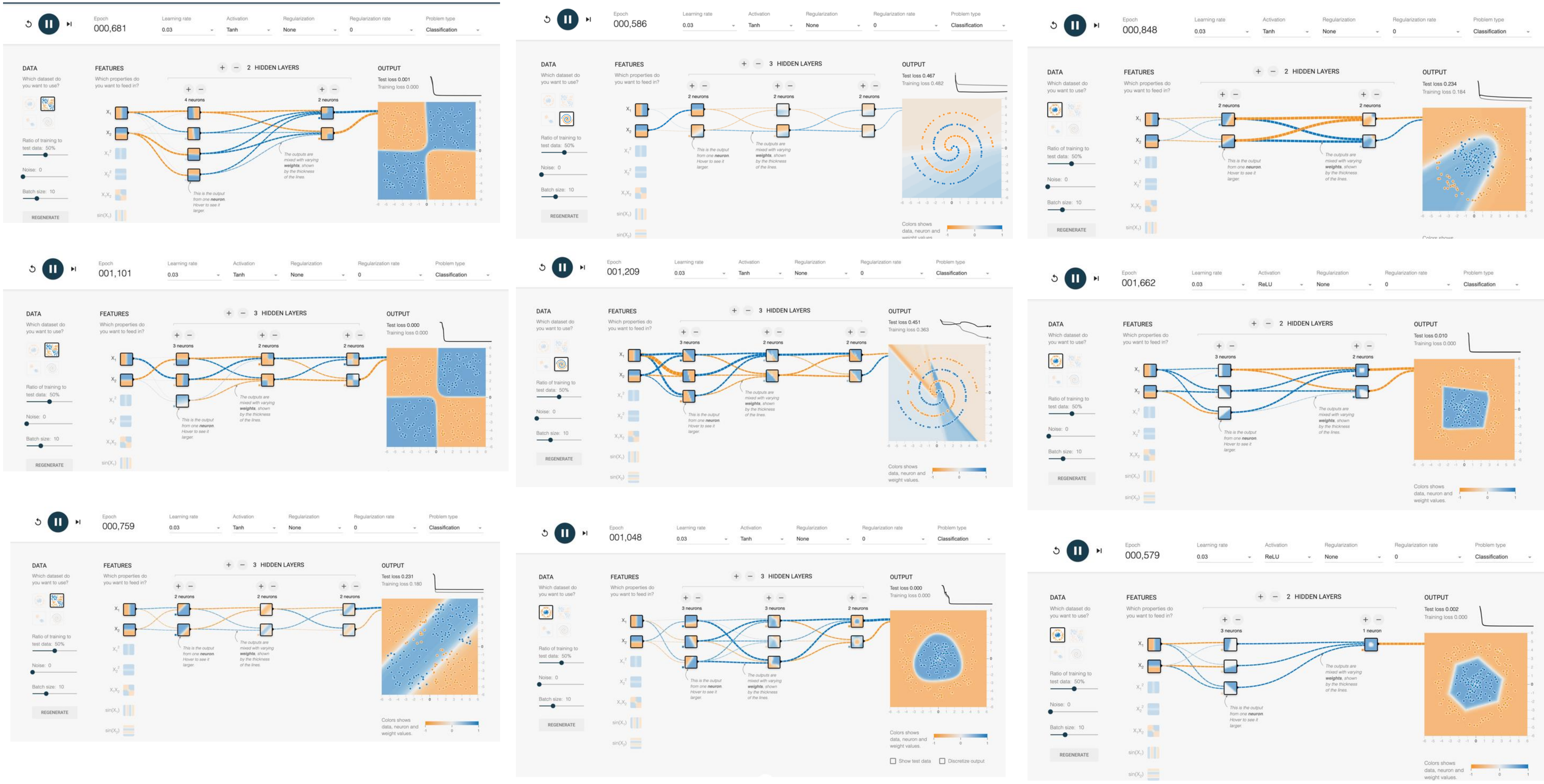


Epoch
000,681
Learning rate
0.03
Activation
Tanh
Regularization
None
Regularization rate
0
Problem type
Classification
DATA
Which dataset do you want to use?
FEATURES
Which properties do you want to feed in?
2 HIDDEN LAYERS
OUTPUT
Test loss 0.001
Training loss 0.000
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE
Epoch
000,586
3 HIDDEN LAYERS
Test loss 0.467
Training loss 0.482
Epoch
000,848
2 HIDDEN LAYERS
Test loss 0.234
Training loss 0.184
Epoch
001,101
3 HIDDEN LAYERS
Test loss 0.000
Training loss 0.000
Epoch
001,209
3 HIDDEN LAYERS
Test loss 0.451
Training loss 0.363
Epoch
001,662
ReLU
2 HIDDEN LAYERS
Test loss 0.010
Training loss 0.000
Epoch
000,759
3 HIDDEN LAYERS
Test loss 0.231
Training loss 0.180
Epoch
001,048
3 HIDDEN LAYERS
Test loss 0.000
Training loss 0.000
Epoch
000,579
ReLU
2 HIDDEN LAYERS
Test loss 0.002
Training loss 0.000
Colors shows data, neuron and weight values.
Show test data
Discretize output

Optimization Algorithms and Loss functions

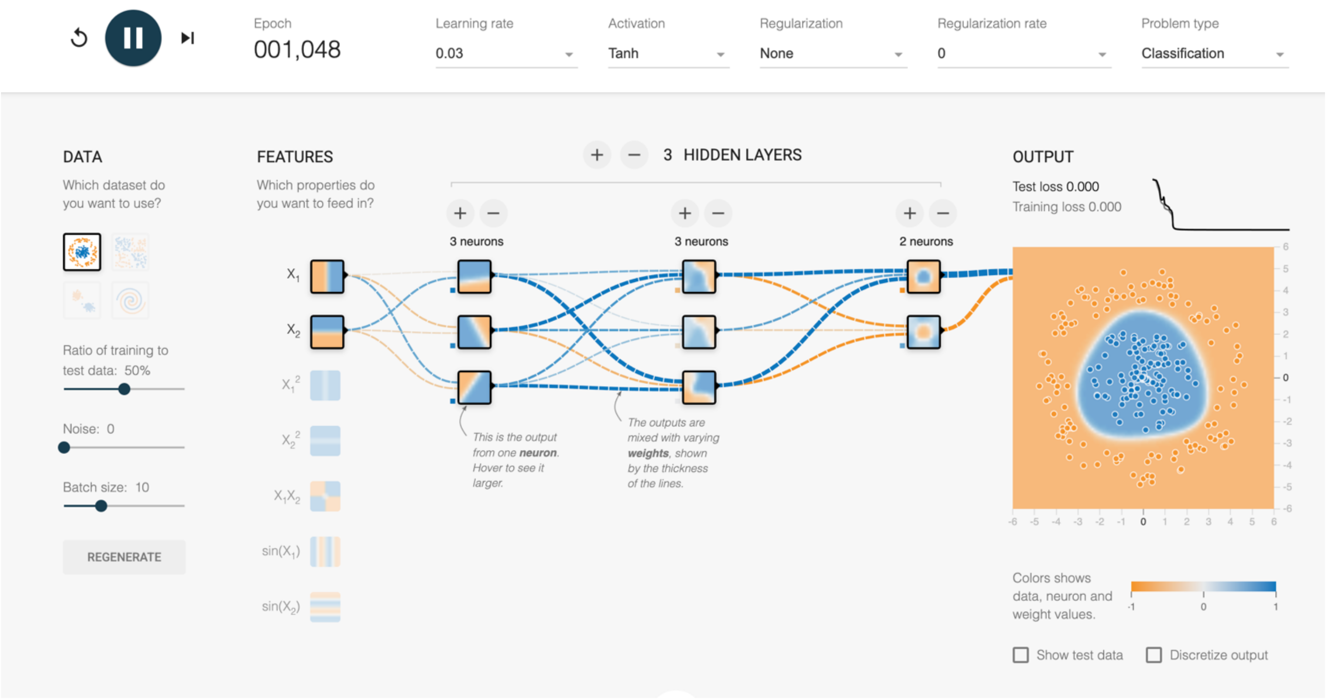
Machine Learning Theory (MLT)

Edinburgh

Rik Sarkar

How does this program find good models?

- In a neural network, models are defined by weights on the edges
- Good models correspond to right selection $\boldsymbol{w}$ of weights



Optimization: finding good models

- Our goal is to find $h \in \mathcal{H}$
- Such that $|L(h) - L(h^*)|$ is small
  - Where h^* is the best possible model

- Optimization algorithms try to find a good h (represented by weight vector $\boldsymbol{w}$)
  - That have a low loss

Today’s lecture

- Finding weights for a single neuron (linear models)
  - Logistic regression
- Convex functions and convex learning
- Gradient descent and Stochastic gradient descent
  - Main training algorithms in ML and Deep learning
- Continuity properties of loss functions
- Regularization
- Stability

Course

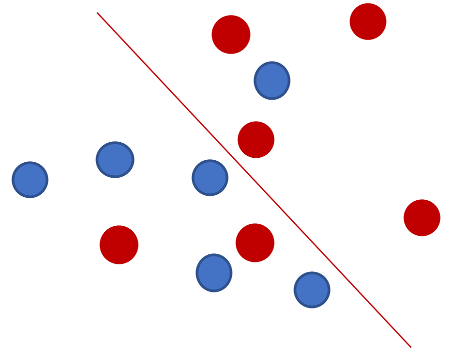
- Tutorial 1 next week
- Tutorial sheet will be out soon (by thursday).
- Please go over it before the tutorial

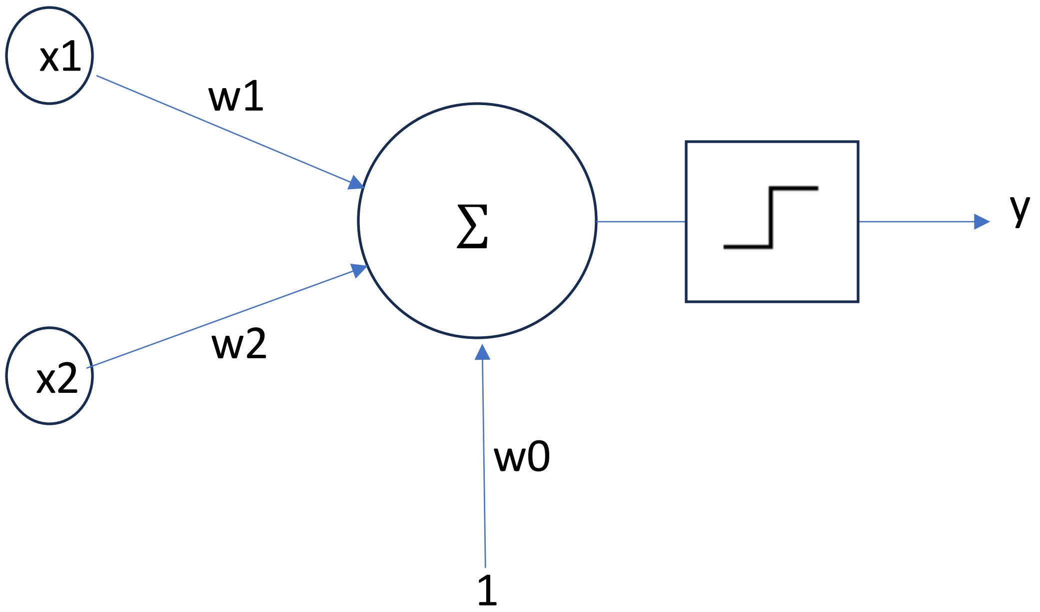
Single neuron

- Perceptron with threshold activation
  - $w_1, w_2, b \in \mathbb{R}$
- $y = (w_1 x_1 + w_2 x_2 + w_0 . 1 \geq 0)$
  - Truth value 0/1 (0r, -1/+1)

- We write
  - $z = w \cdot x$

- Optimization problem:
  - Find the best possible w
  - Represents model h_w



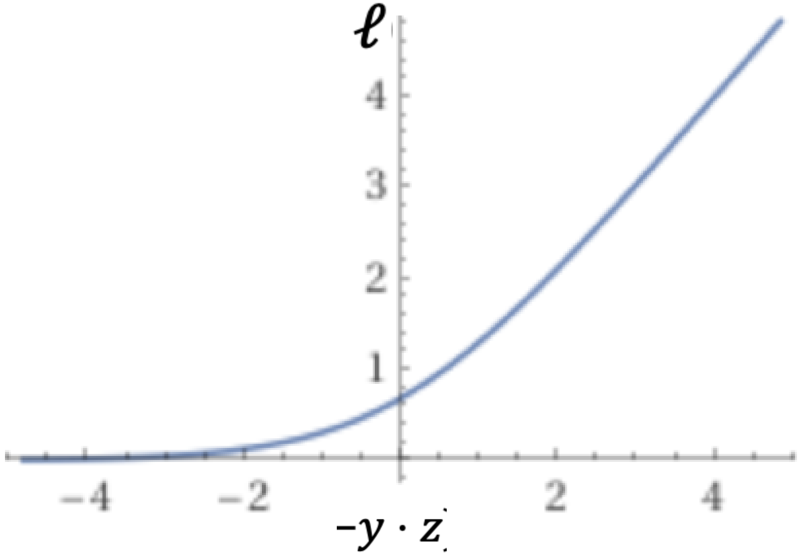


Logistic regression (used for classification!)

- Suppose point x has label $y \in \{-1, 1\}$

- If z and y have the same sign
  - Then the classification is correct
- If z and y have different signs
  - Then classification is incorrect

- The logistic loss function is:
  - $\ell\big(h_w, (x, y)\big) = \log(1 + \exp(-y \cdot z))$
  - If y, z are same sign, ℓ gets smaller with z
  - y, z are different signs, ℓ is larger with z



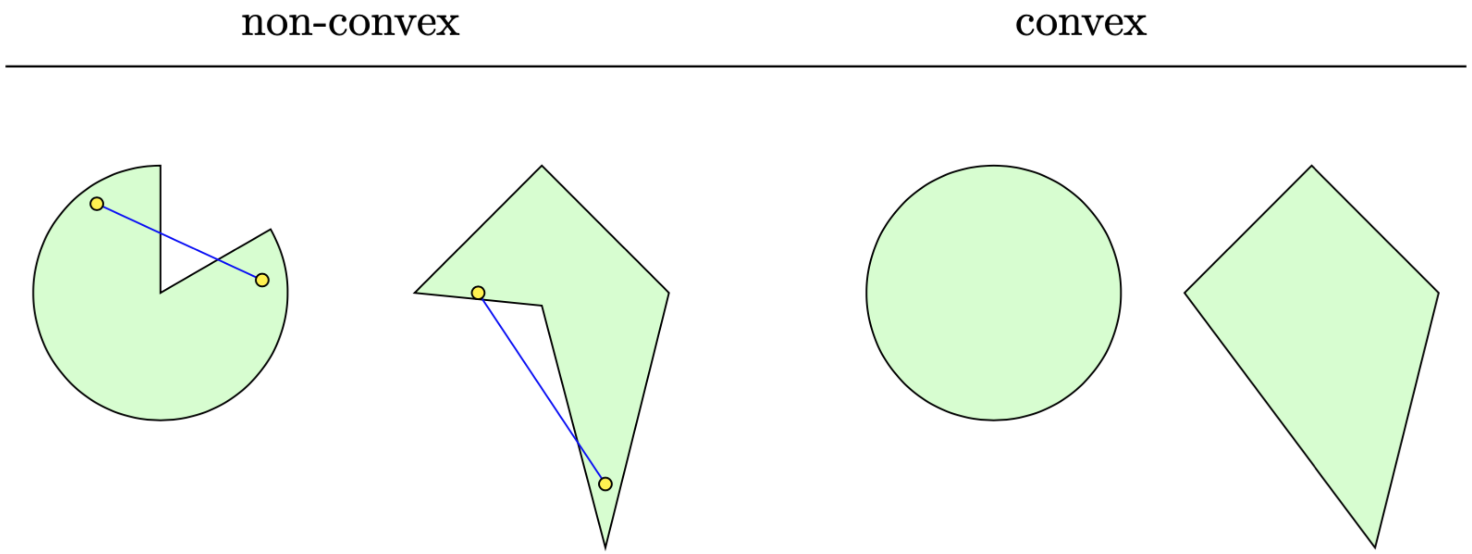
Logistic loss of S

- For a training dataset S
- We use the average logistic loss
- So, the best model w is the one with min logistic loss:
  - $argmin_w \frac{1}{m} \sum_{i=1}^{m} \log(1 + e^{-yz})$

- But we still need an algorithm to find this best w

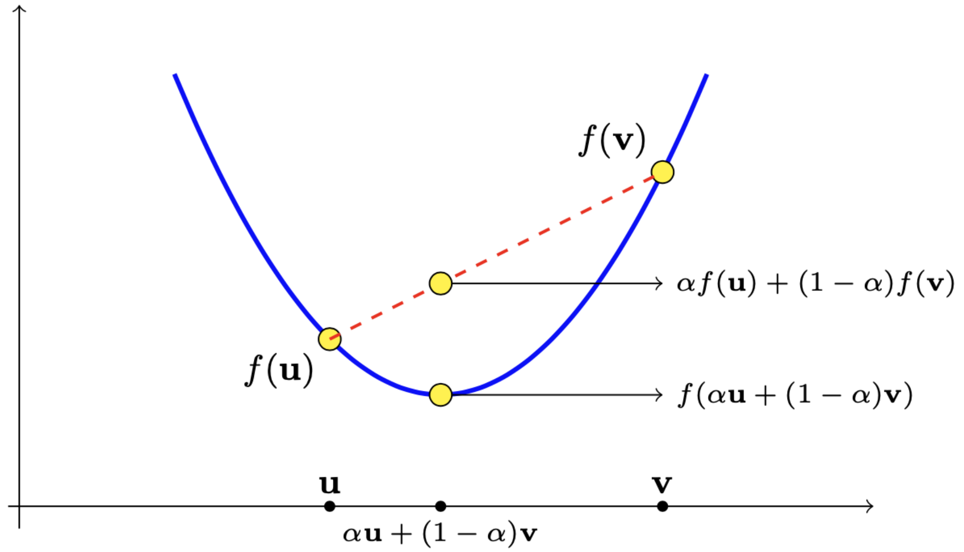
Convexity and convex learning

- A set C is convex if for any $u, v \in C$, the line segment connecting $\boldsymbol{u}, \boldsymbol{v}$ is in C. (Any intermediate point is in C)
  - Can be written formally as:
  - For any $\alpha \in [0,1]$, it is true that $\alpha\boldsymbol{u} + (1-\alpha)\boldsymbol{v} \in C$



Convex function

- For a convex C, a function $f: C \rightarrow \mathbb{R}$ is convex if
- $f(\alpha \boldsymbol{u} + (1-\alpha)\boldsymbol{v}) \leq \alpha f(\boldsymbol{u}) + (1-\alpha) f(\boldsymbol{v})$
- The graph of f lies below the straight line connecting u and v

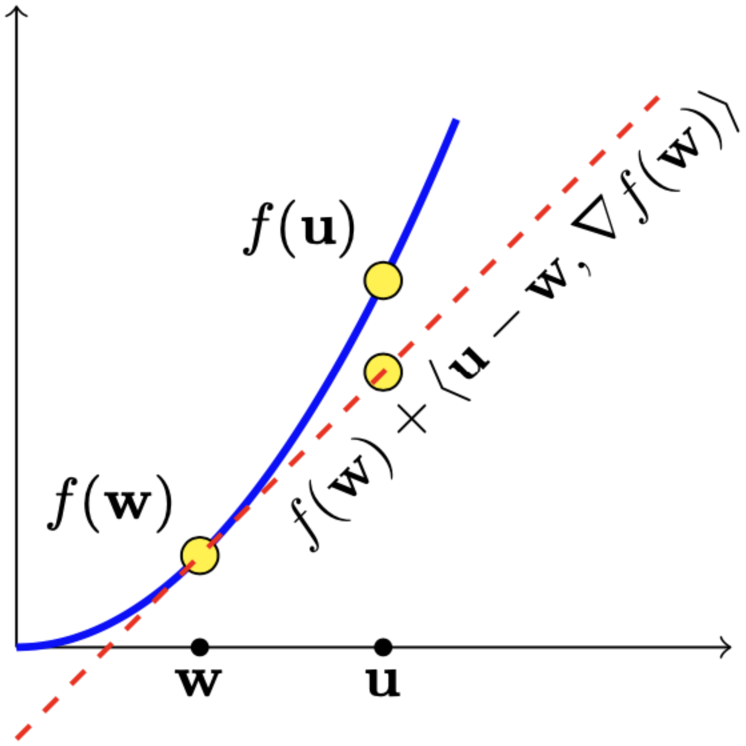


Properties of convex functions

- Every local minimum is also a global minimum
  - Question: is the global minimum unique?

- For every $\boldsymbol{w}$ the tangent at $\boldsymbol{w}$ lies below f:
  - $\forall \boldsymbol{u}, f(\boldsymbol{u}) \geq f(\boldsymbol{w}) + \langle \nabla f(\boldsymbol{w}), \boldsymbol{u} - \boldsymbol{w} \rangle$

- If $f: \mathbb{R} \to \mathbb{R}$ is twice differentiable, then
  - f is convex
  - f' is monotone nondecreasing
  - f'' is nonnegative
- Are equivalent



Examples of convex functions

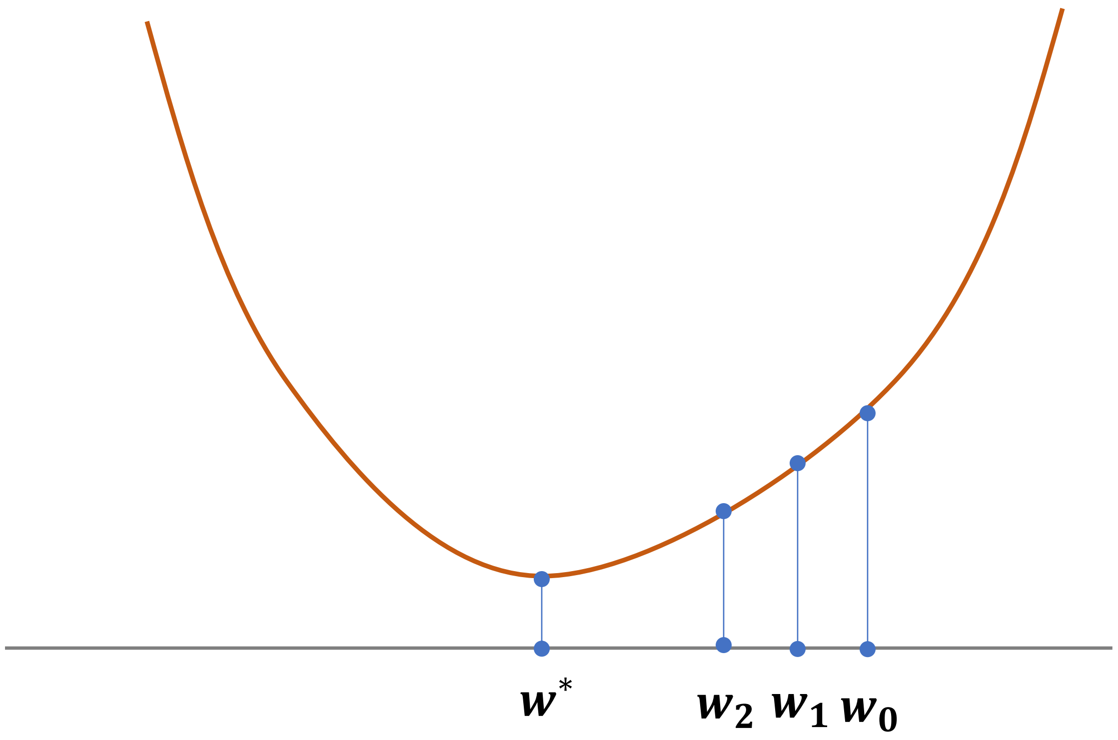
Examples of functions that are not convex

Combining convex functions

- If g is convex, then $f(\boldsymbol{w}) = g(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + y)$ is convex
- If f_i are convex functions
- $g(x) = \max_i f_i(x)$ is convex
- $g(x) = \sum_i w_i f_i(x)$ is convex
  - What is the consequence for loss functions?

Convex learning is easy!

- Start with any model $\boldsymbol{w}_0$
- Take a step in a direction that makes the loss smaller
- Repeat until we are close to $\boldsymbol{w}^*$ with smallest loss
- Gradient descent
  - Compute the derivative at current $\boldsymbol{w}$, move a step in that direction



Gradient

- Gradient (a vector derivative in multiple dimensions)
  - The direction and speed of fastest increase

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d} \right)$$

  - (here a w_i is a parameter or dimension of the model)

- Partial derivatives
  - Compute the derivative along each dimension, put them in a vector

Gradient descent

- Gradient is $$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w[1]}, \cdots, \frac{\partial f(\mathbf{w})}{\partial w[d]} \right)$$
- Gradient represents the direction in which f increases fastest
- Gradient Descent: At every step t :
  - $w^{t+1} = w^t - \eta \nabla f(w^t)$
    - (Move in the direction that f decreases fastest With a step scale of η)
- After T steps, output the average vector $\bar{\boldsymbol{w}} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{w}^t$
- Other version: output final vector $\boldsymbol{w_T}$
- For us, f is the average loss L

Theorem (14.2 in book)

- For convex lipschitz bounded learning
- Setting $\eta = \sqrt{\frac{B^2}{\rho^2 T}}$
- We can get $$f(\bar{\mathbf{w}}) - f(\mathbf{w}^\star) \;\leq\; \frac{B\,\rho}{\sqrt{T}}.$$

- Alternatively, to achieve $f(\overline{\boldsymbol{w}}) - f(\boldsymbol{w}^*) \leq \epsilon$ the number of rounds is:

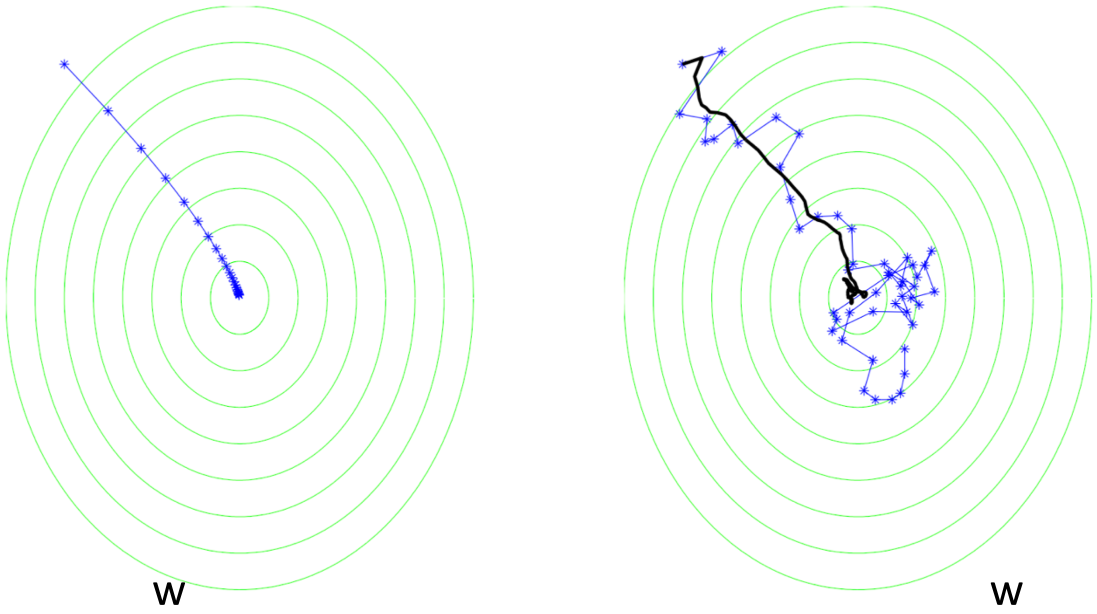
$$T \geq \frac{B^2 \rho^2}{\epsilon^2}$$

Stochastic gradient descent

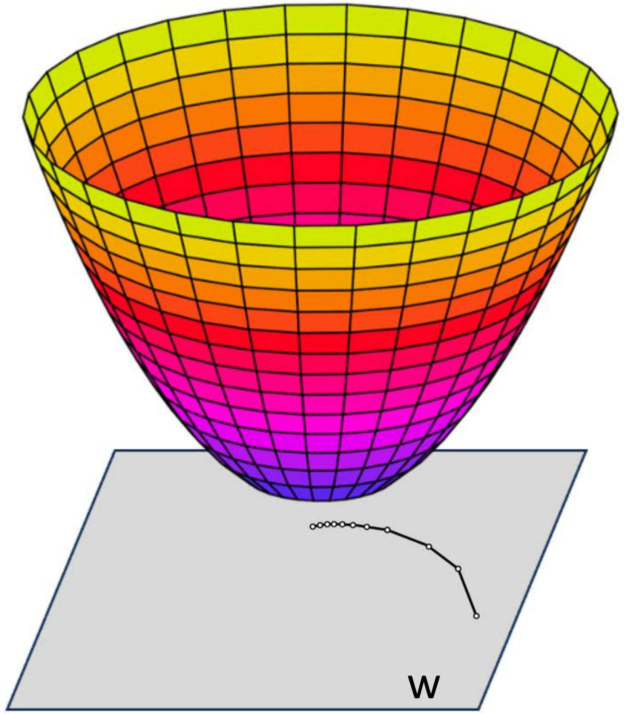
- Computing the gradient of empirical loss is expensive
  - Because empirical loss depends on all training data
  - And every step requires a pass through entire dataset
- Idea: Instead of computing gradient on the entire dataset each time, compute them on small samples:
  - Small batches of data
  - Or even a single data point
    - (Each i.i.d data point is treated like a tiny sample of data)
  - While any single data point does not represent the set, on average they behave simmilarly

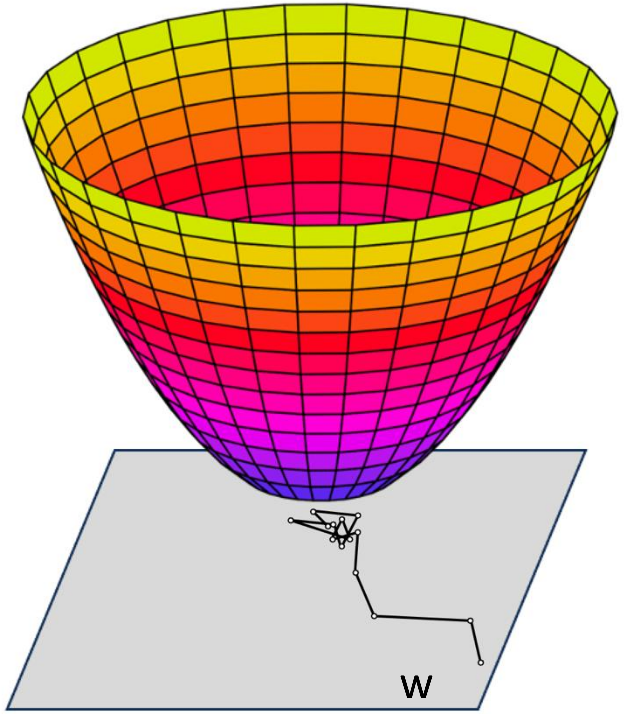
GD vs SGD

- SGD takes a more random path, but follows similar trends



Finding the best model by looking for the lowest point of the loss function





Stochastic gradient descent (from book)

Stochastic Gradient Descent (SGD) for minimizing

$$f(\mathbf{w})$$

parameters: Scalar $\eta > 0$, integer $T > 0$
initialize: $\mathbf{w}^{(1)} = \mathbf{0}$
for $t = 1, 2, \ldots, T$
  choose $\mathbf{v}_t$ at random from a distribution such that $\mathbb{E}[\mathbf{v}_t \mid \mathbf{w}^{(t)}] \in \partial f(\mathbf{w}^{(t)})$
  update $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{v}_t$
output $\bar{\mathbf{w}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{w}^{(t)}$

Stochastic gradient descent other version

- Initialize $\boldsymbol{w}^1$ randomly (uniform or gaussian)

- For $t = 1 \ldots T$
  - Take a random small sample of data (mini batch)
  - Compute gradient $\boldsymbol{v}^t$ on this sample
  - Update $\boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \eta \boldsymbol{v}^t$
- Output $\boldsymbol{w}^T$

Theorem (14.8)

- Similar result to deterministic GD:

$$\mathbb{E}\left[f(\bar{\mathbf{w}})\right] - f(\mathbf{w}^\star) \leq \frac{B\,\rho}{\sqrt{T}}$$

Practical modifications

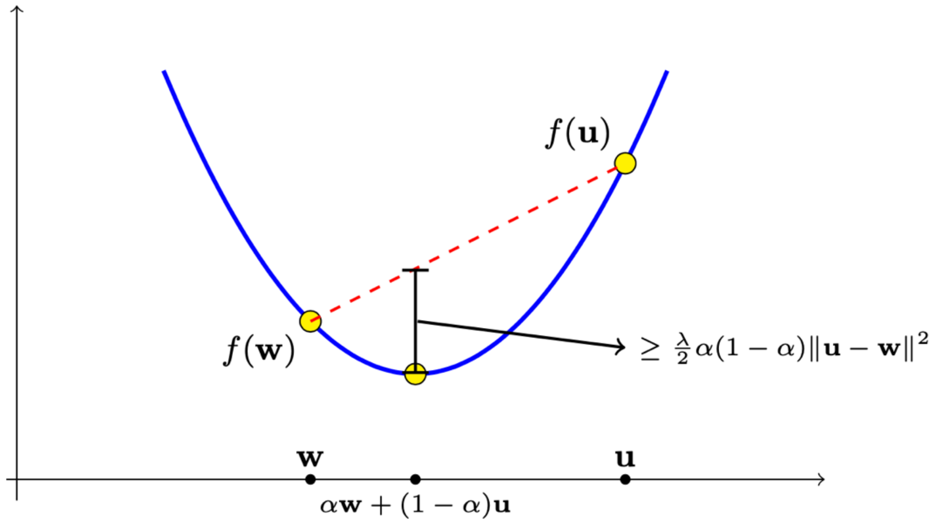
- For large neural nets
  - Simply put all edge weights in the same vector
  - SGD algorithm does not depend on what your model type is
  - As along as all parameters are real valued
- Mini batching:
  - Instead of one data item at time, take them in batches of a few at a time.
  - Faster, and fewer unhelpful moves
- Run in epochs. In each epoch
  - Order the data points in a random permutation
  - For each data point (or mini-batch)
    - Compute the gradient and move the modes
- Other modifications:
  - Change learning rates
  - Add momentum, add dropout etc

Other properties of loss functions

Strong Convexity

- Function f is λ-strongly convex if

$$f(\alpha \mathbf{w} + (1-\alpha)\mathbf{u}) \leq \alpha f(\mathbf{w}) + (1-\alpha) f(\mathbf{u}) - \frac{\lambda}{2}\alpha(1-\alpha)\|\mathbf{w}-\mathbf{u}\|^2$$



Lipschitzness

- A function f is ρ-Lipschitz if
  - $\left|\left|f(\boldsymbol{w}_1) - f(\boldsymbol{w}_2)\right|\right| \leq \rho||\boldsymbol{w}_1 - \boldsymbol{w}_2||$

- A function that does not change too fast
  - If the derivative is bounded by ρ,
    - What can we say about its lipschitzness?
    - Then the function is also ρ-Lipschitz
    - But lipschitzness can be defined/computed even when the derivative does not exist

Smoothness

- Gradient (a vector derivative in multiple dimensions)
  - The direction and speed of fastest increase

$$\nabla f(\mathbf{w}) = \left(\frac{\partial f(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial f(\mathbf{w})}{\partial w_d} \right)$$

- f is β-smooth if ∇f is β-Lipschitz:
  - $\left|\left|\nabla f(\boldsymbol{v}) - \nabla f(\boldsymbol{w})\right|\right| \le \beta ||\boldsymbol{v} - \boldsymbol{w}||$

Convex-Lipschitz-Bounded learning problems

- A learning problem $(\mathcal{H}, \mathcal{Z}, \ell)$ where:

- $\mathcal{H}$ is convex, $\forall \boldsymbol{w} \in \mathcal{H}, \left||\boldsymbol{w}|\right| \leq B$
- $\forall z \in \mathcal{Z}$ the loss $\ell(\cdot, z)$ is convex and ρ-Lipschitz (for some ρ)

Convex-smooth-bounded learning

- A learning problem $(\mathcal{H}, \mathcal{Z}, \ell)$ where:

- $\mathcal{H}$ is convex, $\forall \boldsymbol{w} \in \mathcal{H}, \left|\left|\boldsymbol{w}\right|\right| \leq B$
- $\forall z \in \mathcal{Z}$ the loss $\ell(\cdot, z)$ is convex, nonnegative and β-smooth (for some β)

Why do we want convexity, smoothness, lipschitzness etc?

Why do we want convexity, smoothness, lipschitzness etc?

- Avoids sudden changes in function and its gradients
- Easier to compute and apply gradients as optimization steps

- Most theoretical analysis assume some of these properties
- Most practical situations have similar properties
  - For most regions of data space and model space
  - It is hard to make SGD, or any algorithm work if it does not

What is the problem of 0-1 empirical risk as loss function?

- Remember that we had defined the average empirical error as the loss.
  - Can we use that for gradient descent?

Surrogate loss functions

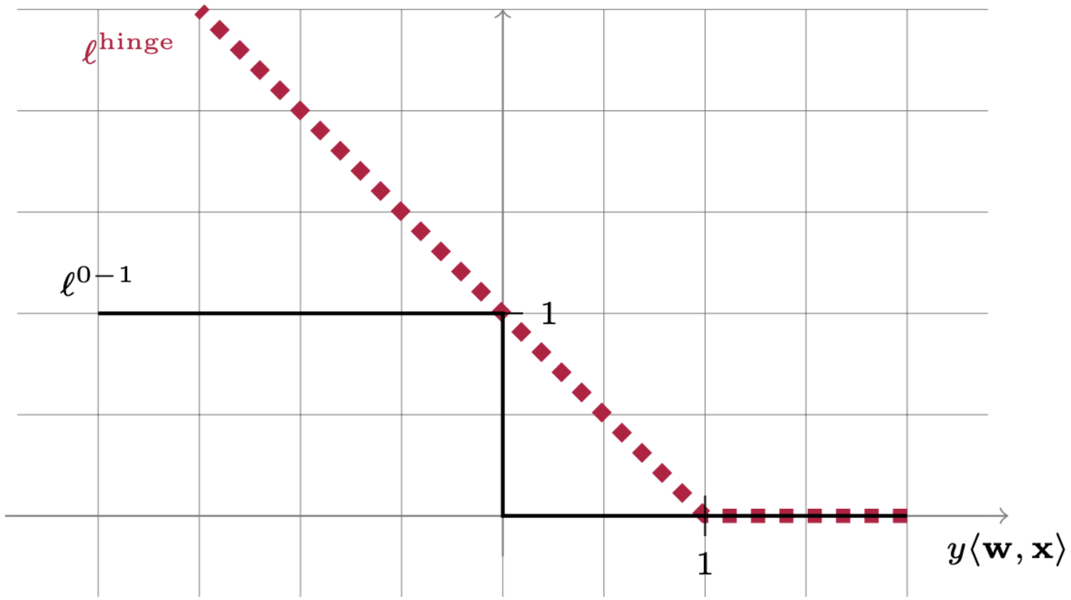
- Some loss functions are hard to work with. E.g.
  - They are not convex
  - They are hard to optimize for
  - E.g. 0-1 loss in linear classification

- Solution
  - Use a “surrogate” loss function
  - That is kind of similar, but easier to manage, e.g. convex

- Usual rule for surrogate loss
  - Should be convex
  - Should upper bound (be larger than original loss.)

Example: Hinge loss

$$\ell^{\text{hinge}}(\mathbf{w}, (\mathbf{x}, y)) \stackrel{\text{def}}{=} \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle\}$$



Regularization

- Instead of the pure loss, minimize loss with a regularization term:

$$\underset{\mathbf{w}}{\operatorname{argmin}}\,(L_S(\mathbf{w}) + R(\mathbf{w}))$$

- Commonly used: $R(\boldsymbol{w}) = \lambda||\boldsymbol{w}||^2$
    - Called Tikhonov regularization

Try yourself:

Go to wolfram alpha and plot a polynomial: $y = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$

- With numbers of your choice in place of coefficients a_i
- Now scale the coefficients: multiply all the coefficients with the same number (may be fractions too). What do you see?

Ridge regression

- Linear regression with Tikhonov regularization

$$\underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmin}} \left(\lambda\|\mathbf{w}\|_2^2 + \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(\langle \mathbf{w}, \mathbf{x}_i\rangle - y_i)^2 \right)$$

- $R(\boldsymbol{w}) = \lambda\left||\boldsymbol{w}|\right|^2$ is 2λ-strongly convex
- If f is λ-strongly convex and g is convex, then $f + g$ is λ-strongly convex

- Thus, Ridge regression is strongly convex
- Strongly convex loss implies stability – useful property in SGD and other methods

Stability

- Intuitively: A learning algorithm is stable if
  - A small change to training set does not cause a big change to the output (model or hypothesis)

- This is a desirable property because…

Stability

- Intuitively: A learning algorithm is stable if
  - A small change to training set does not cause a big change to the output (model or hypothesis)
- This is a desirable property because
  - It implies that it is not too sensitive to specific S. does not overfit
  - If we continue to use it, it will not abruptly change behavior

- Suppose in S, we replace z_i with $z' \sim \mathcal{D}$
- Let us write this as S^i

- A good algorithm A should have small value for
    - $|\ell\left(A\left(S^i\right), z_i\right) - \ell(A(S), z_i)|$

- The loss on z_i does not depend too much on it being in the sample

Stability definition and result

- Algorithm A is on-average-replace-one-stable with rate $\epsilon(m)$
- If
  - $\mathbb{E}\left[\ell\left(A\left(S^i\right), z_i\right) - \ell(A(S), z_i)\right] \leq \epsilon(m)$

Stability definition and result

- Algorithm A is on-average-replace-one-stable with rate $\epsilon(m)$
- If
  - $\mathbb{E}\left[\ell\left(A\left(S^i\right), z_i\right) - \ell(A(S), z_i)\right] \leq \epsilon(m)$

- Theorem:
  - $\mathbb{E}\left[L_{\mathcal{D}}\left(A(S)\right) - L_S\left(A(S)\right)\right] = \mathbb{E}\left[\ell\left(A\left(S^i\right), z_i\right) - \ell(A(S), z_i)\right]$

  - The generalization gap is bounded by the stability

Generalisation Gap

- Empirical or training loss: $L_S(h)$
- Generalisation loss or true loss : $L_{\mathcal{D}}(h)$

- $L_{\mathcal{D}}(h) - L_S(h)$
- A measure of overfitting
  - (sometimes generalization gap is referred to as generalization loss)

Uniform Stability

- Suppose we get S^i by replacing one element z_i at position i of S with a new element z_i'
- And suppose that $z \in \mathcal{Z}$ is some possible input element
- As before, $A(S)$ refers to the model that algorithm A computes using S
- We can write the loss on z as $\ell(A(S), z)$

- Algorithm A is ϵ-uniformly stable if
  - $\mathrm{Sup}_{z \in \mathcal{Z}} \left[E_A \ell\left(A(S^i), z\right) - E_A \ell(A(S), z) \right] \leq \epsilon$
- E_A means expectation taken over all possible random behaviour of A

Uniform Stability implies generalization

- Theorem:
- If Randomized Algorithm A is ϵ-uniformly stable then
  - $E_S E_A \ell(A(S), \mathcal{D}) \leq E_S E_A \ell(A(S), S) + \epsilon$
  - True loss $\leq$ Training loss + ϵ

Uniform Stability implies generalization

- Theorem:
- If Algorithm A is ϵ-uniformly stable then
  - $E_S E_A \ell(A(S), \mathcal{D}) \leq E_S E_A \ell(A(S), S) + \epsilon$
  - Expected true loss $\leq$ Training loss + ϵ
- Proof: Omitted (for now).

Observe

- Regularization creates strong convexity
- Strong convexity implies stability
- Stability implies generalization

Next

- Larger neural networks
  - Losses are non-convex
- What happens with non-convex loss?
- Shapes of loss functions
- Overfitting and overparameterization in neural networks