Deep Learning with S-shaped Rectified Linear Activation Units

Andrea Lombardo
Luca Tomei

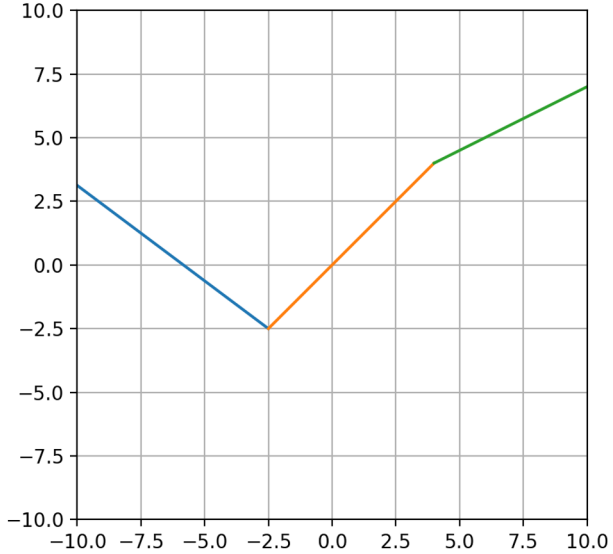
USED ACTIVATION FUNCTIONS

- **RELU**: $\max(0, x_i)$
- **PARAMETRIC RELU**: $\min(0, ax_i) + \max(0, x_i)$
- **LEAKY RELU** : $\min(0, 0.01x_i) + \max(0, x_i)$
- **ELU**: $f(x) = \begin{cases} a(e^x - 1) & \forall x \leq 0 \\ x & \forall x \geq 0 \end{cases}$
- **PARAMETRIC ELU**: $f(x) = \begin{cases} a\left(e^{\frac{x}{b}} - 1\right) & \forall x \leq 0 \\ \frac{a}{b}x & \forall x \geq 0 \end{cases}$
- **SRELU**

INTRODUCTION OF THE SRELU FUNCTION

S-shaped Rectified Linear Unit

$$f(x_i) = \begin{cases} t_i^l + a_i^l(x_i - t_i^l) & \forall\ x_i \leq t_i^l \\ x_i & \forall\ t_i^l < x_i < t_i^r \\ t_i^r + a_i^r(x_i - t_i^r) & \forall\ x_i \geq t_i^r \end{cases}$$



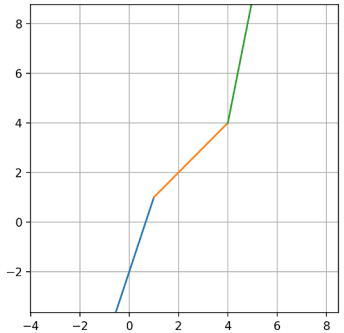
PROPERTIES OF THE SRELU FUNCTION

The SReLU activation function has some remarkable properties:

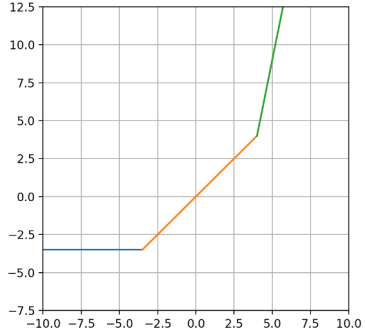
It has four trainable parameters:

$$a_i^r,\ t_i^r,\ a_i^l,\ t_i^l$$

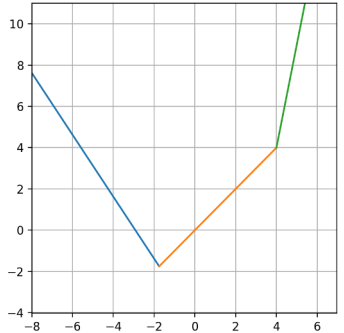
$a_l > 0$



$a_l = 0$



$a_l < 0$

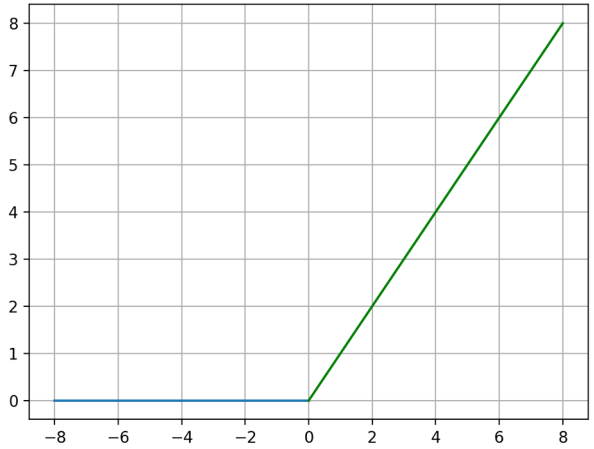


PROPERTIES OF THE SRELU FUNCTION

The SReLU activation function has some remarkable properties:

It generalizes the ReLU function:

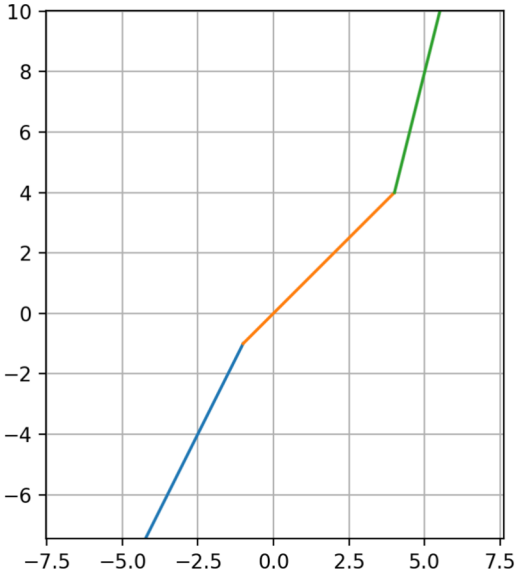
$$\begin{cases} t_l = t_r = 0 \\ a_l = 0 \\ a_r = 1 \end{cases} \Longrightarrow \mathrm{SReLU} = \mathrm{ReLU}$$

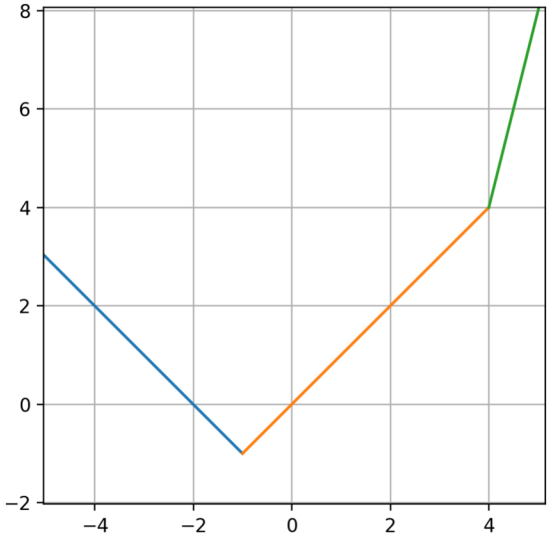


PROPERTIES OF THE SRELU FUNCTION

The SReLU activation function has some remarkable properties:

It can be settled to be either convex or non-convex:

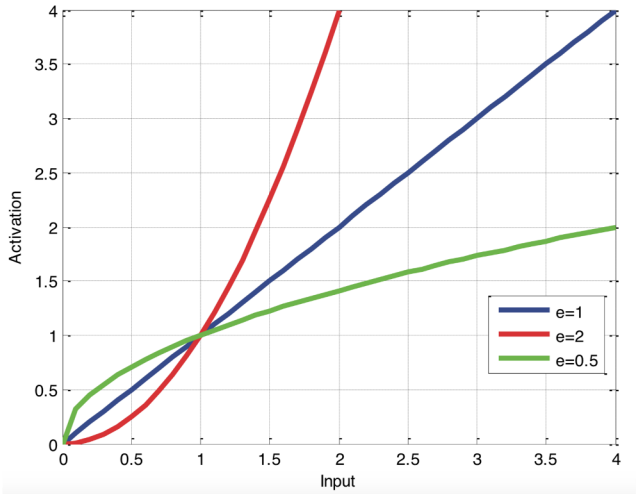


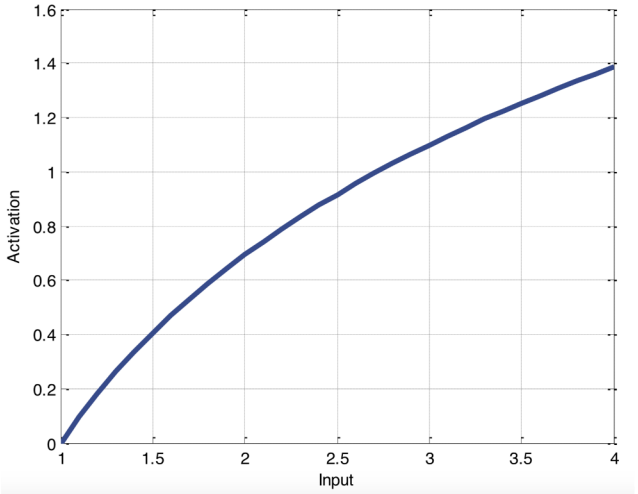


PROPERTIES OF THE SRELU FUNCTION

The SReLU activation function has some remarkable properties:

It can be settled to emulate the human perception mechanisms:





EXPERIMENTAL SETTINGS

IMPLEMENTED ARCHITECTURES ON CIFAR10

CNN 1

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
activation (Activation)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 30, 30, 32)	9248
activation_1 (Activation)	(None, 30, 30, 32)	0
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
dropout (Dropout)	(None, 15, 15, 32)	0
conv2d_2 (Conv2D)	(None, 15, 15, 64)	18496
activation_2 (Activation)	(None, 15, 15, 64)	0
conv2d_3 (Conv2D)	(None, 13, 13, 64)	36928
activation_3 (Activation)	(None, 13, 13, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 6, 6, 64)	0
dropout_1 (Dropout)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 512)	1180160
activation_4 (Activation)	(None, 512)	0
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5130
activation_5 (Activation)	(None, 10)	0

Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0

CNN 2

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
activation (Activation)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9248
activation_1 (Activation)	(None, 32, 32, 32)	0
max_pooling2d (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18496
activation_2 (Activation)	(None, 16, 16, 64)	0
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36928
activation_3 (Activation)	(None, 16, 16, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73856
activation_4 (Activation)	(None, 8, 8, 128)	0
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147584
activation_5 (Activation)	(None, 8, 8, 128)	0
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262272
activation_6 (Activation)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290
activation_7 (Activation)	(None, 10)	0

Total params: 550,570
Trainable params: 550,570
Non-trainable params: 0

IMPLEMENTED ARCHITECTURES ON MNIST

CNN 1

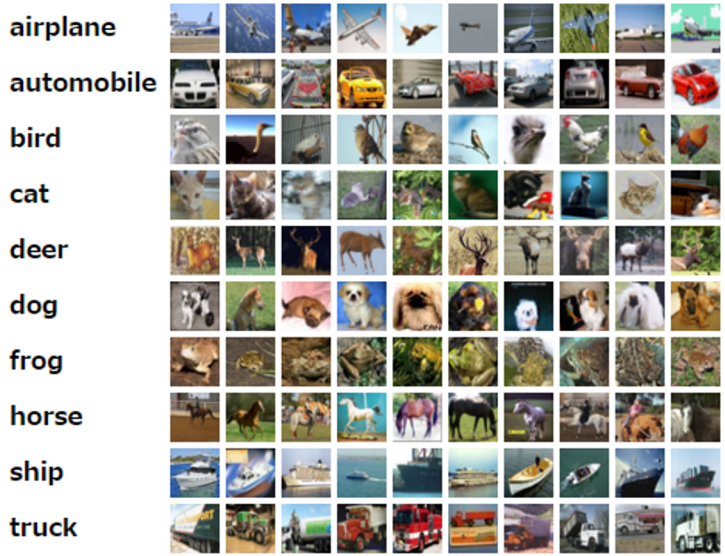
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 26, 26, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 24, 24, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               1179776
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_4 (Activation)    (None, 10)                0
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

CNN 2

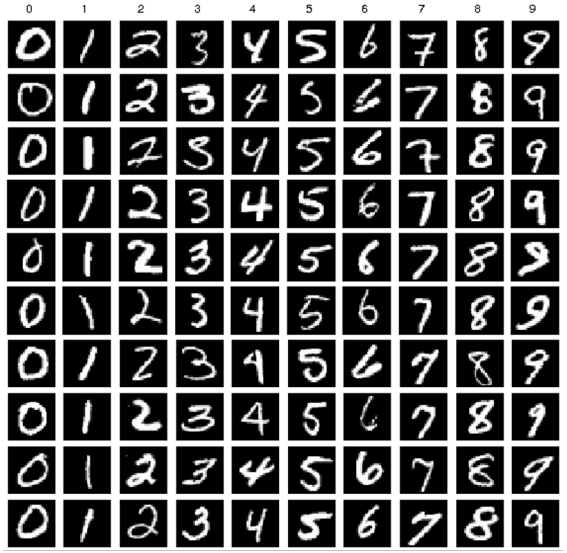
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
activation (Activation)      (None, 26, 26, 32)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 5408)              0
_________________________________________________________________
dense (Dense)                (None, 100)               540900
_________________________________________________________________
activation_1 (Activation)    (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 542,230
Trainable params: 542,230
Non-trainable params: 0
_________________________________________________________________
```

DATASETS

CIFAR10



MNIST



IMPLEMENTATION DETAILS

NEURAL NETWORKS IMPLEMENTATION

CNN1 on CIFAR10

```
model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]))

#model.add(Activation('relu'))
model.add(LeakyReLU())

model.add(Conv2D(32, (3, 3)))

#model.add(Activation('relu'))
model.add(LeakyReLU())

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding='same'))

#model.add(Activation('relu'))
model.add(LeakyReLU())

model.add(Conv2D(64, (3, 3)))

#model.add(Activation('relu'))
model.add(LeakyReLU())

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512))

#model.add(Activation('relu'))
model.add(LeakyReLU())

model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

CNN1 on MNIST

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), input_shape=input_shape))

#model.add(Activation('relu'))
model.add(LeakyReLU())

model.add(Conv2D(64, (3, 3)))

#model.add(Activation('relu'))
model.add(LeakyReLU())

model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128))

#model.add(Activation('relu'))
model.add(LeakyReLU())

model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

NEURAL NETWORKS IMPLEMENTATION

CNN2 on CIFAR10

```
model = Sequential()

model.add(Conv2D(32, (nb_conv, nb_conv), padding='same', input_shape=x_train.shape[1:]))
model.add(Activation('relu'))

model.add(Conv2D(32, (nb_conv, nb_conv), padding='same'))
model.add(Activation('relu'))

model.add(MaxPooling2D((nb_pool, nb_pool)))

model.add(Conv2D(64, (nb_conv, nb_conv), padding='same'))
model.add(Activation('relu'))


model.add(Conv2D(64, (nb_conv, nb_conv), padding='same'))
model.add(Activation('relu'))

model.add(MaxPooling2D((nb_pool, nb_pool)))

model.add(Conv2D(128, (nb_conv, nb_conv), padding='same'))
model.add(Activation('relu'))

model.add(Conv2D(128, (nb_conv, nb_conv), padding='same'))
model.add(Activation('relu'))

model.add(MaxPooling2D((nb_pool, nb_pool)))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))

model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

CNN2 on MNIST

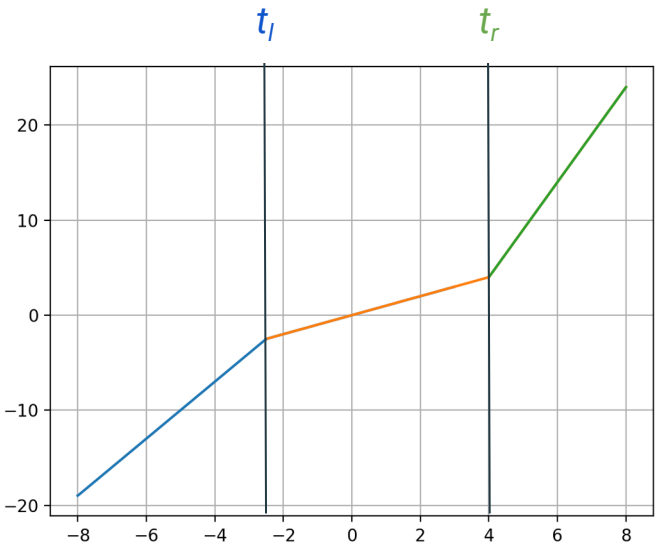
```
model = Sequential()
model.add(Conv2D(nb_filters, (nb_conv, nb_conv), input_shape=input_shape)
model.add(Activation('relu'))
#model.add(LeakyReLU())

model.add(MaxPooling2D((nb_pool, nb_pool)))
model.add(Flatten())

model.add(Dense(100))
model.add(Activation('relu'))
#model.add(LeakyReLU())

model.add(Dense(num_classes, activation='softmax'))
```

MySReLU IMPLEMENTATION

```python
from keras import backend as K
from keras.layers import Layer

class MySReLU(Layer):

    def __init__(self, **kwargs):
        super(MySReLU, self).__init__(**kwargs)

    def build(self, input_shape):
        param_shape = tuple(list(input_shape[1:])) # input_shape is: (batch, height, width, channels)

        self.tl = self.add_weight(shape=param_shape, name='tl', initializer='zeros', trainable=True)
        self.al = self.add_weight(shape=param_shape, name='al', initializer='uniform', trainable=True)
        self.delta = self.add_weight(shape=param_shape, name='delta', initializer='uniform', trainable=True)
        self.ar = self.add_weight(shape=param_shape, name='ar', initializer='ones', trainable=True)

        super(MySReLU, self).build(input_shape)

    def call(self, x):
        tr = self.tl + K.abs(self.delta)
        tl = self.tl
        al = self.al
        ar = self.ar

        eps=0.000001
        if_x_gtr_tr = K.relu(x-tr)/(x-tr+eps);
        if_x_lss_tl = K.relu(tl-x)/(tl-x+eps);
        if_x_btw_tlr = (1-if_x_gtr_tr)*(1-if_x_lss_tl);

        return if_x_gtr_tr*(ar*(x-tr) + tr) + if_x_lss_tl*(al*(x-tl) + tl) + if_x_btw_tlr*x

    def compute_output_shape(self, input_shape):
        return input_shape
```

MySReLU IMPLEMENTATION

```
if_x_lss_tl*(al*(x-tl) + tl)  +  if_x_btw_tlr*x  +  if_x_gtr_tr*(ar*(x-tr) + tr)
```



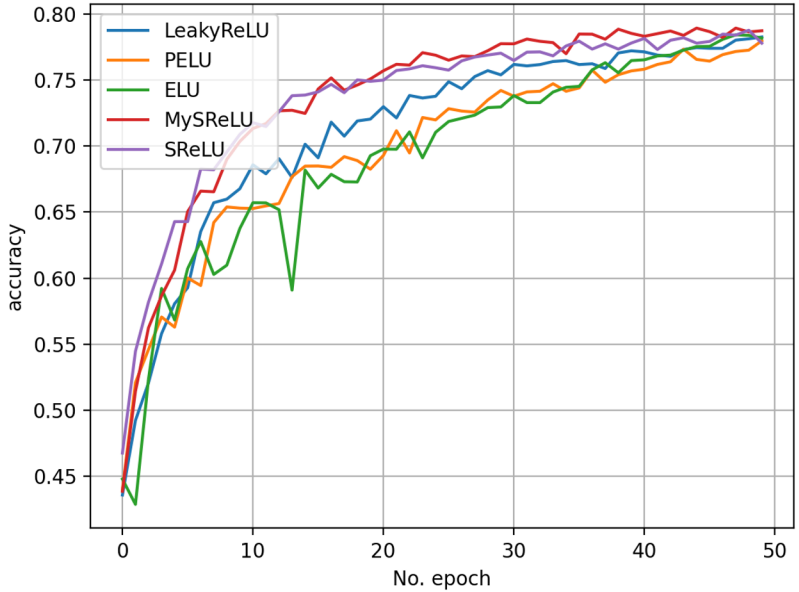

```
if_x_gtr_tr = K.relu(x-tr)/(x-tr+eps)
```

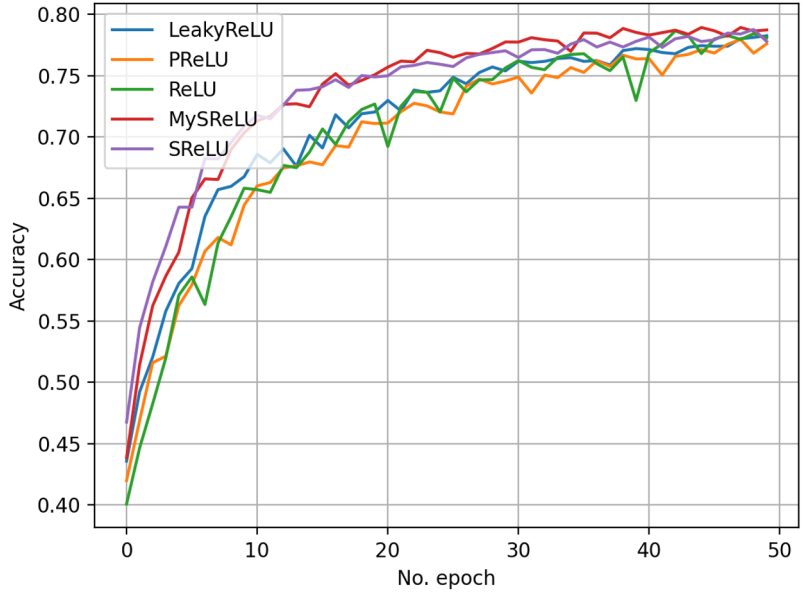
$$\max\left(0, \frac{(x-t_r)}{(x-t_r)+\varepsilon}\right) \simeq \text{Heaviside}(x-t_r)$$

RESULTS

RESULTS: CNN1 ON CIFAR10

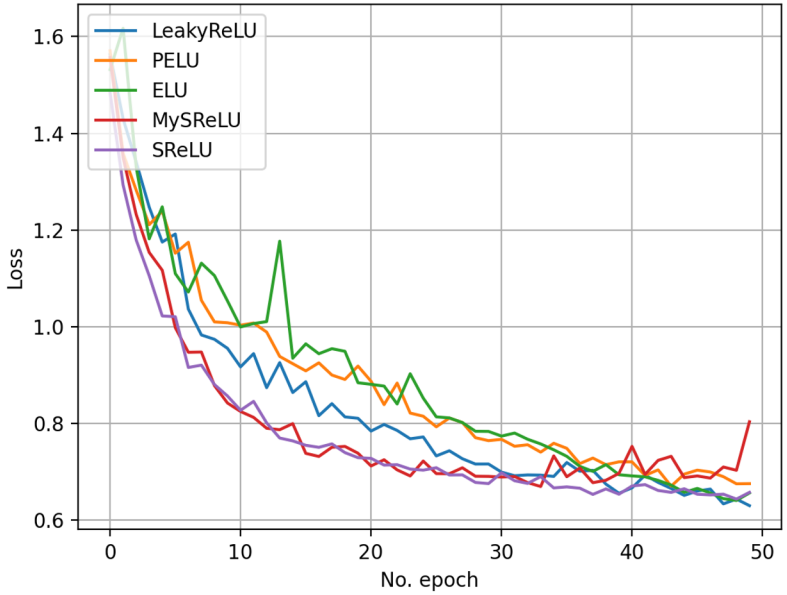
ACCURACY: rectified vs exponential activation functions

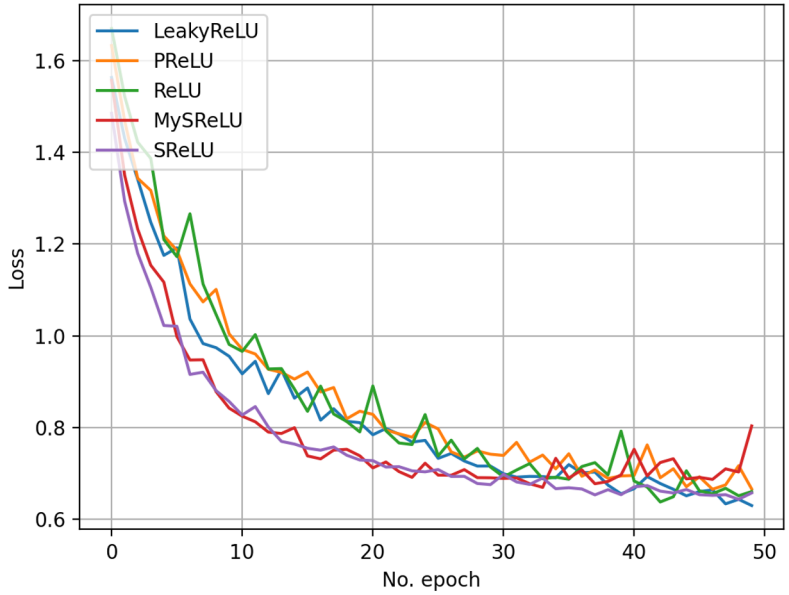




RESULTS: CNN1 ON CIFAR10

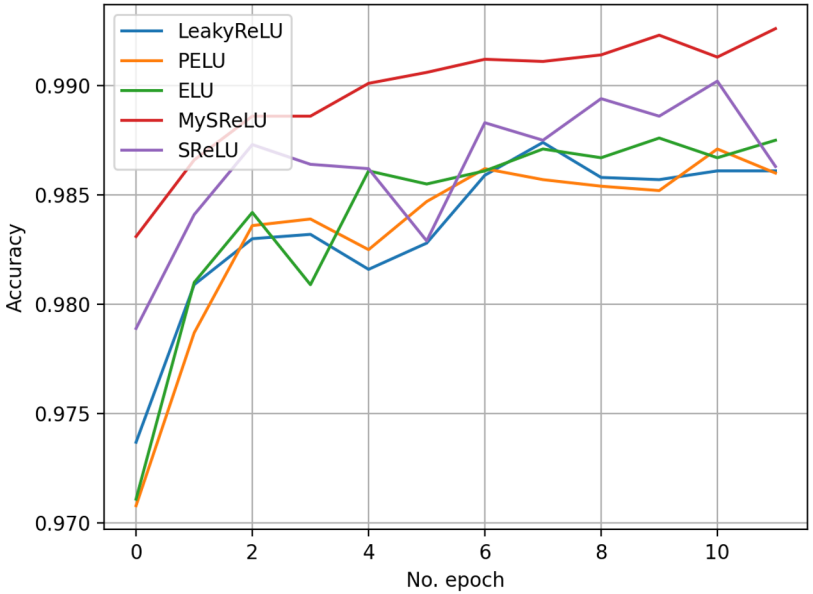
LOSS: rectified vs exponential activation functions

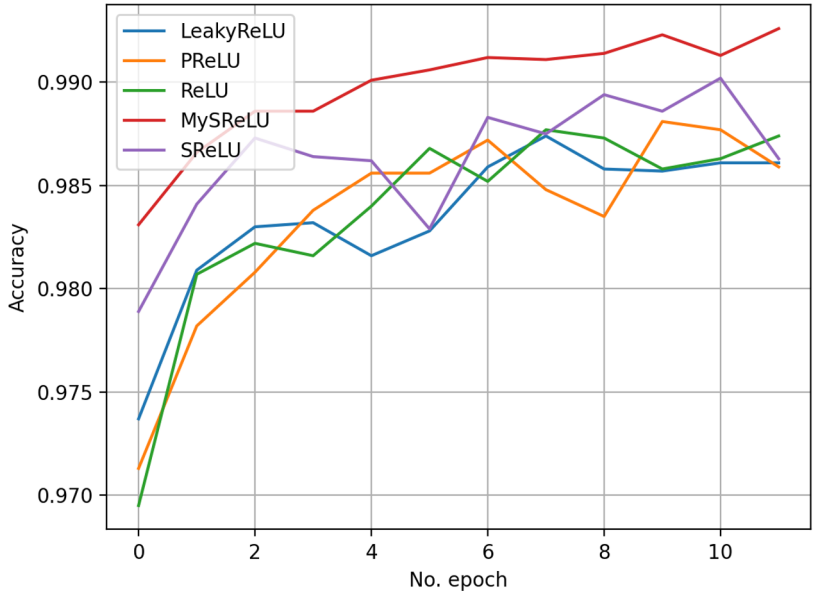




RESULTS: CNN2 ON MNIST

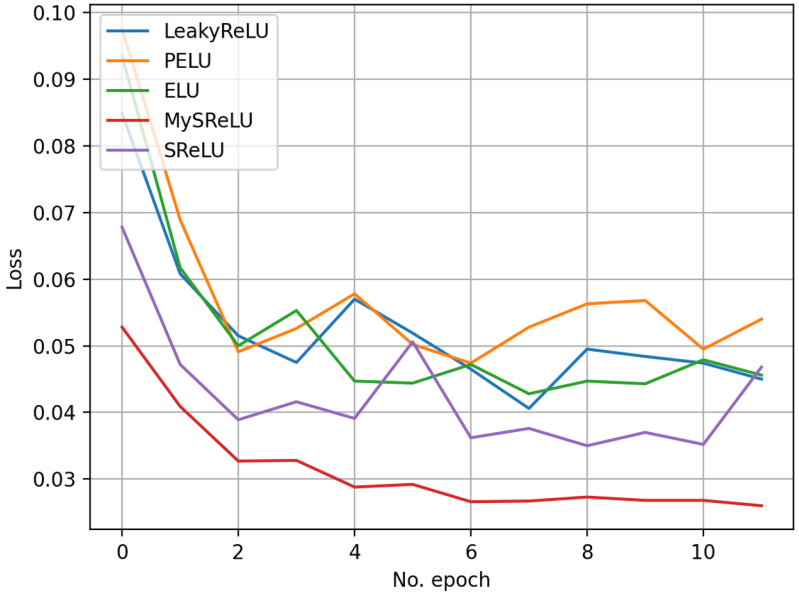
ACCURACY: rectified vs exponential activation functions

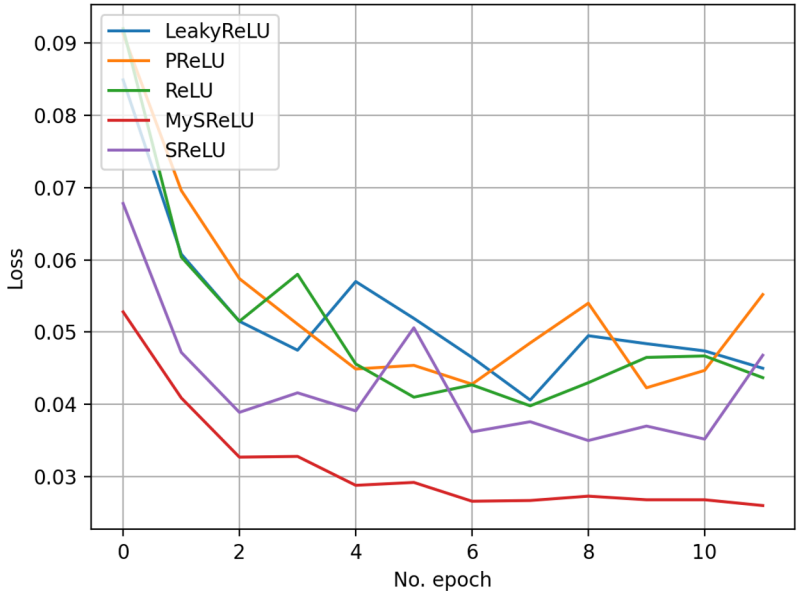




RESULTS: CNN2 ON MNIST

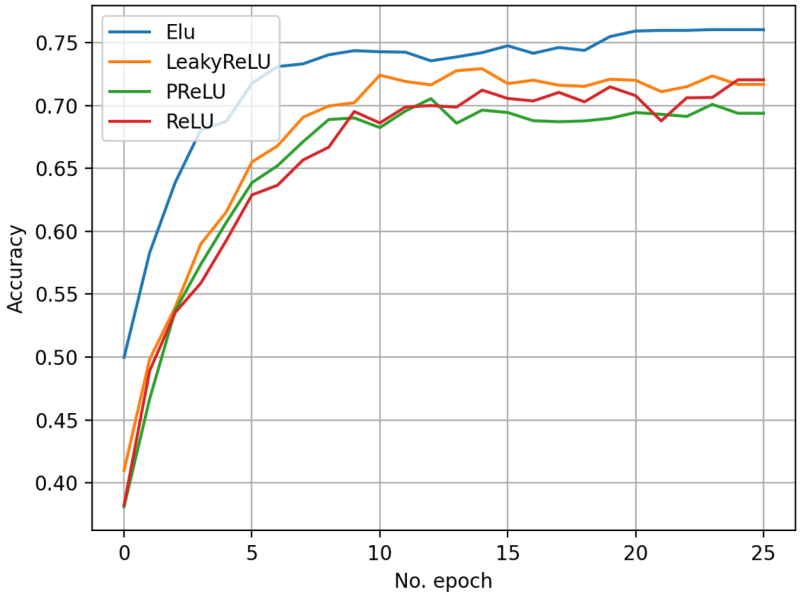
LOSS: rectified vs exponential activation functions

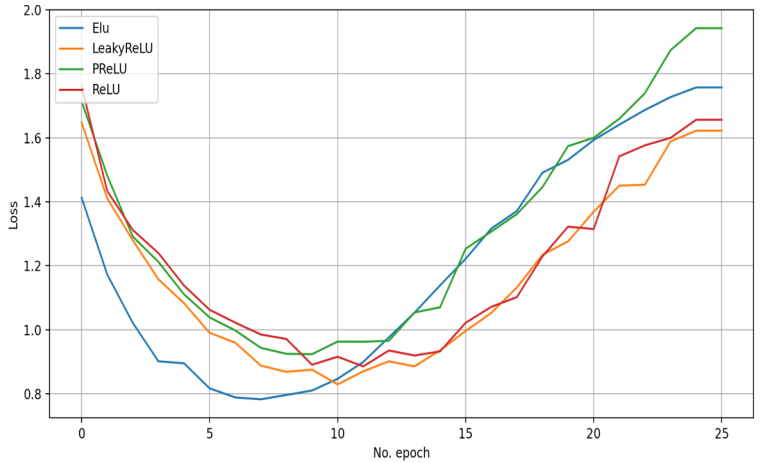




RESULTS: CNN2 ON CIFAR10

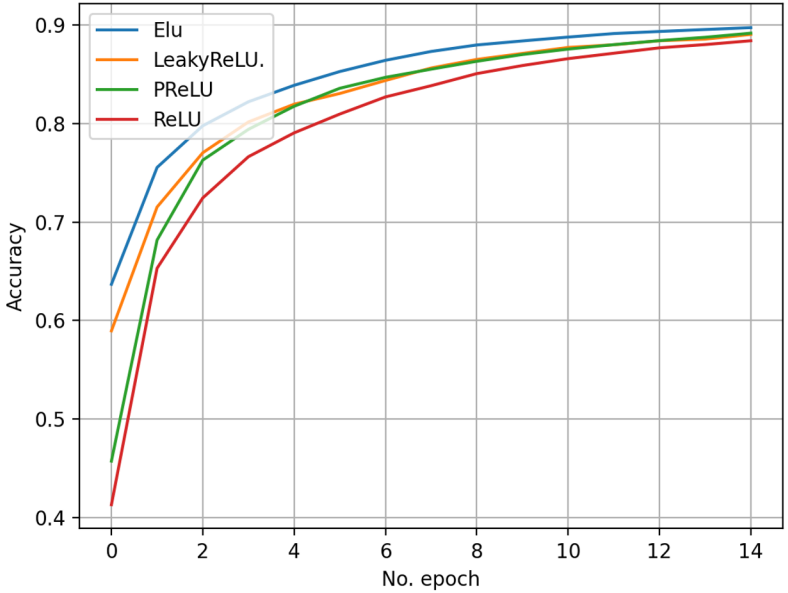
ACCURACY & LOSS

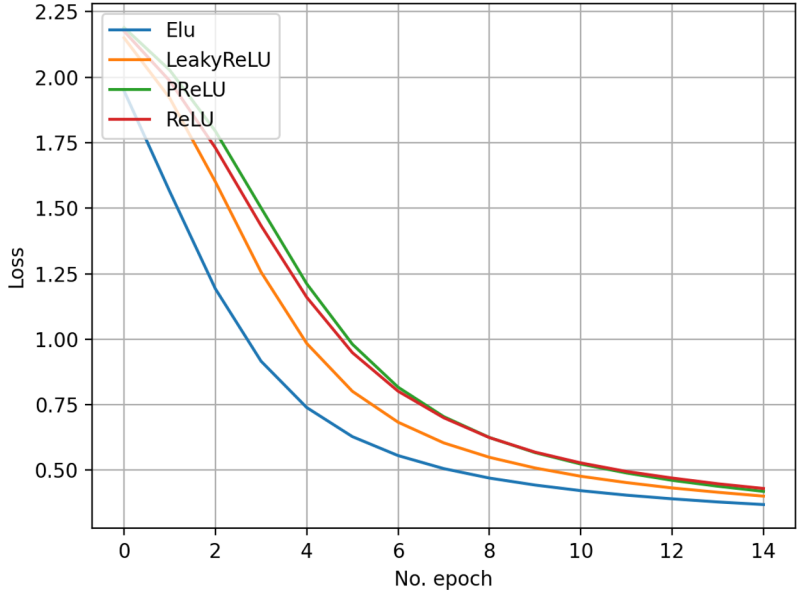




RESULTS: CNN1 ON MNIST

ACCURACY & LOSS





Grazie dell’attenzione