INTRODUCTION TO MACHINE LEARNING COMPSCI 4ML3

Lecture 22

Hassan Ashtiani

UNIVERSAL APPROXIMATION THEOREM

Any potential drawback for neural nets?

- More flexible models require more training data
  - "No free lunch"
- Computational complexity

How can we address these drawbacks? Other models?

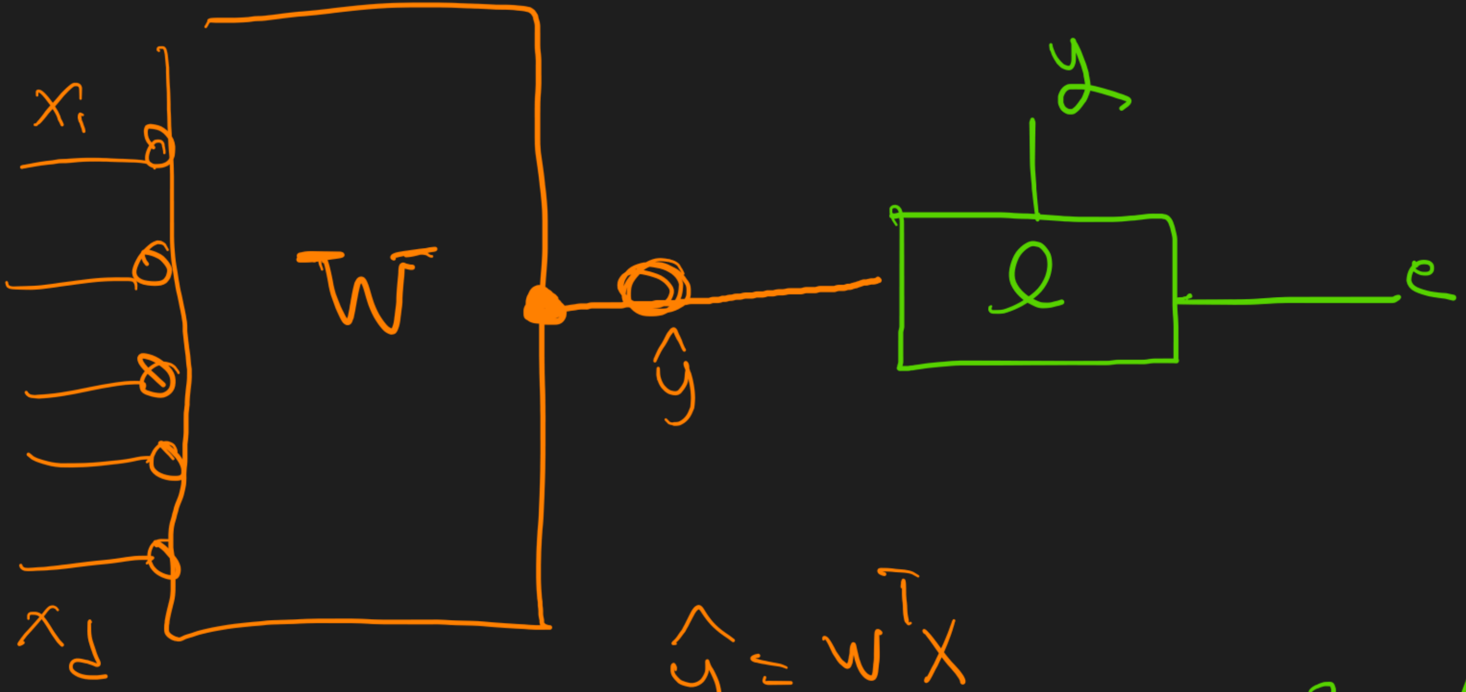
- Incorporate domain knowledge
- More on this later

- Is there a point in having more than one hidden layer?

FEED-FORWARD NNS FOR SUPERVISED LEARNING

- Loss function?
  - Regression
  - Classification
- Output layer?
  - Regression
  - Classification
- Other kinds of layers/architectures?

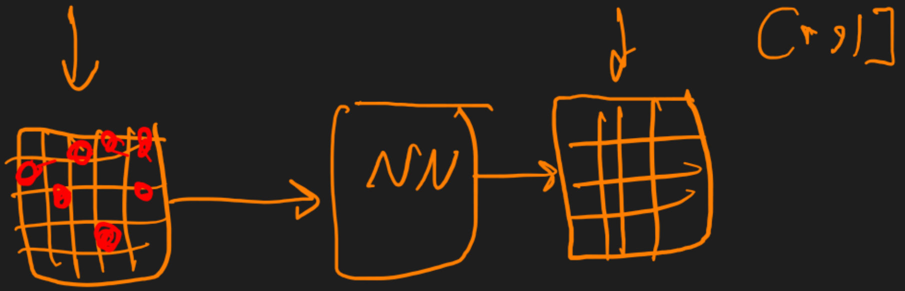
REGRESSION: LEAST SQUARES EXAMPLE



$$\hat{y} = w^T x$$

$$\ell = (\hat{y} - y)^2$$

```python
class LinearRegression(nn.Module):
  def __init__(self, input_dim, output_dim) -> None:
    super().__init__()
    self.fc = nn.Linear(input_dim, output_dim)

  def forward(self, x):
    return self.fc(x)

model = LinearRegression(input_size, output_size)

loss = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

for epoch in range(n_iters):
  y_pred = model(X)
  l = loss(Y,y_pred)
  l.backward()
  optimizer.step()
  optimizer.zero_grad()
```



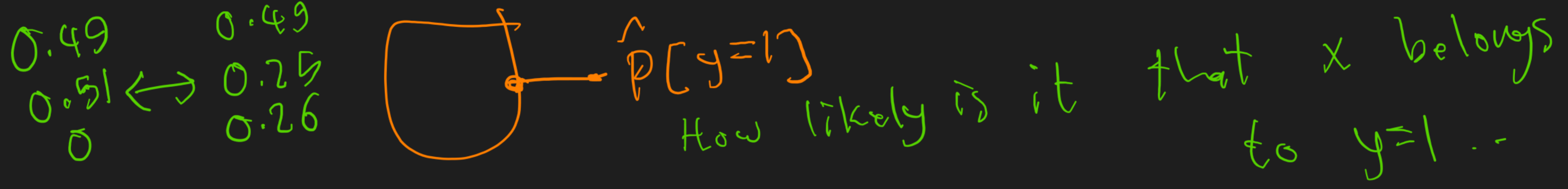
REGRESSION WITH NNS

- The output layer can be
  - Just linear (no activation function)
  - Linear + Rectified Linear Units (ReLU)
  - Linear + sigmoid
  - Linear + Hyperbolic tangent
- Loss function
  - Squared loss
  - Absolute loss (ℓ_1)
  - …

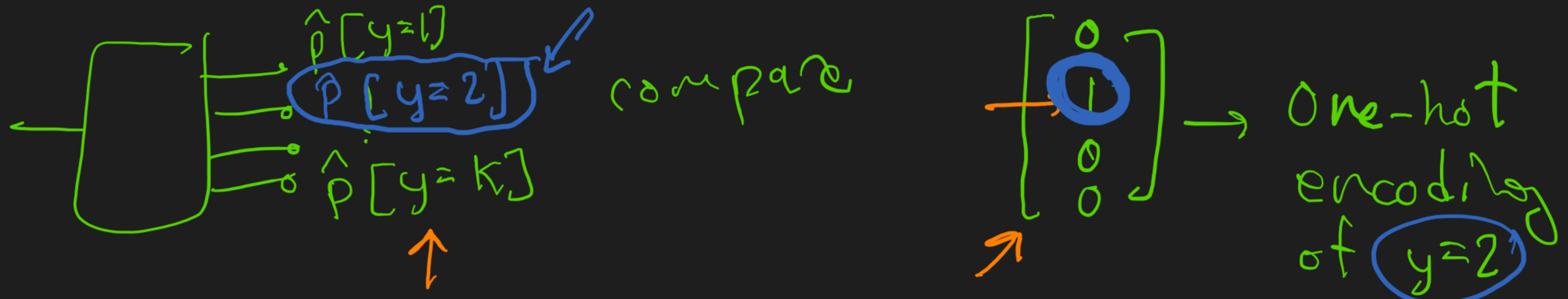
CLASSIFICATION WITH NNS

Number of outputs?

- Single output (binary classification)



- One-hot encoding (binary or multiclass)



CLASSIFICATION WITH NNS

OUTPUT LAYER AND LOSS FUNCTION?

- OUTPUT LAYER: LINEAR + "THRESHOLD/ARGMAX"?
- LOSS: 0-1 FUNCTION?

Hard to optimize --> so it is not used for training.

Classification with NNs

Output layer and loss function?

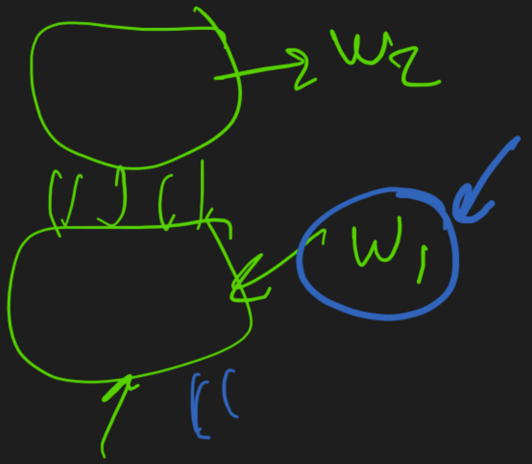
- Output layer: linear + Softmax
- Loss: negative log-likelihood (cross-entropy)
  - In pytorch, crossEntropy loss includes a softmax

LOGISTIC REGRESSION

```
class LogisticRegression(nn.Module):
    def __init__(self, input_size, num_classes):
        super(LogisticRegression, self).__init__()
        self.fc = nn.Linear(input_size, num_classes, bias=True)

    def forward(self, x):
        return self.fc(x)

model = LogisticRegression(input_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=learning_rate)
```

MULTIPLE LAYERS MODELS

```python
class NonlinearModel(nn.Module):
    def __init__(self, input_size, num_classes):
        super(ModifiedModel, self).__init__()
        self.fc1 = nn.Linear(input_size, 5000)
        self.fc2 = nn.Linear(5000, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        return self.fc2(x)
```



Kernels vs multilayer NNs

LINEAR CLASSIFICATION WITH HINGE LOSS

STOCHASTIC GRADIENT DESCENT

Algorithm 8.1 Stochastic gradient descent (SGD) update

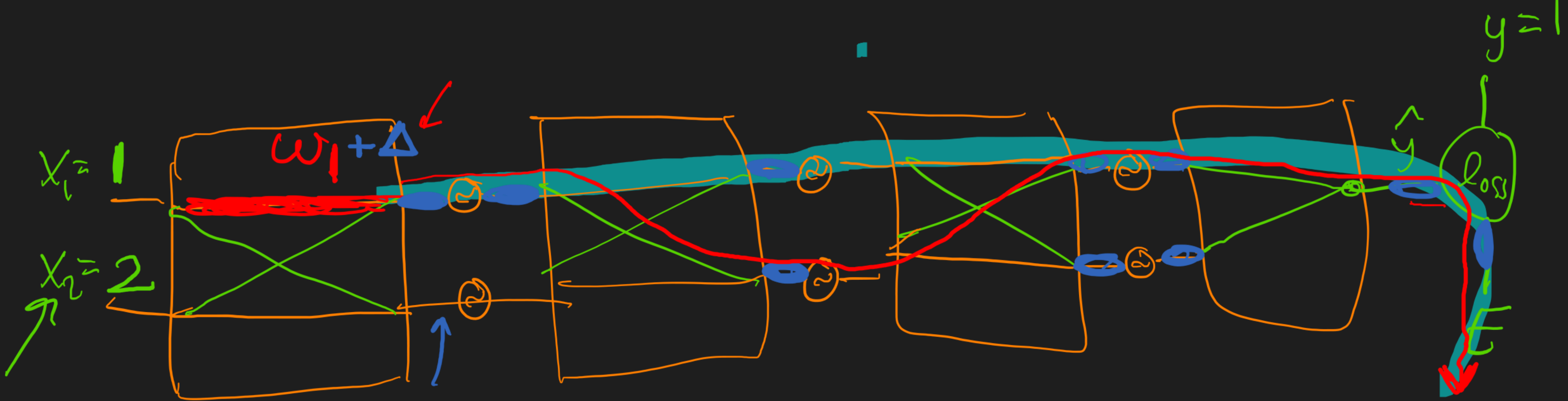
Require: Learning rate schedule $\epsilon_1, \epsilon_2, \ldots$
Require: Initial parameter $\boldsymbol{\theta}$

$k \leftarrow 1$
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$
  $k \leftarrow k + 1$
end while

WHAT IS MISSING?

- $\nabla_{\mathrm{w}}\left(\mathrm{E}(w,b)\right) = \sum_i \nabla_{\mathrm{w}}\left(l\left(f_{w,b}(x^i), y^i\right)\right)$
- How to calculate $\nabla_{\mathrm{w}}\left(l\left(f_{w,b}(x^i), y^i\right)\right)$?
- This can be computationally expensive

NAÏVE APPROACH



$$\frac{\partial E}{\partial w_1} =$$

There are exponentially many paths from w_1 to E.

COMPUTING THE GRADIENT?

- Naïve approach:
  - Computationally expensive for deep models
  - Some of the computations are repetitive
- What to do?
  - A kind of "dynamic programming"
  - Back propagation

BACK-PROPAGATION

- Use chain rule (for vector-valued functions)
- Linear time in terms of the number of weights!
- Forward phase
  - Compute the input/outputs of activation functions
- Backward phase
  - Compute the gradients, layer-by-layer

MULTI-VARIATE CHAIN RULE AND A MODULAR APPROACH