ENSAE

Deep learning

Unpacking Transformers, LLMs and image generation

Common mistakes

Avoid large losses

Do

```
loss = -probs[0:][ys].log().mean()
```

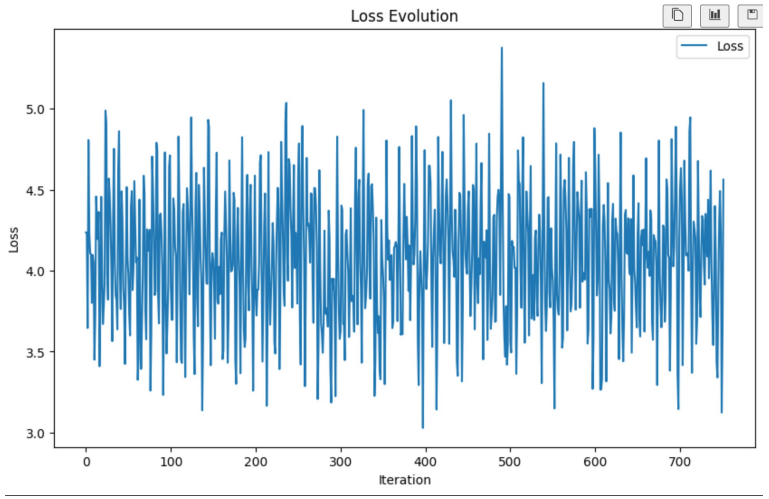
Don’t

```
loss = -probs[0:][ys].log().sum()
```

Plot your loss and don’t pick hyper-params at random

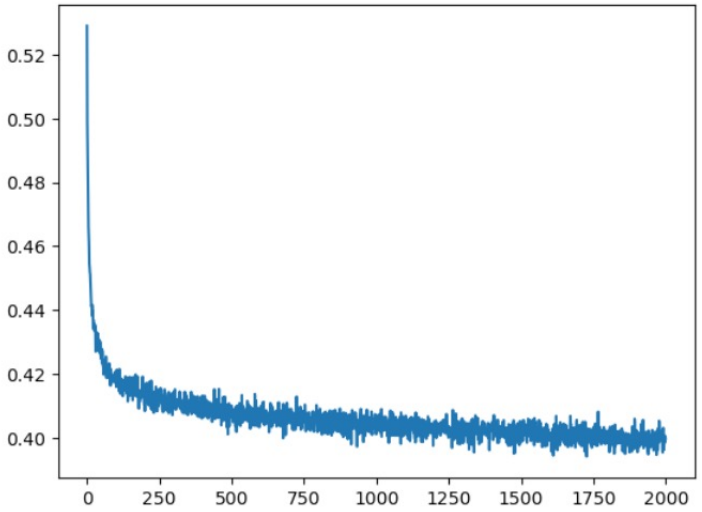
Do

```
lr = 50.0 # cross fingers
```



Don’t

```
lr = 1.0 # after tuning
```



Don’t forget the keepdim = True

Do

```
X = torch.tensor([[1, 2], [3, 4]])
U = X.sum(axis=1, keepdim=True)
X / U
```

Don’t

```
X = torch.tensor([[1, 2], [3, 4]])
U = X.sum(axis=1)
X / U
# this code only works because X is square!
```

Don’t forget the non-linearity

Do

```
h = torch.tanh(emb.view(-1, 30) @ W1 + b1)
logits = h @ W2 + b2
```

Don’t

```
logits = (h @ W1 + b1 ) @ W2 + b2
```

torch.no_grad() is for inference code only

Do

```
# inside training loop (to compute validation loss)
with torch.no_grad():
    logits = ...
```

Don’t – this should not ne wrapped in a torch.no_grad()

```
# inside training loop
with torch.no_grad():
    W -= 0.01 * W.grad
```

torch.no_grad() is for inference code only

Do

```
# inside training loop (to compute validation loss)
with torch.no_grad():
    emb_val =...
```

Don’t -- this should be wrapped in a torch.no_grad()!

```
# forward pass for validation data
emb_val = C[X_dev[val_ix].flatten()].view(-1, emb_size *
block_size)
h_val = torch.tanh(torch.matmul(emb_val, W1) + b1)
```

Reset of gradients should be before gradient descent step

Do

```
loss = ...
W.grad.zero_()
loss.backward()
```

Don’t

```
loss = ...
loss.backward()
W.grad.zero_()
```

A few tips

1. Your code has bugs (always)
2. Debug at very small scale
3. Print tensors shapes
4. Plot internals (loss, gradients, etc.)
5. Use unit test sets