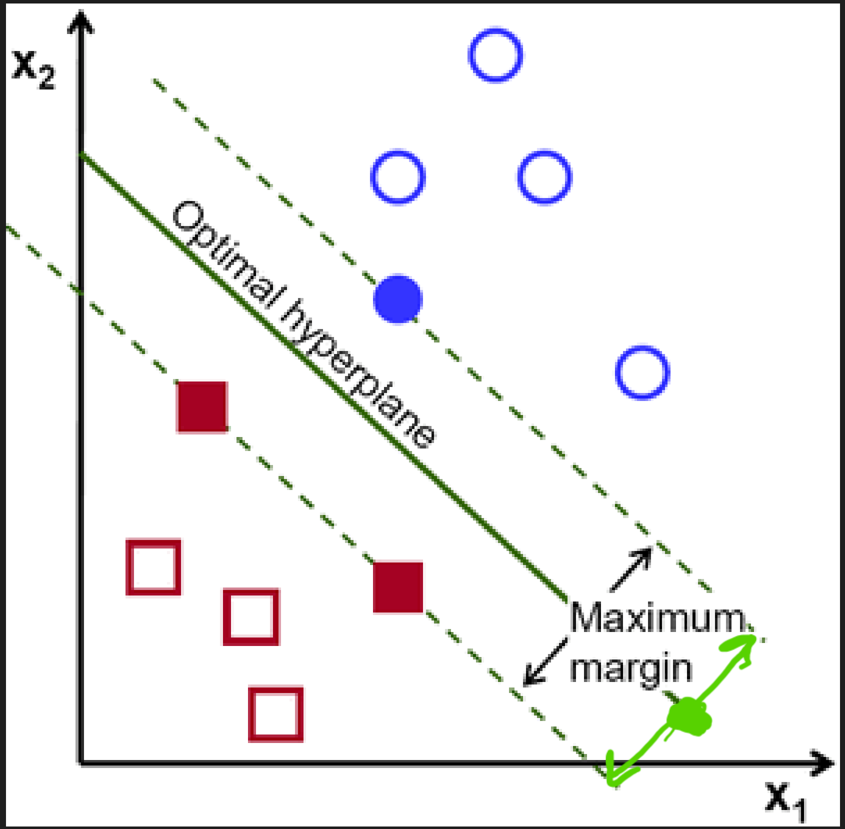
INTRODUCTION TO MACHINE LEARNING COMPSCI 4ML3

LECTURE 16

HASSAN ASHTIANI

SUPPORT VECTOR MACHINES

- Pick the linear separator that maximizes the "margin"
- More robust to "perturbation"
- Less overfitting problem!
  - Works well for high-dimensional data (?)
  - More on that later!



DISTANCE OF A POINT TO A HYPERPLANE

The Euclidean distance between a point x and the hyperplane parametrized by W is (why?)

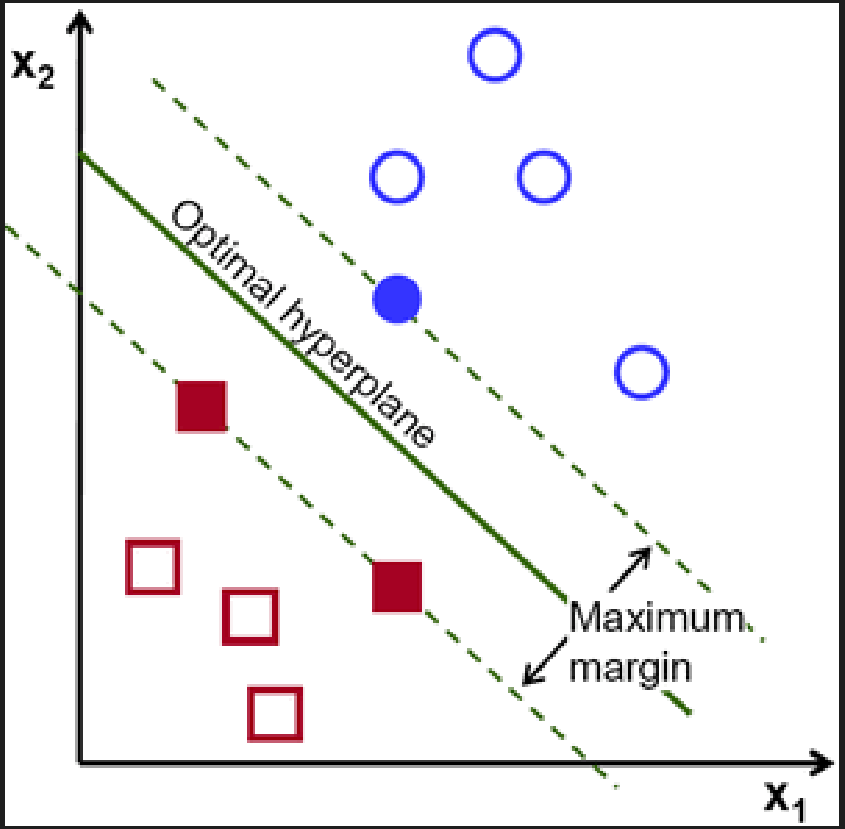
$$\frac{|W^T x + b|}{||W||_2}$$

- Only the direction of W matters (not $||W||_2$)
- Assume $||W||_2$=1, then the distance is

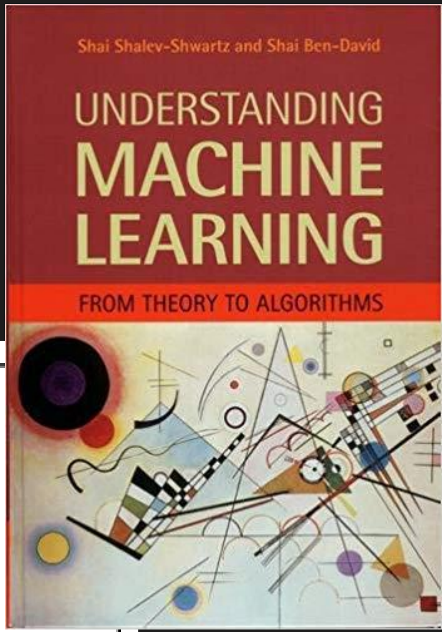
$$|W^T x + b|$$

MAXIMUM MARGIN HYPERPLANE

- LET THE HYPERPLANE BE PARAMETRIZED BY W AND b
- ASSUME $\|W\|_2 = 1$
- (W, b) HAS A γ MARGIN IF
  - $W^T x + b > \gamma$ FOR EVERY BLUE x, AND
  - $W^T x + b < -\gamma$ FOR EVERY RED x



THE VERSION WITH "BIAS"



Hard-SVM

$w' = \frac{w}{\|w\|}$ $\quad \|w'\| = 1$

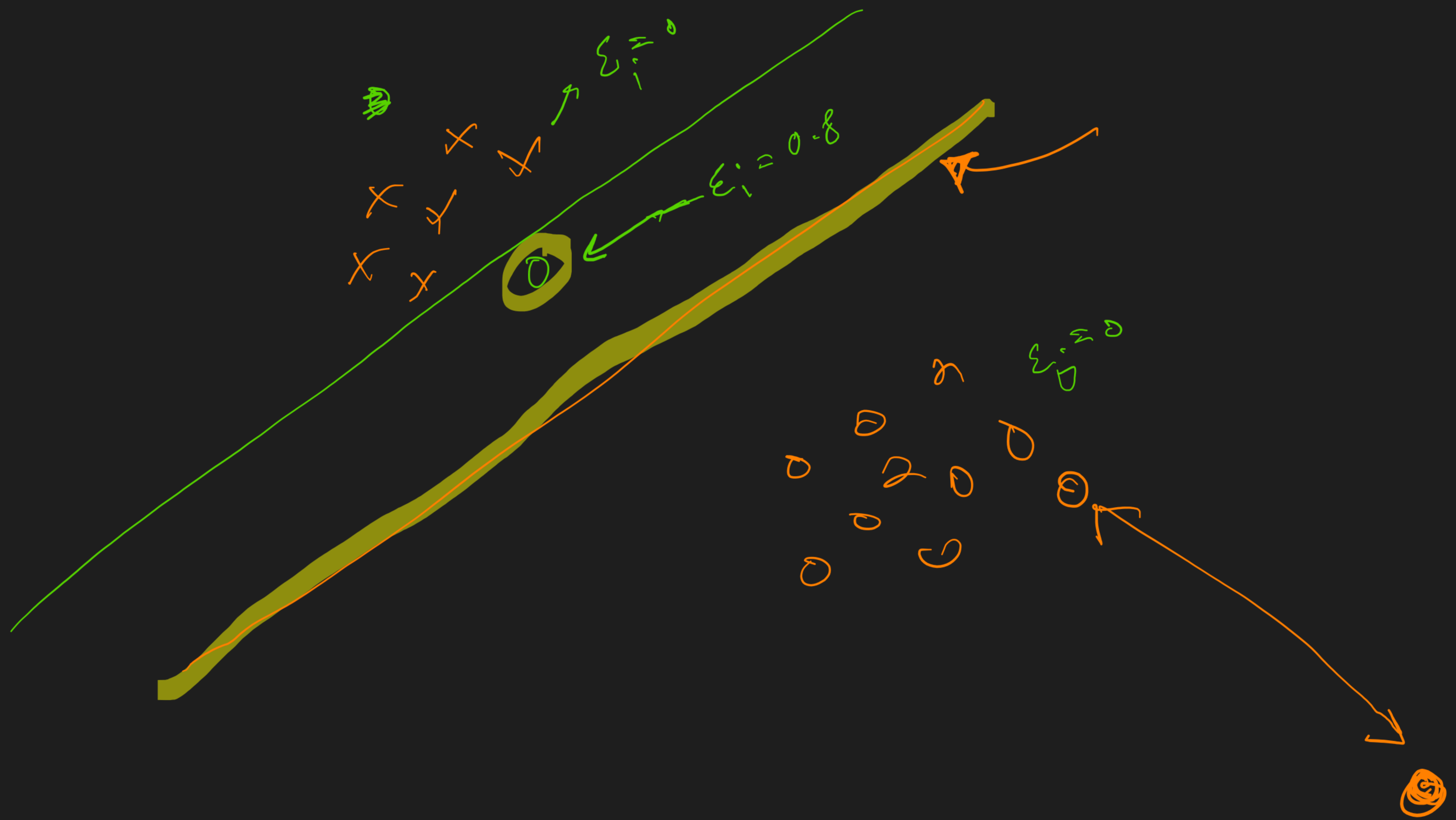
input: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$

solve:

$$(\mathbf{w}_0, b_0) = \underset{(\mathbf{w},b)}{\operatorname{argmin}} \|\mathbf{w}\|^2 \text{ s.t. } \forall i, \; y_i(\langle \underset{\gamma}{\mathbf{w}}, \mathbf{x}_i \rangle + b) \geq \underset{\gamma}{1} \qquad (15.2)$$

output: $\hat{\mathbf{w}} = \frac{\mathbf{w}_0}{\|\mathbf{w}_0\|}, \quad \hat{b} = \frac{b_0}{\|\mathbf{w}_0\|}$

- SENSITIVE TO OUTLIERS…


ξi = 0
ξi = 0.8
ξi = 0

SOFT-MARGIN SVM

Soft-SVM

input: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
parameter: $\lambda > 0$
solve:

$$\min_{\mathbf{w}, b, \boldsymbol{\xi}} \left(\lambda \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right) \tag{15.4}$$

$$\text{s.t. } \forall i, \quad y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \ \text{ and } \ \xi_i \geq 0$$

output: $\mathbf{w}, b$

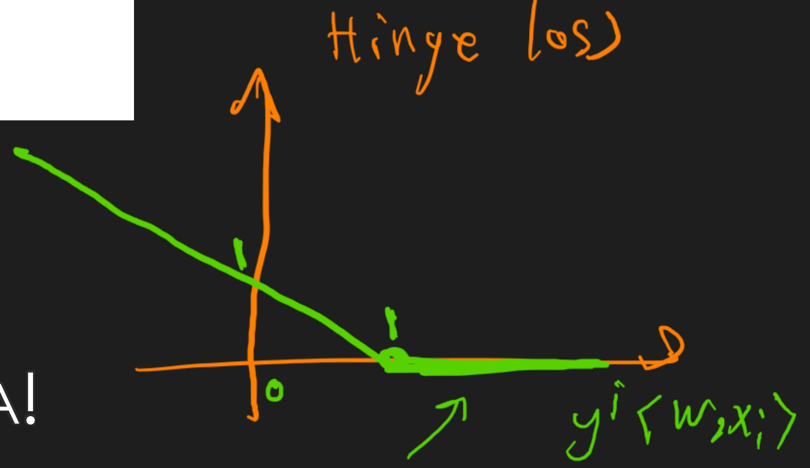
EQUIVALENT FORM OF SOFT-MARGIN SVM

$$\min_{\mathbf{w}} \left(\lambda \|\mathbf{w}\|^2 + L_S^{\text{hinge}}(\mathbf{w}) \right),$$

Hinge loss

$$L_S^{\text{hinge}}(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - \overbrace{y\langle \mathbf{w}, \mathbf{x}_i \rangle}^{\text{margin for data point } i}\}.$$

- No constraints!
- Regularization!
    - SVMs are good for high-dimensional data!



EXERCISE

- Prove that these two forms of Soft-SVM are Equivalent

$$\xi_i = \max(0, 1 - y^i < w, x^i >)$$

Soft-SVM

input: $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
parameter: $\lambda > 0$
solve:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \left(\lambda \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \right) \quad (15.4)$$
$$\text{s.t. } \forall i, \quad y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0$$

output: $\mathbf{w}, b$

$$\min_{\mathbf{w}} \left(\lambda \|\mathbf{w}\|^2 + L_S^{\text{hinge}}(\mathbf{w}) \right),$$

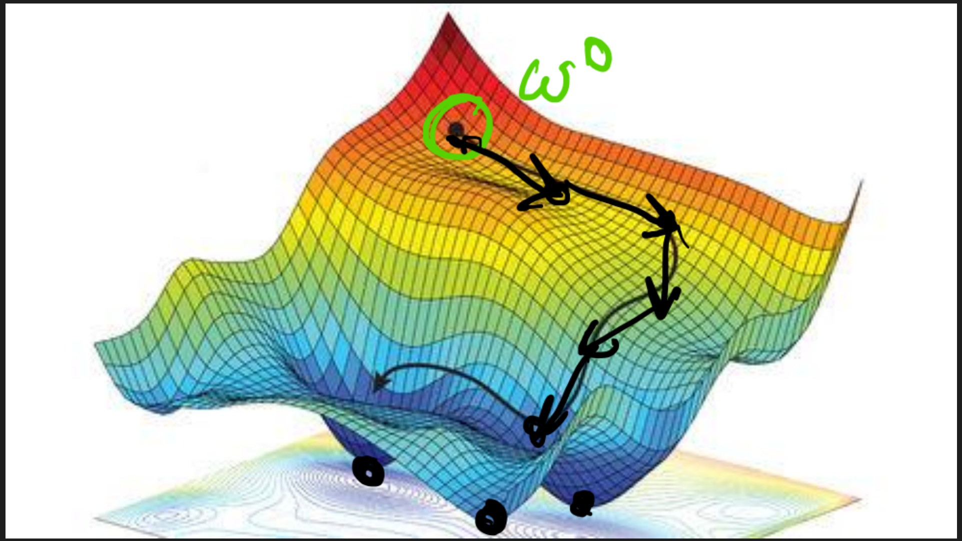
$$L_S^{\text{hinge}}(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x}_i \rangle\}.$$

OPTIMIZATION

- No constraints!
- We can use special-purpose SVM solvers
    - …Or we can just use "gradient descent"!

$$\min_{\mathbf{w}} \left(\lambda\|\mathbf{w}\|^2 + L_S^{\text{hinge}}(\mathbf{w})\right),$$

$$L_S^{\text{hinge}}(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x}_i\rangle\}.$$



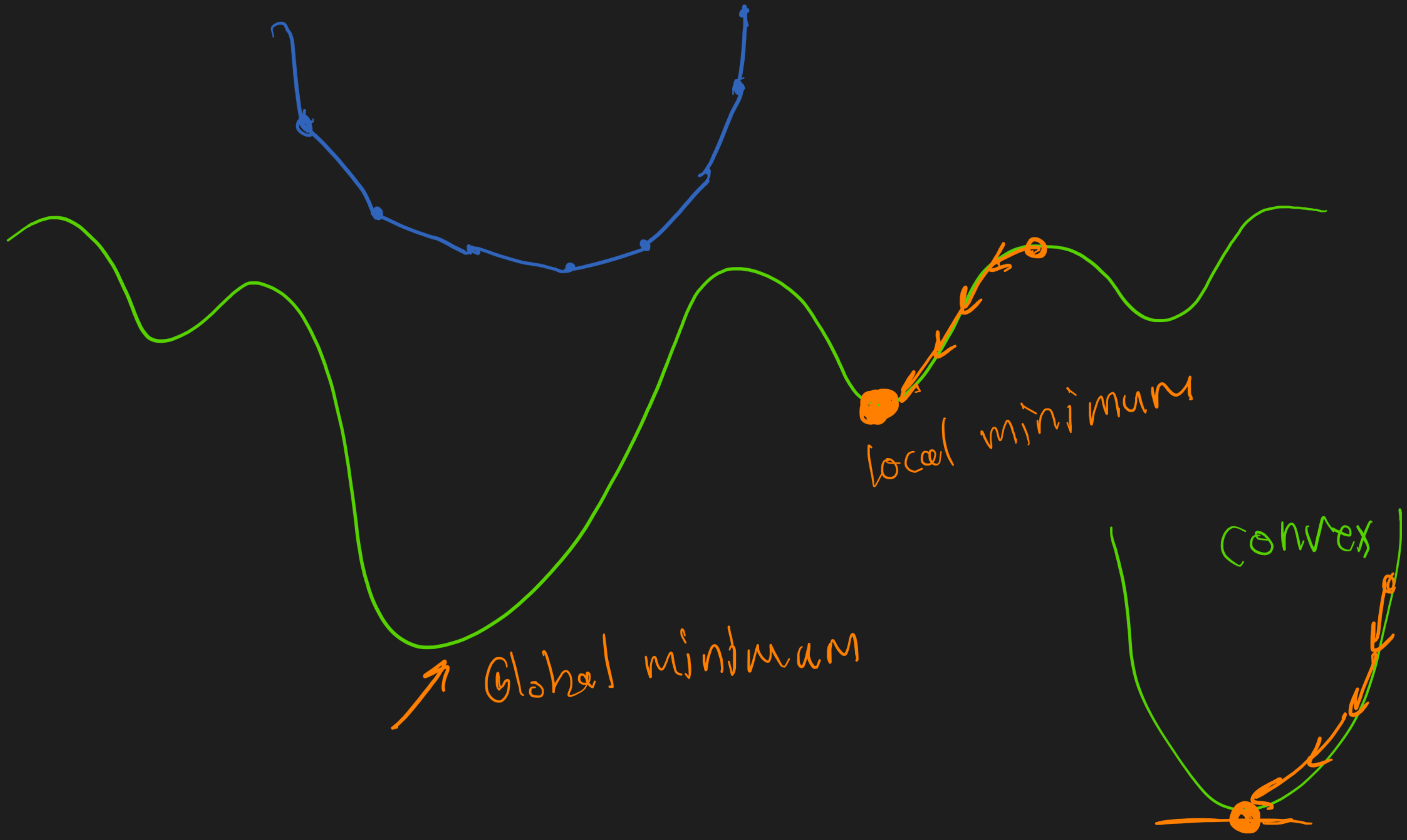
https://www.mit.edu/~amini/projects.html

GRADIENT DESCENT

- Minimize objective function $\mathrm{E}(w)$ with single real-valued output?
- Move in the direction of steepest descent
  - Gradient of a function at a point is the direction of steepest ascent at that point.
- Gradient

$$\nabla_w \left[E(w) \right] = \left[\frac{\partial E(w)}{\partial w_1}, \ldots, \frac{\partial E(w)}{\partial w_d} \right]^T$$

- Gradient Decent
  - Initialize w^0
  - For $t = 1, \ldots$
    - $w^{t+1} = w^t - \alpha \nabla_{\mathrm{w}}(\mathrm{E}(w))$
    - Terminate after a few iterations (or until convergence)
- α : Step size or learning rate
  - An important parameter to set…can depend on t (α_t)


local minimum
Global minimum
convex

Gradient Descent: Guarantees

- Converges to a local minimum/saddle point
  - With appropriate learning rate (can depend on t)
- Not necessarily global minimum
- Guaranteed global minimum for convex functions
  - Soft SVM, LS, logistic regression (?)...
- May get stuck in local minimum for non-convex
  - Widely used in neural networks (more about this later)

CALCULATING THE GRADIENT

- $E(w) = L(w) + \lambda. Reg(w)$
  - E.G., FOR SOFT SVM WE HAVE $Reg(w) = \|w\|_2^2$
  - IT IS OFTEN EASY TO TAKE THE DERIVATIVE OF THE REGULARIZATION TERM
  - WHAT ABOUT THE LOSS TERM?
- $L(w) = \sum_i l(f_w(x^i), y^i)$
  - $\hat{y} = f_w(x)$ IS THE PREDICTED VALUE
  - $l(\hat{y}, y)$ IS THE LOSS FUNCTION
- $\nabla_w L(w) = \nabla_w \left(\sum_i l(f_w(x^i), y^i)\right) = \sum_i \nabla_w \left(l(f_w(x^i), y^i)\right)$
  - COMPUTE THE GRADIENT FOR EACH DATA POINT AND THEN SUM THEM UP
  - A BIT TOO SLOW…. MANY ITERATIONS

CALCULATING THE GRADIENT

- $\nabla_{\mathrm{w}}\big(L(w)\big) = \sum_i \nabla_{\mathrm{w}}\Big(l\big(f_w(x^i), y^i\big)\Big)$
- DIVIDE THE TRAINING DATA INTO A NUMBER OF "BATCHES"
- $S = S_1 \cup S_2 \cup \cdots \cup S_m$
- $\nabla_{\mathrm{w}}^{j} = \sum_{(x,y)\in S_j} \nabla_{\mathrm{w}}\big(l(f_w(x), y)\big)$
- $\nabla_{\mathrm{w}}\big(\mathrm{E}(w)\big) = \sum_j \nabla_{\mathrm{w}}^{j}$

MINI-BATCH GRADIENT DESCENT

- $\nabla_{\mathrm{W}}\big(\mathrm{L}(w)\big) = \sum_j \nabla_{\mathrm{W}}^j$
- STANDARD GRADIENT DESCENT: $w^{t+1} = w^t - \alpha\left(\sum_j \nabla_{\mathrm{W}}^j\right)$
- MINI-BATCH GRADIENT DESCENT:
  - $For\ j = 1\ to\ m$
    - $w = w - \alpha\left(\nabla_{\mathrm{W}}^j\right)$
- BENEFITS?
  - MEMORY
  - PARALLELIZATION
- POTENTIAL DISADVANTAGES
  - CAN BE LESS STABLE

Epoch: one time going over the whole dataset

STOCHASTIC GRADIENT DESCENT

Algorithm 8.1 Stochastic gradient descent (SGD) update

Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
    $k \leftarrow k + 1$
  end while

- Instead of dividing the data into batches, randomly select a subset as a batch
- Another benefit: adds a kind of randomness

WHAT IS MISSING?

- $\nabla_{\mathrm{w}}\big(\mathrm{E}(w)\big) = \sum_i \nabla_{\mathrm{w}}\left(l\big(f_w(x^i), y^i\big)\right) + \nabla_{\mathrm{w}} \|w\|_2^2$
    - How to calculate $\nabla_{\mathrm{w}}\left(l\big(f_w(x^i), y^i\big)\right)$?
    - How to calculate $\nabla_{\mathrm{w}} \|w\|_2^2$?
- Will get back to this later but for now…

- "Automatic differentiation"

AUTOMATIC DIFFERENTIATION IN PYTORCH

```python
# Define a linear SVM model
class LinearSVM(nn.Module):
    def __init__(self, input_size, num_classes):
        super(LinearSVM, self).__init__()
        self.fc = nn.Linear(input_size, num_classes, bias=True)

    def forward(self, x):
        # Flatten the image
        x= x.view(-1, input_size)
        return self.fc(x)
```

$\hat{f}_w(x) = w^T x$

```python
# Initialize the model, loss function, and optimizer
model = LinearSVM(input_size, num_classes)
criterion = nn.MultiMarginLoss()  # A Multi-class version of Hinge loss
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

    # Training the model
    for epoch in range(num_epochs):
        for i, (images_batch, labels_batch) in enumerate(train_loader):
            optimizer.zero_grad() # Clear the gradients
            outputs = model(images_batch) # Forward pass
            loss = criterion(outputs, labels_batch) # Calculate loss
            loss.backward() # Backward pass
            optimizer.step() # Update weights
```