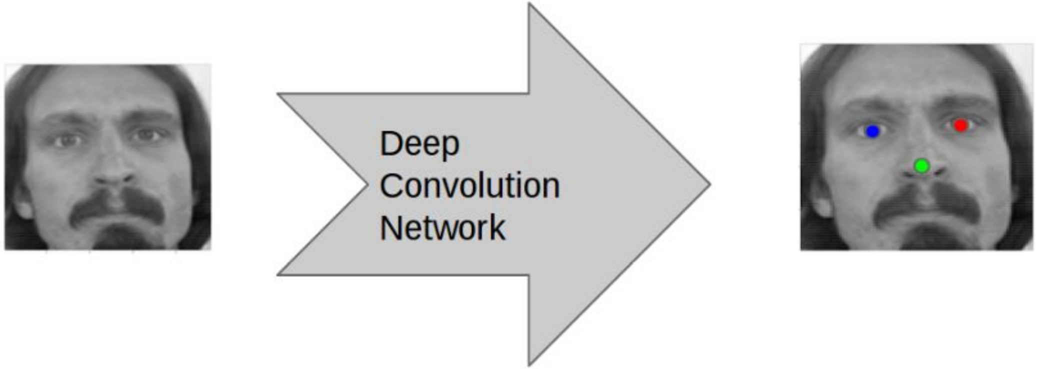
Hands-on
Deep Learning in Python



Imry Kissos

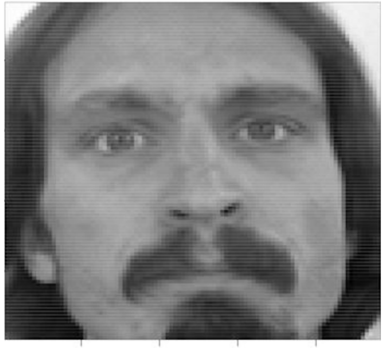
Deep Learning Meetup
TLV August 2015

Outline

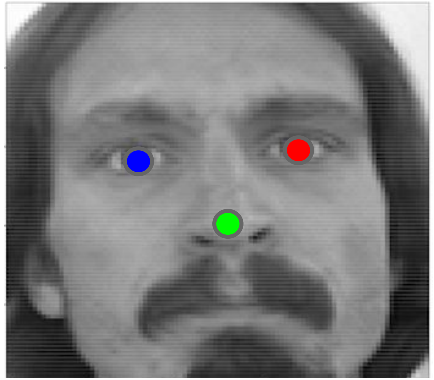
- Problem Definition
- Training a DNN

- Improving the DNN
- Open Source Packages
- Summary

Problem Definition



Deep
Convolution
Network



1 http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/

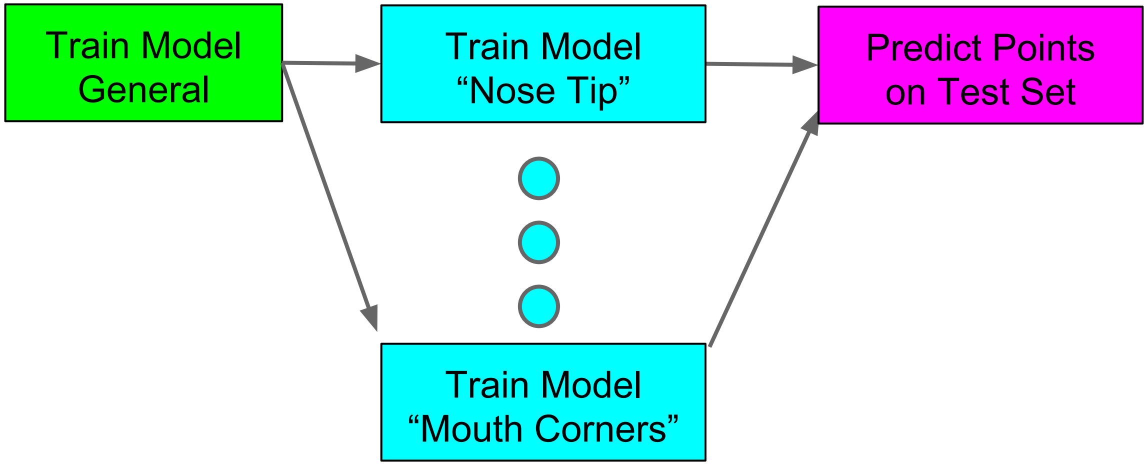
Tutorial

- Goal: Detect facial landmarks on (normal) face images
- Data set provided by Dr. Yoshua Bengio
- Tutorial code available:

https://github.com/dnouri/kfkd-tutorial/blob/master/kfkd.py

Flow

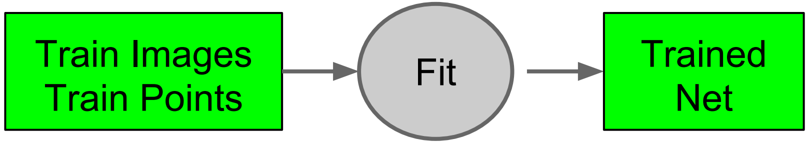
```python
if __name__ == '__main__':
    fit() '''train your first model'''
    fit_specialists(net.pickle)'''train specialists, intiliaze weights from your first model'''
    plot_learning_curves('net-specialists.pickle')
    predict('net-specialists.pickle')'''make predictions to submit to Kaggle'''
```



Flow

```
if __name__ == '__main__':
    fit() '''train your first model'''
    fit_specialists(net.pickle)'''train specialists, intiliaze weights from your first model'''
    plot_learning_curves('net-specialists.pickle')
    predict('net-specialists.pickle')'''make predictions to submit to Kaggle'''
```

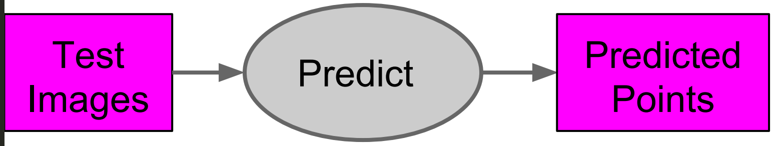
```
def fit():
    X, y = load2d()
    net.fit(X, y)
    with open('net.pickle', 'wb') as f:
        pickle.dump(net, f, -1)
```



Flow

```
if __name__ == '__main__':
    fit() '''train your first model'''
    fit_specialists(net.pickle)'''train specialists, intiliaze weights from your first model'''
    plot_learning_curves('net-specialists.pickle')
    predict('net-specialists.pickle')'''make predictions to submit to Kaggle'''
```

```
def predict(fname_specialists='net-specialists.pickle'):
    with open(fname_specialists, 'rb') as f:
        specialists = pickle.load(f)
    X = load2d(test=True)[0]
    y_pred = np.empty((X.shape[0], 0))
    for model in specialists.values():
        y_pred1 = model.predict(X)
        y_pred = np.hstack([y_pred, y_pred1])
```



Python Deep Learning Framework

High Level

nolearn - Wrapper to Lasagne

Lasagne - Theano extension for Deep Learning

Theano - Define, optimize, and mathematical expressions

Efficient Cuda GPU for DNN

Low Level

HW Supports: GPU & CPU
OS: Linux, OS X, Windows

Training a Deep Neural Network

1. Data Analysis
2. Architecture Engineering
3. Optimization
4. Training the DNN

Training a Deep Neural Network

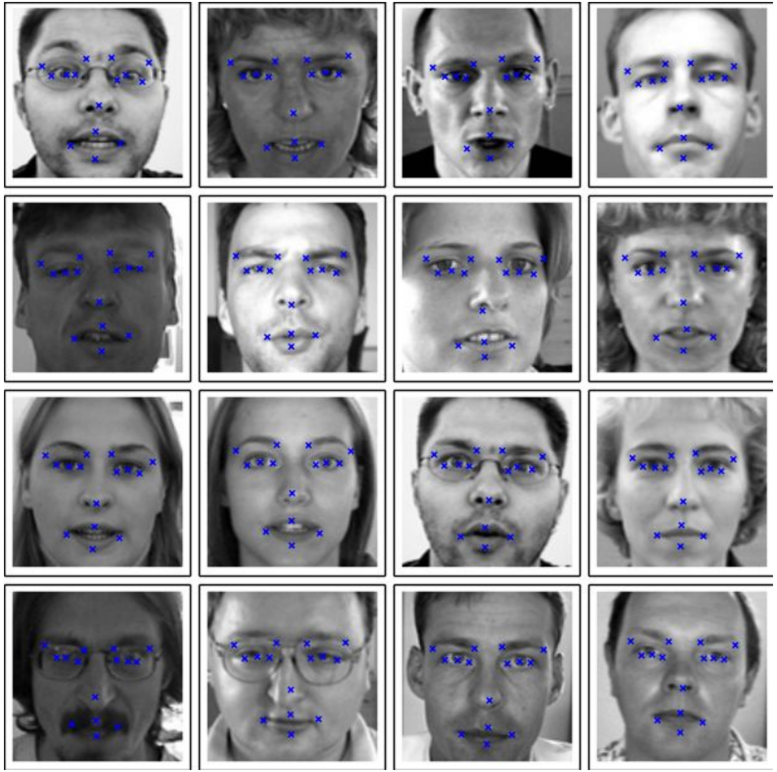
1. **Data Analysis**
    a. Exploration + Validation
    b. Pre-Processing
    c. Batch and Split
2. Architecture Engineering
3. Optimization
4. Training the DNN

Data Exploration + Validation

1

Data:

- 7K gray-scale images of detected faces
- 96x96 pixels per image
- 15 landmarks per image (?)



Data validation:

-

right_eye_center_x	7032
right_eye_center_y	7032
left_eye_inner_corner_x	2266
left_eye_inner_corner_y	2266

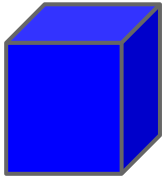
Pre-Processing

```python
def load(test=False, cols=None):
    fname = FTEST if test else FTRAIN
    df = read_csv(os.path.expanduser(fname))  # load pandas dataframe
    df['Image'] = df['Image'].apply(lambda im: np.fromstring(im, sep=' '))
    if cols:  # get a subset of columns
        df = df[list(cols) + ['Image']]
    print(df.count())  # prints the number of values for each column
    df = df.dropna()  # drop all rows that have missing values in them
    X = np.vstack(df['Image'].values) / 255.  # scale pixel values to [0, 1]
    X = X.astype(np.float32)
    if not test:  # only FTRAIN has any target columns
        y = df[df.columns[:-1]].values
        y = (y - 48) / 48  # scale target coordinates to [-1, 1]
        X, y = shuffle(X, y, random_state=42)  # shuffle train data
        y = y.astype(np.float32)
    else:
        y = None
    return X, y
```

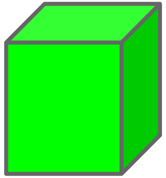
Data Normalization

Shuffle train data

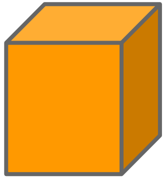
Batch



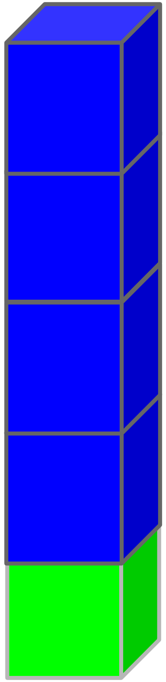
- train batch



- validation batch



- test batch



⇐One Epoch’s data

train/valid/test splits are constant

Train / Validation Split

```
regression=True,
```

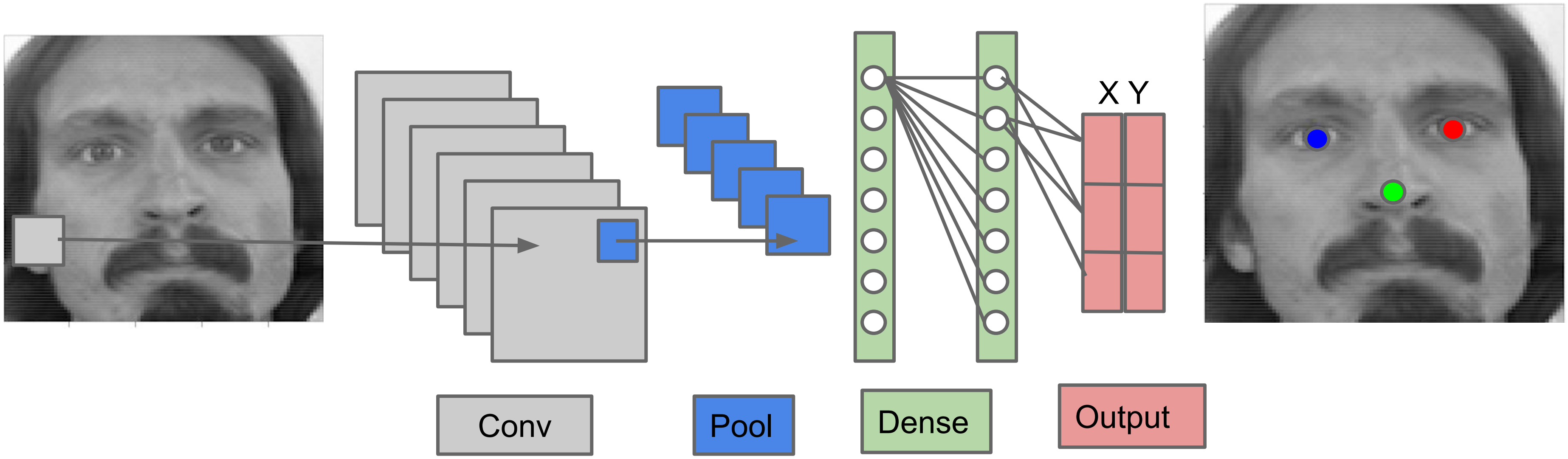
```
class TrainSplit(object):
    def __init__(self, eval_size):
        self.eval_size = eval_size

    def __call__(self, X, y, net):
        if self.eval_size:
            if net.regression:
                kf = KFold(y.shape[0], round(1. / self.eval_size))
            else:
                kf = StratifiedKFold(y, round(1. / self.eval_size))
```

Classification - Train/Validation preserve classes proportion

Training a Deep Neural Network

1. Data Analysis
2. **Architecture Engineering**
   a. **Layers Definition**
   b. **Layers Implementation**
3. Optimization
4. Training

Architecture



Layers Definition

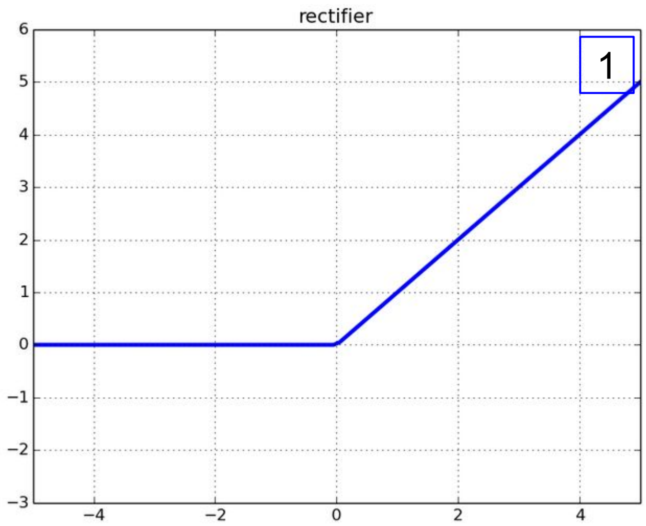
```
net = NeuralNet(
    layers=[
        ('input', layers.InputLayer),
        ('conv1', Conv2DLayer),
        ('pool1', MaxPool2DLayer),
        ('dropout1', layers.DropoutLayer),
        ('conv2', Conv2DLayer),
        ('pool2', MaxPool2DLayer),
        ('dropout2', layers.DropoutLayer),
        ('conv3', Conv2DLayer),
        ('pool3', MaxPool2DLayer),
        ('dropout3', layers.DropoutLayer),
        ('hidden4', layers.DenseLayer),
        ('dropout4', layers.DropoutLayer),
        ('hidden5', layers.DenseLayer),
        ('output', layers.DenseLayer),
        ],
```

```
input_shape=(None, 1, 96, 96),
conv1_num_filters=32, conv1_filter_size=(3, 3), pool1_pool_size=(2, 2),
dropout1_p=0.1,
conv2_num_filters=64, conv2_filter_size=(2, 2), pool2_pool_size=(2, 2),
dropout2_p=0.2,
conv3_num_filters=128, conv3_filter_size=(2, 2), pool3_pool_size=(2, 2),
dropout3_p=0.3,
hidden4_num_units=1000,
dropout4_p=0.5,
hidden5_num_units=1000,
output_num_units=30, output_nonlinearity=None,
```

Activation Function

ReLU

$$X = max(0, X)$$



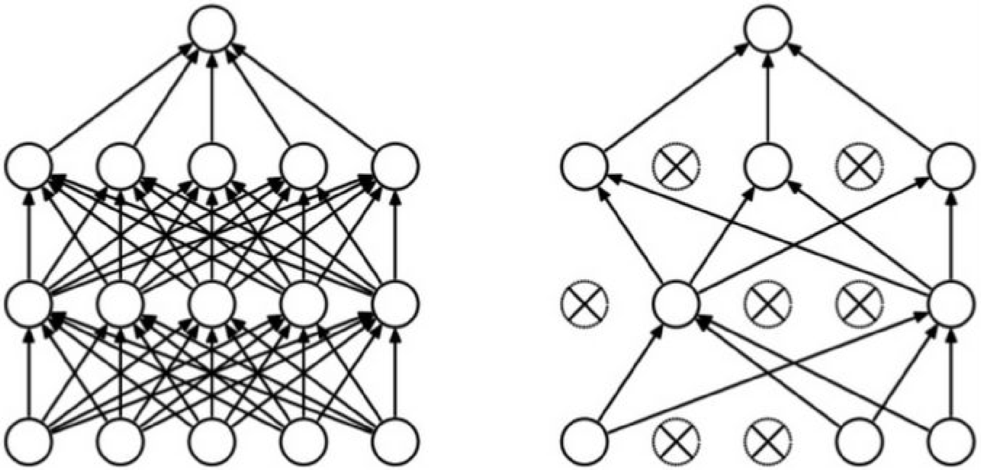

```
def rectify(x):
    """Rectify activation function :math:`\\varphi(x) = \\max(0, x)`

    # The following is faster than T.maximum(0, x),"""
    return 0.5 * (x + abs(x))
```

Dense Layer

$$Out = ReLU(I \cdot W + b)$$

```python
class DenseLayer(Layer):
    """
    lasagne.layers.DenseLayer(incoming, num_units,
    W=lasagne.init.GlorotUniform(), b=lasagne.init.Constant(0.),
    nonlinearity=lasagne.nonlinearities.rectify, **kwargs)

    A fully connected layer """
    def get_output_for(self, input, **kwargs):
        if input.ndim > 2:
            # if the input has more than two dimensions, flatten it into a
            # batch of feature vectors.
            input = input.flatten(2)
        activation = T.dot(input, self.W)
        if self.b is not None:
            activation = activation + self.b.dimshuffle('x', 0)
        return self.nonlinearity(activation)
```

Dropout



(a) Standard Neural Net

(b) After applying dropout.

Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

Dropout

```python
class DropoutLayer(Layer):
    '''During training you should set deterministic to false and during
    testing you should set deterministic to true.'''
```

```python
    def get_output_for(self, input, deterministic=False, **kwargs):
        if deterministic or self.p == 0:
            return input
        else:
            retain_prob = 1 - self.p
            if self.rescale:
                input /= retain_prob
            # use nonsymbolic shape for dropout mask if possible
            input_shape = self.input_shape
            if any(s is None for s in input_shape):
                input_shape = input.shape
            return input * self._srng.binomial(input_shape, p=retain_prob,
                                               dtype=theano.config.floatX)
```

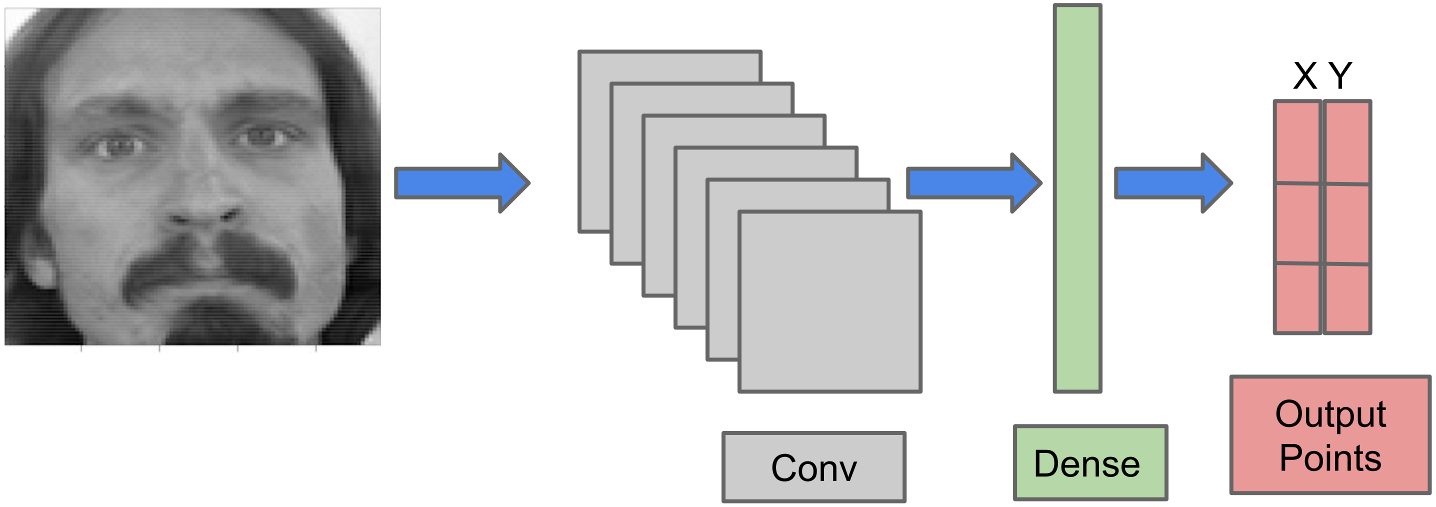
$$test: output = input$$

$$train: output = Input/prob \cdot RandMask$$

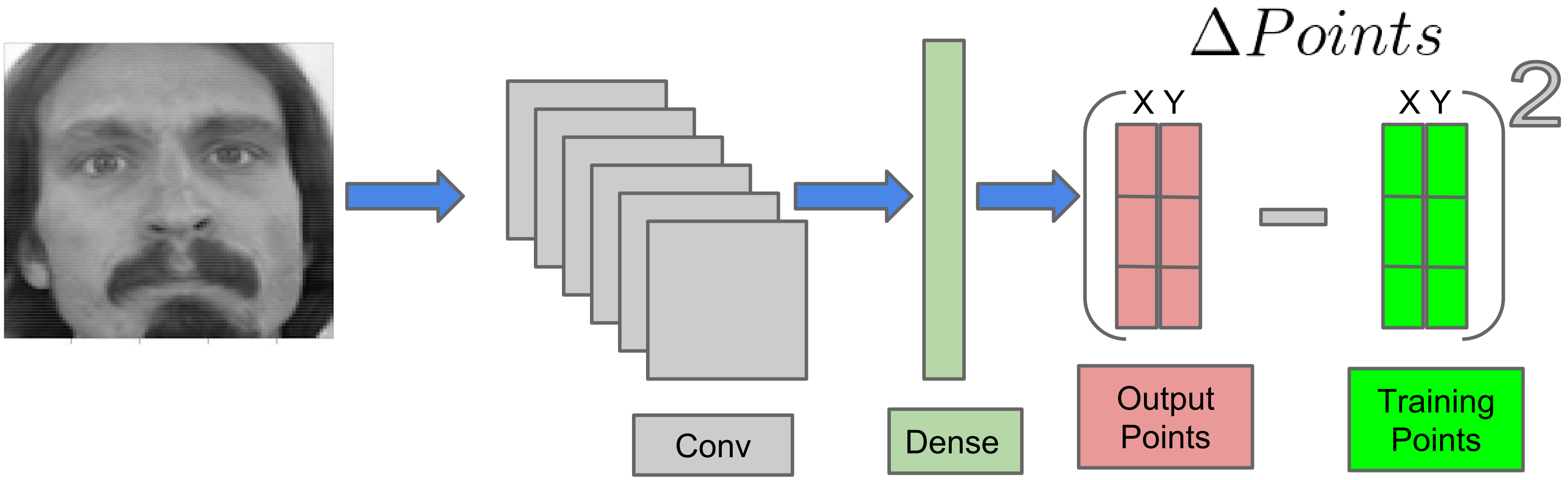
Training a Deep Neural Network

1. Data Analysis
2. Architecture Engineering
3. **Optimization**
   a. **Back Propagation**
   b. **Objective**
   c. **SGD**
   d. **Updates**
   e. **Convergence Tuning**
4. Training the DNN

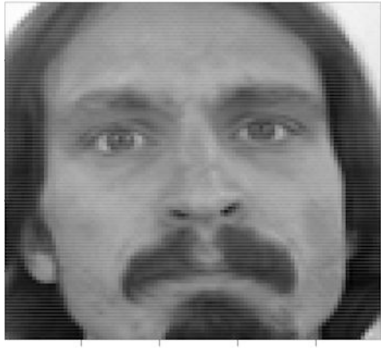
Back Propagation
Forward Path

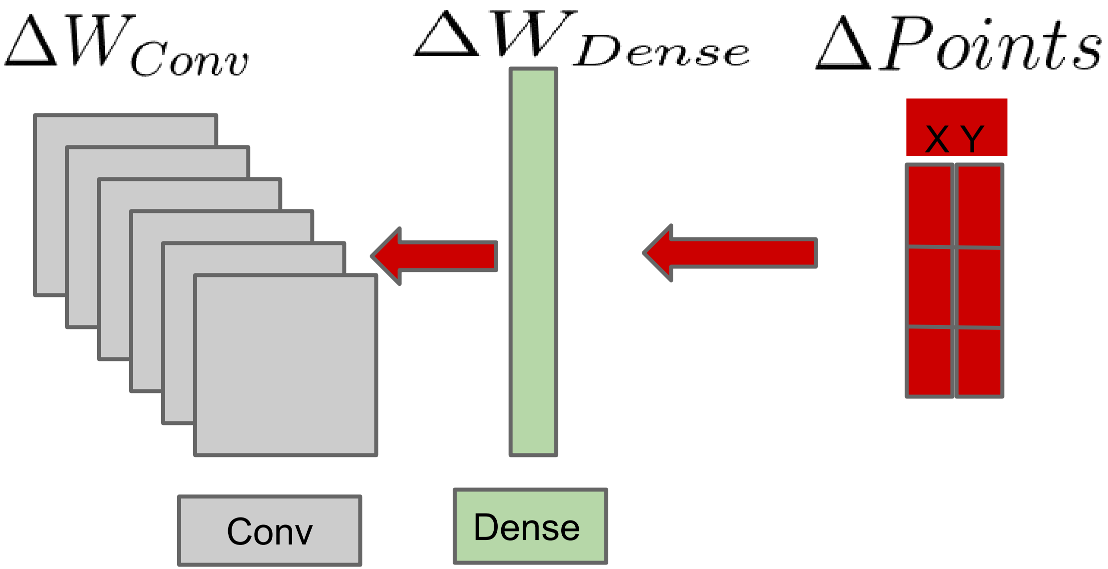


Back Propagation
Forward Path



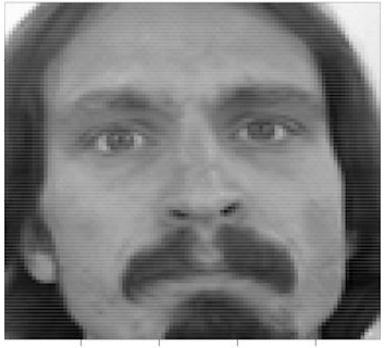
Back Propagation
Backward Path

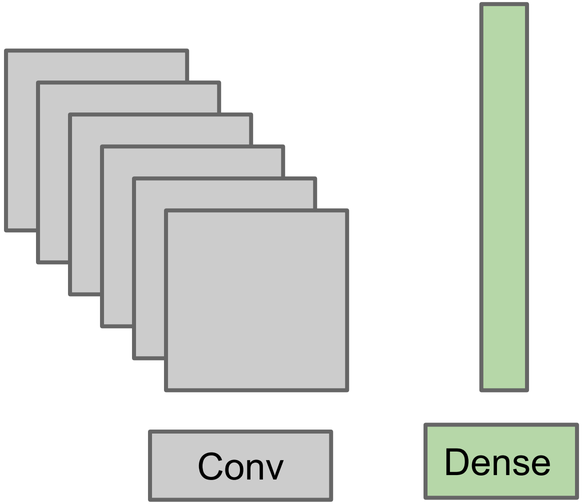




Back Propagation
Update

For All Layers: $$W^{updated} = W^{original} - \Delta W$$





Objective

```
regression=True,
```

```
if objective_loss_function is None:
    objective_loss_function = (
        squared_error if regression else categorical_crossentropy)
```

```
def squared_error(a, b):
    """Computes the element-wise squared difference between two tensors.
    .. math:: L = (p - t)^2
    This is the loss function of choice for many regression problems"""
    return (a - b)**2
```

S.G.D

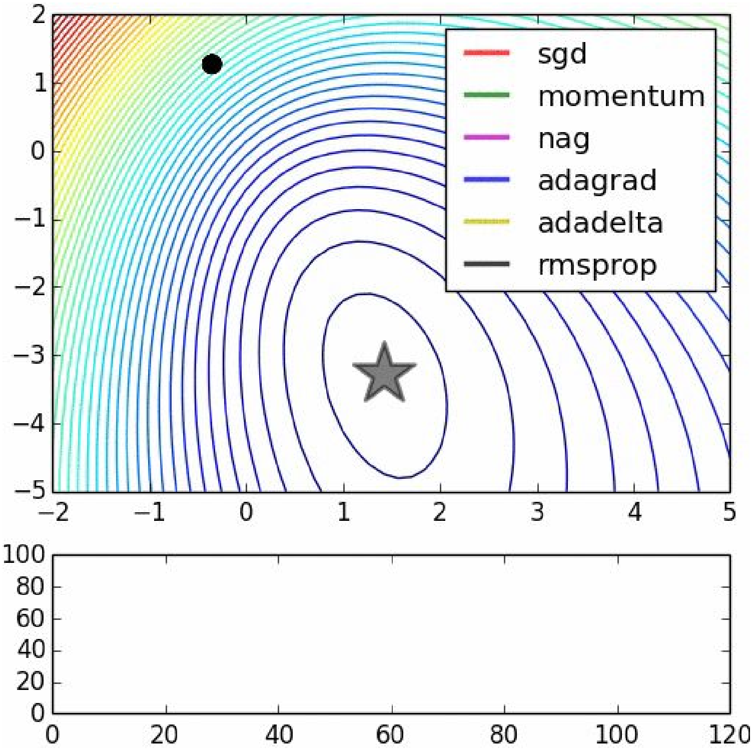
Updates the network after each **batch**

$$W^{updated} = W^{original} - \Delta W$$

```python
def sgd(loss_or_grads, params, learning_rate):
    """Stochastic Gradient Descent (SGD) updates
    * ``param := param - learning_rate * gradient``"""
    grads = get_or_compute_grads(loss_or_grads, params)
    updates = OrderedDict()

    for param, grad in zip(params, grads):
        updates[param] = param - learning_rate * grad

    return updates
```

Karpathy - “Babysitting”: weights/updates ~1e3

Optimization - Updates



Alec Radford

```
sgd
momentum
nesterov_momentum
adagrad
rmsprop
adadelta
adam
```

Adjusting Learning Rate & Momentum

```
on_epoch_finished=[
    AdjustVariable('update_learning_rate', start=0.03, stop=0.0001),
    AdjustVariable('update_momentum', start=0.9, stop=0.999),
    EarlyStopping(patience=200),
    ],
```

```
class AdjustVariable(object):
    def __init__(self, name, start=0.03, stop=0.001):
        self.name = name
        self.start, self.stop = start, stop
        self.ls = None
    def __call__(self, nn, train_history):
        if self.ls is None:
            self.ls = np.linspace(self.start, self.stop, nn.max_epochs)
        epoch = train_history[-1]['epoch']
        new_value = np.cast['float32'](self.ls[epoch - 1])
        getattr(nn, self.name).set_value(new_value)
```

Linear in epoch

Convergence Tuning

```
class EarlyStopping(object):
    def __init__(self, patience=100):
        self.patience = patience
        self.best_valid = np.inf
        self.best_valid_epoch = 0
        self.best_weights = None

    def __call__(self, nn, train_history):
        current_valid = train_history[-1]['valid_loss']
        current_epoch = train_history[-1]['epoch']
        if current_valid < self.best_valid:
            self.best_valid = current_valid
            self.best_valid_epoch = current_epoch
            self.best_weights = nn.get_all_params_values()
        elif self.best_valid_epoch + self.patience < current_epoch:
            print("Early stopping.")
            print("Best valid loss was {:.6f} at epoch {}.".format(
                self.best_valid, self.best_valid_epoch))
            nn.load_params_from(self.best_weights)
            raise StopIteration()
```

```
on_epoch_finished=[
    AdjustVariable('update_learning_rate', start=0.03, stop=0.0001),
    AdjustVariable('update_momentum', start=0.9, stop=0.999),
    EarlyStopping(patience=200),
],
```

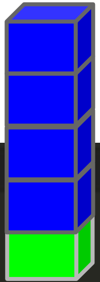
stops according to **validation** loss

returns best weights

Training a Deep Neural Network

1. Data Analysis
2. Architecture Engineering
3. Optimization
4. **Training the DNN**
   a. **Fit**
   b. **Fine Tune Pre-Trained**
   c. **Learning Curves**

Fit



```python
def fit():
    X, y = load2d()
    net.fit(X, y)
    with open('net.pickle', 'wb') as f:
        pickle.dump(net, f, -1)
```

```python
while epoch < self.max_epochs:
    epoch += 1
    valid_losses = []
    valid_accuracies = []
    custom_score = []
    t0 = time()
    for Xb, yb in self.batch_iterator_train(X_train, y_train):
        batch_train_loss = self.apply_batch_func(
            self.train_iter_, Xb, yb)
        train_losses.append(batch_train_loss)
    for Xb, yb in self.batch_iterator_test(X_valid, y_valid):
        batch_valid_loss, accuracy = self.apply_batch_func(
            self.eval_iter_, Xb, yb)
        valid_losses.append(batch_valid_loss)
        valid_accuracies.append(accuracy)
    avg_train_loss = np.mean(train_losses)
    avg_valid_loss = np.mean(valid_losses)
```

Loop over train batchs

Forward+BackProp

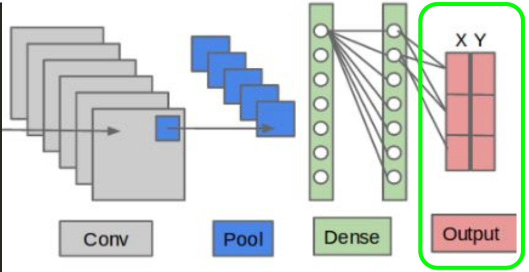
Loop over validation batchs

Forward

Fine Tune Pre-Trained

```
if __name__ == '__main__':
    fit() '''train your first model'''
    fit_specialists(net.pickle)'''train specialists, intiliaze weights from your first model'''
    plot_learning_curves('net-specialists.pickle')
    predict('net-specialists.pickle')'''make predictions to submit to Kaggle'''
```

```
def fit_specialists(fname_pretrain=None):
    with open(fname_pretrain, 'rb') as f:
        net_pretrain = pickle.load(f)
    specialists = OrderedDict()
    for setting in SPECIALIST_SETTINGS:
        cols = setting['columns']
        X, y = load2d(cols=cols)
        model = clone(net)
        model.output_num_units = y.shape[1]
        model.batch_iterator_train.flip_indices = setting['flip_indices']
        model.max_epochs = int(4e6 / y.shape[0])
        model.load_params_from(net_pretrain)
        print("Training model for columns {} for {} epochs".format(
            cols, model.max_epochs))
        model.fit(X, y)
        specialists[cols] = model
```



change output layer

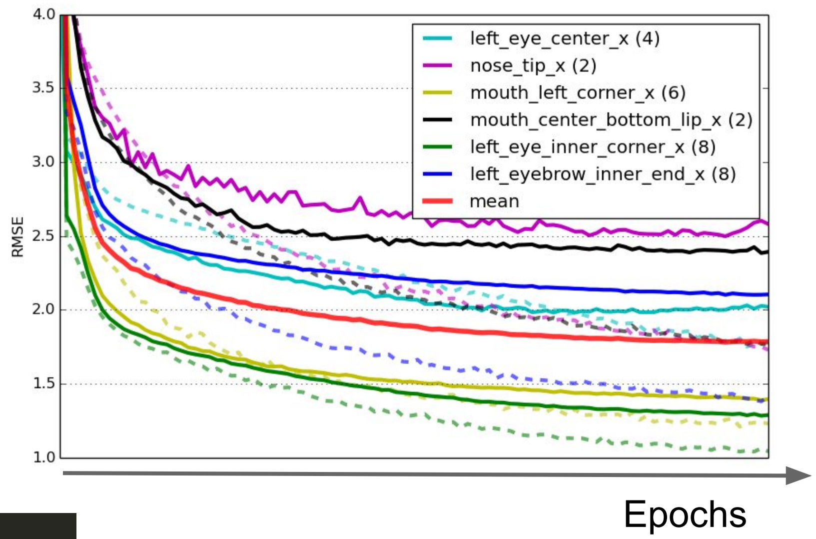
load pre-trained weight

fine tune specialist

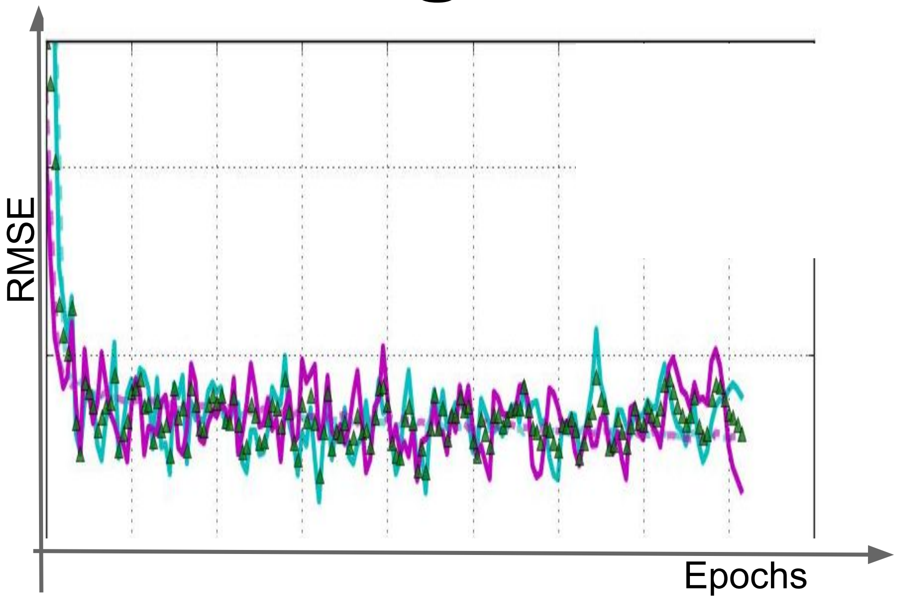
Learning Curves

Loop over 6 Nets:

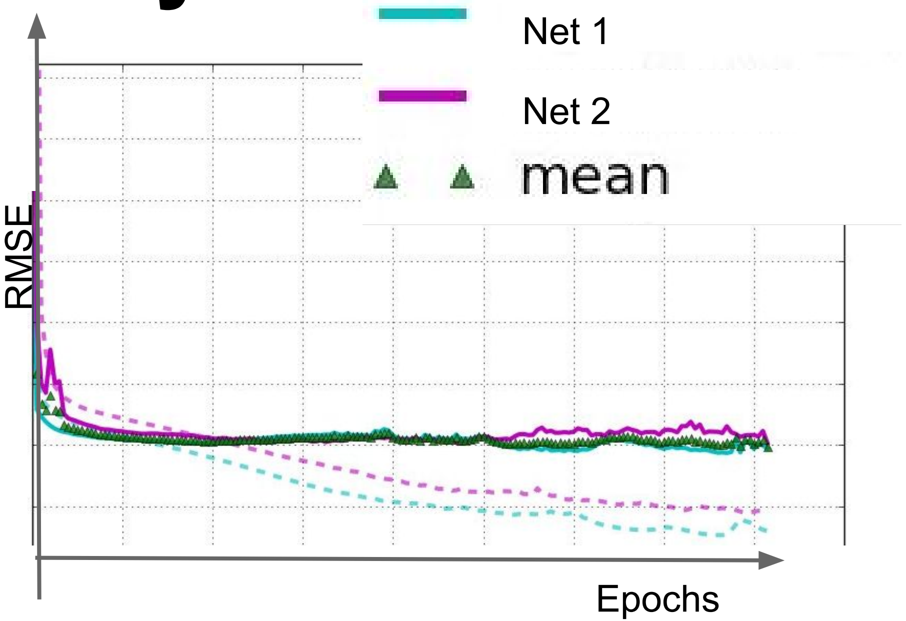
```
    ax.plot(valid_loss,
            label='{} ({})'.format(cg[0], len(cg)), linewidth=3)
    ax.plot(train_loss,
            linestyle='--', linewidth=3, alpha=0.6)
    ax.set_xticks([])
weights = np.array([m.output_num_units for m in models.values()],
                   dtype=float)
weights /= weights.sum()
mean_valid_loss = (
    np.vstack(valid_losses) * weights.reshape(-1, 1)).sum(axis=0)
ax.plot(mean_valid_loss, color='r', label='mean', linewidth=4, alpha=0.8)
ax.legend()
ax.set_ylim((1.0, 4.0))
ax.grid()
pyplot.ylabel("RMSE")
pyplot.show()
```



Learning Curves Analysis



Convergence
Jittering



Overfitting

Part 1 Summary

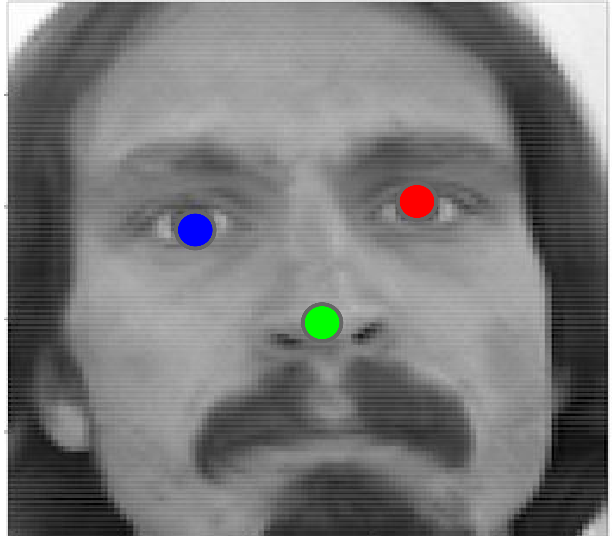
Training a DNN:

```
if __name__ == '__main__':
    fit() '''train your first model'''
    fit_specialists(net.pickle)'''train specialists, intiliaze weights from your first model'''
    plot_learning_curves('net-specialists.pickle')
    predict('net-specialists.pickle')'''make predictions to submit to Kaggle'''
```

Part 1 End

Break

- Improving the DNN
- Open Source Packages
- Summary



Part 2

Beyond Training

Outline

- Problem Definition
- Motivation
- Training a DNN

- **Improving the DNN**
- **Open Source Packages**
- **Summary**

Beyond Training

1. **Improving the DNN**
    a. **Analysis Capabilities**
    b. **Augmentation**
    c. **Forward - Backward Path**
    d. **Monitor Layers' Training**
2. Open Source Packages
3. Summary

Improving the DNN

Very tempting:

- >1M images
- >1M parameters
- Large gap: Theory ↔ Practice

⇒Brute force experiments?!

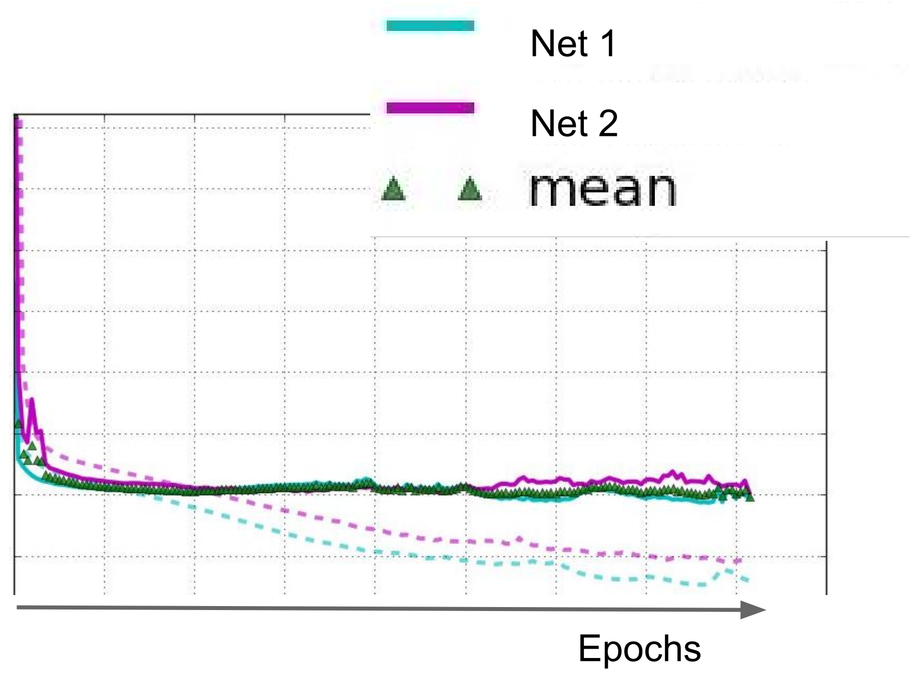
Analysis Capabilities

1. Theoretical explanation
    a. Eg. dropout and augmentation decrease overfit
2. Empirical claims about a phenomena
    a. Eg. normalization improves convergence
3. Numerical understanding
    a. Eg. exploding / vanishing updates

Reduce Overfitting

Solution:

Data Augmentation



Overfitting

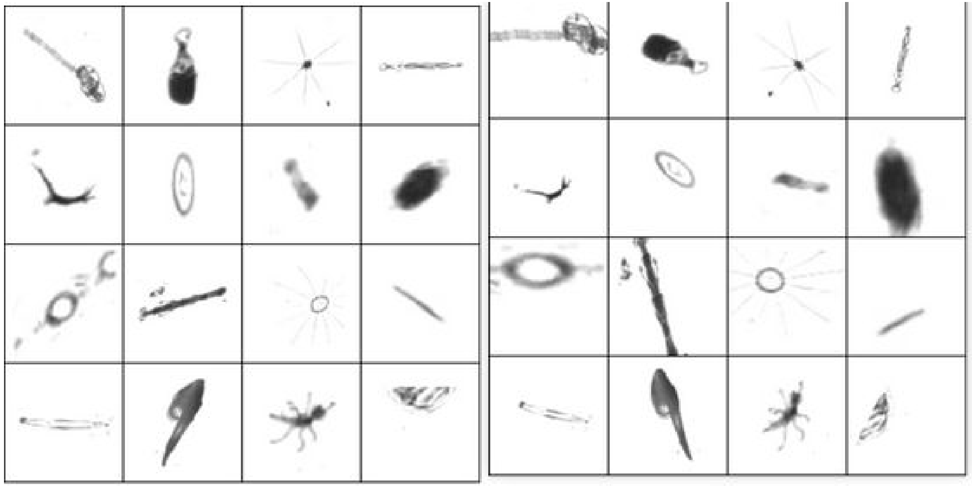
Data Augmentation

```
batch_iterator_train=FlipBatchIterator(batch_size=128),
```

Horizontal Flip **Perturbation**

1

Advanced Augmentation

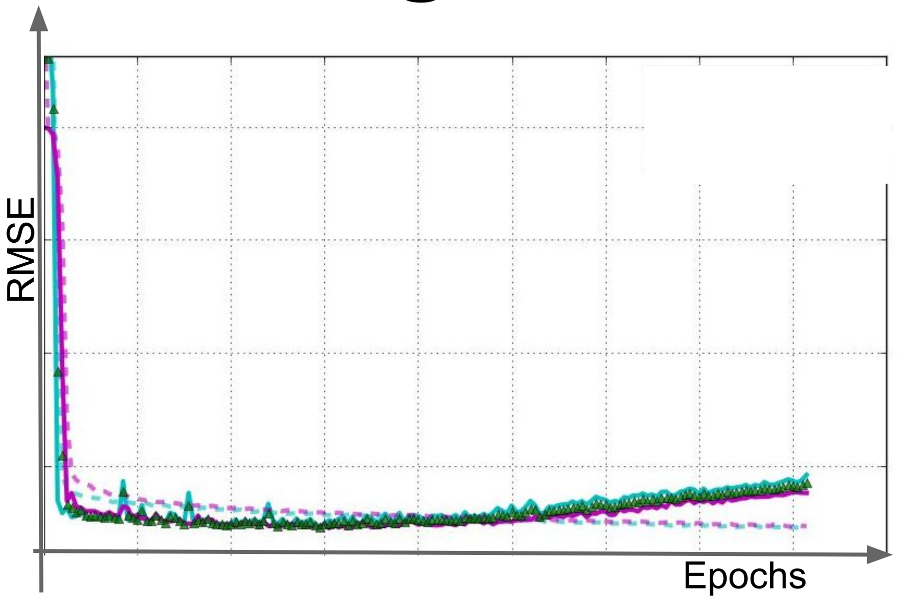


Pre-processed images (left) and augmented versions of the same images (right).

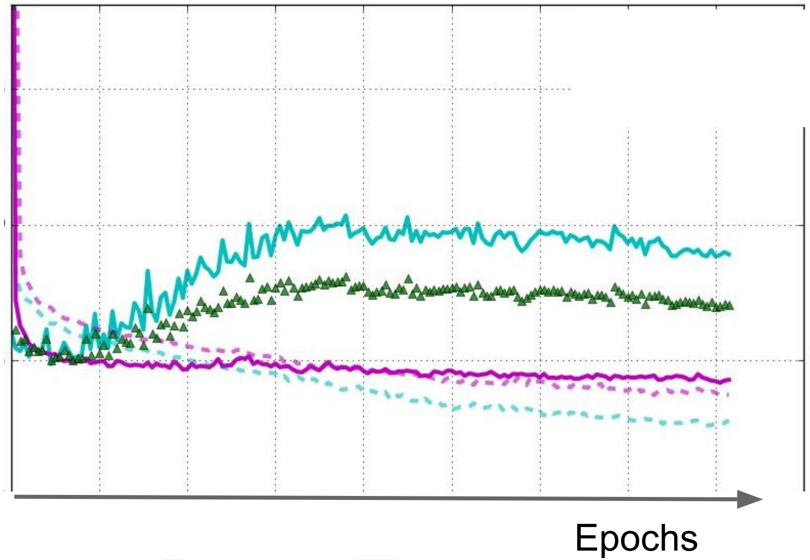
- **rotation**: random with angle between 0° and 360° (uniform)
- **translation**: random with shift between -10 and 10 pixels (uniform)
- **rescaling**: random with scale factor between 1/1.6 and 1.6 (log-uniform)
- **flipping**: yes or no (bernoulli)
- **shearing**: random with angle between -20° and 20° (uniform)
- **stretching**: random with stretch factor between 1/1.3 and 1.3 (log-uniform)

http://benanne.github.io/2015/03/17/plankton.html

Convergence Challenges



Normalization



Data Error

Need to monitor forward + backward path

Forward - Backward Path

Forward

```
def get_output_for(self, input, **kwargs):
```

Backward:

Gradient w.r.t parameters

```
def get_or_compute_grads(loss_or_grads, params):
    """
    Parameters
    ----------
    loss_or_grads : symbolic expression or list of expressions
        A scalar loss expression, or a list of gradient expressions
    params : list of shared variables
        The variables to return the gradients for
    """
```

$$\nabla_W F(W^{n-1}, b^{n-1})$$

Monitor Layers' Training

nolearn - visualize.py

```
def plot_conv_weights(layer, figsize=(6, 6)):
    """Plot the weights of a specific layer.
```

```
def plot_conv_activity(layer, x, figsize=(6, 8)):
    """Plot the acitivities of a specific layer.
```

Monitor Layers' Training

X. Glorot ,Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks:*

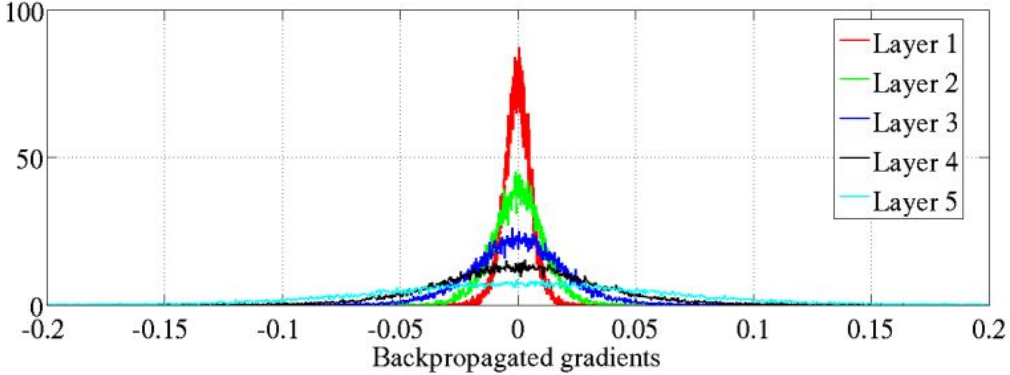
*"Monitoring **activation** and **gradients** across layers and training iterations is a powerful investigation tool"*

Easy to monitor in Theano Framework

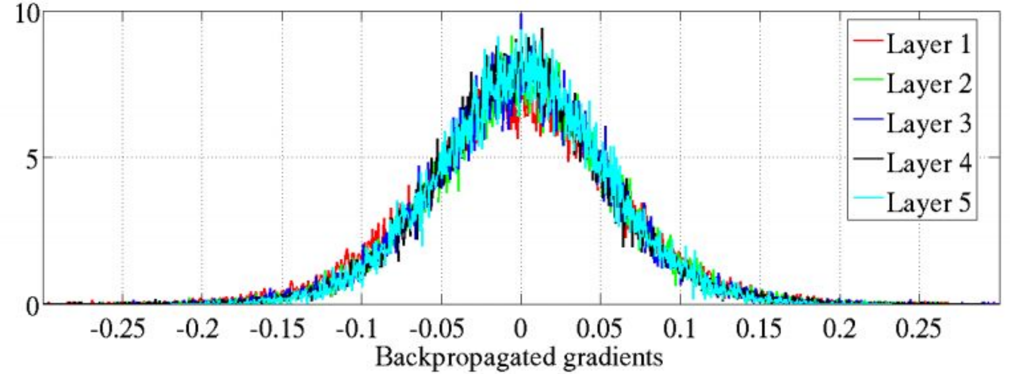
Weight Initialization matters (1)

Layer 1- Gradient are close to zero - vanishing gradients

$$W_j \sim U\left[-\frac{1}{\sqrt{n_j}}, \frac{1}{\sqrt{n_j}}\right]$$



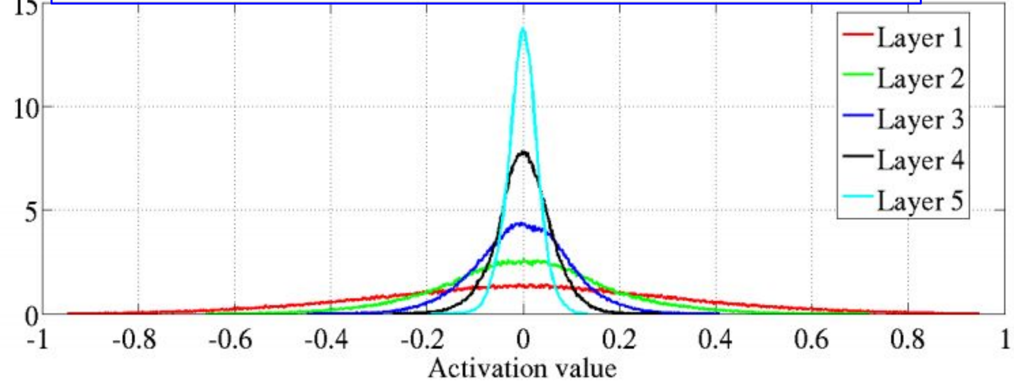
$$W_j \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$



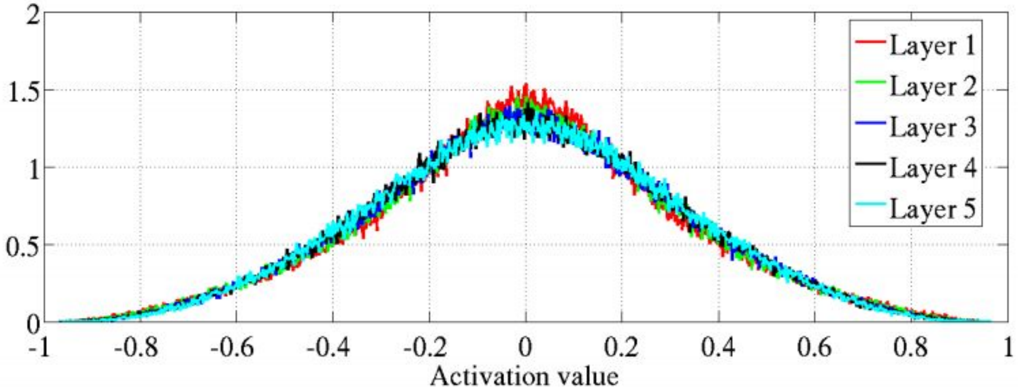
Weight Initialization matters (2)

$$W_j \sim U\left[-\frac{1}{\sqrt{n_j}}, \frac{1}{\sqrt{n_j}}\right]$$

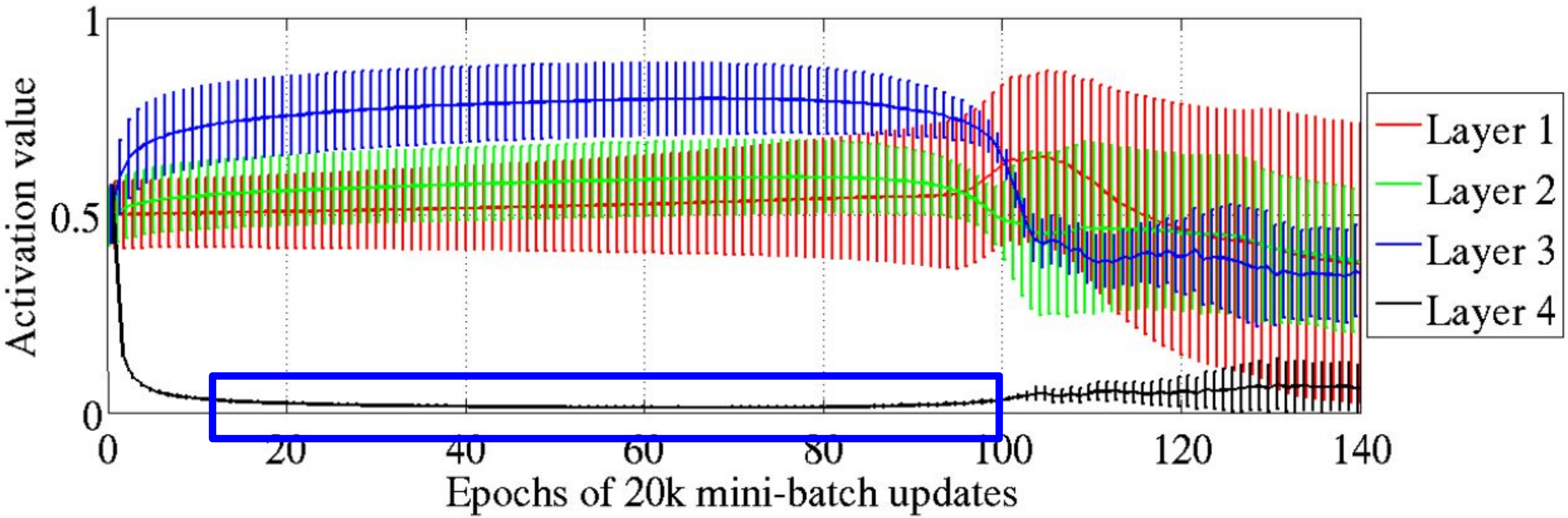
Network returns close to zero values for all inputs



$$W_j \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}\right]$$



Monitoring Activation

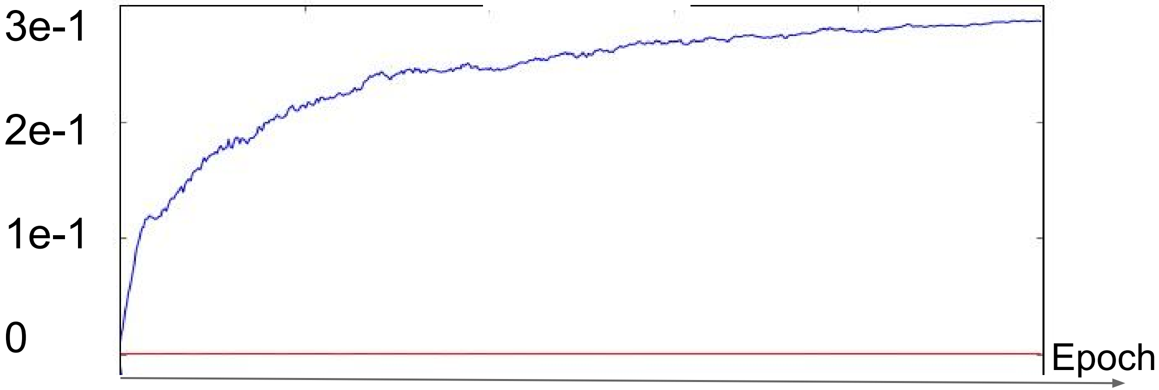


For most epochs the network returns close to zero output for all inputs

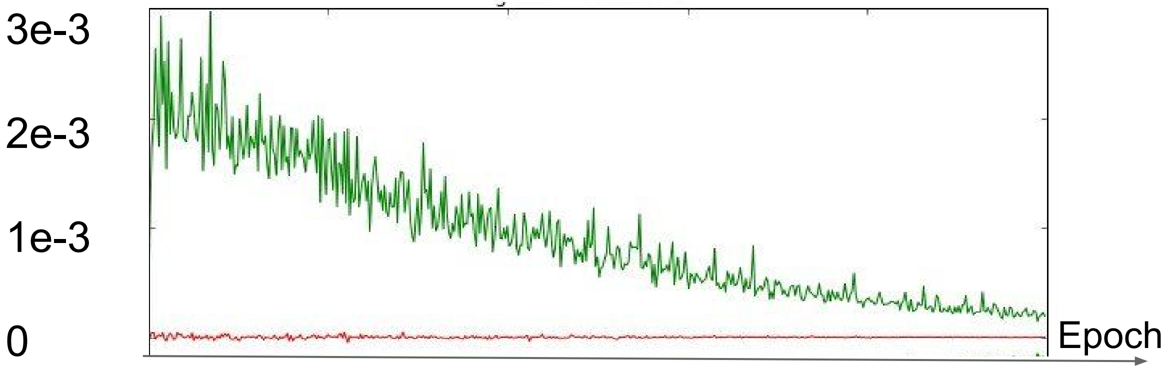
Objective plateaus sometimes can be explained by saturation

Monitoring weights/update ratio

Max of **Weights** of Conv1:



Max of **Updates** of Conv1:



http://cs231n.github.io/neural-networks-3/#baby

Beyond Training

1. Improving the DNN
2. **Open Source Packages**
   a. **Hardware and OS**
   b. **Python Framework**
   c. **Deep Learning Open Source Packages**
   d. **Effort Estimation**
3. Summary

Hardware and OS

- Amazon Cloud GPU:

AWS Lasagne GPU Setup

Spot ~ $0.0031 per GPU Instance Hour

- IBM Cloud GPU:

http://www-03.ibm.com/systems/platformcomputing/products/symphony/gpuharvesting.html

- Your Linux machine GPU:

```
pip install -r https://raw.githubusercontent.com/dnouri/kfkd-tutorial/master/requirements.txt
```

- Window install

```
http://deeplearning.net/software/theano/install_windows.html#install-windows
```

Starting Tips

- Sanity Checks:
    - DNN Architecture : “Overfit a tiny subset of data” Karpathy
    - Check Regularization ↗ Loss ↗
- Use pre-trained VGG as a base line
- Start with ~3 conv layer with ~16 filter each - quickly iterate

Python

- Rich eco-system
- State-of-the-art
- Easy to port from prototype to production

Podcast : http://www.reversim.com/2015/10/277-scientific-python.html

Python Deep Learning Framework

Keras ,pylearn2, OpenDeep, Lasagne - common base

Tips from Deep Learning Packages

torch code organization

- `main.lua` (~30 lines) - loads all other files, starts training.
- `opts.lua` (~50 lines) - all the command-line options and de
- `model.lua` (~80 lines) - creates AlexNet model and criterion
- `train.lua` (~190 lines) - logic for training the network. we ha
  produces good results.
- `test.lua` (~120 lines) - logic for testing the network on valid
- `dataset.lua` (~430 lines) - a general purpose data loader, n

Caffe's separation configuration ↔code

```
net = NeuralNet(
    layers=[
```

NeuralNet ⟶ YAML text format defining experiment's configuration

Deep Learning
Open Source Packages

Open source progress rapidly⟶ impossible to predict industry's standard

Caffe for applications
Torch and Theano for research on Deep Learning itself
http://fastml.com/torch-vs-theano/

Disruptive Effort Estimation

<u>Feature Eng</u>

<u>Deep Learning</u>

Still requires algorithmic expertise

Summary

- Dove into Training a DNN
- Presented Analysis Capabilities
- Reviewed Open Source Packages

References

Hinton Coursera Neuronal Network

https://www.coursera.org/course/neuralnets

Technion Deep Learning course

http://moodle.technion.ac.il/course/view.php?id=4128

Oxford Deep Learning course

https://www.youtube.com/playlist?list=PLE6Wd9FR--EfW8dtjAuPoTuPcqmOV53Fu

CS231n CNN for Visual Recognition

http://cs231n.github.io/

Deep Learning Book

http://www.iro.umontreal.ca/~bengioy/dlbook/

Montreal DL summer school

http://videolectures.net/deeplearning2015_montreal/

Questions?