NYU

NYU Summer Machine Learning Program

Presenter Name Here
Date Here

Neural Networks

Day 6

Learning Objectives

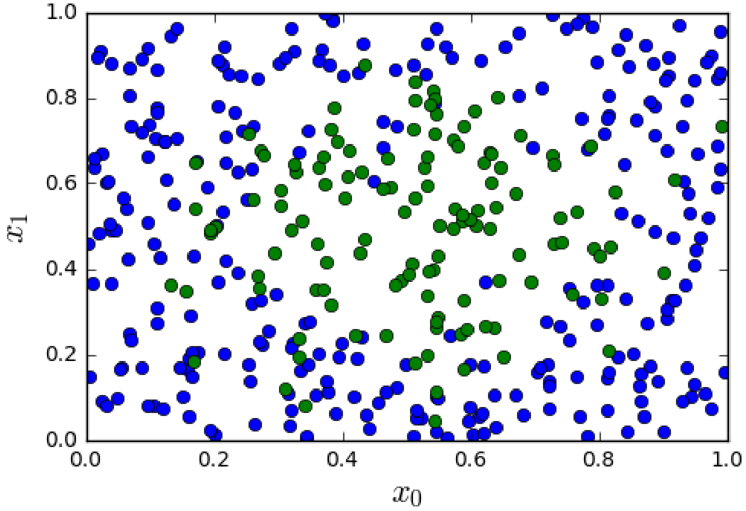
- Mathematically describe a neural network with a single hidden layer
  - Describe mappings for the hidden and output units
- Manually compute output regions for very simple networks
- Select the loss function based on the problem type
- Build and train a simple neural network in Keras
- Write the formulas for gradients using backpropagation
- Describe mini-batches in stochastic gradient descent

Outline

- Motivating Idea: Nonlinear classifiers from linear features
- Neural Networks
- Neural Network Loss Function
- Building and Training a Network in Keras
  - MNIST
- Backpropagation Training

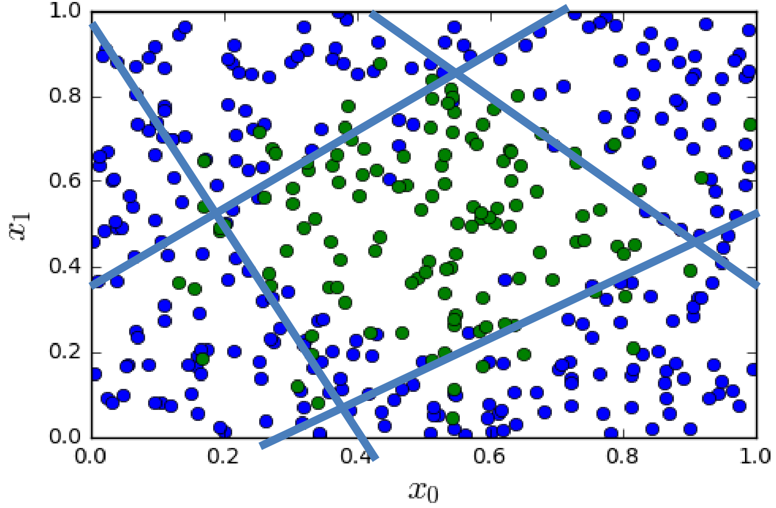
Motivation - Most Datasets are not Linearly Separable

- Consider simple synthetic data
  - See figure to the right
  - 2D features
  - Binary class label
- Not separated linearly

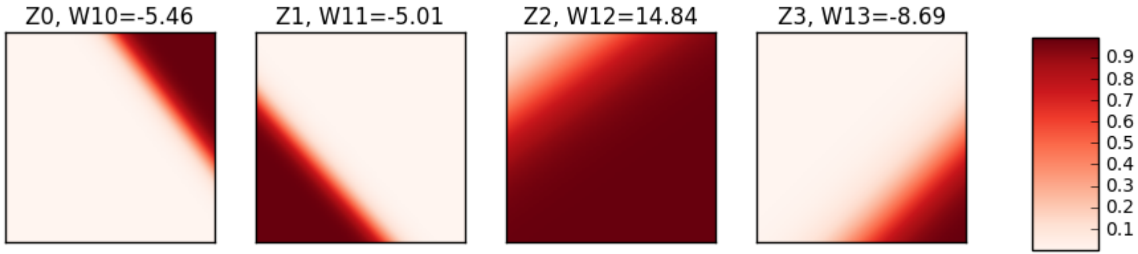


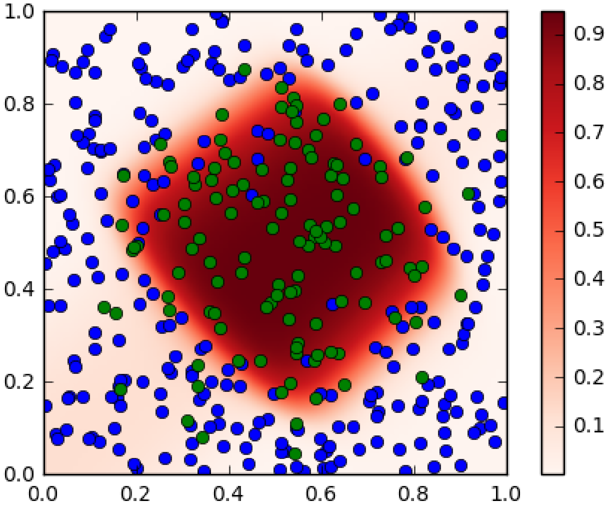
Motivation - From Linear to Nonlinear

- Idea: Build nonlinear region from linear decisions
- Possible form for a classifier:
  - Step 1: Classify into small number of linear regions
  - Step 2: Predict class label from step 1 decisions



Step 1 Outputs and Step 2 Outputs





- Each output from step 1 is from a linear classifier with soft decision
  - Like logistic regression
- Final output is a weighted average of step 1 outputs using the weights
  - Weights are indicated on top of the figures

A Possible Two Stage Classifier

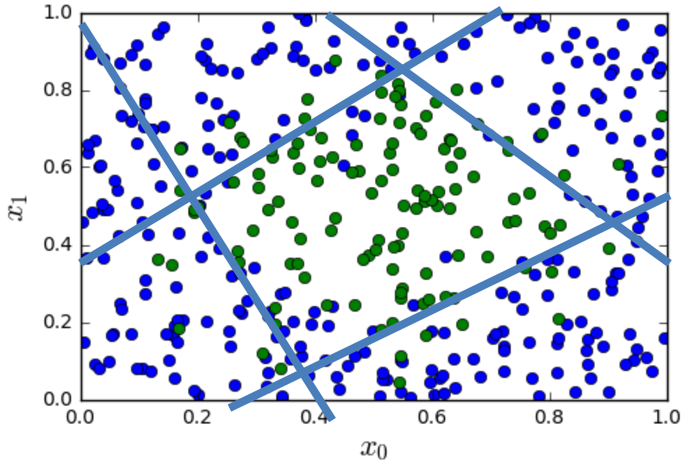
- Input sample: $\boldsymbol{x} = (x_1, x_2)^T$
- First step: Hidden layer
  - Take $N_H = 4$ linear discriminants

$$z_{H,1} = \mathbf{w}_{H,1}^T x + b_{H,1}$$
$$\vdots$$
$$z_{H,N_H} = \mathbf{w}_{H,M}^T x + b_{H,M}.$$

  - Make a soft decision on each linear region

$$u_{H,m} = g(z_{H,m}) = \frac{1}{1 + e^{-z_{H,m}}}$$

- Second step: Output layer
  - Linear step $z_O = w_O^T u_H + b_O$
  - Soft decision: $u_O = g(z_O)$

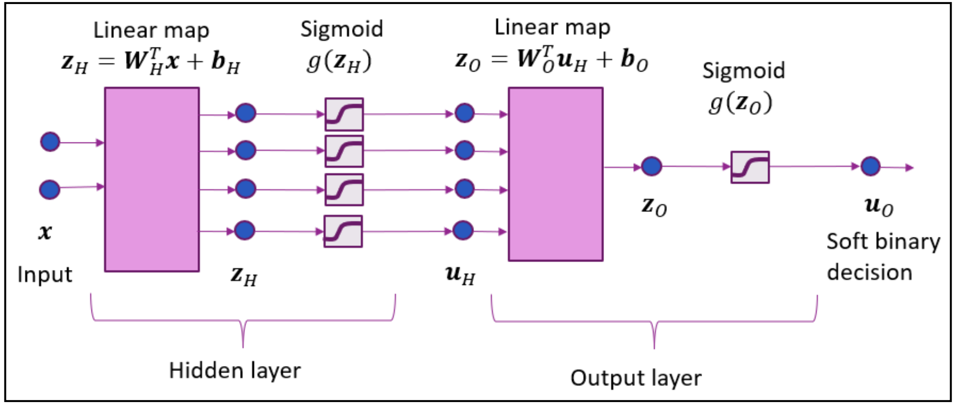


Model Block Diagram

- ❑ Hidden layer: $\boldsymbol{z}_H = \boldsymbol{W}_H^T \boldsymbol{x} + \boldsymbol{b}_H, \quad \boldsymbol{u}_H = g(\boldsymbol{z}_H)$
- ❑ Output layer: $\boldsymbol{z}_O = \boldsymbol{W}_O^T \boldsymbol{u}_H + \boldsymbol{b}_O, \; u_O = g(\boldsymbol{z}_O)$

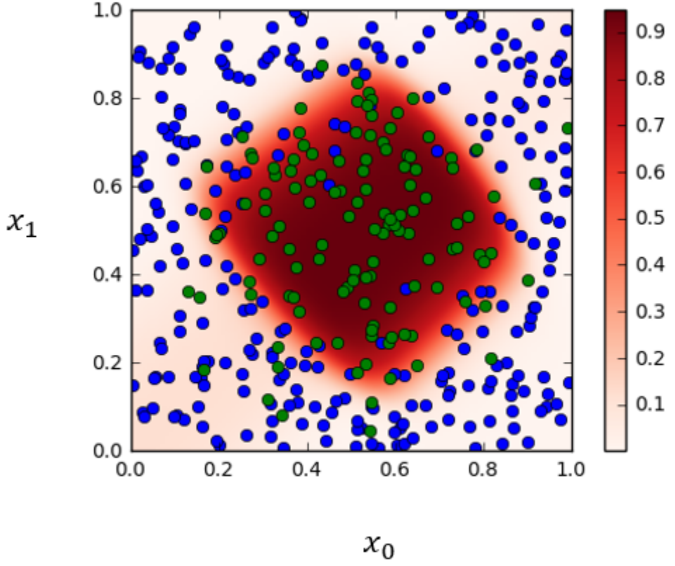
❑Each hidden node is a linear classifier with soft decision (Logistic regression)

❑Final output is a weighted average of step 1 outputs using the weights indicated on top of the figures



Training the Model

- Model in matrix form:
  - Hidden layer: $\boldsymbol{z}_H = \boldsymbol{W}_H^T \boldsymbol{x} + \boldsymbol{b}_H, \quad \boldsymbol{u}_H = g(\boldsymbol{z}_H)$
  - Output layer: $z_O = \boldsymbol{W}_O^T \boldsymbol{u}_H + \boldsymbol{b}_O, \; u_O = g(z_O)$
- $z_O = F(\boldsymbol{x}, \theta)$: Linear output from final stage
  - Parameters: $\theta = (\boldsymbol{W}_H, \boldsymbol{W}_O, b_H, b_O)$
- Get training data $(\boldsymbol{x}_i, y_i), \; i = 1, \ldots, N$
- Define loss function: $L(\theta) \coloneqq -\sum_{i=1}^{N} y_{true} \ln P(y_i | x_i, \theta)$
- Pick parameters to minimize loss
- Will discuss how to do this minimization later



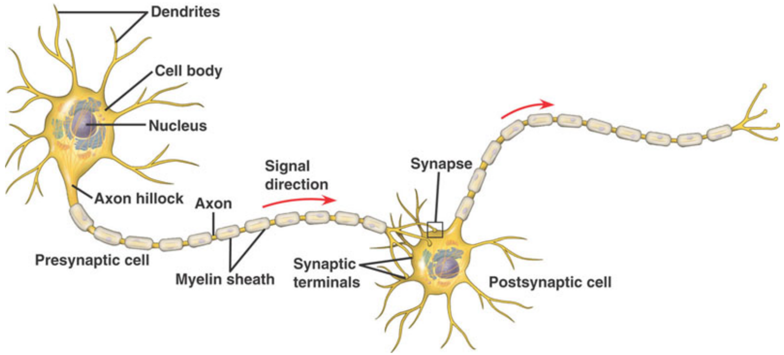
NYU

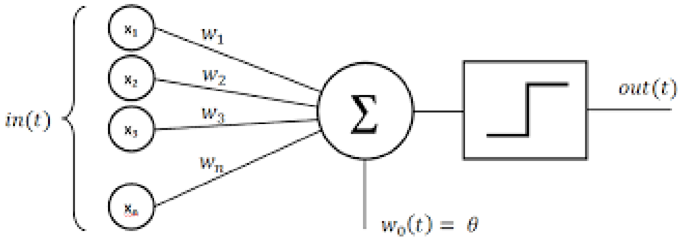
Outline

- Motivating Idea: Nonlinear classifiers from linear features
- Neural Networks
- Neural Network Loss Function
- Building and Training a Network in Keras
  - MNIST
- Backpropagation Training

Neural Networks - Inspiration from Biology

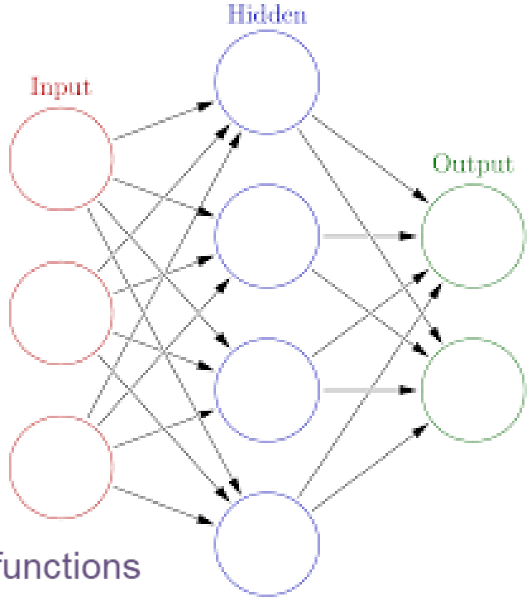
- Simple model of neurons
  - Dendrites: Input currents from other neurons
  - Soma: Cell body, accumulation of charge
  - Axon: Outputs to other neurons
  - Synapse: Junction between neurons
- Operation:
  - Take weighted sum of input current
  - Outputs when sum reaches a threshold
- Each neuron is like one unit in neural network





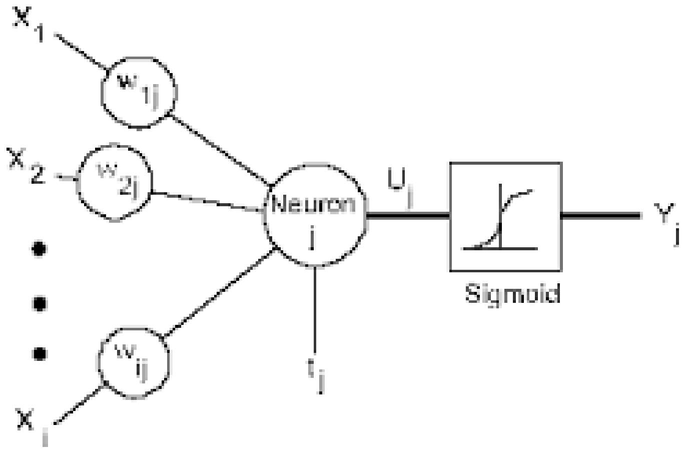
General Structure

- Input: $\boldsymbol{x} = (x_1, \cdots, x_d)$
  - N_I = number of features
- Hidden layer:
  - Linear transform: $\boldsymbol{z}_H = \boldsymbol{W}_H^T \boldsymbol{x} + \boldsymbol{b}_H$
  - Activation function: $\boldsymbol{u}_H = g_{act}(\boldsymbol{z}_H)$
  - Dimension: N_H hidden units
- Output layer:
  - Linear transform: $\boldsymbol{z}_O = \boldsymbol{W}_O^T \boldsymbol{u}_H + \boldsymbol{b}_O$
  - Output function: $\boldsymbol{u}_O = g_{out}(\boldsymbol{z}_O)$
  - Dimension: N_O = number of classes / outputs
- Can be used for classification or regression, with different decision functions



Terminology

- Hidden variables: the variables $\boldsymbol{z}_H, \boldsymbol{u}_H$
  - These are not directly observed
- Hidden units: The functions that compute:
  - $z_{H,i} = \sum_j W_{H,ji} x_j + b_{H,i}\,,\;\; u_{H,i} = g(z_{H,i})$
  - The function $g(z)$ called the activation function
- Output units: The functions that compute
  - $z_{O,i} = \sum_j W_{O,ji} u_{H,j} + b_{O,i}$

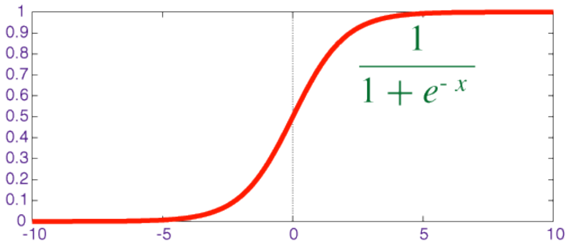


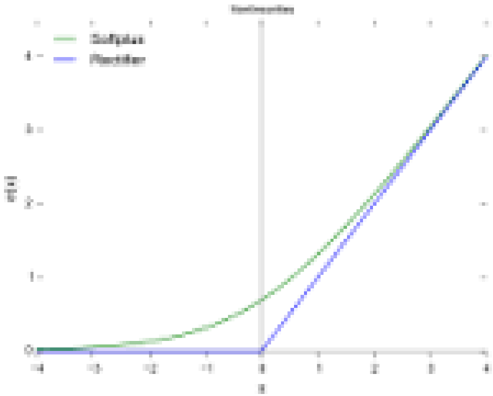
Response Map or Output Activation

- Last layer depends on type of response
- Binary classification: $y=\pm 1$
  - z_O is a scalar
  - Hard decision: $y_{pred} = \text{sign}\,(z_O)$
  - Soft decision: $P(y = 1 \,|\, x) = 1/(1 + e^{-z_O})$
- Multi-class classification: $y=1,\ldots,K$
  - $\mathbf{z}_O = \left[z_{O,1}, \cdots, z_{O,K}\right]^T$ is a vector
  - Hard decision: $\hat{y} = \arg\max_k z_{O,k}$
  - Soft decision: $P(y = k|x) = S_k(\mathbf{z}_O)$, $S_k(\mathbf{z}_O) = \frac{e^{z_{O,k}}}{\sum_\ell e^{z_{O,\ell}}}$ (SoftMax)
- Regression: $\mathbf{y} \in R^K$
  - $\mathbf{y}_{pred} = \mathbf{z}_O$ (linear output layer)

Hidden Activation Function

- Two common activation functions
- Sigmoid:
  - $g_{act}(z) = 1/(1+e^{-z})$
  - Benefits: Values are bounded
  - Often used for small networks
- Rectified linear unit (ReLU):
- $g_{act}(z) = \max(0, z)$
  - Often used for larger networks





Number of Parameters

Layer	Parameter	Symbol	Number parameters
Hidden layer	Bias	b_H	N_H
	Weights	W_H	$N_H N_I$
Output layer	Bias	b_O	N_O
	Weights	W_O	$N_O N_H$
Total			$N_H(N_I + 1) + N_O(N_H + 1)$

- Sizes:
  - N_I = input dimension, N_H = number of hidden units, N_O = output dimension
- N_H = number of hidden units is a free parameter

Outline

- Motivating Idea: Nonlinear classifiers from linear features
- Neural Networks
- Neural Network Loss Function
- Building and Training a Network in Keras
  - MNIST
- Backpropagation Training

Training a Neural Network

- Given data: $(\boldsymbol{x}_i, y_i)$, i=1,…,N
- Learn parameters: θ = (W_H, b_H, W_o, b_o)
  - Weights and biases for hidden and output layers
- Will minimize a loss function: $L(\theta)$

$$\theta' = \arg\min_{\theta} L(\theta)$$

- $L(\theta)$ = measures how well parameters θ fit training data $(\boldsymbol{x}_i, y_i)$

Selecting the Right Loss Function

- Depends on the problem type
- Always compare final output z_{Oi} with target y_i

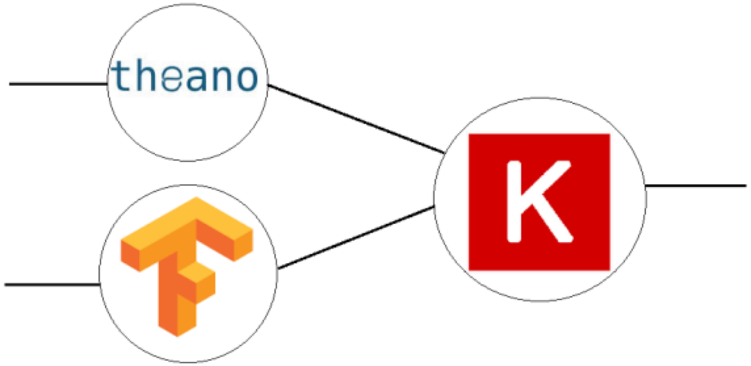
Problem	Target y_i	Output z_{Oi}	Loss function	Formula
Regression	y_i = Scalar real	z_{Oi} = Prediction of y_i Scalar output / sample	Squared / L2 loss	$\sum_i (y_i - z_{Oi})^2$
Regression with vector samples	$\boldsymbol{y}_i = (y_{i1}, \ldots, y_{iK})$	z_{Oik} = Prediction of y_{ik} K outputs / sample	Squared / L2 loss	$\sum_{ik} (y_{ik} - z_{Oik})^2$
Binary classification	$y_i = \{0,1\}$	z_{Oi} = "logit" score Scalar output / sample u_i is the SoftMax of z_{Oi}	Binary cross entropy	$\sum_i y_i \, ln(u_i) + (1 - y_i)\ln(1 - u_i)$
Multi-class classification	$y_i = \{1, \ldots, K\}$	z_{Oik} = "logit" scores K outputs / sample	Categorical cross entropy	$\sum_{k=1} r_{ik} \ln[P(y_i = k \mid x_i, \theta)]$

Outline

- Motivating Idea: Nonlinear classifiers from linear features
- Neural Networks
- Neural Network Loss Function
- Building and Training a Network in Keras
  - MNIST
- Backpropagation Training

Python Deep Learning Library - Keras

- High-level neural network language in Python
- Runs on top of a backend
    - Much simpler than raw backend language
    - Very fast coding
    - Uniform language for all backend
- Keras has been incorporated into TF
- But…
    - Slightly less flexible
    - Not as fast sometimes
- In this class, we use Keras

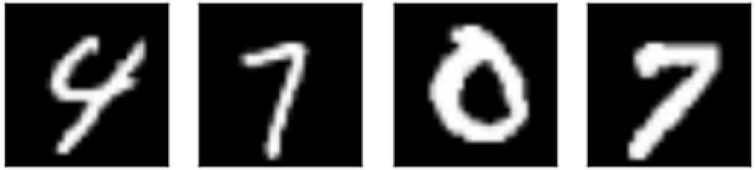


Keras Recipe

- Step 1. Describe model architecture
  - Number of hidden units, output units, activations, …
- Step 2. Select an optimizer
- Step 3. Select a loss function and compile the model
- Step 4. Fit the model
- Step 5. Test / use the model

Example: MNIST data

- Classic MNIST problem:
  - Detect hand-written digits
  - Each image is 28 x 28 = 784 pixels
- Dataset size:
  - 50,000 training digits
  - 10,000 test
  - 10,000 validation (not used here)
- Can be loaded with sklearn and many other packages

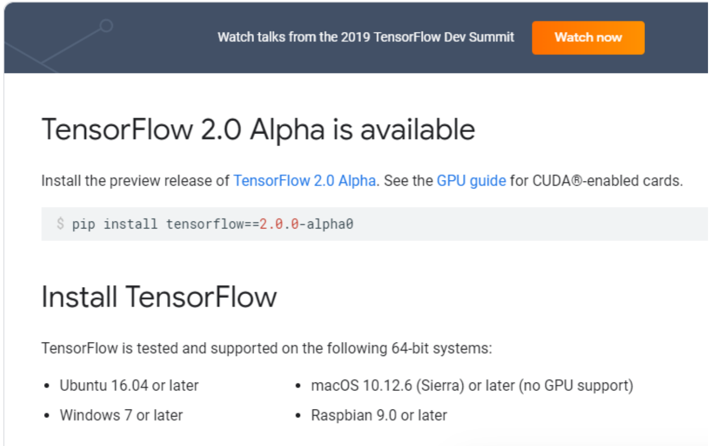


Step 0: Import the Packages

https://www.tensorflow.org/install

- Install TensorFlow
- For this lab, you can use the CPU version
- If you are using Google Collaboratory, TF is pre-installed

```
import tensorflow as tf
```



Step 1: Define Model

```
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Activation
```

```
import tensorflow.keras.backend as K
K.clear_session()
```

- Load modules for layers
- Clear graph (extremely important!)
- Build model
  - This example: dense layers
  - Give each layer a dimension, name & activation

```
nin = Xtr.shape[1]  # dimension of input data
nh = 100      # number of hidden units
nout = int(np.max(ytr)+1)    # number of outputs = 10 since there are 10 classes
model = Sequential()
model.add(Dense(units=nh, input_shape=(nin,), activation='sigmoid', name='hidden'))
model.add(Dense(units=nout, activation='softmax', name='output'))
```

Step 2, 3: Select an Optimizer & Compile

- Adam optimizer generally works well for most problems
  - In this case, had to manually set learning rate
  - You often need to play with this.
- Use binary cross-entropy loss
- Metrics indicate what will be printed in each epoch

```
from tensorflow.keras import optimizers

opt = optimizers.Adam(lr=0.001) # beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

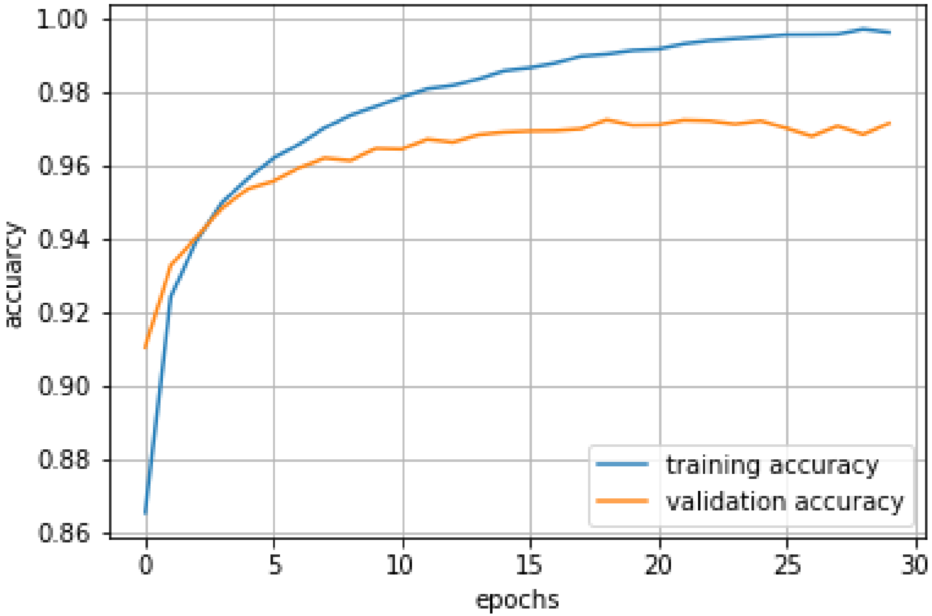
Step 4: Fit the Model

- Use Keras fit function
  - Specify number of epoch & batch size
- Prints progress after each epoch
  - Loss = loss on training data
  - Acc = accuracy on training data

```
Epoch 1/30
60000/60000 [==============================] - 3s 58us/sample - loss: 0.5095 - acc: 0.8685 - val_loss: 0.2804 - val_acc: 0.9214
Epoch 2/30
60000/60000 [==============================] - 3s 52us/sample - loss: 0.2570 - acc: 0.9268 - val_loss: 0.2155 - val_acc: 0.9386
Epoch 3/30
60000/60000 [==============================] - 3s 50us/sample - loss: 0.2018 - acc: 0.9435 - val_loss: 0.1806 - val_acc: 0.9474
Epoch 4/30
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1656 - acc: 0.9528 - val_loss: 0.1522 - val_acc: 0.9560
Epoch 5/30
60000/60000 [==============================] - 3s 50us/sample - loss: 0.1423 - acc: 0.9595 - val_loss: 0.1398 - val_acc: 0.9603
Epoch 6/30
60000/60000 [==============================] - 3s 49us/sample - loss: 0.1250 - acc: 0.9651 - val_loss: 0.1274 - val_acc: 0.9629
Epoch 7/30
60000/60000 [==============================] - 3s 49us/sample - loss: 0.1087 - acc: 0.9691 - val_loss: 0.1129 - val_acc: 0.9659
Epoch 8/30
60000/60000 [==============================] - 3s 49us/sample - loss: 0.0980 - acc: 0.9718 - val_loss: 0.1097 - val_acc: 0.9663
Epoch 9/30
60000/60000 [==============================] - 3s 50us/sample - loss: 0.0884 - acc: 0.9751 - val_loss: 0.0987 - val_acc: 0.9713
Epoch 10/30
60000/60000 [==============================] - 3s 50us/sample - loss: 0.0800 - acc: 0.9777 - val_loss: 0.1009 - val_acc: 0.9701
```

Training and Validation Accuracy

- Training accuracy continues to increase
- Validation accuracy eventually flattens and sometimes starts to decrease.
- Should stop when the validation accuracy starts to decrease.
- This indicates overfitting.



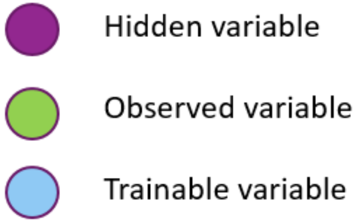

```
tr_accuracy = hist.history['acc']
val_accuracy = hist.history['val_acc']

plt.plot(tr_accuracy)
plt.plot(val_accuracy)
plt.grid()
plt.xlabel('epochs')
plt.ylabel('accuarcy')
plt.legend(['training accuracy', 'validation accuracy'])
```

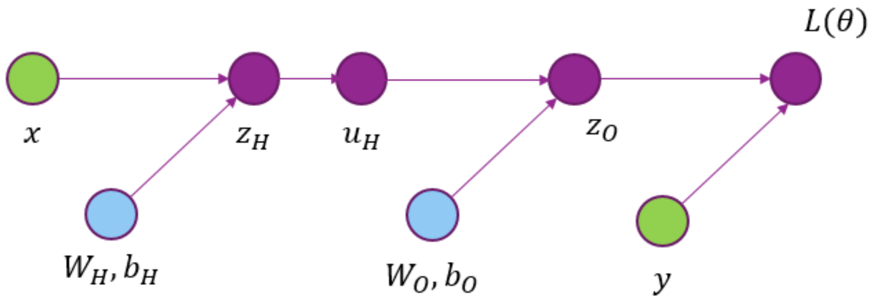
Outline

- Motivating Idea: Nonlinear classifiers from linear features
- Neural Networks
- Neural Network Loss Function
- Building and Training a Network in Keras
  - MNIST
- Backpropagation Training

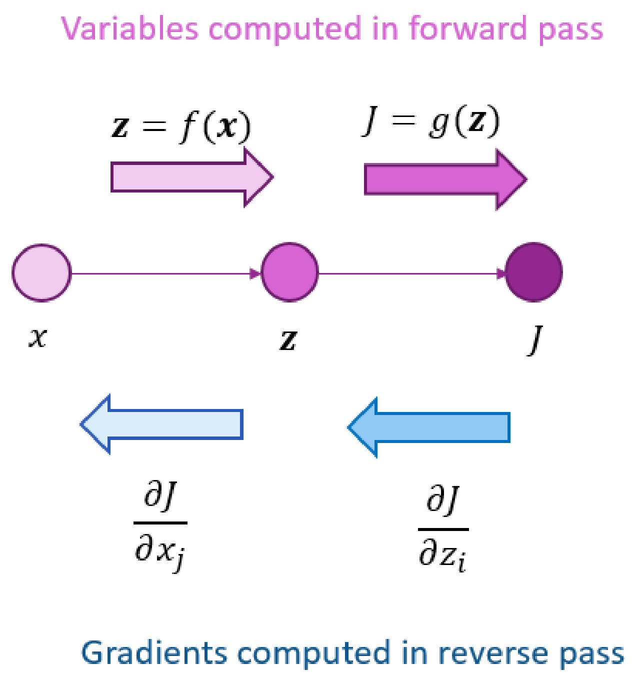
Computation Graph & Forward Pass

- Neural network loss function can be computed via a computation graph
- Sequence of operations starting from measured data and parameters
- Loss function computed via a forward pass in the computation graph
  - $z_{H,i} = W_H\, x_i + b_H$
  - $u_{H,i} = g_{act}\,(z_{H,i})$
  - $z_{O,i} = W_O\, u_{H,i} + b_O$
  - $L = \sum_i L_i(z_{O,i}, y_i)$

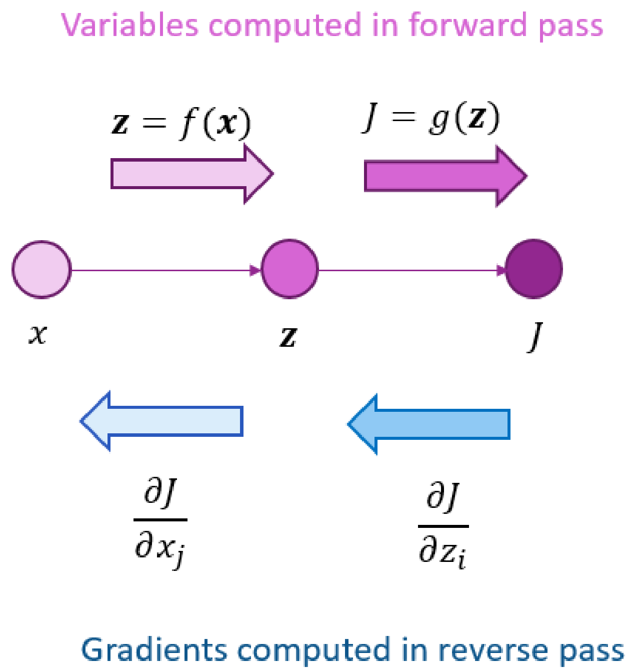




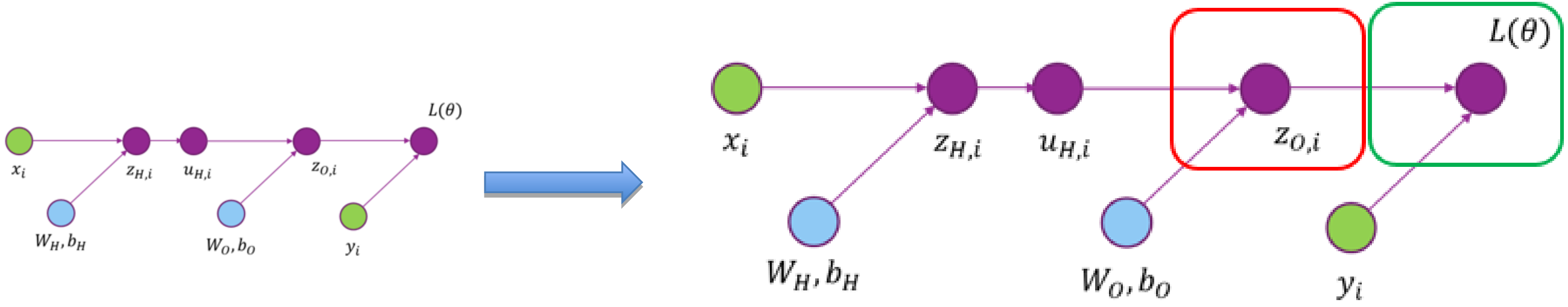
Back-Propagation on A Two Node Graph



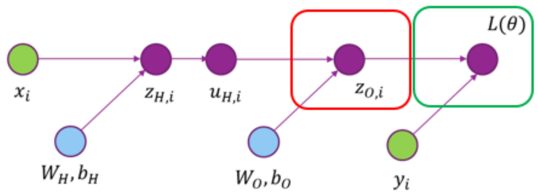
Back-Propagation on A Two Node Graph

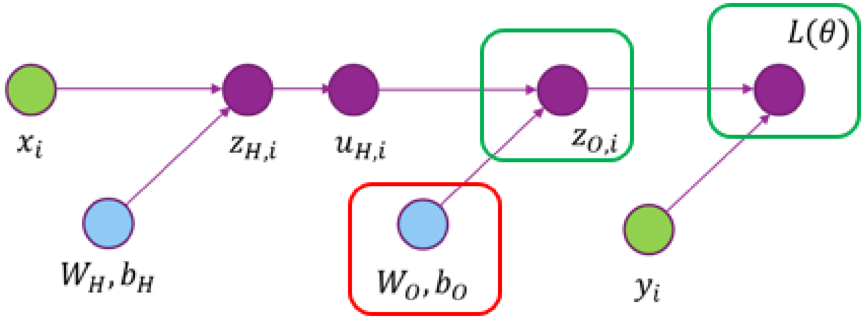


Back-Prop on a General Computation Graph

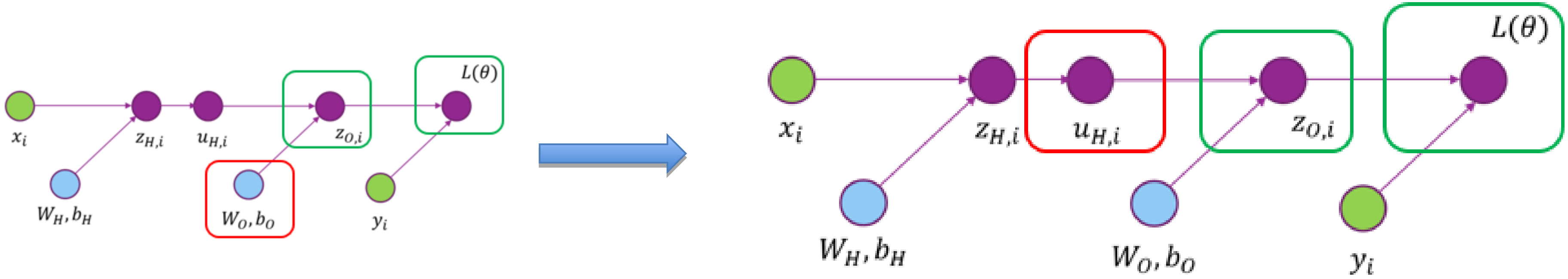


Back-Prop on a General Computation Graph

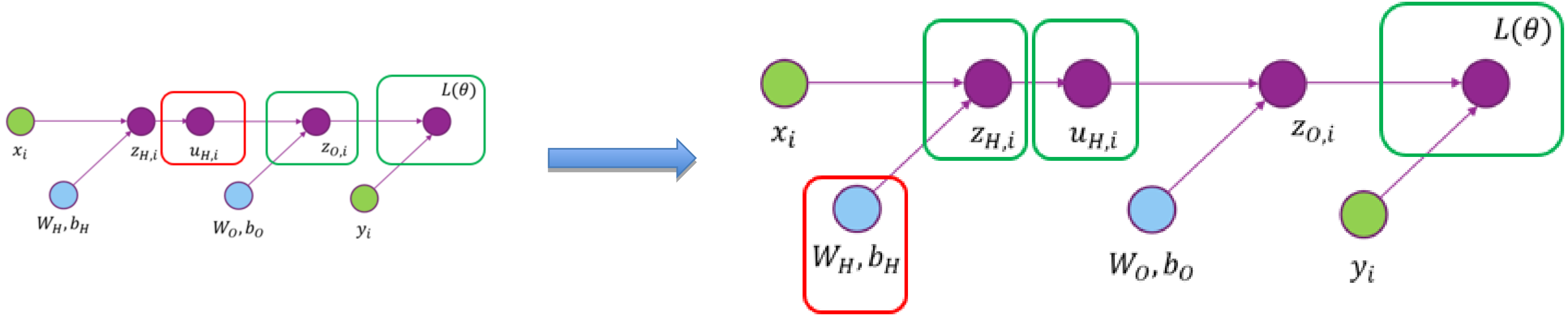




Back-Prop on a General Computation Graph



Back-Prop on a General Computation Graph



Gradients Descent

- For neural net problem: $\theta = (W_H,\ b_H,\ W_o,\ b_o)$
- Gradient is computed using back-propagation
- Gradient descent is performed on each parameter:

$$W_H \leftarrow W_H - \alpha \nabla_{W_H} L(\theta),$$

$$b_H \leftarrow b_H - \alpha \nabla_{b_H} L(\theta),$$

….

Regularization in Keras

- Activity regularization tries to make the output at each layer small or sparse.

- `kernel_regularizer` : instance of `keras.regularizers.Regularizer`
- `bias_regularizer` : instance of `keras.regularizers.Regularizer`
- `activity_regularizer` : instance of `keras.regularizers.Regularizer`

Example

```
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01),
                activity_regularizer=regularizers.l1(0.01)))
```

Available penalties

```
keras.regularizers.l1(0.)
keras.regularizers.l2(0.)
keras.regularizers.l1_l2(0.)
```

Choice of network parameters

- Number of layers (typically not more than 2)
- Number of hidden units in the hidden layer
- Regularization level
- Learning rate
- Determined by maximizing the cross-validation error through typically exhaustive search

Learning Objectives

- Mathematically describe a neural network with a single hidden layer
    - Describe mappings for the hidden and output units
- Manually compute output regions for very simple networks
- Select the loss function based on the problem type
- Build and train a simple neural network in Keras
- Describe mini-batches in stochastic gradient descent
- Importance of regularization
- Hyperparameter optimization

Thank You!