CPEN455: Deep Learning
Homework 1

Nolan McCleary - 33634312

Submitted: 2025 Sep. 27

Note: We use lowercase bold letters to represent vectors and capital letters to represent matrices. Checking shapes is always helpful to recognize vectors, matrices, and high order tensors, and understand the operations that are applied to them. $\mathbf{x}[i]$ means the i-th element of a vector $\mathbf{x}$ and $X[i, j]$ means the (i, j)-th element of a matrix X. $X[i, :]$ means the i-th row of a matrix X. We can index over high order tensors similarly.

- **It is required to use the notations given in problem descriptions and follow the above conventions**.
- **It is required to use provided LaTeX template to finish the homework**.

1 Dropout [25pts]

Let us consider a single hidden layer MLP with M hidden units. Suppose the input vector $\mathbf{x} \in \mathbb{R}^{N\times 1}$. The hidden activations $\mathbf{h} \in \mathbb{R}^{M\times 1}$ are computed as follows,

$$\mathbf{h} = \sigma(W\mathbf{x} + \mathbf{b}) \tag{1}$$

where weight matrix $W \in \mathbb{R}^{M\times N}$, bias vector $\mathbf{b} \in \mathbb{R}^{M\times 1}$, and σ is the nonlinear activation function.

Dropout [1] is a technique to help reduce overfitting for neural networks. In PyTorch, Dropout is implemented as follows. During training, we independently zero out elements of $\mathbf{h}$ with probability p and then rescale it with $\frac{1}{1-p}$, *i.e.*,

$$\tilde{\mathbf{h}} = \frac{\mathbf{m}}{1-p} \odot \mathbf{h} \tag{2}$$

$$\mathbf{m}[i] \sim \text{Bernoulli}(1-p) \qquad \forall i = 1, \ldots, M, \tag{3}$$

where $\odot$ is the Hadamard product (*a.k.a.*, element-wise product) and Bernoulli$(1-p)$ is the Bernoulli distribution where the random variable takes the value 1 with the probability $1-p$. During testing, we just use $\tilde{\mathbf{h}} = \mathbf{h}$.

1.1 [5pts] Explain why we need to rescale the hidden activations by $\frac{1}{1-p}$.

If we don't scale down, the expected value of the hidden layer output during training will be $(1-p)$ that of the hidden layer output during testing (assuming dropout is off during testing). If this is the case, then the training process will not accurately reflect the test conditions and it's likely that the learnable parameters will not be appropriately calibrated.

1.2 [15pts] Assume $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, I)$, $\mathbf{b} = \mathbf{0}$, $WW^\top = I_M$ (I_M is an indentity matrix with size $M \times M$), and we use rectified linear units (ReLU) as the nonlinear activation function, *i.e.*, $\sigma(x) = \max(x, 0)$, derive the variance of the activations before Dropout (*i.e.*, $\mathbf{h}$) and after Dropout (*i.e.*, $\tilde{\mathbf{h}}$).

EXPECTATION BEFORE DROPOUT:

I. Given COV(x) is an Identity matrix and W is orthonormal, we can say that VAR(Wx) is also an identity matrix of dimensionality mxm i.e. I_m.

II. Given that $\mathbb{E}x = 0$ and $b = 0$, we can say that $\mathbb{E}[Wx] = 0$

III. Thus, $z = Wx \sim N(0, I_m)$

ReLU operating on z compresses the entire LHS of the distribution onto a point-mass at x=0:
$\mathbb{E}[ReLU(z)] = \mathbb{E}h_i = \int_{-\infty}^{\infty} \frac{z}{\sqrt{COV(z)*2\pi}} e^{\frac{-1}{2}\frac{z-\mu}{\sqrt{COV(z)}}^2} = \int_{0}^{\infty} \frac{z}{\sqrt{COV(z)*2\pi}} e^{\frac{-1}{2}\frac{z-\mu}{\sqrt{COV(z)}}^2}$
Where $\mu = 0, COV(z) = I_m$ and thus $\sqrt{COV(z)} = I_m$ as well.

We use u-substitution where $u = \frac{-z^2}{2}$ to evaluate the integral and end up getting:
$\mathbb{E}[ReLU(z)] = \frac{1}{\sqrt{2\pi}}(-e^{-u})|_0^\infty = \frac{1}{\sqrt{2\pi}}$

VARIANCE BEFORE DROPOUT:

VAR(x) simplifies to $\mathbb{E}x^2 - (\mathbb{E}x)^2$ as a result of operator linearity.

Evaluating $\mathbb{E}[z^2]$ with $u = \frac{z^2}{2}$ we get the following function:

$\frac{1}{\sqrt{2\pi}} \int_0^\infty u^{\frac{1}{2}} e^{-u} du$

Which allows us to substitute the following gamma function identity:
$\int_0^\infty e^{\frac{-u}{2}} du = \Gamma(\frac{3}{2}) = \frac{1}{2}\Gamma(\frac{1}{2}) = \sqrt{\pi}$

Thus: $\text{VAR}(h_i) = \text{VAR}(ReLU(z)) = \frac{1}{2} - \frac{1}{2\pi}$

EXPECTATION AFTER DROPOUT:

$\mathbb{E}[xy] = \mathbb{E}X * \mathbb{E}Y + \text{COV}[XY]$

Given that dropout is uncorrelated with our activation function, we can assume that covariance between the two is zero.

We use Hadamard mask $m_i \sim \text{Bernouli}(1-p)$ and normalize by $\frac{1}{1-p}$ as per 1.1:

$\tilde{h_i} = \frac{m_i}{1-p} \odot h_i$

Thus $\mathbb{E}[\tilde{h_i}] = (1 * (1-p) + 0 * p)\frac{1}{1-p}\mathbb{E}[h_i] = \mathbb{E}[h_i] = \frac{1}{\sqrt{2\pi}}$

VARIANCE AFTER DROPOUT:

$\mathbb{E}[\tilde{h_i}^2] = \mathbb{E}[\frac{m_i^2 h_i^2}{(1-p)^2}] = \frac{1}{(1-p)^2}\mathbb{E}m_i^2\mathbb{E}h_i^2$ once again due to activation and dropout functions being uncorrelated.

m_i is a random bernouli mask where each value inside it is randomly chosen to be either zero or one. As such, m_i^2 must equal m_i, thus:

$\mathbb{E}[\tilde{h_i}^2] = \mathbb{E}[\frac{m_i h_i^2}{(1-p)^2}] = \frac{1}{(1-p)}\mathbb{E}[h_i^2] = \frac{1}{2(1-p)}$

$\therefore \text{VAR}(\tilde{h_i}) = \frac{1}{2(1-p)} - \frac{1}{2\pi}$

1.3 [5pts] What is the expected number of hidden units that are kept (*i.e.*, those with $\mathbf{m}[i] = 1$) by Dropout? Derive the probability distribution (*i.e.*, the probability mass function) of the number of kept hidden units.

We have M hidden units, each with probability of 1-p of being kept i.e. $\mathbb{E}k = M(1-p)$.

Any given run, the number of kept hidden units falls on a line somewhere between 0 and M, the chance that any k units are activated is given by the expression $f(M,k) = \frac{M!}{(M-k)!k!}(1-p)^k p^(M-k)$ where $0 \leq k \leq m$

From the above we can clearly see that this distribution is a binomial distribution.

The first term gives the number of possible ways we can turn on k hidden units given a total number of M hidden units. The second term is the associated probability that exactly k units are on and the rest are off.

1.4 [Bonus 10pts] Assume the number of hidden units M goes to infinity and the probability of keeping units $1-p$ goes to 0 in a way that their product $M(1-p)$ stays fixed. Derive the probability distribution of the number of kept hidden units.

Let $f(M,k) = \frac{M!}{(M-k)!k!}(1-p)^k p^{M-k}$ where $0 \leq k \leq m$

We want to get $\lim_{M\to\infty} f(M,k)$

Let $u = \frac{\lambda}{m} = 1-p$ where λ is the number of cases where the neuron is activated, then

$\lim_{M\to\infty} f(M,k) = \frac{M!}{(M-k)!k!}(u)^k(1-u)^{M-k} = \lim_{M\to\infty} \frac{M!}{(M-k)!k!}\frac{\lambda^k}{M^k}(1-\frac{\lambda}{M})^M(1-\frac{\lambda}{M})^{-k}$

Now we consider the convergence properties of the following components:

1. $\lim_{M\to\infty} \frac{M!}{(M-k)!} = \lim_{M\to\infty} \prod_{i=1}^{k}(M-i+1) = \prod_{i=1}^{k} M = M^k$
2. $\lim_{M\to\infty}(1-\frac{\lambda}{M})^k = 1$
3. Given $\lim_{x\to\infty}(1+\frac{1}{x})^x = e$, let $x = \frac{-M}{\lambda}$, then $\lim_{M\to\infty}(1-\frac{\lambda}{M})^M = e^{-\lambda}$

$\therefore \lim_{M\to\infty} f(M,k) = \frac{\lambda^k e^{-\lambda}}{k!} = \textbf{Poisson}(\lambda)$

This is known convergence behavior for a binomial distribution.

1.5 [Bonus 10pts] Suppose the number of hidden units M follows a Poisson distribution with parameter λ, *i.e.*, the probability mass function is $\mathbb{P}(M=k) = \frac{\lambda^k e^{-\lambda}}{k!}$. Derive the probability distribution of the number of kept hidden units.

From the problem description, we see that $P(M=m) = \frac{\lambda^m e^{-\lambda}}{m!}$. This is the probability that we have a total of m hidden units in our given layer.

From the above, we know that $P(K=k|m=M) = \binom{m}{k}(1-p)^k p^{m-k}$ where $k \leq m$. This represents the probability that k units will be turned on given that we have m total units.

To find the distribution of kept hidden units i.e. $P(K=k)$, we need to integrate the conditional PDF with respect to all valid values of m i.e. all $m \geq k$

$P(K=k) = \sum_{m=k}^{\infty} \binom{m}{k}(1-p)^k p^{m-k} \frac{\lambda^m e^{-\lambda}}{m!}$

Using the convergent series $\sum_{r=0}^{\infty} \frac{(p\lambda)^r}{r!} = e^{p\lambda}$:

$P(K=k) = \frac{(\lambda(1-p))^k e^{(p-1)\lambda}}{k!} = \textbf{Poisson}((1-p)\lambda)$

2 Batch Normalization [35pts]

Let us again consider a single hidden layer MLP with M hidden units. Suppose we now have B input vectors packed as a matrix such that each row is a sample, *i.e.*, $X \in \mathbb{R}^{B\times N}$. The hidden activations $H \in \mathbb{R}^{B\times M}$ are computed as follows,

$$Y = XW^\top + \mathbf{b}^\top \tag{4}$$

$$H = \sigma(Y) \tag{5}$$

where weight matrix $W \in \mathbb{R}^{M\times N}$, bias vector $\mathbf{b} \in \mathbb{R}^{M\times 1}$, and σ is the nonlinear activation function. $XW^\top + \mathbf{b}^\top$ leverages the *broadcasting addition*. In particular, the shapes of $XW^\top$ and $\mathbf{b}^\top$ are $B\times M$ and $1\times M$, respectively. You can imagine that the broadcasting addition first duplicates $\mathbf{b}^\top$ for B times to create a matrix of shape $B\times M$ and then performs element-wise addition. Note that in the actual implementation, one does not perform duplication since it is a waste of memory. This is just a helpful way to understand how broadcasting operations work in a high level.

Batch normalization (BN) [2], is a method that makes training of deep neural networks faster and more stable via normalizing the inputs of individual layers by re-centering and re-scaling. In particular, if we apply BN to pre-activations[1] Y in Eq. (5), we have

$$\mathbf{m} = \frac{1}{B}\sum_{i=1}^{B} Y[i,:] \tag{6}$$

$$\mathbf{v}[j] = \frac{1}{B}\sum_{i=1}^{B}(Y[i,j] - \mathbf{m}[j])^2 \tag{7}$$

$$\hat{Y}[i,j] = \boldsymbol{\gamma}[j]\frac{Y[i,j] - \mathbf{m}[j]}{\sqrt{\mathbf{v}[j] + \epsilon}} + \boldsymbol{\beta}[j], \tag{8}$$

where the normalized input $\hat{Y} \in \mathbb{R}^{B\times M}$. $\mathbf{m} \in \mathbb{R}^{M\times 1}$ and $\mathbf{v} \in \mathbb{R}^{M\times 1}$ are the mean and the (dimension-wise) variance vectors. $\boldsymbol{\gamma} \in \mathbb{R}^{M\times 1}$ and $\boldsymbol{\beta} \in \mathbb{R}^{M\times 1}$ are learnable parameters associated with the BN layer. ϵ is a hyperparameter.

2.1 [5pts] Explain why we need the hyperparameter ϵ.

Epsilon is necessary to ensure numerical stability. Two things can happen if it is not used:

1. Given a variance $v[j]$ of exactly zero, we get a divide-by-zero error when normalizing and our system crashes.

2. Given a very small non-zero $v[j]$, our normalized variables can explode in size. This can saturate our activation functions and make it very difficult to compute the gradient.

2.2 [10pts] Derive the mean and the variance of $\hat{Y}[i,j]$ (ignore the effect of ϵ only for this question).

$\hat{m}[j] = \frac{1}{B}\sum_{i=1}^{B}\gamma[j]\frac{Y[i,j]-m[j]}{\sqrt{v[j]}} + \beta[j] = \gamma[j]\frac{m[j]-m[j]}{\sqrt{v[j]}} + \beta[j] = \beta[j]$

$\hat{v}[j] = \frac{1}{B}\sum_{i=1}^{B}(\hat{Y}[i,j] - \beta[j])^2 = \frac{\gamma^2[j]}{v[j]}\frac{1}{B}\sum_{i=1}^{B}(Y[i,j] - m[j])^2 = \gamma^2[j]$

Therefore the variance and mean of our normalized inputs are themselves learnable parameters.

2.3 [20pts] Suppose the nonlinear activation function is ReLU and the gradient of the training loss ℓ with respect to (w.r.t.) the normalized hidden activations H is $\frac{\partial \ell}{\partial H}$. Derive the gradients of the training loss w.r.t. learnable parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$, the mean $\mathbf{m}$, the variance $\mathbf{v}$, and the input Y.

Given $h = \text{ReLU}(\hat{Y})$

$$\nabla\ell = \begin{cases} 0, & \hat{Y} < 0 \\ \nabla\hat{Y}, & \hat{Y} \geq 0 \end{cases}$$

$$\nabla\hat{Y} = \frac{\partial\hat{Y}}{\partial\gamma} + \frac{\partial\hat{Y}}{\partial\beta} + \frac{\partial\hat{Y}}{\partial v} + \frac{\partial\hat{Y}}{\partial Y} + \frac{\partial\hat{Y}}{\partial m}$$

[1] We call Y pre-activations since it is before the activation function σ. Normalization operations that are applied to pre-activations are often called *pre-norm*. Similarly, one can define *post-norm*.

$$\frac{\partial \hat{Y}[i,j]}{\partial \gamma[j]} = \frac{Y[i,j]-m[j]}{\sqrt{v[j]+\epsilon}}$$

$$\frac{\partial \hat{Y}[i,j]}{\partial \beta[j]} = 1$$

$$\frac{\partial \hat{Y}[i,j]}{\partial v[j]} = \frac{-1}{2}\gamma[j](Y[i,j]-m[j])(v[j]+\epsilon)^{\frac{-3}{2}}$$

$$m[j] := \frac{1}{B}\sum_{i=1}^{B} Y[i,j] \in \mathbb{R}^{M\times 1}$$

$$\frac{\partial m[j]}{\partial Y[i,j]} = \frac{1}{B}$$

$$v := \text{DIAG}(\frac{1}{B}\sum_{i=1}^{B}(Y[i,j]-m[j])(Y[i,j]-m[j])^T) \in \mathbb{R}^{M\times 1}$$

$$\frac{\partial v[j]}{\partial m[j]} = 0$$

$$\frac{\partial v[j]}{\partial Y[i,j]} = \frac{2}{B}(Y[i,j]-m[j])$$

$$\frac{\partial \hat{Y}[i,j]}{\partial m[j]} = -\frac{\gamma}{\sqrt{v[j]+\epsilon}}$$

$$\text{Let } s = (v[j]+\epsilon)^{\frac{-1}{2}}$$

$$\therefore \frac{\partial s}{\partial Y[i,j]} = \frac{-1}{2}(v[j]+\epsilon)^{\frac{-3}{2}}\frac{2}{B}(Y[i,j]-m[j])$$

Then $\frac{\partial \hat{Y}[i,j]}{\partial Y[i,j]} = s + (Y[i,j]-m[j])\frac{\partial s}{\partial Y[i,j]} = (v[j]+\epsilon)^{\frac{-1}{2}} + (Y[i,j]-m[j])\frac{-1}{2}(v[j]+\epsilon)^{\frac{-3}{2}}\frac{2}{B}(Y[i,j]-m[j])$

2.4 [Bonus 10pts] BN only normalizes the hidden units in a dimension-wise fashion, *e.g.*, the variance is computed per-dimension. Derive the equations of BN where the covariance matrix of the normalized input $\hat{Y}$ is normalized to an identity. Compare this version of BN to the original BN and discuss the pros and cons.

Let $Y_c = Y - \mu$

Let $\Sigma = \text{COV}(Y_c) = \frac{1}{B} Y_c^T Y_c = UAU^T$ by the property of eigen-decomposition.

Then $\Sigma^{\frac{-1}{2}} = UA^{\frac{-1}{2}}U^T$ where A is the eigenvalue diagonal.

Let $\tilde{Y} = Y_c \Sigma^{\frac{-1}{2}} \in \mathbb{R}^{B \times M}$

Then $\frac{1}{B}\tilde{Y}^T Y_c = \Sigma^{\frac{-1}{2}} \Sigma \Sigma^{\frac{-1}{2}} = I_M$

PROS:

1. Better normalization of features inside hidden layers. Activations become uncorrelated with unit variance.
2. Normalizes to full identity covariance, not just per-dimension variance.

CONS:

1. Computationally expensive
2. Less numerically stable

3 Back-Propagation and Initialization [40pts]

Suppose we have a deep MLP classifier and L hidden layers as follows.

$$\mathbf{h}_i = \sigma(W_i \mathbf{h}_{i-1} + \mathbf{b}_i) \qquad \forall i = 1, \ldots, L \tag{9}$$

where the input data vector $\mathbf{h}_0 = \mathbf{x} \in \mathbb{R}^{D_0 \times 1}$ and σ is the nonlinear activation function. Weights and the bias vector of the i-th layer are $W_i \in \mathbb{R}^{D_i \times D_{i-1}}$ and $\mathbf{b}_i \in \mathbb{R}^{D_i \times 1}$ respectively. The readout function is the *softmax*. In other words, the probability of the input $\mathbf{x}$ belong to the class k is,

$$\mathbb{P}(\mathbf{x} \text{ belong to class } k) = \mathbf{y}[k] = \frac{\exp(\mathbf{h}_L[k])}{\sum_j \exp(\mathbf{h}_L[j])} \tag{10}$$

where the input to the softmax $\mathbf{h}_L \in \mathbb{R}^{D_L \times 1}$ is often called *logits*. The dimension of the logits should be the same as the total number of classes K, *i.e.*, $D_L = K$. To train the classifier, we use cross-entropy loss as below,

$$\ell(\bar{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} \bar{\mathbf{y}}[k] \log \mathbf{y}[k], \tag{11}$$

where $\bar{\mathbf{y}} \in \mathbb{R}^{K \times 1}$ is the ground-truth probability vector of the input $\mathbf{x}$. Typically, it is a one-hot encoded vector, *i.e.*, if input $\mathbf{x}$ of the training set belongs to the class c, then $\bar{\mathbf{y}}[c] = 1$ and $\bar{\mathbf{y}}[k] = 0, \forall k \neq c$.

N.B.: For simplicity, we only consider the case of a single sample in the above context, *i.e.*, the batch dimesnion is ignored.

3.1 [15pts] Derive the gradients of loss ℓ w.r.t. all hidden activations $\mathbf{h}_i$. In particular, for any layer $1 \leq i \leq L$, you need to show how to derive $\frac{\partial \ell}{\partial \mathbf{h}_i}$. You need to write down the shapes of all tensors involved in the final expressions.

$$\frac{\partial \ell}{\partial h_i} = \prod_{j=i+1}^{L} \frac{\partial h_j}{\partial h_{j-1}}^T \frac{\partial \ell}{\partial h_L}, h_i \in \mathbb{R}^{D_i \times 1}$$

$$h_i = \sigma(W_i h_{i-1} + b_i) \rightarrow \frac{\partial h_i}{\partial h_{i-1}} = \text{Diag}(\sigma'(W_i h_{i-1} + b_i)) W_i \in \mathbb{R}^{D_i \times D_{i-1}}$$

3.2 [10pts] Derive the gradients of loss ℓ w.r.t. all weights W_i and biases $\mathbf{b}_i$. In particular, for any layer $1 \leq i \leq L$, you need to show how to derive $\frac{\partial \ell}{\partial W_i}$ and $\frac{\partial \ell}{\partial \mathbf{b}_i}$. You need to write down the shapes of all tensors involved in the final expressions.

$$\text{Let } z_i = W_i h_{i-1} + b_i \in \mathbb{R}^{D_i \times 1}$$

$$\frac{\partial h_i}{\partial z_i} = \sigma'(W_i h_{i-1} + b_i)$$

$$\frac{\partial z_i}{\partial W_i} = h_{i-1}$$

$$\frac{\partial z_i}{\partial b_i} = 1$$

$$\frac{\partial h_i}{\partial W_i} = \frac{\partial h_i}{\partial z_i}\frac{\partial z_i}{\partial W_i} = \sigma'(W_i h_{i-1} + b_i) h_{i-1} \in \mathbb{R}^{D_i \times D_i \times D_{i-1}}$$

$$\frac{\partial h_i}{\partial b_i} = \frac{\partial h_i}{\partial z_i}\frac{\partial z_i}{\partial b_i} = \sigma'(W_i h_{i-1} + b_i) \in \mathbb{R}^{D_i \times D_{i-1}}$$

3.3 [15pts] Derive the Kaiming initialization [3] in our context, *i.e.*, applying the same variance analysis technique we learned from Xavier initialization [4] to this deep MLP with ReLU activations.

Show

$$\forall h, \text{VAR}(z_i)^{(h)} = \text{VAR}(z_j)^{(h-1)}, z \in \mathbb{R}^{d_h \times 1}, 0 \leq i \leq d_{h+1}, 0 \leq j \leq d_h$$

Let

$$W_{i,j} \sim N(0, \frac{2}{d_h}), b^{(h)} = 0, h(z) = ReLU(z)$$

Then given $\mathbb{E}W = 0$ and $b^{(h)} = 0$:

$$\mathbb{E}z_i^{(h)} = 0$$

$$\text{VAR}(z_i^{(h)}) = d_h \text{VAR}(W_{i,j}^{(h)}) \mathbb{E}[(x_j^h)^2]$$

$$\mathbb{E}[(x_j^{(h)})^2] = \int_{-\infty}^{\infty} (x_j^{(h)})^2 P_x(x_j^{(h)}) dx_j^{(h)}$$

$$\mathbb{E}[(x_j^{(h)})^2] = \int_{0}^{\infty} (z_j^{(h-1)})^2 P_z(z_j^{(h-1)}) dz_j^{(h-1)}$$

$$= \frac{1}{2}\text{VAR}(z_j^{(h-1)})$$

$$\therefore \text{VAR}(z_i^{(h)}) = d_h \text{VAR}(W_{i,j}^{(h)}) \mathbb{E}[(x_j^h)^2] = d_h * \frac{2}{d_h} * \frac{1}{2}\text{VAR}(z_j^{(h-1)}) = \text{VAR}(z_j^{(h-1)})$$

References

[1] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), pp.1929-1958.

[2] Ioffe, S. and Szegedy, C., 2015, June. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (pp. 448-456).

[3] He, K., Zhang, X., Ren, S. and Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1026-1034).

[4] Glorot, X. and Bengio, Y., 2010, March. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the International Conference on Artificial Intelligence and Statistics (pp. 249-256).