Cracking the AI Mind: Exploring Adversarial Examples, Neural Networks, and Linear Regression Models

Narayan Lamichhane, Professor Varun Chandrasekaran
Department of Chemical and Computer Engineering, The Grainger College of Engineering, University of Illinois at Urbana-Champaign

Introduction/Background

This poster presents an in-depth exploration of three pivotal concepts in artificial intelligence and machine learning: adversarial examples, neural networks, and linear regression models.

Neural Networks

Computational models inspired by the human brain's network of neurons

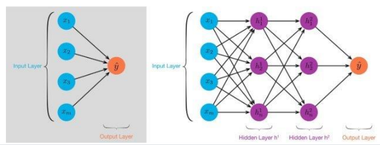
•Composed of three main structures: **neurons, layers, weights.**

• Neurons are things that hold numbers. They have an input layer which is typically shown as a vector

X = [x1, x2. …. X3]

•The Neurons carry a number from 0-1 (called activations) :in the input, the Neurons "light up" depending on the pixels that are brightest.

In the output the Neurons are a number from 0-1 but its meaning is the probability the Neural Network is confident that the prediction is accurate with the actual target

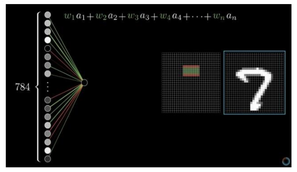


(Kang)

•Neural Networks have layers, that work together to piece numbers for example together to see if they make loops or lines to potentially determine the number.

•To connect the first layer to the second you can assign weights. Weights are adjustable parameters (like knobs)

•To go from one layer to the next, you assign weights to determine the strength of the connection between neurons.



$Z = n \sum i = 1 (w_i * x_i + b)$

$w_1x_1 + w_2x_2 + \ldots..w_nx_x$

(Sanders, Grant)

If you make the weights all 0 except the pixels that are colored, the weighted sum is just you adding the values in the region that as the shape on the grid. Bias is there if you want the total to be specifically (> 10)

Linear Regression In Neural Networks

•Statistical method used to model the relationship between a dependent variable and one or more independent variables.

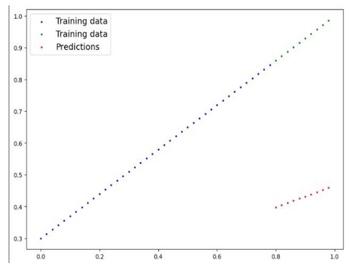
y=wx+b

•Splitting data into training and test sets. Machine learning goes from training set -> validation set -> test set

```
# Create *known* parameters
weight = 0.7 # same as b in Y = a + bX
bias = 0.3 # same as a in Y = a + bX

# Create
start = 0
end = 1
step = 0.02 # Gap
X = torch.arange(start, end, step).unsqueeze(dim=1)
y = weight * X + bias

X[:10], y[:10]
```

```
# create a train/test split
train_split = int(0.8 * len(X)) # splits data into two parts, 80% for training and 20% for testing
X_train, y_train = X[:train_split], y[:train_split] # this means X_train = X[:80] (it gets the data from train_split above)
X_test, y_test = X[train_split:], y[train_split:]

len(X_train), len(y_train), len(X_test), len(y_test) # output is (80 is split into two, 40 for x 40 for y) (20 is split to 10)
```

```
class LinearRegressionModelV2(nn.Module):
  def __init__(self):
    super().__init__()

    self.linear_layer = nn.Linear(in_features=1,
                                  out_features=1)

  # Forward method
  def forward(self, x: torch.Tensor) -> torch.Tensor:
    return self.linear_layer

# Set manual seed
torch.manual_seed(42)
model_1 = LinearRegressionModelV2()
model_1, model_1.state_dict()
```



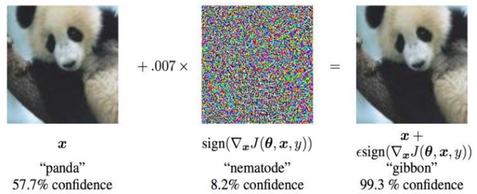

```
# Setup loss function
loss_fn = nn.L1Loss() # L1 loss

#set up optimizer
optimizer = torch.optim.SGD(lr=0.01, params=model_1.parameters()) # lr = learning rate
```

```
torch.manual_seed(42)

epochs = 200

for epoch in range(epochs):
  model_1.train()

  # 1. Forward Pass
  y_pred = model_1(X_train)

  # 2. Loss, how wrong our models predictions are
  loss = loss_fn(y_pred, y_train)

  # 3. Optimizer
  optimizer.zero_grad()
  # 4. Perform backprop
  loss.backward()

  optimizer.step
```

(Bourke, Daniel)

Adversarial Examples

•Goodfellow et al. found that many machine learning models, including state-of-the-art neural networks, are vulnerable to adversarial examples. These examples are inputs that are slightly perturbed but cause the models to misclassify them



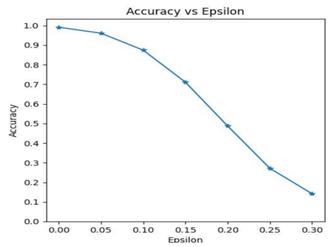
(Adversarial Example Generation/ Pytorch)

•Fast Gradient Sign Method (FGSM)

Method: This technique generates adversarial examples by adding a small perturbation in the direction of the gradient of the loss function with respect to the input.

•The linear nature of models in high-dimensional spaces means that small perturbations can accumulate to cause significant changes in output.

Mathematical Explanation: The dot product between the weight vector and an adversarial example grows with the dimensionality, making it possible to create a anges in ir



(Adversarial Example Generation/ Pytorch)

•Adversarial examples illustrates the model's robustness to adversarial perturbations. Epsilon (ϵ) represents the magnitude of the perturbations applied to the input data.

RESULTS/CONCLUSION:

•As ϵ increases, the strength of the adversarial attack increases.
•Small ϵ values cause minor perturbations, while large ϵ values cause significant changes.

```
# Define the LeNet model
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

```
# FGSM attack code
def fgsm_attack(image, epsilon, data_grad):
    sign_data_grad = data_grad.sign()
    perturbed_image = image + epsilon * sign_data_grad
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    return perturbed_image

# Test function
def test(model, device, test_loader, epsilon):
    correct = 0
    adv_examples = []

    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        data.requires_grad = True

        output = model(data)
        init_pred = output.max(1, keepdim=True)[1]

        if init_pred.item() != target.item():
            continue

        loss = F.nll_loss(output, target)
        model.zero_grad()
        loss.backward()
        data_grad = data.grad.data

        perturbed_data = fgsm_attack(data, epsilon, data_grad)
        output = model(perturbed_data)

        final_pred = output.max(1, keepdim=True)[1]
        if final_pred.item() == target.item():
            correct += 1
            if epsilon == 0:
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                adv_examples.append((init_pred.item(), final_pred.item(), adv_ex))
        else:
            adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
            adv_examples.append((init_pred.item(), final_pred.item(), adv_ex))

    final_acc = correct / float(len(test_loader))
    print(f"Epsilon: {epsilon}\tTest Accuracy = {correct} / {len(test_loader)} = {final_acc}")

    return final_acc, adv_examples
```

```
# Set device
print("CUDA Available: ", torch.cuda.is_available())
device = torch.device("cuda" if use_cuda and torch.cuda.is_available() else "cpu")

# Download the pretrained model
!wget {pretrained_model_url} -O {pretrained_model_path}

# Initialize the network
model = Net().to(device)

# Load the pretrained model
model.load_state_dict(torch.load(pretrained_model_path, map_location=device))
model.eval()

# MNIST Test dataset and dataloader
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, download=True, transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,)),
    ])),
    batch_size=1, shuffle=True)
```

```
# Run attack for each epsilon
accuracies = []
examples = []

for eps in epsilons:
    acc, ex = test(model, device, test_loader, eps)
    accuracies.append(acc)
    examples.append(ex)

# Plot accuracy vs epsilon
plt.figure(figsize=(8, 6))
plt.plot(epsilons, accuracies, "*-")
plt.title("Accuracy vs Epsilon")
plt.xlabel("Epsilon")
plt.ylabel("Accuracy")
plt.show()

# Plot several examples of adversarial samples at each epsilon
cnt = 0
plt.figure(figsize=(8, 10))
for i in range(len(epsilons)):
    for j in range(len(examples[i])):
        cnt += 1
        plt.subplot(len(epsilons), len(examples[0]), cnt)
        plt.xticks([], [])
        plt.yticks([], [])
        if j == 0:
            plt.ylabel(f"Eps: {epsilons[i]}", fontsize=14)
        orig, adv, ex = examples[i][j]
        plt.title(f"{orig} -> {adv}")
        plt.imshow(ex, cmap="gray")
plt.tight_layout()
plt.show()
```

References

Sanders, Grant. "3Blue1Brown." YouTube, YouTube, www.youtube.com/c/3blue1brown. Accessed 23 July 2024.

Bourke, Daniel. "Learn Pytorch for Deep Learning in a Day. Literally." YouTube, YouTube, 24 July 2022, www.youtube.com/watch?v=Z_ikDlimN6A&pp=ygUeZGFuaWVsIGJvdXJrZSBtYWNoaW5lIGxlYXJuaW5n.

Goodfellow, Ian J., et al. Explaining and Harnessing Adversarial Examples, arxiv.org/pdf/1412.6572. Accessed 24 July 2024.

Tibshirani, Ryan. "Lecture 3: Linear Regression and Prediction." Lecture 3: Linear Regression and Prediction, www.stat.berkeley.edu/~ryantibs/timeseries-f23/lectures/regression.pdf. Accessed 24 July 2024.

"Adversarial Example Generation¶." Adversarial Example Generation - PyTorch Tutorials 2.4.0+cu124 Documentation, pytorch.org/tutorials/beginner/fgsm_tutorial.html. Accessed 24 July 2024.

OpenAI. (2024). ChatGPT (July 2024 version) [Large language model]. Retrieved from https://www.openai.com/chatgpt

Kang, Nahua. "Multi-Layer Neural Networks with Sigmoid Function- Deep Learning for Rookies (2)." Medium, Towards Data Science, 4 Feb. 2019, towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f.

Future Work

Extend the study by incorporating adversarial training into the model. See how this impacts the model's robustness from other datasets