



MNIST:

An introductory example using NNX



Build and train a simple CNN with NNX

Download the MNIST dataset

```
#!/wget -nc https://huggingface.co/datasets/ylecun/mnist/resolve/main/mnist/train-00000-of-00001.parquet
```

```
#!/wget -nc https://huggingface.co/datasets/ylecun/mnist/resolve/main/mnist/test-00000-of-00001.parquet
```

```
import pandas as pd
```

```
train_file_path = '/content/train-00000-of-00001.parquet'
```

```
test_file_path = '/content/test-00000-of-00001.parquet'
```

```
mnist_train_df = pd.read_parquet(train_file_path)
```

```
mnist_test_df = pd.read_parquet(test_file_path)
```

Get the datasets ready

```
class Dataset:
    def __init__(self, df):
        self.df = df

    def __len__(self):
        return len(self.df)

    def __getitem__(self, index):
        return convert_to_numpy(self.df.iloc[index])

def convert_to_numpy(data_dict):
    png_bytes = data_dict['image']['bytes']
    image = Image.open(io.BytesIO(png_bytes))
    image_array = np.array(image, dtype=np.float32) / 255.0
    label_array = np.array(data_dict['label'])
    return {'image': image_array[:, :, np.newaxis], 'label': label_array}
```

Create the Grain dataloaders

```
mnist_train = Dataset(mnist_train_df)
```

```
mnist_test = Dataset(mnist_test_df)
```

```
sampler = grain.SequentialSampler(  
    num_records=len(mnist_train),  
    shard_options=grain.NoSharding())
```

```
train_dl = grain.DataLoader(  
    data_source=mnist_train,  
    sampler=sampler,  
    operations=[grain.Batch(batch_size=batch_size, drop_remainder=True)]  
)
```

```
test_dl = grain.DataLoader(  
    data_source=mnist_test,  
    sampler=sampler,  
    operations=[grain.Batch(batch_size=batch_size, drop_remainder=True)]  
)
```

Define a CNN

```
class CNN(nnx.Module):
    """A simple CNN model."""

    def __init__(self, *, rngs: nnx.Rngs):
        self.conv1 = nnx.Conv(1, 32, kernel_size=(3, 3), rngs=rngs)
        self.conv2 = nnx.Conv(32, 64, kernel_size=(3, 3), rngs=rngs)
        self.avg_pool = partial(nnx.avg_pool, window_shape=(2, 2), strides=(2, 2))
        self.linear1 = nnx.Linear(3136, 256, rngs=rngs)
        self.linear2 = nnx.Linear(256, 10, rngs=rngs)

    def __call__(self, x):
        x = self.avg_pool(nnx.relu(self.conv1(x)))
        x = self.avg_pool(nnx.relu(self.conv2(x)))
        x = x.reshape(x.shape[0], -1) # flatten
        x = nnx.relu(self.linear1(x))
        x = self.linear2(x)
        return x
```

Display the CNN

Instantiate the model.

```
model = CNN(rngs=nnx.Rngs(0))
```

Visualize it.

```
nnx.display(model)
```

```
▼ CNN( # Param: 824,458 (3.3 MB)
conv1= Conv(kernel_shape=(3, 3, 1, 32), kernel=Param(value=<jax.Array float32(3, 3, 1, 32) ≈0.00089 ±0.33 [≥-0.75, ≤0.73] nonzero:288>), bias=Param(value=<jax.Array float32(32,) ≈0.00038 ±0.059 [≥-0.13, ≤0.13] nonzero:18_432>), bias=Param(value=<jax.Array float32(32,) ≈0.00038 ±0.059 [≥-0.13, ≤0.13] nonzero:18_432>),
conv2= Conv(kernel_shape=(3, 3, 32, 64), kernel=Param(value=<jax.Array float32(3, 3, 32, 64) ≈0.00038 ±0.059 [≥-0.13, ≤0.13] nonzero:18_432>), bias=Param(value=<jax.Array float32(64,) ≈0.00038 ±0.059 [≥-0.13, ≤0.13] nonzero:18_432>), bias=Param(value=<jax.Array float32(64,) ≈0.00038 ±0.059 [≥-0.13, ≤0.13] nonzero:18_432>),
avg_pool=functools.partial(<function avg_pool at 0x2e3528e50>, window_shape=(2, 2), strides=(2, 2)), # <functools.partial object at 0x2fca16430>
linear1= Linear( # Param: 803,072 (3.2 MB)
  kernel=Param(value=<jax.Array float32(3136, 256)>),
  bias=Param(value=<jax.Array float32(256,) ≈0.0 ±0.0 [≥0.0, ≤0.0] zero:256>),
  in_features=3136,
  out_features=256,
  use_bias=True,
  dtype=None,
  param_dtype=jax.numpy.float32,
  precision=None,
  kernel_init=<function variance_scaling.<locals>.init at 0x2e420f490>, # Defined at line 316 of /Users/cris/repos/cristian/flax/.venv/lib/python3.10/site-packages/jax/_src/numpy/lax_numpy.py:316:0
  bias_init=zeros,
  dot_general=jax.lax.dot_general,
),
linear2= Linear(kernel=Param(value=<jax.Array float32(256, 10) ≈0.0016 ±0.063 [≥-0.14, ≤0.14] nonzero:2_560>), bias=Param(value=<jax.Array float32(10,) ≈0.0 ±0.0 [≥0.0, ≤0.0] zero:10>),
```

Test run the CNN

```
import jax.numpy as jnp # JAX NumPy
```

```
y = model(jnp.ones((1, 28, 28, 1)))
```

```
# Array([[ -0.06820839, -0.14743432,  0.00265857, -0.2173656 ,  0.16673787,  
#          -0.00923921, -0.06636689,  0.28341877,  0.33754364, -0.20142877]],  
#        dtype=float32)
```

Define optimizer and metrics

```
import optax
```

```
learning_rate = 0.005
```

```
momentum = 0.9
```

```
optimizer = nnx.Optimizer(model, optax.adamw(learning_rate, momentum), wrt=nnx.Param)
```

```
metrics = nnx.MultiMetric(
```

```
    accuracy=nnx.metrics.Accuracy(),
```

```
    loss=nnx.metrics.Average('loss'),
```

```
)
```


Define optimizer and metrics

```
nnx.display(optimizer)
```

```
▼Optimizer( # Param: 824,458 (3.3 MB), OptArray: 1 (4 B), OptVariable: 1,648,916 (6.6 MB), OptState: 1 (4 B), Total: 2,473,376 (9.9 MB)
  step=OptState(value=<jax.Array(0, dtype=uint32)>),
  model=▼CNN( # Param: 824,458 (3.3 MB)
    conv1=Conv(kernel_shape=(3, 3, 1, 32), kernel=Param(value=<jax.Array float32(3, 3, 1, 32) ≈0.00089 ±0.33 [≥-0.75, ≤0.73] nonzero:288>), bias=Param(value=<jax.Array float32(32,) ≈0.0 ±0.0 [≥0.0, ≤0.0] zero:32>)),
    conv2=Conv(kernel_shape=(3, 3, 32, 64), kernel=Param(value=<jax.Array float32(3, 3, 32, 64) ≈0.00038 ±0.059 [≥-0.13, ≤0.13] nonzero:18_432>), bias=Param(value=<jax.Array float32(64,) ≈0.0 ±0.0 [≥0.0, ≤0.0] zero:64>)),
    avg_pool=functools.partial(<function avg_pool at 0x2e3528e50>, window_shape=(2, 2), strides=(2, 2)),
    linear1=Linear(kernel=Param(value=<jax.Array float32(3136, 256)>), bias=Param(value=<jax.Array float32(256,) ≈0.0 ±0.0 [≥0.0, ≤0.0] zero:256>), in_features=3136, out_features=256),
    linear2=Linear(kernel=Param(value=<jax.Array float32(256, 10) ≈0.0016 ±0.063 [≥-0.14, ≤0.14] nonzero:2_560>), bias=Param(value=<jax.Array float32(10,) ≈0.0 ±0.0 [≥0.0, ≤0.0] zero:10>)),
  ),
  tx=▼GradientTransformationExtraArgs(
    init=<function chain.<locals>.init_fn at 0x303660040>,
    update=<function chain.<locals>.update_fn at 0x303660280>,
  ),
  opt_state=▼(
    ▶ScaleByAdamState(count=OptArray(value=<jax.Array(0, dtype=int32)>), mu=State({'conv1': {'bias': OptVariable(value=<jax.Array float32(32,) ≈0.0 ±0.0 [≥0.0, ≤0.0] zero:32>, source_t=0)}, state=EmptyState(), EmptyState()), EmptyState()),
  ),
  wrt=Param,
)
```

Define training step functions

```
def loss_fn(model: CNN, batch):  
    logits = model(batch['image'])  
    loss = optax.softmax_cross_entropy_with_integer_labels(  
        logits=logits, labels=batch['label']  
    ).mean()  
    return loss, logits
```

@nnx.jit

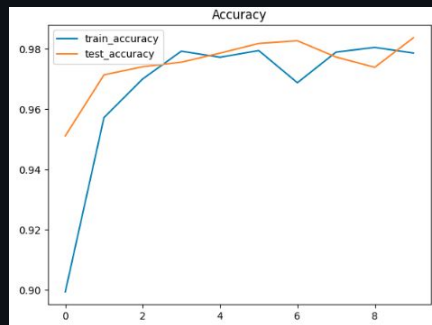
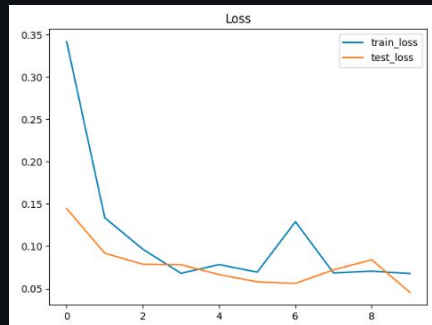
```
def train_step(model: CNN, optimizer: nnx.Optimizer, metrics: nnx.MultiMetric, batch):  
    """Train for a single step."""  
    grad_fn = nnx.value_and_grad(loss_fn, has_aux=True)  
    (loss, logits), grads = grad_fn(model, batch)  
    metrics.update(loss=loss, logits=logits, labels=batch['label']) # In-place updates.  
    optimizer.update(model, grads) # In-place updates.
```

@nnx.jit

```
def eval_step(model: CNN, metrics: nnx.MultiMetric, batch):  
    loss, logits = loss_fn(model, batch)  
    metrics.update(loss=loss, logits=logits, labels=batch['label']) # In-place updates.
```

Create the training loop

```
for epoch in range(num_epochs):  
    for batch in train_dl:  
        train_step(model, optimizer, metrics, batch)  
  
    if step > 0 and (step % eval_every == 0 or step == train_steps - 1):  
        for metric, value in metrics.compute().items():  
            metrics_history[f'train_{metric}'].append(value)  
        metrics.reset() # Reset the metrics for the test set.  
  
    for test_batch in test_dl:  
        eval_step(model, metrics, test_batch)
```



Run a few test images

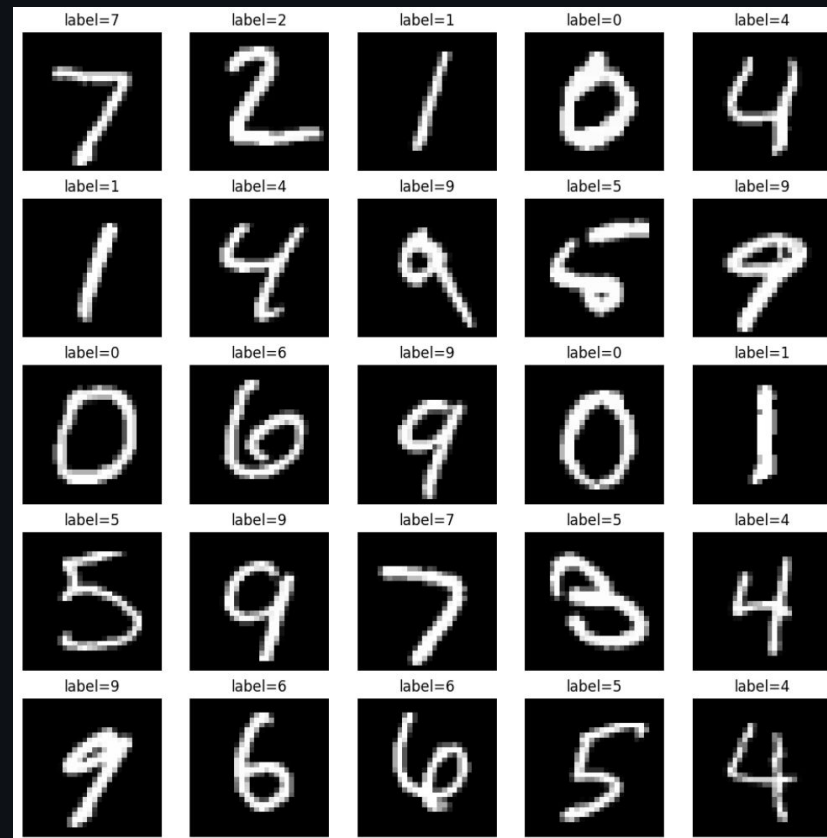
```
model.eval() # Switch to evaluation mode.
```

```
@nnx.jit
```

```
def pred_step(model: CNN, batch):  
    logits = model(batch['image'])  
    return logits.argmax(axis=1)
```

```
test_batch = next(iter(test_dl))  
pred = pred_step(model, test_batch)
```

```
fig, axs = plt.subplots(5, 5, figsize=(12, 12))  
for i, ax in enumerate(axs.flatten()):  
    ax.imshow(test_batch['image'][i, ..., 0], cmap='gray')  
    ax.set_title(f'label={pred[i]}')  
    ax.axis('off')
```



Learning Resources

Code Exercises, Quick References, and Slides

- <https://goo.gle/learning-jax>



Community and Docs

Community:

- <https://goo.gle/jax-community>

Docs

- JAX AI Stack: <https://jaxstack.ai>
- JAX: <https://jax.dev>
- Flax NNX: <https://flax.readthedocs.io>