Deep Learning – Exercise 1

Salomon Benhamo 332651918

Yarden Carmel 208520288

Question 1 – Fully connected neural network

We have a mini-batch of size m. Each sample has 10 input features, there is one hidden layer with 50 neurons and an output layer with 3 neurons. All neurons use ReLU.

Section a - Shape of the input matrix X

$$X \in \mathbb{R}^{m \times 10}$$

Each of the m rows is one sample; each row has 10 input features.

Section b - Shapes of the hidden layer parameters

Let $W^{(h)}$ be the weight matrix of the hidden layer and $b^{(h)}$ its bias.

$$W_h \in \mathbb{R}^{10 \times 50}$$
$$b_h \in R^{1 \times 50}$$

The input size is 10 and the hidden layer output size is 50. The weight matrix maps the input features to the hidden neurons, and the bias vector corresponds to the number of hidden neurons.

Section c - Shapes of the output layer parameters

Let $W^{(o)}$ be the weight matrix of the output layer and $b^{(o)}$ its bias.

$$W_0 \in \mathbb{R}^{50 \times 3}$$
$$b_0 \in \mathbb{R}^{1 \times 3}$$

The hidden layer (size 50) connects to the output layer (size 3)

Section d - Shape of the network's output matrix Y

$$Y \in \mathbb{R}^{m \times 3}$$

For a batch of m examples, the network produces 3 output values for each example4.

Section e - Explicit expression for Y as a function of X, W_h, b_h, W_0, b_0

Let $ReLU(z) = max(0, z)$ applied elementwise and let $1_m \in \mathbb{R}^{m\times 1}$ be a column vector of ones (used to broadcast the biases). Then:

Hidden layer pre-activation:

$$Z_h = XW_h + 1_m b_h$$

Hidden layer activation:

$$H = ReLU(Z_h)$$

Output layer pre-activation:

$$Z_o = HW_0 + 1_m b_o$$

Output layer activation:

$$Y = ReLU\left(Z^{(o)}\right)$$

Equivalently,

$$Y = ReLU(ReLU(XW_h + b_h)W_0 + b_o)$$

where b_h and b_o are broadcast along the batch dimension.

Question 2 – Convolutional network parameter count

We have three convolutional layers. Each use:

- 3x3 kernels
- stride 2
- SAME padding

The number of feature maps (channels) is:

- Layer 1 (lowest): 100 output feature maps
- Layer 2 (middle): 200 output feature maps
- Layer 3 (top): 400 output feature maps

The input images are RGB, with 3 channels and spatial size 200×300.

For a convolutional layer with:

- C_{in} input channels
- C_{out} output channels (filters)

- kernel size K×K
  and one bias per output channel, the number of parameters is

$$C_{out} \cdot (C_{in} \cdot K \cdot K + 1)$$

Layer 1

C_{in}= 3 (RGB)
C_{out}= 100
K = 3

$$weights = 3 \cdot 3 \cdot 3 \cdot 100 = 27 \cdot 100 = 2{,}700$$
$$biases = 100$$

Total parameters in layer 1:

$$2{,}700 + 100 = 2{,}800$$

Layer 2

Input channels are the 100 feature maps from layer 1.

C_{in}= 100
C_{out}= 200
K = 3

$$weights = 3 \cdot 3 \cdot 100 \cdot 200 = 9 \cdot 100 \cdot 200 = 180{,}000$$
$$biases = 200$$

Total parameters in layer 2:

$$180{,}000 + 200 = 180{,}200$$

Layer 3

Input channels are the 200 feature maps from layer 2.

C_{in} = 200
C_{out} = 400
K = 3

$$weights = 3 \cdot 3 \cdot 200 \cdot 400 = 9 \cdot 200 \cdot 400 = 720{,}000$$
$$biases = 400$$

Total parameters in layer 3:

$$720{,}000 + 400 = 720{,}400$$

Total number of parameters in the three convolutional layers

$$= 2{,}800 + 180{,}200 + 720{,}400$$
$$= 903{,}400\ parameters$$

Question 3 – Batch Normalization gradients

We consider 1D Batch Normalization for a single scalar feature with a mini-batch of size m. Let the inputs to BN be x_1, …, x_m. The forward pass is:

batch mean:

$$\mu_B = (1/m)\sum_{i=1}^{m} x_i$$

batch variance:

$$\sigma_B^2 = (1/m)\sum_{i=1}^{m}(x_i - \mu_B)^2$$

normalized input:

$$\hat{x}_i = \frac{(x_i - \mu_B)}{\sqrt{(\sigma_B^2 + \varepsilon)}}$$

BN output:

$$y_i = \gamma\hat{x}_i + \beta$$

We assume ε is a small constant and does not depend on x.

We are given the upstream gradients $\partial L/\partial y_i$ for i = 1, …, m.

Define for convenience:

$$g_i^y = \partial f/\partial y_i$$
$$g_i^{\hat{x}} = \partial f/\partial \hat{x}_i$$

$\partial f/\partial \gamma$

From $y_i = \gamma\hat{x}_i + \beta$ we have $\partial y_i/\partial\gamma = \hat{x}_i$, so

$$\partial f/\partial\gamma = \sum_{i=1}^{m}(\partial f/\partial y_i)(\partial y_i/\partial\gamma) = \sum_{i=1}^{m} g_i^y\hat{x}_i$$

$\partial L/\partial \beta$

From $y_i = \gamma \hat{x}_i + \beta$ we have $\partial y_i/\partial \beta = 1$, hence

$$\partial f/\partial \beta = \sum_{i=1}^{m} (\partial f/\partial y_i)(\partial y_i/\partial \beta) = \sum_{i=1}^{m} g_i^y$$

$\partial L/\partial \hat{x}_i$

Again, from $y_i = \gamma \hat{x}_i + \beta$,

$$\partial y_i/\partial \hat{x}_i = \gamma$$

so for each i,

$$\partial f/\partial \hat{x}_i = (\partial f/\partial y_i)(\partial y_i/\partial \hat{x}_i) = g_i^y \cdot \gamma$$

In other words, $g_i^{\hat{x}} = \gamma g_i^y$.

$\partial L/\partial \sigma^2$

We have:

$$\hat{x}_i = \frac{(x_i - \mu_B)}{\sqrt{(\sigma_B^2 + \varepsilon)}}$$

Treating μ_B as constant when differentiating w.r.t. σ^2,

$$\partial \hat{x}_i/\partial \sigma^2 = (x_i - \mu) \cdot \frac{\partial\left[(\sigma^2 + \varepsilon)^{-1/2}\right]}{\partial \sigma^2} = (x_i - \mu_B) \cdot \left(-\frac{1}{2}\right)(\sigma^2 + \varepsilon)^{-3/2}$$

Using the chain rule,

$$\partial f/\partial \sigma^2 = \sum_{i=1}^{m} (\partial f/\partial \hat{x}_i)(\partial \hat{x}_i/\partial \sigma_B^2) = \sum_{i=1}^{m} g_i^{\hat{x}} (x_i - \mu_B)\left(-\frac{1}{2}\right)(\sigma^2 + \varepsilon)^{-3/2}$$

Therefore,

$$\partial f/\partial \sigma^2 = -\frac{1}{2}\sum_{i=1}^{m} g_i^{\hat{x}} (x_i - \mu_B)(\sigma^2 + \varepsilon)^{-3/2}$$

$\partial f/\partial \mu$

μ influences the loss via $\hat{x}_i$ and via σ^2. Using the chain rule,

$$\partial f/\partial \mu_B = \sum_{i=1}^{m} (\partial f/\partial \hat{x}_i)(\partial \hat{x}_i/\partial \mu_B) + (\partial f/\partial \sigma_B^2)(\partial \sigma_B^2/\partial \mu_B)$$

First,

$$\hat{x}_i = \frac{(x_i - \mu_B)}{\sqrt{(\sigma_B^2 + \varepsilon)}}$$

So, treating σ² as constant,

$$\partial \hat{x}_i/\partial \mu = -1/\sqrt{(\sigma^2 + \varepsilon)}$$

Also,

$$\sigma^2 = (1/m)\sum_{j=1}^{m}\left(x_j - \mu\right)^2$$

Hence:

$$\partial \sigma^2/\partial \mu = (1/m)\sum_{j=1}^{m} 2\left(x_j - \mu\right)(-1) = -(2/m)\sum_{j=1}^{m}\left(x_j - \mu\right)$$

But by definition of μ, $\sum_{j=1}^{m}\left(x_j - \mu_B\right) = 0$, so the second term vanishes.

Therefore,

$$\partial f/\partial \mu = \sum_{i=1}^{m} g_i^{\hat{x}}\left(-1/\sqrt{(\sigma^2 + \varepsilon)}\right) = -\sum_{i=1}^{m} g_i^{\hat{x}}/\sqrt{(\sigma^2 + \varepsilon)}$$

$\partial L/\partial x_i$

Each x_i influences the loss through $\hat{x}_i$, σ² and μ. Using the chain rule,

$$\partial f/\partial x_i = (\partial f/\partial \hat{x}_i)(\partial \hat{x}_i/\partial x_i) + (\partial f/\partial \sigma^2)(\partial \sigma^2/\partial x_i) + (\partial f/\partial \mu)(\partial \mu/\partial x_i)$$

We have:

$$\mu = (1/m)\sum_{j=1}^{m} x_j \Rightarrow \partial\mu/\partial x_i = 1/m$$

using the fact that $\sum_j (x_j - \mu_B) = 0$

$$\sigma^2 = (1/m)\sum_{j=1}^{m} (x_j - \mu)^2 \Rightarrow \partial\sigma^2/\partial x_i = 2(x_i - \mu)/m$$

Also,

$$\hat{x}_i = \frac{(x_i - \mu_B)}{\sqrt{(\sigma_B^2 + \varepsilon)}}$$

so, treating μ and σ² as independent intermediate variables,

$$\partial\hat{x}_i/\partial x_i = 1/\sqrt{(\sigma^2 + \varepsilon)}$$

Combining everything,

$$\partial f/\partial x_i = g_i^{\hat{x}} \cdot 1/\sqrt{(\sigma^2 + \varepsilon)} + (\partial f/\partial\sigma^2) \cdot 2(x_i - \mu)/m + (\partial f/\partial\mu) \cdot 1/m$$

written directly in terms of sums,

$$\frac{\partial f}{\partial x_i} = \frac{\left[g_i^{\hat{x}} - \left(\frac{1}{m}\right)\sum_{j=1}^{m} g_j^{\hat{x}}\right]}{\sqrt{(\sigma^2 + \varepsilon)}} - \frac{(x_i - \mu)}{(\sigma^2 + \varepsilon)^{3/2}} \cdot \frac{1}{m}\sum_{j=1}^{m} g_j^{\hat{x}} (x_j - \mu)$$

This expression uses only the upstream gradients $g_j^{\hat{x}}$ (and hence g_j^{y}) and the forward-pass quantities x_j, μ and σ².

Question 4

To see full code, clone it and run it, see:

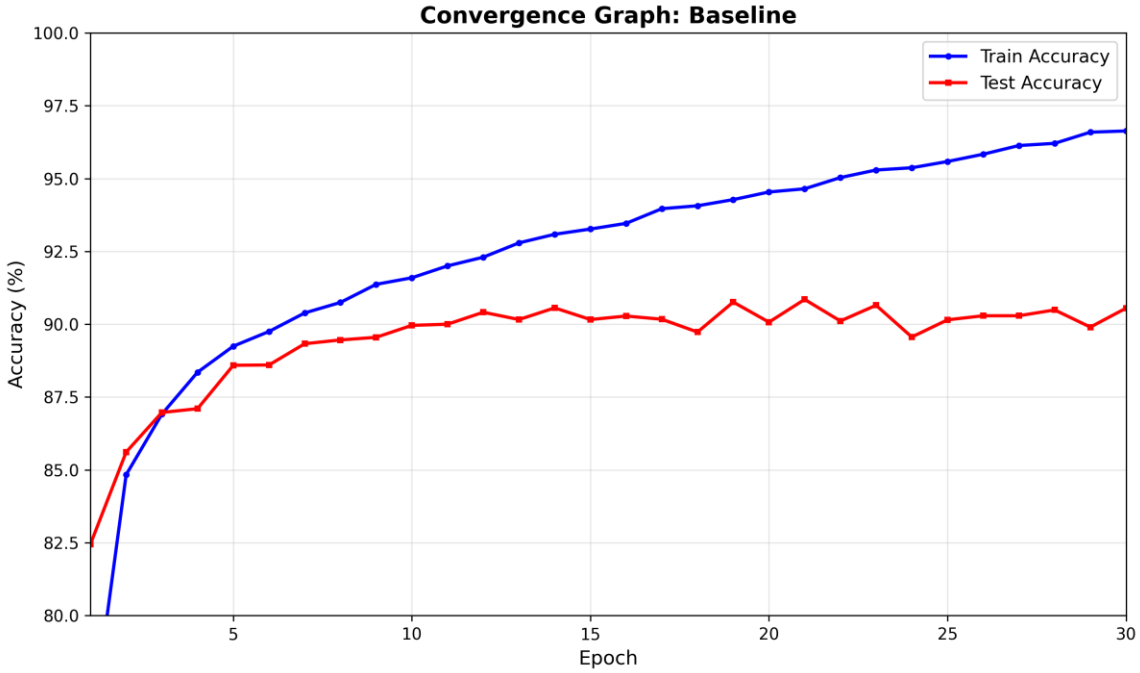
https://github.com/yardencarmel/deep-learning-course

We trained a LeNet-5 style CNN on Fashion-MNIST (28×28 grayscale images, 10 classes) with four different configurations:

1. Baseline – no explicit regularization.
2. Dropout – dropout in the fully-connected layers.
3. Weight decay (L2) – L2 penalty via the optimizer.
4. Batch normalization (BN) – BN after each convolutional and fully-connected layer.

All models were trained with the same optimizer (Adam, lr = 0.001), batch size 128 and 30 epochs, so the curves are directly comparable.

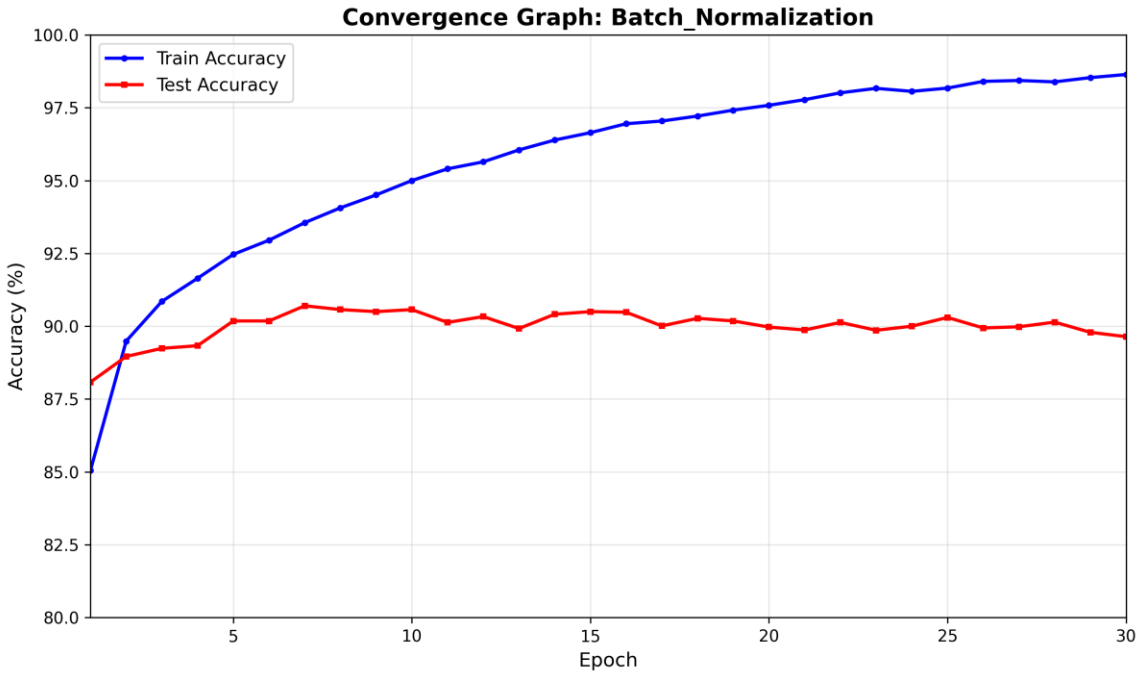
Baseline



The baseline model converges quickly: training accuracy jumps from ~80% to above 90% within the first few epochs and then slowly approaches ≈96% by epoch 30. Test accuracy rises to around 90–91% and then stabilizes.
There is a consistent train–test gap of roughly 5–6 percentage points, which

indicates some overfitting, but not severe. This setup already achieves around 90%+ test accuracy without any regularization.

Batch Normalization

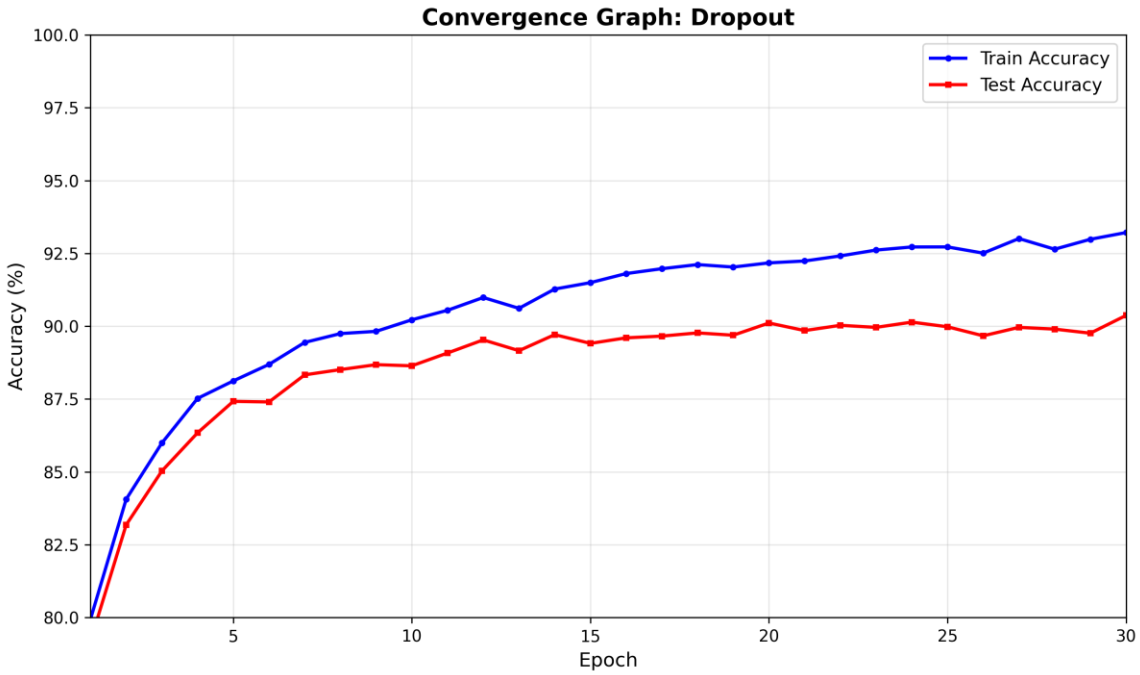


With batch normalization, training converges faster and to a higher value: the train curve reaches above 90% within ~3 epochs and ends close to 98–99%. This shows that BN makes optimization easier and stabilizes the gradients.

However, the test accuracy does not improve accordingly. It stays around ~90% and is slightly below or comparable to the baseline across most epochs, despite the much higher training accuracy. The train–test gap is now even larger.

In conclusion, this setting (relatively small LeNet-5 on a not-too-hard dataset), BN mainly helps optimization, not generalization. It allows the network to fit the training data better but does not translate into noticeably higher test accuracy; if anything, it slightly increases overfitting.

Dropout

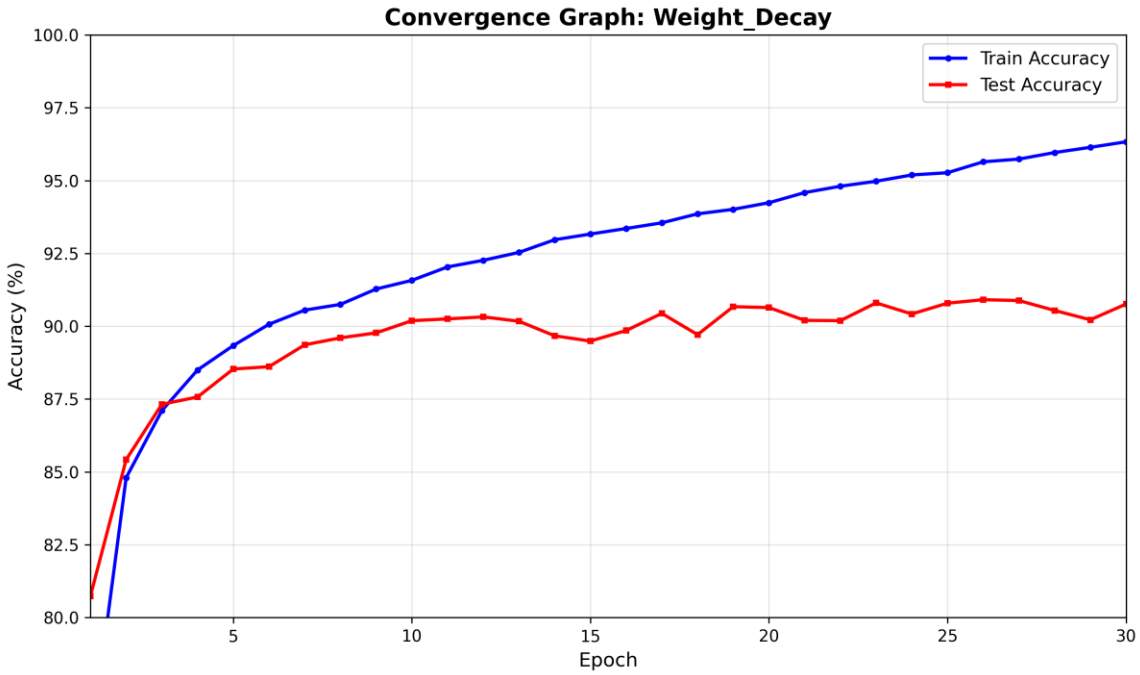


With dropout, the training accuracy is consistently lower than the baseline, ending around ≈93–94% instead of ≈96%. This is expected: randomly dropping units during training makes the effective model noisier and harder to fit perfectly.

The test accuracy, however, is very similar to the baseline (around 90–91%), sometimes slightly below but with a smaller gap between train and test. The curves are smoother, and the gap is reduced from ~5–6 percentage points (baseline) to only ~3–4.

In conclusion: dropout works as intended as a regularizer. It limits the network's capacity to overfit the training set (lower train accuracy) while keeping test accuracy essentially unchanged, slightly improving the generalization gap.

Weight Decay (L2)



With weight decay, the training curve is close to the baseline but saturates a bit lower (≈95–96% instead of ≈96+%). The test accuracy is again around 90–91%, very similar to or marginally better than the baseline, and the train–test gap is slightly smaller.

Because L2 penalizes large weights, it discourages overly complex solutions and pushes the model toward smoother decision boundaries. In the graphs, this shows up as:

- Slightly reduced training accuracy compared to baseline.
- Test accuracy that is comparable to the best configuration and at least as stable as the baseline.

In conclusion: weight decay provides mild regularization. It does not dramatically change performance, but it slightly reduces overfitting without hurting test accuracy.

Overall comparison

- All four models reach very similar test accuracies (~90–91%). None of the techniques dramatically outperforms the others in terms of final test accuracy on Fashion-MNIST with this architecture.
- Batch normalization gives the fastest and highest training accuracy but does not improve test accuracy; it mainly helps optimization.
- Dropout and weight decay both act as true regularizers: they lower training accuracy yet keep test accuracy roughly the same as the baseline and slightly reduce the generalization gap.
- Since the baseline already generalizes well on this task, the main effect of the regularization methods is to change the trade-off between train accuracy, test accuracy and overfitting, rather than to boost test performance by several percentage points.

In summary, on this relatively small LeNet-5/Fashion-MNIST experiment, batch normalization improves convergence speed and training performance, while dropout and weight decay primarily control overfitting. The test accuracy remains around 90% across all configurations, showing that the problem is already well-posed and that regularization provides more subtle improvements in robustness rather than dramatic gains in accuracy.