Lab 9

Exercise 1

In this exercise, we first train a logistic regression classifier on a subset of the MNIST dataset, containing only the digits zero and one. Then, we look for adversarial examples that can fool this classifier.

Training logistic regression

First of all, we download the dataset (using Keras's utility), discard the samples we do not need, normalize the inputs, turn them into vectors, add a bias term, and encode the labels to ± 1:

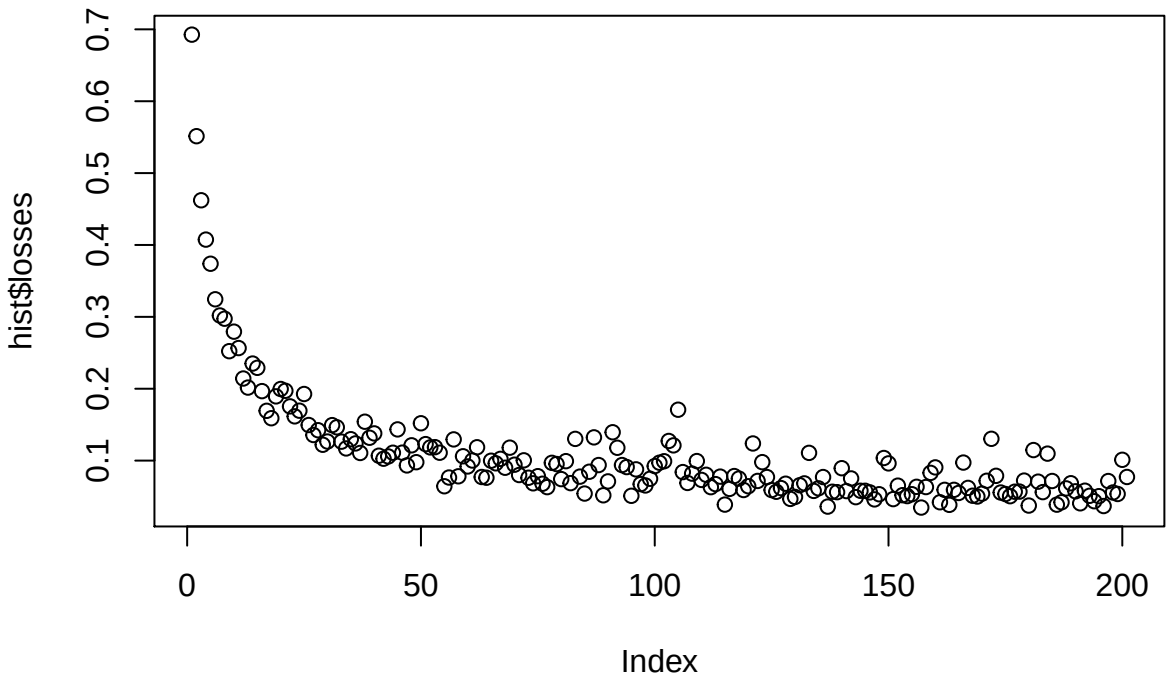
```
library(keras)

mnist = dataset_mnist()

train_mask = (mnist$train$y == 1) | (mnist$train$y == 2)
train_x = mnist$train$x[train_mask,,] / 255.0
dim(train_x) <- c(nrow(train_x), 28 * 28)
train_x = cbind(train_x, rep(1, nrow(train_x)))
train_y = 2 * mnist$train$y[train_mask] - 3

test_mask = (mnist$test$y == 1) | (mnist$test$y == 2)
test_x = mnist$test$x[test_mask,,] / 255.0
dim(test_x) <- c(nrow(test_x), 28 * 28)
test_x = cbind(test_x, rep(1, nrow(test_x)))
test_y = 2 * mnist$test$y[test_mask] - 3
```

We now implement and train logistic regression, with loss function

$$\mathcal{L}(y, f(\mathbf{x}|\theta)) = \log\left(1 + \exp\left(-y \cdot \theta^T \mathbf{x}\right)\right)$$

```
train_logreg = function(data_x, data_y, max_steps, lrate, batch_size) {
  theta = rnorm(ncol(data_x), sd=0.001)

  losses = sapply(1:(max_steps + 1), function(i) {
    batch_idx = sample(nrow(data_x), batch_size)

    batch_x = data_x[batch_idx,]
    batch_y = data_y[batch_idx]

    linear = batch_x %*% theta
    loss = sum(log(1 + exp(-batch_y * linear[,1]))) / batch_size

    es = exp(-batch_y * linear[,1])
    grad = -batch_y * es / (1 + es) / batch_size
    grad = colSums(sweep(batch_x, MARGIN = 1, grad, '*'))

    theta <<- theta - lrate * grad
```

```
    loss  # return the loss
  })

  list(theta = theta, losses = losses)
}


hist = train_logreg(
  train_x, train_y, max_steps = 200, lrate = 0.1, batch_size = 128
)
theta = hist$theta
plot(hist$losses)
```



We then compute the accuracy on the test set:

```
# compute test accuracy
test_predictions = ifelse(test_x %*% theta > 0, 1, -1)[, 1]
mean(test_predictions == test_y)
```

```
## [1] 0.9893862
```

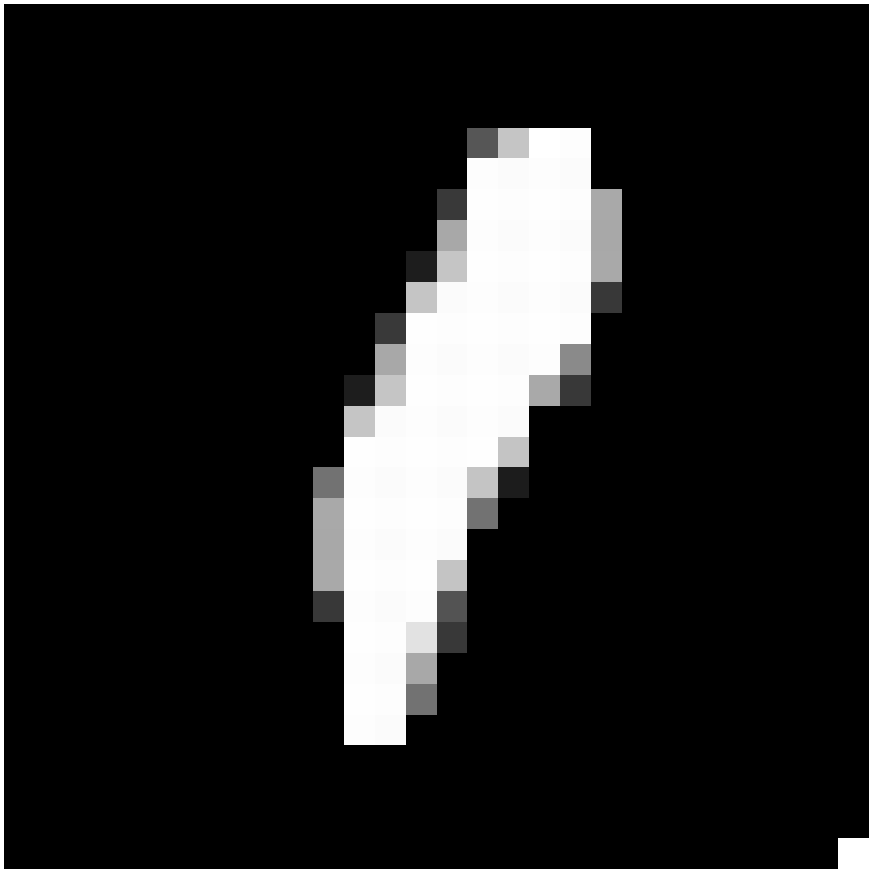
Randomly searching adversarial examples

Now that the classifier is trained, we can construct adversarial examples. The first strategy we try is to randomly generate vectors of a given length, and check whether they result in a different classification.

First, let's select an example that is classified correctly:

```
predict_class = function(x) {
  ifelse(x %*% theta > 0, 1, -1)
}

# choose an example that is correctly classified
repeat {
```

```
  idx = floor(runif(1, 0, nrow(train_x)))

  sample = train_x[idx,]
  label = train_y[idx]

  if(predict_class(sample) == label) {
    break
  }
}

library(grid)
grid.raster(matrix(tail(sample, -1), nrow = 28), interpolate = FALSE)
```



Next, we write a function that perturbs this sample with a vector whose elements are all $\pm\epsilon$, and checks whether the class is changed:

```
perturb_and_check = function(eps) {
  delta = (
    eps * sign(rnorm(length(sample)))
  )

  prediction = (
    predict_class(sample + delta)
  )

  ifelse(prediction == label, 1, -1)
```

```
}
```

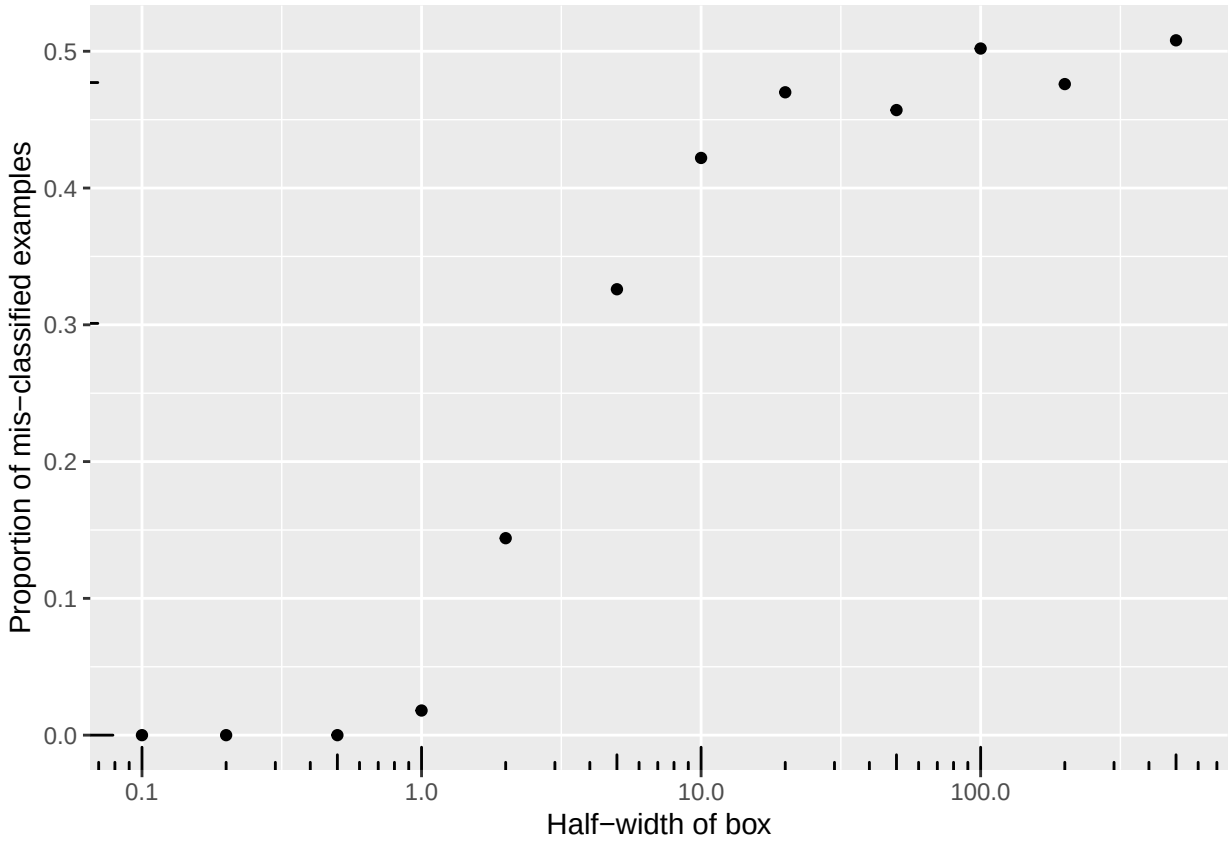
Now, we try different values for `eps`. For each of them, we generate one thousand different perturbations, and compute the proportion that result in a change of class:

```
count = 1000
eps_range = c(1e-1, 2e-1, 5e-1, 1e0, 2e0, 5e0, 1e1, 2e1, 5e1, 1e2, 2e2, 5e2)

p_wrong = sapply(eps_range, function(eps) {
  sum(sapply(1:count, function(i) {
    ifelse(perturb_and_check(eps) > 0, 0, 1)
  })) / count
})

p_wrong
```

```
##  [1] 0.000 0.000 0.000 0.018 0.144 0.326 0.422 0.470 0.457 0.502 0.476 0.508
```

We can also plot this:

```
library(ggplot2)
ggplot() +
  geom_point(aes(x = eps_range, y = p_wrong)) +
  scale_x_log10() +
  annotation_logticks() +
  xlab("Half-width of box") +
  ylab("Proportion of mis-classified examples")
```



Clearly, there are no points with a different classification up to a certain distance from the original point.

The specific range depends, obviously, on the point in question. Further than that, and the proportion of points with a different class grows larger and larger, until it stabilizes to a value smaller than one. Such range can be thought of as the distance of the point to the decision boundary. If you now imagine a hyper-cube centered on this point, the proportion of mis-classified examples is (an estimation of) the amount of surface of this cube that is on the other side of the separating hyper-plane. This intuition is valid regardless of the specific classification model employed, with the only difference that the decision boundary can be arbitrarily more complex than a hyper-plane.

Creating an adversarial example via gradient ascent

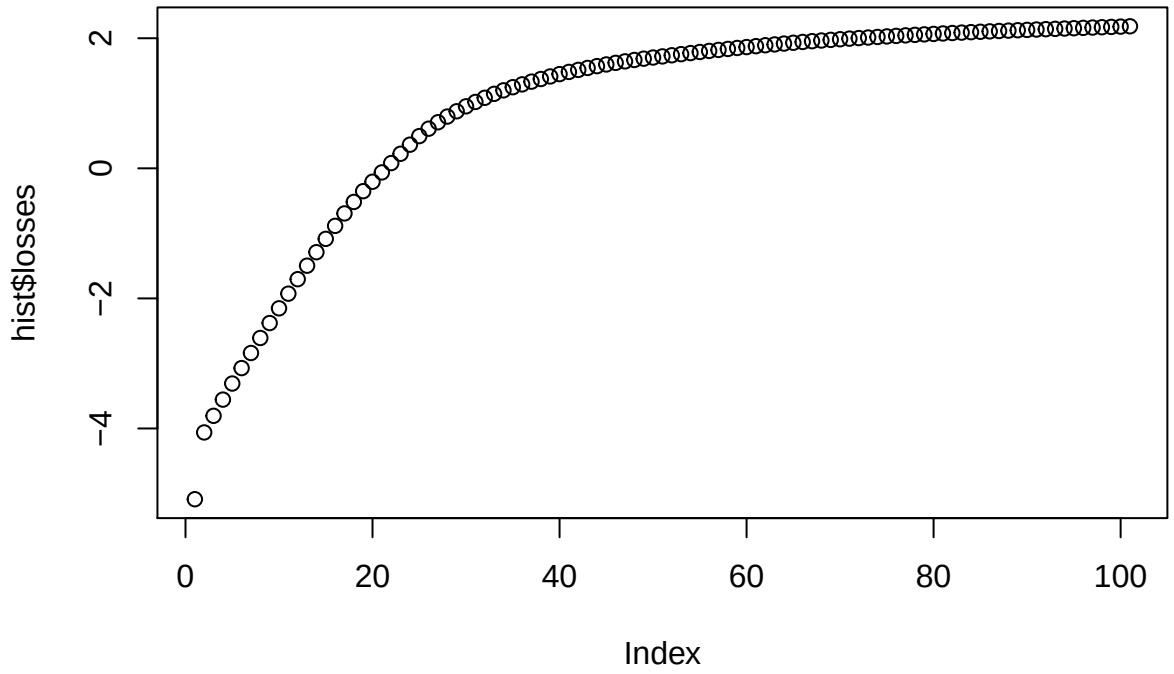
Now, we can try to look for adversarial examples more intelligently. Specifically, we want to find a vector δ^* such that

$$\delta^* = \text{argmax}_\delta \left[\mathcal{L}(y, f(\mathbf{x})) - \mathcal{L}(y, f(\mathbf{x} + \delta))\right]$$

and $\delta \in \mathcal{B}_\epsilon^\infty$, with $f(x)$ being our trained logistic regression classifier. The function above can be maximized with gradient ascent, by slightly modifying the training procedure we employed earlier. In particular, note that we can use values for epsilon for which a random perturbation has practically no chance to result in a change of class.

```
make_adversarial = function(theta, sample, true_class, eps, max_steps, lrate) {
  delta = rnorm(length(sample))

  wanted_class = -1 * true_class

  losses = sapply(1:(max_steps + 1), function(i) {

    linear = (sample + delta) %*% theta

    loss_true = log(1 + exp(-true_class * linear[,1]))
    loss_wanted = log(1 + exp(-wanted_class * linear[,1]))

    est = exp(-true_class * linear[,1])
    grad_true = -true_class * theta * est / (1 + est)

    esw = exp(-wanted_class * linear[,1])
    grad_wanted = -wanted_class * theta * esw / (1 + esw)

    delta <<- delta + lrate * (grad_true - grad_wanted)

    delta <<- pmin(eps, pmax(delta,-eps))

    loss_true - loss_wanted  # return difference in loss
  })

  list(delta = delta, losses = losses)
}

hist = make_adversarial(
  theta,
  sample = sample,
  true_class = label,
  eps = 0.2,
  max_steps = 100,
```

```
  lrate = 0.1
)
delta = hist$delta
plot(hist$losses)
```



The label changes when the new loss becomes positive. We can now check that the class was indeed changed:

```
c(
  predict_class(sample),
  predict_class(sample + delta)
)
```

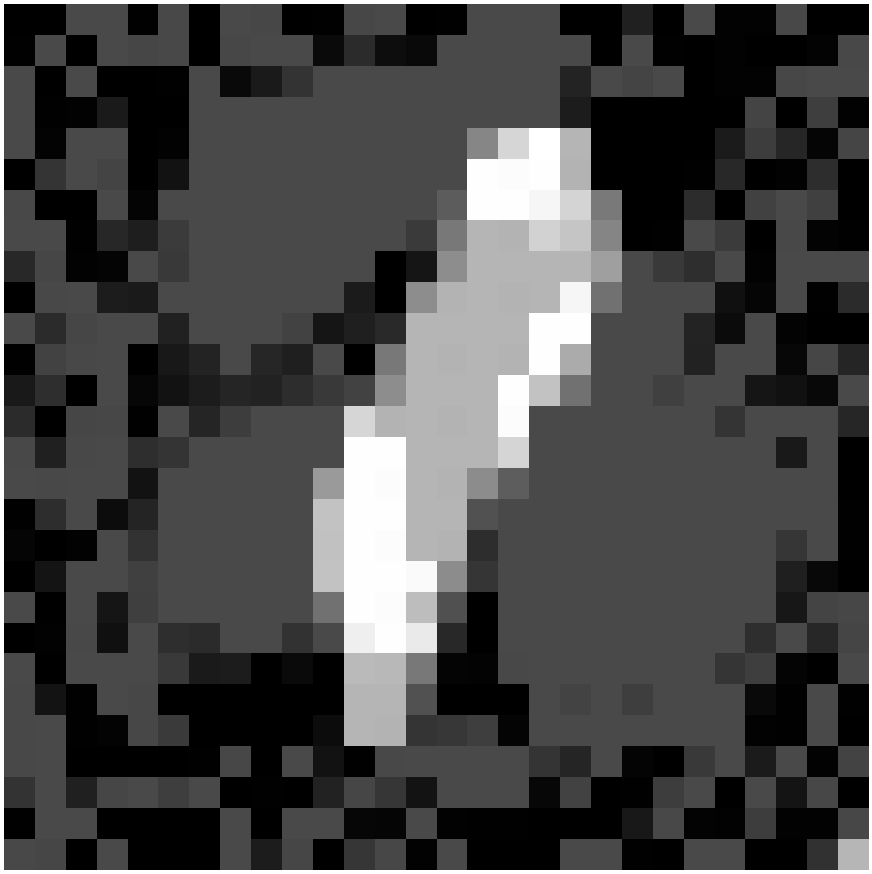
```
## [1] -1  1
```

We can also visualize the resulting perturbation vector:

```
dd = (delta - min(delta)) / (max(delta) - min(delta))
grid.raster(matrix(tail(dd, -1), nrow = 28), interpolate = FALSE)
```

and the perturbed sample:

```
ss = sample + delta
dd = (ss - min(ss)) / (max(ss) - min(ss))
grid.raster(matrix(tail(dd, -1), nrow = 28), interpolate = FALSE)
```



Can you interpret these images?

Exercise 2

Suppose that the adversarial examples for a logistic regression classifier are generated in $\mathcal{B}_\epsilon^2(\mathbf{x})$ instead of $\mathcal{B}_\epsilon^\infty(\mathbf{x})$. Show that the adversarial risk becomes

$$\mathcal{L}(y, f(\mathbf{x}|\theta)) = \Psi\left(y(\theta^T\mathbf{x}) - \epsilon||\theta||_2\right)$$

Solution

We can follow the same reasoning in the slides, i.e.

$$\delta^* = \max_{\delta\in\Delta(\mathbf{0})} \mathcal{L}(y, f(\mathbf{x}+\delta|\theta)) = \ldots = \min_{\delta\in\Delta(\mathbf{0})} y(\theta^T\delta)$$

Now, however

$$\Delta(\mathbf{0}) = \mathcal{B}_\epsilon^2(\mathbf{0}) = \{\delta : \delta \in \mathbb{R}^n \wedge ||\delta||_2 \leq \epsilon\}$$

When $y = 1$, we have to find a vector δ of length ϵ that minimizes the dot-product of θ; such δ is headed in the opposite direction of θ. When $y = -1$, the dot-product has to be maximized, i.e. δ points in the same direction of θ. This means that

$$\delta^* = -y\epsilon\frac{\theta}{||\theta||_2} = \begin{cases} -\epsilon\frac{\theta}{||\theta||_2} & y = 1 \\ \epsilon\frac{\theta}{||\theta||_2} & y = -1 \end{cases}$$

Replacing this in the adversarial risk yields

$$\mathcal{L}(y, f(\mathbf{x}|\theta)) = \Psi\left(y(\theta^T\mathbf{x}) - y\theta^T\delta_*\right) = \Psi\left(y(\theta^T\mathbf{x}) - \epsilon\frac{\theta^T\theta}{||\theta||_2}\right) = \Psi\left(y(\theta^T\mathbf{x}) - \epsilon||\theta||_2\right)$$