Numerical Methods and Machine Learning for Image Processing

Week 7: Basic Neural Nets, Part 2

November 14, 2021

Damon M. Chandler and Yi Zhang

Last time: Basic Neural Nets, Part 1b

- **Perceptron model**
  1. Perceptron code
- **Multilayer Perceptron model**
  1. MLP math
  2. MLP code

Today: Basic Neural Nets, Part 2

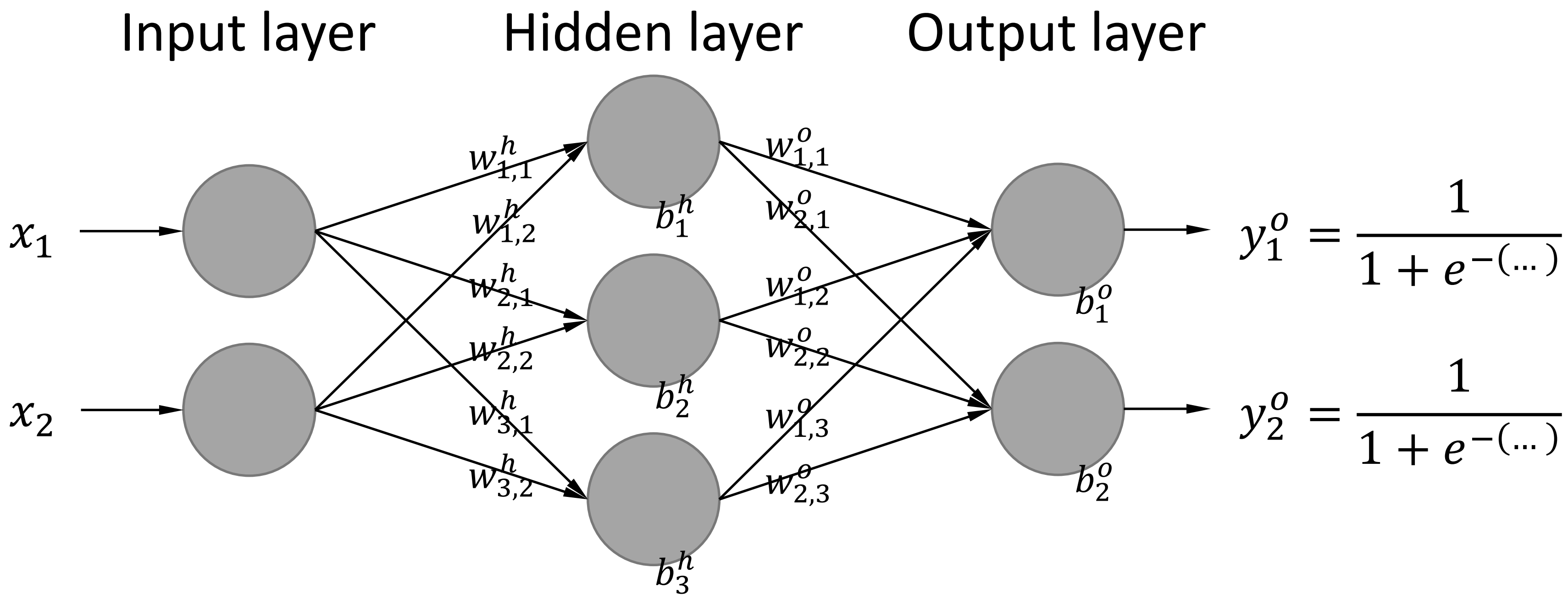
- **Multilayer Perceptron model**
  1. MLP with multiple outputs (math)
  2. MLP with multiple outputs for regression (code)
  3. MLP with multiple outputs for classification (code)

Today: Basic Neural Nets, Part 2

- **Multilayer Perceptron model**
  1. MLP with multiple outputs (math)
  2. MLP with multiple outputs for regression (code)
  3. MLP with multiple outputs for classification (code)

MLP with multiple outputs

MLP with multiple outputs



$$y^o_1 = \frac{1}{1+e^{-\left(b^o_1+w^o_{1,1}y^h_1+w^o_{1,2}y^h_2+w^o_{1,3}y^h_3\right)}}$$

$$y^o_2 = \frac{1}{1+e^{-\left(b^o_2+w^o_{1,1}y^h_1+w^o_{1,2}y^h_2+w^o_{1,3}y^h_3\right)}}$$

$$y^h_1 = \frac{1}{1+e^{-\left(b^h_1+w^h_{1,1}x_1+w^h_{1,2}x_2\right)}}$$

$$y^h_2 = \frac{1}{1+e^{-\left(b^h_2+w^h_{2,1}x_1+w^h_{2,2}x_2\right)}}$$

$$y^h_3 = \frac{1}{1+e^{-\left(b^h_3+w^h_{3,1}x_1+w^h_{3,2}x_2\right)}}$$

MLP with multiple outputs

$$E = \frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2 = \frac{1}{2}\sum_{k=1}^{2}\left(y_k - \frac{1}{1 + e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}\right)^2$$

$$\frac{\partial E}{\partial w_{k,j}^o} = \frac{\partial E}{\partial y_k^o}\frac{\partial y_k^o}{\partial w_{k,j}^o} = -(y_k - y_k^o)\frac{\partial y_k^o}{\partial w_{k,j}^o}$$

$$\frac{\partial E}{\partial y_1^o} = \frac{\partial}{\partial y_1^o}\left\{\frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2\right\} = \frac{\partial}{\partial y_1^o}\left\{\frac{1}{2}\left((y_1 - y_1^o)^2 + (y_2 - y_2^o)^2\right)\right\}$$

$$= -(y_1 - y_1^o)$$

$$\frac{\partial E}{\partial y_2^o} = \frac{\partial}{\partial y_2^o}\left\{\frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2\right\} = \frac{\partial}{\partial y_2^o}\left\{\frac{1}{2}\left((y_1 - y_1^o)^2 + (y_2 - y_2^o)^2\right)\right\}$$

$$= -(y_2 - y_2^o)$$

i = input node index
j = hidden node index
k = output node index

MLP with multiple outputs

$$E = \frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2 = \frac{1}{2}\sum_{k=1}^{2}\left(y_k - \frac{1}{1+e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}\right)^2$$

$$\frac{\partial E}{\partial w_{k,j}^o} = \frac{\partial E}{\partial y_k^o}\frac{\partial y_k^o}{\partial w_{k,j}^o} = -(y_k - y_k^o)\frac{\partial y_k^o}{\partial w_{k,j}^o} = -(y_k - y_k^o)y_j^h y_k^o(1 - y_k^o)$$

$$\frac{\partial E}{\partial y_k^o} = \frac{\partial}{\partial y_k^o}\left\{\frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2\right\} = -(y_k - y_k^o)$$

$$\frac{\partial y_k^o}{\partial w_{k,1}^o} = \frac{y_1^h e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}{\left(1+e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}\right)^2} = y_1^h y_k^o(1 - y_k^o)$$

$$\frac{\partial y_k^o}{\partial w_{k,2}^o} = \frac{y_2^h e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}{\left(1+e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}\right)^2} = y_2^h y_k^o(1 - y_k^o)$$

$$\frac{\partial y_k^o}{\partial w_{k,3}^o} = \frac{y_3^h e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}{\left(1+e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}\right)^2} = y_3^h y_k^o(1 - y_k^o)$$

i = input node index
j = hidden node index
k = output node index

MLP with multiple outputs

$$E = \frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2 = \frac{1}{2}\sum_{k=1}^{2}\left(y_k - \frac{1}{1+e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}\right)^2$$

$$\frac{\partial E}{\partial b_k^o} = \frac{\partial E}{\partial y_k^o}\frac{\partial y_k^o}{\partial b_k^o} = -(y_k - y_k^o)\frac{\partial y_k^o}{\partial b_k^o} = -(y_k - y_k^o)1y_k^o(1 - y_k^o)$$

$$\frac{\partial E}{\partial y_k^o} = \frac{\partial}{\partial y_k^o}\left\{\frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2\right\} = -(y_k - y_k^o)$$

$$\frac{\partial y_k^o}{\partial b_k^o} = \frac{1e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}{\left(1+e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}\right)^2} = 1y_k^o(1 - y_k^o)$$

i = input node index
j = hidden node index
k = output node index

MLP with multiple outputs

$$\frac{\partial E}{\partial w_{k,j}^{o}} = -(y_k - y_k^o)y_j^h y_k^o(1 - y_k^o) \qquad \frac{\partial E}{\partial b_k^{o}} = -(y_k - y_k^o)1y_k^o(1 - y_k^o)$$

How to change the weights (*update rule*):

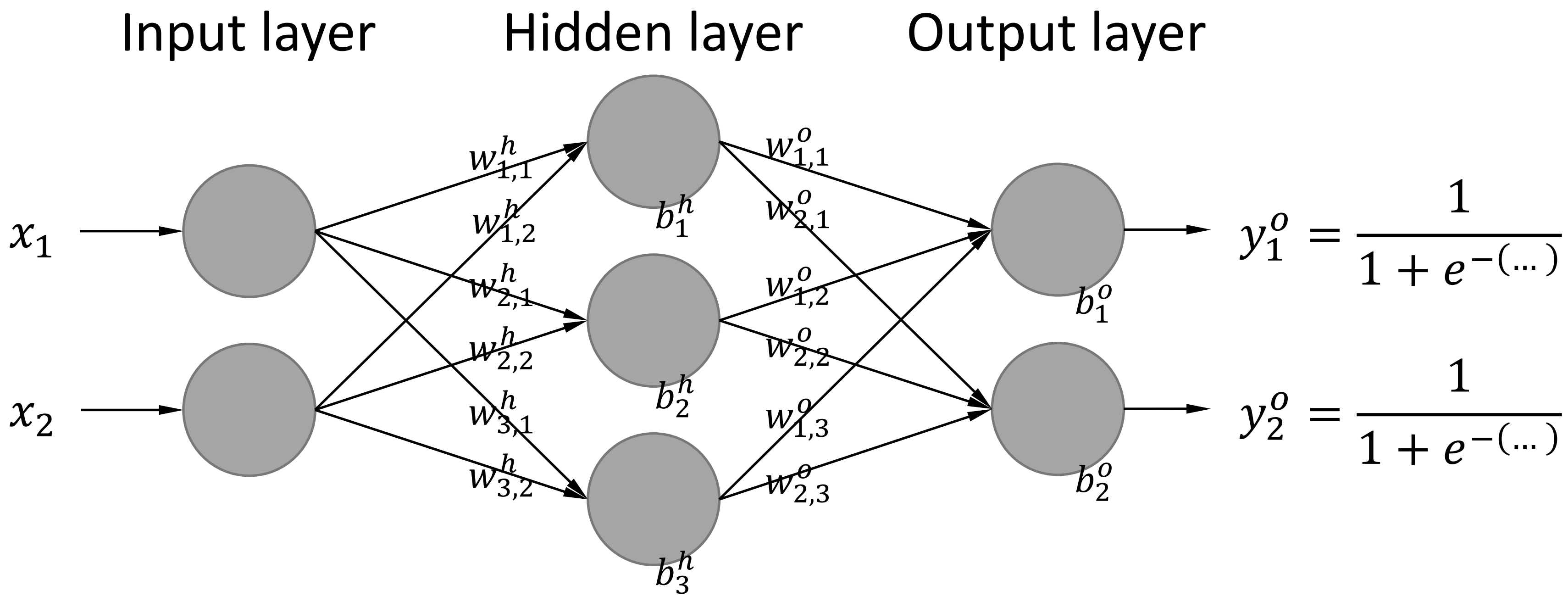
$$w_{k,j}^o = w_{k,j}^o - \eta\frac{\partial E}{\partial w_{k,j}^{o}} = w_{k,j}^o + \eta(y_k - y_k^o)y_j^h y_k^o(1 - y_k^o)$$

$$b_k^o = b_k^o - \eta\frac{\partial E}{\partial b_k^{o}} = b_k^o + \eta(y_k - y_k^o)y_k^o(1 - y_k^o)$$

- lrate is η
- tgt is y
- out is y^o
- err is $(y - y^o)$
- hid[h_idx] is y_j^h
- wo[h_idx] is w_j^o
- bo is b^o
- i_idx is input index i
- h_idx is input index j
- o_idx is input index k

```
err = tgt - out # vector-vector subtraction
delta_out[o_idx] = out[o_idx] * (1.0 - out[o_idx]) * err[o_idx]

wo[o_idx, h_idx] += lrate * hid[h_idx] * delta_out[o_idx]
bo[o_idx] += lrate * delta_out[o_idx]
```

MLP with multiple outputs



$$y^o_1 = \frac{1}{1+e^{-\left(b^o_1 + w^o_{1,1}y^h_1 + w^o_{1,2}y^h_2 + w^o_{1,3}y^h_3\right)}}$$

$$y^o_2 = \frac{1}{1+e^{-\left(b^o_2 + w^o_{1,1}y^h_1 + w^o_{1,2}y^h_2 + w^o_{1,3}y^h_3\right)}}$$

$$y^h_1 = \frac{1}{1+e^{-\left(b^h_1 + w^h_{1,1}x_1 + w^h_{1,2}x_2\right)}}$$

$$y^h_2 = \frac{1}{1+e^{-\left(b^h_2 + w^h_{2,1}x_1 + w^h_{2,2}x_2\right)}}$$

$$y^h_3 = \frac{1}{1+e^{-\left(b^h_3 + w^h_{3,1}x_1 + w^h_{3,2}x_2\right)}}$$

MLP with multiple outputs

$$E = \frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2 = \frac{1}{2}\sum_{k=1}^{2}\left(y_k - \frac{1}{1+e^{-\left(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h\right)}}\right)^2$$

$$\frac{\partial E}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial w_{j,i}^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial w_{j,i}^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial w_{j,i}^h}$$

$$\frac{\partial E}{\partial y_1^o} = \frac{\partial}{\partial y_k^o}\left\{\frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2\right\} = -(y_1 - y_1^o)$$

$$\frac{\partial E}{\partial y_2^o} = \frac{\partial}{\partial y_k^o}\left\{\frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2\right\} = -(y_2 - y_2^o)$$

$$y_j^h = \frac{1}{1+e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}$$

i = input node index
j = hidden node index
k = output node index

MLP with multiple outputs

$$E = \frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2 = \frac{1}{2}\sum_{k=1}^{2}\left(y_k - \frac{1}{1+e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}\right)^2$$

$$\frac{\partial E}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial w_{j,i}^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial w_{j,i}^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial w_{j,i}^h}$$

$$\frac{\partial E}{\partial y_1^o} = -(y_1 - y_1^o) \qquad \frac{\partial E}{\partial y_2^o} = -(y_2 - y_2^o)$$

$$\frac{\partial y_1^o}{\partial y_j^h} = \frac{w_{1,j}^o e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}{\left(1+e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}\right)^2} = w_{1,j}^o y_1^o (1 - y_1^o)$$

$$\frac{\partial y_2^o}{\partial y_j^h} = \frac{w_{2,j}^o e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}{\left(1+e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}\right)^2} = w_{2,j}^o y_2^o (1 - y_2^o)$$

$$y_j^h = \frac{1}{1+e^{-(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2)}}$$

i = input node index
j = hidden node index
k = output node index

MLP with multiple outputs

$$E = \frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2 = \frac{1}{2}\sum_{k=1}^{2}\left(y_k - \frac{1}{1 + e^{-\left(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h\right)}}\right)^2$$

$$\frac{\partial E}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial w_{j,i}^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial w_{j,i}^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial w_{j,i}^h}$$

$$\frac{\partial E}{\partial y_1^o} = -(y_1 - y_1^o) \qquad \frac{\partial E}{\partial y_2^o} = -(y_2 - y_2^o)$$

$$\frac{\partial y_1^o}{\partial y_j^h} = w_{1,j}^o y_1^o(1 - y_1^o) \qquad \frac{\partial y_2^o}{\partial y_i^h} = w_{2,j}^o y_2^o(1 - y_2^o)$$

$$\frac{\partial y_j^h}{\partial w_{j,i}^h} = \frac{x_i e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}{\left(1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}\right)^2} = x_i y_j^h\left(1 - y_j^h\right)$$

$$y_j^h = \frac{1}{1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}$$

i = input node index
j = hidden node index
k = output node index

MLP with multiple outputs

$$E = \frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2 = \frac{1}{2}\sum_{k=1}^{2}\left(y_k - \frac{1}{1+e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}\right)^2$$

$$\frac{\partial E}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial w_{j,i}^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial w_{j,i}^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial w_{j,i}^h}$$

$$\frac{\partial E}{\partial y_1^o} = -(y_1 - y_1^o) \qquad \frac{\partial E}{\partial y_2^o} = -(y_2 - y_2^o)$$

$$\frac{\partial y_1^o}{\partial y_j^h} = w_{1,j}^o y_1^o(1 - y_1^o) \qquad \frac{\partial y_2^o}{\partial y_j^h} = w_{2,j}^o y_2^o(1 - y_2^o)$$

$$\frac{\partial y_j^h}{\partial w_{j,i}^h} = x_i y_j^h(1 - y_j^h)$$

$$y_j^h = \frac{1}{1+e^{-(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2)}}$$

i = input node index
j = hidden node index
k = output node index

MLP with multiple outputs

$$E = \frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2 = \frac{1}{2}\sum_{k=1}^{2}\left(y_k - \frac{1}{1+e^{-\left(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h\right)}}\right)^2$$

$$\frac{\partial E}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial w_{j,i}^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial w_{j,i}^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial w_{j,i}^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial w_{j,i}^h}$$

$$= \left(-(y_1 - y_1^o)w_{1,j}^o y_1^o(1-y_1^o) + -(y_2 - y_2^o)w_{2,j}^o y_2^o(1-y_2^o)\right)x_i y_j^h(1-y_j^h)$$

$$= \left(-\sum_{k=1}^{2}(y_k - y_k^o)w_{k,j}^o y_k^o(1-y_k^o)\right)x_i y_j^h(1-y_j^h)$$

$$\frac{\partial E}{\partial y_1^o} = -(y_1 - y_1^o) \qquad \frac{\partial E}{\partial y_2^o} = -(y_2 - y_2^o)$$

$$\frac{\partial y_1^o}{\partial y_j^h} = w_{1,j}^o y_1^o(1-y_1^o) \qquad \frac{\partial y_2^o}{\partial y_j^h} = w_{2,j}^o y_2^o(1-y_2^o)$$

$$\frac{\partial y_j^h}{\partial w_{j,i}^h} = x_i y_j^h(1-y_j^h)$$

$$y_j^h = \frac{1}{1+e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}$$

i = input node index
j = hidden node index
k = output node index

MLP with multiple outputs

$$E = \frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2 = \frac{1}{2}\sum_{k=1}^{2}\left(y_k - \frac{1}{1 + e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}\right)^2$$

$$\frac{\partial E}{\partial b_j^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial b_j^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial b_j^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial b_j^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial b_j^h}$$

$$= \left(-(y_1 - y_1^o)w_{1,j}^o y_1^o(1 - y_1^o) + -(y_2 - y_2^o)w_{2,j}^o y_2^o(1 - y_2^o)\right)\frac{\partial y_j^h}{\partial b_j^h}$$

$$= \left(-\sum_{k=1}^{2}(y_k - y_k^o)w_{k,j}^o y_k^o(1 - y_k^o)\right)\frac{\partial y_j^h}{\partial b_j^h}$$

$$\frac{\partial y_j^h}{\partial b_j^h} = \frac{1e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}{\left(1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}\right)^2} = 1y_j^h\left(1 - y_j^h\right)$$

$$y_j^h = \frac{1}{1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}$$

i = input node index
j = hidden node index
k = output node index

MLP with multiple outputs

$$E = \frac{1}{2}\sum_{k=1}^{2}(y_k - y_k^o)^2 = \frac{1}{2}\sum_{k=1}^{2}\left(y_k - \frac{1}{1 + e^{-(b_k^o + w_{k,1}^o y_1^h + w_{k,2}^o y_2^h + w_{k,3}^o y_3^h)}}\right)^2$$

$$\frac{\partial E}{\partial b_j^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial b_j^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial b_j^h} = \frac{\partial E}{\partial y_1^o}\frac{\partial y_1^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial b_j^h} + \frac{\partial E}{\partial y_2^o}\frac{\partial y_2^o}{\partial y_j^h}\frac{\partial y_j^h}{\partial b_j^h}$$

$$= \left(-(y_1 - y_1^o)w_{1,j}^o y_1^o(1 - y_1^o) + -(y_2 - y_2^o)w_{2,j}^o y_2^o(1 - y_2^o)\right) y_j^h(1 - y_j^h)$$

$$= \left(-\sum_{k=1}^{2}(y_k - y_k^o)w_{k,j}^o y_k^o(1 - y_k^o)\right) y_j^h(1 - y_j^h)$$

$$\frac{\partial y_j^h}{\partial b_j^h} = \frac{1e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}{\left(1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}\right)^2} = 1y_j^h(1 - y_j^h)$$

$$y_j^h = \frac{1}{1 + e^{-\left(b_j^h + w_{j,1}^h x_1 + w_{j,2}^h x_2\right)}}$$

i = input node index
j = hidden node index
k = output node index

MLP with multiple outputs

$$\frac{\partial E}{\partial w_{j,k}^h} = \left(-\sum_{k=1}^{2} (y_k - y_k^o) w_{k,j}^o y_k^o (1 - y_k^o) \right) x_i y_j^h (1 - y_j^h)$$

How to change the hidden weights (*update rule*):

$$w_{j,k}^h = w_{j,k}^h - \eta \frac{\partial E}{\partial w_{j,k}^h} = w_{j,k}^h + \eta \left(\sum_{k=1}^{2} (y_k - y_k^o) w_{k,j}^o y_k^o (1 - y_k^o) \right) x_i y_j^h (1 - y_j^h)$$

- `lrate` is η
- `tgt` is y
- `out` is y^o
- `err` is $(y - y^o)$
- `hid[h_idx]` is y_j^h
- `wo[h_idx]` is w_j^o
- `bo` is b^o
- `i_idx` is input index i
- `h_idx` is input index j
- `o_idx` is input index k

```
err = tgt - out # vector-vector subtraction

err_prop[h_idx] = 0
for o_idx in range(NUM_OUT_NODES):
  delta_out[o_idx] = out[o_idx] * (1.0 - out[o_idx]) * err[o_idx]
  err_prop[h_idx] += (delta_out[o_idx] * Who[o_idx, h_idx])

delta_hid[h_idx] = hid[h_idx] * (1.0 - hid[h_idx]) * err_prop[h_idx]

wh[h_idx, i_idx] += lrate * inp[i_idx] * delta_hid[h_idx]
bh[h_idx] += lrate * delta_hid[h_idx]
```

MLP with multiple outputs

$$\frac{\partial E}{\partial b_j^h} = \left(- \sum_{k=1}^{2} (y_k - y_k^o) w_{k,j}^o y_k^o (1 - y_k^o) \right) y_j^h (1 - y_j^h)$$

<u>How to change the hidden bias weights (*update rule*):</u>

$$b_j^h = b_j^h - \eta \frac{\partial E}{\partial b_j^h} = b_j^h + \eta \left(\sum_{k=1}^{2} (y_k - y_k^o) w_{k,j}^o y_k^o (1 - y_k^o) \right) y_j^h (1 - y_j^h)$$

- lrate is η
- tgt is y
- out is y^o
- err is $(y - y^o)$
- hid[h_idx] is y_j^h
- wo[h_idx] is w_j^o
- bo is b^o
- i_idx is input index i
- h_idx is input index j
- o_idx is input index k

```
err = tgt - out # vector-vector subtraction

err_prop[h_idx] = 0
for o_idx in range(NUM_OUT_NODES):
  delta_out[o_idx] = out[o_idx] * (1.0 - out[o_idx]) * err[o_idx]
  err_prop[h_idx] += (delta_out[o_idx] * Who[o_idx, h_idx])

delta_hid[h_idx] = hid[h_idx] * (1.0 - hid[h_idx]) * err_prop[h_idx]

wh[h_idx, i_idx] += lrate * inp[i_idx] * delta_hid[h_idx]
bh[h_idx] += lrate * delta_hid[h_idx]
```

Today: Basic Neural Nets, Part 2

- **Multilayer Perceptron model**
  1. MLP with multiple outputs (math)
  2. MLP with multiple outputs for regression (code)
  3. MLP with multiple outputs for classification (code)

Using a neural net to predict missing pixels

Original image

Image showing missing pixels
MSE = 1585.2

DIY code for missing pixels prediction

```python
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams["figure.dpi"] = 150
from ipcv_utils.utils import imshow as ipcv_imshow


#%% UTILITY FUNCTIONS AND MLP CODE
NUM_INP_NODES = 60
NUM_OUT_NODES = 4
BLOCK_SIZE = 8

def parse_img(img, inps, tgts):
  num_rows = img.shape[0]
  num_cols = img.shape[1]

  inp = np.zeros((NUM_INP_NODES))
  tgt = np.zeros((NUM_OUT_NODES))

  for r0 in range(0, num_rows, BLOCK_SIZE):
    for c0 in range(0, num_cols, BLOCK_SIZE):
      pos1 = 0
      pos2 = 0
      for r in range(BLOCK_SIZE):
        for c in range(BLOCK_SIZE):
          if (r >= 3 and r <= 4 and c >= 3 and c <= 4):
            tgt[pos1] = img[r0 + r, c0 + c]
            pos1 += 1
          else:
            inp[pos2] = img[r0 + r, c0 + c]
            pos2 += 1

      inps.append(inp.copy() / 255.0)
      tgts.append(tgt.copy() / 255.0)

def neural_response(inp, wh, bh, wo, bo):
  num_inp_nodes = wh.shape[1]
  num_hid_nodes = wh.shape[0]
  num_out_nodes = wo.shape[0]

  hid = np.zeros((num_hid_nodes))
  out = np.zeros((num_out_nodes))

  # compute the hidden layer activation
  for h_idx in range(num_hid_nodes):
    val = 0
    for i_idx in range(num_inp_nodes):
      val += inp[i_idx] * wh[h_idx, i_idx]
    val += bh[h_idx]

    hid[h_idx] = 1.0 / (1.0 + np.exp(-val))

  # compute the output layer activation
  for o_idx in range(num_out_nodes):
    val = 0
    for h_idx in range(num_hid_nodes):
      val += hid[h_idx] * wo[o_idx, h_idx]
    val += bo[o_idx]

    out[o_idx] = 1.0 / (1.0 + np.exp(-val))

  # return the output
  return (hid, out)


def update_network(hid, out, err, wh, bh, wo, bo):
  num_inp_nodes = wh.shape[1]
  num_hid_nodes = wh.shape[0]
  num_out_nodes = wo.shape[0]

  err_prop = np.zeros((num_hid_nodes))
  delta_hid = np.zeros((num_hid_nodes))
  delta_out = np.zeros((num_out_nodes))

  # compute the output layer deltas
  for o_idx in range(num_out_nodes):
    delta_out[o_idx] = \
      out[o_idx]*(1.0 - out[o_idx]) * err[o_idx]

  # compute the error to propagate back to the hidden layer
  for h_idx in range(num_hid_nodes):
    val = 0
    for o_idx in range(num_out_nodes):
      val += (delta_out[o_idx] * wo[o_idx, h_idx])
    err_prop[h_idx] = val

  # compute the hidden layer deltas
  for h_idx in range(num_hid_nodes):
    delta_hid[h_idx] = \
      hid[h_idx]*(1.0 - hid[h_idx]) * err_prop[h_idx]

  # adjust wo
  for h_idx in range(num_hid_nodes):
    for o_idx in range(num_out_nodes):
      wo[o_idx, h_idx] += \
        lrate * (hid[h_idx] * delta_out[o_idx])

  # adjust bo
  for o_idx in range(num_out_nodes):
    bo[o_idx] += lrate * delta_out[o_idx]

  # adjust wh
  for i_idx in range(num_inp_nodes):
    for h_idx in range(num_hid_nodes):
      wh[h_idx, i_idx] += \
        lrate * (inp[i_idx] * delta_hid[h_idx])

  # adjust bh
  for h_idx in range(num_hid_nodes):
    bh[h_idx] += lrate * delta_hid[h_idx]
```

DIY code for missing pixels prediction

```
#%% LOAD THE DATA
img_path = "buf_imgs/"
img_names = [
  "baby_small.dbl_buf",
  "beach_small.dbl_buf",
  "bike_small.dbl_buf",
  "church_small.dbl_buf",
  "horse_small.dbl_buf",
  "flowers_small.dbl_buf",
  "football_small.dbl_buf",
  "kids_small.dbl_buf",
  "pumpkins_small.dbl_buf",
  "sail_small.dbl_buf",
  "table_small.dbl_buf",
  "train_small.dbl_buf"
  ]

inps = []
tgts = []

for img_name in img_names:
  print(img_name)
  with open(img_path + img_name, "rb") as fid:
    w = np.fromfile(fid, np.uint32, 1)
    h = np.fromfile(fid, np.uint32, 1)
    img = np.fromfile(fid, np.float64)
    img = img.reshape(h[0], w[0])

    parse_img(img, inps, tgts)

print("Num inputs = ", len(inps))
print("Num features = ", len(inps[0]))
print("Num targets = ", len(tgts))
print("Num tgt pixels = ", len(tgts[0]))

X_trn = np.zeros((len(inps), NUM_INP_NODES))
y_trn = np.zeros((len(tgts), NUM_OUT_NODES))
for idx in range(len(inps)):
  X_trn[idx, :] = inps[idx]
  y_trn[idx] = tgts[idx]


#%% TRAINING
NUM_HID_NODES = 6
```

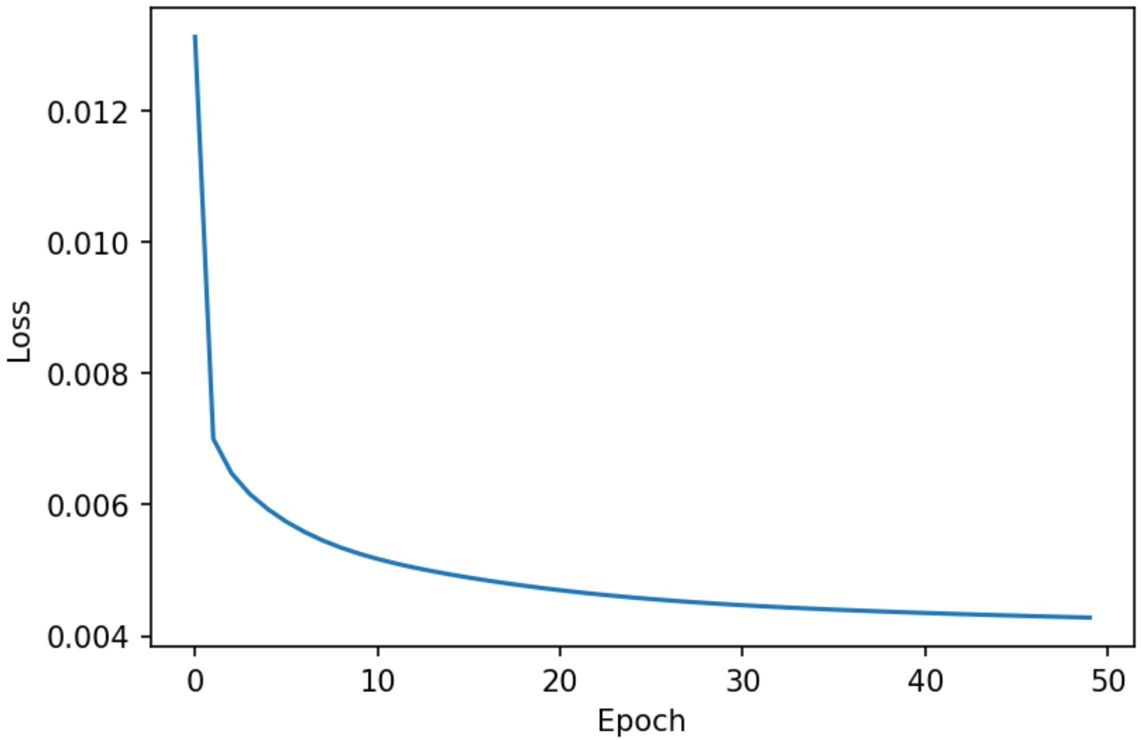
```
lrate = 0.1
num_epochs = 50

# randomize the (input to) hidden weights
wh = 2*np.random.rand(NUM_HID_NODES, NUM_INP_NODES) - 1
# randomize the (hidden to) ouput weights
wo = 2*np.random.rand(NUM_OUT_NODES, NUM_HID_NODES) - 1
# initialize the bias weights
bh = np.zeros((NUM_HID_NODES))
bo = np.zeros((NUM_OUT_NODES))

# for error tracking and propagation
err = np.zeros((NUM_OUT_NODES))
mses = np.zeros((num_epochs))

for epoch in range(num_epochs):
  print("Epoch", epoch+1, "of", num_epochs)

  # for each input/target pattern combo...
  epoch_mses = list()
  for inp, tgt in zip(X_trn, y_trn):

    # compute the MLP's response(s)
    hid, out = neural_response(inp, wh, bh, wo, bo)

    # compute the o/p node's MSE
    err = tgt - out
    epoch_mses.append((err**2).mean())

    # update the weights
    update_network(hid, out, err, wh, bh, wo, bo)

  # show/store the MSE for all patterns
  epoch_mse = np.mean(epoch_mses)
  print("Epoch MSE =", epoch_mse)
  mses[epoch] = epoch_mse

plt.plot(mses)
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
```

DIY code for missing pixels prediction

```
#%% SHOW THE RESULTS
# load a test image and put the missing and GT pixels into X_tst and y_tst
img_name = "balloon_small.dbl_buf"
print(img_name)
with open(img_path + img_name, "rb") as fid:
  w = np.fromfile(fid, np.uint32, 1)
  h = np.fromfile(fid, np.uint32, 1)
  img = np.fromfile(fid, np.float64)
  img = img.reshape(h[0], w[0])

inps = []
tgts = []
parse_img(img, inps, tgts)

X_tst = np.zeros((len(inps), NUM_INP_NODES))
y_tst = np.zeros((len(tgts), NUM_OUT_NODES))
for idx in range(len(inps)):
  X_tst[idx, :] = inps[idx]
  y_tst[idx] = tgts[idx]

# predict the missing pixels of the test image
y_tst_prd = np.zeros(y_tst.shape)
for idx in range(X_tst.shape[0]):
  _, y_tst_prd[idx] = neural_response(X_tst[idx], wh, bh, wo, bo)

# put the predicted pixels into the large image
num_rows = img.shape[0]
num_cols = img.shape[1]
rec_img = np.zeros((num_rows, num_cols))

blk_idx = 0
for r0 in range(0, num_rows, BLOCK_SIZE):
  for c0 in range(0, num_cols, BLOCK_SIZE):

    out = y_tst_prd[blk_idx]
    blk_idx += 1

    pos = 0
    for r in range(BLOCK_SIZE):
      for c in range(BLOCK_SIZE):
        if (r >= 3 and r <= 4 and c >= 3 and c <= 4):
          rec_img[r0 + r, c0 + c] = 255*out[pos]
          pos += 1
        else:
          rec_img[r0 + r, c0 + c] = img[r0 + r, c0 + c]

ipcv_imshow(rec_img, cmap="gray", vmin=0, vmax=255, zoom=2)
err = rec_img - img
print((err**2).mean())
```

```
Epoch 1 of 50
Epoch MSE = 0.013120898726167804
Epoch 2 of 50
Epoch MSE = 0.0070013703179305845
Epoch 3 of 50
Epoch MSE = 0.006476387636694087
Epoch 4 of 50
Epoch MSE = 0.006160339735151922
Epoch 5 of 50
Epoch MSE = 0.0059276033118941815
Epoch 6 of 50
Epoch MSE = 0.005739257691917149
Epoch 7 of 50
Epoch MSE = 0.00558308575888282
Epoch 8 of 50
Epoch MSE = 0.005453148396274893
Epoch 9 of 50
Epoch MSE = 0.005344146407257218
Epoch 10 of 50
Epoch MSE = 0.0052517196646226965
Epoch 11 of 50
Epoch MSE = 0.0051723064638762055
Epoch 12 of 50
Epoch MSE = 0.00510297872446081
Epoch 13 of 50
show more (open the raw output data in a text editor) ...

Epoch MSE = 0.004293714652989423
Epoch 49 of 50
Epoch MSE = 0.004286667437600425
Epoch 50 of 50
Epoch MSE = 0.004279825468131098
```

DIY code for missing pixels prediction



Predicted image
MSE = 21.5

Keras code for missing pixels prediction

```
import cv2 as cv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

#%% UTILITY FUNCTION
NUM_INP_NODES = 60
NUM_OUT_NODES = 4
BLOCK_SIZE = 8

def parse_img(img, inps, tgts):
  num_rows = img.shape[0]
  num_cols = img.shape[1]

  inp = np.zeros((NUM_INP_NODES))
  tgt = np.zeros((NUM_OUT_NODES))

  for r0 in range(0, num_rows, BLOCK_SIZE):
    for c0 in range(0, num_cols, BLOCK_SIZE):
      pos1 = 0
      pos2 = 0
      for r in range(BLOCK_SIZE):
        for c in range(BLOCK_SIZE):
          if (r >= 3 and r <= 4 and c >= 3 and c <= 4):
            tgt[pos1] = img[r0 + r, c0 + c]
            pos1 += 1
          else:
            inp[pos2] = img[r0 + r, c0 + c]
            pos2 += 1

      inps.append(inp.copy() / 255.0)
      tgts.append(tgt.copy() / 255.0)


#%% LOAD THE TRAINING DATA
img_names = [
  "baby_small.dbl_buf",
  "beach_small.dbl_buf",
  "bike_small.dbl_buf",
  "church_small.dbl_buf",
  "horse_small.dbl_buf",
  "flowers_small.dbl_buf",
  "football_small.dbl_buf",
  "kids_small.dbl_buf",
  "pumpkins_small.dbl_buf",
  "sail_small.dbl_buf",
  "table_small.dbl_buf",
  "train_small.dbl_buf"
  ]

inps = []
tgts = []

for img_name in img_names:
  print(img_name)
  with open(img_path + img_name, "rb") as fid:
    w = np.fromfile(fid, np.uint32, 1)
    h = np.fromfile(fid, np.uint32, 1)
    img = np.fromfile(fid, np.float64)
    img = img.reshape(h[0], w[0])

    parse_img(img, inps, tgts)

print("Num inputs = ", len(inps))
print("Num features = ", len(inps[0]))
print("Num targets = ", len(tgts))
print("Num tgt pixels = ", len(tgts[0]))

X_trn = np.zeros((len(inps), NUM_INP_NODES))
y_trn = np.zeros((len(tgts), NUM_OUT_NODES))
for idx in range(len(inps)):
  X_trn[idx, :] = inps[idx]
  y_trn[idx] = tgts[idx]
```

Keras code for missing pixels prediction

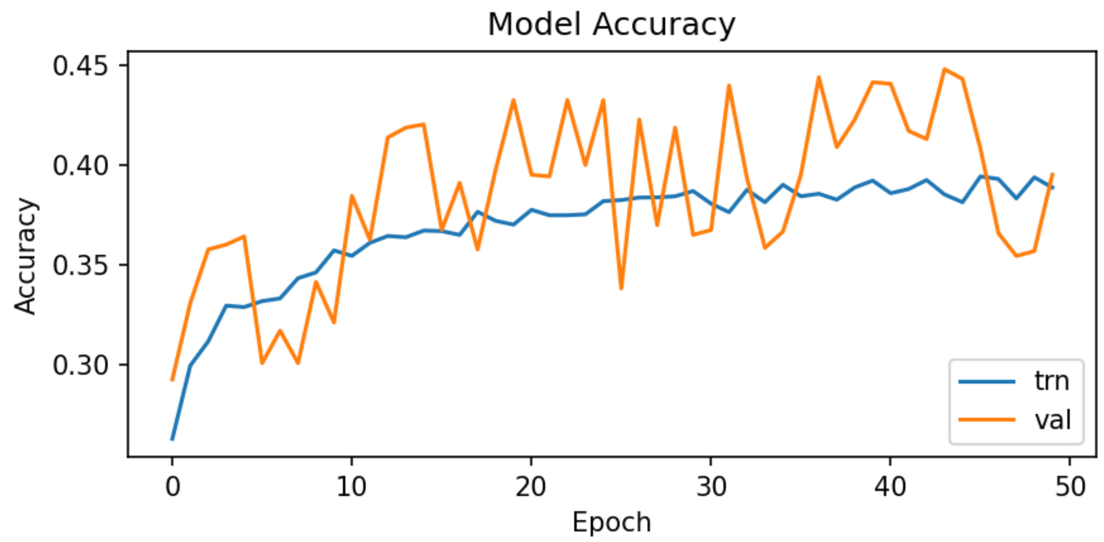
```python
#%% MODEL DEFINITION
NUM_HID_NODES = 6

model = Sequential()
hid_layer = Dense(NUM_HID_NODES,
  input_shape=(NUM_INP_NODES,), activation="relu")
out_layer = Dense(NUM_OUT_NODES,
  activation="sigmoid")
model.add(hid_layer)
model.add(out_layer)

model.compile(loss="mean_squared_error",
  optimizer="adam", metrics=["accuracy"])
model.summary()


#%% TRAINING
history = model.fit(X_trn, y_trn, epochs=50,
  batch_size=32, validation_split=0.1)

fig = plt.figure()
plt.subplot(2,1,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['trn', 'val'], loc='lower right')

plt.subplot(2,1,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['trn', 'val'], loc='upper right')
plt.tight_layout()


#%% SHOW THE RESULTS
# load a test image and put the missing and GT pixels into X_tst and y_tst
img_name = "balloon_small.dbl_buf"
print(img_name)
with open(img_path + img_name, "rb") as fid:
  w = np.fromfile(fid, np.uint32, 1)
  h = np.fromfile(fid, np.uint32, 1)
  img = np.fromfile(fid, np.float64)
  img = img.reshape(h[0], w[0])

inps = []
tgts = []
parse_img(img, inps, tgts)

X_tst = np.zeros((len(inps), NUM_INP_NODES))
y_tst = np.zeros((len(tgts), NUM_OUT_NODES))
for idx in range(len(inps)):
  X_tst[idx, :] = inps[idx]
  y_tst[idx] = tgts[idx]

# predict the missing pixels of the test image
y_tst_prd = model.predict(X_tst)

# put the predicted pixels into the large image
num_rows = img.shape[0]
num_cols = img.shape[1]
rec_img = np.zeros((num_rows, num_cols))

blk_idx = 0
for r0 in range(0, num_rows, BLOCK_SIZE):
  for c0 in range(0, num_cols, BLOCK_SIZE):

    out = y_tst_prd[blk_idx]
    blk_idx += 1

    pos = 0
    for r in range(BLOCK_SIZE):
      for c in range(BLOCK_SIZE):
        if (r >= 3 and r <= 4 and c >= 3 and c <= 4):
          rec_img[r0 + r, c0 + c] = 255*out[pos]
          pos += 1
        else:
          rec_img[r0 + r, c0 + c] = img[r0 + r, c0 + c]

ipcv_imshow(rec_img, cmap="gray", vmin=0, vmax=255, zoom=2)
err = rec_img - img
print((err**2).mean())
```

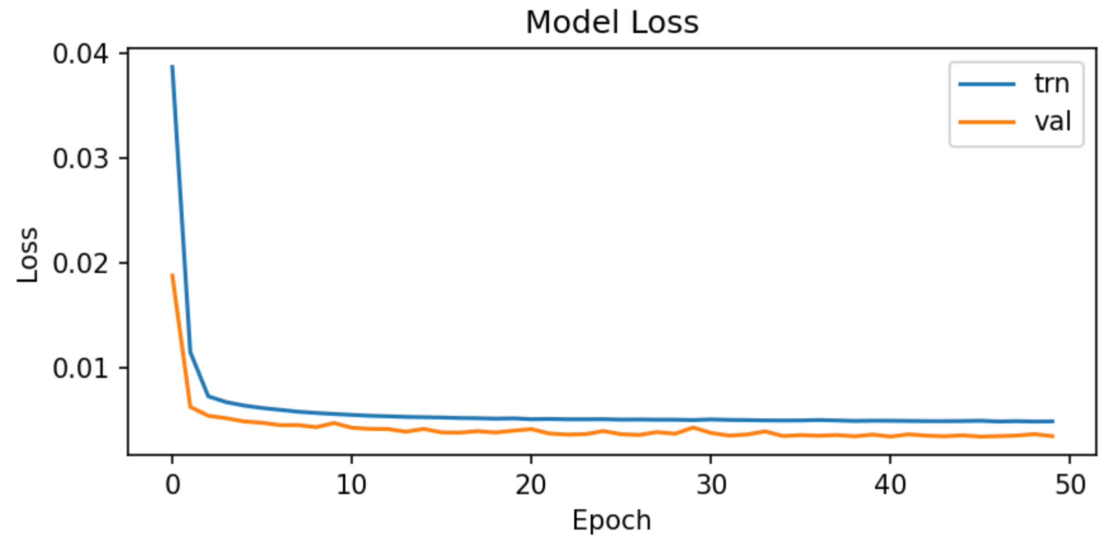
Keras code for missing pixels prediction

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 6)                 366
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 28
=================================================================
Total params: 394
Trainable params: 394
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/50
461/461 [==============================] - 3s 5ms/step - loss: 0.0387 -
accuracy: 0.2631 - val_loss: 0.0188 - val_accuracy: 0.2929
Epoch 2/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0115 -
accuracy: 0.2998 - val_loss: 0.0062 - val_accuracy: 0.3312
Epoch 3/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0072 -
accuracy: 0.3120 - val_loss: 0.0054 - val_accuracy: 0.3580
Epoch 4/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0067 -
accuracy: 0.3299 - val_loss: 0.0051 - val_accuracy: 0.3605
Epoch 5/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0063 -
accuracy: 0.3291 - val_loss: 0.0048 - val_accuracy: 0.3645
Epoch 6/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0061 -
accuracy: 0.3321 - val_loss: 0.0047 - val_accuracy: 0.3011
Epoch 7/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0059 -
accuracy: 0.3335 - val_loss: 0.0045 - val_accuracy: 0.3173
Epoch 8/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0057 -
accuracy: 0.3436 - val_loss: 0.0045 - val_accuracy: 0.3011
Epoch 9/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0056 -
accuracy: 0.3465 - val_loss: 0.0043 - val_accuracy: 0.3417
Epoch 10/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0055 -
accuracy: 0.3575 - val_loss: 0.0047 - val_accuracy: 0.3214
Epoch 11/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0054 -
accuracy: 0.3548 - val_loss: 0.0042 - val_accuracy: 0.3849
Epoch 12/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0054 -
accuracy: 0.3613 - val_loss: 0.0041 - val_accuracy: 0.3629
Epoch 13/50
show more (open the raw output data in a text editor) ...

461/461 [==============================] - 2s 4ms/step - loss: 0.0048 -
accuracy: 0.3836 - val_loss: 0.0035 - val_accuracy: 0.3548
Epoch 49/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0048 -
accuracy: 0.3942 - val_loss: 0.0036 - val_accuracy: 0.3572
Epoch 50/50
461/461 [==============================] - 2s 4ms/step - loss: 0.0048 -
accuracy: 0.3891 - val_loss: 0.0034 - val_accuracy: 0.3954
```

DIY code for missing pixels prediction

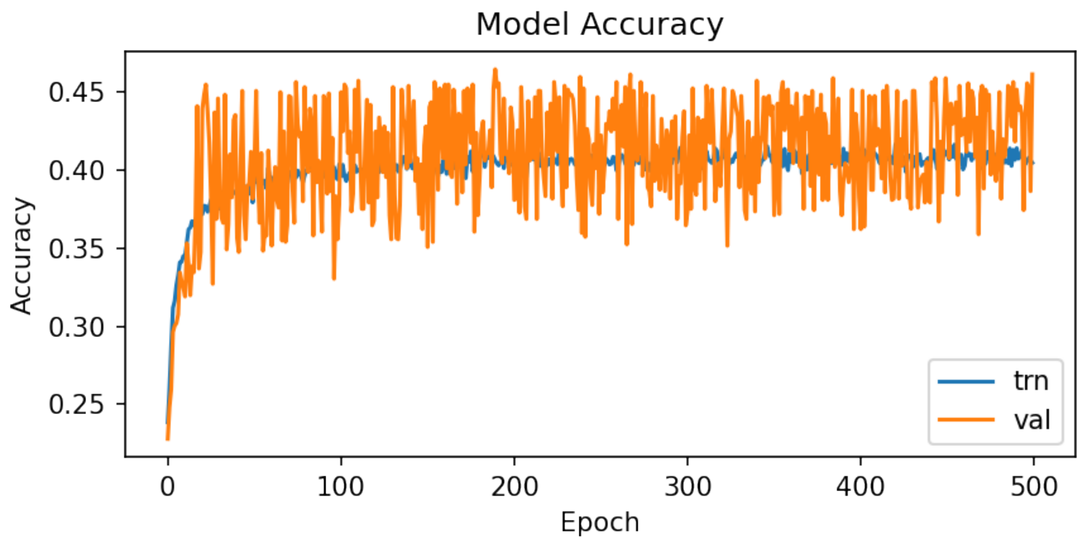


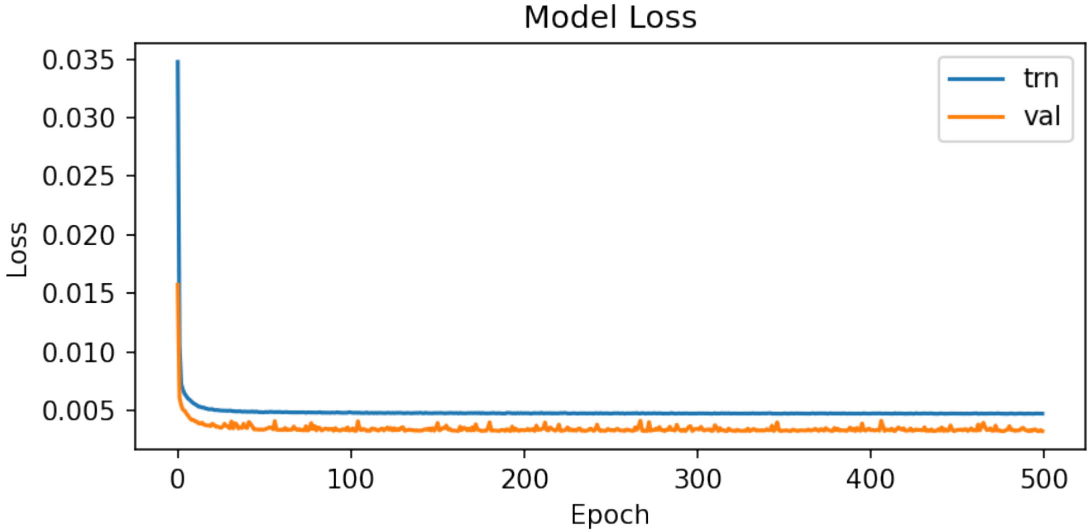


Predicted image
MSE = 23.0

DIY code for missing pixels prediction

500 epochs of training doesn't do any better!

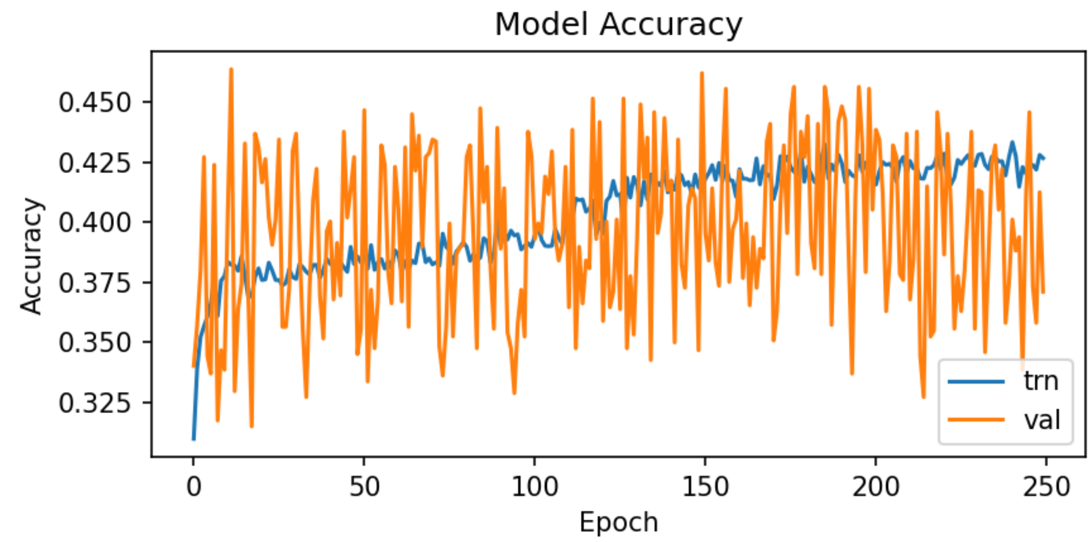


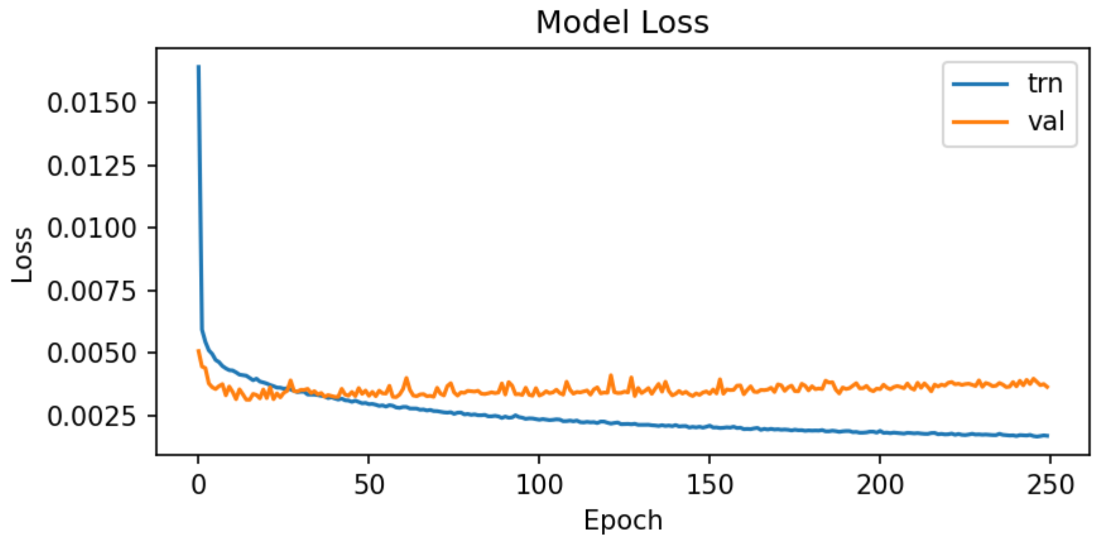


Predicted image
MSE = 23.8

DIY code for missing pixels prediction

Using two hidden layers, each with 60 nodes → OK, but still overfitting (need more training data)





Predicted image

MSE = 22.7

Today: Basic Neural Nets, Part 2

- **Multilayer Perceptron model**
  1. MLP with multiple outputs (math)
  2. MLP with multiple outputs for regression (code)
  3. MLP with multiple outputs for classification (code)

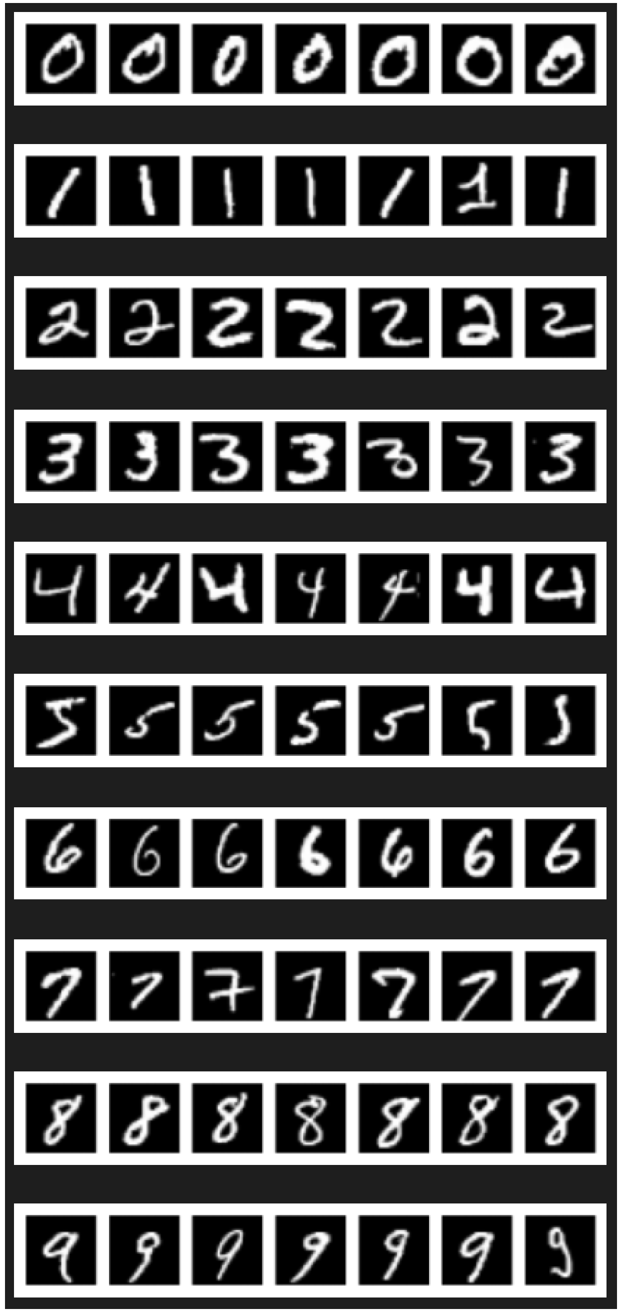
Keras code for MINST digits classification

```
import os
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import seaborn as sns

import numpy as np
import matplotlib.pyplot as plt

import keras
from tensorflow.keras.models import load_model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.callbacks import ModelCheckpoint


#%% LOAD THE DATA
(X_trn, y_trn), (X_tst, y_tst) = keras.datasets.mnist.load_data()

for cat_idx in range(10):
  idxs = np.argwhere(y_trn == cat_idx)

  plt.figure()
  for pos in range(7):
    plt.subplot(1, 7, pos+1)
    plt.imshow(X_trn[idxs[pos][0]], cmap="gray")
    plt.axis("off")
  plt.show()

X_trn = X_trn.astype('float32')
X_tst = X_tst.astype('float32')
X_trn /= 255.0
X_tst /= 255.0
X_trn = X_trn.reshape((X_trn.shape[0],
  X_trn.shape[1]*X_trn.shape[2]))
X_tst = X_tst.reshape((X_tst.shape[0],
  X_tst.shape[1]*X_tst.shape[2]))
```

```
print("Training set size (length, dims):", X_trn.shape)
print("Testing set size (length, dims):", X_tst.shape)


#%% CREATE OR LOAD THE MODEL
reuse_pre_model = True

# previous model exists?
model_fname = "saved_models/minst_digits.0.h5"
if reuse_pre_model and os.path.exists(model_fname):
  print("Reloading previous model: ", model_fname)
  model = load_model(model_fname)
else:
  print("Creating new model")
  model = Sequential()
  model.add(Dense(512, input_shape=(784,),
    activation="relu"))
  model.add(Dense(10, activation="softmax"))

  # compile the model
  model.compile(loss="sparse_categorical_crossentropy",
    optimizer="sgd", metrics=["accuracy"], )

# show the model info
model.summary()


#%% DO THE TRAINING
checkpoint = ModelCheckpoint(model_fname,
  monitor='loss', verbose=1, save_best_only=True)
callbacks_list = [checkpoint]

# fit the model
history = model.fit(X_trn, y_trn, epochs=20, batch_size=256,
  validation_split=0.15, callbacks=callbacks_list)
```

Keras code for MINST digits classification

```
#%% SHOW THE ACCURACY/LOSS HISTORY
fig = plt.figure(figsize=(6,6))
plt.subplot(2,1,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['trn', 'val'], loc='lower right')

plt.subplot(2,1,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['trn', 'val'], loc='upper right')
plt.tight_layout()


#%% SHOW THE CLASSIFICATION SUMMARY
# predict the class labels
y_tst_prd = np.argmax(model.predict(X_tst), axis=1)

print("")
print("Classification Report")
print(classification_report(y_tst, y_tst_prd))

plt.figure()
cm = confusion_matrix(y_tst, y_tst_prd)
sns.set(font_scale=0.75)
sns.heatmap(cm.T, square=True, annot=True, fmt='d',
  cbar=False, linewidths=0.5)
plt.xlabel('GT label')
plt.ylabel('Predicted label')
```

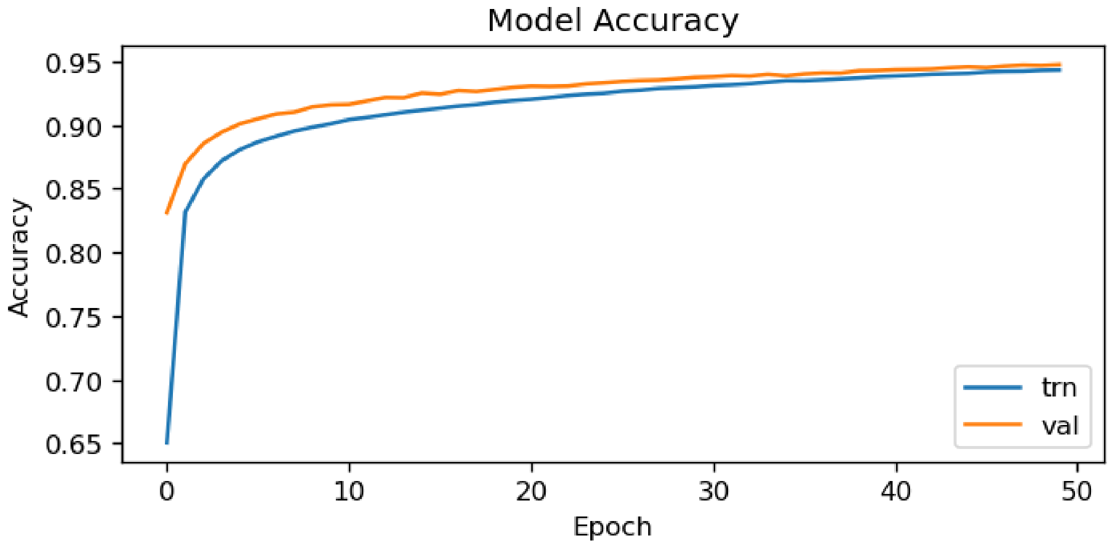


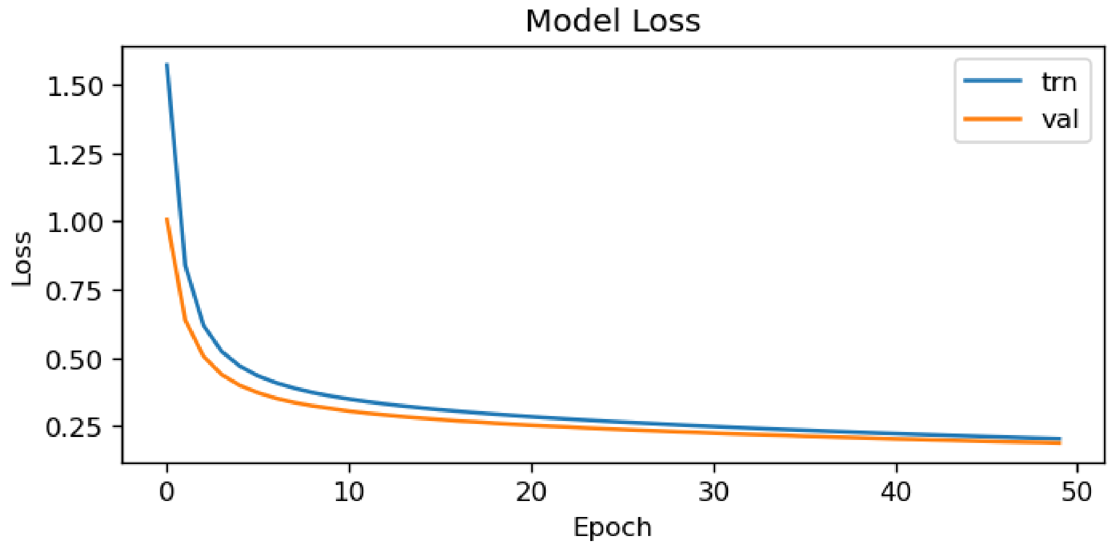
Keras code for missing pixels prediction

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/50
200/200 [==============================] - 1s 5ms/step - loss: 1.5718 -
accuracy: 0.6505 - val_loss: 1.0061 - val_accuracy: 0.8316

Epoch 00001: loss improved from inf to 1.57180, saving model to
saved_models\minst_digits.0.h5
Epoch 2/50
200/200 [==============================] - 1s 4ms/step - loss: 0.8386 -
accuracy: 0.8317 - val_loss: 0.6376 - val_accuracy: 0.8698

Epoch 00002: loss improved from 1.57180 to 0.83857, saving model to
saved_models\minst_digits.0.h5
Epoch 3/50
200/200 [==============================] - 1s 4ms/step - loss: 0.6186 -
accuracy: 0.8578 - val_loss: 0.5058 - val_accuracy: 0.8858

Epoch 00003: loss improved from 0.83857 to 0.61863, saving model to
saved_models\minst_digits.0.h5
Epoch 4/50
200/200 [==============================] - 1s 5ms/step - loss: 0.5235 -
accuracy: 0.8722 - val_loss: 0.4388 - val_accuracy: 0.8947

Epoch 00004: loss improved from 0.61863 to 0.52349, saving model to
saved_models\minst_digits.0.h5
Epoch 5/50
200/200 [==============================] - 1s 5ms/step - loss: 0.4694 -
accuracy: 0.8809 - val_loss: 0.3988 - val_accuracy: 0.9012

Epoch 00005: loss improved from 0.52349 to 0.46940, saving model to
saved_models\minst_digits.0.h5
Epoch 6/50
200/200 [==============================] - 1s 5ms/step - loss: 0.4338 -
accuracy: 0.8871 - val_loss: 0.3721 - val_accuracy: 0.9052

Epoch 00006: loss improved from 0.46940 to 0.43378, saving model to
saved_models\minst_digits.0.h5
Epoch 7/50
show more (open the raw output data in a text editor) ...


Epoch 00049: loss improved from 0.20738 to 0.20547, saving model to
saved_models\minst_digits.0.h5
Epoch 50/50
200/200 [==============================] - 1s 5ms/step - loss: 0.2035 -
accuracy: 0.9437 - val_loss: 0.1881 - val_accuracy: 0.9479

Epoch 00050: loss improved from 0.20547 to 0.20349, saving model to
saved_models\minst_digits.0.h5
```

Keras code for MINST digits classification






```
Classification Report
              precision    recall  f1-score   support

           0       0.95      0.98      0.97       980
           1       0.98      0.98      0.98      1135
           2       0.95      0.93      0.94      1032
           3       0.93      0.94      0.93      1010
           4       0.94      0.94      0.94       982
           5       0.95      0.91      0.93       892
           6       0.94      0.96      0.95       958
           7       0.95      0.94      0.94      1028
           8       0.93      0.93      0.93       974
           9       0.93      0.93      0.93      1009

    accuracy                           0.95     10000
   macro avg       0.94      0.94      0.94     10000
weighted avg       0.95      0.95      0.95     10000
```