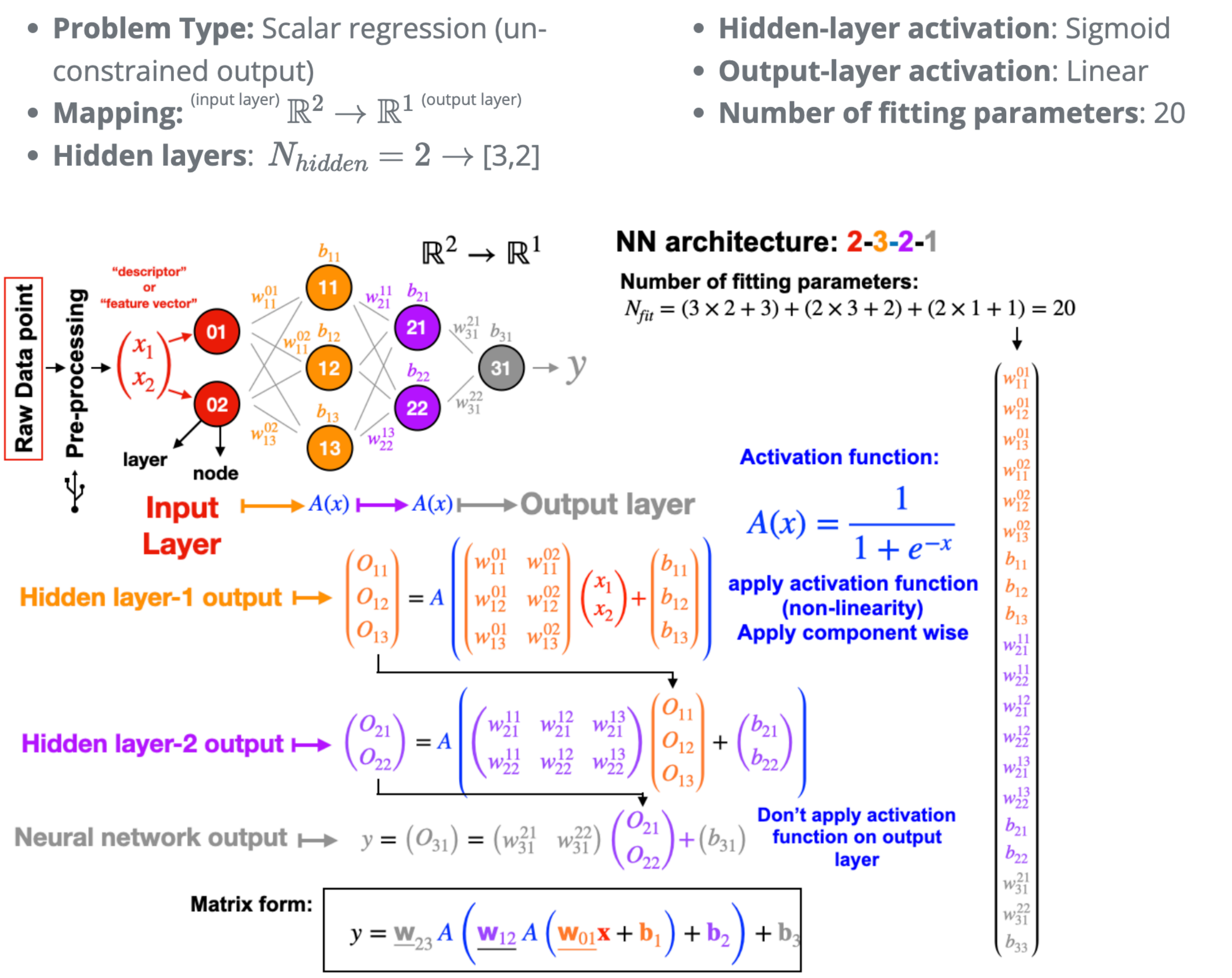
Exam study-guide: Deep learning

Shallow Neural Networks

- Be able to draw shallow models (e.g. linear and logistic regression as simple networks)
  - Study the following notes section: click here

Dense Feed-Forward Neural Networks

- Understand the architecture of dense feed-forward neural networks.
  - Study the following notes section: click here
- **For example**:
- **Simple Neural network for scalar regression: 2 feature input and 1 target output**
  - Note a **better** choice for hidden layer activate would be RELU here (to help combat the vanishing gradient effect)



- **Problem Type:** Scalar regression (un-constrained output)
- **Mapping:** (input layer) $\mathbb{R}^2 \to \mathbb{R}^1$ (output layer)
- **Hidden layers:** $N_{hidden} = 2 \to$ [3,2]
- **Hidden-layer activation**: Sigmoid
- **Output-layer activation**: Linear
- **Number of fitting parameters**: 20

NN architecture: 2-3-2-1

Number of fitting parameters:

$$N_{fit} = (3 \times 2 + 3) + (2 \times 3 + 2) + (2 \times 1 + 1) = 20$$

Activation function:

$$A(x) = \frac{1}{1 + e^{-x}}$$

apply activation function (non-linearity) Apply component wise

Hidden layer-1 output $\mapsto \begin{pmatrix} O_{11} \\ O_{12} \\ O_{13} \end{pmatrix} = A\left(\begin{pmatrix} w_{11}^{01} & w_{11}^{02} \\ w_{12}^{01} & w_{12}^{02} \\ w_{13}^{01} & w_{13}^{02} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_{11} \\ b_{12} \\ b_{13} \end{pmatrix} \right)$

Hidden layer-2 output $\mapsto \begin{pmatrix} O_{21} \\ O_{22} \end{pmatrix} = A\left(\begin{pmatrix} w_{21}^{11} & w_{21}^{12} & w_{21}^{13} \\ w_{22}^{11} & w_{22}^{12} & w_{22}^{13} \end{pmatrix} \begin{pmatrix} O_{11} \\ O_{12} \\ O_{13} \end{pmatrix} + \begin{pmatrix} b_{21} \\ b_{22} \end{pmatrix} \right)$

Neural network output $\mapsto y = (O_{31}) = \begin{pmatrix} w_{31}^{21} & w_{31}^{22} \end{pmatrix} \begin{pmatrix} O_{21} \\ O_{22} \end{pmatrix} + (b_{31})$

Don't apply activation function on output layer

Matrix form:

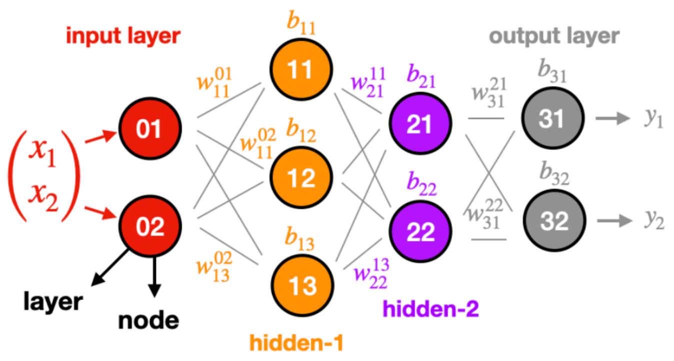
$$\boxed{y = \underline{\mathbf{w}}_{23} A\left(\underline{\mathbf{w}}_{12} A\left(\underline{\mathbf{w}}_{01} \mathbf{x} + \mathbf{b}_1 \right) + \mathbf{b}_2 \right) + \mathbf{b}_3}$$

$$\begin{pmatrix} w_{11}^{01} & w_{12}^{01} & w_{13}^{01} & w_{11}^{02} & w_{12}^{02} & w_{13}^{02} & b_{11} & b_{12} & b_{13} & w_{21}^{11} & w_{22}^{11} & w_{21}^{12} & w_{22}^{12} & w_{21}^{13} & w_{22}^{13} & b_{21} & b_{22} & w_{31}^{21} & w_{31}^{22} & b_{33} \end{pmatrix}^T$$

- **Simple Neural network for vector regression: 2 feature input and 2 target output**

- **Problem Type:** Vector regression
- **Mapping:** $^{\text{(input layer)}}\ \mathbb{R}^2 \rightarrow \mathbb{R}^2\ ^{\text{(output layer)}}$
- **Hidden layers:** $N_{hidden} = 2 \rightarrow$ [3,2]
- **Hidden-layer activation**: ReLu
- **Output-layer activation**: Linear
- **Number of fitting parameters**: 23

Computational graph



$$\mathbf{y} = (y_1, y_2) = \underline{\mathbf{w}_{23}}\, A\left(\underline{\mathbf{w}_{12}}\, A\left(\underline{\mathbf{w}_{01}}\mathbf{x} + \mathbf{b}_1\right) + \mathbf{b}_2\right) + \mathbf{b}_3$$

$$A(x) = max\,(0,x) \quad (relu)$$

Note: We didn't have to use `relu` on the *hidden layers*, other activations would also work

Be able to do the following:

- Determine the number of parameters in each layer, and in the total network, given node counts for various layers.
- Given node counts, activation functions, and weight matrices, be able to:
  - Draw a network accurately.
  - Perform step-by-step computations to evaluate a very small dense feed forward model:
    1. Matrix multiplication
    2. Vector addition (bias)
    3. Component-wise activation
- Understand and be able to draw common hidden-layer activation functions:
  - Sigmoid, Tanh, ReLU, Leaky ReLU
  - Clearly label critical points and characteristics of activation functions.
  - See the following section of the notes for examples: click here

Learning Tasks and Output Layer Design

- Given specific learning tasks, know how to determine:
  - Number of output layer nodes.
  - Appropriate loss functions:
    - Mean Squared Error (MSE) for scalar/vector regression
    - Binary Cross-Entropy (BCE) for binary classification
    - Categorical Cross-Entropy (CCE) for multi-class classification
  - Choosing the correct activation functions for the output layer.
  - **For example:**
    - Vector regression mapping from R^{50} to R^{20} where R^n is the continuous spaced defined by the intersection of n mutually orthogonal coordinate axes (i.e. number lines)
      - Input layer size: 50
      - Output layer size: 20
      - Output activation: Linear
      - Loss: MSE
    - Sentiment analysis (binary classification) with a 10000 word vocabulary and OHE vectorization
      - Input layer size: 10000
      - Output layer size: 1 (probability of positive class)
      - Output activation: Sigmoid
      - Loss: BCE
    - MNIST, 10 class multi class classification problem (with 28x28 images (matrices), reshaped into (784,1) vectors)
      - Nodes in input layer: 784
      - Nodes in output layer: 10 node output layer
      - Output activation: Soft-max activation

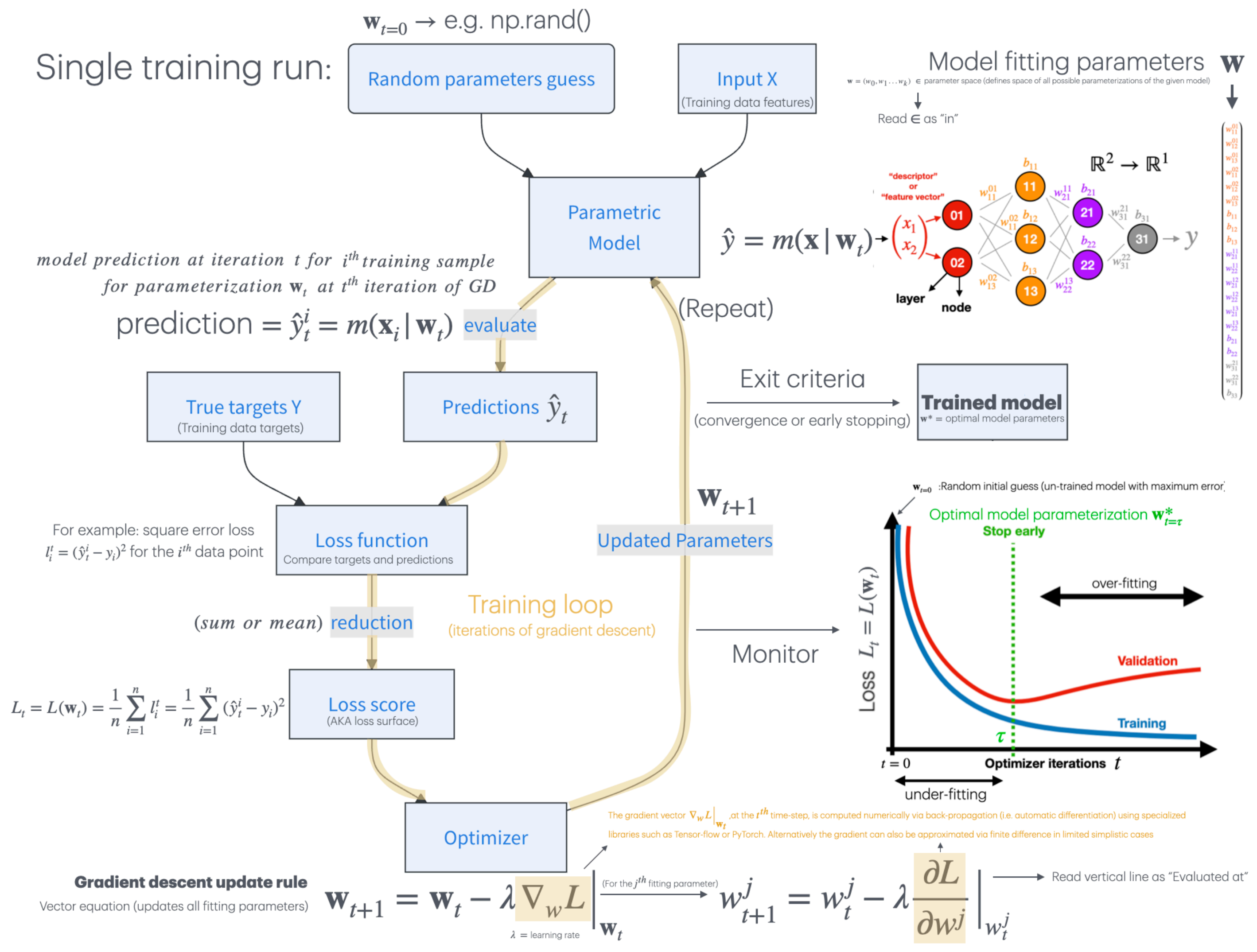
- Loss: CCE

Metrics vs. Loss

- Understand the difference between:
  - Loss: directly optimized by the training process (seen by optimizer, used to find optimal model parameters).
  - Metrics: monitored but not directly optimized (not seen by optimizer).
- Be familiar with common metrics used for Regression (MSE, MAE, MAPE, etc) & Classification (accuracy, precision, recall, F1)

Training Process Diagram

- Be able to describe, and annotate the following diagram which summarizes the training process, and possibly evaluate the progression for a very simple model, by taking a few steps of gradient descent manually (given the gradient vector)
- Understand training curves for training and validation loss, i.e. loss as a function of gradient decent iteration index t, understand what is happening in different regions of the curve (under-fitting, optimal fit, over-fitting), and the Concept of early stopping to avoid overfitting
- **Study the following workflow diagram carefully and ensure you fully understand it. If you do, you will have a solid grasp of how the vast majority of modern deep learning methods operate (e.g., MLP, CNN, RNN, GNN, LSTM, Auto-regression, ARIMA, Transformer models, etc.).** Remember, for parametric machine learning the goal is to learn optimal model parameters via optimization, i.e. **Training = Optimization**. Most models follow some variation of this workflow. In special cases (e.g., linear regression), the optimization problem can be solved analytically (by hand), and numerical optimization isn't necessary. However, this is an exception. Typically, we must solve the optimization numerically, such as through gradient descent or its variations, as outlined in the following diagram.



$\mathbf{w}_{t=0} \rightarrow$ e.g. np.rand()
Single training run:
Random parameters guess
Input X
(Training data features)
Model fitting parameters $\mathbf{w}$
$\mathbf{w} = (w_0, w_1 \dots w_k) \in$ parameter space (defines space of all possible parameterizations of the given model)
Read ∈ as "in"
$\mathbb{R}^2 \rightarrow \mathbb{R}^1$
"descriptor" or "feature vector"
$\hat{y} = m(\mathbf{x} \,|\, \mathbf{w}_t)$
layer
node
Parametric Model
model prediction at iteration t for i^{th} training sample for parameterization $\mathbf{w}_t$ at t^{th} iteration of GD
prediction $= \hat{y}^i_t = m(\mathbf{x}_i \,|\, \mathbf{w}_t)$
evaluate
(Repeat)
True targets Y
(Training data targets)
Predictions $\hat{y}_t$
Exit criteria
(convergence or early stopping)
Trained model
$\mathbf{w}^*$ = optimal model parameters
$\mathbf{w}_{t+1}$
Updated Parameters
For example: square error loss
$l^t_i = (\hat{y}^i_t - y_i)^2$ for the i^{th} data point
Loss function
Compare targets and predictions
(sum or mean) reduction
Training loop
(iterations of gradient descent)
Monitor
$L_t = L(\mathbf{w}_t) = \frac{1}{n}\sum_{i=1}^{n} l^t_i = \frac{1}{n}\sum_{i=1}^{n} (\hat{y}^i_t - y_i)^2$
Loss score
(AKA loss surface)
Optimizer
$\mathbf{w}_{t=0}$:Random initial guess (un-trained model with maximum error)
Optimal model parameterization $\mathbf{w}^*_{t=\tau}$
Stop early
over-fitting
Validation
Training
Loss $L_t = L(\mathbf{w}_t)$
τ
$t=0$
Optimizer iterations t
under-fitting
The gradient vector $\nabla_w L|_{\mathbf{w}_t}$ at the t^{th} time-step, is computed numerically via back-propagation (i.e. automatic differentiation) using specialized libraries such as Tensor-flow or PyTorch. Alternatively the gradient can also be approximated via finite difference in limited simplistic cases
Gradient descent update rule
Vector equation (updates all fitting parameters)
$\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda \nabla_w L \big|_{\mathbf{w}_t}$
λ = learning rate
(For the j^{th} fitting parameter)
$w^j_{t+1} = w^j_t - \lambda \frac{\partial L}{\partial w^j}\Big|_{w^j_t}$
Read vertical line as "Evaluated at"

Standard Deep Learning Workflow

- Understand the standard deep learning workflow, both **conceptually**, and as **implemented in Keras,** i.e. the following should be very familiar and intuitive to you at this point in the course (note how this is a very general process that applies to most parametric models in some form or another)
  - Implementation of the workflow in Keras can be found here: click here

1. **Raw Data Analysis**:

   - Input data: from the input data X and target Y data be able to tell what type of design decisions can be made about the neural network modeling
      1. e.g. size of input layer, size of output layer, activation on output layer, loss function
2. **Data Preprocessing**: Normalization and One-hot encoding
3. **Data Splitting**: Training, Validation, Testing
4. **Parametric Model Definition**: $y_{pred} = \hat{y} = m(\mathbf{x}|\mathbf{w})$
   1. e.g. linear regression, logistic regression, or neural network
5. **Loss Function Definition**:
   - Loss function definition $L(\mathbf{w})$, for given learning task, to score the quality of a particular parameterization $\mathbf{w}$.
6. **Parameter Initialization**:
   - Random initial guess for model parameters $\mathbf{w}_{t=0}$ where t is an iteration variable for gradient descent (this instantiates a particular version, i.e. parameterization, of the model). Since it is un-trained this random model parameterization will generally be bad (high loss/error)
7. **Training Loop (Gradient Descent)**: over iteration index t
   - Model is trained through the training data to optimize model parameters via gradient descent or one of it's "cousins" (Adam, RMSprop, etc)
   - Iteratively optimize parameters:
     - Full vector update: $\mathbf{w}_{t+1} = \mathbf{w}_t - \lambda \nabla_{\mathbf{w}} L(\mathbf{w}_t)$
       - This is the vector expression to update all fitting parameters at once
     - For a single fitting parameter, i.e. the j^{th} fitting parameter, the formula is just $w^j_{t+1} = w^j_t - \lambda \frac{\partial L}{\partial w^j_t}$
   - Parameter stops updating when the gradient is zero.
   - Keep iterating until the convergence or the onset of overfitting (terminate run via early stopping if needed)
   - Monitor validation loss to avoid over fitting
8. **Final Model Evaluation**:
   - Final model transferability check: evaluate on the test data and compare metrics to validation and training set

Regularization Techniques

- Understand the principles and effects of:
  - L1 regularization
  - L2 regularization
- Explain how these regularizations modify the loss function and influence model parameters.
- Note: Dropout regularization is not included on the exam.

Back-propagation and Differentiation

- **Not required**: no back-propagation computations on the exam.