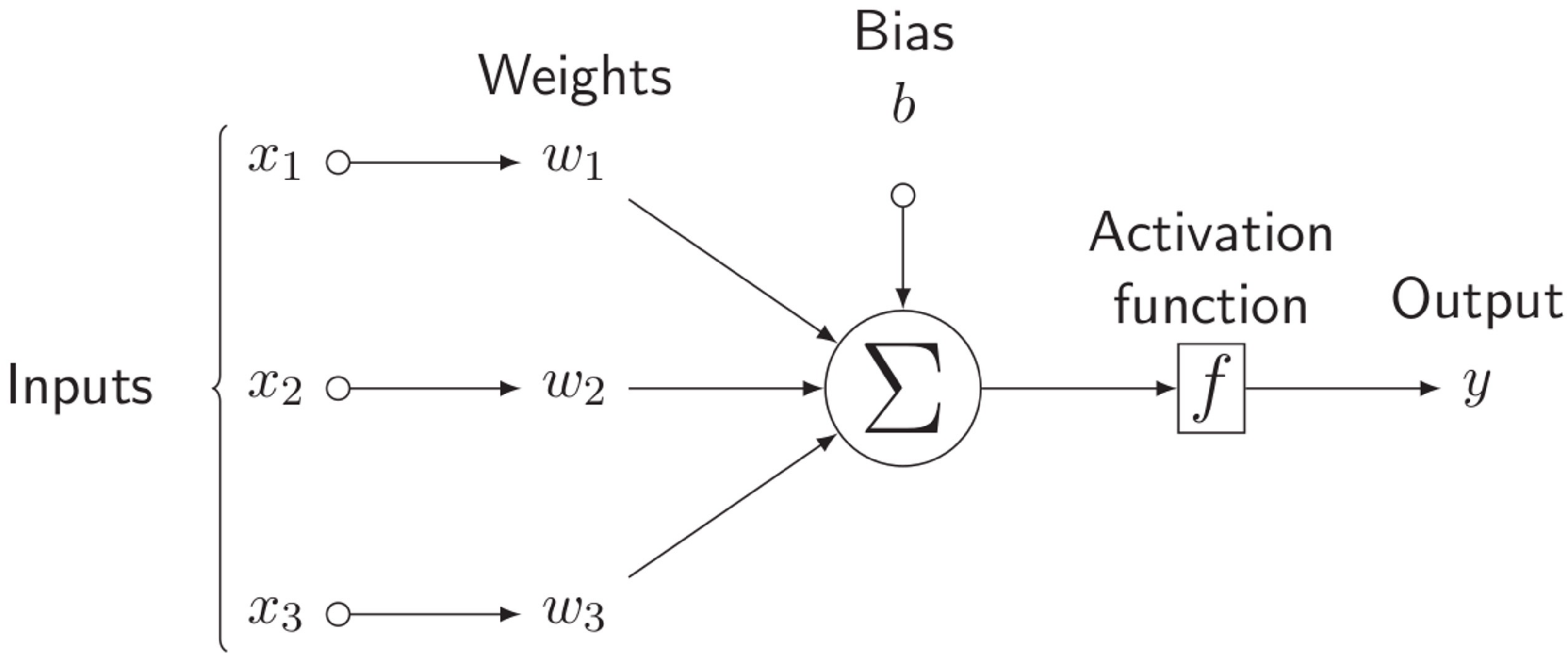
Perceptron: Threshold Logic





Perceptron: Threshold Logic

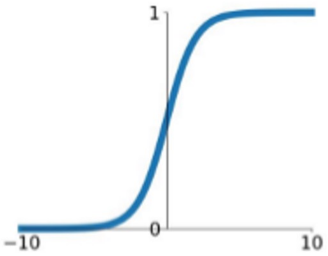
$$\mathcal{L}_{\text{perc}}(\mathbf{x}, y) = \begin{cases} 0 & \text{if } y\mathbf{w}^\top \mathbf{f}(\mathbf{x}) > 0 \\ -y\mathbf{w}^\top \mathbf{f}(\mathbf{x}) & \text{if } y\mathbf{w}^\top \mathbf{f}(\mathbf{x}) \leq 0 \end{cases}$$



Activation functions

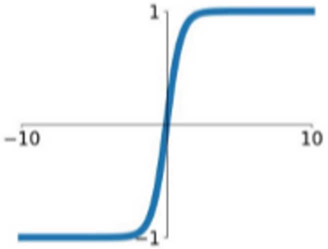
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



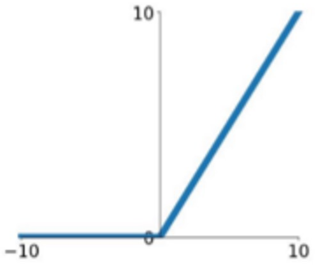
tanh

$\tanh(x)$



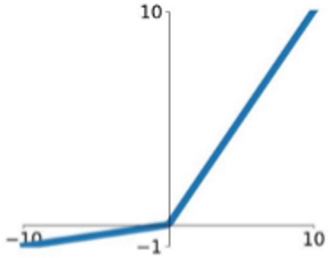
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

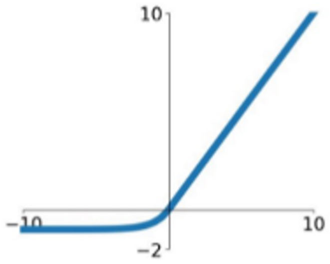


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Optimizers (pt1)

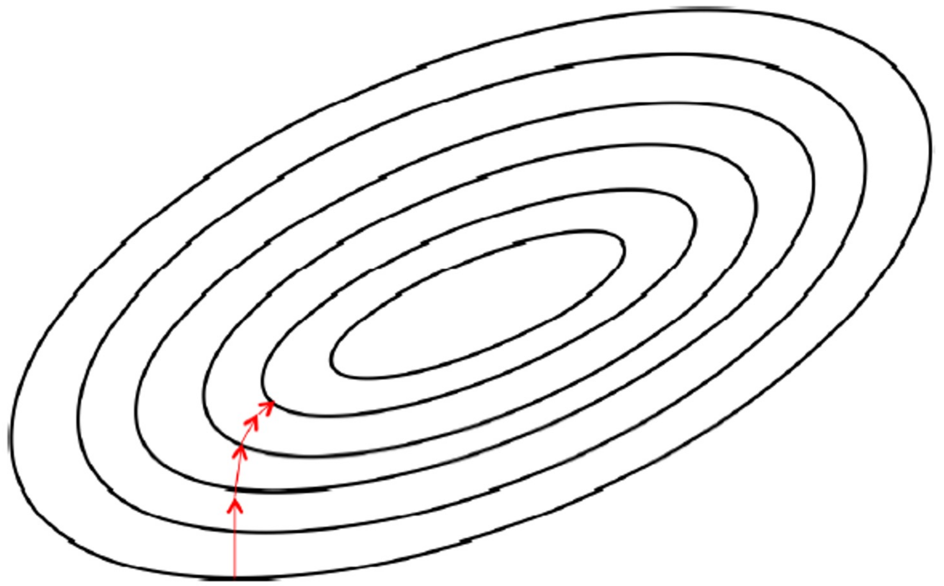
Gradient

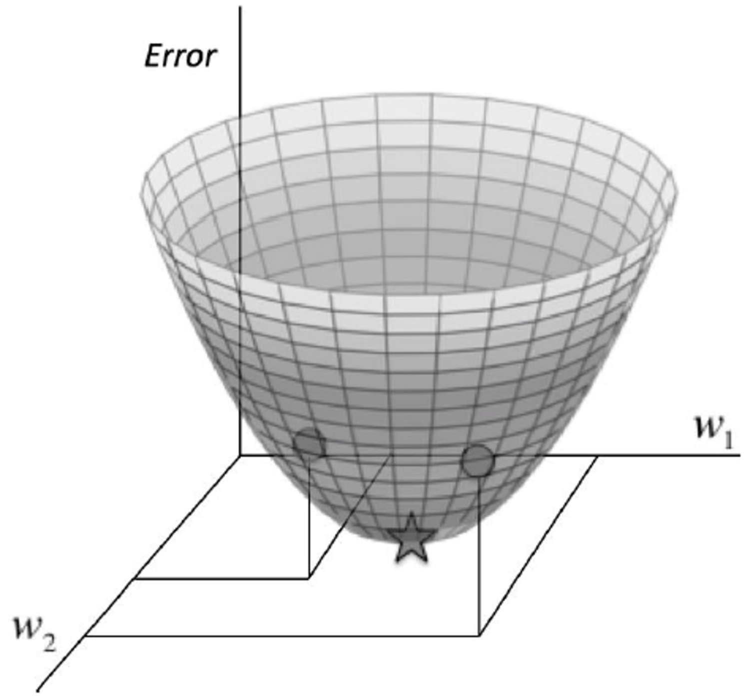
$$\Delta w_k = -\frac{\partial E}{\partial w_k}$$

$$= -\frac{\partial}{\partial w_k}\left(\frac{1}{m}\sum_i (w^T X_i - y_i)_i^2\right)$$

$$w_{i+1} = w_i + \Delta w_k$$

Stochastic gradient descent (**SGD**)





Optimizers (pt1)

Practical test:
lr_val = [1; 0.1; 0.01]
momentum_val = 0
nesterov_val = ‘False'
decay_val = 1e-6

Hyperparameters

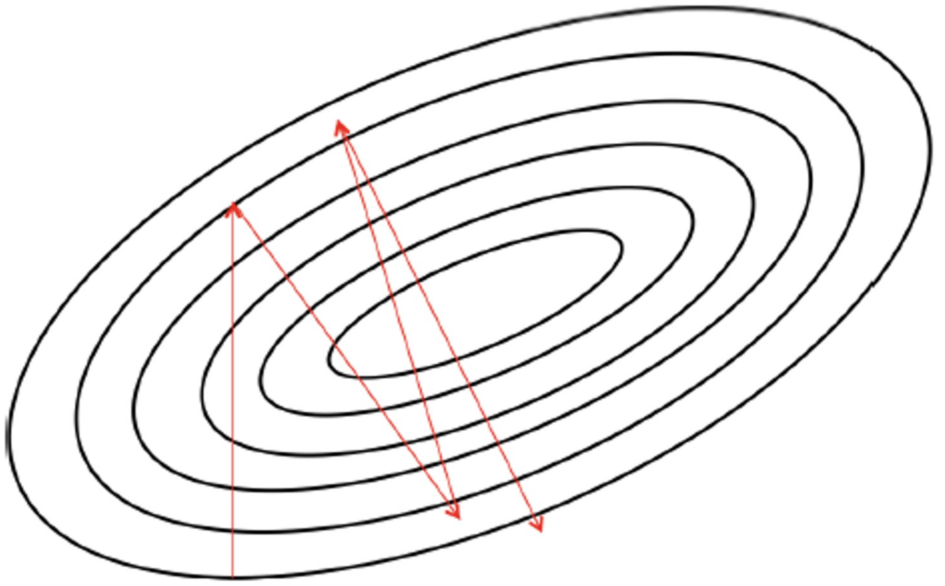
- Learning rate (α)

$$\Delta w_k = -\alpha \frac{\partial E}{\partial w_k}$$

$$= -\alpha \frac{\partial}{\partial w_k}\left(\frac{1}{m}\sum_i (w^T X_i - y_i)_i^2\right)$$

$$w_{i+1} = w_i + \Delta w_k$$

Stochastic gradient descent (**SGD**)



Result of a large learning rate α

Optimizers

Watch out for local minimal areas

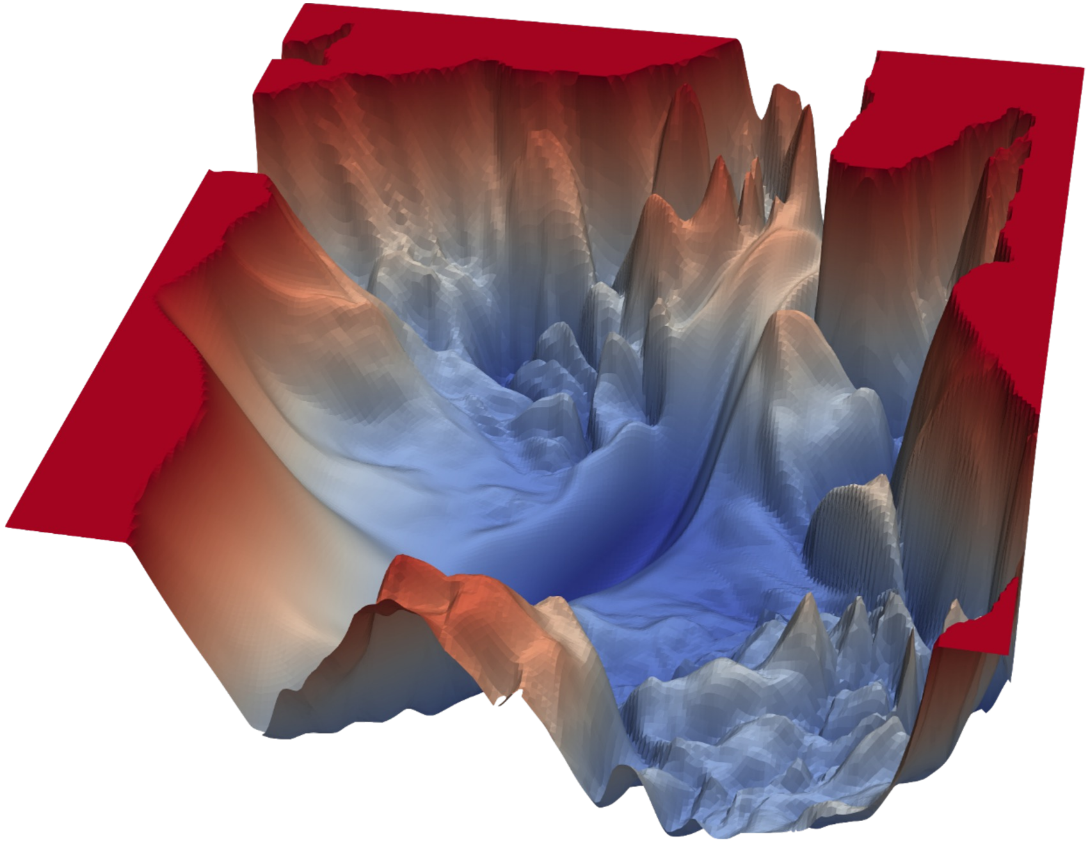
Hyperparameters

- Learning rate (α)

$$\Delta w_k = -\alpha \frac{\partial E}{\partial w_k}$$

$$= -\alpha \frac{\partial}{\partial w_k}\left(\frac{1}{m}\sum_i (w^T X_i - y_i)_i^2\right)$$

$$w_{i+1} = w_i + \Delta w_k$$



Stochastic gradient descent (**SGD**)

Gradient Descent

- Gradient descent refers to taking a step in the direction of the ***gradient (partial derivative)*** of a weight or bias with respect to the loss function
- Gradients are propagated backwards through the network in a process known as ***backpropagation***
- The size of the step taken in the direction of the gradient is called the ***learning rate***

Time for a quiz and tutorial!

https://tinyurl.com/GeoComp2024

Optimizers

Optimizers

Hyperparameters

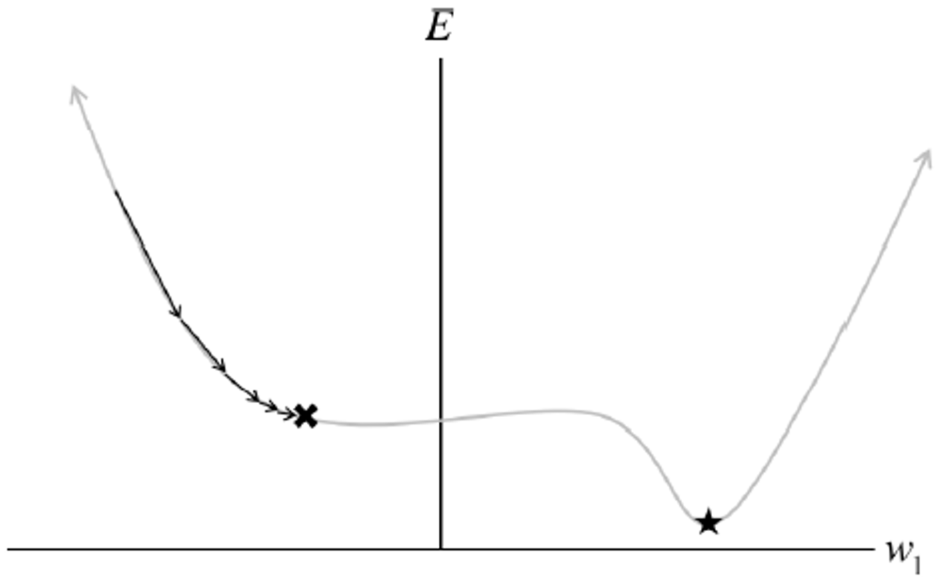
- Learning rate (α)

$$\Delta w_k = -\alpha \frac{\partial E}{\partial w_k}$$

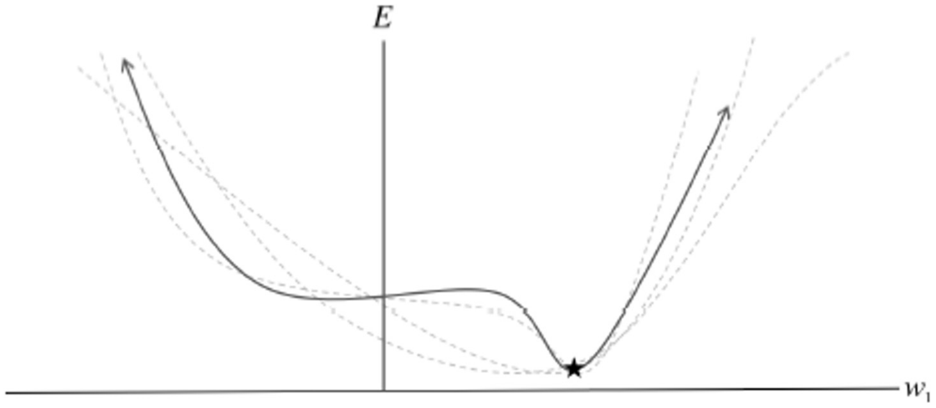
$$= -\alpha \frac{\partial}{\partial w_k}\left(\frac{1}{m}\sum_i (w^T X_i - y_i)_i^2\right)$$

$$w_{i+1} = w_i + \Delta w_k$$

Stochastic gradient descent (**SGD**)



Local Minima



Multiple samples

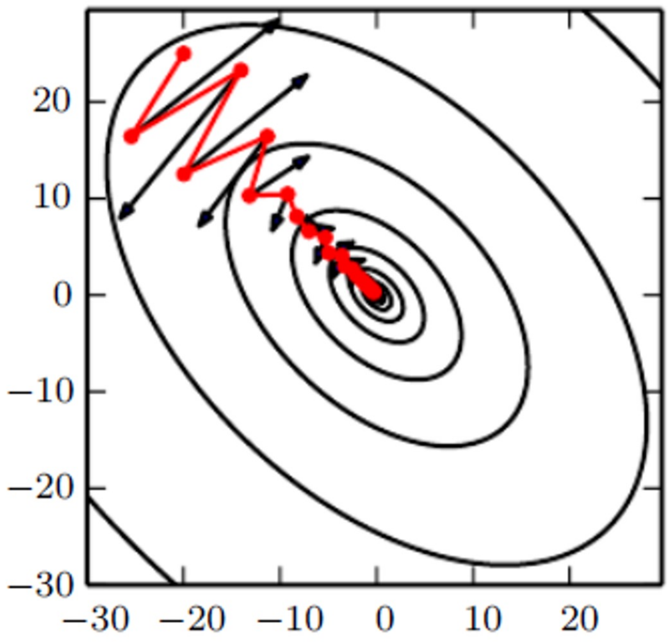
Optimizers

Hyperparameters

- Learning rate (α)
- Momentum (β)

$$v_{i+1} = v\beta - \alpha \frac{\partial}{\partial w_k}\left(\frac{1}{m}\sum_i (w^T X_i - y_i)_i^2\right)$$

$$w_{i+1} = w_i + v$$



SGD

SGD+Momentum

Stochastic gradient descent with momentum (**SGD+Momentum**)

Optimizers

Hard to pick right hyperparameters

- Small learning rate: long convergence time
- Large learning rate: convergence problems

Adagrad: adapts learning rate to each parameter

$$\Delta w_{k,t} = -\alpha \frac{\partial E_t}{\partial w_{k,t}} = -\alpha \nabla_w E(w_t)$$

- Learning rate might decrease too fast
- Might not converge

$$g_{t,i} = \nabla_w E(w_{t,i})$$

$$G_{t+1,i} = G_{t,i} + g_{t,i} \odot g_{t,i}$$

$$w_{t+1,i} = w_{t,i} - \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} g_{t,i}$$

Optimizers

RMSprop: decaying average of the past squared gradients

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

Exponentially decaying average

Adadelta

$$E[\Delta_w^2]_t = \gamma E[\Delta_w^2]_{t-1} + (1-\gamma)\Delta_w^2$$

$$\Delta w_t = \frac{\sqrt{E[\Delta_w^2]_t + \epsilon}}{\sqrt{G_{t,i} + \epsilon}} g_t$$

$$\Delta w_{k,t} = -\alpha \frac{\partial E_t}{\partial w_{k,t}} = -\alpha \nabla_w E(w_t) = -\alpha g_{t,i}$$

$$g_{t,i} = \nabla_w E(w_{t,i})$$

$$G_{t+1,i} = \gamma G_{t,i} + (1-\gamma) g_{t,i} \odot g_{t,i}$$

$$\Longrightarrow$$

$$w_{t+1,i} = w_{t,i} - \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} g_{t,i}$$

Optimizers

ADAM: decaying average of the past squared gradients and momentum

RMSprop / Adadelta

$$g_{t,i} = \nabla_w E(w_{t,i})$$

$$G_{t+1,i} = \gamma G_{t,i} + (1-\gamma) g_{t,i} \odot g_{t,i}$$

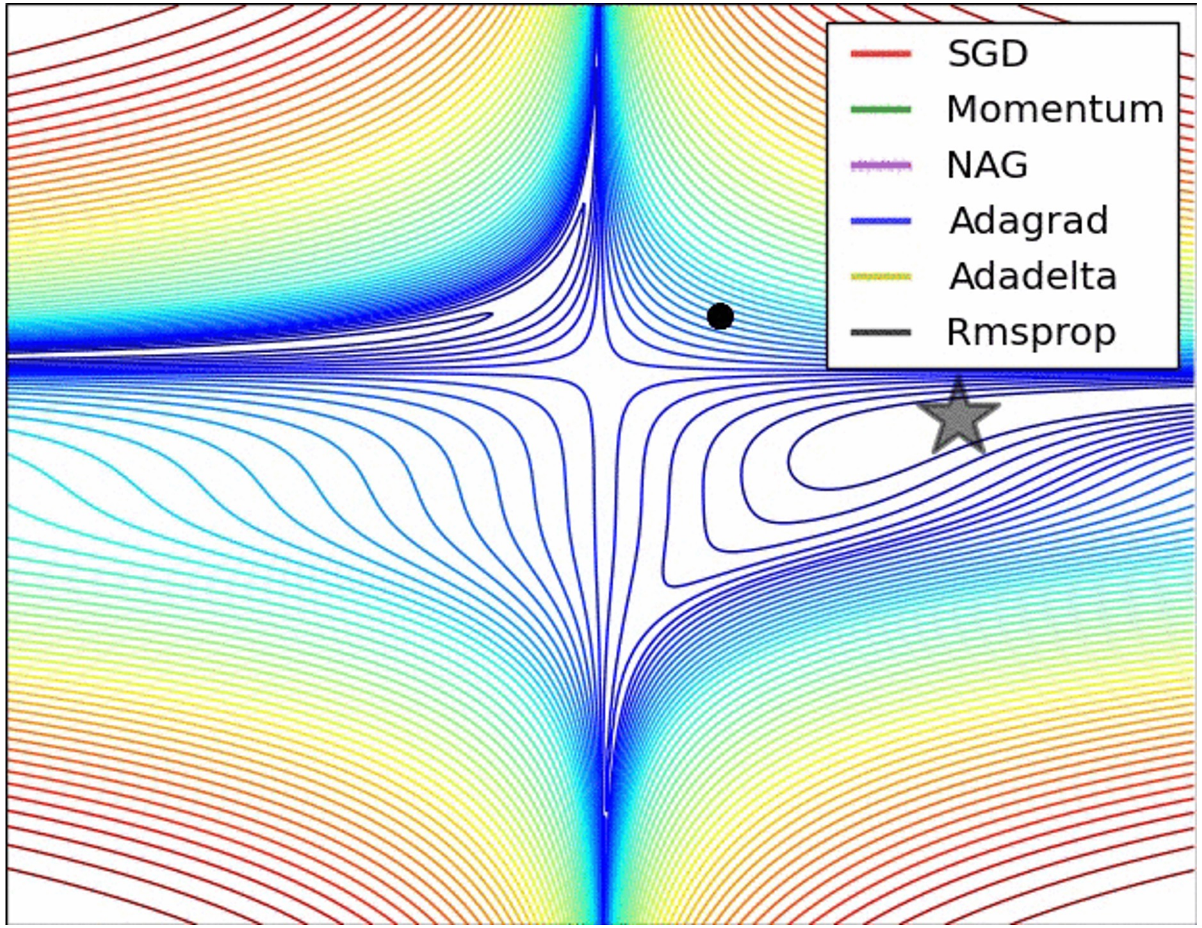
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

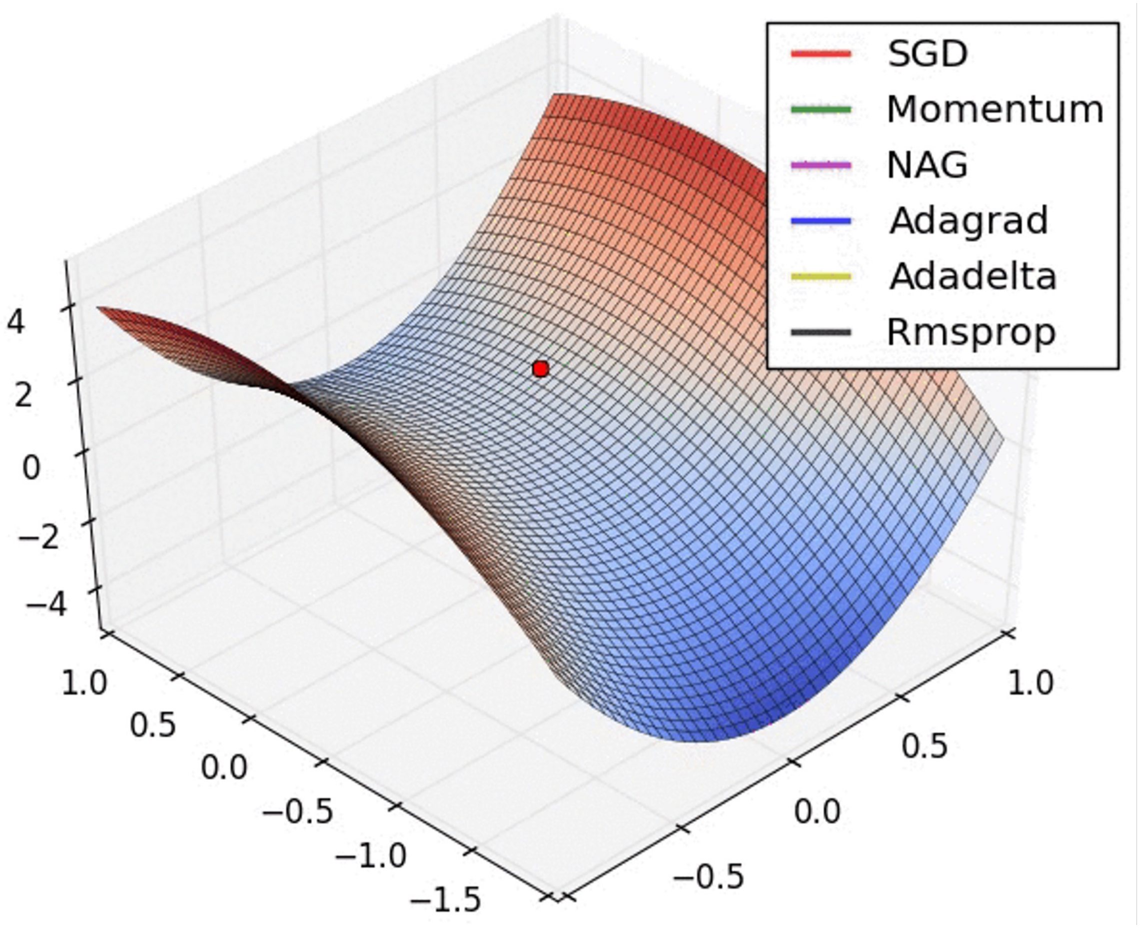
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$w_{t+1,i} = w_{t,i} - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$





Which optimizer is the best?