CS5489 - Machine Learning

Lecture 9b - Regularization

Prof. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

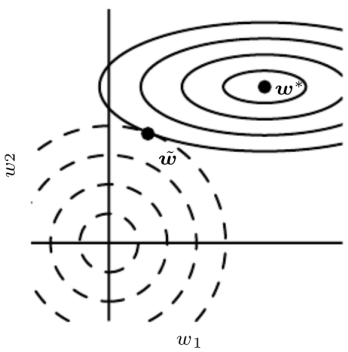
- Convolutional neural network (CNN)
- **Regularization**

Regularization with L2-norm

- One way to regularize the network is to add an L2-norm penalty on the weights
  - Add a penalty term to the loss function
    - larger weights have higher penalty
    - $\hat{L} = L + \lambda \frac{1}{2} ||\mathbf{w}||^2$

- Taking the gradient of $\hat{L}$,
  - $\frac{d\hat{L}}{d\mathbf{w}} = \frac{dL}{d\mathbf{w}} + \lambda \mathbf{w}$
- Applying gradient descent
  - $\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{d\hat{L}}{d\mathbf{w}}$
  - $\mathbf{w} \leftarrow \mathbf{w} - \eta (\frac{dL}{d\mathbf{w}} + \lambda \mathbf{w})$
  - $\mathbf{w} \leftarrow (1 - \eta\lambda)\mathbf{w} - \eta \frac{dL}{d\mathbf{w}}$
- the weights shrink by $(1 - \eta\lambda)$ in each iteration of gradient descent.
  - prevents the $\mathbf{w}$ from getting too large.
  - Sometimes this is called **weight-decay** regularization.

- The optimal solution is an equilibrium point between L2-norm (dashed-line) and the loss (solid-line).
  - $\lambda \mathbf{w} = -\frac{dL}{d\mathbf{w}}$.



Example: L2 regularization

- Apply L2 regularization on all layers.
  - `kernel_regularizer` keyword

In [9]:
```
K.clear_session()  # cleanup
random.seed(4487); tf.random.set_seed(4487) # initialize random seed

# build the network
nn = Sequential()
nn.add(Conv2D(10, (5,5), strides=(2,2), activation='relu',
              input_shape=(28,28,1),
              kernel_regularizer=keras.regularizers.l2(0.0001),  # L2 regularizer
              padding='same'))
nn.add(Conv2D(40, (5,5), strides=(2,2), activation='relu',
              kernel_regularizer=keras.regularizers.l2(0.0001),  # L2 regularizer
              padding='same'))
nn.add(Conv2D(80, (5,5), strides=(1,1), activation='relu',
              kernel_regularizer=keras.regularizers.l2(0.0001),  # L2 regularizer
              padding='same'))
nn.add(Flatten())
nn.add(Dense(units=50, activation='relu',
             kernel_regularizer=keras.regularizers.l2(0.0001))) # L2 regularizer
nn.add(Dense(units=10, activation='softmax',
       kernel_regularizer=keras.regularizers.l2(0.0001)))        # L2 regularizer

# setup early stopping callback function
earlystop = keras.callbacks.EarlyStopping(
    monitor='val_loss',     # look at the validation loss
    min_delta=0.0001,       # threshold to consider as no change
    patience=5,             # stop if 5 epochs with no change
    verbose=1, mode='auto'
)
callbacks_list = [earlystop]

# compile and fit the network
nn.compile(loss=keras.losses.categorical_crossentropy,
           optimizer=keras.optimizers.SGD(learning_rate=0.02, momentum=0.9, nesterov=True),
          metrics=['accuracy'])
history = nn.fit(vtrainI, vtrainYb, epochs=100, batch_size=50,
                 callbacks=callbacks_list,
                 validation_data=validsetI, verbose=False)
```

```
2023-01-22 23:11:20.828749: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:305] Could not identify NUMA node of platform GPU ID 0, de
faulting to 0. Your kernel may not have been built with NUMA support.
2023-01-22 23:11:20.828902: I tensorflow/core/common_runtime/pluggable_device/plug
gable_device_factory.cc:271] Created TensorFlow device (/job:localhost/replica:0/t
ask:0/device:GPU:0 with 0 MB memory) -> physical PluggableDevice (device: 0, name:
METAL, pci bus id: <undefined>)
2023-01-22 23:11:20.933188: W tensorflow/core/platform/profile_utils/cpu_utils.cc:
128] Failed to get CPU frequency: 0 Hz
```

```
Metal device set to: Apple M1 Max
```

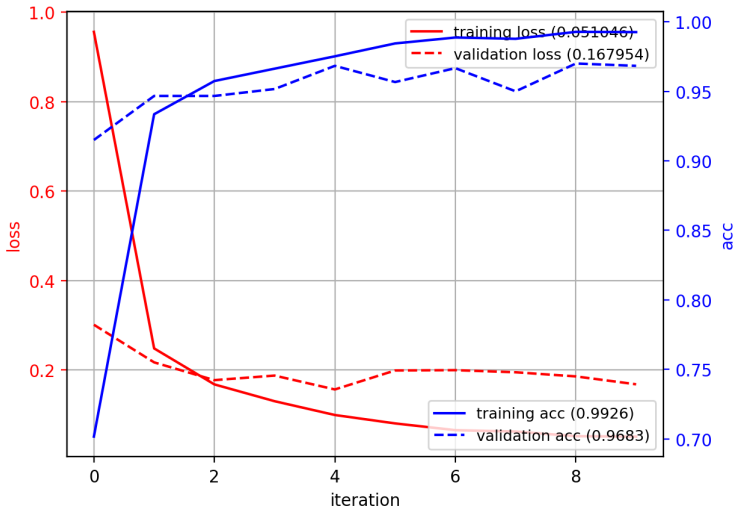
```
2023-01-22 23:11:21.071509: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-22 23:11:22.033856: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

```
Epoch 10: early stopping
```

In [10]:
```
plot_history(history)

predY = argmax(nn.predict(testI, verbose=False), axis=-1)
acc = metrics.accuracy_score(testY, predY)
print("test accuracy:", acc)
```

```
2023-01-22 23:11:29.347366: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

```
test accuracy: 0.966
```



Which layers should we apply L2 regularization?

- Consider the following network: $f(\mathbf{x}) = \mathbf{A}^T r(\mathbf{B}^T \mathbf{x})$
  - $r(\cdot)$ is the ReLU activation.
- Consider the scaling value $\epsilon > 0$:

$$f(\mathbf{x}) = \mathbf{A}^T r(\mathbf{B}^T \mathbf{x}) = \mathbf{A}^T r(\frac{\epsilon}{\epsilon} \mathbf{B}^T \mathbf{x}) = \frac{1}{\epsilon} \mathbf{A}^T r(\epsilon \mathbf{B}^T \mathbf{x})$$

  - the last line is because for $b > 0$, $r(ba) = \max(0, ba) = b \max(0, a)$

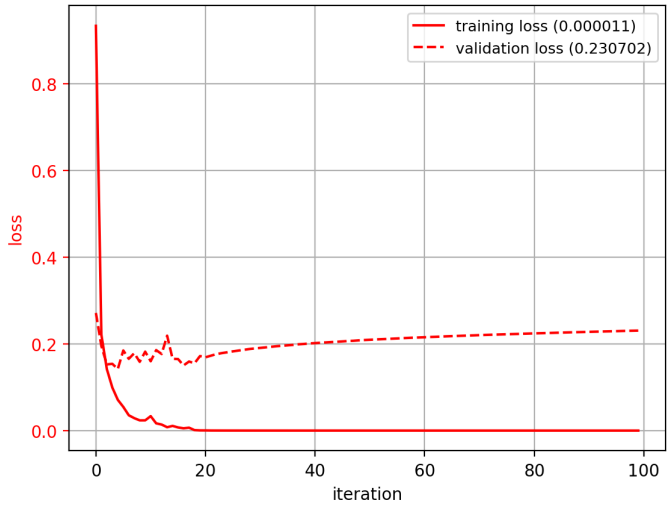
- If we decrease $\mathbf{B}$ by $\epsilon < 1$, we can increase $\mathbf{A}$ correspondingly to create the same network (with the same loss L).
  - $\mathbf{B} \to \epsilon \mathbf{B}$
  - $\mathbf{A} \to \frac{1}{\epsilon} \mathbf{B}$
- Thus, when applying L2 regularization on only one layer (e.g., $\mathbf{B}$), the other layer (e.g., $\mathbf{A}$) can be adjusted to get the same network.
  - The L2-norm is artificially reduced without changing the network function.
  - i.e., it has no actual effect.

- **Answer**: apply L2 regularization to *all* layers!

Overfitting

- occurs when validation curve starts to increase
  - network is specializing only on the training data, losing generalization ability.

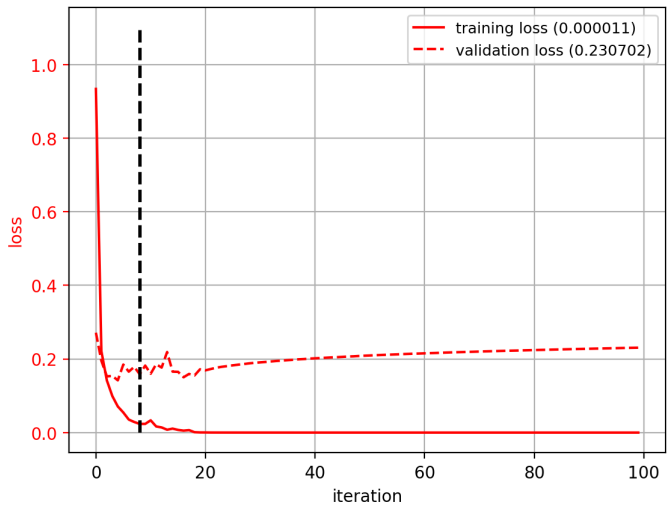
In [12]:
```
plot_history(history, showacc=False)
```



Early Stopping

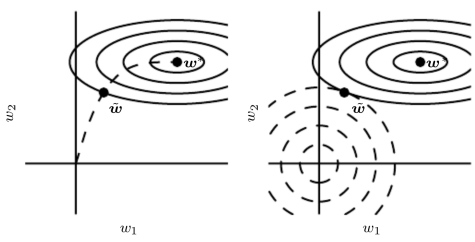
- Stopping training when the validation loss increases
  - equivalently, selecting the model/iteration with lowest validation loss.

In [13]:
```
plot_history(history, showacc=False)
plt.plot([8,8], [0,1.1], 'k--', lw=2);
```



Early Stopping as Regularization

- Early stopping is a form of regularization
  - limit the number of training iterations = limiting how far parameters $\mathbf{w}$ can move.
    - e.g., if we initialize $\mathbf{w} = 0$, then early stopping limits the L2-norm of $\mathbf{w}$.
  - the amount $\mathbf{w}$ can move is adaptive to the data via the validation loss.



Improving Generalization with Ensembles

- **Bagging**: Bootstrap Aggregation
  - combine predictions from several models (model averaging).
- Suppose we have K models, each model has error ϵ_i
  - MSE of each model: $\text{MSE}_i = \mathbb{E}[\epsilon_i^2] = V$
- Combining the models:
  - MSE of the combined model: $\text{MSE}_c = \frac{V}{K} + \frac{K-1}{K}C$
    - $C = \mathbb{E}[\epsilon_i \epsilon_j]$ is the correlation between errors of the i-th and j-th models.
- If the errors are uncorrelated ($C = 0$), then the MSE decreases!
  - $\text{MSE}_c = \frac{V}{K}$

Bagging for NN

- Typically bagging needs to create a new dataset by sampling with replacement of the original dataset, and then learning a new model on each generated dataset.
- For NN, there are enough solution points
  - we can use the same dataset, and different initializations (or change hyperparameters).
  - this should give models with partially uncorrelated errors.
- This is a **very** popular way to boost performance.

Example: Ensembles

- train 5 CNNs with same architectures but different initializations

In [15]:
```
def ensemble_prediction(nns, testI):
    # compute probabilities from all nns
    allprob = [mynn.predict(testI, verbose=False) for mynn in nns]
    bagprob = sum(allprob, axis=0) / len(nns)

    # compute predictions (assume class labels are 0,1,2,...)
    allpred = [argmax(myprob, axis=1) for myprob in allprob]
    bagpred = argmax(bagprob, axis=1)

    return allpred, bagpred
```

In [16]:
```
allpred, bagpred = ensemble_prediction(allnn, testI)
```

```
2023-01-22 23:13:48.384017: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-22 23:13:49.031485: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-22 23:13:49.700232: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-22 23:13:50.374741: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-22 23:13:51.027375: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

- various errors for the individual NNs

In [17]:
```
# compute error for individual models
for i,predY in enumerate(allpred):
    acc = metrics.accuracy_score(testY, predY)
    print("model {} test accuracy: {}".format(i, acc))
```

```
model 0 test accuracy: 0.966
model 1 test accuracy: 0.9693
model 2 test accuracy: 0.9621
model 3 test accuracy: 0.9634
model 4 test accuracy: 0.9629
```

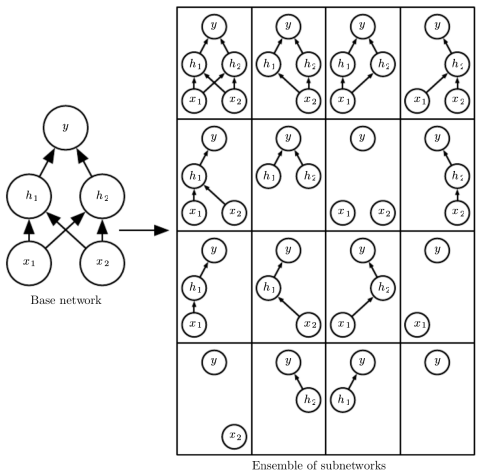
- ensembling the predictions improves accuracy

In [18]:
```
# compute error for ensemble
acc = metrics.accuracy_score(testY, bagpred)
print("combined model test accuracy: {}".format(acc))
```

```
combined model test accuracy: 0.9723
```

Dropout

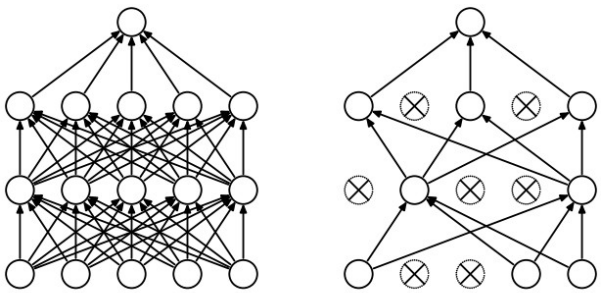
- **Problem:** training a large number of NNs for an ensemble is time consuming.
- **Goal:** approximate a really large ensemble of models
  - create an ensemble by randomly removing an input/hidden node with some probability.



- number of models is exponential in the number of nodes.
- share the same parameters for each sub-model.
  - we can train all models in the ensemble at the same time.

Dropout Training

- During training, randomly "drop out" each node with probability p
  - a dropped-out node is not used for calculating the prediction or weight updating.
  - trains a reduced network in each iteration.


(a) Standard Neural Net (b) After applying dropout.

- During test time, use all the nodes for prediction and scale output by $1 - p$.
  - this keeps the expected value of the node to be the same as during training.
  - analogous to computing predictions from all models, and then averages the predictions.

- Dropout is implemented as a layer.
- Example:

- apply dropout layer after last feature layer and 1st dense layer.

In [19]:
```
K.clear_session()
random.seed(4487); tf.random.set_seed(4487)  # initialize random seed

# build the network
nn = Sequential()
nn.add(Conv2D(10, (5,5), strides=(2,2), activation='relu',
              input_shape=(28,28,1),
              padding='same'))
nn.add(Conv2D(40, (5,5), strides=(2,2), activation='relu',
              padding='same'))
nn.add(Conv2D(80, (5,5), strides=(1,1), activation='relu',
              padding='same'))
nn.add(Dropout(rate=0.5, seed=44))   # dropout layer! (need to specify the seed)
nn.add(Flatten())
nn.add(Dense(units=50, activation='relu'))
nn.add(Dropout(rate=0.5, seed=45))   # dropout layer!
nn.add(Dense(units=10, activation='softmax'))

# compile and fit the network
nn.compile(loss=keras.losses.categorical_crossentropy,
           optimizer=keras.optimizers.SGD(learning_rate=0.02, momentum=0.9, nesterov=True),
          metrics=['accuracy'])
history = nn.fit(vtrainI, vtrainYb, epochs=100, batch_size=50,
                 callbacks=callbacks_list,
                 validation_data=validsetI, verbose=False)
```

```
2023-01-22 23:14:03.010821: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-22 23:14:03.795372: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

```
Epoch 15: early stopping
```

In [20]:
```
nn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 14, 14, 10)        260

 conv2d_1 (Conv2D)           (None, 7, 7, 40)          10040

 conv2d_2 (Conv2D)           (None, 7, 7, 80)          80080

 dropout (Dropout)           (None, 7, 7, 80)          0

 flatten (Flatten)           (None, 3920)              0

 dense (Dense)               (None, 50)                196050

 dropout_1 (Dropout)         (None, 50)                0

 dense_1 (Dense)             (None, 10)                510

=================================================================
Total params: 286,940
Trainable params: 286,940
Non-trainable params: 0
_________________________________________________________________
```

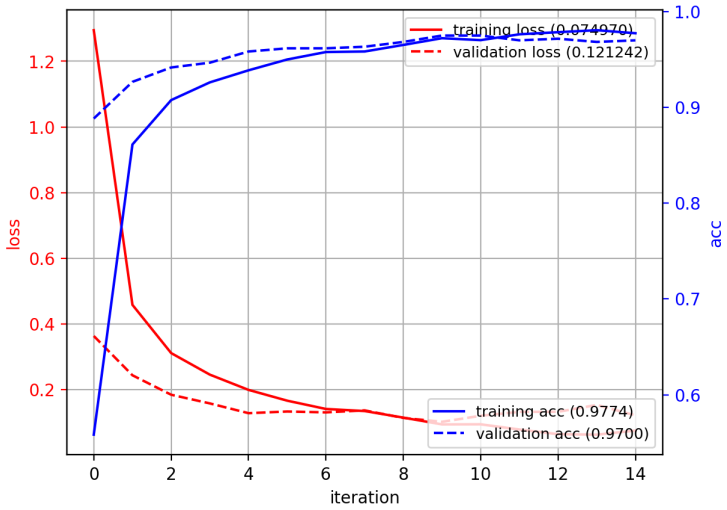
- accuracy improves

In [21]:
```
plot_history(history)
predY = argmax(nn.predict(testI, verbose=False), axis=-1)
acc = metrics.accuracy_score(testY, predY)
print("test accuracy:", acc)
```

```
2023-01-22 23:14:14.243689: I tensorflow/core/grappler/optimizers/custom_graph_opt
```

```
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

```
test accuracy: 0.9713
```



Effects of Dropout

- dropping a node means the node's output is 0
  - i.e., the gradient is 0, which blocks some paths when computing gradients.
  - helps with vanishing gradient problem
- model averaging reduces the error.
- Dropout *reduces* the effective capacity of the network.
  - may need to increase the model size.
- At any time a node h will be removed, i.e., that feature is missing.
  - then the other features must be sufficient to solve the classification problem
  - classifier doesn't depend exclusively on one feature.
  - increases classifier robustness.

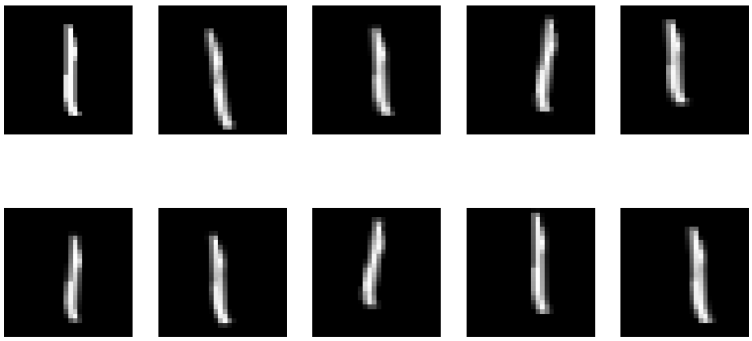
Data augmentation

- Problem: not enough data
- Solution: artificially permute the data to increase the dataset size
  - goal: make the network invariant to the permutations
  - examples: translate image, flip image, add pixel noise, rotate image, deform image, etc.

In [22]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# build the data augmenter
datagen = ImageDataGenerator(
    rotation_range=10,         # image rotation
    width_shift_range=0.1,     # image shifting
    height_shift_range=0.1,    # image shifting
    shear_range=0.1,           # shear transformation
    zoom_range=0.1,            # zooming
    data_format='channels_last')

# fit (required for some normalization augmentations)
datagen.fit(vtrainI)
```

- Example of original (top-left) and augmented data

In [24]:
```
plt.figure(figsize=(8,4))
show_imgs(imgs, nc=5)
```



- Any transformation is okay, as long as the transformation doesn't confuse the classes.
  - e.g. don't rotate "6" so that it looks like a "9"
- Data augmentation improves robustness or invariance to selected transformations
  - encoded in the parameters of the model.

In [25]:
```
# initialize random seed
K.clear_session()
random.seed(4487); tf.random.set_seed(4487)

# build the network
nn = Sequential()
nn.add(Conv2D(10, (5,5), strides=(2,2), activation='relu',
              input_shape=(28,28,1), padding='same'))
nn.add(Conv2D(40, (5,5), strides=(2,2), activation='relu', padding='same'))
nn.add(Conv2D(80, (5,5), strides=(1,1), activation='relu', padding='same'))
nn.add(Dropout(rate=0.5, seed=44))
nn.add(Flatten())
nn.add(Dense(units=50, activation='relu'))
nn.add(Dropout(rate=0.5, seed=45))
nn.add(Dense(units=10, activation='softmax'))

# compile the network
nn.compile(loss=keras.losses.categorical_crossentropy,
           optimizer=keras.optimizers.SGD(learning_rate=0.02, momentum=0.9, nesterov=True),
          metrics=['accuracy'])
```

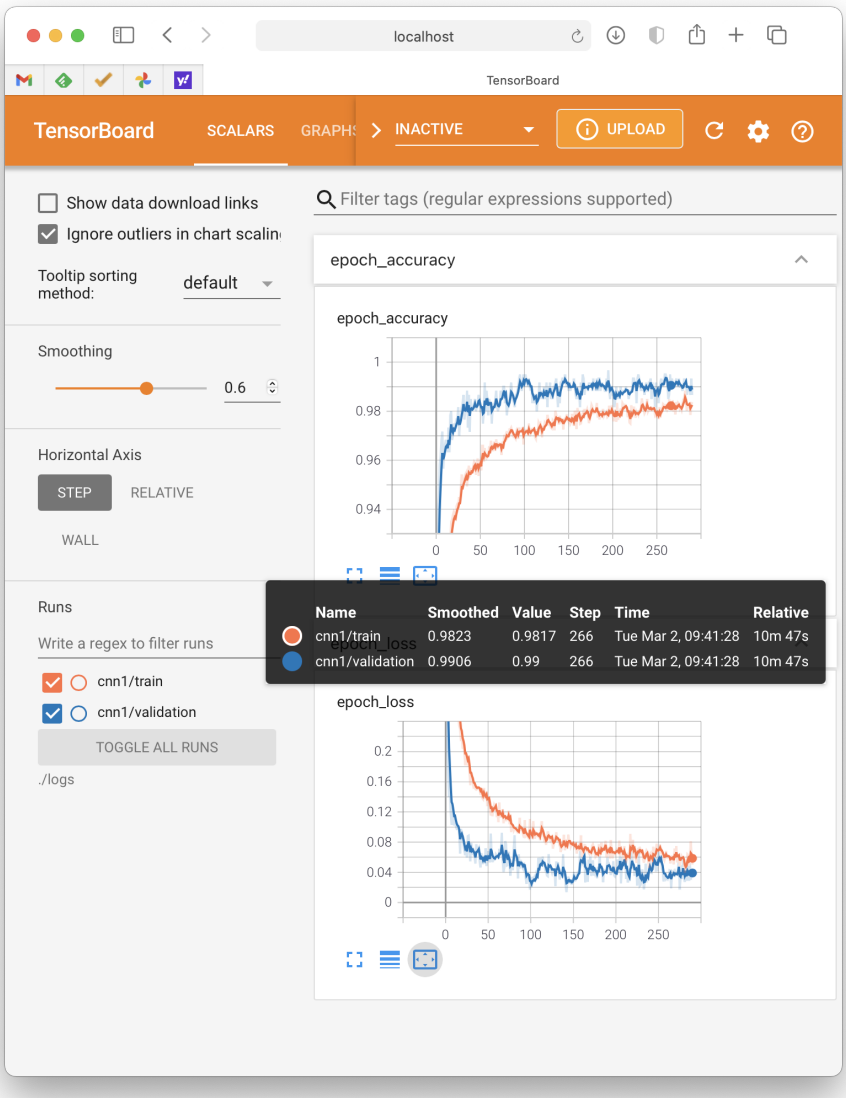
- Train with `fit`
  - passes data through the generator first (`datagen.flow(...)`)
  - runs generator and fit in parallel
- the dataset changes each epoch, so validation error will change a lot.
  - we disable early stopping and just let it run for 100 epochs.
- use TensorBoard to view the current results in real-time.
  - Create a `TensorBoard` object and use it as the callback.

In [26]:
```
from tensorflow.keras.callbacks import TensorBoard

tensorboard = TensorBoard(log_dir='./logs/cnn1', histogram_freq=0,
                          write_graph=False, write_images=False)

# pass data through augmentor and fit
# runs data-generator and fit in parallel
history = nn.fit(
            datagen.flow(vtrainI, vtrainYb, batch_size=50),  # data from generator
            steps_per_epoch=len(vtrainI)/50,    # should be number of batches per epoch
            epochs=300,
            callbacks=[tensorboard],
            validation_data=validsetI, verbose=False)
```

```
2023-01-22 23:14:40.819597: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-22 23:14:41.936169: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

Tensorboard

- Tensorboard runs as a local webserver to view the results in the log directory.
  - to start Tensorboard, run the following command in the shell: `tensorboard --logdir ./logs`
  - then connect to the local webserver using a browser (the URL is printed out when Tensorboard starts).

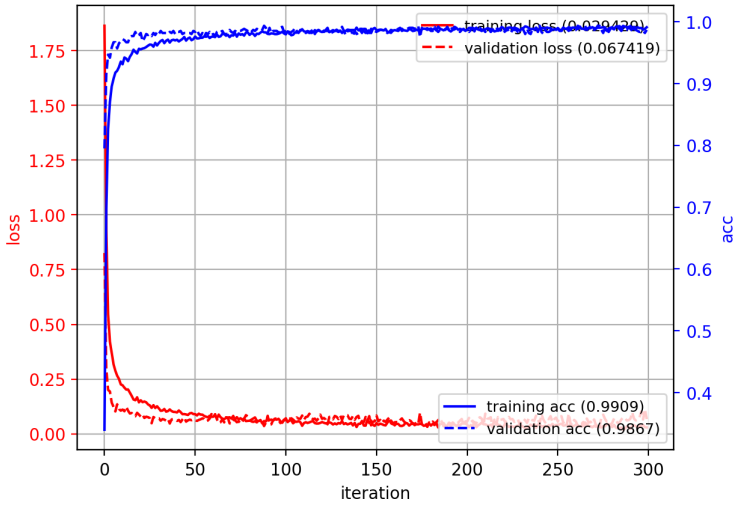


- data augmentation increases accuracy.

In [27]:
```
plot_history(history)

predY = argmax(nn.predict(testI, verbose=False), axis=-1)
acc = metrics.accuracy_score(testY, predY)
print("test accuracy:", acc)
```

```
2023-01-22 23:19:40.551225: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

```
test accuracy: 0.9893
```



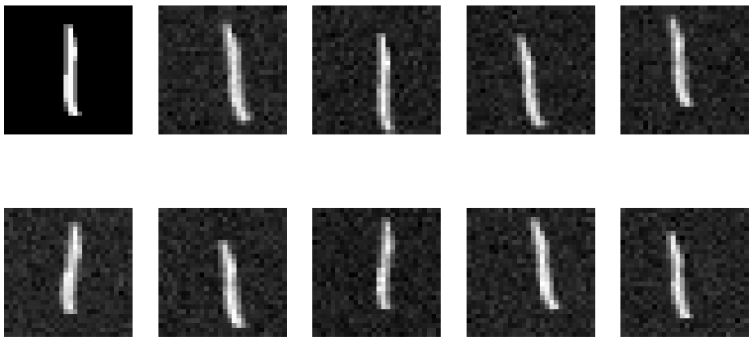
Data augmentation with noise

- Also add per-pixel noise to the image for data augmentation.
  - define a function to add noise
  - set it as the `preprocessing_function`

In [28]:
```
def add_gauss_noise(X, sigma2=0.05):
    # add Gaussian noise with zero mean, and variance sigma2
    return X + random.normal(0, sigma2, X.shape)

# build the data augmenter
datagen = ImageDataGenerator(
    rotation_range=10,          # image rotation
    width_shift_range=0.1,      # image shifting
    height_shift_range=0.1,     # image shifting
    shear_range=0.1,            # shear transformation
    zoom_range=0.1,             # zooming
    preprocessing_function=add_gauss_noise,
    data_format='channels_last'
)

# fit (required for some normalization augmentations)
datagen.fit(vtrainI)
```

- Example: original image (top-left) and augmented data

In [30]:
```
plt.figure(figsize=(8,4))
show_imgs(imgs, nc=5)
```



- Train with augmented data: transformations and per-pixel noise

In [31]:
```
K.clear_session()
random.seed(4487); tf.random.set_seed(4487) # initialize random seed

# build the network
nn = Sequential()
nn.add(Conv2D(10, (5,5), strides=(2,2), activation='relu',
              input_shape=(28,28,1), padding='same'))
nn.add(Conv2D(40, (5,5), strides=(2,2), activation='relu', padding='same'))
nn.add(Conv2D(80, (5,5), strides=(1,1), activation='relu', padding='same'))
nn.add(Dropout(rate=0.5, seed=44))
nn.add(Flatten())
nn.add(Dense(units=50, activation='relu'))
nn.add(Dropout(rate=0.5, seed=45))
nn.add(Dense(units=10, activation='softmax'))

# compile the network
nn.compile(loss=keras.losses.categorical_crossentropy,
           optimizer=keras.optimizers.SGD(learning_rate=0.02, momentum=0.9, nesterov=True),
          metrics=['accuracy'])

tensorboard = TensorBoard(log_dir='./logs/cnn2', histogram_freq=0,
                          write_graph=False, write_images=False)

# pass data through augmentor and fit
# runs data-generator and fit in parallel
history = nn.fit(
            datagen.flow(vtrainI, vtrainYb, batch_size=50),  # data from generator
            steps_per_epoch=len(vtrainI)/50,    # should be number of batches per epoch
            epochs=300,
            callbacks=[tensorboard],
            validation_data=validsetI, verbose=False)
```

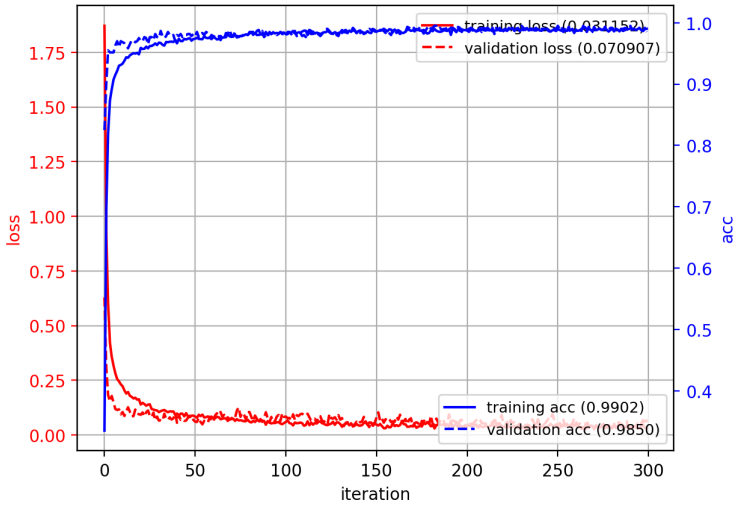
```
2023-01-22 23:19:41.560058: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
2023-01-22 23:19:42.578966: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

- Similar results.

In [32]:
```
plot_history(history)

predY = argmax(nn.predict(testI, verbose=False), axis=-1)
acc = metrics.accuracy_score(testY, predY)
print("test accuracy:", acc)
```

```
2023-01-22 23:24:33.698356: I tensorflow/core/grappler/optimizers/custom_graph_opt
imizer_registry.cc:113] Plugin optimizer for device_type GPU is enabled.
```

```
test accuracy: 0.9895
```



Data generator options

- Sometimes the dataset is too large to store completely in RAM:
  - `datagen.flow_from_directory()` - iterate over images in a directory
  - `datagen.flow_from_dataframe()` - same, but use a pandas dataframe as the metadata.

Comparison on MNIST

- For image data, CNN works better and has less parameters.

Type	No.Layers	Architecture	No.Parameters	Test Accuracy
LR	1	output(10)	7,850	0.8911
MLP	2	ReLu(50), output(10)	39,760	0.9380
MLP	2	Relu(200), output(10)	159,010	0.9437
MLP	2	Relu(1000), output(10)	795,010	0.9462
MLP	3	ReLu(500), Relu(500), output(10)	648,010	0.9529
CNN	3	Conv(10x5x5), ReLu(50), output(10)	98,820	0.9534
CNN	5	Conv(10x5x5), Conv(50x3x3), Conv(80x3x3), ReLu(50), output(10)	286,940	0.9564
CNN (w/ weight decay)	5	Conv(10x5x5), Conv(50x3x3), Conv(80x3x3), ReLu(50), output(10)	286,940	0.9660
CNN (ensemble)	5	Conv(10x5x5), Conv(50x3x3), Conv(80x3x3), ReLu(50), output(10)	5x286,940	0.9723
CNN (w/ dropout)	5	Conv(10x5x5), Conv(50x3x3), Conv(80x3x3), ReLu(50), output(10)	286,940	0.9713
CNN (w/ dropout, data-augmentation)	5	Conv(10x5x5), Conv(50x3x3), Conv(80x3x3), ReLu(50), output(10)	286,940	0.9893
CNN (w/ dropout, data-augmentation, noise)	5	Conv(10x5x5), Conv(50x3x3), Conv(80x3x3), ReLu(50), output(10)	286,940	0.9895

Summary

- **Convolutional neural network (CNN)**
  - convolution filters for extracting local image features
    - global translation equivariance
  - local translation invariance using max-pooling
  - classifier using MLP on the image features.
- **Regularization methods**
  - L2-norm regularization (weight-decay) - keep weights from becoming too large
  - Early-stopping - adaptively keep weights from becoming too large
  - Ensembles - reduce error by bagging
  - Dropout - approximate an ensemble of networks
  - Data Augmentation - make the NN robust to various image transformations