NPFL114, Lecture 13

Practical Methodology, TF Development, Advanced Architectures

Milan Straka

Practical Methodology

Hyperparameters Selection

- Grid search

Hyperparameters Selection

- Grid search
- Random search

Hyperparameters Selection

- Grid search
- Random search
- Reinforcement learning

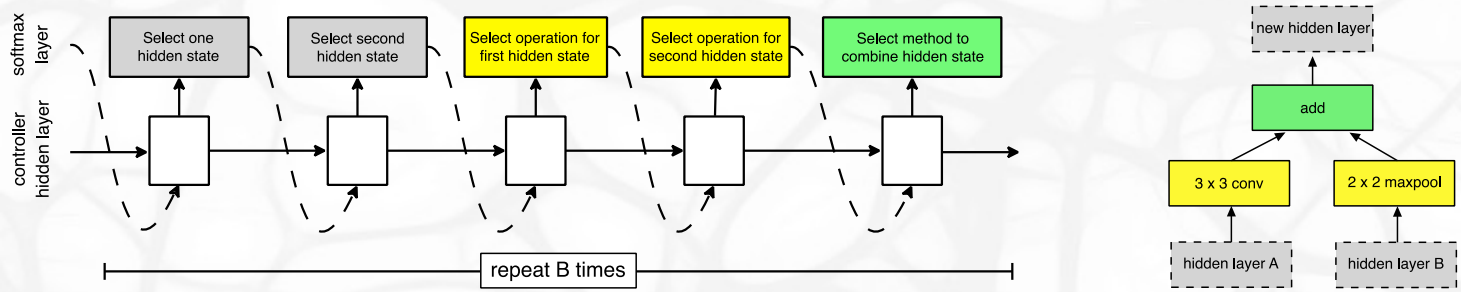


Figure 3. Controller model architecture for recursively constructing one block of a convolutional cell. Each bl ck requires selecting 5 discrete parameters, each of which corresponds to the output of a softmax layer. Example constructed block shown on right. A convolutional cell contains B blocks, hence the controller contains $5B$ softmax layers for predicting the architecture of a convolutional cell. In our experiments, the number of blocks B is 5.

Hyperparameters Selection

- Grid search
- Random search
- Reinforcement learning

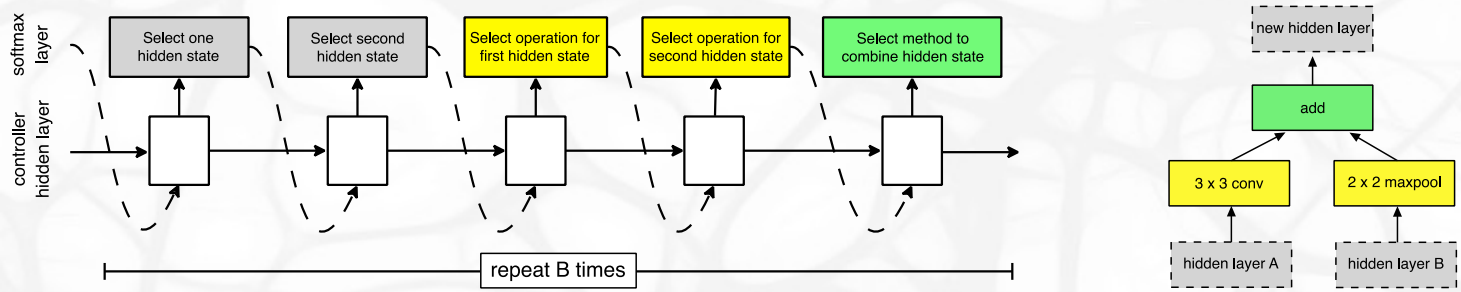


Figure 3. Controller model architecture for recursively constructing one block of a convolutional cell. Each bl ck requires selecting 5 discrete parameters, each of which corresponds to the output of a softmax layer. Example constructed block shown on right. A convolutional cell contains B blocks, hence the controller contains $5B$ softmax layers for predicting the architecture of a convolutional cell. In our experiments, the number of blocks B is 5.

- Bayesian optimization using Gaussian processes

Bayesian optimization using GP

Stochastic process $X_{\mathcal{T}}$ is a *Gaussian process* if for every finite set of indices $t_1, \ldots, t_k$ from $\mathcal{T}$,

$$Y = (X_{t_1}, \ldots, X_{t_k})$$

is a multivariate Gaussian random variable.

Bayesian optimization using GP

We model neural network loss function as a Gaussian process. Prior distribution is usually parametrized using a Matérn 5/2 kernel

$$K_{M52}(x, x') = \theta_0 \left(1 + \sqrt{5}||x - x'||_2 + \frac{5}{3}||x - x'||_2^2\right) e^{-\sqrt{5}||x-x'||_2}.$$

Bayesian optimization using GP

We model neural network loss function as a Gaussian process. Prior distribution is usually parametrized using a Matérn 5/2 kernel $K_{M52}(x, x') = \theta_0 \left(1 + \sqrt{5}||x - x'||_2 + \frac{5}{3}||x - x'||_2^2\right) e^{-\sqrt{5}||x-x'||_2}$.

The posterior distribution (i.e., distribution over all possible functions after sampling several points of the loss function) can be computed in closed form.

Bayesian optimization using GP

We model neural network loss function as a Gaussian process. Prior distribution is usually parametrized using a Matérn 5/2 kernel $K_{M52}(x, x') = \theta_0 \left(1 + \sqrt{5}||x - x'||_2 + \frac{5}{3}||x - x'||_2^2\right) e^{-\sqrt{5}||x-x'||_2}$.

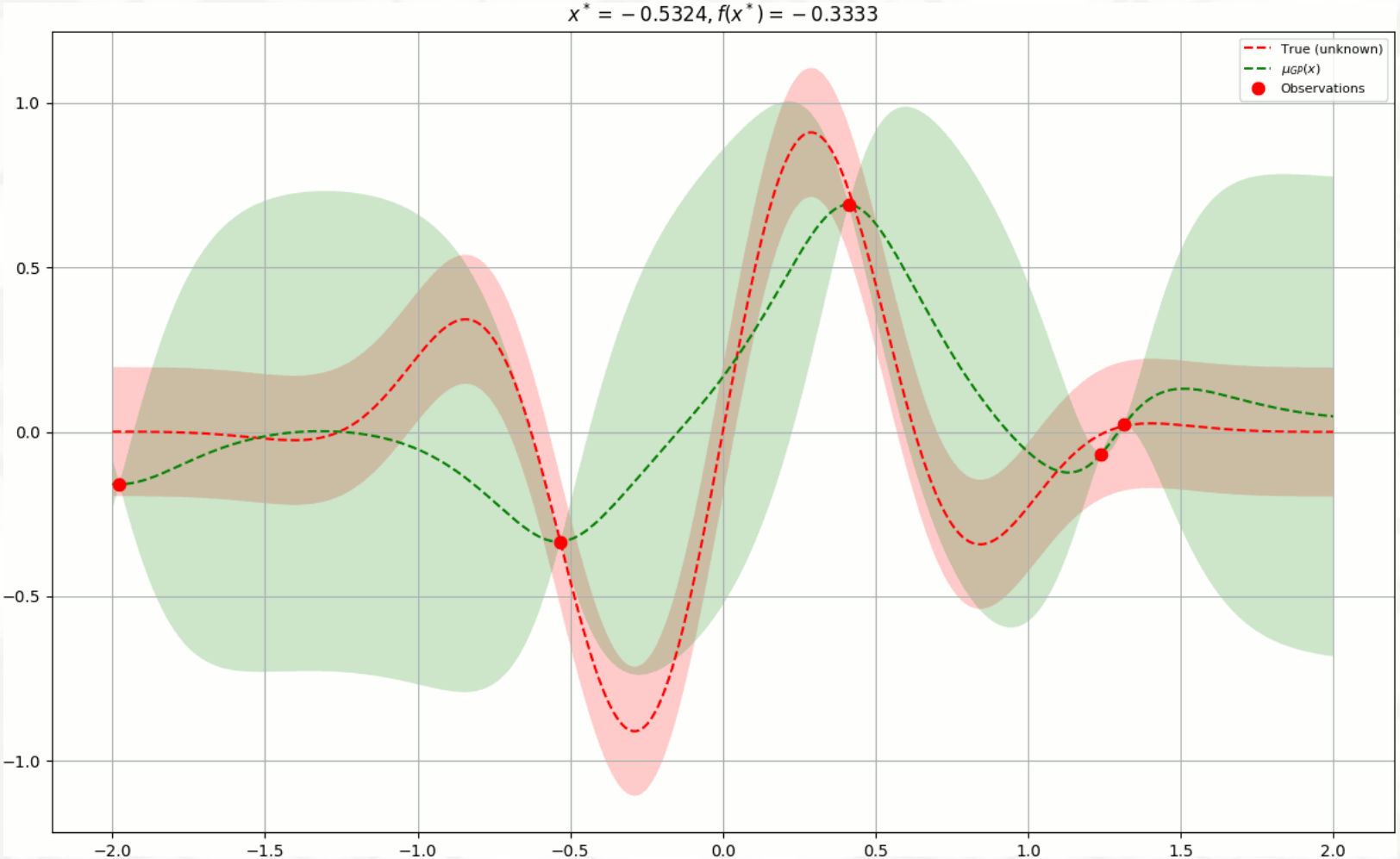
The posterior distribution (i.e., distribution over all possible functions after sampling several points of the loss function) can be computed in closed form.

We choose next candidates from the posterior using an *acquisition function*. One of the commonly used ones is *expected improvement*:

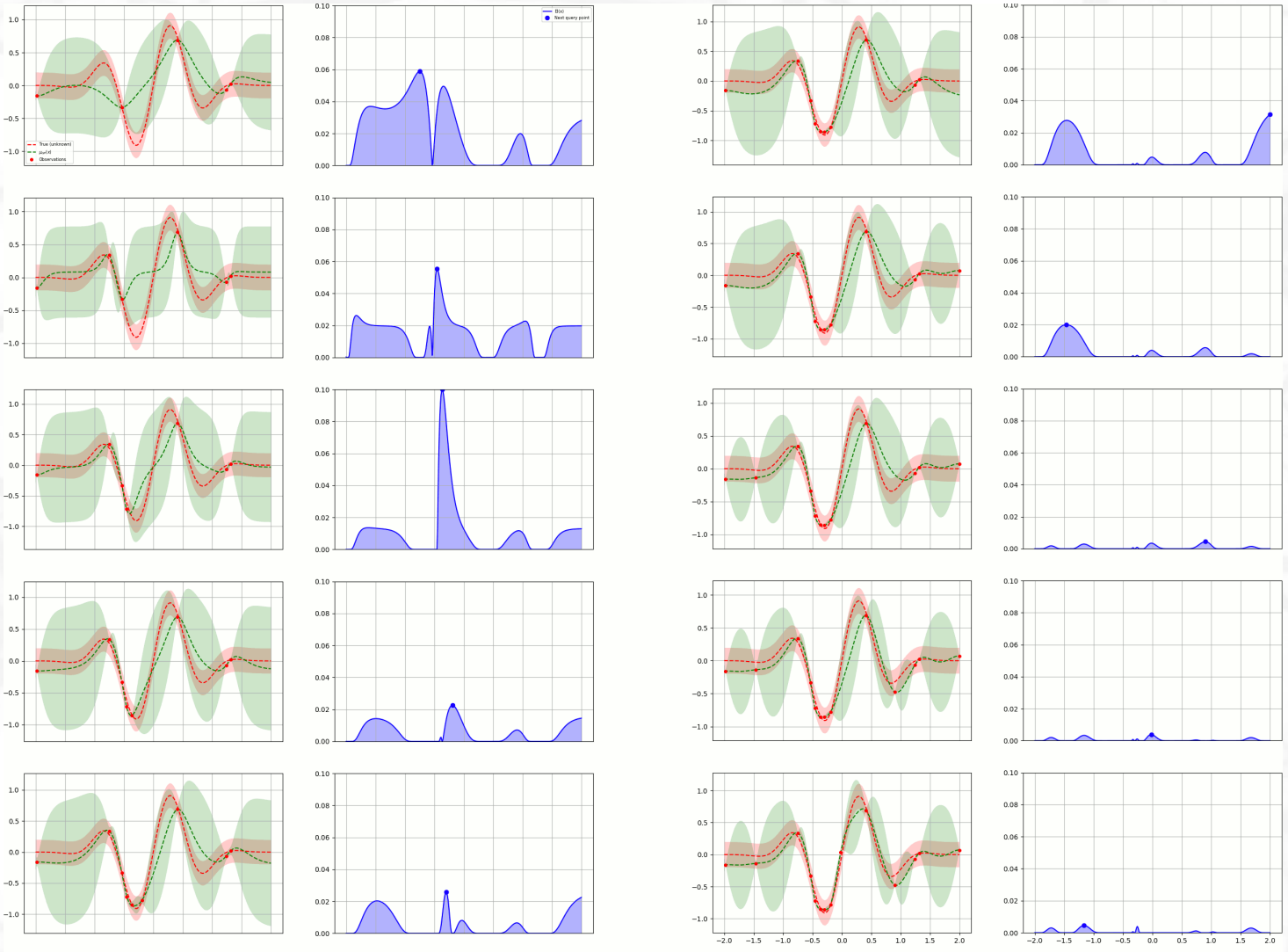
$$\boldsymbol{x}_{next} = \arg\max_{\boldsymbol{x}} \mathbb{E}[\max(best\ solution\ so\ far - model(\boldsymbol{x}), 0)].$$

Luckily, such acquisition function can also be expressed in closed form.

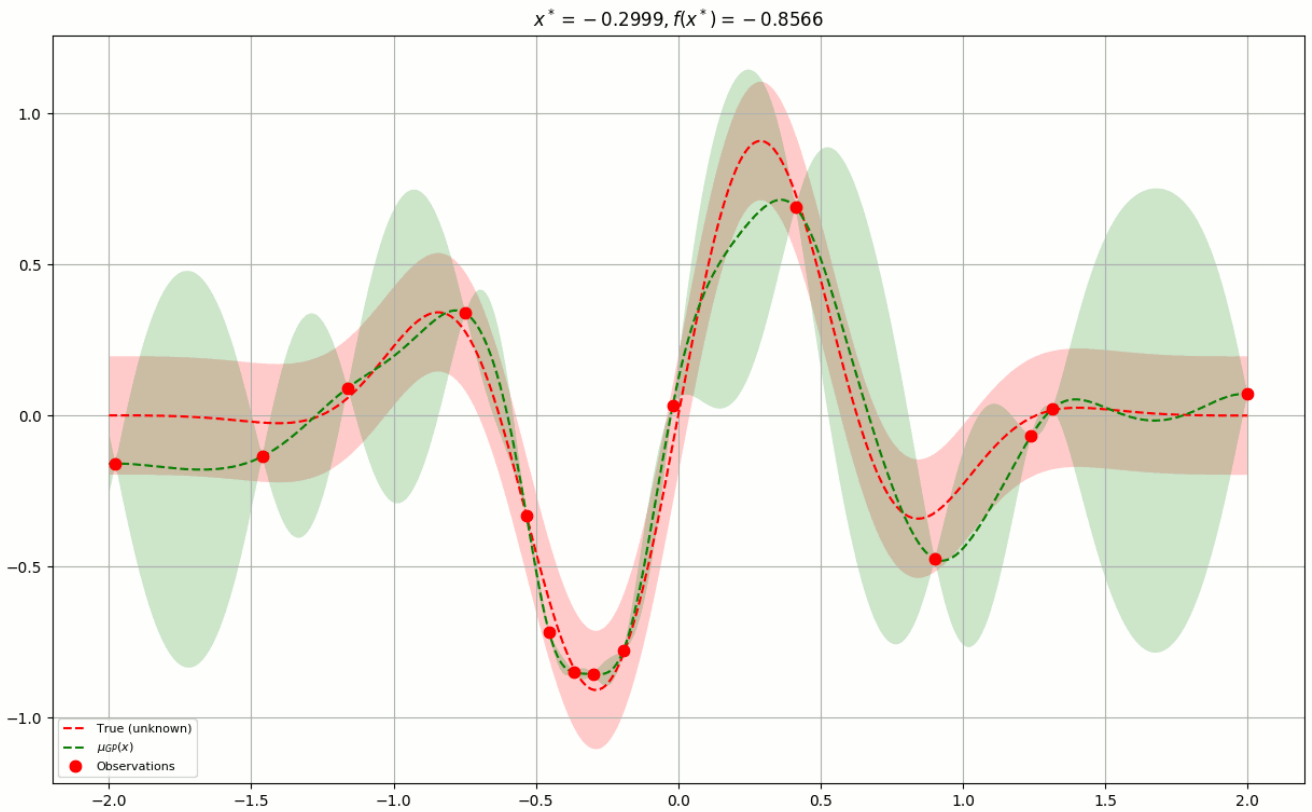
Bayesian optimization using GP



Bayesian optimization using GP



Bayesian optimization using GP



Bayesian optimization using GP

No open-source *out of the box* solution integrated with TensorFlow.

Bayesian optimization using GP

No open-source *out of the box* solution integrated with TensorFlow.

Google has Google Vizier (see *Google Vizier: A Service for Black-Box Optimization* paper for description).

Bayesian optimization using GP

No open-source *out of the box* solution integrated with TensorFlow.

Google has Google Vizier (see *Google Vizier: A Service for Black-Box Optimization* paper for description).

Implementation of GP minimization available in multiple packages, for example `scikit-optimize` (`skopt.gp_minimize`), `GPyOpt`, and many others.

Bayesian optimization using GP

No open-source *out of the box* solution integrated with TensorFlow.

Google has Google Vizier (see *Google Vizier: A Service for Black-Box Optimization* paper for description).

Implementation of GP minimization available in multiple packages, for example `scikit-optimize` (`skopt.gp_minimize`), `GPyOpt`, and many others.

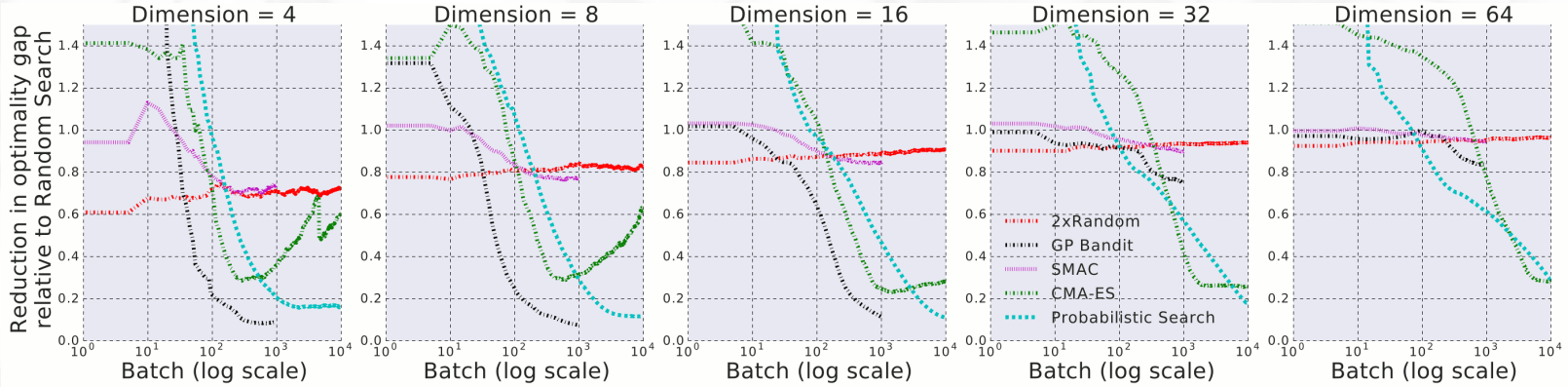


Figure 6: Ratio of the average optimality gap of each optimizer to that of Random Search at a given number of samples. The 2×Random Search is a Random Search allowed to sample two points at every step (as opposed to a single point for the other algorithms).

TF Development

TF Eager

TF Eager mode discards (at least publicly) computation graphs and sessions; all computations are performed immediately.

```
import tensorflow.contrib.eager as tfe

tfe.enable_eager_execution()

print(tf.random_normal([2, 3]))
```

TF Eager

TF Eager mode discards (at least publicly) computation graphs and sessions; all computations are performed immediately.

```
import tensorflow.contrib.eager as tfe

tfe.enable_eager_execution()

print(tf.random_normal([2, 3]))
```

- Asynchronous mode in TF 1.8 and later.

TF Eager

- Only object interface, not the functional one

```
dense = tf.layers.Dense(10)

dense(input)
```

TF Eager

- Only object interface, not the functional one

```
dense = tf.layers.Dense(10)

dense(input)
```

- helper classes for tracking layers (`tfe.Network` and `tf.keras.Model`):

```
class CNN(tfe.Network):
    def __init__(self):
        super(CNN, self).__init__(name='')
        self.dense1 = self.track_layer(tf.layers.Dense(100))
        self.dense2 = self.track_layer(tf.layers.Dense(100))
    def call(self, inputs):
        inputs = self.dense1(inputs)
        return self.dense2(inputs)

cnn=CNN(); ...; print(cnn.variables)
```

TF Eager

- Gradients are computed using explicit `tfe.GradientTape`:

```
with tfe.GradientTape() as tape:
    logits = model(input)
    loss = tf.losses.sparse_softmax_cross_entropy(labels,
                                                  logits)

gradients = tape.gradients(loss, model.variables)
```

TF Eager

- Gradients are computed using explicit `tfe.GradientTape`:

```
with tfe.GradientTape() as tape:
    logits = model(input)
    loss = tf.losses.sparse_softmax_cross_entropy(labels,
                                                  logits)

gradients = tape.gradients(loss, model.variables)
```

- Summaries only with the new interface `tensorflow.contrib.summary`.

TF Eager

- Gradients are computed using explicit `tfe.GradientTape`:

```
with tfe.GradientTape() as tape:
    logits = model(input)
    loss = tf.losses.sparse_softmax_cross_entropy(labels,
                                                  logits)

gradients = tape.gradients(loss, model.variables)
```

- Summaries only with the new interface `tensorflow.contrib.summary`.
- New interface for metrics `tfe.metrics`

```
accuracy = tfe.metrics.Accuracy("test/accuracy")
for data in test_set:
    ...
    accuracy(gold_labels, predictions)

return accuracy.result() # Also generates a summary
```

TF Eager

The eager mode is great for processing structured data like sentences/words/characters, trees, graphs, …

However, it is unsuitable for many HW architectures (GPUs, TPUs, mobile phones, etc).

TF Estimator

The `tf.estimator.Estimator` offers high-level API.

TF Estimator

The `tf.estimator.Estimator` offers high-level API.

- Distributed computation.

TF Estimator

The `tf.estimator.Estimator` offers high-level API.

- Distributed computation.
- Dataset manipulation.

TF Estimator

The `tf.estimator.Estimator` offers high-level API.

- Distributed computation.
- Dataset manipulation.
- Automatic summaries.

TF Estimator

The `tf.estimator.Estimator` offers high-level API.

- Distributed computation.
- Dataset manipulation.
- Automatic summaries.
- checkpoints and restoration.

TF Estimator

The `tf.estimator.Estimator` offers high-level API.

- Distributed computation.
- Dataset manipulation.
- Automatic summaries.
- checkpoints and restoration.
- Train, evaluation and predict modes.

TF Estimator

The `tf.estimator.Estimator` offers high-level API.

- Distributed computation.
- Dataset manipulation.
- Automatic summaries.
- checkpoints and restoration.
- Train, evaluation and predict modes.
- However, currently *not compatible* with eager mode.

TF Estimator

The `tf.estimator.Estimator` is created using a `model_fn`, `params` and many options.

```
def model_fn(features, labels, mode, params):
    images = features["images"]
    ...
    if mode == tf.estimator.ModeKeys.PREDICT:
        return tf.estimator.EstimatorSpec(
            mode=mode, predictions={"labels": predictions})
    ...
    if mode == tf.estimator.ModeKeys.TRAIN:
        return tf.estimator.EstimatorSpec(mode=mode,
        loss=loss, train_op=train_op, eval_metric_ops={
            "accuracy": tf.metrics.accuracy(labels,
                                            predictions)})
    ...
    if mode == tf.estimator.ModeKeys.EVAL:
        return tf.estimator.EstimatorSpec(mode=mode,
        loss=loss, eval_metric_ops={
            "accuracy": tf.metrics.accuracy(labels,
                                            predictions)})
```

TF Estimator

- The individual operations are executed using `train`, `predict` and `evaluate`.

TF Estimator

- The individual operations are executed using `train`, `predict` and `evaluate`.
- All these operations accept many hooks which can customize the behaviour.

TF Estimator

- The individual operations are executed using `train`, `predict` and `evaluate`.
- All these operations accept many hooks which can customize the behaviour.
- The input of these operations is taken from `input_fn`, which should return a `tf.data.Dataset` generating pairs `(features, labels)`, where `features` and `labels` can be either `Tensors`, or dictionaries of `Tensors`.

tf.data API

The `tf.data.Dataset` represents a sequence of elements.

- Can be created from a source
  - `tf.data.Dataset.from_tensor_slices`,
  - `tf.data.Dataset.from_generator`
  - `tf.data.TextLineDataset`
  - `tf.data.TFRecordDataset`
- Can be generated by applying transformations to existing datasets
  - `tf.data.Dataset.shuffle`
  - `tf.data.Dataset.batch`
  - `tf.data.Dataset.repeat`
  - `tf.data.Dataset.{map, flat_map, filter}`
  - `tf.data.Dataset.prefetch`
  - many more

tf.data API

The `tf.data.Iterator` allows iterating a dataset.

tf.data API

The `tf.data.Iterator` allows iterating a dataset.

- The `Iterator.get_next()` returns an operation yielding next element (usually a batch).

tf.data API

The `tf.data.Iterator` allows iterating a dataset.

- The `Iterator.get_next()` returns an operation yielding next element (usually a batch).
- The simplest way to create an `Iterator` is `dataset.make_one_shot_iterator`.

tf.data API

The `tf.data.Iterator` allows iterating a dataset.

- The `Iterator.get_next()` returns an operation yielding next element (usually a batch).
- The simplest way to create an `Iterator` is `dataset.make_one_shot_iterator`.
- The iterator throws `tf.errors.OutOfRangeError` when reaching the end.

```
for _ in range(episodes):
  iterator = dataset.make_one_shot_iterator()

  while True:
    try:
      sess.run(...)
    except tf.errors.OutOfRangeError:
      break
```

Advanced Architectures

WaveNet

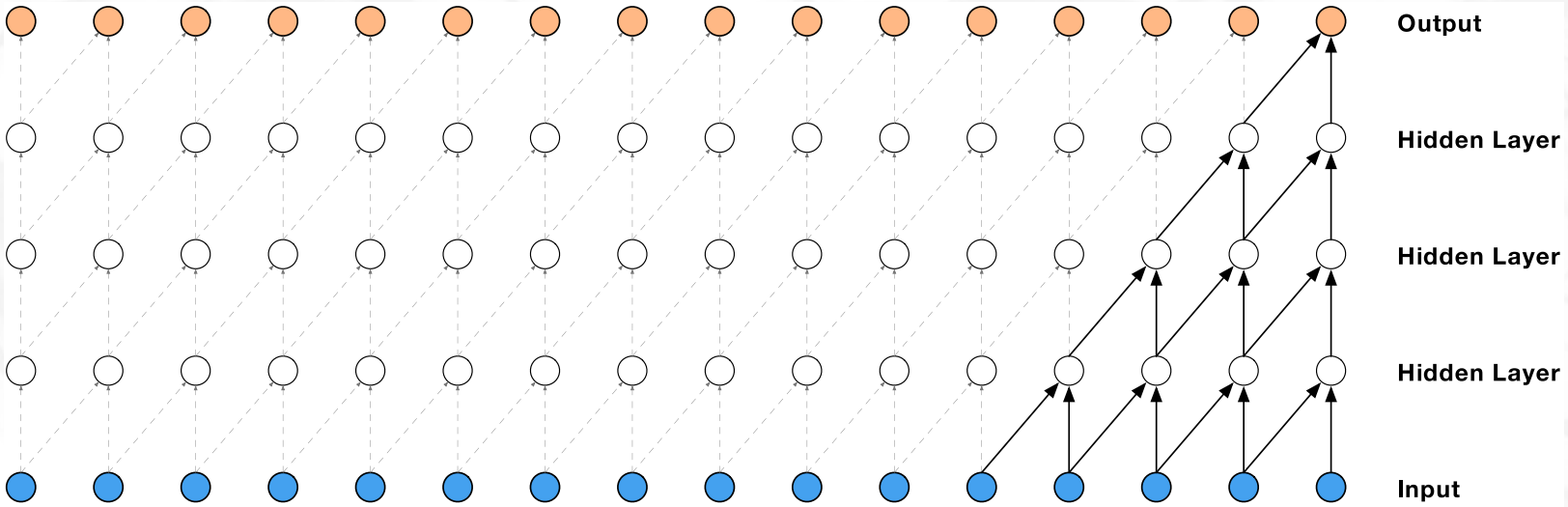


Figure 2: Visualization of a stack of causal convolutional layers.

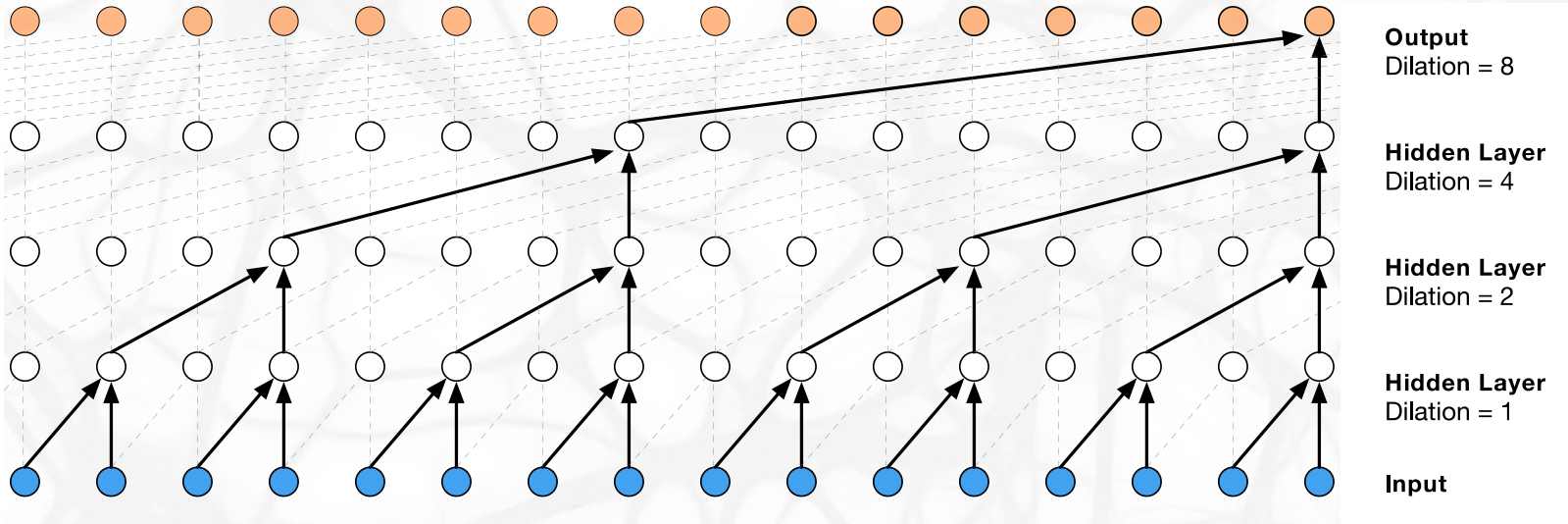


Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

WaveNet

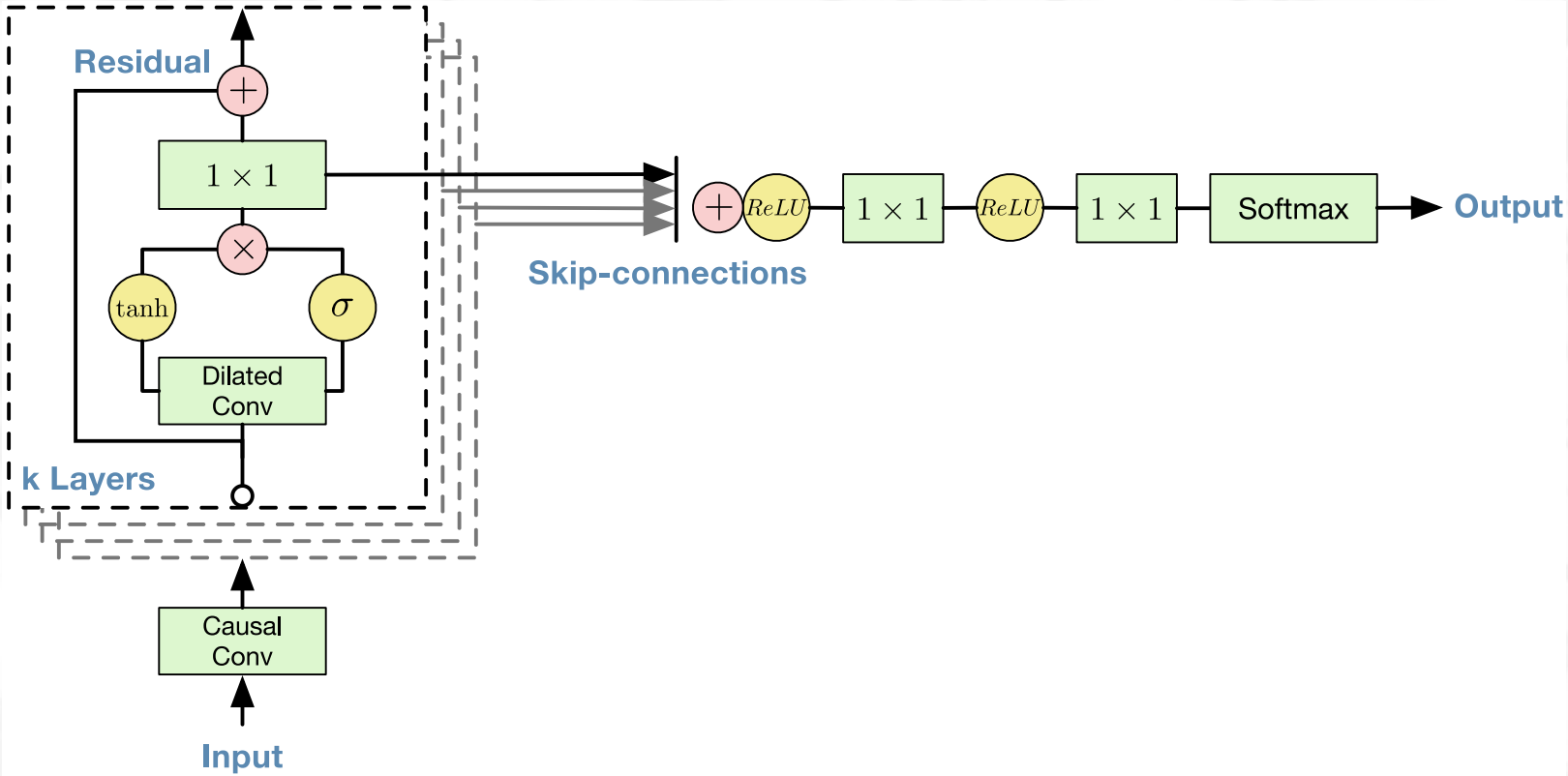


Figure 4: Overview of the residual block and the entire architecture.

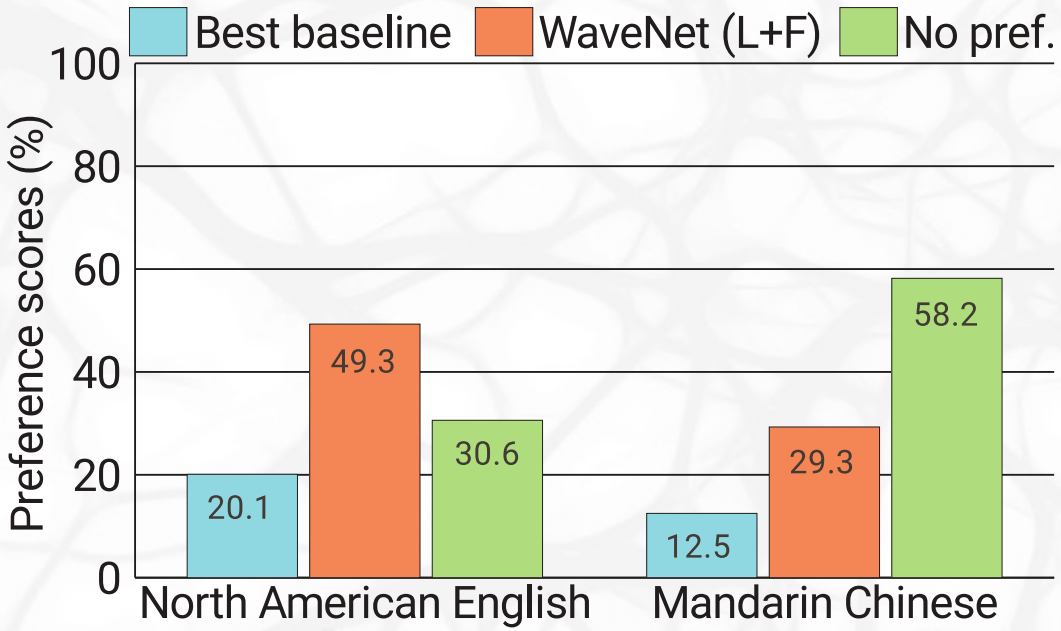
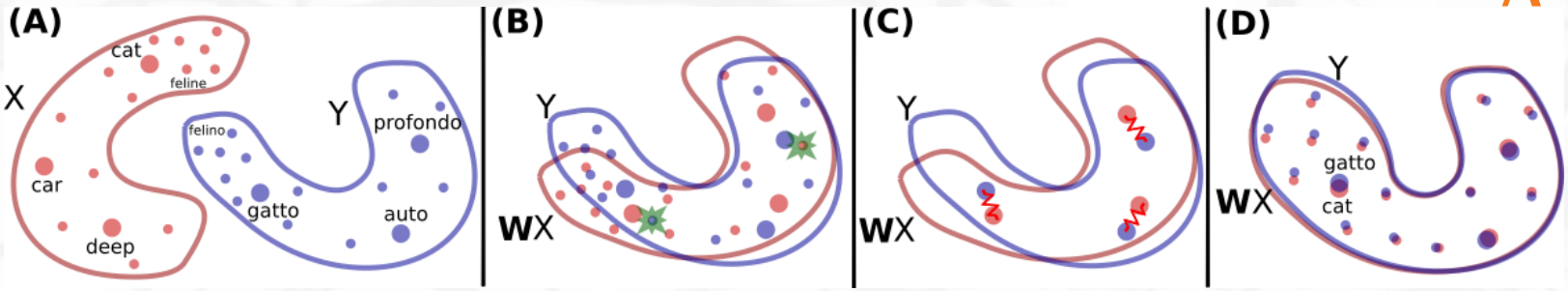
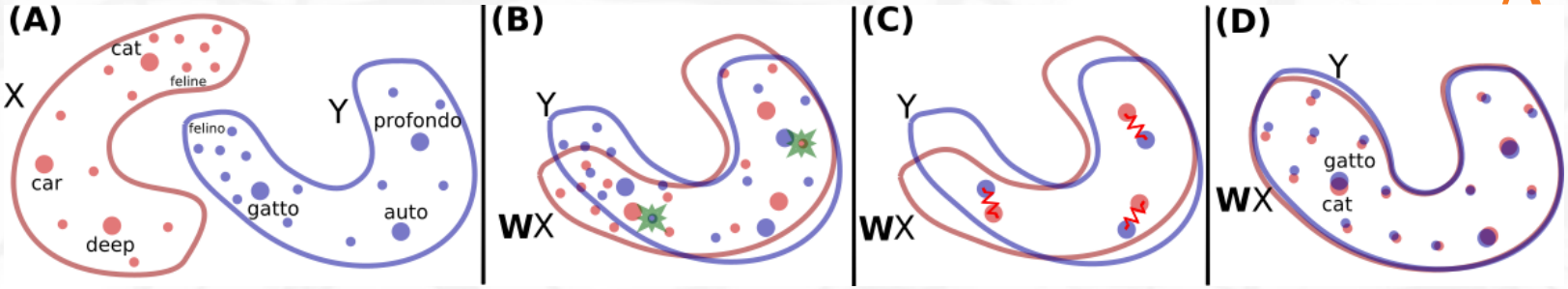


Figure 5: Subjective preference scores (%) of speech samples between (top) two baselines, (middle) two WaveNets, and (bottom) the best baseline and WaveNet. Note that LSTM and Concat correspond to LSTM-RNN-based statistical parametric and HMM-driven unit selection concatenative baseline synthesizers, and WaveNet (L) and WaveNet (L+F) correspond to the WaveNet conditioned on linguistic features only and that conditioned on both linguistic features and $\log F_0$ values.

Word Translation Without Parallel Data

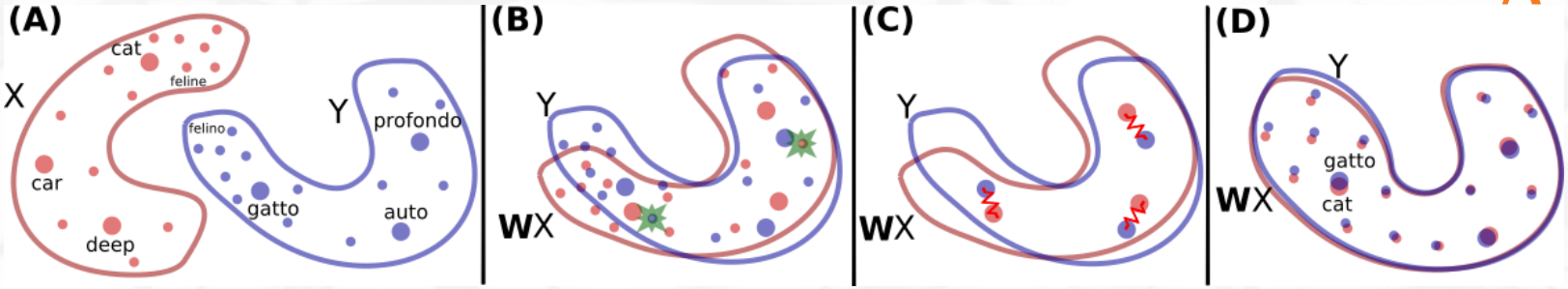


Word Translation Without Parallel Data



$$\boldsymbol{W} = \arg\min_{\boldsymbol{W}'} ||\boldsymbol{W}'\boldsymbol{X} - \boldsymbol{Y}||_{\mathrm{F}}$$

Word Translation Without Parallel Data

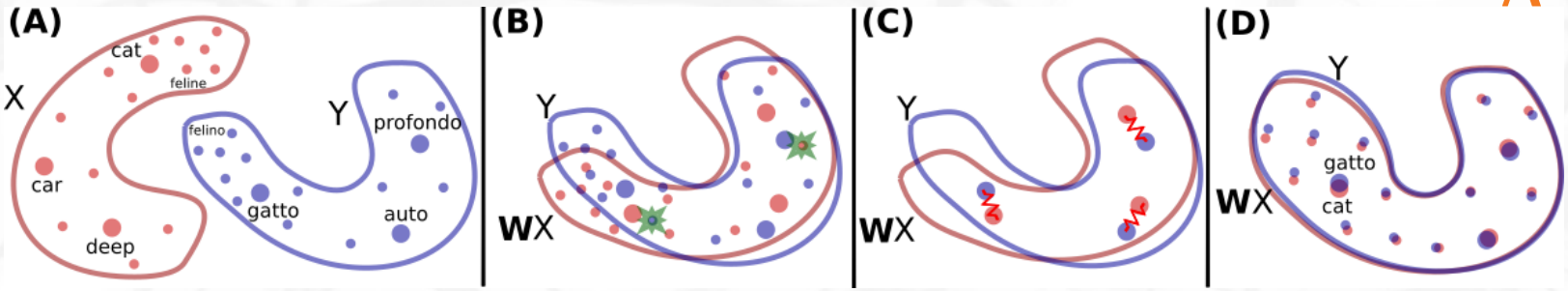


$$\boldsymbol{W} = \arg\min_{\boldsymbol{W}'} ||\boldsymbol{W}'\boldsymbol{X} - \boldsymbol{Y}||_{\mathrm{F}}$$

Close form solution:

$$\boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^T, \text{ where } \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \mathrm{SVD}(\boldsymbol{Y}\boldsymbol{X}^T)$$

Word Translation Without Parallel Data



$$\boldsymbol{W} = \arg\min_{\boldsymbol{W}'} ||\boldsymbol{W}'\boldsymbol{X} - \boldsymbol{Y}||_{\mathrm{F}}$$

Close form solution:

$$\boldsymbol{W} = \boldsymbol{U}\boldsymbol{V}^T, \text{ where } \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T = \mathrm{SVD}(\boldsymbol{Y}\boldsymbol{X}^T)$$

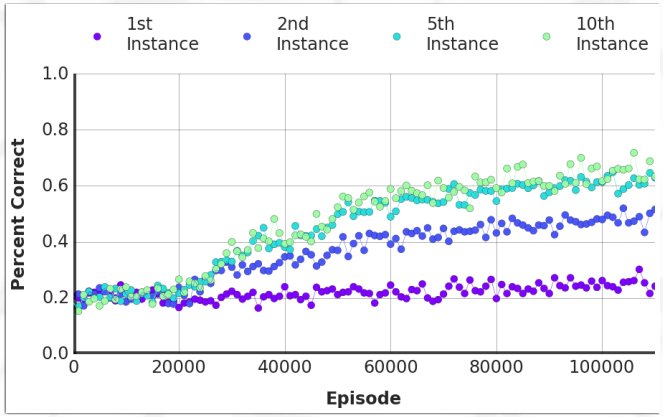
However, such solution needs aligned data. Instead, we employ adversarian training.

Word Translation Without Parallel Data

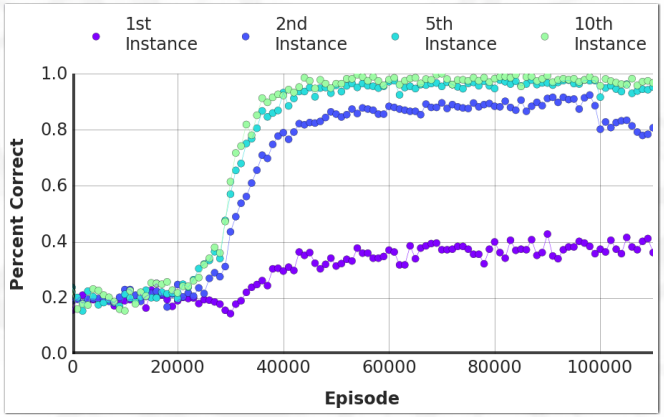
	en-es	es-en	en-fr	fr-en	en-de	de-en	en-ru	ru-en	en-zh	zh-en	en-eo	eo-en
Methods with cross-lingual supervision and fastText embeddings												
Procrustes - NN	77.4	77.3	74.9	76.1	68.4	67.7	47.0	58.2	40.6	30.2	22.1	20.4
Procrustes - ISF	81.1	82.6	81.1	81.3	71.1	71.5	49.5	63.8	35.7	**37.5**	29.0	27.9
Procrustes - CSLS	81.4	82.9	81.1	**82.4**	73.5	**72.4**	**51.7**	**63.7**	**42.7**	36.7	**29.3**	25.3
Methods without cross-lingual supervision and fastText embeddings												
Adv - NN	69.8	71.3	70.4	61.9	63.1	59.6	29.1	41.5	18.5	22.3	13.5	12.1
Adv - CSLS	75.7	79.7	77.8	71.2	70.1	66.4	37.2	48.1	23.4	28.3	18.6	16.6
Adv - Refine - NN	79.1	78.1	78.1	78.2	71.3	69.6	37.3	54.3	30.9	21.9	20.7	20.6
Adv - Refine - CSLS	**81.7**	**83.3**	**82.3**	82.1	**74.0**	72.2	44.0	59.1	32.5	31.4	28.2	**25.6**

Table 1: Word translation retrieval P@1 for our released vocabularies in various language pairs. We consider 1,500 source test queries, and 200k target words for each language pair. We use fastText embeddings trained on Wikipedia. NN: nearest n ighbors. ISF: inverted softmax. ('en' is English, 'fr' is French, 'de' is German, 'ru' is Russian, 'zh' is classical Chinese and 'eo' is Esperanto)

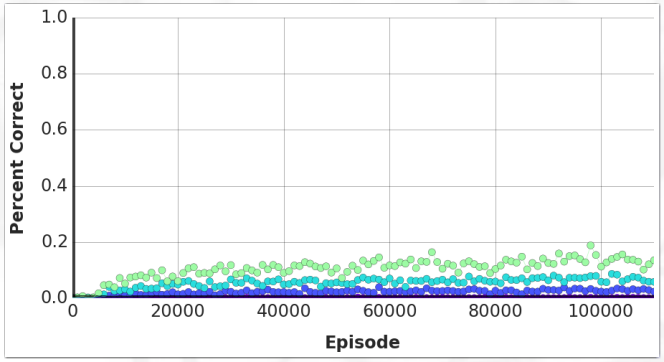
Memory-augmented NNs



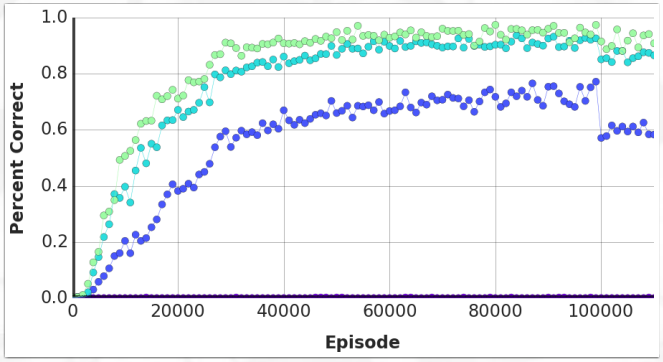
(a) LSTM, five random classes/episode, one-hot vector labels



(b) MANN, five random classes/episode, one-hot vector labels



(c) LSTM, fifteen classes/episode, five-character string labels



(d) MANN, fifteen classes/episode, five-character string labels

Figure 2. Omniglot classification. The network was given either five (a-b) or up to fifteen (c-d) random classes per episode, which were of length 50 or 100 respectively. Labels were one-hot vectors in (a-b), and five-character strings in (c-d). In (b), first instance accuracy is above chance, indicating that the MANN is performing "educated guesses" for new classes based on the classes it has already seen and stored in memory. In (c-d), first instance accuracy is poor, as is expected, since it must make a guess from 3125 random strings. Second instance accuracy, however, approaches 80% during training for the MANN (d). At the 100,000 episode mark the network was tested, without further learning, on distinct classes withheld from the training set, and exhibited comparable performance.