DLA1

404018

December 2020

1 Building a simple perceptron

1.1 Training and test set

I generated the set of all possible inputs using a function which created all permutations of a vector containing -1 and 1.

I used 80% of this for the training set and the other 20% for the test set.

1.2 Perceptron

Using the code provided (from per.R) I adapted the perceptron as below:

```
perceptron <- function(weights, epsilon, epsilon_degrad, epochs, error_print = FALSE,
    sd_print = FALSE) {

    errors <- rep(NULL, times = epochs)
    sd_weights <- rep(NULL, times = epochs)

    # Make perceptron function
    for (iteration in 1:epochs) {
        order = sample(length(results.train))
        error = 0

        # Reduce epsilon each epoch, make sure it doesn't fall below floating point
        # erro
        epsilon <- epsilon * epsilon_degrad
        ifelse(epsilon < 1e-16, epsilon <- 1e-16, epsilon <- epsilon)

        for (i in order) {
            x = training[, i]
            t = results.train[i]
            y = sum(x * weights)
            y <- y > 0
            error = error + (0.5 * (t - y)^2)
            dw <- epsilon * (t - y) * x
            weights <- weights + dw
        }

        # Round weights
        weights <- round(weights, digits = 6)
        errors[iteration] <- error
        sd_weights[iteration] <- sd(weights)

        # Print error every so often
        if (iteration %in% seq(epochs/1000, epochs, by = epochs/1000)) {
            # print(error) print(sd(weights))
        }
```

```
        if (error == 0)
            break
    }

    # Use output options to output metrics we are interested in
    if (error_print == T) {
        if (sd_print == F) {
            return(list(error, weights, iteration, errors))
        } else {
            return(list(error, weights, iteration, errors, sd_weights))
        }
    } else {
        if (sd_print == F) {
            return(list(error, weights, iteration))
        } else {
            return(list(error, weights, iteration, sd_weights))
        }
    }
}
```

Given that the perceptron functions by summing the dot product:

$$y = \sum_{0}^{i} w_i x_i \tag{1}$$

$$f(y) = z \tag{2}$$

Where $f()$ is the transfer function and z is the output of the perceptron. As the question in this case is asking to classify the the sign of the sum of the input values you would therefore expect that:

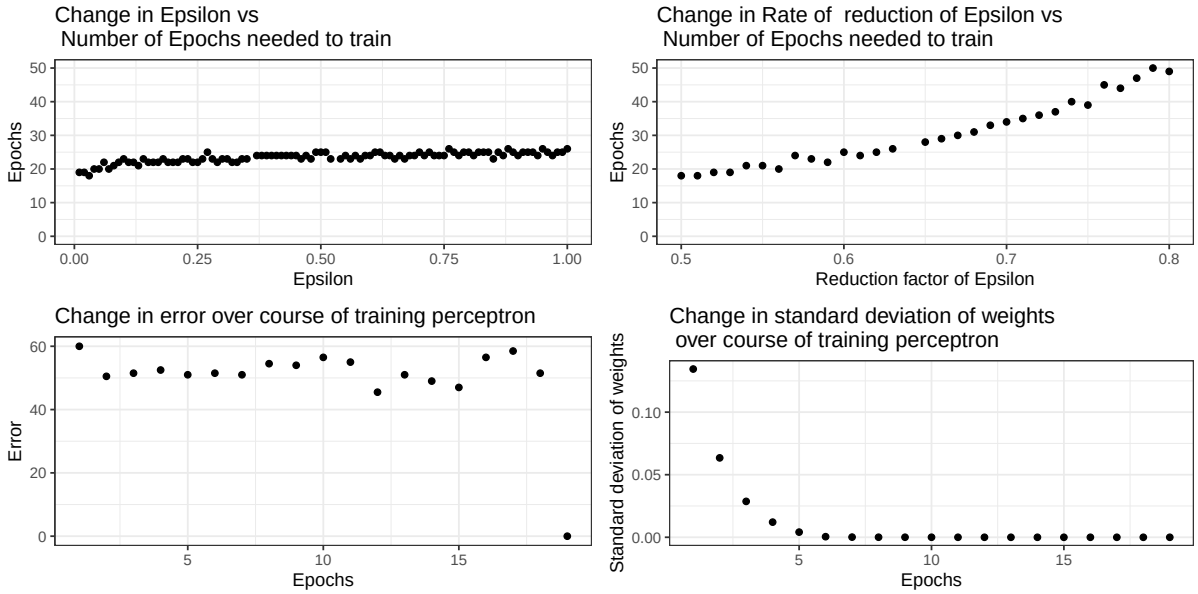
$$f(\sum_{0}^{i} x_i) = f(\sum_{0}^{i} w_i x_i) \tag{3}$$

if $f()$ is the threshold function you would expect the weights to converge to a single value. For this reason I also tracked the error using the standard deviation of the weights.

I initiated the weights with a randomly generated sequnce from 0 to 1 (and used set.seed for reproducibility).

1.3 Testing parameters

I tested the parameters of the perceptron including starting epsilon and the degree by which epsilon falls each time. I worked out the optimum parameters and then used these to train the network.



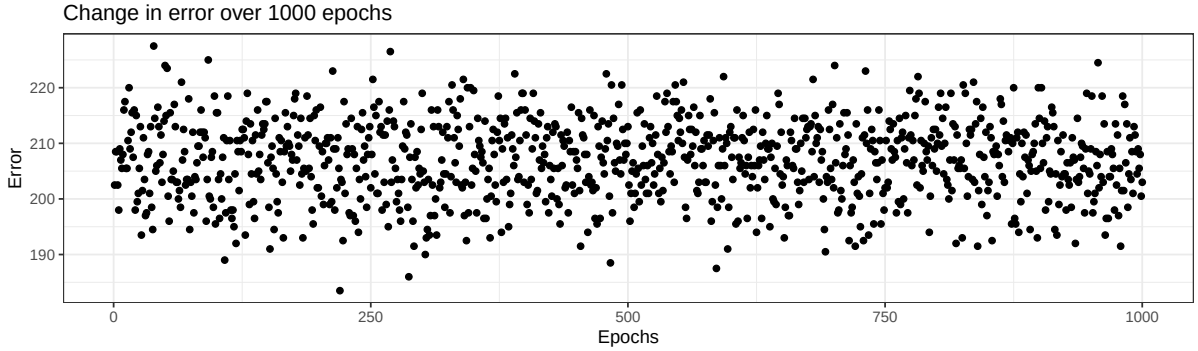
1.4 Testing

Following this I validated the perceptron on the test set. This outputs the error which was:

```
## [1] 0
```

1.5 Perceptron to test product of points

I then used the perceptron to test the parity of points using the same optimised parameters as above. However error did not reduce (see figure) - this is not a linearly separable problem so it would need more than one layer in the perceptron to solve this problem.



2 Training a multilayer perceptron

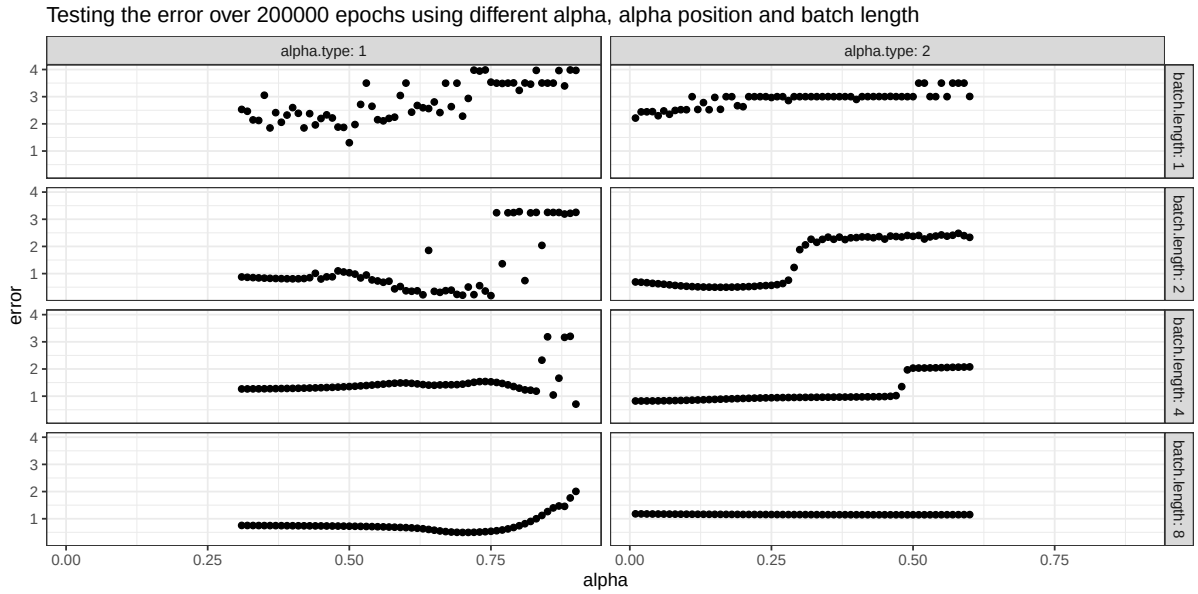
The question asked to train a multilayer neural network with 8 input units, 3 hidden units and 8 output units. I adapted the code from xor.R.

I set up the network so it can be used with different batch lengths, epsilon, rate of reduction of epsilon and momentum. In the script I put a momentum term (α) in 2 different places - updated online (α_1) and updated at the end of each batch (α_2). In this case (α_1) was only applied to the hidden unit weights whilst (α_2) was applied to the output and hidden unit activations. I then tested this network with a number of different hyperparameters to optimise its learning.

2.1 Testing network

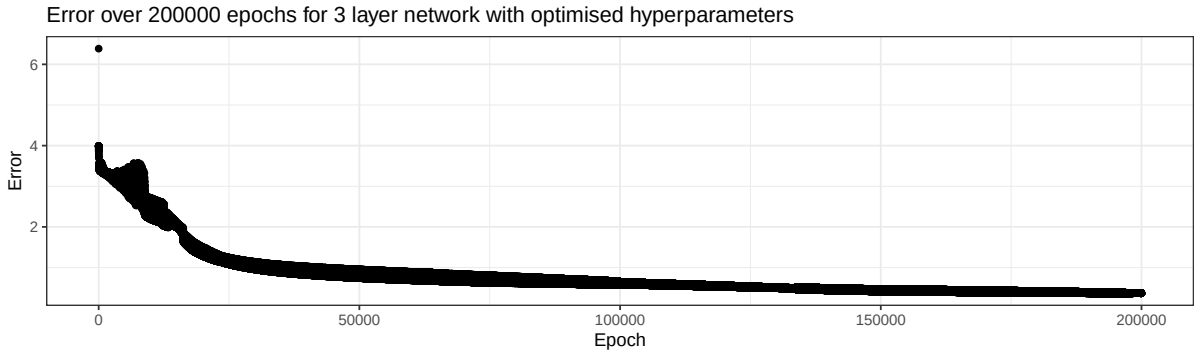
I tested the network with different batch lengths and momentum terms to see for which set of parameters the network trained best. As you can see below using a batch of 2, the α_1 momentum term set to between

0.6 and 0.75 the network trains the best.



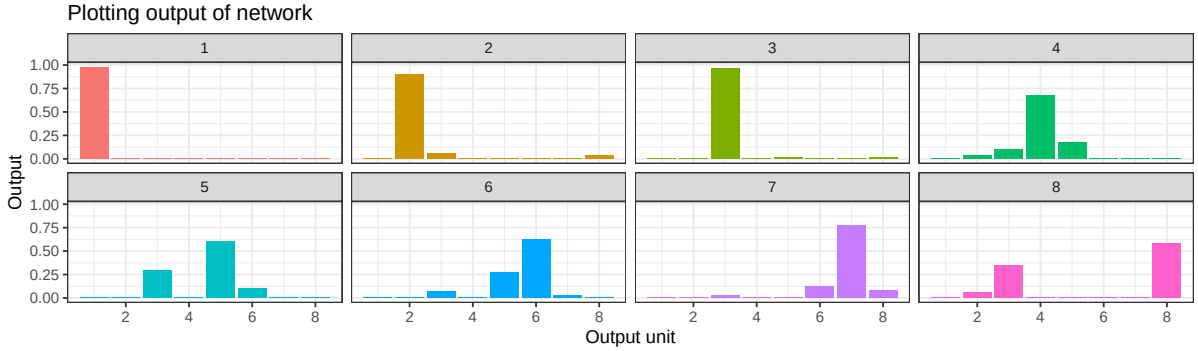
Similarly to above I tested different epsilon and rate of reduction in epsilon. I found that the best starting epsilon was 4.5 with it reducing by a factor of 0.999995 every epoch (data not shown).

Using these parameters ($\alpha_1 = 0.65$, epsilon $= 4.5$, epsilon reducing by 0.999995 every epoch) I then plotted the error for this network as it trained.



2.2 Test whether network correctly outputs for each input

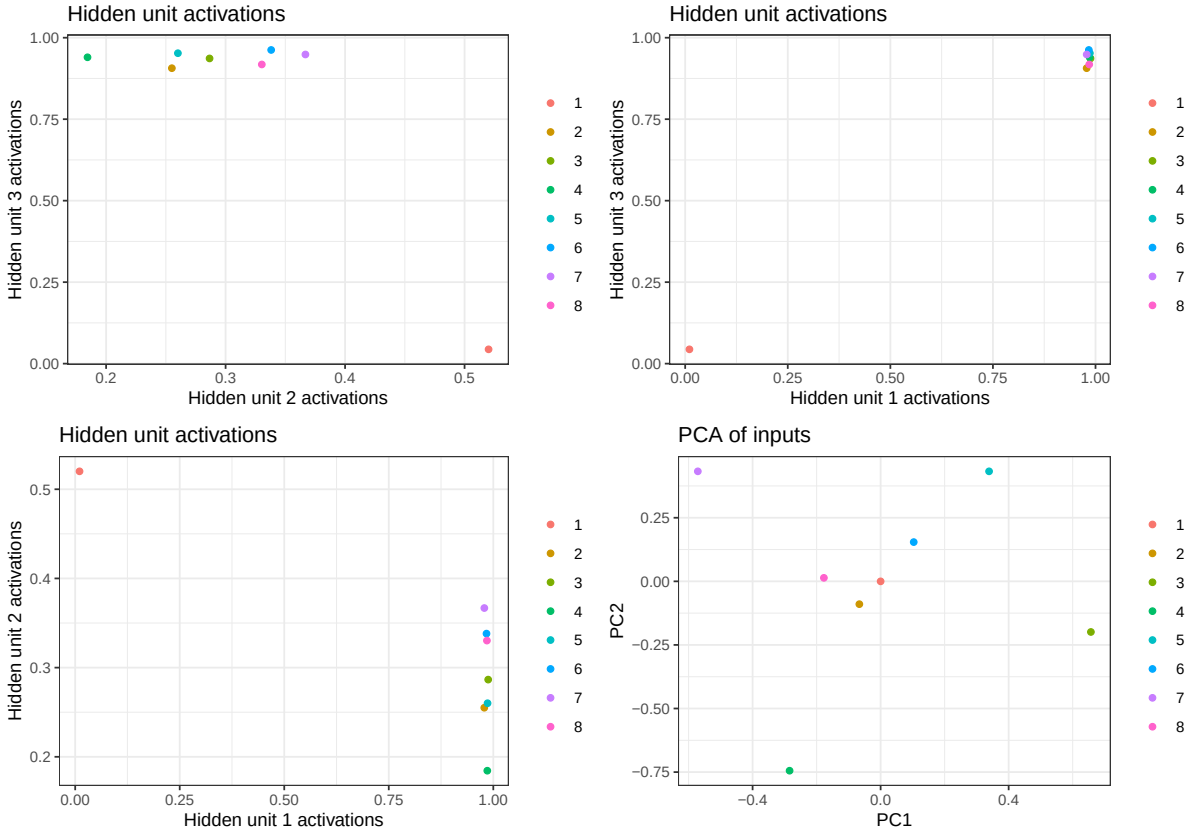
Using the optimum parameters I tested the trained network to see whether it allocates correctly. Here you can see that it correctly outputs for every input when trained using the above parameters with the above error.



2.3 Visualising internal activations

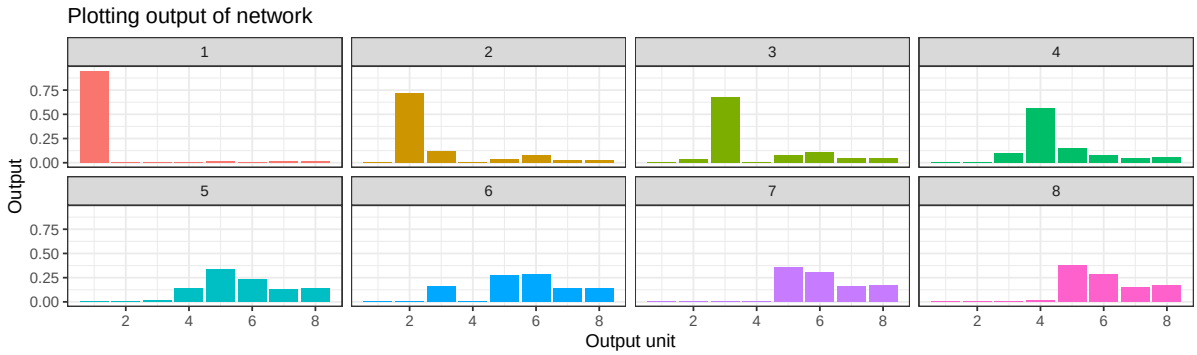
To visualise how the trained network works I plotted the activations of the hidden layer. I compared the performance of the trained autoencoder to that of a PCA. Here you can see that two of the hidden units

seem to do the same thing and only one of the hidden units differentiates between all of the different inputs significantly. By comparison the PCA is able to separate all of the inputs. However the network nonetheless is able to correctly encode all of the inputs.



2.4 Testing with 2 hidden units

As 2 of the hidden units seem to be doing the same thing in this case I then trained the network using only 2 hidden units. However over the same number of epochs it trained significantly less well and was not able to allocate correctly. Therefore it seems that all 3 hidden units are needed for training. However I did not optimise for this setup so optimising this setup would likely improve this. Another approach to this would be training with 3 hidden units then removing one which is duplicated then retraining the network. Additionally, the weights in this network were much larger so using an L2 regulariser might improve the performance of this network.



3 Training a network on Keras

I used the keras package in R to train a network on the MNIST dataset. I started with the code in the MNIST.Rmd file.

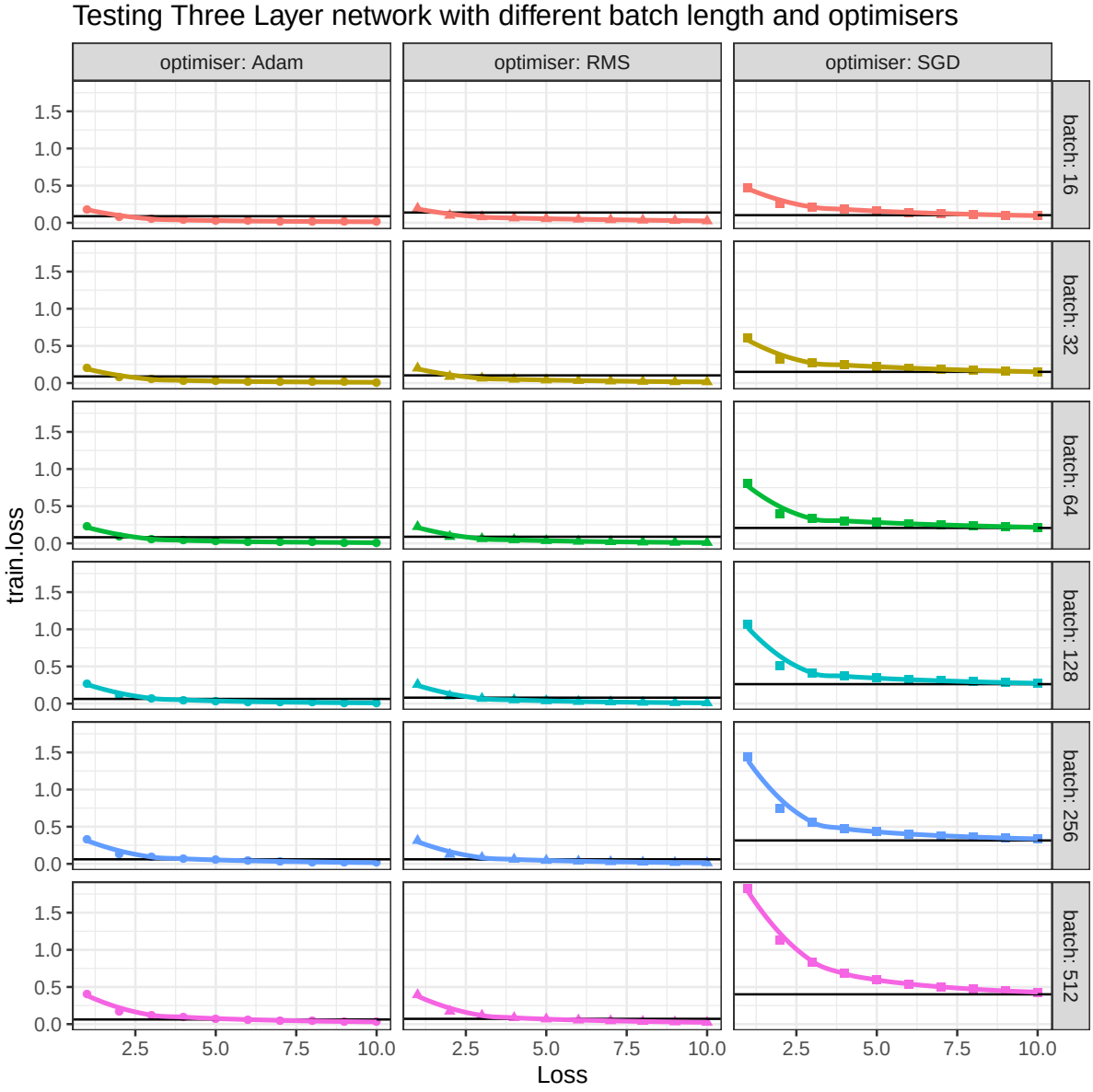
3.1 Neural network with 2 layers

First I trained a 2 layer network with a number of different parameters (data not shown). I found that the optimiser Adam is the most effective and a smaller batch size causes the network to train more quickly, but that netowrks trained with a smaller batch size generalised less well

3.2 Network with hidden layer

I repeated the above for a network with a hidden layer. Here you can see that adding in the hidden layer causes the network to train more slowly. Adam remains the most effective optimiser and a smaller batch size again causes the network to train more quickly. However the larger batch size produces the network that has the best generalisability (black line is loss on test set).

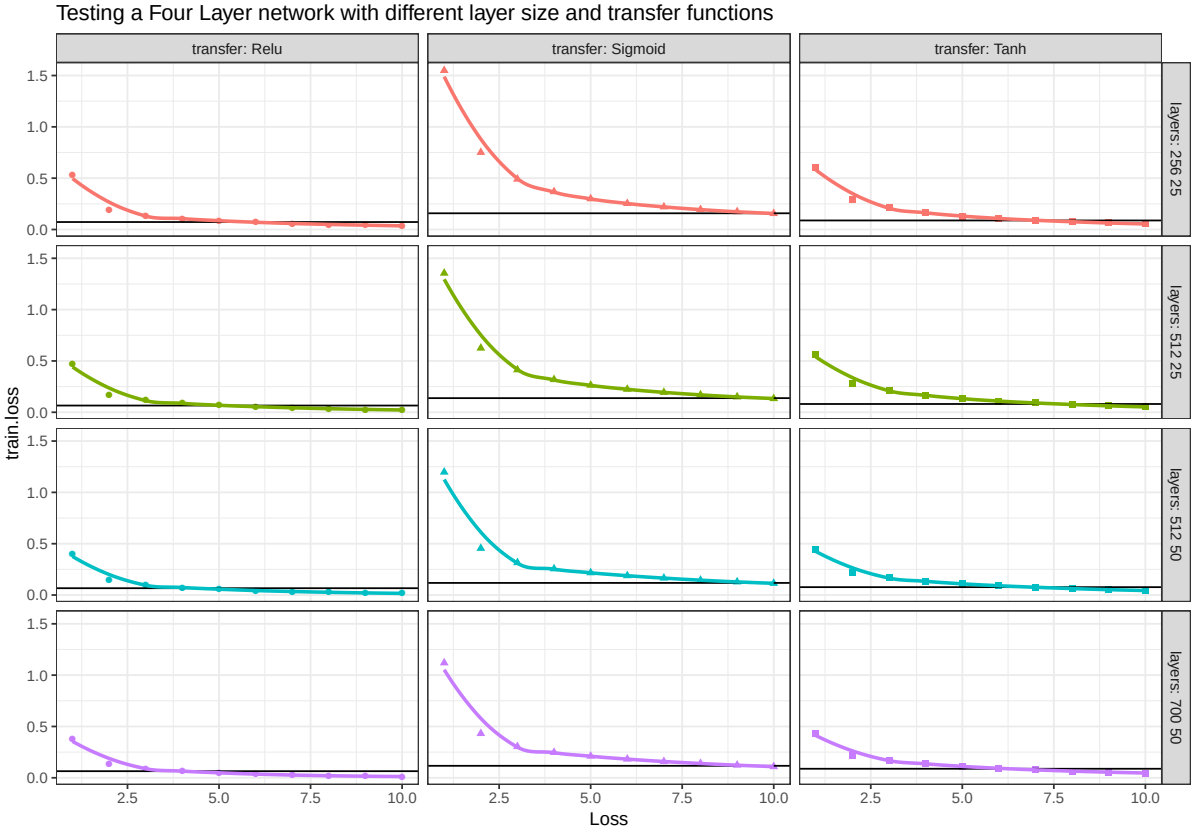
For the network using the optimiser adam and a batch size of 512 the loss was 0.06675 and the accuracy was 0.9797 on the test set.



3.3 Neural network with 2 hidden layers

I next trained a neural network with 2 hidden layers. This time I tested changing the transfer function and the number of hidden units in the hidden layers. I used a large batch size and the optimiser Adam as these performed best in the previous network.

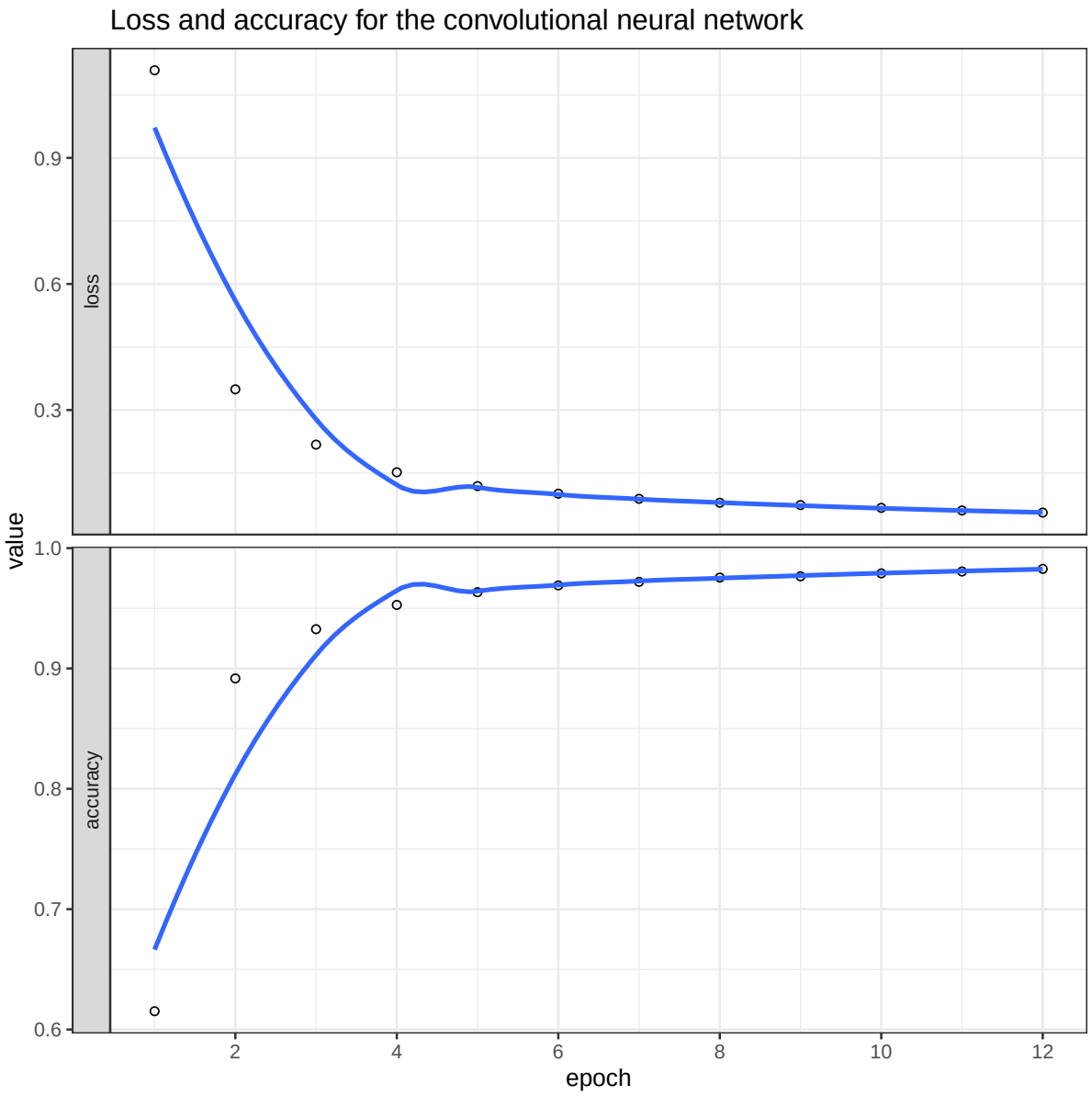
Here we can see that the of the transfer function Relu seems to perform best, but is only slightly better than tanh, and the larger layers (first hidden layer with 700 or 512 units, second with 50) seems to be the best set up. On the test set these set ups performed almost identically had a loss of 0.063 and an accuracy of 0.981 (note that this can vary slightly between identical training runs). This network performed only slightly better than the network with only one hidden layer.



3.4 Training a convolutional neural network

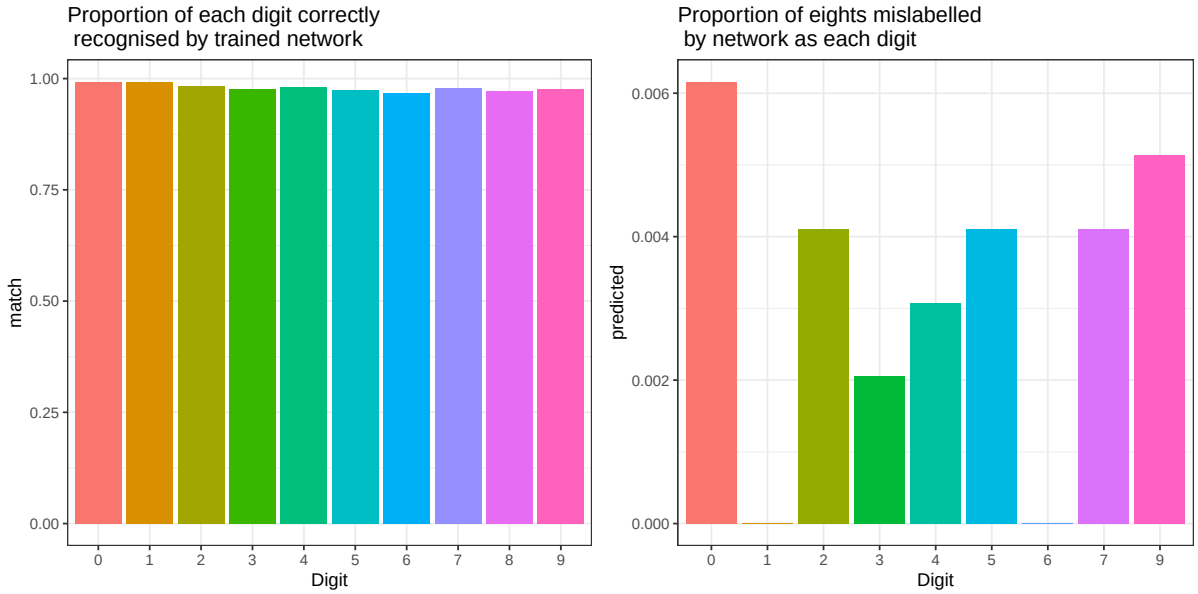
Next I trained a convolutation neural network to label MNIST. I used the CNN from here https://tensorflow.rstudio.com/tu as a model for this. This had an accuracy of 0.9655, which meant it performed slightly less well than the neural network without convolutional layers. This is possibly because simple low resolution images in MNIST need less spacial preservation of elements than more complex images.

```
## `geom_smooth()` using formula 'y ~ x'
```



3.5 Finding which letters are hardest to catagorise

Finally I used the optimum paramemeters for the 3 layer neural network to see how well the network labels MNIST images and which it is most likely to get wrong. The most poorly recognised digit was 8, followed by 7, 9 and 4. While 7, 9 and 4 might be confused for each other I wanted to investigate what digits 8 was confused for. I have plotted here the digits which 8 is most commonly mislabelled as.



3.6 Other hyperparameters to change

There are other parameters that I did not change but could be experimented with. I could have used more layers to test if this improved performance. I could have used more transfer functions or optimiser functions to test with. I could have changed the convolutional network more, including changing the size of the convolution, changing the number of layers, changing the number or size of the pooling layers. I also could have changed the output function to another function such as softplus.

4 All code used

```
# Function to make all possible permutations
permutations <- function(inputs, length = NA) {
    if (is.na(length)) {
        length <- length(inputs)
    }
    perm <- matrix(nrow = length, ncol = length(inputs)^length)
    for (i in 1:length) {
        perm[i, ] <- rep(inputs, each = length(inputs)^(i - 1), length = length(inputs)^length)
    }
    return(perm)
}

all.data <- permutations(c(1, -1), 10)
results <- apply(all.data, 2, sum)
results <- ifelse(results > 0, 1, 0)

# Split data into training and testing
training.cols <- sample(length(results), size = length(results) * 0.8)
training <- all.data[, training.cols]
results.train <- results[training.cols]
testing <- all.data[, -training.cols]
results.test <- results[-training.cols]

# Set weights and epsilon
weights <- rep(c(1, 0), 5)

perceptron <- function(weights, epsilon, epsilon_degrad, epochs, error_print = FALSE,
    sd_print = FALSE) {

    errors <- rep(NULL, times = epochs)
    sd_weights <- rep(NULL, times = epochs)

    # Make perceptron function
    for (iteration in 1:epochs) {
        order = sample(length(results.train))
        error = 0

        # Reduce epsilon each epoch, make sure it doesn't fall below floating point
        # erro
        epsilon <- epsilon * epsilon_degrad
        ifelse(epsilon < 1e-16, epsilon <- 1e-16, epsilon <- epsilon)

        for (i in order) {
            x = training[, i]
            t = results.train[i]
            y = sum(x * weights)
            y <- y > 0
            error = error + (0.5 * (t - y)^2)
            dw <- epsilon * (t - y) * x
            weights <- weights + dw
        }

        # Round weights
        weights <- round(weights, digits = 6)
        errors[iteration] <- error
```

```
        sd_weights[iteration] <- sd(weights)

        # Print error every so often
        if (iteration %in% seq(epochs/1000, epochs, by = epochs/1000)) {
            # print(error) print(sd(weights))
        }
        if (error == 0)
            break
    }

    # Use output options to output metrics we are interested in
    if (error_print == T) {
        if (sd_print == F) {
            return(list(error, weights, iteration, errors))
        } else {
            return(list(error, weights, iteration, errors, sd_weights))
        }
    } else {
        if (sd_print == F) {
            return(list(error, weights, iteration))
        } else {
            return(list(error, weights, iteration, sd_weights))
        }
    }
}

# Set weights and epsilon
set.seed(1)
weights <- runif(10)

epsilon.seq <- seq(0.01, 1, by = 0.01)
change.epsilon <- mclapply(epsilon.seq, perceptron, weights = weights, 0.5,
    10000, mc.cores = 16)
change.epsilon <- matrix(unlist(change.epsilon), nrow = 12)
a <- ggplot() + geom_point(aes(x = epsilon.seq, y = change.epsilon[12, ])) +
    theme_bw() + # geom_smooth(aes(x = epsilon.seq, y = change.epsilon[12,]), se = F) +
xlab("Epsilon") + ylab("Epochs") + labs(title = "Change in Epsilon vs \n Number of Epochs needed to
    ylim(c(0, 50))

# Use the degrad value
degrad.seq <- seq(0.5, 0.8, by = 0.01)
change.degrad <- mclapply(degrad.seq, perceptron, weights = weights, epsilon = 0.01,
    10000, mc.cores = 16)
change.degrad <- matrix(unlist(change.degrad), nrow = 12)
b <- ggplot() + geom_point(aes(x = degrad.seq, y = change.degrad[12, ])) + # geom_smooth(aes(x = deg
theme_bw() + xlab("Reduction factor of Epsilon") + ylab("Epochs") + labs(title = "Change in Rate of
    ylim(c(0, 50))

errors <- perceptron(weights, 0.01, 0.5, 1000, error_print = T, sd_print = TRUE)
c <- ggplot() + geom_point(aes(y = errors[4][[1]], x = 1:length(errors[4][[1]]))) +
    theme_bw() + xlab("Epochs") + ylab("Error") + labs(title = "Change in error over course of train

d <- ggplot() + geom_point(aes(y = errors[5][[1]], x = 1:length(errors[5][[1]]))) +
    theme_bw() + xlab("Epochs") + ylab("Standard deviation of weights") + labs(title = "Change in st

grid.arrange(a, b, c, d, nrow = 2)
```

```r
# Test function:

test.perceptron <- function(weights) {

    order = sample(length(results.test))
    error = 0

    for (i in order) {
        x = testing[, i]
        t = results.test[i]
        y = sum(x * weights)
        y <- y > 0
        error = error + (0.5 * (t - y)^2)
    }

    return(error)
}

test.perceptron(errors[2][[1]])



g <- function(x) {
    1/(1 + exp(-x))
}

## be careful about interpreting gprime!!!
gprime <- function(x) {
    y <- g(x)
    y * (1 - y)
}

# curve(g, xlim=c(-3, 3)) curve(gprime, xlim=c(-3, 3))
bias = -1

new.data <- matrix(rep(c(1, rep(0, 8)), length.out = 64), 8, 8, byrow = TRUE)
new.data.bias <- cbind(new.data, bias)

inputs <- t(new.data.bias)
targets = new.data


I = 8  #excluding bias
K = 8

multilayer_perceptron <- function(nepoch, J, alpha1, alpha2, batch.len, epsilon,
    epsilon_degrad, seed = 1, error_show = F) {
    set.seed(seed)
    W1 = matrix(runif(J * (I + 1)), J, I + 1)  #+1 for bias
    W2 = matrix(runif(K * (J + 1)), K, J + 1)


    z_j = matrix(0, J + 1, 1)
    delta_j = rep(0, J)

    errors = rep(0, nepoch)
    prev_DW1 = matrix(0, J, I + 1)
    prev_DW2 = matrix(0, K, J + 1)
```

```
for (epoch in 1:nepoch) {
    DW1 = matrix(0, J, I + 1)
    DW2 = matrix(0, K, J + 1)

    epoch_err = 0

    # Reduce value of epsilon each epoch
    epsilon <- epsilon * epsilon_degrad

    # Update the weights each batch
    for (batch in 1:(ncol(inputs)/batch.len)) {

        iterator <- seq((batch - 1) * batch.len + 1, batch * batch.len)

        for (i in iterator) {
            ## forward activation
            z_i = inputs[, i, drop = FALSE]  # keep as col vector
            t_k = targets[, i, drop = FALSE]

            ## input to hidden
            x_j = W1 %*% z_i

            for (q in 1:J) {
              z_j[q] = g(x_j[q])
            }
            z_j[J + 1] = bias


            ## hidden to output

            x_k = W2 %*% z_j
            z_k = g(x_k)

            error = sum(0.5 * (t_k - z_k)^2)

            epoch_err = epoch_err + error

            ## backward error propagation.
            delta_k = gprime(x_k) * (t_k - z_k)
            DW2 = DW2 + outer(as.vector(delta_k), as.vector(z_j))


            ## Now get deltas for hidden layer.
            delta_j_prev <- delta_j
            for (q in 1:J) {

              # First method for having momentum
              delta_j[q] = gprime(x_j[q]) * delta_k[1] * W2[1, q] + alpha1 *
                delta_j_prev[q]
            }

            DW1 = DW1 + outer(delta_j, as.vector(z_i))
        }

        # Include momentum term here
        W2 = W2 + (epsilon * (DW2 + alpha2 * prev_DW2))
        W1 = W1 + (epsilon * (DW1 + alpha2 * prev_DW1))
```

```
            prev_DW1 <- DW1
            prev_DW2 <- DW2
        }


        ## end of an epoch.
        errors[epoch] = epoch_err
        if ((epoch%%50) == 0) {
            # print(epoch_err)
        }
        ## print(W1)
    }
    if (error_show == TRUE) {
        return(list(W1, W2, epoch_err, errors))
    } else {
        return(list(W1, W2, epoch_err))
    }
}

# Batch of 2
alpha1.seq <- seq(0.31, 0.9, 0.01)
change.alpha1 <- mclapply(alpha1.seq, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha2 = 0, 2, 4, 0.999995, mc.cores = 32)
change.alpha1.mat <- matrix(unlist(change.alpha1), nrow = 60)
# ggplot() + geom_point(aes(x = alpha1.seq, y =
# change.alpha1.mat[dim(change.alpha1.mat)[1],]))

alpha2.seq <- seq(0.01, 0.6, 0.01)
change.alpha2 <- mclapply(alpha2.seq, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha1 = 0, 2, 4, 0.999995, mc.cores = 32)
change.alpha2.mat <- matrix(unlist(change.alpha2), nrow = 60)
# ggplot() + geom_point(aes(x = alpha2.seq, y =
# change.alpha2.mat[dim(change.alpha2.mat)[1],]))

# Batch of 4
alpha1.seq4 <- seq(0.31, 0.9, 0.01)
change.alpha1.4 <- mclapply(alpha1.seq4, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha2 = 0, 4, 4, 0.999995, mc.cores = 32)
change.alpha1.4mat <- matrix(unlist(change.alpha1.4), nrow = 60)
# ggplot() + geom_point(aes(x = alpha1.seq4, y =
# change.alpha1.4mat[dim(change.alpha1.4mat)[1],]))

alpha2.seq4 <- seq(0.01, 0.6, 0.01)
change.alpha2.4 <- mclapply(alpha2.seq4, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha1 = 0, 4, 4, 0.999995, mc.cores = 32)
change.alpha2.4mat <- matrix(unlist(change.alpha2.4), nrow = 60)
# ggplot() + geom_point(aes(x = alpha2.seq4, y =
# change.alpha2.4mat[dim(change.alpha2.4mat)[1],]))

# Batch of 1
alpha1.seq1 <- seq(0.31, 0.9, 0.01)
change.alpha1.1 <- mclapply(alpha1.seq1, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha2 = 0, 1, 4, 0.999995, mc.cores = 32)
change.alpha1.1mat <- matrix(unlist(change.alpha1.1), nrow = 60)
# ggplot() + geom_point(aes(x = alpha1.seq1, y =
# change.alpha1.1mat[dim(change.alpha1.1mat)[1],]))
```

```r
alpha2.seq1 <- seq(0.01, 0.6, 0.01)
change.alpha2.1 <- mclapply(alpha2.seq1, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha1 = 0, 1, 4, 0.999995, mc.cores = 32)
change.alpha2.1mat <- matrix(unlist(change.alpha2.1), nrow = 60)
# ggplot() + geom_point(aes(x = alpha2.seq1, y =
# change.alpha2.1mat[dim(change.alpha2.1mat)[1],]))

# Batch of 8
alpha1.seq8 <- seq(0.31, 0.9, 0.01)
change.alpha1.8 <- mclapply(alpha1.seq8, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha2 = 0, 8, 4, 0.999995, mc.cores = 32)
change.alpha1.8mat <- matrix(unlist(change.alpha1.8), nrow = 60)
# ggplot() + geom_point(aes(x = alpha1.seq8, y =
# change.alpha1.8mat[dim(change.alpha1.1mat)[1],]))

alpha2.seq8 <- seq(0.01, 0.6, 0.01)
change.alpha2.8 <- mclapply(alpha2.seq8, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha1 = 0, 8, 4, 0.999995, mc.cores = 32)
change.alpha2.8mat <- matrix(unlist(change.alpha2.8), nrow = 60)
# ggplot() + geom_point(aes(x = alpha2.seq8, y =
# change.alpha2.8mat[dim(change.alpha2.1mat)[1],]))

change.alpha.batch <- data.frame(alpha = c(alpha1.seq1, alpha2.seq1, alpha1.seq,
    alpha2.seq, alpha1.seq4, alpha2.seq4, alpha1.seq8, alpha2.seq8), error = c(change.alpha1.1mat[di
    ], change.alpha2.1mat[dim(change.alpha2.1mat)[1], ], change.alpha1.mat[dim(change.alpha1.mat)[1]
    ], change.alpha2.mat[dim(change.alpha2.mat)[1], ], change.alpha1.4mat[dim(change.alpha1.4mat)[1]
    ], change.alpha2.4mat[dim(change.alpha2.4mat)[1], ], change.alpha1.8mat[dim(change.alpha1.8mat)[
    ], change.alpha2.8mat[dim(change.alpha2.8mat)[1], ]), batch.length = rep(c(1,
    2, 4, 8), each = length(alpha1.seq1) * 2), alpha.type = rep(c(1, 2, 1, 2,
    1, 2, 1, 2), each = length(alpha1.seq1)))

ggplot(change.alpha.batch) + geom_point(aes(x = alpha, y = error)) + facet_grid(batch.length ~
    alpha.type, labeller = label_both) + theme_bw() + labs(title = "Testing the error over 200000 ep


# Test network using different epsilon Batch of 2
epsilon.seq <- seq(1, 7, 0.1)
epsilon.degrad <- c(0.999, 0.9999, 0.99999, 0.999992, 0.999994, 0.999996, 0.999999,
    0.9999995)
change.epsilon1 <- mclapply(epsilon.seq, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha1 = 0.57, alpha2 = 0, 2, epsilon.degrad[1], mc.cores = 32)
change.epsilon1.mat <- matrix(unlist(change.epsilon1), nrow = 60)

change.epsilon2 <- mclapply(epsilon.seq, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha1 = 0.57, alpha2 = 0, 2, epsilon.degrad[2], mc.cores = 32)
change.epsilon2.mat <- matrix(unlist(change.epsilon2), nrow = 60)

change.epsilon3 <- mclapply(epsilon.seq, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha1 = 0.57, alpha2 = 0, 2, epsilon.degrad[3], mc.cores = 32)
change.epsilon3.mat <- matrix(unlist(change.epsilon3), nrow = 60)

change.epsilon4 <- mclapply(epsilon.seq, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha1 = 0.57, alpha2 = 0, 2, epsilon.degrad[4], mc.cores = 32)
change.epsilon4.mat <- matrix(unlist(change.epsilon4), nrow = 60)

change.epsilon5 <- mclapply(epsilon.seq, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha1 = 0.57, alpha2 = 0, 2, epsilon.degrad[5], mc.cores = 32)
```

```
change.epsilon5.mat <- matrix(unlist(change.epsilon5), nrow = 60)

change.epsilon6 <- mclapply(epsilon.seq, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha1 = 0.57, alpha2 = 0, 2, epsilon.degrad[6], mc.cores = 32)
change.epsilon6.mat <- matrix(unlist(change.epsilon6), nrow = 60)

change.epsilon7 <- mclapply(epsilon.seq, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha1 = 0.57, alpha2 = 0, 2, epsilon.degrad[7], mc.cores = 32)
change.epsilon7.mat <- matrix(unlist(change.epsilon7), nrow = 60)

change.epsilon8 <- mclapply(epsilon.seq, multilayer_perceptron, nepoch = 2e+05,
    J = 3, alpha1 = 0.57, alpha2 = 0, 2, epsilon.degrad[8], mc.cores = 32)
change.epsilon8.mat <- matrix(unlist(change.epsilon8), nrow = 60)

change.epsilon.degrad <- data.frame(epsilon = rep(epsilon.seq, times = 8), error = c(change.epsilon1
    ], change.epsilon2.mat[dim(change.epsilon2.mat)[1], ], change.epsilon3.mat[dim(change.epsilon3.m
    ], change.epsilon4.mat[dim(change.epsilon4.mat)[1], ], change.epsilon5.mat[dim(change.epsilon5.m
    ], change.epsilon6.mat[dim(change.epsilon6.mat)[1], ], change.epsilon7.mat[dim(change.epsilon7.m
    ], change.epsilon8.mat[dim(change.epsilon8.mat)[1], ]), epsilon.degrad)

ggplot(change.epsilon.degrad) + geom_point(aes(x = epsilon, y = error)) + facet_grid(epsilon.degrad)
    theme_bw() + labs(title = "Testing the error over 200000 epochs using different epsilon and rate


errors <- multilayer_perceptron(2e+05, 3, 0.65, 0, 2, 4.5, 0.999995, error_show = TRUE)
ggplot() + geom_point(aes(x = 1:length(errors[4][[1]]), y = errors[4][[1]])) +
    theme_bw() + # geom_smooth(aes(x = 1:length(errors[4][[1]]), y = errors[4][[1]]), se =
# FALSE) +
xlab("Epoch") + ylab("Error") + labs(title = "Error over 200000 epochs for 3 layer network with opti

W1 <- errors[[1]]
W2 <- errors[[2]]
J <- 3
outputs <- matrix(nrow = 8, ncol = 8)
z_j = matrix(0, J + 1, 1)

for (i in 1:ncol(inputs)) {
    z_i = inputs[, i, drop = FALSE]  # keep as col vector
    t_k = targets[, i, drop = FALSE]

    ## input to hidden
    x_j = W1 %*% z_i

    for (q in 1:J) {
        z_j[q] = g(x_j[q])
    }
    z_j[J + 1] = bias


    ## hidden to output

    x_k = W2 %*% z_j
    z_k = g(x_k)
    outputs[, i] <- z_k/sum(z_k)
}

input2output <- data.frame(inputs = rep(c(1:8), each = 8), output.loc = rep(c(1:8),
```

```
    times = 8), output = as.vector(outputs))

ggplot(input2output) + geom_col(aes(x = output.loc, y = output, fill = as.factor(inputs))) +
    theme_bw() + facet_wrap(~inputs, nrow = 2) + ylab("Output") + xlab("Output unit") +
    labs(title = "Plotting output of network") + theme(legend.position = "none")



errors <- multilayer_perceptron(2e+05, J = 2, 0.57, 0, 2, 4, 0.999995, error_show = TRUE)
# ggplot() + geom_point(aes(x = 1:length(errors[4][[1]]), y =
# errors[4][[1]])) + theme_bw() + geom_smooth(aes(x =
# 1:length(errors[4][[1]]), y = errors[4][[1]]), se = FALSE) + xlab('Epoch')
# + ylab('Error') + labs(title = 'Error over 200000 epochs for 3 layer
# network with optimised hyperparameters')

W1 <- errors[[1]]
W2 <- errors[[2]]
J <- 2
outputs <- matrix(nrow = 8, ncol = 8)
z_j = matrix(0, J + 1, 1)

for (i in 1:ncol(inputs)) {
    z_i = inputs[, i, drop = FALSE]  # keep as col vector
    t_k = targets[, i, drop = FALSE]

    ## input to hidden
    x_j = W1 %*% z_i

    for (q in 1:J) {
        z_j[q] = g(x_j[q])
    }
    z_j[J + 1] = bias


    ## hidden to output

    x_k = W2 %*% z_j
    z_k = g(x_k)
    outputs[, i] <- z_k/sum(z_k)
}

input2output <- data.frame(inputs = rep(c(1:8), each = 8), output.loc = rep(c(1:8),
    times = 8), output = as.vector(outputs))
# Consider doing facet grid thing here?
ggplot(input2output) + geom_col(aes(x = output.loc, y = output, fill = as.factor(inputs))) +
    theme_bw() + facet_wrap(~inputs, nrow = 2) + ylab("Output") + xlab("Output unit") +
    labs(title = "Plotting output of network") + theme(legend.position = "none")


mnist <- dataset_mnist()
train_images <- mnist$train$x
train_labels <- mnist$train$y
test_images <- mnist$test$x
test_labels <- mnist$test$y

## reshape
train_images <- array_reshape(train_images, c(60000, 28 * 28))
```

```
train_images <- train_images/255  #normalize pixel to [0,1]

test_images <- array_reshape(test_images, c(10000, 28 * 28))
test_images <- test_images/255  #normalize pixel to [0,1]


## train_images is now a NxL matrix, where N=number of samples, L=28*28
train_labels <- to_categorical(train_labels)
test_labels <- to_categorical(test_labels)

# Network 1 - rms, batch change - 128, rms
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.128.rms <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 128)
network %>% evaluate(train_images, train_labels)

metrics.128.rms <- network %>% evaluate(test_images, test_labels)

# Batch 64
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.64.rms <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 64)
network %>% evaluate(train_images, train_labels)

metrics.64.rms <- network %>% evaluate(test_images, test_labels)

# Batch 32
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.32.rms <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 32)
network %>% evaluate(train_images, train_labels)

metrics.32.rms <- network %>% evaluate(test_images, test_labels)

# Batch 16
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.16.rms <- network %>% fit(train_images, train_labels, epochs = 10,
```

```
    batch_size = 16)
network %>% evaluate(train_images, train_labels)

metrics.16.rms <- network %>% evaluate(test_images, test_labels)

# Batch 256

network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.256.rms <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 256)
network %>% evaluate(train_images, train_labels)

metrics.256.rms <- network %>% evaluate(test_images, test_labels)

# Network 1 - rms, batch change - 128, rms
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "sgd", loss = "categorical_crossentropy", metrics = c("accuracy"))

learning.128.sgd <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 128)
network %>% evaluate(train_images, train_labels)

metrics.128.sgd <- network %>% evaluate(test_images, test_labels)

# Batch 64
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "sgd", loss = "categorical_crossentropy", metrics = c("accuracy"))

learning.64.sgd <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 64)
network %>% evaluate(train_images, train_labels)

metrics.64.sgd <- network %>% evaluate(test_images, test_labels)

# Batch 32
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "sgd", loss = "categorical_crossentropy", metrics = c("accuracy"))

learning.32.sgd <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 32)
network %>% evaluate(train_images, train_labels)

metrics.32.sgd <- network %>% evaluate(test_images, test_labels)

# Batch 16
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
```

```
    input_shape = c(28 * 28))

network %>% compile(optimizer = "sgd", loss = "categorical_crossentropy", metrics = c("accuracy"))

learning.16.sgd <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 16)
network %>% evaluate(train_images, train_labels)

metrics.16.sgd <- network %>% evaluate(test_images, test_labels)

# Batch 256

network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "sgd", loss = "categorical_crossentropy", metrics = c("accuracy"))

learning.256.sgd <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 256)
network %>% evaluate(train_images, train_labels)

metrics.256.sgd <- network %>% evaluate(test_images, test_labels)

# Network 1 - adam, batch change - 128, rms
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = c("accuracy"))

learning.128.adam <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 128)
network %>% evaluate(train_images, train_labels)

metrics.128.adam <- network %>% evaluate(test_images, test_labels)

# Batch 64
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = c("accuracy"))

learning.64.adam <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 64)
network %>% evaluate(train_images, train_labels)

metrics.64.adam <- network %>% evaluate(test_images, test_labels)

# Batch 32
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = c("accuracy"))

learning.32.adam <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 32)
network %>% evaluate(train_images, train_labels)
```

```
metrics.32.adam <- network %>% evaluate(test_images, test_labels)

# Batch 16
network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = c("accuracy"))

learning.16.adam <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 16)
network %>% evaluate(train_images, train_labels)

metrics.16.adam <- network %>% evaluate(test_images, test_labels)

# Batch 256

network <- keras_model_sequential() %>% layer_dense(units = 10, activation = "softmax",
    input_shape = c(28 * 28))

network %>% compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = c("accuracy"))

learning.256.adam <- network %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 256)
network %>% evaluate(train_images, train_labels)

metrics.256.adam <- network %>% evaluate(test_images, test_labels)

network1.metrics <- data.frame(epoch = rep(c(1:10), times = 15), loss = c(learning.16.rms$metrics$lo
    learning.32.rms$metrics$loss, learning.64.rms$metrics$loss, learning.128.rms$metrics$loss,
    learning.256.rms$metrics$loss, learning.16.sgd$metrics$loss, learning.32.sgd$metrics$loss,
    learning.64.sgd$metrics$loss, learning.128.sgd$metrics$loss, learning.256.sgd$metrics$loss,
    learning.16.adam$metrics$loss, learning.32.adam$metrics$loss, learning.64.adam$metrics$loss,
    learning.128.adam$metrics$loss, learning.256.adam$metrics$loss), batch = rep(c(16,
    32, 64, 128, 256), each = 10, times = 3), optimiser = rep(c("RMS", "SGD",
    "Adam"), each = 50))

ggplot(network1.metrics) + geom_point(aes(x = epoch, y = loss)) + facet_grid(batch ~
    optimiser, labeller = label_both) + theme_bw() + labs(title = "Testing 2 Layer network with diff


# Network 2 - rms, batch change - 128, rms
network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.128.rms <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 128)
network2 %>% evaluate(train_images, train_labels)

metrics.128.rms <- network2 %>% evaluate(test_images, test_labels)

# Batch 64
network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
```

```r
network2

learning.64.rms <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 64)
network2 %>% evaluate(train_images, train_labels)

metrics.64.rms <- network2 %>% evaluate(test_images, test_labels)

# Batch 32
network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
network2

learning.32.rms <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 32)
network2 %>% evaluate(train_images, train_labels)

metrics.32.rms <- network2 %>% evaluate(test_images, test_labels)

# Batch 16
network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
network2

learning.16.rms <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 16)
network2 %>% evaluate(train_images, train_labels)

metrics.16.rms <- network2 %>% evaluate(test_images, test_labels)

# Batch 256

network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
network2

learning.256.rms <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 256)
network2 %>% evaluate(train_images, train_labels)

metrics.256.rms <- network2 %>% evaluate(test_images, test_labels)

# Batch 512

network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "rmsprop", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
network2

learning.512.rms <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 512)
```

```
network2 %>% evaluate(train_images, train_labels)

metrics.512.rms <- network2 %>% evaluate(test_images, test_labels)

# Network 2 - rms, batch change - 128, rms
network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "sgd", loss = "categorical_crossentropy", metrics = c("accuracy"))
network2

learning.128.sgd <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 128)
network2 %>% evaluate(train_images, train_labels)

metrics.128.sgd <- network2 %>% evaluate(test_images, test_labels)

# Batch 64
network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "sgd", loss = "categorical_crossentropy", metrics = c("accuracy"))
network2

learning.64.sgd <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 64)
network2 %>% evaluate(train_images, train_labels)

metrics.64.sgd <- network2 %>% evaluate(test_images, test_labels)

# Batch 32
network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "sgd", loss = "categorical_crossentropy", metrics = c("accuracy"))
network2

learning.32.sgd <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 32)
network2 %>% evaluate(train_images, train_labels)

metrics.32.sgd <- network2 %>% evaluate(test_images, test_labels)

# Batch 16
network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "sgd", loss = "categorical_crossentropy", metrics = c("accuracy"))
network2

learning.16.sgd <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 16)
network2 %>% evaluate(train_images, train_labels)

metrics.16.sgd <- network2 %>% evaluate(test_images, test_labels)

# Batch 256

network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "sgd", loss = "categorical_crossentropy", metrics = c("accuracy"))
```

```
network2

learning.256.sgd <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 256)
network2 %>% evaluate(train_images, train_labels)

metrics.256.sgd <- network2 %>% evaluate(test_images, test_labels)

# Batch 512

network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "sgd", loss = "categorical_crossentropy", metrics = c("accuracy"))
network2

learning.512.sgd <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 512)
network2 %>% evaluate(train_images, train_labels)

metrics.512.sgd <- network2 %>% evaluate(test_images, test_labels)

# network2 - adam, batch change - 128, rms
network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
network2

learning.128.adam <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 128)
network2 %>% evaluate(train_images, train_labels)

metrics.128.adam <- network2 %>% evaluate(test_images, test_labels)

# Batch 64
network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
network2

learning.64.adam <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 64)
network2 %>% evaluate(train_images, train_labels)

metrics.64.adam <- network2 %>% evaluate(test_images, test_labels)

# Batch 32
network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
network2

learning.32.adam <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 32)
network2 %>% evaluate(train_images, train_labels)
```

```
metrics.32.adam <- network2 %>% evaluate(test_images, test_labels)

# Batch 16
network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
network2

learning.16.adam <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 16)
network2 %>% evaluate(train_images, train_labels)

metrics.16.adam <- network2 %>% evaluate(test_images, test_labels)

# Batch 256

network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
network2

learning.256.adam <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 256)
network2 %>% evaluate(train_images, train_labels)

metrics.256.adam <- network2 %>% evaluate(test_images, test_labels)

# Batch 512

network2 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 10, activation = "softmax")
network2 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
network2

learning.512.adam <- network2 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 512)
network2 %>% evaluate(train_images, train_labels)

metrics.512.adam <- network2 %>% evaluate(test_images, test_labels)


network2.metrics <- data.frame(epoch = rep(c(1:10), times = 18), train.loss = c(learning.16.rms$metr
    learning.32.rms$metrics$loss, learning.64.rms$metrics$loss, learning.128.rms$metrics$loss,
    learning.256.rms$metrics$loss, learning.512.rms$metrics$loss, learning.16.sgd$metrics$loss,
    learning.32.sgd$metrics$loss, learning.64.sgd$metrics$loss, learning.128.sgd$metrics$loss,
    learning.256.sgd$metrics$loss, learning.512.sgd$metrics$loss, learning.16.adam$metrics$loss,
    learning.32.adam$metrics$loss, learning.64.adam$metrics$loss, learning.128.adam$metrics$loss,
    learning.256.adam$metrics$loss, learning.512.adam$metrics$loss), test.loss = rep(c(metrics.16.rm
    metrics.32.rms[[1]], metrics.64.rms[[1]], metrics.128.rms[[1]], metrics.256.rms[[1]],
    metrics.512.rms[[1]], metrics.16.sgd[[1]], metrics.32.sgd[[1]], metrics.64.sgd[[1]],
    metrics.128.sgd[[1]], metrics.256.sgd[[1]], metrics.512.sgd[[1]], metrics.16.adam[[1]],
    metrics.32.adam[[1]], metrics.64.adam[[1]], metrics.128.adam[[1]], metrics.256.adam[[1]],
    metrics.512.adam[[1]]), each = 10), batch = rep(c(16, 32, 64, 128, 256,
    512), each = 10, times = 3), optimiser = rep(c("RMS", "SGD", "Adam"), each = 60))
```

```
ggplot(network2.metrics) + geom_point(aes(x = epoch, y = train.loss, col = as.factor(batch),
    shape = optimiser)) + facet_grid(batch ~ optimiser, labeller = label_both) +
    theme_bw() + labs(title = "Testing Three Layer network with different batch length and optimiser
    geom_hline(aes(yintercept = test.loss)) + geom_smooth(aes(x = epoch, y = train.loss,
    col = as.factor(batch)), method = "loess", formula = "y  ~ x", se = FALSE) +
    theme(legend.position = "none") + xlab("Loss")

# Relu - 512 - 50
network3 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 50, activation = "relu") %>%
    layer_dense(units = 10, activation = "softmax")
network3 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.relu.51250 <- network3 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 500)
network3 %>% evaluate(train_images, train_labels)
metrics.relu.51250 <- network3 %>% evaluate(test_images, test_labels)

# Relu 256 25

network3 <- keras_model_sequential() %>% layer_dense(units = 256, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 25, activation = "relu") %>%
    layer_dense(units = 10, activation = "softmax")
network3 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.relu.25625 <- network3 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 500)
network3 %>% evaluate(train_images, train_labels)
metrics.relu.25625 <- network3 %>% evaluate(test_images, test_labels)
# Relu

network3.use <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 25, activation = "relu") %>%
    layer_dense(units = 10, activation = "softmax")
network3.use %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.relu.51225 <- network3.use %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 500)
network3.use %>% evaluate(train_images, train_labels)
metrics.relu.51225 <- network3.use %>% evaluate(test_images, test_labels)

# Relu

network3 <- keras_model_sequential() %>% layer_dense(units = 700, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 50, activation = "relu") %>%
    layer_dense(units = 10, activation = "softmax")
network3 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.relu.70050 <- network3 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 500)
network3 %>% evaluate(train_images, train_labels)
metrics.relu.70050 <- network3 %>% evaluate(test_images, test_labels)
```

```
# tanh - 512 - 50
network3 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "tanh",
    input_shape = c(28 * 28)) %>% layer_dense(units = 50, activation = "tanh") %>%
    layer_dense(units = 10, activation = "softmax")
network3 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.tanh.51250 <- network3 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 500)
network3 %>% evaluate(train_images, train_labels)
metrics.tanh.51250 <- network3 %>% evaluate(test_images, test_labels)

# tanh 256 25

network3 <- keras_model_sequential() %>% layer_dense(units = 256, activation = "tanh",
    input_shape = c(28 * 28)) %>% layer_dense(units = 25, activation = "tanh") %>%
    layer_dense(units = 10, activation = "softmax")
network3 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.tanh.25625 <- network3 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 500)
network3 %>% evaluate(train_images, train_labels)
metrics.tanh.25625 <- network3 %>% evaluate(test_images, test_labels)

# tanh

network3 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "tanh",
    input_shape = c(28 * 28)) %>% layer_dense(units = 25, activation = "tanh") %>%
    layer_dense(units = 10, activation = "softmax")
network3 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.tanh.51225 <- network3 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 500)
network3 %>% evaluate(train_images, train_labels)
metrics.tanh.51225 <- network3 %>% evaluate(test_images, test_labels)

# tanh

network3 <- keras_model_sequential() %>% layer_dense(units = 700, activation = "tanh",
    input_shape = c(28 * 28)) %>% layer_dense(units = 50, activation = "tanh") %>%
    layer_dense(units = 10, activation = "softmax")
network3 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.tanh.70050 <- network3 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 500)
network3 %>% evaluate(train_images, train_labels)
metrics.tanh.70050 <- network3 %>% evaluate(test_images, test_labels)

# sigmoid - 512 - 50
network3 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "sigmoid",
    input_shape = c(28 * 28)) %>% layer_dense(units = 50, activation = "sigmoid") %>%
    layer_dense(units = 10, activation = "softmax")
network3 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))
```

```
learning.sigmoid.51250 <- network3 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 500)
network3 %>% evaluate(train_images, train_labels)
metrics.sigmoid.51250 <- network3 %>% evaluate(test_images, test_labels)

# sigmoid 256 25

network3 <- keras_model_sequential() %>% layer_dense(units = 256, activation = "sigmoid",
    input_shape = c(28 * 28)) %>% layer_dense(units = 25, activation = "sigmoid") %>%
    layer_dense(units = 10, activation = "softmax")
network3 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.sigmoid.25625 <- network3 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 500)
network3 %>% evaluate(train_images, train_labels)
metrics.sigmoid.25625 <- network3 %>% evaluate(test_images, test_labels)

# sigmoid

network3 <- keras_model_sequential() %>% layer_dense(units = 512, activation = "sigmoid",
    input_shape = c(28 * 28)) %>% layer_dense(units = 25, activation = "sigmoid") %>%
    layer_dense(units = 10, activation = "softmax")
network3 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.sigmoid.51225 <- network3 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 500)
network3 %>% evaluate(train_images, train_labels)
metrics.sigmoid.51225 <- network3 %>% evaluate(test_images, test_labels)

# sigmoid

network3 <- keras_model_sequential() %>% layer_dense(units = 700, activation = "sigmoid",
    input_shape = c(28 * 28)) %>% layer_dense(units = 50, activation = "sigmoid") %>%
    layer_dense(units = 10, activation = "softmax")
network3 %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

learning.sigmoid.70050 <- network3 %>% fit(train_images, train_labels, epochs = 10,
    batch_size = 500)
network3 %>% evaluate(train_images, train_labels)
metrics.sigmoid.70050 <- network3 %>% evaluate(test_images, test_labels)

network3.metrics <- data.frame(epoch = rep(c(1:10), times = 12), train.loss = c(learning.relu.25625$
    learning.relu.51225$metrics$loss, learning.relu.51250$metrics$loss, learning.relu.70050$metrics$
    learning.tanh.25625$metrics$loss, learning.tanh.51225$metrics$loss, learning.tanh.51250$metrics$
    learning.tanh.70050$metrics$loss, learning.sigmoid.25625$metrics$loss, learning.sigmoid.51225$me
    learning.sigmoid.51250$metrics$loss, learning.sigmoid.70050$metrics$loss),
    test.loss = rep(c(metrics.relu.25625[[1]], metrics.relu.51225[[1]], metrics.relu.51250[[1]],
        metrics.relu.70050[[1]], metrics.tanh.25625[[1]], metrics.tanh.51225[[1]],
        metrics.tanh.51250[[1]], metrics.tanh.70050[[1]], metrics.sigmoid.25625[[1]],
        metrics.sigmoid.51225[[1]], metrics.sigmoid.51250[[1]], metrics.sigmoid.70050[[1]]),
        each = 10), layers = rep(c("256 25", "512 25", "512 50", "700 50"),
        each = 10, times = 3), transfer = rep(c("Relu", "Tanh", "Sigmoid"),
        each = 40))
```

```r
ggplot(network3.metrics) + geom_point(aes(x = epoch, y = train.loss, col = as.factor(layers),
    shape = transfer)) + facet_grid(layers ~ transfer, labeller = label_both) +
    theme_bw() + labs(title = "Testing a Four Layer network with different layer size and transfer f
    geom_hline(aes(yintercept = test.loss)) + geom_smooth(aes(x = epoch, y = train.loss,
    col = as.factor(layers)), method = "loess", formula = "y  ~ x", se = FALSE) +
    theme(legend.position = "none") + xlab("Loss")



train_images_CNN <- mnist$train$x
test_images_CNN <- mnist$test$x

## reshape
train_images_CNN <- array_reshape(train_images_CNN, c(60000, 28, 28, 1))
train_images_CNN <- train_images_CNN/255  #normalize pixel to [0,1]

test_images_CNN <- array_reshape(test_images_CNN, c(10000, 28, 28, 1))
train_images_CNN <- train_images_CNN/255  #normalize pixel to [0,1]


model <- keras_model_sequential() %>% layer_conv_2d(filters = 32, kernel_size = c(3,
    3), activation = "relu", input_shape = c(28, 28, 1)) %>% layer_max_pooling_2d(pool_size = c(2,
    2)) %>% layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
    layer_max_pooling_2d(pool_size = c(2, 2)) %>% layer_conv_2d(filters = 64,
    kernel_size = c(3, 3), activation = "relu") %>% layer_flatten() %>% layer_dense(units = 64,
    activation = "relu") %>% layer_dense(units = 10, activation = "softmax") %>%
    compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = "accuracy")

history_tf <- model %>% fit(train_images_CNN, train_labels, epochs = 12, batch_size = 128)
history2_tf <- model %>% evaluate(train_images_CNN, train_labels)

metrics_cnn_tf <- model %>% evaluate(test_images_CNN, test_labels)

plot(history_tf) + theme_bw() + labs(title = "Loss and accuracy for the convolutional neural network



network3.use <- keras_model_sequential() %>% layer_dense(units = 512, activation = "relu",
    input_shape = c(28 * 28)) %>% layer_dense(units = 25, activation = "relu") %>%
    layer_dense(units = 10, activation = "softmax")
network3.use %>% compile(optimizer = "adam", loss = "categorical_crossentropy",
    metrics = c("accuracy"))

network3.use %>% fit(train_images, train_labels, epochs = 10, batch_size = 500)

predictions <- network3.use %>% predict_classes(test_images)

testing.network <- data.frame(predictions, labels = mnist$test$y)

# Split by number
percent.predict <- data.frame(match = rep(NA, 10), digit = 0:9)
for (i in 1:10) {
    locs <- which(testing.network$labels == (i - 1))
    matches <- which(testing.network$predictions[locs] == testing.network$labels[locs])
    percent.predict$match[i] <- length(matches)/length(locs)
}
a <- ggplot(percent.predict) + geom_col(aes(x = as.factor(digit), y = match,
```

```
    fill = as.factor(digit))) + theme_bw() + theme(legend.position = "none") +
    labs(title = "Proportion of each digit correctly \n recognised by trained network") +
    xlab("Digit")

# See what 8 was predicted as
eight.predict <- data.frame(predicted = rep(NA, 10), digit = 0:9)
eights <- which(testing.network$labels == 8)
for (i in 1:10) {
    eight.predict$predicted[i] <- length(which(testing.network$predictions[eights] ==
        (i - 1)))/length(eights)
}

b <- ggplot(eight.predict[-9, ]) + geom_col(aes(x = as.factor(digit), y = predicted,
    fill = as.factor(digit))) + theme_bw() + theme(legend.position = "none") +
    labs(title = "Proportion of eights mislabelled \n by network as each digit") +
    xlab("Digit")

grid.arrange(a, b, nrow = 1)
```