Sprawozdanie

Dominik Rafacz

18.04.2020

Przygotowanie

Załadowanie niezbędnych pakietów

```
library(MIOwAD)      # pakiet z sieciami
library(dplyr)       # transformacja danych
library(ggplot2)     # wykresy
```

Wczytywanie danych

```
# wyspecyfikowanie danych do poszczegolnych eksperymentow
dat_regr_names <- c("square-simple", "square-large", "steps-small",
                    "steps-large", "multimodal-large", "multimodal-sparse")
dat_classif_names <- c("easy", "rings3-regular", "rings5-regular", "xor3",
                       "rings5-sparse", "rings3-balance", "xor3-balance")
dat_lab1_names <- c("square-simple", "steps-large")
dat_lab2_names <- c("square-simple", "steps-small", "multimodal-large")
dat_lab3_names <- c("square-large", "steps-large", "multimodal-large")
dat_lab4_names <- c("easy", "rings3-regular", "xor3")
dat_lab5_names <- c("steps-large", "multimodal-large", "rings3-regular", "rings5-regular")
dat_lab6_names <- c("multimodal-sparse", "rings5-sparse", "rings3-balance", "xor3-balance")

dat_lab5_regr_names <- c("steps-large", "multimodal-large")
dat_lab5_classif_names <- c("rings3-regular", "rings5-regular")
dat_lab6_regr_names <- "multimodal-sparse"
dat_lab6_classif_names <- c("rings5-sparse", "rings3-balance", "xor3-balance")

# wczytanie datasetow
dat <- c(
  nlapply(dat_regr_names, function(name)
    list(train = read.csv(paste0("../data/regression/", name, "-training.csv")),
         test = read.csv(paste0("../data/regression/", name, "-test.csv")))),
  nlapply(dat_classif_names, function(name)
    list(train = read.csv(paste0("../data/classification/", name, "-training.csv")),
         test = read.csv(paste0("../data/classification/", name, "-test.csv"))))
)

# wydobycie zbiorow X
X <- c(
  nlapply(dat_regr_names, function(name) {
    train <- scale(dat[[name]][["train"]][["x"]]) # ze skalowaniem
    list(train = as.matrix(train),
         test = as.matrix(scale(dat[[name]][["test"]][["x"]],
                                center = attr(train, "scaled:center"),
                                scale = attr(train, "scaled:scale"))))
  }),
  nlapply(dat_classif_names, function(name) {
    train <- scale(dat[[name]][["train"]][, 1:2])
    list(train = as.matrix(train),
```

```
        test = as.matrix(scale(dat[[name]][["test"]][, 1:2],
                        center = attr(train, "scaled:center"),
                        scale = attr(train, "scaled:scale"))))
  })
)

# wydobycie wektorow y
y <- c(
  nlapply(dat_regr_names, function(name) {
    train <- scale(dat[[name]][["train"]][["y"]])
    list(train = as.matrix(train),
        test = as.matrix(scale(dat[[name]][["test"]][["y"]],
                        center = attr(train, "scaled:center"),
                        scale = attr(train, "scaled:scale"))))
  }),
  nlapply(dat_classif_names, function(name)
    list(train = as.matrix(dat[[name]][["train"]][, 3]),
        test = as.matrix(dat[[name]][["test"]][, 3]))
  )
)

# utworzenie one-hot-encodowanych zmiennych do klasyfikacji
y_enc <- nlapply(dat_classif_names, function(name)
  list(train = one_hot_encode_y(y[[name]][["train"]],
                                as.vector(unique(y[[name]][["train"]]))),
       test = one_hot_encode_y(y[[name]][["test"]],
                                as.vector(unique(y[[name]][["test"]])))))
```

Laboratorium 1

Cel: zbudowanie prostych sieci z algorytmem feedforward

Warto zaznaczyć na wstępie, że dane musiałem przeskalować, gdyż bez tego praktycznie nie dało się dopasować odpowiednich wartości. Skalowane były zbiory treningowe, a następnie na tej podstawie – zbiory testowe

Tworzenie architektury sieci

Dla każdego ze zbiorów przygotujemy po trzy sieci o różnych architekturach, które później będę uzupełniał wagami. Będą to następujące sieci:

- 1 warstwa ukryta (5 neuronów),
- 1 warstwa ukryta (10 neuronów),
- 2 warstwy ukryte (po 5 neuronów każda).

```
nets[["lab1"]] <- nlapply(dat_lab1_names, function(name) {
  list(
    small =
      neural_network(1) +
      hidden_layer(5, "sigmoid") +
      output_layer(1),
    medium =
      neural_network(1) +
      hidden_layer(10, "sigmoid") +
      output_layer(1),
    big =
      neural_network(1) +
      hidden_layer(5, "sigmoid") +
      hidden_layer(5, "sigmoid") +
      output_layer(1)
  )
```

```
})
```

Ustawianie wag sieci

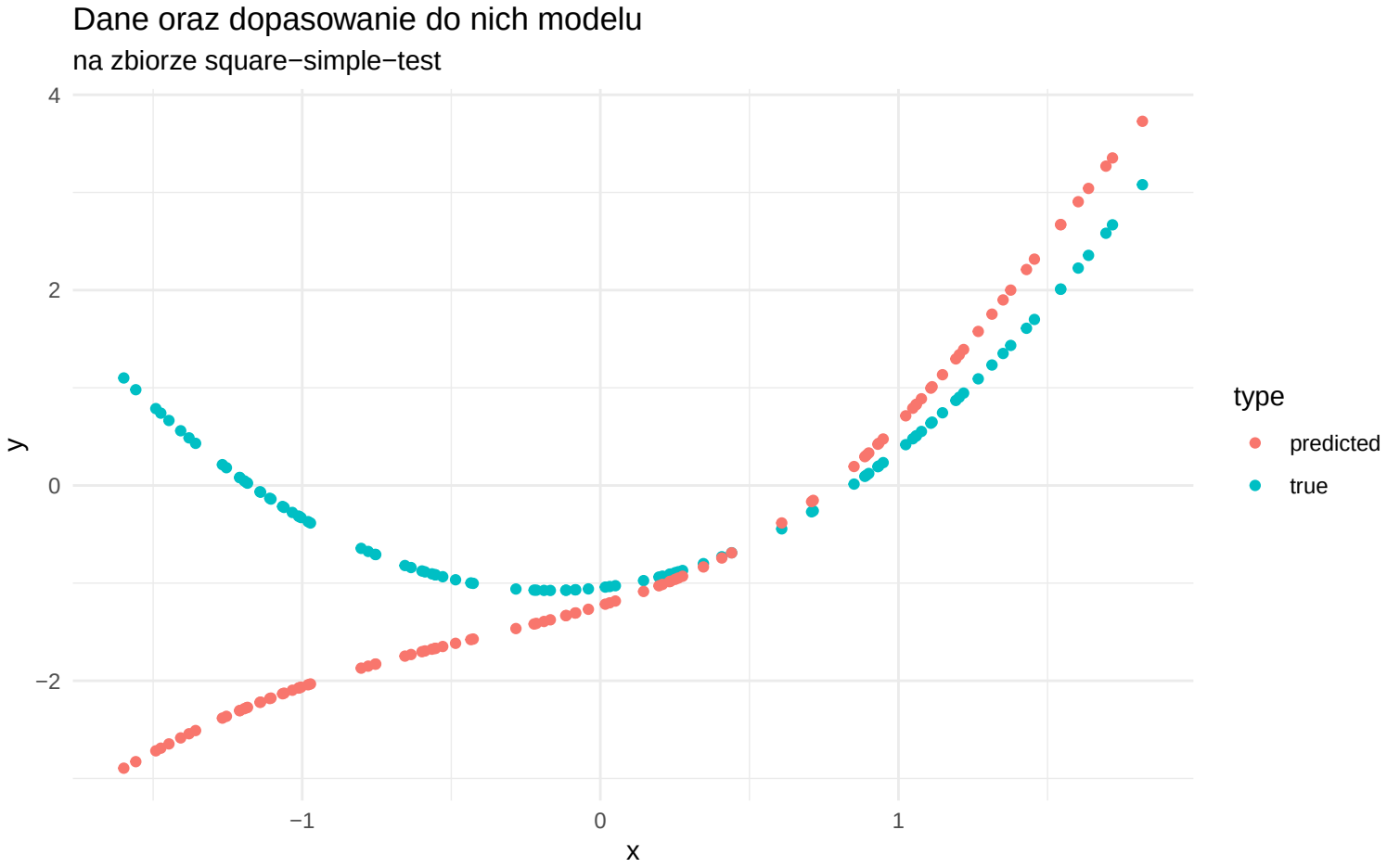
Wagi sieci dobrałem częściowo ręcznie, częściowo sugerując się tym, jak zostały w wyniku późniejszej rozbudowy wytrenowane.

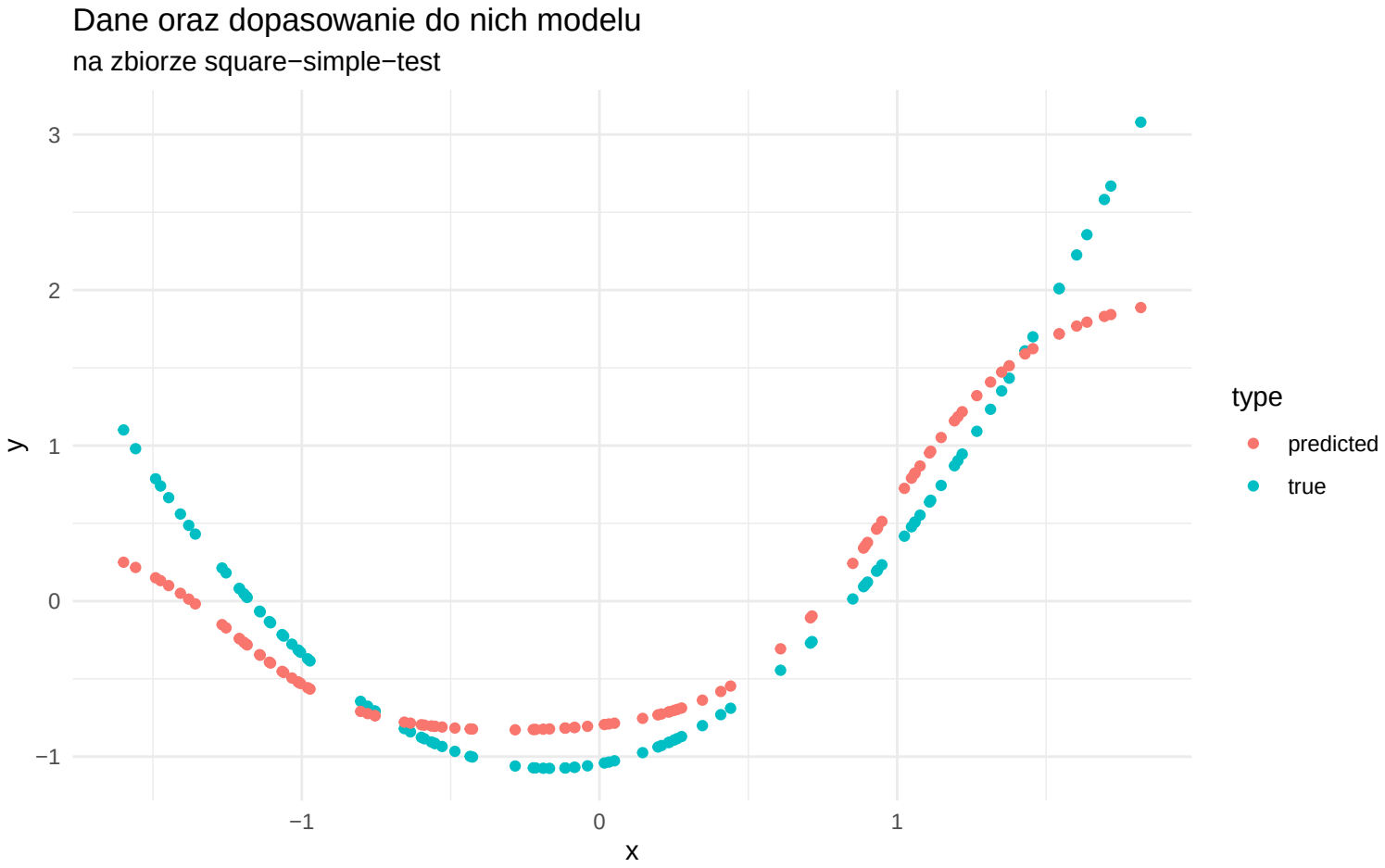
```
nets[["lab1"]][["square-simple"]][["small"]]$weights <- list(
  matrix(c(-4, -2, -7, -3, 8, 4, 3, -2, -9, -3), nrow = 2),
  matrix(c(-0.5, -7, 17, 7, -8, 16), nrow = 6))

nets[["lab1"]][["square-simple"]][["medium"]]$weights <- list(
  matrix(c(6, 5, 2, 0.05, 1.36, -0.2, 1, 0.9, -5.8, -4.6,
           1.5, 0.06, -0.5, 0.5, 3.8, -4, -0.2, -0.1, 3.9, -4), nrow = 2),
  matrix(c(0.5, -0.6, 1.1, 0, -0.2, 0.7, 1.6, 0.12, -1.3, -0.3, -1.5), nrow = 11))

nets[["lab1"]][["square-simple"]][["big"]]$weights <- list(
  matrix(c(-2.9, -1.7, -0.2, -0.4, -1.9, 1.3, -1.9, 1.1, 0.2, -0.4), nrow = 2),
  matrix(c(-0.2, -0.7, 0, -0.4, -2.2, 0.9, -1.5, 2.7, 0, 0.4, 1.9,
           0.1, 1, 0.6, 2.5, -2.8,-1.9, 1.8, 0.6, 0.3, 1.3, -1.6,
           0.1, 0, -2.1, 2.5, 0.6, 0.9, 0.8, 0.8), nrow = 6),
  matrix(c(2.5, -3.5, 3.9, -3.4, -1.2, 4.5), nrow = 6))

nets[["lab1"]][["steps-large"]][["small"]]$weights <- list(
  matrix(c(-0.5, 1, 1, -0.5, 0.3, 0.5, -0.5, 0.5, 0, -0.5), nrow = 2),
  matrix(c(-0.2, 2, -2, 1, 1.5, -0.75), nrow = 6))

nets[["lab1"]][["steps-large"]][["medium"]]$weights <- list(
  matrix(c(-0.8, 0.8, -0.1, -0.1, -1.9, 1.4, 1.2, -0.7, 0.6, -0.1, 1.8,
           0.1, 0.9, -0.1, -0.2, -0.9, 1.4, -0.9, -0.5, -0.8), nrow = 2),
  matrix(c(0.9, 1.4, -0.5, 0.7, -0.7, -0.2, 1, -0.3, -1.2, -0.3, -1.2), nrow = 11))

nets[["lab1"]][["steps-large"]][["big"]]$weights <- list(
  matrix(c(-4.8, -6.3, -1.2, -1.8, 2.4, -1.3, 0.1, -4, -0.9, -0.3), nrow = 2),
  matrix(c(-1.3, 0.1, -0.9, 3.2, 0.9, 1.4, 0.4, -1.1, 2.3, -3, -2.4, -0.7,
           -0.2, 2.2, -0.2, 0.5, -1.1, 0, 2.3, -7.3, -0.1, 1.4, -0.3, 0.6,
           -1.1, 1.4, 0.2, 0.4, 0.3, 0.5), nrow = 6),
  matrix(c(0.9, -2.9, 6.6, 0.9, 1.2, -0.2), nrow = 6))
```

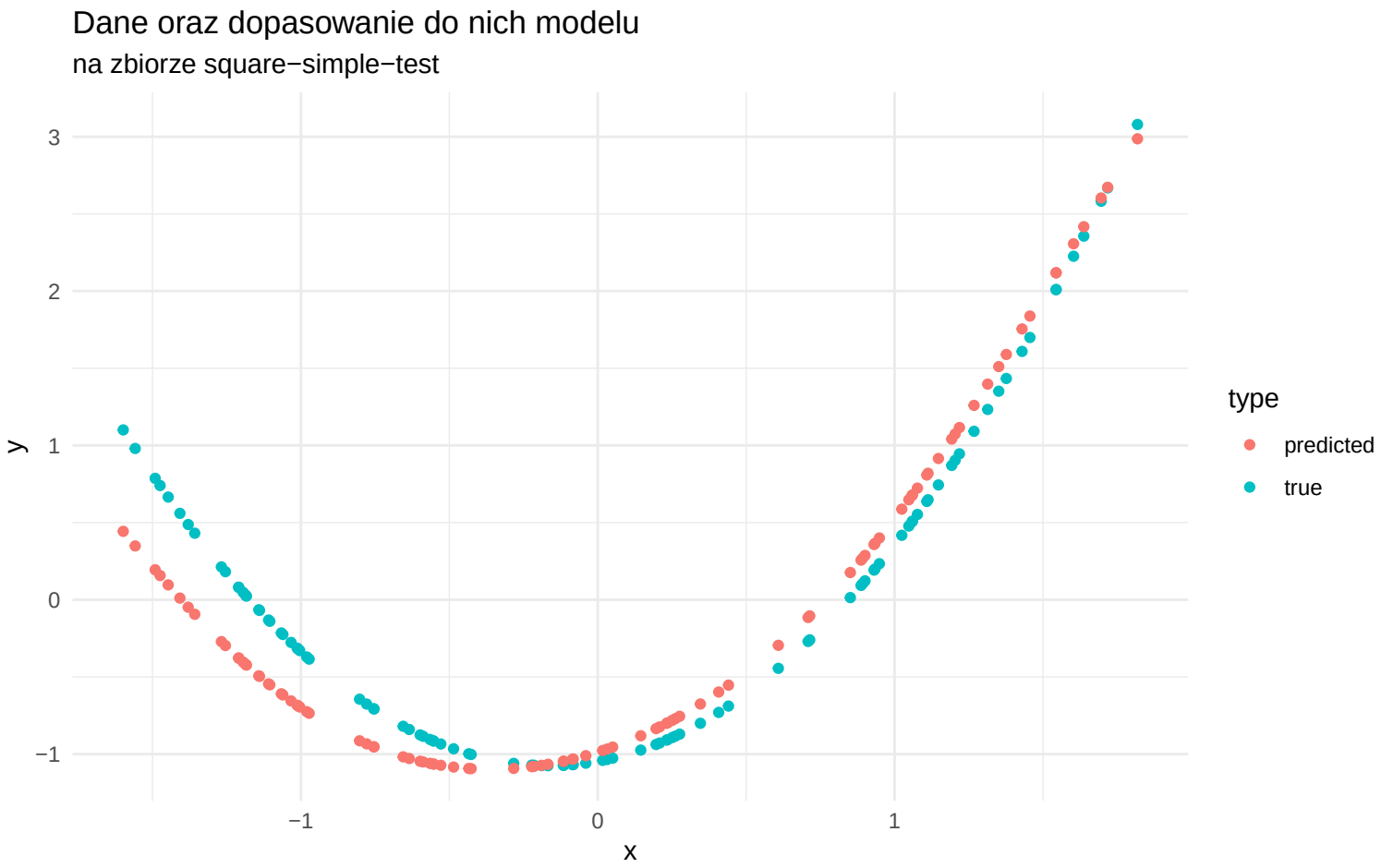
Sprawdzanie dopasowania

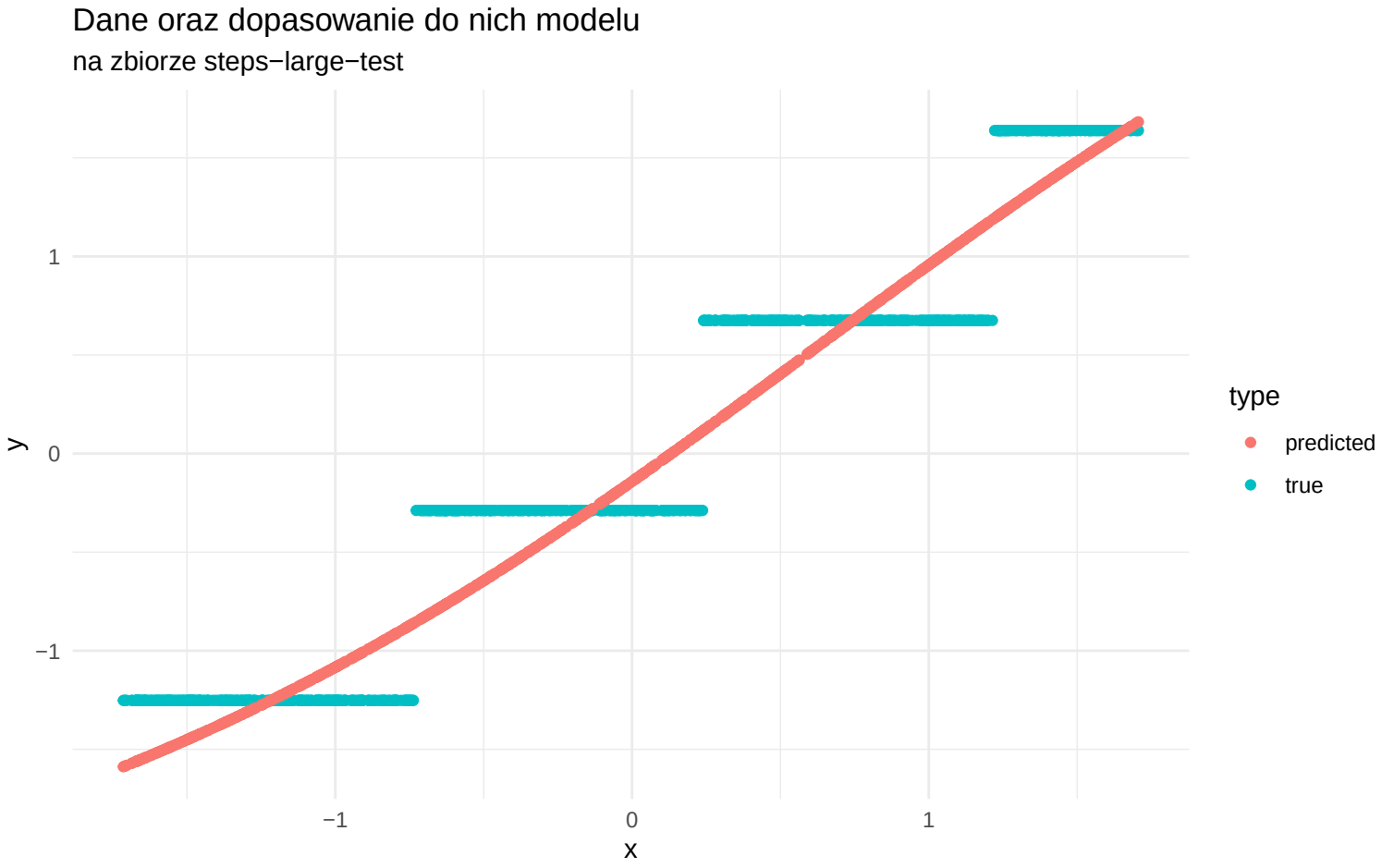
Teraz zaprezentuję, jak sieci się dopasowały do danych na zbiorze testowym.

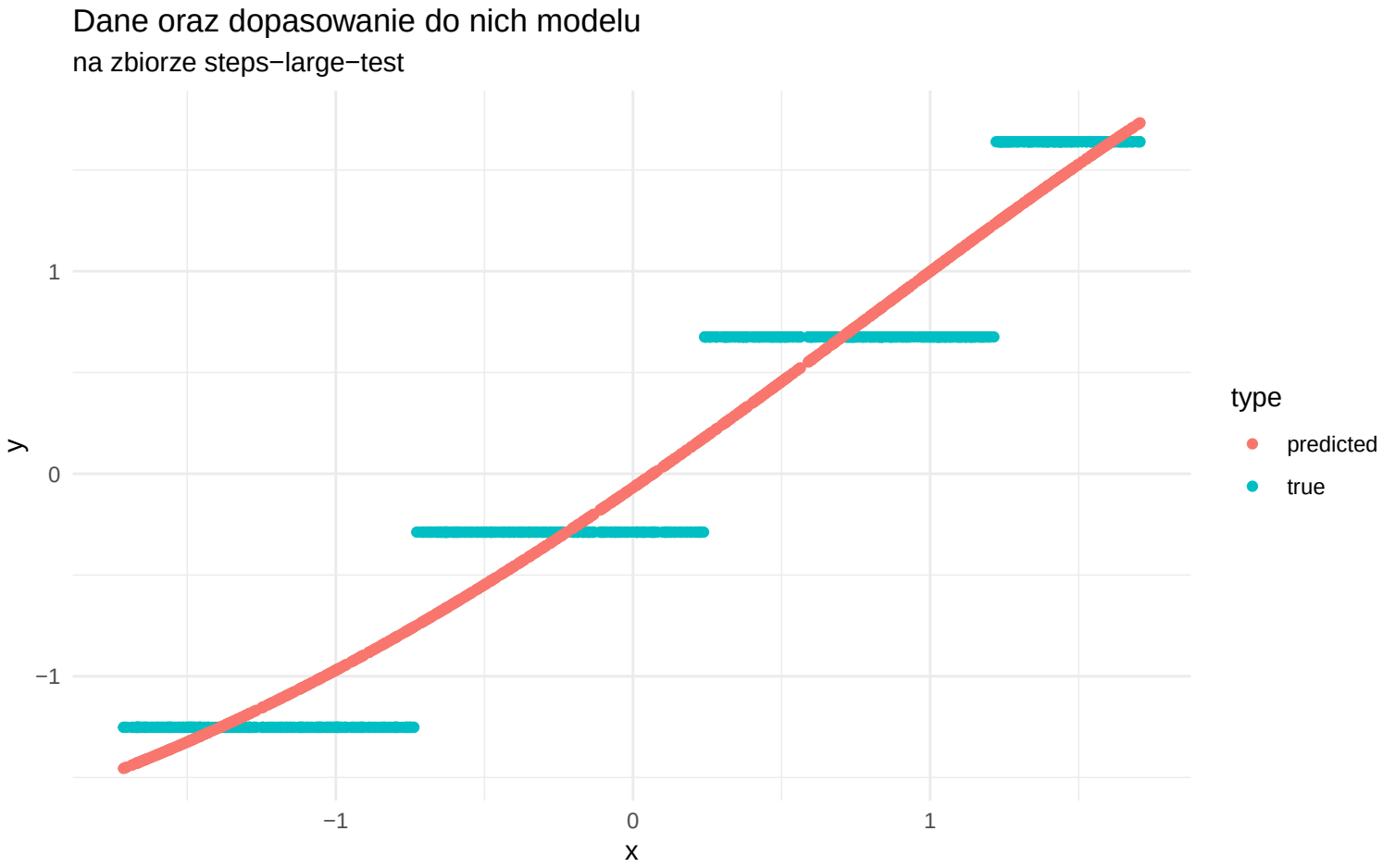
```
plots[["lab1"]] <- nlapply(dat_lab1_names, function(name)
  nlapply(c("small", "medium", "big"), function(architecture) {

    # wyliczenie wartosci na zbiorze testowym
    nets[["lab1"]][[name]][[architecture]] %>%
      feed_network(X[[name]][["test"]]) -> fit

    # wykonanie wykresow
    ggplot(data.frame(x = rep(X[[name]][["test"]], 2),
                      y = c(y[[name]][["test"]], fit),
                      type = rep(c("true", "predicted"),
                                 each = nrow(X[[name]][["test"]])))),
           aes(x = x, y = y, color = type)) +
      geom_point() +
      theme_minimal() +
      ggtitle("Dane oraz dopasowanie do nich modelu",
              paste0("na zbiorze ", name, "-test"))
  })
)
```

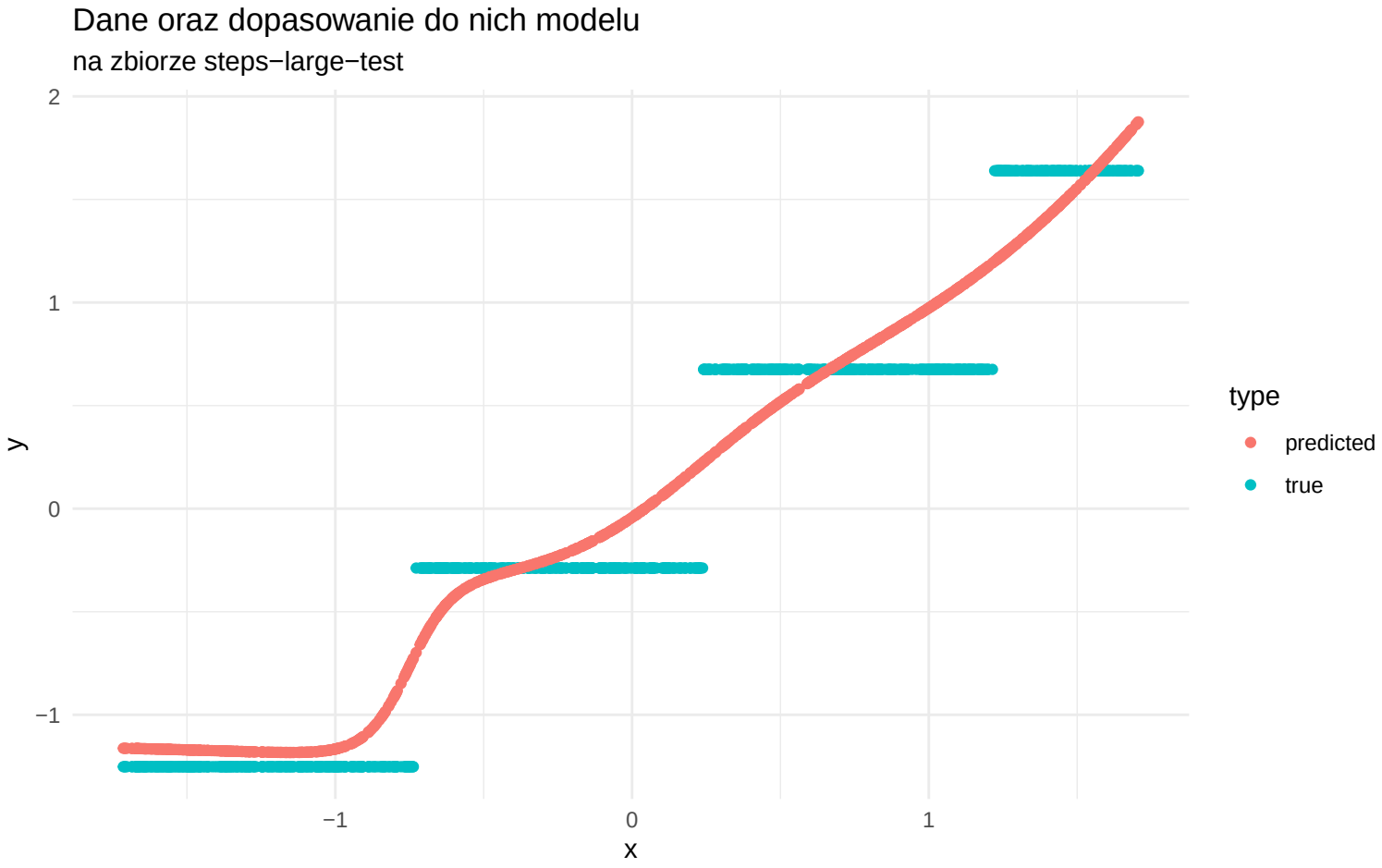

Dane oraz dopasowanie do nich modelu
na zbiorze square−simple−test
4
2
0
−2
y
−1
0
1
x
type
predicted
true


Dane oraz dopasowanie do nich modelu
na zbiorze square−simple−test
3
2
1
0
−1
y
−1
0
1
x
type
predicted
true


Dane oraz dopasowanie do nich modelu
na zbiorze square−simple−test
3
2
1
0
−1
y
−1
0
1
x
type
predicted
true


Dane oraz dopasowanie do nich modelu
na zbiorze steps−large−test
1
0
−1
y
−1
0
1
x
type
predicted
true


Dane oraz dopasowanie do nich modelu
na zbiorze steps−large−test
1
0
−1
y
−1
0
1
x
type
predicted
true


Dane oraz dopasowanie do nich modelu
na zbiorze steps−large−test
2
1
0
−1
y
−1
0
1
x
type
predicted
true

Jak możemy zobaczyć po wykresach dopasowania, im bardziej złożona sieć, tym lepiej może się ona dopasować do danych. Co prawda trenujemy je na zbiorach treningowych, a rysujemy wykresy na testowych, jednak zbiory te zostały sztucznie wygenerowane i mają prawie identyczne rozkłady, więc kwestia przeuczenia nie wchodzi tutaj w grę.

Laboratorium 2

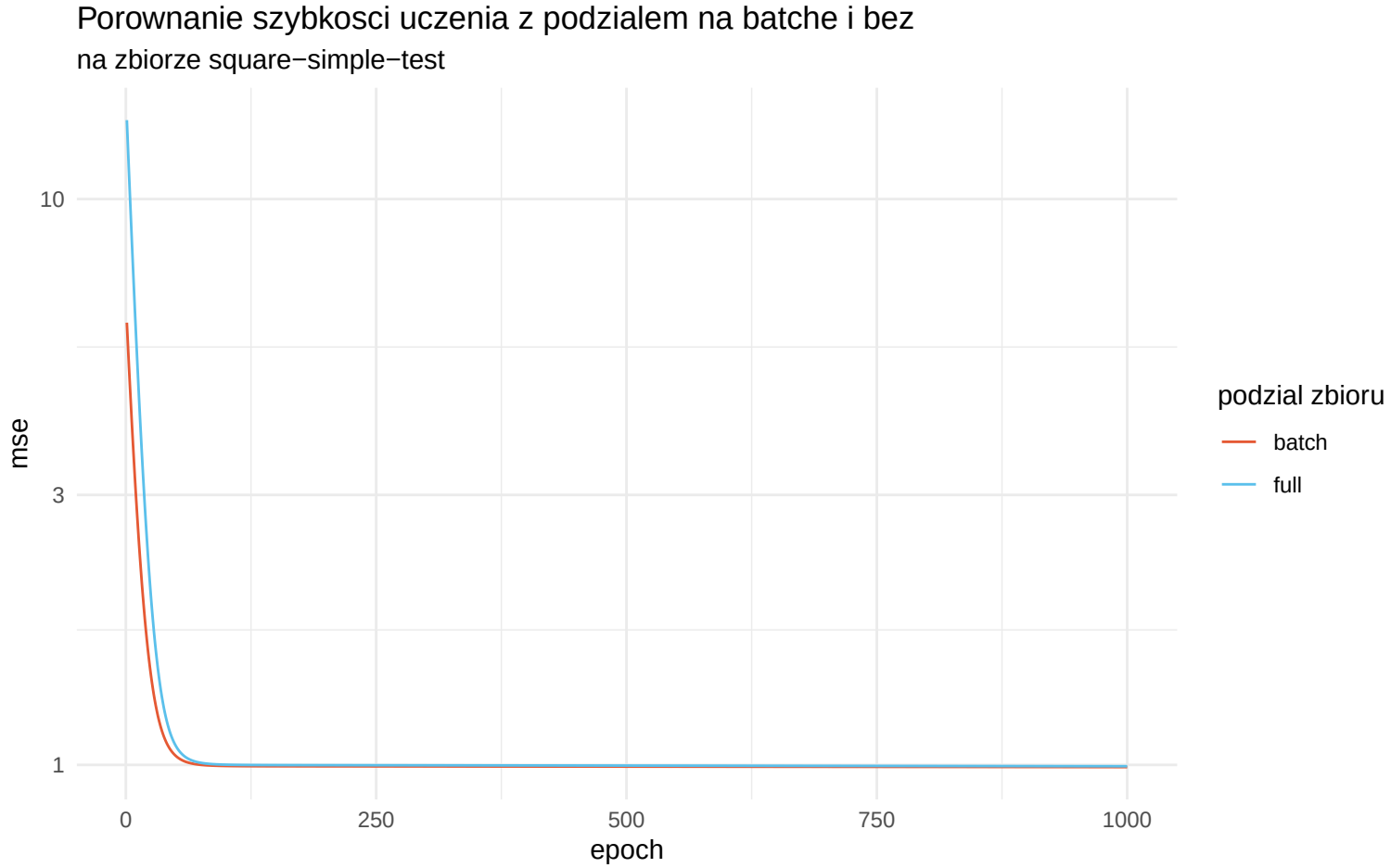
Cel: implementacja algorytmu backpropagation, metod inicjalizacji wag i batch

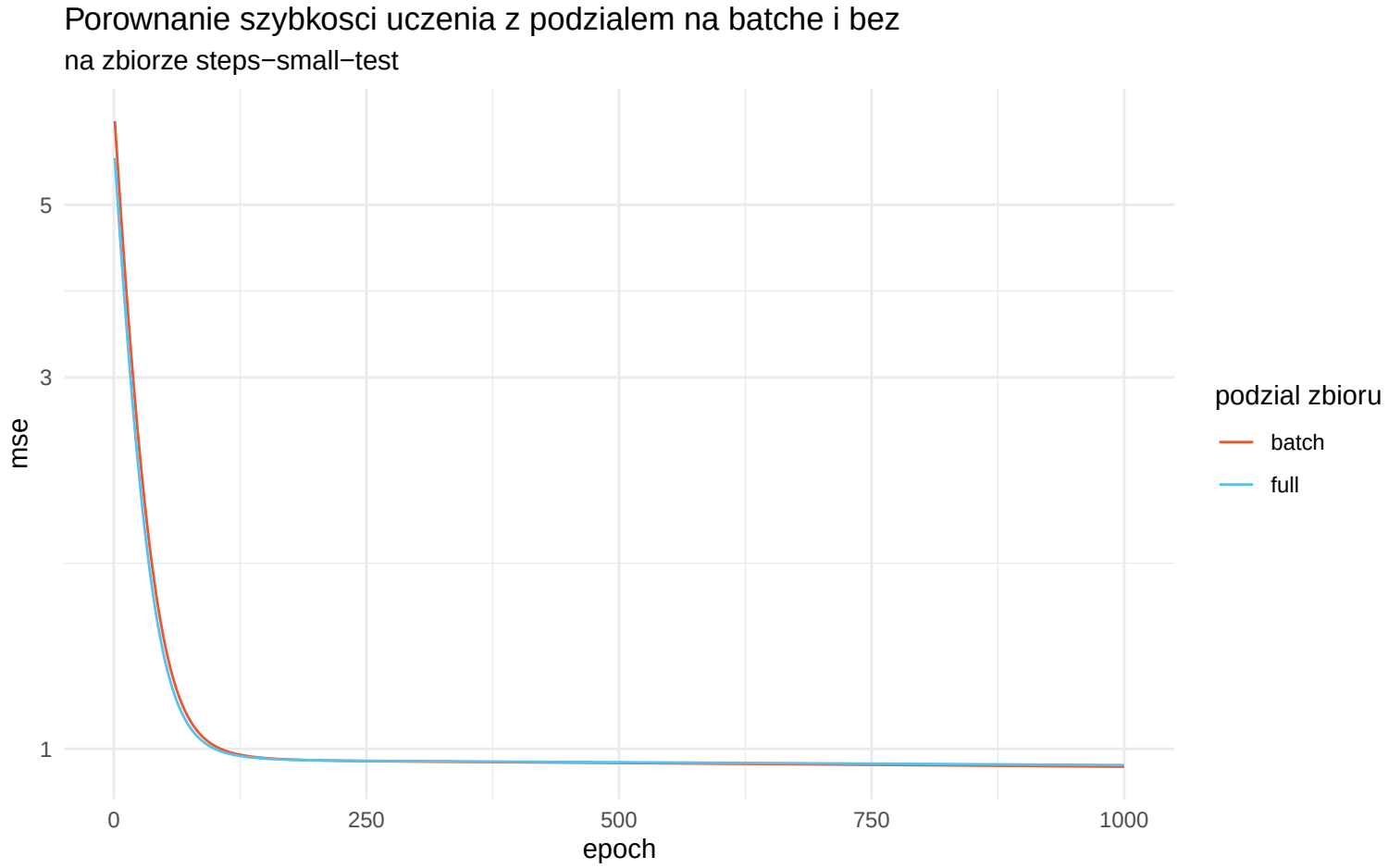
Porównywanie tempa uczenia przy uczeniu na całym zbiorze i na fragmentach (batch)

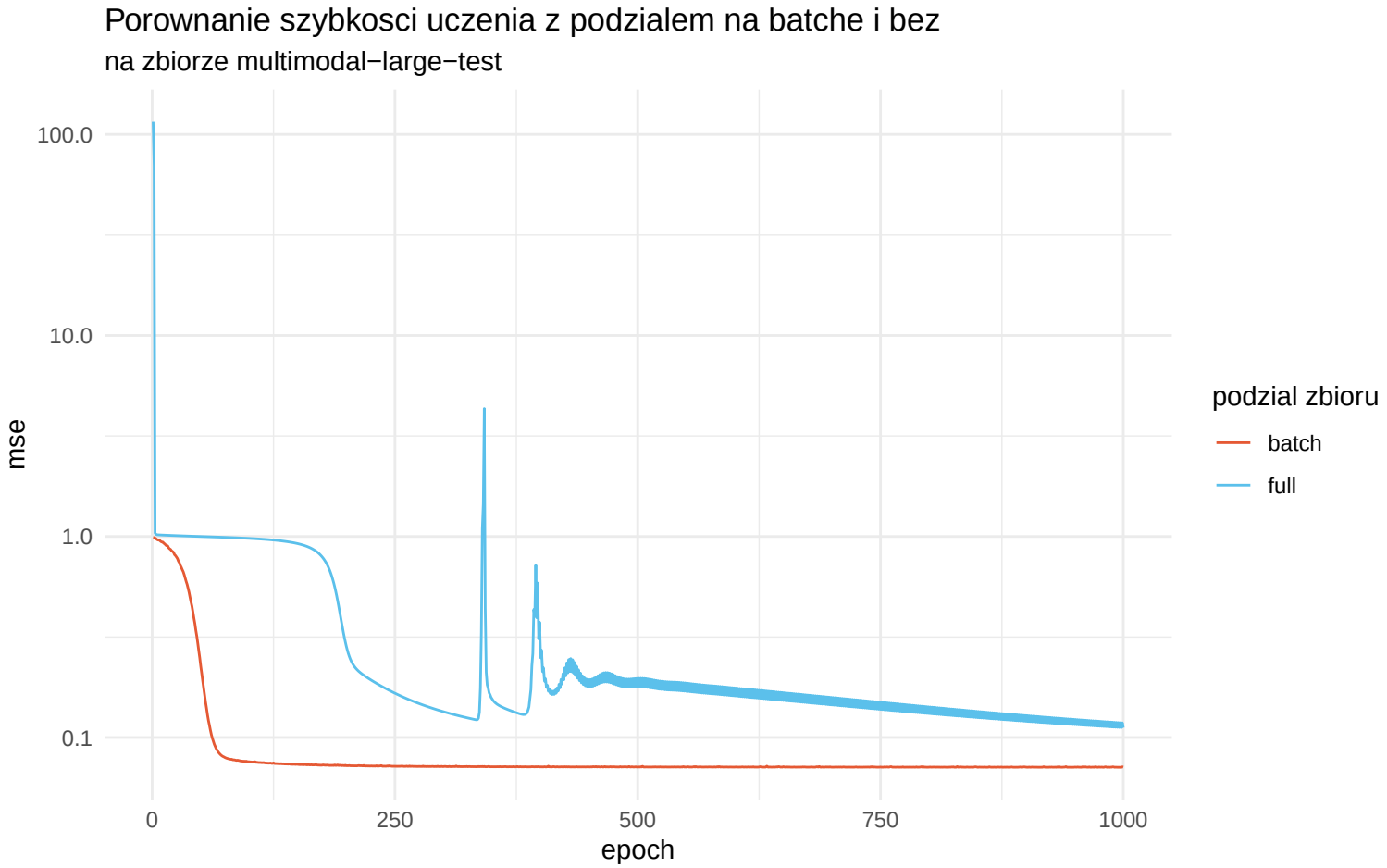
W celu porównania tempa uczenia się sieci w przypadku aktualizacji gradientu po całym zbiorze i po jego częściach (tzw. batching), zbadamy to tempo na dwuwarstwowej sieci z poprzedniego eksperymentu, z wylosowanymi wagami z rozkładu normalnego, trenując ją przez 100 epok.

```
nets[["lab2"]] <- list()
nets[["lab2"]][["batch"]] <- nlapply(dat_lab2_names, function(name) {
  net <- neural_network(1) +                  # tworzymy siec
         hidden_layer(5, "sigmoid") +
         hidden_layer(5, "sigmoid") +
         output_layer(1)
  list(full = net %>%
         randomize_weights_runif() %>%              # losujemy wagi wg rozkl. jednost.
         train_network_sgd(X[[name]][["train"]],    # trenujemy bez batchowania
                           y[[name]][["train"]],
                           num_epochs = 1000,
                           eta = 1e-4,
                           verbose = FALSE)
         ,
       batch = net %>%
         randomize_weights_runif() %>%              # losujemy wagi wg rozkl. jednost.
         train_network_sgd(X[[name]][["train"]],    # trenujemy z batchami
                           y[[name]][["train"]],
                           batch_size = ceiling(nrow(X[[name]][["train"]]) / 5),
                           num_epochs = 1000,
                           eta = 1e-4,
                           verbose = FALSE)
  )
})
```

```
plots[["lab2"]] <- list()
plots[["lab2"]][["batch"]] <- nlapply(dat_lab2_names, function(name){
  data.frame(mse =
               c(nets[["lab2"]][["batch"]][[name]][["full"]]$training_history$training,
                 nets[["lab2"]][["batch"]][[name]][["batch"]]$training_history$training),
             epoch = rep(1:1000, 2),
             type = rep(c("full", "batch"), each = 1000)) %>%
    ggplot(aes(x = epoch, y = mse, color = type)) +
    geom_line() +
    legendary_palette() +
    scale_y_log10() +
    ggtitle("Porownanie szybkosci uczenia z podzialem na batche i bez",
            paste0("na zbiorze ", name, "-test")) +
    labs(color = "podzial zbioru") +
    theme_minimal()

})
```


Porownanie szybkosci uczenia z podzialem na batche i bez
na zbiorze square−simple−test
mse
10
3
1
0
250
500
750
1000
epoch
podzial zbioru
batch
full


Porownanie szybkosci uczenia z podzialem na batche i bez
na zbiorze steps−small−test
mse
5
3
1
0
250
500
750
1000
epoch
podzial zbioru
batch
full

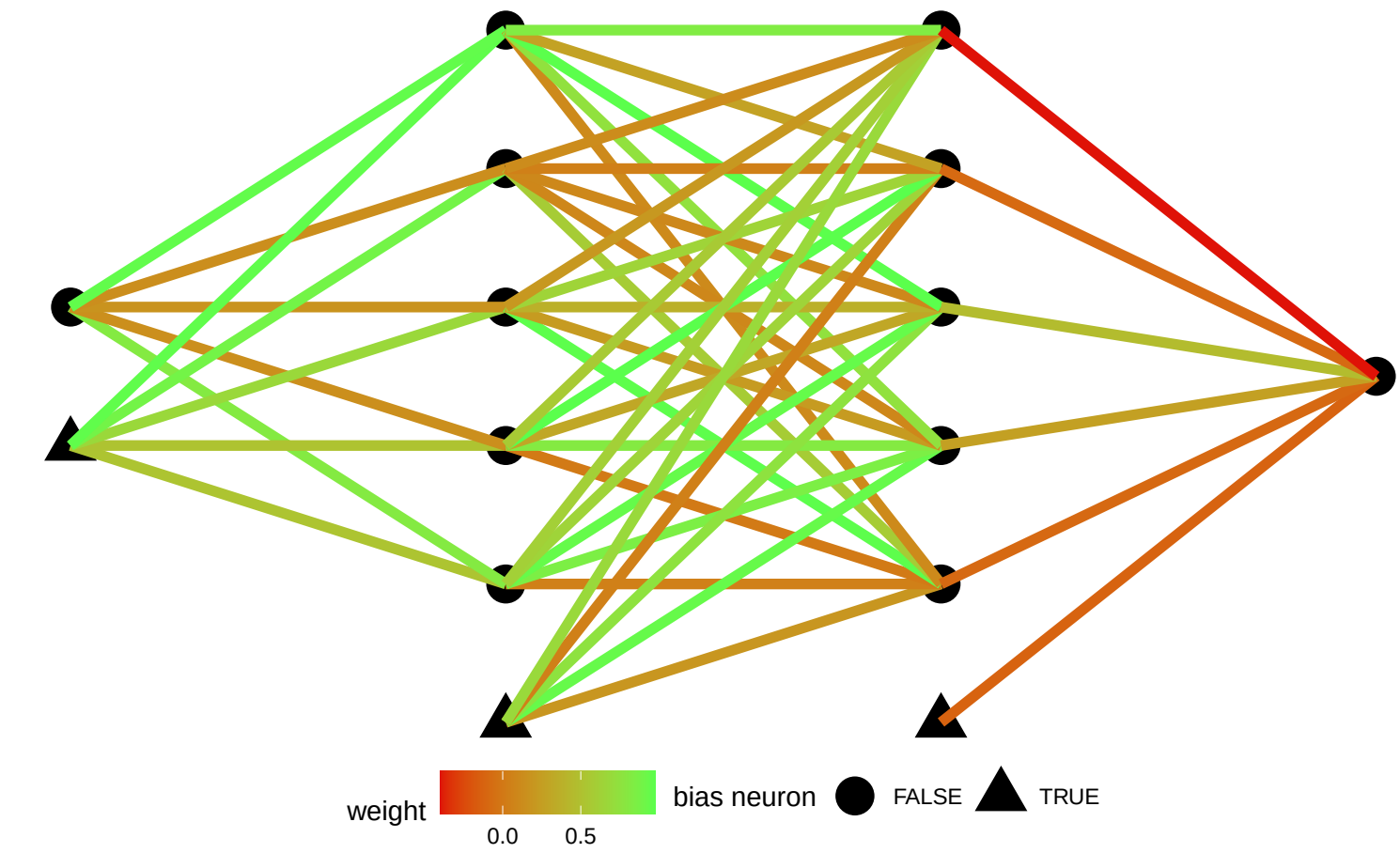


Widzimy, że choć użycie batchy przyspiesza proces uczenia (a przynajmniej go nie spowalnia) i zdecydowanie nieco go "stabilizuje"

Wizualizacja wag

Możemy zwizualizować sobie wagi modelu:

```
plots[["lab2"]][["weigths"]] <- plot_weigths(nets[["lab2"]][["batch"]][["square-simple"]][["batch"]])

plots[["lab2"]][["weigths"]]
```



Porównanie metod losowania inicjalnych wag

Teraz porównamy trzy sposoby losowania wag:

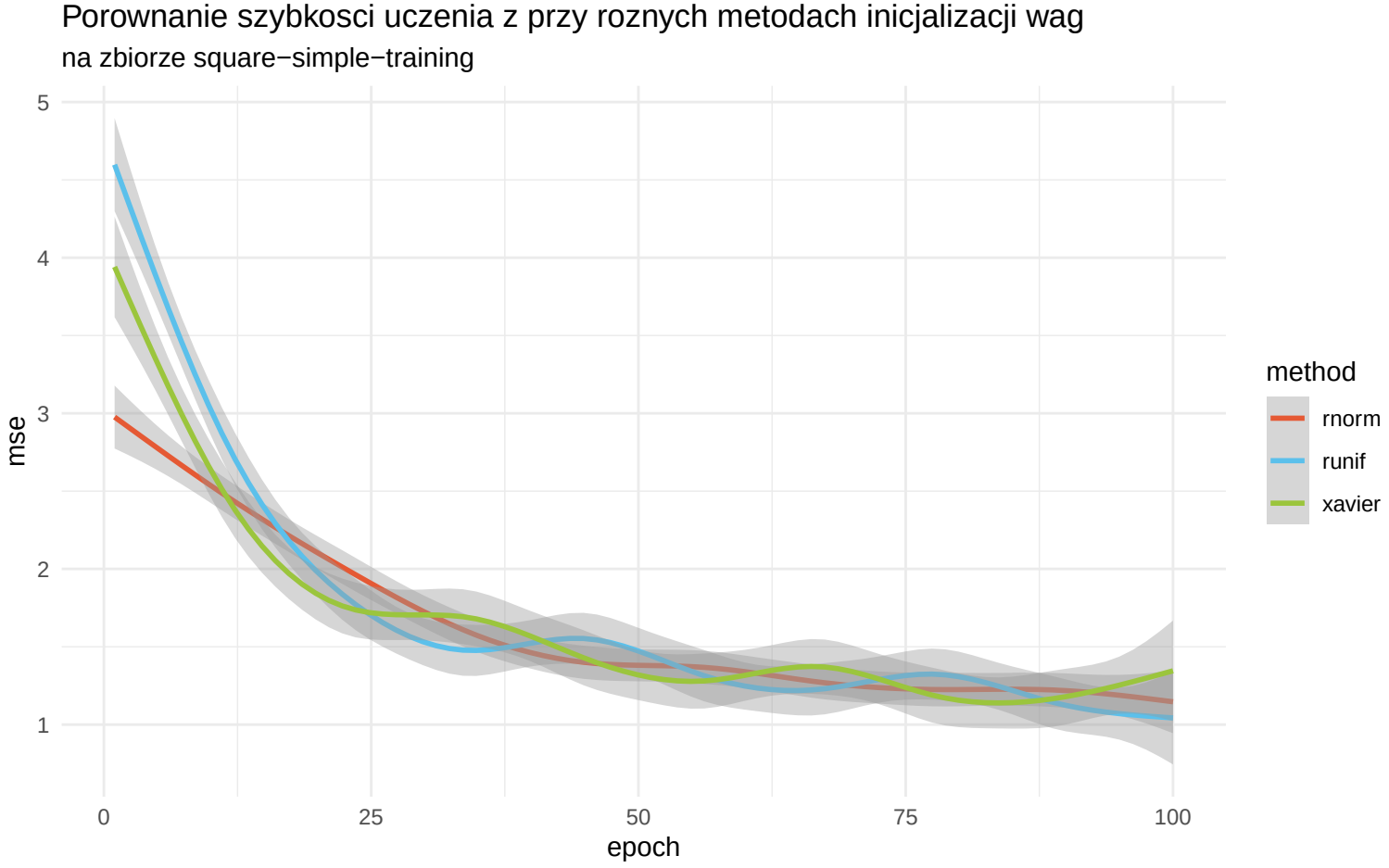
- losowanie z rozkładu normalnego o średniej 0 z odchyleniem 1,
- losowanie z rozkładu jednostajnego na przedziale $[0, 1]$,
- metoda Xavier

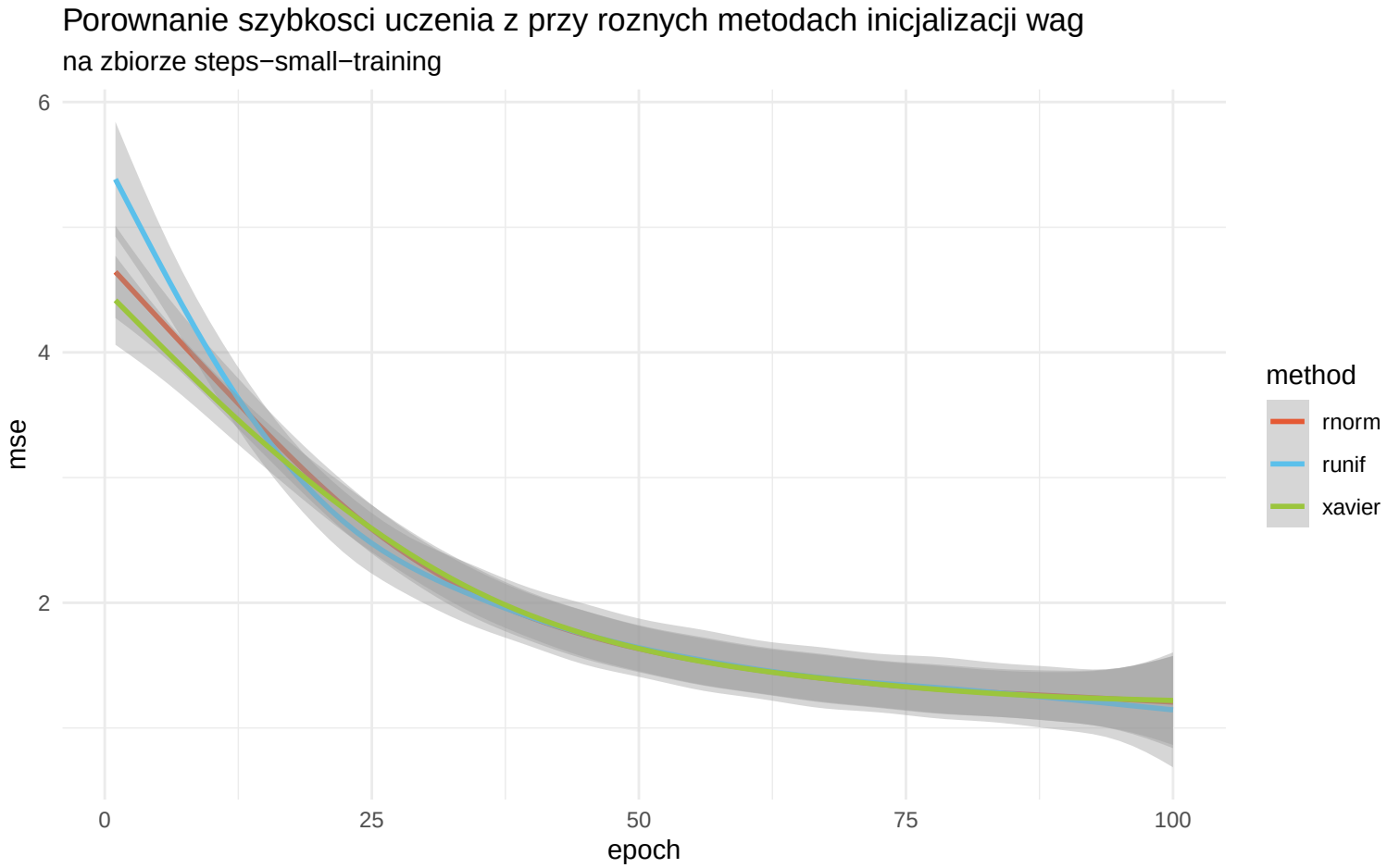
Architektura sieci pozostanie jak wcześniej; aby móc nieco uogólnić wynik, będziemy losować po 10 razy każdą metodą na każdym zbiorze.

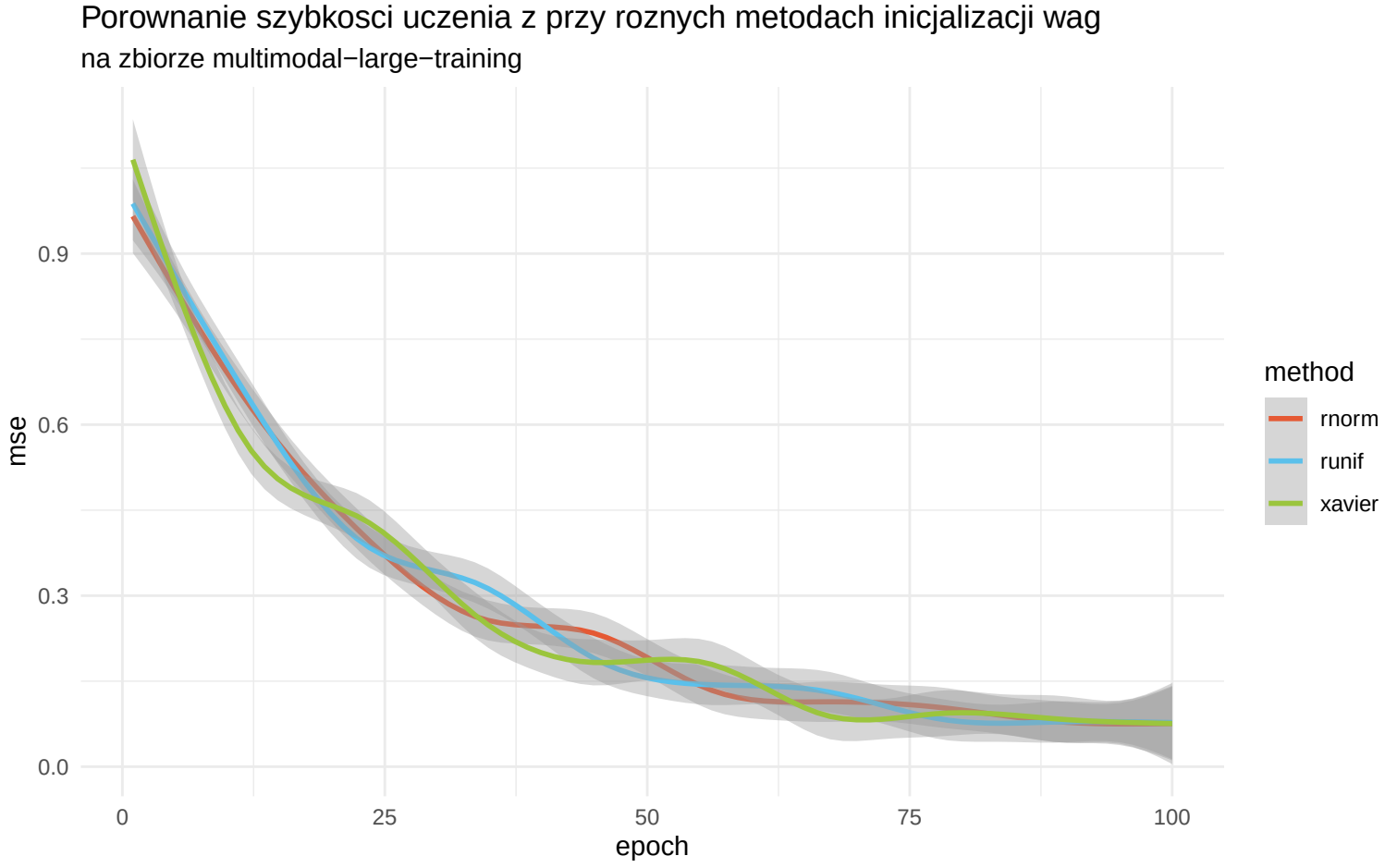
```
r_methods <- list(
  runif = randomize_weights_runif,
  rnorm = randomize_weights_rnorm,
  xavier = randomize_weights_xavier
)

nets[["lab2"]][["init"]] <- nlapply(dat_lab2_names, function(name) {
  net <- neural_network(1) +                    # tworzymy siec
    hidden_layer(5, "sigmoid") +
    hidden_layer(5, "sigmoid") +
    output_layer(1)

  nlapply(r_methods, function(method)
    nlapply(1:10, function(iteration)
      net %>%
        method() %>%                                  # losujemy wagi wg wybranej metody
        train_network_sgd(X[[name]][["train"]],     # trenujemy z batchowaniem
                          y[[name]][["train"]],
                          num_epochs = 100,
                          eta = 1e-4,
                          batch_size = ceiling(nrow(X[[name]][["train"]]) / 5),
```

```
                          verbose = FALSE)
    )
  )
})
```

```
plots[["lab2"]][["init"]] <- nlapply(dat_lab2_names, function(name){
  data.frame(
    mse = unlist(lapply(names(r_methods), function(method)
      lapply(nets[["lab2"]][["init"]][[name]][[method]], function(net)
        net$training_history$training)
    )),
    epoch = rep(1:100, 30),
    method = rep(names(r_methods), each = 1000),
    iteration = rep(rep(1:10, each = 100), 3)) %>%
    ggplot(aes(x = epoch, y = mse, color = method)) +
    geom_smooth() +
    legendary_palette() +
    ggtitle("Porownanie szybkosci uczenia z przy roznych metodach inicjalizacji wag",
            paste0("na zbiorze ", name, "-training")) +
    theme_minimal()

})
```







Jak widzimy, uśredniając po iteracjach, nie ma aż tak drastycznych różnic pomiędzy rozkładem normalnym wokół 0 a metodą

Xavier, jednak obie są wyraźnie lepsze niż rozkład jednostajny na przedziale $[0, 1]$.

Laboratorium 3

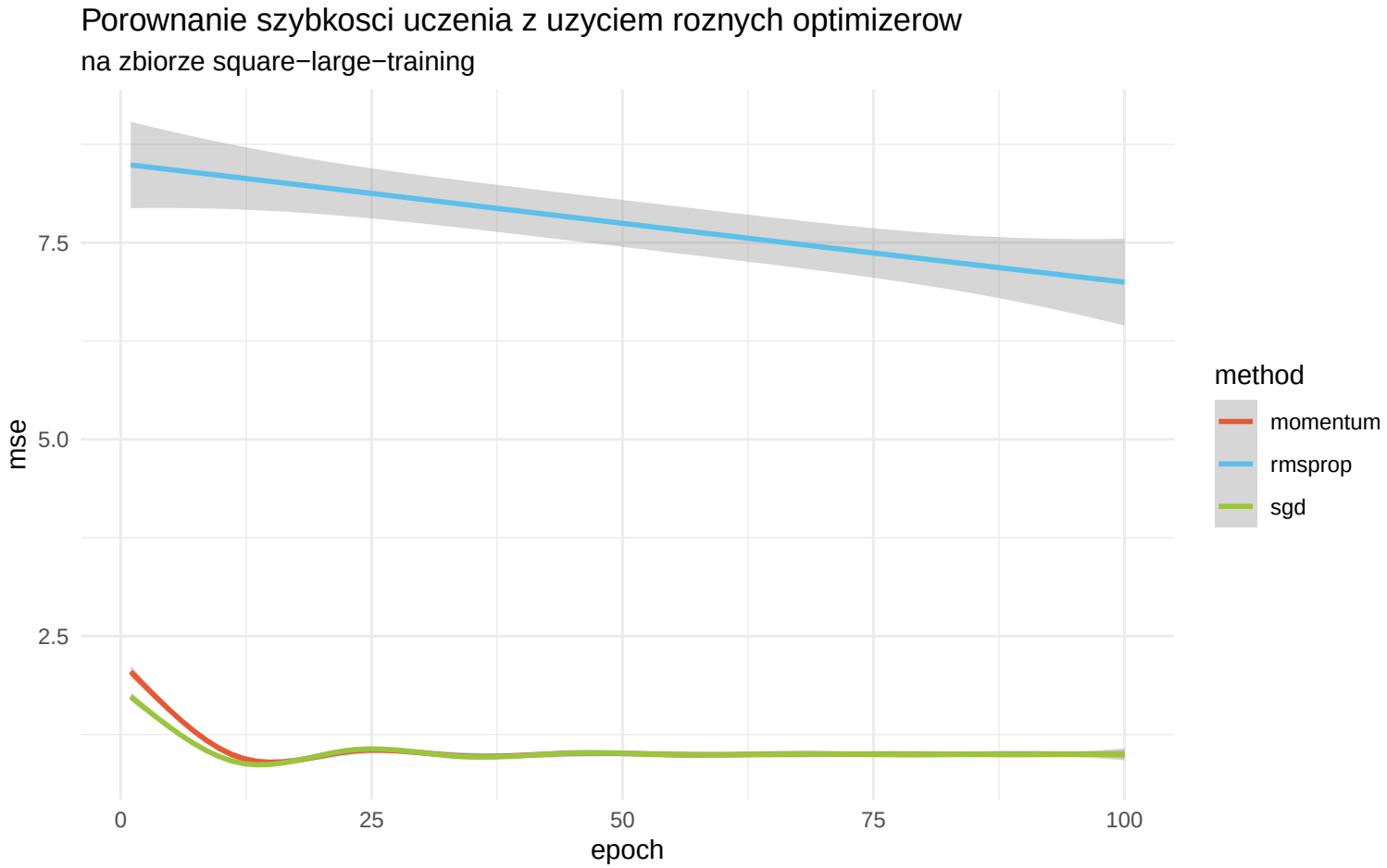
Cel: implementacja momentum i RMSprop oraz porównanie tych metod ze zwykłym sgd

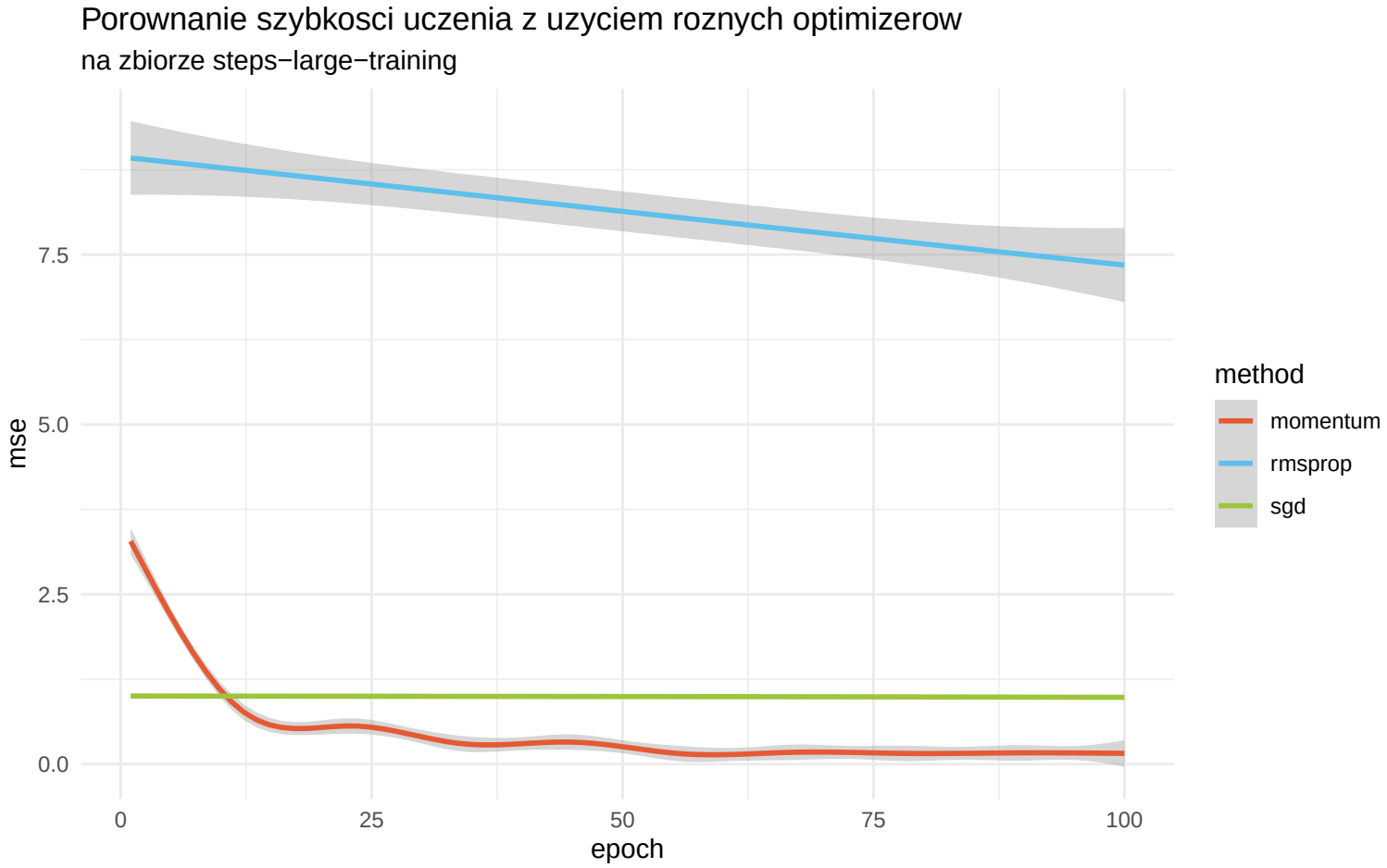
W tej części będziemy badać dwie metody mające w założeniu przyspieszyć osiąganie globalnego minimum. Będą to RMSprop i momentum. Zbadamy je na sieci z trzema warstwami po pięć neuronów na kilku zbiorach – na każdym dziesięć razy uruchomimy trening przez 100 epok,

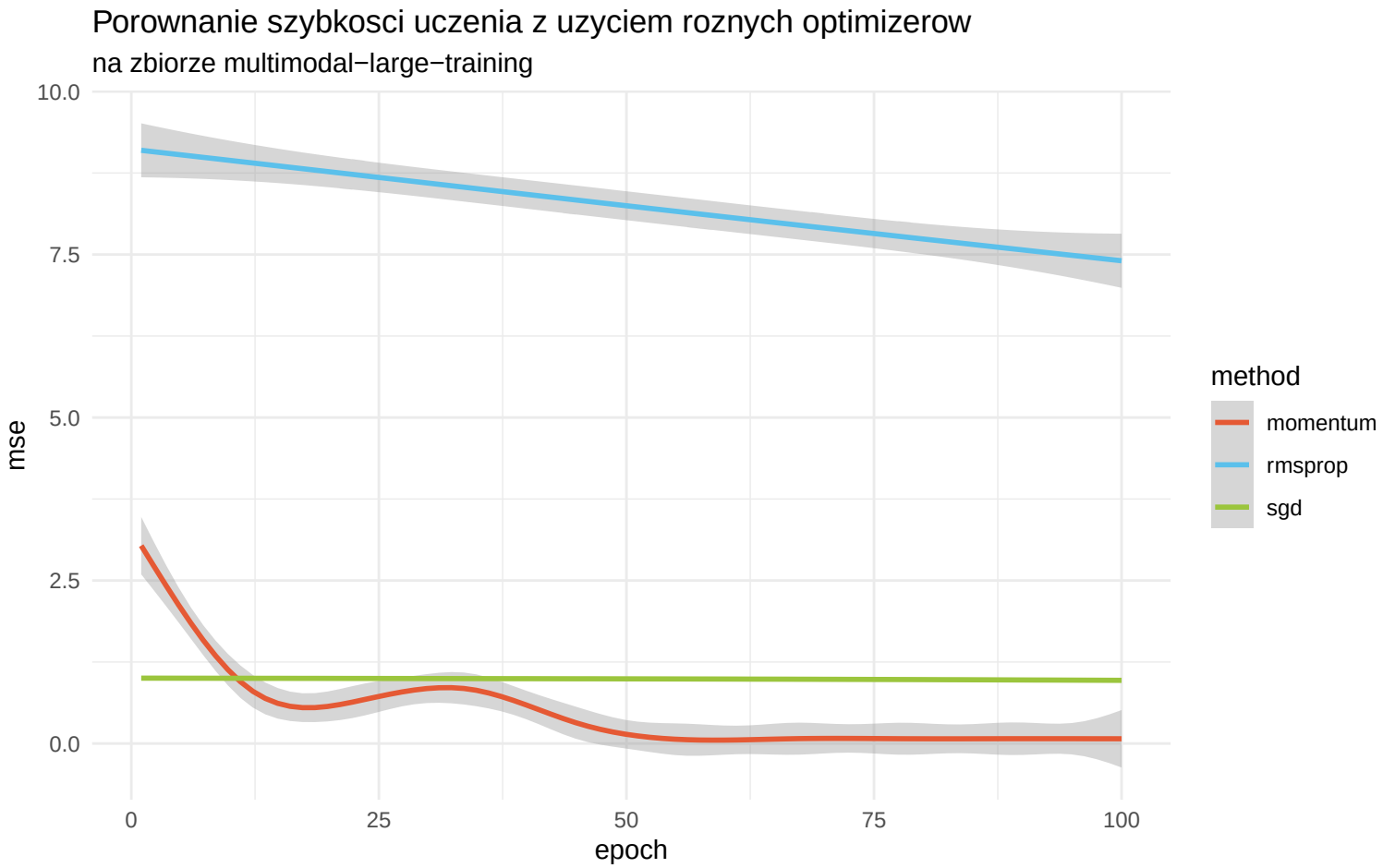
Porównanie metod optymalizacji zbieżności

```
t_methods <- list(
  sgd = train_network_sgd,
  momentum = train_network_momentum,
  rmsprop = train_network_rmsprop
)

nets[["lab3"]] <- nlapply(dat_lab3_names, function(name) {
  net <- neural_network(1) +                  # tworzymy siec
    hidden_layer(5, "sigmoid") +
    hidden_layer(5, "sigmoid") +
    hidden_layer(5, "sigmoid") +
    output_layer(1)

  nlapply(t_methods, function(method)
    nlapply(1:10, function(iteration)
      net %>%
        randomize_weights_runif() %>%              # losujemy wagi z rozkladu runif
        method(X[[name]][["train"]],    # trenujemy z batchowaniem
               y[[name]][["train"]],
               num_epochs = 100,
               eta = 1e-4,
               batch_size = ceiling(nrow(X[[name]][["train"]]) / 5),
               verbose = FALSE)
    )
  )
})
```

```
plots[["lab3"]] <- nlapply(dat_lab3_names, function(name){
  data.frame(
    mse = unlist(lapply(names(t_methods), function(method)
      lapply(nets[["lab3"]][[name]][[method]], function(net)
        net$training_history$training)
    )),
    epoch = rep(1:100, 30),
    method = rep(names(t_methods), each = 1000),
    iteration = rep(rep(1:10, each = 100), 3)) %>%
    ggplot(aes(x = epoch, y = mse, color = method)) +
    geom_smooth() +
    legendary_palette() +
    ggtitle("Porownanie szybkosci uczenia z uzyciem roznych optimizerow",
            paste0("na zbiorze ", name, "-training")) +
    theme_minimal()

})
```


Porownanie szybkosci uczenia z uzyciem roznych optimizerow
na zbiorze square−large−training
mse
7.5
5.0
2.5
0
25
50
75
100
epoch
method
momentum
rmsprop
sgd


Porownanie szybkosci uczenia z uzyciem roznych optimizerow
na zbiorze steps−large−training
mse
7.5
5.0
2.5
0.0
0
25
50
75
100
epoch
method
momentum
rmsprop
sgd



Jak widzimy, metoda rmsprop spada bardzo jednostajnie, jednak przy domyślnych parametrach nie jest szybka. Metoda momentum natomiast charakteryzuje się dużo większą szybkością niż sgd i nie wpada tak łatwo w lokalne optima.

Laboratorium 4

Cel: zbadanie różnych wyjściowych funkcji aktywacji na zadaniu klasyfikacji

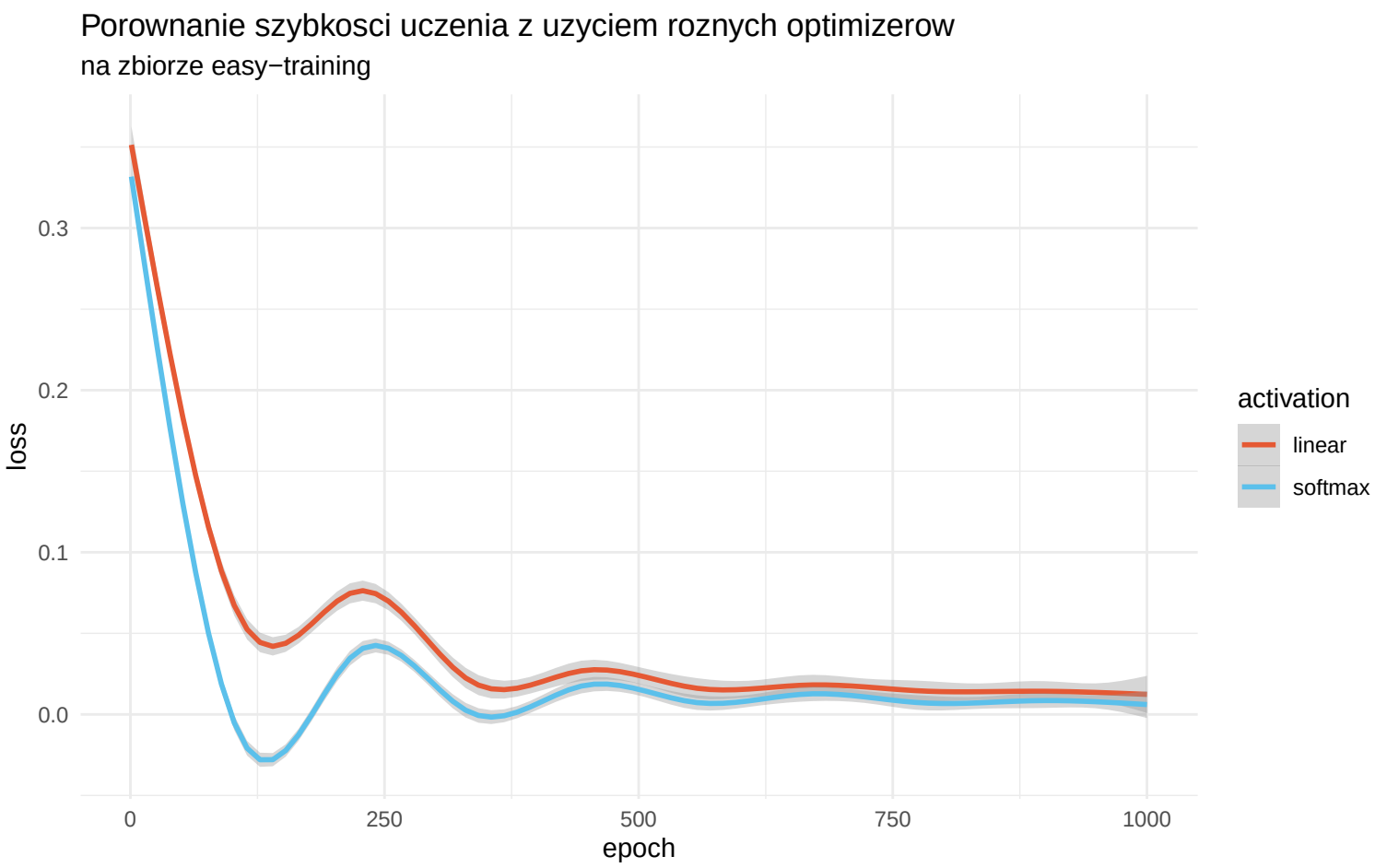
Porównanie tempa zbieżności przy różnych wyjściowych funkcjach aktywacji

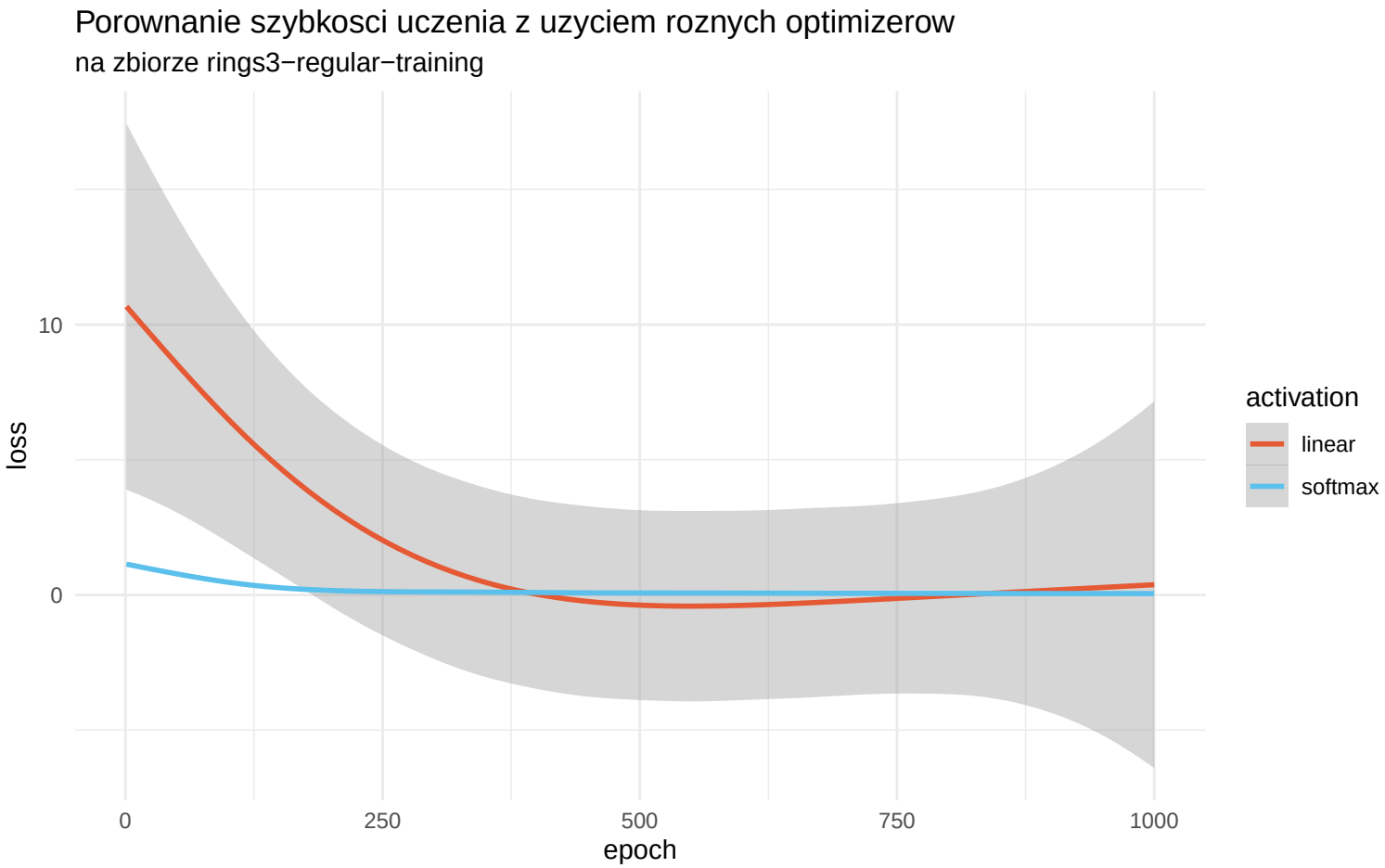
Porównamy teraz tempo uczenia przy zadaniu klasyfikacji z użyciem różnych funkcji aktywacji na ostatniej warstwie – liniowej oraz softmax. Zaznaczmy, że przy softmax używamy ponadto funkcji straty crossentropy zamiast mse. Eksperymentu dokonamy przez pięciokrotne wytrenowanie dwóch sieci na każdym zbiorze.

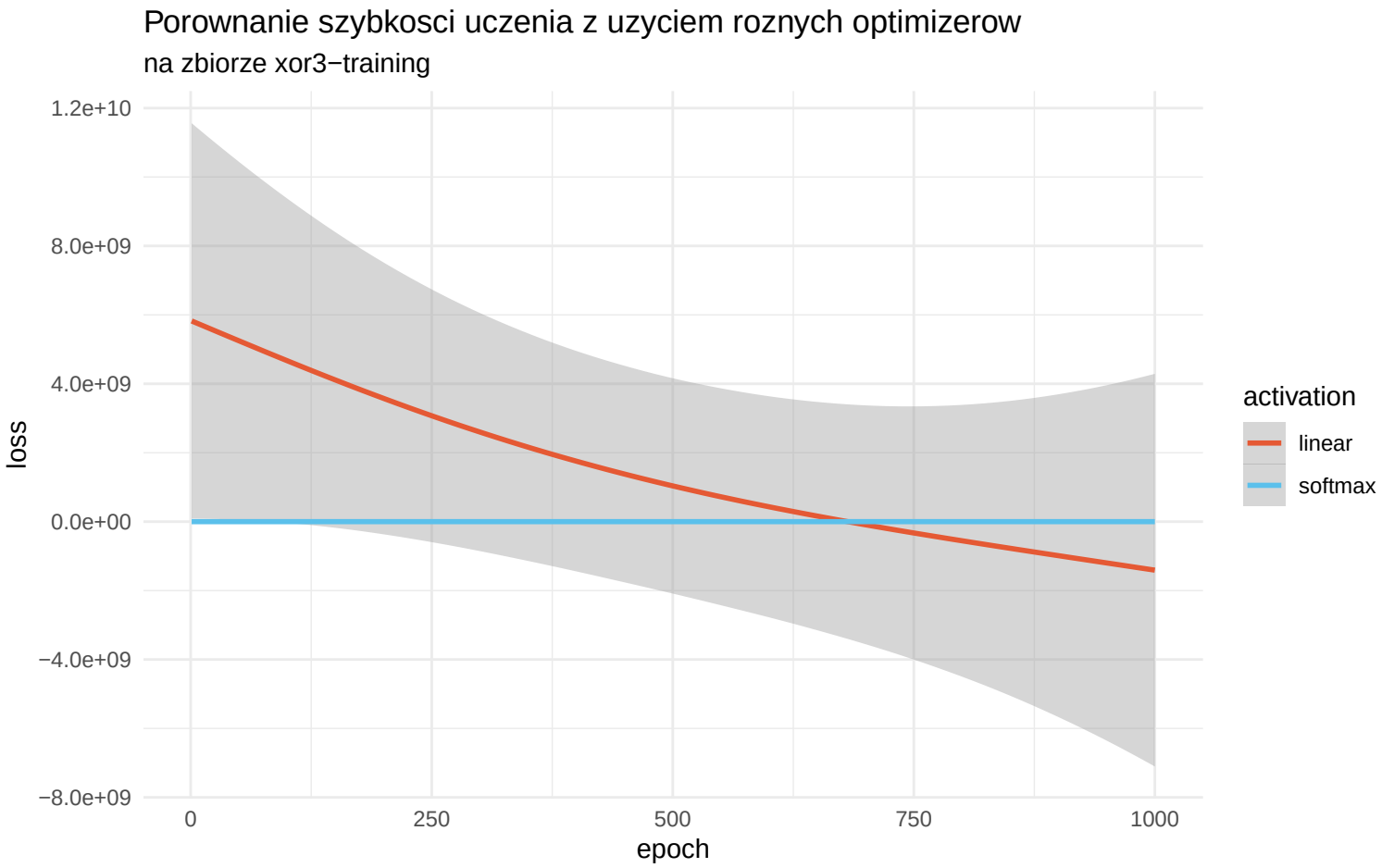
```
nets[["lab4"]] <- list()
activations <- c("linear", "softmax")

nets[["lab4"]][["classif"]] <- nlapply(dat_lab4_names, function(name)
  nlapply(activations, function(activation) {
    net <- neural_network(2) +                  # tworzymy siec
      hidden_layer(30, "sigmoid") +
      hidden_layer(30, "sigmoid") +
      hidden_layer(30, "sigmoid") +
      output_layer(ncol(y_enc[[name]][["train"]]), activation = activation)
    nlapply(1:5, function(iteration)
      net %>%
        randomize_weights_xavier() %>%   # losujemy wagi wg metody Xavier
        train_network_momentum(X[[name]][["train"]],    # trenujemy z batchowaniem
                               y_enc[[name]][["train"]],
                               num_epochs = 1000,
                               eta = 3e-3,
                               batch_size = 100,
```

```
                              verbose = FALSE,
                              loss = if (activation == "softmax") "crossentropy" else "mse")
    )
  })
)
```

```
plots[["lab4"]] <- list()
plots[["lab4"]][["classif"]] <- nlapply(dat_lab4_names, function(name){
  data.frame(
    loss = unlist(lapply(activations, function(activation)
      lapply(nets[["lab4"]][["classif"]][[name]][[activation]], function(net)
        net$training_history$training)
    )),
    epoch = rep(1:1000, 10),
    activation = rep(activations, each = 5000),
    iteration = rep(rep(1:5, each = 1000), 2)) %>%
    ggplot(aes(x = epoch, y = loss, color = activation)) +
    geom_line() +
    scale_y_log10() +
    legendary_palette() +
    facet_wrap(~iteration, scales = "free_y") +
    ggtitle("Porownanie szybkosci uczenia z uzyciem roznych optimizerow",
            paste0("na zbiorze ", name, "-training")) +
    theme_minimal()

})
```







Jak widzimy, softmax, mimo że czasami blokuje się na lokalnych optimach, spada szybciej niż aktywacja liniowa. Duża niestabilność

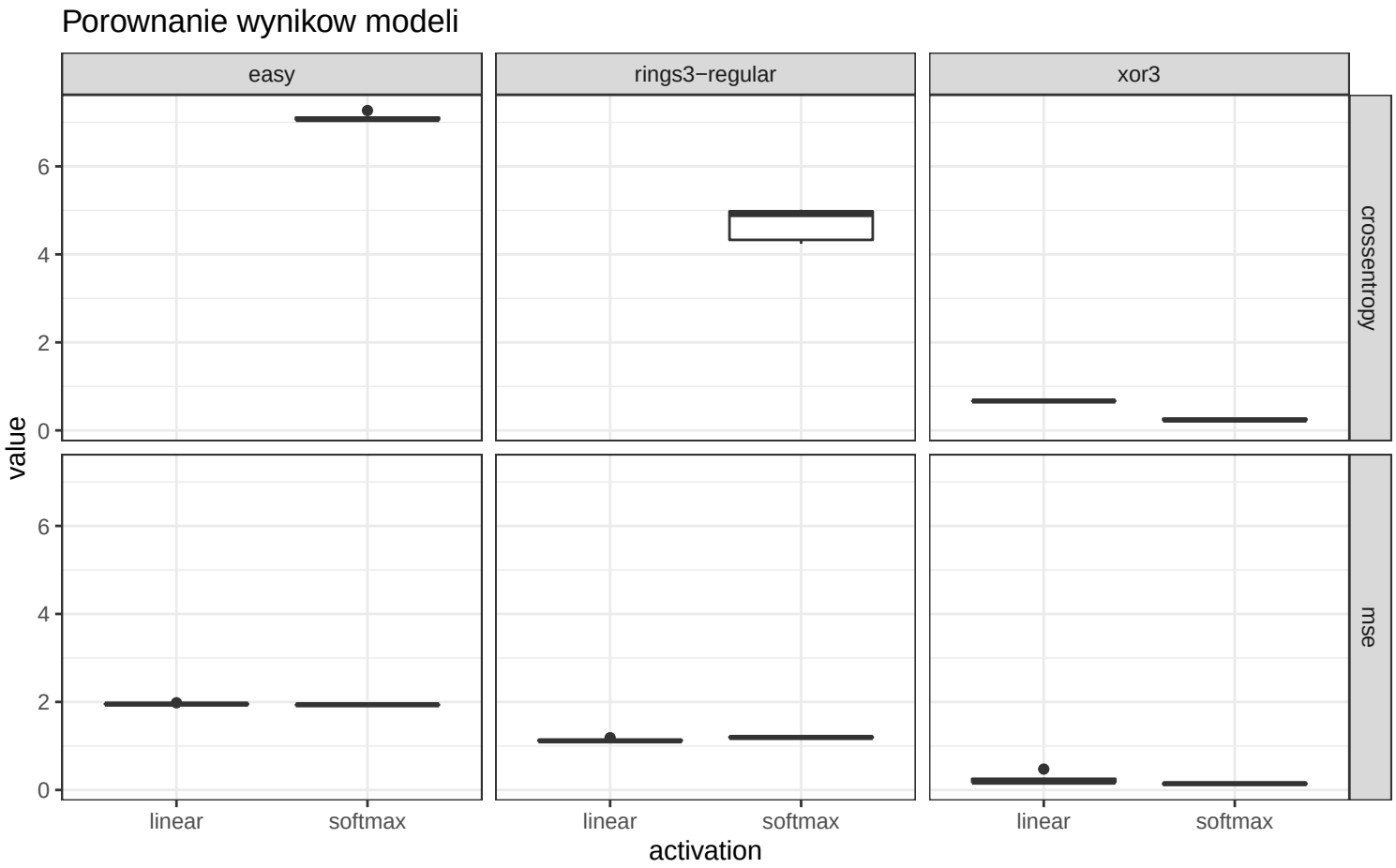
na wykresie to konsekwencja skali logarytmicznej na osi OY oraz faktu, że gradient softmaxu generalnie jest większy, więc trudniej trafić w optimum, kiedy jesteśmy już blisko.

Porównanie wyników uczenia

Teraz jeszcze spojrzymy na wyniki na zbiorze testowym i to, jak dane dopasowały się do zbiorów po 1000 epok:

```
test_preds_lab4 <- nlapply(dat_lab4_names, function(name)
  nlapply(activations, function(activation)
    nlapply(1:5, function(iteration)
      nets[["lab4"]][["classif"]][[name]][[activation]][[iteration]] %>%
        feed_network(X[[name]][["test"]]) %>%
        select_max()
    )
  )
)
```

```
plots[["lab4"]][["results"]] <- do.call(rbind, nlapply(dat_lab4_names, function(name)
  do.call(rbind, nlapply(activations, function(activation)
    do.call(rbind, nlapply(1:5, function(iteration)
      data.frame(dataset = name, activation = activation, loss = c("mse", "crossentropy"),
                 value =
                   c(mse =
                     nets[["lab4"]][["classif"]][[name]][[activation]][[iteration]] %>%
                       mse(X[[name]][["test"]], y_enc[[name]][["test"]]),
                     crossentropy =
                       nets[["lab4"]][["classif"]][[name]][[activation]][[iteration]] %>%
                       crossentropy(X[[name]][["test"]], y_enc[[name]][["test"]])))
    ))
  ))
)) %>%
  ggplot(aes(x = activation, group = activation, y = value)) +
  facet_grid(loss~dataset) +
  geom_boxplot() +
  theme_minimal() +
  ggtitle("Porownanie wynikow modeli")
```

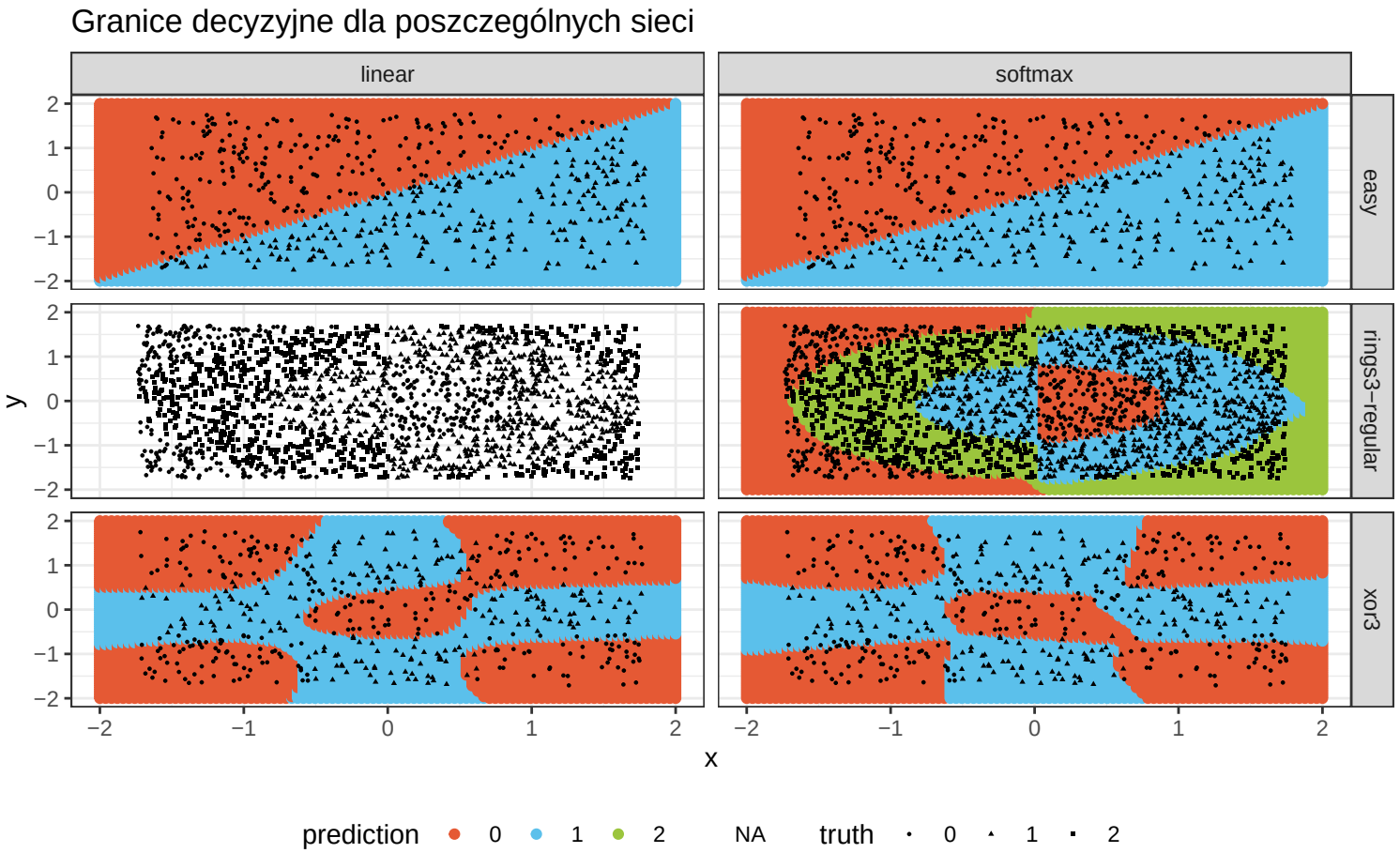


Jak widzimy, generalnie softmax osiąga lepsze wyniki. Nie wszystkie wyniki w przypadku miary crossentropy są widoczne, gdyż nie da się jej policzyć, gdy wartość na wyjściowym neuronie jest ujemna, co jest możliwe w przypadku aktywacji liniowej.

Przykładowe granice decyzyjne:

```
plot_grid <- data.frame(x = rep(seq(-2, 2, length.out = 100), each = 100),
                        y = rep(seq(-2, 2, length.out = 100), times = 100))

plot_bg <- cbind(plot_grid, do.call(rbind, lapply(dat_lab4_names, function(name)
  do.call(rbind, lapply(activations, function(activation)
    data.frame(y_pred = nets[["lab4"]][["classif"]][[name]][[activation]][[3]] %>%
        feed_network(as.matrix(plot_grid)) %>%
        select_max(),
      activation = activation,
      dataset = name))
  ))
))

plot_points <- do.call(rbind, lapply(dat_lab4_names, function(name)
  do.call(rbind, lapply(activations, function(activation)
    data.frame(
      truth = as.numeric(y[[name]][["test"]]),
      x = X[[name]][["test"]][, "x"],
      y = X[[name]][["test"]][, "y"],
      activation = activation,
      dataset = name
    )
  ))
))

plots[["lab4"]][["boundaries"]] <-
  ggplot(plot_bg, aes(x = x, y = y, color = as.factor(y_pred))) +
  geom_point() +
```

```
  legendary_palette() +
  facet_grid(dataset~activation) +
  theme_minimal() +
  labs(color = "prediction", shape = "truth") +
  ggtitle("Granice decyzyjne dla poszczególnych sieci") +
  geom_point(data = plot_points, aes(x = x, y = y, shape = as.factor(truth)),
             inherit.aes = FALSE, size = 0.5) +
  theme(legend.position = "bottom")
```



Jak widzimy, sieć z sigmoidem dopasowuje się lepiej.

Laboratorium 5

Cel: porównanie różnych fukcji aktywacji

Dla każdego zbioru danych wytrenujemy sieć z każdą kombinacją parametrów: - cztery możliwe funkcje aktywacji (linear, sigmoid, relu, tanh), - trzy możliwe liczby warstw ukrytych (one, two, three), - trzy możliwe rozmiary każdej z warstw (3, 5, 10).

Każdą z tych sieci będziemy trenować przez 100 epok, korzystając z momentum i batchowania.

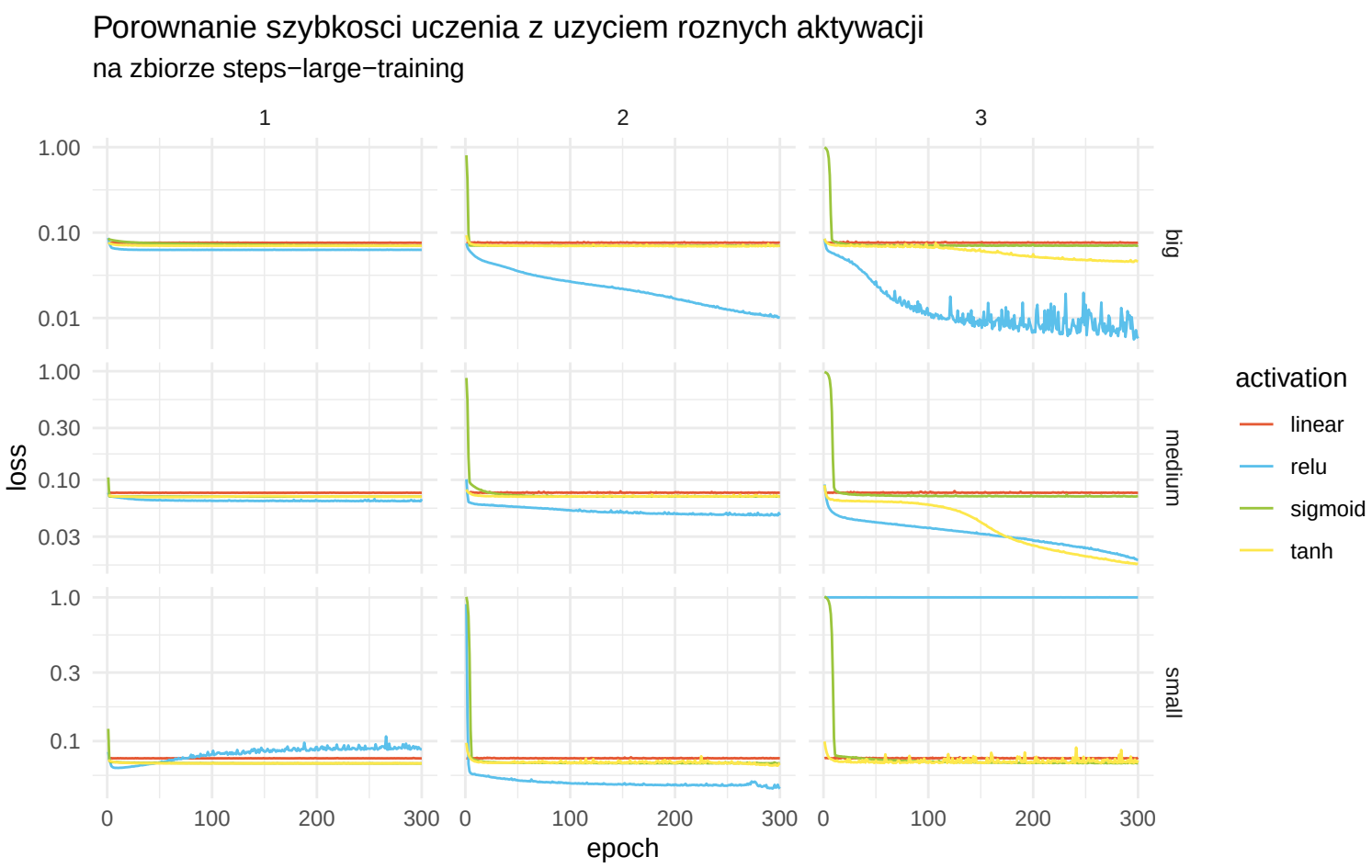
Porównanie czasu zbieżności

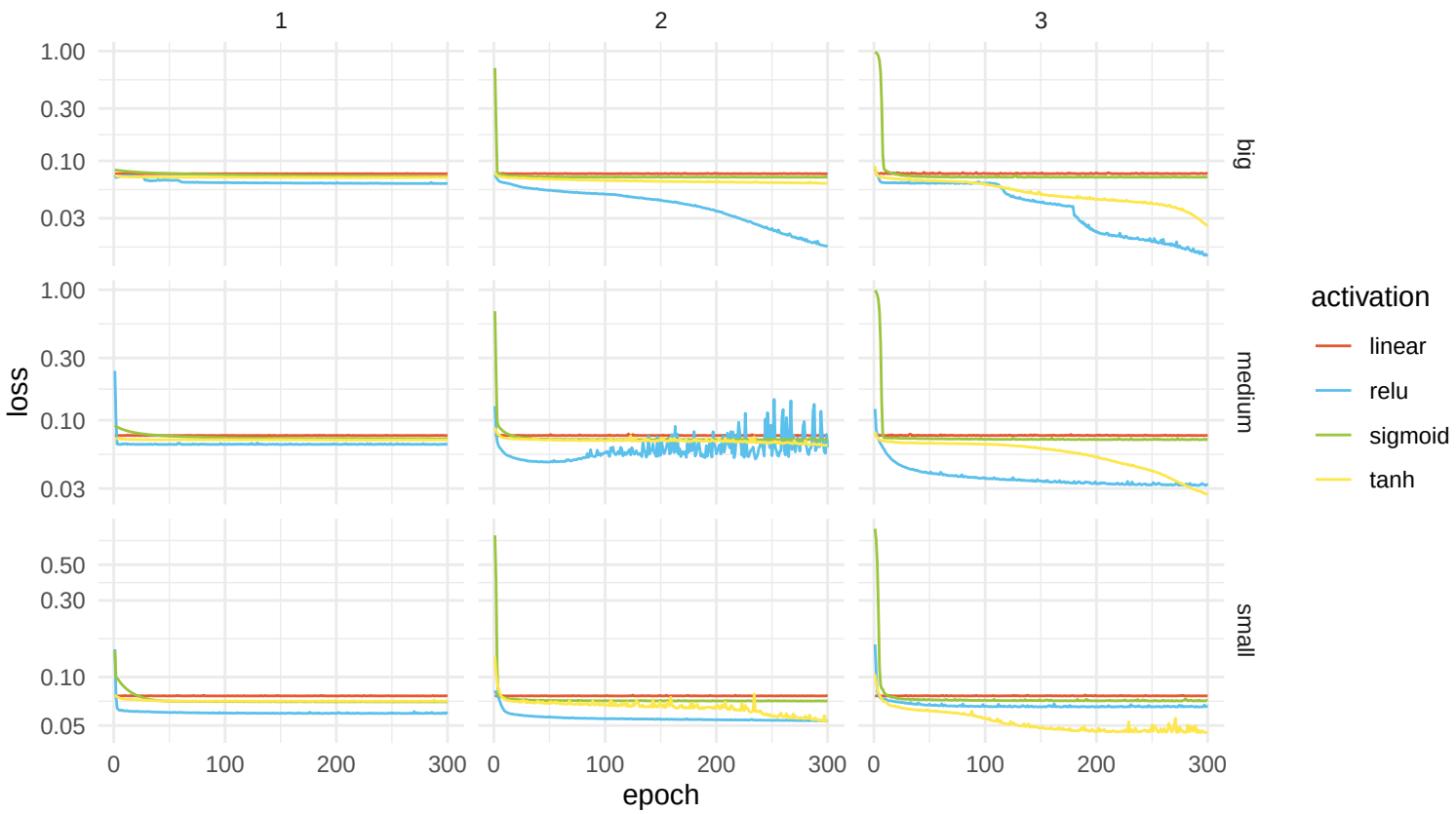
```
first_layer <- list(
  `steps-large` = neural_network(1),
  `multimodal-large` = neural_network(1),
  `rings3-regular` = neural_network(2),
  `rings5-regular` = neural_network(2)
)

last_layer <- list(
  `steps-large` = output_layer(1, "linear"),
```

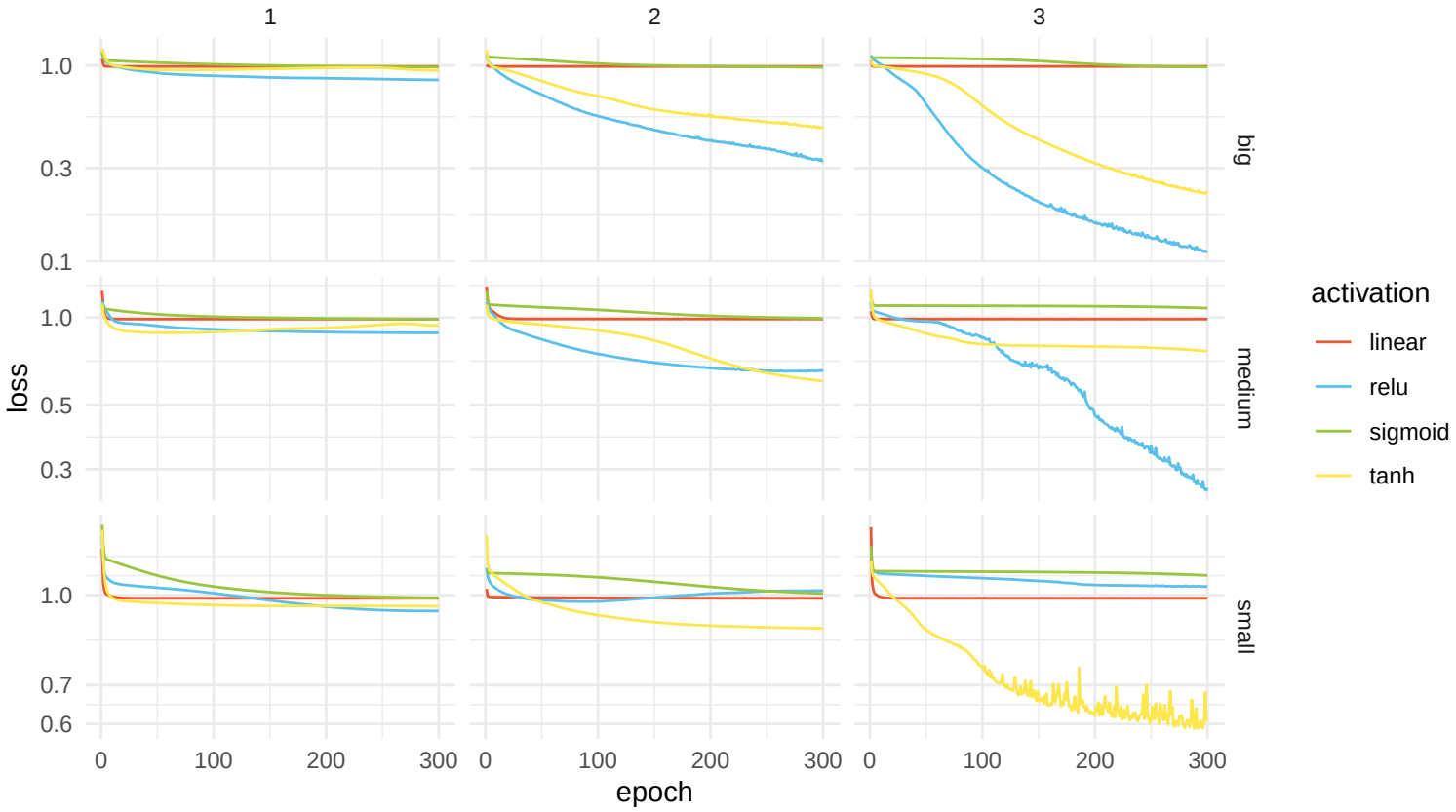
```
  `multimodal-large` = output_layer(1, "linear"),
  `rings3-regular` = output_layer(3, "softmax"),
  `rings5-regular` = output_layer(5, "softmax")
)

# set possible parameters
activations <- c("linear", "sigmoid", "relu", "tanh")
ns_layers <- c(1:3)
sizes_layers <- c(small = 3, medium = 5, big = 10)

nets[["lab5"]] <- nlapply(dat_lab5_names, function(name)
  nlapply(activations, function(activation)
    nlapply(sizes_layers, function(size)
      lapply(ns_layers, function(n) {
        net <- switch (n,
                       "1" = first_layer[[name]] +
                         hidden_layer(size, activation),
                       "2" = first_layer[[name]] +
                         hidden_layer(size, activation) +
                         hidden_layer(size, activation),
                       "3" = first_layer[[name]] +
                         hidden_layer(size, activation) +
                         hidden_layer(size, activation) +
                         hidden_layer(size, activation)) +
          last_layer[[name]]
        net %>%
          randomize_weights_xavier() %>%
          train_network_momentum(
            X[[name]][["train"]],
            if (name %in% dat_lab5_classif_names) y_enc[[name]][["train"]]
            else y[[name]][["train"]],
            num_epochs = 300,
            eta = 1e-4,
            batch_size = 100,
            verbose = FALSE,
            loss = if (name %in% dat_lab5_classif_names) "crossentropy" else "mse"
          )
      })
    )
  )
)
```

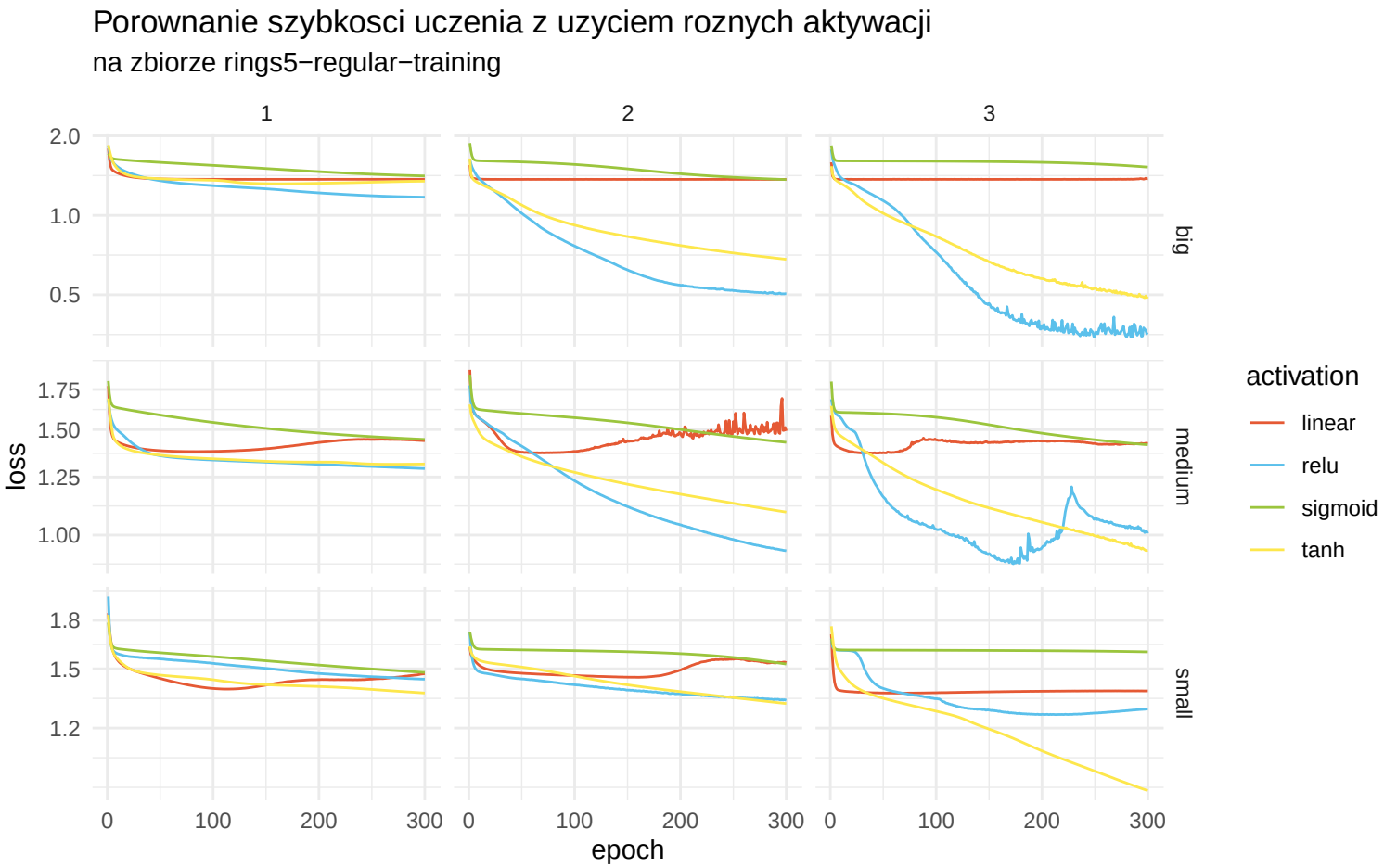
```
plots[["lab5"]] <- list()
plots[["lab5"]][["convergence"]] <- nlapply(dat_lab5_names, function(name){
  data.frame(
    loss = unlist(lapply(activations, function(activation)
      unlist(lapply(names(sizes_layers), function(size)
        unlist(lapply(ns_layers, function(n)
          nets[["lab5"]][[name]][[activation]][[size]][[n]]$training_history$training
        ))
      ))
    )),
    activation = rep(activations, each = 3 * 3 * 300),
    size = rep(names(sizes_layers), each = 3 * 300),
    n = rep(ns_layers, each = 300),
    epoch = 1:300) %>%
    filter(!is.nan(loss), loss < 10) %>%
    ggplot(aes(x = epoch, y = loss, color = activation)) +
    geom_line() +
    scale_y_log10() +
    legendary_palette() +
```

```
    facet_grid(size~n, scales = "free_y") +
    ggtitle("Porownanie szybkosci uczenia z uzyciem roznych aktywacji",
            paste0("na zbiorze ", name, "-training")) +
    theme_minimal()

})
```




Porownanie szybkosci uczenia z uzyciem roznych aktywacji
na zbiorze multimodal−large−training
1
2
3
big
medium
small
loss
epoch
activation
linear
relu
sigmoid
tanh


Porownanie szybkosci uczenia z uzyciem roznych aktywacji
na zbiorze rings3−regular−training
1
2
3
big
medium
small
loss
epoch
activation
linear
relu
sigmoid
tanh



Fakt że oscylacje krzywej są względnie większe w dolnych częściach wykresu, wynika z zastosowania skali logarytmicznej.

Skrajnie duże wartości (powyżej 10) zostały usunięte dla czytelności.

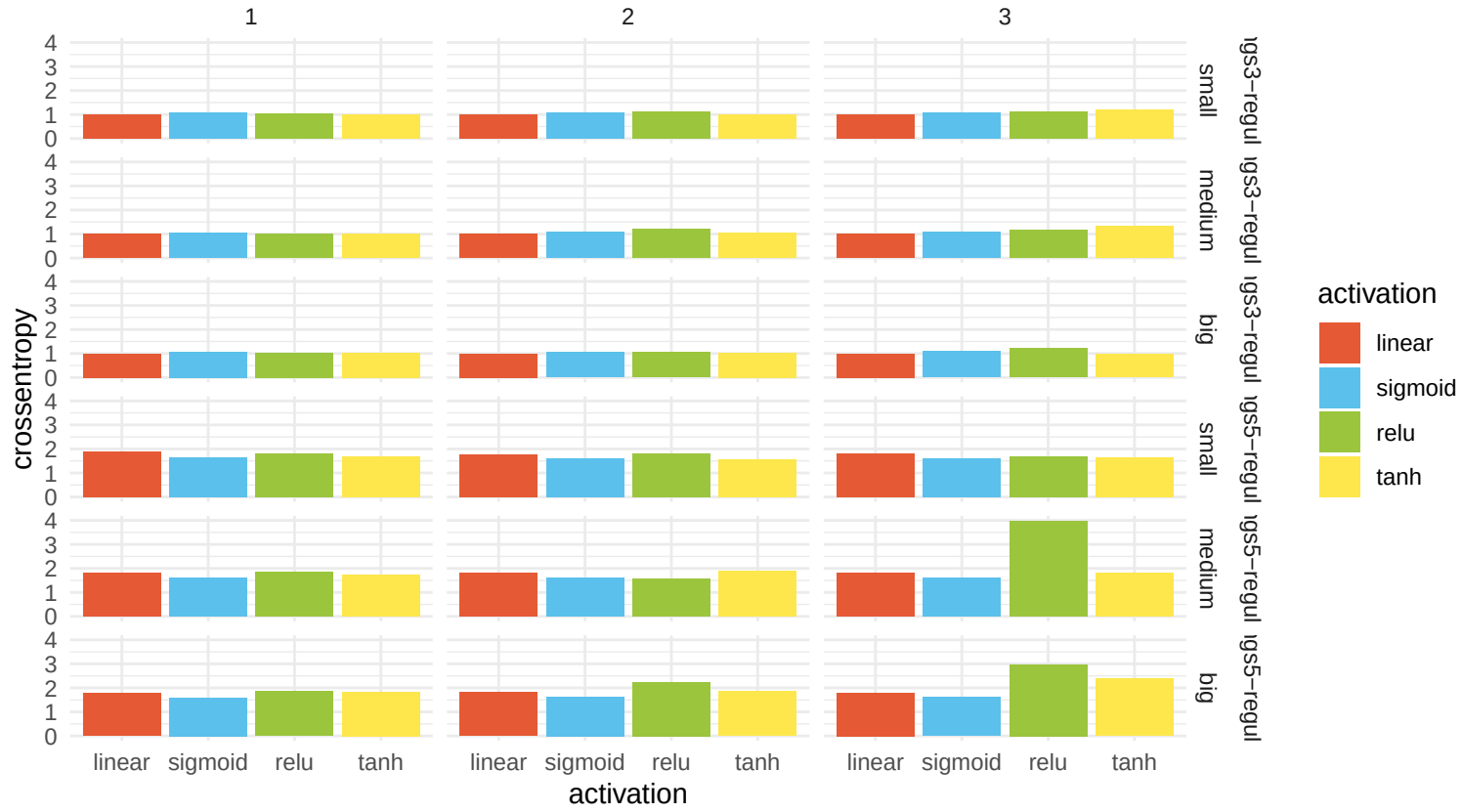
Możemy na podstawie tych wykresów wyciągnąć kilka wniosków:

- Funkcja liniowa sprawdza się nienajgorzej w przypadku regresji, jednak jest bardzo niestabilna w przypadku klasyfikacji i wagi często "eksplodują".
- ReLU łatwo wpada w lokalne optimum (co jest spowodowane tym, jak wygląda jego pochodna – daltego często w praktyce zamiast ReLU używa się ReLU modyfikowanego)
- Dla klasyfikacji prawie w każdym przypadku sigmoid jest lepszy niż tanh, w przypadku regresji bywa na odwrót (widać to wyraźniej, kiedy zbadamy większą liczbę epok).
- Im więcej warstw, tym sigmoid skuteczniejszy, a ReLU mniej skuteczne i funkcja liniowa łatwiej rozbiega.
- Im więcej warstw i neuronów, tym większa niestabilność (lokalne oscylacje mają większe amplitudy).

Porównanie wyników na zbiorze testowym

```
func_selector <- function(results, name) {
  if (name %in% dat_lab5_classif_names) select_max(results)
  else results
}

test_preds_lab5 <- nlapply(dat_lab5_names, function(name)
  nlapply(activations, function(activation)
    nlapply(names(sizes_layers), function(size)
      nlapply(ns_layers, function(n)
        nets[["lab5"]][[name]][[activation]][[size]][[n]] %>%
          feed_network(X[[name]][["test"]]) %>%
          func_selector(name)
      )
    )
  )
)
```

```r
plots[["lab5"]][["classif-results"]] <- do.call(rbind, nlapply(dat_lab5_classif_names, function(name)
  do.call(rbind, nlapply(activations, function(activation)
    do.call(rbind, nlapply(names(sizes_layers), function(size)
      do.call(rbind, nlapply(ns_layers, function(n)
        data.frame(dataset = name,
                   activation = activation,
                   n = n,
                   size = size,
                   crossentropy = nets[["lab5"]][[name]][[activation]][[size]][[n]] %>%
                         crossentropy(X[[name]][["test"]], y_enc[[name]][["test"]]))
      ))
    ))
  ))
)) %>%
  ggplot(aes(x = activation, group = activation, fill = activation, y = crossentropy)) +
  facet_grid(dataset+size~n) +
  legendary_fill_palette() +
  geom_bar(stat = "identity") +
  theme_minimal() +
  ggtitle("Porownanie wynikow modeli na zbiorach klasyfikacyjnych",
          "dla różnych kombinacji rozmiarow i liczby warstw oraz roznych funkcji aktywacji")

plots[["lab5"]][["regr-results"]] <- do.call(rbind, nlapply(dat_lab5_regr_names, function(name)
  do.call(rbind, nlapply(activations, function(activation)
    do.call(rbind, nlapply(names(sizes_layers), function(size)
      do.call(rbind, nlapply(ns_layers, function(n)
        data.frame(dataset = name,
                   activation = activation,
                   n = n,
                   size = size,
                   mse = nets[["lab5"]][[name]][[activation]][[size]][[n]] %>%
                         mse(X[[name]][["test"]], y[[name]][["test"]]))
      ))
    ))
  ))
)) %>%
  ggplot(aes(x = activation, group = activation, fill = activation, y = mse)) +
  facet_grid(dataset+size~n) +
  legendary_fill_palette() +
  geom_bar(stat = "identity") +
  theme_minimal() +
  ggtitle("Porownanie wynikow modeli na zbiorach regresyjnych",
          "dla różnych kombinacji rozmiarow i liczby warstw oraz roznych funkcji aktywacji")
```



Wyniki na zbiorze treningowym zdają się potwierdzać wcześniejsze obserwacje.

Laboratorium 6

Cel: zbadanie przeuczania sieci i regularyzacji

Będziemy testować trzy metody mające przeciwdziałać przeuczeniu się sieci:

- Zatrzymywanie procesu uczenia po wzroście błędu na zbiorze walidacyjnym
- Regularyzację L2
- Dropout

Możemy z nich korzystać dzięki dodatkowym argumentom w funkcjach `train_network_sgd`, `train_network_momentum` i `train_network_rmsprop`.

```
train_network_sgd(network, X, y,

                  # monitorowanie zbioru walidacyjnego:
                  X_validation,         # zbior X do walidacji
                  y_validation,         # zbior y do walidacji
                  validation_threshold, # prog walidacji
                  min_epochs,           # minimalna liczba epok przed przerwaniem

                  # regularyzajca L2
                  lambda,               # wspolczynnik regularyzacji L2

                  # dropout
                  dropout_rate          # wspolczynnik odrzucania neuronow
                  )
```

Monitorowanie zbioru walidacyjnego działa w ten sposób – jeśli podamy parametry X_validation i y_validation, to w każdej epoce, począwszy od min_epochs (domyślnie num_epochs / 10) będziemy sprawdzać, czy na zbiorze walidacyjnym wartość błędu nie wzrosła względem poprzedniej epoki o współczynnik validation_threshold. Jeśli tak – prerywamy proces uczenia.

Porównanie metod dla regresji

Najpierw przyjrzymy się dokładnie jednemu zbiorowi regresyjnemu. Wytrenujemy jedną sieć bez zastosowania regularyzacji i po trzy sieci dla każdej z trzech metod zapobiegających przeuczeniu z różnymi zestawami parametrów.

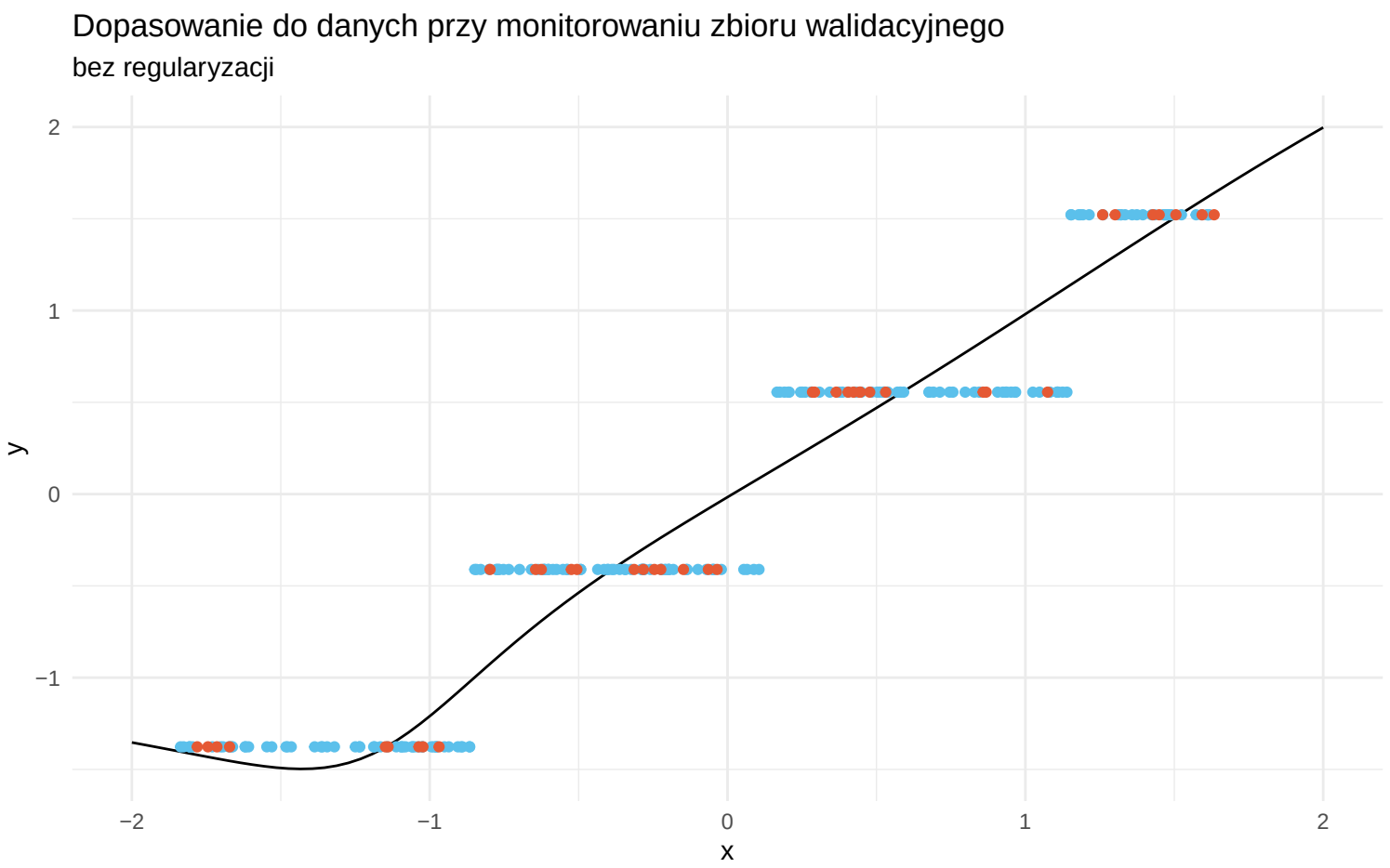
Oto porównanie przebiegów:

```
methods <- c("impr-stop", "lambda", "dropout")

par_impr_stop <- c(`1.1` = 1.1, `1.5` = 1.5, `2.5` = 2.5)
par_lambda <- c(`0.01` = 0.01, `0.1` = 0.1, `1` = 1)
par_dropout <- c(`0.1` = 0.1, `0.2` = 0.2, `0.4` = 0.4)
pars <- list(`impr-stop` = par_impr_stop,
             lambda = par_lambda,
             dropout = par_dropout)

net_lab6_regr <- (neural_network(1) +
                    hidden_layer(10, "sigmoid") +
                    hidden_layer(10, "sigmoid") +
                    hidden_layer(10, "sigmoid") +
                    output_layer(1))

nets[["lab6"]] <- list()
nets[["lab6"]][["regr"]] <-
  list(
    `no-reg` = net_lab6_regr %>%
      randomize_weights_xavier() %>%
      train_network_momentum(
        X[["multimodal-sparse"]][["train"]],
        y[["multimodal-sparse"]][["train"]],
        num_epochs = 5000,
        eta = 1e-2,
        batch_size = 10,
```

```
      verbose = FALSE
    ),
  `impr-stop` = nlapply(par_impr_stop, function(impr_stop) {
    val_inds <- sample(1:nrow(X[["multimodal-sparse"]][["train"]]),
                       ceiling(nrow(X[["multimodal-sparse"]][["train"]]) / 5))
    net_lab6_regr %>%
      randomize_weights_xavier() %>%
      train_network_momentum(
        X[["multimodal-sparse"]][["train"]][-val_inds, , drop = FALSE],
        y[["multimodal-sparse"]][["train"]][-val_inds, , drop = FALSE],
        num_epochs = 5000,
        eta = 1e-2,
        batch_size = 10,
        verbose = FALSE,
        X_validation = X[["multimodal-sparse"]][["train"]][val_inds, , drop = FALSE],
        y_validation = y[["multimodal-sparse"]][["train"]][val_inds, , drop = FALSE],
        validation_threshold = impr_stop
      )
  }),
  lambda = nlapply(par_lambda, function(lambda)
    net_lab6_regr %>%
      randomize_weights_xavier() %>%
      train_network_momentum(
        X[["multimodal-sparse"]][["train"]],
        y[["multimodal-sparse"]][["train"]],
        num_epochs = 5000,
        eta = 1e-2,
        batch_size = 10,
        verbose = FALSE,
        lambda = lambda
      )
  ),
  dropout = nlapply(par_dropout, function(dropout)
    net_lab6_regr %>%
      randomize_weights_xavier() %>%
      train_network_momentum(
        X[["multimodal-sparse"]][["train"]],
        y[["multimodal-sparse"]][["train"]],
        num_epochs = 5000,
        eta = 1e-3,
        batch_size = 10,
        verbose = FALSE,
        dropout_rate = dropout
      )
  )
)
```

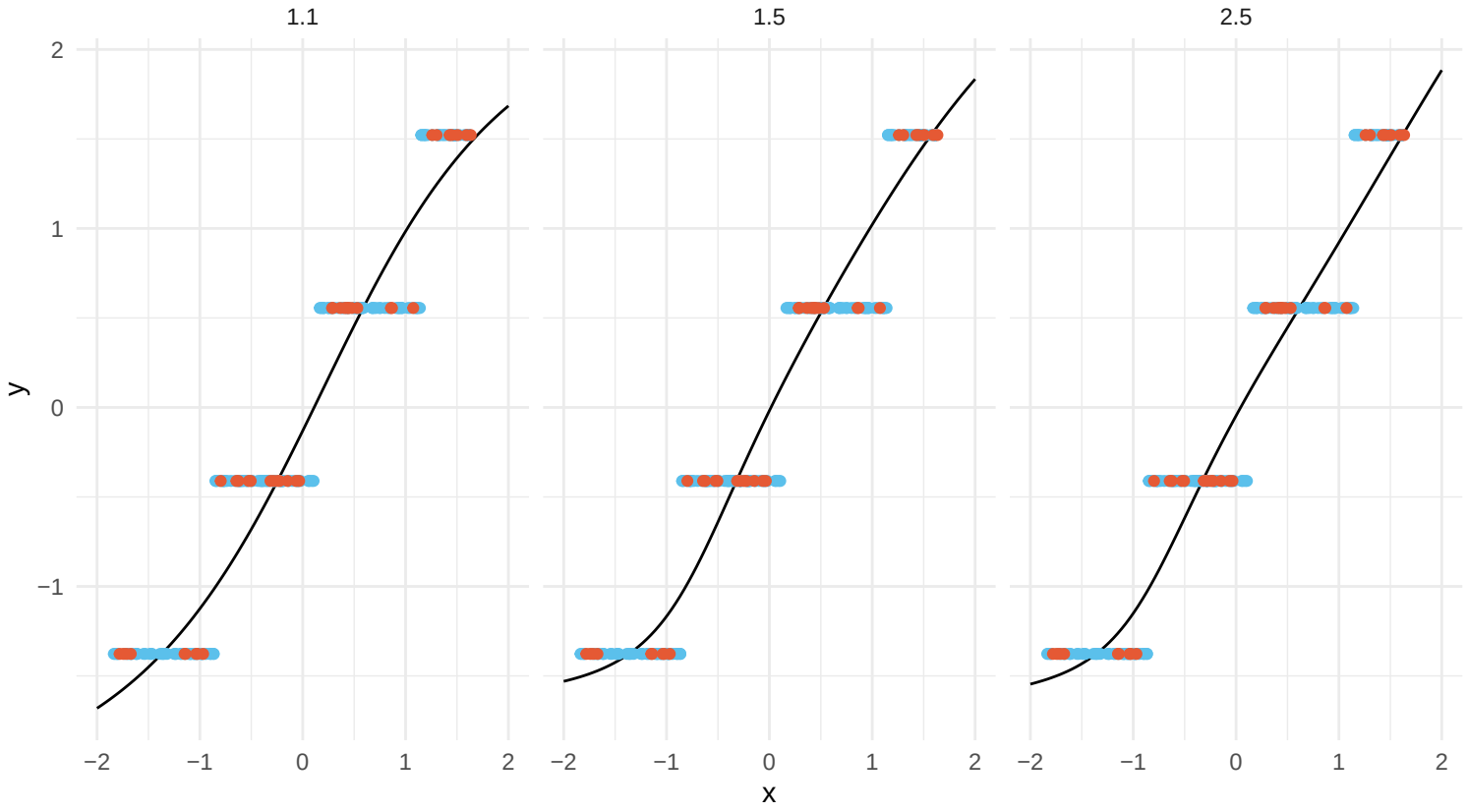
```
plots[["lab6"]] <- list()
plots[["lab6"]][["regr"]] <- c(
  list(`no-reg` =
         data.frame(
           x = plot_grid[, "x", drop = FALSE],
           y = nets[["lab6"]][["regr"]][["no-reg"]] %>%
             feed_network(as.matrix(plot_grid[, "x", drop = FALSE]))
         ) %>%
         ggplot(aes(x = x, y = y)) +
         geom_line() +
         geom_point(data = data.frame(x = X[["multimodal-sparse"]][["test"]],
                                      y = y[["multimodal-sparse"]][["test"]]),
                    color = "#5bc0eb") +
         geom_point(data = data.frame(x = X[["multimodal-sparse"]][["train"]],
```

```
                                    y = y[["multimodal-sparse"]][["train"]]),
                    color = "#e55934") +
        theme_minimal()),
  nlapply(methods, function(method)
    do.call(rbind, lapply(names(pars[[method]]), function(par)
      data.frame(
        x = plot_grid[, "x", drop = FALSE],
        y = nets[["lab6"]][["regr"]][[method]][[par]] %>%
          feed_network(as.matrix(plot_grid[, "x", drop = FALSE])),
        param = par
      ))) %>%
      ggplot(aes(x = x, y = y)) +
      geom_line() +
      facet_grid(~param) +
      geom_point(data = data.frame(x = X[["multimodal-sparse"]][["test"]],
                                   y = y[["multimodal-sparse"]][["test"]]),
                 color = "#5bc0eb") +
      geom_point(data = data.frame(x = X[["multimodal-sparse"]][["train"]],
                                   y = y[["multimodal-sparse"]][["train"]]),
                 color = "#e55934",
                 size = 1.5) +
      theme_minimal()
  )
)
```



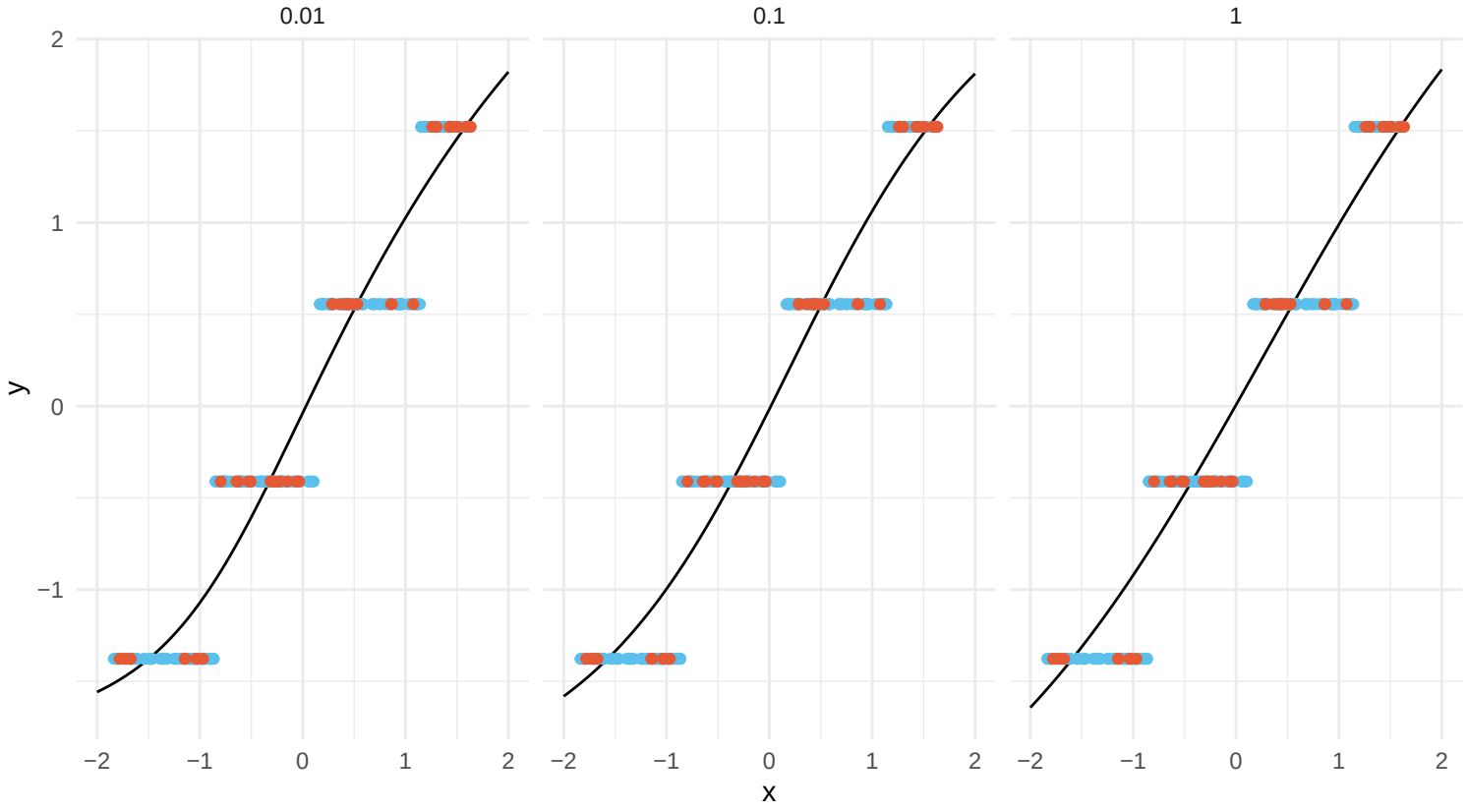
Dopasowanie do danych przy monitorowaniu zbioru walidacyjnego

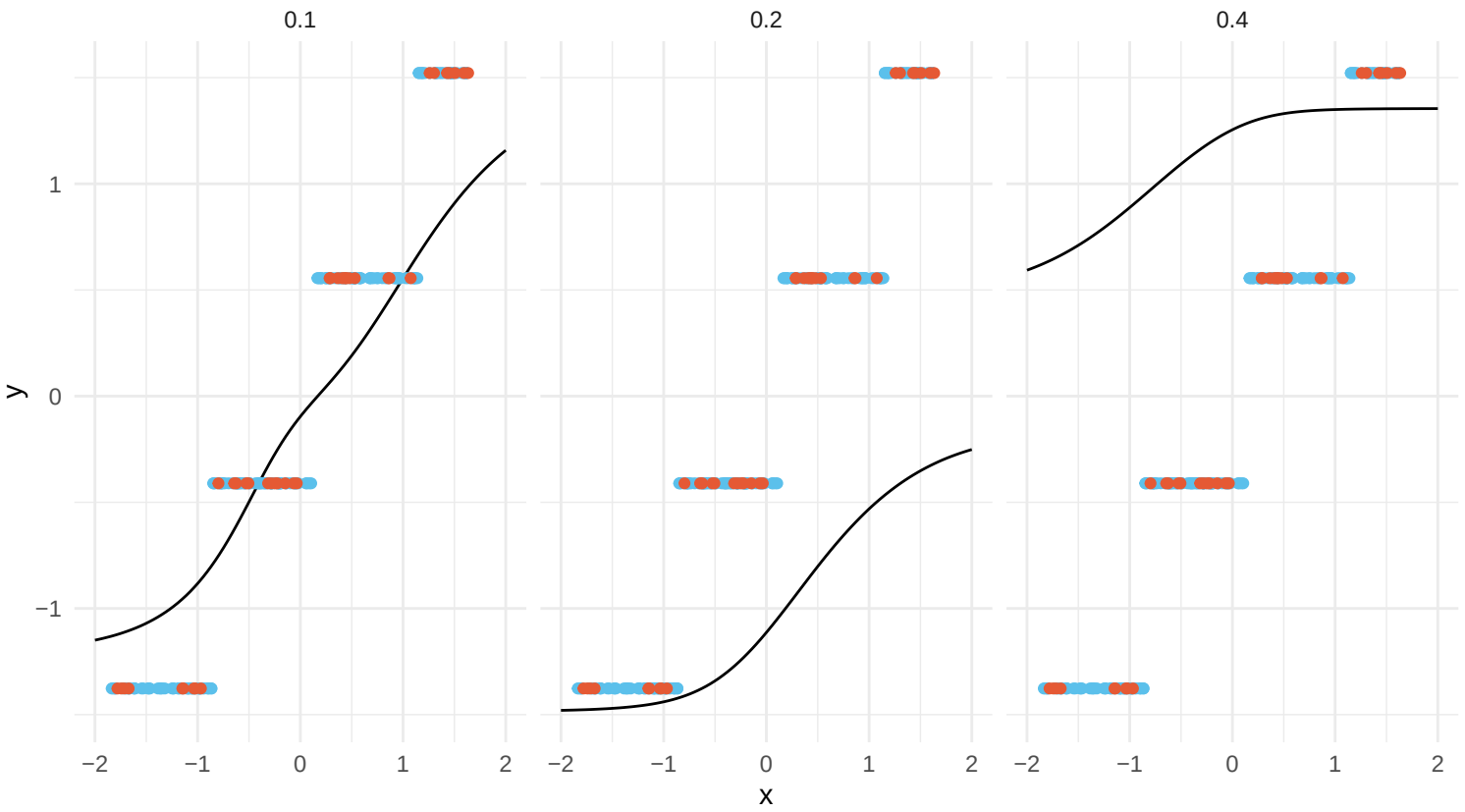
dla roznych wartosci progu zatrzymania



Dopasowanie do danych przy regularyzacji L2

dla roznych wartosci parametru lambda





Na powyższych rysunkach czarną linią zaznaczona jest predykcja modelu dla poszczególnych wartości na wejściu, na czerwono punkty ze zbioru treningowego, a na niebiesko - ze zbioru testowego.

Możemy zobaczyć z wizualizacji, dodanie elementów regularyzacji istotnie upraszcza model, jednak trzeba uważać, gdyż nieraz nawet za bardzo, jeśli nieodpowiednio dobierze się parametry.

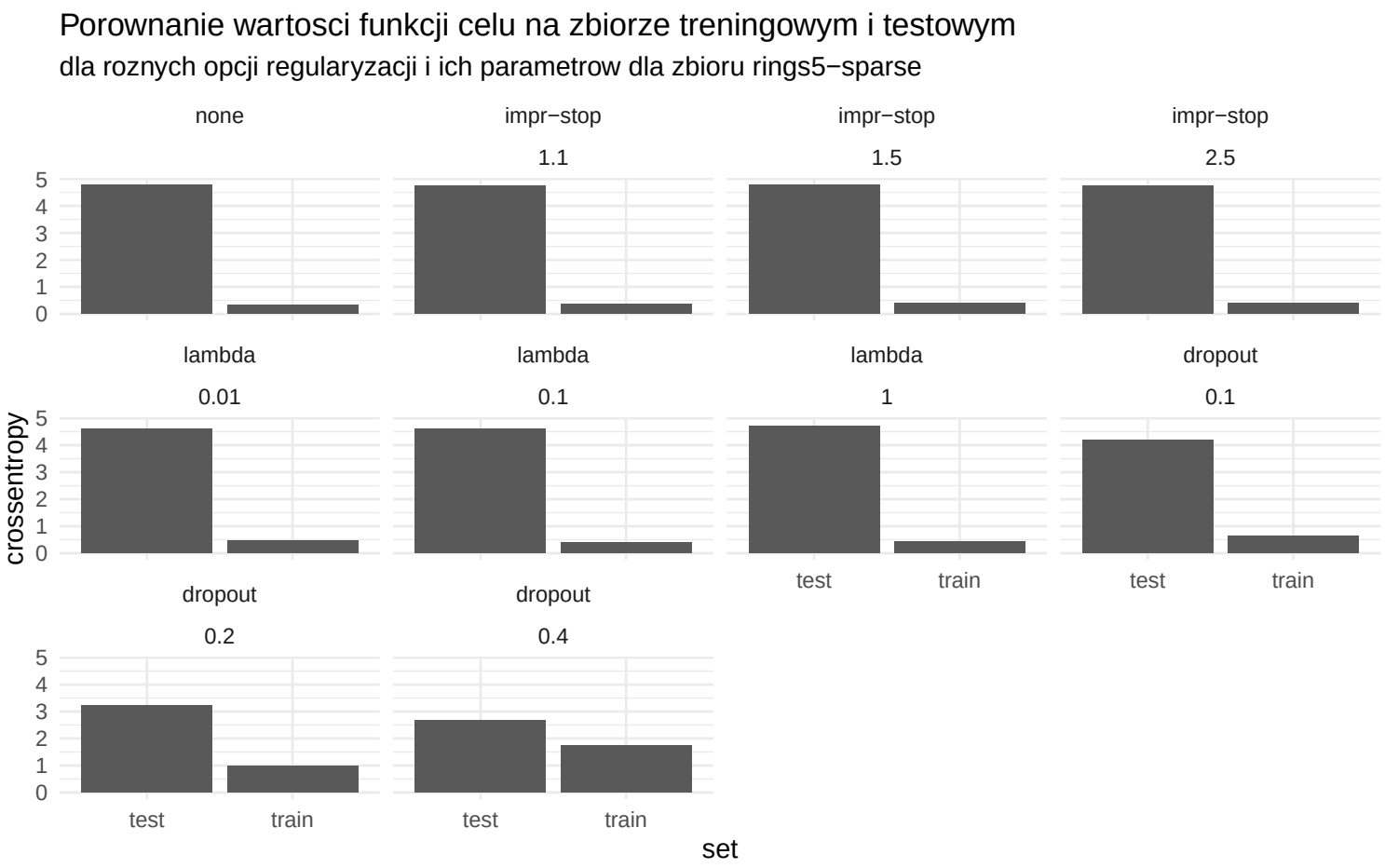
Regularyzacja klasyfikacji

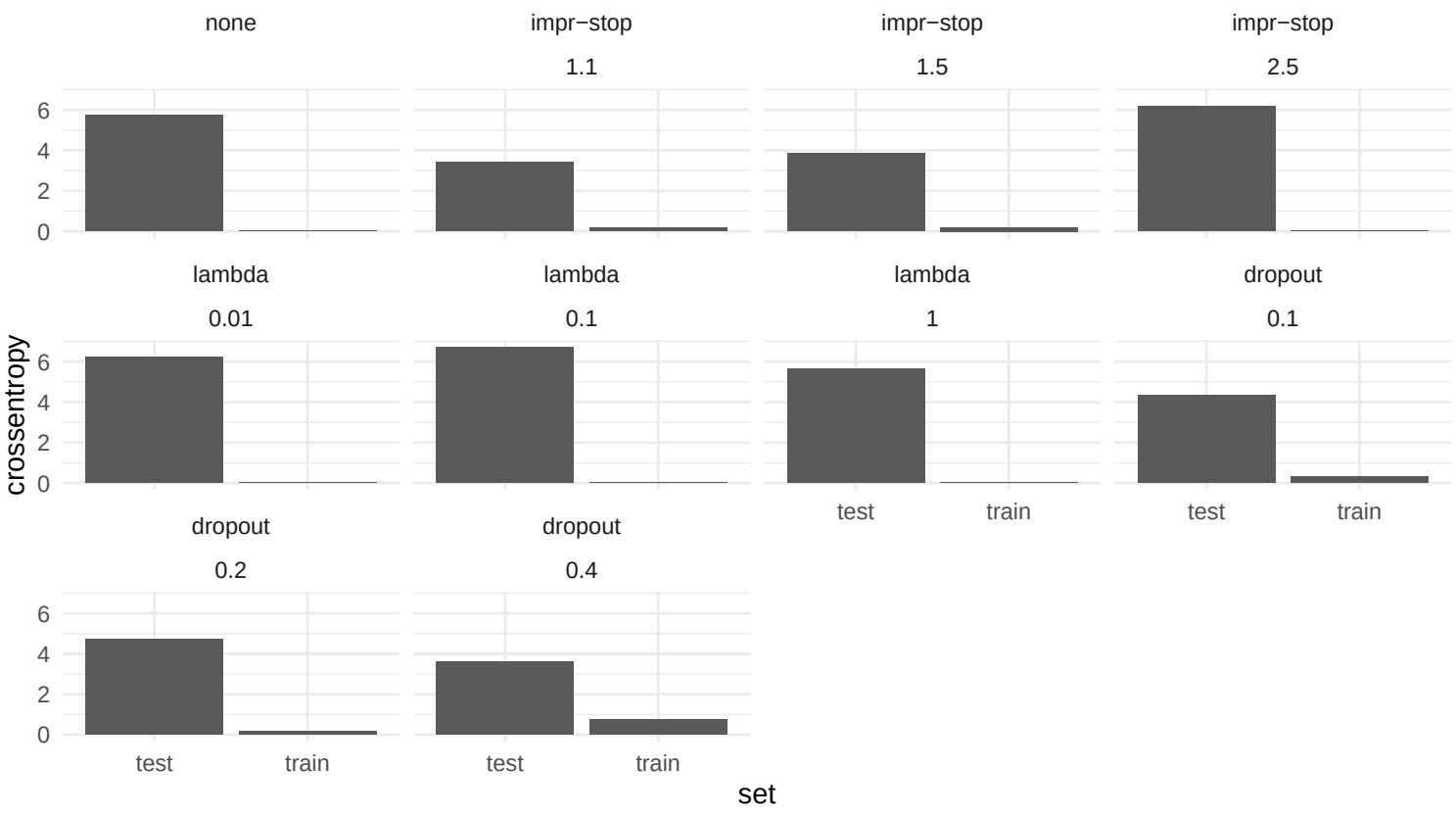
```
nets[["lab6"]][["classif"]] <- nlapply(dat_lab6_classif_names, function(name) {
  net <- neural_network(2) +
    hidden_layer(20, "sigmoid") +
    hidden_layer(20, "sigmoid") +
    hidden_layer(20, "sigmoid") +
    output_layer(ncol(y_enc[[name]][["train"]]), "softmax")
  list(
    `no-reg` = net %>%
        randomize_weights_xavier() %>%
        train_network_momentum(
          X[[name]][["train"]],
          y_enc[[name]][["train"]],
          num_epochs = 3000,
          eta = 1e-3,
          batch_size = 100,
          verbose = FALSE,
          loss = "crossentropy"
        ),
    `impr-stop` = nlapply(par_impr_stop, function(impr_stop) {
      val_inds <- sample(1:nrow(X[[name]][["train"]]),
                         ceiling(nrow(X[[name]][["train"]]) / 5))
      net %>%
        randomize_weights_xavier() %>%
        train_network_momentum(
```

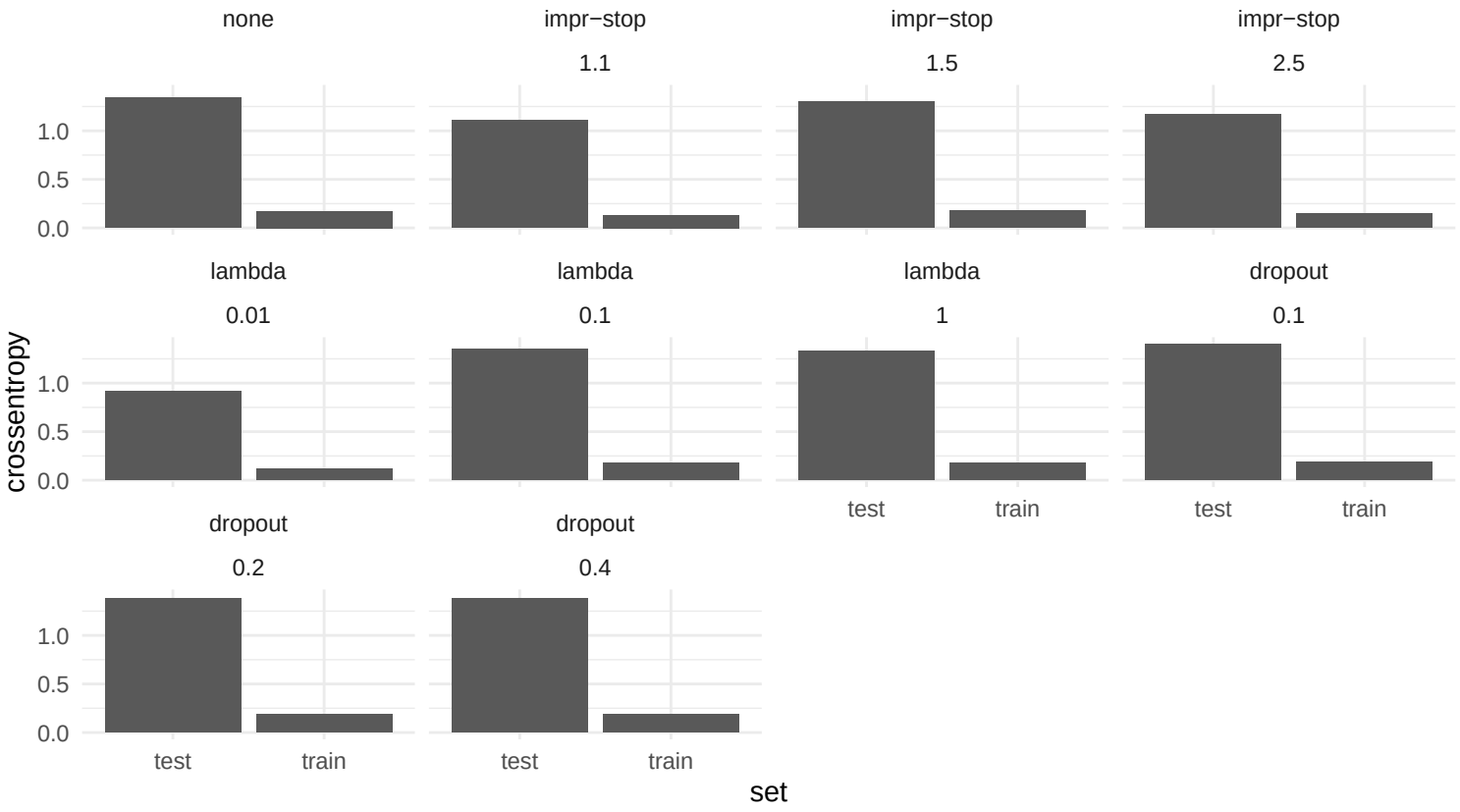
```
          X[[name]][["train"]][-val_inds, , drop = FALSE],
          y_enc[[name]][["train"]][-val_inds, , drop = FALSE],
          num_epochs = 3000,
          eta = 1e-3,
          batch_size = 100,
          verbose = FALSE,
          X_validation = X[[name]][["train"]][val_inds, , drop = FALSE],
          y_validation = y_enc[[name]][["train"]][val_inds, , drop = FALSE],
          validation_threshold = impr_stop,
          loss = "crossentropy"
        )
    }),
    lambda = nlapply(par_lambda, function(lambda)
      net %>%
        randomize_weights_xavier() %>%
        train_network_momentum(
          X[[name]][["train"]],
          y_enc[[name]][["train"]],
          num_epochs = 3000,
          eta = 1e-3,
          batch_size = 100,
          verbose = FALSE,
          lambda = lambda,
          loss = "crossentropy"
        )
    ),
    dropout = nlapply(par_dropout, function(dropout)
      net %>%
        randomize_weights_xavier() %>%
        train_network_momentum(
          X[[name]][["train"]],
          y_enc[[name]][["train"]],
          num_epochs = 3000,
          eta = 1e-3,
          batch_size = 100,
          verbose = FALSE,
          dropout_rate = dropout,
          loss = "crossentropy"
        )
    )
  )
})
```

```
plots[["lab6"]][["classif"]] <- nlapply(dat_lab6_classif_names, function(name)
  do.call(rbind, c(
    list(data.frame(
      crossentropy = c(
        nets[["lab6"]][["classif"]][[name]][["no-reg"]] %>%
          crossentropy(X[[name]][["train"]], y_enc[[name]][["train"]]),
        nets[["lab6"]][["classif"]][[name]][["no-reg"]] %>%
          crossentropy(X[[name]][["test"]], y_enc[[name]][["test"]])),
      set = c("train", "test"),
      regularization = "none",
      parameter = ""
    )),
    lapply(methods, function(method)
      do.call(rbind, lapply(names(pars[[method]]), function(par)
        data.frame(
          crossentropy = c(
            nets[["lab6"]][["classif"]][[name]][[method]][[par]] %>%
              crossentropy(X[[name]][["train"]], y_enc[[name]][["train"]]),
```

```
            nets[["lab6"]][["classif"]][[name]][[method]][[par]] %>%
              crossentropy(X[[name]][["test"]], y_enc[[name]][["test"]])),
          set = c("train", "test"),
          regularization = method,
          parameter = par
        )
      ))
    ))
  ) %>%
    ggplot(aes(x = set, y = crossentropy)) +
    facet_wrap(~regularization+parameter) +
    geom_bar(stat = "identity") +
    ggtitle("Porownanie wartosci funkcji celu na zbiorze treningowym i testowym",
            paste0("dla roznych opcji regularyzacji i ich parametrow dla zbioru ", name)) +
    theme_minimal()
)
```







Błąd na zbiorze treningowym jest mniejszy w przypadku bez regularyzacji, jednak, jak możemy zobaczyć na wykresach, na zbiorze testowym może on być niższy w przypadku odpowiednio parametrów. Jeśli wybierzemy nieodpowiednie parametry – np.

jak tutaj za duży parametr lambda lub za duży dropout_rate.